MEng Individual Project - Final Report

Imperial College London

Department of Electrical and Electronic Engineering

Analysis and Implementation of Data Imputation Techniques for Laboratory Data

Project Supervisor:
Dr Bernard Hernandez Perez

Project Co-Supervisor:
Dr Pantelis Georgiou

Project Second Marker:
Dr Timothy Constandinou

Author:
Mr. Agrim Manchanda
CID: 01333674

A thesis submitted for the degree of

MEng Electronic and Information Engineering

June 21, 2021

Final Report Plagiarism Statement

I affirm that I have submitted, or will submit, electronic copies of my final year project report to both Blackboard and the EEE coursework submission system.

I affirm that the two copies of the report are identical.

I affirm that I have provided explicit references for all material in my Final Report which is not authored by me and represented as my own work.

Abstract

Missing data is a ubiquitous problem in the clinical domain which impacts the fidelity of predictive modelling as many of the models rely on the completeness of data. Simple methods such as mean or median imputation exist but they do not preserve the relations between variables. As such, they do not capture and exploit feature correlations which exists between variables in laboratory data. This project presents the design and implementation of an imputation framework which provides a robust methodology to investigate two data imputation techniques, Machine Learning and Bayesian Networks, that inherently exploit feature relations to infer missing data values from those that are observed. The project carries out an empirical study on a real-life laboratory data set to compare the performance of the aforementioned techniques against simple median imputation. The project concludes with an evaluation of the studied techniques and recommendations on how the methods can be integrated as part of an existing clinical decision support system.

Acknowledgements

Firstly, I would like to thank Dr Pantelis Georgiou for providing me an opportunity to undertake this project. This project has allowed me to use my skills in the best possible way.

Next, I would like to express deep gratitude to Dr Bernard Hernandez Perez for his conscientious guidance, constant support and invaluable feedback throughout the entirety of my project. I especially thank him for his excellent cooperation, useful suggestions and generously giving me his time. I have thoroughly enjoyed working and learning from him.

Then, I would like to thank Dr Timothy Constandinou for being another marker of this project.

I would also like to thank Dr Damien Keng Yen Ming for his feedback on the clinical aspects of my project.

Finally, I would like to express my sincere gratitude to my parents, grandparents and my sister for their unconditional love, constant prayers, countless sacrifices and heartfelt blessings. They have always pushed me to strive for the highest and I remain forever indebted to the effort they have put in my upbringing.

Above all, I thank **Almighty God** for showering his blessings and grace towards me.

Contents

Chapter 1

Introduction

1.1 Motivation

Over the last 20 years, the healthcare domain has observed a continuous growth in the adoption of electronic health record (EHR) systems [1] which store patient medical information such as demographics, medical history and laboratory test data [2]. Consequently, it has led to a significant improvement in the quality and efficiency of delivering patient clinical care [3]. This rise has been further accelerated by the proliferation of emerging technologies such as *big data*, *machine learning (ML)* and *artificial intelligence (AI)* which allow clinicians to better predict patient outcomes and manage treatment plans accordingly [4]. As such, predictive modelling is vital across all healthcare services and forms an integral part of clinical decision support systems (CDSSs) [5].

However, many of these models rely on the completeness of the underlying data for correct inferences and hence are susceptible to inaccurate conclusions if the data is missing or incomplete. Missing data can also introduce *bias* which can significantly skew results and potentially invalidate conclusions derived from those results [6]. This presents a clear limitation at a time when there are significant ongoing developments in increasing the availability of patient medical data for predictive modelling. It is an ever-present challenge and an active area of research for many secondary users of EHRs [7].

The prevalent issue of missing data, particularly in the clinical domain, has been under scrutiny for many years [8]. Studies have found several reasons for why this is the case [8], yet it is difficult to prevent them from happening. Some solutions have been proposed but they have clear limitations. A study conducted by Raaijmakers (1999) [9] found that eliminating or discarding 30% of records with missing values leads to a 98% loss in statistical power. Instead, researchers have been trying to investigate feasible methods by filling otherwise known as *imputing* the missing values [8].

Simple imputation methods such as using the mean or median of the observed values for each feature are commonly used, though they do not preserve relations between the variables [10]. Such techniques are not ideal for imputing missing values in pathology laboratory data whereby the variables, *biochemical markers* or *analytes*, are known to be correlated with one another [11]. It is important to capture and preserve these relations when imputing data to prevent distortion of the underlying data distribution and subsequently improve the accuracy of predictive models.

1.2 Project Objectives

Enhanced, Personalised and Integrated Care for Infection Management at the Point of Care (EPiC IMPOC) is an intelligent CDSS developed by Hernandez *et al.* (2018) [12] to improve clinical management of infectious diseases. The online tool has been piloted successfully across three North West London NHS hospitals. The server side of the application comprises of six modules; one of them is *probabilistic inference (PI)* which generates the predictive models to assist clinicians in assessing the severity of infections. A limitation of this module is that it uses the median to impute missing values in pathology laboratory test data [12].

The aim of this project is to investigate and implement better data imputation techniques to enhance the performance of predictive models generated in EPiC IMPOC. Imputing missing test results with the median is a generic approach which may be skewed by the results of other patients with specific underlying health conditions and diseases. It also does not account for any contextual information such as if the patients are subject to specific treatments. As such, the relations between the biochemical markers are not preserved which inevitably leads to inaccurate classifier predictions in CDSSs. Hence, there is an opportunity to explore and identify underlying relations between biochemical markers and exploit these relations to infer missing data values from those that have already been collected. Furthermore, the inclusion of the implemented imputation methods in EPiC IMPOC in the future should increase the availability of data to help the clinicians better evaluate the severity of infections in patients [12].

This project will develop an imputation framework suitable for laboratory data and carry out an empirical study to compare the performance of using traditional ML methods and probabilistic graphical models (PGMs) against simple median imputation. ML has been studied and applied extensively within the clinical domain which warrants its selection for this project [4]. On the other hand, PGMs inherently deal with missing data and provide an intuitive way to reason and visualise relations between analytes. As such, the scope of this project extends beyond its proposed use case - the imputation methods investigated in this project will serve as tools to help clinicians better understand the rationale behind the predicted values which are otherwise perceived as “black-box algorithms” [13].

1.3 Report Structure

This section presents a brief overview of the main content discussed in each chapter of this report. Each chapter is concluded with a brief summary of the key points covered in the chapter.

- *Chapter 2*: this chapter covers the background and preliminary materials for this project. It presents material on classifying missing data types (Section 2.1), common imputation methods (Section 2.2), machine learning topics (Section 2.3), probabilistic graphical models (Section 2.4) and software tools (Section 2.5).
- *Chapter 3*: this chapter covers the requirements and deliverables in this project which include design (Section 3.1), implementation (Section 3.2), experimentation (Section 3.3) and code documentation (Section 3.4).
- *Chapter 4*: this chapter covers the analysis and design of the proposed imputation framework. The material includes: overview of design (Section 4.1), feature selection (Section 4.2), prefilling (Section 4.4), pre-processing (Section 4.3), model learning (Section 4.5) and imputation (Section 4.6).
- *Chapter 5*: this chapter covers implementation details for the designed framework. This section includes: overview of implementations (Section 5.1), feature selection (Section 5.2), prefilling (Section 5.3), pre-processing (Section 5.4), ML methods (Section 5.5) and BN methods (Section 5.6).
- *Chapter 6*: this chapter presents the experiments and their respective methodologies. First, an overview of the data set is provided (Section 6.1) followed by an overview of experiments I - IV (Section 6.2). Experiments are presented in order: I (Section 6.3), II (Section 6.4), III (Section 6.5) and IV (Section 6.6).
- *Chapter 7*: this chapter presents the results for the experiments described in Chapter 6. First, the chapter covers necessary prerequisites (Section 7.1) followed by the experiment results in order: II (Section 7.2), III (Section 7.3) and IV (Section 7.4). Note the results of experiment I are covered in Section 6.3.
- *Chapter 8*: this chapter presents an evaluation of the requirements outlined in Chapter 3. First, an evaluation of project requirements is presented (Section 8.1) followed by comparison of the imputation methods (Section 8.2).
- *Chapter 9*: this chapter presents the project conclusions (Section 9.1) and areas of future work (Section 9.2).

It should be noted that in this report, the terms *project* and *study* are used interchangeably.

Chapter 2

Background

This aim of this chapter is to cover the relevant background material and theory required to understand the two approaches that will be investigated in this project. First, the chapter explains three types of missing data (Section 2.1) to understand the underlying assumptions behind the choice of imputation methods. Then, three types of common imputation methods (Section 2.2) are presented with a facilitating discussion on their benefits and drawbacks. Next, preliminaries for the two approaches are covered: machine learning (ML) (Section 2.3) and probabilistic graphical models (PGMs) (Section 2.4) to understand the methods and algorithms that underpin their use. Literature reviews are presented with each section. The software tools selected for this project are also covered (Section 2.5). Finally, a summary of the chapter is presented (Section 2.6).

2.1 Classifying missing data types

Before analysing imputation methods from the literature, it is important to establish and classify missing data types. This will help when evaluating the models as it may offer plausible explanations for the obtained results. This section will provide an overview of the three main types of missing data identified from the literature [14]:

2.1.1 Missing completely at random

The data is said to be missing completely at random (MCAR) when the reason for the missing data is completely independent of whether any value was or was not observed in the data [14]. For example, the laboratory test results of some patients may be missing from the EHR because a data entry was not made (for any arbitrary reason). Here, the reason for the missing data is completely random and hence independent of the inclusion of other patient's data. From a statistical perspective, this case only leads to a loss of data (which may impact the fidelity of statistical modelling) but does not introduce any systematic bias into the data [14].

2.1.2 Missing at random

The data is said to be missing at random (MAR) when there is a correlation between the missing data and the other *observed* values from a different variable in the data [14]. For example, patients who have Trypanophobia (fear of needles) are less likely to (voluntarily) get a blood test. A more relevant example may be the clinician not requesting for a laboratory test based on the (already received) results of another test. Here, the missingness of data is correlated with the (diagnosed) condition of the patient. It is important in such cases to account for the reasons of missing data otherwise it may introduce *bias* if the incomplete data is used in subsequent studies [14].

2.1.3 Missing not at random

The data is said to be missing not at random (MNAR) when there is a correlation between the missing data and other unobserved data outside the scope of the collected data [14]. For example, patients are more likely to drop out of clinical studies if their condition (due to an unrecorded illness) worsens. Here, it is important to note that the state of their condition (or diagnosis of illness) is not recorded as part of the study and hence falls under unobserved data. Like MAR, this may introduce bias into the data [14].

2.1.4 Summary

While the features of missing data types are clearly presented in the literature, classifying real (incomplete) data sets is not a straightforward process. While these mechanisms are applicable for classifying collected data in clinical trials [15], the classifying criteria is not universal across healthcare [7] or other domains. There are several statistical methods proposed in the literature [16] to determine missing data types but they fall outside of the scope of this project. On a more general note, most incomplete data sets are assumed to be MAR as it forms the basis of assumption for the application of several imputation methods [17].

2.2 Common imputation methods

This section will provide an overview of common data imputation methods which can be categorised into three main types: Complete Case Analysis (CCA), Single Imputation (SI) and Model Based Imputation (MBI) [18].

2.2.1 Complete case analysis

Complete case analysis (CCA), as the name suggests, involves only the study of participants or variables with only complete records and no missing values. As such, this reduces the sample size of the data set and subsequently the statistical power of any modelling [18]. While this is undesirable, it is still a commonly employed technique because it does not rely on any assumptions about the underlying data distributions and saves computational overheads to impute missing values. It is also commonly used in studies because it provides a (reliable) source of ground-truth values using which the performance of any predictive model can be determined.

2.2.2 Single imputation methods

In single imputation (SI) methods, the missing values are only replaced once. This presents a simple and computationally efficient approach that is scalable with large data sets. Two of the most common SI methods used in the clinical domain are: simple and regression.

Simple: As aforementioned, simple methods such as mean and median are commonly used to impute missing data [10] though are routinely outperformed by other more complex methods [8]. Mean imputation is implemented by imputing the missing value(s) of a variable by the mean of the other observed value(s). The steps are the same for mode/median except the respective statistic method is used. While their simplicity is desirable (and often preferred), these methods tend to distort the underlying distributions in the data set, reduce variability and disregard variable relations [10] [19]. This does imply, however, that they are suitable for data sets where the correlations between variables are weak or only a few values are missing from the data set [19].

Regression: Regression imputation consists of two steps: generation of regression model (usually linear) and subsequent use of the model to predict (and impute) missing value(s) [19]. This method addresses the drawbacks described with mean imputation as it preserves the underlying data distribution and leverages on the relations between variables to create an imputation model. However, this method only tends to be used with MCAR data and not MAR/MNAR data as it may lead to bias results [19]. See Section 2.3.4 for more details on regression.

2.2.3 Model-based methods

The second type of methods to be studied is model based imputation (MBI) because they use an underlying modelling methods to predict values. Many MBI methods extend the regression model to create more sophisticated algorithms; there can be many variations of MBI methods but there are two common approaches described in literature [19].

Maximum Likelihood: Maximum Likelihood, also referred as Expectation-Maximisation (EM) method in the literature [17], was first proposed by Dempster *et al.* (1997) [20] and is a "general method of finding maximum likelihood estimate of parameters of an underlying distribution" [17]. In other words, this method tries to find the most likely estimate value that should be imputed with respect to the observed distribution in the data set. The EM algorithm iterates through two steps: Expectation (E) and Maximisation (M) until it converges on the most likely estimate value. Figure 2.1 provides an overview of the method.

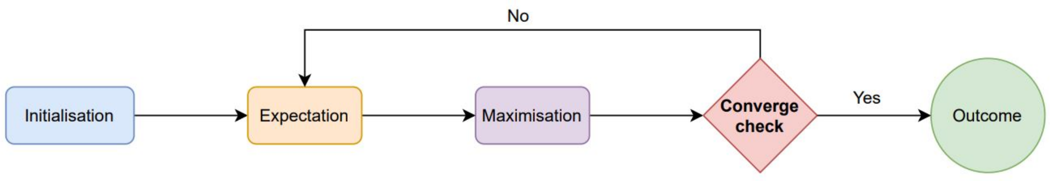


Figure 2.1: Flow diagram showing the steps in EM algorithm. The model is first initialised with parameter values. In the E-step, the algorithm generates an imputation model based on other observed data and the initialised parameters to find an estimate value. The following M-step uses the *likelihood* function to check if the estimate is most likely. The two steps continue to repeat until the estimate converges to produce a single final value [21].

For a more rigorous mathematical derivation of the steps in EM algorithm refer to [21].

EM algorithms have been widely used with other distribution models (such as Gaussian) in the ML domain [22]. Therefore, they have been well optimised for fast operation [21]. Furthermore, the convergent nature of the algorithm means the estimate values are deterministic which increases confidence for use. However, a drawback of the algorithm is that it assumes the data is MAR and the variables exhibit a Gaussian distribution. Both of these assumptions may not necessarily be true for all data sets [21].

Multiple imputation: While the SI methods solve the problem at hand, they do not account for the error variance that is inherently introduced into the data [19]. Multiple imputation (MI) methods, first proposed by Rubin (1977) [23], address this issue. Figure 2.2 provides an illustration of the main steps in this method.

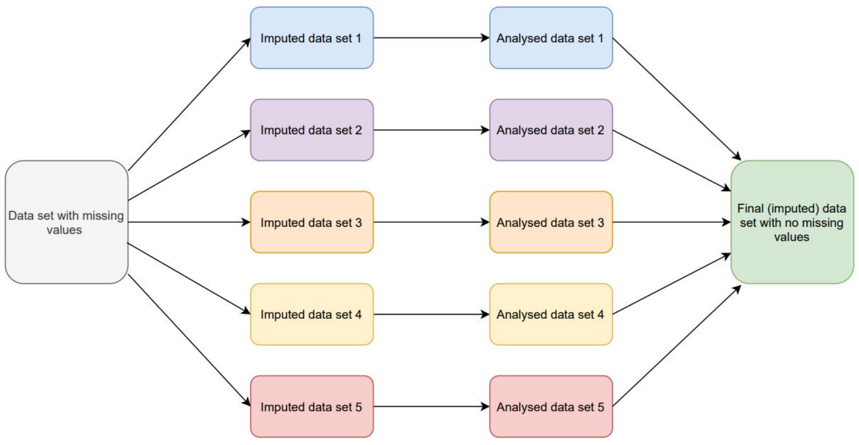


Figure 2.2: Diagram showing the different steps in MI model. The first step is to generate m (here $m = 5$) imputed copies of the data set using an imputation model that incorporates random variation. This produces m-complete data sets, each with a different imputed value. Then, each data set is analysed using "standard complete-data methods", for example linear regression. At the final stage, m different estimates are averaged "to produce a single point estimate". [23]

The described method has many advantages. First, the random variation in the first stage helps to achieve unbiased estimates of all parameters [24]. It also accounts for uncertainty due to missing data and can be universally used for any type of data analysis [24]. However, this model shares its limitations with maximum likelihood as it assumes MAR data and Gaussian distribution of variables [24]. Furthermore, the user has to carefully select the imputation model to use in step 2 so that it is suitable for the type of problem at hand [24].

2.2.4 Use cases in clinical domain

Important clinical studies have been carried out to evaluate the performance of the described methods; a selected subset of those studies will be presented here. A simulation study conducted by Lodder *et al.* (2014) [19] compared four (above described) methods: mean imputation, regression imputation, maximum likelihood and multiple imputation. The study concluded that multiple imputation was most effective and provides "unbiased estimates of Cronbach's alpha", which is a measure of reliability [25]. More importantly, the study also concluded that the type of missing data (MCAR, MAR, MNAR) nor the proportion of missing data (10%, 20%, 30% in entire data set) did not significantly influence the results. Mean imputation, however, scored consistently below the required Cronbach alpha threshold.

A separate simulation study conducted by Waljee *et al.* (2013) [26] also concluded that mean imputation had the highest imputation error in filling missing biochemical marker values for detection of Cirrhosis. The same study also concluded that a MI based method, missForest [27], yielded lowest imputation error on MCAR data. There have been other studies which have supported this conclusion, such as the one conducted by Tang *et al.* (2017) [28], which found that the performance of these methods increase with variable correlation. Such studies indicate that MI methods have proven (as expected) to be more effective in preserving variable relations which inherently lead to stronger performances than their counterparts.

2.3 Machine Learning

Machine learning (ML) facilitates building of predictive models which learn, capture and use patterns from seen (trained) data to predict values when given an unseen (test) input data [29]. In other words, ML models aim to leverage information that they have previously been exposed to in order to make predictions on the true values of new data. This section will cover the ML preliminaries to understand the selection of algorithms for this approach and provide an overview of how they function.

2.3.1 Overview

There are three main categories of machine learning [22]: *supervised, unsupervised and reinforcement learning*. Supervised learning occurs when the model trains on a set of labelled input variables to infer a function f that maps the input variables to a set of correct output labels; in other words, the model has full knowledge of the input and output feature space. The trained model can then be applied to any unseen data to infer the (correct) output value. Supervised learning can be further divided into two sub-categories: *classification* and *regression*. In classification, the aim is to map the input variables to discrete/categorical outputs, whereas in regression, the aim is to map the inputs to real-valued/continuous outputs [22].

On the other hand, unsupervised learning occurs when the model has no knowledge of the output labels and hence relies on inferring hidden patterns between the input variables. Finally, reinforcement learning occurs when the model trains with unlabelled data (as with unsupervised learning) and uses a feedback loop (by interacting with the environment) to maximise its total reward pay off [22].

Biochemical marker values are usually recorded as continuous values in laboratory tests [30]. As such, the remainder of this project will focus on regression-based algorithms and their respective performance evaluation metrics.

2.3.2 Training, test and validation data sets

Before analysing different machine learning models, it is important to understand how data sets are prepared for supervised learning tasks. Figure 2.3 provides an overview of how three (different) models are evaluated using different splits of a single data set. Large data sets are typically divided into three sets [31]: *training, test and validation*. Training data, as the name suggests, is used to train the machine learning model to infer the mapping function f. Then, the performance of the trained model is evaluated on the (unseen) test data set. During the training stage, the model may learn patterns that are specific to the training data (also known as *noise*) which leads to *over-fitting* [31].

Therefore, a validation set is used to evaluate the performance of the trained model to ensure it *generalises* well and to tune any *hyperparameters*. These are parameters which can control the learning rate of a model and are usually initialised before training [22]. A simple example is the *K* in *K-Nearest Neighbours* algorithm. There are a few suggestions in the literature for common split ratios [31]. The experiments methodologies will outline the chosen split ratios respectively.

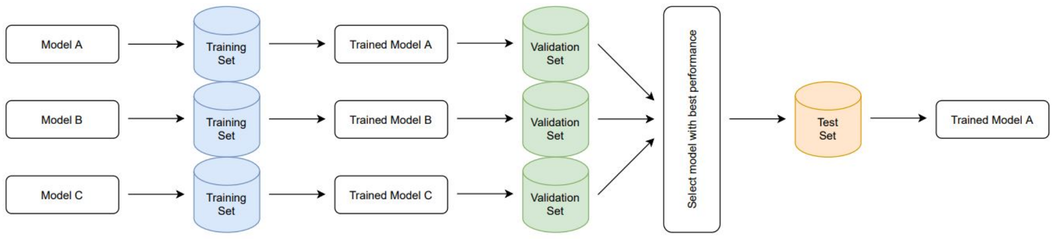


Figure 2.3: Diagram showing an overview of the evaluation pipeline. Three models are trained with the training set to generate the trained models. Then, their performance is evaluated (and hyperparameters tuned) on the validation set. The model with the best performance on the validation set is exposed to the unseen test set at which point its final performance is reported.

2.3.3 Cross Validation

One of the challenges in splitting data sets is finding the optimum trade off between the proportion of values in training/test set. Indeed, the goal is to maximise the number of points in each set so the model has maximum data to train and be tested on. Another evaluation strategy which accounts for this issue is stratified K-fold Cross Validation (CV) [32]. In this method, the entire data set is divided into k subsets and iterated k times using the training/test pipeline. In each iteration, one of the k subset is held out as test set while the remaining (k-1) subsets are used for training. At the end of all iterations, the error from each fold is averaged to produce a single CV accuracy error for the model. Stratified sampling is used to ensure each fold has the same class distribution to prevent bias learning but this is only applicable to classification tasks [32]. As such, only K-Fold CV will be used for this project as the variables will be continuous. Figure 2.4 [33] provides a clear illustration of a 5-fold CV process.

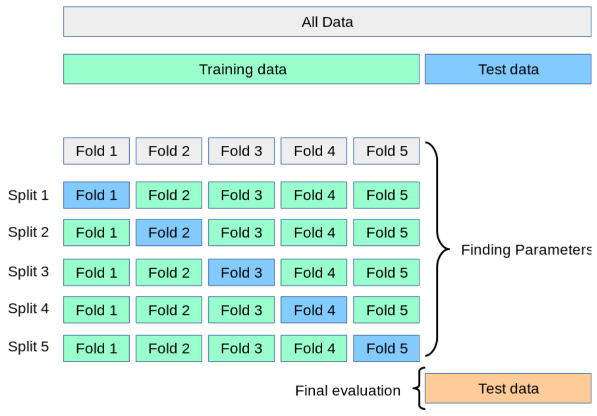


Figure 2.4: Diagram showing 5-fold CV [33]. The data set is split into training and test. The training data is further split into five folds which iterate through one test and four training sets in each split. The final evaluation is done on the held-out (unseen) test data.

2.3.4 Linear and polynomial models

As discussed in Section 2.3.1, the purpose of regression models is to infer a function that describes the relationship between the input and output feature space. One of the most common model is *linear regression* whereby the model depicts a linear relationship (using a *line of best fit*) between the input and output values. Likewise, *polynomial regression* model finds a polynomial curve (order of magnitude 2 or above) to fit the data. *Multiple linear regression* (many input variables), can be mathematically modelled as per Equation 2.1 below [34].

$$y = \beta_0 + \sum_{n=1}^{m} \beta_n x_n + \epsilon_n \tag{2.1}$$

where:

y	= output values of the model
β_0	= the y-intercept of the model
β_n	= coefficients defining slope of the model
x_n	= input values of the model
ϵ_n	= error term, typically modelled using Gaussian distribution

Regression models are popular because they are simple to implement and provide a robust method to find the most optimal coefficients, β, during model training [34]. However, they are susceptible to outliers in the data set which can skew the weights of the coefficients and hence the slope of *curve of best fit*.

2.3.5 Decision Tree models

Decision Trees (DTs) can be used to address some of the limitations of linear regression, specifically to model non-linear relationships [34]. DT, as the name suggests, aim to build a tree-like structure to model the relationship between the input and output variables by "learning simple data rules inferred from the data features" [33]. In other words, a set of *if-then-else* rules are recursively constructed using partitioning. Figure 2.5 provides an illustration of a simplified DT. The algorithms used to build the nodes vary according to the type of problem at hand; for regression problems the *classification and regression tree (CART)* algorithm is commonly used to find the optimal splitting attributes using *variance reduction* [35]. DT learning algorithms use top-down greedy search to build the structure [35]. They can be further optimised by *pruning* which is a technique to remove redundant nodes and improve performance [35].

DTs have been widely used in the clinical domain because of their simplicity and low training time [12]. They also do not require any data pre-processing so can be used for fast prototyping. However, the models have a propensity to over-fit because of their greedy search approach where the algorithm only optimises locally (at each splitting node) without considering the global optimum. This leads to a high variance in the output value [12].

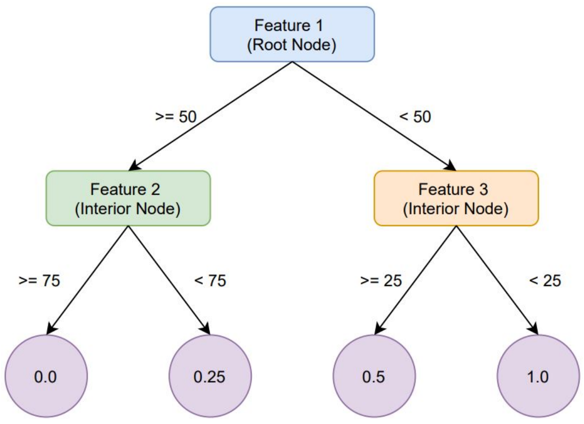


Figure 2.5: Diagram showing the structure of a (very simplified) decision tree. The *root node* partitions the data set into two. In this example, all records for which the value of Feature 1 is above 50 will go to the left-subtree else to the right, called *interior nodes*. Feature 2 and 3 further split the partitioned data based on their respective conditions (above/below 75 and 25) to arrive at the *leaf nodes* which are the (four possible) real-valued outputs of this algorithm.

2.3.6 Random Forests

Random Forests (RFs) are an ensemble of DTs, first proposed by Breiman (2001) [36], where each DT is allocated a different portion of the complete data set as its training set to generate a model. Each DT uses the steps described in Section 2.3.5 to predict an output (that is real-valued). The predictions are then averaged across all DTs to generate a single output value. Figure 2.6 illustrates an example of a RF constructed using Figure 2.5.

RFs address the over-fitting problem discussed with DTs because each model trains on a different subset of the complete data set and hence the averaged output has a lower variance [36]. However, the benefits come at a cost of increased complexity of the model as each DT exhibits a different structure which is difficult to interpret. Nevertheless, their low variance property makes them more desirable for predictive modelling than using a single DT.

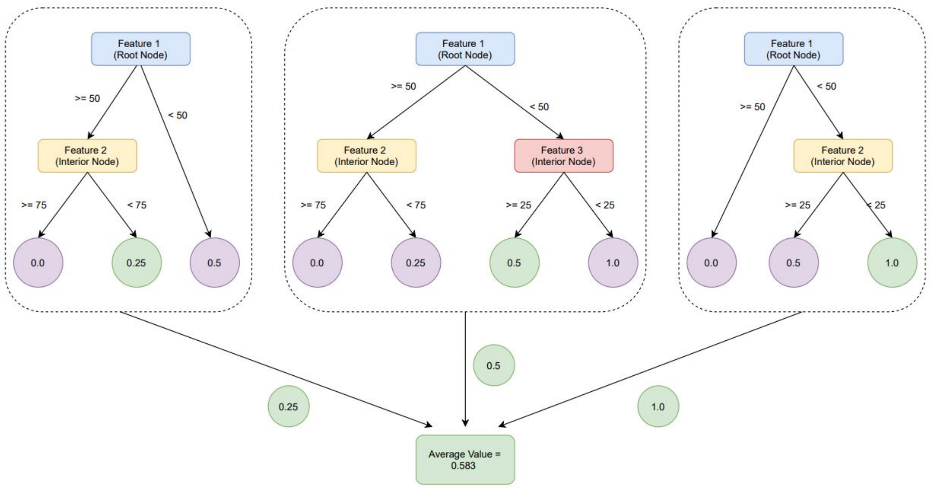


Figure 2.6: Diagram showing the structure of a random forest with three DTs. Each DT has a different structure and generates a different real-value prediction: 0.25, 0.5 and 1.0 respectively. These values are aggregated and averaged to find a single output: 0.583 (3 significant figures).

Gradient Boosting: RFs are one the most widely known ensemble methods but there are also other techniques such as stacking, bagging and boosting [37]. For brevity, only the boosting method is discussed in this section. While simple models can be effective, there are times when they are not able to capture enough information about the problem at hand. Boosting is a technique that tries to combine several simple or weak models to generate a composite predictor that encapsulates the power of each of the simple models [37]. It build the composite model in a "stage-wise additive" manner such that more models are added to the predictor until the optimisation problem of minimising the loss function is solved [37]. The gradient descent algorithm is used to converge to a final value. XGBoost is an open-source algorithm that uses boosting to enhance the performance of DTs [38]. It computes the best split based on a histogram and pre-sorted algorithm [37]. The advantage of this method is that stochastic gradient descent is well suited to minimise the error and the inclusion of multiple models inherently targets predictions which are not accurate [37]. At the same time, this increases their propensity to over-fit and the boosting procedure can be computationally expensive (memory and time) [39].

2.3.7 Artificial Neural Networks

Artificial Neural Networks (ANNs) are popular supervised learning models inspired by the structure of human neurons [40]. Neural networks (NN) usually consist of at least three layers: input, hidden and output. Each layer consists of multiple neurons connected in parallel and the purpose of each neuron is to learn a different part of the data [22]. When the neurons are connected in a specified sequence, they can be used to "learn from higher order features" and can "theoretically model an arbitrary function over a set of inputs" [22]. This makes them suitable for use in regression tasks. Figure 2.7 provides an illustration of an example of an ANN.

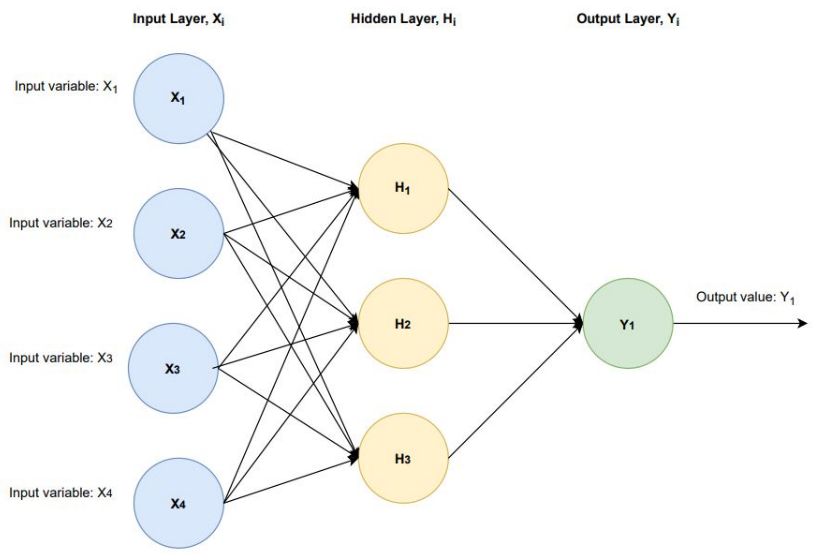


Figure 2.7: Diagram showing the structure of a 4-3-1 ANN. The ANN has three layers: input, hidden and output. The input layer X_i consists of four input variables $X_{1...4}$ which are all connected to the three neurons in the hidden layer H_i. Finally, the three neurons $H_{1...3}$ are all connected to a single output neuron Y_1 which produces the (continuous) output value.

NNs operate in bi-directional ways: forwards and backwards. The "forward-pass" is used to generate a prediction where the NN traverses through all the neurons from the input to output layer [40]. Each input layer has one neuron per input variable, one or multiple hidden layers (consisting of one or many neurons) and one output layer (consisting of typically one but sometimes multiple neurons). Connections between neurons are assigned weights which determines the rate of transfer of information and hence induces a learning effect [22]. Therefore, the "backward-pass" is used to adjust the weights between the neurons using a back propagation algorithm which uses the gradient of the loss function (at the output layer) to propagate the error and adjust the weights such that the error is minimised [40].

ANNs have been widely used across many domains for regression tasks primarily because of their high prediction accuracy when compared to other supervised methods [41]. There have been many studies investigating complex and dense architectures in the field of deep learning [42]. However, they also have some known drawbacks. NNs typically require a lot of data to train which can be problematic with smaller data sets [41]. This also means that training NNs is time consuming though this is not a problem once they are deployed. On a more general level, NNs are difficult to interpret because they usually involve many hidden layers (each with multiple neurons) [41]. This may deter their use in the clinical domain.

2.3.8 K Nearest Neighbours

K-Nearest Neighbours (KNN) is another popular supervised learning algorithm that uses a (user-specified) distance function to cluster K nearest data points together and averages all points in those groups to make a prediction [43]. In the literature, they are described as *lazy learners* because they keep a local cache of training examples and postpone the prediction until "an explicit request is made at test time" [22]. This is also known as *instance-based learning* [43]. There are three commonly used distance functions for regression problems [43]: Euclidean, Manhattan and Minkowski.

