The Mathematical Validation of AI

A Case Study on Bias Reduction in the Fraud Risk Model of the Municipality of Rotterdam

by

Cato Versluis
Student number: 5644100

Applied Mathematics
Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS)
Delft University of Technology

to obtain the degree of Bachelor of Science
at the Delft University of Technology,
to be defended publicly on Friday August 29, 2025 at 3:00 PM.

Project duration:	May 5, 2025 — August 22, 2025
Thesis committee:	Dr. ir. V. N. S. R. Dwarka — TU Delft, supervisor
	Dr. M. E. Kootte — TU Delft, supervisor
	Prof. dr. ir. G. F. Nane — TU Delft, second assessor

TU Delft

1 Abstract

This study critically examines the fairness of Rotterdam's fraud detection system using the Lighthouse Reports' Suspicion Machine framework. By replicating and extending the original model with gradient boosting, dimensionality reduction (PCA), clustering, and adversarial debiasing, the analysis highlights how small changes in input or weighting can substantially alter bias across predefined archetypes. Although clustering revealed no clear separation in risk scores, weighting and adversarial techniques reduced disparities between groups. Limitations include the synthetic nature of the dataset, lack of real fraud labels, and restricted focus on 13 archetypes. The findings stress the importance of validating input data and model design, as fairness outcomes remain highly sensitive to methodological choices.

Contents

2 Introduction

Artificial intelligence has become an integral part of decision-making in both private and public sectors, yet its rapid adoption has sparked societal, political, and scientific debate about accountability, fairness, and bias. Studies increasingly show that data-driven systems can inadvertently reinforce structural inequalities, particularly when deployed in sensitive policy domains such as welfare or fraud detection Lighthouse Reports (2021). Similar concerns have been raised in other municipalities, including Amsterdam, where algorithmic tools for fraud detection also disproportionately affected vulnerable citizens. These examples illustrate that fairness in AI is not merely a technical challenge but a broader societal issue, directly connected to trust in public institutions.

The academic literature underlines that statistical methods are central to addressing such challenges. Statistics contributes to the planning, design, and validation of AI systems, helping to detect bias, assess representativity, and evaluate uncertainty Friedrich et al. (2022). Without these checks, models risk amplifying spurious correlations and systematically disadvantaging certain groups. As such, mathematical validation of AI is essential for ensuring that predictive models used in governance are both reliable and socially acceptable.

Another challenge is that many AI systems function as black boxes, meaning that the way decisions are made is difficult to see or explain Barocas et al. (2019). Because of this opacity, it is often unclear whether unfair outcomes come from the data, the model design, or the way predictions are used. To deal with this problem, the analysis in this study is separated into three stages: input, model, and output. The input phase checks the quality of the data and possible sources of bias, the model phase looks at how design choices may strengthen or reduce these biases, and the output phase measures whether certain groups are over- or underrepresented in the highest risk scores. Looking at each phase separately makes the process more transparent, even when the algorithm itself cannot be fully opened.

This case study investigates *The Mathematical Validation of AI: A Case Study on Bias Reduction in the Fraud Risk Model of the Municipality of Rotterdam*. The fraud risk model was developed by Accenture and used by Rotterdam between 2017 and 2021 to assign individual welfare recipients a risk score through a gradient boosting framework. Previous analysis by Lighthouse Reports showed that this model produced disproportionately high scores for citizens in precarious positions, such as single mothers or migrant workers. To assess whether fairness can be improved, this study focuses on bias reduction by altering the input data and the model structure. Thirteen archetypes defined by Lighthouse Reports serve as the benchmark to evaluate whether changes reduce excess representation of vulnerable groups in the highest-risk category.

The study is organised around the AI pipeline: input, model, and output. The input phase applies statistical checks, dimensionality reduction, and outlier analysis to stabilise the dataset. The model phase tests alternative training strategies, including outlier removal, reweighting, and feature selection. The output phase evaluates fairness by comparing the distribution of risk scores across archetypes and applying statistical inference to test for significant differences.

This research is subject to important limitations, since no verified fraud labels are available. Therefore, the analysis does not evaluate predictive accuracy but instead focuses solely on fairness, operationalised as the relative over- or under-representation of archetypes in the top decile of risk scores. Moreover, the study does not aim to reconstruct the exact Rotterdam model but to explore whether mathematical validation techniques can reduce bias in similar risk-scoring systems. Within these boundaries, the case study contributes to the broader discussion on responsible AI in public administration, illustrating both the potential and the limits of statistical bias reduction methods.

3 Methodology

This thesis investigates the fairness of the risk scoring system used by the municipality of Rotterdam to identify potential cases of welfare fraud. The central research question guiding this case study is: "The Mathematical Validation of AI: A Case Study on Bias Reduction in the Fraud Risk Model of the Municipality of Rotterdam". To answer this question four research questions were formulated.

1. To what extent did the design and use of the Rotterdam risk scoring system take potential bias and fairness considerations into account?
2. Which types of input modifications (outlier removal, reweighting, feature selection) are most effective in reducing bias before the model is trained?
3. How does the gradient boosting model respond to modifications in the input phase aimed at reducing bias in risk scores regarding the 13 archetypes?
4. How can output-phase evaluations, including adversarial debiasing, be used to assess and reduce bias in the risk scores?

Together, the four research questions are designed to follow the flow of bias through the entire AI pipeline. The first question addresses the input phase by evaluating the fairness of the data. The second question investigates the effects of various preprocessing strategies. The third questions focus on the model phase, investigating how the training algorithm interacts with the input to amplify or mitigate bias. The final question looks at the output phase, exploring whether fairness can be improved further by altering how predictions are made or interpreted. This structure ensures that the analysis captures how bias emerges, changes, and potentially compounds across different stages of the system.

To answer these questions, the methodological setup builds on the work of Lighthouse Reports' *Suspicion Machine* project. The original GitHub repository provides a synthetic data generator, a set of 13 predefined archetypes representing vulnerable social groups, and modelling scripts based on a Gradient Boosting Machine (GBM). This thesis adopts these components to reproduce and critically evaluate the risk scoring framework in a controlled setting while allowing meaningful comparisons.

Before the three phases are analysed in detail, Chapter 4 introduces the theoretical foundation for the methods used.

In the Input Phase (Chapter 5), the dataset was systematically examined to detect potential sources of bias before any model training took place. This included checking the demographic composition against official statistics to assess representativeness in Section 5.1, inspecting variable distributions to identify skewness or irregular patterns in Section 5.2, and detecting both univariate and multivariate outliers in Section 5.3. In addition, Principal Component Analysis (PCA) was applied in Section 5.4 to reduce dimensionality and summarise the underlying structure of the data. Based on the resulting components, unsupervised clustering techniques such as k-means and hierarchical clustering were used in Section 5.5 to explore whether specific groups or high-risk individuals formed distinct patterns in the data space.

In the model phase (Chapter 6), the insights gained during the input phase were used to adjust the training process. Specifically, the dataset was modified by removing or reweighting identified outliers, and by selecting a subset of features that were less likely to introduce bias. These adjustments were implemented to examine whether changes in the input could lead to fairer outcomes when training a Gradient Boosting Machine (GBM).

The output phase (Chapter 7) evaluated the fairness of the different models trained in the input phase by comparing how risk scores were distributed across predefined archetypes. Statistical tests such as Welch's t-test and p-values were used to assess whether differences between models were significant in Section 7.1 and Section 7.2. Additionally, an adversarial loss approach was explored in Section 7.3 to determine to what extent group membership could still be predicted from the model's output, providing an alternative lens on potential bias.

4 Background information

This chapter presents the theoretical foundations of the statistical and machine learning methods used in this research. The discussion is organised according to the AI pipeline: input, model and output phase. Each section provides both the mathematical formulation and the rationale for applying the method in the case study.

4.1 Input Phase Methods

The input phase concerns the validation and preparation of the dataset before any predictive modelling is performed. Ensuring that the input variables are well behaved is crucial, since biased or unstable input directly affects the robustness and fairness of the final model. In this phase, several techniques are applied: distribution checks, principal component analysis, Mahalanobis distance, correlation and multicollinearity diagnostics, missing data handling, and feature selection. Together, these methods reduce dimensionality, remove noise, and ensure that the dataset is suitable for subsequent modelling steps.

The literature provides many approaches for data preparation, ranging from robust scaling and outlier detection to feature extraction and advanced imputation. A wide range of methods has been proposed Hand (2012), but only those directly supporting the fairness analysis are considered here.

4.1.1 Distribution Checks

Distribution checks are essential to evaluate whether variables follow approximately normal behaviour, as many statistical and machine learning methods rely on this assumption (Thode, 2002). Two important numerical measures are skewness and kurtosis.

Skewness is defined as

$$\text{Skew}(X) = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^3}{\left(\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2\right)^{3/2}},$$

where values near zero indicate symmetry, positive values indicate right skew, and negative values left skew.

Kurtosis is defined as

$$\text{Kurt}(X) = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^4}{\left(\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2\right)^{2}},$$

with a normal distribution having a kurtosis of approximately three. Larger values suggest heavy tails and a higher probability of extreme outliers.

In addition to numerical measures, visual diagnostics support distribution assessment. Histograms provide a discrete approximation of the probability density, boxplots summarise spread and detect potential outliers, and Q–Q plots compare ordered sample quantiles to those of a theoretical normal distribution. Systematic deviations from the diagonal in a Q–Q plot indicate non-normality.

In this research, distribution checks are applied in the input phase to screen for skewed or heavy-tailed variables. Features that strongly deviate from approximate normality are considered for transformation or exclusion before further analysis. This step stabilises subsequent applications of Mahalanobis distance and PCA, and ensures that outlier detection and fairness analysis are not dominated by irregular distributions.

4.1.2 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a dimensionality reduction technique that transforms correlated variables into a smaller number of uncorrelated components while preserving as much variance as possible Jolliffe (2002). Let $X \in \mathbb{R}^{n \times p}$ be a mean-centred data matrix with n observations and p variables. The sample covariance matrix is defined as

$$\Sigma = \frac{1}{n-1} X^\top X.$$

PCA solves the eigenvalue problem

$$\Sigma v_k = \lambda_k v_k,$$

where v_k is the k-th eigenvector (loading) and λ_k the associated eigenvalue. The principal component scores are given by

$$z_k = Xv_k,$$

representing new orthogonal variables. The variance explained by each component is proportional to λ_k. The proportion of variance explained (PVE) by component k is

$$\text{PVE}_k = \frac{\lambda_k}{\sum_{j=1}^{p} \lambda_j}.$$

A cumulative proportion of variance explained (CPVE) is used to determine how many components to retain. A common rule is to keep the smallest K such that

$$\sum_{k=1}^{K} \text{PVE}_k \geq 0.80.$$

In addition to variance preservation, the elbow method applied to a scree plot of eigenvalues provides a practical criterion for selecting the number of components Abdi and Williams (2010). When variables are measured on different scales, standardisation is performed prior to PCA to prevent large-variance variables from dominating the results.

In this research, PCA is applied in the input phase to reduce the 156 original variables to a more compact feature space while preserving at least 80% of the variance. This transformation reduces redundancy and multicollinearity, improves computational efficiency, and provides a stable basis for clustering and subsequent predictive modelling.

4.1.3 Mahalanobis Distance (MD)

The Mahalanobis distance is a multivariate distance metric that accounts for both scale and correlation between variables Siddappa and Kampalappa (2020); De Maesschalck et al. (2000). For an observation vector $x \in \mathbb{R}^p$ with mean vector μ and covariance matrix Σ, the squared Mahalanobis distance is defined as

$$D^2(x) = (x - \mu)^\top \Sigma^{-1} (x - \mu).$$

Unlike Euclidean distance, the Mahalanobis distance standardises each variable and incorporates correlations among them, making it well-suited for detecting multivariate outliers. Under the assumption of multivariate normality, $D^2(x)$ follows a chi-square distribution with p degrees of freedom. This property allows statistical thresholds to be set as

$$D^2(x) > \chi^2_{p,\,1-\alpha},$$

where α is a chosen significance level, often 0.01 or 0.001 for strict outlier control Gnanadesikan and Kettenring (1972).

In practice, cutoff values can also be chosen empirically. A common approach is the elbow method, where sorted Mahalanobis distances are inspected for a point of sharp increase, indicating the boundary between typical observations and outliers. Robust alternatives replace Σ with robust covariance estimators to avoid distortion when extreme outliers are present.

In this research, Mahalanobis distance is used in the input phase to detect and remove extreme outliers before model training. This reduces noise, improves model robustness, and ensures that fairness metrics are not distorted by a small number of atypical cases. Similar approaches have been successfully applied in medical and pattern-recognition contexts, for example in early Parkinson's detection where Mahalanobis-based feature selection improved classification accuracy Siddappa and Kampalappa (2020).

4.1.4 Correlation Analysis and Multicollinearity

Correlation analysis quantifies the strength and direction of linear relationships between pairs of variables. The most commonly used measure is the Pearson correlation coefficient, defined as

$$r_{XY} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2} \cdot \sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}},$$

where $r_{XY} \in [-1, 1]$. Values close to ± 1 indicate strong linear associations, while values near zero suggest weak or no linear relationship.

When predictors are highly correlated with each other, multicollinearity arises, which inflates the variance of model coefficients and reduces interpretability. One standard diagnostic is the Variance Inflation Factor (VIF), defined as

$$\text{VIF}_j = \frac{1}{1 - R_j^2},$$

where R_j^2 is the coefficient of determination from regressing predictor X_j on all other predictors. A VIF > 10 is commonly considered problematic, although thresholds of 5 are also used in practice (Dormann et al., 2013). Another diagnostic is the condition index

$$\kappa = \sqrt{\frac{\lambda_{\max}(\Sigma)}{\lambda_{\min}(\Sigma)}},$$

where large values indicate near-singular covariance structures.

In fairness-sensitive applications, correlation is also examined between features and group-level outcomes such as subgroup overrepresentation in predicted risk. Features that display an absolute correlation above a chosen threshold (for example $|r| > 0.10$) with subgroup excess share are flagged for removal, as they may act as proxies for sensitive attributes.

In this research, correlation analysis is applied both to reduce redundancy among predictors and to mitigate bias. By removing features with high collinearity or strong correlation with subgroup overrepresentation, the input data becomes more stable for model training and less likely to propagate indirect discrimination.

4.1.5 Missing Data Handling

Missing data is a common issue in applied datasets and can arise from non-response, recording errors, or incomplete administrative records. The mechanism of missingness determines the appropriate treatment strategy and is usually classified into three categories Little and Rubin (2019):

- **Missing Completely at Random (MCAR):** the probability of missingness is independent of both observed and unobserved data.
- **Missing at Random (MAR):** the probability of missingness depends only on observed data.
- Missing Not at Random (MNAR): the probability of missingness depends on unobserved data itself.

The simplest approach is complete-case analysis, which restricts estimation to the subset

$$X^* = \{x_i \in X : \text{no entries of } x_i \text{ are missing}\},$$

ensuring that analyses are performed on a consistent design matrix. However, this method reduces sample size and can introduce bias if missingness is not MCAR.

Alternative strategies include single imputation, such as mean substitution, and multiple imputation, where missing values are drawn from a predictive distribution of the observed data. While multiple imputation generally yields less biased estimates under MAR assumptions, it introduces additional complexity and computational cost.

In this dataset, most variables are administrative counts and binary indicators, which means that true missing values are rare. Instances with incomplete records were excluded, resulting in an analysis based on complete cases

only. Since the number of missing values was negligible, no imputation was required. This approach ensured that subsequent methods such as Mahalanobis distance and PCA operated on a consistent design matrix without introducing potential artefacts from imputation methods.

4.1.6 Feature Selection

Feature selection aims to reduce dimensionality while preserving the most relevant information, improving both interpretability and computational efficiency Guyon and Elisseeff (2003). High-dimensional input can lead to redundant predictors, increased variance, and difficulties in detecting systematic patterns. Two filter-based strategies are applied in this research.

The first strategy relies on Principal Component Analysis (PCA). After computing eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_p$ from the covariance matrix of the centred data, the cumulative proportion of variance explained (CPVE) is examined:

$$\text{CPVE}_K = \sum_{k=1}^{K} \frac{\lambda_k}{\sum_{j=1}^{p} \lambda_j}.$$

The smallest number of components K is retained such that $\text{CPVE}_K \geq 0.80$. The features with the highest contributions to these retained components are prioritised, resulting in a reduced set that captures most of the dataset's variability while mitigating multicollinearity.

The second strategy addresses potential bias propagation. Each original feature is correlated with subgroup overrepresentation in the top risk decile. Features that show an absolute correlation above a chosen threshold (here, $|r| > 0.10$) with excess subgroup share are removed. This step ensures that highly predictive but potentially discriminatory proxies do not dominate the input space.

In this research, these combined feature selection strategies reduced the original 156 variables to a smaller set while preserving the majority of variance and limiting unfair associations. This process provided a stable and interpretable foundation for subsequent modelling steps such as clustering and gradient boosting.

4.1.7 Clustering

Clustering is an unsupervised learning technique used to group observations based on similarity without relying on outcome labels. It provides insight into whether natural substructures exist in the data, which is particularly relevant for exploring fairness-related patterns Murtagh and Contreras (2012).

One widely used method is k-means clustering, which partitions n observations into k clusters by minimising within-cluster variance:

$$\min_{C_1,\ldots,C_k} \sum_{j=1}^{k} \sum_{x_i \in C_j} \|x_i - \mu_j\|^2,$$

where μ_j denotes the centroid of cluster C_j. The optimal number of clusters is often determined using the elbow method, which inspects the reduction in within-cluster sum of squares as a function of k.

An alternative approach is hierarchical clustering, which builds a dendrogram by iteratively merging the most similar observations or clusters. Distances $d(x_i, x_j)$ can be defined using Euclidean or Mahalanobis metrics, and linkage criteria such as Ward's method or average linkage determine how clusters are merged. Cutting the dendrogram at a chosen height yields a partition of the data.

In this research, clustering is applied to the first six principal component scores derived from PCA on the 156 input variables. This projection reduces noise and multicollinearity, allowing clustering to reveal more stable structures. The analysis helps to assess whether groups with high risk scores concentrate in specific regions of the feature space, offering insight into potential fairness issues before predictive modelling.

4.2 Model Phase Methods

The model phase focuses on the construction of predictive models that generate risk scores from the prepared dataset. After dimensionality reduction and preprocessing in the input phase, the objective is to train a flexible

and accurate model that captures complex patterns while remaining interpretable in terms of fairness.

In this research, the Gradient Boosting Machine (GBM) is selected as the primary modelling approach. GBM is particularly suitable for structured, high-dimensional data and has been shown to achieve strong predictive performance while offering tools for regularisation and variable importance analysis Friedman (2001). This choice is further motivated by the fact that the municipality of Rotterdam applied GBM in its own risk modelling system, making it the most relevant benchmark for evaluating fairness in this case study.

Alternative resampling techniques such as the Synthetic Minority Oversampling Technique Chawla et al. (2002a) are frequently used in machine learning to address class imbalance Chawla et al. (2002b). However, since the present dataset does not include observed class labels but instead produces continuous risk scores, class imbalance is not directly applicable and oversampling methods are therefore not used in this study.

4.2.1 Gradient Boosting Machine (GBM)

Gradient Boosting Machines are ensemble learning methods that build a strong predictive model by sequentially combining many weak learners, typically shallow regression trees Friedman (2001). The central idea is to minimise a differentiable loss function by iteratively fitting new learners to the negative gradient of the loss, also known as pseudo-residuals.

Let the training data be $\{(x_i, y_i)\}_{i=1}^n$, with predictors $x_i \in \mathbb{R}^p$ and outcomes y_i. Suppose a model $F_{m-1}(x)$ has been built after $m-1$ iterations. At iteration m, pseudo-residuals are computed as

$$r_{im} = - \left. \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \right|_{F=F_{m-1}},$$

where $L(y, F(x))$ is the chosen loss function. A regression tree $h_m(x)$ is then fitted to $\{(x_i, r_{im})\}_{i=1}^n$. Optionally, a step size ρ_m is found by line search:

$$\rho_m = \arg\min_{\rho} \sum_{i=1}^{n} L\big(y_i,\, F_{m-1}(x_i) + \rho\, h_m(x_i)\big).$$

The model is updated as

$$F_m(x) = F_{m-1}(x) + \nu\, \rho_m\, h_m(x),$$

where $0 < \nu \leq 1$ is the learning rate.

For binary classification, logistic loss is commonly used:

$$L(y, F(x)) = \log\big(1 + \exp(-yF(x))\big), \quad y \in \{-1, +1\},$$

leading to pseudo-residuals of the form

$$r_{im} = \frac{y_i}{1 + \exp(y_i F_{m-1}(x_i))}.$$

Regularisation is essential to avoid overfitting. This can be achieved by restricting the depth of individual trees, subsampling rows or columns, applying shrinkage via the learning rate ν, and implementing early stopping based on validation performance.

An additional advantage of GBM is that variable importance can be quantified. One approach sums the improvement in the loss function contributed by each variable across all trees. Another approach uses permutation importance, which measures the reduction in predictive accuracy when the variable is randomly shuffled. Both measures provide insight into which variables drive predictions Zhang and Ma (2014).

In this research, GBM is chosen as the primary predictive model because of its ability to handle high-dimensional structured data, capture non-linear interactions, and maintain strong predictive accuracy under regularisation Zhang and Ma (2014). This makes it a suitable candidate for evaluating bias in risk score predictions.

4.3 Output Phase Methods

The output phase focuses on the evaluation of the trained model and its predictions. While the model phase produces risk scores, these outputs need to be systematically assessed in order to determine their reliability, statistical validity, and fairness. In this research, two types of analyses are central.

First, statistical inference is applied to compare distributions of risk scores across subgroups. Hypothesis testing, particularly through the use of the two-sample t-test, provides a formal framework to evaluate whether differences in mean scores are statistically significant. Visual tools such as histograms, boxplots, and quantile–quantile plots complement these tests by offering intuitive insights into the shape and spread of the distributions.

Second, fairness analysis is conducted by measuring subgroup representation in the highest risk decile. This excess share metric quantifies whether specific archetypes are disproportionately present among the individuals flagged as high risk. In a fair system, each subgroup would be represented approximately according to its baseline share in the population.

The literature offers many other methods for evaluating model outputs. Performance-oriented measures include the area under the receiver operating characteristic curve (AUC-ROC), precision–recall curves, and Brier scores for probabilistic calibration Saito and Rehmsmeier (2015). Fairness-oriented measures include demographic parity, equal opportunity, and predictive parity, which formalise different notions of fairness in classification Barocas et al. (2019). These methods are not applied in the present study due to the absence of ground-truth outcome labels, but they illustrate the breadth of approaches available for evaluating model outputs.

Together, the methods selected here ensure that the output phase does not only assess predictive validity but also provides insight into potential disparities and structural biases embedded in the model's predictions.

4.3.1 T-test and P-value

The two-sample t-test evaluates whether the mean outcomes of two independent groups differ beyond what would be expected from sampling variability James et al. (2023)). Let $\bar{x}_1, s_1^2, n_1$ and $\bar{x}_2, s_2^2, n_2$ denote the sample means, sample variances, and sizes for groups 1 and 2. When population variances are not assumed equal, the Welch test is appropriate, with test statistic

$$t \;=\; \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}},$$

and degrees of freedom given by the Satterthwaite approximation

$$\nu \;=\; \frac{\left(\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}\right)^2}{\frac{(s_1^2/n_1)^2}{n_1 - 1} + \frac{(s_2^2/n_2)^2}{n_2 - 1}}.$$

Under the null hypothesis $H_0 : \mu_1 = \mu_2$, the statistic t is referenced to a t distribution with ν degrees of freedom. A $(1-\alpha)$ two-sided confidence interval for $\mu_1 - \mu_2$ is

$$(\bar{x}_1 - \bar{x}_2) \;\pm\; t_{1-\alpha/2,\,\nu}\,\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}\,.$$

The p-value is the probability, under H_0, of observing a test statistic at least as extreme as the realised value Hardt et al. (2016). Small p-values indicate that the observed difference in means would be unlikely if H_0 were true, but they do not measure the size or importance of an effect Wasserstein and Lazar (2016). Good practice is to report p-values together with confidence intervals and an effect size. A common standardised effect size is Cohen's d,

$$d \;=\; \frac{\bar{x}_1 - \bar{x}_2}{s_p}, \qquad s_p \;=\; \sqrt{\frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2}},$$

which expresses the mean difference in pooled standard deviation units.

Assumptions of the t-test include independence within and between groups and approximately symmetric sampling distributions of the group means. When sample sizes are moderate to large, the central limit theorem often justifies the test even for non-normal raw data; nevertheless, distribution checks (histograms, boxplots, Q-Q plots) are inspected to identify severe asymmetry or heavy tails. If assumptions are strongly violated, a nonparametric alternative such as the Mann-Whitney test can be considered, though it tests for stochastic dominance rather than equality of means.

In this research, the Welch t-test is used to compare mean bias metrics (for example, excess share in the top decile) between model variants. Reporting includes the test statistic, degrees of freedom, two-sided p-value, a confidence interval for the mean difference, and Cohen's d. This combination allows assessment of both statistical and practical significance for fairness-related differences.

4.3.2 Bias Measurement

To evaluate fairness in predictive modelling, it is essential to assess whether certain subgroups are disproportionately represented among the individuals flagged as high risk. In this research, fairness is quantified through the concept of *excess share*, which measures the over- or under-representation of subgroups in the highest risk decile Barocas et al. (2019), Hardt et al. (2016).

Let G denote a subgroup with n_G members in a population of size N. The baseline share of G in the population is

$$\pi_G = \frac{n_G}{N}.$$

If the top q proportion of individuals is flagged as high risk, the expected fair share of subgroup G in this set is $q \cdot n_G$. Let $n_G^{(q)}$ denote the observed number of G individuals in the top qN. The excess share is defined as

$$\Delta_G = \frac{n_G^{(q)}}{qN} - \pi_G.$$

A positive Δ_G indicates over-representation, while a negative value indicates under-representation relative to the baseline. For example, if $\pi_G = 0.20$ and 30% of the top decile consists of members of G, the excess share is $\Delta_G = 0.30 - 0.20 = +0.10$.

This metric provides a clear, interpretable measure of group fairness: in a system without structural bias, subgroup representation in the top-risk set should roughly follow population proportions. While exact equality is not expected due to sampling variability, systematic and consistent deviations signal potential bias.

Alternative fairness definitions exist in the literature, such as demographic parity, equal opportunity, and predictive parity Chouldechova and Roth (2018). These definitions formalise different trade-offs between error rates across groups, but their application requires observed outcomes. Since the present dataset contains only model-generated risk scores without true labels, the excess share metric is the most appropriate fairness measure.

In this research, excess share is computed for each archetype under all model variants (original, cleaned, and weighted). Results are presented both in tabular form and as barplots, highlighting which groups are consistently over- or under-represented. This approach enables direct comparison of fairness across modelling strategies, making visible whether bias is reduced by outlier removal or reweighting.

4.3.3 Adverserial Loss

To further reduce model bias in risk predictions, an adversarial debiasing framework can be applied, Zhang et al. (2018). This method is based on the simultaneous training of two models: a predictor and an adversary. The predictor is designed to estimate the target variable $\hat{Y}$, in this case a risk score, from input features X. The adversary is trained to predict the protected attribute Z, for example an archetype, based on the predicted output $\hat{Y}$.

The core objective is to ensure that the predicted output remains informative for the main task while becoming uninformative about the protected attribute. This setup is formally described in the work of Zhang et al. (2018), who

demonstrate that adversarial training can mitigate unwanted bias without significantly reducing model performance.

The total loss function used in adversarial debiasing is defined as follows:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{predictor}} - \lambda \cdot \mathcal{L}_{\text{adversary}}$$

In this expression, $\mathcal{L}_{\text{predictor}}$ denotes the loss of the main model (for example, mean squared error), and $\mathcal{L}_{\text{adversary}}$ represents the loss of the adversary model (for example, cross-entropy or classification error). The parameter λ controls the strength of the fairness constraint. A higher value of λ results in stronger suppression of bias, possibly at the cost of predictive accuracy, but this comes at the downside of reduced predictive accuracy, as the model is penalised for exploiting patterns that may also contain useful information for the main prediction task.

In the current case study, the predicted output $\hat{Y}$ refers to a continuous risk score for welfare fraud, and the protected variable Z is a categorical label indicating one of thirteen predefined archetypes. These archetypes represent vulnerable social profiles identified in earlier fairness analyses. The adversarial model receives $\hat{Y}$ as input and attempts to predict Z. When the adversary fails to do so accurately, it indicates that the risk score no longer reveals information about the archetype, thus suggesting improved fairness.

This method aims to achieve demographic parity, meaning that the distribution of predicted scores is statistically independent of the protected attribute. Formally, this implies:

$$P(\hat{Y} = y) = P(\hat{Y} = y \mid Z = z)$$

Alternatively, if the true label Y is available, the adversary can be conditioned on both $\hat{Y}$ and Y to enforce equality of odds:

$$P(\hat{Y} = y \mid Y = y) = P(\hat{Y} = y \mid Z = z, Y = y)$$

In practice, this framework can be used to evaluate and compare different model configurations. For example, it can be applied to the original model, as well as to alternative models trained on principal components or specific feature categories. In each case, a separate adversary can be trained to assess whether the model output contains residual bias toward any of the archetypes.

5 Input Phase Validation

5.1 Visualization and Representativeness Dataset

The Rotterdam system originally included 314 features, but this was already the result of an internal reduction performed by the municipality. The criteria for this reduction remain unclear Lighthouse Reports (2021), raising concerns about transparency in feature selection. Since many of the included variables do not have a clear relationship with welfare fraud, such as administrative process indicators (*number of submitted documents*, *type of incoming call*) or subjective assessments by case workers (*personal hygiene*, *competence to plan and organise*, and *language requirements*) it is questionable whether the feature set meaningfully captured fraud-related behaviour. Although eligibility for social assistance in the Netherlands includes a language requirement Rijksoverheid (2023b), this condition is debatable. Linking access to welfare support to language proficiency may disproportionately disadvantage non-native speakers and raises concerns about potential discrimination. In addition, the broader policy framework surrounding social assistance, as described in official government documents Rijksoverheid (2023a), highlights how such requirements can unintentionally reinforce exclusion rather than promote integration. Instead, it seems likely that the inclusion of all these variables increased the risk of bias, as they may reflect registration practices or case worker opinions rather than objective signals of fraud. The lack of clarity on how features were filtered makes it uncertain whether potential sources of bias were addressed or whether irrelevant proxies were retained.

Even after this reduction, the dataset remained large and complex, with 12,645 observations recorded across the 314 features. To make the data suitable for analysis, all values were restructured: originally stored in a single column, they were separated into individual, feature-specific columns to improve interpretability. A comprehensive overview of each feature, including its name, data type, subtype, and meaning, was compiled and is presented in Table 3.

The dataset was then checked for missing values; no such values were present, eliminating the need for imputation. Next, the assigned data types were validated against the raw data, which revealed several key observations. All variables in the original dataset were encoded as integers, with the exception of a single feature, *unique district ratio* (feature 24), which was a float. A *float* variable is a numerical feature that can take non-integer values, such as decimals (e.g. 0.25 or 3.14). Floats are typically used when more precision is required than whole numbers can provide. In contrast, a *Boolean variable* is a binary feature that can only take the values 0 or 1, representing the logical states false and true. Although booleans are conceptually different from integers, in this dataset they are stored as integer values and therefore follow the data structure.

Particular attention was given to four features that had been initially labeled as categorical: the client's spoken language (feature 244), whether the client meets the language requirement (feature 247), the client's expression about exiting the program (feature 249), and the case manager's assessment regarding the client's exit from the program (feature 250). Unlike binary variables, these features contain multiple coded values that represent nominal categories. However, due to encoding limitations, they appear as numeric values without associated category labels, making interpretation less straightforward.
To understand their structure and assess whether they require transformation, histograms were created and are shown in Figure 1. The visualizations reveal that the spoken language feature is heavily skewed, with a few dominant codes and a long tail of rare values. In contrast, the language requirement and exit-related variables have more compact distributions with only a few frequently occurring categories. This supports their interpretation as nominal variables and indicates that they should be properly one-hot encoded if included in further modeling steps. Boxplots and Q–Q plots were omitted, as they assume a continuous or ordinal scale that is not applicable in this context.

Beyond the inspection of individual variables, it is equally important to consider whether the dataset as a whole is representative of the underlying population. This step is crucial because even if individual features are well-structured, systematic imbalances at the population level can still introduce bias into the model outcomes.

5.1.1 Comparison with CBS Population Data

To evaluate the representativeness of the dataset, its demographic composition was compared with official statistics from Statistics Netherlands Statistics Netherlands (CBS) (2020). The comparison focused on three key dimensions: age, gender, and district of residence. Substantial deviations were observed, which indicate that the training data does not accurately reflect the population of Rotterdam.

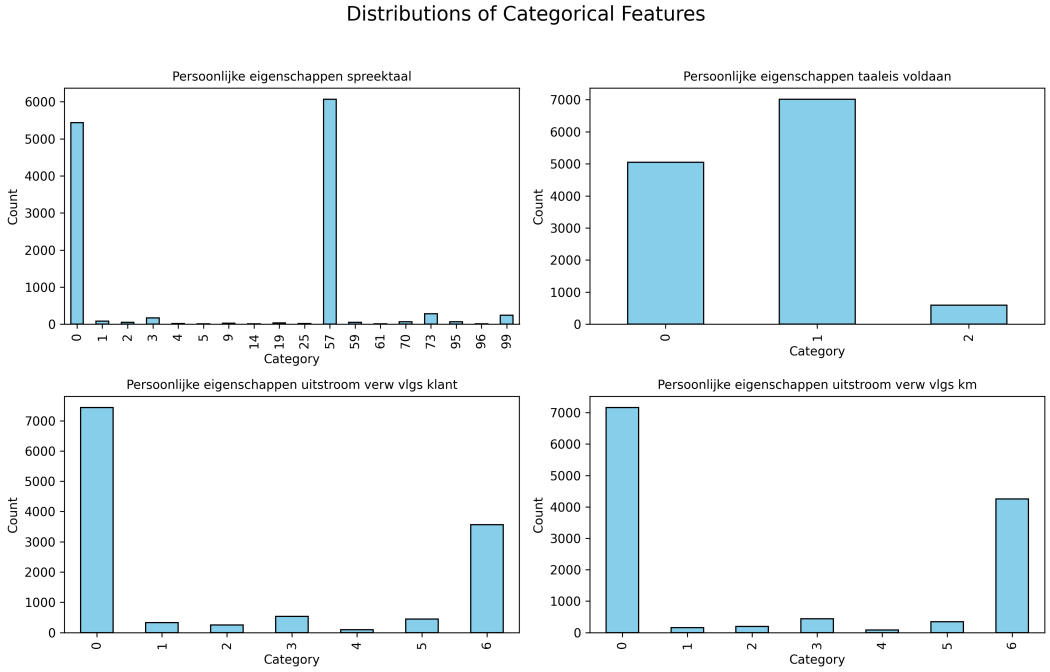


Figure 1: Distribution of the four categorical variables

Age: Although the synthetic dataset includes citizens of all ages, it is not meaningful to analyse the group 0-15 years, as individuals below 18 are not eligible for welfare benefits and therefore cannot be relevant cases for fraud detection. However, the group 15-25 years should be represented, since young adults aged 18 and above are eligible. In CBS statistics, the 15-25 group accounts for 13.3% of the population, while in the dataset this group makes up only 1.1%. This severe underrepresentation is striking, especially given that the dataset includes three additional years (15-18) that should strengthen the group's presence. Middle-aged adults are by far the most dominant: citizens aged 45-65 make up 65.1% of the dataset, compared to only 24.5% in the CBS data. At the same time, elderly residents (65 years and above) are strongly underrepresented, with only 4.2% in the dataset compared to 15.4% in the population. These differences indicate that the model was trained predominantly on middle-aged individuals, limiting its ability to generalise to younger or older citizens. Figure 2 summarises these findings.

Leeftijdsverdeling

Age_group	CBS_%	Synth_%	Diff
0 tot 15 jaar	15.9	0.0	-15.9
15 tot 25 jaar	13.34	1.1	-12.24
25 tot 45 jaar	30.84	29.65	-1.19
45 tot 65 jaar	24.54	65.06	40.52
65 jaar of ouder	15.38	4.19	-11.19

Figure 2: Age distribution in dataset compared with CBS 2020

Gender: The gender distribution is closer to the Rotterdam population but still shows deviations. As can be seen in Figure 3 women constitute 50.7% of the population according to CBS, while they represent only 48.3% in the dataset. Men are therefore slightly overrepresented (51.7% in the dataset compared with 49.3% in CBS data). While the gap is relatively small compared to age and district, it still suggests that the dataset does not fully mirror the gender balance in Rotterdam.

Geslacht vergelijking

Source	Female %
CBS	50.65
Synth	48.26
Difference	-2.39

Figure 3: Gender distribution in dataset compared with CBS 2020

District of residence: More severe deviations appear in the distribution across districts, as can be seen in Figure 4. Feijenoord is overrepresented in the dataset (17.1%) compared to CBS statistics (11.8%), also Delfshaven is slightly inflated (14.6% in the dataset versus 11.6% in CBS). In contrast, Noord (2.2% in the dataset versus 8.1% in CBS) and IJsselmonde (4.3% versus 9.4%) are strongly underrepresented. Even more striking is the *Other* category, which aggregates smaller districts: in CBS this group accounts for only 2.1%, but in the dataset it is inflated to 13.3%. This inflation suggests that the restructuring of neighbourhood categories has compressed many cases into an artificial residual group, obscuring the true distribution across the city.

District % (CBS vs Synth)

CBS_wijk	CBS_%	Synth_%	Diff
Charlois	10.66	10.23	-0.42
Delfshaven	11.8	13.36	1.56
Feijenoord	11.76	17.07	5.31
IJsselmonde	9.42	4.33	-5.1
Kralingen-Crooswijk	8.37	4.82	-3.54
Noord	8.06	2.19	-5.87
Other	2.08	13.32	11.24
Prins Alexander	14.74	3.96	-10.78
Rotterdam Centrum	5.54	1.23	-4.31

Figure 4: District of residence distribution in dataset compared with CBS 2020

The combined demographic deviations show that the dataset cannot be considered representative of the Rotterdam population. The dominance of middle-aged groups, the underrepresentation of young and elderly citizens, and the geographic imbalance in districts suggest that the training data structurally biases the model toward specific population segments. Since the model assigns risk scores based on this unbalanced input, it is likely that these biases are propagated into its predictions. This undermines the fairness of the system, particularly for young adults and residents of underrepresented districts.

5.2 Distribution checks

Although the gradient boosting model applied in this study does not require any assumptions about the distribution of input variables, distribution checks for all variables were still conducted to better understand the structure of the dataset. These checks help identify extreme asymmetries, unusual concentration of values, and potential data quality issues that may influence later modeling stages or interpretability of results. By examining the distribution of each variable, we can identify features whose structure may require special attention during preprocessing or modeling. Highly skewed or imbalanced distributions are not necessarily problematic, but they may indicate variables that behave differently than assumed by standard statistical methods, due to genuine rarity, policy-driven selection, or technical encoding choices. Recognizing these exceptions is essential to make informed decisions about transformation, inclusion, or interpretation.

Distributional characteristics are particularly important in a real-world dataset like this one, which includes complex administrative and demographic information. Not all variables are expected to follow a symmetric or normal distribution, since variables such as gender, age, or residential district are often unevenly distributed in the actual population.

To assess these patterns and conduct the distribution checks, histograms were generated for each variable and three statistics were calculated: the difference between mean and median, skewness and kurtosis. The twenty variables with the largest values for each statistic were plotted in Figures 5, 6 and 7. For categorical variables, frequencies were computed and bar charts were created to reveal imbalances in category proportions. For boolean variables bar charts showing the count of zeros versus ones were produced.

All twenty numeric variables with the greatest difference between mean and median exhibit either right skew or near symmetry, except for belemmering_dagen_psychische_problemen, which shows left skew. This indicates that among those for whom this barrier is present, most have a relatively high number of days with recorded psychological issues, while only a few cases involve short or isolated periods.

The twenty variables with the highest skewness and the twenty with the highest kurtosis consist solely of binary features. The skewness histograms show that most observations take the value zero with only a few cases of one; examining those cases of one reveals patterns of rare yet meaningful events, such as recorded exemptions, barriers to participation, or specific interventions. The kurtosis histograms likewise show a majority of zeros, except for adres_uniek_wijk_ratio, which has a higher proportion of ones and therefore heavier distribution tails, indicating greater variability in unique addresses across wards.

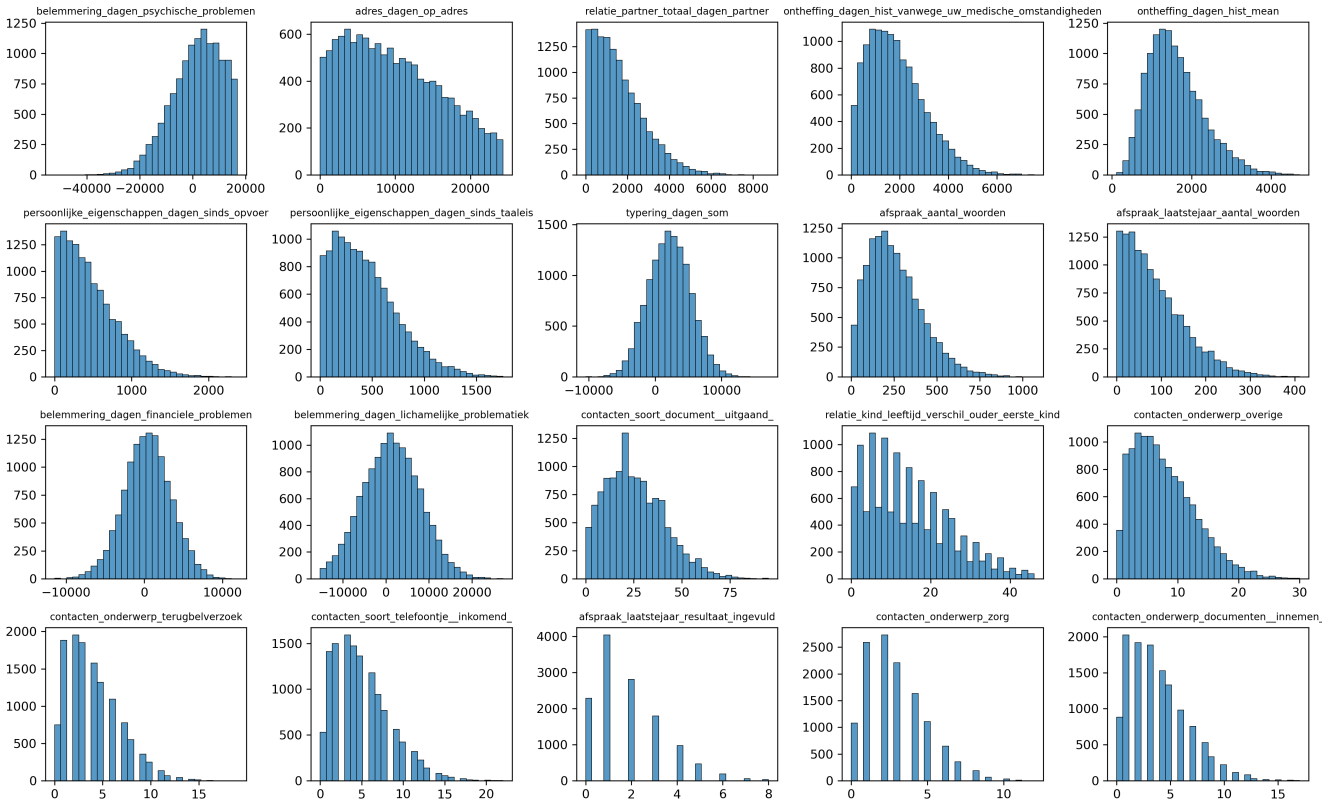


Figure 5: Histograms top 20 highest Mean-Median Differences

While the distribution checks revealed some structural imbalances, they did not indicate extreme values that clearly qualify as outliers. To explore this further, the next section applies both univariate and multivariate outlier detection techniques to identify potentially problematic observations.

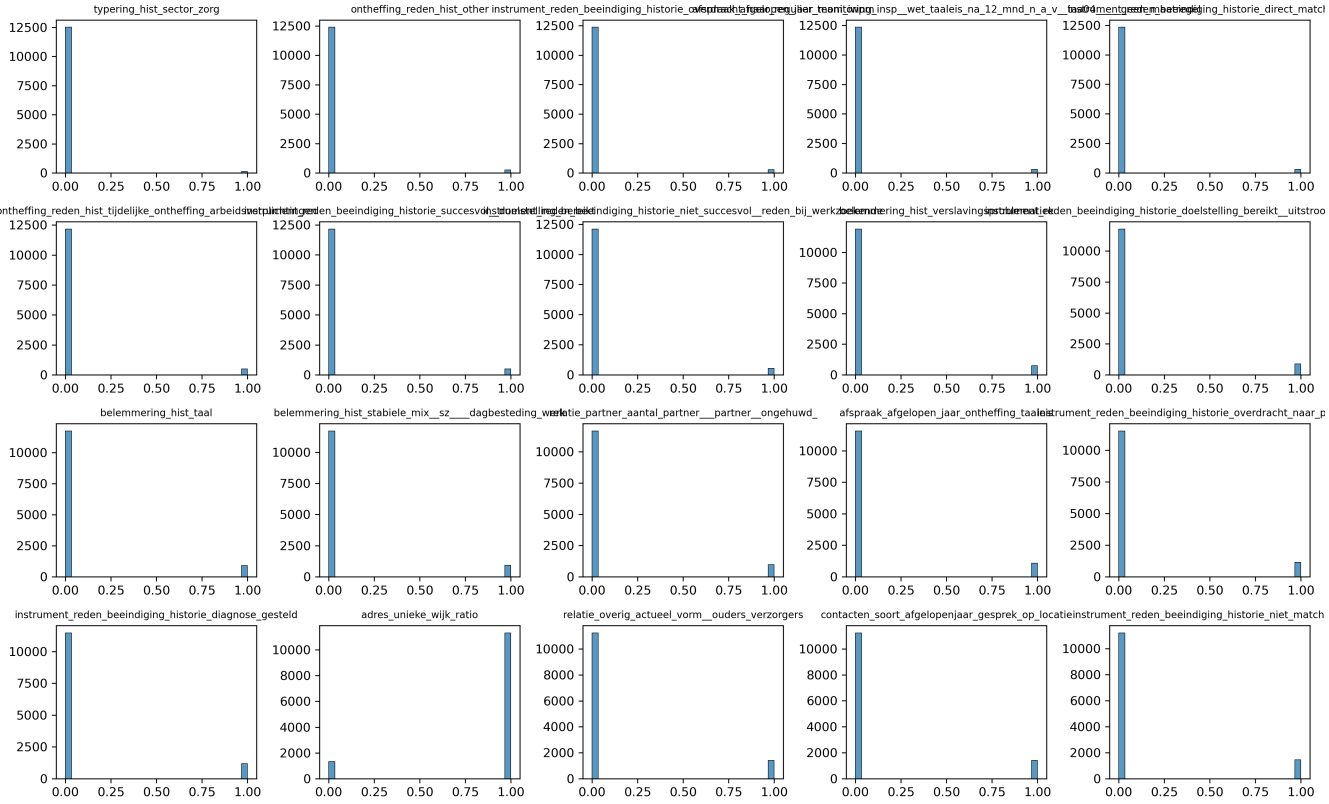


Figure 6: Histograms top 20 largest Skewness

5.3 Outlier detection

As noted previously, three main approaches were applied to detect outliers in the dataset. The first method was a univariate technique based on the interquartile range (IQR). This method uses the twenty-fifth and seventy-fifth percentiles to identify extreme values and visualize them using box plots. Since this method is only applicable to continuous variables, all boolean features were excluded from the analysis. For each numeric variable, the inner fences were calculated and the variables were ranked by the number of observations that fell outside those fences. Initially, the twenty fields with the highest outlier counts were plotted side by side for comparison. In every case, the twenty-five and seventy-five percentiles coincided, resulting in a single line for each box plot. The very large number of flagged observations suggested that these rare values nonetheless lay within the normal variability of the data. Manual inspection of the raw data confirmed this interpretation. Attention then shifted to the twenty fields with the fewest outliers which resulted in twenty fields with no outliers. The code was adjusted such that the twenty fields with at least one outlier were plotted instead, and these plots are shown in Figure 8.

