Modelling Heart Conditions and Train Delays Using Machine Learning Methods

MATH4069 Statistical Machine Learning

Group Project Report

2023/24

School of Mathematical Sciences

University of Nottingham

Group D:

Alastair Harrison

Timi Folaranmi

Ying Zhan

Weiyun Wu

Contents

Abstract

This report presents two distinct studies, utilising machine learning techniques to address disparate challenges: predicting heart conditions from electrocardiogram (ECG) readings and modelling train delays for trains arriving at Nottingham station. Both challenges are effectively tackled using decision trees and random forests. As such, insights into regression and classification problems using Random Forests are obtained.

1 Introduction

This report contains two different studies with the aim to use machine learning methods to model first heart conditions based on ECG readings and second train delay for trains arriving into Nottingham station on the Leeds to Nottingham line. Despite the vastly different nature and applications of these two challenges both lend themselves well to decision trees and thus random forests. In this report two different approaches to manipulating data, extracting features and developing random forest models will be highlighted. The differences in these challenges allow for random forests to be examined from both a regression and a classification point of view. The two studies should allow the reader to see the similarities (for example similar parameter tuning methods) and differences (for example different metrics used for measuring success) for approaching a regression and classification problem while using random forests.

Section 2 of this report contains the study on ECGs and modelling heart conditions. This study first introduces what an ECG is and the aims of this challenge in section 2.1, followed by the extensive data manipulation and feature extraction required for this challenge in section 2.2. Next model development is discussed in section 2.3 before discussion of the results from these models in section 2.4. Finally in section 2.5 the conclusions from this study are given.

Section 3 contains the study for modelling train delay on the Leeds to Nottingham line based on historical data. First in section 3.1 background context and the aims of this study are outlined. Next data manipulation and feature extraction are shown in section 3.2, followed by the ideas and processes surrounding model development, highlighted in section 3.3. Results from the models developed for this challenge can then be seen in section 3.4 before finally conclusions for the trains study in section 3.5.

At the end of this report general conclusions can be found in section 4 before the Appendix and references. The Appendix contains sections of code for both challenges and some further EDA and derivations for the trains challenge.

2 Diagnosing Heart Conditions from ECGs

2.1 Introduction

2.1.1 Background

Various heart conditions are common and potentially life-threatening. They are some of the leading causes of death worldwide. Two common heart conditions are Myocardial infarction (MI) and Cardiomyopathy. MI is essentially the medical term for a heart attack while Cardiomyopathy is a general term for diseases of the heart muscle. Both of these conditions have characteristics such as high morbidity, disability, recurrence and mortality. Reducing the socio - economic burden caused by these conditions has thus become an important global public health priority. Electrocardiogram (ECG) is a basic basis for determining the condition of someone's heart. Willem Einthoven is credited with inventing the ECG in the early 1900s. An ECG looks at a persons heart's rate, rhythm and electrical activity. If you are having a heart attack, have symptoms of coronary heart disease, have a history of previous heart disease or are taking certain medications, you are likely to need an ECG. So it is necessary to detect and predict heart diseases and heart attacks in time through ECG, which will help healthcare professionals to diagnose the patients and reduce the number of deaths caused by these heart conditions.

2.1.2 Aims

The aim for this section of the report is to develop a machine learning model that can be used to diagnose heart conditions for patients based solely on their ECG graph. The model should take as input a vector of length $30,000$ that when plotted shows the ECG of the patient and output a classification of the condition of the patients heart. Either healthy ($y = 0$), myocardial infarction ($y = 1$) and cardiomyopathy ($y = 2$). The method should aim create a function $\hat{y} = f(x)$ that maximises the number of patients classified correctly, i.e. maximises

$$\sum_{i=1}^{n_{test}} I(\hat{y} = y) \tag{2.1}$$

where $I(x)$ is the indicator function such that,

$$I(x) = \begin{cases} 1 & \text{if } x \text{ is true.} \\ 0 & \text{if } x \text{ is false.} \end{cases}$$

To achieve this aim the first step will be to process the data into a more useful format. This is explored in the next section.

2.2 Data Preprocessing: ECGs

2.2.1 Data Provided

The data provided for this challenge consists of a training set and a test set. The training set contains ECGs for 115 patients along with the diagnosed heart condition for each patient. There are three given heart conditions, healthy (coded by $y = 0$ in the data set), myocardial infarction ($y = 1$) and cardiomyopathy ($y = 2$). In the training data 26 of the patients are healthy, 80 have myocardial infarction and 9 have cardiomyopathy. The ECGs themselves are given by a vector of length $30,000$ which when plotted shows the ECG time series for the patient. A plot for patient 1 from the training set can be seen in Figure 1.

The test data set contains the ECGs for 100 patients but does not contain the response variable (y) corresponding to the condition of the patients heart. This is what is to be modelled in this report.

2.2.2 Challenges with the data

There are a few key issuse that become clear when looking at the data sets. These issues are: (i) the small number of patients in the training set increases the risk of over fitting and renders the use of cross validation more challenging. (ii) The data itself is not in a convenient form for integration into a machine learning method, implying data manipulation is required to allow for feature extraction.

2.2.3 Reading ECGs

Literature online provides many suggestions as to what features can be used to diagnose conditions based on ECGs. As feature extraction is a reasonably convoluted process for

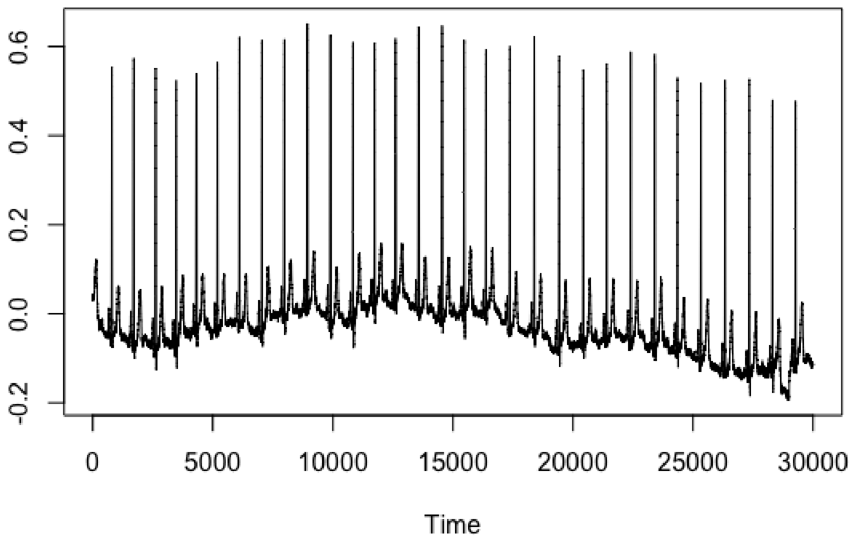


Figure 1: Plot of ECG for Patient 1

this challenge it is important to select sensible features to extract at the beginning to prevent wasting time. Once the features have been extracted from the data exploratory data analysis (EDA) can be preformed to determine if these features should be included in the model. This is done in section 2.3.1.

An ECG shows multiple heart beats, however more insight into the condition of the heart can be gained by looking at the individual heartbeats. A diagram of an individual heartbeat can be seen in Figure 2. This diagram shows some of the important features to look at for this challenge. Namely the P and T-wave alongside different segments such and the PR interval and QRS complex. Another feature that can indicate a heart condition is elevation of the ST segment which means that the heartbeat takes longer than normal to return to the baseline after the QRS complex. [1] These are all sensible suggestion for features to try and extract, the next sections will show the process used to extract some of them.

2.2.4 Splitting the ECGs into Individual Heart Beats

The first manipulation of the data is to split the ECGs into individual heart beats. This has been done by splitting the ECG at the maximum point of each heartbeat (at R in

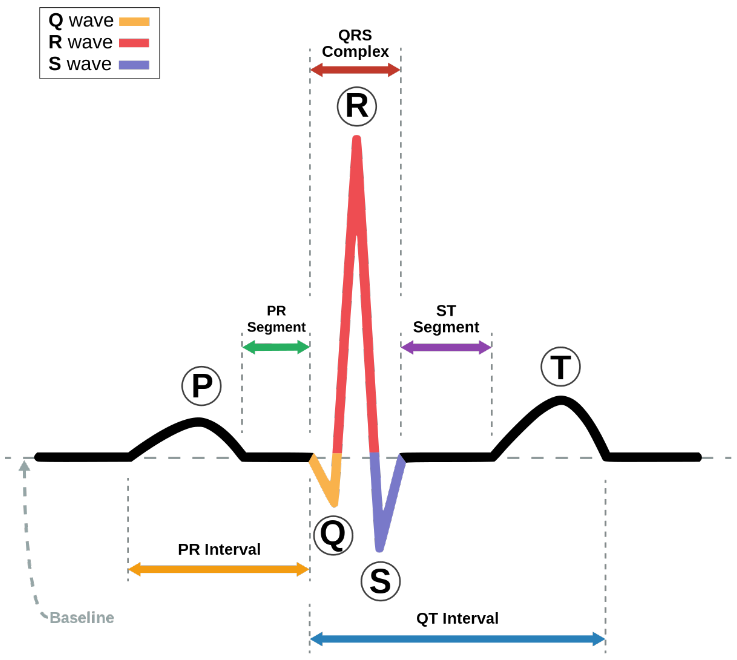
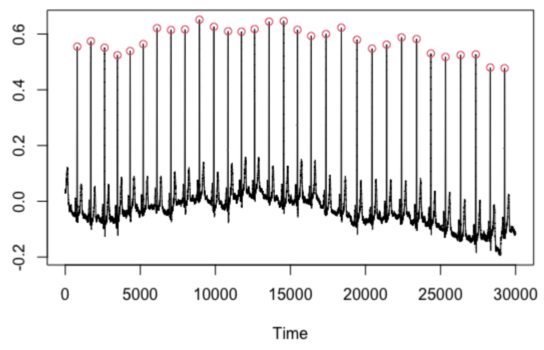


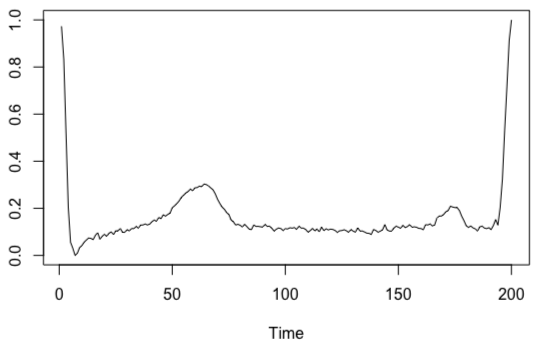
Figure 2: Diagram of an individual heart beat [2]

Figure 2). To do this, an algorithm in `R` has been set up that takes a threshold value and identifies all points higher than this, it also looks to see if these point are higher than the points either side of it. If this is the case, the index on the time scale is noted. In Figure 3a these maximum values have been plotted. The 4th and 5th maximum are selected to be used as the splitting point for the ECGs. This choice is reasonably arbitrary, and any other maximums could have been used, but the first two points would not be recommended, as the patient may still be be adjusting to the machine causing discrepancies in the ECG.