The value of K is a hyperparameter that can be tuned to achieve best model performance. Due to the nature of the method, this will often depend on the underlying distribution of data in the training set [43]. There exists a trade off between high and low values as the former makes the model "less sensitive to training data (lower variance) and increases the smoothness of decision boundaries (higher bias)" and vice versa for the latter [22].

While KNNs are typically used for classification problems, their strengths are equally applicable for regression problems. The instance-based learning means that the models are robust to changing distributions in the data since the algorithm is only called at test time [43]. At the same time, this is a limitation of the model because of space and time complexities as they have to store all training examples and re-calculate the neighbours in every test call [43]. They are also sensitive to feature scale as they rely on distance-based metrics to evaluate nearest neighbours in the data set.

2.3.9 Support Vectors

Support Vector Machines (SVMs) are models that use a *hyperplane*, which is a subset plane defined in high dimensional vector space, to separate data points while maximising the margin between the two clusters [44]. The data points which are used to define the hyperplane are known as *support vectors*. SVMs are often used to address an ubiquitous problem: non-linearity [44].

If the training data exhibits non-linear characteristics, in other words a hyperplane cannot be used in the same dimension to separate the data, then SVMs are able to map the data points into a higher dimensional space using *Kernel functions* [44]. These functions preserve the underlying distributions while introducing linearity for hyperplane separation. There are two commonly used kernel functions: polynomial functions and Radial Basis Functions (RBFs) [44]. Their purpose is to enable the algorithm to run in the original dimensional space without the need for explicitly translating to a higher dimension. This is especially useful for regression problems with a large feature space.

The ability of SVMs to handle large and non-linear data gives them an advantage over many of the other supervised learning models. However, the advantages also come at a cost. While the use of kernel functions is intelligent, they also increase the space and algorithmic complexity of the model so they are not suitable for fast prototyping [44]. Likewise, the user has to specify the kernel functions which are complex to understand. This makes SVMs, in general, difficult to interpret and understand. Like ANNs, this is a limiting factor.

2.3.10 Evaluation metrics

Regression models require a specific set of evaluation metrics to statistically measure their performance. These metrics provide a good basis for comparison and are defined as *loss functions* in the literature [45]. All ML models are objectively designed to minimise the error defined by the respective loss functions. Many of these metrics already exist as part of the Scikit-learn library [33]. This section will provide an outline of the metrics.

The notation used throughout the section will be as follows:

n = total number of data points
y_i = actual output value
$\hat{y}_i$ = predicted output value

Mean Absolute Error (MAE): One of the most widely used evaluation metric is MAE [46]. It calculates the sum of the *absolute* difference between the actual and predicted values averaged across the total number of data points. It is mathematically expressed using Equation 2.2:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{2.2}$$

MAE is popular because it is less sensitive to outliers in the data set and does not consider the direction (i.e. negative or positive difference between the values) [46]. Low MAE scores indicate high model accuracy.

Mean Squared Error (MSE): Like MAE, MSE calculates the sum of the *squared* difference between the actual and predicted values averaged across the total number of data points. Equation 2.3 shows the mathematical formula of MSE:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.3}$$

Low MSE scores indicate high model accuracy. The use of squared sum means MSE penalises large differences between the actual and predicted values more harshly. It also means that the unit of error is squared which may make it difficult for direct comparisons. Hence, the root of MSE is often preferred.

Root Mean Square Error (RMSE): RMSE provides a direct measurable quantity of error that is consistent with the units of the values in the data set. It is defined as per Equation 2.4:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{2.4}$$

However, like MSE, RMSE is also susceptible to outliers which can skew the error values. Hence, a further modified version of RMSE is studied.

Root Mean Squared Log Error (RMSLE): The difference between RMSE and RMSLE is that a log function is taken of the actual and predicted values, as shown in Equation 2.5 [47]:

$$RMSLE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(\hat{y}_i + 1) - \log(y_i + 1))^2} \tag{2.5}$$

There are two notable differences between RMSE and RMSLE. First, RMSLE is more robust to outliers because of the log term which scales down the magnitude of difference. Second, RMSLE penalises underestimation of values more harshly than overestimation [47].

Relative Squared Error (R^2): R^2, also known as *coefficient of determination*, is used to evaluate "the percentage of variation explained by the relationship between the input and output variables" [48]. In other words, it measures how well the regression model fits on the data set. It is mathematically expressed as in Equation 2.6 [48]. Note: the denominator term is the sum of difference between the actual value and mean value of model outputs ($\bar{y}$).

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{2.6}$$

The R^2 metric is usually used if the model exhibits linear characteristics as in with linear regression. It gives an indication of how well the data points are scattered around the regression model [48].

2.3.11 Use cases in clinical domain

There are numerous applications of ML in the clinical domain; this section will focus on previous studies that used ML methods to predict missing laboratory test values. A study conducted by Luo *et al.* (2016) [30] used four regression methods: linear, Bayesian linear, RFs and Lasso to predict missing Ferritin values in laboratory test data. These values were subsequently used to classify the predicted value as "normal" or "abnormal". The study concluded that RFs (with MICE [49], a type of MI) achieved the best prediction performance, closely followed by Lasso (with missForest). Both led to a high classification accuracy.

Another study carried out by Bertismas *et al.* (2018) [50] formulated the imputation problem using an optimisation approach. The authors tested KNN, SVM and DTs against a suite of other imputation methods (mean, Bayesian principle component analysis), and concluded that the former methods consistently produced low average MAE and RMSE scores. The study also found that the proposed methods were computationally faster (during prediction) with SVM converging to an estimate within seconds for data sets of magnitude 100,000. Other studies have also shown promising performance results when using KNN [51] and SVM [52].

A different study, conducted by Sharpe *at al.* (2005) [53], used NNs to predict missing values in Thyroid laboratory test results (Thryotoxic, Hypothyroid and others). The study concluded that using NNs was effective - producing a correctness score of 92.4% with a similarly high classification accuracy rate. A different study on NN, carried out by Smieja *et al.* (2018) [54] demonstrated that adding a dedicated input layer which estimates missing values using the EM algorithm leads to a better performance on incomplete data than using standard imputation techniques.

All experimental studies and literature indicate that many ML models have been successful in imputing values in laboratory test data. These findings will help to select the methods that will be used in the project to compare with simple median imputation and probabilistic graphical models (PGMs).

2.4 Probabilistic Graphical Models (PGMs)

Probabilistic Graphical Models (PGMs) are models that capture conditional dependencies between (random) variables and provide a concise way to encode joint variable probabilities using graphical structures [55]. In the context of this project, probability provides tools to model *uncertainty* which arises from missing data. This section will present the key concepts and ideas behind building PGMs that explains how these models function and key considerations that need to be made.

2.4.1 Overview

PGMs use the underlying probability distributions in the data to represent complex relations between different random variables [55]. They have been widely adopted across many domains because of their ability to model complex inter-dependent relations in a graphical format [56]. The structure of a PGM can be learned by training it with data (similar to ML methods) or by (manually) specifying the probabilities to build the structure. The structure can be further modelled by two types: *Directed Graphical Models (DGMs)* or *Undirected Graphical Models (UGMs)* [57]. This project will focus on the former also known as *Bayesian Networks (BNs)*.

BNs can be modelled using two components: nodes and (directed) edges. Each node represents a random variable and hence has an associated probability distribution [58]. Likewise, the edges represent dependency/*causality* relationships between the random variables; in other words, *conditional* probabilities [58]. When a BN is learned, each node is assigned a *Conditional Probability Distribution (CPD)* that encodes all the relevant joint distributions for that node [57]. Note: BNs are classified as *Directed Acyclic Graphs (DAGs)* because they (theoretically) cannot model cyclic relations between variables [58]. Figure 2.8 illustrates a (very simple) example of a BN.

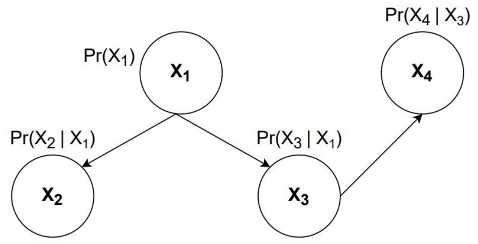


Figure 2.8: Diagram showing a very simplified example of a BN. There are four random variables $X_{1...4}$ in this BN each with an associated probability. The arrows represent *causal* relationships and are used to determine the CPD for each of $X_{2...4}$. X_1 is independent and hence cannot be modelled by any CPDs - its probability only depends on itself.

For the purposes of this project, there are three main concepts that need to be addressed: structure learning, parameter learning and inference. The former two relate to building the graphical structure of BNs and the latter is used to predict missing values using those that have been observed.

2.4.2 Structure learning

In Section 2.4.1, an overview of structure learning was provided which can happen in one of three ways: *knowledge discovery* where domain expertise is used to specify the nodes and causality relationships, *automated learning* where the model discerns patterns from the data to build the structure to capture the dependencies or *hybrid* where a combined approach is used. Knowledge discovery is suitable for scenarios where the task has been well studied and defined with existing evidence on how different variables influence each other. For example, a clinician can specify symptoms of a disease which can be used to build a parents-child relationship (where parents are symptoms and child is disease) to model the causality. On the other hand, automated learning uses an automated approach to learn the dependencies which are mapped to a graphical structure.

There are advantages and disadvantages to both approaches. While it is desirable to build a BN manually using expertise, it is equally difficult to source both domain and probabilistic expertise and it is unlikely to come from a single source. Moreover, information and data changes continuously which would require the structure of the models to be regularly changed which is time consuming and mundane. This is especially true in the clinical domain where symptoms and diseases are continuously reviewed (and changed) based on ongoing research and discoveries. In contrast, building BNs by learning from data is comparatively easier because of widespread availability of patient data in the form of EHRs [1]. Automated structure learning, however, requires careful selection of algorithms and methods that is suitable for the domain and nature of the data set. As such, the project will predominantly focus on automated structure learning methods but discuss the merits of a hybrid approach where appropriate.

In the literature there are three main categories of automated structure learning algorithms [59]: *score-based* methods which formulate the structure search as an optimisation problem, *constraint-based* methods which use conditional independence tests to model dependencies between random variables and *hybrid* methods which use both. These algorithms can be mathematically studied using probability theory but for sake of brevity the intuition behind these algorithms is presented here. For complete mathematical derivations see [59].

Score-based: Score-based methods first estimate an initial structure of a BN and iteratively change the structure by adding, removing and reversing the direction of the edges until a candidate BN is selected with the maximum score. The scoring usually measures "goodness of fit" [60]. A simple way to estimate these structure is exhaustive search where each permutation of the graph is scored. However, this approach is NP-hard and has a factorial-exponential complexity that scales with the number of nodes (features) [61]. As such, the problem is simplified using greedy search algorithms such as *Hill Climbing Search* (HCS) and using scoring heuristics such as log-likelihood, Bayesian score and Bayesian Information Criterion (BIC) [60]. Some scoring metrics such as BIC are better suited when prior or "expert knowledge" is not available as they inherently use *information entropy* (measure of gain in information) for scoring [62]. Despite using heuristics, ultimately they pose and solve an optimisation problem which may not converge to a final solution [61].

Constraint-based: Constraint-based methods use conditional independence between the features to find the structure that explains those relations [60]. The method works by creating an undirected graph and conducting a series of independence tests to check the significance level is above a defined threshold [61] for all the nodes. The edges which do not pass the significance level of the tests are removed until a final structure is left. The PC algorithm is the most widely regarded constraint-based algorithm which has been successfully used in various domains to generate optimal BN structures [63]. However, their biggest limitation is that they scale exponentially with number of nodes making them unsuitable for high dimensional data sets. Further, the topology of the graph is dependent on the variable ordering in the data set meaning they give non-deterministic results [60].

Hybrid: Hybrid methods combine both score-based and constraint-based methods to find the most optimum structure. The constraint-based method is first used to initialise an undirected graph structure and carry out conditional independence tests. This reduces the search space for score-based greedy algorithms which further use scoring heuristics to find the candidate BN. The Max-Min Hill Climbing (MMHC) [64] is the most popular example in literature but other algorithms such as Hybrid HPC (H2PC) [65] have also been proposed. While the hybrid approach best utilises both methods, they often do not converge to the structure with maximum score and return a sub-optimal result [65].

2.4.3 Parameter learning

Parameter learning is the second concept that follows structure learning. Parameter learning uses the learned BN structure and data set to estimate conditional probabilities for each state and builds the CPDs. In the literature there are two main approaches: *Frequentist* and *Bayesian* [66]. An important underlying assumption in both approaches is the availability of complete data set to infer the probabilities. The Frequentist approach most commonly uses *Maximum Likelihood Estimator* (MLE) which uses the log-likehood function to assign probabilities to values that maximise the function [66]. Ultimately, this solves an optimisation problem where the probability distribution assigned to each variable/node best fits the observed distribution for that variable. This is an intuitive method to assign probabilities because it relies on the frequency of values within each variable. At the same time, this approach has a propensity to over fit because the values with highest frequencies dictate the probability distributions [66].

The second method is using Bayesian statistics to estimate the parameters. Bayesian estimation relies on Bayes's Theorem which relies on using *prior*, *evidence* and *likelihood* to estimate the posterior probability [67]. This can be mathematically expressed as:

$$Pr(\theta|E) = \frac{Pr(E|\theta) \times Pr(\theta)}{Pr(E)} \tag{2.7}$$

where $Pr(\theta|E)$ is posterior, $Pr(E|\theta)$ is conditional likelihood, $Pr(\theta)$ is prior and $Pr(E)$ is evidence.

Bayes's Theorem characterises the estimation of parameters because the method tries to use the prior probability distribution to calculate a posterior for each variable. Evidently, this requires the prior to be defined either using human expertise or through automatic learning methods. In automatic learning the priors are estimated with uniform distribution which gives each variable an equal probability so the method obtains different likelihoods based on the values found for each variable from the data [66]. The advantage of this approach is that it utilises Bayesian statistics which can inherently relies on statistics (such as likelihood, evidence) to make a robust estimation. This method also only requires a single data point to compute the posterior distribution making it ideal for smaller data sets [67]. However, the prior dictates the value of the posterior so it needs to be as accurate as possible.

For mathematical proof and derivation of MLE and Bayesian parameter learning methods see [67].

2.4.4 Inference

In Sections 2.4.1 and 2.4.3, the concepts of structure and parameter learning were covered which form the model learning process. Once the structure of a BN is established, it can be used to infer data that is not explicitly modelled. This makes it suitable for data imputation tasks. In the literature, there are two methods commonly used for inference: *exact* and *approximate* [68]. The former is an analytical technique whereby the joint CPDs are used to compute conditional ($Pr(X|Y)$) and maximum posterior probabilities (value of X that maximises $Pr(X|Y)$) [68]. A selection of algorithms can be used but most notably *variable elimination* is used [68]. This algorithm also uses a heuristic to find the best order of elimination as different permutations can impact computational complexity (memory and time) [69]. Approximate inference uses statistical techniques and sampling methods to obtain the same results [68].

The biggest advantage of BNs is that they are interpretable making them very suitable for application in the clinical domain. Secondly, knowledge discovery can be added such that the models use data-driven and elicit knowledge from clinicians to create an enhanced structure which is used to make the predictions. One drawback, however, is that they have a high computational complexity because of increased number of parameters which need to learned to build the structure (for example CPDs) [57].

2.4.5 Use cases in clinical domain

There have been limited studies which have tried to use BNs with laboratory test data. A preliminary study conducted by Guenfoud *et al.* (2018) [70] confirmed that BNs can be used to successfully model relations between biochemical markers and integrated as part of CDSSs. However, an in-depth empirical study on their performance is yet to be carried out. A different study conducted by Shen *et al.* (2018) [71] concluded their superiority over other algorithms in a clinical application. These studies encourage use of BNs in this project.

2.5 Software tools

This section will provide an overview of the software and libraries that will be used during the implementation phase of the project. Most of these tools have been chosen based on familiarity and past experience in other academic and industrial projects.

2.5.1 Python 3

According to the TIOBE index [72], Python [73] was awarded "programming language of the year 2020" recording a growth of 2.01%. This accolade is a testament to the widespread adoption of the language by the scientific community, particularly in the domains of data science and machine learning. Python is renowned for its easy-to-learn syntax and fast prototyping, favoured by many developers in these domains, because it allows them to focus on the development of the algorithms as opposed to handling complex work flows [74]. Perhaps the most desirable feature is the extensive range of computation packages, libraries and frameworks that are supported by the language, along with the abundance of documentations, tutorials and user community support [74]. As such, Python is a suitable and appropriate choice for this project and it should easily integrate with the PI module of EPiC IMPOC (which is also written in Python [12]).

2.5.2 Pytest

Pytest is a testing framework for Python that provides all the tools and infrastructure to run automated test cases [75]. It provides an intuitive setup to run several tests in parallel with custom configurations that suits the domain purpose. This makes it an appropriate choice for this project so that simple unit tests can be written for the implemented methods and tested in parallel at once.

2.5.3 SciPy

Scientific Python (SciPy) is a "Python-based ecosystem of open-source software for mathematics, science and engineering" [76]. It encompasses many of the libraries and tools discussed in this section, as well as its own scientific computation package for numerical methods [76].

2.5.4 Pandas

Pandas is a "fast, flexible and easy to use open source data analysis and manipulation tool" [77]. One of its biggest capabilities is handling missing data by providing tools to identify, remove and impute missing data with programmer specified values. Aside from handling missing data, the library provides the *DataFrame* data structure that is very powerful in handling different types of data sets. The library is also renowned for its performance as some methods are written in low-level C language to optimise operations [78]. All the above attributes should help to simplify data handling operations in this project.

2.5.5 NumPy

Numerical Python (NumPy) is one of the most popular computation libraries in Python and provides methods for highly complex multidimensional array operations [79]. It is most commonly used as a substitute for Python *List* because of its reduced memory space, increased performance (fast vectorised operations) and better functionality (such as linear algebra operations). Like pandas, many methods have been optimised for speed and written in C-language [79]. Again, these features are desirable for this project and will work well in tandem with the other libraries selected.

2.5.6 Matplotlib

Matplotlib is a "comprehensive library for creating static, animated, and interactive visualizations in Python" [80]. It is a graphical package that will be very useful in this project to visualise and analyse implementation results. In particular, the `pyplot` framework will be used as it includes all the necessary graphical plots and will integrate easily with the other libraries [80].

2.5.7 Scikit-learn

Scikit-learn is a machine learning library in Python that provides "simple and efficient tools for predictive data analysis" [33]. Many methods and features in this library make use of the previously described tools NumPy, SciPy and Matplotlib [33]. This library will be used universally across different aspects of the project, particularly during the early stages to generate baseline results using traditional ML methods which the Bayesian Networks can be evaluated against. An example is the (experimental) `IterativeImputer` which will be used in the early stages of the project.

2.5.8 Pgmpy

Pgmpy is an open source Python library specifically built for Probabilistic Graphical Models [81]. The library supports many different graphical models; this project will aim to use the `BayesianModel` class which provides methods to initialise, build and infer using the learned structure. The documentation is well maintained and provides several tutorials on model implementations [81]. The library is ideal for this project because it provides all the infrastructure to build the graphical models so that the focus of experimentation can be on maximising the performance of the imputation methods.

2.5.9 Networkx

Networkx is a comprehensive Python package that provides all the infrastructure to build and visualise graphical structures [82]. It provides a set of data structures and methods which can create graphical objects that can integrate with the Matplotlib library. The library is also integrated within Pgmpy which should allow visual representations of the BNs to be easily plotted. Graphs can also be configured in terms of their layout, node properties and edges which will be needed during later stages of the project.

2.6 Summary

In this chapter, the key background and preliminary materials were covered so that motivations behind the content presented in Chapters 3 - 9 can be understood. To summarise, the key topics studied in this chapter were:

- *Types of missing data*: which covered three main types - MCAR, MAR and MNAR.
- *Common imputation methods*: which covered three main methods - complete case analysis, single imputation and model based.
- *Machine learning preliminaries*: which covered data preparation, cross validation, regression models (linear, decision tree, random forests, artificial neural networks, support vectors) and evaluation metrics.
- *Probabilistic graphical model preliminaries*: which covered three main topics - structure learning, parameter learning and inference.
- *Selection of software tools*: which covered a breadth of different software packages and libraries selected for this projected.

Chapter 3

Requirements Capture

This chapter outlines a project specification which consolidates key findings from the background material and literature reviews. The main objective of the project is to design, implement and analyse data imputation methods suitable for laboratory data. Ideally, these methods should outperform simple median imputation in terms of imputation accuracy; that is, the imputed values should be closer to true values than simple median imputation. This chapter presents the requirements for the design of the imputation framework (Section 3.1) followed by the implementation details on how the methods will be programmed (Section 3.2). This chapter will also discuss how the designed and implemented methods will be evaluated via experiments (Section 3.3) to facilitate discussions on their suitability for laboratory data. The project will also document all code and findings in supporting documentation (Section 3.4) that forms part of the deliverables. Finally, a summary of the key deliverables for this project is presented (Section 3.5).

3.1 Design of imputation framework

The first objective of the project is to design an imputation framework that provides a methodology to impute missing values in laboratory data. The only design constraint that the framework needs to adhere to is that it must support two imputation approaches, ML based methods and BNs, so that they can be studied and compared in terms of their imputation performance. Both approaches should exploit the relations between variables to impute missing values using their respective mechanisms. As mentioned in Section 2.3.1, test results in laboratory data are continuous values so the framework should formulate the problem as a regression task. The regression models and BNs studied in background material should be tested and evaluated for that purpose. The project should evaluate the suitability of the imputation framework and its application during the experiments.

3.2 Implementation of imputation framework

The second objective of the project is to implement the steps designed in the imputation framework to facilitate their testing on a real-life laboratory data set. The project code and implementations should be compatible with Scikit-learn [33]. The library provides robust infrastructure for most of the steps that will be required in the framework including optimised implementations for many vanilla regression models that can be directly used. This will create a unified framework to enable all methods to be evaluated with the same steps in a single Python script. This means that all other libraries such as `pgmpy` [81] which are (by design) not compatible with Scikit-learn will need to be wrapped in methods that Scikit-learn requires to build a systematic pipeline. This will reduce redundancy in scripting and make the implementations more readable. The wrappers should exploit (OOP) *inheritance* to build on top of existing functionality provided by the external libraries. This means, in effect, a Python library will be created which provides all of the imputation methods and can be easily imported as a package for future use.

The project will focus on the analysis of (experimental) results though the implementation of the framework steps is a prerequisite. The ultimate aim is for the project findings and implementations to be embedded into the pipeline of EPiC IMPOC's probabilistic inference module but evaluating the performance of its classifers remains outside of the project scope. It will be sufficient to show that the implemented methods outperform the median approach currently used. Nevertheless, as with any software projects, the aim is to write clean, readable and well documented code that can be easily maintained and extended in the future. The best way to achieve that is to use an *object-orientated* approach to encapsulate the core functionalities of the implemented methods.

3.3 Experiments on a real-life laboratory data set

To test the two approaches, a series of experiments should be carried out to study the performance of the different methods on a real-life laboratory data set. This will validate the functionality of the framework but also provide an insight into the strengths and weaknesses of the respective methods. The findings from the literature review indicate that state-of-the-art ML methods have previously outperformed simple imputation methods but there are fewer examples evaluating the performance of BNs. This motivates the need for this study as there is no previous work (to the best of knowledge) directly comparing the two approaches especially in a clinical application. The experimental results will be presented and discussed as part of this report; key results and metrics will also be included as part of supporting code documentation.

3.4 Supporting code documentation

Writing concise, easy-to-understand and meaningful code documentation is part of good software engineering practice. A GitHub repository was set up for this project to organise all code development and experiment results. Supporting code documentation for the implemented methods should be provided in a *Hyper Text Markup Language* (HTML) rendered format using the *Sphinx* library. This style of documentation is universally used for Python open-source contributions, libraries and packages. The documentation should accompany the code such that future collaborators can use and maintain the library in the future. The experiments will serve as examples on how to use the implemented imputation methods. The ultimate aim of providing documentation is to create a single reference point for the project where all code and experiment results can be easily accessed.

3.5 Summary

To summarise, the following are the key deliverables for this project:

- *Design*: formulation of an imputation framework which provides a methodology to investigate two approaches, ML based and BN methods, on a real-life laboratory data set.
- *Implement*: implementation of all the stages in the designed framework using Python. Development of required Python `wrappers` for universal compatibility of all the implemented method with Scikit-learn.
- *Test*: planning, execution and collection of results from experiments on a real-life laboratory data set with supporting analysis and discussions.
- *Document*: provision of supporting code documentation that is a reference point for all project code and experiment results.

Chapter 4

Analysis and Design

This chapter presents analysis and design of the imputation framework which provides a methodology suitable to investigate the outlined approaches in this project. Firstly, an overview of the imputation framework is provided (Section 4.1) which is followed by more detailed justifications of the key design decisions for all the framework stages. This includes discussion on feature selection (Section 4.2), selection of prefilling strategies (Section 4.3), pre-processing techniques (Section 4.4), model learning which trains the respective ML and BN models (Section 4.5) and imputing the values by predicting them using the other features (Section 4.6). Finally, a summary of the chapter is presented (Section 4.7).

4.1 Overview of imputation framework

Figure 4.1 provides a graphical overview of the stages in the imputation framework which were designed to investigate the two approaches: state-of-the-art ML based methods and BNs. The framework has been formulated as a classic ML workflow because the aim is to train the model(s) to capture the variable relations and exploit those to predict (impute) the missing values. The framework was designed in a modular way so that: (a) dependencies between steps could be minimised, (b) code implementation could be parallelised and (c) framework can be extended or reduced as per use case including application outside of the healthcare domain.

The framework comprises of five main stages with two (distinct) workflows to account for the two approaches. In Figure 4.1, this is depicted by one workflow on the left (ML based) and one on the right (BNs). In stage one, the data set is formatted and prepared for processing following which the features of interests are specified to reduce the dimensionality of data. The framework splits for the next three stages. In stage two, separate strategies are used to prefill the incomplete data set using either the feature median for ML based (left) or EM algorithm for BNs (right) methods respectively. In stage three, the data is pre-processed to facilitate model learning. In the following stage, for ML model learning, a selection of regression models (studied from background material) are learned from the (training) data for each variable in the feature space and subsequently used to impute missing values. The same step is applied for learning BNs which require an extra intermediate parameter learning step to generate conditional probabilities for all the nodes. As such, it is important to separate the two workflows to accurately differentiate between the two approaches. Finally, in stage five, the trained models are used to predict or estimate the missing values.

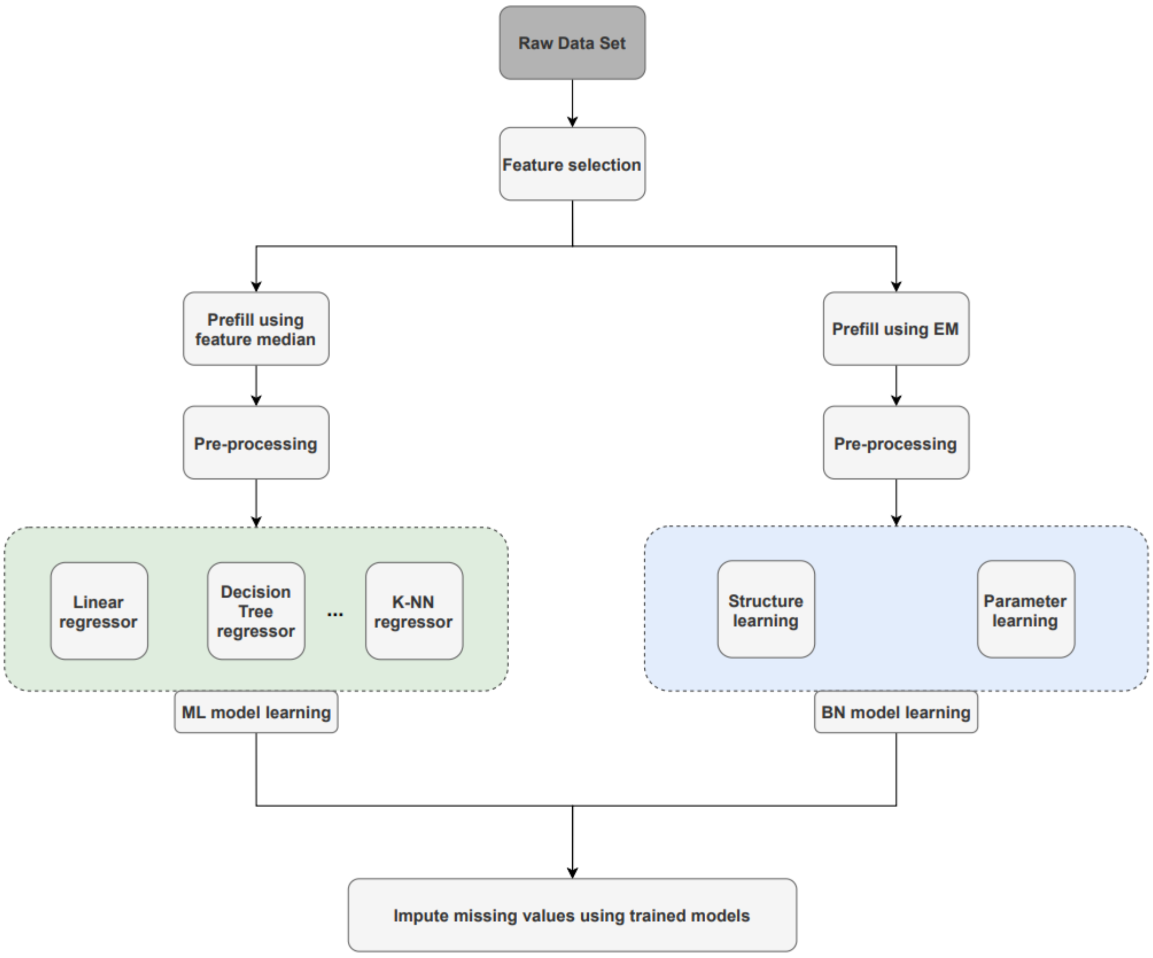
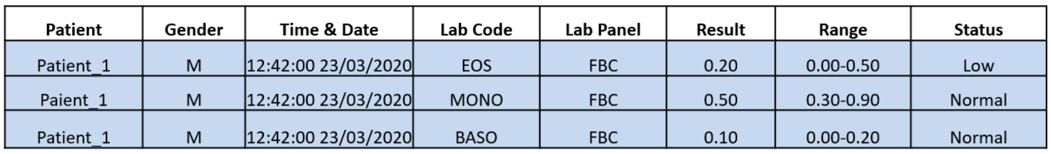


Figure 4.1: Diagram providing an overview of the designed imputation framework. There are five main stages in the workflow: (a) selecting features to impute, (b) prefilling using feature median for ML based (left) or EM algorithm for BNs (right) methods, (c) pre-processing the data set (left, right) to make it suitable for model learning, (d) model learning for ML based (left) and BNs (right) methods and (e) imputing missing values using the trained models.

The framework has been designed with the intention of direct usability on a real-life laboratory data set that contains missing values. However, for the experiments, the framework will be modified as a complete data set (which contains ground-truth values) will be used to measure the error of the imputed values. Therefore, there would be a need to (artificially) remove values from a complete data set to simulate missing data taking into account the missingness mechanisms studied in Section 2.1. The modular design of the framework means such methodology changes can be easily introduced without drastic changes to the rest of the pipeline. It should be noted that under the MAR assumption, the prefilling strategy does not introduce bias and hence does not impact pre-processing (which transforms and scales the data with the prefilled values). In fact, pre-processing methods in Scikit-learn require complete data sets else they either (a) drop rows with missing values or (b) return an error.

4.2 Feature Selection

Laboratory data is longitudinal by design which means each row or record in the data set represents a single timestamped test result for a single patient as shown in Figure 4.2. It is common to find several variables, both continuous and categorical, that measure some parameter related to each patient. As such, the raw data set usually contains range of test results for different biochemical markers which are grouped into their respective panel codes. Examples of test panels include Full Blood Count (FBC), Hepatic Function (Liver) and Renal Function (Kidney). Clinicians use the biochemical marker values within each test panel to medically evaluate the status of a patient and accordingly devise a treatment plan. For this reason, the availability of the actual value for each biochemical marker is imperative for accurate medical diagnosis. Prior to feature selection, the raw data needs to be cleaned to discard variables that are not of interest (for example Gender) and transformed using table pivots so that it is suitable for further processing.



Patient	Gender	Time & Date	Lab Code	Lab Panel	Result	Range	Status
Patient_1	M	12:42:00 23/03/2020	EOS	FBC	0.20	0.00-0.50	Low
Paient_1	M	12:42:00 23/03/2020	MONO	FBC	0.50	0.30-0.90	Normal
Patient_1	M	12:42:00 23/03/2020	BASO	FBC	0.10	0.00-0.20	Normal

Figure 4.2: Diagram showing a snippet of raw laboratory data. There are many fields in the data set which contain different types of data and the test results are presented one per row.

Feature selection requires the test panel to be specified so that the biochemical markers for that specific panel are selected as feature inputs to the imputation methods. This increases the likelihood of the models finding and exploiting relations between biochemical markers under the specified panel. This does mean that panel codes need to be defined but this information is widely available and easily accessible. With this being said, feature selection is not strictly imposed in the framework but rather a (manual) method to reduce dimensionality of the data to make the results more interpretable. It is also likely that clinicians examine one panel at a time so the same rationale is applied here.

4.3 Prefilling

Prefilling is a strategy which temporarily fills missing values using the specified imputation method. It is important to note that prefilling is just a technique to generate complete data set while the models (which are subsequently learned) are used to perform the actual imputation. The fidelity of ML and BN models are directly related to the quality and quantity of data (assuming regularisation is used to prevent over fitting) and therefore this results in better imputation accuracy. The workflow splits at the prefilling stage as different methods were selected for the two approaches.

4.3.1 Machine learning methods

For ML methods, the missing values in the data set are prefilled using the feature median value. For clarity, the feature median is the median value of all other *observed* values for that feature (variable). While this is a simple imputation method, previous studies have empirically shown that it significantly improves the accuracy of imputation [83] [84]. In this case, the median was selected instead of the mean as it is more robust to outliers [85]. Other approaches such as using multiple imputation methods were considered but not eventually used because they have large computation overheads. The benefits of using a simple method like median outweigh its limitations discussed in Section 2.1.