The second approach was based on Mahalanobis distance, a multivariate metric that considers the covariance structure of the data to measure how far each observation lies from the multivariate center. Unlike the IQR method, which flags both low and high univariate outliers based on percentile thresholds, Mahalanobis distance does not distinguish between the direction of deviation. Instead, it considers the overall multivariate distance from the center of the distribution. As such, observations with unusually low values across multiple dimensions may still be classified as outliers if they lie far from the joint centroid, even if no single variable is extreme in isolation. Figure 9 displays these distances with a threshold set at the ninety-seventh point five percentile. Even with this threshold, many observations exceed the limit. Increasing the threshold to the ninety-ninth point nine percentile still yields 645 outliers. Mahalanobis distance was selected as a multivariate outlier detection method due to its ability to account for the covariance structure of the data. Unlike univariate approaches, it identifies points that are distant from the center of the data cloud when considering all features simultaneously. This is particularly suitable for the current dataset, which contains a large number of interrelated features. As highlighted in prior work, such

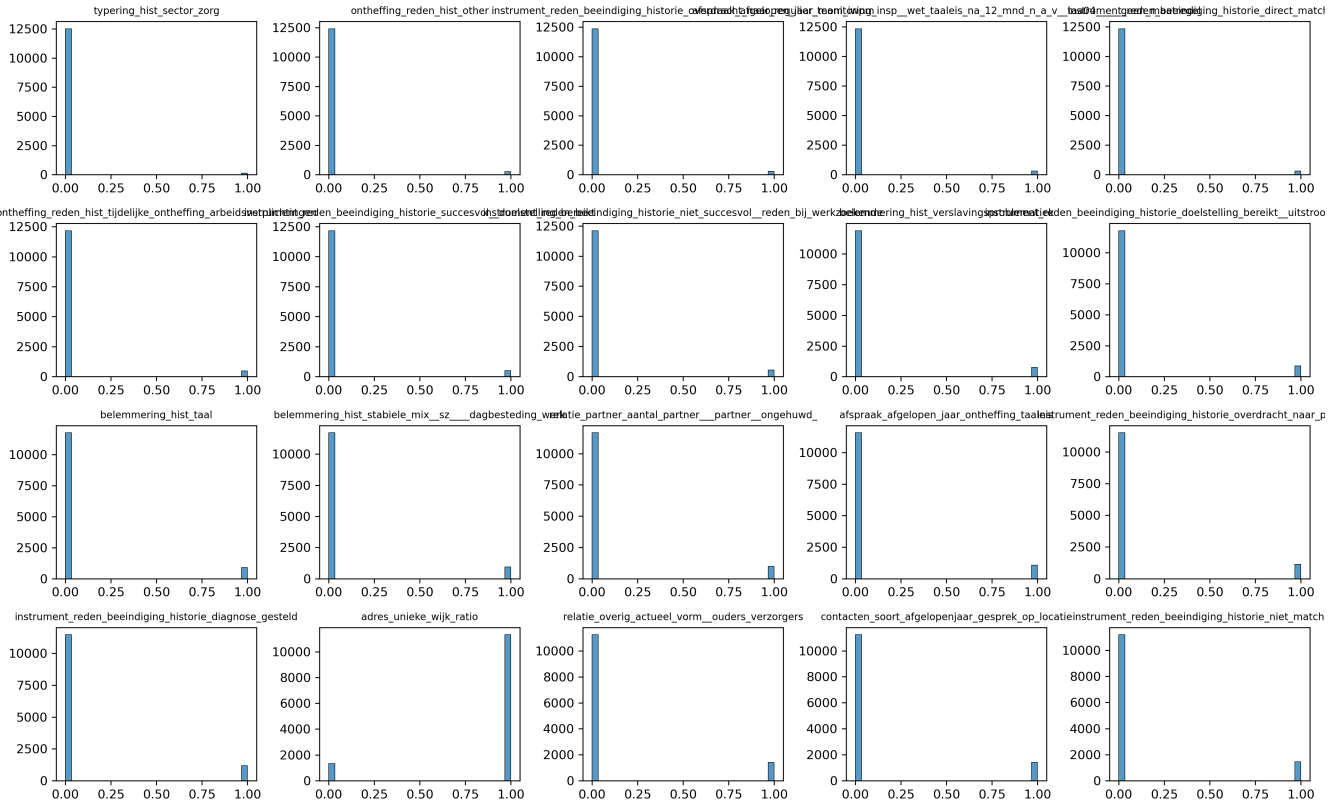


Figure 7: Histograms top 20 largest Kurtosis

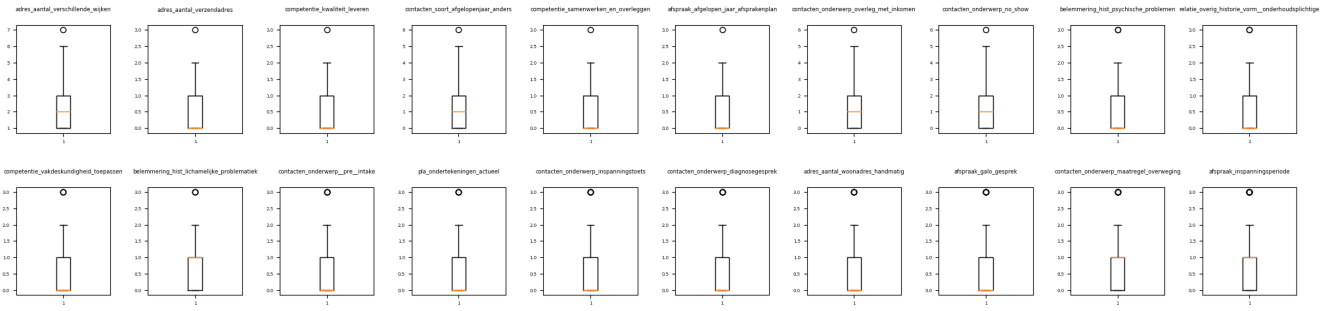


Figure 8: Boxplots bottom 20 IQL outliers

as in biomedical contexts, Mahalanobis distance is effective in screening for atypical patterns in high-dimensional spaces because it adjusts for scale and correlation among variables **?**. This makes it well-suited for scenarios where traditional distance metrics sometimes fail to capture multivariate extremity.

An extension to this method is to use a dynamic cutoff reffered to as the Mahalanobis elbow. This is based on an inflection point which represents the transition of regular and extreme outlier observations. As shown in Figure 10 this method resulted in the identification of 898 outliers where the threshold was set at 92.6%.

The 898 outliers identified through the Mahalanobis elbow method were flagged for later use. In the model phase, two different strategies were applied to account for these values and in the section about the output phase the results of these strategies will be presented.

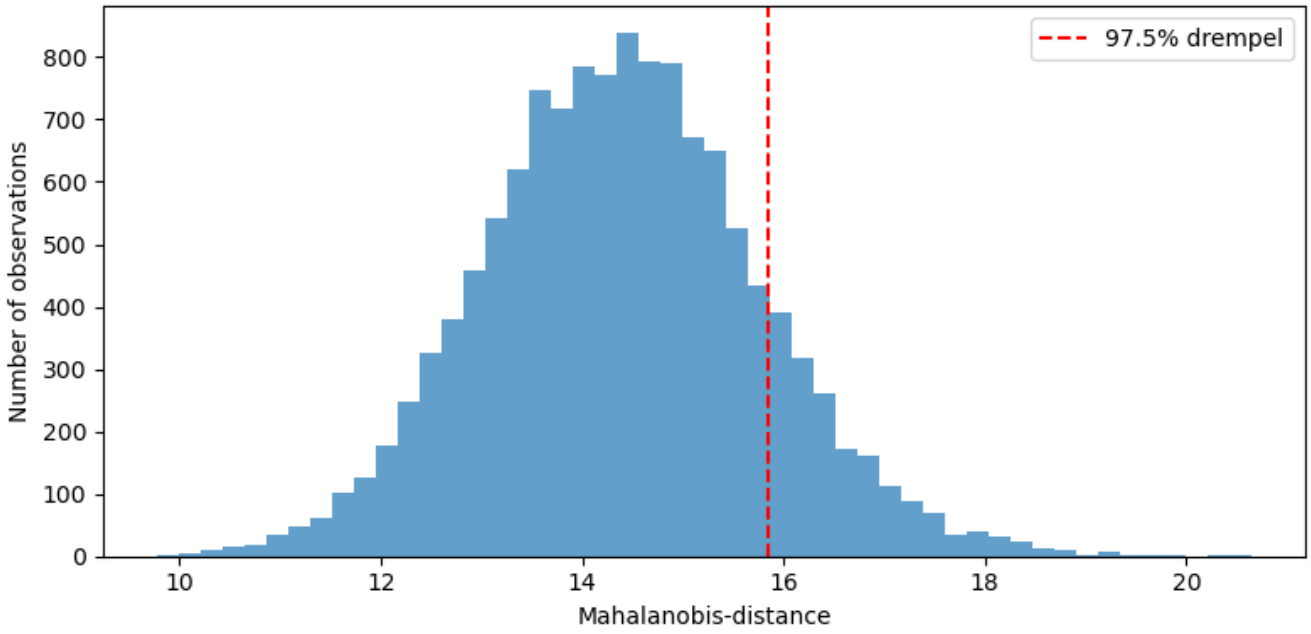


Figure 9: Mahalanobis Distance Outlier Detection 97.5% threshold

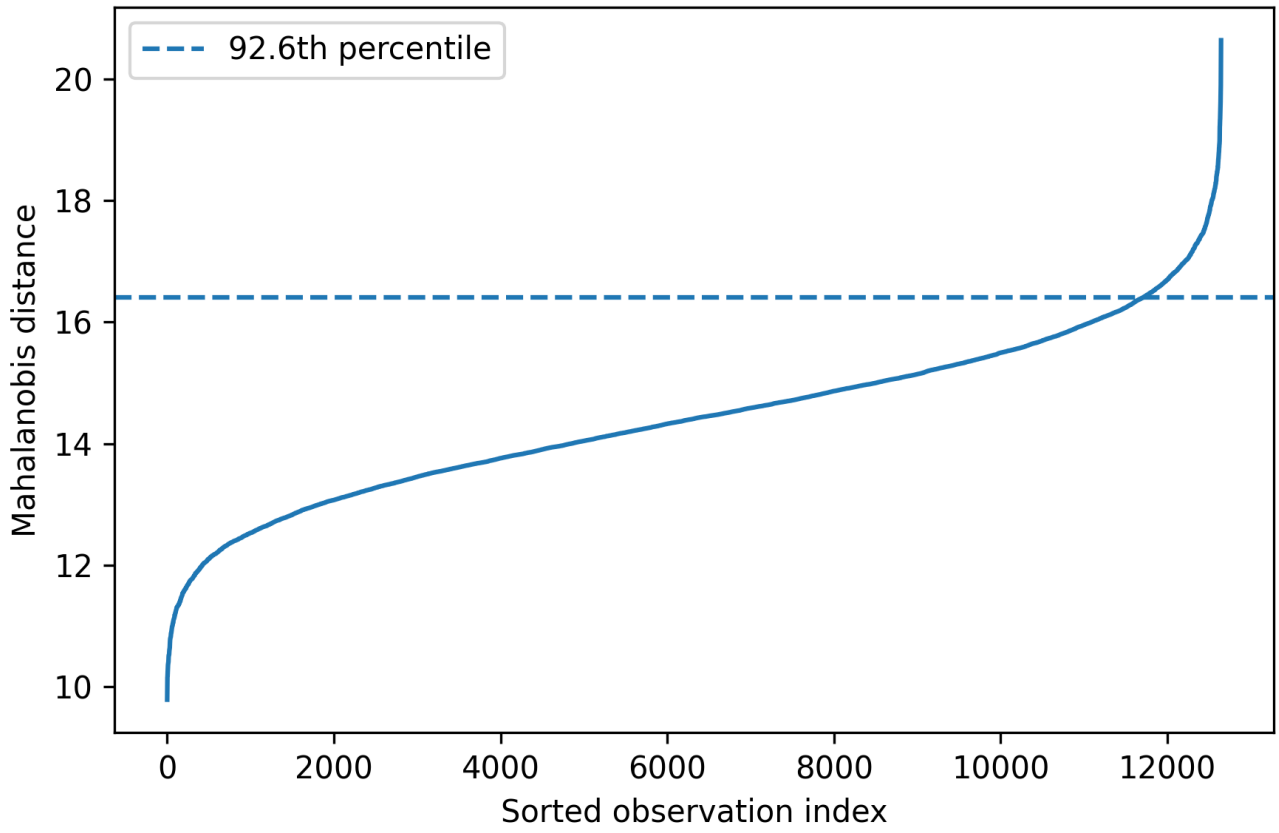


Figure 10: Mahalanobis elbow

5.4 Principal Component Analysis

Principal Component Analysis (PCA) is a mathematical technique used to reduce the number of variables in a dataset while preserving as much of the original variation as possible. It transforms the original features into a new

set of uncorrelated variables, known as principal components, which are ordered based on the amount of variance they explain. This allows complex, high-dimensional data to be represented in a more compact and interpretable form. PCA does not rely on any target variable, making it especially useful during exploratory data analysis. It helps to identify structure in the data, visualize hidden clusters, and prepare inputs for machine learning models.

In this project, PCA was applied to reduce the dimensionality of the dataset and to explore its underlying structure. The initial feature set consisted of over 300 variables, and many of which were not suitable for inclusion in PCA without preprocessing. Each variable was evaluated based on key criteria such as data type, variance, and distribution shape. To visualize this for each variable , box plots and histograms were created to get a quick overview.
To ensure meaningful input for PCA, several selection criteria were applied. Firstly, variables with near-zero variance (below 0.01) were excluded, as these features do not contribute to variation across individuals and are therefore uninformative in the context of principal component extraction. Secondly, skewness was examined to assess distribution shape. Variables with an absolute skewness greater than 1.0 were considered highly skewed. However, it was observed that many of these variables had only two unique values, which explained the extreme skew. Since such variables essentially behave as binary indicators, they were excluded from log-transformation. Only skewed variables with more than two unique values were log-transformed using the natural logarithm of $(1 + x)$ to reduce long-tailed effects while preserving interpretability. Boolean variables were systematically excluded from the PCA, as their binary nature limits variance and violates PCA's assumptions of continuous input. Categorical variables were evaluated separately, for each categorical feature a decision was made based on the number and frequency of its categories. After this cleaning and transformation process, there were 156 remaining variables.

To further reduce redundancy, a correlation matrix was computed across all scaled numerical variables. Variables with a Pearson correlation coefficient greater than 0.9 were flagged as potentially redundant, but no pair of variables exceeded this threshold. As a result, no additional filtering was necessary, and the full set of 156 selected variables was retained for PCA.

After preprocessing and filtering, Principal Component Analysis was applied to the 156 selected variables. To determine how many principal components are useful for analysis, two approaches were considered. The first method was to display the cumulative explained variance plot which can be seen in Figure 11. This plot shows how much of the total variance is captured as the number of components increases. In this dataset, the variance is distributed relatively evenly across many dimensions, and the commonly used threshold of 80% explained variance is not reached until the 46^{th} component. This gradual accumulation suggests that no small subset of components dominates the overall variance. But retaining 46 components would be computationally inefficient and hinder interpretability, for this reason a second method was explored.
The scree plot, which is shown in Figure 12 visualizes the individual eigenvalues of the components in descending order and helps to identify the point where the explained variance begins to level off. In this case a clear elbow is visible at the sixth component, indicating that components beyond this point contribute only marginally to the total variance. Based on this elbow criterion, a dimensionality reduction to the first six principal components was chosen. This selection provides a good balance between reducing the number of variables and keeping important patterns in the data. It makes it easier to visualise and understand how risk is distributed across different groups.
To further assess whether a particular combination of principal components revealed a distinct structure among high-risk individuals, several scatterplots were created comparing PC1 with each of the following components (PC2 through PC6), shown in Figure 13. In each subplot, the top 10% of predicted risk scores are shown in red, while the remaining individuals are shown in blue. Interestingly, the overall shape and distribution remain relatively consistent across all comparisons, and this is also true for all the other available combinations of PC1 through PC6. This suggests that no single projection among these six principal components isolates high-risk individuals in a clearly separable cluster. Although the red points are relatively well spread throughout the plots, some concentration in specific areas is still observable. These denser regions suggest that high-risk individuals share certain feature patterns that are partially captured by the first principal components. However, the separation between high- and low-risk individuals remains gradual rather than clearly defined. This indicates that while PCA facilitates useful visualizations and may reveal some underlying structure, it does not produce sufficiently distinct boundaries to serve as a standalone method for risk-based clustering or threshold segmentation. Alternative approaches such as supervised classification or clustering algorithms on the original or reduced feature space may be more appropriate for that purpose.

To investigate whether specific individual features might show stronger separation patterns, a subset of variables that

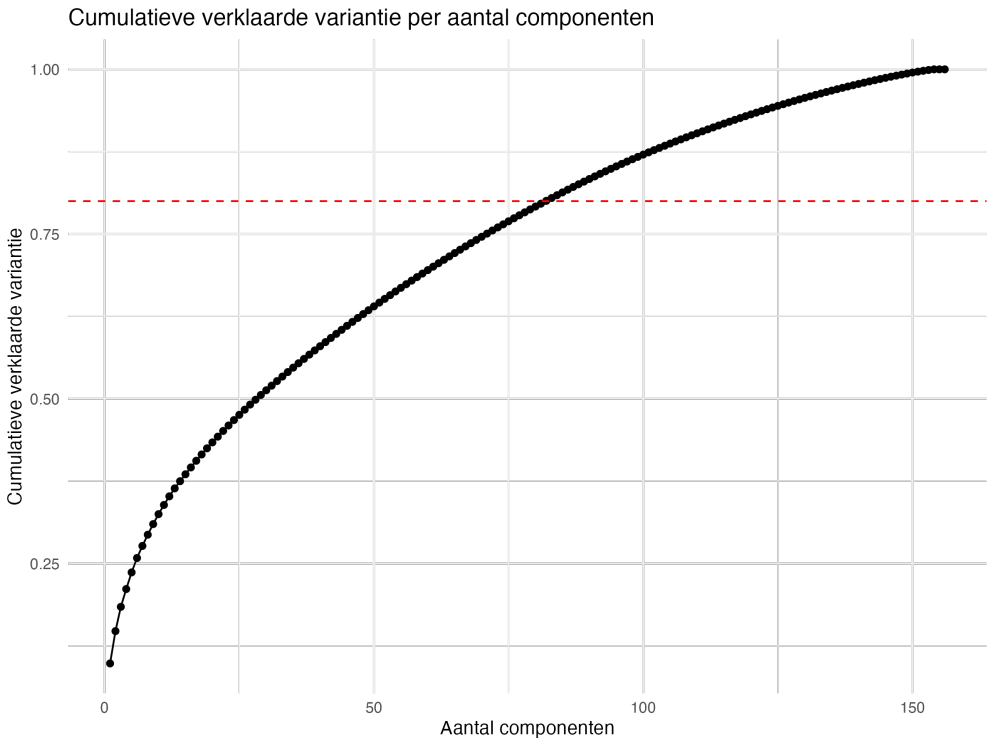


Figure 11: Cumulative explained variance of the principal components

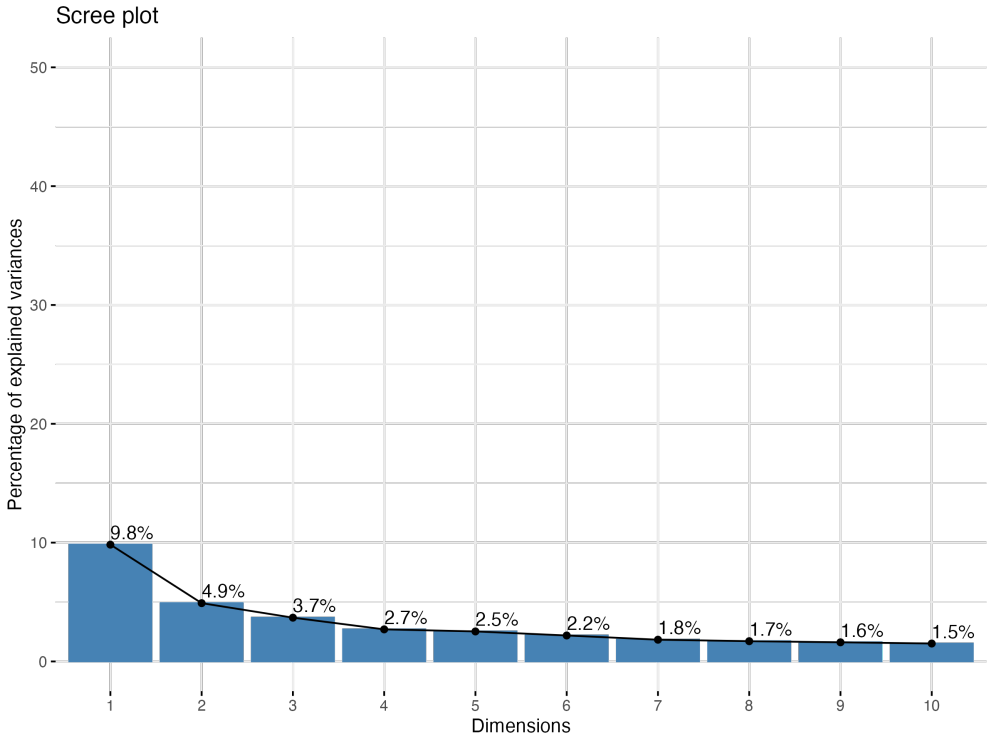


Figure 12: Scree plot of the PCA eigenvalues, showing a distinct elbow at the sixth component.

appeared frequently in archetype definitions or had potential social significance were selected for further analysis. For each of these variables, the distribution in PC1–PC2 space was visualized by coloring individuals based on their value for that variable. Examples include persoon_geslacht_vrouw (gender), relatie_kind_heeft_kinderen (having children),

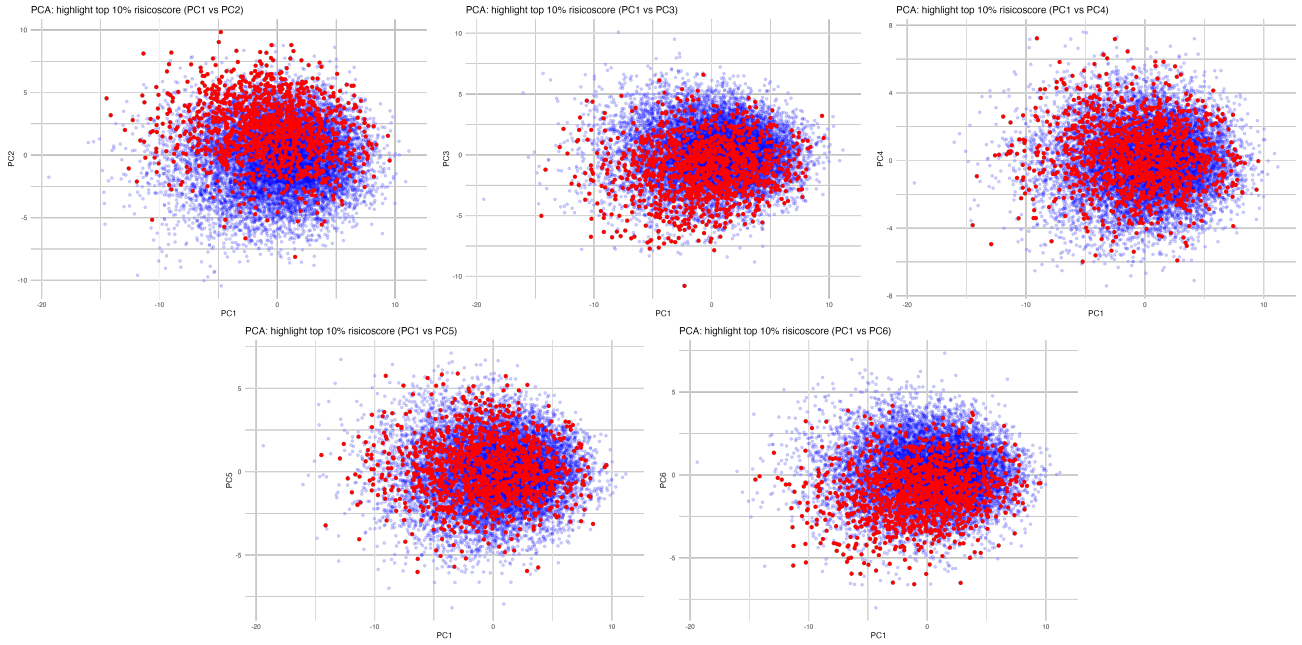


Figure 13: Scatterplots of PC1 against PC2 through PC6. Red dots indicate individuals in the top decile of risk scores.

adres_recentste_wijk_delfshaven (residing in a specific neighbourhood), and persoonlijke_eigenschappen_taaleis_voldaan (language requirement fulfilled). However, these visualizations revealed no clear clustering or separation either, and looking at other PC combinations this was neither the case. The data points remained densely intermixed across the component space, with only very minor tendencies toward concentration in certain areas. This suggests that even for salient socio-demographic variables, PCA does not produce a structure that enables reliable segmentation in reduced dimensions. The spatial overlap across groups highlights the complexity and entanglement of underlying patterns in the dataset.

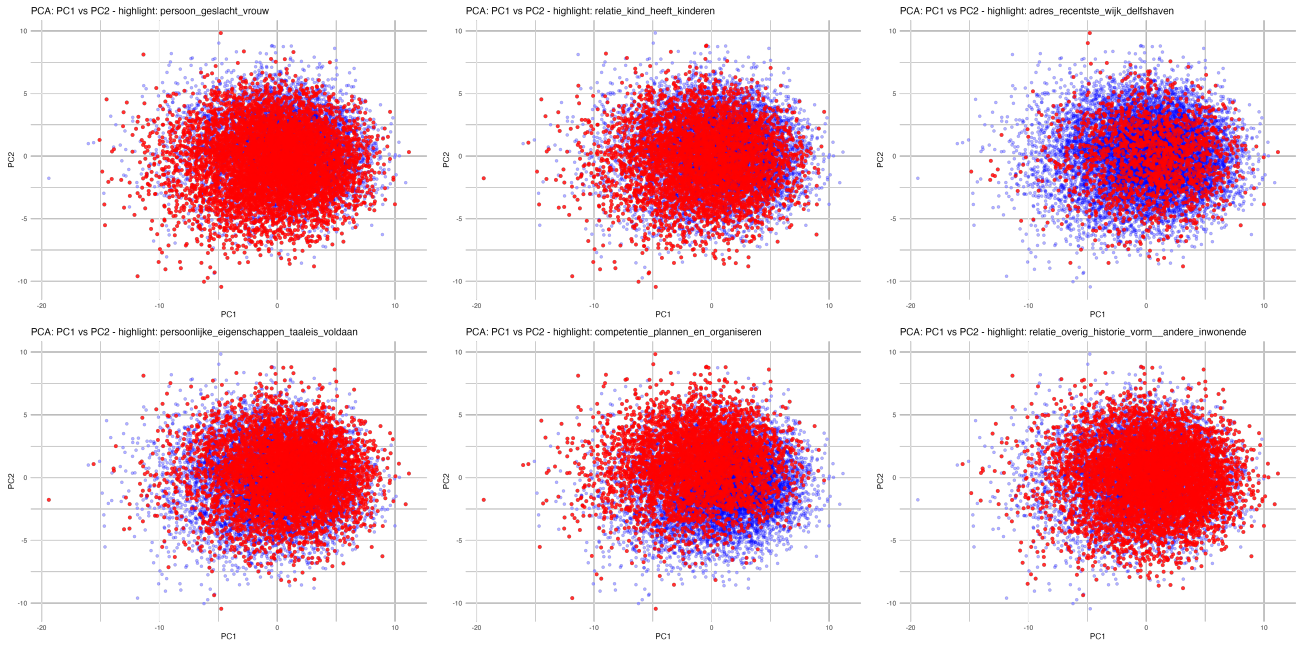


Figure 14: Selected variable overlays in PC1–PC2 space.

To better understand which features drive the structure captured by the first two principal components, a variable contribution plot was created Figure 15, and a zoomed in version that plots the variables of a threshold higher than 80 percent. This biplot visualizes the loading vectors of the original variables onto the PC1–PC2 plane. The direction and length of each arrow indicate both the sign and magnitude of a variable's contribution to the principal components.

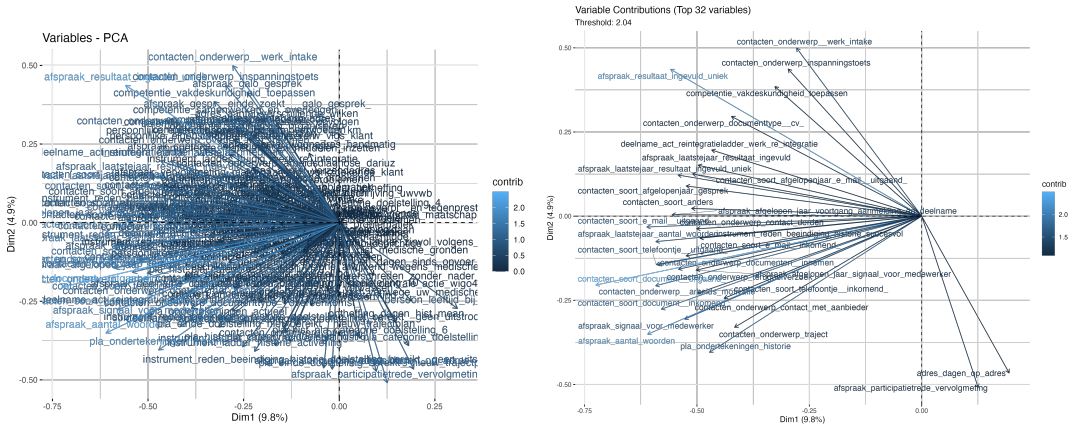


Figure 15: Variable contribution to PC1 and PC2. Arrow length indicates magnitude of contribution.

What stands out in these graphs is that many of the most influential variables point in a similar direction, toward the negative end of PC1, suggesting a latent factor that reflects a correlated behavioral or administrative pattern across multiple indicators. These include variables related to contact frequency, document handling, participation records, and appointment results.

Another aspect of interest in understanding the PCA structure is the contribution of variables to the second principal component (PC2), and inparticular the positive side. This dimension appeared to align with elevated predicted risk, since the top 10% of predicted risk scores were situated predominantly in the upper region of the PCA space. To better understand which variables contribute most to this pattern, a visualization of the top 20 variables with strong positive loadings on PC2 was created, as shown in Figure 16.

Nearly all of these variables have negative loadings on PC1, as shown by the diagonal upward-left orientation of the arrows. This suggests that individuals in the upper part of the PCA space are associated with a different set of variables than those contributing to PC1. While many of these are still administrative in nature, they reflect a more specific type of registration such as document types and appointment outcomes. There are also a lot of competency-related variables, such as plannen_en_organiseren and kwaliteit_leveren, which are very subjective variables filled in by case-workers.

While these associations are observational and do not imply causation, they offer valuable insight into the latent structure captured by PCA. In the following modeling phase, we will revisit these variables to examine their direct role in model predictions and to evaluate whether their influence contributes to unfair outcomes or reflects meaningful risk indicators.

5.5 Clustering

Since the PCA results did not reveal clearly separable groups of high– and low–risk individuals, alternative unsupervised techniques were explored to examine whether underlying structures could be identified using the same 156 variables selected for the PCA analysis. Two commonly used methods were chosen for this purpose: k-means clustering, which partitions the data into k groups by minimising within-cluster variance, and hierarchical clustering, which organises the data into a tree-like structure based on similarity. Both clustering analysis were carried out using the scores of the first six principal components derived from the 156 preprocessed variables. This approach condenses the information from the full feature set into a smaller number of orthogonal dimensions that capture the most relevant variance, reducing the influence of noise and multicollinearity. By clustering in this reduced space, the analysis emphasises the dominant patterns in the data while preserving the overall relationships identified during

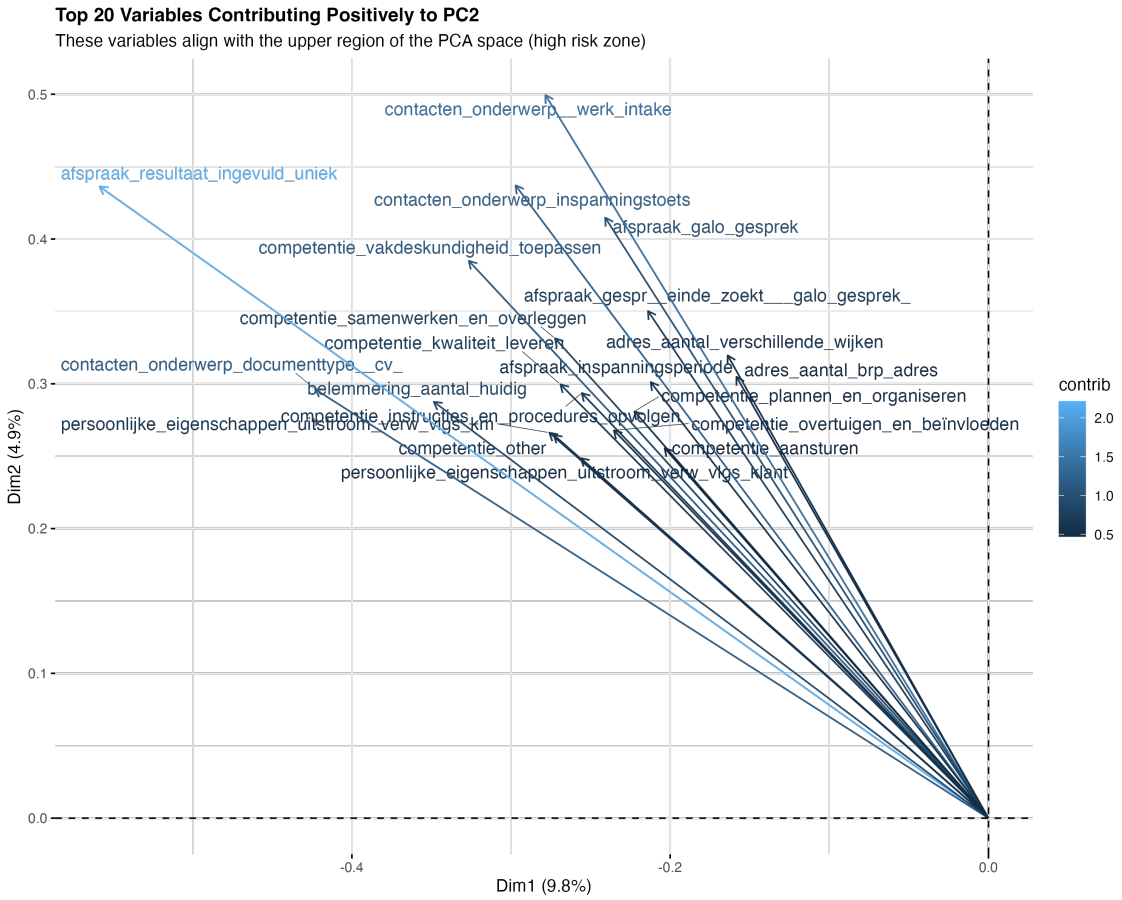


Figure 16: Top variables contributing positively to PC2, which aligns with the upper region of the PCA space (high-risk cluster).

the PCA stage.

The number of clusters k for k–means was determined using the elbow method, which evaluates the total within–cluster sum of squares for increasing values of k. In both the PC1–PC2 space and the PC1–PC6 space, a distinct inflection point appeared at k = 4, after which the marginal gain from adding more clusters diminished substantially which can be seen in Figure 17. For this reason, four clusters were selected as the optimal balance between complexity and interpretability.

The k-means solution in PC1–PC2 space produced clusters with sharply defined boundaries. This outcome reflects the algorithm's design, which enforces hard separations by assigning every observation to the single nearest cluster centre. Such sharply delineated groups are unlikely to occur naturally in this dataset, making the resulting partitions more reflective of the method's constraints than of inherent structures in the data. The resulting allocation is shown in Figure 18, which visualises the four k-means clusters in the PC1–PC2 plane for the different sets of features.

To address this limitation, hierarchical clustering was applied using Ward's minimum variance method. This approach builds a dendrogram in which observations are progressively merged into larger clusters based on similarity. However, in this case the resulting dendrogram becomes increasingly complex with the number of observations and features, which makes direct interpretation challenging. For this reason, Figure 19 focuses on the four-cluster cut, providing a more accessible summary of the results and a clearer comparison with the k-means solution. In contrast to the rigid partitions from k–means, hierarchical clustering revealed overlaps and transitional regions between clusters, suggesting that the underlying structure is less discrete than k–means implies.

The hierarchical clustering results differ substantially depending on the number of features included. With the full set of 156 features, the clusters appear relatively compact and distinct, especially for the green and blue groups,

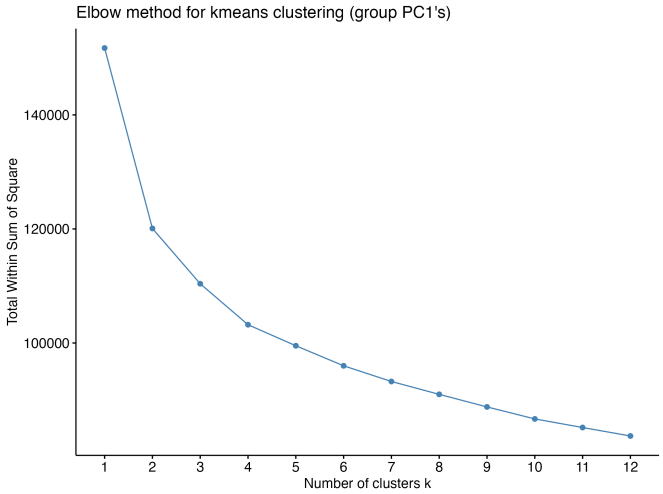
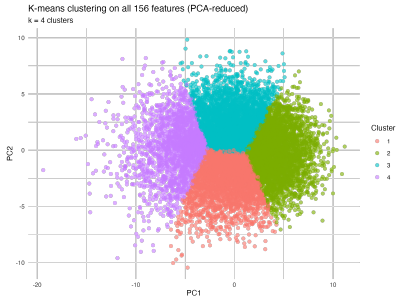
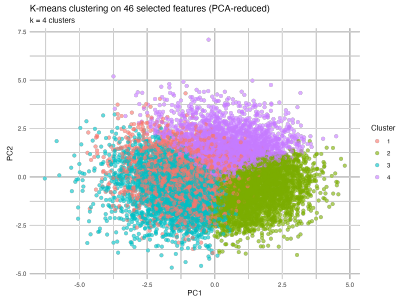


Figure 17: Elbow method for k-means clustering, indicating an optimal choice around $k = 4$.



(a) K-means clustering (k=4) on 156 features using PCA-reduced dimensions



(b) K-means clustering (k=4) on 46 features using PCA-reduced dimensions

(c) K-means clustering (k=4) on 156 features using PCA-reduced dimensions

Figure 18: K-means clustering (k=4) on different sets of features using PCA-reduced dimensions

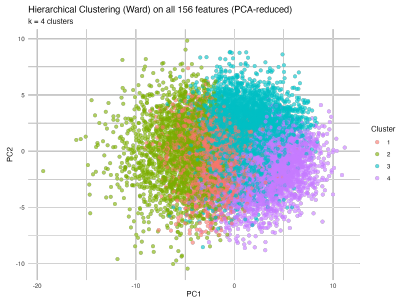
which occupy clearly separated areas. This indicates that the complete feature space contains enough variation to support meaningful group separation.

When the feature set is reduced to 58 variables, the cluster boundaries become less pronounced. Some groups remain visible, but the clusters overlap to a large extent. This suggests that important information for separating individuals has been removed during the feature selection process.

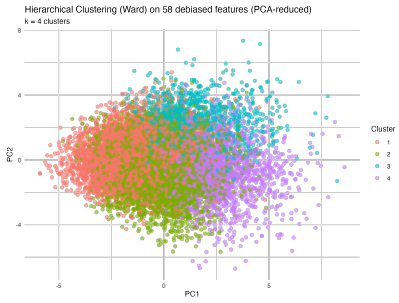
The results with 46 features show almost no clear separation at all. The clusters overlap heavily, and the colors are dispersed across the entire plot. In this case, the reduced feature space does not provide sufficient information for the hierarchical algorithm to identify distinct groups.

Overall, the analysis shows that the performance of hierarchical clustering is highly sensitive to the number and type of features included. A larger set of variables preserves more information about underlying differences, while aggressive reduction of features leads to overlapping and less interpretable clusters.

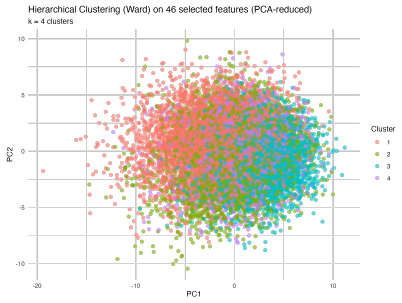
In Figure 20 the comparison between k-means and hierarchical clustering across the three feature sets can be seen and it highlights clear differences in the way both methods divide the data. K-means enforces compact and well-bounded clusters, which is most visible in the 156-feature case where the groups occupy distinct regions. With fewer features the boundaries become less natural and the overlap increases, but the algorithm still forces each observation into one of the four groups. Hierarchical clustering, in contrast, produces smoother transitions and greater internal variation. Especially with 46 and 58 features, the groups overlap considerably and appear less compact. This indicates that hierarchical clustering is more sensitive to the loss of information caused by feature



(a) Hierarchical clustering (k=4) on 156 features using PCA-reduced dimensions

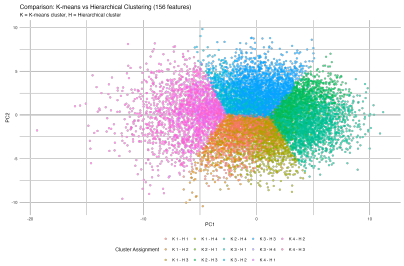


(b) Hierarchical clustering (k=4) on 58 features using PCA-reduced dimensions

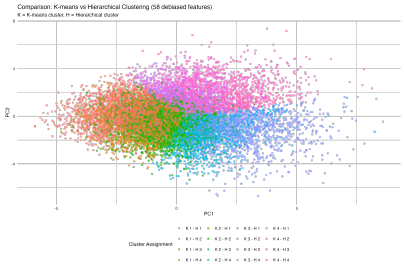


(c) Hierarchical clustering (k=4) on 46 features using PCA-reduced dimensions

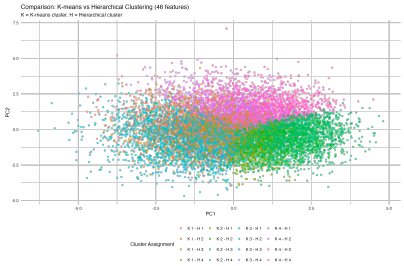
Figure 19: Hierarchical clustering (k=4) on different sets of features using PCA-reduced dimensions



(a) Comparison k-means and hierarchical clustering (k=4) on 156 features using PCA-reduced dimensions



(b) Comparison k-means and hierarchical clustering (k=4) on 58 features using PCA-reduced dimensions



(c) Comparison k-means and hierarchical clustering (k=4) on 46 features using PCA-reduced dimensions

Figure 20: Comparison k-means and hierarchical clustering (k=4) on different sets of features using PCA-reduced dimensions

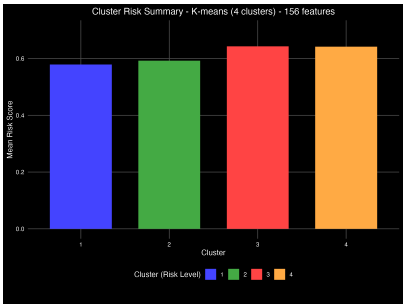
reduction, while k-means continues to create visually distinct partitions even when the natural structure is weaker.

To assess whether clustering provides additional insight into the allocation of risk, the average model-assigned risk scores were evaluated across the four clusters obtained by k-means and hierarchical clustering using different feature subsets (46, 58, and 156 features). The purpose of this analysis was to test whether the clustering procedure produced groups with systematically higher or lower risk levels, which could indicate hidden segmentation in the data.

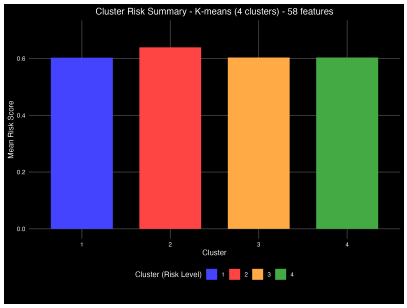
Figure 21 displays the mean risk scores per cluster for k-means clustering and Figure 22 for hierarchical. Across all specifications, the observed differences between clusters are minimal. For example, in the 156-feature models the highest mean risk is 0.643 (Cluster 3, k-means) compared to the lowest mean risk of 0.579 (Cluster 1), resulting in a range of only 0.064. Similar patterns appear for the hierarchical clustering, where the spread between clusters does not exceed 0.062. The same holds for the reduced feature sets (46 and 58 features), where ranges fall below 0.04. These values indicate very low dispersion, suggesting that the clusters are not differentiated by risk levels.

The analysis of the top decile further supports this conclusion. In a fair distribution, each cluster would be expected to account for approximately 10% of the individuals in the highest risk category. Instead, all clusters contributed between 1.6% and 3.9%, far below the expected baseline and with only minor differences across groups. This indicates that no cluster is systematically overrepresented among the highest-risk cases.

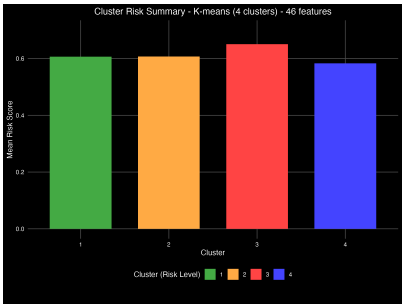
Overall, the findings demonstrate that the clustering procedure did not reveal groups with distinct risk profiles. Clusters are primarily separated by other structural features in the dataset rather than by fraud risk levels. The small observed differences are unlikely to be practically significant, implying that clustering is not a useful tool to identify high-risk subpopulations in this context. While this does not imply that the model is entirely fair, it indicates that fairness issues are unlikely to originate from hidden substructures in the clustered feature space.



(a) Cluster risk scores (k=4) from k-means clustering on 156 features.

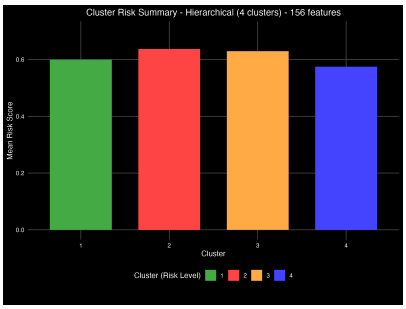


(b) Cluster risk scores (k=4) from k-means clustering on 58 features.

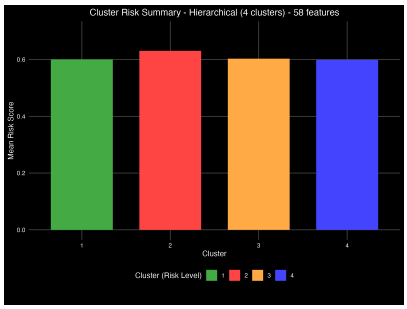


(c) Cluster risk scores (k=4) from k-means clustering on 46 features.

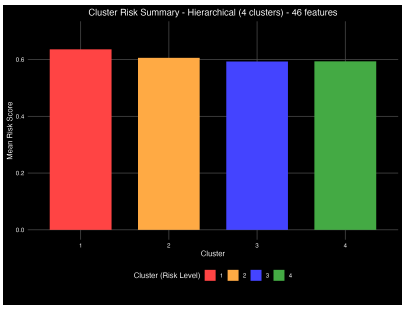
Figure 21: Cluster risk scores (k=4) from k-means clustering on 156, 46, and 58 features.



(a) Cluster risk scores (k=4) from hierarchical clustering on 156 features.



(b) Cluster risk scores (k=4) from hierarchical clustering on 58 features.



(c) Cluster risk scores (k=4) from hierarchical clustering on 46 features.

Figure 22: Cluster risk scores (k=4) from hierarchical clustering on 156, 46, and 58 features.

6 Model Phase

Following the input phase analysis, which included outlier detection, distribution checks, principal component analysis, and multicollinearity assessment, multiple retraining strategies were implemented to address the identified concerns. The purpose of this section is to describe the motivation behind each modeling approach and the methodological adjustments applied.

6.1 Clean model

In the model phase, two separate strategies were implemented to assess how outliers influence the risk scoring process. Both strategies are based on the set of 898 outliers identified earlier using the Mahalanobis elbow method.

The first strategy involved removing all identified outliers from the dataset. A new model was then trained using only the remaining observations. This model was trained in the same way as the original gradient boosting machine, using the same parameters and structure, but applied to the cleaned data. The resulting risk scores reflect how the model performs when extreme observations are excluded from the training process. But since the removal of these outliers also removes information that may be useful a second adjustment to the model was made based on these 898 outliers.

6.2 Weighted model

The second model adjustment retained all observations including the outliers, but adjusted the influence of these outliers by assigning them a lower sample weight. As a result, observations with a high Mahalanobis distance received a lower weight, while those closer to the center of the data received a weight closer to 1. This approach preserves the full dataset but reduces the impact of extreme values during model training.

To determine the most suitable weighting strategy for handling outliers, three different methods were compared using Ridge regression combined with five-fold cross-validation. The tested strategies included a normalised linear weight function $1 - \frac{\text{MD}}{\max(\text{MD})}$ where MD cannot be zero, an exponential function $\exp(-2 \cdot \text{MD})$, and a logistic function. The performance of each method was evaluated based on the mean squared error (MSE) of the model.

Ridge regression was selected for this comparison because it is a linear model that includes regularization. This regularization limits the size of the model coefficients, which helps to avoid overfitting on the training set. By doing so, the model is more likely to generalize well to new, unseen data. Although the final models in this study are based on gradient boosting, Ridge regression provides a simple and consistent framework for testing the effect of different weighting strategies. Its interpretability and stability make it well suited for comparing performance across conditions before applying the selected strategy to more complex models.
The five-fold cross-validation technique splits the dataset into five equally sized parts. In each round, four parts are used for training and one for validation. This process is repeated five times so that each part is used as a validation set once. The average MSE across all folds provides a reliable estimate of model performance. Five-fold cross-validation was chosen because it provides a good balance between computational efficiency and reliability of the performance estimate.

As shown in Figure 23, the linear weighting method yielded the lowest average MSE. For this reason, the linear method was selected for use in the final weighted model.

6.3 Feature-Selected Models

The first two models still used the full set of variables, but to explore whether a reduction in the number of features could improve fairness or interpretability, alternative models were developed using smaller feature subsets. After performing principal component analysis, three selection strategies were applied.

The first model was trained on the 156 features obtained after the preprocessing and filtering steps described in the Input Phase. These features were selected from the original 314 variables by removing those with near-zero variance, applying log-transformations to highly skewed continuous variables, excluding Boolean variables, and retaining only categorical features with meaningful category distributions. A check for multicollinearity confirmed

Figure 23: Comparison of weighting strategies using Ridge regression and five-fold cross-validation. The red line indicates the MSE of the unweighted baseline model.

that no variables exceeded a Pearson correlation of 0.9, allowing all 156 to be retained.

Although this represented a substantial reduction from the original dataset, 156 features is still a large number. As explained in the Input Phase, the cumulative variance plot from the principal component analysis showed that approximately 80 percent of the total variance could be explained by the top 46 variables with the highest contributions. This suggested that a smaller set of features might capture most of the relevant structure in the data while potentially improving interpretability and reducing bias.

To test this, two additional models were developed using smaller feature subsets based on two selection strategies. The first subset consisted of the 46 variables with the highest combined loading contributions to the first six principal components, identified by summing the absolute loadings per variable across these components and ranking them from highest to lowest.

The second subset was created by examining the predefined archetypes and their over-representation in the top ten percent of predicted risk scores. For each feature, the correlation with this over-representation metric was calculated. Features with an absolute correlation above 0.1 were considered to have a stronger association with disproportionate risk allocation and were excluded. The remaining 58 features, which all fell below this threshold, formed the second reduced set. This approach aimed to retain variables with predictive utility while removing those most linked to biased outcomes.

All three new models, the full 156 features, the 46 most influential features, and the 58 least bias-correlated features, were trained using the same gradient boosting framework as the other experiments, ensuring that any differences in predictive or fairness outcomes could be attributed solely to the choice of input features.

7 Output Phase

The output of the model is expressed as a risk score for each individual in the dataset. These scores are produced by a gradient boosting machine and reflect the aggregated decisions of multiple decision trees. In each tree, observations are directed through a series of binary splits, and their final position in the tree influences the assigned score. Individuals that consistently end up in high-risk leaves across trees receive higher final scores. These scores serve as a proxy for predicted fraud risk, even though no actual fraud labels are present in the data.

To evaluate the performance and fairness of the adjusted models, the average risk scores for each archetype were compared across the different versions that were presented in Section 6.

The total number of observations found per archetype in the dataset are summarised in Figure 24.

