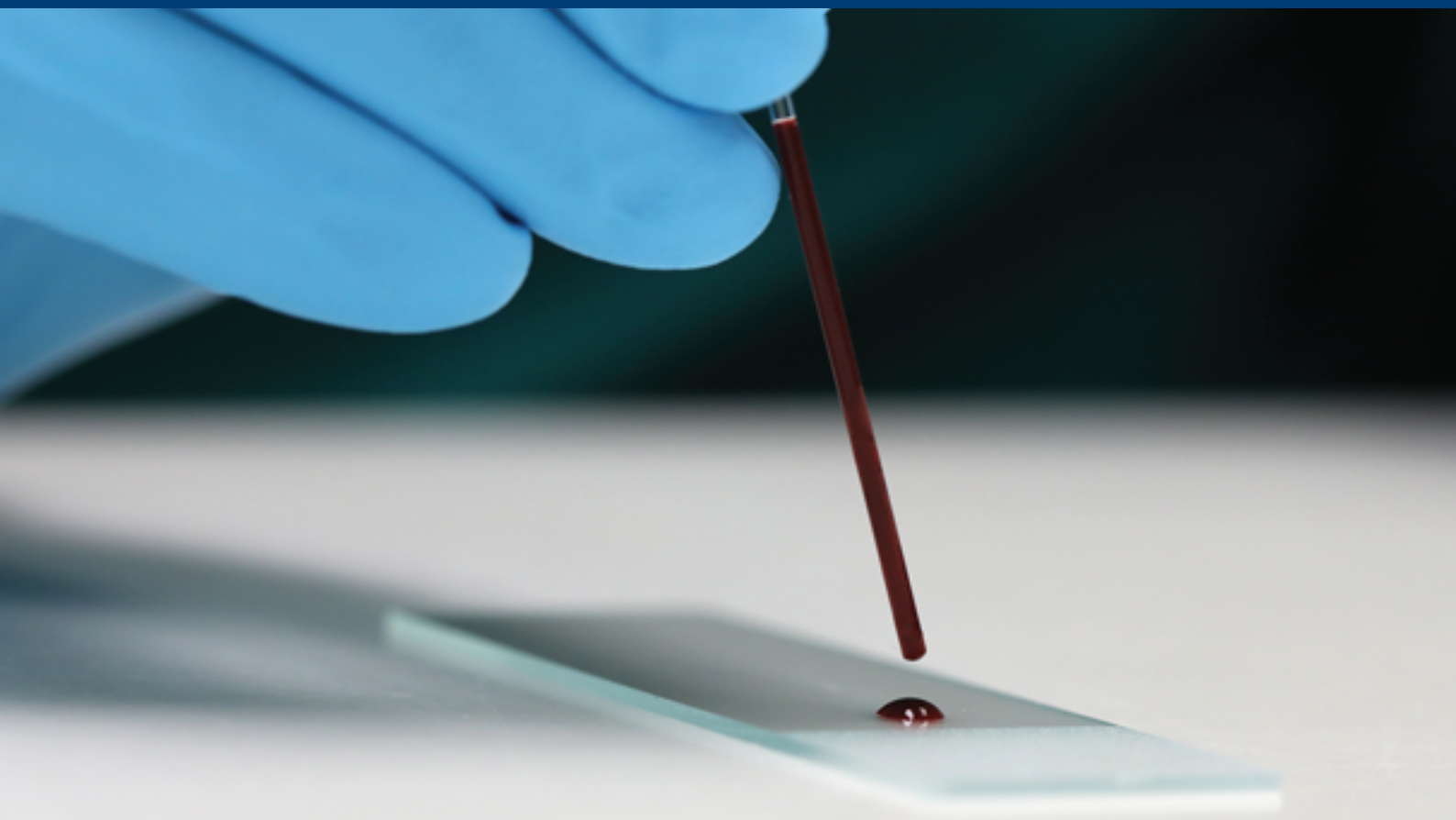
Dating blood traces using Bayesian Networks

Si-Jing Chen

Bachelor Thesis
Applied Mathematics

TU Delft



TU Delft

Dating blood traces using Bayesian Networks

by

Si-Jing Chen

to obtain the degree of Bachelor of Science
at the Delft University of Technology,
to be defended publicly on Friday July 14, 2023 at 10:30 AM.

Student number:	5124840	
Project duration:	April 24, 2023 – July 14, 2023	
Thesis committee:	Prof. dr. ir. G.F. Nane,	TU Delft, supervisor
	Mr. ir. drs. V.N.S.R Dwarka,	TU Delft

This thesis is confidential and cannot be made public until July 14, 2023.

TU Delft

Preface

This Bachelor Thesis has given me the opportunity to truly see what it is like to be an Applied Mathematician. Being able to work on a project with such a real-life significance and getting my hands on such raw data, it has been an eye-opening experience. Especially as it has has also given me the chance to combine both my skills in programming and mathematics which I have gained throughout my Applied Mathematics Bachelor and Computer Science Minor at TU Delft.

I would like to sincerely thank Prof. Dr. ir. Tina Nane for all of her support and guidance throughout the project. Regardless of her busy schedule, she has still been able to provide me with so much of her time and for that I am truly thankful.

I would also like to thank Dr. Federica Galli and her team for giving me the opportunity to take part in their research on the dating of blood traces, and their frequent correspondence.

Si-Jing Chen
Delft, July 2023

Abstract

At crime scenes, various methods can be used to determine how much time has passed since the act happened, for example a body left at the scene, camera footage or eye witnesses. In this thesis, however, a different method shall be used to determine the time of the crime. The time will be determined using blood evidence.

To be exact, the goal of this thesis is to analyse the aging of bloodstains through colour analysis, and constructing a Bayesian network (BN) which can accurately make predictions on the time of deposition of a bloodstain.

The data used for the construction of the BNs was obtained from images provided by the Leiden Institute of Physics (LION). To obtain the data, the bloodstains in the images first need to be isolated from the background. Afterwards, the bloodstains are split into two parts: the inner part of the bloodstain and the complete bloodstain. Both the isolation and the splitting are done using a method called masking. Afterwards they can be converted to RGB (red, green, blue) values and using these RGB values, the following data can be collected for each colour channel of both the inner part of the bloodstain and the whole bloodstain: mean, min, max, variance, and the 5%, 20%, 50%, 80% and 95% quantiles.

Using various subsets of the data, BNs can be constructed. The structure of the BN is determined using a structure-learning algorithm called hill-climbing . To determine the validity of a given structure k-fold cross-validation can be performed using a given loss function. In this thesis, k=5 has been used, and the Mean Squared Error (MSE) has been taken as the loss function. Upon comparison of the MSE, it seems that the best model is given by the red values of a bloodstain. However, even the best performing model found in this thesis still has a considerably poor performance as the BN for the red variables has an MSE of 15149.81.

Contents

1 Introduction

Time is of the essence in crime scene investigations. Be it homicide or kidnapping, knowing how much time has passed since the incident can provide investigators with crucial information, which can contribute to an arrest or even a conviction. There are many methods to determine the time of the crime [14]. For a homicide, one of the most common methods would be using the body. However, when a body is absent, investigators will be required to analyse other pieces of evidence. One possible option being blood remains which could have been left at the site.

Research on the connection between blood and time has been attempted ever since 1907 [13], and since then countless attempts have been made to find a method to trace a bloodstain's age. However, no method has yet been found with sufficient accuracy and validation [14]. Initially, research on the time since deposition focused on the macroscopic changes of blood, i.e. the changes which are visible to the naked eye. For example the colour of the blood [13], and the solubility of the blood in water [7]. More recent research has developed to a more microscopic scale, and has been involving more complicated methods. An example of a more recent study on this topic uses digital image analysis and the statistical classification technique called Random Forests [12]. Although this study attained a rather low error rate, it is good to mention that using Random Forests does not give an exact estimate of how much time has passed. For Random Forests, an image is classified at certain time-point, for example 15 min, 30 min, 1 h, 6 h, 1 day etc. So this method does provide an estimate of the time since deposition, but the accuracy depends fully on the chosen time stamps over which is classified.

In this thesis, the focus will also be on digital image processing / visual data processing, and the usage of mathematical constructs to create a model. To be exact, the goal of this thesis is to obtain the best possible forecast to date blood traces, using Bayesian networks. Bayesian networks are probabilistic graphical models capable of modeling the dependencies between various random variables. In comparison to the aforementioned Random Forests, Bayesian networks are be able to make exact predictions of the time, rather than only classifying the time since deposition of a bloodstain. The Bayesian networks in this thesis will be constructed using colour data obtained from images which tracked the ageing of the blood traces. The images were provided by the Leiden Institute of Physics (LION) and consisted of images for 10 different bloodstains. Each image is converted to RGB values (red, green and blue), and various pieces of data shall be collected, such as the mean, minimum and maximum.

To attain the goal of accurately being able to determine a bloodstain's age, it is first important to achieve a better understanding of what currently has been researched, and the effectiveness of these methods. Both determining the time since deposition (TSD) of a bloodstain and the implementation of Bayesian networks in the forensic field have been investigated to some extent. The spectrum of implementations for Bayesian networks within forensics is rather broad. Some studies focus more on the usage of the visual aspect of the model. Whereas others make use of the model to evaluate evidence at a crime scene [4]. In Chapter 2 the advances in these fields will be further described. After having giving some background on the relevant research and implementations, Chapter 3 will elaborate on the data that will be used (RGB), the methods used to obtain this data, and an preliminary analysis

will be performed on the data. After having collected and analysed the data, a start can be made on the Bayesian networks. It is necessary to first gain a better understanding of the theory involved when constructing, analysing and using a Bayesian network. In Chapter 4, the required theory on graph theory, probability theory, and Bayesian networks themselves shall be provided. Afterwards, in Chapter 5 the collected data will be used to construct various Bayesian networks. These Bayesian networks shall be analysed to see which variables are most relevant to the dating of the blood traces, and the most optimal structure of the Bayesian network will be determined. Finally, the conclusion and discussion will be compiled in chapters 6 and 7. In the Appendices, the used Python and R code can be found, and there will also be more additional information, plots and tables.

2

Related work

This thesis focuses on dating blood traces using Bayesian networks. As mentioned in the Introduction, both determining the time since deposition (TSD) of a bloodstain, and the implementation of Bayesian networks in the forensic field are well researched topics. However, the combination of these topics is still unexplored. In this chapter, the past and current developments of dating blood traces, and some implementations of Bayesian networks in the forensic field will be discussed.

2.1. Time since Deposition of Blood

In the Introduction, some examples were given of past and more recent attempts at determining a bloodstain's age. None of these methods, however, have been successful enough to be used by both scientists and in court. Even though these previous studies have not necessarily found a method to reliably determine a bloodstain's age, they can still provide valuable information.

Going back to the earliest studies on the tracing of a bloodstain's age, it was discussed earlier that Tomellini [13] and Leers [7] focused on the colour and the solubility, respectively. As blood ages, the colour of the stain becomes darker and less vibrant, and the solubility of the blood in water decreases. This is caused by the decay of hemoglobin (Hb), Hb being one of the main components of the red blood cells in blood. Following these studies, various other scientists further expanded on these findings, and were made aware of the effects of environmental influences, like UV-exposure and temperature, on the properties of blood [14].

With time, more advanced methods were starting to be used, like atomic force microscopy (AFM) which focuses on the elasticity of the red blood cells [11], or fluorescence analysis which focuses on the changes in the fluorescence of the blood due to the protein degradation in the blood plasma.

Some more recent studies, which are closely related to the topic of this thesis, focused on a digital image analysis / colour analysis approach [12] [10]. Both have attempted to date blood traces using images captured by different smartphones, however, with greatly differing results. Thanakiatkrai et al. [12] used the M-values (magenta) and the statistical classification technique Random Forests. Using this, Thanakiatkrai et al. [12] was able to obtain an error rate of 12% for up to 42 days. As noted in the Introduction, however, this method is only able to classify the time since deposition for a bloodstain. So the prediction of the time since deposition is more limited, as it depends on the chosen time stamps for the classification (15 min, 30 min, 6 h, 1 day etc.). On the other hand, Shin et al. [10] analysed the V-values (brightness) of the bloodstains, and their method had difficulties providing accurate predictions after the 42 hours mark.

This smartphone colour analysis approach could prove to be very useful, as due to its simplicity the analysis of a bloodstain does not take long. Additionally, no professional is required to use this method, and the method is non-invasive, so it does not compromise the evidence itself. However, there are still many other factors which need to taken into consideration before this could become a viable option.

For example, environmental factors like humidity, or contamination of the bloodstain by dust or soil [14]. Thanakiatkrai et al. [12] has analysed a considerable number of environmental factors. They, for example, looked at different phone cameras, temperature, humidity, and light exposure. But also the effects when blood was placed on different surfaces and materials. However, these factors were tested for different constants, so fluctuations, e.g. the temperature fluctuations of day and night, are not taken into consideration.

It is clear that the methods have been becoming increasingly more advanced. Not only in the method used to collect the data, but also in the ways that the acquired data is evaluated and processed. More on the various techniques, used to determine the age of blood traces, and more details on the aging of blood can be found in the article by Weber and Lednev [14].

2.2. Bayesian Networks and Forensics

Bayesian networks have been employed in countless fields, for example, in medicine, artificial intelligence, agriculture and, of course, forensic science. As mentioned before the combination of blood tracing and Bayesian networks have not been investigated before, but Bayesian networks have definitely caught the eye of forensic scientists. The visual aspect of a Bayesian network is very intuitive, and can therefore be useful when making a case in court. Additionally, various hypotheses can be added to the Bayesian network in the form of probabilities and used for inference. The scope of problems that can be treated using Bayesian networks is endless, and even when singling out a single topic the questions that can be asked and answered can be very diverse. Take evidence evaluation for example, a model can be constructed to draw inferences on the source of DNA evidence found at the site, for example.

But the problem could also be taken more broadly. The question of whether evidence left at a crime scene is relevant to the case could also be modeled using a Bayesian network (Biedermann and Taroni [3]). So, for example, if fingerprints are found on a knife, was it actually used to commit a murder or was it used to cut some vegetables? Or perhaps if fingerprints are found on a balcony railing, does this indicate that someone climbed up, or was someone just leaning on the railing. These type of more general questions to fingerprint evidence left at a scene were explored by de Ronde [4] using Bayesian networks. In this dissertation, various factors such as the location and direction of the fingerprints were taken into consideration. But also things like the exerted pressure and the transfer of the fingerprint. Using Bayesian networks a high accuracy could be obtained to distinguish between writing and reading a letter (98.0%), however, the accuracy became highly volatile when more actions were performed such as the folding of the letter and would drop by 64.4%. This research definitely shows promise in the usage of Bayesian networks for forensic science, but it does seem that additional research is required.

Comparable research on Bayesian networks for the transfer of evidence from the criminal to the scene has been performed by Garbolino and Taroni [5], and shows how the likelihood ratios can be determined for some node of interest for the varying scenarios. This definitely showcases the usefulness of Bayesian networks for these types of problems, however, these examples are all left very general and have not been tested for real-life scenarios. Or at least no exact statistics on the successful usage rate has been given. Thus in this case the accuracy of the Bayesian networks in practice is not known.

3

Image Processing of Blood Traces

In this chapter, the process of acquiring and processing the data is described. All used data was collected using a climate control chamber, and was provided by the Leiden Institute of Physics (LION).

First, the setting of the experiment will be described. Some basic information will be given on the circumstances within the climate control chamber, the samples themselves, and also how images could be converted to data. More specifically, the conversion of images to the RGB colour system. Secondly, the processing of the images themselves will be further elaborated on. The irregularities within the samples will be described, and it will be shown how these irregularities are dealt with during the processing of the images. After that, an overview will be given of all data that is collected during the processing, and a description of each collected variable will be given. Finally, an exploratory analysis will be performed on the collected data. Various plots will be shown, and the observations from these plots will be further elaborated on.

3.1. Image Collection Setting

To model the ageing of bloodstains, data is required. This data was collected through an experiment performed in a climate control chamber at the Leiden Institute of Physics (LION) (see Appendix A.1 Figure A.1). Within this chamber, the temperature and humidity can be controlled. These factors were kept constant for the data used in this thesis:

temperature: $25°C$

humidity: 35%

Our data was collected from a total of 12 samples. 10 of these samples are actual blood traces deposited onto a piece of control paper. Control paper is used, as the exact colour values (RGB) for this piece of paper are known. The remaining 2 samples are control samples for which no blood has been deposited onto it. These control samples can be used to detect any irregularities, e.g. changes in the background colour or lighting. For each of these samples an image was taken every 13 / 14 minutes for approximately 20 days. Which totals up to 2178 images for each sample. (See [2] for further details on the used equipment and the image capturing.)

Each image has a resolution of 640x480, i.e. each image consists of 640 pixels per column and 480 pixels per row. The colour of each pixel can be described using various colour systems. For our data the RGB (red, green, blue) colour system was used. In this case, the colour of each pixel is determined by assigning each colour channel R, G and B a value between 0 and 255. The value of the colour represents its presence, i.e. the higher the value, the stronger the colour's presence. The combination (0, 0, 0) and (255, 255, 255) represent black (absence of all colours) and white (presence of all colours) respectively. Additionally, the data for the grayscale (black and white) images were also collected. Converting an RGB image to a grayscale image can be done using the following formula:

$$\text{GRAY} = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \tag{3.1}$$

There also exist other colour systems, e.g. the CMYK (cyan, magenta, yellow, key) and HSV (hue, saturation, value) colour systems. In this thesis, we did not make use of the CMYK and the HSV colour systems, but as mentioned in Section 2.1 the chosen colour system varies for different papers and gives varying results.

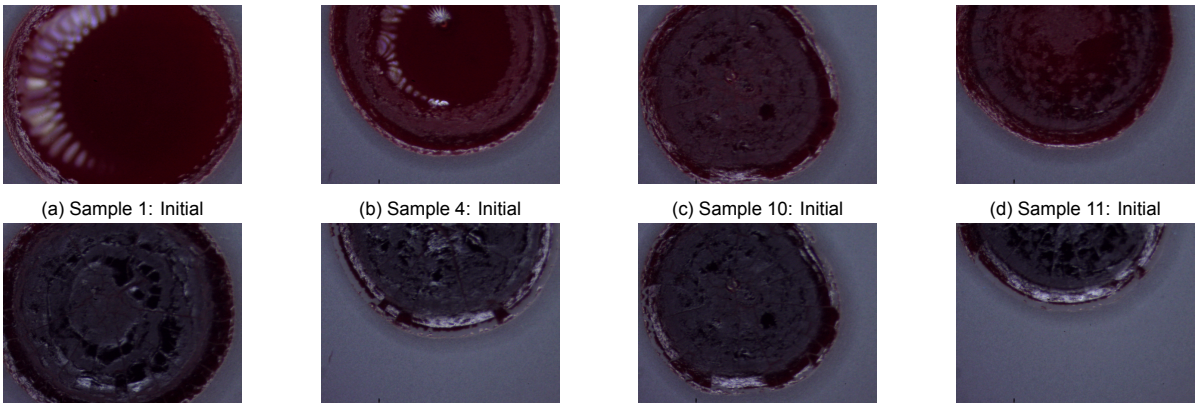
3.2. Images

In the previous section, some basic information was provided on the environmental factors, the blood samples, and the conversion of the images of these blood samples. Now the images themselves will be discussed, i.e. the images shall be inspected, and the factors which affect the consistency of the images and the samples shall be addressed. Also the processing of these images using masks will be described. An explanation will be given on what masks are, how they are constructed using thresholding, and what an image looks like when a mask is applied.

3.2.1. Images of Blood Samples

In Figure 3.1, some samples can be seen to gain an understanding of what the images from LION look like. From these images a few noteworthy matters should be addressed, i.e. the location of the bloodstain within the image and the lighting.

As can be seen when comparing the samples, it is clear that the bloodstains are varying sizes, and different areas of the bloodstain are captured within the images. Most important is the area of the bloodstain captured within the image, as the captured area is directly connected to the distribution of the colours within the image. In Figure 3.1e and 3.1h, for example, a substantial difference in captured area can be observed when comparing two different samples. This difference is caused by the centering of the bloodstain with respect to the camera. But also when we look at a single sample, significant changes to observed area can occur, e.g. in Sample 11. In this case, the shift in area is most likely caused by the drying and therefore shrinking of the bloodstain. The exact effects of the captured area on the data will become more apparent when plotting the data, and will be further analysed during the exploratory analysis (Section 3.4). This problem of non-aligned samples could be solved to some extent using methods like edge detection, but this shall be at the cost of losing data from the images. Methods to remedy this problem will not be implemented in this thesis due to the time constraint on the project, but in the Discussion some possible methods will be mentioned.



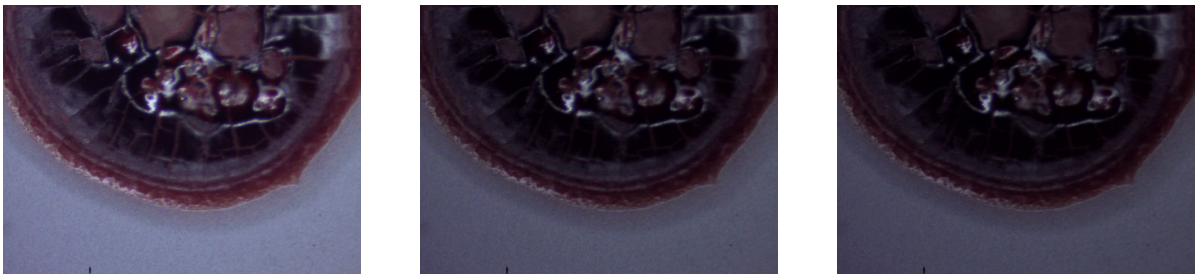
(a) Sample 1: Initial (b) Sample 4: Initial (c) Sample 10: Initial (d) Sample 11: Initial

(e) Sample 1: After 1 day (f) Sample 4: After 1 day (g) Sample 10: After 1 day (h) Sample 11: After 1 day

Figure 3.1

Another factor, which has led to some inconsistencies within the images, is the lighting. The samples are affected by the lighting in two ways. Firstly, the amount of data provided by the bloodstain can be influenced. When the bloodstain is wet (Figure 3.1a) or when certain areas turn dry (Figure 3.1h), the blood can express reflective behaviour. These areas will reflect the lighting, and lead to a white glare on top of the bloodstain. This glare can be removed using masking, but the colour at that area cannot be determined. So no information will be obtained from those areas. In Subsection 3.2.2, an explanation will be given on what masking is and how it is done. Secondly, the RGB values within an image could

be changed as a whole. Changes in the brightness of the lighting can overexpose or underexpose the image which will lead to an increase or decrease in the RGB values, respectively. Although the light equipment maintains a constant illumination, multiple outliers have still been detected. In Figure 3.2, the difference in illumination is visualised. In Section 3.4, a more in-depth analysis will be executed to determine the frequency of these outliers, and also the significance of their influence. The cause for this fluctuation in lighting is still unknown, but will be addressed in future experiments.



(a) Overexposure (b) Normal exposure (c) Underexposure

Figure 3.2: Image of bloodstains with overexposure, normal exposure and underexposure to light

3.2.2. Masks of Images

Every pixel within the image has RGB values, even the areas which are not part of the bloodstain, e.g. the background and the white glare on the bloodstain caused by the lighting (for example in Figure 3.1a). To remove the obstructive areas of the image, a mask is constructed using image thresholding.

In our case, a mask describes whether a pixel is part of a bloodstain. A mask is the same format as the image it covers, as each pixel in the mask refers to the corresponding pixel of the image. The pixels in the mask have 2 states; the pixel is in the mask, i.e. the pixel is part of the bloodstain. In this case the RGB pixel values of the original image are retained. Or the pixel is not in the mask, so it is not part of the bloodstain and the RGB pixel values in the image are all set to a predetermined value, which is (0, 0, 0) in this report. Each pixel of the mask can be described as following:

$$x_{i,j} = \begin{cases} 1, & \text{if pixel } x_{i,j} \text{ is contained within the mask} \\ 0, & \text{otherwise} \end{cases}$$

Determining whether the pixel is retained, i.e. whether it is included in the mask, is done using image thresholding. To perform image thresholding, an RGB image is first converted to a grayscale image, this can be done using Equation 3.1. Each pixel of the grayscale image will then be compared to a certain threshold, and depending on whether the pixel value is larger or smaller, the corresponding pixel will be retained by the mask or not. The threshold can be chosen as a constant T, but there are also automatic thresholding methods, i.e. thresholding methods which use algorithms to determine the optimal threshold for each (group of) pixels. One such method is called Otsu's method, and this is also the method that was used during the processing of the data (see [1] for more information on Otsu's method). In Example 1, however, a fixed threshold T shall be taken to give an idea of what thresholding looks like.