4.3.2 Bayesian Network methods

The EM algorithm was selected for BN methods instead of imputing the feature median because it inherently maximises the log-likelihood parameters. It is used as a prefill strategy to improve the quality structure and parameter learning upholding the MAR assumption in laboratory data. By design, EM uses the complete values to estimate the parameters and iteratively adjusts the estimated parameters until they converge to a global maxima. Empirical studies have also shown the improvements in imputation results when EM was used prior to structure learning [86] [87]. The benefits of this approach is that it inherently uses conditional expectations to generate the parameters (that maximise log-likelihood) which maximises the probability of finding the most optimal BN structure [21]. At the same time, the rate of convergence is largely dictated by the first parameter estimate which may converge slowly and only to a local maxima.

4.4 Pre-processing

In ML applications, pre-processing is a common step that prepares the data in the right format for model learning and increases model generalisation [88]. It is even more important in the clinical domain, such as for laboratory data, where the biochemical markers are (usually) continuous values with different ranges. This can be problematic in two ways: the first is the scale of the variables which is unsuitable for running some ML models. More precisely, distance-based ML algorithms (such as K-NN) are sensitive to the magnitude of the values meaning that they will give erroneous results when evaluated on the test set. The second challenge is the presence of outliers typically due to misreporting (human error) or inaccurate recording of test results (diagnostic device error). The latter also contributes towards the MAR assumption for missingness in laboratory data. Other algorithms such as DTs are invariant to feature scale and do not need any pre-processing.

To tackle the discussed problems, feature scaling and outlier removal techniques can be used. For feature scaling two approaches are considered: data standardisation and normalisation. The former, shown in Equation 4.1, transforms each feature so that it has zero mean (μ) and unit variance (σ^2) (i.e. a Gaussian distribution) while the latter, shown in Equation 4.2, transforms such that feature values lie between [0, 1]. Note the denominator in Equation 4.1 shows the standard deviation (σ).

$$X' = \frac{X - \mu}{\sigma} \tag{4.1}$$

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{4.2}$$

The framework uses Z-Score based methods to detect and remove outliers. Outliers are classified using the inter-quartile rule (IQR) with the lower and upper Z-Scores defined as [$Q1$ - $\alpha * IQR$] and [$Q3$ + $\alpha * IQR$] where α is the strategy coefficient respectively. Note that $Q1$ and $Q3$ are the lower (25th percentile) and upper (75th percentile) bounds respectively. By default, the framework uses the most widely accepted value of α as 1.5 but this can be increased or decreased depending on an aggressive (or passive) strategy. This emphasises an important point that the ultimate choice of pre-processing steps are largely driven by the nature of the data and the candidate algorithms being studied. It should be noted that the framework removes outliers *before* applying feature scaling to prevent them from skewing the statistics used to calculate the standardisation or normalisation formulas. For BNs, pre-processing only constitutes removal of outliers as they do not require the data to be scaled prior to model learning.

4.5 Model learning

Model learning takes place on the (complete) data set generated from the prefilling and pre-processing stages. This is the core of the framework because these models are used to subsequently replace the prefilled values using predictions from the respective estimators. Model learning is separate for the two approaches to reflect their different training mechanisms.

4.5.1 Machine learning methods

In ML model learning, the generated data set from prior stages is used to learn the regression models. The benefit of this approach is that multiple regression models are learned, each for a different set of observed variables, but at the same time may lead to a higher spacial and temporal complexity depending on the size of the feature space and permutations generated. For instance, given a data set has X features and n features are missing for a particular record then $X - n$ features will be used to build a regression model to predict n values. The process repeats for each record in the data set and therefore scales linearly with the size of the data set.

Simple models such as linear regressor are inherently faster (in terms of training time) than ensemble methods such as RFs making them more suitable for imputing large laboratory data sets. The latter, however, are more robust to outliers and may be better able to model non-linear relations between the variables. Table 4.1 summarises the default regression models supported by the framework though this selection is non-exhaustive as the modular design allows more or less models to be used. The selection of these models is dictated by findings from background material and separate empirical studies discussed in Section 2.3.11.

4.5.2 Bayesian Network methods

As discussed in Section 2.4, BN model learning consists of two steps: structure and parameter learning. The framework separates the two steps as they each have a distinct purpose; the former captures the dependencies or causalities between the variables while the latter estimates the *conditional probability distributions* CPDs for the learned structure. The framework was initially designed with both steps combined however this eliminated the possibility of the user manually defining the BN structure. This was changed to allow the user to review and change the structure if needed. The rationale behind this design change was largely enforced by its intended application where clinicians may potentially benefit from having the ability to use their domain expertise to remove erroneous or add any missed relations between the features. This would enhance the quality of modelling and better fulfill the purpose of BNs. Nevertheless, the framework is completely automated and by default runs

Regression model	Imputation method	Imputation type
Linear (LR)	Linear	Multiple
Decision Trees (DT)	Decision tree	Multiple
Random Forests (RF)	Ensemble	Multiple
XGBoost (XGB)	Ensemble	Multiple
Support Vector (SVR)	Support vector	Multiple
K-Nearest Neighbours (K-NN)	Nearest neighbour	Multiple
Multi-Layer Perceptron (MLP)	Neural network	Multiple
Simple Median	Feature Statistic	Single

Table 4.1: Table showing the regression models supported by the framework. Eight regression models are supported with their respective imputation method and type described. Multiple imputation means multiple models are learned for each set of missing features. This is different from simple imputation methods such as median which only impute once. Note that simple median is included as a regression model as it can directly be used to impute missing values.

structure and parameter learning consecutively.

The framework uses a score-based method to learn the structure of the BN from the data. A design decision was made to use this method on the premise of an empirical study carried out by Stefano *et al.* (2017) [89] which found that constraint-based method generally tend to under fit. For the same reason, hybrid methods were not considered either. Following the selection of score-based method, two further design choices were made namely the search strategy and scoring metric. *Hill Climb Search* (HCS) was selected as a suitable search heuristic which uses a greedy local search strategy to iterate through different BN structures (by adding, removing and reversing directions of different edges) until the search converges to a structure with the highest possible corresponding score [81]. Aside from the greedy search this heuristic is efficient for data sets with large number of features.

Selecting the scoring metric was more difficult as it usually varies on the intended use case. In this framework, *Bayesian Information Criterion* (BIC) was selected because like EM it uses the maximum log-likelihood method to score the models. More importantly, BIC penalises higher model complexity [90] and therefore in combination with HCS only selects the most simplest model. This is desirable for panels which have large features such as FBC (that have 16 biochemical markers). It is important to note that the fidelity of BIC improves with the size of the training set which is ideal as laboratory data sets are usually large. This does imply that BIC can oversimplify the model structure if trained on a smaller data set as the greedy search would not produce enough combinations to reach an optimal model [91].

Parameter learning estimates the CPDs for each of the nodes in the structure found from the previous step. This is a prerequisite for inference which relies on CPDs to estimate values. The framework learns the parameters by maximising the Bayesian Dirichlet (BD) using the *Bayesian-Dirichlet equivalent uniform* (BDeu) score [92]. The intuition behind this scoring function is that it maximises the posterior probability of the structure starting with the assumption that all nodes have a uniform prior i.e. they all have the same probability. This scoring function is frequently used in practice and usually when no prior knowledge is available as is the case here. While it is possible for the user to also manually specify the CPDs it is infeasible for this task because (a) test values vary depending on the individual physiology of the patient and (b) it is impractical to map probabilities for all types of cases/patients.

Despite the extra intermediate parameter learning step, one clear strength of this method is that only a single model is learned for the entire data set. That is, a single BN that captures all of the relations as opposed to learning a new regression model for a different set of missing features. This simplifies the imputation process by making the flow of information more tractable.

4.6 Imputing missing values

The final stage uses the trained models to predict and replace the prefilled values to return a complete imputed data set. The process of predicting, however, is inherently different for the two approaches. Prediction is more straightforward in ML methods because the trained regression models simply use values of all the other observed features (independent variables) to predict a value for the dependent variable. This process repeats for all the features until the entire data set has been imputed.

Prediction in BNs is an inference task because the joint probabilities of the observed variables are used to infer values for features of interest. The actual value is inferred by first computing the probability distribution for the feature of interest and then selecting the value that corresponds to the highest computed probability. The framework uses the *variable elimination* (VE) algorithm based on findings from Section 2.4. This is an exact inference algorithm meaning that the probabilities that are learned during parameter learning are actually used instead of approximating new values. This reduces memory overheads and redundancy in calculating new parameters for the final stage of the framework.

The intuition behind VE is to marginalise by combining all factors (probability distributions) that are related to the feature being predicted to create a single factor that describes the probability distribution of all values that feature can take [69]. The value which has the highest corresponding probability is selected and returned as the predicted value. An important design decision was selecting the evaluation heuristic to decide the ordering of elimination for which the *minimum-fill* method was selected because the removal of nodes from the graph during elimination should minimise impact on nodes that are not eliminated [69]. As such, the VE algorithm is best suited for inference on laboratory data.

The imputed data at this stage can be analysed to determine the performance of the applied methods including a comparison of the two approaches. The unified framework design means that both approaches share the same initial steps and therefore start with the same (incomplete) data set. This validates any comparisons that are made between the two approaches. It should be noted that the output data set is returned in the same format as the input raw data set without any pre-processing applied.

4.7 Summary

To summarise, this chapter presented the five main stages designed in the imputation framework. These stages represent a methodology which should be followed to correctly impute missing values in laboratory data. The following summarises the key stages in the designed framework:

- *Stage I*: feature selection to specify the laboratory panel of interest.
- *Stage II*: prefilling of the incomplete data set for ML based (feature median) and BNs (EM algorithm).
- *Stage III*: pre-processing the data set by applying transformations and scaling.
- *Stage IV*: model learning which trains the models to capture patterns and relations in the data.
- *Stage V*: imputing missing values using the trained models.

Chapter 5

Implementation

In Chapter 4, the analysis and design of the proposed imputation framework was presented including a justification for selection of methods and algorithms for each of the five stages. This chapter provides the implementation details on how the different stages of the framework were implemented. Since Python was selected for this project, all implementations were done in the same language using the appropriate selection of libraries where necessary (Section 2.5). This chapter provides an overview of the implementation process and methodology (Section 5.1) and then provides implementation details for each of the five stages in the framework including: feature selection (Section 5.2), prefilling (Section 5.3), pre-processing (Section 5.4), model learning and predictions for both ML (Section 5.5) and BN (Section 5.6) methods. Finally, a summary of the main implementation contributions are presented (Section 5.7).

5.1 Overview of implementation

Implementation of the project code was done in an incremental and agile way. The methodology for each of the stages was to implement a simple working model at first to ensure it functions correctly and then iteratively improve the code keeping in mind best development practices. Basic unit testing was carried out using the `pytest` framework [75] to aid development. Where possible, the implementations directly use available libraries from Scikit-learn to fulfill the requirements set out in Section 3.2. The library is widely used and regularly maintained which justifies its use for the framework to build a robust infrastructure. Scikit-learn was predominantly used for stages one and three with more implementations done for stages two, four and five of the framework. The implementations routinely use the `pandas` library to handle data using the `DataFrame` data structure which can easily manage large volumes of data. This section describes the implementation details for each stage in the framework.

5.2 Feature Selection

A `TidyWidget` class was provided for this project which transforms the raw data set into a neat structure where the features form the columns of the data. As mentioned previously, the raw data set is unsuitable for modelling because it is provided in a longitudinal format where the test values are not aggregated by test type. `TidyWidget` first collects test values for each test type and then uses table pivots to order the test results by column. See Section

6.3.1 which visualises this transformation on a real-life laboratory data set.

An optional but recommended feature in the framework is for the laboratory panel to be defined. This can be defined as a list of strings containing the abbreviated laboratory test codes; an example is "RBC" for Red Blood Cell. This list can be subsequently used to only select a subset of laboratory test results corresponding to that panel. The advantage of using a `DataFrame` allows this to be easily done using the *indexing operator*. A selection of laboratory panels and their corresponding test codes have been defined in the experiment scripts. For added flexibility, the test panels are not verified by the code so the user has the freedom to define a custom test panel list as long as the test codes are part of the raw data set.

5.3 Prefilling

Prefilling is a strategy to impute an incomplete data set to simulate a complete data set. The framework supports two prefilling methods which are separate for the two workflows. As such, different prefilling methods were implemented. Scikit-learn provides a set of *transformers* that can be used for prefilling [33]. In Scikit-learn, each transformer is implemented as a class which inherits from `BaseEstimator` and `TransformerMixin`. These parent classes provide Scikit's *Application Programming Interface* (API) for compatibility. Every transformer has (at least) the following methods:

- `fit`: fits the transformer on the data and learns the relevant parameters. For example, median values of every feature in the data set.
- `transform`: uses the learned model to transform the data. It is necessary for `fit` to be called before `transform`.
- `fit_transform`: performs `fit` followed by `transform` on the same data for increased efficiency.

For the ML based work flow, Scikit-learn's `SimpleImputer` transformer was selected because it provides an efficient way to impute missing features with the specified strategy - in this case it was the median. In fact, the feature used for ML model learning which is described in Section 5.5 already has this imputer integrated within it.

For the BN workflow, implementation was required for the EM prefilling algorithm which was written as: (a) a callable function that returns the imputed data set for it to be used directly and (b) a transformer class which inherits Scikit-learn's the aforementioned parent classes to make it compatible with the library. In both implementations, array-like structures such as `DataFrame` are supported but the algorithm largely operates using Numpy arrays as it is computationally optimised.

The algorithm follows the same steps outlined in Section 2.2.3 and iterates until the estimates converge. Figure 5.1 shows the main steps of the designed algorithm. At the initialisation step, an estimate for the mean (μ) and standard deviation (σ) is computed for all non-missing values in each column. In the Expectation-step, a new estimate is found for the same statistics using the previous estimated value. This step is only valid after the first iteration as there is no previous estimate on the first run. The Maximisation-step uses those computed statistics to impute the missing value with an estimate value drawn from a Gaussian distribution using Numpy's `random.normal` function. This repeats until the parameter estimates converge i.e. the percentage change between the current and previous imputed value is less than the *tolerance* threshold. By default, the tolerance threshold is defined at 0.01 but this can be configured at function call or object instantiation (when used as a transformer). There is also a defined maximum iteration parameters (*max_iter*, default at 50) to prevent an infinite loop in the case the algorithm diverges. This can happen if the initial estimate is poor which would usually happen if the feature has large number of missing values.

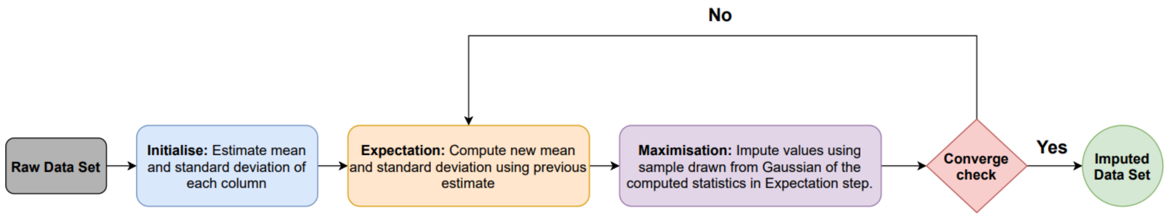

No
Raw Data Set
Initialise: Estimate mean and standard deviation of each column
Expectation: Compute new mean and standard deviation using previous estimate
Maximisation: Impute values using sample drawn from Gaussian of the computed statistics in Expectation step.
Converge check
Yes
Imputed Data Set

Figure 5.1: Diagram showing the main steps of the EM imputation algorithm. There are four main steps in the algorithm: (a) an initial estimate is made for the mean and standard deviation for each column with missing values, (b) the E-step computes a new set of the same statistics using the previous estimate, (c) the M-step uses those statistics to draw a sample from a Gaussian and impute the missing value and (d) a convergence check to determine the percentage change between the current and previous estimated value. The outcome is an imputed (complete) data set.

Some simple optimisation techniques were used to mitigate the computational limitations of the standard EM algorithm. This implementation imputes at the first iteration so that a complete data set is used to compute new statistics (during the E-step) in the next iteration. This means the algorithm constantly adjusts the imputed values, similar to MICE [49], which significantly reduces the number of iterations required to converge to a final value. It does mean that multiple estimates are made which increases computational overheads but this is negligible against the increase in statistical power which subsequently improves the structure learning process.

5.4 Pre-processing

Scikit-learn has a robust set of *transformers* that can be used for different pre-processing methods [33]. The first pre-processing step in the framework is the removal of outliers using the Z-Score bound method discussed in Section 4.4. A custom utility function was implemented to support this feature as Scikit-learn does not currently provide any methods to do so. The function signature takes in the data (in any array-like structure) and the threshold coefficient to select the Z-Score scaling (by default it is set to 1.5). The function calculates the *inter-quartile range* (IQR) for each variable but does not immediately remove the outliers based on the Z-Score bound. Instead, the function assigns *Not a Number* (NaN) to each value that is classed as an outlier. Any row (test record) with at least one NaN value is subsequently discarded but this threshold can be increased to deliberately allow missing values to remain in the data set. This provides a method to simulate missing data as well as assess the robustness of the algorithms in learning with missing values.

For feature scaling, a subset of the available transformers namely `StandardScaler` and `MinMaxScaler` for standardisation and normalisation respectively were selected. By default, the framework supports standardisation as a feature scaling method for scale-sensitive ML algorithms such as K-NN. Standardisation is also favoured by SVR which uses stochastic gradient descent because it increases the likelihood of the algorithm of converging faster [44].

5.5 Machine learning methods

An experimental feature called `Iterative Imputer` from Scikit-learn was selected for this workflow [33]. This is actually a transformer class designed for pre-processing data. Indeed, imputation is typically done to handle missing values prior training ML algorithms but for the purposes of this project it is the core of what is trying to be achieved. As such, the feature was directly used with slight modifications to its parameters for this use case.

5.5.1 Iterative Imputer

`Iterative Imputer` uses an iterative *round-robin* method to impute missing values. First, the imputer prefills each feature that has missing values with the median value for that feature, exactly as described in the prefilling stage of the framework. This generates a complete data set which is subsequently used to train an estimator (for example linear regressor) as a predictor. Figure 5.2 shows a simple example of model learning using three features A, B, C. The training happens in a *round-robin* fashion meaning that each variable is assigned as the dependent variable (test set) exactly once while the remaining variables are assigned as independent variables (train set) to build the regressor models. The learned models are used to predict and replace the prefilled values to perform the actual imputation. All these set of steps constitute a single run of the imputer which repeats until a stopping threshold condition is met or the iterator reaches maximum number of iterations. Both convergence conditions are hyperparameters defined by the user.

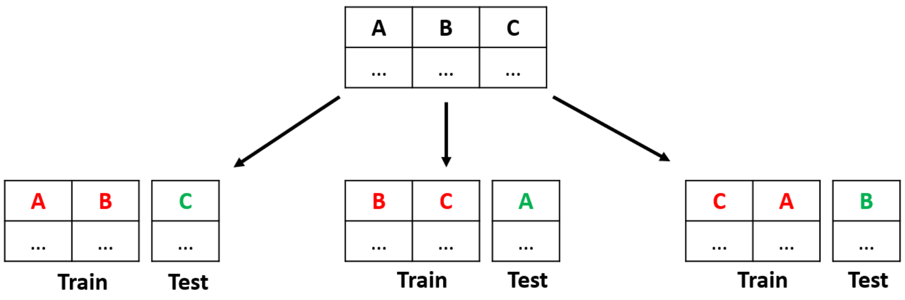


Figure 5.2: Diagram illustrating Iterative Imputer's *round-robin* procedure for a single run. There are n permutations for n features; here each feature becomes the test set exactly once while the others form the training set to fit the regressor. The regressor model is used to predict and replace the prefilled values with the new value to perform the actual imputation. This repeats until convergence.

5.5.2 Iterative Imputer Regressor

For purposes of the upcoming experiments, a wrapper class was implemented to use `Iterative Imputer` as a regressor instead of a transformer. There are distinct differences between the two - the former is used to build a regression model to then predict values. Like a transformer, `Scikit-learn` implements each regressor as a class inherited from `BaseEstimator` and `RegressorMixin`. Every regressor has (at least) the following methods:

- `fit`: fits the model on training data exactly like *transformers*.
- `predict`: uses the fit model to predict values for a target variable in test set. The `fit` method needs to be called before `predict`.

There is no `fit_predict` method as that is typically only used for unsupervised learning algorithms [93]. Following these requirements a wrapper class, called `IterativeImputerRegressor` (IIR), was implemented which inherits from `IterativeImputer` class which itself inherits from `BaseEstimator` and `TransformerMixin`. Figure 5.3 shows a UML diagram summarising the member variables and functions added or overriden in IIR child class. The member variables were overriden for use with laboratory data with each change explained as:

- *estimator*: linear regressor was set as default as Bayesian ridge was not selected for this project.
- *initial_strategy*: set to median instead of mean to fulfill prefilling strategy.
- *max_iter*: increased iterations to 100 to allow ensemble methods to reach good accuracy.
- *min_value*: to prevent predicting negative values as all laboratory test results should be positive.

The `fit` member function was overriden to allow the estimators to fit on the entire data set - that is, the independent and dependent variable. Regressors usually train on independent variables only and predict values for the dependent but in this instance the model training needs to happen on both to capture all relations to be able to impute missing values. A `predict` method was added which adds another column with all NaN values and passes it to the `transform` method of the parent class to emulate the imputation process. Only the imputed column is returned - exactly as intended and desired.

The exact same wrapper was also implemented for Scikit-learn's `SimpleImputer` which is a more popular imputation transformer. The same methodology was followed with the only member function overriden being *strategy* which was set to median. The `fit` method was overriden and `predict` method implemented in the same way as IIR to use the `SimpleImputer` as a regressor in the experiments.

The `pytest` framework was used to write and execute unit tests to test the functionality of IIR. The tests were written to compare the `fit` and `predict` methods independently as well as using them consecutively to simulate how a regressor would be used. The unit tests compare the results from IIR (using linear regressor as its estimator) to that of vanilla linear regressor from Scikit-learn.

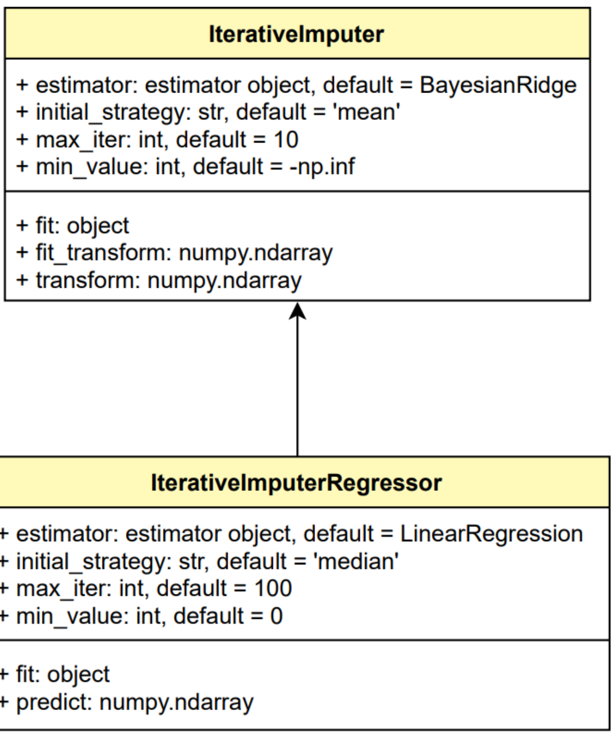

IterativeImputer
+ estimator: estimator object, default = BayesianRidge
+ initial_strategy: str, default = 'mean'
+ max_iter: int, default = 10
+ min_value: int, default = -np.inf
+ fit: object
+ fit_transform: numpy.ndarray
+ transform: numpy.ndarray
IterativeImputerRegressor
+ estimator: estimator object, default = LinearRegression
+ initial_strategy: str, default = 'median'
+ max_iter: int, default = 100
+ min_value: int, default = 0
+ fit: object
+ predict: numpy.ndarray

Figure 5.3: UML diagram showing `IterativeImputerRegressor` (IIR) inheriting from `IterativeImputer` transformer class. The child class overrides member variables estimator, initial_strategy, max_iter and min_value which are passed to the parent constructor using the `super()` method which returns a proxy object. The `fit` member function is overriden and a new `predict` method is added.

5.5.3 Regression models

The framework supports seven regression models described in Table 4.1. Scikit-learn provides vanilla implementations of six of those seven regressor models as shown by Table 5.1. XGBoost regressor was sourced from Python's `xgboost` library [83]. The hyperparameters for these models were tuned using GridSearchCV [33] which carries out an exhaustive search over the specified range of hyperparameters. The best combination of hyperparameters can be selected based on the evaluation metric that is most important for instance minimising RMSE. Separate Python scripts were written for each experiment to tune the hyperparameters.

Regression model	**Scikit-learn method**
Linear (LR)	`LinearRegression()`
Decision Trees (DT)	`DecisionTreeRegressor()`
Random Forests (RF)	`ExtraTreesRegressor()`
XGBoost (XGB)	`XGBRegressor()*`
Support Vector (SVR)	`SGDRegressor(SVR)`
K-Nearest Neighbours (K-NN)	`KNeighborsRegressor()`
Multi-Layer Perceptron (MLP)	`MLPRegressor()`
Simple Median	`SimpleImputerRegressor()`

Table 5.1: Table showing the Scikit-learn methods for each of the eight regression models. All methods were used directly from Scikit-learn's library except `XGBRegressor()` which was used directly from Python's `xgboost` library [38].

5.6 Bayesian Network methods

As part of the methodology outlined in the framework (Section 4.3), there are two main steps that are required to build: structure and parameter learning. This is then followed by inference which uses the probabilities and structure of BNs to predict values. In this project, BNs were implemented using the `pgmpy` library which already contains implementations of algorithms selected in the framework. Two classes were implemented, `BNImputer` and `BNRegressor`, which transform and regress respectively as shown in Figure 5.4.

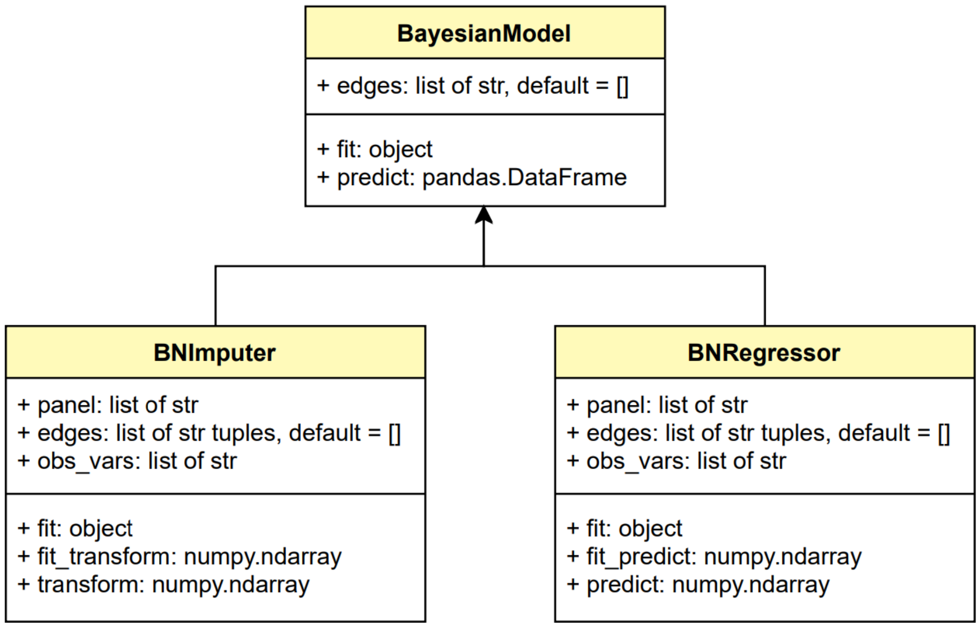

BayesianModel
+ edges: list of str, default = []
+ fit: object
+ predict: pandas.DataFrame
BNImputer
+ panel: list of str
+ edges: list of str tuples, default = []
+ obs_vars: list of str
+ fit: object
+ fit_transform: numpy.ndarray
+ transform: numpy.ndarray
BNRegressor
+ panel: list of str
+ edges: list of str tuples, default = []
+ obs_vars: list of str
+ fit: object
+ fit_predict: numpy.ndarray
+ predict: numpy.ndarray

Figure 5.4: UML diagram showing `BNImputer` and `BNRegressor` classes inheriting from `pgmpy`'s `BayesianModel` class. Both child classes initialise different member variables namely *panel*, *edges* and *obs_vars* of which *edges* is passed to the parent constructor using the `super()` method which returns a proxy object. Both child classes override the `fit` member function and implement new `transformer` and `predict` methods respectively.

`BNImputer` was implemented to directly impute missing values similar to `IterativeImputer` while `BNRegressor` was developed for the purpose of the experiments. Both classes inherit from pgmpy's BayesianModel class which provides the tools to build the structure of a desired BN. They also inherit BaseEstimator as well as TransformerMixin and RegressorMixin respectively for compatibility with Scikit-learn.

5.6.1 Model learning

BNImputer and BNRegressor were implemented as classes to facilitate manual, automatic or hybrid structure learning. For instance, the classes can be instantiated with the *edges* passed as a list of tuples where (A,B) denotes a directed edge between node A and node B. When the edges are specified (by the user), an important check takes place to ensure that there are no loops in the structure which violate the acyclic property of BNs. This functionality is inherited from the parent class constructor. Specifically for this project, the classes also require the laboratory panel to be specified as a list of strings to ensure that the model enforces a structure with all the nodes specified under the panel.

Model learning takes place using the fit method which is common for both classes and satisfies the requirements for Scikit-learn's API. The first step requires finding the topology of the network. If the edges were not provided during instantiation of the class then automatic learning takes place else the provided edges are assumed to be the complete structure of the BN which are then used for parameter learning. In automatic learning, the heuristic *Hill Climb Search* (HCS) algorithm in conjunction with *Bayesian Information Criterion* (BIC) scoring method, which are provided as part of pgmpy's libary, are used directly. The structure of the best model is then passed to the parent class constructor to initialise the BN. To ensure consistency with pgmpy's methods, any array-like structure is converted to DataFrame because it provides a neat way to assign names to each column (feature) which is used to build nodes and their connections to other nodes to create a BN. Since the panel (and associated analytes) is stored as a member variable during instantiation, it can be easily used to assign names to columns in the conversion of arrays to DataFrame. The fit method returns the self object so that the BN structure can be plotted using the networkx package and nodes/edges can be manually added or removed using methods inherited from BayesianModel. Plotting the network also helps to verify that structure learning took place.

The second step in model learning is parameter learning which generates the probability distributions for each of the nodes in the BN structure. In the implementations, parameter learning happens straight after structure learning with the nodes and edges passed directly to the _param_learn method. The framework uses the Bayesian method to learn the parameters of the model using the *Bayesian-Dirichlet equivalent uniform* (BDeu) scoring function. Fortunately, both methods were available as part of pgmpy's estimator class which was imported and used directly. It should be noted that probability distributions for each node can also be user defined in exactly the same way as defining the structure of the BN. The completion of parameter learning can be verified by iterating through the get_CPD() method which returns the probability table generated for each node. Parameter learning was also manually tested using the get_model_cdf() member function which returns a boolean if the conditional probabilities assigned to each node add to 1.

5.6.2 Model inference

The third step supported by the framework is inference where the BN uses the probabilities estimated from parameter learning and the topology structure to make predictions. The framework uses the *variable elimination* (VE) exact inference algorithm which was imported from pgmpy's `inference` class. The methods had to be adapted and wrapped appropriately to work with the transformer (`BNImputer`) and regressor (`BNRegressor`) classes respectively.

The imputation of values in `BNImputer` happens using the `transform` member function in three stages described by Figure 5.5. At the first stage, the function processes the input data (passed as a `DataFrame`) to extract the missing (*query*) and observed (*evidence*) variables and their corresponding values. Iterating by row means that each test record is treated independently; in other words, only the other observed values are used to make inferences of missing values for each record which is exactly as proposed in the framework. Both query and evidence can support one or more variables meaning that multiple variables can be imputed using the evidence from remaining variables each time. The second stage performs the imputation by passing the collected information to pgmpy's `inference` class to perform VE. The final stage imputes the missing variables by replacing the NaN values with inferred values. The three steps repeat for each row in the data set that contains missing values and hence generates a complete imputed data set that is returned.

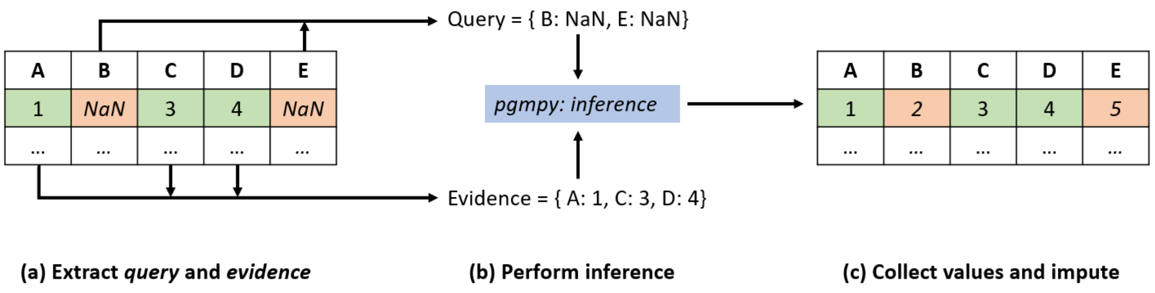

Query = { B: NaN, E: NaN}
A
B
C
D
E
1
NaN
3
4
NaN
...
pgmpy: inference
Evidence = { A: 1, C: 3, D: 4}
A
B
C
D
E
1
2
3
4
5
...
(a) Extract query and evidence
(b) Perform inference
(c) Collect values and impute

Figure 5.5: Diagram illustrating imputation procedure followed by `BNImputer`. There are three main stages to impute values: (a) the query and evidence variables are extracted from each row, (b) the collected parameters are passed to pgmpy's `inference` method and (c) the predicted values are collected and used to impute the NaN values returning a complete row.