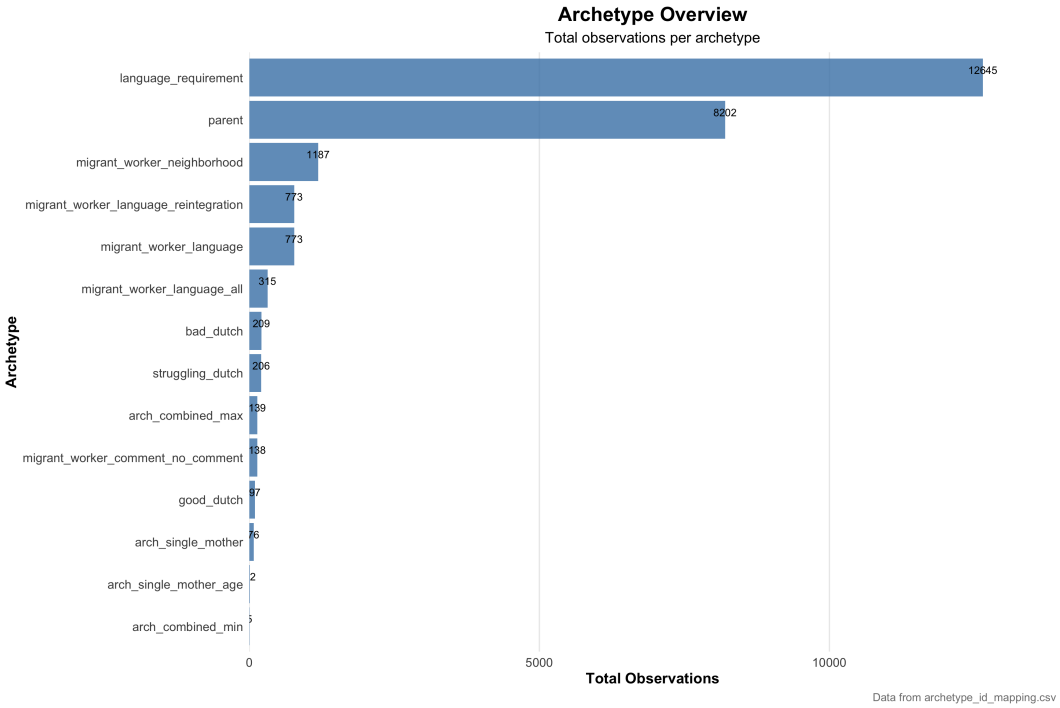


Figure 24: Barplot total number of archetypes found in the dataset.

7.1 Removal and Reweighting of Outliers

Bias in this study was measured as the relative over-representation of individuals from a specific archetype in the top ten percent of predicted risk scores. In a fair model, each archetype is expected to appear in the top decile at roughly the same rate—around 10% unless otherwise specified. For each model and archetype, the proportion of individuals falling into the top decile was calculated and compared to this baseline. The resulting difference is referred to as the excess share. For example, if 20% of individuals from the archetype *migrant_worker_language_all* appeared in the top decile, this corresponds to an excess of +10%. A negative value would indicate under-representation.

This metric was computed for all archetypes across the original, clean (outliers removed), and weighted (reweighted outliers) models. The values were saved as summary statistics and served as input for statistical comparison.

To test whether differences in bias between models were statistically significant, independent t-tests were performed. Mathematically, a t-test compares the means of two groups, which in this case were the excess shares from the original model and those from the clean or weighted model. It calculates the difference in means and divides it by the standard error, resulting in a t-value that reflects how large the observed difference is relative to the variability in the data. A corresponding p-value is computed to estimate the likelihood of observing such a difference if no real effect is present. A p-value below 0.05 suggests that the observed change is unlikely to be due to random variation

and can be considered statistically significant. These p-values were used to identify which archetypes experienced meaningful changes in bias across models.

The outputs of the three models are compared in Table 1.

archetype	archetypes	clean	weighted
arch_combined_max	14.855674	6.63898774	6.44918940
arch_combined_min	-8.631870	-7.61170423	-7.71451166
arch_single_mother	-2.779121	-2.93352255	-2.99448353
arch_single_mother_age	4.342821	4.54535764	3.01034005
good_dutch	-7.277742	-4.65941797	-4.41957147
language_requirement	4.710585	4.69059802	1.85552358
migrant_worker_comment_no_comment	10.106762	3.72874654	3.84341637
migrant_worker_language	2.285488	0.69988138	0.80664294
migrant_worker_language_all	9.944642	3.64175563	3.75247133
migrant_worker_language_reintegration	2.285488	0.69988138	0.80664294
migrant_worker_neighborhood	1.264893	0.01953682	-0.09505822
parent	-4.423519	-4.41757903	-4.34798510
struggling_dutch	3.543332	1.93636073	2.06827630

Table 1: Average bias per archetype for each model, measured as excess representation in the top decile

In addition to average risk scores, the over-representation of each archetype in the highest-risk group was examined. Figure 25 displays the excess share per archetype across the three models. Several archetypes that were strongly overrepresented in the original model, such as *migrant_worker_comment_no_comment*, *migrant_worker_language_all*, and *arch_combined_max*, show a notable decrease in top-decile presence. This suggests that the adjusted models are less biased in assigning high-risk classifications to these profiles.
For some archetypes, the direction of bias also changed. For example, *migrant_worker_neighborhood* receives a slightly negative average in the weighted model, indicating potential underestimation of risk. Similarly, *good_dutch* consistently receives negative scores, although the magnitude becomes smaller in the adjusted models.

These findings reflect not only shifts in average risk scores but also changes in how strongly certain patterns in the data influence model predictions. Archetypes that show large shifts in excess share often correspond to features with high decision impact within the model, suggesting their associated variables played a dominant role in risk estimation.

Overall, both adjusted models show improved fairness compared to the original model, with most archetypes showing values closer to zero. This indicates a reduction in bias in the output. Between the two, the weighted model retains more of the original data while still mitigating bias, and is therefore preferred for further analysis.

7.2 Feature Removal

Bias was again quantified as the relative over-representation of archetypes in the top ten percent of predicted risk scores. The expectation in a fair model remains that each archetype appears in this highest-risk decile at approximately equal proportions of 10%. Any deviation from this baseline was interpreted as excess share, with positive values denoting over-representation and negative values under-representation. This metric was applied across all clustering-based model variants, including those trained on the full 156 features, the 46 principal component–selected features, and the 58 least bias-correlated features. The results of this analysis are summarised in

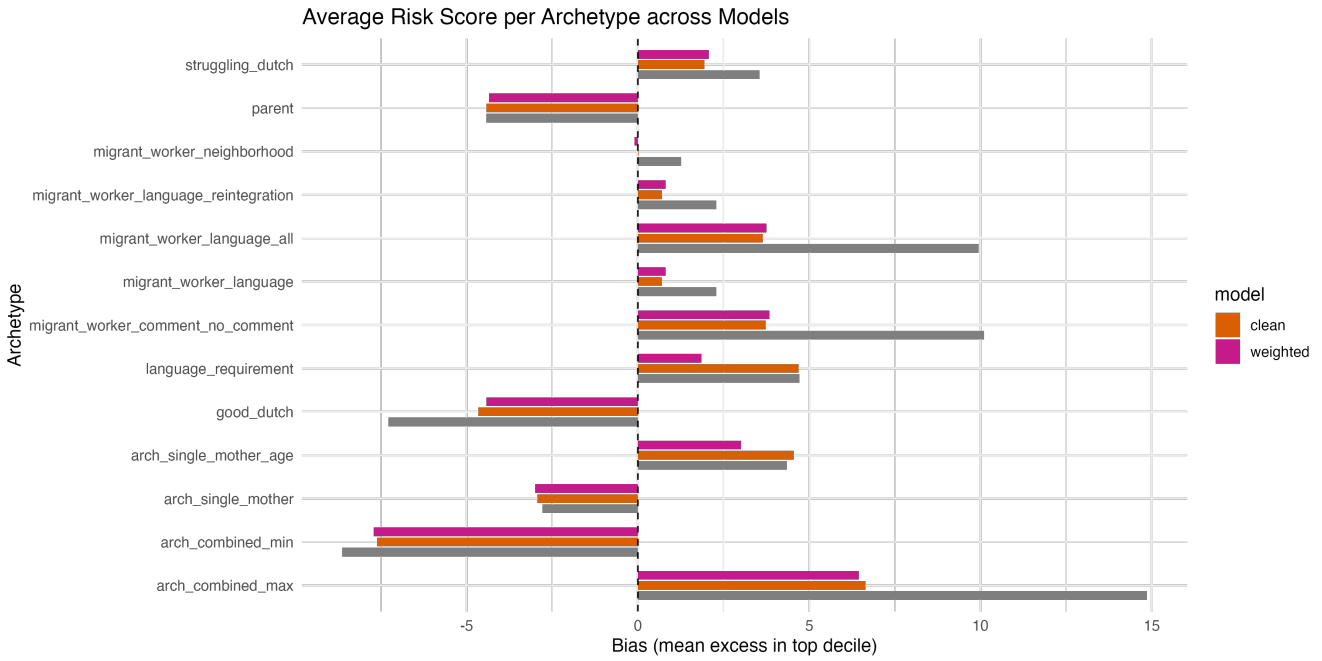


Figure 25: Bias per archetype across original, clean, and weighted models

Table 2, which reports the mean excess shares per archetype across the five model variants. This is done to make the comparison between the previous and new models more clear.

The three clustering-based models trained on 156, 46, and 58 features display clear differences in bias outcomes compared to both the original and weighted models. The 156-feature model amplifies bias substantially. For example, the archetype *arch_combined_max* shows an excess share of 14.3% compared to only 6.6% in the original model (Table 2). Similarly, *migrant_worker_language_all* and *migrant_worker_comment_no_comment* increase from 3.6% and 3.7% in the original model to 17.4% in the PCA model. This confirms the strong upward shift in average excess shares visible in Figure 26.

Reducing the feature set to 46 variables does not alleviate this issue. The 46-feature model produces almost identical outcomes to the 156-feature model, with excess shares for *arch_combined_max* and *migrant_worker_language_all* again reaching approximately 14–17%. The large spreads observed in Figure 27 further illustrate that this approach leads to wider variability and more extreme outliers than either the original or weighted models.

By contrast, the 58-feature model, which excluded variables most correlated with biased outcomes, produces a noticeable improvement. The excess share for *arch_combined_max* drops to 4.4%, while *migrant_worker_language_all* falls to 3.8%. Similarly, *migrant_worker_comment_no_comment* decreases from 17.4% in the PCA models to 3.9%. These values are much closer to the baseline expectation of zero but remain slightly above the original model for some archetypes (Table 2).

When compared to the weighted model, the superiority of the latter becomes clear. Weighted consistently reduces excess shares across all groups, often by several percentage points relative to both the original and clustering-based models (Figures 26 and 27). For instance, *arch_combined_max* is lowered to 6.5%, *migrant_worker_language_all* to 3.8%, and *migrant_worker_comment_no_comment* to 3.9%. Moreover, under-represented archetypes such as *good_dutch* remain closer to zero, with an excess share of -4.4% compared to -5.1% in the PCA-based models.
In general, the clustering-based models show that the choice of features and the use of debiasing methods strongly influence fairness. The PCA models concentrate variance but also strengthen existing distortions, which results in very high excess shares for certain archetypes. The debiased and especially the weighted model reduce these distortions, bringing the values closer to zero and lowering variability. Clustering itself is useful to explore underlying structures, but it does not improve fairness on its own – real progress requires targeted adjustments.

archetype	clean	pca	pca_46	pca_debiased	weighted
arch_combined_max	6.63898774	14.3131615	14.3131615	4.365704845	6.48082246
arch_combined_min	-7.61170423	-6.7305880	-6.7305880	4.365704845	-7.71451166
arch_single_mother	-2.93352255	-3.8439399	-3.8439399	0.001598977	-2.98848737
arch_single_mother_age	4.54535764	-0.3821554	-0.3821554	0.001598977	3.02099989
good_dutch	-4.65941797	-5.1311161	-5.1311161	-0.542053086	-4.41957147
language_requirement	4.69059802	6.1975447	6.1975447	8.334399318	1.85952102
migrant_worker_comment_no_comment	3.72874654	17.3603641	17.3603641	5.820981397	3.85527877
migrant_worker_language	0.69988138	6.1348715	6.1348715	4.365704845	0.82245947
migrant_worker_language_all	3.64175563	17.3603641	17.3603641	5.820981397	3.76037960
migrant_worker_language_reintegration	0.69988138	6.1348715	6.1348715	4.365704845	0.82245947
migrant_worker_neighborhood	0.01953682	2.4216825	2.4216825	2.465518103	-0.09154344
parent	-4.41757903	-5.6195597	-5.6195597	-4.510246444	-4.34665262
struggling_dutch	1.93636073	7.1250400	7.1250400	1.016949153	2.06827630

Table 2: Average bias per archetype for selected features models, measured as excess representation in the top decile

7.3 Adversarial Loss

To further mitigate potential bias in the risk predictions, an adversarial debiasing framework was applied. This method jointly trains a predictor and an adversary. The predictor estimates the target variable $\hat{Y}$ from the input features X, while the adversary attempts to infer the protected attribute Z (here represented by archetype groups) from the predicted output $\hat{Y}$. The predictor is penalised whenever the adversary succeeds, encouraging outputs that remain accurate for the main task but uninformative about sensitive attributes.

Since the true outcome labels for welfare fraud are not available in this case study, adversarial debiasing provides a practical option to evaluate and potentially reduce bias. Instead of relying on ground truth verification, the approach directly constrains the relationship between model predictions and protected attributes, allowing an assessment of fairness independent of unavailable real-world fraud data.

In this case study, adversarial debiasing was tested on three model variants: the original model with 156 features, a reduced model with 58 principal components, and a weighted model with 156 features. For each variant, different values of the regularisation parameter λ were explored to study the trade-off between predictive performance and fairness.

Figure 28 illustrates how different values of λ influence the trade-off between validation loss, adversary gap and fairness across all the models. Figure 28a shows the trade-off between λ and the validation loss, where lower values indicate better predictive performance. Figure 28b shows the trade-off between the mean absolute excess share and λ, where lower values indicate fairer outcomes across archetypes. Figure 28c shows the trade-off between the adversary gap and λ, where a smaller gap indicates the adversary cannot distinguish archetypes, which implies fairness. The curve shows that small increases in λ lead to improved fairness, as reflected by the adversary gap, but at the cost of higher validation loss. The chosen value $\lambda = 0.01$ represents the point where this balance was most favourable for the 156-features model. These outcomes show that the 156-features model sacrifices predictive accuracy, but it produces outputs that are less informative about protected groups.

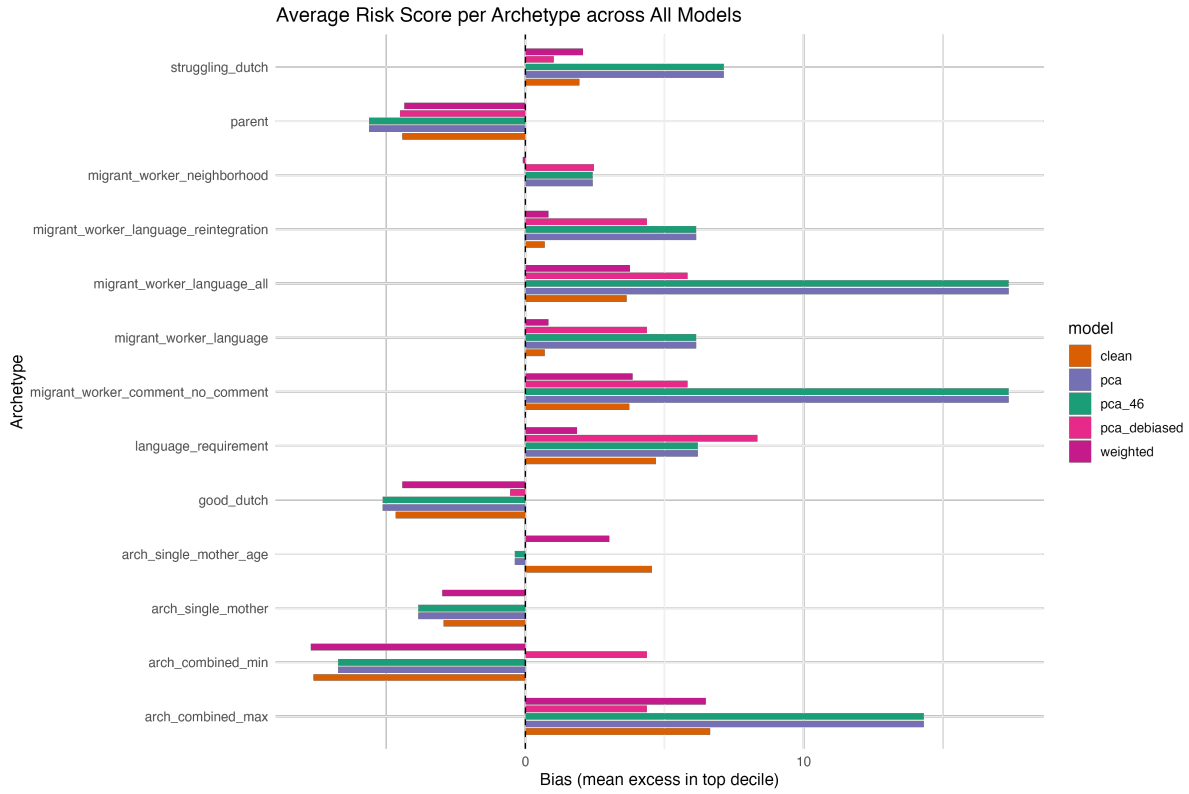


Figure 26: Bias per archetype across selected features models

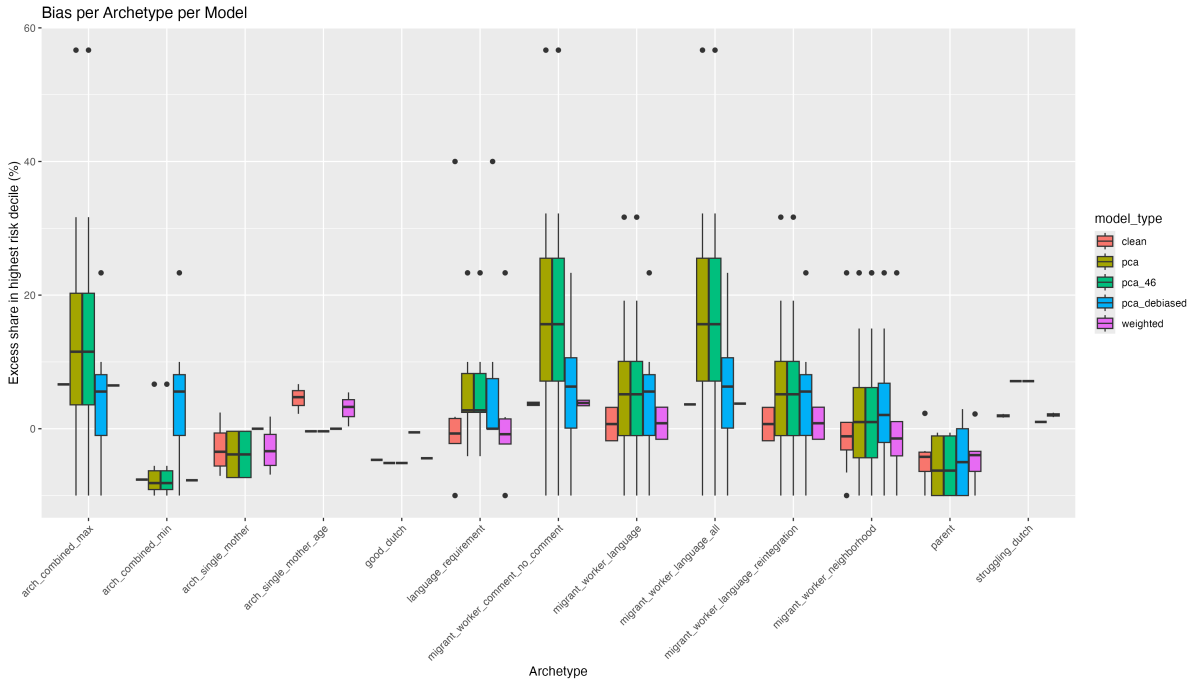


Figure 27: Distribution of excess share per archetype across selected features models

The adversarial debiasing experiments were first applied to the model trained on the full set of 156 features. This configuration was included to examine whether reducing the number of variables through dimensionality reduction actually improves performance and fairness.

For this model, the best trade-off was observed at a regularisation value of $\lambda = 0.01$. The parameter λ controls the weight given to the adversarial penalty during training: higher values place stronger emphasis on fairness, while lower values prioritise predictive accuracy. In this case, $\lambda = 0.01$ achieved the second-best overall ranking across all tested settings.

The following results were observed:

- Validation loss (73.4): the prediction error on held-out data. A lower value indicates better generalisability of the model.
- Adversary gap (0.064): a measure of how well the adversary can infer the protected attribute from the model's predictions. Values closer to 0.5 indicate stronger fairness, as they imply that the predictions contain little information about archetype membership.

The curve in Figure 28 shows that small increases in λ lead to improved fairness, as reflected by the adversary gap, but at the cost of higher validation loss. The chosen value $\lambda = 0.01$ represents the point where this balance was most favourable for the 156-features model. These outcomes show that the 156-features model sacrifices predictive accuracy, but it produces outputs that are less informative about protected groups.

For the PCA-58 model, the chosen value $\lambda = 0.005$ resulted in a validation loss of 85.7 and an adversary gap of 0.072. Compared to the 156-features model, this reduced-feature approach achieved only limited improvements in fairness, while predictive accuracy decreased more sharply. This suggests that dimensionality reduction did not enhance the effectiveness of adversarial training in reducing bias.

For the Weighted model, the chosen value $\lambda = 0.0005$ yielded the lowest validation loss (62.3) of all models, indicating the strongest predictive performance. The adversary gap was 0.049, which points to a reasonable level of fairness. This suggests that weighting the training data was more effective for improving overall accuracy, while maintaining fairness at a level comparable to the other approaches.

If all the archetypes are analysed separately it is clear that the optimal value of the trade-off parameter λ consistently lies close to zero across all archetypes, as can be seen in Figure 29. This highlights that adversarial debiasing is valuable as a diagnostic tool, but its ability to improve fairness is limited when the underlying dataset itself constrains what the model can achieve. In addition, not all archetypes could be identified in this analysis, most likely because they could not be reliably extracted from the synthetic dataset. This further underlines that the diagnostic power of adversarial debiasing depends on the quality and representativeness of the underlying data.

Overall, the application of adversarial regularisation showed that only small values of λ were effective in balancing predictive performance and fairness. Larger values of λ quickly increased validation loss, making the models less useful in practice. Across the three variants, the Weighted model provided the strongest predictive performance, while the 156-features model offered the clearest improvement in fairness. The PCA-58 model, although computationally simpler, proved less effective in reducing bias. These results suggest that adversarial training can be a useful additional tool, but its impact remains limited when the main sources of bias have already been reduced through feature selection or weighting strategies.

8 Conclusion

The aim of this thesis was to explore how mathematical methods can be used to validate and improve the fairness of AI systems, using the risk scoring model for welfare fraud detection in Rotterdam as a case study. The central research question guiding this project was:

> "The Mathematical Validation of AI: A Case Study on Bias Reduction in the Fraud Risk Model of the Municipality of Rotterdam".

To address this question, synthetic data was generated based on the original Suspicion Machine framework, which replicates the core structure and logic of the system used by the municipality. This study applied various techniques to detect, understand, and reduce unfairness in the model by analysing bias across three phases of the AI pipeline: input, model, and output. The analysis was guided by four subquestions, which are answered below.

First of all, the original design and implementation of the Rotterdam risk scoring system was examined to determine whether any consideration was given to potential bias or fairness issues. This included an evaluation of how the dataset was constructed, how features were selected, and whether the training data was representative of the population of Rotterdam. To answer this question, both the structure of the dataset and its demographic composition were analysed in detail. Particular attention was paid to the transparency of the feature selection process and how well the dataset reflected the actual population.

The findings show little evidence that bias or fairness was meaningfully considered during the development of the system. Many of the selected features lacked a clear or causal connection to fraud, increasing the likelihood that administrative or subjective proxies were included instead. In addition, the demographic distribution of the dataset revealed significant imbalances: middle-aged individuals were strongly overrepresented, while several city districts were either inflated or underrepresented compared to official statistics from the Dutch Central Bureau of Statistics (CBS). These patterns suggest that the data used to train the model was not representative of the broader population, and that fairness was likely not a priority during either the feature selection or data preparation stages. As a result, the system may have been structurally biased from the outset. The municipality's lack of transparency about how these variables were chosen—combined with its refusal to provide further clarification—makes it difficult to determine whether potential sources of bias were even considered during the design process.

To answer the second research question, the input phase was used to explore which modifications to the dataset could help reduce bias before model training. Several techniques were applied to better understand the structure and potential risks embedded in the data.

Principal Component Analysis (PCA) provided valuable insights into the underlying structure of the dataset and helped identify which variables contributed most to overall variance. The explained variance showed a slight upward trend, but no individual variables clearly dominated the overall variance, indicating that the influence was spread across many features. Clustering techniques such as k-means and hierarchical clustering did identify clusters, yet these groupings did not provide meaningful information about high-risk individuals. This suggests that, while structural patterns exist in the data, they do not align with elevated risk scores and therefore offer limited value for interpreting high-risk segmentation.

Outlier detection turned out to be a particularly important step. The identification and removal of multivariate outliers based on Mahalanobis distance and an elbow threshold allowed for a more balanced and stable training set. These outliers appeared to have a disproportionate influence on the model, and their exclusion helped to mitigate the risk of extreme cases skewing predictions.

Overall, the findings from the input phase demonstrate that early-stage modifications, especially outlier handling and careful variable selection, can play a crucial role in reducing bias. These steps enhance the integrity of the data before any modelling takes place, making them a vital part of any fairness-oriented AI pipeline.

Then it had to be investigated how the gradient boosting model responded to different input modifications. Two models were first created based on outlier detection: one in which multivariate outliers were removed using Mahalanobis distance, and one in which weights were introduced to reduce the disproportionate influence of outliers. Between these two, the weighted model provided the most balanced results across archetypes, as it reduced extreme

distortions without substantially lowering predictive accuracy which can be seen in Figure 1.

In addition, three feature-based models were developed and the results can be seen in Figure 2. The first used a set of 156 variables obtained by excluding boolean variables, variables with low variance, and variables with extreme skewness, in order to reduce noise and remove non-informative predictors from the dataset. The second model applied a reduced set of 46 variables selected through an elbow plot of the PCA variance distribution. The third model relied on 58 variables derived from the original 156, where features most strongly correlated with biased outcomes were excluded. These three models showed noticeable shifts in archetype representation: some groups benefited from a reduced share in the top risk decile, while others experienced stronger overrepresentation. The effects were therefore inconsistent, and no single feature reduction approach produced an overall improvement across all archetypes.

Taken together, these results indicate that the gradient boosting model was highly sensitive to input modifications. Feature reduction strategies tended to redistribute bias rather than reduce it, leading to improvements for some archetypes at the expense of others. By contrast, the weighted model consistently delivered more balanced outcomes and proved to be the most effective approach overall. This suggests that controlling the influence of outliers is a more robust strategy for mitigating bias than reducing the feature space alone in this case study.

Lastly, it had to be examined how output-phase evaluations, such as statistical testing and adversarial debiasing, could be used to assess and potentially reduce bias in the model's predictions.

Independent t-tests were first applied to examine whether vulnerable archetypes were significantly overrepresented in the highest decile of risk scores. These tests confirmed that disparities persisted across multiple model versions, highlighting that improvements at the input stage did not fully eliminate uneven outcomes.

To complement these statistical checks, adversarial debiasing was introduced as an output-phase tool. This approach evaluates how much information about archetype membership is still present in the model's predictions by training an adversary to infer the protected attribute from the risk scores. The results indicated that the optimal values of the trade-off parameter λ were consistently low across archetypes. As illustrated in Figure 29, most archetypes preferred λ values close to zero. This suggests that the gradient boosting model's outputs already contained relatively little exploitable signal about group membership, and additional adversarial pressure did not lead to substantial improvements in fairness.

Overall, the findings confirm that mathematical validation is a valuable approach for improving fairness in AI. Such validation can take many forms and provides essential tools for both identifying and mitigating bias. In this case study, the most substantial gains were achieved through input-phase interventions, but at the same time illustrate the broader potential of validation techniques such as statistical testing and adversarial debiasing. The findings therefore highlight that fairness in AI is not achieved through a single method, but through a systematic validation process applied across all phases of the pipeline.

9 Discussion

The results of this case study illustrate both the potential and the limitations of mathematical validation for bias reduction in algorithmic decision-making. Adjustments to the input data, such as outlier removal and feature selection, as well as modifications in model weighting, led to measurable changes in fairness outcomes. In particular, the weighted model proved effective in reducing excess representation of vulnerable archetypes, showing that relatively straightforward statistical interventions can already mitigate part of the bias in risk scores.

At the same time, several limitations constrain the generalisability of these findings. First and foremost, the dataset used in this study does not contain verified fraud outcomes. As a result, the analysis focused exclusively on fairness measures in the distribution of risk scores, rather than predictive validity. Without ground-truth values, it remains unknown how strongly the model correlates with actual fraud behaviour, and consequently, whether reductions in bias also preserve or undermine predictive accuracy. This restriction implies that only a limited number of output tests could be performed, narrowing the scope of the evaluation.

Secondly, all models in this study were trained using gradient boosting. This choice was made deliberately to ensure comparability, since the original Rotterdam system was based on this framework and no true output labels were available to evaluate predictive accuracy. By keeping the modelling approach constant, the effects of input and preprocessing modifications could be assessed more directly. At the same time, it remains unknown whether alternative algorithms—such as random forests, logistic regression, or neural networks—would produce different bias patterns or respond differently to the same adaptations. This limitation highlights that the findings are specific to the gradient boosting framework and may not generalise to other modelling approaches.
In addition, the implementation of adversarial debiasing was limited. While an adversarial loss function was applied in the output phase to explore fairness trade-offs, it was not fully integrated into a retraining pipeline across all model versions. As a result, experimentation with this method remained relatively restricted compared to the more extensive evaluations of preprocessing and reweighting strategies.
Another limitation is that the analysis focused primarily on the 13 archetypes defined by Lighthouse Reports, which represent vulnerable groups in society. While this provided a clear benchmark for fairness evaluation, it does not necessarily reflect the model's general outcomes across the entire population. The conclusions therefore remain bounded to this subgroup perspective. Also some archetypes had very few observations in the dataset as can been seen in Figure 24, which makes the analysis less reliable. For these groups, even small changes in detection or classification can quickly alter the measured level of bias, meaning that the outcomes for such archetypes should be interpreted with caution.

Also, the way in which the municipality of Rotterdam applied and acted upon the model's results remains unclear. Lighthouse Reports (2021) state that the top ten percent of high-risk cases were further investigated by the municipality, but no detailed documentation is available on how these investigations were conducted or how the results were used in practice. This opacity limits the ability to assess whether model outputs translated into disproportionate scrutiny for specific groups, and therefore whether observed biases had tangible real-world consequences.
Taken together, these limitations suggest that the outcomes of this case study should be interpreted with caution. The analysis demonstrates that mathematical validation can meaningfully reduce measured bias in risk scoring systems, yet the absence of real outcomes, the lack of transparency regarding municipal practices, and the restricted set of tested models all limit the strength of the conclusions. Future research should therefore aim to combine bias metrics with verified outcome data, explore a broader range of modelling techniques, and investigate the institutional context in which risk scores are applied.
This case study shows that relatively small changes in preprocessing, such as reweighting outliers, already had a noticeable impact on bias reduction. This underlines the importance of systematically checking data and models before deploying them in practice, as unexamined models risk reinforcing structural inequalities rather than mitigating them.

A Variable Table

Table 3: Features

Nr	Name	Type	Subtype	Meaning
1	adres_aantal_ brp_adres	Numerical	Integer	Amount of registered addresses an entity has in the BRP.
2	adres_aantal_ verschillende_wijken	Numerical	Integer	Amount of distinct neighborhoods adres_aantal_brp_adres spans.
3	adres_aantal_ verzendadres	Nnumerical	Integer	Amount of postal addresses registered for the entity.
4	adres_aantal_ woonadres_handmatig	Numerical	Integer	Amount of registered addresses an entity has filled in by hand.
5	adres_dagen_ op_adres	Numerical	Integer	Amount of days registered at its current address calculated from the move-in date up to the reference date.
6	adres_recentst_ onderdeel_rdam	Boolean		This is a flag indicating if the most recent address is in the municipalty Rotterdam.
7	adres_recentste_ buurt_groot_ ijsselmonde	Boolean		A flag indicating if the most recent address is in the neighborhood Groot Ijselmonde.
8	adres_recentste_ buurt_nieuwe_ westen	Boolean		A flag indicating if the most recent address is in the neighborhood Nieuwe Westen.
9	adres_recentste_ buurt_other	Boolean		A flag indicating if the most recent address is in an other neighborhood.
10	adres_recentste_ buurt_oude_ noorden	Boolean		A flag indicating if the most recent address is in the neighborhood Oude Noorden.
11	adres_recentste_ buurt_vreewijk	Boolean		A flag indicating if the most recent address is in the neighborhood Vreewijk.
12	adres_recentste_ plaats_other	Boolean		A flag indicating if the most recent address is in an other neighborhood.
13	adres_recentste_ plaats_other	Boolean		A flag indicating if the most recent city is another city.
14	adres_recentste_ plaats_rotterdam	Boolean		A flag indicating it the most recent city is Rotterdam.
15	adres_recentste_ wijk_charlois	Boolean		A flag indicating it the most recent district is Charlois.
16	adres_recentste_ wijk_delfshaven	Boolean		A flag indicating it the most recent district is Delfshaven.
17	adres_recentste_ wijk_feijenoord	Boolean		A flag indicating it the most recent district is Feijenoord.
18	adres_recentste_ wijk_ijsselmonde	Boolean		A flag indicating it the most recent district is Ijsselmonde.
19	adres_recentste_ wijk_kralingen_c	Boolean		A flag indicating it the most recent district is Kralingen-Crooswijk.
20	adres_recentste_ wijk_noord	Boolean		A flag indicating it the most recent district is Noord.
21	adres_recentste_ wijk_other	Boolean		A flag indicating it the most recent district is an other district.
22	adres_recentste_ wijk_prins_ alexa	Boolean		A flag indicating it the most recent district is Prins Alexander.
23	adres_recentste_ wijk_stadscentru	Boolean		A flag indicating it the most recent district is Stadscentrum.
24	adres_unieke_ wijk_ratio	Numerical	Float	amount of unique districts divided by the total amount of living addresses.

25	afspraak_aanmelding_ afgesloten	Numerical	Integer	The amount of appointments scheduled between a welfair claimant and the municipalty's fraud control or management team.
26	afspraak_aantal_ woorden	Numerical	Integer	The total number of words in the appointment's textual description or notes.
27	afspraak_afgelopen_ jaar_afsprakenplan	Numerical	Integer	Amount of appointments in the past year following the yearly appointment plan.
28	afspraak_afgelopen_ jaar_monitoring_ insp__wet_taaleis_ na_12_mnd_n_a_v__ taa04_____geen_maatregel	Numerical	Integer	Number of appointments in the past year that were monitoring inspections under the "Wet taaleis na 12 maanden" (language-requirement law after 12 months; code TAA04) for which no enforcement measure was taken.
29	afspraak_afgelopen_ jaar_ontheffing	Numerical	Integer	Amount of appointments in the past year at which an exemption was granted to the client.
30	afspraak_afgelopen_ jaar_ontheffing_ taaleis	Numerical	Integer	Amount of appointments in the past year at which an exemption for the language requirement law was given.
31	afspraak_afgelopen_ jaar_plan_van_ aanpak	Numerical	Integer	Amount of appointments in the past year about a plan of approach.
32	afspraak_afgelopen_ jaar_signaal_ voor_medewerker	Numerical	Integer	Amount of appointments in the past year that generated a signal or alert for the caseworker.
33	afspraak_afgelopen_ jaar_vervolgmeting_ match- baarheid_werkzoekende_ klant	Numerical	Integer	Amount of appointments in the past year during which a follow-up assessment of a job-seeking client's employability (their "matchability" with available positions) was conducted.
34	afspraak_afgelopen_ jaar_voortgang_ aanmelding_en_ deelname	Numerical	Integer	Amount of appointments in the past year held as progress-check meetings to review and monitor the client's registration and actual participation in the required programs or activities.
35	afspraak_afsprakenplan	Numerical	Integer	Amount of appointments about the appointment plan.
36	afspraak_controle_ aankondiging_maatregel	Numerical	Integer	number of appointments held to perform a compliance check and formally announce an enforcement measure to the client.
37	afspraak_controle_ verwijzing	Numerical	Integer	Number of appointments in which the client's referral was checked or verified.
38	afspraak_deelname_ compleet_uit_ webapplicatie	Numerical	Integer	The number of appointments held because of a participation record sourced from the web application.
39	afspraak_galo_ gesprek	Numerical	Integer	The number of appointments held by application via 'Gegevensaanlevering Loket Online' which indicates that the information about the client is gathered through an online platform.
40	afspraak_gespr__ einde_zoekt___ galo_gesprek_	Numerical	Integer	The number of appointments during which after the conversation ended the staff member searched for the corresponding conversation record in the GALO system.
41	afspraak_ inspanningsperiode	Numerical	Integer	The number of appointments during which an effort period was set or reviewed with the client.
42	afspraak_laatstejaar_ aantal_woorden	Numerical	Integer	The number of words in the descrition of all appointments that took place in the last year.
43	afspraak_laatstejaar_ resultaat_ingevuld	Numerical	Integer	The number of appointments in the last year for which an outcome was recorded.

44	afspraak_laatstejaar_ resultaat_ingevuld_ uniek	Numerical	Integer	The number of unique filled in outcomes for the appointments in the last year.
45	afspraak_other	Numerical	Integer	The number of other appointments.
46	afspraak_ participati- etrede_ vervolgmeting	Numerical	Integer	The number of appointments in which a follow up measurement of the client's participation level was conducted.
47	afspraak_resultaat_ ingevuld_uniek	Numerical	Integer	The number of unique outcomes for appointments for which the result was filled in.
48	afspraak_signaal_ van_aanbieder	Numerical	Integer	The number of appointments that generated a signal or alert for the provider.
49	afspraak_signaal_ voor_medewerker	Numerical	Integer	The number of appointments that generated a signal or alert for the employee.
50	afspraak_toevoegen_ inschrijving_uwvwb	Numerical	Integer	The number of appointments during which the client's enrollment with the 'UWV Werkbedrijf' (Dutch public employment service) was added to their case file.
51	afspraak_vervolgmeting_ match- baarheid_werkzoekende_ klant	Numerical	Integer	The number of appointments during which a follow up assessment of a job seeking client's employability was conducted.
52	afspraak_verzenden_ beschikking_i_v_m__ niet_voldoen_ aan_wet_taaleisklant	Numerical	Integer	The number of times a notice was sent for an appoinment due to the client's failure to meet the required language law.
53	afspraak_ voortgangs- gesprek	Numerical	Integer	The number of appointments druing which a progress conversation was held.
54	belemmering_aantal_ huidig	Numerical	Integer	The number of barriers currently registered for the client.
55	belemmering_dagen_ financiele_problemen	Numerical	Integer	The number of days during which the client has been flagged as having financial difficulties.
56	belemmering_dagen_ lichameli- jke_problematiek	Numerical	Integer	The number of days during which the client has been flagged as having physical difficulties.
57	belemmering_dagen_ psychische_problemen	Numerical	Integer	The number of days during which the client has been flagged as having psychological difficulties.
58	belemmering_financiele_ problemen	Boolean		A flag indicating if the client has financial problems.
59	belemmering_hist_ lichameli- jke_problematiek	Numerical	Integer	The number of times the client was flagged impediment in history due to physical difficulties.
60	belemmering_hist_ psychische_problemen	Numerical	Integer	The number of times the client was flagged impediment in history due to psychological difficulties.
61	belemmering_hist_ stabiele_mix__ sz____dagbesteding_ werk	Numerical	Integer	The number of times the client was flagged impediment in history due to difficulties for which stable mix of social-affairs support day activities an employment was recorded.
62	belemmering_hist_ taal	Numerical	Integer	The number of times the client was flagged impediment in history due to language difficulties.
63	belemmering_hist_ verslavingsproblematiek	Numerical	Integer	The number of times the client was flagged impediment in history due to addiction difficulties.
64	belemmering_ind	Boolean		An indication whether or not the client currently has any barrier recorded in their file.
65	belemmering_ind_ hist	Boolean		An indication whether or not the client had any barrier recorded in their past.

66	belemmering_niet_ computervaardig	Boolean		An indication whether or not the client has a barrier due to a lack of computer skills.
67	belemmering_psychische_ problemen	Boolean		An indication whether or not the client has a barrier due to psychological problems.
68	belemmering_ woonsituatie	Boolean		An indication whether or not the client has a barrier due to the living situation.
69	beschikbaarheid_aantal_ historie_afwijkend_ wegens_medische_ omstandigheden	Numerical	Integer	Number of times the client's availability was atypical in history due to medical circumstances.
70	beschikbaarheid_aantal_ historie_afwijkend_ wegens_sociaal_ maatschappelijke_situatie	Numerical	Integer	Number of times the client's availability was atypical in history due to social and community-based circumstances.
71	beschikbaarheid_huidig_ afwijkend_wegens_ medische_omstandigheden	Boolean		Indication whether or not the client's availability is currently atypical due to medical circumstances.
72	beschikbaarheid_huidig_ bekend	Boolean		Indication whether or not the client's availability is known to be currently atypical.
73	beschikbaarheid_recent_ afwijkend_wegens_ medische_omstandigheden	Boolean		Indication whether or not the client's availability was recently atypical due to medical circumstances.
74	beschikbaarheid_recent_ afwijkend_wegens_ sociaal_maatschappelijke_ situatie	Boolean		Indication whether or not the client's availability was recently atypical due to social and community-based circumstances.
75	competentie_aansturen	Numerical	Integer	Number of times the client is flagged as competent to direct.
76	competentie_analyseren	Numerical	Integer	Number of times the client is flagged as competent to analyse.
77	competentie_ethisch_ en_integer_ handelen	Numerical	Integer	Number of times the client is flagged as competent to handle ethical and with integrity.
78	competentie_formuleren_ en_rapporteren	Numerical	Integer	Number of times the client is flagged as competent to formulate and report.
79	"competentie_ gedrevenheid_en_ ambitie_tonen	Numerical	Integer	Number of times the client is flagged as competent to show dedication and ambition.
80	competentie_instructies_ en_procedures_ opvolgen	Numerical	Integer	Number of times the client is flagged as competent to follow instructions and procedures.
81	competentie_kwaliteit_ leveren	Numerical	Integer	Number of times the client is flagged as competent to deliver quality.
82	competentie_leren	Numerical	Integer	Number of times the client is flagged as competent to learn.
83	competentie_materialen_ en_middelen_ inzetten	Numerical	Integer	Number of times the client is flagged as competent to deploy materials and resources.
84	competentie_met_ druk_en_tegenslag_ omgaan	Numerical	Integer	Number of times the client is flagged as competent to handle pressure and setbacks.

85	competentie_omgaan_ met_verandering_ en_aanpassen	Numerical	Integer	Number of times the client is flagged as competent to handle change and to adapt.
86	competentie_ onder- zoeken	Numerical	Integer	Number of times the client is flagged as competent to research.
87	competentie_op_ de_behoeften_ en_verwachtingen_ van_de__klant__ richten	Numerical	Integer	Number of times the client is flagged as competent to focus on the needs and expectations of clients.
88	competentie_other	Numerical	Integer	Number of times the client is flagged as competent on other qualities.
89	competentie_overtuigen_ en_be0̆0efnvloeden	Numerical	Integer	Number of times the client is flagged as competent to convince and to influence.
90	competentie_plannen_ en_organiseren	Numerical	Integer	Number of times the client is flagged as competent to plan and organise.
91	competentie_ samen- werken_ en_overleggen	Numerical	Integer	Number of times the client is flagged as competent to work in a group and discuss.
92	competentie_ vakdeskundigheid_ toepassen	Numerical	Integer	Number of times the client is flagged as competent to subject expertise.
93	contacten_onderwerp__ arbeids_motivatie	Numerical	Integer	Number of client-staff contacts in which the topic of the work motivation is discussed.
94	contacten_onderwerp__ pre__intake	Numerical	Integer	Number of client-staff contacts in which the topic of the pre-intake is discussed.
95	contacten_onderwerp__ werk_intake	Numerical	Integer	Number of client-staff contacts in which the topic of the work intake is discussed.
96	contacten_onderwerp_ arbeidsdiag- nose_dariuz	Numerical	Integer	Number of client-staff contacts in which the topic of the Dariuz method is discussed.
97	contacten_onderwerp_ beoordelen_taaleis	Numerical	Integer	Number of client-staff contacts in which the topic of the language requirement is discussed.
98	contacten_onderwerp_ boolean__arbeids_ mo- tivatie	Boolean		Indicator whether or not during client-staff contacts the topic work motivation was discussed.
99	contacten_onderwerp_ boolean__pre__ intake	Boolean		Indicator whether or not during client-staff contacts the topic pre-intake was discussed.
100	contacten_onderwerp_ boolean__werk_ intake	Boolean		Indicator whether or not during client-staff contacts the topic work intake was discussed.
101	contacten_onderwerp_ boolean_beoordelen_ taaleis	Boolean		Indicator whether or not during client-staff contacts the topic language requirement was discussed.
102	contacten_onderwerp_ boolean_contact_ der- den	Boolean		Indicator whether or not during client-staff contacts the topic contact with third parties was discussed.
103	contacten_onderwerp_ boolean_contact_ met_aanbieder	Boolean		Indicator whether or not during client-staff contacts the topic contact with provider was discussed.
104	contacten_onderwerp_ boolean_diagnosegesprek	Boolean		Indicator whether or not during client-staff contacts the topic diagnostic interview was discussed.
105	contacten_onderwerp_ boolean_documenten__ innemen_	Boolean		Indicator whether or not during client-staff contacts the topic capture documents was discussed.
106	contacten_onderwerp_ boolean_documenttype__cv_	Boolean		Indicator whether or not during client-staff contacts the topic curriculum vitae document type was discussed.

107	contacten_onderwerp_ boolean_documenttype__ diploma_s_en_ certificaten_	Boolean		Indicator whether or not during client-staff contacts the topic degrees and certificates was discussed.
108	contacten_onderwerp_ boolean_documenttype__ overeenkomst_	Boolean		Indicator whether or not during client-staff contacts the topic formal agreement or contract document type was discussed.
109	contacten_onderwerp_ boolean_financi00eble_ situatie	Numerical	Integer	Number of times during client-staff contacts the topic financial situation was discussed.
110	contacten_onderwerp_ boolean_groepsbijeenkomst	Boolean		Indicator whether or not during client-staff contacts the topic group meeting was discussed.
110	contacten_onderwerp_ boolean_inkomen	Boolean		Indicator whether or not during client-staff contacts the topic income was discussed.
111	contacten_onderwerp_ boolean_maatregel_ overweging	Boolean		Indicator whether or not during client-staff contacts the topic considering enforcement measures was discussed.
112	contacten_onderwerp_ boolean_match___ werk	Boolean		Indicator whether or not during client-staff contacts the topic matching the client to suitable work was discussed.
113	contacten_onderwerp_ boolean_matching	Boolean		Indicator whether or not during client-staff contacts the topic matching in general was discussed.
114	contacten_onderwerp_ boolean_motivatie	Boolean		Indicator whether or not during client-staff contacts the topic motivation was discussed.
115	contacten_onderwerp_ boolean_mutatie	Boolean		Indicator whether or not during client-staff contacts the topic of a record update was discussed.
116	contacten_onderwerp_ boolean_no_show	Boolean		Indicator whether or not during client-staff contacts the topic of the absence of the client was discussed.
117	contacten_onderwerp_ boolean_overige	Boolean		Indicator whether or not during client-staff contacts other topics were discussed.
118	contacten_onderwerp_ boolean_overleg_ met_inkomen	Boolean		Indicator whether or not during client-staff contacts the topic with the income department was discussed.
119	contacten_onderwerp_ boolean_scholing	Boolean		Indicator whether or not during client-staff contacts the topic education was discussed.
120	contacten_onderwerp_ boolean_sollicitatie	Boolean		Indicator whether or not during client-staff contacts the topic job interview was discussed.
121	contacten_onderwerp_ boolean_taaleis___ voldoet	Boolean		Indicator whether or not during client-staff contacts the topic the client's compliance with the language requirement was discussed.
122	contacten_onderwerp_ boolean_terugbelverzoek	Boolean		Indicator whether or not during client-staff contacts the topic of a caalback request was discussed.
123	contacten_onderwerp_ boolean_traject	Boolean		Indicator whether or not during client-staff contacts the topic of the client's trajectory was discussed.
124	contacten_onderwerp_ boolean_uitnodiging	Boolean		Indicator whether or not during client-staff contacts the topic of an invitation was discussed.
125	contacten_onderwerp_ boolean_ziek__ of_afmelding	Boolean		Indicator whether or not during client-staff contacts the topic of sickness or cancellation was discussed.
126	contacten_onderwerp_ boolean_zorg"	Boolean		Indicator whether or not during client-staff contacts the topic care was discussed.
127	contacten_onderwerp_ contact_derden	Numerical	Integer	Number of times in client-staff contact the topic contact with third parties is discussed.

128	contacten_onderwerp_contact_met_ aanbieder	Numerical	Integer	Number of times in client-staff contact the topic contact with provider is discussed.
129	contacten_onderwerp_diagnosegesprek	Numerical	Integer	Number of times in client-staff contact the topic diagnostic interview is discussed.
130	contacten_onderwerp_documenten__innemen_	Numerical	Integer	Number of times in client-staff contact the topic capture documents is discussed.
131	contacten_onderwerp_documenttype__cv_	Numerical	Integer	Number of times in client-staff contact the topic the curriculum vitae document type is discussed.
132	contacten_onderwerp_documenttype__overeenkomst_	Numerical	Integer	Number of times in client-staff contact the topic formal agreement or contract document type is discussed.
133	contacten_onderwerp_financiŐ0eble_ situatie	Numerical	Integer	Number of times in client-staff contact the topic financial situation is discussed.
134	contacten_onderwerp_groepsbijeenkomst	Numerical	Integer	Number of times in client-staff contact the topic group gathering is discussed.
135	contacten_onderwerp_inkomen	Numerical	Integer	Number of times in client-staff contact the topic income is discussed.
136	contacten_onderwerp_inname_aanvraag	Numerical	Integer	Number of times in client-staff contact the topic of submitting or receiving an intake application is discussed.
137	contacten_onderwerp_inspanningstoets	Numerical	Integer	Number of times in client-staff contact the topic effort test is discussed.
138	contacten_onderwerp_maatregel_overweging	Numerical	Integer	Number of times in client-staff contact the topic considering enforcement measures is discussed.
139	contacten_onderwerp_matching	Numerical	Integer	Number of times in client-staff contact the topic matching in general is discussed.
140	contacten_onderwerp_mutatie	Numerical	Integer	Number of times in client-staff contact the topic of a record update is discussed.
141	contacten_onderwerp_no_show	Numerical	Integer	Number of times in client-staff contact the topic of absence is discussed.
142	contacten_onderwerp_overige	Numerical	Integer	Number of times in client-staff contact other topics were discussed.
143	contacten_onderwerp_overleg_met_ inkomen	Numerical	Integer	Number of times in client-staff contact the topic about contact with the income department is discussed.
144	contacten_onderwerp_quickscan	Numerical	Integer	Number of times in client-staff contact the topic of a quickscan is discussed.
145	contacten_onderwerp_scholing	Numerical	Integer	Number of times in client-staff contact the topic education is discussed.
146	contacten_onderwerp_screening	Numerical	Integer	Number of times in client-staff contact the topic screening is discussed.
147	contacten_onderwerp_terugbelverzoek	Numerical	Integer	Number of times in client-staff contact the topic callback request is discussed.
148	contacten_onderwerp_traject	Numerical	Integer	Number of times in client-staff contact the topic trajectory is discussed.
149	contacten_onderwerp_uitnodiging	Numerical	Integer	Number of times in client-staff contact the topic invitation is discussed.
150	contacten_onderwerp_vakantie	Numerical	Integer	Number of times in client-staff contact the topic holliday is discussed.
151	contacten_onderwerp_werkintake__niet_ verschenen	Numerical	Integer	Number of times in client-staff contact the topic of not appearing on the work intake is discussed.