Example 1. *Let us consider an image of* $3x2$ *pixels with the following RGB and grayscale arrays.*

$$\textit{RGB} = \begin{bmatrix} (90,3,24) & (66,20,26) & (40,15,33) \\ (87,8,20) & (58,17,4) & (20,3,18) \end{bmatrix} \Longrightarrow \textit{GRAY} = \begin{bmatrix} 31.407 & 34.438 & 24.527 \\ 32.989 & 27.777 & 9.793 \end{bmatrix}$$

If a fixed threshold $T = 25$ *is taken, then the mask for the grayscale image would be as following;*

$$\begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix}$$

Using this binary mask, the masked grayscale and RGB arrays for the image can be obtained applying elementwise multiplication. So the masked image would be represented by the following arrays:

$$\textit{Masked RGB} = \begin{bmatrix} (90,3,24) & (66,20,26) & (0,0,0) \\ (87,8,20) & (58,17,4) & (0,0,0) \end{bmatrix}, \textit{Masked GRAY} = \begin{bmatrix} 31.407 & 34.438 & 0 \\ 32.989 & 27.777 & 0 \end{bmatrix}$$

After having seen an example, the idea of thresholding should be clear and now it is time to actually apply it to our images. In Figure 3.3, an image can be seen of Sample 1 (May 3rd 2023, 07:51) which shall be used as an example to visualise what the masking process looks like. The code used to construct the mask and to determine the masked image can be found in Appendix C. As mentioned above, this code uses Otsu's method for the thresholding. In Figure 3.4 the mask obtained from running the code can be seen, and also the corresponding masked image. In the masked image, the area outside of the mask is set to white for the sake of visualisation. Keep in mind that this image should not be run with the code, as the white area does not translate to the correct RGB values. When visually analysing the figure, the bloodstain has been fully isolated from the background and the white glare has been removed.

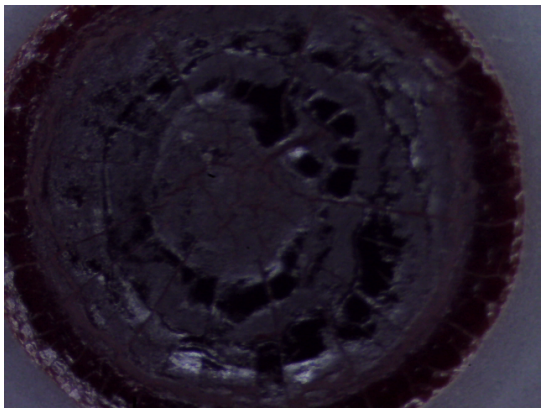
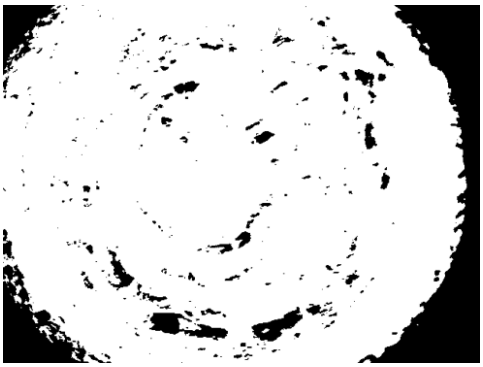
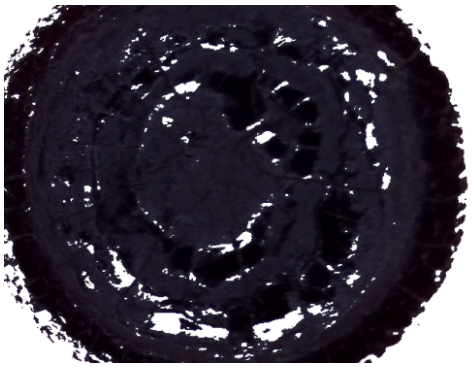


Figure 3.3: Image of Sample 1
(May 5th 2023, 07:51)

However, there are still some irregularities within the bloodstain itself which have been retained. Take the edge for example, there is a clear dark red border around the bloodstain. And there are also dark patches and cracks spread throughout the middle of the bloodstain. To further rid the bloodstain of its irregularities, the image could be masked another time. In Figure 3.5, the mask can be found in the case that we were to mask twice in a row, and also the corresponding masked image. The mask obtained when masking twice shall be referred to as the "inner mask", and the obtained image / bloodstain shall be referred to as the "inner image / bloodstain". When analysing the inner bloodstain visually, it seems that the bloodstain is more consistent in colour, but now the shape of the bloodstain has been compromised. A large number of holes are present, and the information provided by the edge has become very limited.



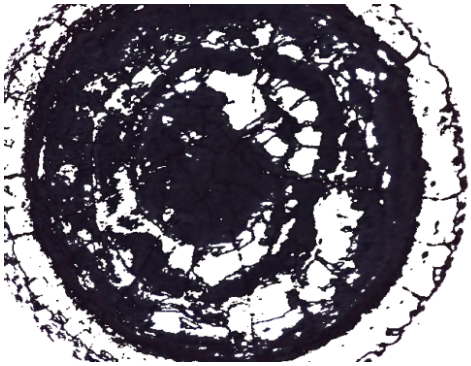
(a) Mask



(b) Masked image

Figure 3.4: Images of the mask and the corresponding masked image of Sample 1

(a) Mask



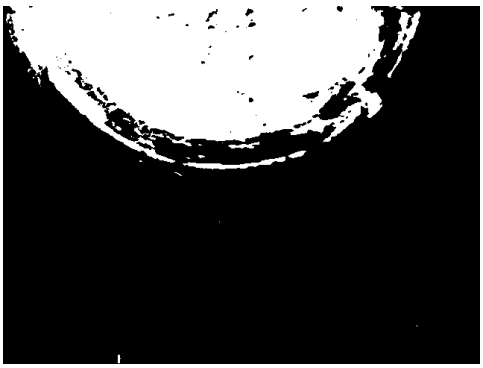
(b) Masked image

Figure 3.5: Images of the inner mask and the corresponding masked image of Sample 1

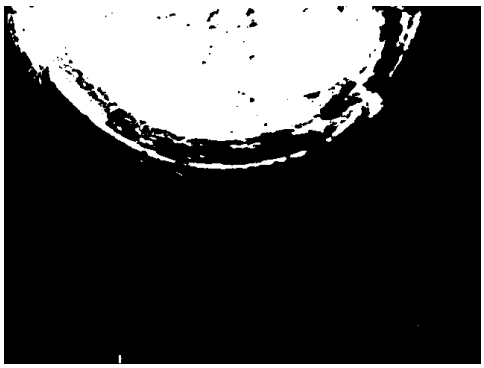
Whether masking once or twice, there are clear sacrifices made for either technique. Masking once gives us a more consistent area, but less consistent colouring. Whereas masking twice gives us more consistent colouring, but less consistent area. How the consistency of these two factors relate to the time since deposition of the bloodstain is still unknown. For this reason, data will be collected for both the complete and inner masked bloodstain. Which information is collected shall be further elaborated on in Section 3.3, and the influence of the images shall be discussed in Chapter 5,

To further expand on the construction of the mask, it is good to refer to one of the problems mentioned back in Subsection 3.2.1, i.e. the problem with the inconsistent areas within a single sample. As mentioned, the area of a bloodstain can change throughout the time, due to the drying of the blood. However, shifts caused by the drying of the blood are usually most prominent within the first day, and afterwards the area of the bloodstain will stabilise. With the sheer amount of images that are taken, the amount of greatly shifted images should be small. To confirm this theory, the choice was made to look at an universal mask with a specified coverage percentage. A pixel is contained in this universal mask if it is present in a certain percentage of all the masks for a sample. This universal mask shall be referred to as a ”coverage mask”.

In Figure 3.6 the 95% and 99% coverage masks can be found for Sample 11. In the previous subsection, it was already mentioned that this sample underwent a large shift. However, when changing the coverage percentage of the mask, the mask barely seems to change regardless of the significant change in coverage. So it seems that there are only a small number of images for which the sample shifts greatly. In Appendix A.1.2 more coverage masks can be seen for other samples. For these samples too it seems that the number of shifted images is minimal. In this thesis, the decision was made to collect the data using a 95% coverage mask rather than constructing a separate mask for each image. This was done as only a small number of images seem to be shifted, and also to keep the observed area during the aging of the bloodstain consistent.



(a) 95% coverage



(b) 99% coverage

Figure 3.6: Image of the masks of Sample 11 with 95% and 99% coverage

3.3. Relevant Variables

As described in the previous section, each image is masked using a mask with a 95% coverage. Now in this section, we can finally decide on which data should be collected (see Appendix C for the used code), and used in the Bayesian networks. In Table 3.1 all collected data can be found, the abbreviation used in the constructed data set and in the Bayesian network, and a description for each variable. The data was collected using the Python code in Appendix C.

To construct a Bayesian network, data should be acquired for various variables. However, it is still unknown in what way the colours within a bloodstain are related to the time of deposition. For this reason, a large number of variables have been collected. For the RGB and GRAY colour channels the following data is collected for each image: mean, min, max, variance, and the 5%, 20%, 50%, 80% and 95% quantiles. (Note that these variables are collected for both the complete bloodstain and the inner bloodstain.) Collecting the aforementioned variables for the colour values, and collecting the time passed since deposition and also the volume of the bloodstain gives us a total of 74 variables.

Note that for the mean and variance, the value is divided by the number of pixels. For both cases this would mean that the acquired value does not necessarily have to be an integer anymore. For the variance, the acquired value is not even an RGB value anymore, i.e. it does not need to be between 0 and 255. As the mean and variance do not take on a finite number of values, the variables are continuous random variables. For the quantiles, however, the acquired value will be a value obtained from the bloodstain, and therefore this value will be a valid RGB value. There are only a finite number of RGB values, thus the distributions of the quantiles will be discrete. Likewise, the minimum and maximum will also be discrete random variables.

There is also some additional data that is collected or could be collected, but which will not be part of the Bayesian networks. As mentioned before in 3.1 the temperature and humidity were kept constant at $25°C$ and 35%, respectively. However, in the future it could be possible that the temperature and the humidity will vary throughout the experiment. In this case, the two factors would be represented by continuous random variables.

The sample number *samp_num* is also collected for each observation. However, this shall only be used for during the plotting or to filter on given samples. Note that it will not be included in the Bayesian network.

Variable name	Random variable type	Description
time	Continuous	Time since deposition of the bloodstain in hours (h).
volume	Discrete	Volume in microlitre (μL). (2, 4, or $8\mu L$)
(in_)X_mean	Continuous	Mean of the X-values, where X is R, G, B or GRAY.
(in_)X_min	Discrete	Minimum of the X-values, where X is R, G, B or GRAY.
(in_)X_max	Discrete	Maximum of the X-values, where X is R, G, B or GRAY.
(in_)X_variance	Continuous	Variance of the X-values, where X is R, G, B or GRAY.
(in_)X_qi	Discrete	i% quantile of the X-values, where $i \in \{5, 20, 50, 80, 95\}$, and X is R,G,B or GRAY.

Table 3.1: List of all collected data. (X can be swapped with R, G, B or GRAY. And adding "_in" refers to the inner bloodstain.)

3.4. Exploratory Analysis

Now that it is known what information is collected from the images, the data can be visualised, and a preliminary analysis can be performed. The visualisation of this data will be done through various plots, and also by visualizing the colours themselves. The data can be split into two groups: the blood samples and the control samples. These groups will be discussed separately in this section.

3.4.1. Blood Samples

Due to the large number of variables, not all variables shall be discussed in depth, nor shall all colour channels. Showing a certain behaviour for a single colour channel can often be extended to the other colour channels, and can be confirmed using the additional plots and images in Appendix A. For any variables which have not been plotted, the provided code in Appendix D can be used. These remaining variables have been checked to some extent, but shall not be discussed extensively in this thesis due to the lack of information provided by the variable.

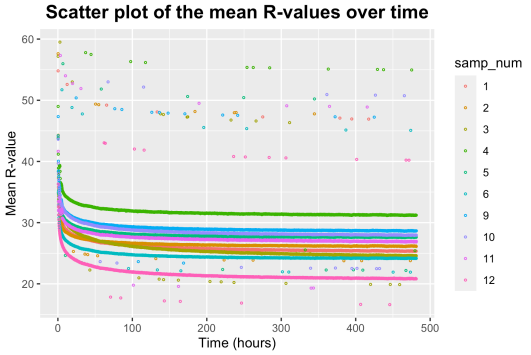
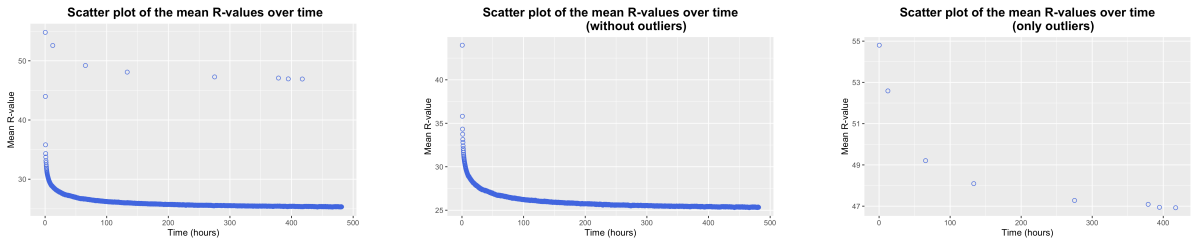


Figure 3.7: Scatter plot of the mean R-values over time

Before analysing specific variables, some commonly shared peculiarities within the data will be pointed out, and clarification shall be given on the causes for these peculiarities. The scatter plot in Figure 3.7 only shows the behaviour for the R_mean with respect to time, but this behaviour is shared with most, if not all, variables. When looking at the plot, the observations from Subsection 3.2.1 seem to have materialised visually. In that subsection it was pointed out that there were inconsistencies in the lighting of the images and the location of the samples. When looking at the image, there seem to be numerous outliers spread throughout. To determine whether these outliers result from the lighting, a manual inspection was performed on Sample 1. In Figure 3.8, Sample 1 has been plotted and the outliers have been separated. In this case, any points with $R\_mean > 45$ are considered outliers. Upon manual inspection of the images for these outliers, it has been found that each of these outliers resulted from images that were overexposed to light. It is therefore not unjustified to assume that most of the outliers are caused by the inconsistent lighting. Especially as the majority of the data does follow a clear trend. As the number of outliers are rather insignificant, when considering the large number of images that we have, they shall not be removed from the data set. However, in the Discussion a method shall be discussed to remove these outliers efficiently.

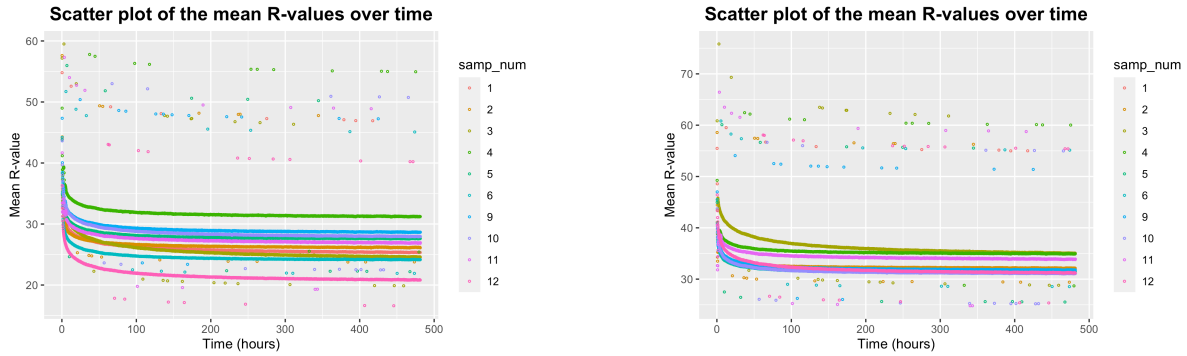
The influence of the location of the sample can also clearly be seen in the image. Let us consider the plot of the R_mean over the $time$ (Figure 3.7). Within the plot, each sample has been assigned a different colour, and it can be seen that each sample follows its own trend. This is not only the case for the R_mean, but also for (almost all) other variables. From these plots it can clearly be seen that here is a rather large sample to sample variance in the variables. This can definitely influence the accuracy of the constructed Bayesian network.



(a) Whole sample (b) Without outliers (c) With outliers

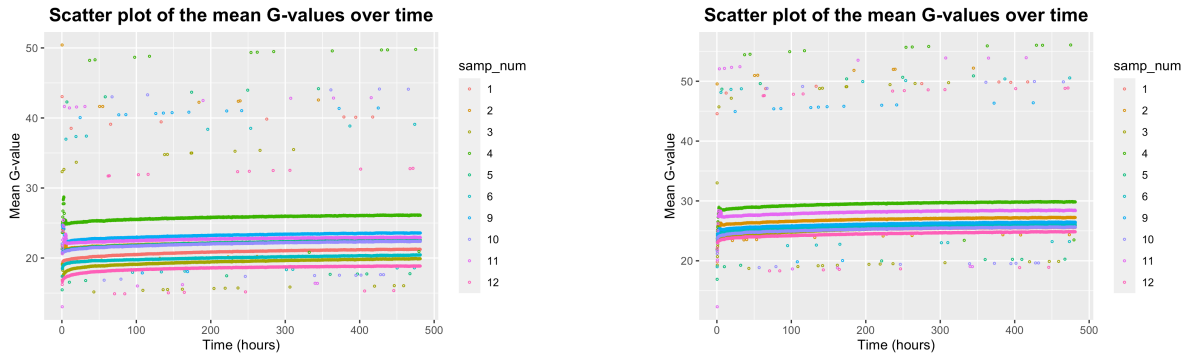
Figure 3.8: Scatter plots of Sample 1 for the mean R-value over the time

Now the focus shall be on the examination of the mean values. For each colour channel the means of both the complete and inner bloodstain have been plotted against the time, and these plots can be seen in Figure 3.9, 3.10, 3.11 and 3.12. A few noteworthy things can be observed. First, it is clear that each sample of a fixed colour channel shows a similar trend. But as mentioned earlier, a large sample to sample variance is present. Whilst the R_mean and the $GRAY\_mean$ show a decreasing trend for each sample, the G_mean and the B_mean are constant or increasing slightly. When a mask is applied a second time and the mean is once again determined, it can be seen that the trend of the samples stays relatively the same, if not becoming slightly less steep. It is also notable that the variance of the means over the different samples becomes smaller when applying a mask a second time. This shows that the variance over the samples occurs mainly in the edges of the bloodstain. Two things could follow from this observation. Firstly, it could mean that removing the irregularities from the bloodstain will decrease the variance and therefore giving us more consistent RGB values regardless of which sample is being analysed. Secondly, it could also have an opposite effect. If the changes in the removed irregularities are related to the aging of the bloodstain, then removing these irregularities could actually worsen any predictions. From the plots it will be difficult to determine whether these irregularities contribute to the predictions or will impede them. This will be further investigated when constructing the Bayesian networks in Chapter 5.



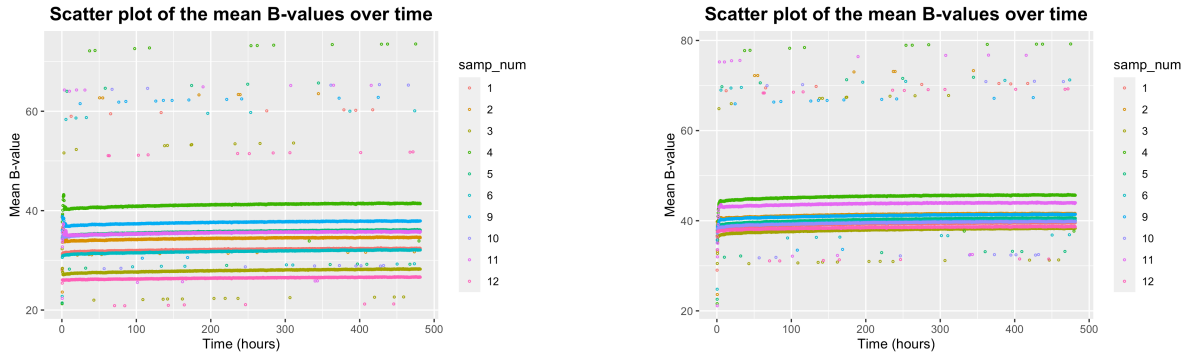
(a) Complete (b) Inner

Figure 3.9: Scatter plot of the time and mean R-values for the complete and inner bloodstain



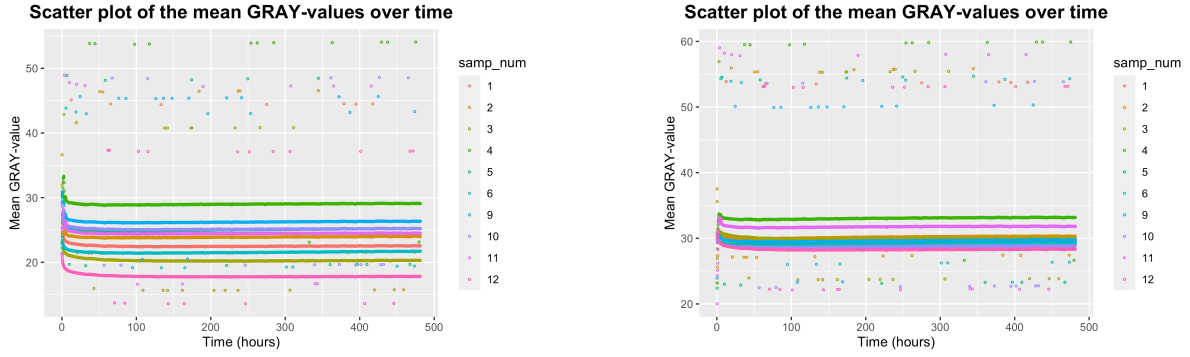
(a) Complete (b) Inner

Figure 3.10: Scatter plot of the time and mean G-values for the complete and inner bloodstain



(a) Complete (b) Inner

Figure 3.11: Scatter plot of the time and mean B-values for the complete and inner bloodstain



(a) Complete (b) Inner

Figure 3.12: Scatter plot of the time and mean GRAY-values for the complete and inner bloodstain

Another interesting variable is the maximum. When plotting the maximum for the complete and inner bloodstain, a clear linear trend can be seen for each colour channel (see Figure 3.14). This linear behaviour indicates that the effect of masking an image a second time does not affect the maximum significantly. Masking an image twice was done with the intention of removing irregularities from the bloodstain. As the maximum is largely unaffected, even after removing the irregularities, it seems that the maximum is influenced less by time-dependent inconsistencies.