Once the maximum values being used for the splitting have been selected, the points in between them are used to create an interpolation for the heart beat. The interpolation is used to create a new vector of length 200 which is then stored in a new data set. During this process, the values in the new vector are all scaled to be in the range $[0, 1]$. This is allowed as the actual values recorded by the ECGs are not important when reading them as machines are often not calibrated suitably, or are prone to drifting and therefore it makes it easier to compare ECGs once they have all been moved to the same scale (standardised). A plot of the new data for patient 1 is shown in Figure 3b.



(a) Maximums plotted on patient 1



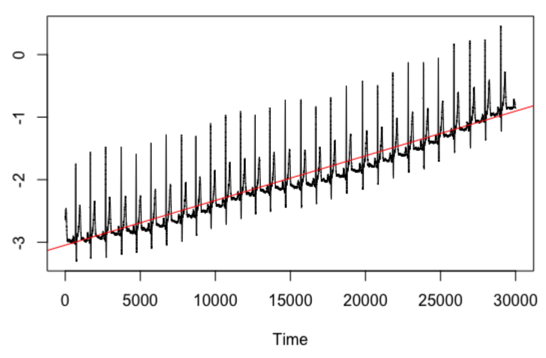
(b) Individual heartbeat from patient 1

Figure 3: Plots from splitting patient 1s ECG

2.2.5 Issues with Splitting Method

This method for splitting the ECGs works well for a limited number of the patients given to us however there are two issued that are encountered when trying to do this for all the patients. One issue is, as mentioned earlier, the machines used for recording ECGs are prone to drifting while the readings are taking place. An example of this can be seen in Figure 4 which shows patient 4 from the training data. To remove the drift the line of best fit for the data is found using the `lm` function in `R`. This is plotted in red in Figure 4a. The line of best fit can be used to remove the drift by transforming each point in the vector using $m =$ the gradient and $c =$ the y-intercept. Each point x was transformed using $x \leftarrow x - m * i - c$ where i is the point on the time axis. A plot of the transformed data can be seen in figure 4b.

The next issue encountered while splitting the heartbeats is that using the method described in Section 2.2.4 to find the maximum points fails when there is a lot of artifact in the reading. Artifact is the name given to the small variations visible in some ECGs caused by the patient moving as the ECG reading is taken (Note: not P-wave, T-wave and QRS complex shown in Figure 2). Figure 5a shows a plot of the first $10,000$ points for patient 3 in the training set along with maximums created using the method explained in section 2.2.4. It can be seen that each peak now contains multiple maximum points, this is further highlighted in Figure 5b which shows a zoomed in plot of one of the peaks from


(a) ECG with drift plotted with line of best fit

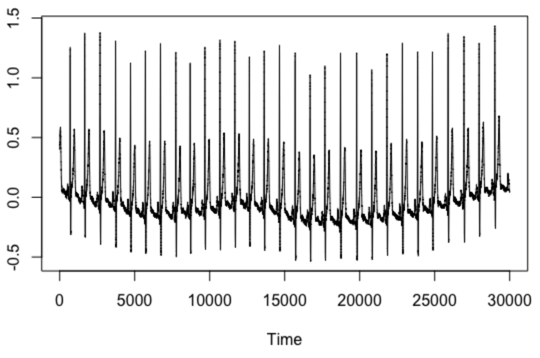
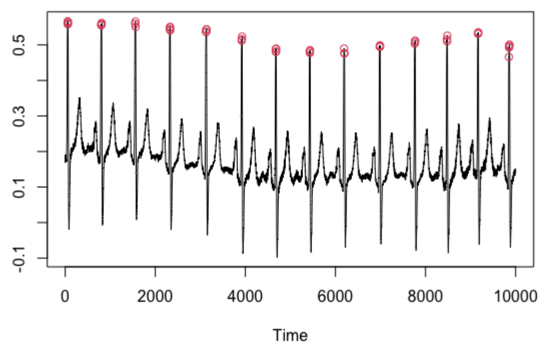

(b) ECG with drift removed

Figure 4: Plots showing drift for patient 4 being removed


(a) ECG with multiple maximums at each peak

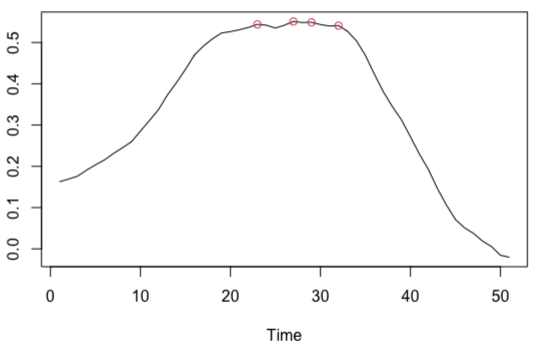
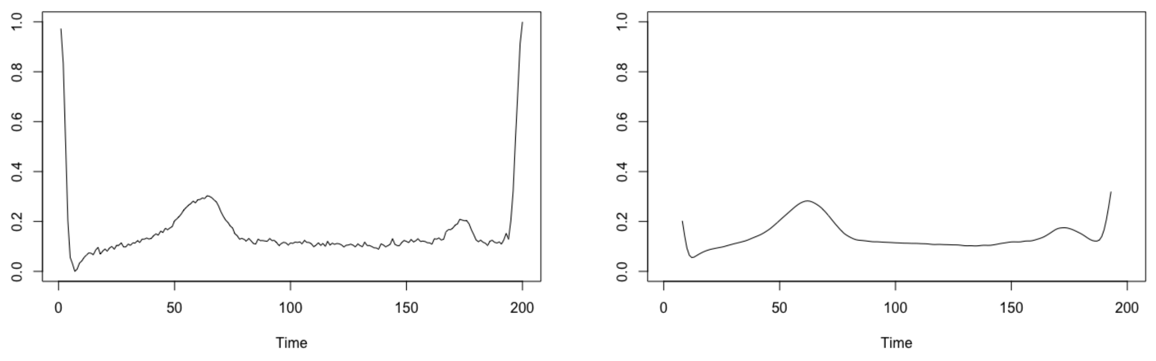

(b) Zoomed in plot of a peak

Figure 5: ECG of Patient 3 showing multiple maximums at each peak

patient 3's ECG. This causes issues as now it is not clear which maximums to use to split the heart beats at. However as these maximums are so close together it does not matter which maximum is picked so when looking at the whole ECG if a maximum is close to another maximum then one of them is selected and the other removed. This is repeated until no maximums are left close to any others. The tolerance used in this report was any maximums within 100 time steps of another maximum. Code showing how the splitting is done is presented in section 5.1.1 of the Appendix.



(a) Patient 1 heartbeat, unsmoothed (b) Patient 1 heartbeat, smoothed

Figure 6: Plots showing the smoothing of patient 1's heartbeat

2.2.6 Smoothing the Curves

With ECG readings for patients now having been split into single heartbeats, the artifact mentioned in Section 2.2.5 can be removed. This is done by smoothing the functions. The method used in this report to smooth the functions is the moving averages method, commonly used in economics. This method has been selected because just as when looking at a stock chart one might want to understand the general trends of the chart instead of what is happening over a very short period of time, for this report the general trends of the heart beat are desired and not information relating to the artifact. So as this method preforms well for a similar problem it has been implemented here.

The moving averages method requires a window size w to be selected (usually an odd number). Here $w = 15$ has been selected. Once a window size has been selected each value x in the data is looked at in turn and assigned a new value $\hat{x}$ such that,

$$\hat{x} \leftarrow \frac{\sum_{i=1}^{n} \hat{x}_i + x + \sum_{j=1}^{n} \tilde{x}_j}{w} \tag{2.2}$$

where $n = \frac{w-1}{2}$, $\hat{x}_i$ are the n values to the left of x and, $\tilde{x}_j$ are the n values to the right of x. Essentially this method is mapping all values to the average of the 15 closest values, thus smoothing the curve while still encapsulating the general trends. Figure 6 shows the change in patient 1's heart beat after the smoothing takes place.

There are two issues with the moving averages method. Firstly, the initial $n = 7$ and

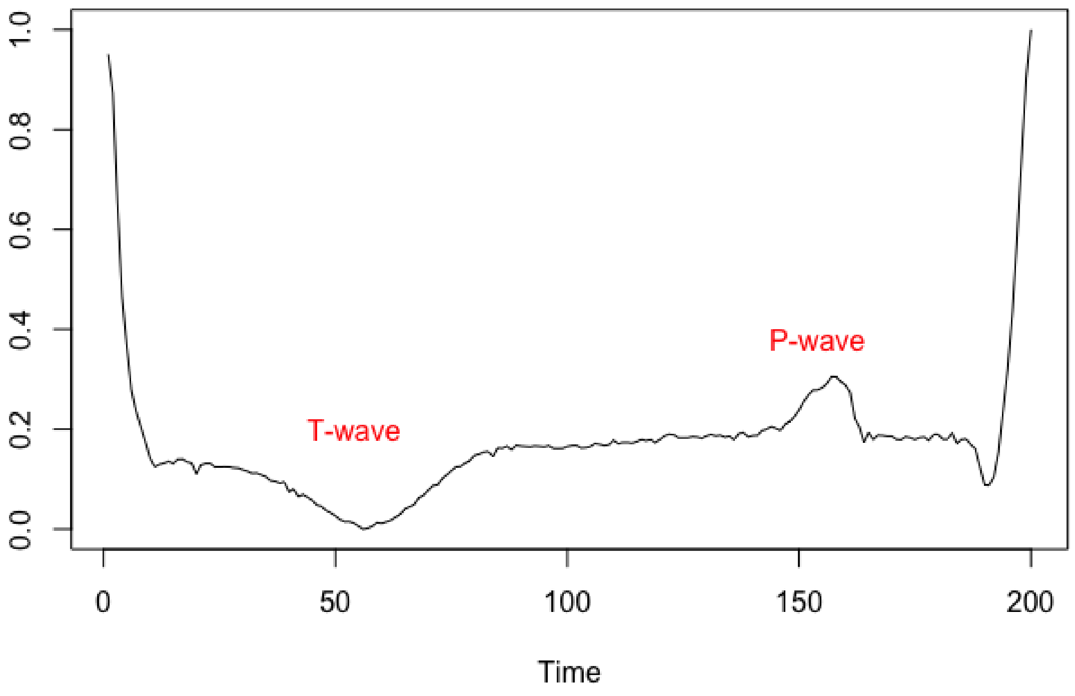


Figure 7: Patient 43's heartbeat with an inverted T-wave

last $n = 7$ points do not get assigned new values as there are not n values on either side of them. This, however, is not a problem here as the behaviour of the ECGs is known in this region. As all of the heartbeats have been scaled, the values approach 1 here. The second issue with this method is the selection of the window size w as the trade off for selecting a higher value for w is the risk that general trends in the graph will be lost. Here, $w = 15$ is selected as it is high enough to remove the artifact from the heartbeats but does not lose the general trends of the graph as shown in Figure 6b.