One of the challenges of implementing the `transform` method was reducing the time complexity because the iteration of each row scales linearly with the size of the data set (with complexity of $\mathcal{O}(n)$). A neat solution was to use Python's `Parallel` and `joblib` multi-processing tools which provide multi-threading functionality. Since inference happens on each row separately multi-threading provides a way to run this method concurrently utilising as many processors supported by the target machine. This significantly reduces the time complexity and makes inference on large data sets such as laboratory data feasible.

Similarly, model inference in `BNRegressor` class happens using the `predict` method. In contrast to `transform`, the `predict` method is more straightforward because only a single value is inferred per row. The input data set is directly passed to the `predict` method provided by the parent class (`BayesianModel`) which returns a `Series` containing the estimated values. In line with Scikit-learn's conventions this is returned as a 1D Numpy array. One of the requirements of this method is that the input data set must be complete so that the algorithm can utilise all the other features to predict the value of the dependent variable. In the context of regression, this means that the test data set must have complete values to predict the values of the dependent variable.

5.7 Summary

To summarise, the following implementations were made in this project with respect to the five stages of the framework:

- *Feature selection*: application of `TidyWidget` class and development of outlier removal utility function.
- *Prefilling*: development of `EMImputer` transformer class for prefilling missing values in BN workflow.
- *Pre-processing*: implementation of a utility function to remove outlier values from the data set.
- *Machine learning models*: development of `IterativeImputerRegressor`, unit testing using `pytest` framework and configuration of Scikit-learn's regressor models using GridSearchCV.
- *Bayesian Network models*: development of two classes `BNImputer` and `BNRegressor` which provide functionality for structure learning using HCS and BIC scoring method, parameter learning using Bayesian estimator and inference/prediction using VE exact algorithm.

It should be noted that the last two points cover stages four and five of the framework together. All implementations were tested for compatibility with Scikit-learn's API.

Chapter 6

Testing and Experiments

In Chapter 5, implementation details were provided on how each stage of the framework (Chapter 4) was implemented. This chapter outlines a series of experiments on a real-life laboratory data set to test the implemented imputation methods. The experiment steps were largely based on the proposed framework methodology (Section 4.1) but also on literature reviews and prior practical experience with the Scikit-learn library. This section presents an overview of the pathology data set used for this project (Section 6.1) followed by an overview of four experiments conducted in this project (Section 6.2). The chapter then provides the aim, materials and methods used in experiments I-IV (Sections 6.3 - 6.6). Finally, a summary of the main experiment is presented (Section 6.7).

6.1 Pathology data set

The pathology data for this project was provided by Imperial College NHS Healthcare Trust which comprises of date of collection, anonymised patient ID, test code, result, unit, normal reference range and status. For computational reasons only a subset of annual data containing patient records from March to May 2020 was selected. This data set had 5, 461, 798 records for 77, 350 patients where each record (row) contains one test result (analyte value) for a single patient. The high number of patients (and test records) can likely be explained by the first wave of COVID-19 in the UK which began in March 2020 [94] during which the frequency of testing would have increased. There was no explicit information provided in the data set to confirm this nor whether, if any, of the patients were positive for COVID-19. As such, on the balance of probability, the data set was assumed to be for patients with COVID-19 and not generic. Before using this data set for the experiments a decision was made to omit 1, 054, 327 test records as they did not have a valid corresponding patient ID. Therefore, modified raw data set had 4, 407, 471 test records for 45, 672 patients in total.

6.2 Overview of Experiments I-IV

Figure 6.1 provides a graphical overview of the experiments described in this section. Experiment I transforms the data set to create *static* patient daily profiles after which feature selection and exploratory data analysis is carried out. The daily profiles contain only complete test results with no missing values and are used for the subsequent experiments. Experiment II uses the implemented `IterativeImputerRegressor` (Section 5.5) with the standard eight regression models (Table 5.1) to investigate ML based imputation. Experiment III investigates the same using the implemented `BNRegressor` (Section 5.6) to investigate BNs. It is worth mentioning that simple median imputation is used as a reference or baseline in experiments II and III. Experiment IV compares both ML based and BNs to facilitate direct comparison. Experiments II-IV carry out two sub experiments each which include single and multiple feature removal to simulate different patterns of missingness in laboratory data. Features are removed under the MAR assumption which facilitates the use of ML based and BN imputation methods to exploit feature relations to impute the missing values.

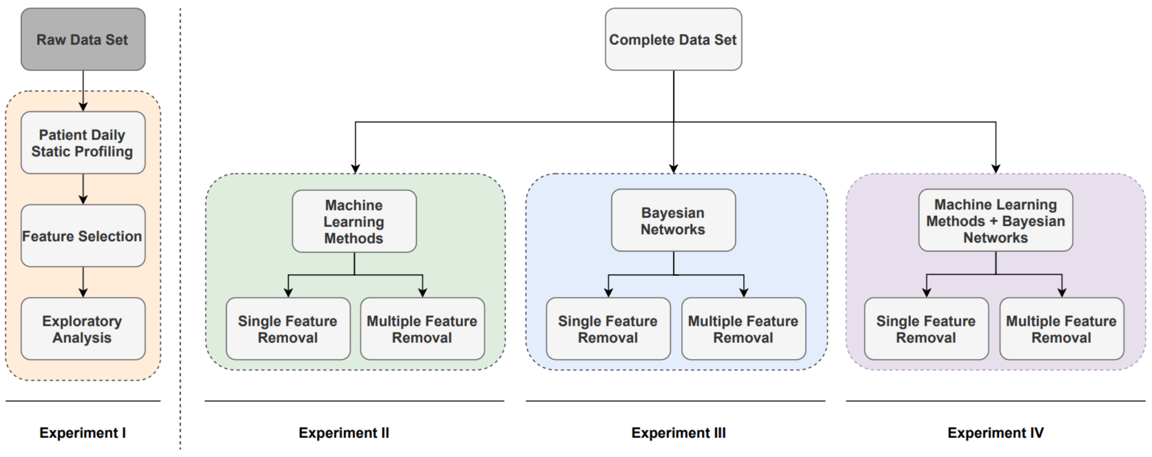


Figure 6.1: Diagram providing a high level overview of the experiments. The experiments shown in the diagram are: (i) Overview of pathology data (ii) Imputation using ML based methods, (iii) Imputation using BNs and (iv) Comparison between ML based and BNs. Experiment I prepares (and analyses) the raw pathology data for use in experiments II-IV.

All experiments were carried out using *static* daily patient profiles meaning that the temporal evolution of values for a given patient (who has more than one test panel result) were not considered. This assumption facilitates the use of the imputation methods from the day that the patients are admitted (which is usually when they are most severely ill) so that the predictions are made only using the other observed values for a single time instance on that day. For instance, it is extremely likely that clinicians will request FBC on the day that the patient is admitted. Therefore, it is imperative that the methods treat each panel request independently and impute accordingly.

Experiments II-IV were conducted mainly using Scikit-learn's `Pipeline` class which provides a neat way of combining and applying transformations and estimators to the data set [33]. It encapsulates many of the steps described in the experiments but requires all transformers and estimators to be compatible with Scikit-learn's API. A summary of implementations is provided in Section 5.7 which explains that all implemented methods are compliant with Scikit-learn.

6.2.1 Single feature removal

The first sub-experiment removes all values for a single analyte from the data set and uses the other analytes to predict its values. This simulates a multiple regression problem where only one feature is missing but this is rarely the case in practice. However, it is important to carry out this experiment to validate the imputation methods where feature relations are explicitly used to impute missing values. For the same reason, it was decided not to investigate different proportions of missing values with single feature removal as it would only reduce the amount of data that is imputed which would ultimately impact the fidelity of analysis.

6.2.2 Multiple feature removal

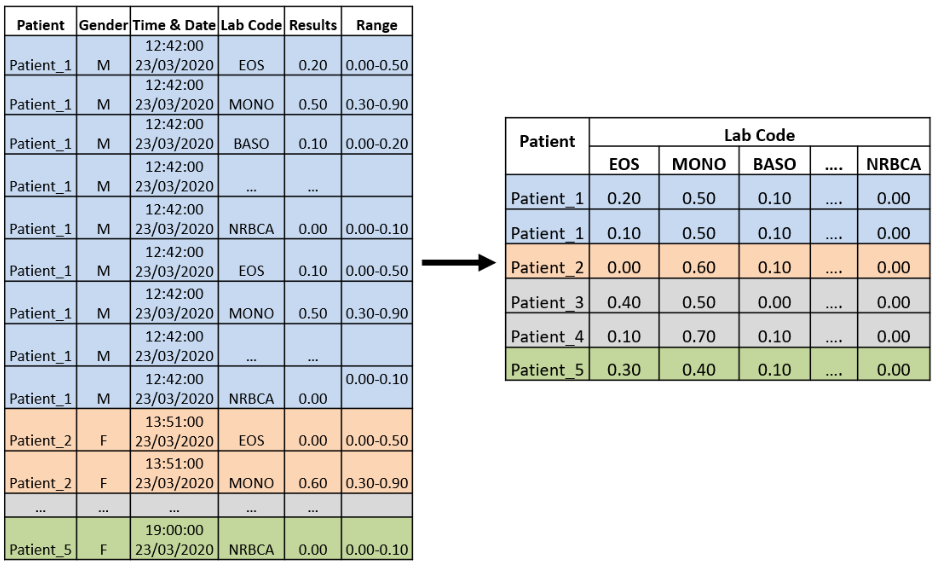
The second sub-experiment removes multiple (multi) features to simulate missingness in laboratory data closer to a real-life scenario where multiple test values are missing (at random). Multi feature removal is done systematically where different proportions (but same for each analyte) are removed from the data set to satisfy the MAR assumption. It was decided to only investigate three proportions of missing values: 10%, 30% and 50% as that would provide sufficient information to test the robustness of the algorithms with increasing uncertainty. The same approach was followed by a similar empirical study conducted by Waljee *at al.* (2013) which tested 10%, 20% and 30% missing values. It should be noted that errors are calculated exclusively on the imputed data to ensure homogeneity between different levels of missingness.

6.3 Experiment I: Overview of pathology data

Aim: The aim of the experiment is to prepare the data for *static* profiling, perform *feature selection* and carry out some of the pre-processing steps for the subsequent experiments.

6.3.1 Patient daily static profiling

The raw data set had 18 variables of which only four were of interest: patient ID, date of result, test code and test result. The `TidyWidget` class is used to transform the data set so that the test results are aggregated by row. This transformation is required so that the biochemical markers and their corresponding values are recognised as *feature* inputs to the imputation methods. Figure 6.2 provides an illustration of this transformation and shows the layout of daily patient profiles. For purposes of static profiling, the time and date of test results were removed.



Patient	Gender	Time & Date	Lab Code	Results	Range
Patient_1	M	12:42:00 23/03/2020	EOS	0.20	0.00-0.50
Patient_1	M	12:42:00 23/03/2020	MONO	0.50	0.30-0.90
Patient_1	M	12:42:00 23/03/2020	BASO	0.10	0.00-0.20
Patient_1	M	12:42:00 23/03/2020	...	...	
Patient_1	M	12:42:00 23/03/2020	NRBCA	0.00	0.00-0.10
Patient_1	M	12:42:00 23/03/2020	EOS	0.10	0.00-0.50
Patient_1	M	12:42:00 23/03/2020	MONO	0.50	0.30-0.90
Patient_1	M	12:42:00 23/03/2020	...	...	
Patient_1	M	12:42:00 23/03/2020	NRBCA	0.00	0.00-0.10
Patient_2	F	13:51:00 23/03/2020	EOS	0.00	0.00-0.50
Patient_2	F	13:51:00 23/03/2020	MONO	0.60	0.30-0.90
...	...	...	...	...	
Patient_5	F	19:00:00 23/03/2020	NRBCA	0.00	0.00-0.10

Patient	Lab Code				
	EOS	MONO	BASO	....	NRBCA
Patient_1	0.20	0.50	0.10	....	0.00
Patient_1	0.10	0.50	0.10	....	0.00
Patient_2	0.00	0.60	0.10	....	0.00
Patient_3	0.40	0.50	0.00	....	0.00
Patient_4	0.10	0.70	0.10	....	0.00
Patient_5	0.30	0.40	0.10	....	0.00

Figure 6.2: Diagram showing the transformation from raw data (left) to static daily profile (right). For each patient, their FBC test results are aggregated for their respective row. This generates a daily profile for all patients in the data set grouped by time and day of test result (starting from admission and first panel request).

6.3.2 Feature selection

The raw data set contained many test results for different test panels. It should be noted that the same set of panels were not requested (and therefore not recorded) for all patients. For sake of specificity, this project only focused on FBC which is the most frequently requested test panel [95]. As such, the data set was further reduced to 94, 984 test records. Table B.2 provides a description of the 16 biochemical markers under this panel. Fortunately, every patient in the data set had at least one set of FBC test results and there were no missing values for any of those tests. While this is rare in practice (and is the problem being addressed in this project), it is in fact favourable for the fidelity of the experiments as the statistical power of the data is retained.

6.3.3 Distributions of FBC biochemical markers

The density distribution for each analyte is presented in Figure 6.3 using violin plots. Each violin plot is superimposed with a box-plot which provides additional robust measures such as central tendency (median) and statistical dispersion (IQR). It is evident that all features contain a large number of outliers (defined using Z-Score bounds in Section 4.2) which seemingly makes their distributions highly skewed. While it is not strictly related to the project, it is worth discussing these distributions from a clinical perspective as most of the patients in the data set likely had COVID-19.

For brevity, all analytes can be broadly classified into three categories shown in Table 6.1. It is interesting to note that all analytes under the white blood cell category are extremely positively skewed with large number of outlier values towards the high unit range. This is expected as the quantity of these type of cells increases as part of an immune response to pathogenic infection. On the other hand, red blood cells have outlier values on both sides of the tail indicating that patients equally suffered from anemia and B-12 deficiency; HCT and HGB show more proportion of values in the extreme low range showing that anemia was more prominent. A recent study found this disease to be associated with severe COVID-19 [96]. The analytes under platelets also have extreme high values which are also in line with recent studies showing increased platelet production in COVID-19 patients [97]. Therefore, the distributions support the earlier COVID-19 assumptions made about the data set.

Cell type	Analytes
Red blood cell	HCT, HGB, MCH, MCHC, MCV, NRBCA, RBC, RDW
White blood cell	BASO, EOS, LY, MONO, NEUT, WBC
Platelet	MPV, PLT

Table 6.1: Categorisation of FBC biochemical markers. The analytes have been grouped to show their wider function and the cell type that they are associated with.

However, for the purposes of analysis, the outliers for each analyte were removed using the Z-Score method (with coefficient value of 1.5). For clarity, if any test result for a given patient had at least one outlier value then the entire record was discarded. The new data set was reduced by 40.76% and contained 56, 271 test records.

Figure 6.4 presents the same set of violin plots with the outliers discarded. It is immediately evident that BASO and NRBCA are constant analytes that show no data dispersion and therefore not suitable to predict other values. As such, these features were removed from the data set. EOS and MONO show notable discontinuities in their distribution as take discrete values over a continuous range. Regardless, these features were not discarded as they may still add predictive power because the models can benefit from combining the information from these analytes with others for enhanced performance. A trade off exists between the inclusion of these analytes and risk of adding noise (which increases likelihood of over fitting) but the benefits outweigh the limitations. Apart from the distributions attributing to Gaussian-like shapes, there are no discernable patterns amongst the other analytes.

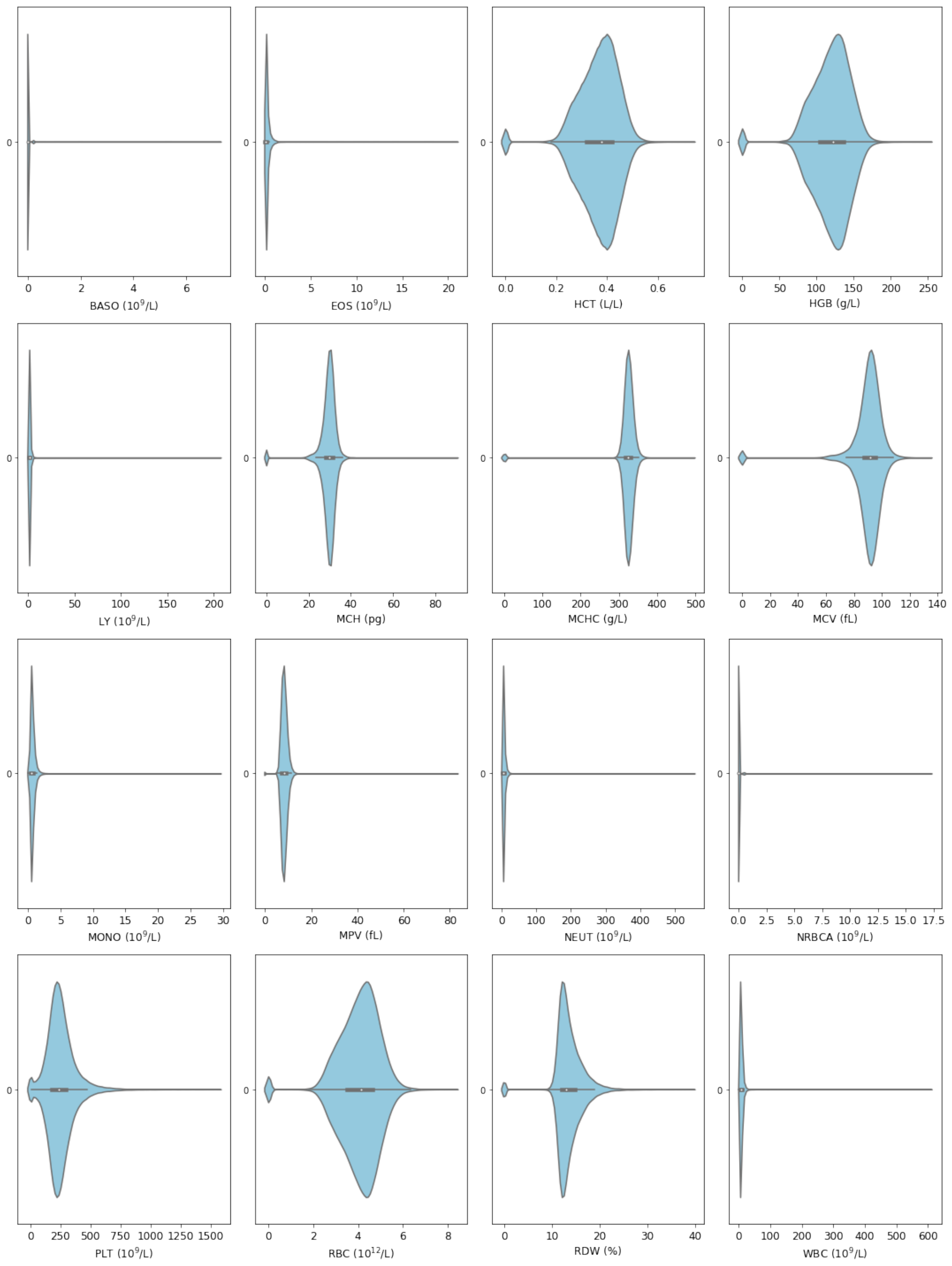


Figure 6.3: Diagram showing the violin plot for each biochemical marker in FBC panel with presence of outliers (raw data set). Each plot contains a box-plot which is superimposed with the kernel density estimation (KDE) plot to show the distribution of values for that analyte.

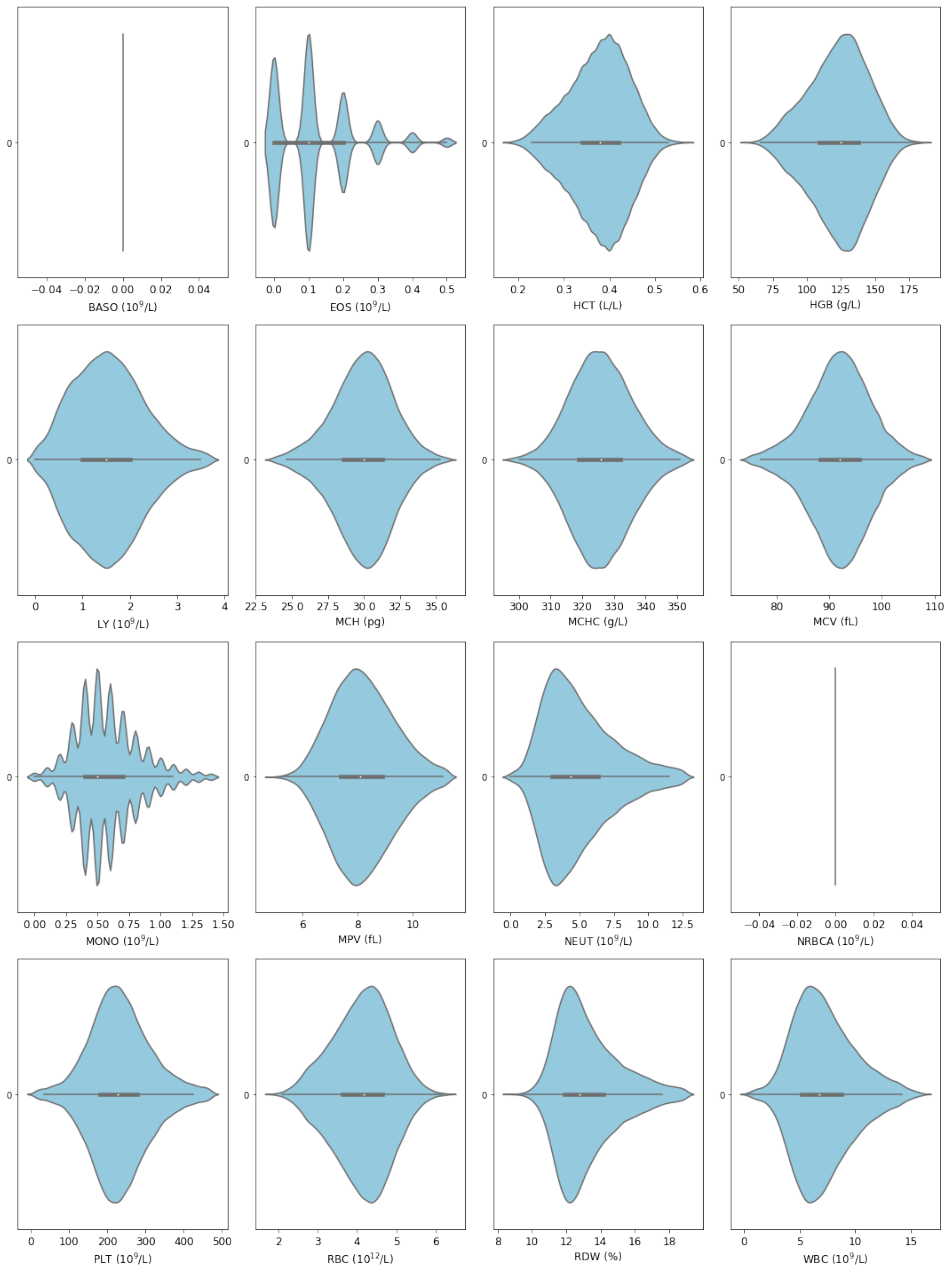


Figure 6.4: Diagram showing the violin plot for each biochemical marker in FBC panel without outliers (modified raw data set). The outliers are discarded using the Z-Score method to each feature and the same information as Figure 6.3 is presented.

Test for Gaussianity

Given the Gaussian-like distributions of most analytes, a *Jarque-Bera* (JB) test was conducted which is suitable for more than 2000 samples as is the case here [98]. It measures the skew (symmetrical properties of the distribution about its mean) and kurtosis (tail size at either ends of the distribution) against a standard Gaussian distribution which has zero skew and three kurtosis. Table B.1 shows that with the significance level set at $p < 0.05$, all analytes have a p-value of zero. This means there is sufficient evidence to state that the skewness and kurtosis of the analytes is significantly different from a Gaussian distribution. This result contradicts the observations made about the distribution of the analytes (after removing the outliers). One explanation may be the large sample size of the data causing the sampling distribution of the mean to approach Gaussian distribution as stated by the Central Limit Theorem [99]. This does assume that the samples are independent and identically distributed (*iid*) which is not unreasonable given that static profiling treats each record independent from others. Fortunately, Gaussianity is not an underlying assumption for any of the approaches being investigated so it does not impact the experiments.

6.3.4 Correlations of FBC biochemical markers

Feature correlations were obtained using Pearson's correlation coefficient (r) which finds linear correlations between pairs of variables (x, y) and can be expressed using Equation 6.1.

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2(y_i - \bar{y})^2}} \tag{6.1}$$

Figure 6.5 shows the lower half of the correlation matrix for the 16 features selected for the subsequent experiments. Only the cross-feature correlations are shown labelled with their value of r and colour-coded according the corresponding colour map where a darker shade of red indicates high correlation value. The self-correlations (which trivially have a score of one) is omitted as is the upper half of the correlation matrix because it is symmetric to the lower half. In general, most of the feature correlations lie in the range $[-0.5, 0.5]$ with some notable strong positive covariates including (HCT, HGB), (HGT, RBC), (HGB, RBC), (MCH, MCV) and (NEUT, WBC).

From a medical viewpoint the strong correlation pairs make sense. Both HCT and HGB are strongly correlated to RBC because they measure attributes related to the same cell; the former is a measure of the protein contained in red blood cells while the latter measures the percentage of total red blood cells in the whole blood. In fact, HCT can either be determined using laboratory diagnostic machines or calculated directly using by taking the product of RBC and MPV [100]. Likewise, the high correlation between MCH and MCV is plausible as an increase in size of red blood cells (measured by MCH) is likely correlated with an increase in the amount of HGB per red blood cell (measured by MCV). This idea will be explored even further with BNs which can model this exact *cause-effect* relationship. For the same reasons as above, the positive correlation between NEUT and WBC is seen as they are both attributes of the same type of cell (white blood cells). It was interesting, however, to not see a strong positive correlation between other analytes in white blood cell given that they all help to fight infection. Perhaps this highlights how some white blood cells are more important than others to fight this type of virus.

Since the absolute feature correlations values lie in the range $[0, 0.5]$, network graphs were plotted using the `networkx` package to visualise the correlations shown in Figure 6.6. Each node in the network graph corresponds to an analyte while the colour of the edges are weighted by the strength of correlation between the two nodes - darker shades indicate stronger correlation. Likewise, the size of the nodes are dynamic to their degree (number of nodes it is connected with). Four sets of correlation graphs (graphs 1 - 4) are shown corresponding to absolute correlation value range $[0.1, 0.2, 0.3, 0.4]$. These graphs are more useful for lower threshold values as it is more difficult to discern the relations between analytes for the same values from the correlation matrix. It should be noted that the topology of the correlation graphs is invariant to the scale or ordering of the data (assuming corresponding (x_i, y_i) values stay together). As such, calculating the correlation matrix before or after scaling should not impact the correlation coefficients.

It is interesting to note that for thresholds $[0.1, 0.2]$, LY has the largest degree size meaning that it has the highest number of relations to other analytes. This is expected for analytes that are classed under the white blood cell type but unexpected for all others. For example, graph 2 shows correlations of LY with each of RBC, HCT and HGB even though they are analytes from different cell types. A plausible explanation for this correlation may be the effects of COVID-19 lowering RBC and LY simultaneously especially because low LY count has shown to be correlated with a high risk of lung disease [101]. Graph 2 also shows a small sub-graph of MCH, MCHC and MCV because they are all derived values which can be used interchangeably [102].

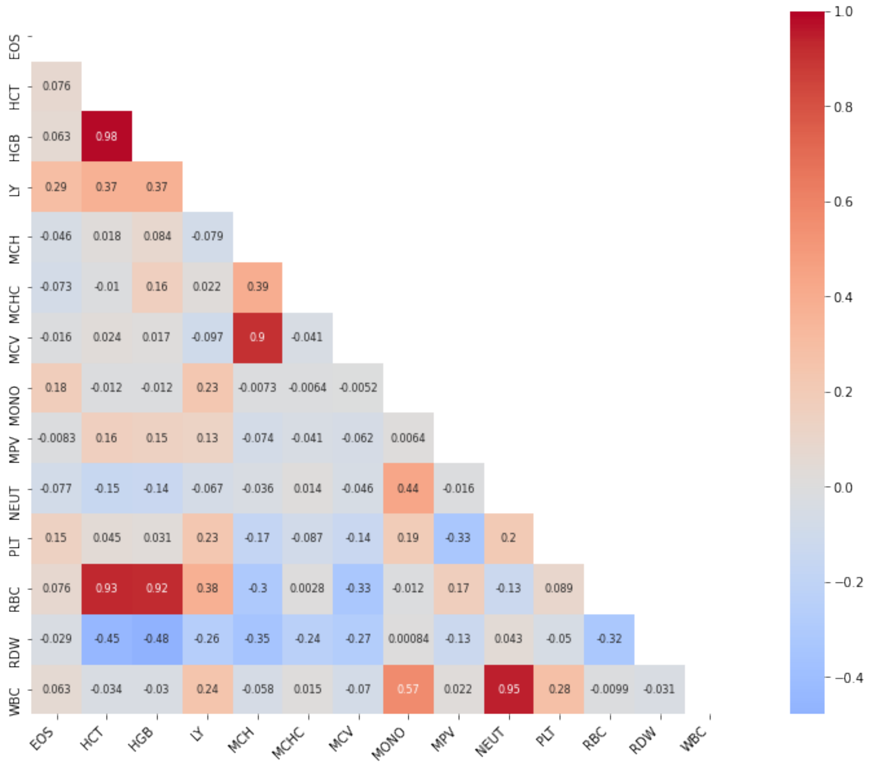


Figure 6.5: Diagram showing correlation matrix for FBC biochemical markers. The colour bar shows the strength of the correlation values with stronger positive values having a darker red shade while negative values having a blue shade.

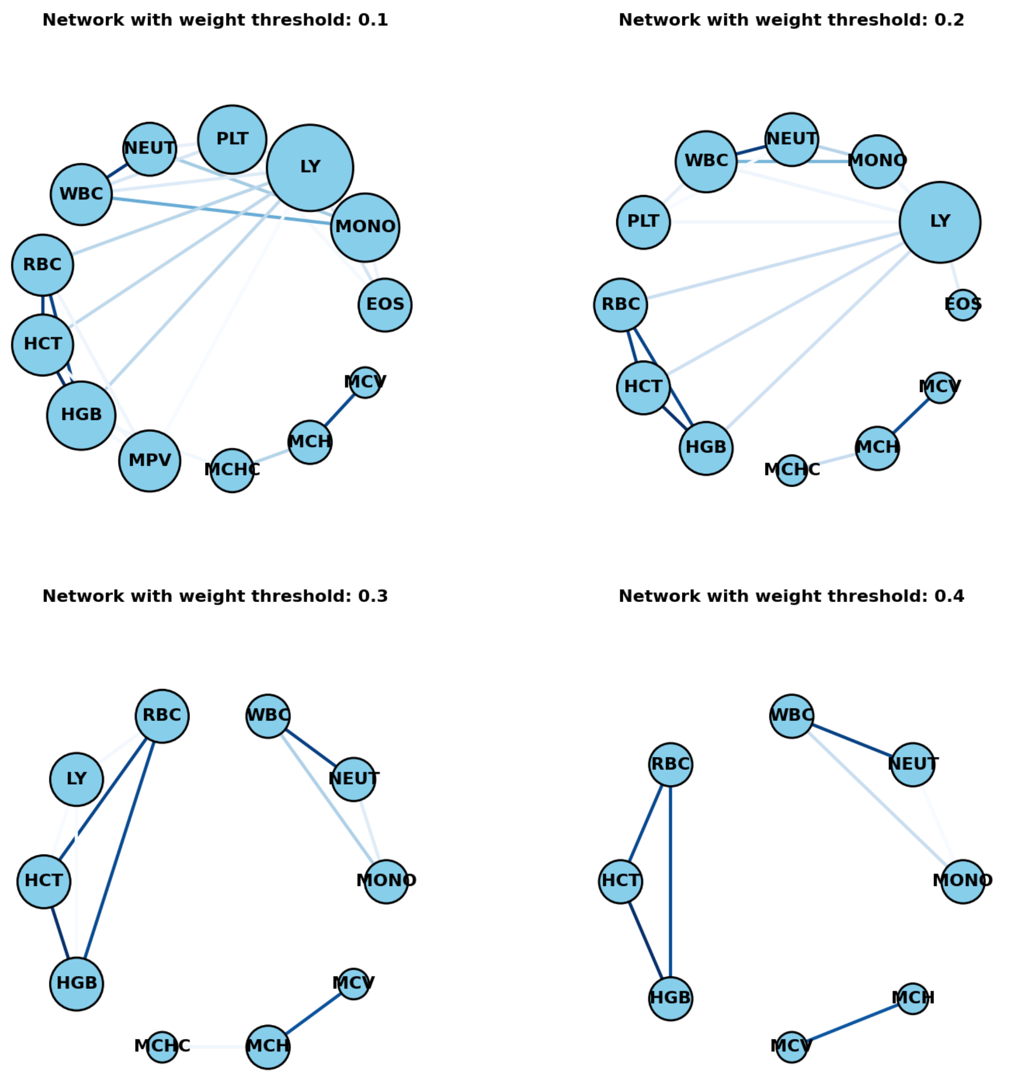


Figure 6.6: Diagram showing correlation network graphs for FBC biochemical markers. There are four graphs (1 - 4) each with a different weight threshold corresponding to the range of absolute correlation values between the features $[0.1, 0.2, 0.3, 0.4]$.

6.4 Experiment II: Imputation using Machine learning methods

Aim: The aim of this experiment is to investigate ML based imputation methods using the implemented `IterativeImputerRegressor` two sub experiments (a) single and (b) multiple feature removal. The complete data set generated from experiment I was used for this experiment containing 56, 271 test records.