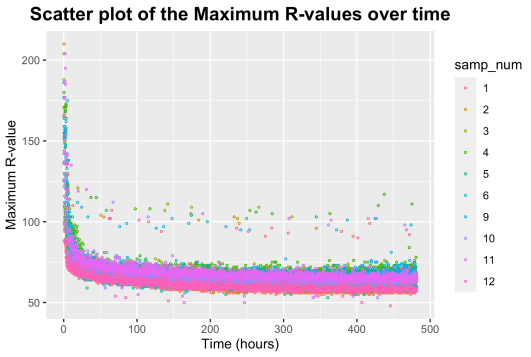
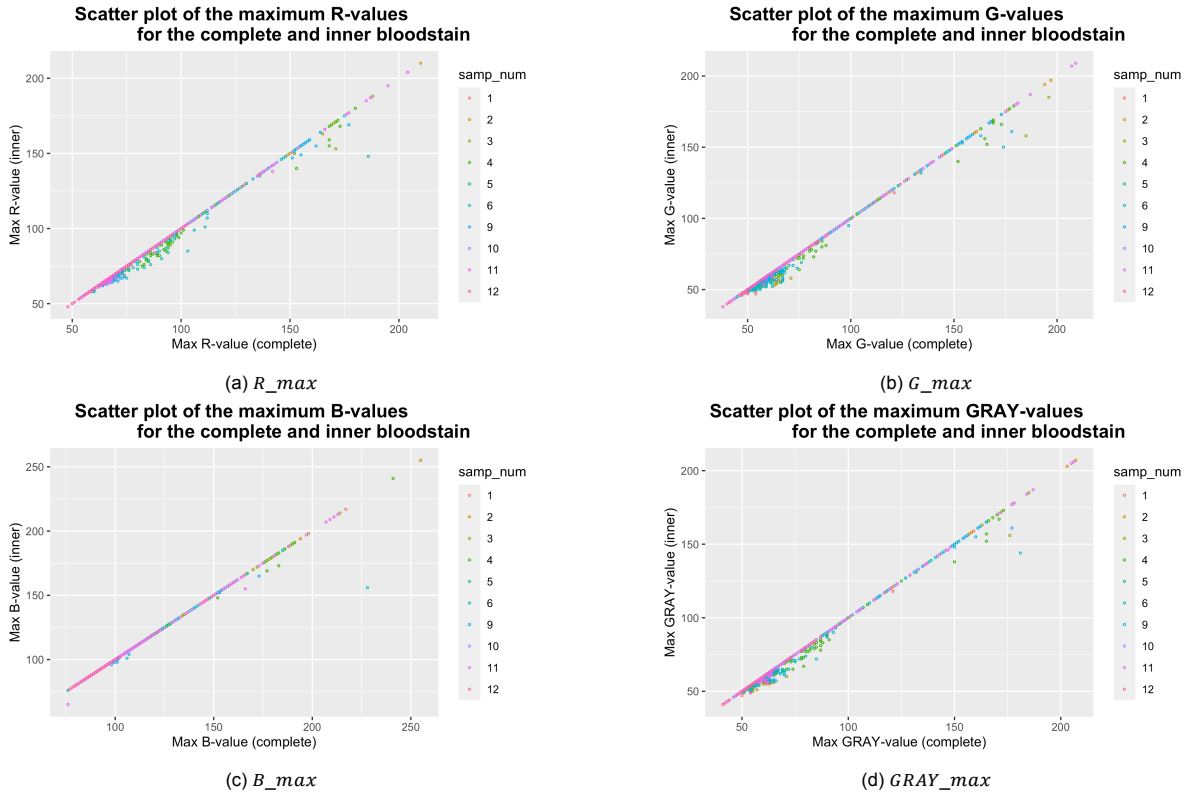


Figure 3.13: Scatter plot of the maximum R-values over time

To further look into this, a scatter plot of the R_max has been plotted over time (see Figure 3.13). When looking at the plot, a similar decreasing trend can be observed as for the mean (in this case the R_mean, see Figure 3.7). However, for the maxima the points are spread out significantly more. The variance of the points within a single sample seems to be larger. This is an understandable observation, as the maximum only looks at one pixel from the whole image. A disruption in a single pixel will not necessarily be removed by a mask. In the case of the R_mean, it can be said that a single disruption would be spread out over the whole image, as we average over the number of pixels contained in the bloodstain. For the maximum, however, the full effect of such a disruption can be carried by a single pixel. So if the maximum is affected, then the disruption is more likely to be larger.

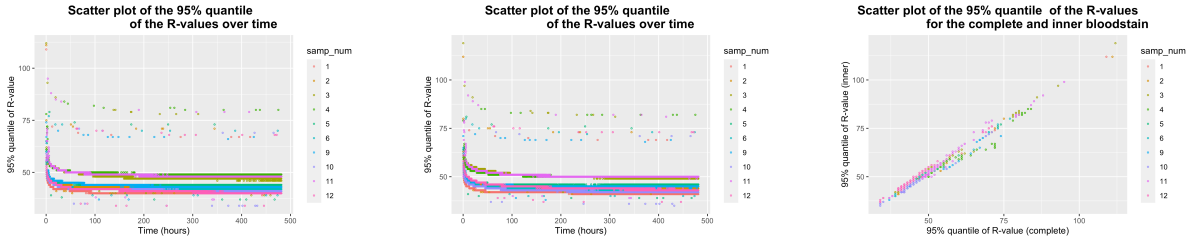


(a) R_max (b) G_max

(c) B_max (d) $GRAY\_max$

Figure 3.14: Scatter plot of the maximum colour values of the complete and inner bloodstain

To counteract the volatility of the maximum, looking at the quantiles is also a possibility. To keep a similar behaviour to the maximum, the quantile would need to be of a high percentage. For the variables that have been collected, the highest quantile is 95%. When constructing similar plots for the R_q95, the behaviour of the data does seem to change slightly. First, from the scatter plots in Figure 3.15a and 3.15b, it is immediately clear that in comparison to the R_max, there no longer seems to be a large variance in the R-values. However, when creating a scatter plot which compares the complete and inner bloodstains (see Figure 3.15c), it seems that more variance has been introduced as can be seen from the scattering of the points. Choosing either the maximum or the 95% quantile shall be further explored during the construction of the Bayesian network.



(a) Complete (b) Inner (c) Both

Figure 3.15

Finally, some more general visualisations of the data shall be discussed. Only having a number represent an RGB value does not give a clear perception on the intensity of the colour associated to the number. In this part, the image of Sample 1 in Figure 3.16 shall be used. In Figures 3.17 and 3.18, all R variables which take on actual colour values, i.e. the discrete random variables, have been plotted for both the complete and inner bloodstain of Sample 1. The same has been done for the G and B colour variables (see Appendix A.1.3).

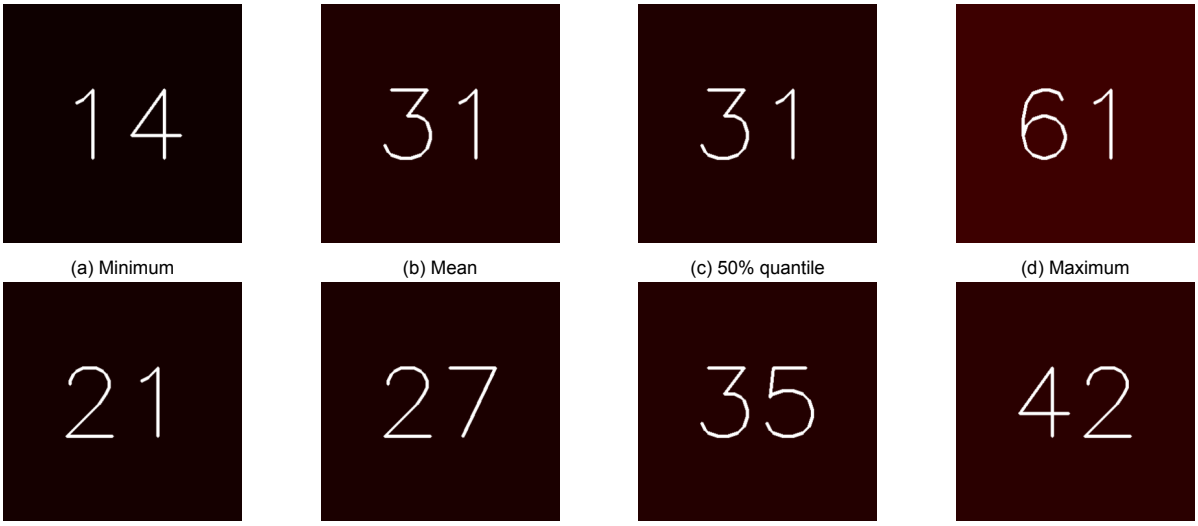
Figure 3.16: Image of Sample 1
(May 9th 2023, 10:47)

When looking at these figures, it can be seen that the value for almost each variable increases when comparing the complete and the inner bloodstain. It is also notable that the increase is the largest for the variables that take on a low value, e.g. the minimum and the 5% quantile. Whereas, for the variables that take on high values, e.g. the maximum and the 95%, this increase seems to weaken significantly. For high colour values there is almost no change when masking the image a second time. These observations of the colours also seem to correspond to previously obtained information, i.e. the unaffectedness of the maximum when applying a mask a second time.

(a) Minimum (b) Mean (c) 50% quantile (d) Maximum

(e) 5% Quantile (f) 20% Quantile (g) 80% Quantile (h) 95% Quantile

Figure 3.17: Visual representation of the data from the R values.
(Sample 1, 2023 May 9th, 10:47)



(a) Minimum (b) Mean (c) 50% quantile (d) Maximum

(e) 5% Quantile (f) 20% Quantile (g) 80% Quantile (h) 95% Quantile

Figure 3.18: Visual representation of the inner bloodstain data from the R values.
(Sample 1, 2023 May 9th, 10:47)

3.4.2. Control Samples

The images of the control samples are all images of only the background paper (see Figure 3.19). It is expected that the background colour should therefore be constant, i.e. the RGB values of the background should be staying constant, for both time and also for different variables.

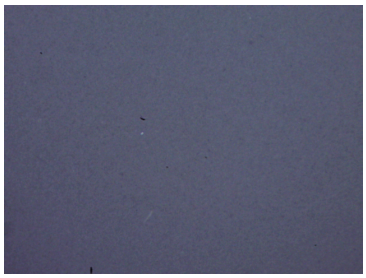
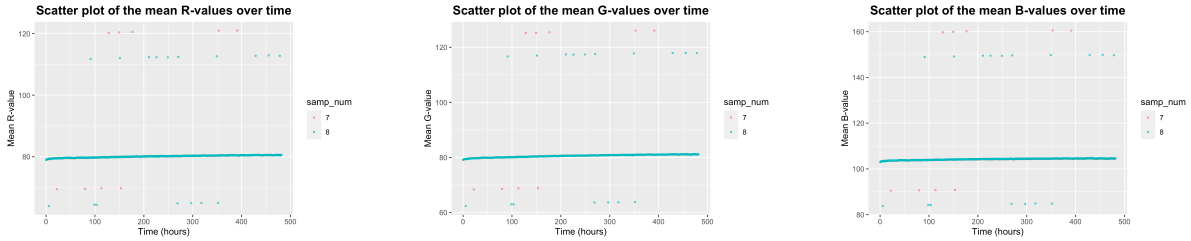


Figure 3.19: Image of Sample 7
(control sample)

Expecting that the background colour stays constant over the time, is not uncalled for. Changes in the background colour, when the environmental factors should be unchanged, indicate that there is some underlying cause which can still influence the results within the climate control chamber. When plotting the mean values of the RGB colour channels, it seems that such an unknown influence is present. In Figure 3.20, a slight increasing trend can be detected in the mean for each colour channels. The cause for this increase is still unknown, but it is hypothesised that this is caused by changes in the lighting settings. Throughout the experiment the mean value increases with approximately a value of 2 to 4, which is relatively small. This increase shall therefore not be taken into consideration for this thesis. However, in the Discussion more will be said on the significance of this increase, and how to deal with it.

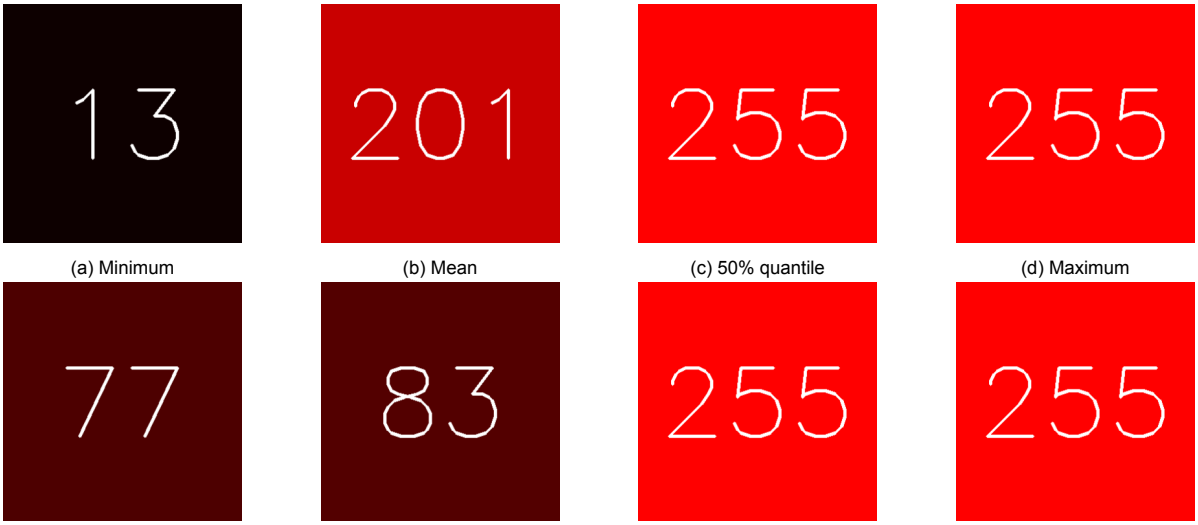
A less obvious expectation is that the RGB values should be constant over the different variables of a colour channel, this is referring to the mean, min, max and the quantiles of the image. This should be the case, as the background should be uniform in colour, so each colour channel of the RGB array should have the same value for each pixel, or at least similar values when taking slight equipment errors into consideration. Taking the mean, min, max, etc. from this array should then also give points which should all be relatively close together. In Figure 3.21, it can be seen that the spread of the R-values for Sample 7 (control sample) is still quite broad. The same can be said for the G- and B-values (see Figure A.10 and A.11). For all three channels it seems that the maximum and any quantile above 50% seems to attain the maximum possible values for RGB values within our setting, i.e. 255. Whereas the remaining variables do seem to be more spread out over the spectrum of 0 to 255. The very low minima could be attributed to some of the small black specks on the control paper, but the rather low values for the 5% and 20% quantile indicate that quite a few pixels seem to have low RGB values. This could possibly be due to the lighting of the sample itself not being completely uniform. Of course, if the lighting is not uniform for the control samples, then this could also affect the images of the bloodstains. Fixing this non-uniform illumination of the image would most likely require changes during the data

collection process, or it would require extensive coding. These methods shall be further discussed in the Discussion.



(a) R_mean (b) G_mean (c) B_mean

Figure 3.20: Scatter plots for the means of the control samples



(a) Minimum (b) Mean (c) 50% quantile (d) Maximum

(e) 5% Quantile (f) 20% Quantile (g) 80% Quantile (h) 95% Quantile

Figure 3.21: Visual representation of the data from the R values (Sample 7)

4

Theory: Bayesian Networks

Now that the data has been acquired, a Bayesian network can be constructed. Before actually explaining what Bayesian networks are, some prior knowledge on probability and graphs should be touched upon. After that, it will be possible to formally define a Bayesian network, and its properties and usage can be discussed. Afterwards, clarification shall be given on the different types of BNs and how they can be constructed. The theory on Bayesian networks in this chapter is heavily based on the book "Bayesian networks: with examples in R" (Scutari and Denis [9]). The information on the learning of a Bayesian network is also further supported by the articles from Scanagatta et al. [8] and Ji et al. [6].

4.1. Prerequisite Knowledge

Before a formal definition can be given for BNs and the methodology behind the usage of the BN, it is important to introduce some terms and theorems from both graph theory and probability theory. A BN gives a visual representation of a joint distribution through the use of graphs. More specifically, a BN has the requirement that it should be represented by a directed acyclic graph (DAG), and using this BN various probabilities can be determined through e.g. inference, i.e. distributions of nodes can be updated given observations. To use inference some definitions and theorems from probability should be introduced.

In this section, directed acyclic graphs will be explained, and some terminology will be elaborated on. Afterwards, various definitions and theorems from probability will be listed. These shall be used in later sections to show how certain probabilities can be calculated within a BN.

4.1.1. Graphs

A Bayesian network is able to give a visual representation of a joint distribution through the use of graphs. However, not any graph can be used. Before defining what a DAG is, it is good to first touch upon graphs themselves.

A **graph** G is an ordered pair (V, A), where the elements of V are called **nodes** (or vertices, or points), and the elements of A are called **edges** (or lines). The set of edges A is a subset of $\{\{a, b\} : a, b \in V\}$ (unordered pairs of elements from V). The edges between nodes, can be illustrated by drawing lines between the nodes. An example of a graph with node set $V = \{1, 2, 3, 4, 5\}$ and edge set $A = \{\{1, 5\}, \{2, 4\}, \{3, 5\}, \{4, 5\}\}$ can be seen in Figure 4.1.

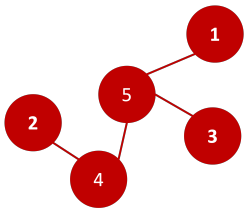


Figure 4.1: A graph with $V = \{1, 2, 3, 4, 5\}$ and $E = \{\{1, 5\}, \{2, 4\}, \{3, 5\}, \{4, 5\}\}$

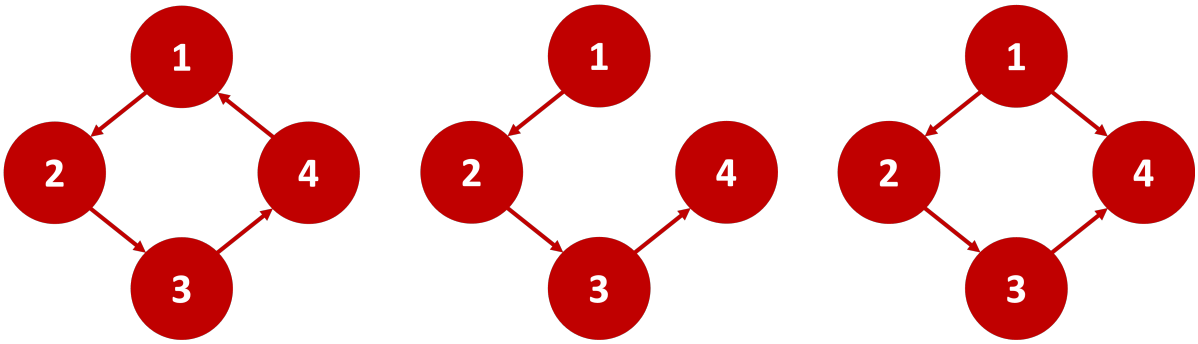
Now that we know what a graph is, we can took a look at DAGs.

Definition 1. *A graph $G = (V, A)$ is a **directed acyclic graph** (DAG) if it is both a directed and an acyclic graph.*

- *A graph is **directed** if all the edges in A are directed, i.e. instead of unordered pairs, we have ordered pairs. So A is a subset of VxV. Elements in A will be called **arcs**, and are depicted by arrows, rather than lines. When considering an element (a, b) from A, the arrow is drawn from node a to node b.*
- *A graph is **acyclic** if the graph does not contain any cycles, i.e. for any node in the graph there is no path back to the node itself.*

Example 2. *In Figure 4.2a a directed graph can be seen with node set $V = \{1, 2, 3, 4\}$ and arc set $A = \{(1, 2), (2, 3), (3, 4), (4, 1)\}$. This graph is not a DAG, however, as the graph contains a cycle for each node in V. Let us start at node 1 for example, then by following arcs $(1, 2)$, $(2, 3)$, $(3, 4)$ and $(4, 1)$ it is possible to arrive back at node 1. So node 1 has a cycle, which means that the graph is not acyclic.*

In this case, the graph can be made acyclic by removing an arc, or changing the direction of an arc. In Figure 4.2b, arc $(4, 1)$ has been removed, and in Figure 4.2c arc $(4, 1)$ has been replaced by $(1, 4)$. In both cases, it is clear that none of the nodes in V have a path to themselves, so therefore both graphs are acyclic. Which means that both graphs are DAGs.



(a) Not acyclic (b) Acyclic (c) Acyclic

Figure 4.2: Example of a directed graph which is not acyclic, and two similar graphs which are acyclic

For a node within a graph, two more terms should also be introduced: the parents and children of a node. These terms will be used when using the conditional independence within the Bayesian network. We will further elaborate on what the conditional independence is in Subsection 4.1.2.

Definition 2. *Consider a directed graph $G = (V, A)$, and take a node $x \in V$.*

- *The **parents** of node x, $\mathcal{P}(x)$, are the direct predecessors of node x, i.e. the nodes for which there are outgoing arcs going to node x. This can also be written as following:*

$$\mathcal{P}(x)\} = \{\{(b, x) : \exists b \in V : (b, x) \in A\}$$

- *The **children** of node x, $\mathcal{C}(x)$, are the direct successors of node x, i.e. the nodes for which there are incoming arcs coming from node x. This can also be written as following:*

$$\mathcal{C}(x) = \{(x, a) : \exists a \in V : (x, a) \in A\}$$

Example 3. *Consider the graph in Figure 4.2c. Then in Table 4.1 the parents and children for each node can be found.*

Node	1	2	3	4
Parents	-	1	2 and 4	1
Children	2 and 4	3	-	3

Table 4.1: Table of the parents and children of the nodes in Figure 4.2c

4.1.2. Probability

Before listing the various properties and theorems that are used when determining probabilities within the Bayesian network, first we shall recall some basic definitions.

Definition 3. *Let A and B be two events with $\mathbb{B} > 0$.*

- *The **conditional probability of** A **given** B is given by*

$$\mathbb{P}(A|B) = \frac{\mathbb{P}(A \cap B)}{\mathbb{P}(B)}$$

- *A and B are **independent**, i.e. $A \perp B$, if*

$$\mathbb{P}(A|B) = \mathbb{P}(A)$$

- *A and B are **conditionally independent** if there is some event C such that*

$$\mathbb{P}((A \cap B)|C)) = \mathbb{P}(A|C)\mathbb{P}(B|C)$$

In this case, we write $A \perp B|C$.

The following theorems are useful when making calculations in the BNs. The below terms can often be simplified to a certain extent due to the conditional independence within the BN. Further elaboration on when we have conditional independence in a BN will be given in Section 4.2.

Theorem 1. *(Law of total probability) For any two events A and B we have the following:*

$$\mathbb{P}(A) = \mathbb{P}(A \cap B) + \mathbb{P}(A \cap B^C)$$

Theorem 2. *(Bayes' theorem) Consider two events A and B such that $\mathbb{P} \neq 0$. Then we have the following:*

$$\mathbb{P}(A|B) = \frac{\mathbb{P}(B|A)\mathbb{P}(A)}{\mathbb{P}(B)}$$

Theorem 3. *(Chain Rule for probability) Consider two events A and B. Then we have the following:*

$$\mathbb{P}(A \cap B) = \mathbb{P}(B|A)\mathbb{P}(A)$$

Note that this immediately follows from rewriting the definition of the conditional probability.

- *Expanding this to any finite number of events $A_1,...,A_n$ would give us:*

$$\mathbb{P}(\bigcap_{i=1}^{n} A_i) = \prod_{k=1}^{n} \mathbb{P}(A_k|\bigcap_{j=1}^{k-1} A_j)$$

4.2. Bayesian Networks Basics

In the previous section, some important terms and theorems were discussed. Now, it is possible to formally define Bayesian networks

Definition 4. *Consider a graph G for which $V = \{X_1,...,X_n\}$. Assume that the nodes X_i are all random variables. Then G is a **Bayesian network** (BN) if it satisfies the following requirements:*

- *G is a directed acyclic graph (DAG).*
- *And for a density or mass function p the following holds:*

$$p(x_1,...,x_n) = \prod_{i=1}^{n} p(x_i|\mathcal{P}(x_i))$$

  - *Note that $\mathcal{P}(x_i)$ refers to the parent nodes of x_i.*
  - *If $\mathcal{P}(x_i) = \emptyset$, then $p(x_i|\mathcal{P}(x_i)) = p(x_i)$*

In the case of a BN, the nodes are defined by random variables, and the arcs between these nodes represent the conditional dependencies between these random variables. A BN can be said to consist of two parts:

- *The qualitative part, i.e. the DAG G which depicts the dependencies between the various nodes. We shall refer to the graph as the **structure** of the BN.*
- *The quantitative part, i.e., the (conditional) probability functions for each random variable within the BN.*

The second requirement of the Bayesian network is very important, and follows from the chain rule for probability (see Theorem 3) by using the conditional independence within BNs.

Within a BN many conditional independencies are present. For each node, the Local Markov Property is able to give one such conditional independence relation (see Definition 5).