2.2.7 Feature Extraction

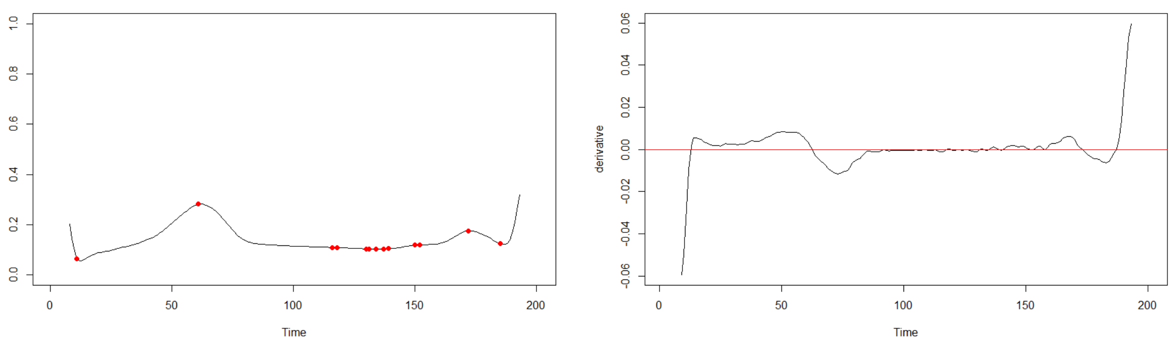
The work done so far in Section 2.2 allows features to be extracted from the data and put into a new design matrix. In this section, two methods for extracting features will be highlighted.

The first feature to extract is weather the T-wave is inverted or not. An example of a patient with an inverted T-wave is shown in Figure 7. It is trivial to identify whether a patient has an inverted T-wave by visual inspection. However, in practice, this is not a viable method for use on large training and tests sets, and therefore this process requires automation. As a result of efforts so far, the heartbeats for each patient can be approximately differentiated using the inbuilt `R` function `diff()` which takes a vector $\mathbf{x}$ as input and returns the difference between the points x_i of $\mathbf{x}$. In this case, this is an approximation of the gradient at the different time-steps as the values of $\mathbf{x}$ are all plotted 1 time-step apart, thus using the approximation for the derivative at a time-step t_i and that $t_{i+1} - t_i = 1$,

$$f'(t_i) \approx \frac{f(t_{i+1}) - f(t_i)}{t_{i+1} - t_i} = f(t_{i+1}) - f(t_i).$$

This approximation of the derivative can then be used to find the stationary points of the heartbeat. A plot with the approximate stationary points is shown in Figure 8a the accompanying plot in Figure 8b shows a plot of the derivative with the x-axis plotted in red. Finally, this can be used to determine if the T-wave is inverted by looking at the value of the second derivative at the second stationary point as this is typically the stationary point in the middle of the T-wave. If this value is negative, then it is at a maximum and the T-wave is not inverted (coded as 0 in the new data frame) and if this value is positive, then it is at a minimum and thus the T-wave is inverted (coded as 1). This method is not entirely robust, however, as there is a non - zero probability that the stationary point on the T-wave is not the second one. In these cases, the number of the stationary point must be inputted manually. In practice, this is not heavily time consuming, as it is straightforward to identify the patients this happens to, simply by observing the plots similar to that in Figure 8a. For the data used in this report, only around 12% of patients fell into this category.

The second feature extracted that will be discussed in this report is the length of the QRS complex. The QRS complex is shown in Figure 2. To extract this feature a similar method to above can be used. However instead of looking at the second stationary point, the first and last stationary point are used. Due to how the heartbeats have been split the graphs do not contain a full QRS complex. However as these graphs should in theory be periodic the length of the QRS complex q can be calculated using the formula



(a) Patient 1 heartbeat with stationary points (b) Derivative of Patient 1 heartbeat

Figure 8: Plots showing how stationary points for Patient 1's heartbeat are found

$q \leftarrow TP_{initial} + (200 - TP_{final})$ where TP is the vector of positions of the stationary points. As with the inverted T-wave feature the positions of the stationary points should be checked manually with a plot. Only around 9% of these needed to be changed manually.

Other features such as the length of the PR interval can be extracted using similar methods to the ones highlighted above. In the next section the features that have been extracted here will be used in model development.

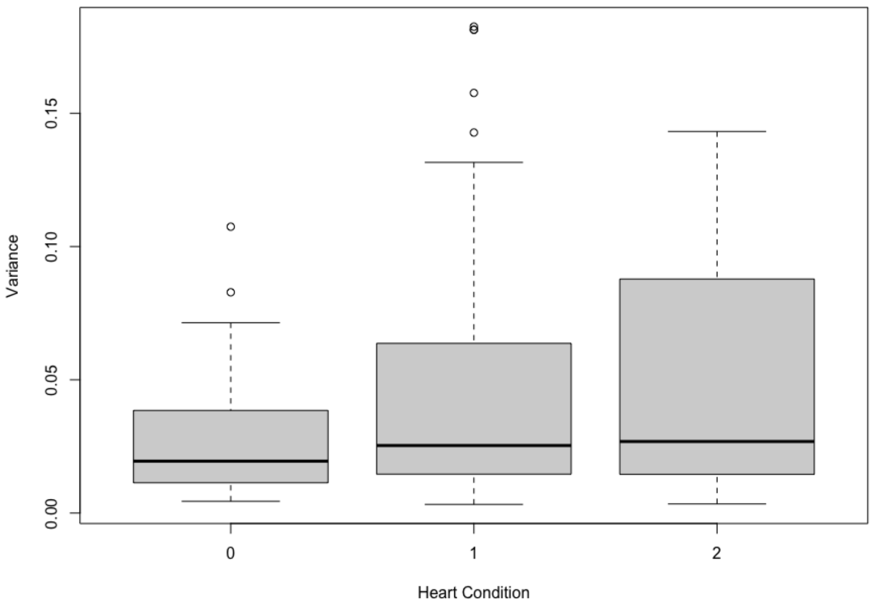
2.3 Model Development

2.3.1 Exploratory Data Analysis (EDA)

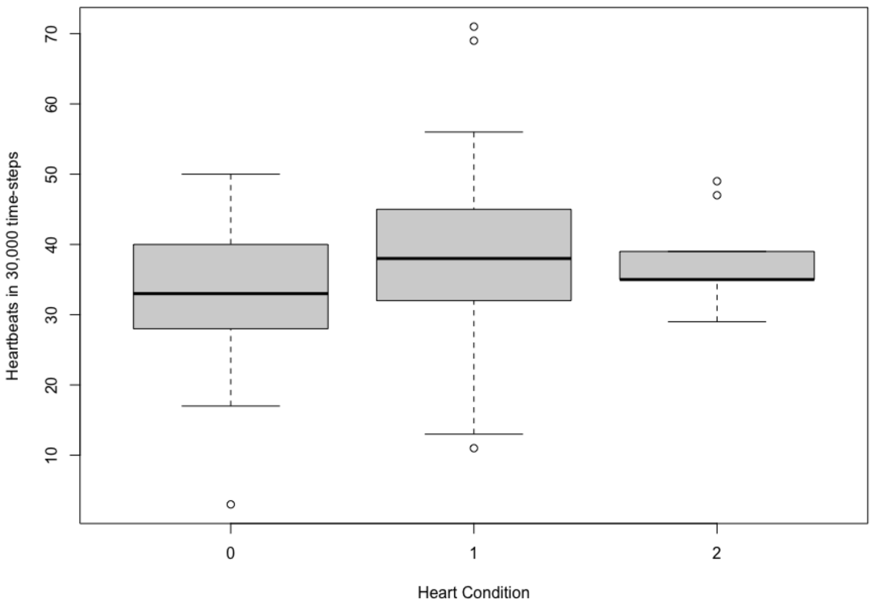
Figure 9 and 10 contain a series of boxplots for different potential features to be used in a model for predicting heart conditions. These plots will be referenced in future sections.

2.3.2 Benchmark Model

To find a simple benchmark model for this data, consider the data before all the complex features have been extracted. One feature that is trivial to extract with out any data manipulation is the variance of the vectors for each patient. This should provide some insight into the condition of the heartbeats, namely if the ST segment takes a long time to return to the baseline this would increase the variance and if the T or P-waves are larger than normal this would also increase the variance. Therefore, this seems like a sensible starting point for a model. This is supported by Figure 9a which shows that the mean