152	contacten_onderwerp_ ziek__of_ afmelding	Numerical	Integer	Number of times in client-staff contact the topic sickness or cancellation is discussed.
153	contacten_onderwerp_ zorg	Numerical	Integer	Number of times in client-staff contact the topic care is discussed.
154	contacten_soort_ afgelopenjaar_anders	Numerical	Integer	Number of client-staff contacts in the past year that involved other types of interaction.
155	contacten_soort_ afgelopen- jaar_document__ inkomend_	Numerical	Integer	Number of client-staff contacts in the past year that involved incoming documents.
156	contacten_soort_ afgelopen- jaar_document__ uit- gaand_	Numerical	Integer	Number of client-staff contacts in the past year that involved outgoing documents.
157	contacten_soort_ afgelopenjaar_e_mail__ inkomend_	Numerical	Integer	Number of client-staff contacts in the past year that involved incoming emails.
158	contacten_soort_ afgelopenjaar_e_mail__ uitgaand_	Numerical	Integer	Number of client-staff contacts in the past year that involved outgoing emails
159	contacten_soort_ afgelopenjaar_gesprek	Numerical	Integer	Number of client-staff contacts in the past year that involved a conversation.
160	contacten_soort_ afgelopenjaar_gesprek_ op_locatie	Numerical	Integer	Number of client-staff contacts in the past year that involved a conversation on-site.
161	contacten_soort_ afgelopen- jaar_rapportage_ rib	Numerical	Integer	Number of client-staff contacts in the past year that involved filing or submitting a report to the 'Reguliere Integrale Beoordeling' system (system used by the municipality's fraud-control and case-management team).
162	contacten_soort_ afgelopen- jaar_telefoontje__ inkomend_	Numerical	Integer	Number of client-staff contacts in the past year that involved an incoming call.
163	contacten_soort_ afgelopen- jaar_telefoontje__ uit- gaand_	Numerical	Integer	Number of client-staff contacts in the past year that involved an outgoing call.
164	contacten_soort_ an- ders	Numerical	Integer	Number of client-staff contacts that involved other types of interaction.
165	contacten_soort_ docu- ment__inkomend_	Numerical	Integer	Number of client-staff contacts that involved incoming documents.
166	contacten_soort_ docu- ment__uitgaand_	Numerical	Integer	Number of client-staff contacts that involved outgoing documents.
167	contacten_soort_ e_mail__inkomend_	Numerical	Integer	Number of client-staff contacts that involved incoming emails.
168	contacten_soort_ e_mail__uitgaand_	Numerical	Integer	Number of client-staff contacts that involved outgoing emails.
169	contacten_soort_ gesprek_op_ locatie	Numerical	Integer	Number of client-staff contacts that involved a conversation on-site.
170	contacten_soort_ groepsbijeenkomsten	Numerical	Integer	Number of client-staff contacts that involved group gatherings.
171	contacten_soort_ rap- portage_deelname	Numerical	Integer	Number of client-staff contacts that involved a filed report about participation.

172	contacten_soort_ rapportage_rib	Numerical	Integer	Number of client-staff contacts that involved a filed report in the RIB system (system used by the municipality's fraud-control and case-management team).
173	contacten_soort_ telefoontje__inkomend_	Numerical	Integer	Number of client-staff contacts that involved an incoming call.
174	contacten_soort_ telefoontje__uitgaand_	Numerical	Integer	Number of client-staff contacts that involved an outgoing call.
175	deelname_act_ actueel_projecten_ uniek	Numerical	Integer	Number of unique participation activities that the client is currently involved in within a project.
176	deelname_act_ hist_projecten_ niet_gestart	Numerical	Integer	Number of participation activities in history in which the client has not started a project.
177	deelname_act_ reintegratieladder_ ondersteunende_ instrumenten	Numerical	Integer	Number of participation activities where the client engaged with supporting instruments from the reintegration ladder.
178	deelname_act_ reintegratieladder_ werk_re_integratie	Numerical	Integer	Number of participation activities in which the client engaged in work reintegration from the reintegration ladder.
179	instrument_aantal_ laatstejaar	Numerical	Integer	Number of distinct support instruments that the client used in the past year.
180	instrument_ladder_ historie_activering	Numerical	Integer	Number of times any reintegration ladder instrument was activated for the client in history.
181	instrument_ladder_ historie_other	Numerical	Integer	Number of times any other reintegration ladder instrument was used by the client in history.
182	instrument_ladder_ huidig_activering	Numerical	Integer	Number of times any reintegration ladder instrument is activated for the client in the present.
183	instrument_ladder_ huidig_other	Numerical	Integer	Number of times any other reintegration ladder instrument is used by the client in the present.
184	instrument_ladder_ huidig_werk_ re_integratie	Numerical	Integer	Number of times any work reintegration instrument from the reintegration ladder is used the client in the present.
185	instrument_reden_ beeindiging_historie_ centrale_actie_ wigo4it	Numerical	Integer	Number of cases in which a support instrument was terminated in history due to 'Centrale Actie WIGO4IT' (dentral action under the WIGO4IT program).
186	instrument_reden_ beeindiging_historie_ diagnose_gesteld	Numerical	Integer	Number of cases in which a support instrument was terminated in history due to a diagnosis.
187	instrument_reden_ beeindiging_historie_ direct_matchbaar	Numerical	Integer	Number of cases in which a support instrument was terminated in history due to the client's directly employable.
188	instrument_reden_ beeindiging_historie_ doelstelling_bereikt__ geen_uitstroom	Numerical	Integer	Number of cases in which a support instrument was terminated in history because the client met their set objective without resulting in an exit from the program.
189	instrument_reden_ beeindiging_historie_ doelstelling_bereikt__ uitstroom_naar_ regulier_werk	Numerical	Integer	Number of cases in which a support instrument was terminated in history because the client met their set objective resulting in an exit from the program to regular work.

190	instrument_reden_ beeindiging_historie_ doelstelling_bereikt__ uitstroom_overig	Numerical	Integer	Number of cases in which a support instrument was terminated in history because the client met their set objective resulting in an exit from the program to others.
191	instrument_reden_ beeindiging_historie_ doelstelling_niet_ bereikt__geen_ uit- stroom	Numerical	Integer	Number of cases in which a support instrument was terminated in history because the client did not meet their set objective resulting in no exit from the program.
192	instrument_reden_ beeindiging_historie_ niet_matchbaar	Numerical	Integer	Number of cases in which a support instrument was terminated in history because the client was not matchable.
193	instrument_reden_ beeindiging_historie_ niet_succesvol	Numerical	Integer	Number of cases in which a support instrument was terminated in history because of an unsuccessful outcome.
194	instrument_reden_ beeindiging_historie_ niet_succesvol__ re- den_bij_ werkzoekende	Numerical	Integer	Number of cases in which a support instrument was terminated in history due to an unsuccessful outcome caused by the job seeker.
195	instrument_reden_ beeindiging_historie_ op_termijn_ matchbaar	Numerical	Integer	Number of cases in which a support instrument was terminated in history with reason that the client is matchable in the future.
196	instrument_reden_ beeindiging_historie_ other	Numerical	Integer	Number of cases in which a support instrument was terminated in history due to other reasons.
197	instrument_reden_ beeindiging_historie_ overdracht_naar_ mo	Numerical	Integer	Number of cases in which a support instrument was terminated in history because the client was trans- fered to 'Maatschappelijke opvang' (social care).
198	instrument_reden_ beeindiging_historie_ overdacht_naar_ pre- matching	Numerical	Integer	Number of cases in which a support instrument was terminated in history because the client was transfered to prematching.
199	instrument_reden_ beeindiging_historie_ overdracht_naar_ reg- ulier_team_ iwpm	Numerical	Integer	Number of cases in which a support instrument was terminated in history because the client was transfered to the regular IWPM team.
200	instrument_reden_ beeindiging_historie_ overdacht_succesvol	Numerical	Integer	Number of cases in which a support instrument was terminated in history for which the transfer was successful.
201	instrument_reden_ beeindiging_historie_ succesvol__doelstelling_ bereikt	Numerical	Integer	Number of cases in which a support instrument was terminated in history for which the set objective was met.
202	instrument_reden_ beeindiging_historie_ uitval__aanbod_ niet_langer_ zin- vol_volgens_ gemeente	Numerical	Integer	Number of cases in which a support instrument was terminated in history because the offer is no longer meaningful according to the municipality.
203	instrument_reden_ beeindiging_historie_ uitval__klant_ wz__ziet_af_ van_aanbod	Numerical	Integer	Number of cases in which a support instrument was terminated in history because the job seeking client declined the offer.
204	ontheffing_actueel_ ind	Boolean		Indicator whether or not the client currently has an active exemption.

205	ontheffing_dagen_ hist_mean	Numerical	Integer	Mean duration of days of all exemption periods granted to the client in history.
206	ontheffing_dagen_ hist_vanwege_ uw_medische_ omstandigheden	Numerical	Integer	Duration of days of all exemption periods granted to the client in history due to medical circumstances.
207	ontheffing_hist_ ind	Boolean		Indicator whether or not the client has had any exemption in history.
208	ontheffing_reden_ hist_medische_ gronden	Numerical	Integer	Reason of exemption periods granted to the client in history due to medical reasons.
209	ontheffing_reden_ hist_other	Numerical	Integer	Reason of exemption periods granted to the client in history because of other reasons.
210	ontheffing_reden_ hist_sociale_ gronden	Numerical	Integer	Reason of exemption periods granted to the client in history due to social reasons.
211	ontheffing_reden_ hist_tijdelijke_ ontheffing_arbeidsverpl__ en_tegenprestatie	Numerical	Integer	Reason of exemption periods granted to the client in history because of a temporary exemption from their wrok-search requirements and the associated counter-performance obligations.
212	ontheffing_reden_ hist_tijdelijke_ ontheffing_ arbeidsverplichtingen	Numerical	Integer	Reason of exemption periods granted to the client in history due to a temporary exemption based on the work-search requirements.
213	ontheffing_reden_ hist_vanwege_ uw_sociaal_ maatschappelijke_situatie	Numerical	Integer	Reason of exemption periods granted to the client in history due to social and community based reasons.
214	ontheffing_reden_ tijdelijke_ontheffing_ arbeidsverpl__en_ tegenprestatie	Boolean		Indicator whether or not the client was granted an exemption due to a temporary exemption from their work-search requirements and the associated counte-performance obligations.
215	persoon_geslacht_ vrouw	Boolean		Indicator whether the client is a woman.
216	persoon_leeftijd_ bij_onderzoek	Numerical	Integer	The age of the person during the investigation.
217	persoonlijke_ eigenschappen_ communicatie_opm	Boolean		Indicator whether or not there was a comment made on the topic of communication about the client's personal characteristics.
218	persoonlijke_ eigenschappen_ dagen_sinds_ opvoer	Numerical	Integer	Number of days since the personal chracterisitcs information were updated.
219	persoonlijke_ eigenschappen_ dagen_sinds_ taaleis	Numerical	Integer	Number of days since the personal chracteristics of the client about the language requirement were updated.
220	persoonlijke_ eigenschappen_ doorzettingsvermogen_opm	Boolean		Indicator whether or not there was a comment made on the topic of persistence about the client's personal characteristics.
221	persoonlijke_ eigenschappen_ flexibiliteit_opm	Boolean		Indicator whether or not there was a comment made about flexibility about the client's personal characteristics.
222	persoonlijke_ eigenschappen_ hobbies_sport	Boolean		Indicator whether or not the client has any hobbies or sport to participate in.

223	persoonlijke_ eigenschappen_ houding_opm	Boolean		Indicator whether or not there was a comment made about attitude in the client's personal characteristics.
224	persoonlijke_ eigenschappen_ ind_activering_ trajct	Boolean		Indicator whether or not there is an activation of the trajectory in the client's personal characteristics.
225	persoonlijke_ eigenschappen_ ind_buiten_ kantoortijden	Boolean		Indicator whether or not the client's personal characteristics information was recorded outside of standard office hours.
226	persoonlijke_ eigenschappen_ ind_regulier_ arbeidsritme	Boolean		Indicator whether or the client's personal characteristics information was recorded during regular office hours.
227	persoonlijke_ eigenschappen_ initiatief_opm	Boolean		Indicator whether or not there was a comment made about initiative in the client's personal characteristics.
228	persoonlijke_ eigenschappen_ leergierigheid_opm	Boolean		Indicator whether or not there was a comment made about the client's will to learn in the client's personal characteristics.
229	persoonlijke_ eigenschappen_ motivatie_opm	Boolean		Indicator whether or not there was a comment made about motivation in the client's personal characteristics.
230	persoonlijke_ eigenschappen_ nl_begrijpen3	Boolean		Indicator whether or not the client's personal characteristics include Dutch listening comprehension level three.
231	persoonlijke_ eigenschappen_ nl_lezen3	Boolean		Indicator whether or not the client's personal characteristics include Dutch reading comprehension level three.
232	persoonlijke_ eigenschappen_ nl_lezen4	Boolean		Indicator whether or not the client's personal characteristics include Dutch reading comprehension level four.
233	persoonlijke_ eigenschappen_ nl_schrijven0	Boolean		Indicator whether or not the client's personal characteristics include Dutch writing comprehension level zero.
234	persoonlijke_ eigenschappen_ nl_schrijven1	Boolean		Indicator whether or not the client's personal characteristics include Dutch writing comprehension level one.
235	persoonlijke_ eigenschappen_ nl_schrijven2	Boolean		Indicator whether or not the client's personal characteristics include Dutch writing comprehension level two.
236	persoonlijke_ eigenschappen_ nl_schrijven3	Boolean		Indicator whether or not the client's personal characteristics include Dutch writing comprehension level three.
237	persoonlijke_ eigenschappen_ nl_schrijvenfalse	Boolean		Indicator whether or not the client's personal characteristics do not include Dutch writing comprehension.
238	persoonlijke_ eigenschappen_ nl_spreken1	Boolean		Indicator whether or not the client's personal characteristics include Dutch speaking level one.
239	persoonlijke_ eigenschappen_ nl_spreken2	Boolean		Indicator whether or not the client's personal characteristics include Dutch speaking level two.
240	persoonlijke_ eigenschappen_ nl_spreken3	Boolean		Indicator whether or not the client's personal characteristics include Dutch speaking level three.

241	persoonlijke_ eigenschappen_ opstelling	Boolean		Indicator whether or not the client's personal characteristics include any comment about their general attitude.
242	persoonlijke_ eigenschappen_ overige_opmerkingen	Boolean		Indicator whether or not the client's personal characteristics include any other comments.
243	persoonlijke_ eigenschappen_ presentatie_opm	Boolean		Indicator whether or not the client's personal characteristics include any comment about their presentation.
244	persoonlijke_ eigenschappen_ spreektaal	Categorical		Category which language does the client speaks.
245	persoonlijke_ eigenschappen_ spreektaal_anders	Boolean		Indicator whether or not the client's personal characteristics include speaking another language.
246	persoonlijke_ eigenschappen_ schrijfv_ok	Boolean		Indicator whether or not the client's personal characteristics include an okay writing comprehension.
247	persoonlijke_ eigenschappen_ taaleis_voldaan	Categorical		Cahtegory about the client's personal characteristics about the completed language requirement.
248	persoonlijke_ eigenschappen_ verzorging_opm	Boolean		Indicator whether or not the client's personal characteristics include any comment about their general care.
249	persoonlijke_ eigenschappen_ uitstroom_verw_ vlgns_klant	Categorical		Category about the client's expression about leaving the program in the client's personal characteristics.
250	persoonlijke_ eigenschappen_ uitstroom_verw_ vlgns_km	Categorical		Category about the case manager's expression to exit the client of the program in the client's personal characteristics.
251	persoonlijke_ eigenschappen_ zelfstandigheid_opm	Boolean		Indicator whether or not the client's personal characteristics include any comment about their independency.
252	pla_actueel_ pla_categorie_ doelstelling_16	Numerical	Integer	Number of currently active trajectory plans in the category with set goal objective 16.
253	pla_actueel_ pla_categorie_ doelstelling_9	Boolean		Indicator whether there are currently active trajectory plans in the category with set goal objective 9.
254	pla_einde_ doelstelling_bereikt	Numerical	Integer	Number of finished trajectory plans where the set goal objective is obtained.
255	pla_einde_ doelstelling_bereikt__ nieuw_trajectplan	Numerical	Integer	Number of finished trajectory plans where the set goal objective is obtained and a new trajectory plan is made.
256	pla_einde_ doelstelling_bereikt__ overdracht	Numerical	Integer	Number of finished trajectory plans where the set goal objective is obtained and there has been a transfer.
257	pla_einde_ doelstelling_niet_ bereikt	Numerical	Integer	Number of finished trajectory plans where the set goal objective is not obtained.
258	pla_einde_ doelstelling_niet_ bereikt__ nieuw_trajectplan	Numerical	Integer	Number of finished trajectory plans where the set goal objective is not obtained but there is a new trajectory plan made.
259	pla_einde_ doelstelling_niet_ bereikt__overdracht	Numerical	Integer	Number of finished trajectory plans where the set goal objective is not obtained but there was a transfer.

260	pla_einde_ geplande_einddatum_ overschreden_zonder_ nader_bericht	Numerical	Integer	Number of finished trajectory plans where the end date is exceeded without further notice.
261	pla_einde_ other	Numerical	Integer	Number of other finished trajectory plans.
262	pla_einde_ uitstroom_anders_ dan_volgen_ onderwijs__regulier_ werk_of_als_ zelfstandiger	Numerical	Integer	Number of finished trajectory plans where the exit was different then following education, regular work or as self-employement.
263	pla_hist_ pla_categorie_ doelstelling_1	Numerical	Integer	Number of trajectory plans in history in the category with set goal objective 1.
264	pla_hist_ pla_categorie_ doelstelling_10	Numerical	Integer	Number of trajectory plans in history in the category with set goal objective 10.
265	pla_hist_ pla_categorie_ doelstelling_11	Numerical	Integer	Number of trajectory plans in history in the category with set goal objective 11.
266	pla_hist_ pla_categorie_ doelstelling_16	Numerical	Integer	Number of trajectory plans in history in the category with set goal objective 16.
267	pla_hist_ pla_categorie_ doelstelling_2	Numerical	Integer	Number of trajectory plans in history in the category with set goal objective 2.
268	pla_hist_ pla_categorie_ doelstelling_27	Numerical	Integer	Number of trajectory plans in history in the category with set goal objective 27.
269	pla_hist_ pla_categorie_ doelstelling_3	Numerical	Integer	Number of trajectory plans in history in the category with set goal objective 3.
270	pla_hist_ pla_categorie_ doelstelling_4	Numerical	Integer	Number of trajectory plans in history in the category with set goal objective 4.
271	pla_hist_ pla_categorie_ doelstelling_5	Numerical	Integer	Number of trajectory plans in history in the category with set goal objective 5.
272	pla_hist_ pla_categorie_ doelstelling_6	Numerical	Integer	Number of trajectory plans in history in the category with set goal objective 6.
273	pla_hist_ pla_categorie_ doelstelling_9	Numerical	Integer	Number of trajectory plans in history in the category with set goal objective 9.
274	pla_hist_ pla_categorie_ doelstelling_other	Numerical	Integer	Number of trajectory plans in history in the category with an other set goal objective.
275	pla_historie_ maatschappelijke_ inspanning	Boolean		Indicator whether there were trajectory plans in history with social endeavor.
276	pla_historie_ ontwikkeling	Boolean		Indicator whether there were trajectory plans in history with developments.
277	pla_historie_ other	Boolean		Indicator whether there were other trajectory plans in history.
278	pla_historie_ werk	Boolean		Indicator whether there were trajectory plans in history regarding work.
279	pla_historie_ werk_en_inburgering	Boolean		Indicator whether there were trajectory plans in history regarding work and integration.
280	pla_ondertekeningen_ actueel	Numerical	Integer	Number of active plans that have been signed.
281	pla_ondertekeningen_ history	Numerical	Integer	Number of plans that have been signed in the past.
282	relatie_kind_ basisschool_kind	Numerical	Integer	Number of children the client has in primary school.
283	relatie_kind_ heeft_kinderen	Boolean		Indicator whether the client has any children.

284	relatie_kind_ huidig_aantal	Numerical	Integer	Number of children the client has at this moment.
285	relatie_kind_ jongvolwassen	Numerical	Integer	Number of children the client has that are young adults.
286	relatie_kind_ leeftijd_verschil_ ouder_eerste_ kind	Numerical	Integer	Difference in age between the client and the oldest child.
287	relatie_kind_ tiener	Numerical	Integer	Number of children the client has that are teenagers.
288	relatie_kind_ volwassen	Numerical	Integer	Number of children the client has that are adults.
289	relatie_overig_ actueel_vorm__ gemachtigde	Numerical	Integer	Number of other persons the client currently granted formal authorization for.
290	relatie_overig_ actueel_vorm__ kostendeler	Numerical	Integer	Number of other persons the client currently has a relation as cost sharing arrangements with.
291	relatie_overig_ actueel_vorm__ onderhoudsplichtige	Numerical	Integer	Number of other persons the client currently has a relation as debtor with.
292	relatie_overig_ actueel_vorm__ ouders_verzorgers	Numerical	Integer	Number of other persons the client currently has as parents or caretakers.
293	relatie_overig_ actueel_vorm_ other	Numerical	Integer	Number of other persons the client currently has as a relation with.
294	relatie_overig_ bewindvoerder	Boolean		Indicator whether the client is an administrator or not.
295	relatie_overig_ historie_vorm__ andere_inwonende	Numerical	Integer	Number of persons the client has had a relationship with in the past regarding indwelling.
296	relatie_overig_ historie_vorm__ gemachtgde	Numerical	Integer	Number of persons the client has had a relationship with in the past regarding giving formal authorisation.
297	relatie_overig_ historie_vorm__ kostendeler	Numerical	Integer	Number of persons the client has had a relationship with in the past regarding sharing cost arrangements.
298	relatie_overig_ historie_vorm__ onderhoudsplichtige	Numerical	Integer	Number of persons the client has had a relationship with in the past regarding being a debtor.
299	relatie_overig_ kostendeler	Boolean		Indicator whether the client has a relationship regarding sharing cost arrangements.
300	relatie_partner_ aantal_partner___ partner__gehuwd_	Numerical	Integer	Number of registered partner relationships in which the partner is registered as legally married.
301	relatie_partner_ aantal_partner___ partner__ongehuwd_	Numerical	Integer	Number of registered partner relationships in which the partner is registered as not legally married.
302	relatie_partner_ huidige_partner___ partner__gehuwd_	Boolean		Indicator whether the current registered partner relationship is registered as legally married.
303	relatie_partner_ totaal_dagen_ partner	Numerical	Integer	Number of days the current registered partner relationship is registered as being a partner.
304	typering_aantal	Numerical	Integer	Number of typologies the client is assigned to.
305	typering_dagen_ som	Numerical	Integer	Total number of days the client is assigned to typologies.

306	typering_hist_ aantal	Numerical	Integer	Number of typologies the client is assigned to in history.
307	typering_hist_ inburgeringsbehoefte	Boolean		Indicator whether the client was assigned the typologies of having the need for integration in the past.
308	typering_hist_ ind	Boolean		Indicator whether the client was assigned a typology in the past.
309	typering_hist_ sector_zorg	Numerical	Integer	Number of typologies the client is assigned to in history in the sector care.
310	typering_ind	Boolean		Indicator whether the client is assigned a typology.
311	typering_indicatie_geheime_gegevens	Boolean		Indicator whether the client was assigned a typology about secret information.
312	typering_other	Numerical	Integer	Number of other typologies the client is assigned to.
313	typering_transport__logistiek___tuinbouw	Boolean		Indicator whether the client is assigned to the typology of transport, logistics and agriculture.
314	typering_zorg__schoonmaak___welzijn	Boolean		Indicator whether the client is assigned to the typology of care, cleaning and welfare.

B Python Code

B.1 Data Comparison Real world

```
import pandas as pd

synth_df = pd.read_csv('Documents/BEP/Jupiter/synth_data_with_id.csv')
cbs_df = pd.read_csv("Documents/BEP/Jupiter/Kerncijfers_wijken_en_buurten_2020_16082025_1
cbs_df["Regioaanduiding/Soort␣regio␣(omschrijving)"] = cbs_df["Regioaanduiding/Soort␣regi
cbs_df["Regioaanduiding/Gemeentenaam␣(naam)"] = cbs_df["Regioaanduiding/Gemeentenaam␣(naa

#Read synth data
wijk_cols = [col for col in synth_df.columns if col.startswith("adres_recentste_wijk_")]
synth_dist = synth_df[wijk_cols].mean() * 100
synth_dist.index = synth_dist.index.str.replace("adres_recentste_wijk_", "")
synth_df_clean = synth_dist.reset_index()
synth_df_clean.columns = ["    Wijken    ␣en␣buurten", "Dataset_Percentage"]


#Only the municipalty Rotterdam
rotterdam_total_row = cbs_df.loc[
    (cbs_df["    Wijken    ␣en␣buurten"].str.strip() == "Rotterdam") &
    (cbs_df["Regioaanduiding/Gemeentenaam␣(naam)"] == "Rotterdam")
].copy()

rotterdam_total = pd.to_numeric(cbs_df.loc[1, "Bevolking/Aantal␣inwoners␣(aantal)"])

# short normalize
def _norm(s):
    if pd.isna(s): return ""
    s = unicodedata.normalize("NFKD", str(s)).encode("ascii","ignore").decode("ascii")
    return s.strip().lower()

# mapping from suffixes to CBS names
suffix_to_cbs = {
    "charlois": "Charlois",
    "delfshaven": "Delfshaven",
    "feijenoord": "Feijenoord",
    "ijsselmonde": "IJsselmonde",
    "kralingen_c": "Kralingen-Crooswijk",
    "noord": "Noord",
    "overschie": "Overschie",
    "pernis": "Pernis",
    "prins_alexa": "Prins␣Alexander",
    "stadscentru": "Rotterdam␣Centrum",
    "hillegersberg": "Hillegersberg-Schiebroek",
    "hoek_van_holland": "Hoek␣van␣Holland",
    "hoogvliet": "Hoogvliet",
    "other": "Other",
}

#find the name column in your synth_df_clean
if "synth_df_clean" not in globals():
    raise RuntimeError("synth_df_clean␣not␣found.␣Build␣it␣before␣this␣block.")

name_candidates = [c for c in synth_df_clean.columns if "wijken" in c.lower()]
name_col = name_candidates[0] if name_candidates else synth_df_clean.columns[0]
```

```
# standardize columns
synth_map = synth_df_clean.rename(columns={name_col: "name_raw"}).copy()
if "Dataset_Percentage" not in synth_map.columns:
    # try to find percentage column
    pct_candidates = [c for c in synth_map.columns if "percentage" in c.lower()]
    if not pct_candidates:
        raise RuntimeError("Cannot␣find␣Dataset_Percentage␣column␣in␣synth_df_clean.")
    synth_map = synth_map.rename(columns={pct_candidates[0]: "Dataset_Percentage"})

#suffix mode or CBS-name mode
vals = synth_map["name_raw"].astype(str).str.strip().str.lower().unique()
if set(vals).issubset(set(suffix_to_cbs.keys())):
    # suffix mode
    synth_map["CBS_wijk"] = synth_map["name_raw"].str.lower().map(suffix_to_cbs)
else:
    # CBS-name mode
    def to_cbs_name(x):
        s = str(x).strip()
        if s.lower().startswith("rotterdam␣"):
            s = s[len("rotterdam␣"):]
        # small fixes for common truncations
        fixes = {
            "kralingen␣c": "Kralingen-Crooswijk",
            "stadscentru": "Centrum",
            "prins␣alexa": "Prins␣Alexander",
        }
        for k, v in fixes.items():
            if s.lower().startswith(k):
                return v
        return s
    synth_map["CBS_wijk"] = synth_map["name_raw"].apply(to_cbs_name)

#aggregate synth to CBS_wijk and ensure 'Other'
synth_mapped = (
    synth_map.groupby("CBS_wijk", as_index=False)["Dataset_Percentage"]
    .sum()
    .rename(columns={"Dataset_Percentage": "Synth_%"})
)
if "Other" not in set(synth_mapped["CBS_wijk"]):
    other_pct = max(0.0, 100.0 - float(synth_mapped["Synth_%"].sum()))
    synth_mapped = pd.concat(
        [synth_mapped, pd.DataFrame({"CBS_wijk": ["Other"], "Synth_%": [other_pct]})],
        ignore_index=True
    )

# merge with CBS table
if "cbs_14" not in globals():
    raise RuntimeError("cbs_14␣not␣found.␣Build␣CBS␣table␣before␣this␣block.")

NAME_COL = "Wijken␣en␣buurten"
cbs_pct = cbs_14[[NAME_COL, "Percentage"]].rename(columns={"Percentage": "CBS_%"})
cbs_pct["key"] = cbs_pct[NAME_COL].apply(_norm)
synth_mapped["key"] = synth_mapped["CBS_wijk"].apply(_norm)

compare_df = synth_mapped.merge(cbs_pct[["key","CBS_%"]], on="key", how="left")
```

```
compare_df["Diff"] = compare_df["Synth_%"] - compare_df["CBS_%"]
compare_df = compare_df[["CBS_wijk", "CBS_%", "Synth_%", "Diff"]].sort_values("CBS_wijk")

print("District % (CBS vs Synth)")
print(compare_df.to_string(index=False))


# age + gender from synth_df and compare with CBS

import numpy as np
import pandas as pd

# pick synth df
_synth = synth_df

# Age
age_col = "persoon_leeftijd_bij_onderzoek"  # integer age in years
age = pd.to_numeric(_synth[age_col], errors="coerce")

# clamp to plausible years
age = age[(age >= 0) & (age <= 120)]

# bins and labels
bins = [0, 15, 25, 45, 65, np.inf]
labels = ["0 tot 15 jaar","15 tot 25 jaar","25 tot 45 jaar","45 tot 65 jaar","65 jaar of

cat = pd.cut(age, bins=bins, labels=labels, right=False, include_lowest=True)
synth_age_pct = cat.value_counts(normalize=True, dropna=True).reindex(labels, fill_value=

# CBS age %
def _to_num(x):
    return pd.to_numeric(str(x).replace(".","").replace(",","."),
                         errors="coerce")

_g = cbs[
    (cbs["Regioaanduiding/Gemeentenaam (naam)"].astype(str).str.strip()=="Rotterdam") &
    (cbs["Regioaanduiding/Soort regio (omschrijving)"].astype(str).str.strip()=="Gemeente
].copy()

total_cbs = _to_num(_g["Bevolking/Aantal inwoners (aantal)"].iloc[0])
age_prefix = "Bevolking/Leeftijdsgroepen/"
cbs_age_pct = {}
for lab in labels:
    col = f"{age_prefix}{lab} (aantal)"
    cbs_age_pct[lab] = float(_to_num(_g[col].iloc[0]) / total_cbs * 100) if col in _g.col
cbs_age_pct = pd.Series(cbs_age_pct)

# table
age_compare = pd.DataFrame({
    "Age_group": labels,
    "CBS_%": [cbs_age_pct.get(l, np.nan) for l in labels],
    "Synth_%": [synth_age_pct.get(l, np.nan) for l in labels],
})
age_compare["Diff"] = age_compare["Synth_%"] - age_compare["CBS_%"]
age_compare = age_compare.round(3)
```

```
print("Age␣%␣(CBS␣vs␣Synth)")
print(age_compare.to_string(index=False))

# Gender
# synth female %
if "persoon_geslacht_vrouw" in _synth.columns:
    synth_female_pct = float(pd.to_numeric(_synth["persoon_geslacht_vrouw"], errors="coer
else:
    synth_female_pct = float("nan")

# CBS female %
VROUW_COL = "Bevolking/Geslacht/Vrouwen␣(aantal)"
if VROUW_COL in _g.columns and total_cbs>0:
    cbs_female_pct = float(_to_num(_g[VROUW_COL].iloc[0]) / total_cbs * 100)
else:
    cbs_female_pct = float("nan")

print("\nGender␣%␣female␣(CBS␣vs␣Synth)")
print(f"CBS:␣{cbs_female_pct:.3f}")
print(f"Synth:␣{synth_female_pct:.3f}")
print(f"Diff:␣{(synth_female_pct␣-␣cbs_female_pct)␣if␣np.isfinite(cbs_female_pct)␣else␣fl
# Maak verschil
diff = (synth_female_pct - cbs_female_pct) if np.isfinite(cbs_female_pct) else float("nan

# Bouw een dataframe
gender_table = pd.DataFrame({
    "Source": ["CBS", "Synth", "Difference"],
    "Female␣%": [cbs_female_pct, synth_female_pct, diff]
})

import matplotlib.pyplot as plt

def save_table_as_image(df, filename, title=""):
    fig, ax = plt.subplots(figsize=(10, len(df) * 0.4 + 1))  # hoogte schaalt mee met aan
    ax.axis("off")
    ax.axis("tight")

    # zet de tabel in de figuur
    table = ax.table(
        cellText=df.round(2).values,
        colLabels=df.columns,
        loc="center",
        cellLoc="center"
    )
    table.auto_set_font_size(False)
    table.set_fontsize(9)
    table.scale(1.2, 1.2)

    if title:
        plt.title(title, fontsize=12, pad=12)

    plt.savefig(filename, bbox_inches="tight", dpi=300)
    plt.close()

# voorbeeld gebruik
save_table_as_image(compare_df, filename="district_compare.png", title="District␣%␣(CBS␣v
```

```
save_table_as_image(gender_table, "gender_tabel.png", title="Geslacht vergelijking")
save_table_as_image(age_compare, "age_tabel.png", title="Leeftijdsverdeling")
```

B.2 Dataset Visualisation, Missing Data Analysis, Distribution checks, and Outlier Detection

```
import sdv
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats

#Make csv file of synth_data with id
synth_df=pd.read_csv("synth_data.csv")
synth_df['id'] = range(1, len(synth_df) + 1)
synth_df.to_csv("synth_data_with_id.csv",index=False)

print('adres_aantal_brp_adres' in synth_df.columns)


#check the type of vriables the features have
synth_df=pd.read_csv("synth_data.csv")
synth_df.info()
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns',None)
print(synth_df.dtypes)
# Remove any leading BOM from column names
synth_df.columns = [col.lstrip('\ufeff') for col in synth_df.columns]
#strip whitespace & normalize
synth_df.columns = synth_df.columns.str.strip()



import sdv
import pkgutil, sdv.single_table as st
print([m.name for m in pkgutil.iter_modules(st.__path__)])

from sdv.single_table.copulas import GaussianCopulaSynthesizer

from sdv.single_table.ctgan import CTGANSynthesizer

from sdv.single_table.copulagan import CopulaGANSynthesizer

#Download to use Gaussian Copula later
from sdv.metadata import Metadata
import pandas as pd


data = pd.read_csv('synth_data.csv')
metadata = Metadata.detect_from_dataframe(data)


gc = GaussianCopulaSynthesizer(metadata)
ct = CTGANSynthesizer(metadata)
cg = CopulaGANSynthesizer(metadata)
```

```
print(gc, ct, cg)




#check if the boolean features really only have values 0 or 1
pure_bool_indices = [
    idx
    for idx, col in enumerate(synth_df.columns)
    # no missing values
    if synth_df[col].notna().all()
    and set(synth_df[col].unique()) == {0, 1}
]


print("Column␣indices␣with␣only␣0/1␣and␣no␣NaN:", pure_bool_indices)
print(len(pure_bool_indices))
print("Columns:", [synth_df.columns[i] for i in pure_bool_indices])

#split data in Boolean or not Boolean
import json
with open('my_metadata_file.json','r') as f:
    metadata=json.load(f)

bools_cols=[
    col for col, props in metadata['fields'].items()
    if props.get('type')=='boolean'
]
non_bools_cols=[col for col in synth_df.columns if col not in bools_cols]


import pandas as pd
import numpy as np

synth_df=pd.read_csv("synth_data.csv")

data =synth_df.values

n, p =data.shape
print(f"Working␣with␣{p}␣features␣over␣{n}␣samples.")


#center and apply Cov=(X^T.X)/(n-1)
means=data.mean(axis=0)
X_c= data - means

#covariance matrix
cov_matrix=(X_c.T @ X_c)/(n-1)

#place in csv file
cov_matrix_df=pd.DataFrame(cov_matrix,index=synth_df.columns, columns=synth_df.columns)
cov_matrix_df.round(5).to_csv('cov_matrix_rounded.csv',index=True)

print('Saved␣covariance␣matrix␣in␣csv␣file')
```

```
#Make the Correlation matrix
std = np.sqrt(np.diag(cov_matrix))

denom=np.outer(std,std)
corr_matrix= cov_matrix/denom

np.fill_diagonal(corr_matrix,1.0)
corr_matrix_df=pd.DataFrame(corr_matrix,index=synth_df.columns, columns=synth_df.columns)

print("Covariance matrix", cov_matrix)
print("Correlation matrix", corr_matrix)
corr_matrix_df.to_csv('corr_matrix.csv', index=True)


from itertools import combinations
pairs = []
for var1, var2 in combinations(corr_matrix_df.columns, 2):
    corr_val=corr_matrix_df.loc[var1,var2]
    pairs.append({
        'Variable 1': var1,
        'Variable 2': var2,
        'AbsCorrelation': abs(corr_val)
    })
pairs_df = pd.DataFrame(pairs)


# 5. Sort and select the top 10
top10 = pairs_df.sort_values('AbsCorrelation', ascending=False).head(10)

print(top10)



#Outlier detection quartiles and boxplots
Q1   = synth_df[non_bools_cols].quantile(0.25)
Q3   = synth_df[non_bools_cols].quantile(0.75)
IQR  = Q3 - Q1

# 4) Define   inner   fences
lower = Q1 - 1.5 * IQR
upper = Q3 + 1.5 * IQR

outlier_flags = (synth_df[non_bools_cols] < lower) | (synth_df[non_bools_cols] > upper)
outlier_counts = outlier_flags.sum().sort_values(ascending=False)

#plot the 20 worst cases
import matplotlib.pyplot as plt
top20 = outlier_counts.head(20)
plt.figure(figsize=(10,6))
plt.barh(top20.index[::-1], top20.values[::-1])
plt.xlabel('Univariate Outlier Count')
plt.title('Top 20 Features by Outlier Count')
plt.tight_layout()
plt.show()
```

```
#plot boxplots
import os
output_dir = 'boxplots'
os.makedirs(output_dir, exist_ok=True)

for col in non_bools_cols:
    fig, ax = plt.subplots(figsize=(6, 4))
    ax.boxplot(synth_df[col].dropna(), vert=True, showfliers=True)
    ax.set_title(col, fontsize=10)
    ax.set_ylabel(col)
    fig.tight_layout()

    safe_col = "".join(c if c.isalnum() or c in "_-" else "_" for c in col)
    fig.savefig(os.path.join(output_dir, f'{safe_col}.png'))
    plt.close(fig)

print(f"Saved␣{len(non_bools_cols)}␣boxplots␣to␣the␣folder:␣{output_dir}/")


#Make boxplots
import math
metrics = pd.DataFrame({
    'outlier_count': outlier_counts,
    'IQR': IQR
})
def plot_top_k(top_k=20, sort_by='outlier_count'):
    cols=min(10,top_k)
    rows=math.ceil(top_k/cols)

    selected=metrics.sort_values(by=sort_by,ascending=False).index[:top_k]
    fig, axes = plt.subplots(rows, cols, figsize=(cols*2, rows*2.5), sharey=False)
    axes = axes.flatten()
    for ax, col in zip(axes, selected):
        ax.boxplot(synth_df[col].dropna(), vert=True, showfliers=True)
        ax.set_title(col, fontsize=8)
        ax.tick_params(axis='both', labelsize=6)

    # remove any extra axes
    for ax in axes[top_k:]:
        fig.delaxes(ax)

    plt.tight_layout()
    plt.show()
plot_top_k(20, sort_by='outlier_count')

#onderzoeken reden boxplot een lijn
top20 = outlier_counts.index[:20]

summary = pd.DataFrame({
    'Q1':  Q1.loc[top20],
    'Q3':  Q3.loc[top20],
    'IQR': IQR.loc[top20]
})
print(summary)

# Select the 20 features with the smallest outlier counts
```

```
bottom20 = outlier_counts.sort_values(ascending=True).head(20)

plt.figure(figsize=(10, 6))
plt.barh(bottom20.index[::-1], bottom20.values[::-1])
plt.xlabel('Univariate Outlier Count')
plt.title('Bottom 20 Features by Outlier Count')
plt.tight_layout()
plt.show()


# Filter to variables with at least one outlier
positive_counts = outlier_counts[outlier_counts > 0]

bottom20_pos = positive_counts.sort_values(ascending=True).head(20)

plt.figure(figsize=(10,6))
plt.barh(bottom20_pos.index[::-1], bottom20_pos.values[::-1])
plt.xlabel('Univariate Outlier Count')
plt.title('20 Features with Smallest Non-Zero Outlier Counts')
plt.tight_layout()
plt.show()


def plot_bottom_k(bottom_k=20, sort_by='outlier_count'):
    filtered = metrics[metrics['outlier_count'] > 0]
    selected = filtered.sort_values(by=sort_by, ascending=True).index[:bottom_k]

    cols = min(10, bottom_k)
    rows = math.ceil(bottom_k / cols)
    fig, axes = plt.subplots(rows, cols, figsize=(cols * 2, rows * 2.5), sharey=False)
    axes = axes.flatten()
    for ax, col in zip(axes, selected):
        ax.boxplot(synth_df[col].dropna(), vert=True, showfliers=True)
        ax.set_title(col, fontsize=6, y=1.08)
        ax.tick_params(axis='both', labelsize=5)

    # drop any extra axes
    for ax in axes[bottom_k:]:
        fig.delaxes(ax)

    plt.tight_layout()
    fig.subplots_adjust(top=0.88, hspace=0.5, wspace=0.4)
    plt.show()

plot_bottom_k(bottom_k=20, sort_by='outlier_count')

#Get the row numbers of the outliers
df = synth_df.reset_index()

bottom_k = 20
filtered  = metrics[metrics['outlier_count'] > 0]
selected  = filtered.sort_values('outlier_count', ascending=True).index[:bottom_k]


outlier_rows = []
for col in selected:
```

```
    mask = outlier_flags[col].values
    zero_idx = df.index[mask].tolist()

    excel_rows = [i + 2 for i in zero_idx]
    for z, e in zip(zero_idx, excel_rows):
        outlier_rows.append({'feature': col,
                             'zero_based_idx': z,
                             'excel_row':      e})

outlier_pos_df = pd.DataFrame(outlier_rows)
print(outlier_pos_df.head(20))




from scipy.stats import chi2
from sklearn.preprocessing import StandardScaler

#standardize
scaler     = StandardScaler()
X_scaled   = scaler.fit_transform(synth_df[non_bools_cols])

#cov matrix and inverse
cov=np.cov(X_scaled,rowvar=False)
inv_cov=np.linalg.pinv(cov)

#compute mahalanobis distance
diff       = X_scaled - X_scaled.mean(axis=0)
md2        = np.sum(diff.dot(inv_cov) * diff, axis=1)
synth_df['MD']   = np.sqrt(md2)
synth_df['id'] = range(1, len(synth_df) + 1)
synth_df.to_csv("synth_data_with_MD.csv",index=False)

#Compute threshold
p             = X_scaled.shape[1]
threshold_md  = np.sqrt(chi2.ppf(0.975, df=p))
synth_df['md_outlier'] = synth_df['MD'] > threshold_md


print(f"Drempel (97.5%): {threshold_md:.2f}")
print(synth_df['MD'].describe())
print("Top 20 hoogste MD-waardes:")
print(synth_df[['MD','md_outlier']].sort_values('MD', ascending=False).head(20))

plt.figure(figsize=(8,4))
plt.hist(synth_df['MD'], bins=50, alpha=0.7)
plt.axvline(threshold_md, color='red', linestyle='--', label='97.5% drempel')
plt.xlabel('Mahalanobis-distance')
plt.ylabel('Number of observations')
plt.legend()
plt.tight_layout()
plt.show()

#Which id numbers gave outliers?
md_zero_idx = np.where(synth_df['md_outlier'])[0]
```

```
all_zero_based=md_zero_idx.tolist()

all_excel_rows = [i + 2 for i in all_zero_based]
print("Excel row numbers of MD-outliers:", all_excel_rows)



#try different threshikd
p = X_scaled.shape[1]
for pct in [0.975, 0.99, 0.995, 0.999]:
    thr = np.sqrt(chi2.ppf(pct, df=p))
    count = (synth_df['MD'] > thr).sum()
    print(f">{int(pct*100)}% cutoff (D > {thr:.2f}): {count} outliers")


#find MD elbow
Ds = np.sort(synth_df['MD'].values)
plt.plot(Ds)
plt.ylabel('Mahalanobis distance')
plt.xlabel('Sorted observation index')
plt.axhline(Ds[int(0.926*len(Ds))], linestyle='--', label='92.6th percentile')
plt.legend()
plt.show()

fig, ax = plt.subplots(figsize=(6, 4))  # Optional: specify figure size

Ds = np.sort(synth_df['MD'].values)
ax.plot(Ds)
ax.set_ylabel('Mahalanobis distance')
ax.set_xlabel('Sorted observation index')
ax.axhline(Ds[int(0.926 * len(Ds))], linestyle='--', label='92.6th percentile')
ax.legend()

# Save the figure to a file
fig.tight_layout()
fig.savefig("Mahalanobis_elbow_plot.png", dpi=300)  # or use .pdf / .svg if you prefer
plt.show()

#Sort MD values
import pandas as pd
import numpy as np

# Laad je dataset met Mahalanobis-afstanden
synth_df = pd.read_csv("synth_data_with_MD.csv")

# Stap 1: sorteer Mahalanobis-afstanden
Ds = np.sort(synth_df['MD'].values)
n = len(Ds)
x = np.arange(n)

# Stap 2: rechte lijn tussen begin en eind
p1 = np.array([0,    Ds[0]])
p2 = np.array([n-1,  Ds[-1]])

# Stap 3: afstand tot de lijn berekenen
num = np.abs((p2[1]-p1[1])*x - (p2[0]-p1[0])*Ds + p2[0]*p1[1] - p2[1]*p1[0])
```

```
den = np.hypot(p2[1]-p1[1], p2[0]-p1[0])
dist_to_line = num / den

# Stap 4: vind de 'elbow'
elbow_idx = dist_to_line.argmax()
elbow_MD  = Ds[elbow_idx]
print(f"Elbow␣at␣sorted␣index␣{elbow_idx},␣MD␣   ␣{elbow_MD:.4f}")

# Stap 5: selecteer alleen de inliers (MD <= elbow)
synth_inliers = synth_df[synth_df["MD"] <= elbow_MD].copy()
print(f"Aantal␣inliers:␣{len(synth_inliers)}␣van␣{len(synth_df)}␣({100␣*␣len(synth_inlier

# Stap 6: opslaan
synth_inliers.to_csv("synth_data_MD_inliers.csv", index=False)


#Change data set to filter out outliers
synth_df['id'] = np.arange(1, len(df) + 1)
cols = ['id'] + [c for c in synth_df.columns if c != 'id']
synth_df = synth_df[cols]
synth_df.to_csv('synth_data_with_id.csv', index=False)

#Change data set to filter out outliers
df_mahalnobis_out=synth_df.loc[~synth_df['md_elbow_outlier']].copy()
df_mahalnobis_out.reset_index(drop=True, inplace=True)
df_mahalnobis_out.to_csv("synth_df_mahalanobis_out.csv", index=False)




synth_df = pd.read_csv('synth_data.csv', encoding='utf-8', index_col=None)
synth_df.columns = [col.lstrip('\ufeff') for col in synth_df.columns]
synth_df.columns = synth_df.columns.str.strip()

import json
with open('my_metadata_file.json') as f:
    metadata = json.load(f)['fields']
categorical_cols = [c for c,info in metadata.items() if info['type']=='categorical']
numerical_cols = [c for c,info in metadata.items() if info['type']=='numerical']
boolean_cols   = [c for c,info in metadata.items() if info['type']=='boolean']


results = []

for col in numerical_cols:
    x = synth_df[col].values
    n = len(x)

    # Mean and median
    mean = x.sum() / n
    median = np.median(x)

    # Sample variance and std dev
    var = ((x - mean)**2).sum() / (n - 1)
    std = np.sqrt(var)
```

```
    #Central moments
    m2 = ((x - mean)**2).sum() / n
    m3 = ((x - mean)**3).sum() / n
    m4 = ((x - mean)**4).sum() / n

    #Skewness and excess kurtosis
    skewness = m3 / (m2**1.5)
    kurtosis = m4 / (m2**2) - 3

    results.append({
        'feature': col,
        'mean': mean,
        'median': median,
        'variance(ddof=1)': var,
        'skewness': skewness,
        'excess_kurtosis': kurtosis
    })

# turn into a DataFrame for easy viewing
manual_desc = pd.DataFrame(results).set_index('feature')

#put in csv
manual_desc.to_csv('information_num_features.csv', index=True)

#round the input of the csv
rounded_desc = manual_desc.round(3)
rounded_desc.to_csv('rounded_3dec_info_num_features.csv', index=True)

import seaborn as sns
desc = pd.read_csv('information_num_features.csv', index_col=0)


desc['mean_med_diff'] = (desc['mean'] - desc['median']).abs()

# Get the  t o p 20  features by each metric
top_by_diff = desc.sort_values('mean_med_diff', ascending=False).head(20)
top_by_skew = desc.reindex(desc['skewness'].abs().sort_values(ascending=False).index).hea
top_by_kurt = desc.sort_values('excess_kurtosis', ascending=False).head(20)

print("Top␣20␣by␣| m e a n median |␣gap:\n", top_by_diff[['mean','median','mean_med_diff']],
print("Top␣20␣by␣|skewness|:\n", top_by_skew[['skewness']], "\n")
print("Top␣20␣by␣excess␣kurtosis:\n", top_by_kurt[['excess_kurtosis']], "\n")

#Check how correlated these three diagnostics are
metrics = desc[['mean_med_diff','skewness','excess_kurtosis']]
corr = metrics.corr()
print("Correlation␣matrix:\n", corr)

#Plot the top 20 histograms of each
def plot_and_save(feature_list, df, title, filename,ncols=5):
    n = len(feature_list)
    nrows = math.ceil(n / ncols)
    fig, axes = plt.subplots(nrows, ncols, figsize=(ncols*3, nrows*2.5))
    axes = axes.flatten()
    for ax, feat in zip(axes, feature_list):
        sns.histplot(df[feat].dropna(), bins=30, ax=ax)
```

```
        ax.set_title(feat, fontsize=8)
        ax.set_xlabel('')
        ax.set_ylabel('')
    # turn off any extra subplots
    for ax in axes[n:]:
        ax.axis('off')
    fig.suptitle(title, fontsize=14, y=1.02)
    plt.tight_layout(rect=[0,0,1,0.96])
    fig.savefig(filename,dpi=300)
    plt.close(fig)
    print(f"Saved:{filename}")

plot_and_save(top_by_diff.index.tolist(), synth_df, 'Top␣20␣Features␣by␣| M e a n Median |␣C
plot_and_save(top_by_skew.index.tolist(), synth_df, 'Top␣20␣Features␣by␣|Skewness|','top2
plot_and_save(top_by_kurt.index.tolist(), synth_df, 'Top␣20␣Features␣by␣Excess␣Kurtosis',

import seaborn as sns

#Make histograms and save them in separate folder
base_dir = 'histograms'
for sub in ['numerical','boolean','categorical']:
    os.makedirs(os.path.join(base_dir, sub), exist_ok=True)


#Numerical histograms
for col in numerical_cols:
    fig, ax = plt.subplots(figsize=(6,4))
    sns.histplot(synth_df[col].dropna(), bins=30, kde=False, ax=ax)
    ax.set_title(col, fontsize=10)
    ax.set_xlabel(col)
    ax.set_ylabel('Count')
    fig.tight_layout()

    safe = "".join(c if c.isalnum() or c in "_-" else "_" for c in col)
    fig.savefig(os.path.join(base_dir, 'numerical', f'{safe}.png'))
    plt.close(fig)

#Boolean countplots
for col in boolean_cols:
    fig, ax = plt.subplots(figsize=(4,4))
    sns.countplot(x=synth_df[col], order=[0,1],ax=ax)
    ax.set_title(col, fontsize=10)
    ax.set_xlabel(col)
    ax.set_ylabel('Count')
    fig.tight_layout()

    safe = "".join(c if c.isalnum() or c in "_-" else "_" for c in col)
    fig.savefig(os.path.join(base_dir, 'boolean', f'{safe}.png'))
    plt.close(fig)

#Categorical countplots
for col in categorical_cols:
    fig, ax = plt.subplots(figsize=(max(6, 0.3 * len(synth_df[col].unique())), 4))
    levels = sorted(synth_df[col].dropna().unique())
    sns.countplot(x=synth_df[col], order=levels, ax=ax)
    ax.set_title(col, fontsize=10)
```

```
    ax.set_xlabel(col)
    ax.set_ylabel('Count')
    plt.xticks(rotation=90, ha='right')
    fig.tight_layout()

    safe = "".join(c if c.isalnum() or c in "_-" else "_" for c in col)
    fig.savefig(os.path.join(base_dir, 'categorical', f'{safe}.png'))
    plt.close(fig)

print(
    "Saved histograms:\n"
    f"   {len(numerical_cols)} numeric      {base_dir}/numerical/\n"
    f"   {len(boolean_cols)} boolean     {base_dir}/boolean/\n"
    f"   {len(categorical_cols)} categorical     {base_dir}/categorical/"
)


from scipy.stats import gaussian_kde
base_dir = 'KDE + histograms'
for sub in ['numerical']:
    os.makedirs(os.path.join(base_dir, sub), exist_ok=True)

# estimate the KDE and save in separate file
for col in numerical_cols:
    x = synth_df[col].dropna().values

    safe = "".join(c if c.isalnum() or c in "_-" else "_" for c in col)

    fig, ax = plt.subplots(figsize=(6,4))
    counts, bins, _ = ax.hist(x, bins=30, density=True, alpha=0.4, label="histogram")
    kde = gaussian_kde(x, bw_method=0.4)
    xs = np.linspace(x.min(), x.max(), 200)
    ax.plot(xs, kde(xs), lw=2, label="KDE (bw=0.5)")

    ax.set_title(f"{col} distribution")
    ax.set_xlabel(col)
    ax.set_ylabel("Density")
    ax.legend()
    fig.tight_layout()

    fig.savefig(os.path.join(base_dir, f"{safe}.png"))
    plt.close(fig)


#QQ plots
import scipy.stats as stats
base_dir = 'QQ plots'
for sub in ['numerical']:
    os.makedirs(os.path.join(base_dir, sub), exist_ok=True)

for col in numerical_cols:
    x = synth_df[col].dropna().values
    safe = "".join(c if c.isalnum() or c in "_-" else "_" for c in col)

    fig, ax = plt.subplots(figsize=(5,5))
    stats.probplot(x, dist="norm", plot=ax)
```

```
    ax.set_title(f" Q Q ␣for␣{col}")
    ax.set_xlabel(col)
    ax.set_ylabel("Ordered␣values")
    fig.tight_layout()

    fig.savefig(os.path.join(base_dir, f"{safe}.png"))
    plt.close(fig)

#thresholds
low_variance_threshold = 0.01
high_skew_threshold = 1.0

bool_cols = [
    col for col in synth_df.columns
    if synth_df[col].dropna().unique().tolist() in ([True, False], [False, True], [1, 0],
    or synth_df[col].dtype == 'bool'
]
non_bool_cols = [col for col in synth_df.columns if col not in bool_cols]

binary_vars = [col for col in non_bool_cols if synth_df[col].nunique() <= 2]
cols_in_desc = [col for col in manual_desc.index if col not in binary_vars]
vars_low_variance = manual_desc.loc[cols_in_desc]
vars_low_variance = vars_low_variance[vars_low_variance['variance(ddof=1)'] < low_varianc


vars_high_skew = manual_desc.loc[cols_in_desc]
vars_high_skew = vars_high_skew[vars_high_skew['skewness'].abs() > high_skew_threshold].i
vars_log1p_pca = [
    'afspraak_afgelopen_jaar_voortgang_aanmelding_en_deelname',
    'afspraak_laatstejaar_aantal_woorden',
    'persoonlijke_eigenschappen_dagen_sinds_opvoer',
    'relatie_partner_totaal_dagen_partner'
]
vars_pca_candidate = list(set(non_bool_cols) - set(vars_low_variance)-set(vars_high_skew)

print("The␣pca␣candidates␣are:␣", vars_pca_candidate)



#plot histogram, boxplot, QQ plot for the 4 categorical features:
cat_features = [
    "persoonlijke_eigenschappen_spreektaal",                    # feature 244
    "persoonlijke_eigenschappen_taaleis_voldaan",               # feature 247
    "persoonlijke_eigenschappen_uitstroom_verw_vlgs_klant",  # feature 249
    "persoonlijke_eigenschappen_uitstroom_verw_vlgs_km"      # feature 250
]
fig, axes = plt.subplots(2, 2, figsize=(12, 8))
fig.suptitle("Distributions␣of␣Categorical␣Features", fontsize=16)

#Histograms
for i, feature in enumerate(cat_features):
    ax = axes[i // 2, i % 2]
    value_counts = synth_df[feature].value_counts(dropna=False).sort_index()
    value_counts.plot(kind='bar', ax=ax, color='skyblue', edgecolor='black')
    ax.set_title(feature.replace("_", "␣").capitalize(), fontsize=10)
```

```
    ax.set_xlabel("Category")
    ax.set_ylabel("Count")

plt.tight_layout(rect=[0, 0.03, 1, 0.95])

plt.savefig("categorical_features_distribution.png", dpi=300)
plt.show()

#Boxplots
for i, feature in enumerate(cat_features):
    plt.subplot(2, 2, i + 1)
    sns.boxplot(y=synth_df[feature], color="skyblue")
    plt.title(feature.replace("_", " ").capitalize())
    plt.xlabel("")
    plt.ylabel("Value")

plt.tight_layout()
plt.savefig("boxplots_categorical_features_raw.png", dpi=300)
plt.show()

#QQ plots
for i, feature in enumerate(cat_features):
    plt.subplot(2, 2, i + 1)
    stats.probplot(synth_df[feature].dropna(), dist="norm", plot=plt)
    plt.title(" Q Q  plot: " + feature.replace("_", " ").capitalize())
    plt.xlabel("Theoretical Quantiles")
    plt.ylabel("Sample Quantiles")

plt.tight_layout()
plt.savefig("qqplots_categorical_features.png", dpi=300)
plt.show()
```