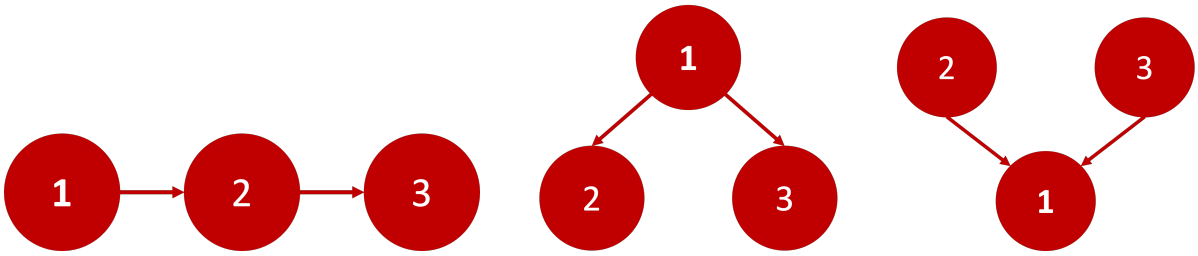
Definition 5. *(Local Markov Property) Each node X_i within a BN is conditionally independent of its non-descendants given its parents. The non-descendants of X_i are the nodes X_j for which there is no path from X_i to X_j.*

However, there can also be many other conditional independence relations between the nodes. Using the d-separation criterion, all independencies within a BN can be determined. To explain the d-separation criterion, we will first need to define the various types of connections between nodes.

Definition 6. *Consider a graph G. When considering three nodes 1, 2 and 3 we can categorise three types of elementary connections: a **serial**, **diverging** and a **converging** connection.*

- *Serial connection: One incoming arc and one outgoing arc*
- *Diverging connection: Two outgoing arcs*
- *Converging connection: Two incoming arrows*

An example of what each of these connections look like can be found in Figure 4.3.



(a) Serial (b) Diverging (c) Converging

Figure 4.3: Types of connections in a Bayesian network

Now using the below definition for d-separation, and Theorem 4 any conditional dependency within a BN can be determined.

Definition 7. *Let us consider a DAG G. Consider three disjoint subsets of G: A, B and C. C **d-separates** A from B if for any path between a node in A and a node in B there exists a node v satisfying one of the following two conditions:*

- *v has converging arcs and neither v nor any of its descendants are in C.*
- *v is in C and does not have converging arcs.*

In more intuitive terms, it can be said that C blocks of any undirected path from A to B. So there is no way to go from one set to the other and then back to itself.

Theorem 4. *(Conditional independence and d-separation) Consider a Bayesian network, and let X, Y and Z be three disjoint sets of nodes. If Z d-separates X and Y, then*

$$X \perp Y | Z$$

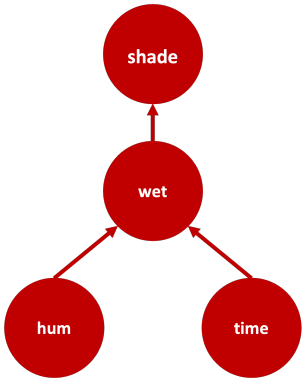


Figure 4.4: Example Bayesian network for a blood stain

Example 4. *Let us consider the Bayesian network in Figure 4.4 and assume that the nodes within the graph are discrete random variables. In Table 4.2 each node is described.*

Node name	Description	Notation
hum	Humidity of the air (low or high)	hum^- = low and hum^+ = high
$time$	Time since deposition (short or long)	$time^-$ = short and $time^+$ = long
wet	Wetness of the stain (yes or no)	wet^- = no and wet^+ = yes
$shade$	Shade of the stain (dark or light)	$shade^-$ = light and $shade^+$ = dark

Table 4.2: Description of the nodes of the Bayesian network in Figure 4.4

For each node, the conditional probability can be found in Table 4.3. (The chosen values are not representative, and were chosen arbitrarily for the sake of the example.) In this example, the following probabilities will be determined using varying methods: $\mathbb{P}(hum = x_1, time = x_2, wet = x_3, shade = x_4) = \mathbb{P}(x_1, x_2, x_3, x_4)$ *and* $\mathbb{P}(wet^+)$*,*

- *For convenience, we shall write*

$$\mathbb{P}(hum = x_1, time = x_2, wet = x_3, shade = x_4) = \mathbb{P}(x_1, x_2, x_3, x_4)$$

. Then we can use the chain rule for probability (Theorem 3), and the conditional independence to simplify the expression:

$$\begin{aligned}\mathbb{P}(x_1, x_2, x_3, x_4) &= \mathbb{P}(x_4|x_1, x_2, x_3) \cdot \mathbb{P}(x_3|x_1, x_2) \cdot \mathbb{P}(x_2|x_1) \cdot \mathbb{P}(x_1) \\ &= \mathbb{P}(x_4|x_3) \cdot \mathbb{P}(x_3|x_1, x_2) \cdot \mathbb{P}(x_2) \cdot \mathbb{P}(x_1)\end{aligned}$$

For any combination (x_1, x_2, x_3, x_4)*, the above expression can be filled in using provided tables with the conditional probabilities. For example:*

$$\begin{aligned}\mathbb{P}(hum^+, time^+, wet^+, shade^+) &= \mathbb{P}(shade^+|wet^+) \cdot \mathbb{P}(wet^+|hum^+, time^+) \cdot \mathbb{P}(time^+) \cdot \mathbb{P}(hum^+) \\ &= 0.4 \cdot 0.4 \cdot 0.6 \cdot 0.8 = 0.0768\end{aligned}$$

- $\mathbb{P}(wet^+)$ *can be calculated by using the law of total probability on the parents of the* wet *node (Theorem 1), and using the chain rule for probability (Theorem 3). Also note that* hum *and* $time$ *are independent. Then we can calculate the following:*

$$\begin{aligned}\mathbb{P}(wet^+) =& \mathbb{P}(wet^+, hum^+, time^+) + \mathbb{P}(wet^+, hum^+, time^-) \\ &+ \mathbb{P}(wet^+, hum^-, time^+) + \mathbb{P}(wet^+, hum^-, time^-) \\ =& \mathbb{P}(wet^+|hum^+, time^+)\mathbb{P}(hum^+, time^+) + \mathbb{P}(wet^+|hum^+, time^-)\mathbb{P}(hum^+, time^-) \\ &+ \mathbb{P}(wet^+|hum^-, time^+)\mathbb{P}(hum^-, time^+) + \mathbb{P}(wet^+|hum^-, time^-)\mathbb{P}(hum^-, time^-) \\ =& \mathbb{P}(wet^+|hum^+, time^+)\mathbb{P}(hum^+)\mathbb{P}(time^+) + \mathbb{P}(wet^+|hum^+, time^-)\mathbb{P}(hum^+)\mathbb{P}(time^-) \\ &+ \mathbb{P}(wet^+|hum^-, time^+)\mathbb{P}(hum^-)\mathbb{P}(time^+) + \mathbb{P}(wet^+|hum^-, time^-)\mathbb{P}(hum^-)\mathbb{P}(time^-) \\ =& 0.4 \cdot 0.8 \cdot 0.6 + 0.8 \cdot 0.8 \cdot 0.4 + 0.1 \cdot 0.2 \cdot 0.6 + 0.5 \cdot 0.2 \cdot 0.4 \\ =& 0.5\end{aligned}$$

hum	low	high
	0.2	0.8

(a) Humidity

time	short	long
	0.4	0.6

(b) Time

wet	hum	time	yes	no
	low	short	0.5	0.5
	high	short	0.8	0.2
	low	long	0.1	0.9
	high	long	0.4	0.6

(c) Wetness

shade	wet	light	dark
	yes	0.6	0.4
	no	0.2	0.8

(d) Shade

Table 4.3: (Conditional) Probabilities for all the nodes from Figure 4.4

4.3. Types of Bayesian Networks

The type of a Bayesian network depends on the type of random variables within the graph. Bayesian networks can be categorised into three groups:

- Discrete Bayesian networks: All the nodes in the BN are discrete random variables, and the Bayesian network is accompanied by a conditional probability table. As was seen in Example 4, other probabilities can be calculated using various definitions and theorems from probability (see Subsection 4.1.2).
- Continuous Bayesian networks: There are two types of continuous Bayesian networks. For the Gaussian Bayesian network, the conditional probability distribution of the node is normal / Gaussian and the influence of a parent node on its child is regressed over the arcs.

  For a non-parametric Bayesian network the distribution of the data is given by arbitrary, continuous and invertible distributions. In this case, the influence of a parent node is passed onto its child through normal copulas.
- Hybrid Bayesian networks: In this case, there are both discrete and continuous random variables. The probability distribution of the continuous random variables is not restricted to the normal distribution. The combination of both discrete and continuous random variables makes it possible for the BN to cover far more diverse situations, but this comes at the cost of a more complicated BN.

4.4. Structure Learning

In previous sections, the structure of the BN was already given. However, when only data has been provided, the structure of the BN is still unknown. A structure could be determined using two different methods: expert judgement or structure learning algorithms. In this thesis, the structure will be determined through structure learning algorithms. The structure learning algorithms can be categorised into three different groups: score-based, constraint-based and hybrid structure learning.

Let us start with the most used approach, i.e. score-based structure learning. These algorithms search for a DAG which maximizes a certain score function, e.g. the Bayesian Information Criterion (BIC). This score functions shows the goodness-of-fit of the DAG to the data, and is usually rather simple to calculate. The simplicity of these algorithms make the run time relatively shorter than for the other two groups. However, this method does also have weaknesses. Score-based structure learning focuses only on the maximization of the score function. So it does not take the logical aspect of the nodes into consideration. This could lead to a BN for which the dependencies unnatural or even incorrect. Some examples of score-based algorithms are hill-climbing and tabu search.

On the other hand, constraint-based structure learning focuses on analysing the conditional independence of the various variables. These algorithms use conditional independence tests to determine the structure of the BN. This evaluation of arcs between nodes can be done in various ways. One

possible option is to start with a graph for which all possible arcs are present, and then continuously remove arcs based on statistical tests for conditional independence. This method is called a backward selection procedure, and an example of an algorithm which makes use of this is the PC algorithm. Conversely, it is also possible to use a forward selection procedure, i.e. you start with a graph with no arcs, and continuously add arcs between nodes which are not conditionally independent. Algorithms like the Grow-Shrink algorithm make use of forward selection, but there also various other procedures used when constructing the graph with constraint-based structure learning, for example two step procedures where both forward and backward selections are made. Constraint-based structure learning algorithms are exceptional at creating accurate BNs, but due to the more complicated nature of the algorithm it is often a significantly slower algorithm. Especially when compared to the simple score-based algorithms.

And finally there are also hybrid structure learning algorithms which combine constraint-based and score-based algorithms. This makes it possible to balance out the weaknesses for both groups of algorithms. The most well-known algorithms from this type are the Sparse Candidate algorithm (SC) and the Max-Min Hill-Climbing algorithm (MMHC). Both algorithms are based on two steps: restrict and maximise. The restrict phase implements a constraint-based algorithm to reduce the space of possible DAGS, and the maximise phase implements a score-based algorithm which finishes the optimal DAG within the restricted space.

In this thesis, the score-based algorithm *hill-climbing* was used as the main structure learning method. This was due to the fact that the sheer size of the data made it very inefficient to run the other structure learning methods and also prone to running errors.

The algorithm for Hill-Climbing is as following ([9]):

1. Choose an arbitrary network structure G over V. Often chosen empty.
2. Compute the score of G, using the notation $Score_G = Score(G)$ and set it as the maximum score, i.e. $maxscore = Score_G$. In our case, BIC is used as the score function.
3. Repeat the following steps as long as $maxscore$ increases:
   (a) For every possible arc addition, deletion or reversal for which G stays a DAG:
      i. Compute the score of the modified graph G^*, $Score_{G^*} = Score(G^*)$.
      ii. If $Score_{G^*} > Score_G$, set $G = G^*$ and $Score_G = Score_{G^*}$
   (b) Update $maxscore$ with the new value of $Score_G$
4. Return the DAG G

5 Results

In this chapter, various Bayesian networks will be constructed, cross-validated and analysed. This whole process was carried out in RStudio and made use of the *bnlearn* package. The various functions used during the process shall also be described. After having constructed numerous Bayesian networks, the most optimal Bayesian network shall be determined, and shall be discussed in-depth.

5.1. Bayesian Network: Structure and Cross-validation

First, the whole process of determining a Bayesian network, cross-validating it and various parts of the BN shall be analysed. This process will be showcased when all the variables mentioned in 3.3 are used. Afterwards, subsets of the data will be taken to observe the changes in the structure and the predictive quality of the Bayesian network. These Bayesian networks will be compared based on the Mean Squared Error (MSE), and also based on plots of the predictions.

5.1.1. General Process

In this subsection the general process for creating and evaluating the BN shall be described. The process shall be showcased when using the whole data frame that has been provided by LION. In the R-code this data frame is referred to as *data*.

Now that the data has been prepared, the structure of the BN can be determined. As mentioned in Chapter 4, hill-climbing (HC) shall be the chosen structure learning algorithm in this thesis due to the HC algorithm having a significantly better run time for our data frame. The used function in RStudio is *hc*, for which the Bayesian Information Criterion (BIC) is used as the default score function to evaluate the BN. Using the function *hc* gives us the BN in Figure 5.2. Seeing that the determined structure has this many arcs could both be seen as a positive and a negative. Positive, as it indicates that there are dependencies between the variables, so it does show that the time since deposition and the colour of a bloodstain are somewhat related. But it can also have negative effects, as there could be more arcs between factors which should be conditionally independent, for example the *time* and *volume*, or arcs that are not logical, for example arcs mapping from R_mean to $time$. For a small BN seeing a irregularity can be solved easily, but with the large number of arcs in Figure 5.2 it cannot be seen easily, nor is there an efficient way to check all the arcs.

Checking all the arcs between nodes would be inefficient, and additionally not all dependencies between nodes are known or easy to spot. However, there is one node which should be checked to some extent, and that node is the *time* node. A simple check for the *time* node would be to look at both the parents and the children of the node.

In this case, the time node does not have any parent nodes. This sounds quite reasonable, because the change in colour happens because time passes and not the other way around. The children nodes, on the other hand, are

```
> bn.hc_data$nodes$time$children
 [1] "volume"          "GRAY_mean"       "GRAY_min"        "GRAY_max"
 [5] "GRAY_variance"   "GRAY_q20"        "GRAY_q50"        "GRAY_q80"
 [9] "GRAY_q95"        "R_mean"          "R_min"           "R_max"
[13] "R_variance"      "R_q5"            "R_q20"           "R_q50"
[17] "R_q80"           "G_mean"          "G_min"           "G_max"
[21] "G_variance"      "G_q20"           "G_q50"           "G_q80"
[25] "G_q95"           "B_mean"          "B_min"           "B_variance"
[29] "B_q5"            "B_q50"           "B_q80"           "B_q95"
[33] "in_GRAY_mean"    "in_GRAY_min"     "in_GRAY_variance" "in_GRAY_q20"
[37] "in_GRAY_q50"     "in_GRAY_q80"     "in_GRAY_q95"     "in_R_mean"
[41] "in_R_max"        "in_R_variance"   "in_R_q5"         "in_R_q50"
[45] "in_R_q95"        "in_G_mean"       "in_G_min"        "in_G_variance"
[49] "in_G_q5"         "in_G_q20"        "in_G_q50"        "in_G_q95"
[53] "in_B_mean"       "in_B_q5"         "in_B_q50"        "in_B_q80"
[57] "in_B_q95"
```

Figure 5.1: Children nodes of the *time* node in Figure 5.2

less obvious. A large number of the variables are children of the *time* node, but the ones that are not a child do not necessarily seem to follow a certain pattern. For one colour channel the minimum is missing, for another the maximum, some quantiles are retained and others are discarded. This is also one of the cons of hill-climbing which were mentioned in Chapter 4, the fact that the algorithm maximizes the score function blindly, paying no heed to the conditions independence or logic. Once again, due to the large number of arcs it will be difficult to argue which arcs should be retained or not. But it will be good to keep this phenomena in mind.

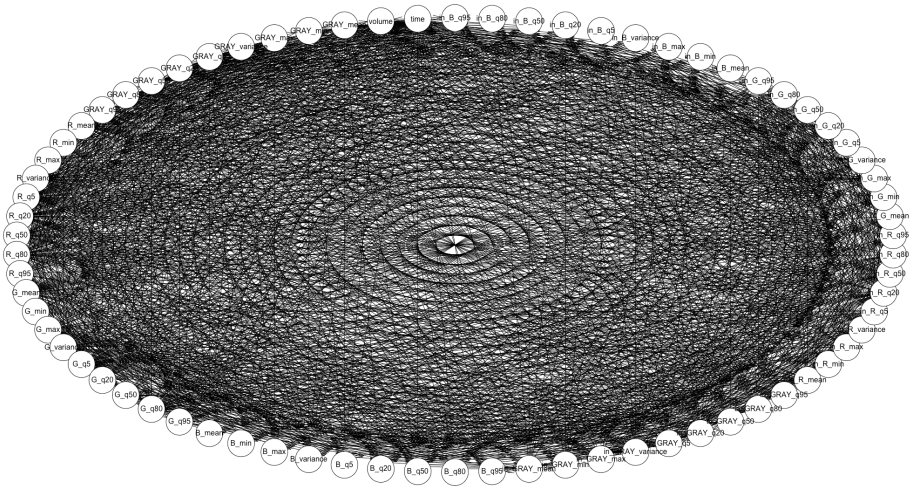


Figure 5.2: Bayesian network constructed using hill-climbing on the complete data frame

The structure of the BN has now been determined and the most important node, the *time* node, has been examined to some extent. Now it will be tested whether the structure actually fits the data or not. This shall be done using **k-fold cross-validation**. K-fold cross-validation is a method to determine the accuracy of a model. The method goes as following:

1. The data is split into a total of k subsets of equal size.
2. The model is fitted using $k-1$ of the sets, these $k-1$ sets are called the **training data**.
3. The fitted model is then used on the remaining subset, which is called the **testing data**, and the loss function is calculated.
4. Step 2 and 3 are performed a total of k times until each subset has been used as the testing set once. Afterwards the average is taken over the calculated losses.

K-fold cross-validation can also be used with multiple runs. In that case, the overall loss is the average of the loss estimates from the different runs. Additionally, the standard deviation of the loss can also be calculated.

In *bnlearn*, k-fold cross validation can be performed using the *bn.cv* function. Now, we can cross validate the Bayesian network for the full data set (Figure 5.2. For the cross-validation the following parameters were chosen:

- k = 5, so the training and testing sets are 80% and 20% of the data respectively.

- loss function: Mean Squared Error (MSE)
- target node forecasts: *time*
- number of runs = 5

Using 5-fold cross-validation on the Bayesian network in Figure 5.2 gives us the following values:

$$\text{MSE} = 19303.45 \implies \text{RMSE} = 138.94$$

$$\text{Standard deviation} = 1.46$$

In Figure 5.3, the reason for the high MSE can immediately be seen. It seems that the prediction for the time is constant, regardless of the provided data. The reason for the prediction being constant, can be found when investigating the coefficients determined during the fitting process. On inspection of the children nodes of the *time* node, it can be seen that the coefficient corresponding to the time is very small. In Table 5.1, some of these coefficients can be found. Seeing how small these coefficients are, it is likely that the effects from any node on the *time* are minimal. And as the signs are also varying, the effects can also neutralize each other. From the MSE and also from the plot it seems that the BN, constructed using all the variables, is not a good fit for our data. In the following section, the process of constructing and evaluating BNs shall be repeated. However, this time for other subsets of the data.

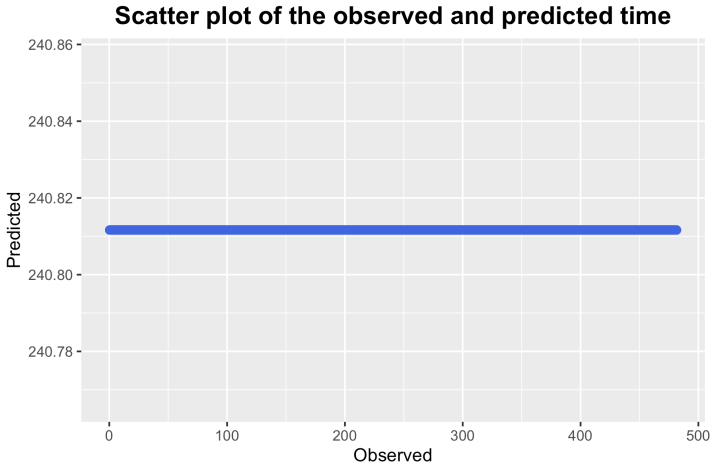


Figure 5.3: Plot of the observed and predicted values for one of the cross-validation runs.

Node	Coefficient
volume	0.0045
GRAY_mean	0.0013
R_mean	- 0.0009
R_max	-0.0186
G_mean	0.0002
B_mean	-0.0001

Table 5.1: Coefficient corresponding to the $time$ for a given node

5.1.2. Other Bayesian Networks

Now that we have seen how a Bayesian network is constructed and evaluated, the same steps can be repeated for subsets of the data. However, in this section the order of the analysis will be slightly different. First, BNs will be constructed and immediately cross-validated. Based on the MSE, the BNs can be compared and a decision is made on whether further analysis of the BN will be interesting.

In Table 5.2, the MSE can be seen for the different colour channels. When looking at the MSE, it seems that the variables by itself, the variables for the colour channels for the red variables seem

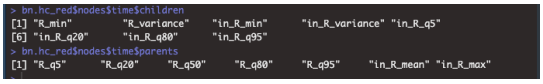
to give us the best results. It seems that the combination the variables of the red and blue variables further improves the MSE. However, when combining the red and green variables, the MSE becomes significantly worse. This behaviour is quite peculiar as the MSE for the green and blue variables separately seem to be somewhat similar. When comparing the figures in Appendix A.1.3, we do note that the spread of the blue values within each individual image is larger. If a larger spread is attained, then the data possibly provides us with more information which could improve a prediction.

On comparison of the BN constructed with only the red variables, and both the red and green variables, the latter does seem to be slightly better. It is noteworthy to mention that the difference is rather small. Especially when converting the MSE to the Root Mean Squared Error (RMSE) and also when comparing the difference to the MSE itself.

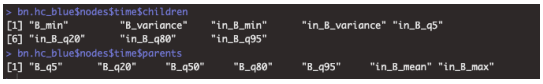
Variables	Mean Squared Error (MSE)	Standard deviation of MSE
Gray variables	19303.48	1.99
Red variables	15149.81	1.99
Green variables	17335.29	11.16
Blue variables	17758.62	8.86
Red and green variables	18087.33	1.43
Red and blue variables	14866.17	2.16

Table 5.2: Table of the MSE and the standard deviation of the MSE obtained from the cross validation of the BNs constructed for each colour channel

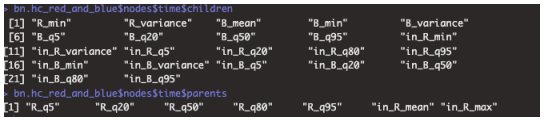
In Figure 5.5, the BNs constructed for the red, blue, and red and blue variables can be found. Although there are significantly less arcs, it is still difficult to actually analyse all of them. So as before the parents and children of the *time* node shall be investigated. In Figure 5.4, the parent and children nodes of the *time* node in the BNs can be found. For all three BNs, an aforementioned problem is immediately visible. In our newly constructed BNs, the *time* node has parent nodes. In reality, this should not be possible as there is no clear link from how the colour of a bloodstain can affect time itself.