Materials and methods: Figure 6.7 presents a graphical overview of the experiment methodology which was devised using the imputation framework. In the first step, the complete data set is split 80:20 into cross validation training set (CVTS) and held out test set (HOTS); the split ratio was primarily determined based on the size of the data set and recommendations from the literature [31]. In total, 45, 016 test records were used for CVTS with 11, 255 for HOTS. It is vital to split the data in the first step to avoid any *data leakage* such as during standardisation (pre-processing) where statistic parameters (such as mean μ and standard deviation σ) are used to transform the data. If the entire data set were to be pre-processed altogether then the model would learn additional information about the test set which should remain completely unseen to validate the evaluation of model performance. Out of the two feature scaling approaches standardisation was chosen as the analytes did not statistically show Gaussian distribution. The next step requires the same proportion of single or multiple features to be removed from both data sets upholding the MAR assumption.

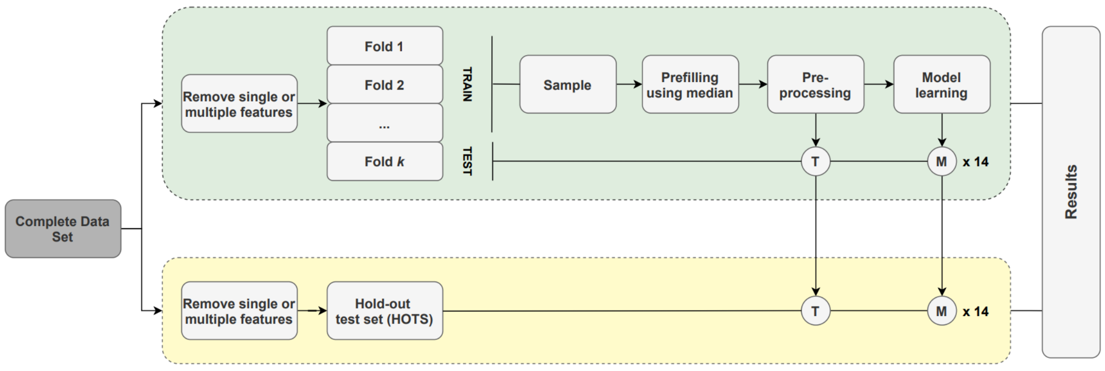


Figure 6.7: Diagram providing a high level overview of experiment II methodology. The complete data set is first split into cross validation training set (CVTS) and held out test set (HOTS). Then, single or multiple features are removed from both data sets. Five-fold CV is performed on the CVTS where each training fold is sampled, prefilled using the feature median, standardised (pre-processed) and trained. The standardisation equation (T) is used to transform HOTS and the best ML model (M, determined from CVTS) is evaluated on the transformed HOTS for each analyte. Finally, the complete data set is used to obtain error performance for the imputed values.

In this project, five-fold CV is used to assess the generalisability of eight regression models, described in Table 4.1, on an independent data set and prevent over fitting. This method further splits the training set into two sets: four training folds and a single test (validation) fold. The training folds are sampled, prefilled using feature medians, standardised and used to learn the model parameters. These steps are carried out for each analyte during which the RMSE scores for each of the eight models are aggregated and used to determine the best ML model for that analyte (M). Effectively, a single ML model (out of the eight tested) is selected for each analyte (14 in total) and used to impute its missing values. This means that the best model which best characterises the relations of an analyte with others is chosen.

It should be noted that the standardisation equation is obtained during pre-processing (T) and used to separately transform HOTS prior to the final step where the performance of the best model is evaluated. The complete data set, which contains the ground-truth values, is used to calculate imputation error (using appropriate evaluation metrics) of the best model for each analyte.

Hyperparameter tuning: Hyperparameter tuning was conducted to find the most optimal set of parameters for each of the seven ML models using GridSearchCV. Five-fold CV was used with the results aggregated to select the hyperparameters that give the lowest RMSE scores. Table 6.2 presents the hyperparameters selected for both single and multi feature removal. Note that one set of hyperparameters were obtained for multi-feature removal based on the results from 30% feature removal.

Model	Hyperparameter	Single	Multiple (10%, 30%, 50%)
DT	Max depth	8	6
	Max leaf nodes	15	12
	Min samples split	8	8
	Min samples leaf	8	8
RF	No of estimators	100	10
	Max depth	10	6
	Min samples split	10	10
SVR	Epsilon	0.05	0.01
	Learning rate	Adaptive	Adaptive
	Loss	Squared	Squared insensitive
K-NN	No of neighbours	8	7
	Weights	Distance	Distance
XGB	No of estimators	100	10
	Max depth	10	6
MLP	Hidden layers	32	(32, 64)
	Solver	Adam	Adam
	Learning rate	0.0001	0.0001

Table 6.2: Table showing the hyperparameters for single and multi feature removal sub-experiments for ML based imputation methods. Linear regression (LR) does not have tunable parameters so it was not included in this table. The most important parameters for each of the other ML models have been included. **Keys:** *DT = decision tree, RF = random forest, SVR = support vector, K-NN = k-nearest neighbours, XGB = XGBoost and MLP = multi-layer perceptron.*

In general, the magnitude of parameters was smaller for multi-feature removal as the ML methods build smaller trees/networks to cope with the loss of data. Reducing the number of estimators for DT/RF/XGB had the largest impact in terms of reducing time complexity but minimal impact on the performance showing that ensemble methods can create complex models if their parameters are not tuned. K-NN naturally reduced from eight to seven neighbours between the two experiments to cope with reduced features. A denser architecture for MLP was preferred in multi feature removal to learn more complex relations that retain the performance even with fewer variables available. Crucially, the denser architecture did not over fit.

6.5 Experiment III: Imputation using Bayesian Networks

Aim: The aim of this experiment is to investigate suitability of BNs for imputing missing values via the same two sub experiments (a) single and (b) multiple feature removal. This experiment uses the implemented `BNRegressor`. As before, the complete data set generated from experiment I was used for this experiment containing 56, 271 test records.

Materials and methods: Figure 6.8 presents a graphical overview of the experiment methodology. In general, the experiment follows the same steps as with ML based methods described in Section 6.4 with three notable differences. First, the CVTS is used to minimise over fitting in structure and parameter learning described in Section 4.5. As such, only a single BN structure is learned which is used to make the predictions as opposed to creating a new model for each analyte. Nevertheless, the experiment evaluates and reports performance of the model for each analyte separately. The second notable difference is pre-processing for which Scikit-learn's `KBinsDiscretiser` [33] transformer is used to discretise the continuous values in uniform bins. This is predominantly done for performance reasons as the implemented BNs treat each value as a discrete state to build the probability tables. As such, this is a preventative measure to avoid using excessive memory at run time (can use in excess of 45 GB RAM). The third notable difference is using the EM algorithm to prefill missing values using the implemented `EMImputer` transformer class which is better suited for structure learning in BN for reasons discussed in Section 4.3.

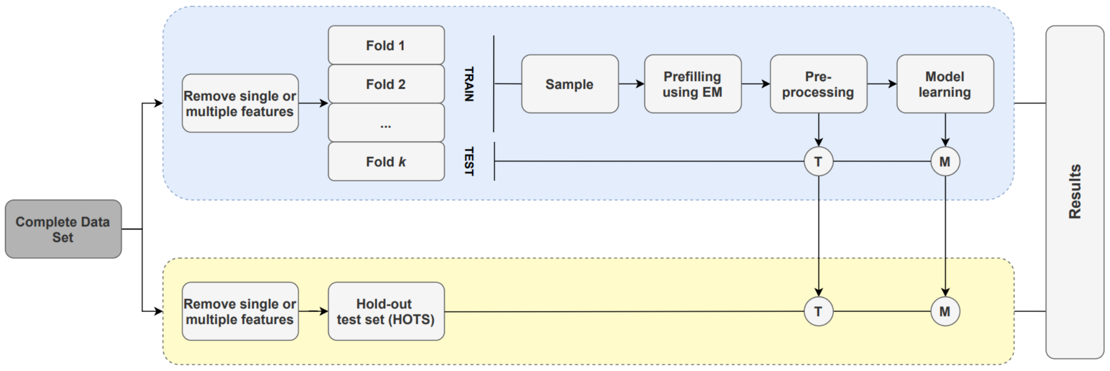


Figure 6.8: Diagram providing a high level overview of experiment III methodology. The complete data set is first split into CVTS. Then, single or multiple features are removed from both data sets. Five-fold CV is performed on the CVTS where each training fold is sampled, prefilled using the EM algorithm, discretised (pre-processed) and learned (via structure and parameter learning). The discretisation method (T) is used to transform HOTS and the best BN structure (M, determined from CVTS) is evaluated on the transformed HOTS for each analyte. Finally, the complete data set is used to obtain error performance for the imputed values.

It should be noted that discretising the data does not affect the regression task being solved even though the BNs internally "classify" the most likely value for the variable being predicted. This methodology is based on empirical findings from a study conducted by Torgo *et al.* (2005) [103] which found that "regression by classification" yields comparable results if the number of bins is selected using n-fold CV. As such, GridSearchCV with five-fold CV was used to find that five number of bins is sufficient for this task as it minimises the RMSE. It is worthwhile mentioning that discretisation happens uniformly such that each class has equal number of samples to avoid *class imbalance*.

6.6 Experiment IV: Comparison of Machine learning and Bayesian Networks

Aim: This experiment aims to compare both ML and BN methods which have been investigated in experiments II (Section 6.4) and III (Section 6.5) to determine which method gives a better performance. This experiment uses the `IterativeImputerRegressor` and `BNRegressor` classes.

Materials and methods: The methodology for this experiment follows that of experiment III (Section 6.5) which outlined the set of steps to investigate BNs. One of the notable differences between this experiment and experiment II (Section 6.5) is that in this experiment the continuous values in the data set are discretised into five uniform bins for each analyte. This facilitates comparison between ML based methods and BNs as it ensures that the data is pre-processed in the same way for both. It should be noted that, as before, the problem is still formulated as a regression task where both methods try to predict the actual value of the variable being regressed.

In experiment II (Section 6.4), the methodology evaluates only the best ML method on HOTS for each analyte with both single and multi feature removal. In the same way, this experiment compares the best ML methods found in experiment II with the best BN structure found in experiment III (Section 6.5) to ensure a valid comparison is made. For the same reason, the same configuration of hyperparameters (Section 6.4) are used here. Despite the similarity between the two methods, it should be noted that the framework methodology (Section 4.1) uses two separate workflows for ML based and BN imputation because of their respective prefilling strategies and model learning mechanisms. As such, this experiment is conducted using two separate Python scripts following the same steps as experiment III.

6.7 Summary

To summarise, this chapter outlined four experiments which were carried out on a real-life laboratory data set provided by Imperial College NHS Healthcare Trust. The methodology steps for experiments II - IV (Sections 6.4 - 6.6) were informed from an overview of the provided data set explored in Section 6.3. The following provides a summary of the key experiments:

- *Experiment I:* Overview of the pathology data set to understand feature distributions and (linear) correlations.
- *Experiment II:* Investigation of ML based imputation methods to compare multiple regression models perform against simple median imputation.
- *Experiment III:* Investigation of BN imputation methods to compare performance of the best BN structure (learned from data) against simple median imputation.
- *Experiment IV:* Comparison between ML based and BN imputation methods to determine which method gives a better performance.

Chapter 7

Results and Discussion

In Chapter 6, the aim, methodology and materials were described for four experiments to evaluate the two imputation methods implemented in this project (Chapter 5). This chapter presents the results of experiments II - IV; the results of experiment I were presented with its methodology to explain the rationale behind decisions in experiments II-IV. Firstly, the chapter provides a description of the prerequisites (Section 7.1) such as the evaluation metrics (Section 7.1.1) and statistical tests (Section 7.1.2) selected for this project. Then, the results of experiments II-IV with single and multiple feature removal are presented and discussed (Sections 7.2 - 7.4). Finally, a summary of the key results is provided (Section 7.5).

7.1 Prerequisites

This section presents the prerequisites that need to be discussed before presenting the results. This includes the evaluation metrics selected for analysing the results (Section 7.1.1) and statistical tests to assess the distribution of imputed values (Section 7.1.2).

7.1.1 Metrics

In Section 2.3.10, an overview of the most common evaluation metrics used in regression tasks was provided. From the studied metrics, RMSE was selected because it is measured in the same unit magnitude of dependent variable which makes it more interpretable than the other metrics. It provides an intuitive way for prospective clinicians to understand how the magnitude of the error relates to each of the analytes which is an important consideration in this project. This metric is sensitive to outlier values but they were removed from the data set as part of experiment I.

To facilitate comparison of the implemented methods in relation to the simple median approach, a custom Δ (Delta) metric was designed which measures the *percentage improvement* in RMSE scores. This metric characterises the relative increase (or decrease) of the implemented methods (ML or BNs) over baseline median imputation. It can be expressed as:

$$\Delta = 100 - \left(100 \times \frac{RMSE_{ML/BN}}{RMSE_{median}}\right) \tag{7.1}$$

The metric ranges from 0 ± 100 % where positive percentages indicate improvement, zero indicates no improvement and negative indicates a worsened score. It is assumed that the implemented methods will outperform simple median imputation so the sign of the percentage value should be assumed as positive (unless otherwise stated). The metric is dictated by the magnitude of the RMSE score for ML and BN methods for which a small score would yield a large percentage improvement.

RMSE is a suitable metric to evaluate intra-analyte performance of the ML models for example when selecting the best model based on the aggregated scores on CVTS. However, it is difficult to determine what classifies as a "good" RMSE score across analytes with different scales. Initially, normalising the RMSE was considered using the spread of the true data to create a scale-free metric but it was later replaced by using a normalised version of MAE called normalised absolute error (NAE). This metric helps to better analyse and compare the distribution of absolute errors. This facilitates inter-analyte comparison of the uncertainty of prediction for each analyte which provides insight into how the models perform on analytes that are inherently difficult to predict. It can be defined as:

$$NAE = \frac{|y - y_i|}{max(y_i) - min(y_i)} \tag{7.2}$$

where y and y_i are the predicted and true values respectively.

7.1.2 Statistical analysis

The implemented imputation methods are compared to the baseline median approach in experiments II (Section 6.4) and III (Section 6.5). The selected evaluation metrics provide one way to characterise the imputation accuracy but do not statistically assess how the imputed values relate to the true values. It is important to assess if the ML based and BN methods actually preserve the distribution of the true data; in other words, the imputation methods should not introduce bias that can change the underlying distribution of data. For this study, two non-parametric tests were considered to measure the differences in the distributions of the analytes: Mann-Whitney U-test and single-tailed paired Wilcoxon rank-signed test [104]. The former assumes the samples are independent and identically distributed (*iid*) while the latter assumes dependency between samples. Given that the experiments were conducted using static profiling (i.e. without temporal information) the former assumption is more plausible and therefore Mann-Whitney U-test was selected.

7.2 Experiment II: Imputation using Machine learning

This section presents the experiment results in using ML based methods to impute values with single and multi feature removal. Throughout this section, imputation improvement is characterised using the Δ (%) metric to facilitate comparison on the improvement made by the ML methods over median imputation.

7.2.1 Single feature removal

Table 7.1 shows the RMSE and Δ (%) metrics with respect to HOTS and CVTS which compares the best ML model with simple median imputation for each analyte. All seven ML models outperformed median imputation for each analyte but only the best model for each is presented. Even then, eight out of 14 analytes had MLP as the best model while LR performed best for the rest. On average, the ML models outperformed the simple median imputation by scoring 74.41% lower (better) across all 14 analytes on HOTS. A slightly lower percentage of 74.38% was obtained on CVTS, which despite having MLP as the dominant model, did not over fit to CVTS.

The scores for all analytes on HOTS are clustered into three main performance ranges: high, middle and low. Figure B.1 visualises this information for easier comparison. High performances were for HCT, HGB, LY, MCH, MCHC, MCV, NEUT, RBC and WBC scoring between 91.18% and 98.11% better than median imputation. EOS and MONO had mediocre performance scoring 49.96% and 75.41% better respectively. The lowest performance was for MPV, PLT and RDW scoring between 13.92% and 21.02% better. There is a (roughly) equal split between LR and MLP in the high performing group with five out of nine analytes scoring best with MLP. At the same time, all analytes in the low performing groups gave the lowest RMSE with MLP.

It is worthwhile comparing these results with the (linear) feature correlations explored in Section 6.3.4. It is no surprise that analytes that had the highest joint correlation coefficients (> 0.90), for example HCT, HGB and RBC, are in the high performing group while the same can be said about the low performing analytes which had almost no correlations with any other analytes. This demonstrates that both LR and MLP are able to inherently exploit feature correlations to impute missing values. The modality of MLP amongst the analytes suggests that many of them have non-linear relations with other analytes which is not captured by Pearson's correlation coefficient. For this reason, analyte pairs such as MCH, MCHC which had weak correlation values actually performed better with MLP than LR.

Analyte	Unit	Best model (BM)			Median (M)		Δ (%)		p-value	
		Type	HOTS	CVTS	HOTS	CVTS	HOTS	CVTS	BM	M
EOS*	10^9/L	LR	**0.058**	0.057	0.116	0.117	49.958	51.055	< 0.05	< 0.05
HCT$^+$	L/L	LR	**0.004**	0.004	0.065	0.065	94.495	94.537	0.358	< 0.05
HGB$^+$	g/L	MLP	**0.575**	0.643	22.078	21.187	97.394	96.965	0.418	< 0.05
LY$^+$	10^9/L	LR	**0.065**	0.066	0.741	0.759	91.180	91.343	< 0.05	< 0.05
MCH$^+$	pg	MLP	**0.058**	0.063	1.912	2.120	96.966	97.037	0.422	< 0.05
MCHC$^+$	g/L	MLP	**0.689**	0.702	9.812	9.729	92.982	92.780	0.380	< 0.05
MCV$^+$	fL	MLP	**0.204**	0.221	5.434	6.017	96.254	96.325	0.387	< 0.05
MONO*	10^9/L	LR	**0.064**	0.064	0.258	0.261	75.414	75.426	< 0.05	< 0.05
MPV$^-$	fL	MLP	**0.964**	1.012	1.120	1.142	13.915	11.381	0.054	< 0.05
NEUT$^+$	10^9/L	LR	**0.066**	0.066	2.733	2.589	97.589	97.442	0.150	< 0.05
PLT$^-$	10^9/L	MLP	**64.177**	67.418	79.060	81.135	18.826	16.907	< 0.05	< 0.05
RBC$^+$	10^{12}/L	MLP	**0.014**	0.018	0.759	0.743	98.109	97.530	0.425	< 0.05
RDW$^-$	%	MLP	**1.239**	1.439	1.569	1.917	21.021	24.968	< 0.05	< 0.05
WBC$^+$	10^9/L	LR	**0.066**	0.066	2.822	2.766	97.663	97.605	0.170	< 0.05
Average	-	-	**4.874**	5.131	9.177	9.325	74.412	74.379	0.198	< 0.05

Table 7.1: Table showing the RMSE, Δ (%) and Mann-Whitney U-test p-values on the held out test set (HOTS) and (five-fold) cross validation training set (CVTS) for each analyte and imputation method (best and median). The lowest HOTS scores (between best and median model) are highlighted in bold. **Keys:** *LR = linear regressor, MLP = multi layer perceptron*; Analyte performances are labelled high$^+$, medium* and low$^-$. $p < 0.05$ represents the significance level of the test.

Table B.3 presents the RMSE on CVTS for all the regression models tested. The final row in the table shows the average RMSE for each model across all the analytes. MLP gives the lowest RMSE which is closely followed by LR and SVR. The difference between the latter two is minimal. It is interesting to note the variety in the models as all three learn and capture relations in the data using different mechanisms. In contrast, the ensemble methods did not perform as well; XGB gave the worst performance overall with a RMSE that was higher than median imputation. Overall, the results indicate that if a single method were to be selected then MLP would give the best performance but if a simpler model is preferred than LR would suffice.

Figure 7.1 presents the NAE distributions for each analyte and corresponding imputation methods. It is evident from both central tendency (median) and statistical dispersion (IQR) that median imputation performed consistently worse than ML methods for all analytes. In general, the box plots show heterogeneous medians for all analytes except EOS and MONO which have positively skewed medians. The same can be observed for the dispersion of median imputation errors with an approximate average IQR of 0.2. In contrast, the error dispersion for all analytes are mostly in line with the RMSE scores. The analytes in the high performing group namely HCT, MCH, MCHC, MCV, NEUT, RBC and WBC all have negligible error dispersion indicating a high model certainty in predicting the values close to the true values. On the other hand, the analytes in the low performing group MPV, PLT and RDW show a higher relative dispersion compared to the other analytes but crucially remain below the median imputation dispersion. Likewise, the median values for the low performing group are below the respective median imputation errors. These results show that even the best performing ML model (MLP out of all seven tested) was not able to capture enough latent information to better predict values for low performing analytes than simply imputing them with their respective medians. It is likely the presence of all the other analytes adds noise to the predictor which degrades the quality of model building and ultimately performance.

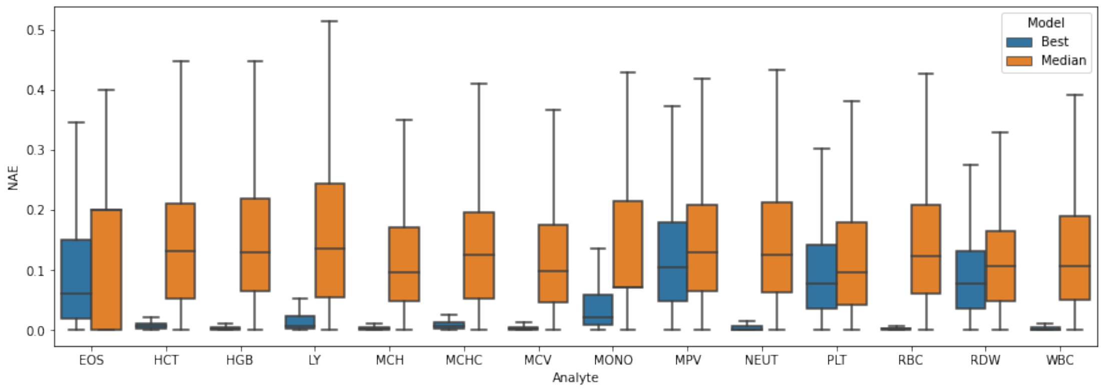


Figure 7.1: Diagram showing the NAE scores on the held out test set (HOTS) for each analyte and imputation method. The central tendency and statistical dispersion for median imputation is consistent but worse than best model for all analytes. There is more variation in the best model for each analyte which corresponds to their RMSE performances.

The final column of Table 7.1 shows the p-values for the Mann-Whitney U-test for each analyte and imputation method carried out on HOTS. As both RMSE and NAE metrics are directly derived from the raw values, the statistical test by extension, is valid for those two metrics as well. With the significance level set at $p < 0.05$, the table shows there is sufficient evidence that distribution of the median values differs from the true values. In contrast, the p-values for best ML methods were higher than the set threshold (nine of out fourteen) indicating there is sufficient evidence that their imputed values matches the distribution of the true data. This an encouraging result and proves that ML methods do not introduce bias for the best performing analytes.

To summarise, the ML methods outperform median imputation for all analytes giving the best performance for prediction of red and white blood cell analytes (in terms of lower RMSE). On the other hand, predicting missing values for platelet analytes is comparatively difficult and yields greater dispersion of NAE. Importantly, for all analytes, the central tendency and dispersion remains below the median imputation method. The Mann-Whitney U-test confirms that the ML methods match the distribution of true values when imputing for the best performing analytes. These findings are in line with the feature correlations obtained earlier (Section 6.3.4) as the best imputation accuracy was achieved for analytes with strong covariates. Likewise, the lack of correlations for platelet analytes means that they are inherently difficult to predict as they are isolated from the other two types of cells. Nevertheless, even in such a simple case, it is evident that simple models like linear regression are able to exploit the feature correlations to improve imputation accuracy which is more favourable than imputing with the median.

7.2.2 Multiple feature removal

Table 7.2 presents the RMSE and Δ (%) in the same way as before with three proportions of missing values: 10%, 30% and 50% for each analyte and imputation method. On average, this accounts to: 1.4, 4.2 and 7.0 analytes missing per test record. Note that the use of "missing values" is a proxy to simulate MAR but in actual fact those values are imputed using the feature median values during the prefilling stage prior to model learning. In general, the RMSE increases (worsens) on average with increasing proportion of missing values. This is expected because with increasing missingness there are fewer variables available (on average) per test record which intrinsically decreases the predictive power of the models. Despite this, MLP still performed best for all analytes with 10% and 30% missingness. This can likely be attributed to the increased density of the hidden layers (Section 6.2) which allows the networks to capture more complex latent relations between the analytes.

With 10% missing values, the performance of the analytes (in terms of Δ, %) most severely impacted the high performing group (from single feature removal). For example, LY and MCHC had large performance drops of 34.20% and 32.40% respectively. In contrast, only HCT and RBC had Δ above 90% with a drop of just 7.05% and 5.76% respectively from single feature removal. This highlights that MLP is a good fit for analytes with strong multi-covariates as it is able to retain its performance despite the "loss" of some true values. The mediocre performing pair, EOS and MONO, also recorded high performance drop to 21.57% and 49.27% respectively. In contrast, the low performing group was not impacted as severely with only a drop of approximately 4% for MPV, PLT and RDW. While consistency in model performance is desirable, in this case it confirms two things: (a) the loss of true data for other analytes does not impact their predictions and (b) the other analytes are likely not contributing towards prediction of their values.

A further performance drop was observed with 30% missing data which had approximately four variables prefilled using the median per test record. As before, the high performing group recorded the highest drop in performance (in terms of Δ, %) . The analytes with high covariates such as HCT and RBC recorded a drop in Δ of 21.22% and 23.76% respectively. It is clearly a consequence of loss of statistical power because the median does not contribute towards any variance for each analyte which subsequently reduces the predictive power of the models. The same observation was made for the mediocre performing analyte pair. In contrast, the extra 20% missingness did not impact the low performing analytes as severely and in the case of RDW the drop in performance was negligible.

It is not usual for laboratory data to have 50% of values for each analyte missing but it provides a way to evaluate the robustness of the methods with half the analytes missing. Regardless, the ML models still outperformed the median imputation for all analytes. It was interesting to note more diversity in the best model for the mediocre performing analytes (EOS and MONO) with RF and XGB marginally outperforming MLP. It is not a surprise that ensemble methods perform better with less variation in the analytes because they inherently create diverse tree models to mitigate against this. This has also been confirmed in recent empirical studies evaluating the use of XGBoost to predict missing values [83].

Figure 7.2 shows that Δ follows a diminishing trend as the proportion of missing data increases. This trend can also be described as an exponential decay as the drop in imputation accuracy between 10 - 30% is much more significant than 30 - 50%. This means that the loss of approximately two analytes (at random) has a more adverse impact on imputation accuracy than losing four. Evidently, the prefilling distorts the ability of the model to capture relevant patterns in the data. This is supported by the discussions from single feature removal which found that median imputation consistently gives errors for all analytes and therefore changes the distribution of the underlying data. At the same time, this likely explains why MLP dominated performance across all the other seven tested models because the dense neural architecture means that there are sufficient number of neurons that are trained with the actual values to give a decent performance overall.

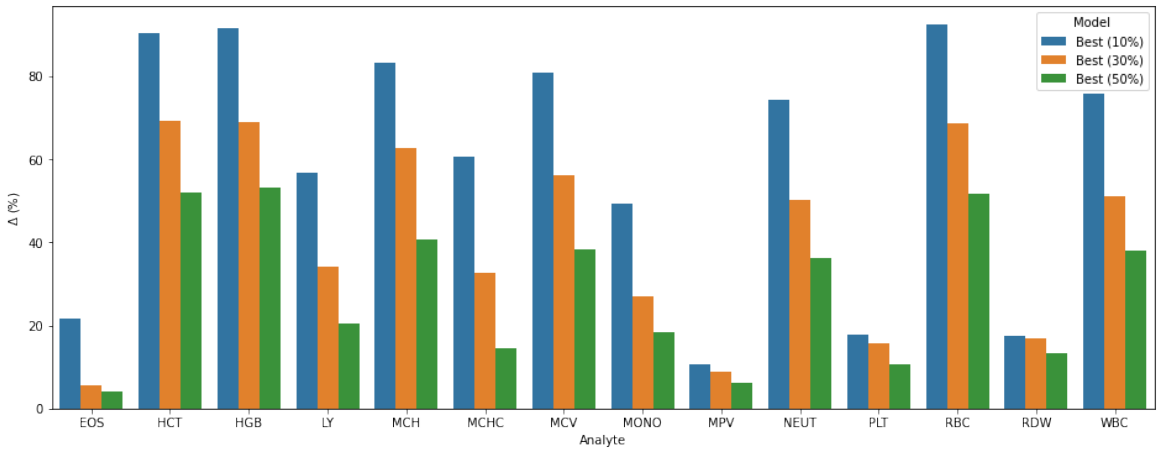


Figure 7.2: Diagram showing Δ (%) improvement on (HOTS) for each analyte and best ML method with different proportions of missing values (10, 30, 50%). In general, with increasing proportion of missingness the Δ drops exponentially.

Table B.4 presents the RMSE on CVTS for all the regression models with different proportion of missing values. The table shows that in many instances the difference between the models was marginal but in general MLP gave the best performance overall. The table shows variation amongst the other analytes and there are no discernable patterns on which models perform better. However, with increasing proportions of missing values the difference becomes slightly more apparent with SVR and RFs performing marginally better than their counterparts. It was disappointing to note the performance of XGB as in some instances, for example MCV, it performed significantly worse than median imputation. One reason could be the stochastic gradient descent algorithm not converging which impacted model learning.

Figure 7.3 shows the NAE for all analyte and imputation methods for different proportions of missing values. In general, increasing the proportion of missing values causes a greater variation in the NAE scores for high and medium performing analytes. The extent of variation is less for analytes such as HCT, HGB and RBC which are strong correlated with each other meaning that the ML models are able to minimise variation in prediction errors for such strong covariates. Crucially, the central tendency for these analytes remains below the median imputation showing robustness of the ML models with greater uncertainty. The analytes in the low performing group: MPV, PLT and RDW show heterogeneous central tendencies and negligible variation in statistical dispersion as they are not impacted by different proportions of missing values. In fact, whether they are imputed with the median or ML based method seems to make little difference to the distribution of errors.

The final column in Table 7.2 shows the p-values for the Mann-Whitney U-test for each analyte and imputation method carried out on HOTS for different proportions of missingness. Using the same significance level as before of $p < 0.05$, in general, the median imputation always has p-values that remain below the threshold except for EOS at 50% missingess and MCH for all missing proportions which shows that their distributions match the true values. The EOS result is likely an anomaly but the p-values are consistent for MCH indicating that median imputation is suitable strategy for that analyte. For the ML methods, the p-values drop between 10 - 50% which is expected as it becomes more difficult for the ML algorithms to preserve the distribution. Nevertheless, the ML methods are able to preserve the distribution up to 30% missing values for the best performing analytes. It is likely that they are robust with greater proportion of missingness (for instance 40%) but this is an interpolated estimate based on the results for 50% missing values.

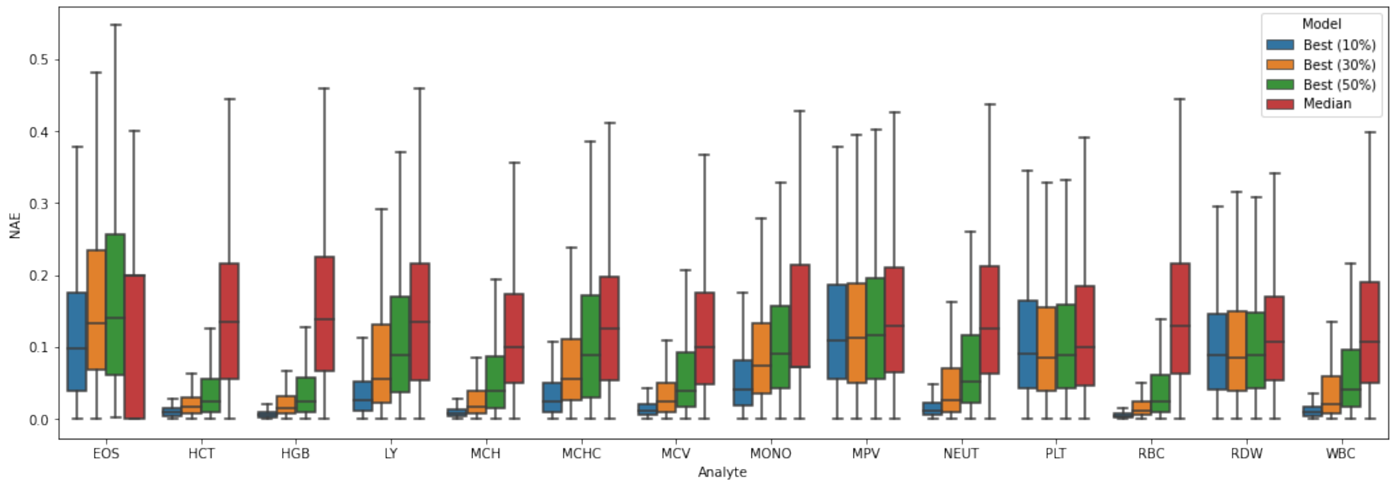


Figure 7.3: Diagram showing the NAE scores on HOTS for each proportion of missing values (10%, 30% and 50%) for each analyte and imputation method. In general, with increasing proportion of missing values the NAE increases linearly for analytes in the high and medium performing group. The low performing analytes consistently have the same NAE.

To summarise, the ML methods are robust (in terms of Δ) against different proportions of missing values for up to 50% for each analyte with a performance improvement of at least 4% based on the worst performing analyte (EOS). On average, the impact on imputation accuracy was more severe between 10 - 30% missing values than 30 - 50 % because it is likely that the most *important analytes* which are used by the models for regression were replaced by their respective median values. The drop in performance was also disproportionate with the high performing analytes being impacted more than the low performing group. This shows that the ML models are unable to discern any patterns for analytes with no covariates and are not significantly better than imputing them with the median. It was also observed that ensemble methods give a better performance than MLP at higher levels of missing values due to their ability to deal with greater uncertainties. Perhaps, ensemble methods will give better performance for more analytes with even greater proportions of missingness though from a practical point of view it is unlikely that a laboratory data set would have that many missing values. The ML methods were also able to preserve the distribution of data for up to 30% of missing values for the best performing analytes. On average across all the analytes with 10%, 30% and 50% missing values, the ML methods gave a Δ of: 58.74%, 40.52% and 28.39% respectively.