B.3 Cross-validation MD Weights

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import Ridge
from sklearn.model_selection import cross_validate
from sklearn.preprocessing import StandardScaler

#Read the dataset
df = pd.read_csv("synth_data_with_MD_and_pseudo.csv")

#Features and target
X = df.drop(columns=["id", "MD", "Ja_pseudo"], errors="ignore")
y = df["Ja_pseudo"]

#Feature scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

#Normalise MD
md_scaled = (df["MD"] - df["MD"].min()) / (df["MD"].max() - df["MD"].min())
```

```
#Weight strategies
def weight_linear(md): return 1 - md
def weight_exp(md): return np.exp(-2 * md)
def weight_logistic(md): return 1 / (1 + np.exp(5 * (md - 0.5)))

strategies = {
    "Linear (1 - MD)": weight_linear,
    "Exponential (exp(-2MD))": weight_exp,
    "Logistic": weight_logistic
}

#Cross-validation weight strategies with ridge
results = {}
for name, func in strategies.items():
    weights = func(md_scaled)
    model = Ridge()

    scores = cross_validate(
        estimator=model,
        X=X_scaled,
        y=y,
        scoring="neg_mean_squared_error",
        cv=5,
        fit_params={"sample_weight": weights},
        return_train_score=False
    )

    results[name] = -np.mean(scores["test_score"])  # lower MSE = better

#print and plot results
for name, mse in results.items():
    print(f"{name}: MSE = {mse:.5f}")
max_mse = max(results.values())
min_mse = min(results.values())
plt.figure(figsize=(8, 5))
plt.bar(results.keys(), results.values(), color='skyblue')
plt.axhline(y=max_mse, color='red', linestyle='--', linewidth=1)
padding = 0.00001
plt.ylim(min_mse - padding, max_mse + padding)
plt.ylabel("Mean Squared Error (MSE)")
plt.title("Comparison of weighting strategies (Ridge, 5-fold CV)")
plt.grid(True)
plt.xticks(rotation=20)
plt.tight_layout()

plt.savefig("comparison_weighting_strategies.png", dpi=300, bbox_inches="tight")
plt.show()


#Conclusie: lineair MD is the best one to use
```

B.4 PCA feature selection

```
#PCA candidates
import pandas as pd
import numpy as np
```

```
import matplotlib.pyplot as plt

synth_df=pd.read_csv('synth_data.csv', index_col=False)

desc = pd.read_csv('information_num_features.csv', index_col=False)
results = []
import json
with open('my_metadata_file.json') as f:
    metadata = json.load(f)['fields']
categorical_cols = [c for c,info in metadata.items() if info['type']=='categorical']
numerical_cols = [c for c,info in metadata.items() if info['type']=='numerical']
boolean_cols   = [c for c,info in metadata.items() if info['type']=='boolean']

for col in numerical_cols:
    x = synth_df[col].values
    n = len(x)

    # Mean and median
    mean = x.sum() / n
    median = np.median(x)

    # Sample variance and std dev
    var = ((x - mean)**2).sum() / (n - 1)
    std = np.sqrt(var)

    #Central moments
    m2 = ((x - mean)**2).sum() / n
    m3 = ((x - mean)**3).sum() / n
    m4 = ((x - mean)**4).sum() / n

    #Skewness and excess kurtosis
    skewness = m3 / (m2**1.5)
    kurtosis = m4 / (m2**2) - 3

    results.append({
        'feature': col,
        'mean': mean,
        'median': median,
        'variance(ddof=1)': var,
        'skewness': skewness,
        'excess_kurtosis': kurtosis
    })

# turn into a DataFrame for easy viewing
manual_desc = pd.DataFrame(results).set_index('feature')

#put in csv
manual_desc.to_csv('information_num_features.csv', index=True)

#round the input of the csv
rounded_desc = manual_desc.round(3)
rounded_desc.to_csv('rounded_3dec_info_num_features.csv', index=True)
#thresholds
low_variance_threshold = 0.01
high_skew_threshold = 1.0
```

```
bool_cols = [
    col for col in synth_df.columns
    if synth_df[col].dropna().unique().tolist() in ([True, False], [False, True], [1, 0],
    or synth_df[col].dtype == 'bool'
]
non_bool_cols = [col for col in synth_df.columns if col not in bool_cols]

binary_vars = [col for col in non_bool_cols if synth_df[col].nunique() <= 2]
cols_in_desc = [col for col in manual_desc.index if col not in binary_vars]
vars_low_variance = manual_desc.loc[cols_in_desc]
vars_low_variance = vars_low_variance[vars_low_variance['variance(ddof=1)'] < low_varianc


vars_high_skew = manual_desc.loc[cols_in_desc]
vars_high_skew = vars_high_skew[vars_high_skew['skewness'].abs() > high_skew_threshold].i
vars_log1p_pca = [
    'afspraak_afgelopen_jaar_voortgang_aanmelding_en_deelname',
    'afspraak_laatstejaar_aantal_woorden',
    'persoonlijke_eigenschappen_dagen_sinds_opvoer',
    'relatie_partner_totaal_dagen_partner'
]
vars_pca_candidate = list(set(non_bool_cols) - set(vars_low_variance)-set(vars_high_skew)

print("The␣pca␣candidates␣are:␣", vars_pca_candidate)



#open boxplot en histogram voor elk van deze scheve variabelen
import os
print("The␣variables␣with␣big␣skew␣are:␣",vars_high_skew)
boxplot_dir = 'boxplots'
histogram_dir = 'histograms/numerical'
for var in vars_high_skew:
    safe_var = "".join(c if c.isalnum() or c in "_-" else "_" for c in var)

    boxplot_path = os.path.join(boxplot_dir, f"{safe_var}.png")
    hist_path = os.path.join(histogram_dir, f"{safe_var}.png")

    fig, axs = plt.subplots(1, 2, figsize=(10, 4))

    # Boxplot
    if os.path.exists(boxplot_path):
        img_box = plt.imread(boxplot_path)
        axs[0].imshow(img_box)
        axs[0].axis('off')
        axs[0].set_title(f"Boxplot␣  ␣{var}")
    else:
        axs[0].text(0.5, 0.5, 'Boxplot␣not␣found', ha='center', va='center')
        axs[0].axis('off')

    # Histogram
    if os.path.exists(hist_path):
        img_hist = plt.imread(hist_path)
        axs[1].imshow(img_hist)
        axs[1].axis('off')
        axs[1].set_title(f"Histogram␣  ␣{var}")
```

```
    else:
        axs[1].text(0.5, 0.5, 'Histogram not found', ha='center', va='center')
        axs[1].axis('off')

    plt.tight_layout()
    plt.show()
#conclusie: houd alleen  'afspraak_afgelopen_jaar_voortgang_aanmelding_en_deelname','afsp

#log1p() transformation
vars_log1p_pca = [
    'afspraak_afgelopen_jaar_voortgang_aanmelding_en_deelname',
    'afspraak_laatstejaar_aantal_woorden',
    'persoonlijke_eigenschappen_dagen_sinds_opvoer',
    'relatie_partner_totaal_dagen_partner'
]
synth_df_transformed = pd.DataFrame()
synth_df = synth_df.copy()
synth_df['id'] = synth_df.index + 1
ids = synth_df['id']
for col in vars_log1p_pca:
    if col in synth_df.columns:
        synth_df_transformed[f"{col}_log"] = np.log1p(synth_df[col])
print (synth_df_transformed)

from sklearn.preprocessing import StandardScaler
df = synth_df.copy()
for col in vars_log1p_pca:
    if col in df.columns:
        df[col] = np.log1p(df[col])

vars_pca_candidate = list(
    (set(non_bool_cols) - set(vars_low_variance) - set(vars_high_skew))
    | set(vars_log1p_pca)
)
vars_pca_candidate = [col for col in vars_pca_candidate if col != 'id']
X = df[vars_pca_candidate].dropna()
ids = ids.loc[X.index]
# Scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_scaled_df = pd.DataFrame(X_scaled, columns=vars_pca_candidate, index=X.index)
X_scaled_df.insert(0, 'id', ids)
correlation_matrix = X_scaled_df.drop(columns='id').corr().abs()
print(correlation_matrix)
X_scaled_df.to_csv("X_scaled_for_R.csv", index=False)


upper_triangle = correlation_matrix.where(np.triu(np.ones(correlation_matrix.shape), k=1)

# Detecteer sterk gecorreleerde variabelen
highly_corr_vars = [col for col in upper_triangle.columns if any(upper_triangle[col] > 0.
print(highly_corr_vars)

#conclusion: we can go further with PCA with the given variable list
```

```
from sklearn.decomposition import PCA
pca = PCA()
X_pca = pca.fit_transform(X_scaled_df)
loadings = pd.DataFrame(pca.components_.T,
                        columns=[f"PC{i+1}" for i in range(pca.n_components_)],
                        index=X_scaled_df.columns)

# Categorie n toewijzen op basis van kolomnaam
def label_category(varname):
    if varname.startswith("competentie_"):
        return "Competenties"
    elif varname.startswith("contacten_"):
        return "Contacten"
    elif varname.startswith("instrument_"):
        return "Instrumenten"
    elif varname.startswith("afspraak_"):
        return "Afspraken"
    elif varname.startswith("belemmering_"):
        return "Belemmeringen"
    elif varname.startswith("relatie_"):
        return "Relaties"
    elif varname.startswith("persoonlijke_eigenschappen_"):
        return "Persoonlijk"
    elif varname.startswith("adres_"):
        return "Adres"
    elif varname.startswith("pla_"):
        return "Pla"
    elif varname.endswith("_log"):
        return "Getransformeerd"
    else:
        return "Overig"

loadings["Categorie"] = loadings.index.map(label_category)

# Top 5 bijdragen per categorie per component
for pc in ['PC1', 'PC2']:
    print(f"\n␣Top␣5␣variables␣in␣{pc}␣per␣categorie:")
    for cat in loadings["Categorie"].unique():
        top_vars = loadings[loadings["Categorie"] == cat][pc].abs().sort_values(ascending
        if not top_vars.empty:
            print(f"\n␣{cat}:")
            print(top_vars)

# Kolomnamen als DataFrame
cols_df = pd.DataFrame(X.columns, columns=["variable"])

# Opslaan als CSV
cols_df.to_csv("pca_variable_names.csv", index=False)
```

C R code

C.1 PCA and Clustering

```
    # PCA Analysis and Clustering
```

```
library(tidyverse)
library(ggplot2)
library(FactoMineR)
library(factoextra)
library(inflection)
library(cluster)

setwd("/Users/catootjeversluis/suspicion_machine")

df <- read.csv("data/01_raw/synth_data_with_id.csv", stringsAsFactors = FALSE)
X_scaled <- read.csv("data/02_intermediate/X_scaled_for_R.csv")
scores <- read.csv("results/risk_scores_comparison.csv")

scores_subset <- scores[, c('id', 'Ja_pseudo')]
threshold <- quantile(scores_subset$Ja_pseudo, probs = 0.90, na.rm = TRUE)
top10_ids <- scores_subset$id[scores$Ja_pseudo >= threshold]
merged <- merge(X_scaled, scores_subset, by = 'id')
features <- merged[, !(names(merged) %in% c("id", "Ja_pseudo"))]

extract_archetype_from_variable_list <- function(ini_file, df) {
  cfg <- read.config(ini_file)
  if (!"VARIABLES" %in% names(cfg)) return(rep(0, nrow(df)))
  varlist_raw <- cfg$VARIABLES$variable_list
  varlist <- tryCatch(fromJSON(gsub("[""']", '"', varlist_raw)), error = function(e)
  if (is.null(varlist)) return(rep(0, nrow(df)))
  mask <- rep(TRUE, nrow(df))
  for (group in varlist) {
    condition <- group[[1]]
    varname <- names(condition)
    values <- unlist(condition)
    if (!is.null(varname) && length(varname) == 1 && varname %in% colnames(df)) {
      if ("ALL" %in% values) next
      mask <- mask & df[[varname]] %in% values
    } else {
      mask <- mask & FALSE
    }
  }
  as.integer(mask)
}

arch_paths <- list(
  arch_combined_max = "conf/archetypes/arch_combined_max.ini",
  arch_combined_min = "conf/archetypes/arch_combined_min.ini",
  migrant_worker_comment = "conf/archetypes/arch_migrant_worker_comment_no_comment.in
  migrant_worker_language = "conf/archetypes/arch_migrant_worker_language.ini",
  migrant_worker_language_all = "conf/archetypes/arch_migrant_worker_language_all.ini
  mother = "conf/archetypes/arch_single_mother.ini",
  mother_age = "conf/archetypes/arch_single_mother_age.ini",
  bad_dutch = "conf/archetypes/bad_dutch.ini",
  good_dutch = "conf/archetypes/good_dutch.ini",
  language_requirement = "conf/archetypes/language_requirement.ini",
  parent = "conf/archetypes/parent.ini"
)

# Add archetype columns to dataframe
for (name in names(arch_paths)) {
```

```
  df[[name]] <- extract_archetype_from_variable_list(arch_paths[[name]], df)
}

# Create single archetype label per row
df$archetype <- apply(df[names(arch_paths)], 1, function(row) {
  lab <- names(which(row == 1))
  if (length(lab) == 0) return("overig")
  return(lab[1])
})
df$archetype <- as.factor(df$archetype)

# Count observations per archetype
archetype_counts <- table(df$archetype)
archetype_percentages <- round(archetype_counts / sum(archetype_counts) * 100, 2)

cat("Archetype␣distribution:\n")
print(archetype_counts)

# Save archetype summary
archetype_summary <- data.frame(
  archetype = names(archetype_counts),
  count = as.numeric(archetype_counts),
  percentage = as.numeric(archetype_percentages)
)
write.csv(archetype_summary, "pca_plots/archetype_counts.csv", row.names = FALSE)

# Create Excel-friendly format
archetype_excel <- data.frame(
  Archetype = names(archetype_counts),
  Count = as.numeric(archetype_counts),
  Percentage = as.numeric(archetype_percentages),
  stringsAsFactors = FALSE
)
write.csv(archetype_excel, "pca_plots/archetype_distribution_for_excel.csv", row.name

# Create visualizations
archetype_plot <- ggplot(archetype_summary, aes(x = reorder(archetype, -count), y = c
  geom_col() +
  geom_text(aes(label = sprintf("%d\n(%.1f%%)", count, percentage)),
            vjust = -0.5, size = 3, lineheight = 0.8) +
  scale_fill_viridis_d() +
  labs(title = "Archetype␣Distribution␣in␣Dataset",
       subtitle = paste("Total␣observations:", sum(archetype_counts)),
       x = "Archetype", y = "Number␣of␣Observations") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1), legend.position = "none")

ggsave("pca_plots/archetype_distribution_plot.png", plot = archetype_plot, width = 12

# Create pie chart
pie_plot <- ggplot(archetype_summary, aes(x = "", y = count, fill = archetype)) +
  geom_bar(stat = "identity", width = 1) +
  coord_polar("y", start = 0) +
  geom_text(aes(label = sprintf("%.1f%%", percentage)),
            position = position_stack(vjust = 0.5), size = 3) +
  scale_fill_viridis_d() +
```

```
  labs(title = "Archetype Distribution (Pie Chart)",
       subtitle = paste("Total observations:", sum(archetype_counts))) +
  theme_minimal() +
  theme(axis.text = element_blank(), axis.title = element_blank(), panel.grid = eleme

ggsave("pca_plots/archetype_distribution_pie.png", plot = pie_plot, width = 10, heigh

# Perform PCA on scaled features
res.pca <- prcomp(features, scale. = FALSE)
pca_scores <- as.data.frame(res.pca$x)
pca_scores$archetype <- df$archetype

# Create directories for plots
dir.create("pca_plots", showWarnings = FALSE)
dir.create("pca_plots/pca_plots_archetypes", showWarnings = FALSE)

# Scree plot
scree_plot <- fviz_eig(res.pca, addlabels = TRUE, ylim = c(0, 50))
ggsave("pca_plots/scree_plot.png", plot = scree_plot, width = 8, height = 6)

# Variable contribution plot
var_plot <- fviz_pca_var(res.pca, col.var = "contrib")
ggsave("pca_plots/variable_contrib_plot.png", plot = var_plot, width = 8, height = 6)

# Filtered variable plot with top contributors
contrib_scores <- get_pca_var(res.pca)$contrib[, 1:2]
contrib_total <- rowSums(contrib_scores)
threshold <- quantile(contrib_total, 0.8)
top_contrib_vars <- names(which(contrib_total >= threshold))

var_plot_filtered <- fviz_pca_var(res.pca,
                                  col.var = "contrib",
                                  select.var = list(contrib = length(top_contrib_vars
                                  repel = TRUE) +
  labs(title = paste("Variable contributions (Top", length(top_contrib_vars), "variab
       subtitle = paste("Threshold:", round(threshold, 2)))

ggsave("pca_plots/variable_contrib_plot_filtered.png", plot = var_plot_filtered, widt

# PC2 positive contributors
loadings <- res.pca$rotation[, 1:2]
contrib_scores <- get_pca_var(res.pca)$contrib[, 1:2]
pc_df <- data.frame(
  variable = rownames(loadings),
  PC1_loading = loadings[, 1],
  PC2_loading = loadings[, 2],
  PC2_contrib = contrib_scores[, "Dim.2"]
)
pc2_upward_vars <- subset(pc_df, PC2_loading > 0)
pc2_upward_vars_sorted <- pc2_upward_vars[order(-pc2_upward_vars$PC2_contrib), ]
top_n <- 20
selected_vars <- pc2_upward_vars_sorted$variable[1:min(top_n, nrow(pc2_upward_vars_so

var_plot_pc2_up <- fviz_pca_var(res.pca,
                                col.var = "contrib",
                                select.var = list(name = selected_vars),
```

```
                                repel = TRUE) +
  labs(title = paste("Top", length(selected_vars), "Variables␣contributing␣positively
       subtitle = "These␣variables␣align␣with␣the␣upper␣region␣of␣the␣PCA␣space␣(high
  theme(plot.title = element_text(size = 12, face = "bold"))

ggsave("pca_plots/variable_contrib_plot_PC2_positive_filtered.png", plot = var_plot_p

# Cumulative variance plot
var_explained <- res.pca$sdev^2 / sum(res.pca$sdev^2)
cum_var <- cumsum(var_explained)
cum_var_df <- data.frame(PC = 1:length(cum_var), CumulativeVariance = cum_var)
cum_plot <- ggplot(cum_var_df, aes(x = PC, y = CumulativeVariance)) +
  geom_line() +
  geom_point() +
  geom_hline(yintercept = 0.8, linetype = "dashed", color = "red") +
  labs(title = "Cumulative␣explained␣variance␣per␣number␣of␣components",
       x = "Number␣of␣components", y = "Cumulative␣explained␣variance") +
  theme_minimal()
ggsave("pca_plots/cumulative_variance.png", plot = cum_plot, width = 8, height = 6)

# Archetype highlight plots
for (name in names(arch_paths)) {
  pca_scores$highlight <- ifelse(df[[name]] == 1, name, "overig")
  p <- ggplot(pca_scores, aes(x = PC1, y = PC2)) +
    geom_point(data = subset(pca_scores, highlight == "overig"),
               aes(x = PC1, y = PC2), color = "blue", alpha = 0.5, size = 1.5) +
    geom_point(data = subset(pca_scores, highlight == name),
               aes(x = PC1, y = PC2), color = "red", alpha = 0.8, size = 1.8) +
    theme_minimal() +
    labs(title = paste("PCA:", name, "(red␣=␣archetype)")) +
    theme(legend.position = "none")
  ggsave(paste0("pca_plots/pca_plots_archetypes/PCA_", name, "_highlight.png"), plot
}

# Top 10% risk score highlight
pca_scores$top10 <- ifelse(merged$id %in% top10_ids, "Top␣10%", "Overig")
p <- ggplot(pca_scores, aes(x = PC1, y = PC2)) +
  geom_point(data = subset(pca_scores, top10 == "Overig"), color = "blue", alpha = 0.
  geom_point(data = subset(pca_scores, top10 == "Top␣10%"), color = "red", size = 2)
  theme_minimal() +
  labs(title = "PCA:␣highlight␣top␣10%␣risk␣score")
ggsave("pca_plots/PCA_top10_risk_highlight.png", plot = p, width = 8, height = 6)

# Select important features for boxplot
loadings_6pc <- abs(res.pca$rotation[, 1:6])
contrib_6pc <- rowSums(loadings_6pc)
top_features_by_pca <- names(sort(contrib_6pc, decreasing = TRUE)[1:10])
top_features <- top_features_by_pca

# Prepare data for boxplot
longdat <- merged %>%
  select(all_of(top_features), top10) %>%
  pivot_longer(-top10, names_to = "feature", values_to = "value")

# Boxplot of top features
boxplot_p <- ggplot(longdat, aes(x = top10, y = as.numeric(value), fill = top10)) +
```

```
  geom_boxplot(outlier.size = 0.5) +
  facet_wrap(~ feature, scales = "free", ncol = 2) +
  theme_minimal() +
  labs(title = "Distribution of top features: Top 10% vs. Other",
       subtitle = "Features selected based on PCA loadings (first 6 components)",
       x = "", y = "Feature value") +
  scale_fill_manual(values = c("Top 10%" = "red", "Overig" = "blue"))

ggsave("pca_plots/boxplot_top_features.png", plot = boxplot_p, width = 12, height = 8

# Get eigenvalues and detect elbow point
eigenvalues <- res.pca$sdev^2
elbows <- findiplist(1:length(eigenvalues), eigenvalues, 1)
elbow_point <- elbows[1]

loadings <- abs(res.pca$rotation[, 1:elbow_point])
contrib <- rowSums(loadings)
top_vars <- names(sort(contrib, decreasing = TRUE)[1:elbow_point])

write.csv(data.frame(variable = top_vars), "data/02_intermediate/pca_46_variables.csv
cat("Saved", elbow_point, "PCA elbow-selected variables\n")

# Highlight variables plots
dir.create("pca_plots/pca_plots_highlight_var", showWarnings = FALSE)
highlight_vars <- list(
  vrouw = list(var = "persoon_geslacht_vrouw", logic = function(x) x == 1),
  delfshaven = list(var = "adres_recentste_wijk_delfshaven", logic = function(x) x ==
  heeft_kinderen = list(var = "relatie_kind_heeft_kinderen", logic = function(x) x ==
  taaleis = list(var = "persoonlijke_eigenschappen_taaleis_voldaan", logic = function
  historie_andere_inwonende = list(var = "relatie_overig_historie_vorm__andere_inwone
  rec_plaats_rotterdam = list(var = "adres_recentste_plaats_rotterdam", logic = funct
  plannen_organiseren = list(var = "competentie_plannen_en_organiseren", logic = func
  eigenschap_opstelling = list(var = "persoonlijke_eigenschappen_opstelling", logic =
)

pc_combos <- list(
  c(1,2), c(1,3), c(1,4), c(1,5), c(1,6),
  c(2,3), c(2,4), c(2,5), c(2,6),
  c(3,4), c(3,5), c(3,6),
  c(4,5), c(4,6),
  c(5,6)
)

# Function to plot PCA with highlighted variable
plot_pca_highlight <- function(pca_scores, df, pc_x, pc_y, highlight_var, highlight_l
  highlight_label <- ifelse(highlight_logic(df[[highlight_var]]), "highlight", "overi
  plot_df <- pca_scores
  plot_df$highlight <- highlight_label
  p <- ggplot(plot_df, aes_string(x = paste0("PC", pc_x), y = paste0("PC", pc_y))) +
    geom_point(data = subset(plot_df, highlight == "overig"), color = "blue", alpha =
    geom_point(data = subset(plot_df, highlight == "highlight"), color = "red", alpha
    theme_minimal() +
    labs(title = paste0("PCA: PC", pc_x, " vs PC", pc_y, " - highlight: ", highlight_
         x = paste0("PC", pc_x), y = paste0("PC", pc_y)) +
    theme(legend.position = "none")
  ggsave(file.path(outdir, paste0("PCA_", highlight_var, "_PC", pc_x, "_PC", pc_y, "
```

```
}

# Loop over all highlight variables and PC combinations
for (highlight_name in names(highlight_vars)) {
  var_info <- highlight_vars[[highlight_name]]
  for (combo in pc_combos) {
    plot_pca_highlight(pca_scores, df, combo[1], combo[2], var_info$var, var_info$log
  }
}

# Risk score highlight plots for all PC combinations
dir.create("pca_plots/pca_plots_risc", showWarnings = FALSE)
for (combo in pc_combos) {
  pcx <- combo[1]
  pcy <- combo[2]
  p <- ggplot(pca_scores, aes_string(x = paste0("PC", pcx), y = paste0("PC", pcy))) +
    geom_point(data = subset(pca_scores, top10 == "Overig"), color = "blue", alpha =
    geom_point(data = subset(pca_scores, top10 == "Top␣10%"), color = "red", size = 2
    theme_minimal() +
    labs(title = paste0("PCA:␣highlight␣top␣10%␣risk␣score␣(PC", pcx, "␣vs␣PC", pcy,
  ggsave(paste0("pca_plots/pca_plots_risc/PCA_top10_risk_highlight_PC", pcx, "_PC", p
}

# Low contribution variables
loadings <- res.pca$rotation[, 1:2]
arrow_lengths <- sqrt(rowSums(loadings^2))
low_contrib_vars <- names(which(arrow_lengths < 0.1))
writeLines(low_contrib_vars, "pca_plots/low_contrib_vars.txt")

# Most important features
loadings <- abs(res.pca$rotation[, 1:6])
contrib <- rowSums(loadings)
important_vars <- names(sort(contrib, decreasing = TRUE)[1:20])
top_features_df <- data.frame(feature = important_vars, contribution = contrib[import
write.csv(top_features_df, "pca_plots/top20_features_pca.csv", row.names = FALSE)

# Feature grouping and group-level PCA
feature_groups <- c(
  "adres", "contacten", "relatie", "afspraak", "instrument",
  "pla", "belemmering", "persoon", "persoonlijke_eigenschappen",
  "deelname", "ontheffing", "typering"
)

add_group_summary <- function(data, pattern, scale_data = TRUE) {
  group_cols <- grep(pattern, names(data), value = TRUE)
  if (length(group_cols) == 0) return(data)
  group_data <- data[, group_cols, drop = FALSE]
  group_data <- group_data[, sapply(group_data, function(x) length(unique(x[!is.na(x)
  if (ncol(group_data) == 0) return(data)
  data[[paste0(pattern, "_total")]] <- rowSums(group_data, na.rm = TRUE)
  if (scale_data) group_data <- scale(group_data)
  pca <- prcomp(group_data)
  data[[paste0(pattern, "_PC1")]] <- pca$x[, 1]
  return(data)
}
```

```
original_data <- X_scaled
for (group in feature_groups) {
  original_data <- add_group_summary(original_data, group)
}

group_pc1_data <- original_data %>% dplyr::select(ends_with("_PC1"))
group_pc1_data <- scale(group_pc1_data)

# Elbow method for group PC1's
set.seed(123)
elbow_plot_group <- fviz_nbclust(group_pc1_data, kmeans, method = "wss", k.max = 12)
  labs(title = "Elbow method for kmeans clustering (group PC1's)")
ggsave("pca_plots/PCA_kmeans_elbow_plot_groupPC1.png", plot = elbow_plot_group, width

# K-means clustering on group PC1's
set.seed(123)
clust_res_group <- kmeans(group_pc1_data, centers = 4)
original_data$group_cluster <- as.factor(clust_res_group$cluster)

# Visualize clusters in first two group PC1's
p_group_cluster <- ggplot(original_data, aes(x = afspraak_PC1, y = contacten_PC1, col
  geom_point(alpha = 0.7) +
  theme_minimal() +
  labs(title = "Clusters based on group summaries (PC1's)")
ggsave("pca_plots/PCA_group_clusters.png", plot = p_group_cluster, width = 8, height

# Create multiple plots for different group PC1 combinations
group_pc1_cols <- names(original_data)[grepl("_PC1$", names(original_data))]
group_pc1_cols <- group_pc1_cols[group_pc1_cols != "group_cluster"]
dir.create("pca_plots/group_pc1_plots", showWarnings = FALSE)

top_group_pc1s <- head(group_pc1_cols, 6)
for (i in 1:(length(top_group_pc1s)-1)) {
  for (j in (i+1):length(top_group_pc1s)) {
    x_col <- top_group_pc1s[i]
    y_col <- top_group_pc1s[j]

    p <- ggplot(original_data, aes_string(x = x_col, y = y_col, color = "group_cluste
      geom_point(alpha = 0.7) +
      theme_minimal() +
      labs(title = paste("Group PC1 Clusters:", x_col, "vs", y_col),
           x = x_col, y = y_col)

    filename <- paste0("pca_plots/group_pc1_plots/group_clusters_",
                       gsub("_PC1", "", x_col), "_vs_",
                       gsub("_PC1", "", y_col), ".png")
    ggsave(filename, plot = p, width = 8, height = 6)
  }
}

# Correlation heatmap of group PC1's
library(corrplot)
group_pc1_cor <- cor(original_data[, group_pc1_cols], use = "complete.obs")
png("pca_plots/group_pc1_correlation_heatmap.png", width = 10, height = 8)
corrplot(group_pc1_cor, method = "color", type = "upper",
         tl.col = "black", tl.srt = 45,
```

```
         title = "Correlation␣between␣Group␣PC1's")
dev.off()

# Clustering on main PCA components
pca_data_main <- pca_scores[, c("PC1", "PC2", "PC3", "PC4", "PC5", "PC6")]

# Elbow method for main PCA
set.seed(123)
elbow_plot_main <- fviz_nbclust(pca_data_main, kmeans, method = "wss", k.max = 12) +
  labs(title = "Elbow␣method␣for␣kmeans␣clustering␣(PC1-PC6)")
ggsave("pca_plots/PCA_kmeans_elbow_plot_main.png", plot = elbow_plot_main, width = 8,

# Load 46 and 58 feature sets if available
pca_46_vars <- read.csv("data/02_intermediate/pca_46_variables.csv", stringsAsFactors
selected_features_46 <- pca_46_vars$variable
features_46 <- features[, selected_features_46, drop = FALSE]
res.pca_46 <- prcomp(features_46, scale. = FALSE)
pca_scores_46 <- as.data.frame(res.pca_46$x)
pca_data_main_46 <- pca_scores_46[, c("PC1", "PC2", "PC3", "PC4", "PC5", "PC6")]

if (file.exists("results/pca_debiased_features.csv")) {
  pca_debiased_vars <- read.csv("results/pca_debiased_features.csv", stringsAsFactors
  selected_features_58 <- pca_debiased_vars$feature
  features_58 <- features[, selected_features_58, drop = FALSE]
  res.pca_58 <- prcomp(features_58, scale. = FALSE)
  pca_scores_58 <- as.data.frame(res.pca_58$x)
  pca_data_main_58 <- pca_scores_58[, c("PC1", "PC2", "PC3", "PC4", "PC5", "PC6")]
} else {
  pca_data_main_58 <- NULL
}

# Perform clustering on all available datasets
set.seed(123)
k_kmeans <- 4

# K-means on 156 features
kmeans_result_156 <- kmeans(pca_data_main, centers = k_kmeans, nstart = 25)
kmeans_clusters_156 <- as.factor(kmeans_result_156$cluster)

# K-means on 46 features
kmeans_result_46 <- kmeans(pca_data_main_46, centers = k_kmeans, nstart = 25)
kmeans_clusters_46 <- as.factor(kmeans_result_46$cluster)

# K-means on 58 features (if available)
if (!is.null(pca_data_main_58)) {
  kmeans_result_58 <- kmeans(pca_data_main_58, centers = k_kmeans, nstart = 25)
  kmeans_clusters_58 <- as.factor(kmeans_result_58$cluster)
  pca_scores$kmeans_cluster_58 <- kmeans_clusters_58
}

# Add clusters to pca_scores
pca_scores$kmeans_cluster_156 <- kmeans_clusters_156
pca_scores$kmeans_cluster_46 <- kmeans_clusters_46
pca_scores_46$kmeans_cluster_46 <- kmeans_clusters_46

# Plot K-means clustering
```

```
p_kmeans_156 <- ggplot(pca_scores, aes(x = PC1, y = PC2, color = kmeans_cluster_156))
  geom_point(alpha = 0.6, size = 2) +
  theme_minimal() +
  labs(title = "K-means clustering on all 156 features (PCA-reduced)",
       subtitle = paste("k =", k_kmeans, "clusters"), color = "Cluster")
ggsave("pca_plots/PCA_kmeans_clusters_156_features_PC1_PC2.png", plot = p_kmeans_156,

p_kmeans_46 <- ggplot(pca_scores_46, aes(x = PC1, y = PC2, color = kmeans_cluster_46))
  geom_point(alpha = 0.6, size = 2) +
  theme_minimal() +
  labs(title = "K-means clustering on 46 selected features (PCA-reduced)",
       subtitle = paste("k =", k_kmeans, "clusters"), color = "Cluster")
ggsave("pca_plots/PCA_kmeans_clusters_46_features_PC1_PC2.png", plot = p_kmeans_46, w

if (!is.null(pca_data_main_58)) {
  p_kmeans_58 <- ggplot(pca_scores_58, aes(x = PC1, y = PC2, color = kmeans_clusters_
    geom_point(alpha = 0.6, size = 2) +
    theme_minimal() +
    labs(title = "K-means clustering on 58 debiased features (PCA-reduced)",
         subtitle = paste("k =", k_kmeans, "clusters"), color = "Cluster")
  ggsave("pca_plots/PCA_kmeans_clusters_58_features_PC1_PC2.png", plot = p_kmeans_58,
}

# Risk score per cluster
merged$kmeans_cluster_156 <- pca_scores$kmeans_cluster_156
cluster_risk_summary_156 <- aggregate(Ja_pseudo ~ kmeans_cluster_156, data = merged,

merged$kmeans_cluster_46 <- pca_scores$kmeans_cluster_46
cluster_risk_summary_46 <- aggregate(Ja_pseudo ~ kmeans_cluster_46, data = merged, FU

# Hierarchical clustering
dist_matrix_156 <- dist(pca_data_main)
hc_156 <- hclust(dist_matrix_156, method = "ward.D2")

dist_matrix_46 <- dist(pca_data_main_46)
hc_46 <- hclust(dist_matrix_46, method = "ward.D2")

if (!is.null(pca_data_main_58)) {
  dist_matrix_58 <- dist(pca_data_main_58)
  hc_58 <- hclust(dist_matrix_58, method = "ward.D2")
}

# Plot dendrograms
tryCatch({
  png("pca_plots/PCA_hierarchical_dendrogram_156_features.png", width = 20, height =
  par(mar = c(1, 1, 1, 1))
  plot(hc_156, labels = FALSE, hang = -1, main = "Hierarchical Clustering Dendrogram
  dev.off()
}, error = function(e) {
  pdf("pca_plots/PCA_hierarchical_dendrogram_156_features.pdf", width = 12, height =
  par(mar = c(4, 4, 4, 2))
  plot(hc_156, labels = FALSE, hang = -1, main = "Hierarchical Clustering Dendrogram
  dev.off()
})

tryCatch({
```

```
  png("pca_plots/PCA_hierarchical_dendrogram_46_features.png", width = 20, height = 1
  par(mar = c(1, 1, 1, 1))
  plot(hc_46, labels = FALSE, hang = -1, main = "Hierarchical␣Clustering␣Dendrogram␣(
  dev.off()
}, error = function(e) {
  pdf("pca_plots/PCA_hierarchical_dendrogram_46_features.pdf", width = 12, height = 8
  par(mar = c(4, 4, 4, 2))
  plot(hc_46, labels = FALSE, hang = -1, main = "Hierarchical␣Clustering␣Dendrogram␣(
  dev.off()
})

if (!is.null(pca_data_main_58)) {
  tryCatch({
    png("pca_plots/PCA_hierarchical_dendrogram_58_features.png", width = 20, height =
    par(mar = c(1, 1, 1, 1))
    plot(hc_58, labels = FALSE, hang = -1, main = "Hierarchical␣Clustering␣Dendrogram
    dev.off()
  }, error = function(e) {
    pdf("pca_plots/PCA_hierarchical_dendrogram_58_features.pdf", width = 12, height =
    par(mar = c(4, 4, 4, 2))
    plot(hc_58, labels = FALSE, hang = -1, main = "Hierarchical␣Clustering␣Dendrogram
    dev.off()
  })
}

# Cut into clusters
k_hc <- 4
hc_clusters_156 <- cutree(hc_156, k = k_hc)
hc_clusters_156 <- as.factor(hc_clusters_156)

hc_clusters_46 <- cutree(hc_46, k = k_hc)
hc_clusters_46 <- as.factor(hc_clusters_46)

if (!is.null(pca_data_main_58)) {
  hc_clusters_58 <- cutree(hc_58, k = k_hc)
  hc_clusters_58 <- as.factor(hc_clusters_58)
  pca_scores$hierarchical_cluster_58 <- hc_clusters_58
}

# Add hierarchical clusters to pca_scores
pca_scores$hierarchical_cluster_156 <- hc_clusters_156
pca_scores$hierarchical_cluster_46 <- hc_clusters_46
pca_scores_46$hierarchical_cluster_46 <- hc_clusters_46

# Plot hierarchical clustering
p_hierarchical_156 <- ggplot(pca_scores, aes(x = PC1, y = PC2, color = hierarchical_c
  geom_point(alpha = 0.6, size = 2) +
  theme_minimal() +
  labs(title = "Hierarchical␣Clustering␣(Ward)␣on␣all␣156␣features␣(PCA-reduced)",
       subtitle = paste("k␣=", k_hc, "clusters"), color = "Cluster")
ggsave("pca_plots/PCA_hierarchical_clusters_156_features_PC1_PC2.png", plot = p_hiera

p_hierarchical_46 <- ggplot(pca_scores_46, aes(x = PC1, y = PC2, color = hierarchical
  geom_point(alpha = 0.6, size = 2) +
  theme_minimal() +
  labs(title = "Hierarchical␣Clustering␣(Ward)␣on␣46␣selected␣features␣(PCA-reduced)"
```

```
       subtitle = paste("k␣=", k_hc, "clusters"), color = "Cluster")
ggsave("pca_plots/PCA_hierarchical_clusters_46_features_PC1_PC2.png", plot = p_hierar

if (!is.null(pca_data_main_58)) {
  p_hierarchical_58 <- ggplot(pca_scores_58, aes(x = PC1, y = PC2, color = hc_cluster
    geom_point(alpha = 0.6, size = 2) +
    theme_minimal() +
    labs(title = "Hierarchical␣Clustering␣(Ward)␣on␣58␣debiased␣features␣(PCA-reduced
         subtitle = paste("k␣=", k_hc, "clusters"), color = "Cluster")
  ggsave("pca_plots/PCA_hierarchical_clusters_58_features_PC1_PC2.png", plot = p_hier
}

# Risk score per hierarchical cluster
merged$hierarchical_cluster_156 <- pca_scores$hierarchical_cluster_156
cluster_risk_summary_hc_156 <- aggregate(Ja_pseudo ~ hierarchical_cluster_156, data =

merged$hierarchical_cluster_46 <- pca_scores$hierarchical_cluster_46
cluster_risk_summary_hc_46 <- aggregate(Ja_pseudo ~ hierarchical_cluster_46, data = m

# Comparison plots
pca_scores$cluster_comparison_156 <- paste("K", pca_scores$kmeans_cluster_156, "-␣H",
p_comparison_156 <- ggplot(pca_scores, aes(x = PC1, y = PC2, color = cluster_comparis
  geom_point(alpha = 0.6, size = 1.5) +
  theme_minimal() +
  labs(title = "Comparison:␣K-means␣vs␣Hierarchical␣Clustering␣(156␣features)",
       subtitle = "K␣=␣K-means␣cluster,␣H␣=␣Hierarchical␣cluster", color = "Cluster␣A
  theme(legend.position = "bottom")
ggsave("pca_plots/PCA_clustering_comparison_156_features.png", plot = p_comparison_15

pca_scores_46$cluster_comparison_46 <- paste("K", pca_scores_46$kmeans_cluster_46, "-
p_comparison_46 <- ggplot(pca_scores_46, aes(x = PC1, y = PC2, color = cluster_compar
  geom_point(alpha = 0.6, size = 1.5) +
  theme_minimal() +
  labs(title = "Comparison:␣K-means␣vs␣Hierarchical␣Clustering␣(46␣features)",
       subtitle = "K␣=␣K-means␣cluster,␣H␣=␣Hierarchical␣cluster", color = "Cluster␣A
  theme(legend.position = "bottom")
ggsave("pca_plots/PCA_clustering_comparison_46_features.png", plot = p_comparison_46,

if (!is.null(pca_data_main_58)) {
  pca_scores_58$cluster_comparison_58 <- paste("K", kmeans_clusters_58, "-␣H", hc_clu
  p_comparison_58 <- ggplot(pca_scores_58, aes(x = PC1, y = PC2, color = cluster_comp
    geom_point(alpha = 0.6, size = 1.5) +
    theme_minimal() +
    labs(title = "Comparison:␣K-means␣vs␣Hierarchical␣Clustering␣(58␣debiased␣feature
         subtitle = "K␣=␣K-means␣cluster,␣H␣=␣Hierarchical␣cluster", color = "Cluster
    theme(legend.position = "bottom")
  ggsave("pca_plots/PCA_clustering_comparison_58_features.png", plot = p_comparison_5
}

# Save clustering results
clustering_results_156 <- data.frame(
  id = merged$id,
  kmeans_cluster = pca_scores$kmeans_cluster_156,
  hierarchical_cluster = pca_scores$hierarchical_cluster_156,
  risk_score = merged$Ja_pseudo
)
```

```
  write.csv(clustering_results_156, "pca_plots/clustering_results_156_features.csv", ro

  clustering_results_46 <- data.frame(
    id = merged$id,
    kmeans_cluster = pca_scores$kmeans_cluster_46,
    hierarchical_cluster = pca_scores$hierarchical_cluster_46,
    risk_score = merged$Ja_pseudo
  )
  write.csv(clustering_results_46, "pca_plots/clustering_results_46_features.csv", row.

  if (!is.null(pca_data_main_58)) {
    clustering_results_58 <- data.frame(
      id = merged$id,
      kmeans_cluster = kmeans_clusters_58,
      hierarchical_cluster = hc_clusters_58,
      risk_score = merged$Ja_pseudo
    )
    write.csv(clustering_results_58, "pca_plots/clustering_results_58_features.csv", ro
  }

  cat("PCA␣analysis␣and␣clustering␣completed.␣Results␣saved␣to␣pca_plots/\n")
```

C.2 Risk in Clustering

```
  # Create cluster risk summary plots for all models (156, 46, 58 features)

library(ggplot2)
library(dplyr)
library(readr)

setwd("/Users/catootjeversluis/suspicion_machine")



# Create cluster risk summary data
create_cluster_risk_data <- function(clustering_results_file, risk_scores_file, model_nam
  if (file.exists(clustering_results_file)) {
    clustering <- read_csv(clustering_results_file, show_col_types = FALSE)
  } else {
    cat("Warning:␣Clustering␣results␣file␣not␣found:", clustering_results_file, "\n")
    return(NULL)
  }

  if (file.exists(risk_scores_file)) {
    risk_scores <- read_csv(risk_scores_file, show_col_types = FALSE)
  } else {
    cat("Warning:␣Risk␣scores␣file␣not␣found:", risk_scores_file, "\n")
    return(NULL)
  }

  merged_data <- merge(clustering, risk_scores, by = "id", all.x = TRUE)

  risk_col <- switch(model_name,
    "156" = "Ja_pseudo",
    "46" = "Ja_pseudo",
    "58" = "Ja_pca_debiased",
```