(a) Red



(b) Blue

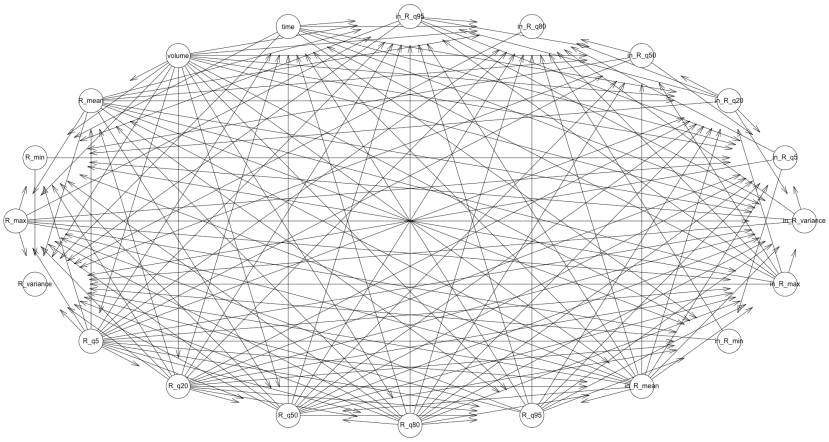


(c) Red and blue

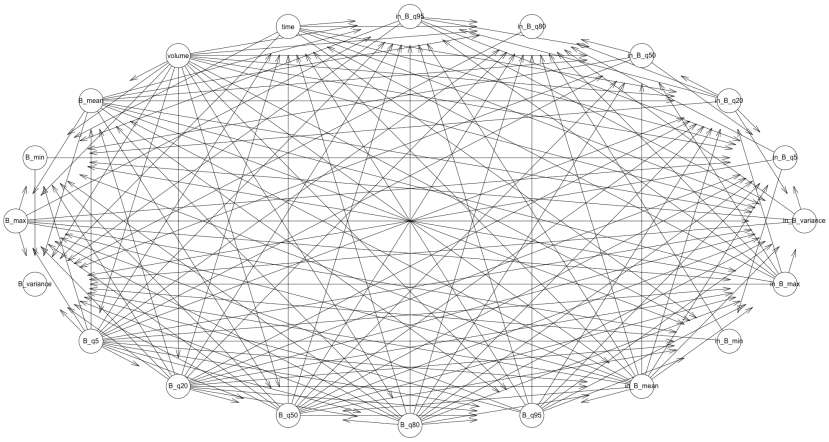
Figure 5.4: Parent and children nodes of the *time* node in the BNs of Figure 5.5

Another anomaly can be found when looking at the predictions made by the model. In Figure 5.6a, 5.6c and 5.6e, the predictions have been plotted for the colour channels. One thing which stands out for each of those Figures is the y-axis. More specifically, the values in the plot take negative y-values, which means that there are negative time predictions. In reality, this is obviously not possible, so these points should not be taken into consideration. Cropping the images, so that only the positive points remain would give use the plots in Figure 5.6b, 5.6d and 5.6f. These plots are already a clear improvement from the constant line that we observed in Figure 5.3. However, there are still problems within these plots. Although the data has more of a trend now, the data seems to follow more of a logarithmic trend, rather than the desired linear trend. And also, the predicted points seem to have a large variance. The data seems to be becoming more spread out, as more time passes.

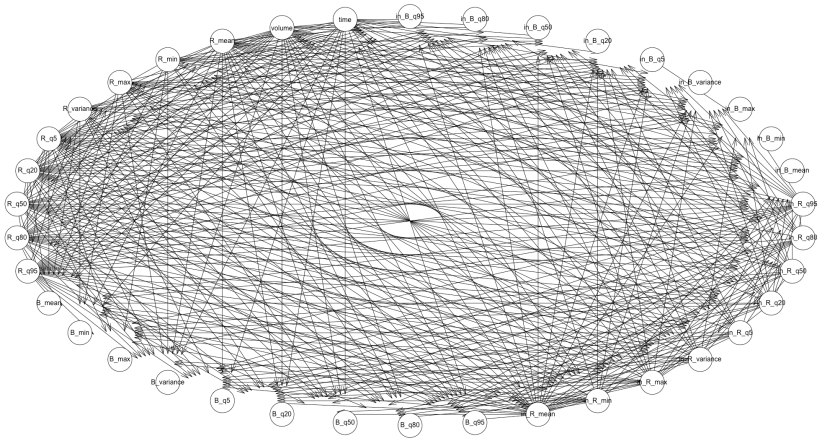
Also when combining the red and blue variables in the BN, the scatter plot of the predictions (Figure 5.6e) starts to take on properties of the predictions of both the red and blue variables. In the earlier time predictions the increasing trend of the red variables can be seen, which does improve the accuracy of the predictions for the beginning. However, for the latter part of the predictions, the constant behaviour from the blue variables seems to be appearing, which worsens the accuracy.



(a) Red

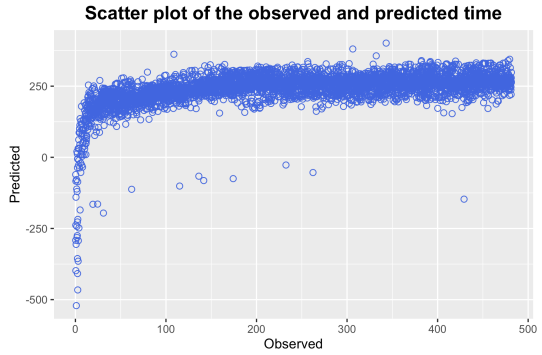


(b) Blue

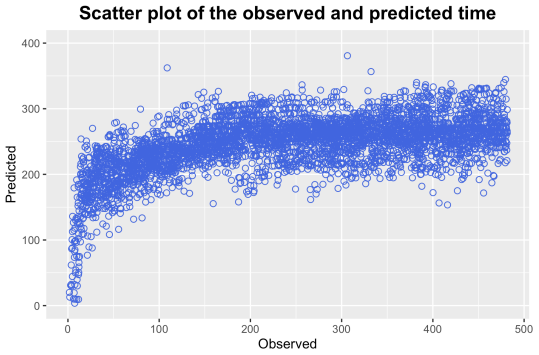


(c) Red and blue

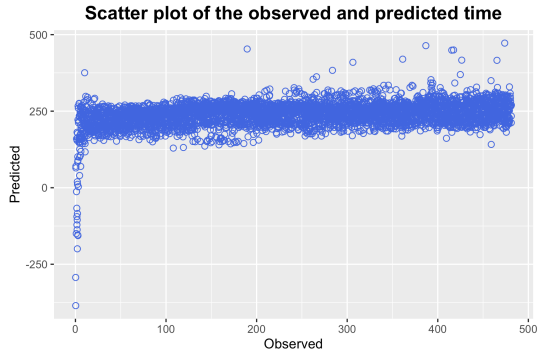
Figure 5.5: Bayesian networks for various colour channels



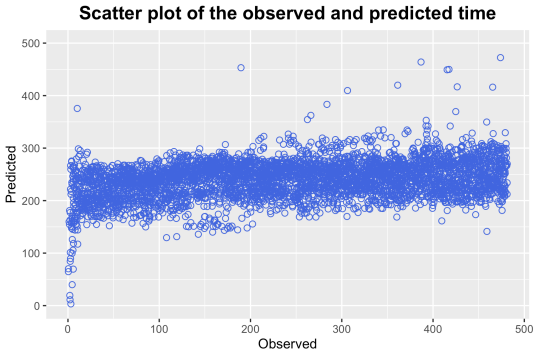
(a) Red



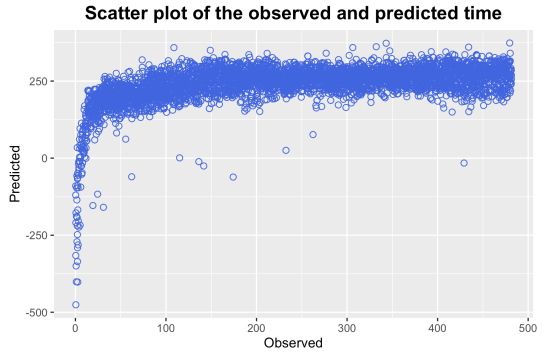
(b) Cropped red



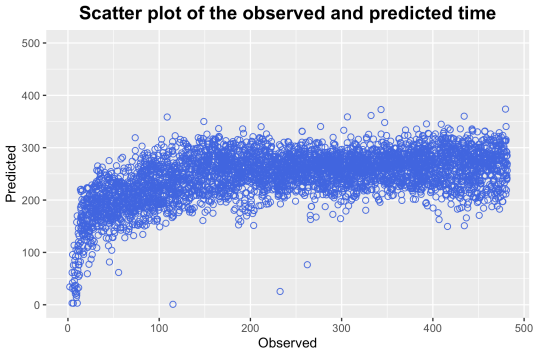
(c) Blue



(d) Cropped blue



(e) Red and blue



(f) Cropped red and blue

Figure 5.6: Plots of the observed and predicted values for various colour channels

Cross validation was also performed to compare the variables for the complete bloodstain and the inner bloodstain. In Appendix B this has been performed for all variables from the complete bloodstain and all from the inner bloodstain, but due to the high MSE this shall not be discussed in this section. Below in Figure 5.3, however, more of an effect can be seen when the complete and inner bloodstain are observed separately for the red colour channel. In this case, only taking the red colour data of the complete bloodstain, significantly worsens the MSE of the BN. Whereas, only taking the data from the inner bloodstain worsens the MSE only slightly. However, in both cases the MSE of the BNs are worse than the BN for the complete red colour data set. This could indicate that there are arcs between the variables from the complete and inner bloodstain, that have a significant influence on the predictions.

Variables	Mean Squared Error (MSE)	Standard deviation of MSE
Red complete variables	18031.04	2.27
Red inner variables	15353.98	11.67

Table 5.3: Table of the MSE and the standard deviation of the MSE obtained from the cross validation of the BNs constructed for each colour channel

In Appendix B, the MSE can be found for other sets of variables. These BNs shall not be elaborated on in this Section, due to high value of the MSE.

5.2. Optimal Bayesian Network

When taking a look at the MSE of all the BNs in Appendix B, it seems that both the BN consisting of only the red variables and the BN consisting of the red and the blue variables are closely matched. As stated in the previous section both the BNs share the following flaws: the unrealistic arcs within the BN and the negative predictions. The BNs differ on two things, i.e., the complexity of the BN itself and also on the trend that the predictions have. The complexity of the model is related to the number of nodes and arcs. The red variable BN has significantly less arcs, than the other BN and is therefore simpler to use, analyse, explain etc. However, the run time for these BNs is rather short so based on that the complexity should not be too much of a hindrance. The predictions can be compared via the plots. On comparison of the scatter plots in Figure 5.3 it seems that there is a trade-off between the accuracy in the early predictions and the later predictions. For early predictions there are still other methods that could be used, so we choose for accuracy in the later predictions.

Therefore the BN constructed using the red variables shall be taken as the optimal BN. In Appendix D.2.1 the code can be found, which constructs the BN for the red variables, and also fits it using the whole data set.

6

Conclusion

This thesis has the goal of analysing the aging of bloodstains through colour analysis, and constructing a Bayesian network which is be able to accurately make predictions on a bloodstain's age. The most optimal Bayesian network in this thesis is created using the following variables of the R colour channel for both the complete and inner bloodstain: mean, min, max, variance, and the 5%, 20%, 50%, 80% and 95% quantiles (see Section 5.2). However, the BNs produced within this thesis, including the optimal BN, are not able to make accurate predictions and have very high Mean Squared Errors (see Table B.1), the lowest Mean Squared Error being 14866.17. Regardless, throughout the process various observations have been made and explained, which could justify the poor results. The model constructed using the red variables has been chosen as the optimal BN. This BN has a slightly higher MSE, but this difference is rather small and this model seems to have slightly better predictions when making predictions over longer periods of time.

In Chapter 3 many observations have been made on the images and the data acquired from these images. First, for the images themselves inconsistencies have been detected. Within the plots there are numerous cases of outliers. The vast majority of these outliers are caused by overexposure or underexposure of the blood traces to light. Additionally, it can also be seen that the blood samples are not consistently centered within the images. This discrepancy in observed area of the blood sample is also reflected in the collected data. This can clearly be seen in the exploratory analysis, as for (almost) all variables, the values get split over the separate samples. Which leads to a large variance of the variable values between the samples.

In Chapter 5 the BNs are constructed and evaluated using the *bnlearn* package in R. The used method for the structure learning is the hill-climbing algorithm. After having constructed many BNs, the number of arcs can be confirmed to be rather large for each one of them. On inspection of the BNs with the highest MSE, it can even be seen that the predictions were almost constant. Apparently, even though the number of arcs is high, the influence of each arc on the *time* node is very small. However, even when the number of variables is decreased, the predictions are still rather poor and the MSE will still remain quite high.

7 Discussion

Although this research has not been able to produce a Bayesian network which can accurately determine the age of a blood trace, the steps that have been performed to obtain an end result are clear. The process could be split into two sections: image processing and Bayesian networks. For both parts, recommendations shall be given on how the results can be improved.

7.1. Image Processing

The process of converting an image to data consists of numerous steps and many decisions have been made based on personal preference. The most important observation from Chapter 3 is the importance of the observed area of a blood sample. The inconsistency of the observed area introduces very large variances to the data, which could (partially) be the cause for the BN's poor predictions. This problem would most likely need to be resolved when the images are being taken during the experiment. However, there are also other methods to mediate this problem. One possibility is to find an area of the bloodstain which overlaps for all samples and cropping the image. In this case, methods like edge detection or the convolution could be used. Finding the overlapping areas of the bloodstains could be rather time intensive, and as mentioned in Subsection 3.2.1 a considerable amount of data could be lost using this method.

The easiest, but also one of the most effective, methods of improving the predictions would be to analyse other variables or more variables. There are still many other colour systems which could be used to convert an image to some numeric value. It is definitely a possibility that a colour channel from a different system is more closely related to the time since deposition. Another possibility would be to further expand on the masking. In this thesis, the complete bloodstain and the inner bloodstain have been analysed, but the rim of the bloodstain could also contain information which could be relevant in the BN. It should also be noted that the inner bloodstain is very inconsistent in shape. It contains many holes, cracks and other irregularly shaped spaces. If possible, looking for a method to more consistently mask the inner area of the bloodstain could be interesting, and will most likely also be useful when masking the rim of the bloodstain.

There are also some other minor changes that could be made, however, I believe that these changes will give very little improvement. First, would be the removal of the outliers. In Section 3.4, it was already mentioned that the number of outliers is relatively small compared to the total number of images and that therefore the influence of these outliers should be minimal. However, improvement is improvement nonetheless, and there are many methods that could be used to remove these outliers efficiently. The easiest method would be to make use of the fact that the RGB value for the background is known. An implementation could be made in the Python code, so that the background is masked and the RGB value of the masked background can be compared to the known RGB value. Afterwards, any observations with a large deviation from the known background value can be removed immediately. Another minor change that could be made is the removal of the trend in the background which has also been observed in Section 3.4. This could be accomplished by fitting a line to the data of the control

group, and subtracting it from the blood sample data.

7.2. Bayesian Network

In Chapter 5, the Bayesian networks mainly focused on the nodes used in the BN. However, due to the large number of variables in each BN, the number of arcs is also by no means small. Determining the strengths of the arcs using bootstrapping, and further refining the BN could be an interesting topic for further research.

Alternatively, it is also a possibility to fit the BN with less and other variables. If less variables are used, perhaps a different structure learning method could be used to obtain a better BN. In this thesis, hill-climbing was used due to the time-intensity, but if the number of variables is less, then that should also decrease the run time. It is also worth constructing BNs for other variables which could perhaps have more rapid time-dependent behaviour.

And finally, the distributions of the variables themselves could be changed. *bnlearn* is limited to only discrete and Gaussian random variables, so perhaps switching to non-parametric BNs could improve the predictions.

A

Images and Plots

A.1. Images

A.1.1. Set-up

Below an image can be found on the used set-up for the experiment at LION. Further detail can be found in Beugelink [2].

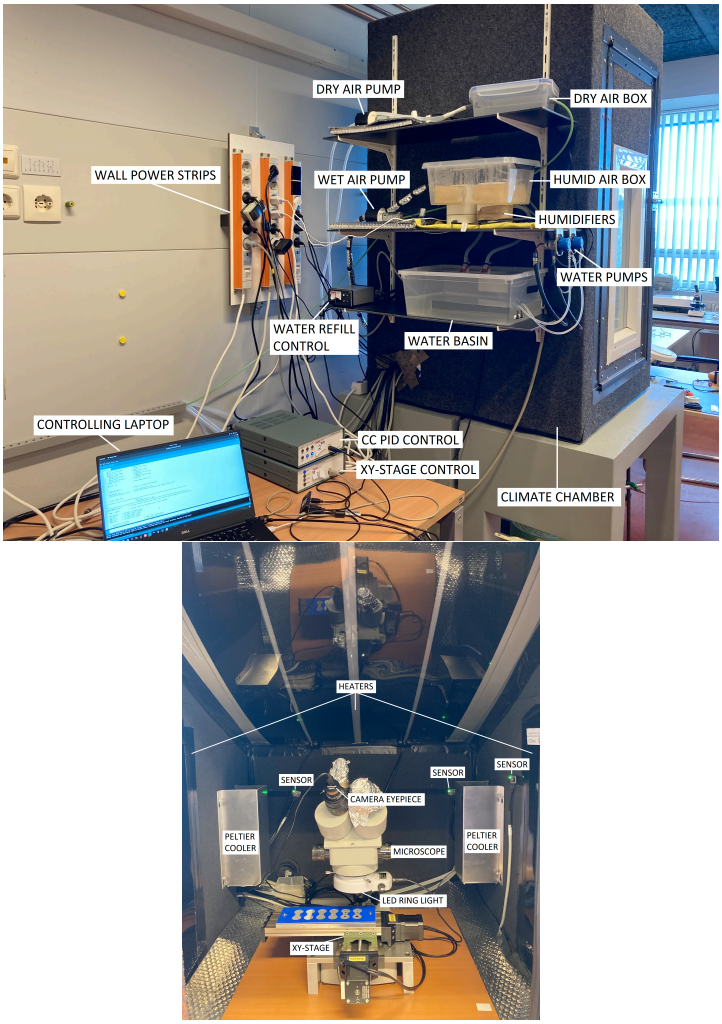
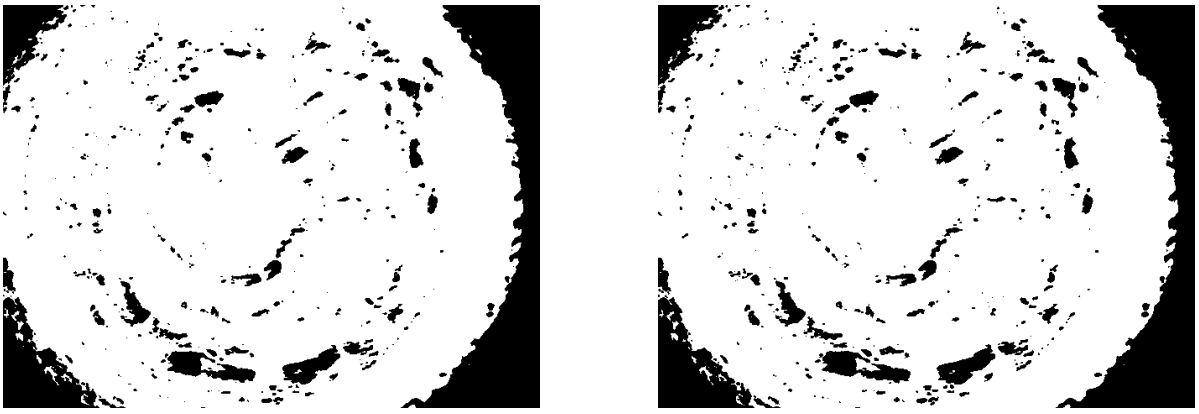


Figure A.1: Experiment set-up at LION [2]

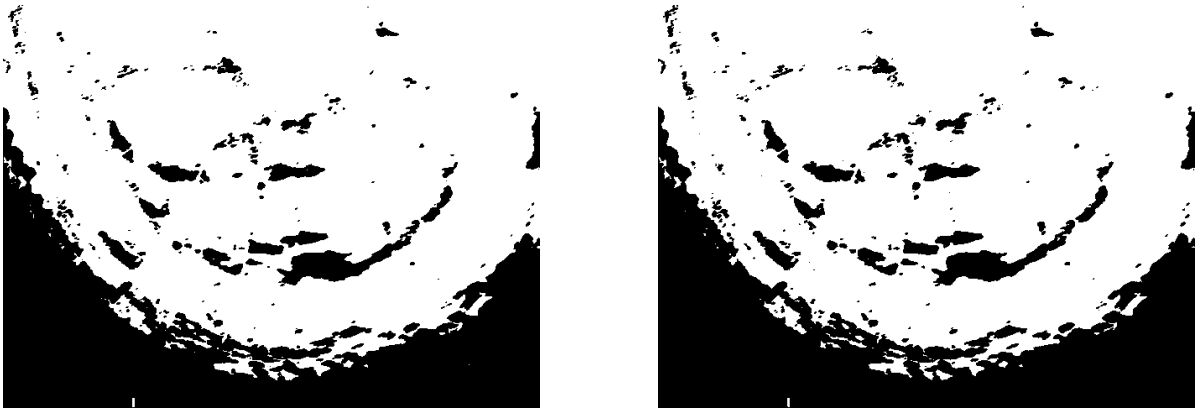
A.1.2. Coverage Masks

Below the 95% and 99% coverage masks can be found for a few of the samples. As in Subsection 3.2.2, the changes in the masks are minimal and therefore the number of greatly shifted bloodstains seem to be small. The code to determine the coverage mask can be found in Appendix C, and can be used to construct the masks for other samples and other coverage percentages.



(a) 95% coverage (b) 99% coverage

Figure A.2: Image of the masks of Sample 1 with 95% and 99% coverage



(a) 95% coverage (b) 99% coverage

Figure A.3: Image of the masks of Sample 2 with 95% and 99% coverage

(a) 95% coverage (b) 99% coverage

Figure A.4: Image of the masks of Sample 3 with 95% and 99% coverage

(a) 95% coverage (b) 99% coverage

Figure A.5: Image of the masks of Sample 4 with 95% and 99% coverage

A.1.3. Collected Data: Visual Representation

Below a visual representation is given of the various G- and B-values collected for both the complete and inner bloodstain of a single image.

(a) Minimum (b) Mean (c) 50% quantile (d) Maximum

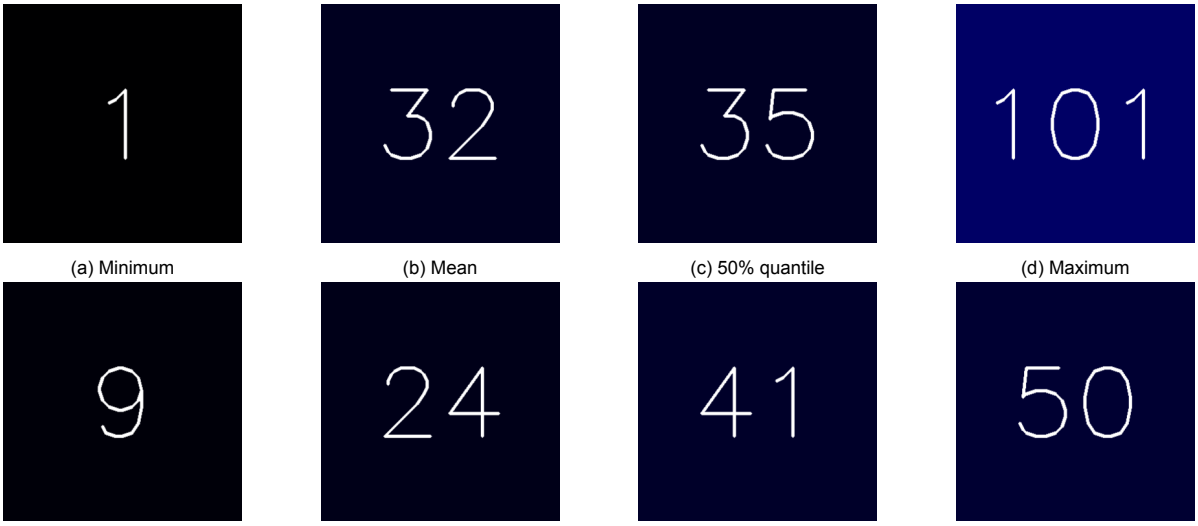
(e) 5% Quantile (f) 20% Quantile (g) 80% Quantile (h) 95% Quantile

Figure A.6: Visual representation of the data from the G values.
(Sample 1, 2023 May 9th, 10:47)

(a) Minimum (b) Mean (c) 50% quantile (d) Maximum

(e) 5% Quantile (f) 20% Quantile (g) 80% Quantile (h) 95% Quantile

Figure A.7: Visual representation of the inner bloodstain data from the G values.
(Sample 1, 2023 May 9th, 10:47)



(a) Minimum (b) Mean (c) 50% quantile (d) Maximum

(e) 5% Quantile (f) 20% Quantile (g) 80% Quantile (h) 95% Quantile

Figure A.8: Visual representation of the data from the B values. (Sample 1, 2023 May 9th, 10:47)

(a) Minimum (b) Mean (c) 50% quantile (d) Maximum

(e) 5% Quantile (f) 20% Quantile (g) 80% Quantile (h) 95% Quantile

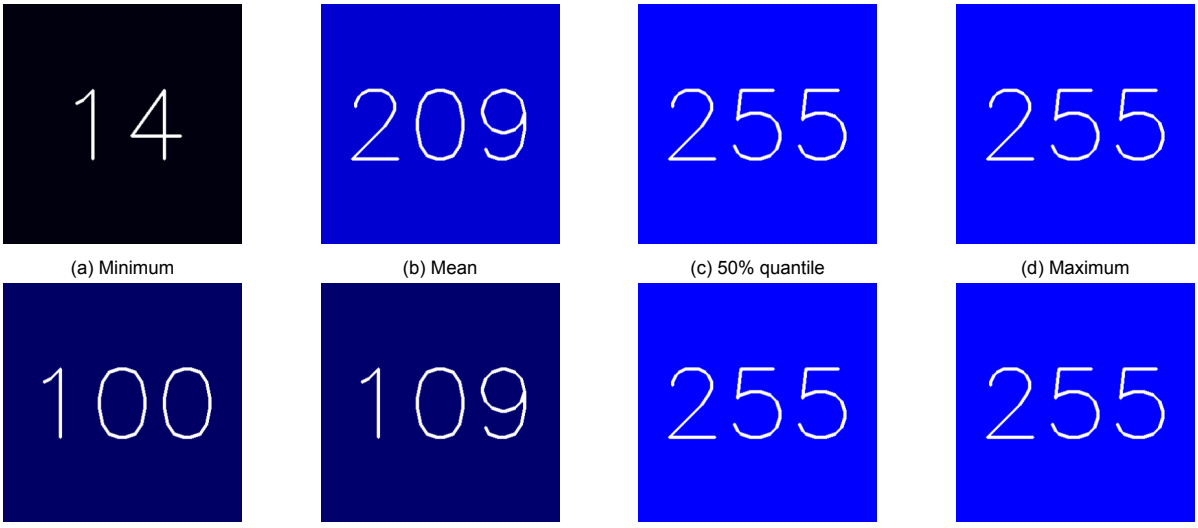
Figure A.9: Visual representation of the inner bloodstain data from the B values.
(Sample 1, 2023 May 9th, 10:47)

Below a visual representation is given of the G- and B-values collected for one of the control samples, i.e. Sample 7.