Analyte	Unit	Missing	Best model (BM)			Median (M)		Δ (%)		p-value	
		(%)	Method	HOTS	CVTS	HOTS	CVTS	HOTS	CVTS	BM	M
		10	MLP	**0.090**	0.087	0.115	0.117	21.567	25.641	< 0.05	< 0.05
EOS*	10^9/L	30	MLP	**0.109**	0.109	0.116	0.117	5.642	7.119	< 0.05	< 0.05
		50	RF	**0.112**	0.111	0.116	0.117	4.073	5.226	< 0.05	0.065
		10	MLP	**0.006**	0.008	0.065	0.065	90.334	87.692	0.458	< 0.05
HCT$^+$	L/L	30	MLP	**0.020**	0.020	0.066	0.065	69.116	68.673	0.195	< 0.05
		50	MLP	**0.031**	0.032	0.064	0.065	52.050	51.188	< 0.05	< 0.05
		10	MLP	**1.922**	2.610	22.781	21.187	91.564	87.681	0.444	< 0.05
HGB$^+$	g/L	30	MLP	**6.903**	6.401	22.106	21.187	68.774	69.790	0.051	< 0.05
		50	MLP	**10.357**	10.126	22.101	21.187	53.138	52.205	< 0.05	< 0.05
		10	MLP	**0.315**	0.336	0.730	0.759	56.791	55.731	0.468	< 0.05
LY$^+$	10^9/L	30	MLP	**0.484**	0.505	0.735	0.759	34.138	33.466	0.060	< 0.05
		50	MLP	**0.585**	0.599	0.735	0.759	20.385	21.037	< 0.05	< 0.05
		10	MLP	**0.330**	0.448	1.948	2.120	83.065	78.868	0.383	0.240
MCH$^+$	pg	30	MLP	**0.715**	0.823	1.920	2.120	62.773	61.197	0.350	0.107
		50	MLP	**1.143**	1.263	1.923	2.120	40.563	40.429	0.227	0.236
		10	MLP	**3.859**	3.883	9.789	9.729	60.578	60.088	0.382	< 0.05
MCHC$^+$	g/L	30	MLP	**6.536**	6.845	9.705	9.729	32.659	29.643	0.291	< 0.05
		50	MLP	**8.444**	8.659	9.870	9.729	14.449	10.992	< 0.05	< 0.05
		10	MLP	**1.047**	1.297	5.426	6.017	80.705	78.444	0.219	< 0.05
MCV$^+$	fL	30	MLP	**2.377**	2.605	5.406	6.017	56.026	56.696	0.312	< 0.05
		50	MLP	**3.369**	3.889	5.456	6.017	38.247	35.355	< 0.05	< 0.05
		10	MLP	**0.127**	0.141	0.249	0.261	49.269	45.977	0.056	< 0.05
MONO*	10^9/L	30	MLP	**0.186**	0.194	0.255	0.261	26.869	25.572	< 0.05	< 0.05
		50	XGB	**0.211**	0.216	0.258	0.261	18.383	17.333	< 0.05	< 0.05
		10	MLP	**0.983**	1.028	1.102	1.142	10.769	9.982	< 0.05	0.073
MPV$^-$	fL	30	MLP	**1.021**	1.054	1.121	1.142	8.858	7.650	0.193	< 0.05
		50	MLP	**1.051**	1.085	1.121	1.142	6.261	4.966	0.274	< 0.05
		10	MLP	**0.666**	0.784	2.598	2.589	74.360	69.718	0.340	< 0.05
NEUT$^+$	10^9/L	30	MLP	**1.340**	1.322	2.692	2.589	50.221	48.922	0.151	< 0.05
		50	MLP	**1.738**	1.713	2.729	2.589	36.305	33.843	0.071	< 0.05
		10	MLP	**65.540**	68.643	79.645	81.135	17.710	15.397	< 0.05	< 0.05
PLT$^-$	10^9/L	30	MLP	**66.889**	70.969	79.218	81.135	15.563	12.530	0.184	< 0.05
		50	MLP	**71.167**	73.578	79.565	81.135	10.556	9.314	< 0.05	< 0.05
		10	MLP	**0.060**	0.085	0.790	0.743	92.354	88.560	0.445	< 0.05
RBC$^+$	10^{12}/L	30	MLP	**0.236**	0.220	0.751	0.743	68.594	70.394	< 0.05	< 0.05
		50	MLP	**0.363**	0.354	0.753	0.743	51.718	52.401	< 0.05	< 0.05
		10	MLP	**1.270**	1.450	1.541	1.917	17.569	24.361	< 0.05	< 0.05
RDW$^-$	%	30	MLP	**1.307**	1.499	1.575	1.917	17.013	21.799	< 0.05	< 0.05
		50	MLP	**1.361**	1.566	1.572	1.917	13.448	18.315	< 0.05	< 0.05
		10	MLP	**0.695**	0.778	2.869	2.766	75.782	71.873	0.430	< 0.05
WBC$^+$	10^9/L	30	MLP	**1.382**	1.356	2.821	2.766	51.016	50.961	0.264	< 0.05
		50	MLP	**1.759**	7.498	2.830	2.766	37.851	35.590	< 0.05	< 0.05

Table 7.2: Table showing the RMSE, Δ (%) and Mann Whitney U-test p-values on HOTS and (five-fold) CVTS for each analyte and imputation method (best and median) for three proportion of missingness: 10%, 30% and 50%. The lowest HOTS scores (between best and median model) are highlighted in bold. **Keys:** *MLP = multi layer perceptron, RF = random forest, XGB = XGBoost*. Analyte performances are labelled high$^+$, medium* and low$^-$. $p < 0.05$ represents the significance level of the test.

7.3 Experiment III: Imputation using Bayesian Networks

This section presents the experiment results in using BNs to impute values with single and multi feature removal. The same evaluation metrics are used for this section including comparison of RMSE, Δ (%) and NAE to facilitate analysis in the same way as ML based methods.

7.3.1 Single feature removal

Figure 7.4 presents the topology of the best structure found using five-fold CVTS which was used to predict values for each analyte in this experiment. Each node in the BN represents an analyte and the presence of an edge (directed arrow) indicates causality. It should be noted that directed arrows sometimes indicate influence for example node A influences node B. In the context of this project, it is sufficient to use them interchangeably because the value of an analyte both influences and causes an effect on the other analytes. For instance, Figure 7.4 shows a directed arrow between HCT and HGB meaning that the value of HGB is influenced by HCT. From a medical perspective, this represents a causality relation as the percentage of red blood cells (measured by HCT) has a direct effect on the volume of haemoglobin (which constitutes red blood cells).

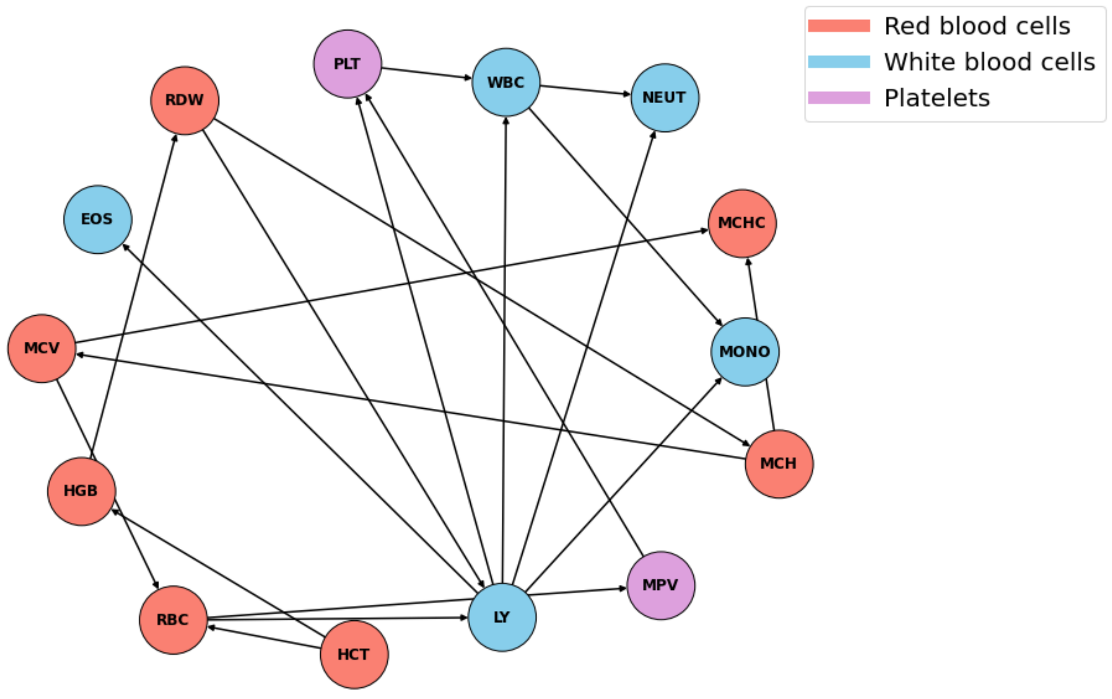


Figure 7.4: Diagram showing the structure of the BN learned from CVTS. The graph is colour coded by the categories: red blood cells (red), white blood cells (light blue) and platelets (violet). The directed arrows represent causality between analytes.

It is worthwhile briefly discussing the topology of the network from a medical viewpoint. In general, the network shows strong relations between analytes within the same cell category. For example, HGB and RBC have a common parent in HCT which shows that the pair of analytes are jointly influenced by HCT. In Section 6.3.4, it was found that the triplet have strong covariates but the BN structure is able to characterise the dependencies between those analytes. From an inference perspective, it means that the values for both HGB and

RBC can be predicted using HCT. The same observation is made for MCHC and MCV which share MCH as their parent analyte. It was interesting to find that the BN structure does not directly find relations between derived analytes, for instance MCHC which can be calculated using HGB and HCT. Instead, the structure finds an indirect relation via RDW and MCH which show dependencies HGB and HCT respectively. In terms of white blood cells, LY has the highest node degree; that is, LY is a parent analyte to all the other white blood cell and platelet analytes. This shows that it is an important analyte which dictates predictions for the other analytes. It is a surprise to see relations between LY and platelet cells as the feature correlations (Section 6.3.4) were not able to capture those. It is likely that this relation is specific to the physiology of the patients in the data set who are assumed to have COVID-19 (Section 6.1).

Table 7.3 presents the RMSE and Δ (%) metrics with respect to HOTS and CVTS which compares results from predicting values using the found BN with simple median imputation for each analyte. In general, the BN outperformed simple median imputation by scoring 38.00% lower (better) across all 14 analytes on HOTS. A slightly lower improvement of 35.91% was obtained for CVTS (across all analytes) so the BN actually generalised well to HOTS with an improved score.

Analyte	**Unit**	**Bayesian Network (BN)**		**Median (M)**		**Δ (%)**		**p-value**	
		HOTS	**CVTS**	**HOTS**	**CVTS**	**HOTS**	**CVTS**	**BN**	**M**
EOS$^-$	10^9/L	**0.103**	0.102	0.119	0.117	13.151	12.820	< 0.05	< 0.05
HCT$^+$	L/L	**0.025**	0.027	0.066	0.065	62.632	58.452	< 0.05	< 0.05
HGB$^+$	g/L	**8.380**	8.372	22.758	21.187	63.176	60.485	< 0.05	< 0.05
LY$^-$	10^9/L	**0.587**	0.590	0.738	0.759	20.475	22.267	0.124	< 0.05
MCH*	pg	**0.919**	0.920	1.917	2.120	52.072	56.603	0.328	< 0.05
MCHC*	g/L	**7.084**	7.080	9.812	9.729	27.800	27.227	< 0.05	< 0.05
MCV*	fL	**2.749**	2.733	5.346	6.017	48.573	54.579	< 0.05	< 0.05
MONO*	10^9/L	**0.217**	0.215	0.295	0.261	26.481	17.625	< 0.05	< 0.05
MPV$^-$	fL	**1.036**	1.033	1.120	1.142	7.533	9.545	< 0.05	< 0.05
NEUT$^+$	10^9/L	**1.137**	1.135	2.930	2.589	61.184	56.161	< 0.05	< 0.05
PLT$^-$	10^9/L	**67.472**	67.469	76.262	75.987	11.526	16.843	< 0.05	< 0.05
RBC$^+$	10^{12}/L	**0.325**	0.323	0.767	0.743	57.637	56.528	< 0.05	< 0.05
RDW*	%	**1.300**	1.298	1.965	1.917	33.808	32.290	< 0.05	< 0.05
WBC$^+$	10^9/L	**1.186**	1.184	2.759	2.766	57.026	57.195	< 0.05	< 0.05
Average	-	**6.609**	6.605	9.061	8.957	38.009	35.909	< 0.05	< 0.05

Table 7.3: Table showing the RMSE, Δ (%) and Mann-Whitney U-test p-values on (HOTS) and (five-fold) CVTS for each analyte and imputation method (BN and median). The lowest HOTS scores (between BN and median model) are highlighted in bold. **Keys:** Analyte performances are labelled high$^+$, medium* and low$^-$. $p < 0.05$ is the significance level of the test.

Following the same convention as experiment II (Section 7.2), the analytes can be clustered into three performance groups: high, medium and low. A notable difference to experiment II is that the proportion of analytes in each performance group for BN is more evenly spread with a ratio of five:five:four respectively. The best performances were for HCT, HGB, NEUT, RBC and WBC scoring between 57.03% to 63.18% better than simple median imputation. The medium performing group included MCH, MCHC, MCV, MONO and RDW scoring between 26.48% to 52.07%. The lowest performances were for EOS, LY, MPV and PLT scoring between 7.53% and 20.48% better. Both high and low performance groups have smaller ranges in comparison to the range of the medium performing group meaning that the distribution of performance is concentrated at both ends of the tail. In other words, the best and lowest performing analytes have comparable analyte performances within each group.

The structure found in Figure 7.4 can provide some plausible explanations for the results. The best performing analytes (HCT, HGB, NEUT, RBC and WBC) usually had multiple parent nodes (with the exception of HCT) so the *variable elimination* (VE) algorithm (Section 4.6) was able to eliminate the other (unconnected) nodes well in order to predominantly use the parent nodes to make the predictions. In contrast, the medium performing analytes did not exhibit the same type of multi-parent relations so the inference algorithm was not able to infer as strongly. For instance, the value of MCHC is derived using MCV which itself is derived from MCH which connects to MCHC so the triplet gave a mediocre performance overall. The performance of the lowest group can largely be attributed to LY which is a common parent to both PLT and MPV. This could be an erroneous relation captured by the BN which ultimately impacted the inferences for values of PLT and MPV. This demonstrates a limitation of BNs but also emphasises the importance of combining domain expertise which can mitigate against this by modifying the structure to better reflect patient physiology. In this experiment, a purely data-driven learning approach was used which works well for some analytes but not as much for others.

Figure 7.5 presents the NAE distributions for each analyte and imputation method. The graph shows that with the exception of EOS, all of the other analytes had lower central tendency (median) and dispersion (IQR) using BNs as compared to median imputation. The variation in the dispersion of errors amongst the analytes is comparable with their RMSE and Δ performances (Table 7.3) . For example, two of the analytes in the best performing group (HCT, HGB) have their medians and dispersion significantly below the NAE for median imputation. In contrast, the low performing group (LY, MPV and PLT) have higher medians and dispersion but they crucially remain below their counterpart median imputation. The performance of EOS can be attributed to its only parent node LY which could not provide as strong inference as expected. For this analyte, the BN is not better at predicting its values than using simple median imputation even though the RMSE of BN was slightly lower.

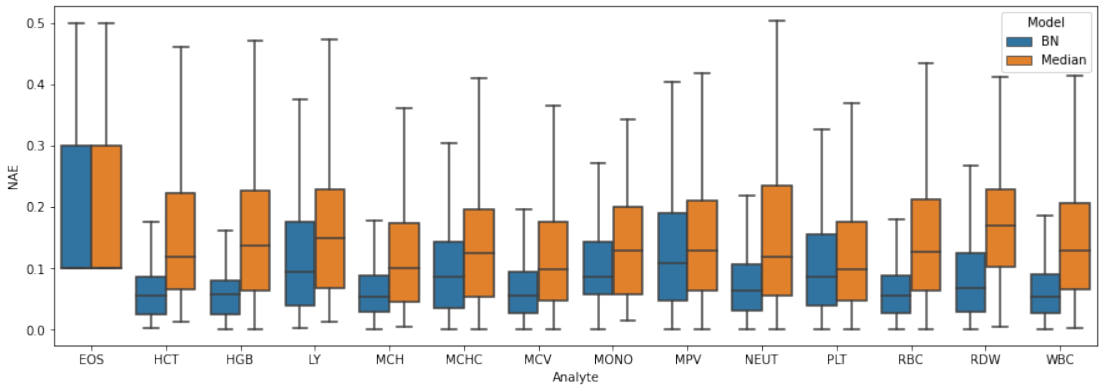


Figure 7.5: Diagram showing the NAE scores on HOTS for each analyte and imputation method. The central tendency and statistical dispersion for median imputation is consistent but worse in terms of higher median and dispersion than imputation using BNs for all analytes except EOS. EOS has median and dispersion matching exactly median imputation.

The final column of Table 7.3 presents the p-value results for the Mann-Whitney U-test for both the BN and median imputation. With the significance level set at $p < 0.05$, only two of the analytes for BN imputation show sufficient evidence of matching the distribution of true values. However, it is important to understand this result in the context of the experiment methodology (Section 6.5) as the continuous values were discretised into five uniform bins. In this case, the imputed values are compared directly with the true (undiscretised) values which yields the results presented here. Even though the size of the discretisation bins was selected based on the best performing results on five-fold CVTS (Section 6.5) it changes the distribution of the data so the statistical test is not as meaningful. In the case of median imputation, a single value is imputed for each variable which is reflected in the p-values showing strong evidence of not matching the distribution of the true values which was also observed in Section 7.2.1.

To summarise, with single feature removal BNs outperform the simple median imputation method for all analytes giving the best performance for prediction of red blood cells. This was a result of the BN capturing the relevant dependencies between the analytes under that cell category and learning an optimal structure which led to a high inference accuracy. In contrast, it was more difficult to predict values for white blood cells and platelets yet they still gave lower RMSEs than median imputation. The experiment also found from the analysis of NAE that all analytes (with exception of EOS) had their medians and dispersions below median imputation which demonstrates that BNs are able to predict with greater accuracy. The Mann Whitney U-test was carried out but on the basis that data was discretised the results were not as meaningful. Nevertheless, the NAE still gives a good indication of the overall performance of BNs across all the analytes.

7.3.2 Multiple feature removal

Figure 7.6 presents the structure of three BNs which were learned from the data with three proportions of missing values: 10% (a), 30% (b) and 50% (c). In general, with increasing proportion of missing values, the relations between the nodes remain fairly similar and the structure remains intact. This highlights the success of EM imputation as a prefilling strategy (Section 4.4) which generates a complete data set that is used for structure learning. For instance, even with increasing proportions of missing values, LY retains its highest node degree status even if some of the other nodes (for example WBC) were not able to do so. Yet, it is interesting to note that the biggest impact of missing values was on white blood cells (WBC, NEUT, EOS) which model fewer relations to other analytes at 50% missing values (as compared to 10%). In contrast, the red blood cells (HCT, HGB, HCT) retain their relations to other analytes at 50% missing values. The platelet analytes were not impacted as much except at 50% the best structure (found using Hill Climb Search algorithm, Section 4.3) reversed the arrow direction but since PLT and MPV are closely related this likely did not impact their inference so much.

Table 7.5 presents the RMSE and Δ (%) metric with the same three proportions of missing values for each analyte and imputation method. In general, with increasing proportion of missing values the RMSE of BNs increases (worsens) but always remains below (better) their counterpart median imputation errors. In other words, BNs are able to outperform median imputation for up to 50% of missing values. This is an encouraging result which validates the use of BNs in this project as they are able to use the structure of the found network to impute values that are closer to the true values even with such high proportions of missing values. It shows how the inference algorithm (Section 4.5.2) is intrinsically able to deal with different combinations of missing values for each test record queried. The random nature of missingness makes this even harder but demonstrates the potential of BNs at such high degree of uncertainty.

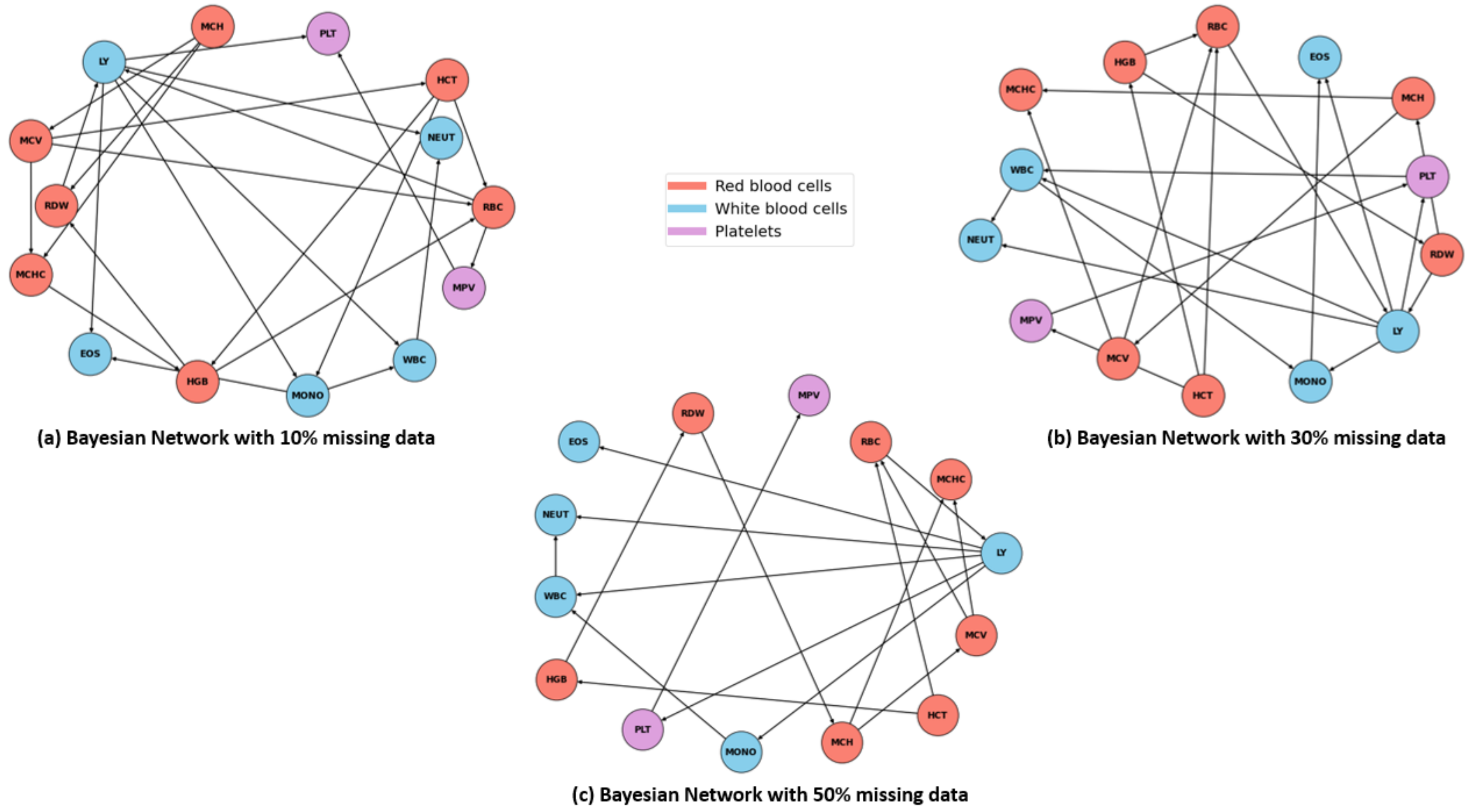


Figure 7.6: Diagram showing the structures of BNs found for three proportions of missing values: 10% (a), 30% (b) and 50% (c). The graph is colour coded by the categories: red blood cells (red), white blood cells (light blue) and platelets (violet). Across all proportions of missingness, the structure roughly remains intact with many of the relations present in all three graphs.

With 10% missing values, the biggest impact on performance (in terms of Δ, %) was for the medium and low performing analytes (when compared to single feature removal). For instance, EOS had a drop in Δ from 13.15% to 5.88% showing a relatively large impact on an already low performing analyte. Likewise, LY recorded a drop from 20.48% to 16.53%. Despite this, both analytes still recorded positive values so they performed marginally better than median imputation. In this instance, since the difference is so small it may be better altogether to use simple median imputation instead. In contrast, the high performing analytes were not impacted as much recording only a small drop in Δ. For example, HCT which had the second highest value for Δ in single feature removal, only fell by 2.63% to 60.00% for 10% missing values. Likewise, the performance for HGB fell by only 3.38% to 59.80%. These performances correlate with the structure of the BN presented in Figure 7.6(a) as the best performing analytes had multiple relations as compared to the low performing group.

At 30% missing data the same trend in performance drop was observed. As with 10% missing values, the performance drop most significantly impacted the low and medium performing analytes. In fact, EOS gave the poorest performance across all the other analytes with just an improvement (in terms of Δ) of 1.70%. From the results of EOS at 50% missingness, it appears that it reached its saturation point with negligible improvement after 30%. For the other low performing analytes, the drop in performance continued linearly as opposed to exponential, which is an expected result given the inference algorithm has fewer available variables. In contrast, the high performing group generally recorded a larger performance drop which could partly be due to the fact they already perform so well (in comparison to others) so the impact of losing values would be more severe.

At the highest proportion of missing values (50%), the biggest performance impact was on high performing analytes because the low performing analytes had already dropped to low values. Nevertheless, the fall was approximately between 10% to 20% when compared to 10% missing values. This is actually a promising result because the BNs are robust against losing 40% of more information. This is not a surprise because parameter learning (Section 4.3) inherently deals with missing values and makes an estimate for the probability distributions based on the captured dependencies and observed values. This is supported by the EM prefilling strategy. An interesting (possibly anomalous) result was observed for MONO which actually had a (slightly) better value for Δ at 50% missingness as compared to 10%. The fact that it has LY and WBC as parent analytes may be a plausible explanation for this result.

Figure 7.7 visualises the diminishing trend in Δ with increasing proportion of missing values. In general, there is variation in the trend for each analyte but the impact of performance is most visible for the best performing analytes which have the highest bar heights. For instance, MCV and NEUT record large drops between 10% to 30%. The magnitude of their fall is larger for this range than 30% to 50% showing that the BNs are more consistent at high proportions of missing values for the best performing analytes. This supports the discussion earlier on the suitability of BNs at high proportions of missing values. In contrast, EOS and MPV recorded a low value of Δ which only deteriorated with increasing proportion of missing values. While Figure 7.6 shows that both analytes have at least one parent in all configurations, they were insufficient to make a more intelligible prediction than imputing with the median. This does not outweigh the strong performance observed for the best performing analytes.

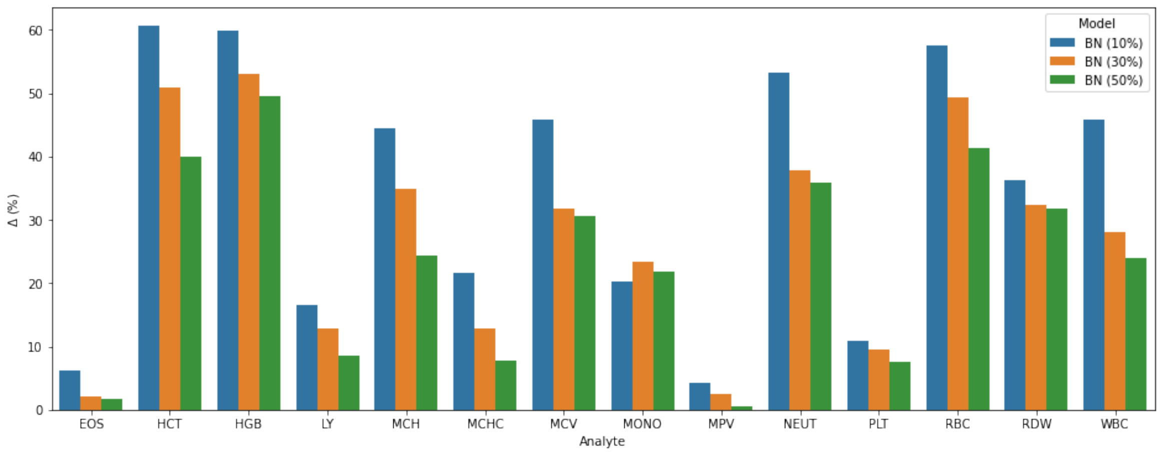


Figure 7.7: Diagram showing Δ (%) improvement on (HOTS) for each analyte and BNs with different proportions of missing values (10%, 30%, 50%). In general, with increasing proportion of missingness the value of Δ drops for most analytes but at different rates.

Figure 7.8 presents the NAE distributions for all analytes and imputation methods with different proportions of missing values. In general, with the exception of EOS, the distribution of NAE for all the analytes at every missing percentage remains below their counterpart median imputation errors. This shows that the BNs are able to infer values that are closer to true values than median imputation. The box plots also show that all analytes have heterogeneous medians and little variation in dispersion (for BNs) with increasing proportion of missing values. This shows the consistency of the model, even if it may not yield to a lower RMSE, it emphasises how BNs can comfortably deal with increasing proportions of uncertainty without increasing in erroneous predictions. The consistency is most notable for the best performing analytes such as HGB which has almost identical box plots at all three proportions of missing values. Even for low performing analytes (LY, PLT), the distribution shows that their central tendencies remain consistent across all three missing proportions.

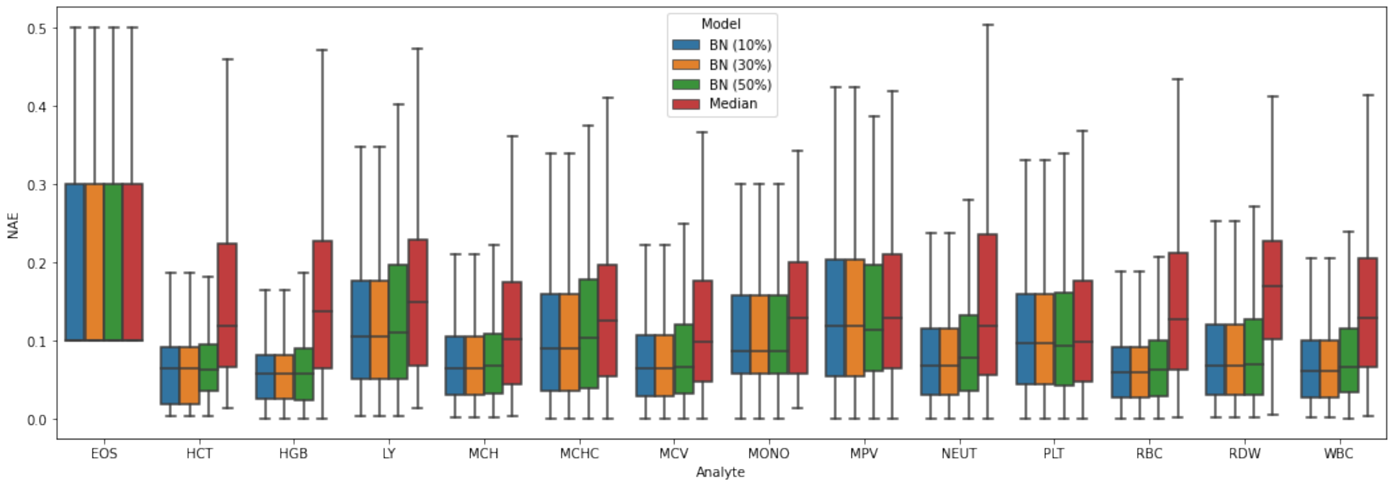


Figure 7.8: Diagram showing the NAE scores on HOTS for each proportion of missing values (10%, 30% and 50%) for each analyte and imputation method. In general, with increasing proportion of missing values the NAE increases linearly for analytes in the high and medium performing group. The low performing analytes consistently have the same NAE.

To summarise, BNs are able to outperform median imputation by reducing the RMSE by at least 24% for the best and medium performing analytes with up to 50% missing values. The best performances (lowest RMSEs) were observed for red blood cells analytes which had the largest value for Δ across all three proportions of missing values. This can be explained by the relations of those analytes modelled in Figure 7.6 which did not change with increasing proportions of missing values. This can be attributed to the strong performance of the EM prefilling method which maximises log-likelihood for optimised structure learning. In contrast, the BNs found it more difficult to predict EOS, LY and PLT despite being able to capture their dependencies correctly. The diminishing trend in Δ was observed for the best performing analytes but this tended to plateau between 30% to 50% as the BNs were able to retain their inference power. The NAE distribution found that all analytes (except EOS) had their error distribution below median imputation which emphasises the success of BNs. On average across all the analytes with 10%, 30% and 50% missing values, the BNs gave a Δ of: 34.55%, 27.13% and 23.23% respectively.