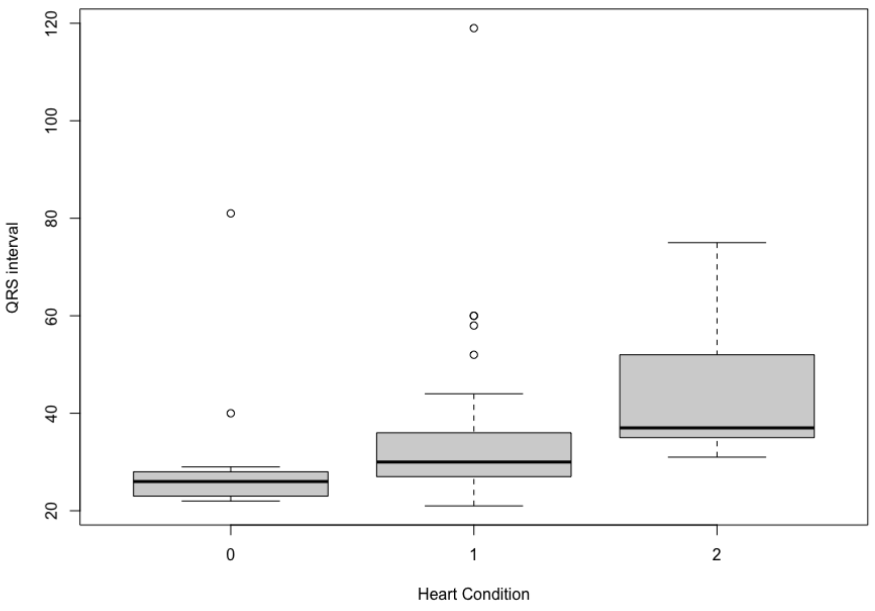


(a) Variance of ECGs

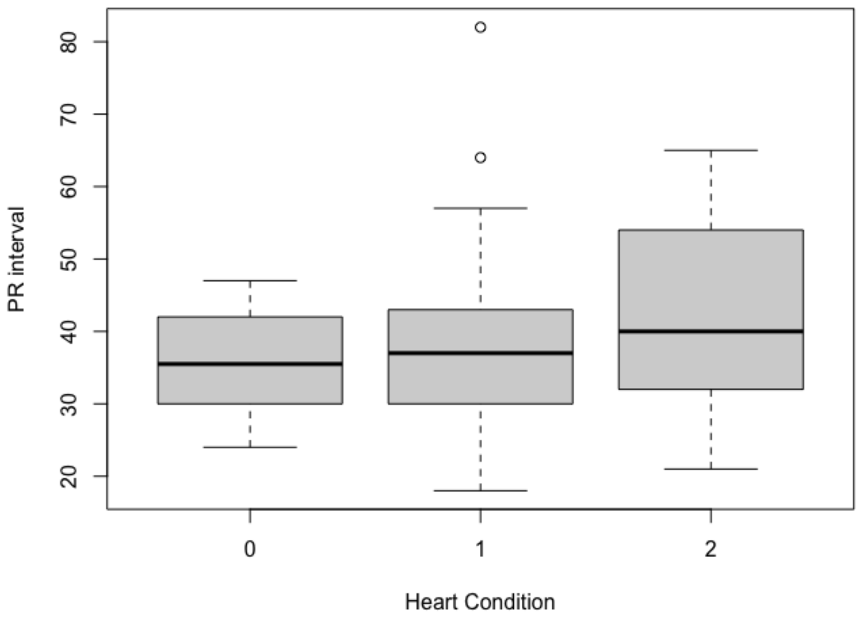


(b) Heart rate

Figure 9: Exploratory data analysis plots for diagnosing heart conditions



(a) QRS complex



(b) PR interval

Figure 10: Exploratory data analysis plots for diagnosing heart conditions

variance is slightly higher for the unhealthy conditions. In hindsight it would have been smart to scaled these vectors $\mathbf{v}$ such that $v_i \in [0, 1]$ to make the comparison of variances more fair.

Implementation is simple, requiring the variance for each patient in the training set to be calculated and added to a data frame along with heart condition of the patient. This can be done easily with a for loop in `R`. A decision tree was made using the `rpart` function to categorise heart conditions based on this variance. A plot of the decision tree can be seen in Figure 11. Each box contains the most abundant category at that node, the split of the data for categories 0, 1 and 2 at that node and the percentage of the full training data set present at that node. From the plot it can be seen that this model will never classify a patient to category 2 (cardiomyopathy). The reason for this could be the small number of data in the training set of this classification. As mention in section 2.2.1 and visible in Figure 11 only 8% of the the training data is for patients suffering from cardiomyopathy.

This model preformed better than expected on the test data with a prediction accuracy of 65%. This was better than expected as the accuracy attained from splitting the training data into new testing and training sets was only 62%.

2.3.3 Random Forests

Before introducing Random Forests, it is helpful to introduce decision trees first. A decision tree is a simple algorithm that can be understood reasonably intuitively. It is a supervised learning algorithm based on setting up nodes at which decisions are made. Figure 12 can more effectively demonstrate its logic.

Random Forests is an algorithm composed of multiple decision trees (There is no correlation between any given pair of decision trees), belonging to the bagging type. When a new input observation comes in, each decision tree in the forest will make judgments and a classifications. Each decision tree will obtain its own classification result. The random forest will take the most popular classification result as the final result. Due to the high accuracy and generalisation performance, the model also has good stability. [4]

Reasons for the strong performance of random forests can be credited to the "random"

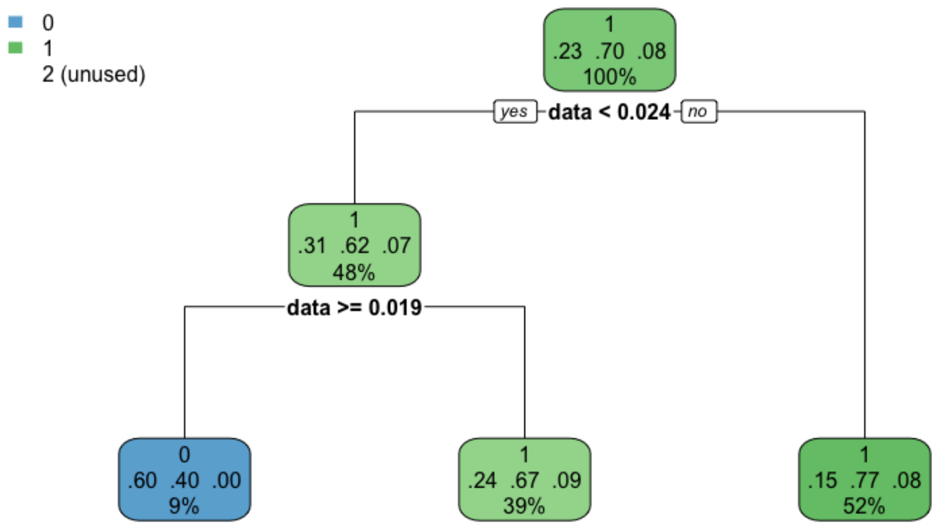


Figure 11: Decision tree based on variance of the whole ECG

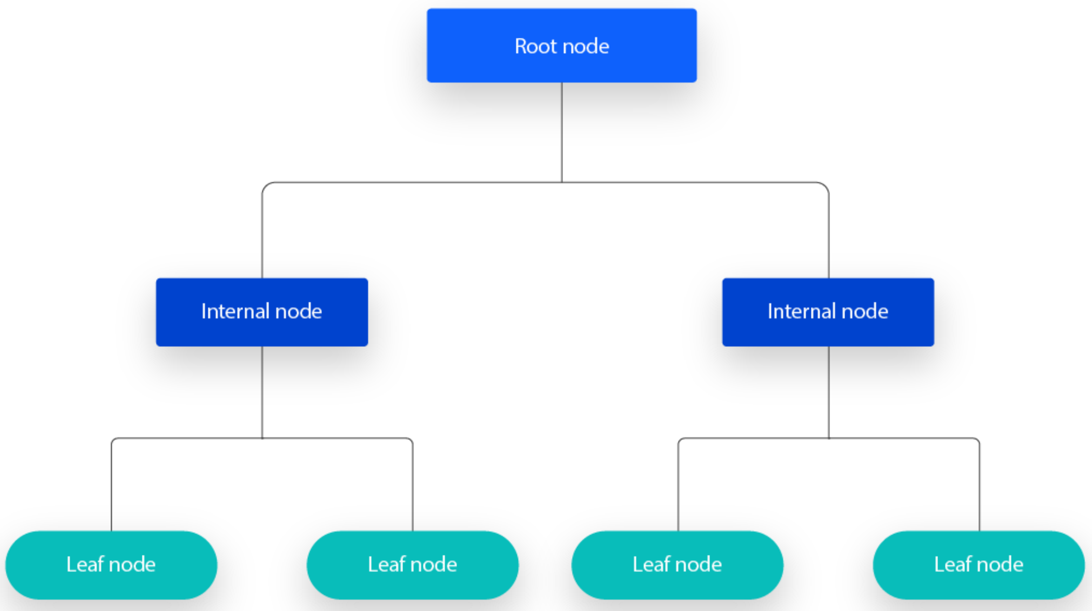


Figure 12: Decision tree [3]

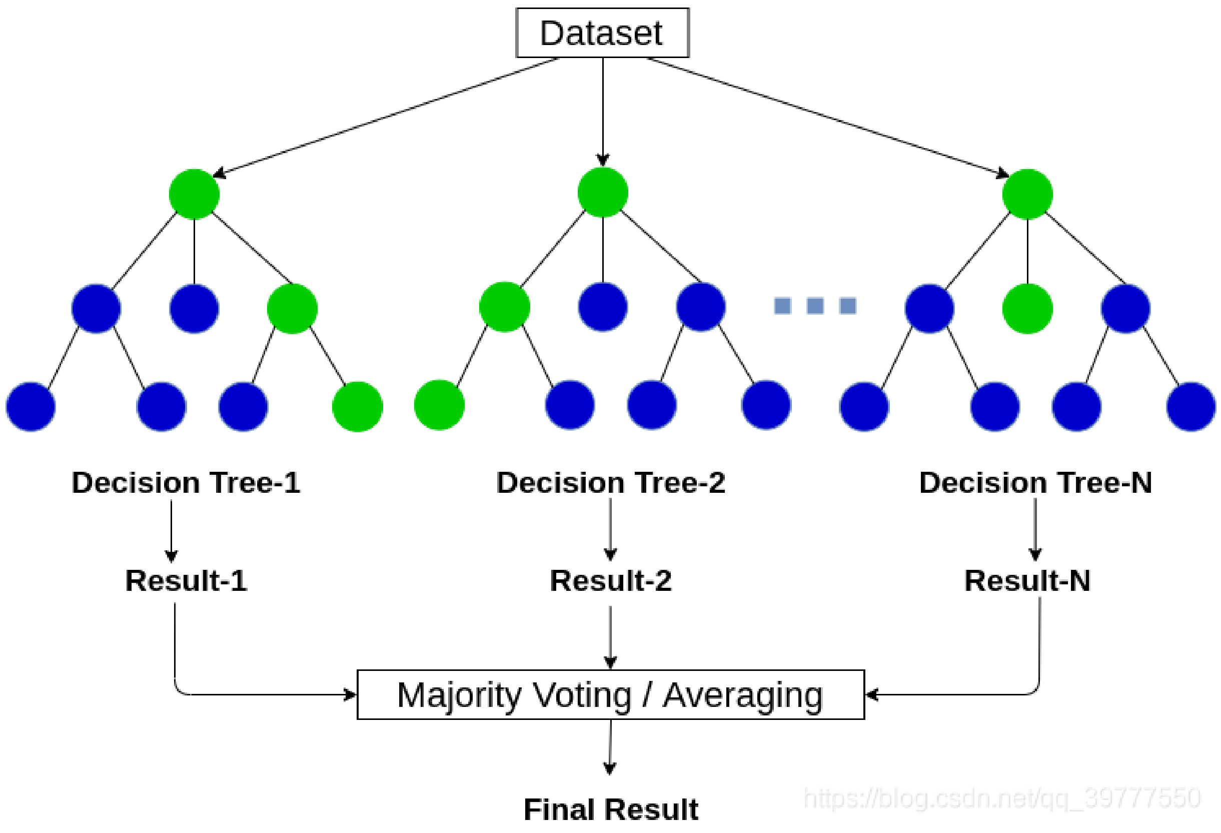


Figure 13: Random forest [5]

and "forest" aspects of the method. Random samples and random features are used in the construction process, and this randomness avoids over-fitting. And it can make predictions using multiple decision trees, so its prediction accuracy is higher than a single decision tree.