(a) Minimum (b) Mean (c) 50% quantile (d) Maximum

(e) 5% Quantile (f) 20% Quantile (g) 80% Quantile (h) 95% Quantile

Figure A.10: Visual representation of the data from the G values. (Sample 7)

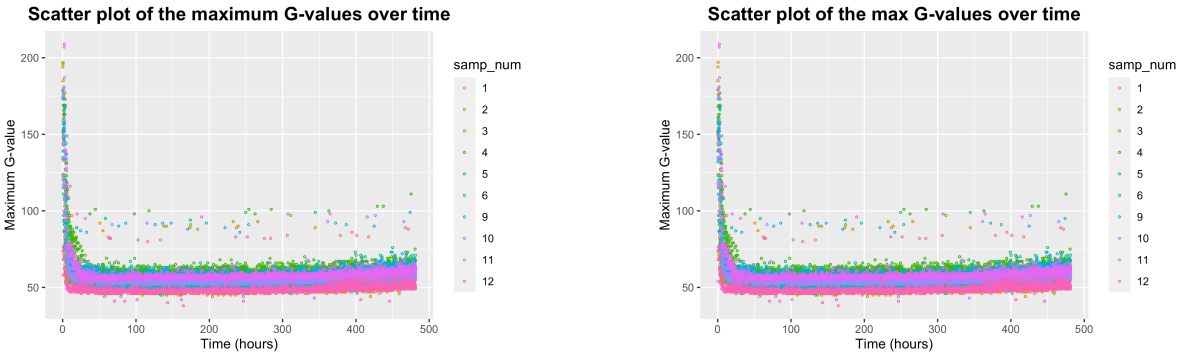


(a) Minimum (b) Mean (c) 50% quantile (d) Maximum

(e) 5% Quantile (f) 20% Quantile (g) 80% Quantile (h) 95% Quantile

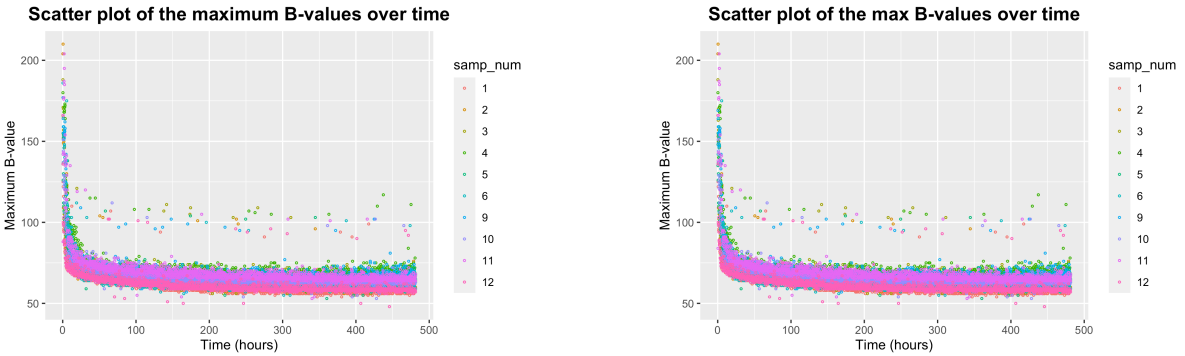
Figure A.11: Visual representation of the data from the B values. (Sample 7)

A.2. Plots



(a) Complete (b) Inner

Figure A.12: Scatter plot of the time and max G-values for the complete and inner bloodstain



(a) Complete (b) Inner

Figure A.13: Scatter plot of the time and max B-values for the complete and inner bloodstain

B

Bayesian Networks

In the table below, the MSE can be found for all the Bayesian networks constructed in the code. Not all these Bayesian networks were discussed in the main text, due to the poor results.

Variables	Mean Squared Error (MSE)	Standard deviation of MSE
All variables	19303.45	1.46
Gray variables	19303.48	1.99
Red variables	15149.81	1.99
Green variables	17335.29	11.16
Blue variables	17758.62	8.86
Red and blue variables	14866.17	2.16
RGB variables	19302.56	0.69
Complete bloodstain	19302.97	0.72
Inner bloodstain	19303.21	1.55
Means	19303.48	1.98
Maxima	19303.31	1.49
q95	19302.56	0.69
Means, maxima, q95	19303.89	1.36
Red complete variables	18031.04	2.27
Red inner variables	15353.98	11.67

Table B.1: Table of the various subsets of the data for which a Bayesian network has been constructed, the corresponding Mean Squared Error (MSE) and also the standard deviation of the MSE.

C

Python Code

In this chapter, the used Python code can be found. Python was used for the data collection process, and the illustration of the images, the corresponding masks, colours etc.

C.1. Functions

```
import os
import cv2
import numpy as np
from tkinter import filedialog
from datetime import datetime
from natsort import natsorted
import pandas as pd
import time
import matplotlib.pyplot as plt


def getdir():
    """
    Opens a dialog box to ask for the folder where data is stored.
    This allows the script to be ran multiple times if necessary.
    :return: Directory chosen by the user
    """
    directory = filedialog.askdirectory()
    return directory


def read(filename):
    """
    Reads an image file and converts it to arrays.
    :param filename: Filename of an image with the following format 'name.png'
    (jpg and other image formats also work)
    :return: Three arrays representing the RSV, RGB and grayscale images
    """
    img = cv2.imread(filename)  # 3 channels, RSV image
    RGB_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # 3 channels, RGB image
    GRAY_img = cv2.cvtColor(RGB_img, cv2.COLOR_RGB2GRAY)  # 1 channel, grayscale image
    return img, RGB_img, GRAY_img


def get_mask(GRAY_img, th):
    """
    Determines a mask for the grayscale image using binary thresholding and Otsu's
    binarization.
    :param GRAY_img: Array of the grayscale image
    :param th: Used threshold value.
    :return: Array representing the mask of the grayscale image (taking values 0 or 1)
    """
    ret, mask = cv2.threshold(GRAY_img, th, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)
    return mask / 255.0
```

```
44
45
46 def parse_filename(filename):
47     """
48     Splits the filename into the sample number and the timestamp of the image
49     :param filename: Filename of an image with the following format
50         "samp...-YYYYMMDDHHMM.png" (or .jpg)
51     :return: The sample number and the timestamp
52     """
53     # This function is written with the assumption that samp_num < 99
54     if filename[5] == "-":
55         samp_num = int(filename[4])
56         dt = filename[6:-4]
57     elif filename[6] == "-":
58         samp_num = int(filename[4:6])
59         dt = filename[7:-4]
60     else:
61         print("An image with incorrect formatting has been found \n Filename:", filename)
62         return "", ""
63     date = datetime.strptime(dt, "%Y%m%d%H%M")
64     return samp_num, date
65
66
67 def final_mask(directory, percentage):
68     """
69     Determines an overlapping mask for all images of each sample.
70     :param directory: Name of the directory in which the images are contained
71     :param percentage: Percentage of which an pixel should be present, to be included in the
       final mask,
72         i.e. a pixel should be contained in ...% of the images.
73     :return: Two dictionaries. In the first dictionary the overlapping mask is stored for
       each separate sample, in the
74         second the mask is stored for the "inner ring" of the sample. I.e. excluding the
       irregularities in the edges
75     """
76     # masks keeps track of how often a pixel is present over all the masks
77     masks = {}
78     inner_masks = {}
79     # count keeps track of how many images there are for each sample.
80     count = {}
81     n = 1
82     # This loop counts how often a pixel is within a mask
83     for file in natsorted(os.listdir(directory)):
84         filename = os.fsdecode(file)
85         if filename.endswith('jpg') or filename.endswith('png'):
86             # Collect the sample number, the time at which the picture was taken, and the
       image information
87             samp_num, date = parse_filename(filename)
88             img, RGB_img, GRAY_img = read(filename)
89
90             # Determine the mask for the given image
91             mask = get_mask(GRAY_img, 0)
92
93             mask_RGB = masked_image(RGB_img, mask)
94
95             # Construct the masked grayscale imager
96             GRAY_masked = (GRAY_img*mask).clip(0, 255).astype(np.uint8)
97
98             # Determine the mask of the masked image, to get rid of further irregular edges.
99             # So only the "inner ring" remains
100            inter_step1 = (get_mask(GRAY_masked, 0) * mask).clip(0, 1)
101            inter_step2 = ((np.ones(np.shape(GRAY_masked)) - get_mask(GRAY_masked, 0)) * mask
       ).clip(0, 1)
102            if np.sum(inter_step1) <= np.sum(inter_step2):
103                inner_mask = inter_step2
104                # plot_mask(inter_step2, f"Intermediate mask 2 (complement) of sample {
       samp_num}")
105            else:
106                inner_mask = inter_step1
107                # plot_mask(inter_step1, f"Intermediate mask 1 of sample {samp_num}")
108
```

```
            # if n <= 3:
            #     plot_mask(mask, f"Complete mask of sample {samp_num}")
            #     plot_mask(inner_mask, f"Inner mask of sample {samp_num}")
            #     mask_RGB = masked_image(RGB_img, mask)
            #     inner_RGB = masked_image(RGB_img, inner_mask)
            #     plot_image(mask_RGB, "")
            #     plot_image(inner_RGB, "")
            #     n += 1

            # If this sample is already in the masks dictionary, then add the masks together, and store the
            # resulting mask.
            # Otherwise create a new item in the masks dictionary
            if samp_num in masks:
                masks[samp_num] = masks[samp_num] + mask
                inner_masks[samp_num] = inner_masks[samp_num] + inner_mask
                count[samp_num] += 1
            else:
                masks[samp_num] = mask
                inner_masks[samp_num] = inner_mask
                count[samp_num] = 1

    # Determine which pixels are present in ...% of the masks
    for samp_num, mask in masks.items():
        masks[samp_num] = (mask > (count[samp_num]*percentage)) * np.ones(RGB_img.shape[:2])
        inner_masks[samp_num] = (inner_masks[samp_num] > (count[samp_num]*percentage)) * np.ones(RGB_img.shape[:2])
        # UNCOMMENT THE BELOW BLOCK TO SHOW THE OVERLAPPING MASK FOR EACH SAMPLE
        # plot_mask(masks[samp_num], f"Combined mask for sample {samp_num}")
        # plot_mask(inner_masks[samp_num], f"Combined mask of the inner ring for sample {samp_num}")
    return masks, inner_masks


def masked_image(RGB_img, mask):
    """
    Constructs the array of a masked RGB image
    :param RGB_img: Array of the RGB image
    :param mask: Binary array which masks the image
    :return: Array of masked RGB image
    """
    mask3D = np.zeros((RGB_img.shape[0], RGB_img.shape[1], 3))
    mask3D[:, :, 0] = mask
    mask3D[:, :, 1] = mask
    mask3D[:, :, 2] = mask
    masked = (RGB_img*mask3D).clip(0, 255).astype(np.uint8)
    for i in range(np.shape(RGB_img)[0]):
        for j in range(np.shape(masked)[1]):
            if (masked[i, j, ] == np.array([0, 0, 0])).all():
                masked[i, j, ] = np.array([255, 255, 255])
    return masked


def data_call(col_channel, num):
    """
    A function which determines whether the mean, minimum, maximum, variance or 5 quantiles of a colour channel
        are calculated (Values of the quantiles can be changed within the function. You need to take exactly 5
        quantiles)
    :param col_channel: Array of any colour channel
    :param num: A number from the set {1, 2, 3, 4, 5}, which determines what is calculated
    :return: Mean, minimum, maximum, variance or 5 quantiles
    """
    # We only consider the non-zero values in the colour channel
    array = col_channel[np.nonzero(col_channel)]
    if num == 0:
        return np.mean(array)
    elif num == 1:
        return np.amin(array)
    elif num == 2:
```

```
175         return np.amax(array)
176     elif num == 3:
177         return np.var(array)
178     elif num == 4:
179         # Change to the desired quantiles (Give exactly 5 quantiles)
180         quantiles = (0.05, 0.25, 0.5, 0.75, 0.95)
181         return np.quantile(array, quantiles)
182     else:
183         print("Wrong input given to data_call")
184         return
185
186
187 def channel_data(col_channel):
188     """
189     Constructs an array with the mean, minimum, maximum, variance and 5 quantiles for a
        single colour channel
190     :param col_channel: Array of any colour channel
191     :return: An array with the mean, minimum, maximum, variance and 5 quantiles
192     """
193     data = np.zeros(9)
194     for i in range(5):
195         if i in range(4):
196             data[i] = data_call(col_channel, i)
197         elif i == 4:
198             data[4:9] = data_call(col_channel, i)
199     return data
200
201
202 def image_data(RGB_image, GRAY_image):
203     """
204     Collects the mean, minimum, maximum, variance and 5 quantiles of the gray and RGB colour
        channels
205     :param RGB_image: Array of the RGB image
206     :param GRAY_image: Array of the grayscale image
207     :return: An array with the mean, minimum, maximum, variance and 5 quantiles of the
        grayscale image, and the
208         RGB image
209     """
210     GRAY = channel_data(GRAY_image)
211     R = channel_data(RGB_image[:, :, 0])
212     G = channel_data(RGB_image[:, :, 1])
213     B = channel_data(RGB_image[:, :, 2])
214     return np.array([np.concatenate((GRAY, R, G, B))])
215
216
217 def other_data(TOD, date, samp_num):
218     """
219     Collects the remaining data
220     :param TOD: Time of deposition
221     :param date: Time at which the image was taken
222     :param samp_num: Sample number of the image
223     :return: An array with the time since deposition, sample number, volume, temperature and
        humidity
224     """
225     # Determines the time since deposition in hours
226     TSD = (date - TOD).total_seconds() / 3600
227
228     # Determines volume, temperature and humidity
229     volume = samp_volume(samp_num)
230     temperature, humidity = read_temp_hum()
231     return np.array([[TSD, samp_num, volume, temperature, humidity]])
232
233
234 def samp_volume(samp_num):
235     """
236     Determines the volume of a sample. If an unknown sample number is given, then we assume
        that the volume is 0
237     :param samp_num: Sample number
238     :return: The volume in micro litre corresponding to the given sample
239     """
240     if samp_num in [1, 2, 3]:
```

```
241         return 8
242     elif samp_num in [4, 5, 6]:
243         return 4
244     elif samp_num in [9, 10, 11, 12]:
245         return 2
246     else:
247         return 0
248 
249 
250 def read_temp_hum():
251     """
252     CURRENTLY HARD CODED TO RETURN THE TEMPERATURE AND PERCENTAGE. WOULD NEED FURTHER CODING,
         TO READ THE DATA
253     FROM THE pid.log FILE
254     :return: The temperature in degrees Celsius and the percentage of humidity
255     """
256     return 25, 35
257 
258 
259 def batch_process(directory, time_dep):
260     """
261     Constructs an array containing the mean, minimum, maximum, variance and 5 quantiles for
        each channel and
262     for each image in the directory.
263     :param directory: Directory in which the to-be-processed images are contained
264     :param time_dep: Time of deposition
265     :return: An array containing the mean, minimum, maximum, variance and 5 quantiles
266     """
267     TOD = datetime.strptime(time_dep, "%Y-%m-%d %H:%M")
268 
269     # Construct the headers for the data array
270     ot_data = np.array([["time", "samp_num", "volume", "temperature", "humidity"]])
271     data = np.array([["GRAY_mean", "GRAY_min", "GRAY_max", "GRAY_variance", "GRAY_q5", "
        GRAY_q20", "GRAY_q50",
272                       "GRAY_q80", "GRAY_q95",
273                       "R_mean", "R_min", "R_max", "R_variance", "R_q5", "R_q20", "R_q50", "
        R_q80", "R_q95",
274                       "G_mean", "G_min", "G_max", "G_variance", "G_q5", "G_q20", "G_q50", "
        G_q80", "G_q95",
275                       "B_mean", "B_min", "B_max", "B_variance", "B_q5", "B_q20", "B_q50", "
        B_q80", "B_q95"]])
276     inner_data = np.array([["in_GRAY_mean", "in_GRAY_min", "in_GRAY_max", "in_GRAY_variance",
         "in_GRAY_q5",
277                             "in_GRAY_q20", "in_GRAY_q50", "in_GRAY_q80", "in_GRAY_q95",
278                             "in_R_mean", "in_R_min", "in_R_max", "in_R_variance", "in_R_q5",
        "in_R_q20", "in_R_q50",
279                             "in_R_q80", "in_R_q95",
280                             "in_G_mean", "in_G_min", "in_G_max", "in_G_variance", "in_G_q5",
        "in_G_q20", "in_G_q50",
281                             "in_G_q80", "in_G_q95",
282                             "in_B_mean", "in_B_min", "in_B_max", "in_B_variance", "in_B_q5",
        "in_B_q20", "in_B_q50",
283                             "in_B_q80", "in_B_q95"]])
284     # Get the overlapping masks
285     masks, inner_masks = final_mask(directory, 0.95)
286     mask_path = directory + "/mask_images"
287 
288     for samp_num in masks:
289         save_mask(mask_path, masks[samp_num], samp_num, 95)
290 
291     # n is used so that we only plot a limited number of images
292     n = 1
293     for file in natsorted(os.listdir(directory)):
294         filename = os.fsdecode(file)
295         # print(f"Filename: {filename}")
296         if filename.endswith('jpg') or filename.endswith('png'):
297             # Read the image, sample number and time at which the image was taken
298             samp_num, date = parse_filename(filename)
299             img, RGB_img, GRAY_img = read(filename)
300 
301             # Determine corresponding mask for the image
```

```
            mask = masks[samp_num]
            inner_mask = inner_masks[samp_num]

            # Construct the masked images
            GRAY_masked = (GRAY_img*mask).clip(0, 255).astype(np.uint8)
            RGB_masked = masked_image(RGB_img, mask)

            inner_GRAY_masked = (GRAY_img*inner_mask).clip(0, 255).astype(np.uint8)
            inner_RGB_masked = masked_image(RGB_img, inner_mask)

            # UNCOMMENT BELOW TO PLOT THE FIRST ... IMAGES. UNMASKED, MASKED AND DOUBLE MASKED
            # if n <= 3:
            #     plot_mask(mask, f"Mask for sample {samp_num}")
            #     plot_image(RGB_img, f"Unmasked image of sample {samp_num}, timestamp {date}")
            #     plot_image(RGB_masked, f"Masked image of sample {samp_num}, timestamp {date}")
            #     plot_image(inner_RGB_masked, f"Double masked image of sample {samp_num}, timestamp {date}")
            #     n += 1

            # Construct the image data and other data, and append it to their respective arrays
            other = other_data(TOD, date, samp_num)
            ot_data = np.concatenate((ot_data, other))
            image = image_data(RGB_masked, GRAY_masked)
            inner = image_data(inner_RGB_masked, inner_GRAY_masked)
            data = np.concatenate((data, image))
            inner_data = np.concatenate((inner_data, inner))
    return np.concatenate((ot_data, data, inner_data), axis=1)


def save_array_as_csv(path, filename, array):
    """
    Converts an array to a Pandas dataframe, and saves it as a CSV file
    :param path: Path to folder in which the dataframe is saved
    :param filename: Chosen name for the CSV file
    :param array: Numpy array that will be saved
    :return: Pandas dataframe
    """
    # Determine current working directory for later
    og_path = os.getcwd()

    # If the desired map does not exist, then it is created
    if not os.path.exists(path):
        os.mkdir(path)

    os.chdir(path)

    # Convert the array to a pandas data frame and proceed to store it in a csv file
    DF = pd.DataFrame(data=array[1:, ], columns=array[0, ])
    DF.to_csv(filename, index=False)

    # Return to original working directory
    os.chdir(og_path)
    return DF


def plot_mask(mask, title):
    """
    Plots the mask
    :param mask: Binary array of the mask
    :param title: Title of the plot
    :return: -
    """
    mask3D = np.zeros((mask.shape[0], mask.shape[1], 3))
    mask3D[:, :, 0] = mask * 255.0
    mask3D[:, :, 1] = mask * 255.0
    mask3D[:, :, 2] = mask * 255.0
    m = mask3D.clip(0, 255).astype(np.uint8)
```

```
    plt.imshow(m)
    plt.title(title)
    plt.show()
    return


def plot_image(RGB_img, title):
    """
    Plots an RGB image
    :param RGB_img: Array of the RGB image
    :param title: Title of the plot
    :return: -
    """
    plt.imshow(RGB_img)
    plt.title(title)
    plt.show()
    return


def save_mask(path, mask, samp_num, percentage):
    """
    Saves a mask as an image
    :param path: Location at which the image will be stored
    :param mask: Binary array of a mask
    :param samp_num: Sample number
    :param percentage: Coverage percentage of the mask
    :return: -
    """
    # Determine current working directory for later
    og_path = os.getcwd()

    # If the desired map does not exist, then it is created
    if not os.path.exists(path):
        os.mkdir(path)

    os.chdir(path)

    mask3D = np.zeros((mask.shape[0], mask.shape[1], 3))
    mask3D[:, :, 0] = mask * 255.0
    mask3D[:, :, 1] = mask * 255.0
    mask3D[:, :, 2] = mask * 255.0
    m = mask3D.clip(0, 255).astype(np.uint8)

    cv2.imwrite(f"mask_samp{samp_num}_perc{percentage}.png", m)
    os.chdir(og_path)
    return


def plot_and_save_colour(array, save_name):
    """
    Plots and saves a given colour value
    :param array: Array of shape (1,1,3), i.e. an array containing only 1 pixel,
    :param save_name: Name that the image should be saves as
    :return: -
    """
    img = np.zeros((369, 369, 3), np.uint8)
    for i in range(3):
        if array[0, 0, i] == 0:
            continue
        else:
            col = array[0, 0, i]
            img[:, :, i] = img[:, :, i] + col
            break
    ### Setup so that the colour value is printed
    # General setup
    font = cv2.FONT_HERSHEY_SIMPLEX
    fontScale = 5
    text_col = (255, 255, 255)
    thickness = 3

    # Getting the boundary of the text
```

```
    textsize = cv2.getTextSize(str(col), font, fontScale, thickness)[0]
    # Get coordinates based on boundary
    textX = int((img.shape[1] - textsize[0]) / 2)
    textY = int((img.shape[0] + textsize[1]) / 2)

    img = cv2.putText(img, str(col), (textX, textY), font, fontScale, text_col, thickness)

    plt.imshow(img)
    plt.axis('off')
    plt.savefig(save_name, transparent=True, bbox_inches='tight', pad_inches=0)
    plt.show()
    return


def plot_and_save_mask(mask, save_name):
    mask3D = np.zeros((mask.shape[0], mask.shape[1], 3))
    mask3D[:, :, 0] = mask * 255.0
    mask3D[:, :, 1] = mask * 255.0
    mask3D[:, :, 2] = mask * 255.0
    m = mask3D.clip(0, 255).astype(np.uint8)
    plt.imshow(m)
    plt.axis('off')
    plt.savefig(save_name, transparent=True, bbox_inches='tight', pad_inches=0)
    plt.show()
    return


def plot_and_save_image(RGB_img, save_name):
    plt.imshow(RGB_img)
    plt.axis('off')
    plt.savefig(save_name, transparent=True, bbox_inches='tight', pad_inches=0)
    plt.show()
    return


def colour_parser(directory, path, masks, inner_masks):
    """

    :param directory: Directory in which the to-be-processed images are contained
    :param path: Path to directory in which the produced images are saved
    :param masks:
    :return:
    """
    colours = ["R", "G", "B"]
    types = ["_mean", "_min", "_max", "_q5", "_q20", "_q50", "_q80", "_q95"]

    for file in natsorted(os.listdir(directory)):
        filename = os.fsdecode(file)
        # print(f"Filename: {filename}")
        if filename.endswith('jpg') or filename.endswith('png'):
            # Read the image, sample number and time at which the image was taken
            samp_num, date = parse_filename(filename)
            img, RGB_img, GRAY_img = read(filename)

            # Determine corresponding mask for the image
            mask = masks[samp_num]
            inner_mask = inner_masks[samp_num]

            # Construct the masked images
            GRAY_masked = (GRAY_img*mask).clip(0, 255).astype(np.uint8)
            RGB_masked = masked_image(RGB_img, mask)

            inner_GRAY_masked = (GRAY_img*inner_mask).clip(0, 255).astype(np.uint8)
            inner_RGB_masked = masked_image(RGB_img, inner_mask)

            # Get the colour data from the image (mean, min, max, quantiles) (also variance,
    but we don't need it)
            # Get the colour data, and remove the GRAY, and variance of each colour channels
            image = image_data(RGB_masked, GRAY_masked)[0]
            image = np.delete(image, (range(9)))
            image = np.delete(image, (3, 12, 21))
```

```
509
510             inner = image_data(inner_RGB_masked, inner_GRAY_masked)[0]
511             inner = np.delete(inner, (range(9)))
512             inner = np.delete(inner, (3, 12, 21))
513
514             if not os.path.exists(path):
515                 os.mkdir(path)
516             os.chdir(path)
517
518             # Now we plot each value, and save it
519             for i in range(len(colours)):
520                 for j in range(len(types)):
521                     # Save the images of the colours for the full bloodstain
522                     filename = f"{colours[i]}{types[j]}_samp{samp_num}.png"
523                     col = np.zeros([1, 1, 3]).astype(np.uint8)
524                     col[0, 0, i] = int(round(image[i*8 + j]))
525                     plot_and_save_colour(col, filename)
526
527                     # Save the images of the colours for the inner bloodstain
528                     filename_inner = "in_" + filename
529                     col_inner = np.zeros([1, 1, 3]).astype(np.uint8)
530                     col_inner[0, 0, i] = int(round(inner[i*8 + j]))
531                     # # Save the colours that were obtained from the processing
532                     plot_and_save_colour(col_inner, filename_inner)
533             os.chdir(directory)
534     return
535
536
537 def mask_image_parser(directory, path, masks, inner_masks):
538     for file in natsorted(os.listdir(directory)):
539         filename = os.fsdecode(file)
540         # print(f"Filename: {filename}")
541         if filename.endswith('jpg') or filename.endswith('png'):
542             # Read the image, sample number and time at which the image was taken
543             samp_num, date = parse_filename(filename)
544             img, RGB_img, GRAY_img = read(filename)
545
546             # Determine corresponding mask for the image
547             mask = masks[samp_num]
548             inner_mask = inner_masks[samp_num]
549
550             # Construct the masked images
551             RGB_masked = masked_image(RGB_img, mask)
552             inner_RGB_masked = masked_image(RGB_img, inner_mask)
553
554             if not os.path.exists(path):
555                 os.mkdir(path)
556             os.chdir(path)
557
558             for tp in ["RGB", "mask"]:
559                 filename_complete = f"{tp}_full_samp{samp_num}.png"
560                 filename_masked = f"{tp}_mask_samp{samp_num}.png"
561                 filename_inner = f"{tp}_in_samp{samp_num}.png"
562                 if tp == "RGB":
563                     plot_and_save_image(RGB_img, filename_complete)
564                     plot_and_save_image(RGB_masked, filename_masked)
565                     plot_and_save_image(inner_RGB_masked, filename_inner)
566                 elif tp == "mask":
567                     plot_and_save_mask(mask, filename_masked)
568                     plot_and_save_mask(inner_mask, filename_inner)
569             os.chdir(directory)
570             return
```