Analyte	Unit	Missing	Bayesian Network (BN)		Median (M)		Delta (%)		p-value	
		(%)	HOTS	CVTS	HOTS	CVTS	HOTS	CVTS	BN	M
		10	**0.112**	0.113	0.119	0.117	5.882	3.419	< 0.05	< 0.05
EOS$^-$	10^9/L	30	**0.116**	0.114	0.118	0.117	1.695	2.564	< 0.05	< 0.05
		50	**0.117**	0.115	0.119	0.117	1.681	1.709	< 0.05	< 0.05
		10	**0.026**	0.025	0.065	0.065	60.000	61.539	< 0.05	< 0.05
HCT$^+$	L/L	30	**0.033**	0.032	0.066	0.065	50.001	50.769	< 0.05	< 0.05
		50	**0.040**	0.037	0.066	0.065	39.394	43.077	< 0.05	< 0.05
		10	**9.148**	9.149	22.758	21.198	59.803	56.840	< 0.05	< 0.05
HGB$^+$	g/L	30	**10.684**	10.687	22.755	21.198	53.054	49.585	< 0.05	< 0.05
		50	**11.480**	11.475	22.758	21.198	49.556	45.867	< 0.05	< 0.05
		10	**0.616**	0.614	0.738	0.748	16.531	17.914	< 0.05	< 0.05
LY$^-$	10^9/L	30	**0.643**	0.639	0.736	0.748	12.636	14.572	< 0.05	< 0.05
		50	**0.676**	0.675	0.738	0.748	8.401	9.759	< 0.05	< 0.05
		10	**1.063**	1.062	1.917	2.109	44.549	49.644	< 0.05	< 0.05
MCH*	pg	30	**1.248**	1.247	1.920	2.109	34.998	40.872	< 0.05	< 0.05
		50	**1.449**	1.446	1.921	2.109	24.571	31.437	< 0.05	< 0.05
		10	**7.684**	7.682	9.812	9.638	21.688	20.295	< 0.05	< 0.05
MCHC*	g/L	30	**8.544**	8.542	9.815	9.638	12.950	11.372	< 0.05	< 0.05
		50	**9.045**	9.043	9.810	9.638	7.799	6.173	< 0.05	< 0.05
		10	**2.897**	2.895	5.346	6.011	45.810	51.838	< 0.05	< 0.05
MCV*	fL	30	**3.651**	3.648	5.346	6.011	31.706	39.311	< 0.05	< 0.05
		50	**3.705**	3.703	5.346	6.011	30.696	38.396	< 0.05	< 0.05
		10	**0.235**	0.232	0.295	0.276	20.339	15.942	< 0.05	< 0.05
MONO*	10^9/L	30	**0.226**	0.224	0.298	0.276	24.161	18.841	< 0.05	< 0.05
		50	**0.231**	0.228	0.296	0.276	21.960	17.391	< 0.05	< 0.05
		10	**1.072**	1.069	1.120	1.141	4.286	6.310	< 0.05	0.073
MPV$^-$	fL	30	**1.091**	1.088	1.122	1.141	2.763	4.645	< 0.05	< 0.05
		50	**1.114**	1.109	1.121	1.141	0.624	2.805	< 0.05	< 0.05
		10	**1.368**	1.364	2.930	2.859	53.311	52.291	< 0.05	< 0.05
NEUT$^+$	10^9/L	30	**1.819**	1.816	2.931	2.859	37.939	36.481	< 0.05	< 0.05
		50	**1.879**	1.875	2.930	2.859	35.870	34.418	< 0.05	< 0.05
		10	**67.929**	67.924	76.262	76.299	10.927	10.977	< 0.05	< 0.05
PLT$^-$	10^9/L	30	**68.949**	68.946	76.269	76.299	9.598	9.637	< 0.05	< 0.05
		50	**70.493**	70.497	76.265	76.299	7.568	7.604	< 0.05	< 0.05
		10	**0.325**	0.324	0.767	0.743	57.627	56.393	< 0.05	< 0.05
RBC$^+$	10^{12}/L	30	**0.388**	0.385	0.768	0.743	49.479	48.183	< 0.05	< 0.05
		50	**0.449**	0.447	0.767	0.743	41.460	39.838	< 0.05	< 0.05
		10	**1.252**	1.254	1.965	1.938	36.285	35.294	< 0.05	< 0.05
RDW*	%	30	**1.328**	1.325	1.966	1.938	32.452	31.631	< 0.05	< 0.05
		50	**1.342**	1.341	1.968	1.938	31.809	30.805	< 0.05	< 0.05
		10	**1.496**	1.494	2.759	2.765	45.777	45.967	< 0.05	< 0.05
WBC$^+$	10^9/L	30	**1.984**	1.981	2.760	2.765	28.116	28.354	< 0.05	< 0.05
		50	**2.101**	2.099	2.759	2.765	23.849	24.087	< 0.05	< 0.05

Table 7.4: Table showing the RMSE, Δ (%) and Mann Whitney U-test p-values on HOTS and (five-fold) CVTS for each analyte and imputation method (BN and median) for three proportion of missingness: 10%, 30% and 50%. The lowest HOTS scores (between BN and median model) are highlighted in bold. **Keys:** Analyte performances are labelled high$^+$, medium* and low$^-$. $p < 0.05$ represents the significance level of the test.

7.4 Experiment IV: Comparison of Machine learning and Bayesian Networks

This section presents the experiment results to compare the performance of ML and BNs to impute values with single and multi feature removal. This section compares the RMSE between the two methods as the Δ metric was used to compare how the improvement over the simple median imputation. Likewise, experiment III (Section 7.3) discussed implications of discretising data on the distributions so analysis for Mann-Whitney U-test is not presented here for the same reasons.

7.4.1 Single feature removal

The results from single feature removal in experiment II (Section 7.2.1) and III (Section 7.3.1) can be briefly compared to understand how the respective methods perform with respect to simple median imputation. In experiment II, the ML methods outperformed (on average, across all analytes) the median imputation methods by 74.41% while BNs outperformed by the same by 38.00%. Just from this result, it can be deduced that ML based methods perform twice as well as BNs. However, it is important to note that the two results are not directly comparable as the data was discretised for BNs (Section 6.6) but not for ML methods (Section 6.5). Therefore, this experiment presents the results of discretising data for ML methods making the analysis more meaningful and valid for a direct comparison.

The RMSE scores for single feature removal with BNs are extracted directly from Section 7.3.1. New results were obtained for the best performing ML methods from experiment II (Section 7.2.1) using the same uniform discretisation pre-processing step to facilitate direct comparison of the two approaches. Table 7.5 presents the RMSE for ML based and BN imputation methods on HOTS and CVTS.

Analyte	Unit	Machine Learning (ML)			Bayesian Network (BN)	
		Type	HOTS	CVTS	HOTS	CVTS
EOS	10^9/L	**LR**	0.108	0.107	**0.103**	0.102
HCT	L/L	**LR**	**0.017**	0.017	0.025	0.027
HGB	g/L	**MLP**	**5.784**	5.863	8.380	8.372
LY	10^9/L	**LR**	0.591	0.536	**0.587**	0.590
MCH	pg	**MLP**	**0.624**	0.624	0.919	0.920
MCHC	g/L	**MLP**	**6.625**	6.598	7.084	7.080
MCV	fL	**MLP**	**2.061**	2.056	2.749	2.733
MONO	10^9/L	**LR**	**0.202**	0.200	0.217	0.215
MPV	fL	**MLP**	**0.993**	0.998	1.036	1.033
NEUT	10^9/L	**LR**	**0.872**	0.844	1.137	1.135
PLT	10^9/L	**MLP**	**65.496**	66.922	67.472	67.469
RBC	10^{12}/L	**MLP**	**0.209**	0.215	0.325	0.323
RDW	%	**MLP**	**1.242**	1.314	1.300	1.298
WBC	10^9/L	**LR**	**0.744**	0.716	1.186	1.184
Average	-	**-**	**6.112**	6.638	6.609	6.605

Table 7.5: Table showing the RMSE on HOTS and five fold CVTS for machine learning (ML) and Bayesian network (BN) methods. The lowest HOTS scores (between ML and BN) are highlighted in bold.

In general, the ML based methods give a comparatively lower RMSE score for 12 out of 14 analytes while BNs score lower for EOS and LY. On average, across all the analytes, the ML based methods perform 8.14% better than their counterpart which shows that there is a small magnitude of difference between the two methods. The best relative performance for the ML methods is the prediction of HGB which has a RMSE score that is 44.88% lower (better) than BN. This is likely because the MLP was able to train its neurons to capture non-linear relations while the BNs solely rely on dependencies between analytes. This raises an important point - the two methods build their respective models differently and inherently use different mechanisms to predict/infer values (Sections 4.3 - 4.5). Having said this, the BN performs better for analytes which were predicted using LR in the ML approach though the difference is marginal. This shows that inference can be powerful and utilising conditional probabilities can sometimes yield better predictions than ML based methods (as is the case here).

The results for the ML models clearly show that discretisation has an impact on performance when comparing the RMSE scores with single feature removal in experiment II (Section 7.2.1). In this experiment, the RMSE score for each analyte is amplified by up to ten times especially for the best performing group from experiment II. For example, HGB gave a RMSE of 0.575 in experiment II but a score of 5.784 in this experiment. The same observation can be made for MCHC which had an approximately ten fold increase. The discretisation does not impact the low performing analytes such as PLT or RDW as the difference to experiment II is minimal. For the same reason, the Mann Whitney U-test shows strong evidence for a different distribution when data is discretised as the values are assigned into bins as opposed to a continuous spectrum.

To summarise, the results from single feature removal indicate that both methods are comparable with ML based methods performing (on average) 8.14% better than BN methods. The discussion presented the impact of dicretisation on the performance on ML based methods, which by extension, imply that BNs were also impacted for the same reason. Nevertheless, the results are encouraging and present a strong use case for both methods.

7.4.2 Multiple feature removal

As before, the results for multiple feature removal from experiment II (Section 7.2.2) and III (Section 7.3.2) can be briefly discussed prior to presenting the results for this experiment. The best way to compare the two approaches is to compare their respective trends in Δ with increasing proportion of missing values as shown by Figures 7.2 and 7.7. This provides an insight into how the models deal with increasing level of uncertainty. There some notable similarities and differences between the results. In general, increasing proportion of missing values led to a performance drop for both methods but impacted to a different extent. For example, Figure 7.2 shows an exponential decay in Δ for all the analytes while Figure 7.7 shows that the greatest impact was between 10% to 30% missing values for only the best performing analytes. This demonstrates that BNs are intrinsically able to deal with higher levels of uncertainty without a significant loss in inference power. However, ML based methods performed better overall as the best performing analyte (HGB) recorded a Δ of 54.14% as compared to the 49.56% with BNs. It is interesting to note that HGB gave the best performance in both methods meaning it was easy for the models to capture its relations with other analytes to predict its values.

Table 7.6 presents the RMSE scores for each analyte and imputation method with the three proportions of missing values. The results for the BNs were obtained from experiment III (Section 7.3.2) while new results for ML based methods are presented here using the same best model for each analyte at each missing proportion as in experiment II (Section 7.2.2) with discretised data. In general, the trends in performance with increasing missing proportions are consistent with the above discussions. The ML based methods, on the whole, perform better than BNs although this diminishes with a linear trend with increasing proportion of missing values. For example, at 10% missing values, the ML methods give a lower (better) relative RMSE score for 12 out of 14 analytes which drops to ten at 30% and finally six at 50%. Likewise, a rising trend is observed for BNs as they perform better than ML based methods at higher levels of uncertainty. This supports and confirms the observations made earlier as BNs are more consistent at maintaining their performance between 30% and 50% missing values.

As with single feature removal (Section 7.4.1), the RMSE scores confirm that discretisation had the largest impact on the performance of best scoring analytes from experiment III (Section 7.3.2). Although it makes the results between the two approaches more comparable, discretisation is unfavourable from a ML perspective because it reduces the precision of data and therefore impacts the fidelity of ML models. Some ensemble methods such as DTs and RFs are inherently invariant to such changes in data and therefore more likely to perform better. With this being said, the performance impact is disproportionate with the best performing analytes (HCT, HGB) showing a ten times increase in RMSE (when compared to Section 7.3.2) while almost no impact on the low performing analytes (PLT, RDW). The range of the analytes could offer a plausible explanation for this; HCT and HGB have smaller range between 0.0 to 0.5 and 0.2 to 0.6 respectively while PLT and RDW have larger ranges between 0 - 500 and 8 - 20 respectively (Section 6.3.3). As such, the RMSE will obtain larger scores for incorrect predictions with smaller range analytes but will not be impacted as much for the larger ranges.

To summarise, two conclusions can be drawn from the results: (a) the performance of BNs is better than their counterpart ML methods at higher proportions of missing values and (b) discretisation has a severe impact on the best performing analytes for ML based methods. The first conclusion emphasises how BNs intrinsically deal with missing data and can utilise the entire structure to make inferences as opposed to ML based methods which would suffer from loss of information. The second conclusion confirms discussion on discretisation with single feature removal (Section 7.4.1) but more importantly implies that BNs could perform better if the data is not discretised.

Analyte	Unit	Missing	Machine Learning (ML)			Bayesian Network (BN)	
		(%)	Method	HOTS	CVTS	HOTS	CVTS
EOS	10^9/L	10	MLP	**0.112**	0.114	**0.112**	0.113
		30	MLP	**0.110**	0.108	0.116	0.114
		50	RF	**0.114**	0.116	0.117	0.115
HCT	L/L	10	MLP	**0.018**	0.019	0.026	0.025
		30	MLP	**0.027**	0.030	0.033	0.032
		50	MLP	**0.038**	0.039	0.040	0.037
HGB	g/L	10	MLP	**5.939**	5.935	9.148	9.149
		30	MLP	**8.836**	8.832	10.684	10.687
		50	MLP	12.598	12.599	**11.480**	11.475
LY	10^9/L	10	MLP	**0.552**	0.550	0.616	0.614
		30	MLP	**0.622**	0.620	0.643	0.639
		50	MLP	0.690	0.698	**0.676**	0.675
MCH	pg	10	MLP	**0.713**	0.708	1.063	1.062
		30	MLP	1.330	1.329	**1.248**	1.247
		50	MLP	1.961	1.964	**1.449**	1.446
MCHC	g/L	10	MLP	**7.132**	7.130	7.684	7.682
		30	MLP	8.966	8.962	**8.544**	8.542
		50	MLP	10.793	10.791	**9.045**	9.043
MCV	fL	10	MLP	**2.298**	2.295	2.897	2.895
		30	MLP	4.101	4.099	**3.651**	3.648
		50	MLP	6.131	6.128	**3.705**	3.703
MONO	10^9/L	10	MLP	**0.202**	0.201	0.235	0.232
		30	MLP	**0.202**	0.204	0.226	0.224
		50	XGB	**0.222**	0.225	0.231	0.228
MPV	fL	10	MLP	**1.020**	1.018	1.072	1.069
		30	MLP	**1.037**	1.035	1.091	1.088
		50	MLP	**1.102**	1.089	1.114	1.109
NEUT	10^9/L	10	MLP	**1.128**	1.125	1.368	1.364
		30	MLP	**1.560**	1.561	1.819	1.816
		50	MLP	1.938	1.935	**1.879**	1.875
PLT	10^9/L	10	MLP	**64.623**	64.623	67.929	67.924
		30	MLP	69.168	69.156	**68.949**	68.946
		50	MLP	73.512	73.506	**70.493**	70.497
RBC	10^{12}/L	10	MLP	**0.223**	0.220	0.325	0.324
		30	MLP	**0.309**	0.304	0.388	0.385
		50	MLP	**0.433**	0.425	0.449	0.447
RDW	%	10	MLP	1.260	1.265	**1.252**	1.254
		30	MLP	**1.315**	1.320	1.328	1.325
		50	MLP	1.363	1.361	**1.342**	1.341
WBC	10^9/L	10	MLP	**1.083**	1.079	1.496	1.494
		30	MLP	**1.534**	1.528	1.984	1.981
		50	MLP	**1.973**	1.970	2.101	2.099

Table 7.6: Table showing the RMSE on HOTS and five five CVTS for each analyte and imputation method (ML and BN) for three proportion of missingness: 10%, 30% and 50%. The lowest HOTS scores (between BN and median model) are highlighted in bold. **Keys:** *MLP = multi layer perceptron, RF = random forest, XGB = XGBoost.*

7.5 Summary

In summary, this chapter presented several key results which provided an insight into how the two models behave with different levels of missing values. Overall, the most encouraging result is that both ML based and BNs were able to outperfom the simple median imputation for each analyte with up to 50% of missing values under the MAR assumption. In experiment II (Section 7.2), the results found that ML based methods scored 74.12% lower (better) with single feature removal while a diminishing trend in Δ was observed for all the analytes with multiple feature removal. Likewise, in experiment III (Section 7.3), the results found that BNs scored 38.00% lower (better) with single feature removal and had a falling trend (but not to the same extent as ML based methods) with multiple feature removal. In comparison, BNs were better at dealing with higher levels of uncertainty than ML based methods. In experiment IV (Section 7.4), a direct comparison was made which found that ML based methods performed 8.14% better with single feature removal. The results with multiple feature removal were more varied but BNs gave a better relative performance for 30% to 50% missing values. All results were facilitated with discussions on the key strengths and limitations of the models with discretisation in experiment III-IV being a limiting factor (Section 7.3 - 7.4). These should be taken into consideration prior to selecting methods for use in a practical setting.

Chapter 8

Evaluation

In Chapter 7, the results of experiments II - IV were presented which evaluated the imputation methods on a real-life laboratory data set. This chapter presents a more formal evaluation of the project requirements (Section 8.1) and provides a comparison of the imputation methods investigated in this project (Section 8.2).

8.1 Meeting project requirements

This section evaluates the contributions made in this project with respect to the project specification outlined in Chapter 3. Overall, the project has met all of its objectives by completing the delivery of four outlined deliverables: design of imputation framework, implementation of steps in the framework including data imputation methods, testing and analysing the performance of the implemented methods via multiple experiments and proving key results as part of supporting online documentation.

8.1.1 Design of imputation framework

The first objective of the project was to design an imputation framework that provides a suitable methodology to impute missing values in laboratory data (Section 3.1). The imputation framework was successfully designed and key design decisions were made for the two workflows (Section 4.1). The design methodology followed an incremental and iterative approach which was continuously improved until it satisfied the design constraint of supporting two imputation approaches: ML and BNs. For example, a key design decision was made to split the workflow after stage one as the two approaches use different prefilling and pre-processing steps to prepare the data prior to model learning.

The framework methodology was used to devise a suitable strategy to test the implemented methods and meticulously followed in the experiments described in Chapter 6. The modular design of the framework made it easier to modify its stages when required, for instance, in experiment I (Section 6.2) where exploratory data analysis was carried out after feature selection to understand feature correlations between the analytes. In fact, the design modularity was best utilised by separating experiments I and II-IV which made it easier to explain the motivations behind the decisions made in the latter experiments. This can be replicated in a practical real-life setting where further exploratory data analysis can be carried out after stage one. This can be useful for clinicians to understand the nature of the underlying data set. Likewise, each of the other stages (two to four), can be modified to select different prefilling (Section 4.2), pre-processing (Section 4.4) and model learning (Section 4.3) strategies. For instance, in this project eight regression models (Section 4.5.1) were selected but other regression models can be added provided they are compliant with Scikit-learn's API. Similarly, different structure (Section 4.5.2) and parameter learning algorithms (Section 4.5.2) can be substituted to test their performance.

8.1.2 Implementation of imputation framework

The second objective of the project follows the first as it was to implement the stages and imputation methods selected in the imputation framework. In Chapter 5, the implementations details describe how each stage of the framework was developed including design (and implementation) of Python `wrappers` to encapsulate functionalities from other libraries such as `pgmpy` (which were used to build BNs). These wrappers inherently exploit OOP inheritance as they build on top of functionality provided by existing libraries. The implementations are compliant with Scikit-learn's API and can be directly used as regressor models or transformers as appropriate (Section 5.7). The Python wrappers were tested manually and using automated testing framework using simple unit tests that verify their functionality.

One of the contributions made through implementations is the creation of a Python library, named `labimputer`, which contains all of the classes and methods implemented in this project. This includes transformer classes: `EMImputer` and `BNImputer` which can prefill using the EM method (Section 4.3.2) and the variable elimination algorithm (Section 4.5.2) respectively. Three regression classes were also implemented: `IterativeImputerRegressor`, `SimpleImputerRegressor` and `BNRegressor`. The regressions models were used extensively in experiments II-IV (Sections 6.4 - 6.6) to collect the results. As such, they have been verified for compatibility with Scikit-learn and therefore functionality in a practical setting.

In hindsight, the decision to use an *object-orientated* approach (in addition to meeting the requirements) was well justified. For example, the modularity of the imputation framework complimented the implementation work as a Python class was written for each stage of the framework. The use of classes facilitated greater flexibility in initialising methods with the desired parameters. For example, the `BNImputer` class can be initialised with a BN structure if provided by the user which can then be used for parameter learning and inference (Section 5.6.1). This way, the user has more control on how to configure the imputation methods for the task at hand. It also increases the extensibility and reusability of the code as the current implementations can be extended to provide additional desired functionality. Since the project code is encapsulated as a package, the relevant classes can be easily imported and directly used in other projects. It is anticipated that there will be more contributions so the provision of the current set up should make this easy to do.

8.1.3 Experiments on real-life laboratory data set

The third objective of the project was to test the implemented imputation methods on a real-life laboratory data set. In Chapter 6, the materials and methodology of four experiments were presented which were carried out on a laboratory data set provided by Imperial College NHS Healthcare Trust. The aim of the experiments was to investigate the performance of two imputation methods (ML based and BNs) using the designed framework and ideally outperform simple median imputation. In Chapter 7, the results were duly presented where it was shown that in all scenarios (single and multiple feature removal), both methods were able to predict values with greater accuracy and thus outperform simple median imputation. The results are meaningful because they validate the design and suitability of the framework for imputing missing values in laboratory data.

The results justify the selection of ML based and BNs for this project. In Section 2.3.11, a selection of empirical studies were presented which motivated the selection of ML based methods but there were fewer examples using BNs for the same purpose. In this project, it has been empirically shown that BNs can be used for data imputation as they can inherently deal with missing values (Section 7.2). While the results satisfy the objective, they can be improved even further if the performance limitations in the inference algorithm provided by `pgmpy` can be overcome. This would prevent the need to discretise the data which evidently impacted the performance of ML based methods (Section 7.4). Nevertheless, the experiments provide a robust set of methodologies which can be replicated in the future to investigate the performance improvement without discretisation.

While the experiments yielded positive results, they also had some limitations which should be considered. First, the project only focused on FBC which was found to have strong covariates that the imputation methods were able to exploit. It would require further prospective work to understand the generalisability of the methods to other laboratory panels. Furthermore, all patient test values were used meaning that the impact of any specific therapy or treatment given to patients was not considered. This may have caused outliers (Section 6.3.3) to be present in the data set. For purposes of this project, it was difficult to capture this information in static profiling so future work would benefit from either excluding results obtained one day after the patient was admitted or including treatment information while performing imputation.

Nevertheless, by carrying out a comprehensive empirical study, this project has consolidated results from investigating a breadth of different ML based methods and BNs. This is a meaningful contribution to research as previous empirical studies have focused on different combination of methods, which are mostly ML based, but not the combination of approaches presented in this project. Secondly, the results from empirical studies are usually not comparable as they are carried out on different data sets with different methodologies. In Section 7.4, the two approaches were tested with exactly the same methodology using the same data set to make the results more comparable.

8.1.4 Supporting code documentation

All implementations and experiments carried out in this project were regularly documented to the GitHub repository set up for this project. This was used extensively to facilitate weekly discussions with the project supervisor on the project progress and experiment results. The provided documentation adheres to the requirements set out in Section 3.4 and is hosted on the same GitHub repository containing the project code. The online documentation is deployed using GitHub Pages and uses the `Sphinx` library to render the documentation in HTML as per requirements. The documentation is organised by experiments where each experiment contains Python scripts with key results and code implementations showing how the implemented methods were used. As with good development practice, the code is clean, readable and well documented so that future collaborators can use, extend and maintain the code base as necessary.

8.2 Comparison of imputation methods

This section presents an evaluation of the strengths and limitations of the imputation methods based on the results of experiments II-IV presented in Chapter 7. Finally, a summary of the key properties of the method is provided for comparison.

8.2.1 Machine learning methods

The results of experiments II (Section 7.2) and IV (Section 7.4) highlighted the strengths of ML based methods. It was shown that ML based methods can impute values with a high degree of accuracy while preserving the underlying distribution of data without introducing bias for the best performing analytes. They can achieve this by capturing and exploiting the feature correlations to predict missing values using the values of features which are observed. This is a characteristic strength of simple models such as linear regression which proved to be the best model for analytes which were strong covariates. For the other analytes, MLP was able to utilise its dense architecture to identify and capture the non-linear combinations. The experiment methodology utilised these strengths well by selecting the best model for each analyte, which in combination with the iterative procedure of `IterativeImputer`, generated multiple imputation estimates. As such, ML based methods performed relatively well when most of the analytes were present.

On the other hand, the experiments also found limitations in the methods with increasing proportion of missing values. Naturally, ML based methods are designed to work with complete data sets and therefore the presence of missing values reduces their predictive capability as they have to rely on fewer features to make predictions with. This leads to the second limitation which is that model learning is completely autonomous and there is no feedback on what features have or have not been learned. As such, ML based methods are not easily interpretable which impacts their perceived trust and utility. They are also not easily extendable as addition of new information or features would require complete re-training to build a new model. Methods which have a large number of hyperparameters (such as MLP) also require a large amount of training data (and time) and need to be regularised to prevent over fitting.

8.2.2 Bayesian Network methods

The intrinsic nature of model learning and inference helps BNs address some limitations of ML based methods. A characteristic strength of BNs is that they are more interpretable as they provide an intuitive way to visualise dependencies between the features. In experiment III (Section 7.3), Figures 7.4 and 7.6 visualised the structure of BNs which offered plausible explanations for the performances of analytes. This also makes it easy to inspect and ensure that BNs are capturing all the domain specific information to make the inferences. In instances where relations are not captured (from the data), they can be manually specified leveraging the ability to combine domain expertise to enhance the utility of the models. As such, BNs are more extensible than ML based methods because new features and information can be easily added without needing to re-train the entire model. This does, however, require accurate prior knowledge which can be difficult to source and may be biased. Another advantage of BNs is that they can perform bi-directional reasoning - that is both predictive (where the child is predicted given evidence about the parent) and diagnostic (where parent is predicted given evidence about the child). This is not feasible in most ML based methods such as MLP which are uni-directional (feed forward) [105].

At the same time, there are limitations to modelling with BNs which should be considered. By design, BNs can only model directed acyclic relations between random variables to satisfy the representation of joint conditional probabilities. In scenarios where a cyclic relation exists, as was the case between HCT, HGB and RBC (Section 6.3.3), it presents shortcomings of this modelling technique. This is also related to the nature of the models; BNs capture feature dependencies and not correlations which may be limiting in cases where the variables do not show those properties. It is suitable for laboratory data as analytes under the same cell category would naturally have dependencies on other analytes. Another challenge, which was a limiting factor in experiments III (Section 7.3) and IV (Section 7.4) is the necessity to discretise data - the implications of this have been discussed in Section 8.1.3. As such, in comparison to ML based methods, BNs were not able to perform to their full potential and gave an acceptable performance overall. This motivates future work to explore continuous variable BNs.

8.2.3 Summary

A summary of the main properties of the two approaches investigated in this project is presented in Table 8.1. It provides a brief overview of their respective characteristic strengths and weaknesses which need to be considered when applying deploying these methods in a practical setting.

Property	Machine Learning	Bayesian Networks
Imputation type	Multiple	Single
Performance*	High	Medium
Interpretability	Low	High
Easily extendable	No	Yes
Combine domain expertise	No	Yes
Training data	High	Low
Training time	High	Low

Table 8.1: Table showing a summary of properties of ML based and BN imputation methods. Keys: * model performance is characterised by the improvement over simple median imputation as measured by the Δ metric (Section 7.1).

Chapter 9

Conclusions and Future Work

9.1 Conclusion

Missing laboratory data is a ubiquitous problem that can impact the fidelity of predictive modelling if the right data imputation techniques are not employed. This project presents the design, implementation and analysis of two data imputation techniques suitable for laboratory data: Machine Learning (ML) and Bayesian Networks (BN). Both methods were extensively evaluated on a real-life laboratory data set and shown to outperform the simple median imputation method with higher accuracy. By doing so, the project successfully delivers the four main deliverables outlined in Chapter 3.

A key contribution in this project is the design of an imputation framework which provides a robust methodology to impute missing values in laboratory data (Chapter 4). The framework is characterised by several advantages. It supports two workflows which enable the aforementioned imputation techniques to be evaluated and its modular design is its unique selling point. The framework can be extended for use cases that are more complex for example to impute values for multiple laboratory panels or modified to study a different subset of ML based and BN model learning methods that may be better suited to the use case. In this project, the flexibility of the framework was demonstrated with the inclusion of exploratory data analysis as discussed in Section 8.1.1. Ultimately, the suitability of the framework was validated by successfully carrying out the experiments outlined in Chapter 6.

Another important contribution in this project is the development of a Python library, denoted `labimputer`, to facilitate prospective evaluation of the designed imputation framework (Chapter 5). The library follows the modular design of the framework and provides implementations for ML based and BN imputation methods. The inclusion of the latter is a notable achievement in this project as Python wrappers were successfully implemented to interface with classes and methods provided by `pgmpy` (Section 5.6). A key characteristic of all implementations is that they are compatible with Scikit-learn and as such can be directly imported from `labimputer` for use with other classes and methods provided by Scikit-learn. The project code is open-source and available as part of the supplementary GitHub repository to enable future collaborators to modify and/or extend functionalities to enhance the performance of the current methods.

The project demonstrated the usability of the designed framework and implemented methods by carrying out an empirical study on a laboratory data set provided by Imperial College

NHS Healthcare Trust (Chapter 6). The performances of both the imputation methods were compared to simple median approach with single and multiple feature removal (10%, 30% and 50%) under the MAR assumption. In all scenarios, both methods consistently produced lower RMSE for all analytes than the median approach and were able to predict values with a higher accuracy. When the two methods were evaluated with the discretised data (Section 7.4), their performances were comparable and highlighted an area of future work to enhance the performance of BNs even more (Section 8.1.3).

The project provides a supporting online documentation which contains all project implementations and key experiment results. This can be used as a centralised reference point for the project but also as a tool for prospective collaborators on how the imputation methods were applied. It is a type of user guide on the methodology steps which were followed in this project so that they can be replicated in the future with other laboratory data sets to assess the generalisability of the imputation methods with different laboratory panels.

Based on the evaluations presented in Section 8.2, the recommendations from this project are twofold. If the intended purpose of imputation is to be used as a pre-processing step - that is, the missing values are imputed to generate a complete data set for further predictive modelling then ML based methods are recommended. This is on the premise that ML based methods offer high prediction accuracy, preserve the distribution of the original data after imputation and are not limited by the need to discretise data. The empirical results show strong evidence that these methods will enhance the accuracy of classifers if they were to be used in the probabilistic inference module of EPiC IMPOC.

On the other hand, if the intended purpose of imputation is to increase the availability of data to help clinicians better evaluate the severity of infections in patients then BNs offer a more appealing solution. The added interpretability makes them suitable in a clinical application as these models present relations between analytes in an intuitive way. At the same time, clinicians can enhance the quality of the models by combining their expertise to build hybrid models that are more likely to lead to better prediction performances. Finally, clinicians can build separate BNs to model separate diseases as this information can be easily encoded as a node in the network. This would make its application more meaningful and increase utility in the intended environment.

9.2 Future work

While this project met all of its objectives, there is scope for further work. The project provides a strong foundation from which other related projects and enhancements can be explored. This section provides a description of future work that can be undertaken.

9.2.1 Integration into EPiC IMPOC

The project was motivated based on some suggested improvements in the probabilistic inference module of EPiC IMPOC which currently uses simple median imputation [12]. A natural extension of this project, as suggested in Section 9.1, is to integrate the implemented ML based methods into the probabilistic inference module in EPiC IMPOC. It would be beneficial (prior to deployment) to carry out a further empirical study to understand the performance improvement that the imputation methods yield in the classifier predictions. The designed framework (Chapter 4) is suitable as a starting point and modifications can be made at stage five (imputation stage) to include evaluation metrics that are suitable for classification problems. The recommended starting model for this study would be MLP as it gave the best performance (for most analytes) in this project across all levels of missingness investigated. Ultimately, the study would aim to determine the best ML based data imputation techniques for CDSSs which is a broader topic that this project can contribute towards.

It would also be interesting to integrate BNs as part of EPiC IMPOC's graphical user interface to facilitate prospective evaluation by clinicians at point of care. This work can be done by adding REST APIs [106] to create endpoints that can be called so that the clinicians can interact with the provided tools. The current implementation plots static graphs (using the `networkx` package) which could be made dynamic using the `plotly` [107] library. This would enhance the utility of the networks as it would make it easier for clinicians to visualise the relations using interactive methods. Similarly, creating, removing or modifying nodes and/or relations in the network structure would allow the domain expertise to be combined with the BNs. This work should be carried out as a pilot study to formally evaluate the usefulness of BNs in clinical decision making.

9.2.2 Extending current experiments

As discussed in Section 9.1, the application of the designed framework to other data laboratory data sets is highly encouraged to evaluate its generalisability. The utility and modularity of the framework can be exploited to other studies perhaps where all the patients are diagnosed with a specific disease. For instance, the same set of experiments can be carried out on patients who had sepsis to see if the model accuracies are comparable with the current experiments. Certainly, it will be interesting to explore the different relations that BNs find between analytes to reflect the physiology of the patients suffering from a particular disease.

In this project, the missing values were simulated under the MAR assumption to facilitate investigation of multiple imputation techniques such as ML based methods and prefilling with the EM algorithm (for BNs). It also forms the basis of the problem investigated in this project - MAR assumes that there is sufficient information in the observed features to be able to predict the missing values. A useful future empirical study would be to carry out the same set of experiments under other missingness mechanisms such as MCAR and MNAR (Section 2.1). The former assumption is more likely to hold for laboratory data as values can be dropped from the data set completely at random for any arbitrary reasons. The latter assumption is unlikely with laboratory data as all patient information is usually recorded so the missingness in data will be related to some observed parameter.

Another worthwhile extension to the current set up would be to investigate the breaking limit of the imputation methods. In other words, increasing the proportion of missing values (under the same MAR assumption) to find the threshold at which the accuracy of the imputation method is no longer any better than the simple median approach. The experiment methodologies (Chapter 6) and provision of the Δ (%) metric (Section 7.1.1) should make this investigation feasible. This would help to evaluate the robustness of the methods with large proportion of missing values.