Random forests have been selected to use in this report as decision trees have been used successfully in the field of medicine many times in the past. As random forests are an extension of decision trees it is a logical model to try. Also due to the small amount of data in the training set the random aspects of random forests will help to prevent over fitting.

Another reason classification trees are a good selection is due to the ability to include a loss matrix in the model. A loss matrix can be added to a model when the consequence of misclassifying an observation differs depending on the actual class of the observation. For example in this report it is much worse to misclassify a patient with a heart condition than a healthy person as if MI or cardiomyopathy is missed then the health of the patient is at risk. Where as if a healthy patient is missclassified this will most likely be corrected once further tests are carried out. More information on the loss matrix can be found in The Elements of Statistical Learning book (p310 [6]). It is important to note that this is important to consider when implementing a model in the real world however as in this report the aim is to maximise the prediction accuracy, seen in equation 2.1, a loss matrix has not been implemented.

Two different results for the test data were attained using the `randomForest` function in `R`. The first predictions were made using only the factors: T-wave inversion and length of the QRS complex. This scored 67% on the test set and 69% when testing on the training set. This was not improved by adding a third factor, length of the PR interval. This produced a prediction accuracy of 67% on the test data and 66% on the training data.

2.3.4 Cross Validation

Cross validation (CV) can be used to tune parameters within the random forest model, namely the optimum number of decision trees to include in the model. For this challenge the main issue while using cross validation is the small number of patients in the training set and the uneven distribution of the heart conditions (80 patients having MI and only 9 with cardiomyopathy). For this reason, the cross validation used in this report is slightly modified and was implemented manually in `R`. The training data is split into 10 folds as usual and then the random forest model is created with, say, n trees in the forest where n iterates through $\{1, 2, 3, ..., 50\}$, for each combination of 9 folds and the accuracy is calculated using the 1 fold not used for training. The number of trees n which has the resulting highest mean accuracy is the optimum number of trees for this split of the data. This is the typical cross validation process thus far, but due to the small number of classifications for cardiomyopathy in the training set, one fold could contain all the patients with cardiomyopathy, therefore this process needs to be repeated with different splits of the data. In this study, it is repeated 1000 times. Figure 14 shows a histogram of the optimum number of trees from each cross validation. The mean optimum number of trees is around 22.

Thus a new random forests model can be created using 22 decision trees, this is

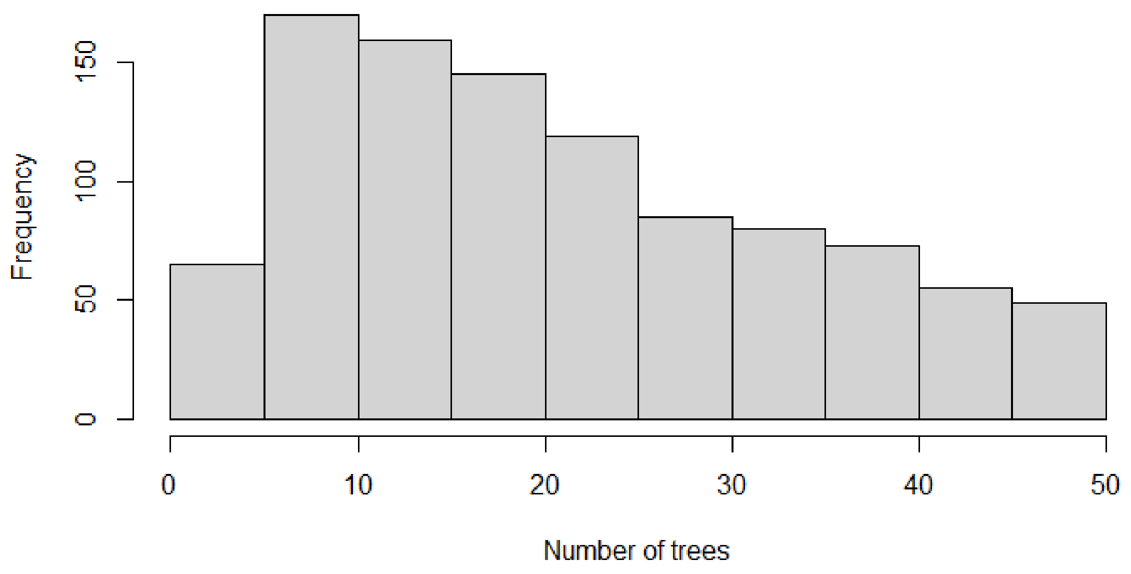


Figure 14: Histogram of optimum number of trees from 1000 CVs

implemented in `R` using the `randomForest` package and function setting the parameter `ntree = 22`. This model was used to create predictions on the test data set using inverted T-wave and QRS complex as the predictor variables as this preformed better than when PR interval was added. These predictions scored significantly worse than expected with an accuracy of 60%. This however was due to an error in the code and not the method, specifically incorrect values in the data frame. Once this was spotted and the correct values re-added to the data frame new predictions were made. These predictions were very similar to the previous random forest predictions highlighted in section 2.3.3. Only 2 values differed so the accuracy of the new predictions with cross validation used are 67% $\pm$ 2. When this method was preformed on splits of the training data the prediction accuracy was 70% which is the highest accuracy achieved in this report, suggesting this method is the best model suggested in this report. In the next section the results will be presented and the models performance evaluated.

Week	Model	Accuracy from Test Set	Accuracy from Cross Validation
1	Variance of whole ECG	65%	62%
2	Variance of individual heartbeat	58%	60%
3	Random Forest	67%	69%
4	Random Forest with more features	67%	66%
5	Tuned Random Forest with 22 trees	60%	70%

Table 1: Accuracy of predictions made on test data

2.4 Model Evaluation and Results

Table 1 shows the results from the predictions made over a 5 - week period. The benchmark prediction discussed in Section 2.3.2 appears to perform well on the test set, better than was suggested by the accuracy attained from just the training data. The accuracy dips in the second week. Week 2's predictions were made based on the variance of the individual scaled heartbeats. The data manipulation to extract these is shown in section 2.2.4. However the focus during this week was more on data manipulation than model improvement which explains the lower prediction accuracies. Week 3 and 4's predictions show an improvement on the benchmark model. Both have prediction accuracies of 67% on the test set. However performance is better on the simpler model according to the prediction accuracies attained from the training data. The lack of improvement from week 3 to 4 may have been attributable to the minimal value added by including PR-interval as a feature. This can be seen from Figure 10b in section 2.3.1, which shows that the correlation between PR-interval and heart condition is weak. Finally, the result for week 5 is unreliable due to an error with the data frame used as discussed in section 2.3.4. This model however is robust and could have raised the prediction accuracy on the test set to 69%. Further evidence that this model should have been the best model presented in this report is seen from the highest prediction accuracy on the training data set with a prediction accuracy of 70%.

Figure 15 shows the improvements of the model over 5 weeks. It can be seen that there is a general upwards trend when using cross validation to calculate the prediction

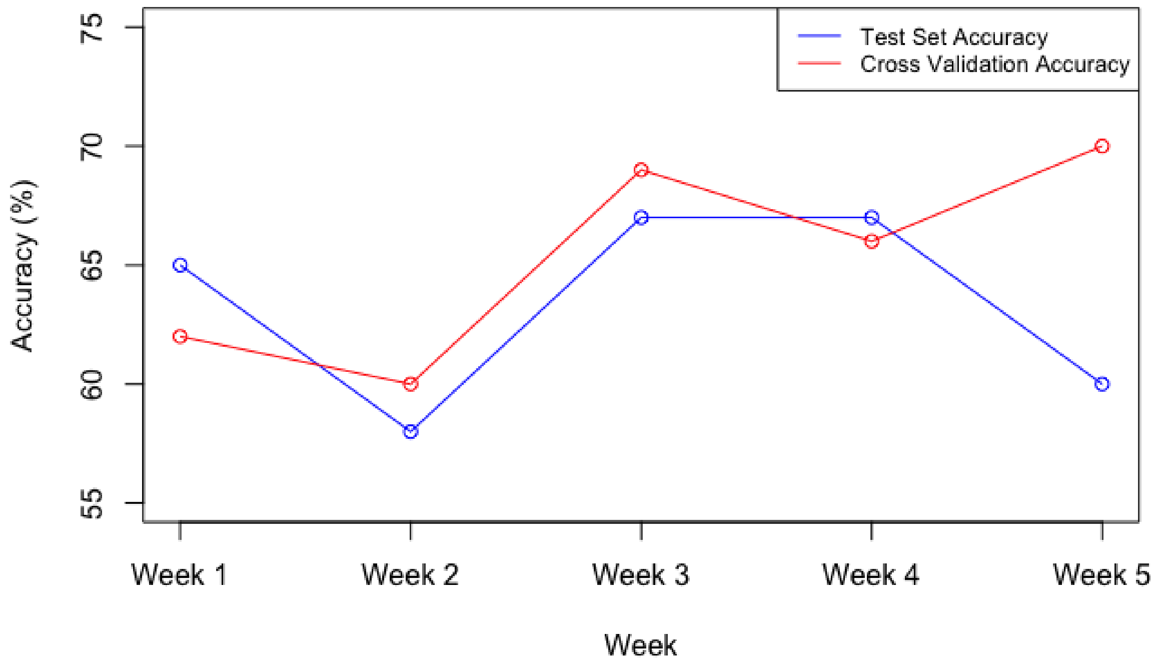


Figure 15: Plot of the Accuracy of the model over 5 weeks

accuracy for the training data set. Unfortunately this trend is not so apparent when looking at the results from the testing data. However if the error in the final submission had been spotted earlier this could have resulted in a more obvious improvement for the predictions from the model on the testing data set.

2.5 ECGs: Conclusion and Discussion

The objective of this study is to diagnose whether a patient is suffering from a heart condition based on their ECG readings. The study develops a machine learning model that makes raises the prediction accuracy higher than the baseline model in an attempt to maximise Formula 2.1 in the Aims section, thus achieving the main aim intended for this study. Since the data itself is not in a form that is easy to integrate into a machine learning approach, some processing of the data was done first in order to extract features. Here, the inversion of the T-wave and length of QRS complex were chosen as the best features to include in the model. Through a few weeks of experimentation, it was found that Random Forest was a suitable model to predict the heart condition of the patient. Also, the best number of decision trees in the random forest was determined by cross validation to be 22, resulting in a prediction accuracy of 67% ± 2 for the test data and 70% for the training data.