Listing C.1: "functions.py" file

C.2. Data Collection

```
from functions import *


# Manually enter TOD, YYYY-MM-DD HH-MM
# Time of Deposition
TOD = '2023-05-02 10:36'       # Second donor measurement

directory = getdir()
data_path = directory + "/datafiles"

os.chdir(directory)
directory = os.getcwd()
print(f"Directory: {directory}")

# Collect all the data for each image
data_array = batch_process(directory, TOD)
dataframe = save_array_as_csv(data_path, "data2.csv", data_array)
```

C.3. Images

```
from functions import *

# Manually enter TOD, YYYY-MM-DD HH-MM
# Time of Deposition
TOD = '2023-05-02 10:36'       # Second donor measurement

directory = getdir()

os.chdir(directory)
directory = os.getcwd()
print(f"Directory: {directory}")

masks, inner_masks = final_mask(directory, 0.95)

colour_path = directory + "/colours"
colour_parser(directory, colour_path, masks, inner_masks)

image_mask_path = directory + "/masks_images"
mask_image_parser(directory, image_mask_path, masks, inner_masks)
```

D

R Code

Throughout this report, the plotting of the data, and the construction of the Bayesian networks was performed in RStudio. In this chapter, all the used R-code can be found. Whenever running any of the code, first run the code in Appendix D.1. Afterwards, the code in Appendix D.2 and D.2.1 can be run. Do keep in mind that the path in the code should be set to the desired directory.

D.1. Functions

```
##### Defining functions #####
histogram_plot <- function(df, var, num_bins, labs, save, filename) {
  # Plot a histogram
  # df: data frame
  # var: variable that will be plotted
  # num_bins: number of bin values for the histogram
  # labs: vector containing the name for the x-axis and the title of
  #       the histogram
  # save: save the image (TRUE or FALSE)
  # filename: name that the image should be saved as
  name <- as.character(substitute(var))
  v_hist_inf <- c(width = (max(df[name])-min(df[name]))/num_bins,
                  bound = min(df[name]))
  p <- ggplot(df, aes({{var}})) +
    geom_histogram(binwidth = v_hist_inf[['width']],
                   boundary = v_hist_inf[['bound']],
                   color="royalblue4", fill="royalblue") +
    ggtitle(labs[2]) + xlab(labs[1]) +
    theme(plot.title = element_text(size = 15))
  if (save == TRUE) {
    ggsave(filename, plot = p, width = 6, height = 4, units = "in")
  }
  p
}

scatter_plot <- function(df, var1, var2, labs, save, filename){
  # Create a scatter plot
  # df: data frame
  # var1: first variable that will be plotted
  # var2: second variable that will be plotted
  # labs: vector containing the name for the x-axis and the y-axis
  #       and the title of the plot
  # save: save the image (TRUE or FALSE)
  # filename: name that the image should be saved as
  p =   ggplot(df, aes({{var1}}, {{var2}})) +
    geom_point(size=2, color="royalblue", shape = 1) +
    ggtitle(labs[3]) + xlab(labs[1]) + ylab(labs[2]) +
    theme(plot.title = element_text(size = 15, face="bold", hjust=0.5))
  if (save == TRUE) {
    ggsave(filename, plot = p, width = 6, height = 4, units = "in")
  }
```

```
  p
}

grouped_scatter_plot <- function(df, var1, var2, var3, title, save, filename){
  # Create a scatter plot grouped over a certain variable
  # df: data frame
  # var1: first variable that will be plotted
  # var2: second variable that will be plotted
  # var3: third variable over which we group
  # labs: vector containing the name for the x-axis and the y-axis
  #       and the title of the plot
  # save: save the image (TRUE or FALSE)
  # filename: name that the image should be saved as
  p = ggplot(df, aes({{var1}}, {{var2}}, color={{var3}})) +
    geom_point(size = 0.5, shape = 1) +
    ggtitle(labs[3]) + xlab(labs[1]) + ylab(labs[2]) +
    theme(plot.title = element_text(size = 15, face="bold", hjust=0.5))
  if (save == TRUE) {
    ggsave(filename, plot = p, width = 6, height = 4, units = "in")
  }
  p
}

obs_pred_plot <- function(obs, pred, title, save, filename){
  # Create a scatter plot with the observed values on the x-axis and
  # the predicted values on the y-axis.
  # obs: vector of the observed values
  # pred: vector of the predicted values
  # title: title of the plot
  # save: save the image (TRUE or FALSE)
  # filename: name that the image should be saved as
  p = ggplot() + geom_point(aes(obs, pred), size=2, color="royalblue",
                            shape=1 ) + ggtitle(title) +
    xlab("Observed") + ylab("Predicted") +
    theme(plot.title = element_text(size = 15, face="bold", hjust=0.5))
  if (save == TRUE) {
    ggsave(filename, plot = p, width = 6, height = 4, units = "in")
  }
  p
}
```

D.2. Bayesian Networks

```
##### Set your working directory with all files, and import packages #####
path = "~/TU Delft/Bachelor Project/R code"
setwd(path)
set.seed(12345)
library(bnlearn)
library(ggplot2)
library(dplyr)

##### Importing and Tidying Data #####
data = read.csv("datafiles/data.csv", header = TRUE)
# Remove control samples
data = data[which(data$samp_num != 7 & data$samp_num != 8),]
# Remove irrelevant columns from data set. I.e. temperature and humidity
# and sample number
data = select(data, -c(temperature, humidity, samp_num))

# Determining the structure may take long, you can also load the
# provided .RData file
load("bn_structure_data.RData")

##### Bayesian Networks #####
##### Whole data set
# Determine the structure of the Bayesian network using hill-climbing
# bn.hc_data = hc(data)

# Cross-validate the structure of the BN. This is done through k-fold cross-
```

```
# validation using hte MSE as the loss function
# The cross-validation is done 5 times
cv_data = bn.cv(data, bn.hc_data, method = "k-fold", k=5, loss = "mse",
                loss.args = list(target = "time"), runs = 5)
cv_data

# Plot the structure of the Bayesian network
plot(bn.hc_data)

# Print the children and parent nodes of the time node
bn.hc_data$nodes$time$children
bn.hc_data$nodes$time$parents

# Plot the observed and predicted time values obtained using the BN
obs_pred_plot(cv_data[[1]][[1]]$observed, cv_data[[1]][[1]]$predicted,
              "Scatter plot of the observed and predicted time", FALSE,
              "obs_pred_all_data.png")

##### Gray variables
gray = data[, c(1:2, 3:11, 39:47)]
# bn.hc_gray = hc(gray)
cv_gray = bn.cv(gray, bn.hc_gray, method = "k-fold", k=5, loss = "mse",
                loss.args = list(target = "time"), runs = 5)
cv_gray
plot(bn.hc_gray)
bn.hc_gray$nodes$time$children
bn.hc_gray$nodes$time$parents
obs_pred_plot(cv_gray[[1]][[1]]$observed, cv_gray[[1]][[1]]$predicted,
              "Scatter plot of the observed and predicted time", FALSE,
              "obs_pred_gray.png")

##### Red variables
red = data[, c(1:2, 12:20, 48:56)]
# bn.hc_red = hc(red)
cv_red = bn.cv(red, bn.hc_red, method = "k-fold", k=5, loss = "mse",
               loss.args = list(target = "time"), runs = 5)
cv_red
plot(bn.hc_red)
bn.hc_red$nodes$time$children
bn.hc_red$nodes$time$parents
obs_pred_plot(cv_red[[1]][[1]]$observed, cv_red[[1]][[1]]$predicted,
              "Scatter plot of the observed and predicted time", FALSE,
              "obs_pred_red.png")

##### Green variables
green = data[, c(1:2, 21:29, 57:65)]
# bn.hc_green = hc(green)
cv_green = bn.cv(green, bn.hc_green, method = "k-fold", k=5, loss = "mse",
                 loss.args = list(target = "time"), runs = 5)
cv_green
plot(bn.hc_green)
bn.hc_green$nodes$time$children
bn.hc_green$nodes$time$parents
obs_pred_plot(cv_green[[1]][[1]]$observed, cv_green[[1]][[1]]$predicted,
              "Scatter plot of the observed and predicted time", FALSE,
              "obs_pred_green.png")

##### Blue variables
blue = data[, c(1:2, 30:38, 66:74)]
# bn.hc_blue = hc(blue)
cv_blue = bn.cv(blue, bn.hc_blue, method = "k-fold", k=5, loss = "mse",
                loss.args = list(target = "time"), runs = 5)
cv_blue
plot(bn.hc_blue)
bn.hc_blue$nodes$time$children
bn.hc_blue$nodes$time$parents
p = obs_pred_plot(cv_blue[[1]][[1]]$observed, cv_blue[[1]][[1]]$predicted,
              "Scatter plot of the observed and predicted time", FALSE,
              "obs_pred_blue.png") + ylim(0,500)
ggsave("zoom_obs_pred_blue.png", plot = p, width = 6, height = 4, units = "in")

```

```

##### Red and blue variables
red_and_blue = data[, c(1:2, 12:20, 30:38, 48:56, 66:74)]
# bn.hc_red_and_blue = hc(red_and_blue)
cv_red_and_blue = bn.cv(red_and_blue, bn.hc_red_and_blue, method = "k-fold",
                        k=5, loss = "mse", loss.args = list(target = "time"),
                        runs = 5)
cv_red_and_blue
plot(bn.hc_red_and_blue)
bn.hc_red_and_blue$nodes$time$children
bn.hc_red_and_blue$nodes$time$parents
obs_pred_plot(cv_red_and_blue[[1]][[1]]$observed,
              cv_red_and_blue[[1]][[1]]$predicted,
              "Scatter plot of the observed and predicted time", FALSE,
              "obs_pred_red_and_blue.png")

##### Red and green variables
red_and_green = data[, c(1:2, 12:29, 48:65)]
# bn.hc_red_and_green = hc(red_and_green)
cv_red_and_green = bn.cv(red_and_green, bn.hc_red_and_green, method = "k-fold",
                        k=5, loss = "mse", loss.args = list(target = "time"),
                        runs = 5)
cv_red_and_green
plot(bn.hc_red_and_green)
bn.hc_red_and_green$nodes$time$children
bn.hc_red_and_green$nodes$time$parents
obs_pred_plot(cv_red_and_green[[1]][[1]]$observed,
              cv_red_and_green[[1]][[1]]$predicted,
              "Scatter plot of the observed and predicted time", FALSE,
              "obs_pred_red_and_green.png") + ylim(0,500)

##### Red, green and blue variables
RGB = data[, c(1:2, 12:38, 48:74)]
# bn.hc_RGB = hc(RGB)
cv_RGB = bn.cv(RGB, bn.hc_RGB, method = "k-fold",
                        k=5, loss = "mse", loss.args = list(target = "time"),
                        runs = 5)
cv_RGB
plot(bn.hc_RGB)
bn.hc_RGB$nodes$time$children
bn.hc_RGB$nodes$time$parents
obs_pred_plot(cv_RGB[[1]][[1]]$observed,
              cv_RGB[[1]][[1]]$predicted,
              "Scatter plot of the observed and predicted time", FALSE,
              "obs_pred_red_and_blue.png") + ylim(0,500)

##### Complete bloodstain
complete = data[, c(1:2, 3:38)]
# bn.hc_complete = hc(complete)
cv_complete = bn.cv(complete, bn.hc_complete, method = "k-fold", k=5,
                    loss = "mse", loss.args = list(target = "time"), runs = 5)
cv_complete

##### Inner bloodstain
inner = data[, c(1:2, 39:74)]
bn.hc_inner = hc(inner)
cv_inner = bn.cv(inner, bn.hc_inner, method = "k-fold", k=5, loss = "mse",
                 loss.args = list(target = "time"), runs = 5)
cv_inner

# Red colour channel of the complete bloodstain
red_complete = data[, c(1:2, 12:20)]
bn.hc_red_complete = hc(red_complete)
cv_red_complete = bn.cv(red_complete, bn.hc_red_complete, method = "k-fold",
                        k=5, loss = "mse",
                        loss.args = list(target = "time"), runs = 5)
cv_red_complete


```

```
# Red colour channel of the inner bloodstain
red_inner = data[, c(1:2, 48:56)]
bn.hc_red_inner = hc(red_inner)
cv_red_inner = bn.cv(red_inner, bn.hc_red_inner, method = "k-fold", k=5,
                     loss = "mse", loss.args = list(target = "time"), runs = 5)
cv_red_inner



means = data[, c(1:2, 3, 12, 21, 30, 39, 48, 57, 66)]
maxs = data[, c(1:2, 5, 14, 23, 32, 41, 50, 59, 68)]
q95s = data[, c(1:2, 11, 20, 29, 38, 47, 56, 65, 74)]


##### Mean variables
means = data[, c(1:2, 3, 12, 21, 30, 39, 48, 57, 66)]
bn.hc_means = hc(means)
cv_means = bn.cv(means, bn.hc_means, method = "k-fold", k=5, loss = "mse",
                 loss.args = list(target = "time"), runs = 5)
cv_means

##### Max variables
maxs = data[, c(1:2, 5, 14, 23, 32, 41, 50, 59, 68)]
bn.hc_maxs = hc(maxs)
cv_maxs = bn.cv(maxs, bn.hc_maxs, method = "k-fold", k=5, loss = "mse",
                 loss.args = list(target = "time"), runs = 5)
cv_maxs

##### q95s variables
q95s = data[, c(1:2, 11, 20, 29, 38, 47, 56, 65, 74)]
bn.hc_q95s = hc(q95s)
cv_q95s = bn.cv(q95s, bn.hc_q95s, method = "k-fold", k=5, loss = "mse",
                loss.args = list(target = "time"), runs = 5)
cv_q95s

##### Means, maxs and q95s variables
mmq = data[c(1:2, 3, 12, 21, 30, 39, 48, 57, 66,
             5, 14, 23, 32, 41, 50, 59, 68,
             11, 20, 29, 38, 47, 56, 65, 74)]
bn.hc_mmq = hc(mmq)
cv_mmq = bn.cv(mmq, bn.hc_mmq, method = "k-fold", k=5, loss = "mse",
                loss.args = list(target = "time"), runs = 5)
cv_mmq

# Uncomment the below to save certain constructed BN structures
# save(bn.hc_data, bn.hc_gray, bn.hc_red, bn.hc_green, bn.hc_blue, bn.hc_gray,
#      bn.hc_complete, bn.hc_inner, bn.hc_group, bn.hc_red_and_blue,
#      bn.hc_red_and_green, bn.hc_RGB,
#      file = "bn_structure_data.RData"))
```

D.2.1. Optimal Bayesian Network

```
##### Set your working directory with all files, and import packages #####
path = "~/TU Delft/Bachelor Project/R code"
setwd(path)
set.seed(12345)
library(bnlearn)
library(ggplot2)
library(dplyr)

##### Importing and Tidying Data #####
data = read.csv("datafiles/data.csv", header = TRUE)
# Remove control samples
data = data[which(data$samp_num != 7 & data$samp_num != 8),]
# Remove irrelevant columns from data set. I.e. temperature and humidity
# and sample number
data = select(data, -c(temperature, humidity, samp_num))

# Determining the structure may take long, you can also load the
# provided .RData file
```

```
load("bn_structure_data.RData")

##### Red variables
red = data[, c(1:2, 12:20, 48:56)]
# bn.hc_red = hc(red)

# Fit the Bayesian network with the all the observations
fit = bn.fit(bn.hc_red, red)
```