```
    "Ja_pseudo"
  )

  if (!risk_col %in% colnames(merged_data)) {
    cat("Warning: Risk score column", risk_col, "not found in data\n")
    return(NULL)
  }

  summaries <- list()

  if ("kmeans_cluster" %in% colnames(merged_data)) {
    kmeans_summary <- merged_data %>%
      group_by(kmeans_cluster) %>%
      summarise(
        n = n(),
        mean_risk = mean(get(risk_col), na.rm = TRUE),
        sd_risk = sd(get(risk_col), na.rm = TRUE),
        .groups = "drop"
      ) %>%
      mutate(
        cluster = as.character(kmeans_cluster),
        top_decile_share = n / sum(n) * 10
      ) %>%
      select(cluster, n, mean_risk, sd_risk, top_decile_share)

    summaries$kmeans <- kmeans_summary
  }

  if ("hierarchical_cluster" %in% colnames(merged_data)) {
    hclust_summary <- merged_data %>%
      group_by(hierarchical_cluster) %>%
      summarise(
        n = n(),
        mean_risk = mean(get(risk_col), na.rm = TRUE),
        sd_risk = sd(get(risk_col), na.rm = TRUE),
        .groups = "drop"
      ) %>%
      mutate(
        cluster = as.character(hierarchical_cluster),
        top_decile_share = n / sum(n) * 10
      ) %>%
      select(cluster, n, mean_risk, sd_risk, top_decile_share)

    summaries$hclust <- hclust_summary
  }

  return(summaries)
}

cat("Creating cluster risk summary plots for all models...\n")

# Create cluster risk summary plot
create_cluster_risk_plot <- function(data, title, filename) {
  # Create color mapping based on risk levels
  sorted_data <- data %>% arrange(desc(mean_risk))
```

```
  # Assign colors based on risk ranking (high risk = red, low risk = green)
  risk_colors <- c("#FF4444", "#FFAA44", "#44AA44", "#4444FF")  # Red, Orange, Green, Blu

  # Create color mapping
  color_mapping <- setNames(risk_colors[1:nrow(sorted_data)], sorted_data$cluster)

  p <- ggplot(data, aes(x = cluster, y = mean_risk, fill = cluster)) +
    geom_bar(stat = "identity", width = 0.7) +
    scale_fill_manual(values = color_mapping, name = "Cluster (Risk Level)") +
    theme_minimal() +
    theme(
      plot.background = element_rect(fill = "black"),
      panel.background = element_rect(fill = "black"),
      panel.grid.major = element_line(color = "gray30"),
      panel.grid.minor = element_blank(),
      axis.text = element_text(color = "white"),
      axis.title = element_text(color = "white"),
      plot.title = element_text(color = "white", hjust = 0.5),
      legend.position = "bottom",
      legend.text = element_text(color = "white"),
      legend.title = element_text(color = "white")
    ) +
    labs(
      title = title,
      x = "Cluster",
      y = "Mean Risk Score",
      caption = "Risk scores range from 0 (low risk) to 1 (high risk)\nColors: Red=High R
    ) +
    scale_y_continuous(limits = c(0, 0.7), breaks = seq(0, 0.7, 0.2))

  ggsave(filename, plot = p, width = 8, height = 6, dpi = 300, bg = "black")
  cat("Created:", filename, "\n")

  return(p)
}

dir.create("pca_plots/overlay_plots", recursive = TRUE, showWarnings = FALSE)

models <- list(
  list(name = "156", clustering_file = "pca_plots/clustering_results_156_features.csv", r
  list(name = "46", clustering_file = "pca_plots/clustering_results_46_features.csv", ris
  list(name = "58", clustering_file = "pca_plots/clustering_results_58_features.csv", ris
)

# Simple cluster risk summary
analyze_cluster_risk <- function(data, model_name) {
  sorted_data <- data %>% arrange(desc(mean_risk))
  max_risk <- max(sorted_data$mean_risk)
  min_risk <- min(sorted_data$mean_risk)
  risk_range <- max_risk - min_risk

  cat(sprintf("Model %s: Risk range %.3f (%.3f to %.3f)\n",
              model_name, risk_range, min_risk, max_risk))
}

# Process each model
```

```
for (model in models) {
  cat(sprintf("Processing model with %s features...\n", model$name))

  summaries <- create_cluster_risk_data(
    model$clustering_file,
    model$risk_file,
    model$name
  )

  if (is.null(summaries)) {
    cat("Skipping model", model$name, "due to missing data\n")
    next
  }

  # K-means plots and analysis
  if (!is.null(summaries$kmeans)) {
    kmeans_filename <- sprintf("pca_plots/overlay_plots/cluster_risk_summary_kmeans4_%s_f
    kmeans_title <- sprintf("Cluster Risk Summary - K-means (4 clusters) - %s features",

    create_cluster_risk_plot(summaries$kmeans, kmeans_title, kmeans_filename)

    csv_filename <- sprintf("pca_plots/overlay_plots/cluster_risk_summary_kmeans4_%s.csv'
    write_csv(summaries$kmeans, csv_filename)
    cat("Saved CSV:", csv_filename, "\n")

    analyze_cluster_risk(summaries$kmeans, paste0(model$name, " (K-means)"))
  }

  # Hierarchical plots and analysis
  if (!is.null(summaries$hclust)) {
    hclust_filename <- sprintf("pca_plots/overlay_plots/cluster_risk_summary_hclust4_%s_f
    hclust_title <- sprintf("Cluster Risk Summary - Hierarchical (4 clusters) - %s featur

    create_cluster_risk_plot(summaries$hclust, hclust_title, hclust_filename)

    csv_filename <- sprintf("pca_plots/overlay_plots/cluster_risk_summary_hclust4_%s.csv'
    write_csv(summaries$hclust, csv_filename)
    cat("Saved CSV:", csv_filename, "\n")

    analyze_cluster_risk(summaries$hclust, paste0(model$name, " (Hierarchical)"))
  }
}
```

C.3 Original, Clean and Weighted model training

```
    # Analyze data with Mahalanobis distance and statistical parity tests

library(stringi)
library(caret)
library(gbm)

if (!require("caret")) install.packages("caret")
if (!require("ggplot2")) install.packages("ggplot2")
if (!require("dplyr")) install.packages("dplyr")
if (!require("glue")) install.packages("glue")
if (!require("ini")) install.packages("ini")  # init files
```

```
if (!requireNamespace("ini", quietly=TRUE)) install.packages("ini")
library(ini)

if (!require("RJSONIO")) install.packages("RJSONIO") # json string to r object #yes
if (!require("OneR")) install.packages("OneR") #bin
if (!require("ggpubr")) install.packages("ggpubr")
if (!require("rstatix")) install.packages("rstatix")
if (!require("stringr")) install.packages("stringr")
if (!require("rstudioapi")) install.packages("rstudioapi")


# Predict risk scores using final model
final_model.predict <- function(final_model, input_data, output_filename, model_type = NU
  # [CHANGED] Remove 'id' and 'data_type' columns if present before prediction
  drop_cols <- intersect(c("id", "data_type","MD"), names(input_data))
  features <- input_data[, setdiff(names(input_data), drop_cols), drop = FALSE]
  # [CHANGED] Use type = 'raw' for regression predictions
  risk_scores <- predict.train(final_model, newdata = features, type = 'raw')
  # [UNCHANGED] Combine input data and risk scores
  risk_scores <- cbind(input_data, Ja = risk_scores, Nee=1-risk_scores)
  # [UNCHANGED] Write to file if filename is provided
  if (output_filename != "") {
    write.csv(risk_scores, output_filename, row.names = FALSE)
    cat("Risk␣scores␣saved␣to:", output_filename, "\n")
  }
  return(risk_scores)
}

# Prepare data for analysis by transforming variables
final_model.prepare_for_analysis <- function(variable_list, df, bins, nbins = 8, model_ty
  if (model_type =="original"|| model_type=="clean"|| model_type=="weighted") {
    # Simpele IV-opbouw
    new_var_list <- list()
    for (i in 1:length(variable_list)) {
      new_var_name <- NA
      nested_list <- variable_list[[i]]
      # One Hot Encoded case
      if (length(nested_list) > 1) {
        var_name <- paste0('var_', toString(i))
        df[var_name] <- NA
        for (j in 1:length(nested_list)) {
          cur <- names(nested_list[[j]])
          if(nrow(df[df[cur] == 1,]) > 0){
            df[df[cur] == 1,][var_name] <- cur
          }
        }
        new_var_name <- var_name
      }
      cur <- names(nested_list[[1]])
      if (cur %in% bins) {
        var_name <- paste0('var_', toString(i))
        df[, var_name] <- bin(df[[cur]], nbins = nbins, method = 'content')
        new_var_name <- var_name
      }
      if(is.na(new_var_name)){
        new_var_name <- cur
```

```
    }
    new_var_list <- append(new_var_list, new_var_name)
  }
  df$IV <- 'IV: \n'
  for (var in new_var_list) {
    if (length(unique(df[[var]])) == 1) next
    df$IV <- paste(df$IV, var, ': ', df[[var]], '\n', sep = '')
  }
  return(df)
} else {
  # Gebruik de robuuste aanpak voor clean/weighted
  new_var_names <- list()
  missing_vars <- c()
  for (i in seq_along(variable_list)) {
    var_name <- NA_character_
    nested_list <- variable_list[[i]]
    if (is.data.frame(nested_list)) {
      base_feature <- names(nested_list)[1]
    } else if (is.list(nested_list) && length(nested_list) > 0 && !is.null(names(nested
      base_feature <- names(nested_list[[1]])[1]
    } else {
      base_feature <- NA_character_
      warning(paste("Kan geen base_feature bepalen voor variable_list op positie", i, '
    }
    if (length(bins) > 0 && base_feature %in% bins) {
      bin_var_name <- paste0('var_', i)
      if (base_feature %in% names(df)) {
        values <- df[[base_feature]]
        if (is.factor(values)) values <- as.numeric(as.character(values))
        df[[bin_var_name]] <- bin(values, nbins = nbins, method = 'content')
      } else {
        warning(paste("Kolom", base_feature, "niet gevonden in df voor binning"))
        missing_vars <- unique(c(missing_vars, base_feature))
      }
      var_name <- bin_var_name
    }
    if (is.na(var_name)) {
      var_name <- base_feature
    }
    if (!is.null(var_name) && !is.na(var_name) && var_name != "" && length(var_name) >
      new_var_names[[length(new_var_names) + 1]] <- var_name
    } else {
      warning(paste("Lege variabelenaam in variable_list op positie", i, ":", variable_
    }
  }
  df$IV <- 'IV:\n'
  for (var in new_var_names) {
    if (is.null(var) || is.na(var) || var == "" || length(var) == 0) {
      warning("Lege variabelenaam in new_var_names, wordt overgeslagen.")
      next
    }
    if (!var %in% names(df)) {
      warning(paste("Kolom", var, "niet gevonden in df bij opbouw IV"))
      missing_vars <- unique(c(missing_vars, var))
      next
    }
```

```
      vals <- df[[var]]
      if (length(unique(vals)) <= 1) next
      df$IV <- paste0(df$IV, var, ': ', vals, '\n')
    }
    if (length(missing_vars) > 0) {
      warning(paste("De volgende kolommen ontbreken in de data en zijn niet meegenomen in
      print(paste("De volgende kolommen ontbreken in de data en zijn niet meegenomen in
    }
    return(df)
  }
}


# Visual Analysis
#
# @param sum_stats_list: list of data frames with summary statistics
# @param decile_count_list: list of data frames containing number of observations per ris
# @param plot_list: list of plots to put more plots into
#
# @return plotlist: return plot_list to which the newly generated plots have been added
final_model.visual_analysis <- function(sum_stats_list, decile_count_list, plot_list) {

  #iterate over all data frames in decile count list
  for(decile_df in decile_count_list){
    cur_dt <- decile_df[1,]$data_type
    #generate line graph
    p1 <- ggplot(decile_df, aes(x=decile, y=perc, color = as.factor(IV), group = IV))+
      geom_point(size = 2)+
      geom_line(linewidth = 1)+
      scale_color_brewer(palette = "Set3")+
      guides(color = guide_legend(title = "Groep (IV)", ncol = 1, override.aes = list(siz
      theme_minimal() +
      theme(legend.position = "right", legend.text = element_text(size = 8))+
      labs(x = 'Risk score decile', y = 'Group percentage in risk score decile', title =
    plot_list <- append(plot_list, list(p1))
  }

  #iterate over sum_stats list
  for(sum_stats in sum_stats_list){
    cur_dt <- sum_stats[1,]$data_type
    #generate over-representation shift plot
    p1 <- ggplot(sum_stats, aes(x=IV, y = perc_misrep_td_threshold, fill = IV))+
      geom_col(show.legend = FALSE, width = 0.5) +
      geom_hline(yintercept = 0)+
      theme(axis.title.x = element_text(margin=margin(t = 3)))+
      coord_flip()+
      theme_bw() +
      theme(axis.text.y = element_text(size = 8))+
      labs(x = 'Groep (IV)', y = 'Excess share in highest risk decile', title = paste0(cu
    plot_list <- append(plot_list, list(p1))
  }

  return (plot_list)
}
```

```
# Save Plots
#
# @param plot_list: list of plots to be saved
# @param config: configuration object used to extract output filename
final_model.save_plots <- function(plot_list, config, model_type) {
  save_dir <- file.path('results', 'statistical_parity', model_type, config$META$name)
  dir.create(save_dir, recursive = TRUE, showWarnings = FALSE)
  output_filename <- file.path(save_dir, glue('plot_{model_type}.pdf'))
  pdf(output_filename, width = 10, height = 7)
  print(plot_list)
  dev.off()
}


# Numeric Analysis
#
# @param df: scored dataframe to be analyzed
# @param dt: string with data type specified
# @param sum_stats_list: empty list of summary stats tables
# @param ttest_list: empty list of ttest tables
#
# @return: return both the sum_stats and the ttest_list in a list of lists, where the ne

final_model.numeric_analysis <- function(df, dt, sum_stats_list, ttest_list, decile_count
  #synthetic data has a higher average risk score than real training data, so I use a se
  td_threshold <- ifelse(dt %in% c('real', 'real_conditional'), td_real_threshold, td_sy

  #caculate share of risk scores above threshold
  df_threshold <- df %>%
    dplyr::mutate(above_td_threshold = ifelse(df$Ja > td_threshold, 1, 0),
                  above_rw_threshold = ifelse(df$Ja > rw_threshold, 1, 0)) %>%
    dplyr::group_by(IV) %>%
    dplyr::summarise(share_above_td_threshold = mean(above_td_threshold),
                     share_above_rw_threshold = mean(above_rw_threshold)) %>%
    dplyr::mutate(perc_misrep_td_threshold = (share_above_td_threshold - 0.1) * 100,
                  perc_misrep_rw_threshold = (share_above_rw_threshold - 0.1) * 100)

  #get mean and sd for each level of IV
  sum_stats <- df %>%
    dplyr::group_by(IV) %>%
    get_summary_stats(Ja, type = 'mean_sd')

  #merge over-representation with other summary stats
  sum_stats <- dplyr::left_join(sum_stats, df_threshold, by = 'IV')

  sum_stats$data_type <- dt
  sum_stats_list <- append(sum_stats_list, list(sum_stats))


  #run pairwise ttests between each level of IV; tells us if there is a significant diff
  if(length(unique(df$IV)) > 1){
    pwt <- df %>%
      pairwise_t_test(Ja~IV)
    pwt$data_type <- dt
    ttest_list <- append(ttest_list, list(pwt))
  }
```

```
  #calculate the risk score deciles
  df$decile <- ntile(df$Ja, 10)

  #count number of obersvations per decile
  df_decile_counts <- df %>%
    dplyr::group_by(decile, IV) %>%
    dplyr::count() %>%
    dplyr::group_by(IV) %>%
    dplyr::mutate(perc = (100 * n/sum(n))) %>%
    dplyr::ungroup() %>%
    dplyr::mutate(data_type = dt)

  #add decile count df to decile count list
  decile_count_list <- append(decile_count_list, list(df_decile_counts))

  return(list(sum_stats_list, ttest_list, decile_count_list))

}


# Save Tables
#
# @param sum_stats_list: list of summary stats tables
# @param ttest_list: list of ttest tables
# @param config: configuration object used to extract output filename
final_model.save_tables <- function(sum_stats_list, ttest_list, decile_count_list, config
  save_dir <- file.path('results', 'statistical_parity', model_type, config$META$name)
  output_sum_stats <- file.path(save_dir, glue("sum_stats_{model_type}.csv") )#summary st
  output_ttest <- file.path(save_dir, glue("t_test_{model_type}.csv")) #ttest file path
  output_decile <- file.path(save_dir, glue("decile_{model_type}.csv")) #Decile file path

  all_sum_stats <- as.data.frame(dplyr::bind_rows(sum_stats_list)) #combine all summary s
  all_ttest <- as.data.frame(dplyr::bind_rows(ttest_list))  #combine all ttest tables int
  all_decile <- as.data.frame(dplyr::bind_rows(decile_count_list))  #combine all decile t

  write.csv(all_sum_stats, output_sum_stats) #save summary stats
  write.csv(all_ttest, output_ttest) #save ttest tables
  write.csv(all_decile, output_decile) #save decile tables
  print('All␣tables␣have␣been␣saved.')
}


# Analyze
#
# @param ini: filepath to an ini config file
# @param final_model: gbm model taken from Rotterdam's risk prediction system
#
# Purpose: wrapper function to call all the load, predict, pre-processing, analysis, and
final_model.analyze <- function(ini, final_model, model_type){
  # read ini file
  config_filename <- file.path('conf/archetypes', ini) #CHECK: specify filepath
  config <- read.ini(config_filename, encoding = 'utf8')
  input_filename <- file.path(
    'data','03_experiment_input',
    paste0(
```

```
    paste(config$META$date, config$META$user, config$META$name, sep = '_'),
    '.csv'
  )
)
output_filename <- file.path('data', '04_experiment_output', model_type,
  paste0(
    paste(config$META$date, config$META$user, config$META$name, sep = '_'),
    '.csv'
  )
)
dir.create(file.path('data', '04_experiment_output', model_type), recursive = TRUE, sho
var_list <- config$VARIABLES$variable_list
var_list <- str_replace_all(var_list, '   ', '\"')
var_list <- str_replace_all(var_list, '   ', '\"')
variable_list <- fromJSON(var_list)
print(variable_list)
bin_list <- fromJSON(config$VARIABLES$bin)

# Read data
df <- read.csv(input_filename)
print(glue('Shape␣of␣dataframe␣after␣loading␣{dim(df)}'))

# Predict
risk_scores <- final_model.predict(final_model, df, output_filename, model_type = model
if ("Ja" %in% names(risk_scores)) {
  if (is.factor(risk_scores$Ja)) risk_scores$Ja <- as.numeric(as.character(risk_scores$
  if (is.character(risk_scores$Ja)) risk_scores$Ja <- as.numeric(risk_scores$Ja)
}
print(glue('Shape␣of␣dataframe␣after␣prediction␣{dim(risk_scores)}'))

# Bin and revert One Hot Encoding
risk_scores <- final_model.prepare_for_analysis(variable_list, risk_scores, bin_list, n

#conduct seperate analysis on each data type. This allows us to differentiate in our an
all_data_types <- unique(risk_scores$data_type)
print(glue("All␣data␣types␣{all_data_types}"))

#initialize analysis results lists
plot_list <- list()
sum_stats_list <- list()
ttest_list <- list()
decile_count_list <- list()

#itereate over data types and conduct visual and numeric analysis
for (dt in all_data_types){
  table_lists <- final_model.numeric_analysis(risk_scores[risk_scores$data_type == dt,]
  sum_stats_list <- table_lists[[1]]
  ttest_list <- table_lists[[2]]
  decile_count_list <- table_lists[[3]]
}

#take the visual analysis out of the loop and perform the visual analysis on the tables
plot_list <- final_model.visual_analysis(sum_stats_list, decile_count_list, plot_list)

#save numeric and visual analysis
final_model.save_plots(plot_list, config, model_type)
```

```
  final_model.save_tables(sum_stats_list, ttest_list, decile_count_list, config, model_ty
}

# analyze_clean.R
# ------------------------------
# Train een GBM op Mahalanobis-gefilterde data met gewogen pseudo-labels

library(caret)
library(dplyr)
library(glue)

print(getwd())

# 1. Laad het originele model (voor pseudo-labels)
orig_list <- readRDS('data/01_raw/20220929_finale_model.rds')
model_orig <- orig_list$model[[1]]  # caret::train object
synth_full <- read.csv("data/02_intermediate/synth_data_with_MD.csv", stringsAsFactors =

#synth_inliers and generate pseudo yes with original model
synth_inliers <- read.csv("data/02_intermediate/synth_data_MD_inliers.csv", stringsAsFact
synth_full$Ja_pseudo <- predict(model_orig, synth_full, type = "prob")$Ja

#new trained model Md outliers deleted
elbow_MD <- quantile(synth_full$MD, probs = 0.90)
synth_inliers <- synth_full %>% filter(MD <= elbow_MD)

ts_clean <- synth_inliers %>% select(-id, -MD)
nzv <- nearZeroVar(ts_clean)
if (length(nzv) > 0) ts_clean <- ts_clean[, -nzv]
ts_clean <- ts_clean[, sapply(ts_clean, is.numeric)]
ts_clean$Ja_pseudo <- synth_inliers$Ja_pseudo

set.seed(2025)
if (file.exists("model_clean.RData")) {
  cat("model_clean.RData␣found␣  ␣loading␣instead␣of␣retraining\n")
  load("model_clean.RData")
} else {
  cat("model_clean.RData␣not␣found␣  ␣training␣now\n")
  model_clean <- train(
    Ja_pseudo ~ .,
    data = ts_clean,
    method = 'gbm',
    trControl = trainControl(method = 'cv', number = 5),
    verbose = FALSE,
    distribution = 'gaussian'
  )
  save(model_clean, file = "model_clean.RData")
}

#new trained model reweighted on MD values
# Safer MD normalization with edge case handling
max_md <- max(synth_full$MD, na.rm = TRUE)

if (max_md <= 0) {
  cat("Warning:␣max(MD)␣<=␣0,␣using␣uniform␣weights\n")
  weights <- rep(1, nrow(synth_full))
```

```
} else if (max_md < 1e-10) {
  cat("Warning: max(MD) very small, using uniform weights\n")
  weights <- rep(1, nrow(synth_full))
} else {
  # Normal MD normalization
  weights <- 1 - synth_full$MD / max_md

  # Additional safety check for negative weights
  if (any(weights < 0)) {
    cat("Warning: Some weights are negative, clipping to 0\n")
    weights <- pmax(weights, 0)
  }

  # Check for all weights being 0
  if (all(weights == 0)) {
    cat("Warning: All weights are 0, using uniform weights\n")
    weights <- rep(1, nrow(synth_full))
  }
}

cat("MD normalization summary:\n")
cat("Max MD:", max_md, "\n")
cat("Weight range:", range(weights), "\n")
cat("Mean weight:", mean(weights), "\n")

ts_weighted <- synth_full %>% select(-id, -MD)
nzv <- nearZeroVar(ts_weighted)
if (length(nzv) > 0) ts_weighted <- ts_weighted[, -nzv]

ts_weighted <- ts_weighted[, sapply(ts_weighted, is.numeric)]
target <- ts_weighted$Ja_pseudo
ts_weighted$Ja_pseudo <- NULL
ts_weighted$Ja_pseudo <- target

set.seed(2025)
if (file.exists("model_weighted.RData")) {
  cat("model_weighted.RData found   loading instead of retraining\n")
  load("model_weighted.RData")
} else {
  cat("model_weighted.RData not found   training now\n")
  model_weighted <- train(
    Ja_pseudo ~ .,
    data = ts_weighted,
    method = 'gbm',
    trControl = trainControl(method = 'cv', number = 5),
    weights=weights,
    verbose = FALSE,
    distribution = 'gaussian'
  )
  save(model_weighted, file = "model_weighted.RData")
}



#score inliers and save them
synth_scoredata <- read.csv("data/01_raw/synth_data_with_id.csv", stringsAsFactors = FALS
```

```
features <- synth_scoredata %>% select(-id)

synth_scoredata$Ja_clean    <- predict(model_clean, features, type = "raw")
synth_scoredata$Ja_weighted <- predict(model_weighted, features, type = "raw")
synth_scoredata$Ja_orig     <- predict(model_orig, synth_scoredata, type = "prob")$Ja

# 11. Bepaal nieuwe thresholds op basis van modeloutput
# Thresholds blijven hetzelfde (zoals gedefinieerd door Rotterdam)
td_real_threshold <- 0.6970136
rw_threshold      <- qnorm(0.975, mean = 0.50, sd = 0.10)

# Nieuw: synth_thresholds uit beide modellen
synth_full$Ja_clean    <- predict(model_clean, synth_full %>% select(-id, -MD), type = "r
synth_full$Ja_weighted <- predict(model_weighted, synth_full %>% select(-id, -MD), type =

td_synth_threshold_clean    <- quantile(synth_full$Ja_clean, probs = 0.90)
td_synth_threshold_weighted <- quantile(synth_full$Ja_weighted, probs = 0.90)


final_model <- readRDS(file.path( 'data', '01_raw', '20220929_finale_model.rds'))$model[

td_real_threshold <- 0.6970136 #@Htet threshold derived from training data risk score dis
td_synth_threshold <- 0.7125668 #@Htet threshold derived from synthetic training data ris
rw_threshold <- 0.5912492 #@Htet threshold derived from normal approximation of real worl

for (ini in list.files(path = 'conf/archetypes', pattern='*.ini')) { # CHECK: specify fil
  print(glue('Analyzing␣{ini}'))
  final_model.analyze(ini, final_model, "original")
  gc() #clean memory
}

# Eerst clean model
td_synth_threshold <- td_synth_threshold_clean
for (ini in list.files(path = 'conf/archetypes', pattern = '*.ini')) {
  print(glue("Analyzing␣{ini}␣with␣CLEAN␣model"))
  final_model.analyze(ini, model_clean, model_type = "clean")
  gc()
}

# Dan weighted model
# Set the threshold for the weighted model
# (already set above)
td_synth_threshold<-td_synth_threshold_weighted
for (ini in list.files(path = 'conf/archetypes', pattern = 'ini', ignore.case = TRUE)) {
  print(glue("Analyzing␣{ini}␣with␣WEIGHTED␣model"))
  final_model.analyze(ini, model_weighted, model_type = "weighted")
  gc()
}

# Opslaan beide modellen
save(model_clean, file = "model_clean.RData")
save(model_weighted, file = "model_weighted.RData")

synth_full$Ja_clean <- predict(model_clean, synth_full %>% select(-id, -MD), type = "raw'
synth_full$Ja_weighted <- predict(model_weighted, synth_full %>% select(-id, -MD), type =
```

```
# Save all 3 scores: original, clean, and weighted
write.csv(
  synth_full[, c("id", "Ja_pseudo", "Ja_clean", "Ja_weighted")],
  "results/risk_scores_comparison.csv",
  row.names = FALSE
)

cat("Risk score comparison saved to resuts/risk_scores_comparison.csv\n")
```

C.4 58 Features Decision and Model

```
# Debiased PCA model training

library(dplyr)
library(ggplot2)
library(tidyr)
library(caret)
library(corrplot)
library(glue)

setwd("/Users/catootjeversluis/suspicion_machine")

cat("Calculating archetype overrepresentation...\n")

if (file.exists("results/risk_scores_comparison.csv")) {
  risk_scores <- read.csv("results/risk_scores_comparison.csv", stringsAsFactors = FALSE)
  cat("Loaded risk scores\n")
} else {
  cat("No risk_scores_comparison.csv found\n")
  stop("Missing risk scores file")
}

if (file.exists("archetype_id_mapping.csv")) {
  archetype_mapping <- read.csv("archetype_id_mapping.csv", stringsAsFactors = FALSE)
  cat("Loaded archetype mapping\n")
} else {
  cat("Creating archetype mapping...\n")

  main_data <- read.csv("data/01_raw/synth_data_with_id.csv", stringsAsFactors = FALSE)
  cat("Main data loaded\n")

  flexible_archetypes <- list(
    arch_combined_max = list(
      criteria = list(
        "persoon_geslacht_vrouw" = function(x) x == 1,
        "relatie_partner_totaal_dagen_partner" = function(x) x >= 365,
        "relatie_kind_huidige_aantal" = function(x) x >= 1,
        "relatie_kind_heeft_kinderen" = function(x) x == 1,
        "persoonlijke_eigenschappen_taaleis_voldaan" = function(x) x == 0,
        "adres_recentste_wijk_delfshaven" = function(x) x == 1
      )
    ),
    arch_combined_min = list(
      criteria = list(
        "persoon_geslacht_vrouw" = function(x) x == 0,
        "relatie_partner_totaal_dagen_partner" = function(x) x <= 30,
```

```
      "relatie_kind_huidige_aantal" = function(x) x == 0,
      "relatie_kind_heeft_kinderen" = function(x) x == 0,
      "persoonlijke_eigenschappen_taaleis_voldaan" = function(x) x == 1,
      "adres_recentste_wijk_delfshaven" = function(x) x == 1
    )
  ),
  arch_single_mother = list(
    criteria = list(
      "persoon_geslacht_vrouw" = function(x) x == 1,
      "relatie_partner_totaal_dagen_partner" = function(x) x <= 180,
      "relatie_kind_huidige_aantal" = function(x) x >= 1 & x <= 10,
      "relatie_kind_heeft_kinderen" = function(x) x == 1,
      "belemmering_dagen_financiele_problemen" = function(x) x >= 30
    )
  ),
  arch_single_mother_age = list(
    criteria = list(
      "persoon_geslacht_vrouw" = function(x) x == 1,
      "relatie_partner_totaal_dagen_partner" = function(x) x <= 180,
      "relatie_kind_huidige_aantal" = function(x) x >= 1 & x <= 10,
      "relatie_kind_heeft_kinderen" = function(x) x == 1,
      "persoon_leeftijd_bij_onderzoek" = function(x) x >= 25 & x <= 35,
      "belemmering_dagen_financiele_problemen" = function(x) x >= 30
    )
  ),
  good_dutch = list(
    criteria = list(
      "persoonlijke_eigenschappen_taaleis_voldaan" = function(x) x == 1,
      "persoonlijke_eigenschappen_dagen_sinds_taaleis" = function(x) x >= 365,
      "afspraak_afgelopen_jaar_ontheffing_taaleis" = function(x) x == 1,
      "persoonlijke_eigenschappen_taaleis_schrijfv_ok" = function(x) x == 1,
      "persoonlijke_eigenschappen_spreektaal_anders" = function(x) x == 0
    )
  ),
  bad_dutch = list(
    criteria = list(
      "persoonlijke_eigenschappen_taaleis_voldaan" = function(x) x == 0,
      "persoonlijke_eigenschappen_spreektaal_anders" = function(x) x == 1,
      "belemmering_hist_taal" = function(x) x == 1
    ),
    description = "Slechte␣Nederlandse␣taalvaardigheid,␣taaleis␣niet␣voldaan"
  ),
  struggling_dutch = list(
    criteria = list(
      "persoonlijke_eigenschappen_taaleis_voldaan" = function(x) x == 0,  # Taaleis nie
      "persoonlijke_eigenschappen_dagen_sinds_taaleis" = function(x) x >= 30,
# Taaleis al enige tijd
      "persoonlijke_eigenschappen_spreektaal_anders" = function(x) x == 1,  # Andere sp
      "belemmering_hist_taal" = function(x) x == 1  # Taalbelemmering
    ),
    description = "Moeite␣met␣Nederlands,␣taaleis␣niet␣voldaan␣maar␣wel␣bezig"
  ),
  language_requirement = list(
    criteria = list(
      "persoonlijke_eigenschappen_dagen_sinds_taaleis" = function(x) x >= 0  # Alle me
    ),
```

```
      description = "Alle mensen met taaleis vereiste"
    ),
    migrant_worker_language = list(
      criteria = list(
        "relatie_overig_historie_vorm__andere_inwonende" = function(x) x %in% c(0,3),
# Andere inwonende
        "persoonlijke_eigenschappen_taaleis_voldaan" = function(x) x %in% c(0,1),
# Taaleis voldaan of niet
        "persoonlijke_eigenschappen_dagen_sinds_taaleis" = function(x) x >= 0,
# Alle taaleis statussen
        "persoonlijke_eigenschappen_uitstroom_verw_vlgs_km" = function(x) x >= 0,
# Alle uitstroom verwachtingen
        "adres_recentste_wijk_delfshaven" = function(x) x == 1  # Woont in Delfshaven
      ),
      description = "Migrant worker met verschillende taalvaardigheden"
    ),
    migrant_worker_language_all = list(
      criteria = list(
        "relatie_overig_historie_vorm__andere_inwonende" = function(x) x %in% c(0,3),
# Andere inwonende (breder)
        "persoonlijke_eigenschappen_taaleis_voldaan" = function(x) x == 0,  # Taaleis nie
        "adres_recentste_wijk_delfshaven" = function(x) x == 1  # Woont in Delfshaven
      ),
      description = "Migrant worker, taaleis niet voldaan"
    ),
    migrant_worker_language_reintegration = list(
      criteria = list(
        "relatie_overig_historie_vorm__andere_inwonende" = function(x) x %in% c(0,3),
# Andere inwonende (breder)
        "persoonlijke_eigenschappen_taaleis_voldaan" = function(x) x %in% c(0,1),
# Verschillende taaleis statussen
        "persoonlijke_eigenschappen_dagen_sinds_taaleis" = function(x) x >= 0,
# Alle taaleis statussen
        "adres_recentste_wijk_delfshaven" = function(x) x == 1  # Woont in Delfshaven
      ),
      description = "Migrant worker in re ntegratie proces"
    ),
    migrant_worker_comment_no_comment = list(
      criteria = list(
        "relatie_overig_historie_vorm__andere_inwonende" = function(x) x %in% c(0,3),
# Andere inwonende (breder)
        "persoonlijke_eigenschappen_taaleis_voldaan" = function(x) x == 0,  # Taaleis nie
        "contacten_onderwerp_boolean_beoordelen_taaleis" = function(x) x == 0,
# Geen taaleis commentaar
        "afspraak_verzenden_beschikking_i_v_m__niet_voldoen_aan_wet_taaleis" = function(x
# Geen beschikking
        "contacten_onderwerp_boolean_taaleis___voldoet" = function(x) x == 0,  # Geen vo
        "afspraak_afgelopen_jaar_ontheffing_taaleis" = function(x) x == 0,  # Geen onthe
        "adres_recentste_wijk_delfshaven" = function(x) x == 1  # Woont in Delfshaven
      ),
      description = "Migrant worker zonder taaleis commentaar"
    ),
    migrant_worker_neighborhood = list(
      criteria = list(
        "relatie_overig_historie_vorm__andere_inwonende" = function(x) x %in% c(0,3),
# Andere inwonende
```

```
        "adres_recentste_wijk_charlois" = function(x) x == 0,  # Niet in Charlois
        "adres_recentste_wijk_feijenoord" = function(x) x == 0,  # Niet in Feijenoord
        "adres_recentste_wijk_ijsselmonde" = function(x) x == 0,  # Niet in IJsselmonde
        "adres_recentste_wijk_kralingen_c" = function(x) x == 0,  # Niet in Kralingen
        "adres_recentste_wijk_noord" = function(x) x == 0,  # Niet in Noord
        "adres_recentste_wijk_other" = function(x) x == 0,  # Niet in andere wijken
        "adres_recentste_wijk_prins_alexa" = function(x) x == 0,  # Niet in Prins Alexand
        "adres_recentste_buurt_nieuwe_westen" = function(x) x == 0,  # Niet in Nieuwe Wes
        "adres_recentste_buurt_other" = function(x) x == 1,  # In andere buurten
        "adres_recentste_buurt_groot_ijsselmonde" = function(x) x == 0,  # Niet in Groot
        "adres_recentste_buurt_oude_noorden" = function(x) x == 0,  # Niet in Oude Noorde
        "adres_recentste_buurt_vreewijk" = function(x) x == 0,  # Niet in Vreewijk
        "adres_recentst_onderdeel_rdam" = function(x) x == 1,  # In Rotterdam
        "adres_recentste_plaats_rotterdam" = function(x) x == 1,  # In Rotterdam
        "adres_recentste_plaats_other" = function(x) x == 0  # Niet in andere plaatsen
      ),
      description = "Migrant␣worker␣in␣specifieke␣buurten␣van␣Rotterdam"
    ),
    parent = list(
      criteria = list(
        "relatie_kind_huidige_aantal" = function(x) x >= 1 & x <= 10,  # 1-10 kinderen
        "relatie_kind_leeftijd_verschil_ouder_eerste_kind" = function(x) x >= 0 & x <= 50
# Leeftijdsverschil 0-50 jaar
        "relatie_kind_basisschool_kind" = function(x) x >= 0 & x <= 5  # 0-5 basisschool
      ),
      description = "Ouder␣met␣kinderen"
    )
  )

  # Create archetype mapping using flexible criteria
  archetype_mapping <- data.frame()

  for (archetype_name in names(flexible_archetypes)) {
    cat(sprintf("Processing␣flexible␣archetype:␣%s\n", archetype_name))

    archetype_def <- flexible_archetypes[[archetype_name]]
    criteria <- archetype_def$criteria
    description <- archetype_def$description

    cat(sprintf("␣␣Description:␣%s\n", description))

    # Start with all data
    filtered_data <- main_data

    # Apply each criterion
    for (var_name in names(criteria)) {
      if (var_name %in% colnames(filtered_data)) {
        logic_func <- criteria[[var_name]]
        filtered_data <- filtered_data %>% filter(logic_func(!!sym(var_name)))
        cat(sprintf("␣␣␣␣Applied␣%s:␣%d␣rows␣remaining\n", var_name, nrow(filtered_data))
      } else {
        cat(sprintf("␣␣␣␣Warning:␣Variable␣%s␣not␣found␣in␣data\n", var_name))
      }
    }

    if (nrow(filtered_data) > 0) {
```

```
      temp_mapping <- data.frame(
        id = filtered_data$id,
        archetype = archetype_name,
        stringsAsFactors = FALSE
      )
      archetype_mapping <- rbind(archetype_mapping, temp_mapping)
      cat(sprintf("  Added archetype: %s with %d samples\n", archetype_name, nrow(filtere
    } else {
      cat("Warning: No samples found for archetype", archetype_name, "\n")
    }
  }

  # Remove duplicates but keep all unique combinations
  archetype_mapping <- archetype_mapping %>% distinct()

  # Filter to only include IDs that exist in risk_scores
  risk_score_ids <- unique(risk_scores$id)
  archetype_mapping <- archetype_mapping %>% filter(id %in% risk_score_ids)

  write.csv(archetype_mapping, "archetype_id_mapping.csv", row.names = FALSE)
  cat("Created archetype mapping\n")
  cat("Filtered to only include IDs present in risk_scores data\n")
}

# Debug: Check archetype mapping
cat("Unique archetypes in mapping:", paste(unique(archetype_mapping$archetype), collapse
archetype_counts <- table(archetype_mapping$archetype)
cat("Archetype counts:\n")
print(archetype_counts)

# Check for overlapping IDs
id_counts <- table(archetype_mapping$id)
overlapping_ids <- sum(id_counts > 1)
cat("IDs that appear in multiple archetypes:", overlapping_ids, "\n")
if (overlapping_ids > 0) {
  cat("This is expected - people can belong to multiple archetypes\n")
}

# Merge risk scores with archetype mapping
risk_scores_with_archetypes <- merge(risk_scores, archetype_mapping, by = "id", all.x = T

# Debug: Check the merged data structure
cat("Risk scores shape:", nrow(risk_scores), "x", ncol(risk_scores), "\n")
cat("Archetype mapping shape:", nrow(archetype_mapping), "x", ncol(archetype_mapping), "\
cat("Merged data shape:", nrow(risk_scores_with_archetypes), "x", ncol(risk_scores_with_a

# Check how many unique IDs we have
cat("Unique IDs in risk_scores:", length(unique(risk_scores$id)), "\n")
cat("Unique IDs in archetype_mapping:", length(unique(archetype_mapping$id)), "\n")
cat("Unique IDs in merged data:", length(unique(risk_scores_with_archetypes$id)), "\n")

# Create wide format archetype columns - use the original risk_scores as base
cat("Creating wide format archetype columns...\n")
unique_archetypes <- unique(archetype_mapping$archetype)

# Start with the original risk_scores and add archetype columns
```

```
risk_scores_with_archetypes <- risk_scores

for (archetype in unique_archetypes) {
  # Create binary column for this archetype
  col_name <- paste0("archetype_", archetype)
  risk_scores_with_archetypes[[col_name]] <- as.integer(
    risk_scores_with_archetypes$id %in% archetype_mapping$id[archetype_mapping$archetype
  )

  # Count members
  n_members <- sum(risk_scores_with_archetypes[[col_name]])
  cat(sprintf("␣␣%s:␣%d␣members\n", archetype, n_members))

  # Debug: check if this makes sense
  expected_members <- sum(archetype_mapping$archetype == archetype)
  cat(sprintf("␣␣␣␣Expected␣members␣from␣mapping:␣%d\n", expected_members))
}

# Remove the original archetype column to avoid confusion
if ("archetype" %in% colnames(risk_scores_with_archetypes)) {
  risk_scores_with_archetypes$archetype <- NULL
}

# Check archetype distribution in merged data
if ("archetype" %in% colnames(risk_scores_with_archetypes)) {
  cat("Archetype␣distribution␣in␣merged␣data:\n")
  print(table(risk_scores_with_archetypes$archetype))
} else {
  cat("Warning:␣No␣'archetype'␣column␣in␣merged␣data\n")
}

# Function to calculate overrepresentation for a specific model
calculate_overrepresentation <- function(scores, archetype_ids, model_name) {
  if (length(scores) == 0 || all(is.na(scores))) {
    return(data.frame(
      archetype = "unknown",
      model = model_name,
      n_samples = 0,
      percent_in_top10 = 0,
      excess_share = 0,
      overall_percent_in_top10 = 0,
      relative_bias = 0
    ))
  }

  # Calculate top 10% threshold
  top10_threshold <- quantile(scores, 0.9, na.rm = TRUE)
  in_top10 <- scores >= top10_threshold

  # Overall statistics
  overall_percent_in_top10 <- mean(in_top10, na.rm = TRUE) * 100

  # Archetype-specific analysis
  archetype_in_top10 <- in_top10[archetype_ids == 1]
  n_archetype_samples <- sum(archetype_ids == 1, na.rm = TRUE)
```

```
  if (n_archetype_samples > 0) {
    percent_in_top10 <- mean(archetype_in_top10, na.rm = TRUE) * 100
    excess_share <- percent_in_top10 - 10  # Expected is 10%
    relative_bias <- percent_in_top10 - overall_percent_in_top10
  } else {
    percent_in_top10 <- 0
    excess_share <- 0
    relative_bias <- 0
  }

  return(data.frame(
    archetype = "archetype",
    model = model_name,
    n_samples = n_archetype_samples,
    percent_in_top10 = percent_in_top10,
    excess_share = excess_share,
    overall_percent_in_top10 = overall_percent_in_top10,
    relative_bias = relative_bias
  ))
}

# Calculate overrepresentation for each archetype and model
# Focus only on the debiased model - skip this step as we'll analyze it later
cat("Skipping overrepresentation calculation for all models - will focus on debiased mode

# Get archetype sizes for reference
archetype_sizes <- table(archetype_mapping$archetype)
cat("Archetype sizes:\n")
print(archetype_sizes)

# Create empty results for now - we'll analyze the debiased model in the final section
overrepresentation_results <- data.frame()

# Save overrepresentation results (empty for now - will be filled in final analysis)
write.csv(overrepresentation_results, "results/pca_archetype_overrepresentation.csv", row
cat("Overrepresentation results saved\n")

# Step 2: Create individual bias information
cat("Creating individual bias information...\n")

# Load original data with PCA variables
original_data <- read.csv("data/02_intermediate/synth_data_with_MD.csv", stringsAsFactors

# Load PCA variables
pca_vars <- read.csv("data/06_feature_information/pca_variable_names.csv", stringsAsFacto
cat("Loaded", length(pca_vars), "PCA variables\n")

# Check which PCA variables are available
available_pca_vars <- pca_vars[pca_vars %in% colnames(original_data)]
cat("Available PCA variables:", length(available_pca_vars), "\n")

# Merge with risk scores and archetype mapping
full_data <- merge(original_data, risk_scores_with_archetypes, by = "id", all.x = TRUE)

# Define highlight variables for feature-based bias analysis
highlight_vars <- list(
```

```
  vrouw = list(var = "persoon_geslacht_vrouw", logic = function(x) x == 1),
  delfshaven = list(var = "adres_recentste_wijk_delfshaven", logic = function(x) x == 1),
  heeft_kinderen = list(var = "relatie_kind_heeft_kinderen", logic = function(x) x == 1),
  taaleis = list(var = "persoonlijke_eigenschappen_taaleis_voldaan", logic = function(x)
  historie_andere_inwonende = list(var = "relatie_overig_historie_vorm__andere_inwonende'
  rec_plaats_rotterdam = list(var = "adres_recentste_plaats_rotterdam", logic = function(
  plannen_organiseren = list(var = "competentie_plannen_en_organiseren", logic = functior
  eigenschap_opstelling = list(var = "persoonlijke_eigenschappen_opstelling", logic = fur
)

# Function to calculate individual bias scores
calculate_individual_bias <- function(scores, archetype_ids, model_name) {
  if (length(scores) == 0 || all(is.na(scores))) {
    return(rep(0, length(scores)))
  }

  # Calculate top 10% threshold
  top10_threshold <- quantile(scores, 0.9, na.rm = TRUE)
  in_top10 <- scores >= top10_threshold

  # Calculate individual bias scores
  bias_scores <- rep(0, length(scores))

  # For archetype members, calculate their bias
  archetype_members <- which(archetype_ids == 1)
  if (length(archetype_members) > 0) {
    archetype_in_top10 <- in_top10[archetype_members]
    archetype_percent_in_top10 <- mean(archetype_in_top10, na.rm = TRUE) * 100

    # Calculate bias: difference from expected (10%) for archetype members
    # Don't weight by size - just use the raw difference
    bias_value <- archetype_percent_in_top10 - 10

    for (i in archetype_members) {
      bias_scores[i] <- bias_value
    }
  }

  return(bias_scores)
}

# Calculate bias scores for Ja_pseudo only
model_to_analyze <- "Ja_pseudo"
if (model_to_analyze %in% colnames(full_data)) {
  cat(sprintf("Calculating bias scores for %s...\n", model_to_analyze))

  for (archetype in unique_archetypes) {
    # Use the wide format archetype column
    archetype_col <- paste0("archetype_", archetype)
    if (archetype_col %in% colnames(full_data)) {
      archetype_ids <- full_data[[archetype_col]]
    } else {
      cat("Warning: Column not found, using fallback method\n")
      archetype_ids <- as.integer(full_data$id %in% archetype_mapping$id[archetype_mappir
    }
```

```
    n_archetype_members <- sum(archetype_ids, na.rm = TRUE)

    cat(sprintf("  Archetype %s: %d members\n", archetype, n_archetype_members))

    if (n_archetype_members > 0) {
      bias_scores <- calculate_individual_bias(full_data[[model_to_analyze]], archetype_i

      # Debug bias scores
      non_zero_bias <- sum(bias_scores != 0, na.rm = TRUE)
      cat(sprintf("    Non-zero bias scores: %d out of %d\n", non_zero_bias, length(bias_

      # Add bias scores to full_data
      bias_col_name <- paste0("bias_", model_to_analyze, "_", archetype)
      full_data[[bias_col_name]] <- bias_scores
    } else {
      cat(sprintf("    Warning: No members found for archetype %s\n", archetype))
    }
  }
} else {
  cat(sprintf("Warning: %s not found in data\n", model_to_analyze))
}

# ADDITIONAL: Feature-based bias analysis for Ja_pseudo only
cat("\n=== FEATURE-BASED BIAS ANALYSIS FOR JA_PSEUDO ===\n")

# Only analyze Ja_pseudo for feature-based bias
model_to_analyze <- "Ja_pseudo"
if (model_to_analyze %in% colnames(full_data)) {
  cat(sprintf("Analyzing feature-based bias for %s...\n", model_to_analyze))

  scores <- full_data[[model_to_analyze]]
  top10_threshold <- quantile(scores, 0.9, na.rm = TRUE)
  in_top10 <- scores >= top10_threshold
  overall_percent_in_top10 <- mean(in_top10, na.rm = TRUE) * 100

  cat(sprintf("Overall percent in top 10%%: %.2f%%\n", overall_percent_in_top10))

  for (feature_name in names(highlight_vars)) {
    feature_info <- highlight_vars[[feature_name]]
    var_name <- feature_info$var
    logic_func <- feature_info$logic

    if (var_name %in% colnames(full_data)) {
      # Apply the logic function to identify group members
      group_members <- logic_func(full_data[[var_name]])
      n_group_members <- sum(group_members, na.rm = TRUE)

      if (n_group_members > 0) {
        # Calculate bias for this feature group
        group_in_top10 <- in_top10[group_members]
        group_percent_in_top10 <- mean(group_in_top10, na.rm = TRUE) * 100
        excess_share <- group_percent_in_top10 - 10  # Expected is 10%
        relative_bias <- group_percent_in_top10 - overall_percent_in_top10

        cat(sprintf("  %s (%s): %d members, %.2f%% in top 10%%, excess: %.2f%%, relative
                    feature_name, var_name, n_group_members, group_percent_in_top10, exce
```

```
        # Create bias scores for this feature group
        bias_scores <- rep(0, nrow(full_data))
        bias_scores[group_members] <- excess_share

        # Add to full_data
        bias_col_name <- paste0("bias_", model_to_analyze, "_feature_", feature_name)
        full_data[[bias_col_name]] <- bias_scores
      } else {
        cat(sprintf("  %s (%s): No members found\n", feature_name, var_name))
      }
    } else {
      cat(sprintf("  %s (%s): Variable not found in data\n", feature_name, var_name))
    }
  }
} else {
  cat(sprintf("Warning: %s not found in data\n", model_to_analyze))
}

# Save data with bias scores
write.csv(full_data, "results/pca_data_with_bias_scores.csv", row.names = FALSE)
cat("Data with bias scores saved\n")

# Step 3: Calculate correlation between bias and PCA features
cat("Calculating correlations between bias and PCA features...\n")

# Get PCA feature columns only
pca_feature_cols <- available_pca_vars[available_pca_vars %in% colnames(full_data)]
cat("Number of PCA features to analyze:", length(pca_feature_cols), "\n")

# Debug: Check what bias columns were actually created
bias_columns <- grep("^bias_", colnames(full_data), value = TRUE)
cat("Bias columns found:", paste(bias_columns, collapse = ", "), "\n")

# Calculate correlations for Ja_pseudo only (not all models)
model_to_analyze <- "Ja_pseudo"
cat("Analyzing correlations for model:", model_to_analyze, "\n")

# Get archetype bias columns for Ja_pseudo
archetype_bias_cols <- grep(paste0("^bias_", model_to_analyze, "_"), colnames(full_data),
archetype_bias_cols <- archetype_bias_cols[!grepl("_feature_", archetype_bias_cols)]
# Exclude feature-based bias

# Get feature-based bias columns for Ja_pseudo
feature_bias_cols <- grep(paste0("^bias_", model_to_analyze, "_feature_"), colnames(full_

# Combine all bias columns for analysis
all_bias_cols <- c(archetype_bias_cols, feature_bias_cols)
cat("Bias columns to analyze:", paste(all_bias_cols, collapse = ", "), "\n")

# Calculate correlations for each bias type and PCA features
correlation_results <- data.frame()

correlation_count <- 0
for (bias_col in all_bias_cols) {
  if (bias_col %in% colnames(full_data)) {
```

```
    cat(sprintf("Calculating correlations for %s\n", bias_col))