However, upon reflection, there are some optimisations that could be made in this project to make the model more accurate. The first is additional feature extraction: simply more features could be extracted to improve the accuracy of the prediction, such as the ST segment, which has been analytically solved, but has not been implemented computationally in code format. Secondly, automating the feature extraction process would improve the repeatability of this study on new data sets. Full automation has not been achieved as the processes outlined in section 2.2 still require screening to ensure accurate feature extraction. Finally, one can try to explore more machine learning methods (potentially multinomial logistic regression) to improve the prediction accuracy.

3 Predicting Train Delays

3.1 Introduction

3.1.1 Background

Deploying accurate train delay prediction is crucial for effective railway traffic planning and management, as well as for providing adequate passenger service quality. In this portion of the study, a model is established that illustrates the interaction between the arrival time and the scheduled arrival time of trains at Nottingham and various factors which affect this. Based on these factors, we identify numeric independent and the key dependent variable. A Random Forest (RF) prediction model in which the actual delay time at Nottingham corresponded to the dependent variable is then established, to capture this phenomenon.

3.1.2 Aims

The aim of this section of the study is to develop a model, which predicts the difference, in seconds, between the arrival time into Nottingham compared with the scheduled arrival time. The goal is to minimise the mean squared error of the predictions, given by

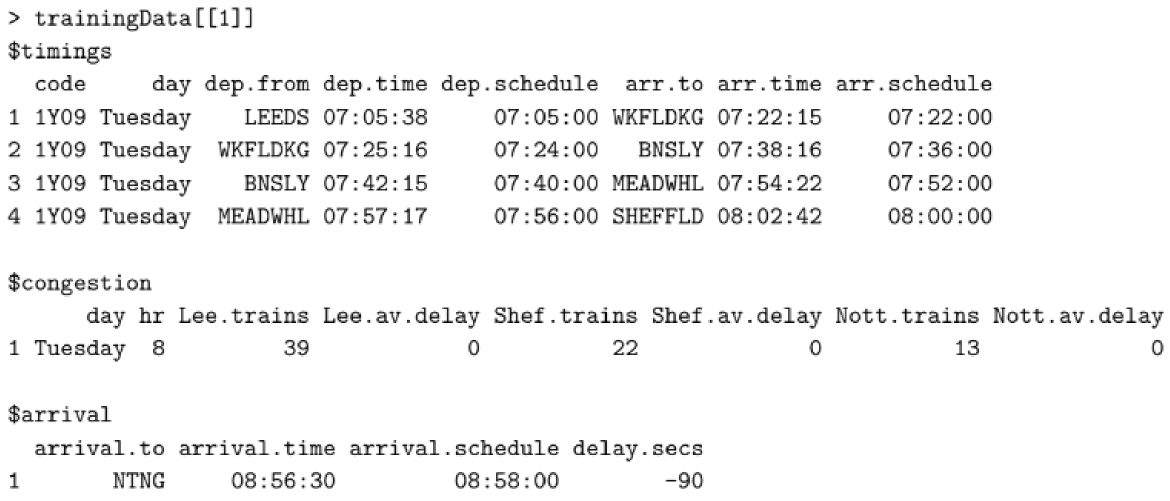
$$\frac{1}{n_{\text{test}}}\sum_{i=1}^{n_{\text{test}}}(y_i-\hat{y}_i)^2$$

Where, for observations in the test dataset, $y_i, i = 1, \ldots, n_{\text{test}}$, denotes the true (hidden) time delays in seconds at Nottingham, and $\hat{y}_i$ denotes the model predictions for it. As such, the aims of this study are to: (i) find an accurate machine learning methodology to predict the train delays and (ii) determine whether it is indeed possible to predict train delays with such a model.

3.2 Data Preprocessing: Train Delays

3.2.1 Exploratory Data Analysis (EDA)

The data provided are three datasets: a training dataset (‘trainingData’), test dataset (‘testData’), and a historical congestion (‘historicalCongestion’) dataset consisting of congestion data across weekdays and hours of the day. Below is the first entry from the training dataset.



```
> trainingData[[1]]
$timings
  code     day dep.from dep.time dep.schedule  arr.to arr.time arr.schedule
1 1Y09 Tuesday    LEEDS 07:05:38     07:05:00 WKFLDKG 07:22:15     07:22:00
2 1Y09 Tuesday  WKFLDKG 07:25:16     07:24:00   BNSLY 07:38:16     07:36:00
3 1Y09 Tuesday    BNSLY 07:42:15     07:40:00 MEADWHL 07:54:22     07:52:00
4 1Y09 Tuesday  MEADWHL 07:57:17     07:56:00 SHEFFLD 08:02:42     08:00:00

$congestion
      day hr Lee.trains Lee.av.delay Shef.trains Shef.av.delay Nott.trains Nott.av.delay
1 Tuesday  8         39            0          22             0          13             0

$arrival
  arrival.to arrival.time arrival.schedule delay.secs
1       NTNG     08:56:30         08:58:00        -90
```

Figure 16: R Output of first train observation in training dataset

There are 1859 observations (train timings on a train line from Leeds to Sheffield) in the training dataset, 465 observations in the test dataset and 126 observations in the historical congestion dataset. In between Leeds and Sheffield, there are train timings (scheduled and actual departures) for each train, for Barnsley, Meadowhall, Wakefield and Normanton station, if the train stops there. Therefore, each train in the training dataset includes: (i) the arrival and departure timings at the aforementioned stations with original schedule for the train, (ii) congestion summaries across Leeds, Sheffield and Nottingham (main stations), and average delay across these stations in the previous 1 hour window, and (iii) the actual and scheduled arrival times at Nottingham, as well as its associated delay. The test dataset has the same format, but does not have the arrival time and delay at Nottingham, as this is what the model will predict.

For the historical congestion dataset, each row corresponds to a day of the week and an hour of the day. Therefore, the dataset displays an average of historical ($\sim$ 5 year) congestion levels at the corresponding day of the week and hour of the day. The congestion levels refers to the number of trains crossing through the main stations, as well as the actual/observed delay for the trains. Thus, for each observation, the historical summaries are comparable to day - congestion data for each observation in the training data. This is useful in the analysis, as it assists in gaining understanding as to the usual congestion levels at the time the train was running.

In order to explore underlying structure within the data, visual plots are utilised. As parts of the data are measured differently (as categorical variables as opposed to continuous), different types of visualisations are used, between box plots and scatter plots. Quantitatively, the Spearman correlations between the dependent variable (delay time at Nottingham) and all potential independent variables are also calculated. Spearman correlations are used, as opposed to Pearson correlations, due to the non - normality of the dependent variable. Indeed, this can be shown as per the plot in Figure 17.

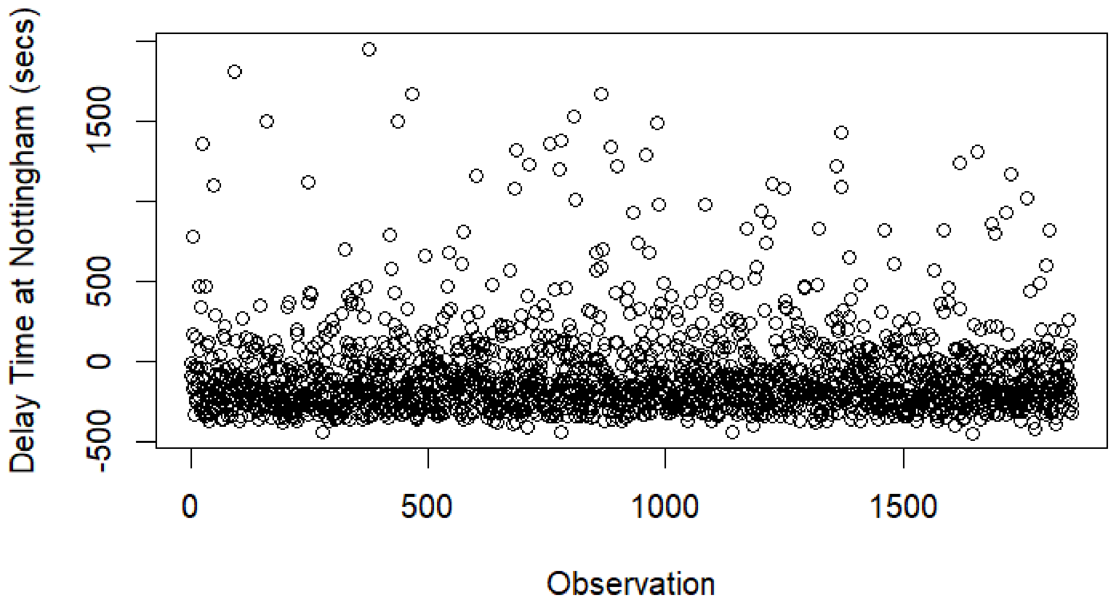


Figure 17: Delay Time at Nottingham in seconds

Clearly, these delay times do not exhibit normality. Note also that here, three outliers are removed. The threshold being a delay time at Nottingham of above 2000 seconds.

First, to explore the relationship between day of the week and the delay time at Nottingham, the day of the week is captured in a binary variable, in such a way such that the train observation is 1 if it runs on a weekday (Monday - Friday) and 0 if it runs on a weekend (Saturday and Sunday). In Figure 18 a graphical representation of the factors is presented.