D.3. Plots

```
##### Set your working directory with all files, and import packages #####
path = "~/TU Delft/Bachelor Project/R code"
setwd(path)
library(bnlearn)
library(ggplot2)
library(dplyr)

##### Importing Data #####
plot_data <- read.csv("datafiles/data.csv", header = TRUE)
# Remove irrelevant columns from data set. I.e. temperature and humidity
plot_data = select(plot_data, -c(temperature, humidity))
# Turns the sample numbers into categorical variables
plot_data$samp_num <- sapply(plot_data$samp_num, as.factor)
plot_data$volume <- sapply(plot_data$volume, as.factor)
control_data = plot_data[which(plot_data$samp_num == 7 |
                                 plot_data$samp_num == 8), ]
plot_data = plot_data[which(plot_data$samp_num != 7 &
                              plot_data$samp_num != 8), ]


##### Outliers Sample 1 #####
sample_1 = plot_data[plot_data$samp_num==1, ]
without_outliers_1 = sample_1[sample_1$R_mean<= 45, ]
outliers_1 = sample_1[sample_1$R_mean>45, ]

scatter_plot(sample_1, time, R_q95,
             c("Time (hours)", "Mean R-value",
               " Scatter plot of the mean R-values over time"),
             FALSE, "samp_1.png")

scatter_plot(without_outliers_1, time, B_mean,
             c("Time (hours)", "Mean R-value",
               " Scatter plot of the mean R-values over time
               (without outliers)"), FALSE, "samp_1_without_outliers.png")

scatter_plot(outliers_1, time, R_mean,
             c("Time (hours)", "Mean R-value",
               " Scatter plot of the mean R-values over time
               (only outliers)"), FALSE, "samp_1_outliers.png")

##### Histograms #####
### R-values
# Complete bloodstain
histogram_plot(plot_data, R_mean, 30,
               c("Mean R-values", "Histogram of the mean R-values"), FALSE,
               "hist_R_mean.png")
histogram_plot(plot_data, R_min, 30,
               c("Minimum R-values", "Histogram of the minimum R-values"),
               FALSE, "hist_R_min.png")
histogram_plot(plot_data, R_max, 60,
               c("Maximum R-values", "Histogram of the maximum R-values"),
               FALSE, "hist_R_max.png")
histogram_plot(plot_data, R_variance, 30,
               c("Variance R-values",
                 "Histogram of the variance of the R-values"), FALSE,
               "hist_R_var.png")
histogram_plot(plot_data, R_q5, 30,
               c("5% quantile of the R-values",
```

```
                  "Histogram of the 5% quantile of the R-values"), FALSE,
                "hist_R_q5.png")
histogram_plot(plot_data, R_q20, 30,
                c("20% quantile of the R-values",
                  "Histogram of the 20% quantile of the R-values"), FALSE,
                "hist_R_q20.png")
histogram_plot(plot_data, R_q50, 30,
                c("50% quantile of the R-values",
                  "Histogram of the 50% quantile of the R-values"), FALSE,
                "hist_R_q50.png")
histogram_plot(plot_data, R_q80, 30,
                c("80% quantile of the R-values",
                  "Histogram of the 80% quantile of the R-values"), FALSE,
                "hist_R_q80.png")
histogram_plot(plot_data, R_q95, 50,
                c("95% quantile of the R-values",
                  "Histogram of the 95% quantile of the R-values"), FALSE,
                "hist_R_q95.png")
# Inner bloodstain
histogram_plot(plot_data, in_R_mean, 70,
                c("Mean R-values of the inner bloodstain",
                  "Histogram of the mean R-values
                  of the inner bloodstain"), FALSE, "hist_in_R_mean.png")
histogram_plot(plot_data, in_R_min, 40,
                c("Minimum R-values of the inner bloodstain",
                  "Histogram of the minimum R-values
                  of the inner bloodstain"), FALSE, "hist_in_R_min.png")
histogram_plot(plot_data, in_R_max, 60,
                c("Maximum R-values of the inner bloodstain",
                  "Histogram of the maximum R-values
                  of the inner bloodstain"), FALSE, "hist_in_R_max.png")
histogram_plot(plot_data, in_R_variance, 60,
                c("Variance R-values of the inner bloodstain",
                  "Histogram of the variance of the R-values
                  of the inner bloodstain"), FALSE, "hist_in_R_var.png")
histogram_plot(plot_data, in_R_q5, 50,
                c("5% quantile of the R-values of the inner bloodstain",
                  "Histogram of the 5% quantile of the R-values
                  of the inner bloodstain"), FALSE, "hist_in_R_q5.png")
histogram_plot(plot_data, in_R_q20, 50,
                c("20% quantile of the R-values of the inner bloodstain",
                  "Histogram of the 20% quantile of the R-values of the inner bloodstain"),
                FALSE, "hist_in_R_q20.png")
histogram_plot(plot_data, in_R_q50, 50,
                c("50% quantile of the R-values of the inner bloodstain",
                  "Histogram of the 50% quantile of the R-values
                  of the inner bloodstain"), FALSE, "hist_in_R_q50.png")
histogram_plot(plot_data, in_R_q80, 50,
                c("80% quantile of the R-values of the inner bloodstain",
                  "Histogram of the 80% quantile of the R-values
                  of the inner bloodstain"), FALSE, "hist_in_R_q80.png")
histogram_plot(plot_data, in_R_q95, 50,
                c("95% quantile of the R-values of the inner bloodstain",
                  "Histogram of the 95% quantile of the R-values
                  of the inner bloodstain"), FALSE, "hist_in_R_q95.png")

### G-values
# Complete bloodstain
histogram_plot(plot_data, G_mean, 30,
                c("Mean G-values", "Histogram of the mean G-values"),
                FALSE, "")
histogram_plot(plot_data, G_min, 30,
                c("Minimum G-values", "Histogram of the minimum G-values"),
                FALSE, "")
histogram_plot(plot_data, G_max, 80,
                c("Maximum G-values", "Histogram of the maximum G-values"),
                FALSE, "hist_G_max.png")
histogram_plot(plot_data, G_variance, 30,
                c("Variance G-values",
                  "Histogram of the variance of the G-values"), FALSE, "")
histogram_plot(plot_data, G_q5, 30,
```

```
                c("5% quantile of the G-values",
                  "Histogram of the 5% quantile of the G-values"), FALSE, "")
histogram_plot(plot_data, G_q20, 30,
                c("20% quantile of the G-values",
                  "Histogram of the 20% quantile of the G-values"), FALSE, "")
histogram_plot(plot_data, G_q50, 30,
                c("50% quantile of the G-values",
                  "Histogram of the 50% quantile of the G-values"), FALSE, "")
histogram_plot(plot_data, G_q80, 30,
                c("80% quantile of the G-values",
                  "Histogram of the 80% quantile of the G-values"), FALSE, "")
histogram_plot(plot_data, G_q95, 30,
                c("95% quantile of the G-values",
                  "Histogram of the 95% quantile of the G-values"), FALSE, "")
# Inner bloodstain
histogram_plot(plot_data, in_G_mean, 30,
                c("Mean G-values of the inner bloodstain",
                  "Histogram of the mean G-values
                  of the inner bloodstain"), FALSE, "")
histogram_plot(plot_data, in_G_min, 30,
                c("Minimum G-values of the inner bloodstain",
                  "Histogram of the minimum G-values
                  of the inner bloodstain"), FALSE, "")
histogram_plot(plot_data, in_G_max, 60,
                c("Maximum G-values of the inner bloodstain",
                  "Histogram of the maximum G-values
                  of the inner bloodstain"), FALSE, "hist_in_G_max.png")
histogram_plot(plot_data, in_G_variance, 30,
                c("Variance G-values of the inner bloodstain",
                  "Histogram of the variance of the G-values
                  of the inner bloodstain"), FALSE, "")
histogram_plot(plot_data, in_G_q5, 30,
                c("5% quantile of the G-values of the inner bloodstain",
                  "Histogram of the 5% quantile of the G-values
                  of the inner bloodstain"), FALSE, "")
histogram_plot(plot_data, in_G_q20, 30,
                c("20% quantile of the G-values of the inner bloodstain",
                  "Histogram of the 20% quantile of the G-values
                  of the inner bloodstain"), FALSE, "")
histogram_plot(plot_data, in_G_q50, 30,
                c("50% quantile of the G-values of the inner bloodstain",
                  "Histogram of the 50% quantile of the G-values
                  of the inner bloodstain"), FALSE, "")
histogram_plot(plot_data, in_G_q80, 30,
                c("80% quantile of the G-values of the inner bloodstain",
                  "Histogram of the 80% quantile of the G-values
                  of the inner bloodstain"), FALSE, "")
histogram_plot(plot_data, in_G_q95, 30,
                c("95% quantile of the G-values of the inner bloodstain",
                  "Histogram of the 95% quantile of the G-values
                  of the inner bloodstain"), FALSE, "")

### B-values
histogram_plot(plot_data, B_mean, 30,
                c("Mean B-values", "Histogram of the mean B-values"),
                FALSE, "")
histogram_plot(plot_data, B_min, 30,
                c("Minimum B-values", "Histogram of the minimum B-values"),
                FALSE, "")
histogram_plot(plot_data, B_max, 60,
                c("Maximum B-values", "Histogram of the maximum B-values"),
                FALSE, "hist_B_max.png")
histogram_plot(plot_data, B_variance, 30,
                c("Variance B-values",
                  "Histogram of the variance of the B-values"), FALSE, "")
histogram_plot(plot_data, B_q5, 50,
                c("5% quantile of the B-values",
                  "Histogram of the 5% quantile of the B-values"), FALSE, "")
histogram_plot(plot_data, B_q20, 30,
                c("20% quantile of the B-values",
                  "Histogram of the 20% quantile of the B-values"), FALSE, "")
```

```
histogram_plot(plot_data, B_q50, 30,
               c("50% quantile of the B-values",
                 "Histogram of the 50% quantile of the B-values"), FALSE, "")
histogram_plot(plot_data, B_q80, 30,
               c("80% quantile of the B-values",
                 "Histogram of the 80% quantile of the B-values"), FALSE, "")
histogram_plot(plot_data, B_q95, 30,
               c("95% quantile of the B-values",
                 "Histogram of the 95% quantile of the B-values"), FALSE, "")
# Inner bloodstain
histogram_plot(plot_data, in_B_mean, 30,
               c("Mean B-values of the inner bloodstain",
                 "Histogram of the mean B-values
                 of the inner bloodstain"), FALSE, "")
histogram_plot(plot_data, in_B_min, 30,
               c("Minimum B-values of the inner bloodstain",
                 "Histogram of the minimum B-values
                 of the inner bloodstain"), FALSE, "")
histogram_plot(plot_data, in_B_max, 60,
               c("Maximum B-values of the inner bloodstain",
                 "Histogram of the maximum B-values
                 of the inner bloodstain"), FALSE, "hist_in_B_max.png")
histogram_plot(plot_data, in_B_variance, 30,
               c("Variance B-values of the inner bloodstain",
                 "Histogram of the variance of the B-values
                 of the inner bloodstain"), FALSE, "")
histogram_plot(plot_data, in_B_q5, 30,
               c("5% quantile of the B-values of the inner bloodstain",
                 "Histogram of the 5% quantile of the B-values
                 of the inner bloodstain"), FALSE, "")
histogram_plot(plot_data, in_B_q20, 30,
               c("20% quantile of the B-values of the inner bloodstain",
                 "Histogram of the 20% quantile of the B-values
                 of the inner bloodstain"), FALSE, "")
histogram_plot(plot_data, in_B_q50, 30,
               c("50% quantile of the B-values of the inner bloodstain",
                 "Histogram of the 50% quantile of the B-values
                 of the inner bloodstain"), FALSE, "")
histogram_plot(plot_data, in_B_q80, 30,
               c("80% quantile of the B-values of the inner bloodstain",
                 "Histogram of the 80% quantile of the B-values
                 of the inner bloodstain"), FALSE, "")
histogram_plot(plot_data, in_B_q95, 30,
               c("95% quantile of the B-values of the inner bloodstain",
                 "Histogram of the 95% quantile of the B-values
                 of the inner bloodstain"), FALSE, "")


### GRAY-values
histogram_plot(plot_data, GRAY_mean, 30,
               c("Mean GRAY-values", "Histogram of the mean GRAY-values"),
               FALSE, "")
histogram_plot(plot_data, GRAY_min, 30,
               c("Minimum GRAY-values", "Histogram of the minimum GRAY-values"),
               FALSE, "")
histogram_plot(plot_data, GRAY_max, 50,
               c("Maximum GRAY-values", "Histogram of the maximum GRAY-values"),
               FALSE, "")
histogram_plot(plot_data, GRAY_variance, 30,
               c("Variance GRAY-values",
                 "Histogram of the variance of the GRAY-values"),
               FALSE, "")
histogram_plot(plot_data, GRAY_q5, 30,
               c("5% quantile of the GRAY-values",
                 "Histogram of the 5% quantile of the GRAY-values"),
               FALSE, "")
histogram_plot(plot_data, GRAY_q20, 30,
               c("20% quantile of the GRAY-values",
                 "Histogram of the 20% quantile of the GRAY-values"),
               FALSE, "")
histogram_plot(plot_data, GRAY_q50, 30,
```

```
                c("50% quantile of the GRAY-values",
                  "Histogram of the 50% quantile of the GRAY-values"),
                FALSE, "")
histogram_plot(plot_data, GRAY_q80, 30,
                c("80% quantile of the GRAY-values",
                  "Histogram of the 80% quantile of the GRAY-values"),
                FALSE, "")
histogram_plot(plot_data, GRAY_q95, 30,
                c("95% quantile of the GRAY-values",
                  "Histogram of the 95% quantile of the GRAY-values"),
                FALSE, "")
# Inner bloodstain
histogram_plot(plot_data, in_GRAY_mean, 30,
                c("Mean GRAY-values of the inner bloodstain",
                  "Histogram of the mean GRAY-values
                  of the inner bloodstain"), FALSE, "")
histogram_plot(plot_data, in_GRAY_min, 30,
                c("Minimum GRAY-values of the inner bloodstain",
                  "Histogram of the minimum GRAY-values
                  of the inner bloodstain"), FALSE, "")
histogram_plot(plot_data, in_GRAY_max, 60,
                c("Maximum GRAY-values of the inner bloodstain",
                  "Histogram of the maximum GRAY-values
                  of the inner bloodstain"), FALSE, "")
histogram_plot(plot_data, in_GRAY_variance, 30,
                c("Variance GRAY-values of the inner bloodstain",
                  "Histogram of the variance of the GRAY-values
                  of the inner bloodstain"), FALSE, "")
histogram_plot(plot_data, in_GRAY_q5, 30,
                c("5% quantile of the GRAY-values of the inner bloodstain",
                  "Histogram of 5% quantile of the GRAY-values
                  of the inner bloodstain"), FALSE, "")
histogram_plot(plot_data, in_GRAY_q20, 30,
                c("20% quantile of the GRAY-values of the inner bloodstain",
                  "Histogram of 20% quantile of the GRAY-values
                  of the inner bloodstain"), FALSE, "")
histogram_plot(plot_data, in_GRAY_q50, 30,
                c("50% quantile of the GRAY-values of the inner bloodstain",
                  "Histogram of 50% quantile of the GRAY-values
                  of the inner bloodstain"), FALSE, "")
histogram_plot(plot_data, in_GRAY_q80, 30,
                c("80% quantile of the GRAY-values of the inner bloodstain",
                  "Histogram of 80% quantile of the GRAY-values
                  of the inner bloodstain"), FALSE, "")
histogram_plot(plot_data, in_GRAY_q95, 30,
                c("95% quantile of the GRAY-values of the inner bloodstain",
                  "Histogram of 95% quantile of the GRAY-values
                  of the inner bloodstain"), FALSE, "")



##### Scatter plots #####
### Scatter plot with time and colour value
# Means
grouped_scatter_plot(plot_data, time, R_mean, samp_num,
                     c("Time (hours)", "Mean R-value",
                       "Scatter plot of the mean R-values over time"), FALSE,
                     "scatter_R_mean.png")
grouped_scatter_plot(plot_data, time, in_R_mean, samp_num,
                     c("Time (hours)", "Mean R-value",
                       "Scatter plot of the mean R-values over time"), FALSE,
                     "scatter_in_R_mean.png")
grouped_scatter_plot(plot_data, time, G_mean, samp_num,
                     c("Time (hours)", "Mean G-value",
                       "Scatter plot of the mean G-values over time"), FALSE,
                     "scatter_G_mean.png")
grouped_scatter_plot(plot_data, time, in_G_mean, samp_num,
                     c("Time (hours)", "Mean G-value",
                       "Scatter plot of the mean G-values over time"), FALSE,
                     "scatter_in_G_mean.png")
grouped_scatter_plot(plot_data, time, B_mean, samp_num,
```

```
                      c("Time (hours)", "Mean B-value",
                        "Scatter plot of the mean B-values over time"), FALSE,
                      "scatter_B_mean.png")
grouped_scatter_plot(plot_data, time, in_B_mean, samp_num,
                      c("Time (hours)", "Mean B-value",
                        "Scatter plot of the mean B-values over time"), FALSE,
                      "scatter_in_B_mean.png")
grouped_scatter_plot(plot_data, time, GRAY_mean, samp_num,
                      c("Time (hours)", "Mean GRAY-value",
                        "Scatter plot of the mean GRAY-values over time"),FALSE,
                      "scatter_GRAY_mean.png")
grouped_scatter_plot(plot_data, time, in_GRAY_mean, samp_num,
                      c("Time (hours)", "Mean GRAY-value",
                        "Scatter plot of the mean GRAY-values over time"),FALSE,
                      "scatter_in_GRAY_mean.png")

# Max
grouped_scatter_plot(plot_data, time, R_max, samp_num,
                      c("Time (hours)", "Maximum R-value",
                        "Scatter plot of the max R-values over time"), FALSE,
                      "scatter_R_max.png")
grouped_scatter_plot(plot_data, time, in_R_max, samp_num,
                      c("Time (hours)", "Maximum R-value",
                        "Scatter plot of the max R-values over time"), FALSE,
                      "scatter_in_R_max.png")
grouped_scatter_plot(plot_data, time, G_max, samp_num,
                      c("Time (hours)", "Maximum G-value",
                        "Scatter plot of the max G-values over time"), FALSE,
                      "scatter_G_max.png")
grouped_scatter_plot(plot_data, time, in_G_max, samp_num,
                      c("Time (hours)", "Maximum G-value",
                        "Scatter plot of the max G-values over time"), FALSE,
                      "scatter_in_G_max.png")
grouped_scatter_plot(plot_data, time, B_max, samp_num,
                      c("Time (hours)", "Maximum B-value",
                        "Scatter plot of the max B-values over time"), FALSE,
                      "scatter_B_max.png")
grouped_scatter_plot(plot_data, time, in_B_max, samp_num,
                      c("Time (hours)", "Maximum B-value",
                        "Scatter plot of the max B-values over time"), FALSE,
                      "scatter_in_B_max.png")


### Scatter plot with colour values for complete and inner bloodstain
# Mean
grouped_scatter_plot(plot_data, R_mean, in_R_mean, samp_num,
                      c("Mean R-value (complete)", "Mean R-value (inner)",
                        "Scatter plot of the mean R-values
                        for the complete and inner bloodstain"), FALSE,
                      "scatter_R_mean_complete_inner.png")

grouped_scatter_plot(plot_data, G_mean, in_G_mean, samp_num,
                      c("Mean G-value (complete)", "Mean G-value (inner)",
                        "Scatter plot of the mean G-values
                        for the complete and inner bloodstain"), FALSE,
                      "scatter_G_mean_complete_inner.png")

grouped_scatter_plot(plot_data, B_mean, in_B_mean, samp_num,
                      c("Mean B-value (complete)", "Mean B-value (inner)",
                        "Scatter plot of the mean B-values
                        for the complete and inner bloodstain"), FALSE,
                      "scatter_B_mean_complete_inner.png")

### Complete and inner bloodstain
grouped_scatter_plot(plot_data, R_max, in_R_max, samp_num,
                      c("Max R-value (complete)", "Max R-value (inner)",
                        "Scatter plot of the maximum R-values
                        for the complete and inner bloodstain"), FALSE,
                      "scatter_R_max_complete_inner.png")

grouped_scatter_plot(plot_data, G_max, in_G_max, samp_num,
```

```
                      c("Max G-value (complete)", "Max G-value (inner)",
                        "Scatter plot of the maximum G-values
                        for the complete and inner bloodstain"), FALSE,
                      "scatter_G_max_complete_inner.png")

grouped_scatter_plot(plot_data, B_max, in_B_max, samp_num,
                      c("Max B-value (complete)", "Max B-value (inner)",
                        "Scatter plot of the maximum B-values
                        for the complete and inner bloodstain"), FALSE,
                      "scatter_B_max_complete_inner.png")
grouped_scatter_plot(plot_data, GRAY_max, in_GRAY_max, samp_num,
                      c("Max GRAY-value (complete)", "Max GRAY-value (inner)",
                        "Scatter plot of the maximum GRAY-values
                        for the complete and inner bloodstain"), FALSE,
                      "scatter_GRAY_max_complete_inner.png")

grouped_scatter_plot(plot_data, R_q95, in_R_q95, samp_num,
                      c("95% quantile of R-value (complete)",
                        "95% quantile of R-value (inner)",
                        "Scatter plot of the 95% quantile  of the R-values
                        for the complete and inner bloodstain"), FALSE,
                      "scatter_R_q95_complete_inner.png")
grouped_scatter_plot(plot_data, G_q95, in_G_q95, samp_num,
                      c("95% quantile of G-value (complete)",
                        "95% quantile of G-value (inner)",
                        "Scatter plot of the 95% quantile  of the G-values
                        for the complete and inner bloodstain"), FALSE,
                      "scatter_G_q95_complete_inner.png")
grouped_scatter_plot(plot_data, B_q95, in_B_q95, samp_num,
                      c("95% quantile of B-value (complete)",
                        "95% quantile of B-value (inner)",
                        "Scatter plot of the 95% quantile  of the B-values
                        for the complete and inner bloodstain"), FALSE,
                      "scatter_B_q95_complete_inner.png")
grouped_scatter_plot(plot_data, GRAY_q95, in_GRAY_q95, samp_num,
                      c("95% quantile of GRAY-value (complete)",
                        "95% quantile of GRAY-value (inner)",
                        "Scatter plot of the 95% quantile  of the GRAY-values
                        for the complete and inner bloodstain"), FALSE,
                      "scatter_GRAY_q95_complete_inner.png")

### Scatter plots for control samples
grouped_scatter_plot(control_data, time, R_mean, samp_num,
                      c("Time (hours)", "Mean R-value",
                        "Scatter plot of the mean R-values over time"), FALSE,
                      "scatter_control_R_mean.png")
grouped_scatter_plot(control_data, time, G_mean, samp_num,
                      c("Time (hours)", "Mean G-value",
                        "Scatter plot of the mean G-values over time"), FALSE,
                      "scatter_control_G_mean.png")
grouped_scatter_plot(control_data, time, B_mean, samp_num,
                      c("Time (hours)", "Mean B-value",
                        "Scatter plot of the mean B-values over time"), FALSE,
                      "scatter_control_B_mean.png")

### Other scatter plots
grouped_scatter_plot(plot_data, time, R_q50, samp_num,
                      c("Time (hours)", "50% quantile of the R-values",
                        "Scatter plot of the 50% quantile
                        of the R-values over time"), FALSE,
                      "scatter_R_q50.png")
grouped_scatter_plot(plot_data, time, in_R_q50, samp_num,
                      c("Time (hours)", "50% quantile of the R-values",
                        "Scatter plot of the 50% quantile
                        of the R-values over time"), FALSE,
                      "scatter_R_min.png")
grouped_scatter_plot(plot_data, time, R_q95, samp_num,
                      c("Time (hours)",
                        "95% quantile of R-value",
                        "Scatter plot of the 95% quantile
                        of the R-values over time"), FALSE,
```

```
                      "scatter_R_q95.png")
grouped_scatter_plot(plot_data, time, in_R_q95, samp_num,
                      c("Time (hours)",
                        "95% quantile of R-value",
                        "Scatter plot of the 95% quantile
                        of the R-values over time"), FALSE,
                      "scatter_in_R_q95.png")
grouped_scatter_plot(plot_data, time, R_min, samp_num,
                      c("Time (hours)", "Minimum R-value",
                        "Scatter plot of the minimum R-values over time"), FALSE,
                      "scatter_R_min.png")
grouped_scatter_plot(plot_data, time, R_variance, samp_num,
                      c("Time (hours)", "Variance of the R-value",
                        "Scatter plot of variance of the R-values over time"),
                      FALSE, "scatter_R_var.png")
```

Bibliography

[1] Sunil L Bangare, Amruta Dubal, Pallavi S Bangare, and Suhas Patil. Reviewing otsu's method for image thresholding. *International Journal of Applied Engineering Research*, 10(9):21777–21783, 2015.

[2] Serafine Beugelink. Age estimation of bloodstains by color analysis in varying climate conditions. Master's thesis, Bachelor's thesis, Leiden University, 2023.

[3] A Biedermann and F Taroni. Bayesian networks for evaluating forensic dna profiling evidence: a review and guide to literature. *Forensic Science International: Genetics*, 6(2):147–157, 2012.

[4] Anouk de Ronde. *What fingermarks reveal about activities*. PhD thesis.

[5] Paolo Garbolino and Franco Taroni. Evaluation of scientific evidence using bayesian networks. *Forensic Science International*, 125(2-3):149–155, 2002.

[6] Zhiwei Ji, Qibiao Xia, and Guanmin Meng. A review of parameter learning methods in bayesian network. In *Advanced Intelligent Computing Theories and Applications: 11th International Conference, ICIC 2015, Fuzhou, China, August 20-23, 2015. Proceedings, Part III 11*, pages 3–12. Springer, 2015.

[7] Otto Leers. *Die forensische Blutuntersuchung: ein Leitfaden für Studierende, beamtete und sachverständige Ärzte und für Kriminalisten*. J. Springer, 1910.

[8] Mauro Scanagatta, Antonio Salmerón, and Fabio Stella. A survey on bayesian network structure learning from data. *Progress in Artificial Intelligence*, 8:425–439, 2019.

[9] Marco Scutari and Jean-Baptiste Denis. *Bayesian networks: with examples in R*. CRC press, 2014.

[10] Joonchul Shin, Seoyeon Choi, Jung-Sik Yang, Jaewoo Song, Jong-Soon Choi, and Hyo-Il Jung. Smart forensic phone: Colorimetric analysis of a bloodstain for age estimation using a smartphone. *Sensors and Actuators B: Chemical*, 243:221–225, 2017. doi: https://doi.org/10.1016/j.snb.2016.11.142.

[11] Stefan Strasser, Albert Zink, Gerald Kada, Peter Hinterdorfer, Oliver Peschel, Wolfgang M Heckl, Andreas G Nerlich, and Stefan Thalhammer. Age determination of blood spots in forensic medicine by force spectroscopy. *Forensic Science International*, 170(1):8–14, 2007.

[12] Phuvadol Thanakiatkrai, Alisa Yaodam, and Thitika Kitpipit. Age estimation of bloodstains using smartphones and digital image analysis. *Forensic science international*, 233(1-3):288–297, 2013. doi: https://doi.org/10.1016/j.forsciint.2013.09.027.

[13] Luigi Tomellini. *De l'emploi d'une table chromatique pour les taches du sang*. A. Rey et Cie, 1907.

[14] Alexis R Weber and Igor K Lednev. Crime clock–analytical studies for approximating time since deposition of bloodstains. *Forensic Chemistry*, 19:100248, 2020. doi: https://doi.org/10.1016/j.forc.2020.100248.