9.2.3 Temporal profiling

In Chapter 6, the experiment methodologies were presented which explained the motivation for using static profiling as a way to treat the results of a patient independently. However, laboratory data is longitudinal which can be exploited to understand how the evolution of disease relates to the evolution of biochemical marker values. This would capture the *contextual* information in the data such as the individual physiology of the patients. For example, a set of experiments could be carried out for both ML based and BN methods to study the evolution of a particular biochemical marker for patients that have results recorded for multiple days. Initial recommendations for these experiments include using *feature engineering* to deal with time-series data by using (a) aggregated values (minimum, maximum) and/or (b) fixed-size rolling window average to understand the most recent evolution in values. The temporal evolution can also be studied for BNs using *Dynamic Bayesian Networks* (DBN) which re-configures the network structure at each time step [108].

9.2.4 Enhancing model performances

In Chapter 7, experiment results for ML based and BN methods were presented which were shown to outperform the simple median approach. The impact of discretisation was also discussed and alternative methods are proposed as future work. Some recommendations are also proposed for ML based methods.

ML based methods: In this project, MLP models performed extremely well and as such a suitable extension would be to study application of deep learning models for this problem. Such models work well with high dimensional data and are able to capture complete representation of the data using their dense architectures [109]. The current project implementations should make this feasible - for example, the `Keras` [110] library can be used to implement deep MLP models which can be encapsulated in Scikit-learn wrappers to be compatible with the current implementations.

BNs: In Section 8.1.3, the limitations of discretising the data were presented. This was largely due to limitations in the *variable elimination* algorithm (Section 4.5) provided by `pgmpy` which uses arrays to compute the marginal probabilities. When the data set contains many variables (for example this project used 14), then the computation is memory expensive. Future work would study how these limitations can be overcome. It could be beneficial to modify the (large) source code to use different data structures for example `PyTorch` [111] tensors which can deal with large dimensional data. This would allow the discretisation to be more granular (possibly not even needed). The idea of *Gaussian Bayesian Networks* can also be explored as they build BNs using Gaussian methods by assuming each feature in the data set follows the distribution [112]. In Section 6.3.3, the analytes did not strictly follow a Gaussian distribution and hence this type of BNs were not explored.

Appendix A

Source code and documentation

Source code: The implementations and code for this project is open source and available on GitHub at: `https://github.com/agrimmanchanda/fyp2020-am9717`.

Brief instructions for use:

1. The project can be cloned from GitHub to local machine using command:
   `git clone https://github.com/agrimmanchanda/fyp2020-am9717.git`
2. Ensure that Python 3.x and `pip` are installed on local machine.
3. Navigate to the root folder of the repository and install all relevant packages using command:
   `pip install -r requirements.txt`.
4. Access all project code by navigating to `labimputer` folder for imputation methods and `examples` folder for experiments.

Documentation: Likewise, the supporting online documentation can be accessed at: `https://agrimmanchanda.github.io/fyp2020-am9717/`.

The documentation includes a `tutorial` section which was provided for this project and can be accessed at: `https://agrimmanchanda.github.io/fyp2020-am9717/tutorials/setup.html`

Appendix B

Supplementary experiment results

B.1 Experiment I: Description of FBC biochemical markers

Analyte	p-value	Analyte	p-value
EOS	$p < 0.05$	MONO	$p < 0.05$
HCT	$p < 0.05$	MPV	$p < 0.05$
HGB	$p < 0.05$	NEUT	$p < 0.05$
LY	$p < 0.05$	PLT	$p < 0.05$
MCH	$p < 0.05$	RBC	$p < 0.05$
MCHC	$p < 0.05$	RDW	$p < 0.05$
MCV	$p < 0.05$	WBC	$p < 0.05$

Table B.1: Table showing results of JB-test carried out on 14 analytes. Keys: $p < 0.05$ represents the significance level of the test.

Analyte and description	Reference range		Units
	Male	Female	
Basophils (BASO) are a type of white blood cell which release granules of enzymes (histamin and heparin) against pathogens.	0.02 - 0.10	0.02 - 0.10	10^9/L
Eosinophils (EOS) like BASO, are also enzyme secreting white blood cells that fight parasitic infections and boost inflammation.	0.10 - 0.40	0.10 - 0.40	10^9/L
Haematocrit (HCT) quantifies the amount of red blood cells in the blood.	0.40 - 0.54	0.37 - 0.47	L/L
Haemoglobin (HGB) is a protein inside the red blood cell that helps it to maintain its biconcave shape and carry oxygen around the body.	130 - 180	115 - 165	g/L
Lymphocytes (LY) are a type of white blood cell that come in mainly two categories T and B cells.	1.00 - 4.00	1.00 - 4.00	10^9/L
Mean Cell Haemoglobin (MCH) measures the average amount of haemoglobin present in red blood cells.	27 - 32	27 - 32	pg
Mean Corpuscular Haemoglobin Concentration (MCHC) measures the average concentration of haemoglobin present in a *single* red blood cell.	320 - 360	320 - 360	g/L
Mean Corpuscular Volume (MCV) measures the average size of red blood cells.	80 - 100	80 - 100	fL
Monocytes (MONO) are a type of white blood cell that fight pathogenic infection and facilitate healing and repair.	0.20 - 0.80	0.20 - 0.80	10^9/L
Mean Platelet Volume (MPV) measures the average volume of platelets in the blood.	7.50 - 12.00	7.50 - 12.00	fL
Neutrophils (NEUT) are a type of white blood cell that fight pathogenic infection and facilitate healing of damaged tissues.	1.80 - 7.50	1.80 - 7.50	10^9/L
Nucleated RBC (NRBCA) measures the amount of nucleated (immature) red blood cells in the blood.	0.00	0.00	10^9/L
Platelets (PLT) are blood cells that used for clotting for example a wound.	140 - 400	140 - 400	10^9/L
Red Blood Cells (RBC) are cells which carry and transport oxygen around the body.	4.50 - 6.50	3.80 - 5.80	10^{12}/L
RBC Distribution Width (RDW) measures the range in the volume and size of the red blood cells.	11.50 - 15.00	11.50 - 15.00	%
White Blood Cells (WBC) are the core of the body's immune response system used to fight pathogenic infections.	3.60 - 11.00	3.60 - 11.00	10^9/L

Table B.2: Table showing a description of all FBC biochemical markers. All reference ranges are provided for adults. **Keys**: *L/L* = litre of cells per litre of blood; *pg* = picograms; *fL* = femtolitre. All information sourced from [113].

B.2 Experiment II: Imputation using ML based methods

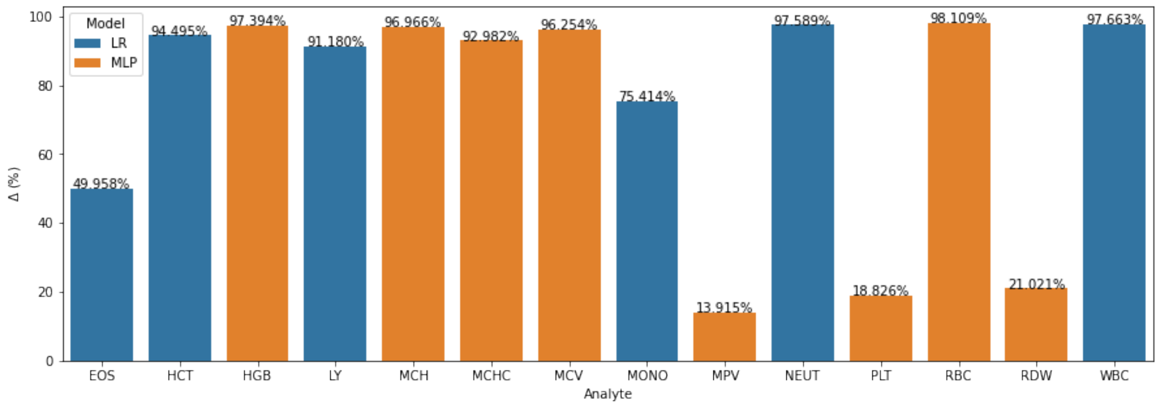


Figure B.1: Diagram showing the Δ (%) improvement of best model over median imputation on the held out test set (HOTS) for each analyte. The bar graph is colour coded by the best model: linear regression (LR) in blue and multi layer perceptron (MLP) in orange.

Analyte	Unit	LR	DT	RF	SVR	K-NN	MLP	XGB	Median
EOS	10^9/L	**0.057**	0.109	0.106	0.058	0.113	0.058	0.111	0.117
HCT	L/L	**0.004**	0.013	0.006	0.004	0.013	0.004	0.017	0.065
HGB	g/L	1.040	3.998	1.989	1.041	4.240	**0.643**	13.267	21.187
LY	10^9/L	**0.066**	0.579	0.520	0.066	0.574	0.068	0.331	0.759
MCH	pg	0.077	0.650	0.235	0.077	0.571	**0.063**	3.179	2.120
MCHC	g/L	0.813	6.130	3.798	0.813	6.986	**0.702**	35.032	9.729
MCV	fL	0.235	1.918	0.865	0.235	1.921	**0.221**	9.889	6.017
MONO	10^9/L	**0.064**	0.202	0.186	0.064	0.214	0.065	0.158	0.261
MPV	fL	1.018	1.042	1.017	1.018	1.075	**1.012**	1.311	1.142
NEUT	10^9/L	**0.066**	0.658	0.365	0.066	0.772	0.076	0.570	2.589
PLT	10^9/L	68.769	71.614	68.440	68.762	71.806	**67.418**	72.196	81.135
RBC	$10^1 2$/L	0.052	0.206	0.057	0.052	0.144	**0.018**	0.396	0.743
RDW	%	1.489	1.531	1.459	1.489	1.525	**1.439**	2.012	1.917
WBC	10^9/L	**0.066**	0.723	0.301	0.066	0.707	0.074	0.789	2.766
Average	-	5.273	6.384	5.667	5.272	6.476	**5.133**	9.947	9.325

Table B.3: Table showing the RMSE for five fold cross validation training set (CVTS) for all eight regression models tested. The lowest CVTS score for each analyte is highlighted in bold. The final row presents the average RMSE for each regression model. **Keys:** *LR = linear regression, DT = decision tree, RF = random forest, SVR = support vector, K-NN = k-nearest neighbours, XGB = XGBoost and MLP = multi-layer perceptron.*

Analyte	Unit	Missing (%)	LR	DT	RF	SVR	K-NN	MLP	XGB	Median
		10	0.109	0.109	0.108	0.109	0.115	**0.087**	0.114	0.117
EOS	10^9/L	30	0.110	0.111	0.110	0.110	0.116	**0.109**	0.116	0.117
		50	0.112	0.111	**0.111**	0.112	0.118	0.111	0.118	0.117
		10	0.016	0.019	0.015	0.017	0.019	**0.008**	0.019	0.065
HCT	L/L	30	0.028	0.028	0.026	0.028	0.029	**0.020**	0.027	0.065
		50	0.037	0.037	0.036	0.037	0.038	**0.032**	0.036	0.065
		10	5.135	6.409	5.131	5.136	6.225	**2.610**	13.612	21.187
HGB	g/L	30	9.073	9.154	8.571	9.073	9.272	**6.401**	14.924	21.187
		50	12.162	12.194	11.975	12.162	12.362	**10.126**	16.909	21.187
		10	0.566	0.629	0.605	0.566	0.603	**0.336**	0.508	0.759
LY	10^9/L	30	0.631	0.661	0.637	0.631	0.645	**0.505**	0.597	0.759
		50	0.662	0.684	0.668	0.662	0.680	**0.599**	0.649	0.759
		10	0.649	0.966	0.794	0.649	0.859	**0.448**	3.243	2.120
MCH	pg	30	1.105	1.304	1.216	1.105	1.238	**0.823**	3.354	2.120
		50	1.433	1.569	1.527	1.433	1.550	**1.263**	3.475	2.120
		10	6.798	7.462	7.068	6.798	7.827	**3.883**	35.356	9.729
MCHC	g/L	30	8.470	8.389	8.173	8.471	8.736	**6.845**	35.726	9.729
		50	9.075	9.060	8.895	9.073	9.419	**8.659**	35.964	9.729
		10	2.089	2.871	2.501	2.090	2.762	**1.297**	10.086	6.017
MCV	fL	30	3.487	3.886	3.694	3.486	3.840	**2.605**	10.393	6.017
		50	4.345	4.556	4.450	4.345	4.655	**3.889**	10.712	6.017
		10	0.207	0.208	0.201	0.207	0.219	**0.141**	0.192	0.261
MONO	10^9/L	30	0.215	0.215	0.211	0.216	0.226	**0.194**	0.205	0.261
		50	0.223	0.222	0.220	0.223	0.235	0.217	**0.216**	0.261
		10	1.034	1.057	1.038	1.034	1.097	**1.028**	1.319	1.142
MPV	fL	30	1.060	1.073	1.065	1.060	1.124	**1.054**	1.341	1.142
		50	1.086	1.096	1.089	1.086	1.159	**1.085**	1.362	1.142
		10	0.881	1.085	0.986	0.881	1.148	**0.784**	0.982	2.589
NEUT	10^9/L	30	1.449	1.526	1.474	1.449	1.611	**1.322**	1.432	2.589
		50	1.799	1.824	1.808	1.799	1.954	**1.713**	1.790	2.589
		10	70.002	73.107	70.735	70.025	73.636	**68.643**	73.514	81.135
PLT	10^9/L	30	72.257	74.721	73.084	72.263	76.081	**70.969**	75.756	81.135
		50	74.593	76.824	75.539	74.599	79.003	**73.578**	78.157	81.135
		10	0.180	0.285	0.190	0.181	0.212	**0.085**	0.409	0.743
RBC	10^{12}/L	30	0.316	0.365	0.313	0.316	0.320	**0.220**	0.465	0.743
		50	0.424	0.452	0.433	0.424	0.432	**0.354**	0.545	0.743
		10	1.520	1.561	1.510	1.519	1.553	**1.450**	2.022	1.917
RDW	%	30	1.570	1.603	1.565	1.570	1.602	**1.499**	2.051	1.917
		50	1.628	1.658	1.632	1.628	1.666	**1.566**	2.099	1.917
		10	0.841	1.160	0.977	0.841	1.123	**0.778**	1.106	2.766
WBC	10^9/L	30	1.441	1.584	1.483	1.441	1.645	**1.356**	1.557	2.766
		50	1.849	1.924	1.882	1.849	2.035	**1.781**	1.934	2.766

Table B.4: Table showing the RMSE for five fold cross validation training set (CVTS) for all eight regression models for three proportions of missingness: 10%, 30% and 50%. The lowest CVTS score for each analyte is highlighted in bold. **Keys:** *LR = linear regression, DT = decision tree, RF = random forest, SVR = support vector, K-NN = k-nearest neighbours, XGB = XGBoost and MLP = multi-layer perceptron.*

Bibliography

[1] Global diffusion of eHealth: making universal health coverage achievable. Report of the third global survey on eHealth. Geneva: World Health Organization., 12 2016. Licence: CC BY-NC-SA 3.0 IGO. pages 6, 22

[2] Marcel Lucadou, Thomas Ganslandt, Hans-Ulrich Prokosch, and Dennis Toddenroth. Feasibility analysis of conducting observational studies with the electronic health record. *BMC Medical Informatics and Decision Making*, 19, 10 2019. pages 6

[3] Bireswar Dutta and Hwang Hsin-Ginn. The adoption of electronic medical record by physicians: A prisma-compliant systematic review. *Medicine (Baltimore)*, 99(8), 02 2020. pages 6

[4] Sabysachi Dash, Sushil Kumar Shakyawar, Mohit Sharma, and Sandeep Kaushik. Big data in healthcare: management, analysis and future prospects. *Journal of Big Data*, 6(54), 6 2019. pages 6, 7

[5] Scheepers-Hoeks A.M.J.W. Wasylewicz A.T.M. *Fundamentals of Clinical Data Science [Internet].*, volume 1 of *1*. Cham (CH): Springer, 1 edition, 2 2019. pages 6

[6] T. Frisell. Sp0187 why missing data is a problem, and what you shouldn't do to solve it. *Annals of the Rheumatic Diseases*, 75(Suppl 2):45–45, 2016. pages 6

[7] Zhen Hu, Genevieve B. Melton, Elliot G. Arsoniadis, Yan Wang, Mary R. Kwaan, and Gyorgy J. Simon. Strategies for handling missing clinical data for automated surgical site infection detection from the electronic health record. *Journal of Biomedical Informatics*, 68:112 – 120, 2017. pages 6, 10

[8] Seyedeh Neelufar Payrovnaziri, Aiwen Xing, Salman Shaeke, Xiuwen Liu, Jiang Bian, and Zhe He. Assessing the impact of imputation on the interpretations of prediction models: A case study on mortality prediction for patients with acute myocardial infarction. 2020. pages 6, 11

[9] Quinten A. W. Raaijmakers. Effectiveness of different missing data treatments in surveys with likert-type data: Introducing the relative mean substitution approach. *Educational and Psychological Measurement*, 59(5):725–748, 1999. pages 6

[10] Salgado CM, Azevedo C, Proenca H, Vieira SM: MIT Critical Data. *Secondary Analysis of Electronic Health Records [Internet]*, volume 1 of *1*. 1 edition, 2016. Cham: Springer International Publishing. pages 6, 10, 11

[11] Muhammad Shahid, Muhammad Idrees, Bilal Nasir, Arsalan Raja, Syed Mohsin Raza, Iram Amin, Afza Rasul, and Ghias Un Nabi Tayyab. Correlation of biochemical markers and hcv rna titers with fibrosis stages and grades in chronic hcv-3a patients. *European journal of gastroenterology hepatology*, 26, 04 2014. pages 6

[12] Bernard Hernandez Perez. *Data-driven web-based intelligent decision support system for infection management at point of care*. PhD thesis, Imperial College London, 2 2019. pages 7, 15, 25, 92

[13] David S Watson, Jenny Krutzinna, Ian N Bruce, Christopher EM Griffiths, Iain B McInnes, Michael R Barnes, and Luciano Floridi. Clinical applications of machine learning algorithms: beyond the black box. *BMJ*, 364, 2019. pages 7

[14] Christina Mack, Zhaohui Su, and Daniel Westreich. *Managing Missing Data in Patient Registries: Addendum to Registries for Evaluating Patient Outcomes: A User's Guide, Third Edition [Internet].*, volume 1 of *1*. AHRQ Publication, 1 edition, 2 2018. Rockville (MD): Agency for Healthcare Research and Quality (US). pages 9, 10

[15] Jim Dziura, Lori Post, Qing (Amanda) Zhao, Zhixuan Fu, and Peter Peduzzi. Strategies for dealing with missing data in clinical trials: From design to analysis. *The Yale journal of biology and medicine*, 86:343–358, 09 2013. pages 10

[16] Cheng Li. Little's test of missing completely at random. *The Stata Journal*, 13(4):795–809, 2013. pages 10

[17] Qinbao Song and Martin Shepperd. Missing data imputation techniques. *IJBIDM*, 2:261–291, 10 2007. pages 10, 11

[18] Robert J. Mislevy. *Journal of Educational Statistics*, 16(2):150–155, 1991. pages 10

[19] Paul Lodder. *To Impute or not Impute: That's the Question*. 01 2014. pages 11, 12

[20] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977. pages 11

[21] T. K. Moon. The expectation-maximization algorithm. *IEEE Signal Processing Magazine*, 13(6):47–60, 1996. pages 11, 12, 34

[22] Antoine Cully, Marek Rei, and Josiah Wang. Lecture notes in introduction to machine learning, December 2020. pages 11, 13, 14, 17, 18

[23] Donald B. Rubin. Multiple imputation after 18+ years. *Journal of the American Statistical Association*, 91(434):473–489, 1996. pages 12

[24] Ines Rombach, Alastair Gray, Crispin Jenkinson, David Murray, and Oliver Rivero-Arias. Multiple imputation for patient reported outcome measures in randomised controlled trials: Advantages and disadvantages of imputing at the item, subscale or composite score level. *BMC Medical Research Methodology*, 18, 08 2018. pages 12

[25] Mohsen Tavakol and Reg Dennick. Making Sense of Cronbach's Alpha. *International Journal of Medical Education*, 2:53–55, 06 2011. pages 12

[26] Akbar K Waljee, Ashin Mukherjee, Amit G Singal, Yiwei Zhang, Jeffrey Warren, Ulysses Balis, Jorge Marrero, Ji Zhu, and Peter DR Higgins. Comparison of imputation methods for missing laboratory data in medicine. *BMJ Open*, 3(8), 2013. pages 12

[27] Daniel J. Stekhoven and Peter Buehlmann. Missforest - non-parametric missing value imputation for mixed-type data. *Bioinformatics*, 28(1):112–118, 2012. pages 13

[28] Fei Tang and Hemant Ishwaran. Random forest missing data algorithms. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 10(6):363–377, 2017. pages 13

[29] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. pages 13

[30] Yuan Luo, Peter Szolovits, Anand Dighe, and Jason Baron. Using Machine Learning to Predict Laboratory Test Results. *American journal of clinical pathology*, 145, 06 2016. pages 13, 20

[31] Rachel Draelos. Best Use of Train/Val/Test Splits, with Tips for Medical Data. *Machine Learning and Medicine*, 2019. pages 13, 14, 59

[32] Sebastian Raschka. Model evaluation, model selection, and algorithm selection in machine learning. *ArXiv*, abs/1811.12808, 2018. pages 14

[33] F. Pedregosa, G. Varoquaux, Michel V. Gramfort, A., et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. pages 14, 15, 19, 26, 28, 40, 41, 42, 44, 50, 61

[34] Christoph Molnar. *Interpretable Machine Learning*. 2019. `https://christophm.github.io/interpretable-ml-book/`. pages 15

[35] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001. pages 15

[36] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. pages 16

[37] Hussain Mujtaba. What is Gradient Boosting and how is it different from AdaBoost? `https://www.mygreatlearning.com/blog/gradient-boosting/`, 6 2020. pages 17

[38] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM. pages 17, 45

[39] Amol Mavuduru. Why XGBoost can't solve all your problems. *Towards Data Science*, 11 2020. pages 17

[40] Mitali S. Mhatre, Dr. Fauzia Siddiqui, Mugdha Dongre, and P. Thakur. A Review paper on Artificial Neural Network: A Prediction Technique. 2015. pages 17, 18

[41] Maad Mijwil. Artificial neural networks advantages and disadvantages. 01 2018. pages 18

[42] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85 – 117, 2015. pages 18

[43] S.B. Imandoust and Mohammad Bolandraftar. Application of K-nearest neighbor (KNN) approach for predicting economic events theoretical background. *Int J Eng Res Appl*, 3:605–610, 01 2013. pages 18

[44] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf. Support vector machines. *IEEE Intelligent Systems and their Applications*, 13(4):18–28, 1998. pages 18, 19, 42

[45] Feiping Nie, Hu Zhanxuan, and Xuelong Li. An investigation for loss functions widely used in machine learning. *Communications in Information and Systems*, 18:37–52, 01 2018. pages 19

[46] Weijie Wang and Yanmin Lu. Analysis of the Mean Absolute Error (MAE) and the Root Mean Square Error (RMSE) in Assessing Rounding Model. *IOP Conference Series: Materials Science and Engineering*, 324:012049, 03 2018. pages 19

[47] Sharoon Saxena. What's the Difference Between RMSE and RMSLE? *Medium.com*, 2019. pages 20

[48] Dalson Figueiredo, Silva Júnior, and Enivaldo Rocha. What is r2 all about? *Leviathan-Cadernos de Pesquisa Polútica*, 3:60–68, 11 2011. pages 20

[49] Stef van Buuren and Karin Groothuis-Oudshoorn. mice: Multivariate imputation by chained equations in r. *Journal of Statistical Software*, 45(3):1–67, 2011. pages 20, 41

[50] D. Bertsimas, C. Pawlowski, and Y.D. Zhuo. From predictive methods to missing data imputation: An optimization approach. *Journal of Machine Learning Research*, 18:1–39, 04 2018. pages 20

[51] Gustavo Batista and Maria-Carolina Monard. An Analysis of Four Missing Data Treatment Methods for Supervised Learning. *Applied Artificial Intelligence*, 17:519–533, 05 2003. pages 21

[52] Honghai Feng, Chen Guoshun, Yin Cheng, Bingru Yang, and Yumei Chen. A svm regression based approach to filling in missing values. volume 3683, pages 581–587, 09 2005. pages 21

[53] P. K. Sharpe and R. Solly. Dealing with missing values in neural network-based diagnostic systems. *Neural Computing Applications*, 3:73–77, 2005. pages 21

[54] Marek Smieja, Łukasz Struski, Jacek Tabor, Bartosz Zieliński, and Przemysław Spurek. Processing of missing data by neural networks, 2019. pages 21

[55] Claude Sammut and Geoffrey I. Webb, editors. *Bayesian Network*, pages 81–81. Springer US, Boston, MA, 2010. pages 21

[56] Khalid Iqbal, Yin xu, Hong-Wei Hao, Qazi Ilyas, and Hazrat Ali. An overview of bayesian network applications in uncertain domains. *International Journal of Computer Theory and Engineering*, 7:416–427, 12 2015. pages 21

[57] Branislav Holländer. Introduction to Probabilistic Graphical Models. *Towards Data Science*, 2 2020. pages 21, 24

[58] Stefano Ermon. Lecture notes in probabilistic graphical models, December 2017. pages 21

[59] Luis M. de Campos. A Scoring Function for Learning Bayesian Networks based on Mutual Information and Conditional Independence Tests. *Journal of Machine Learning Research*, 7(77):2149–2187, 2006. pages 22

[60] Xiaowei Huang. Lecture notes in advanced artificial intelligence, January 2021. pages 23

[61] Md. Faizul Bari. Bayesian Network Structure Learning. 01 2011. pages 23

[62] Dimitris Margaritis. *Learning Bayesian Network Model Structure from Data*. PhD thesis, Carnegie Mellon University, 5 2003. pages 23

[63] Thuc le, Tao Hoang, Jiuyong Li, Lin Liu, and Huawen Liu. A Fast PC Algorithm for High Dimensional Causal Discovery with Multi-Core PCs. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 16, 02 2015. pages 23

[64] Ioannis Tsamardinos, Laura Brown, and Constantin Aliferis. The Max-Min Hill-Climbing Bayesian Network Structure Learning Algorithm. *Machine Learning*, 65:31–78, 10 2006. pages 23

[65] Sonu Jose. *Bayesian Network Structure Learning and its Application on Simulation-based Learning*. PhD thesis, Carnegie Mellon University, 5 2003. pages 23

[66] Anna Goldenberg. Lecture notes in Machine Learning, November 2005. pages 23, 24

[67] Guy Broeck, Karthika Mohan, Arthur Choi, and Judea Pearl. Efficient Algorithms for Bayesian Network Parameter Learning from Incomplete Data. 11 2014. pages 24

[68] Tomás Lozano-Pérez and Leslie Kaelbling. Lecture notes in techniques in artificial intelligence, December 2002. pages 24

[69] Sargur Srihari. Lecture notes in Probabilistic Graphical Models, March 2019. pages 24, 37

[70] Zeyneb Guenfoud and Péter Antal. Bayesian exploration of dependencies of laboratory tests and evaluation of test redundancy. pages 1–6, 10 2018. pages 24

[71] Ying Shen, Lizhu Zhang, Jin Zhang, Min Yang, Buzhou Tang, Yaliang Li, and Kai Lei. CBN: Constructing a Clinical Bayesian Network based on Data from the Electronic Medical Record. *Journal of Biomedical Informatics*, 88, 11 2018. pages 25

[72] TIOBE Software BV. Tiobe Index for January 2021. Accessed from: www.tiobe.com. pages 25

[73] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009. pages 25

[74] Jakub Protasiewicz. Python AI: Why Is Python So Good for Machine Learning? *Netguru*, 8 2018. pages 25

[75] Holger Krekel, Bruno Oliveira, Ronny Pfannschmidt, Floris Bruynooghe, Brianna Laugher, and Florian Bruhin. pytest x.y, 2004. pages 25, 39

[76] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. pages 25

[77] The pandas development team. pandas-dev/pandas: Pandas, February 2020. pages 25

[78] P Asirinaidu. Pandas for Data Analysis and their Benefits. *Medium.com*, 6 2017. pages 25

[79] Charles R. Harris, K. Jarrod Millman, and St'efan J. van der Walt et al. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. pages 26

[80] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. pages 26

[81] Ankur Ankan and Abinash Panda. pgmpy: Probabilistic graphical models using python. In *Proceedings of the 14th Python in Science Conference (SCIPY 2015)*. Citeseer, 2015. pages 26, 29, 36

[82] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring Network Structure, Dynamics, and Function using NetworkX. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008. pages 26

[83] Xinmeng Zhang, Chao Yan, Cheng Gao, Bradley Malin, and You Chen. Predicting Missing Values in Medical Data Via XGBoost Regression. *Journal of Healthcare Informatics Research*, 4, 12 2020. pages 33, 44, 68

[84] Alireza Farhangfar, Lukasz A. Kurgan, and Witold Pedrycz. A Novel Framework for Imputation of Missing Values in Databases. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 37(5):692–709, 2007. pages 33

[85] Peter Rousseeuw. Tutorial to Robust Statistics. *Journal of Chemometrics*, 5:1 – 20, 01 1991. pages 33

[86] Serge Romaric Tembo Mouafo, Sandrine Vaton, Jean-Luc Courant, and Stephane Gosselin. A tutorial on the EM algorithm for Bayesian networks: application to self-diagnosis of GPON-FTTH networks. In *IWCMC 2016 : 12th International Wireless Communications*
Mobile Computing Conference, pages 369 – 376, Paphos, Cyprus, September 2016. pages 34

[87] Andrea Ruggieri, Francesco Stranieri, Fabio Stella, and Marco Scutari. Hard and Soft EM in Bayesian Network Learning from Incomplete Data, 2020. pages 34

[88] Sotiris Kotsiantis, Dimitris Kanellopoulos, and P. Pintelas. Data Preprocessing for Supervised Learning. *International Journal of Computer Science*, 1:111–117, 01 2006. pages 34

[89] Stefano Beretta, Mauro Castelli, Ivo Gonçalves, Roberto Henriques, and Daniele Ramazzotti. Learning the Structure of Bayesian Networks: A Quantitative Assessment of the Effect of Different Algorithmic Schemes. *Complexity*, 04 2017. pages 36

[90] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 1 edition. pages 36

[91] Jason Brownless. Probabilistic Model Selection with AIC, BIC, and MDL, 2020. pages 36

[92] David Heckerman, Dan Geiger, and David Chickering. Learning Bayesian Networks: The Combination of Knowledge and Statistical Data. *Machine Learning*, 20:197–243, 09 1995. pages 36

[93] Giorgos Myrianthous. fit() vs predict() vs fit_predict() in Python scikit-learn. *Towards Data Science*, 3 2021. pages 43

[94] Temitope Fisayo and Sonia Tsukagoshi. Three waves of the COVID-19 pandemic. *Postgraduate Medical Journal*, 97(1147):332–332, 2021. pages 49

[95] Mike Leach. Interpretation of the full blood count in systemic disease–a guide for the physician. author reply. *The journal of the Royal College of Physicians of Edinburgh*, 44:36–41, 03 2014. pages 52

[96] Timotius Hariyanto and Andree Kurniawan. Anemia is associated with severe coronavirus disease 2019 (COVID-19) infection. *Transfusion and Apheresis Science*, 59, 08 2020. pages 53

[97] Timotius Hariyanto and Andree Kurniawan. The Impact of COVID-19 Disease on Platelets and Coagulation. *Pathobiology*, 88:15–27, 01 2021. pages 53

[98] Carlos M. Jarque and Anil K. Bera. Efficient tests for normality, homoscedasticity and serial independence of regression residuals. *Economics Letters*, 6(3):255–259, 1980. pages 56

[99] *Central Limit Theorem*, pages 66–68. Springer New York, New York, NY, 2008. pages 56

[100] Henny Billett. *Clinical Methods: The History, Physical, and Laboratory Examinations*. Boston: Butterworths, 3 edition. pages 56

[101] Cong Ma, Xiaoyan Wang, and Rui Zhao. Associations of lymphocyte percentage and red blood cell distribution width with risk of lung cancer. *The Journal of international medical research*, 47:300060519850417, 06 2019. pages 57

[102] P. Ravi Sarma. *Clinical Methods: The History, Physical, and Laboratory Examinations.*, volume 1 of *1*. Boston Butterworths, 3 edition, 1990. pages 57

[103] Luís Torgo and João Gama. Regression by classification. In Díbio L. Borges and Celso A. A. Kaestner, editors, *Advances in Artificial Intelligence*, pages 51–60, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg. pages 61

[104] Markus Neuhäuser. *Wilcoxon–Mann–Whitney Test*. pages 64

[105] Scott McCloskey. Probabilistic Reasoning and Bayesian Networks. ICSG 755 - Neural Networks and Machine Learning, 1999. pages 90

[106] Leonard Richardson, Mike Amundsen, and Sam Ruby. *RESTful Web APIs*. O'Reilly Media, Inc., 2013. pages 93

[107] Plotly Technologies Inc. Collaborative data science, 2015. pages 93

[108] Zoubin Ghahramani. Learning Dynamic Bayesian Networks, 1997. pages 94

[109] Chung-Yuan Cheng, Wan-Ling Tseng, Ching-Fen Chang, Chuan-Hsiung Chang, and Susan Shur-Fen Gau. A deep learning approach for missing data imputation of rating scales assessing attention-deficit hyperactivity disorder. *Frontiers in Psychiatry*, 11:673, 2020. pages 94

[110] François Chollet et al. Keras. `https://keras.io`, 2015. pages 94

[111] Adam Paszke, Sam Gross, Francisco Massa, et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. pages 94

[112] Asish Ghoshal and Jean Honorio. Learning Identifiable Gaussian Bayesian Networks in Polynomial Time and Sample Complexity. 03 2017. pages 94

[113] myDr. Full blood count (FBC), 6 2017. pages 97