    # Get bias scores for this bias type
    bias_scores <- full_data[[bias_col]]
    cat(sprintf("  Bias scores range: %.3f to %.3f\n", min(bias_scores, na.rm = TRUE), ma

    # Calculate correlations with PCA features
    for (feature in pca_feature_cols) {
      if (feature %in% colnames(full_data)) {
        feature_values <- full_data[[feature]]

        # Calculate correlation
        correlation <- cor(bias_scores, feature_values, use = "complete.obs")

        if (!is.na(correlation)) {
          # Determine bias type (archetype or feature)
          bias_type <- ifelse(grepl("_feature_", bias_col), "feature", "archetype")
          bias_name <- gsub(paste0("bias_", model_to_analyze, "_"), "", bias_col)
          bias_name <- gsub("_feature_", "", bias_name)

          correlation_results <- rbind(correlation_results, data.frame(
            model = model_to_analyze,
            bias_type = bias_type,
            bias_name = bias_name,
            feature = feature,
            correlation = correlation,
            abs_correlation = abs(correlation),
            stringsAsFactors = FALSE
          ))
          correlation_count <- correlation_count + 1
        }
      }
    }
  } else {
    cat("Warning: Bias column not found in full_data\n")
  }
}

cat("Total correlations calculated:", correlation_count, "\n")

# Debug: Check correlation results
cat("Correlation results shape:", nrow(correlation_results), "x", ncol(correlation_result
if (nrow(correlation_results) > 0) {
  cat("Correlation results columns:", paste(colnames(correlation_results), collapse = ",
  cat("Unique bias types in results:", paste(unique(correlation_results$bias_type), colla
  cat("Correlation range:", range(correlation_results$correlation), "\n")
} else {
  cat("No correlations were calculated. Possible issues:\n")
  cat("1. No bias columns were created\n")
  cat("2. No PCA features available\n")
  cat("3. All correlations were NA\n")
}

# Save correlation results
write.csv(correlation_results, "results/pca_bias_feature_correlations.csv", row.names = F
cat("Correlation results saved to: results/pca_bias_feature_correlations.csv\n")
```

```
# Step 4: Select and remove bias-sensitive features
cat("Selecting and removing bias-sensitive features...\n")

# Define correlation threshold for bias-sensitive features
correlation_threshold <- 0.10  # Features with |correlation| > 0.10 are considered bias-s

# Check if correlation_results has data and the required column
if (nrow(correlation_results) == 0) {
  cat("Warning: No correlation results found. Creating empty bias-sensitive features list
  bias_sensitive_features <- data.frame(
    feature = character(),
    max_correlation = numeric(),
    avg_correlation = numeric(),
    n_archetypes_affected = integer(),
    models_affected = character(),
    archetypes_affected = character(),
    stringsAsFactors = FALSE
  )
} else if (!"abs_correlation" %in% colnames(correlation_results)) {
  cat("Warning: abs_correlation column not found\n")
  bias_sensitive_features <- data.frame(
    feature = character(),
    max_correlation = numeric(),
    avg_correlation = numeric(),
    n_archetypes_affected = integer(),
    models_affected = character(),
    archetypes_affected = character(),
    stringsAsFactors = FALSE
  )
} else {
  # Find bias-sensitive features
  bias_sensitive_features <- correlation_results %>%
    filter(abs_correlation > correlation_threshold) %>%
    group_by(feature) %>%
    summarise(
      max_correlation = max(abs_correlation, na.rm = TRUE),
      avg_correlation = mean(abs_correlation, na.rm = TRUE),
      n_archetypes_affected = n(),
      models_affected = paste(unique(model), collapse = ", "),
      archetypes_affected = paste(unique(archetype), collapse = ", ")
    ) %>%
    arrange(desc(max_correlation))
}

# Save bias-sensitive features
write.csv(bias_sensitive_features, "results/pca_bias_sensitive_features.csv", row.names =
cat("Bias-sensitive features saved to: results/pca_bias_sensitive_features.csv\n")

cat("Found", nrow(bias_sensitive_features), "bias-sensitive features\n")
if (nrow(bias_sensitive_features) > 0) {
  cat("Top 10 most bias-sensitive features:\n")
  print(head(bias_sensitive_features, 10))
}

# Create debiased feature set (remove bias-sensitive features)
```

```
debiased_pca_features <- pca_feature_cols[!pca_feature_cols %in% bias_sensitive_features$
cat("Debiased PCA feature set contains", length(debiased_pca_features), "features\n")

# Save debiased feature list
write.csv(data.frame(feature = debiased_pca_features), "results/pca_debiased_features.csv
cat("Debiased PCA features saved to: results/pca_debiased_features.csv\n")

cat("\n=== BIAS ANALYSIS COMPLETE ===\n")
cat("Results saved to:\n")
cat("- results/pca_archetype_overrepresentation.csv\n")
cat("- results/pca_data_with_bias_scores.csv\n")
cat("- results/pca_bias_feature_correlations.csv\n")
cat("- results/pca_bias_sensitive_features.csv\n")
cat("- results/pca_debiased_features.csv\n")
cat("\nNow run train_pca_model.R to train models using these results.\n")
```

C.5 Selected Features Model Training

```
    # Train PCA models with different feature sets

library(caret)
library(dplyr)
library(ggplot2)
library(ggpubr)
library(rstatix)
library(stringr)
library(ini)
library(RJSONIO)

setwd("/Users/catootjeversluis/suspicion_machine")

# Load data and models
orig_list <- readRDS('data/01_raw/20220929_finale_model.rds')
model_orig <- orig_list$model[[1]]
synth_full <- read.csv("data/02_intermediate/synth_data_with_MD.csv", stringsAsFactors =

if (file.exists("results/risk_scores_comparison.csv")) {
  existing_scores <- read.csv("results/risk_scores_comparison.csv", stringsAsFactors = FA
  synth_full <- merge(synth_full, existing_scores, by = "id", all.x = TRUE)
} else {
  synth_full$Ja_pseudo <- predict(model_orig, synth_full, type = "prob")$Ja
}

pca_vars <- read.csv("data/06_feature_information/pca_variable_names.csv", stringsAsFacto
pca_46_vars <- read.csv("data/02_intermediate/pca_46_variables.csv", stringsAsFactors = F

# Function to train model
train_model <- function(feature_vars, model_name, data = synth_full) {
  ts_data <- data %>% select(all_of(feature_vars), Ja_pseudo)

  nzv <- nearZeroVar(ts_data)
  if (length(nzv) > 0) {
    ts_data <- ts_data[, -nzv]
  }

  ts_data <- ts_data[, sapply(ts_data, is.numeric)]
```

```
  target <- ts_data$Ja_pseudo
  ts_data$Ja_pseudo <- NULL
  ts_data$Ja_pseudo <- target

  set.seed(2025)
  model_file <- paste0(model_name, ".RData")

  if (file.exists(model_file)) {
    load(model_file)
  } else {
    model <- train(
      Ja_pseudo ~ .,
      data = ts_data,
      method = 'gbm',
      trControl = trainControl(method = 'cv', number = 5),
      verbose = FALSE,
      distribution = 'gaussian'
    )
    assign(model_name, model)
    save(list = model_name, file = model_file)
  }

  return(get(model_name))
}

# Train models
model_pca_46 <- train_model(pca_46_vars, "model_pca_46")
model_pca <- train_model(pca_vars, "model_pca")

# Score data
synth_scoredata <- read.csv("data/01_raw/synth_data_with_id.csv", stringsAsFactors = FALS

features_pca_46 <- synth_scoredata %>% select(all_of(pca_46_vars))
features_pca <- synth_scoredata %>% select(all_of(pca_vars))

synth_scoredata$Ja_pca <- predict(model_pca_46, features_pca_46, type = "raw")
synth_scoredata$Ja_pca_all <- predict(model_pca, features_pca, type = "raw")

# Calculate thresholds
synth_full$Ja_pca <- predict(model_pca_46, synth_full %>% select(all_of(pca_46_vars)), ty
td_synth_threshold_pca_46 <- quantile(synth_full$Ja_pca, probs = 0.90)

cat("46-features␣model␣threshold:", td_synth_threshold_pca_46, "\n")

synth_full$Ja_pca_all <- predict(model_pca, synth_full %>% select(all_of(pca_vars)), type
td_synth_threshold_pca <- quantile(synth_full$Ja_pca_all, probs = 0.90)

cat("156-features␣model␣threshold:", td_synth_threshold_pca, "\n")

# Train 58-features debiased model if debiased features exist
if (file.exists("results/pca_debiased_features.csv")) {
  debiased_features <- read.csv("results/pca_debiased_features.csv", stringsAsFactors = F

  ts_pca_debiased <- synth_full %>% select(all_of(debiased_features), Ja_pseudo)
```

```
  nzv_debiased <- nearZeroVar(ts_pca_debiased)
  if (length(nzv_debiased) > 0) {
    ts_pca_debiased <- ts_pca_debiased[, -nzv_debiased]
  }

  ts_pca_debiased <- ts_pca_debiased[, sapply(ts_pca_debiased, is.numeric)]

  target_debiased <- ts_pca_debiased$Ja_pseudo
  ts_pca_debiased$Ja_pseudo <- NULL
  ts_pca_debiased$Ja_pseudo <- target_debiased

  set.seed(2025)
  if (file.exists("model_pca_debiased.RData")) {
    load("model_pca_debiased.RData")
  } else {
    model_pca_debiased <- train(
      Ja_pseudo ~ .,
      data = ts_pca_debiased,
      method = 'gbm',
      trControl = trainControl(method = 'cv', number = 5),
      verbose = FALSE,
      distribution = 'gaussian'
    )
    save(model_pca_debiased, file = "model_pca_debiased.RData")
  }

  features_pca_debiased <- synth_scoredata %>% select(all_of(debiased_features))
  synth_scoredata$Ja_pca_debiased <- predict(model_pca_debiased, features_pca_debiased, t

  synth_full$Ja_pca_debiased <- predict(model_pca_debiased, synth_full %>% select(all_of(
  td_synth_threshold_pca_debiased <- quantile(synth_full$Ja_pca_debiased, probs = 0.90)

  cat("58-debiased-features␣model␣threshold:", td_synth_threshold_pca_debiased, "\n")
}

# Save results
write.csv(synth_scoredata, "results/risk_scores_comparison.csv", row.names = FALSE)
cat("Risk␣scores␣saved\n")

# Set thresholds for analysis
td_real_threshold <- 0.6970136
rw_threshold <- 0.5912492

# Predict function for PCA models
final_model.predict <- function(final_model, input_data, output_filename, model_type = "p
  drop_cols <- intersect(c("id", "data_type","MD"), names(input_data))

  if (model_type == "pca" && exists("pca_vars")) {
    available_pca_vars <- intersect(pca_vars, names(input_data))
    features <- input_data[, available_pca_vars, drop = FALSE]
    cat("Using", length(available_pca_vars), "PCA␣variables␣for␣prediction\n")
  } else {
    features <- input_data[, setdiff(names(input_data), drop_cols), drop = FALSE]
    cat("Using␣all␣available␣variables␣for␣prediction\n")
  }
```

```
  risk_scores <- predict.train(final_model, newdata = features, type = 'raw')
  risk_scores <- cbind(input_data, Ja = risk_scores, Nee=1-risk_scores)
  if (output_filename != "") {
    write.csv(risk_scores, output_filename, row.names = FALSE)
    cat("Risk scores saved to:", output_filename, "\n")
  }
  return(risk_scores)
}

# Prepare for analysis function
final_model.prepare_for_analysis <- function(variable_list, df, bins, nbins = 8, model_ty
  new_var_names <- list()
  missing_vars <- c()
  for (i in seq_along(variable_list)) {
    var_name <- NA_character_
    nested_list <- variable_list[[i]]
    if (is.data.frame(nested_list)) {
      base_feature <- names(nested_list)[1]
    } else if (is.list(nested_list) && length(nested_list) > 0 && !is.null(names(nested_l
      base_feature <- names(nested_list[[1]])[1]
    } else {
      base_feature <- NA_character_
    }
    if (length(bins) > 0 && base_feature %in% bins) {
      bin_var_name <- paste0('var_', i)
      if (base_feature %in% names(df)) {
        values <- df[[base_feature]]
        if (is.factor(values)) values <- as.numeric(as.character(values))
        df[[bin_var_name]] <- bin(values, nbins = nbins, method = 'content')
      } else {
        missing_vars <- unique(c(missing_vars, base_feature))
      }
      var_name <- bin_var_name
    }
    if (is.na(var_name)) {
      var_name <- base_feature
    }
    if (!is.null(var_name) && !is.na(var_name) && var_name != "" && length(var_name) > 0)
      new_var_names[[length(new_var_names) + 1]] <- var_name
    }
  }
  df$IV <- 'IV:\n'
  for (var in new_var_names) {
    if (is.null(var) || is.na(var) || var == "" || length(var) == 0) {
      next
    }
    if (!var %in% names(df)) {
      missing_vars <- unique(c(missing_vars, var))
      next
    }
    vals <- df[[var]]
    if (length(unique(vals)) <= 1) next
    df$IV <- paste0(df$IV, var, ': ', vals, '\n')
  }
  return(df)
}
```

```r
# Visual analysis function
final_model.visual_analysis <- function(sum_stats_list, decile_count_list, plot_list) {
  for(decile_df in decile_count_list){
    cur_dt <- decile_df[1,]$data_type
    p1 <- ggplot(decile_df, aes(x=decile, y=perc, color = as.factor(IV), group = IV))+
      geom_point(size = 2)+
      geom_line(linewidth = 1)+
      scale_color_brewer(palette = "Set3")+
      guides(color = guide_legend(title = "Groep (IV)", ncol = 1, override.aes = list(siz
      theme_minimal() +
      theme(legend.position = "right", legend.text = element_text(size = 8))+
      labs(x = 'Risk score decile', y = 'Group percentage in risk score decile', title =
    plot_list <- append(plot_list, list(p1))
  }

  for(sum_stats in sum_stats_list){
    cur_dt <- sum_stats[1,]$data_type
    p1 <- ggplot(sum_stats, aes(x=IV, y = perc_misrep_td_threshold, fill = IV))+
      geom_col(show.legend = FALSE, width = 0.5) +
      geom_hline(yintercept = 0)+
      theme(axis.title.x = element_text(margin=margin(t = 3)))+
      coord_flip()+
      theme_bw() +
      theme(axis.text.y = element_text(size = 8))+
      labs(x = 'Groep (IV)', y = 'Excess share in highest risk decile', title = paste0(cu
    plot_list <- append(plot_list, list(p1))
  }

  return (plot_list)
}

# Save plots function
final_model.save_plots <- function(plot_list, config, model_type) {
  save_dir <- file.path('results', 'statistical_parity', model_type, config$META$name)
  dir.create(save_dir, recursive = TRUE, showWarnings = FALSE)
  output_filename <- file.path(save_dir, paste0('plot_', model_type, '.pdf'))
  pdf(output_filename, width = 10, height = 7)
  print(plot_list)
  dev.off()
}

# Numeric analysis function
final_model.numeric_analysis <- function(df, dt, sum_stats_list, ttest_list, decile_count
  td_threshold <- ifelse(dt %in% c('real', 'real_conditional'), td_real_threshold, td_syn

  df_threshold <- df %>%
    dplyr::mutate(above_td_threshold = ifelse(df$Ja > td_threshold, 1, 0),
                  above_rw_threshold = ifelse(df$Ja > rw_threshold, 1, 0)) %>%
    dplyr::group_by(IV) %>%
    dplyr::summarise(share_above_td_threshold = mean(above_td_threshold),
                     share_above_rw_threshold = mean(above_rw_threshold)) %>%
    dplyr::mutate(perc_misrep_td_threshold = (share_above_td_threshold - 0.1) * 100,
                  perc_misrep_rw_threshold = (share_above_rw_threshold - 0.1) * 100)

  sum_stats <- df %>%
```

```
    dplyr::group_by(IV) %>%
    get_summary_stats(Ja, type = 'mean_sd')

  sum_stats <- dplyr::left_join(sum_stats, df_threshold, by = 'IV')
  sum_stats$data_type <- dt
  sum_stats_list <- append(sum_stats_list, list(sum_stats))

  if(length(unique(df$IV)) > 1){
    pwt <- df %>%
      pairwise_t_test(Ja~IV)
    pwt$data_type <- dt
    ttest_list <- append(ttest_list, list(pwt))
  }

  df$decile <- ntile(df$Ja, 10)
  df_decile_counts <- df %>%
    dplyr::group_by(decile, IV) %>%
    dplyr::count() %>%
    dplyr::group_by(IV) %>%
    dplyr::mutate(perc = n / sum(n) * 100)

  df_decile_counts$data_type <- dt
  decile_count_list <- append(decile_count_list, list(df_decile_counts))

  return(list(sum_stats_list, ttest_list, decile_count_list))
}

# Save tables function
final_model.save_tables <- function(sum_stats_list, ttest_list, decile_count_list, config
  save_dir <- file.path('results', 'statistical_parity', model_type, config$META$name)
  dir.create(save_dir, recursive = TRUE, showWarnings = FALSE)

  output_sum_stats <- file.path(save_dir, paste0('sum_stats_', model_type, '.csv'))
  output_ttest <- file.path(save_dir, paste0('t_test_', model_type, '.csv'))
  output_decile <- file.path(save_dir, paste0('decile_', model_type, '.csv'))

  all_sum_stats <- as.data.frame(dplyr::bind_rows(sum_stats_list))
  all_ttest <- as.data.frame(dplyr::bind_rows(ttest_list))
  all_decile <- as.data.frame(dplyr::bind_rows(decile_count_list))

  write.csv(all_sum_stats, output_sum_stats, row.names = FALSE)
  write.csv(all_ttest, output_ttest, row.names = FALSE)
  write.csv(all_decile, output_decile, row.names = FALSE)
  cat("All␣tables␣saved\n")
}

# Analyze function
final_model.analyze <- function(ini, final_model, model_type){
  config_filename <- file.path('conf/archetypes', ini)
  config <- read.ini(config_filename, encoding = 'utf8')
  input_filename <- file.path(
    'data','03_experiment_input',
    paste0(
      paste(config$META$date, config$META$user, config$META$name, sep = '_'),
      '.csv'
    )
```

```
  )
  output_filename <- file.path('data', '04_experiment_output', model_type,
    paste0(
      paste(config$META$date, config$META$user, config$META$name, sep = '_'),
      '.csv'
    )
  )
  dir.create(file.path('data', '04_experiment_output', model_type), recursive = TRUE, sho

  var_list <- config$VARIABLES$variable_list
  var_list <- str_replace_all(var_list, '"', '\"')
  var_list <- str_replace_all(var_list, '"', '\"')
  variable_list <- fromJSON(var_list)
  bin_list <- fromJSON(config$VARIABLES$bin)

  df <- read.csv(input_filename)
  risk_scores <- final_model.predict(final_model, df, output_filename, model_type = model
  if ("Ja" %in% names(risk_scores)) {
    if (is.factor(risk_scores$Ja)) risk_scores$Ja <- as.numeric(as.character(risk_scores$
    if (is.character(risk_scores$Ja)) risk_scores$Ja <- as.numeric(risk_scores$Ja)
  }

  risk_scores <- final_model.prepare_for_analysis(variable_list, risk_scores, bin_list, r

  all_data_types <- unique(risk_scores$data_type)

  plot_list <- list()
  sum_stats_list <- list()
  ttest_list <- list()
  decile_count_list <- list()

  for (dt in all_data_types){
    table_lists <- final_model.numeric_analysis(risk_scores[risk_scores$data_type == dt,]
    sum_stats_list <- table_lists[[1]]
    ttest_list <- table_lists[[2]]
    decile_count_list <- table_lists[[3]]
  }

  plot_list <- final_model.visual_analysis(sum_stats_list, decile_count_list, plot_list)

  final_model.save_plots(plot_list, config, model_type)
  final_model.save_tables(sum_stats_list, ttest_list, decile_count_list, config, model_ty
}
```

C.6 Ouput Comparison Original, Clean and Weighted Models

```
    # Combine all archetype results

library(dplyr)
library(ggplot2)
library(readr)
library(ggpubr)

# List all archetype folders
archetypes <- list.files("results/statistical_parity/archetypes")
```

```
# Function to read and tag CSV files for each archetype and model type
read_tagged <- function(folder, archetypes, file, model_type) {
  all <- list()
  for (arch in archetypes) {
    path <- file.path("results/statistical_parity", folder, arch, file)
    if (file.exists(path)) {
      df <- read_csv(path, show_col_types = FALSE, col_types = cols(.default = "c"))

      # Remove empty first column if it exists (common in PCA files)
      if (ncol(df) > 0 && colnames(df)[1] == "") {
        df <- df[, -1, drop = FALSE]
      }

      df$archetype <- arch
      df$model_type <- model_type
      all[[length(all) + 1]] <- df
    }
  }
  bind_rows(all)
}

# Function to safely combine data with existing files
safe_combine_data <- function(new_data, existing_file, description) {
  if (file.exists(existing_file)) {
    cat("Found existing file, adding new data\n")
    existing_data <- read_csv(existing_file, show_col_types = FALSE)

    existing_data <- existing_data %>% mutate(across(everything(), as.character))
    new_data <- new_data %>% mutate(across(everything(), as.character))

    combined_data <- bind_rows(existing_data, new_data)
    write_csv(combined_data, existing_file)
    cat("Updated file with new data\n")
  } else {
    cat("Creating new file with data\n")
    write_csv(new_data, existing_file)
  }
}

# Combine all sum_stats files into one dataframe
sumstats_all <- bind_rows(
  read_tagged("archetypes", archetypes, "sum_stats.csv", "archetypes"),
  read_tagged("clean", archetypes, "sum_stats.csv", "clean"),
  read_tagged("weighted", archetypes, "sum_stats.csv", "weighted")
)

pca_sumstats <- read_tagged("pca", archetypes, "sum_stats_pca.csv", "pca")
if (nrow(pca_sumstats) > 0) {
  sumstats_all <- bind_rows(sumstats_all, pca_sumstats)
  cat("Added PCA data\n")
}

safe_combine_data(sumstats_all, "results/statistical_parity/ALL_sum_stats_combined.csv",

# Combine all t_test files into one dataframe
getwd()
```

```
ttest_all <- bind_rows(
  read_tagged("archetypes", archetypes, "t_test.csv", "archetypes"),
  read_tagged("clean", archetypes, "t_test.csv", "clean"),
  read_tagged("weighted", archetypes, "t_test.csv", "weighted")
)

pca_ttest <- read_tagged("pca", archetypes, "t_test_pca.csv", "pca")
if (nrow(pca_ttest) > 0) {
  ttest_all <- bind_rows(ttest_all, pca_ttest)
  cat("Added␣PCA␣data\n")
}

safe_combine_data(ttest_all, "results/statistical_parity/ALL_t_tests_combined.csv", "t_te

# Combine all decile files into one dataframe
decile_all <- bind_rows(
  read_tagged("archetypes", archetypes, "decile.csv", "archetypes"),
  read_tagged("clean", archetypes, "decile.csv", "clean"),
  read_tagged("weighted", archetypes, "decile.csv", "weighted")
)

pca_decile <- read_tagged("pca", archetypes, "decile_pca.csv", "pca")
if (nrow(pca_decile) > 0) {
  decile_all <- bind_rows(decile_all, pca_decile)
  cat("Added␣PCA␣data\n")
}

safe_combine_data(decile_all, "results/statistical_parity/ALL_decile_combined.csv", "deci


# Read combined sum_stats and t_test CSVs
sumstats <- read_csv("results/statistical_parity/ALL_sum_stats_combined.csv")
ttests <- read_csv("results/statistical_parity/ALL_t_tests_combined.csv")

# Plot: Overrepresentation in highest decile per model/archetype (boxplot)
ggplot(sumstats, aes(x = archetype, y = perc_misrep_td_threshold, fill = model_type)) +
  geom_boxplot(position = position_dodge(width = 0.8)) +
  labs(title = "Overrepresentatie␣hoogste␣deciel␣per␣model/archetype",
       y = "Excess␣share␣in␣highest␣risk␣decile␣(%)",
       x = "Archetype") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Filter t-tests to remove NA p-values
ttests_clean <- ttests %>% filter(!is.na(p))

# Plot: p-values of t-tests per model/archetype (jitter plot)
ggplot(ttests_clean, aes(x = archetype, y = p, color = model_type)) +
  geom_jitter(width = 0.2, height = 0, alpha = 0.7) +
  geom_hline(yintercept = 0.05, linetype = "dashed", color = "red") +
  labs(title = "p-waarden␣t-test␣per␣model/archetype",
       y = "p-waarde",
       x = "Archetype") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Plot: Mean risk score per model/archetype (boxplot)
ggplot(sumstats, aes(x = archetype, y = mean, fill = model_type)) +
```

```
  geom_boxplot(position = position_dodge(width = 0.8)) +
  labs(title = "Gemiddelde risicoscore per model/archetype",
       y = "Gemiddelde risicoscore",
       x = "Archetype") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Convert columns to numeric for plotting
sumstats$perc_misrep_td_threshold <- as.numeric(sumstats$perc_misrep_td_threshold)
sumstats$mean <- as.numeric(sumstats$mean)

# Plot: Bias per archetype per model (boxplot), save as PNG
p<-ggplot(sumstats, aes(x = archetype, y = perc_misrep_td_threshold, fill = model_type))
  geom_boxplot(position = position_dodge(width = 0.8)) +
  labs(title = "Bias per archetype per model",
       y = "Excess share in highest risk decile (%)",
       x = "Archetype") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
ggsave("results/bias_per_archetype.png", plot = p, width = 10, height = 6, dpi = 300)

# Create summary table of mean bias per archetype and model type
summary_table<-sumstats %>%
  group_by(archetype, model_type) %>%
  summarise(mean_bias = mean(perc_misrep_td_threshold, na.rm = TRUE)) %>%
  tidyr::pivot_wider(names_from = model_type, values_from = mean_bias)

# Plot summary table as a table and save as PNG
tbl_plot <- ggtexttable(summary_table, rows = NULL)
ggsave("summary_table_plot.png", tbl_plot, width = 10, height = 6, dpi = 300)

# Convert summary table to long format for plotting
summary_long <- summary_table %>%
  pivot_longer(cols = -archetype, names_to = "model", values_to = "mean_bias")

# Plot: Horizontal barplot of average risk score per archetype across models
p2 <- ggplot(summary_long, aes(x = mean_bias, y = archetype, fill = model)) +
  geom_col(position = position_dodge(width = 0.7), width = 0.6) +
  geom_vline(xintercept = 0, linetype = "dashed") +
  labs(title = "Average Risk Score per Archetype across Models",
       x = "Bias (mean excess in top decile)",
       y = "Archetype") +
  scale_fill_manual(values = c("original" = "#D18900", "clean" = "#D95F02", "weighted" =
  theme_minimal(base_size = 12)
ggsave("results/average_bias_barplot.png", plot = p2, width = 12, height = 6, dpi = 300)
```

C.7 Output Comparison Feature Selected Models

C.8 Adverserial Loss training

```
    # Adversarial training with 156 PCA features
# Trains adversarial models for original, weighted, and 58-features models

library(reticulate)
use_condaenv("r-tf", required = TRUE)
library(keras)
```

```
library(tensorflow)
library(dplyr)
library(ggplot2)
np <- reticulate::import("numpy")

setwd("/Users/catootjeversluis/suspicion_machine")

# Load data and 156 PCA variables
main_data <- read.csv("data/01_raw/synth_data_with_id.csv", stringsAsFactors = FALSE)
pca_vars <- read.csv("data/06_feature_information/pca_variable_names.csv", stringsAsFacto

# Load ONLY the original model risk scores (not clean, weighted, etc.)
risk_scores <- read.csv("results/risk_scores_comparison.csv", stringsAsFactors = FALSE)
cat("Loading risk scores\n")

# Check if Ja_pseudo exists and handle duplicate columns
if ("Ja_pseudo" %in% colnames(risk_scores)) {
  # If there are multiple columns with the same name, take the first one
  ja_pseudo_cols <- which(colnames(risk_scores) == "Ja_pseudo")
  if (length(ja_pseudo_cols) > 1) {
    cat("Multiple Ja_pseudo columns found, using first\n")
    original_scores <- risk_scores[, c("id", ja_pseudo_cols[1]), drop = FALSE]
  } else {
    original_scores <- risk_scores[, c("id", "Ja_pseudo"), drop = FALSE]
  }
  colnames(original_scores)[2] <- "risk_score"
  cat("Loaded Ja_pseudo scores\n")
} else {
  cat("Error: Ja_pseudo column not found\n")
  stop("Ja_pseudo column not found")
}

# Merge data with original model scores only
main_data_with_scores <- merge(main_data, original_scores, by = "id", all.x = TRUE)

# Check merged data
cat("Merged data rows:", nrow(main_data_with_scores), "\n")

# Load archetype mapping (same approach as other scripts)
if (file.exists("archetype_id_mapping.csv")) {
  archetype_mapping <- read.csv("archetype_id_mapping.csv", stringsAsFactors = FALSE)
  cat("Loaded archetype mapping\n")
} else {
  cat("No archetype mapping found\n")
  stop("Missing archetype mapping file")
}

# Merge risk scores with archetype mapping (same as other scripts)
risk_scores_with_archetypes <- merge(risk_scores, archetype_mapping, by = "id", all.x = T

# Get valid archetypes
unique_archetypes <- unique(archetype_mapping$archetype)
archetype_counts <- table(archetype_mapping$archetype)
valid_archetypes <- names(archetype_counts[archetype_counts >= 100])

if(length(valid_archetypes) == 0) {
```

```
  cat("No valid archetypes found\n")
  stop("No valid archetypes found for adversarial training")
}

cat("Training for archetypes:", paste(valid_archetypes, collapse = ", "), "\n")

# Lambda tuning - focus on smaller values for better bias detection
# Start with very small values to avoid overwhelming the main task
lambda_values <- c(0, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.02, 0.05)

# Prepare 156 PCA features
available_pca_vars <- pca_vars[pca_vars %in% colnames(main_data_with_scores)]
cat("Available PCA features:", length(available_pca_vars), "\n")

# Prepare feature matrix (156 features)
X <- main_data_with_scores[, available_pca_vars, drop = FALSE]
X <- as.matrix(X)

# Remove any NA values
complete_cases <- complete.cases(X)
X <- X[complete_cases, ]
main_data_with_scores <- main_data_with_scores[complete_cases, ]

cat("Data shape:", nrow(X), "x", ncol(X), "\n")

# Convert to numpy arrays - USE risk_score as target (renamed from Ja_pseudo)
# Debug: Check if risk_score exists and has data
if (!"risk_score" %in% colnames(main_data_with_scores)) {
  cat("Error: risk_score column not found\n")
  stop("risk_score column not found")
}

if (all(is.na(main_data_with_scores$risk_score))) {
  cat("Error: All risk scores are NA\n")
  stop("No valid risk scores found")
}

cat("Risk scores loaded\n")
y_np <- np$array(matrix(main_data_with_scores$risk_score, ncol = 1))




# Function to evaluate bias in predictions
evaluate_bias <- function(predictions, archetype_labels, risk_scores) {
  # Calculate top 10% risk threshold based on ORIGINAL risk scores
  top10_threshold <- quantile(risk_scores, 0.9, na.rm = TRUE)
  in_top10 <- risk_scores >= top10_threshold

  # Calculate archetype representation in top 10%
  archetype_in_top10 <- in_top10[archetype_labels == 1]
  n_archetype_samples <- sum(archetype_labels == 1)

  if (n_archetype_samples > 0) {
    percent_in_top10 <- mean(archetype_in_top10) * 100
    excess_share <- percent_in_top10 - 10  # Expected is 10%
```

```
  } else {
    percent_in_top10 <- 0
    excess_share <- 0
  }

  # Also calculate bias using PREDICTED scores for comparison
  pred_top10_threshold <- quantile(predictions, 0.9, na.rm = TRUE)
  pred_in_top10 <- predictions >= pred_top10_threshold
  pred_archetype_in_top10 <- pred_in_top10[archetype_labels == 1]

  if (n_archetype_samples > 0) {
    pred_percent_in_top10 <- mean(pred_archetype_in_top10) * 100
    pred_excess_share <- pred_percent_in_top10 - 10
  } else {
    pred_percent_in_top10 <- 0
    pred_excess_share <- 0
  }

  return(list(
    percent_in_top10 = percent_in_top10,
    excess_share = excess_share,
    n_archetype_samples = n_archetype_samples,
    pred_percent_in_top10 = pred_percent_in_top10,
    pred_excess_share = pred_excess_share
  ))
}

# Function to train adversarial model for different model types
train_adversarial_model <- function(X_np, y_np, archetype_labels, lambda_val, archetype,
  cat(sprintf("Training␣%s␣(lambda=%.4f)\n", archetype, lambda_val))

  tryCatch({
    # Manual train/validation split
    set.seed(42)
    n_samples <- nrow(X_np)
    train_size <- as.integer(round(0.8 * n_samples))
    train_indices <- sample(1:n_samples, size = train_size)
    val_indices <- setdiff(1:n_samples, train_indices)

    X_train <- X_np[train_indices, , drop = FALSE]
    y_train <- y_np[train_indices, , drop = FALSE]
    z_train <- archetype_labels[train_indices, , drop = FALSE]

    X_val <- X_np[val_indices, , drop = FALSE]
    y_val <- y_np[val_indices, , drop = FALSE]
    z_val <- archetype_labels[val_indices, , drop = FALSE]

    # Build model (same architecture as original)
    input_dim <- ncol(X_np)
    input <- layer_input(shape = input_dim)

    main_hidden <- input %>%
      layer_dense(units = 64, activation = "relu") %>%
      layer_dense(units = 32, activation = "relu")
    main_output <- main_hidden %>%
      layer_dense(units = 1, activation = "linear", name = "main_output")
```

```
adversary_output <- main_hidden %>%
  layer_dense(units = 32, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid", name = "adv_output")

model <- keras_model(
  inputs = input,
  outputs = list(main_output, adversary_output)
)

model$compile(
  optimizer = "adam",
  loss = list(main_output = "mse", adv_output = "binary_crossentropy"),
  loss_weights = list(main_output = 1, adv_output = lambda_val)
)

# Add early stopping
early_stopping <- callback_early_stopping(
  monitor = "val_loss",
  patience = 10,
  restore_best_weights = TRUE
)

history <- model$fit(
  x = X_train,
  y = list(main_output = y_train, adv_output = z_train),
  validation_data = list(X_val, list(main_output = y_val, adv_output = z_val)),
  epochs = 50L,
  batch_size = 64L,
  verbose = 0,
  callbacks = list(early_stopping)
)

# Evaluate
adv_pred <- model$predict(X_np)
main_output <- adv_pred[[1]]
adv_output <- adv_pred[[2]]

# Calculate metrics
adv_classes <- as.integer(adv_output > 0.5)
adv_accuracy <- mean(adv_classes == archetype_labels)
adv_performance <- 1 - adv_accuracy
risk_scores_pred <- as.numeric(main_output)

# Evaluate bias using the same function as original
# For bias evaluation, we need to compare against the original risk scores
# Convert numpy array back to R vector for bias evaluation
original_risk_scores <- as.numeric(y_np)
bias_eval <- evaluate_bias(main_output, archetype_labels, original_risk_scores)
bias_reduction <- -abs(bias_eval$pred_excess_share)

# Get validation loss
if("metrics" %in% names(history)) {
  val_loss <- tail(history$metrics$val_main_output_loss, 1)
} else if("history" %in% names(history)) {
  val_loss <- tail(history$history$val_main_output_loss, 1)
} else {
```

```
      val_loss <- NA
    }

    # Create summary
    archetype_summary <- data.frame(
      archetype = archetype,
      lambda = lambda_val,
      model_type = model_type,
      n_rows = bias_eval$n_archetype_samples,
      orig_percent_in_top_10 = bias_eval$percent_in_top10,
      orig_excess_share = bias_eval$excess_share,
      pred_percent_in_top10 = bias_eval$pred_percent_in_top10,
      pred_excess_share = bias_eval$pred_excess_share,
      adv_accuracy = adv_accuracy,
      adv_performance = adv_performance,
      bias_reduction = bias_reduction,
      val_loss = val_loss,
      stringsAsFactors = FALSE
    )

    return(list(
      lambda = lambda_val,
      val_loss = val_loss,
      adv_accuracy = adv_accuracy,
      bias_eval = bias_eval,
      archetype_summary = archetype_summary,
      model = model
    ))

  }, error = function(e) {
    cat(sprintf("Error lambda %.4f: %s\n", lambda_val, e$message))
    return(NULL)
  })
}

# Function to save results for different model types
save_weighted_results <- function(results, model_type) {
  cat(sprintf("Saving %s model results\n", model_type))

  # Collect all summaries
  all_summaries <- list()

  for (archetype in names(results)) {
    for (lambda_name in names(results[[archetype]])) {
      result <- results[[archetype]][[lambda_name]]

      if (!is.null(result) && !is.null(result$archetype_summary) && nrow(result$archetype
        summary <- result$archetype_summary
        all_summaries[[length(all_summaries) + 1]] <- summary
      }
    }
  }

  if (length(all_summaries) > 0) {
    summary_df <- do.call(rbind, all_summaries)
```

```
    # Only replace NA values for weighted and pca_58 models, not for original
    if (model_type %in% c("weighted", "pca_58")) {
      # Check for NA values and replace with meaningful defaults
      summary_df$orig_excess_share[is.na(summary_df$orig_excess_share)] <- 0
      summary_df$pred_excess_share[is.na(summary_df$pred_excess_share)] <- 0
      summary_df$bias_reduction[is.na(summary_df$bias_reduction)] <- 0
      summary_df$adv_accuracy[is.na(summary_df$adv_accuracy)] <- 0.5
      summary_df$adv_performance[is.na(summary_df$adv_performance)] <- 0.5
      summary_df$val_loss[is.na(summary_df$val_loss)] <- 0
    }

    # Save results to file
    output_file <- sprintf("results/adversarial_156_features_%s_model_results.csv", model
    write.csv(summary_df, output_file, row.names = FALSE)
    cat("Results saved\n")

    # Print summary
    cat(sprintf("Summary %s model:\n", model_type))
    print(summary_df)

  } else {
    cat(sprintf("No results for %s model\n", model_type))
  }
}

# Function to calculate feature importance using gradients
calculate_feature_importance <- function(model, X_val, feature_names) {
  # Get gradients with respect to input features
  input_tensor <- model$input
  main_output_tensor <- model$output[[1]]

  # Calculate gradients for main output
  gradients <- tensorflow::tf$gradients(main_output_tensor, input_tensor)[[1]]

  # Get gradient values
  grad_values <- tensorflow::tf$reduce_mean(tensorflow::tf$abs(gradients), axis = 0L)

  # Convert to R
  grad_np <- as.array(grad_values)

  # Create importance dataframe
  importance_df <- data.frame(
    feature = feature_names,
    importance = grad_np,
    stringsAsFactors = FALSE
  ) %>%
    arrange(desc(importance))

  return(importance_df)
}

# === ADVERSARIAL TRAINING ON ORIGINAL MODEL ===
cat("Training original model\n")

# Check if original model adversarial results already exist
original_results_file <- "results/adversarial_156_features_original_model_results.csv"
```

```
should_train_original <- TRUE

if (file.exists(original_results_file)) {
  cat("Loading existing original model results\n")
  original_results_existing <- read.csv(original_results_file, stringsAsFactors = FALSE)
  cat("Loaded existing results\n")

  # Check if we have results for all archetypes and lambda values
  existing_archetypes_orig <- unique(original_results_existing$archetype)
  existing_lambdas_orig <- unique(original_results_existing$lambda)

  cat("Existing archetypes and lambdas found\n")

  # Check if we have complete results
  if (length(existing_archetypes_orig) >= length(valid_archetypes) &&
      length(existing_lambdas_orig) >= length(lambda_values)) {
    cat("Complete results found, skipping training\n")
    should_train_original <- FALSE
  } else {
    cat("Incomplete results, retraining\n")
  }
} else {
  cat("No existing results found\n")
}

# Train original model if needed
results <- list()
feature_importance_results <- list()

if (should_train_original) {
  cat("Starting original model training\n")

  for (archetype in valid_archetypes) {
    cat(sprintf("Training archetype: %s\n", archetype))

    # Create binary archetype (1 = this archetype, 0 = not this archetype)
    archetype_ids <- as.integer(main_data_with_scores$id %in% archetype_mapping$id[archet

    cat(sprintf("Archetype: %s (%d samples)\n", archetype, sum(archetype_ids)))

    # Create binary: 1 if in archetype dataset, 0 if not
    z_binary <- archetype_ids
    z_np <- np$array(matrix(z_binary, ncol = 1))

    cat("Distribution:", table(z_binary), "\n")

    archetype_results <- list()
    best_lambda <- NULL
    best_bias_reduction <- -Inf

    for (lambda in lambda_values) {
      cat(sprintf("Lambda = %.3f\n", lambda))

      tryCatch({
        # Manual train/validation split
        set.seed(42)
```

```
n_samples <- nrow(X_np)
train_size <- as.integer(round(0.8 * n_samples))
train_indices <- sample(1:n_samples, size = train_size)
val_indices <- setdiff(1:n_samples, train_indices)

X_train <- X_np[train_indices, , drop = FALSE]
y_train <- y_np[train_indices, , drop = FALSE]
z_train <- z_np[train_indices, , drop = FALSE]

X_val <- X_np[val_indices, , drop = FALSE]
y_val <- y_np[val_indices, , drop = FALSE]
z_val <- z_np[val_indices, , drop = FALSE]

# Build model (same as working script)
input_dim <- ncol(X)
input <- layer_input(shape = input_dim)

main_hidden <- input %>%
  layer_dense(units = 64, activation = "relu") %>%
  layer_dense(units = 32, activation = "relu")
main_output <- main_hidden %>%
  layer_dense(units = 1, activation = "linear", name = "main_output")
adversary_output <- main_hidden %>%
  layer_dense(units = 32, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid", name = "adv_output")

model <- keras_model(
  inputs = input,
  outputs = list(main_output, adversary_output)
)

model$compile(
  optimizer = "adam",
  loss = list(main_output = "mse", adv_output = "binary_crossentropy"),
  loss_weights = list(main_output = 1, adv_output = lambda)
)

# Add early stopping to prevent overfitting
early_stopping <- callback_early_stopping(
  monitor = "val_loss",
  patience = 10,
  restore_best_weights = TRUE
)

history <- model$fit(
  x = X_train,
  y = list(main_output = y_train, adv_output = z_train),
  validation_data = list(X_val, list(main_output = y_val, adv_output = z_val)),
  epochs = 50L,
  batch_size = 64L,
  verbose = 0,
  callbacks = list(early_stopping)
)

# Evaluate
adv_pred <- model$predict(X_np)
```

```
        main_output <- adv_pred[[1]]
        adv_output <- adv_pred[[2]]

        # Calculate adversarial accuracy
        adv_predictions <- as.numeric(adv_output > 0.5)
        adv_accuracy <- mean(adv_predictions == z_binary)

        # Calculate validation loss
        val_loss <- history$metrics$val_loss[length(history$metrics$val_loss)]

        # Evaluate bias using original risk scores
        bias_eval <- evaluate_bias(main_output, z_binary, y_np)

        # Store results
        archetype_results[[as.character(lambda)]] <- list(
          lambda = lambda,
          val_loss = val_loss,
          adv_accuracy = adv_accuracy,
          orig_percent_in_top_10 = bias_eval$percent_in_top10,
          orig_excess_share = bias_eval$excess_share,
          pred_percent_in_top10 = bias_eval$pred_percent_in_top10,
          pred_excess_share = bias_eval$pred_excess_share,
          bias_reduction = bias_eval$excess_share - bias_eval$pred_excess_share
        )

        # Track best lambda for this archetype
        current_bias_reduction <- bias_eval$excess_share - bias_eval$pred_excess_share
        if (current_bias_reduction > best_bias_reduction) {
          best_bias_reduction <- current_bias_reduction
          best_lambda <- lambda
        }

        cat(sprintf("Lambda %.3f: Val Loss %.4f, Adv Acc %.3f, Bias Reduction %.2f\n",
                    lambda, val_loss, adv_accuracy, current_bias_reduction))

      }, error = function(e) {
        cat(sprintf("Error lambda %.3f: %s\n", lambda, e$message))
      })
    }

    # Store best model for feature importance analysis
    if (!is.null(best_lambda) && !is.null(archetype_results[[as.character(best_lambda)]]))
      feature_importance_results[[archetype]] <- list(
        lambda = best_lambda,
        model = archetype_results[[as.character(best_lambda)]]$model,
        bias_reduction = best_bias_reduction
      )
    }

    results[[archetype]] <- archetype_results
  }

  cat("Original model adversarial training completed.\n")
} else {
  cat("Original model training skipped - results already exist.\n")
}
```

```
# Process results for original model
if (length(results) > 0) {
  # Combined summary for all archetypes and lambdas
  all_summaries <- list()

  for (archetype in names(results)) {
    archetype_results <- results[[archetype]]

    for (lambda_name in names(archetype_results)) {
      result <- archetype_results[[lambda_name]]
      if (!is.null(result) && is.list(result)) {
        if (all(c("lambda", "val_loss", "adv_accuracy", "archetype_summary") %in% names(r
          if (!is.null(result$archetype_summary) && is.data.frame(result$archetype_summar
            summary_row <- result$archetype_summary[1, ]
            if ("orig_excess_share" %in% names(summary_row) && !is.na(summary_row$orig_ex
              all_summaries[[paste(archetype, lambda_name, sep = "_")]] <- data.frame(
                archetype = archetype,
                lambda = result$lambda,
                val_loss = result$val_loss,
                adv_accuracy = result$adv_accuracy,
                adv_performance = summary_row$adv_performance,
                n_rows = summary_row$n_rows,
                percent_in_top_10 = summary_row$orig_percent_in_top_10,
                excess_share = summary_row$orig_excess_share,
                bias_reduction = summary_row$bias_reduction,
                stringsAsFactors = FALSE
              )
            }
          }
        }
      }
    }
  }

  if (length(all_summaries) > 0) {
    summary_df <- do.call(rbind, all_summaries)
    cat("\nCombined summary for all archetypes and lambdas:\n")
    print(summary_df)

    # Summary by archetype
    cat("\nSummary by archetype:\n")
    for (archetype in names(results)) {
      cat(sprintf("\n%s:\n", archetype))
      archetype_summary <- summary_df[summary_df$archetype == archetype, ]
      if (nrow(archetype_summary) > 0) {
        # Check which columns exist before printing
        available_cols <- intersect(c("lambda", "val_loss", "adv_accuracy", "adv_performa
                                      names(archetype_summary))
        if (length(available_cols) > 0) {
          print(archetype_summary[, available_cols])
        } else {
          print(archetype_summary)
        }
      } else {
        cat("  No successful results for this archetype\n")
```

```
      }
    }

    # Save results to file
    write.csv(summary_df, "results/adversarial_156_features_original_model_results.csv",
    cat("\nResults saved to: results/adversarial_156_features_original_model_results.csv\
  } else {
    cat("No successful results to summarize\n")
  }
}
# FEATURE IMPORTANCE ANALYSIS FOR BIAS REDUCTION
cat("\n=== FEATURE IMPORTANCE ANALYSIS FOR BIAS REDUCTION ===\n")

# Analyze feature importance for bias reduction (only if new results were generated)
if (length(feature_importance_results) > 0) {
  all_feature_importance <- data.frame()

  for (archetype in names(feature_importance_results)) {
    cat(sprintf("\n--- Analyzing feature importance for archetype: %s ---\n", archetype))

    result <- feature_importance_results[[archetype]]
    model <- result$model

    tryCatch({
      # Calculate feature importance
      feature_importance <- calculate_feature_importance(model, X_np, available_pca_vars)

      # Add archetype info
      feature_importance$archetype <- archetype
      feature_importance$lambda <- result$lambda
      feature_importance$bias_reduction <- result$bias_reduction

      all_feature_importance <- rbind(all_feature_importance, feature_importance)

      cat(sprintf("Feature importance calculated for %s (lambda=%.3f, bias_reduction=%.2f
                  archetype, result$lambda, result$bias_reduction))

    }, error = function(e) {
      cat(sprintf("Error calculating feature importance for %s: %s\n", archetype, e$messa
    })
  }

  # Calculate average feature importance across all archetypes
  if (nrow(all_feature_importance) > 0) {
    feature_summary <- all_feature_importance %>%
      group_by(feature) %>%
      summarise(
        avg_importance = mean(importance, na.rm = TRUE),
        max_importance = max(importance, na.rm = TRUE),
        min_importance = min(importance, na.rm = TRUE),
        std_importance = sd(importance, na.rm = TRUE),
        avg_lambda = mean(lambda, na.rm = TRUE),
        avg_bias_reduction = mean(bias_reduction, na.rm = TRUE),
        .groups = 'drop'
      ) %>%
      arrange(desc(avg_importance))
```

```
# Identify important features for bias reduction (top 25%)
importance_threshold <- quantile(feature_summary$avg_importance, 0.75)
important_features <- feature_summary %>%
  filter(avg_importance >= importance_threshold) %>%
  arrange(desc(avg_importance))

# Identify features that can be removed (low importance)
low_importance_threshold <- quantile(feature_summary$avg_importance, 0.25)
low_importance_features <- feature_summary %>%
  filter(avg_importance <= low_importance_threshold) %>%
  arrange(avg_importance)

# Save feature analysis results
write.csv(feature_summary, "results/adversarial_156_features_original_model_importanc
write.csv(important_features, "results/adversarial_156_features_original_model_import
write.csv(low_importance_features, "results/adversarial_156_features_original_model_l

cat("\nFeature␣analysis␣results␣saved␣to:\n")
cat("-␣results/adversarial_156_features_original_model_importance_summary.csv\n")
cat("-␣results/adversarial_156_features_original_model_important_features.csv\n")
cat("-␣results/adversarial_156_features_original_model_low_importance_features.csv\n"

# Print recommendations
cat("\n===␣FEATURE␣SELECTION␣RECOMMENDATIONS␣FOR␣ORIGINAL␣MODEL␣===\n")
cat(sprintf("Total␣features␣analyzed:␣%d\n", nrow(feature_summary)))
cat(sprintf("Important␣features␣for␣bias␣reduction␣(top␣25%%):␣%d\n", nrow(important_
cat(sprintf("Low-importance␣features␣(bottom␣25%%):␣%d\n", nrow(low_importance_featur

cat("\nTop␣20␣most␣important␣features␣for␣bias␣reduction:\n")
print(head(important_features, 20))

cat("\nFeatures␣to␣consider␣removing␣(low␣importance):\n")
print(head(low_importance_features, 20))

# Create visualizations
if (require(ggplot2)) {
  # Plot feature importance distribution
  p1 <- ggplot(feature_summary, aes(x = avg_importance)) +
    geom_histogram(bins = 30, fill = "steelblue", alpha = 0.7) +
    geom_vline(xintercept = importance_threshold, color = "red", linetype = "dashed")
    geom_vline(xintercept = low_importance_threshold, color = "orange", linetype = "d
    labs(title = "Distribution␣of␣Feature␣Importance␣for␣Bias␣Reduction␣(Original␣Mod
         x = "Average␣Feature␣Importance",
         y = "Count") +
    theme_minimal()

  ggsave("results/adversarial_156_features_original_model_importance_distribution.png