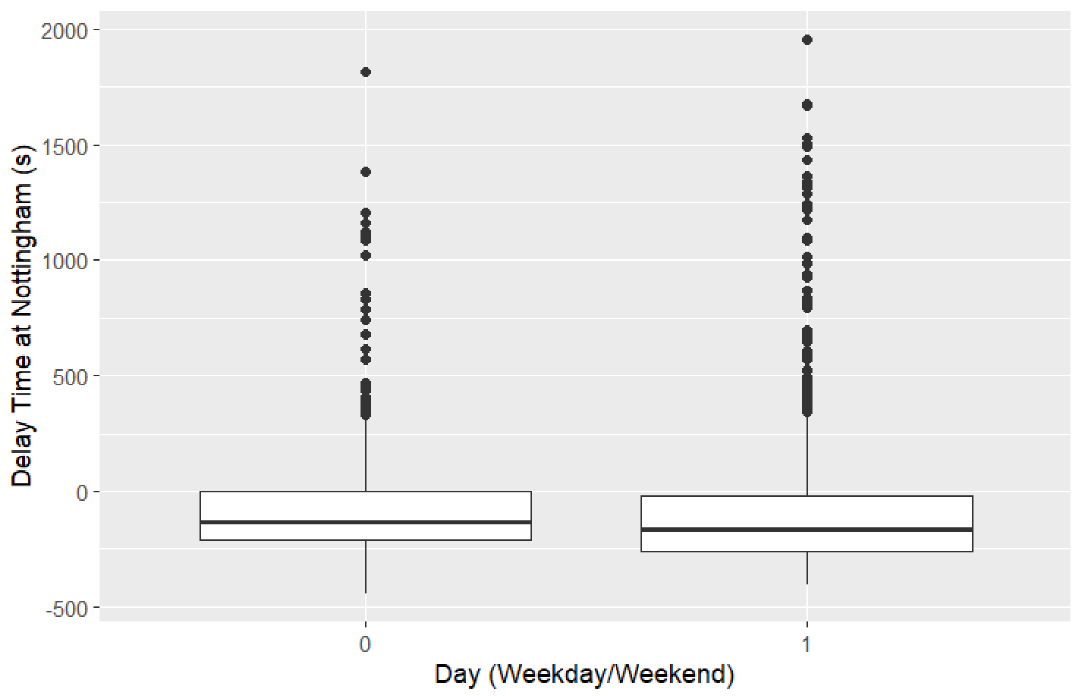


Figure 18: Day of the week compared with delay time (s)

From the box plot, we can infer that trains that run on a weekday would tend to longer delay times at Nottingham, measured in seconds, in general. It also appears that the distribution the delay time of trains that run on a weekday tend to have more outliers, than trains that run on a weekend. There is also a slight difference in the position of the boxes (non - overlapping), but both boxes do overlap slightly, indicating high similarity between the two groups. Next, time of day is explored. In order to ascertain this relationship visually, a box plot is used. This is because times of day can be split into peak and off peak times. Peak times are times of high congestion and train usage. In this model, this is captured with a binary variable, with 1 denoting if a train runs at a peak time (05:00 - 09:00 and 15:00 - 20:00) and 0 if a train runs at an off - peak time. The relationship between the time of day and the delay time at Nottingham can be seen in Figure 19.

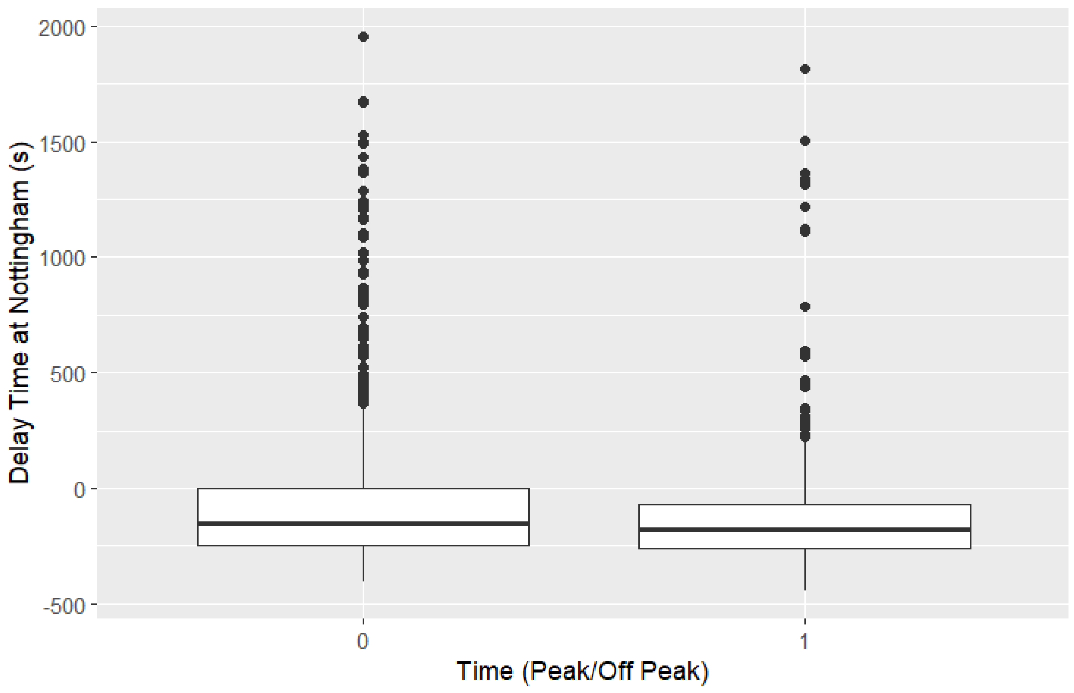


Figure 19: Train running time compared with delay time (s)

The distribution of the delay time at Nottingham for the off - peak (0) times seems to gather a significant amount of more outliers than its peak (1) counterpart. The inter - quarter range is also larger for the off - peak trains, indicated by the larger box in the plot.

It is also interesting to identify structure between the train stops and the delay time at Nottingham. In order to do this, binary variables are used to determine whether a train stops at Meadowhall, Barnsley, Wakefield or Normanton:

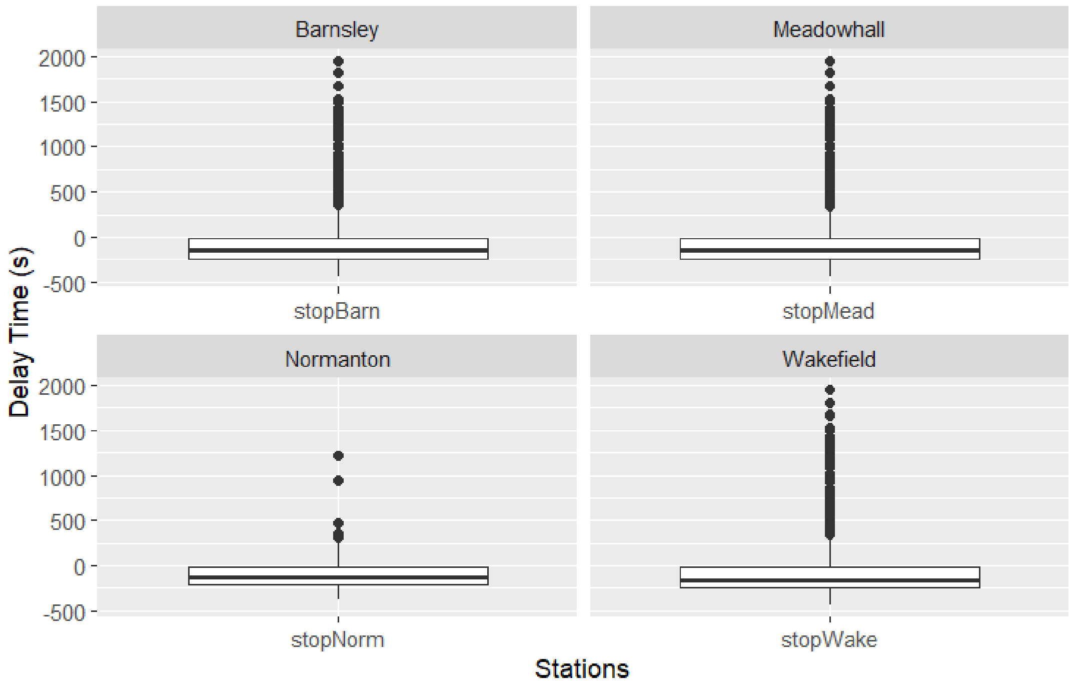


Figure 20: Distribution of delay time for trains which stop at Meadowhall, Barnsley, Wakefield or Normanton

The stations in between Leeds and Sheffield exhibit similar structure, with regard to how they are related to the delay time at Nottingham. In particular, Barnsley, Meadowhall and Wakefield appear to have a larger concentration of outliers than the trains which stop at Normanton. Note that the trains can stop at more than one of these stations in a schedule (for instance, a train may stop at Barnsley, Meadowhall and Normanton in a given journey).

Spearman correlations are used to compute this matrix. As briefly mentioned before, Spearman correlations are used, as opposed to Pearson correlations, due to the non - normality of the distribution of the delay time at Nottingham. In Table 2, the first 6 features and their pair - wise correlations are presented. From this truncated table, there does not appear to be a cause for concern with respect to mutli - collinarity, however, the correlation between ‘stopBarn’ and ‘stopMead’ is relatively high, compared to other values. In general, however, the correlations between the independent variables are not high enough for concern, allowing regression procedure to be suitably implemented.

	day	runTime	stopNorm	stopWake	stopBarn	stopMead
day	1.00	-0.06	0.05	-0.01	0.03	-0.04
runTime	-0.06	1.00	-0.14	-0.06	-0.00	0.00
stopNorm	0.05	-0.14	1.00	0.01	0.01	-0.01
stopWake	-0.01	-0.06	0.01	1.00	0.06	-0.00
stopBarn	0.03	-0.00	0.01	0.06	1.00	0.36
stopMead	-0.04	0.00	-0.01	-0.00	0.36	1.00

Table 2: Correlation matrix of data on first 6 features

In the following section, an in - depth analysis of the challenges with the data is explored, outlining complexities faced particularly in data pre processing.

3.2.2 Challenges with the data

From a relatively - macroscopic perspective, the challenge of predicting train delays at Nottingham is largely how to construct a concise matrix from the relatively unstructured given data, and accordingly form a predictive model. A systematic review of approaches to train - delays prediction in literature written by Tiong et al. (2023) [7] provides a general guideline to the workflow for a model development framework. That is, data pre - processing, methodology selection, data output and evaluation techniques. According to this review, supervised regression is the most widely - used machine learning technique when it comes to train-delays prediction, in the literature. Among all of such ML models, the Random Forest regression is the most common, hence making it a strong first candidate for considering a model to choose in this study. Furthermore, Tiong et al. (2023) [7] listed a table to categorise the literature based on different evaluation techniques, which helped to draw the conclusion that Root Square Mean Error (RMSE) and Mean Absolute Percentage Error (MAE) are commonly - employed factors to evaluate the performance of such predictive models since the goal of our prediction is to minimise the MSE and maximise the accuracy of the predictions from test data. Primarily in this report, the MSE is used. With proper methodology and evaluation techniques having been established, which will be further discussed in the Evaluation and Results section of this paper, most of the challenges were in the procedure of data pre-processing, which is what this portion of the study will largely focus on.

In specific, challenges came from three separate places:

The first type of challenge was to extract the underlying structure of training and test data itself. This seemed a basic criterion, but challenges around this point were throughout the whole modelling process. There are four split challenges the report will mention in this kind.