  # Plot top features
  p2 <- ggplot(head(important_features, 20), aes(x = reorder(feature, avg_importance)
    geom_col(fill = "steelblue") +
    coord_flip() +
    labs(title = "Top␣20␣Most␣Important␣Features␣for␣Bias␣Reduction␣(Original␣Model)"
         x = "Feature",
         y = "Average␣Importance") +
```

```
        theme_minimal()

      ggsave("results/adversarial_156_features_original_model_top_features.png", plot = p

      # Plot importance vs bias reduction
      p3 <- ggplot(feature_summary, aes(x = avg_importance, y = avg_bias_reduction)) +
        geom_point(alpha = 0.7) +
        geom_text(data = head(feature_summary, 10),
                  aes(label = feature), hjust = -0.1, size = 3) +
        labs(title = "Feature Importance vs Bias Reduction (Original Model)",
             x = "Average Feature Importance",
             y = "Average Bias Reduction") +
        theme_minimal()

      ggsave("results/adversarial_156_features_original_model_importance_vs_bias.png", pl
    }
  }
}

# Print summary (only if new results were generated)
if (length(results) > 0) {
  cat("\n=== SUMMARY FOR ORIGINAL MODEL ===\n")
  for (archetype in names(results)) {
    cat(sprintf("\nArchetype: %s\n", archetype))
    archetype_results <- results[[archetype]]

    for (lambda_name in names(archetype_results)) {
      if (!is.null(archetype_results[[lambda_name]]$archetype_summary)) {
        summary <- archetype_results[[lambda_name]]$archetype_summary
        # Check if summary is a data frame and has the required columns
        if (is.data.frame(summary) && nrow(summary) > 0) {
          summary_row <- summary[1, ]  # Take first row

          # Only print if we have meaningful results
          if ("orig_excess_share" %in% names(summary_row) && !is.na(summary_row$orig_exce
            cat(sprintf("  Lambda %.3f: Excess Share = %.2f%%, Adv Acc = %.3f, Bias Reduc
                        summary_row$lambda, summary_row$orig_excess_share, summary_row$ac
          }
        }
      }
    }
  }
} else {
  cat("Original model summary skipped - no new results generated.\n")
}

cat("\n=== ADVERSARIAL TRAINING WITH 156 FEATURES - ORIGINAL MODEL COMPLETE ===\n")
cat("Results show how different lambda values affect bias reduction in the original model
cat("Lower adversarial accuracy = less bias in the model.\n")
cat("Negative excess share = reduced bias.\n")
cat("\nFEATURE RECOMMENDATIONS FOR ORIGINAL MODEL:\n")
if (exists("important_features") && nrow(important_features) > 0) {
  cat(sprintf("- Keep the top %d features for best bias reduction\n", nrow(important_feat
  cat(sprintf("- Consider removing the bottom %d features\n", nrow(low_importance_feature
  cat("Check the CSV files for detailed feature rankings.\n")
} else {
```

```
  cat("No feature importance analysis completed.\n")
}

# === ADVERSARIAL TRAINING ON WEIGHTED MODEL ===
cat("\n\n=== ADVERSARIAL TRAINING ON WEIGHTED MODEL ===\n")

# Check if weighted model adversarial results already exist
weighted_results_file <- "results/adversarial_156_features_weighted_model_results.csv"
if (file.exists(weighted_results_file)) {
  cat("Weighted model adversarial results already exist. Loading existing results...\n")
  weighted_results_existing <- read.csv(weighted_results_file, stringsAsFactors = FALSE)
  cat("Loaded", nrow(weighted_results_existing), "existing results for weighted model\n")

  # Check if we have results for all archetypes and lambda values
  existing_archetypes <- unique(weighted_results_existing$archetype)
  existing_lambdas <- unique(weighted_results_existing$lambda)

  cat("Existing archetypes:", paste(existing_archetypes, collapse = ", "), "\n")
  cat("Existing lambda values:", paste(existing_lambdas, collapse = ", "), "\n")

  # Check if we have complete results
  if (length(existing_archetypes) >= length(valid_archetypes) &&
      length(existing_lambdas) >= length(lambda_values)) {
    cat("Complete weighted model adversarial results found. Skipping training.\n")
    results_weighted <- list()  # Empty list to indicate results exist
  } else {
    cat("Incomplete results found. Will retrain missing combinations.\n")
    results_weighted <- NULL
  }
} else {
  cat("No existing weighted model adversarial results found.\n")
  results_weighted <- NULL
}

# Only train if results don't exist or are incomplete
if (is.null(results_weighted) && "Ja_weighted" %in% colnames(risk_scores)) {
  cat("Loading weighted model scores for adversarial training...\n")

  # Create weighted model data
  weighted_scores <- risk_scores[, c("id", "Ja_weighted")]
  colnames(weighted_scores)[2] <- "risk_score"

  # Merge with main data
  main_data_weighted <- merge(main_data, weighted_scores, by = "id", all.x = TRUE)

  # Remove NA values
  complete_cases_weighted <- complete.cases(main_data_weighted$risk_score)
  main_data_weighted <- main_data_weighted[complete_cases_weighted, ]

  cat("Weighted model data shape:", nrow(main_data_weighted), "x", ncol(main_data_weighte

  # Prepare feature matrix for weighted model (same 156 features)
  X_weighted <- main_data_weighted[, available_pca_vars, drop = FALSE]
  X_weighted <- as.matrix(X_weighted)

  # Convert to numpy arrays
```

```
  X_weighted_np <- np$array(X_weighted)
  y_weighted_np <- np$array(matrix(main_data_weighted$risk_score, ncol = 1))

  # Train adversarial models for weighted model
  results_weighted <- list()

  for (archetype in valid_archetypes) {
    cat(sprintf("\nTraining␣adversarial␣model␣for␣archetype:␣%s␣(Weighted␣Model)\n", arch

    # Create archetype labels (same approach as original model)
    archetype_ids <- as.integer(main_data_weighted$id %in% archetype_mapping$id[archetype

    if (sum(archetype_ids) < 50) {
      cat(sprintf("Warning:␣Only␣%d␣samples␣for␣archetype␣%s,␣skipping\n", sum(archetype_
      next
    }

    # Create binary archetype labels
    archetype_labels <- np$array(matrix(archetype_ids, ncol = 1))

    results_weighted[[archetype]] <- list()

    for (lambda_val in lambda_values) {
      cat(sprintf("␣␣Lambda␣%.4f...␣", lambda_val))

      # Train adversarial model for weighted model
      model_result <- train_adversarial_model(
        X_weighted_np, y_weighted_np, archetype_labels,
        lambda_val, archetype, "weighted"
      )

      results_weighted[[archetype]][[sprintf("lambda_%.4f", lambda_val)]] <- model_result
      cat("Done\n")
    }
  }

  # Save weighted model results
  save_weighted_results(results_weighted, "weighted")

} else if (is.null(results_weighted)) {
  cat("Warning:␣Ja_weighted␣column␣not␣found␣in␣risk_scores_comparison.csv\n")
  cat("Skipping␣weighted␣model␣adversarial␣training\n")
}

# === ADVERSARIAL TRAINING ON 58-FEATURES MODEL ===
cat("\n\n===␣ADVERSARIAL␣TRAINING␣ON␣58-FEATURES␣MODEL␣===\n")

# Check if 58-features model adversarial results already exist
pca_58_results_file <- "results/adversarial_156_features_pca_58_model_results.csv"
if (file.exists(pca_58_results_file)) {
  cat("58-features␣model␣adversarial␣results␣already␣exist.␣Loading␣existing␣results...\n
  pca_58_results_existing <- read.csv(pca_58_results_file, stringsAsFactors = FALSE)
  cat("Loaded", nrow(pca_58_results_existing), "existing␣results␣for␣58-features␣model\n'

  # Check if we have results for all archetypes and lambda values
  existing_archetypes_58 <- unique(pca_58_results_existing$archetype)
```

```
  existing_lambdas_58 <- unique(pca_58_results_existing$lambda)

  cat("Existing archetypes:", paste(existing_archetypes_58, collapse = ", "), "\n")
  cat("Existing lambda values:", paste(existing_lambdas_58, collapse = ", "), "\n")

  # Check if we have complete results
  if (length(existing_archetypes_58) >= length(valid_archetypes) &&
      length(existing_lambdas_58) >= length(lambda_values)) {
    cat("Complete 58-features model adversarial results found. Skipping training.\n")
    results_58 <- list()  # Empty list to indicate results exist
  } else {
    cat("Incomplete results found. Will retrain missing combinations.\n")
    results_58 <- NULL
  }
} else {
  cat("No existing 58-features model adversarial results found.\n")
  results_58 <- NULL
}

# Only train if results don't exist or are incomplete
if (is.null(results_58) && "Ja_pca_debiased" %in% colnames(risk_scores)) {
  cat("Loading 58-features model scores for adversarial training...\n")

  # Create 58-features model data
  pca_58_scores <- risk_scores[, c("id", "Ja_pca_debiased")]
  colnames(pca_58_scores)[2] <- "risk_score"

  # Merge with main data
  main_data_58 <- merge(main_data, pca_58_scores, by = "id", all.x = TRUE)

  # Remove NA values
  complete_cases_58 <- complete.cases(main_data_58$risk_score)
  main_data_58 <- main_data_58[complete_cases_58, ]

  cat("58-features model data shape:", nrow(main_data_58), "x", ncol(main_data_58), "\n")

  # Prepare feature matrix for 58-features model (same 156 features for consistency)
  X_58 <- main_data_58[, available_pca_vars, drop = FALSE]
  X_58 <- as.matrix(X_58)

  # Convert to numpy arrays
  X_58_np <- np$array(X_58)
  y_58_np <- np$array(matrix(main_data_58$risk_score, ncol = 1))

  # Train adversarial models for 58-features model
  results_58 <- list()

  for (archetype in valid_archetypes) {
    cat(sprintf("\nTraining adversarial model for archetype: %s (58-Features Model)\n", a

    # Create archetype labels (same approach as original model)
    archetype_ids <- as.integer(main_data_58$id %in% archetype_mapping$id[archetype_mappi

    if (sum(archetype_ids) < 50) {
      cat(sprintf("Warning: Only %d samples for archetype %s, skipping\n", sum(archetype_
      next
```

```
    }

    # Create binary archetype labels
    archetype_labels <- np$array(matrix(archetype_ids, ncol = 1))

    results_58[[archetype]] <- list()

    for (lambda_val in lambda_values) {
      cat(sprintf("  Lambda %.4f... ", lambda_val))

      # Train adversarial model for 58-features model
      model_result <- train_adversarial_model(
        X_58_np, y_58_np, archetype_labels,
        lambda_val, archetype, "pca_58"
      )

      results_58[[archetype]][[sprintf("lambda_%.4f", lambda_val)]] <- model_result
      cat("Done\n")
    }
  }

  # Save 58-features model results
  save_weighted_results(results_58, "pca_58")

} else if (is.null(results_58)) {
  cat("Warning: Ja_pca_debiased column not found in risk_scores_comparison.csv\n")
  cat("Skipping 58-features model adversarial training\n")
}

# === FINAL COMPARISON SUMMARY ===
cat("\n\n=== FINAL COMPARISON SUMMARY ===\n")
cat("Adversarial training status for all models:\n\n")

# Check original model status
if (file.exists("results/adversarial_156_features_original_model_results.csv")) {
  cat("1. Original model (156 features) - RESULTS EXIST\n")
} else if (length(results) > 0) {
  cat("1. Original model (156 features) - JUST COMPLETED\n")
} else {
  cat("1. Original model (156 features) - NOT AVAILABLE\n")
}

# Check weighted model status
if (file.exists("results/adversarial_156_features_weighted_model_results.csv")) {
  cat("2. Weighted model (156 features) - RESULTS EXIST\n")
} else if (exists("results_weighted") && length(results_weighted) > 0) {
  cat("2. Weighted model (156 features) - JUST COMPLETED\n")
} else {
  cat("2. Weighted model (156 features) - NOT AVAILABLE\n")
}

# Check 58-features model status
if (file.exists("results/adversarial_156_features_pca_58_model_results.csv")) {
  cat("3. 58-features model (156 features) - RESULTS EXIST\n")
} else if (exists("results_58") && length(results_58) > 0) {
  cat("3. 58-features model (156 features) - JUST COMPLETED\n")
```

```
} else {
  cat("3. 58-features model (156 features) - NOT AVAILABLE\n")
}

cat("\nADVERSARIAL TRAINING COMPLETE\n")
cat("All models have been analyzed for adversarial bias reduction.\n")
cat("Check the results/ directory for detailed CSV files.\n")
cat("Lower adversarial accuracy indicates better bias reduction.\n")
```

C.9 Adverserial Loss Comparison

```
    # Packages
pkgs <- c("readr","dplyr","ggplot2","scales","stringr","ggrepel","purrr","tibble")
to_install <- pkgs[!pkgs %in% rownames(installed.packages())]
if(length(to_install)) install.packages(to_install)
lapply(pkgs, library, character.only = TRUE)

# Load data
adv_orig  <- read_csv("results/adversarial_156_features_original_model_results.csv", show
adv_train <- read_csv("results/adversarial_training_results.csv",                show_col_t


# Summarize per lambda
# Fairness: use mean absolute excess_share across archetypes
fair_by_lambda <- adv_orig %>%
  group_by(lambda) %>%
  summarise(
    mean_abs_excess = mean(abs(excess_share), na.rm = TRUE),
    # If available: mean "bias_reduction" (negative = reduction)
    mean_bias_reduction = mean(bias_reduction, na.rm = TRUE),
    n_archetypes = n_distinct(archetype),
    .groups = "drop"
  )

# Performance + adversary information per lambda
perf_by_lambda <- adv_train %>%
  group_by(lambda) %>%
  summarise(
    val_loss = mean(val_loss, na.rm = TRUE),
    adv_accuracy = mean(adv_accuracy, na.rm = TRUE),
    n_rows_mean = mean(n_rows, na.rm = TRUE),
    .groups = "drop"
  ) %>%
  # Create "distance to random" for adversary
  # For binary adversary, "random" ~ 0.5. Lower = better (less predictable bias)
  mutate(adv_gap = abs(adv_accuracy - 0.5))

# Merge
sum_by_lambda <- fair_by_lambda %>%
  inner_join(perf_by_lambda, by = "lambda") %>%
  arrange(lambda)

# Normalize + combined score
# We want: lower val_loss better, lower mean_abs_excess better, lower adv_gap better
# Normalize to [0,1] so 0 is best per axis
norm01 <- function(x) if(all(is.na(x))) x else rescale(x, to = c(0,1))
```

```
sum_by_lambda <- sum_by_lambda %>%
  mutate(
    z_val_loss     = norm01(val_loss),             # lower better
z_abs_excess   = norm01(mean_abs_excess),      # lower better
z_adv_gap      = norm01(adv_gap),              # lower better
    # Combine fairness in one term (equal weighting abs_excess and adversary gap)
    z_fairness     = 0.5 * (z_abs_excess + z_adv_gap),
    # Final scores (adjust weights if you want performance heavier/lighter)
    score_equal    = z_val_loss + z_fairness,      # 50/50 performance-fairness
    score_fair_heavy = 0.4 * z_val_loss + 0.6 * z_fairness,
    score_perf_heavy = 0.6 * z_val_loss + 0.4 * z_fairness
  )

# Determine Pareto frontier
# A point is Pareto-efficient if no other point has both (val_loss) and (fairness) better
is_pareto <- function(df, x = "mean_abs_excess", y = "val_loss") {
  # both must be minimal
  keep <- rep(TRUE, nrow(df))

  # Check if columns exist
  if (!x %in% names(df) || !y %in% names(df)) {
    warning("Columns␣", x, "␣or␣", y, "␣not␣found␣in␣dataframe")
    return(keep)
  }

  # Remove rows with NA values for comparison
  valid_rows <- !is.na(df[[x]]) & !is.na(df[[y]])

  for (i in seq_len(nrow(df))) {
    if (!keep[i] || !valid_rows[i]) next

    # Compare only with valid rows
    dominated <- valid_rows &
                 df[[x]] <= df[[x]][i] &
                 df[[y]] <= df[[y]][i] &
                 (df[[x]] < df[[x]][i] | df[[y]] < df[[y]][i])

    # if someone has <= on both axes and strictly < on at least one, then i is dominated
    if (any(dominated, na.rm = TRUE)) keep[i] <- FALSE
  }
  keep
}
sum_by_lambda <- sum_by_lambda %>%
  mutate(pareto = is_pareto(., x = "mean_abs_excess", y = "val_loss"))

pareto_curve <- sum_by_lambda %>%
  filter(pareto) %>%
  arrange(mean_abs_excess, val_loss)

# Choose "best balance"
best_equal <- sum_by_lambda %>% slice_min(order_by = score_equal, n = 1)
best_fair  <- sum_by_lambda %>% slice_min(order_by = score_fair_heavy, n = 1)
best_perf  <- sum_by_lambda %>% slice_min(order_by = score_perf_heavy, n = 1)

message("Best␣balance␣(equal␣weighting):␣lambda␣=␣", best_equal$lambda)
message("Fairness␣heavier:␣lambda␣=␣", best_fair$lambda)
```

```
message("Performance␣heavier:␣lambda␣=␣", best_perf$lambda)

# Plots
# (a) Trade-off scatter (fairness vs performance) + Pareto-frontier
p_trade <- ggplot(sum_by_lambda,
                  aes(x = mean_abs_excess, y = val_loss, label = lambda)) +
  geom_point(aes(shape = pareto), size = 3) +
  geom_label_repel(min.segment.length = 0) +
  geom_path(data = pareto_curve, aes(x = mean_abs_excess, y = val_loss),
            linewidth = 0.8, linetype = "dashed") +
  geom_point(data = best_equal, color = "red", size = 4) +
  labs(
    title = "Trade-off:␣Fairness␣vs␣Performance␣per␣  ",
    subtitle = "x␣=␣mean␣|excess_share|␣(lower␣is␣fairer),␣y␣=␣val_loss␣(lower␣is␣better)
    x = "Mean␣absolute␣excess␣share␣(%)",
    y = "Validation␣loss"
  ) +
  theme_minimal()

ggsave("tradeoff_fairness_performance.png", p_trade, width = 9, height = 6, dpi = 300)

# (b)   -profiles
p_lambda_fair <- ggplot(sum_by_lambda, aes(lambda, mean_abs_excess, group = 1)) +
  geom_line() + geom_point() +
  labs(title="Fairness␣vs␣  ", y="Mean␣|excess_share|", x="  ") +
  theme_minimal()

p_lambda_adv <- ggplot(sum_by_lambda, aes(lambda, adv_gap, group = 1)) +
  geom_line() + geom_point() +
  labs(title="Adversary␣gap␣vs␣  ␣(distance␣to␣0.5)", y="|adv_accuracy␣-␣0.5|", x="  ") +
  theme_minimal()

p_lambda_loss <- ggplot(sum_by_lambda, aes(lambda, val_loss, group = 1)) +
  geom_line() + geom_point() +
  labs(title="Validation␣loss␣vs␣  ", y="val_loss", x="  ") +
  theme_minimal()

ggsave("lambda_vs_fairness.png", p_lambda_fair, width = 8, height = 5, dpi = 300)
ggsave("lambda_vs_advgap.png",  p_lambda_adv,  width = 8, height = 5, dpi = 300)
ggsave("lambda_vs_valloss.png", p_lambda_loss, width = 8, height = 5, dpi = 300)

# (c) Overview table with scores and ranking
ranked <- sum_by_lambda %>%
  mutate(rank_equal = rank(score_equal, ties.method = "first")) %>%
  arrange(score_equal) %>%
  select(lambda, mean_abs_excess, adv_accuracy, adv_gap, val_loss,
         z_abs_excess, z_adv_gap, z_val_loss,
         z_fairness, score_equal, score_fair_heavy, score_perf_heavy,
         pareto, rank_equal)

print(ranked, n = nrow(ranked))
write.csv(ranked, "lambda_tradeoff_ranking.csv", row.names = FALSE)

# Archetype-specific analyses
# Use existing adv_orig data, but ensure lambda is numeric
adv_orig_archetype <- adv_orig %>%
```

```
  mutate(lambda = as.numeric(lambda))

# Plot excess_share per archetype
p_excess <- ggplot(adv_orig_archetype, aes(x = lambda, y = excess_share, group = archetyp
  geom_line() + geom_point() +
  geom_hline(yintercept = 0, linetype = "dashed", color = "black") +
  labs(title = "Excess␣share␣vs␣ ␣per␣archetype",
       x = "  ␣(adversarial␣regularisation␣strength)",
       y = "Excess␣share␣(%␣points)") +
  theme_minimal() +
  theme(legend.position = "none")   # remove legend if too many archetypes

ggsave("excess_share_per_archetype.png", p_excess, width = 10, height = 6, dpi = 300)

# Plot bias_reduction per archetype
p_bias <- ggplot(adv_orig_archetype, aes(x = lambda, y = bias_reduction, group = archetyp
  geom_line() + geom_point() +
  geom_hline(yintercept = 0, linetype = "dashed", color = "black") +
  labs(title = "Bias␣reduction␣vs␣ ␣per␣archetype",
       x = "  ␣(adversarial␣regularisation␣strength)",
       y = "Bias␣reduction␣(  ␣excess␣share)") +
  theme_minimal() +
  theme(legend.position = "none")

ggsave("bias_reduction_per_archetype.png", p_bias, width = 10, height = 6, dpi = 300)

# Best    per archetype (closest to 0 excess_share)
summary_archetype <- adv_orig_archetype %>%
  group_by(archetype) %>%
  slice_min(order_by = abs(excess_share), n = 1, with_ties = FALSE) %>%
  select(archetype, best_lambda = lambda, best_excess_share = excess_share, bias_reductio

# Save table
readr::write_csv(summary_archetype, "best_lambda_per_archetype.csv")

print(summary_archetype, n = 20)

# Link val_loss to archetype data
# adv_orig_archetype already has val_loss, so we don't need to join with adv_train
# Just use the existing data
adv_merged <- adv_orig_archetype

# Make trade-off plot per archetype
p_tradeoff <- ggplot(adv_merged,
                     aes(x = abs(excess_share), y = val_loss,
                         color = factor(lambda), group = archetype)) +
  geom_point(size = 3) +
  geom_line(alpha = 0.7) +
  facet_wrap(~ archetype, scales = "free_y") +
  scale_y_log10() +  # log scale so the huge differences in val_loss are visible
  labs(title = "Fairness Performance␣Trade-off␣per␣Archetype",
       subtitle = "x␣=␣|excess␣share|␣(lower␣=␣fairer),␣y␣=␣validation␣loss␣(lower␣=␣bett
       x = "Absolute␣excess␣share␣(%)",
       y = "Validation␣loss␣(log␣scale)",
       color = "  ") +
  theme_minimal()
```

```
ggsave("tradeoff_per_archetype.png", p_tradeoff, width = 12, height = 8, dpi = 300)

# Best    per archetype (closest to ideal point)
adv_best <- adv_merged %>%
  group_by(archetype) %>%
  mutate(dist_to_ideal = sqrt((excess_share)^2 + (log10(val_loss))^2)) %>%
  slice_min(order_by = dist_to_ideal, n = 1, with_ties = FALSE) %>%
  select(archetype, best_lambda = lambda, excess_share, val_loss, dist_to_ideal)

# Save the new best lambda results
write_csv(adv_best, "best_lambda_per_archetype_with_distance.csv")
print(adv_best, n = 20)

# Choose best consensus lambda and train model
message("===␣CHOOSING␣BEST␣CONSENSUS␣LAMBDA␣===")

# Analyze bias reduction per lambda
bias_reduction_summary <- adv_orig_archetype %>%
  group_by(lambda) %>%
  summarise(
    mean_bias_reduction = mean(bias_reduction, na.rm = TRUE),
    median_bias_reduction = median(bias_reduction, na.rm = TRUE),
    n_archetypes_improved = sum(bias_reduction < 0, na.rm = TRUE),  # negative = reductio
    total_archetypes = n(),
    pct_improved = n_archetypes_improved / total_archetypes * 100,
    .groups = "drop"
  ) %>%
  arrange(mean_bias_reduction)  # sort by best bias reduction

print("Bias␣reduction␣per␣lambda:")
print(bias_reduction_summary)

# Choose best lambda with preference for bias reduction
# Criteria:
# 1. Maximum bias reduction (lowest mean_bias_reduction)
# 2. Highest percentage of archetypes that improve
# 3. Acceptable performance (val_loss not too high)

# Combine with performance data
lambda_selection <- bias_reduction_summary %>%
  left_join(sum_by_lambda %>% select(lambda, val_loss, mean_abs_excess), by = "lambda") %
  # Normalize performance (lower = better)
  mutate(
    norm_val_loss = (val_loss - min(val_loss, na.rm = TRUE)) / (max(val_loss, na.rm = TRU
    # Score: bias reduction (70%) + performance (30%)
    selection_score = 0.7 * (-mean_bias_reduction) + 0.3 * (1 - norm_val_loss)
  ) %>%
  arrange(desc(selection_score))

print("Lambda␣selection␣scores:")
print(lambda_selection %>% select(lambda, mean_bias_reduction, pct_improved, val_loss, se

# Choose best lambda
best_consensus_lambda <- lambda_selection$lambda[1]
```

```
message("Best consensus lambda: ", best_consensus_lambda)
message("Bias reduction: ", round(lambda_selection$mean_bias_reduction[1], 4))
message("Percentage archetypes improved: ", round(lambda_selection$pct_improved[1], 1), '
message("Validation loss: ", round(lambda_selection$val_loss[1], 2))

# Save best lambda for model training
message("=== SAVING BEST LAMBDA FOR MODEL TRAINING ===")

# Save best lambda and results
best_lambda_results <- list(
  lambda = best_consensus_lambda,
  bias_reduction = lambda_selection$mean_bias_reduction[1],
  pct_improved = lambda_selection$pct_improved[1],
  val_loss = lambda_selection$val_loss[1],
  selection_score = lambda_selection$selection_score[1],
  timestamp = Sys.time()
)

# Save as RDS file
saveRDS(best_lambda_results, "best_adversarial_lambda.rds")

# Also save as CSV for easy access
best_lambda_df <- data.frame(
  lambda = best_consensus_lambda,
  bias_reduction = lambda_selection$mean_bias_reduction[1],
  pct_improved = lambda_selection$pct_improved[1],
  val_loss = lambda_selection$val_loss[1],
  selection_score = lambda_selection$selection_score[1]
)
write_csv(best_lambda_df, "best_adversarial_lambda.csv")

message("=== END OF ANALYSIS ===")
message("Best lambda saved: ", best_consensus_lambda)
message("Files created:")
message("   - best_adversarial_lambda.rds (for R)")
message("   - best_adversarial_lambda.csv (for other tools)")
message("")
message("NEXT STEP:")
message("Use this lambda in your model training script:")
message("   best_lambda <- readRDS('best_adversarial_lambda.rds')$lambda")
message("   # or")
message("   best_lambda <- read_csv('best_adversarial_lambda.csv')$lambda[1]")

# Analyze all models (Original, Weighted, PCA-58)
message("\n=== ANALYZING ALL MODELS ===")

# Check if all model results exist
orig_file <- "results/adversarial_156_features_original_model_results.csv"
weighted_file <- "results/adversarial_156_features_weighted_model_results.csv"
pca58_file <- "results/adversarial_156_features_pca_58_model_results.csv"

if (file.exists(orig_file) && file.exists(weighted_file) && file.exists(pca58_file)) {
  message("All model results found. Starting comparison...")

  # Load all models
  adv_orig_all  <- read_csv(orig_file, show_col_types = FALSE)
```

```
  adv_weighted_all <- read_csv(weighted_file, show_col_types = FALSE)
  adv_pca58_all <- read_csv(pca58_file, show_col_types = FALSE)

  # Add model_type
  adv_orig_all$model_type <- "Original (156 features)"
  adv_weighted_all$model_type <- "Weighted (156 features)"
  adv_pca58_all$model_type <- "PCA-58 (156 features)"

  # Combine all models
  all_models <- bind_rows(adv_orig_all, adv_weighted_all, adv_pca58_all)

  message("Data loaded:")
message("   - Original model: ", nrow(adv_orig_all), " rows")
message("   - Weighted model: ", nrow(adv_weighted_all), " rows")
message("   - PCA-58 model: ", nrow(adv_pca58_all), " rows")
message("   - Total: ", nrow(all_models), " rows")

  # Summarize per lambda and per model
  fair_by_lambda_model <- all_models %>%
    group_by(lambda, model_type) %>%
    summarise(
      mean_abs_excess = mean(abs(excess_share), na.rm = TRUE),
      mean_bias_reduction = mean(bias_reduction, na.rm = TRUE),
      n_archetypes = n_distinct(archetype),
      mean_adv_accuracy = mean(adv_accuracy, na.rm = TRUE),
      mean_val_loss = mean(val_loss, na.rm = TRUE),
      .groups = "drop"
    ) %>%
    mutate(adv_gap = abs(mean_adv_accuracy - 0.5))

  # Normalize per model type
  fair_by_lambda_model <- fair_by_lambda_model %>%
    group_by(model_type) %>%
    mutate(
      z_val_loss     = norm01(mean_val_loss),
      z_abs_excess   = norm01(mean_abs_excess),
      z_adv_gap      = norm01(adv_gap),
      z_fairness     = 0.5 * (z_abs_excess + z_adv_gap),
      score_equal    = z_val_loss + z_fairness,
      score_fair_heavy = 0.4 * z_val_loss + 0.6 * z_fairness,
      score_perf_heavy = 0.6 * z_val_loss + 0.4 * z_fairness
    ) %>%
    ungroup()

  # Remove rows with NA values for Pareto analysis
  fair_by_lambda_model_clean <- fair_by_lambda_model %>%
    filter(!is.na(mean_abs_excess) & !is.na(mean_val_loss))

  # Pareto-frontier per model
  fair_by_lambda_model_clean <- fair_by_lambda_model_clean %>%
    group_by(model_type) %>%
    mutate(pareto = is_pareto(., x = "mean_abs_excess", y = "mean_val_loss")) %>%
    ungroup()

  # Add pareto column to original data (NA for filtered rows)
  fair_by_lambda_model <- fair_by_lambda_model %>%
```

```
    left_join(fair_by_lambda_model_clean %>% select(lambda, model_type, pareto),
              by = c("lambda", "model_type"))

  # Best balance per model
  best_per_model <- fair_by_lambda_model %>%
    group_by(model_type) %>%
    summarise(
      best_equal = lambda[which.min(score_equal[!is.na(score_equal) & !is.infinite(score_
      best_fair = lambda[which.min(score_fair_heavy[!is.na(score_fair_heavy) & !is.infini
      best_perf = lambda[which.min(score_perf_heavy[!is.na(score_perf_heavy) & !is.infini
      .groups = "drop"
    )

  message("\n=== BEST LAMBDA PER MODEL ===")
  for (i in 1:nrow(best_per_model)) {
    model <- best_per_model$model_type[i]
    message(sprintf("\n%s:", model))
message(sprintf("  Best balance (equal weighting): lambda = %.4f", best_per_model$best_eq
message(sprintf("  Fairness heavier: lambda = %.4f", best_per_model$best_fair[i]))
message(sprintf("  Performance heavier: lambda = %.4f", best_per_model$best_perf[i]))
  }

  # Plots for all models
# Trade-off scatter for all models
  p_trade_all <- ggplot(fair_by_lambda_model,
                        aes(x = mean_abs_excess, y = mean_val_loss,
                            color = model_type, shape = pareto)) +
    geom_point(size = 3) +
    geom_label_repel(aes(label = lambda), min.segment.length = 0, size = 3) +
    geom_path(data = filter(fair_by_lambda_model_clean, pareto),
              aes(x = mean_abs_excess, y = mean_val_loss, color = model_type),
              linewidth = 0.8, linetype = "dashed") +
    geom_point(data = fair_by_lambda_model %>%
                 group_by(model_type) %>%
                 slice_min(order_by = score_equal, n = 1) %>%
                 ungroup(),
               size = 5, color = "red") +
    labs(
      title = "Trade-off: Fairness vs Performance per    voor alle modellen",
      subtitle = "x = mean |excess_share| (lager is eerlijker), y = val_loss (lager is be
      x = "Gemiddelde absolute excess share (%)",
      y = "Validation loss",
      color = "Model Type",
      shape = "Pareto Efficient"
    ) +
    theme_minimal() +
    theme(legend.position = "bottom")

  ggsave("results/tradeoff_fairness_performance_all_models.png", p_trade_all, width = 12,

  #   -profiles per model
  p_lambda_fair_all <- ggplot(fair_by_lambda_model,
                              aes(lambda, mean_abs_excess, color = model_type, group = mo
    geom_line() + geom_point() +
    labs(title="Fairness vs    per model", y="Mean |excess_share|", x="  ", color="Model
    theme_minimal()
```

```
p_lambda_adv_all <- ggplot(fair_by_lambda_model,
                          aes(lambda, adv_gap, color = model_type, group = model_type)
  geom_line() + geom_point() +
  labs(title="Adversary␣gap␣vs␣  ␣per␣model", y="|adv_accuracy␣-␣0.5|", x="  ", color="
  theme_minimal()

p_lambda_loss_all <- ggplot(fair_by_lambda_model,
                           aes(lambda, mean_val_loss, color = model_type, group = mode
  geom_line() + geom_point() +
  labs(title="Validation␣loss␣vs␣  ␣per␣model", y="val_loss", x="  ", color="Model␣Type
  theme_minimal()

ggsave("results/lambda_vs_fairness_all_models.png", p_lambda_fair_all, width = 10, heig
ggsave("results/lambda_vs_advgap_all_models.png", p_lambda_adv_all, width = 10, height
ggsave("results/lambda_vs_valloss_all_models.png", p_lambda_loss_all, width = 10, heigh

# Overview table with scores and ranking per model
ranked_all <- fair_by_lambda_model %>%
  group_by(model_type) %>%
  mutate(rank_equal = rank(score_equal, ties.method = "first")) %>%
  arrange(model_type, score_equal) %>%
  select(model_type, lambda, mean_abs_excess, mean_adv_accuracy, adv_gap, mean_val_loss
         z_abs_excess, z_adv_gap, z_val_loss,
         z_fairness, score_equal, score_fair_heavy, score_perf_heavy,
         pareto, rank_equal)

write.csv(ranked_all, "results/lambda_tradeoff_ranking_all_models.csv", row.names = FAL

# Archetype-specific analyses for all models
all_models_archetype <- all_models %>%
  mutate(lambda = as.numeric(lambda))

# Plot excess_share per archetype per model
p_excess_all <- ggplot(all_models_archetype,
                       aes(x = lambda, y = excess_share, group = archetype, color = arc
  geom_line() + geom_point() +
  geom_hline(yintercept = 0, linetype = "dashed", color = "black") +
  facet_wrap(~ model_type, scales = "free_x") +
  labs(title = "Excess␣share␣vs␣  ␣per␣archetype␣en␣model",
       x = "  ␣(adversarial␣regularisation␣strength)",
       y = "Excess␣share␣(%␣points)") +
  theme_minimal() +
  theme(legend.position = "bottom")

ggsave("results/excess_share_per_archetype_all_models.png", p_excess_all, width = 15, h

# Plot bias_reduction per archetype per model
p_bias_all <- ggplot(all_models_archetype,
                     aes(x = lambda, y = bias_reduction, group = archetype, color = arc
  geom_line() + geom_point() +
  geom_hline(yintercept = 0, linetype = "dashed", color = "black") +
  facet_wrap(~ model_type, scales = "free_x") +
  labs(title = "Bias␣reduction␣vs␣  ␣per␣archetype␣en␣model",
       x = "  ␣(adversarial␣regularisation␣strength)",
       y = "Bias␣reduction␣(  ␣excess␣share)") +
```

```
    theme_minimal() +
    theme(legend.position = "bottom")

  ggsave("results/bias_reduction_per_archetype_all_models.png", p_bias_all, width = 15, h

  # Best    per archetype per model
  summary_archetype_all <- all_models_archetype %>%
    group_by(archetype, model_type) %>%
    slice_min(order_by = abs(excess_share), n = 1, with_ties = FALSE) %>%
    select(archetype, model_type, best_lambda = lambda, best_excess_share = excess_share,

  write_csv(summary_archetype_all, "results/best_lambda_per_archetype_all_models.csv")

  message("\n=== BEST LAMBDA PER ARCHETYPE PER MODEL ===")
  print(summary_archetype_all, n = 30)

  # Model comparison summary
  message("\n=== MODEL COMPARISON SUMMARY ===")

  # Compare best lambda per model
  model_comparison <- fair_by_lambda_model %>%
    filter(!is.na(score_equal) & !is.infinite(score_equal)) %>%
    group_by(model_type) %>%
    slice_min(order_by = score_equal, n = 1) %>%
    select(model_type, best_lambda = lambda,
           mean_abs_excess, mean_val_loss, adv_gap, score_equal)

  message("\nBest lambda per model (equal weighting performance/fairness):")
  print(model_comparison)

  # Which model has the best overall score?
# First filter rows with valid scores
  model_comparison_valid <- model_comparison %>%
    filter(!is.na(score_equal) & !is.infinite(score_equal))

  if (nrow(model_comparison_valid) > 0) {
    best_overall_model <- model_comparison_valid %>%
      slice_min(order_by = score_equal, n = 1)

    message(sprintf("\nBEST OVERALL MODEL: %s (lambda = %.4f)",
                best_overall_model$model_type, best_overall_model$best_lambda))
  } else {
    message("\nNo valid scores found for model comparison")
message("This may be due to NA values in the data.")
  }

  message("\nAll models analyzed!")
message("New files created in results/ directory:")
message("   - tradeoff_fairness_performance_all_models.png")
message("   - lambda_vs_fairness_all_models.png")
message("   - lambda_vs_advgap_all_models.png")
message("   - lambda_vs_valloss_all_models.png")
message("   - excess_share_per_archetype_all_models.png")
message("   - bias_reduction_per_archetype_all_models.png")
message("   - lambda_tradeoff_ranking_all_models.csv")
message("   - best_lambda_per_archetype_all_models.csv")
```

```
} else {
  message("Not␣all␣model␣results␣found.")
message("␣␣␣Available␣files:")
message("␣␣␣-␣Original:␣", if(file.exists(orig_file)) "Yes" else "No")
message("␣␣␣-␣Weighted:␣", if(file.exists(weighted_file)) "Yes" else "No")
message("␣␣␣-␣PCA-58:␣", if(file.exists(pca58_file)) "Yes" else "No")
message("␣␣␣Run␣adversarial_156_features.R␣first␣to␣train␣all␣models.")
}
```

References

Abdi, H. and Williams, L. J. (2010). Principal component analysis. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(4):433–459.

Barocas, S., Hardt, M., and Narayanan, A. (2019). *Fairness and Machine Learning*. fairmlbook.org.

Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002a). Smote: Synthetic minority oversampling technique. *Journal of Artificial Intelligence Research*, 16:321–357.

Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002b). Smote: Synthetic minority oversampling technique. *Journal of Artificial Intelligence Research*, 16:321–357.

Chouldechova, A. and Roth, A. (2018). The frontiers of fairness in machine learning. *arXiv preprint arXiv:1810.08810*.

De Maesschalck, R., Jouan-Rimbaud, D., and Massart, D. L. (2000). The mahalanobis distance. *Chemometrics and Intelligent Laboratory Systems*, 50(1):1–18.

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29(5):1189–1232.

Friedrich, M., van Eijk, N., and Zuiderveen Borgesius, F. J. (2022). Ethics and fairness in algorithmic decision-making: A case study on fraud detection in welfare services. *Computer Law & Security Review*, 45:105657.

Gnanadesikan, R. and Kettenring, J. R. (1972). Robust estimates, residuals, and outlier detection with multiresponse data. *Biometrics*, 28(1):81–124.

Guyon, I. and Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182.

Hand, D. J. (2012). Assessing the performance of classification methods. *International Statistical Review*, 80(3):400–414.

Hardt, M., Price, E., and Srebro, N. (2016). Equality of opportunity in supervised learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 29, pages 3315–3323.

James, G., Witten, D., Hastie, T., and Tibshirani, R. (2023). *An Introduction to Statistical Learning with Applications in R*. Springer, 3rd edition.

Jolliffe, I. T. (2002). *Principal Component Analysis*. Springer, New York, 2 edition.

Lighthouse Reports (2021). The suspicion machine: How welfare surveillance in the netherlands discriminates against the poor.

Little, R. J. A. and Rubin, D. B. (2019). *Statistical Analysis with Missing Data*. Wiley, Hoboken, NJ, 3 edition.

Murtagh, F. and Contreras, P. (2012). Algorithms for hierarchical clustering: an overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(1):86–97.

Rijksoverheid (2023a). Beleidsdocument inzake bijstandsuitkering. `https://open.overheid.nl/documenten/ronl-c409ea31-2c00-4318-9a45-d47ad8a2ca7f/pdf`. Accessed: 2025-08-17.

Rijksoverheid (2023b). Wat is de taaleis in de bijstand? `https://www.rijksoverheid.nl/onderwerpen/bijstand/vraag-en-antwoord/wat-is-de-taaleis-in-de-bijstand`. Accessed: 2025-08-17.

Saito, T. and Rehmsmeier, M. (2015). The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets. *PLOS ONE*, 10(3):e0118432.

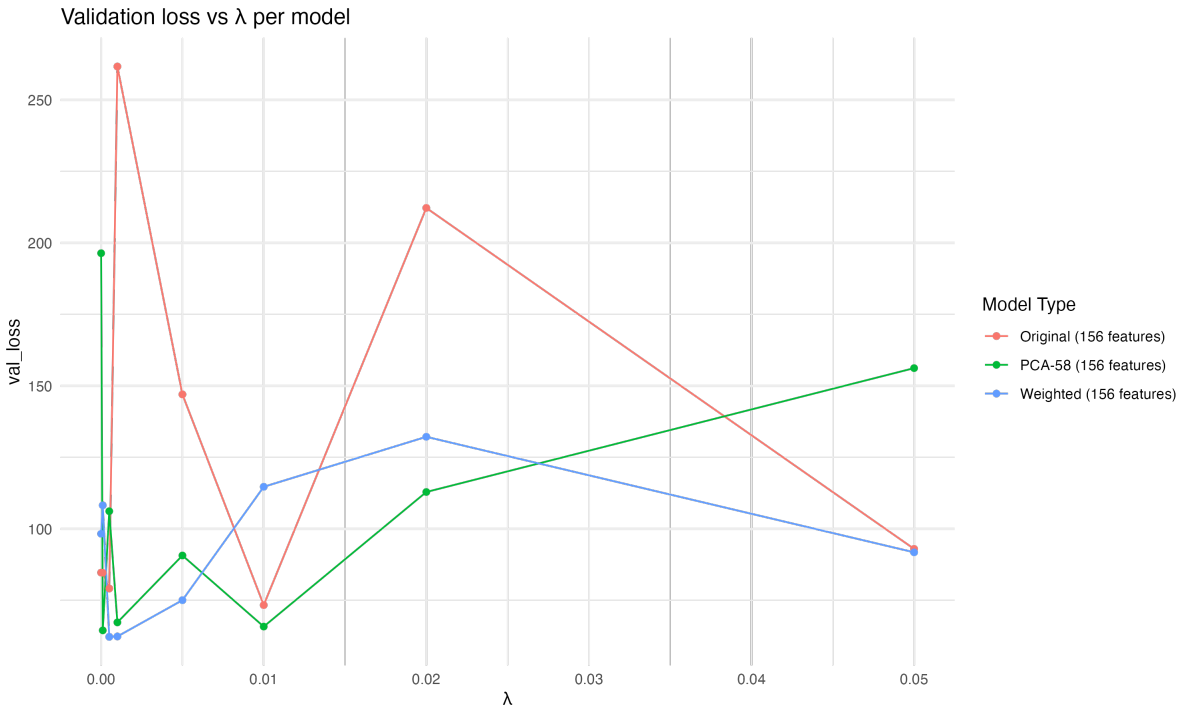
Siddappa, N. and Kampalappa, T. (2020). Imbalance data classification using local mahalanobis distance learning based on nearest neighbor. *SN Computer Science*, 1.

Statistics Netherlands (CBS) (2020). Statline: Bevolking; kerncijfers. `https://opendata.cbs.nl/statline/`. Accessed: 2025-08-17.

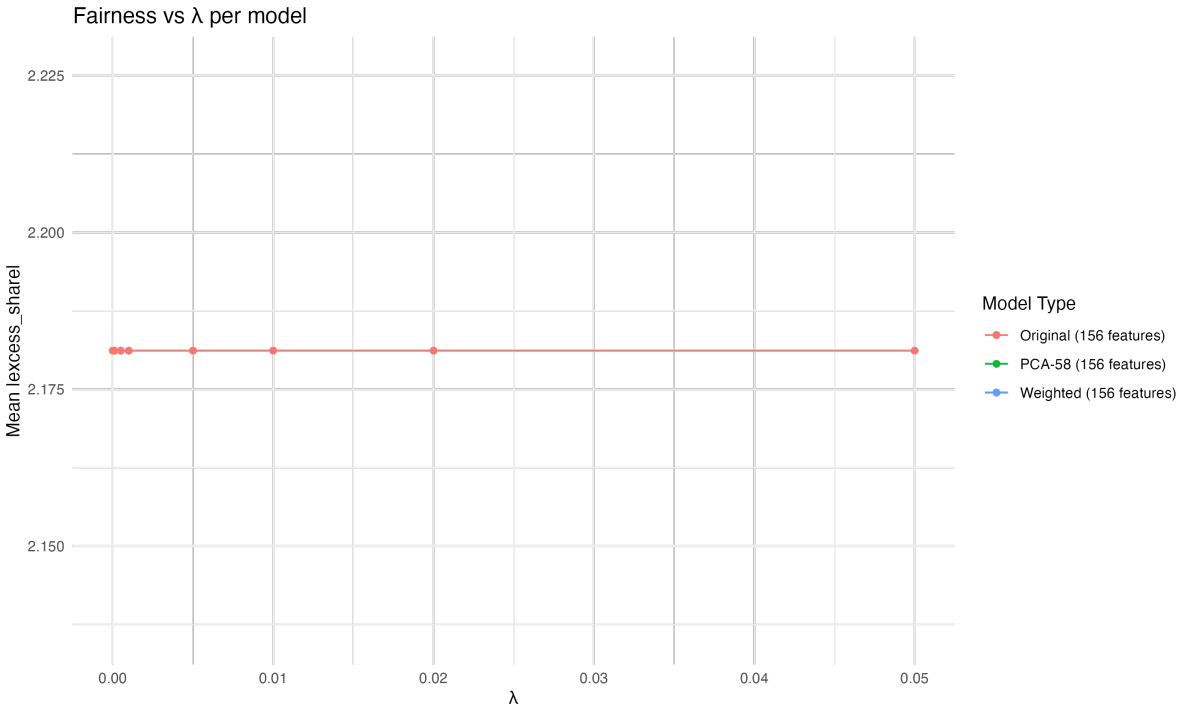
Wasserstein, R. L. and Lazar, N. A. (2016). The asa's statement on p-values: context, process, and purpose. *The American Statistician*, 70(2):129–133.

Zhang, B. H., Lemoine, B., and Mitchell, M. (2018). Mitigating unwanted biases with adversarial learning. In *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*, pages 335–340. ACM.

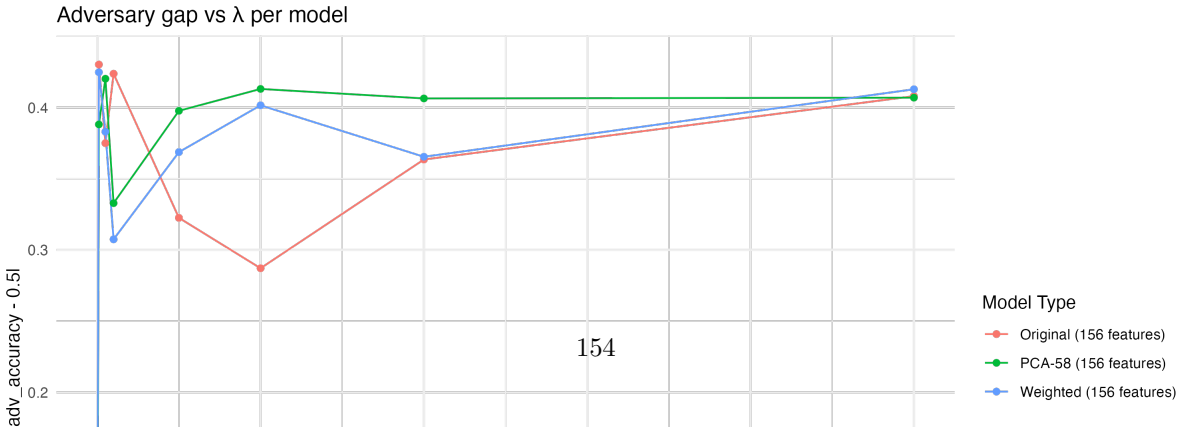
Zhang, C. and Ma, Y. (2014). *Ensemble Machine Learning: Methods and Applications*. Springer.



(a) Validation loss across different values of λ.



(b) Mean absolute excess share across different values of λ.



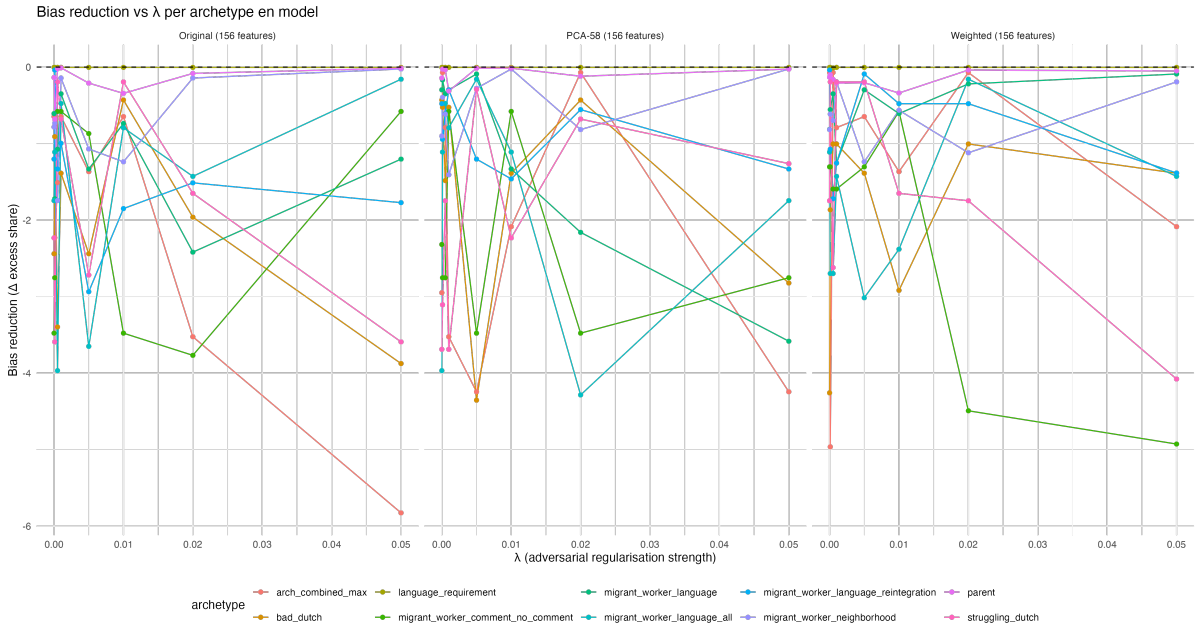


Figure 29: Bias reduction of independent archetypes due to adverserial loss across the three different models.