(i) First of all, after understanding how to correspond each congestion in trainingData with one row in historicalCongestion, it was found difficult in the study to utilise value in the historicalCongestion dataset. In particular, a question arose on how to compare congestion levels of each observation with the historical counterpart. Computing both the ratio and difference of each parameter were taken into consideration, and the latter was finally adopted to present whether the rail network was more or less congested than usual.

(ii) The second challenge was determining how many stations a particular train would stop at. It was not until the code for three fixed stations (Wakefield, Barnsley and Meadowhall) reported errors that this problem was realised. To rectify the situation, the code was created to test the intermediate stop (see Appendix 5.2.2 for the relevant piece of code).

As a result, four binary variables, which will be explored further in the Feature Extraction section, showing whether one observation would stop at each possible station were implemented into the design matrix, and the corresponding delay time was considered zero if it did not stop.

(iii) The third challenge was about Train Codes. From the code shown in Appendix 5.2.3, it was apparent that '1M01' only appeared in trainingData and the Code '1G01' appeared in testData only. It was decided to regard these two train codes as the same code for two reasons: (a) the run time of both Codes are Sunday 19 p.m and (b) there happened to be four observations in trainingData whose Code were 1M01 and one observation in testData whose code was 1G01, which satisfied the ratio of total numbers of observations between two datasets (1859 and 465).

(iv) The fourth challenge was concerned with outliers and influential data. The report identified 3 outliers when it came to delay time at Nottingham. Results showed that they skewed the data and affected model performance to some extent, as will be explored in the Results & Evaluation section.

The second type of challenge came from matrix construction. There were four sub challenges encountered:

(i) One challenge was high dimensionality. This is a topic that will be spoken in greater depth in the Dimensionality Reduction section.

(ii) Another noteworthy point was that there were no NA values in the dataset, which made it easier to prevent any data imputation from taking place. Hence, this advantage was beneficial to consider when constructing the matrix, which meant that 0 could replace NA in the matrix under such a circumstance.

(iii) Thirdly, based on Li et al. (2016) [8], run times are better estimated separately for peak and off-peak hours. Inspired by this paper, many variables in the design matrix matrix assumed binary form, including changing Day of a Week into Weekday (1) or Weekend (0), as will be explored further in the next section, and showing whether one observation stopped at certain station or not and whether one observation had certain Train Code or not. Through this idea, information can be extracted efficiently, but the necessity of dimension reduction increased, which will be mentioned in related section.

(iv) Last but not least, different sorts of delay time should be classified. Specifically, departure and arrival delay time at a same station were distinguished. Additionally, three necessary stations (Leeds, Sheffield and Nottingham) were different from the other four "non-necessary" stations (Normanton, Wakefield, Barnsley and Meadowhall) which had both a departure and arrival delay time. As a result, almost all the information obtained from the dataset could be reflected in the data matrix.

The final kind of challenges was at a computational level. There were numerous complex programming issues during the coding procedure, when pre processing the data.

(i) For example, converting the time period into numeric form. Although relevant code was ready-made, understanding the rule of conversion behind the given `textToSeconds` function was needed.

(ii) Moreover, since there existed certain stations which an observation would not stop at, '0' was set to be returned in this situation. A function called `stopCondConv` was constructed to implement such a functionality, and then another return was added to `textToSeconds` function when the input was not certain time period but '0'. Finally, the `lappy()` function together with the modified `textToSeconds` was applied to converting all these outputs into numeric form.

(iii) The flexible usage of `train()` function in the `caret` package and `set.seed()` function helped significantly.

3.2.3 Feature extraction

After discovering relationships and underlying structure within the data, feature extraction can be done. The following is an explanation of the features used in the model:

Day of the week: For a given train, the day of the week that the train runs on is an important factor in considering its delay time at a given station. This is because differences may be observed in passenger crowding and train utilisation during the week, and on the weekend, and this can then affect delay times due to a multitude of reasons. As mentioned previously, to capture this phenomenon, a binary variable is utilised in a way such that the train observation is 1 if it runs on a weekday (Monday - Friday) and 0 if it runs on a weekend (Saturday and Sunday).

Run time: This is the time period during which the train begins its journey (as given in the dataset, the time of the earliest departure) and ends its journey (the time of its latest arrival at a given station). Empirically, this time can impact delay times at a given station, as at some times of the day, trains are more or less utilised, thus affecting delays. As mentioned previously, this is captured with a binary variable, with 1 denoting if a train runs at a peak time (05:00 - 09:00 and 15:00 - 20:00) and 0 if a train runs at an off

- peak time.

Train stops: Between Leeds and Sheffield, the trains observed in the data stop at none of, or at least one of the following stations: Wakefield, Barnsley, Meadowhall and Normanton. Whether or not a train stops at each of these stations is captured as a binary variable, as there may be a propagation effect from one station affecting the later stations, and then the Nottingham station, making it a suitable explanatory feature. There are 4 features in this class.

Train code: Next, the train code is explored to identify whether there are is an effect of the train code/line on the subsequent delay (if at all) at Nottingham. This is useful, as it would help identify less efficient train lines and conduct appropriate actions to mitigate this. The unique train lines in the training data are: 1Y09, 1Y49, 1Y45, 1Y21, 1Y57, 1Y13, 1Y53, 1Y37, 1Y33, 1Y41, 1Y17, 1Y05, 1Y25, 1Y29 and 1M01. The train line codes are the same in the test data, except the 1M01 train line is assumed to be the same as the 1G01 train line in the test data. Therefore, for the test data, the design matrix remains the same.

Departure delay times: This group of factors captures the departure delay times for trains that depart from, if they stop there, Wakefield, Barnsley, Meadowhall, Normanton and Leeds, where every train departs from. This is a continuous variable, measured in seconds. The departure delay time is the scheduled delay time - actual delay time.

Arrival delay times: This group of factors captures the arrival delay times for trains that make arrivals to Wakefield, Barnsely, Meadowhall, and Normanton. There are also arrival delay times for each train at Nottingham and Sheffield.

Station congestion: For these factors, the congestion (number of trains at respective station) is looked at, for a particular hour on a particular day. For example, in the first observation of the training data, between the hours of $07:00 - 08:00$, on a Tuesday, 39 trains were delayed at Leeds.

Historical congestion difference: This set of factors captures the difference between number of trains delayed at a station, on a particular day of the week, during a particular hour - long window, and its corresponding historical average (the same hour - long window and the same day of the week). For instance, as before, the first observation in the training data reveals that between the hours of $07:00 - 08:00$, on a Tuesday, 39 trains were delayed at Leeds. Historically, however, ($\sim$ 5 years), between the hours of $07:00 - 08:00$ on a Tuesday, 38.58 trains were delayed on average, at Leeds. These values are compared to obtain 0.42.

Due to the high - dimensional nature of the data, a dimension reduction technique is explored in the next section. A full list of the features used in the model can be found in the Appendix (see table 6).

3.2.4 Dimension Reduction

Due to the large number of dimensions/factors discovered in the model, from the previous section, it is beneficial to look to dimension reduction techniques to explore the structure of the data in lower dimensions, to therefore ease computational burden. Here, Principal Component Analysis (PCA) is chosen as such a dimensionality reduction technique. As will be explored in this section, the method is successful in reducing the dimensions of the data, easing the computational strain, but does not actually improve the model in terms of performance.

Geometrically, one can think of PCA as the projection of data points onto a lower dimensional subspace. The basis vectors for this subspace are the eigenvectors of the correlation/covariance matrix for the data. These are the Principal Components (PCs). Formally, this process solves an optimisation problem (explored further in the Appendix), with one key constraint being that the Principal Components are uncorrelated to each other, to maximise explained variance for each component (preventing 'overlap' in explained variation). The projected data points in the new subspace are called the Principal Component Scores. For this particular set of data, due to the difference in scale between the factors, the correlation matrix is chosen, due to its invariance properties with regards to scale. Note that performing PCA on the correlation matrix is similar to performing PCA on the covariance matrix, after the data has been centered/standardised.

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9
day	0.03	-0.03	0.10	-0.24	0.37	-0.26	0.21	-0.09	0.22
runTime	0.01	0.15	-0.67	-0.04	0.05	-0.05	0.01	-0.04	-0.00
stopNorm	-0.01	-0.50	-0.06	-0.03	-0.04	-0.05	0.03	0.00	-0.06
stopWake	-0.00	-0.02	0.08	0.00	-0.06	-0.11	-0.24	-0.51	-0.04
stopBarn	-0.01	0.01	0.01	0.14	-0.30	-0.59	0.09	-0.03	0.05
stopMead	-0.01	0.03	0.01	0.16	-0.31	-0.56	0.11	0.09	-0.06
con_Leed	-0.00	-0.00	-0.04	0.24	-0.06	0.15	-0.23	0.15	0.28
con_Shef	0.05	-0.01	-0.05	0.19	-0.31	0.04	-0.33	0.04	0.25
con_Nott	0.04	-0.02	-0.03	0.28	-0.26	0.18	-0.21	0.04	0.19
con_avLeed	-0.12	0.02	0.01	-0.51	-0.23	-0.00	-0.17	0.11	-0.00
con_avShef	-0.09	-0.01	-0.00	-0.49	-0.28	-0.04	-0.07	0.08	0.06

Table 3: First 9 principal components using 11 of 38 original variables

In table 3, an truncated output of PCA on the correlation matrix of the data is presented, showing the first 9 principal components and 11 (of 38) features of the data. The interpretation is that each element in the output can be seen as a how much 'contribution' a variable has to that particular principal component. For instance, in the first column for PC1, the original variable 'day' can be interpreted as contributing 0.03 units to PC1, the variable 'runTime' contributing 0.01 and the variable 'stopNorm' contributing -0.01, and so on. This is important in capturing how to interpret the principal components themselves, as some may be more correlated to some of the original variables more than others. For PC1, trains that run on a weekday, at a peak time, and do not make stops at Normanton, Wakefield, Barnsley and Meadowhall (so only stop at Leeds, Sheffield and Nottingham) would have a higher principal component score on PC1 than trains that run on a weekend, at an - off peak time, and stop at, at least one of, Normanton, Wakefield, Barnsley and Meadowhall, given all else is equal with the remaining factors. Clearly, as the number of factors increases, interpretability of the model becomes difficult, reinforcing the need for such a dimension reducing technique in the first place.

Next, a suitable number of principal components must be chosen in order to reduce the dimension of the data, whilst still capturing the most important information (through explained variance) in the data. As such, in order to determine such a quantity, a 'scree' plot is employed. The scree plot constructed below captures how the explained variance of the model changes as the number of principal components kept in the model increases. For each principal component, its corresponding eigenvalue denotes its explained variance, or its ability to explain information and variation in the data.

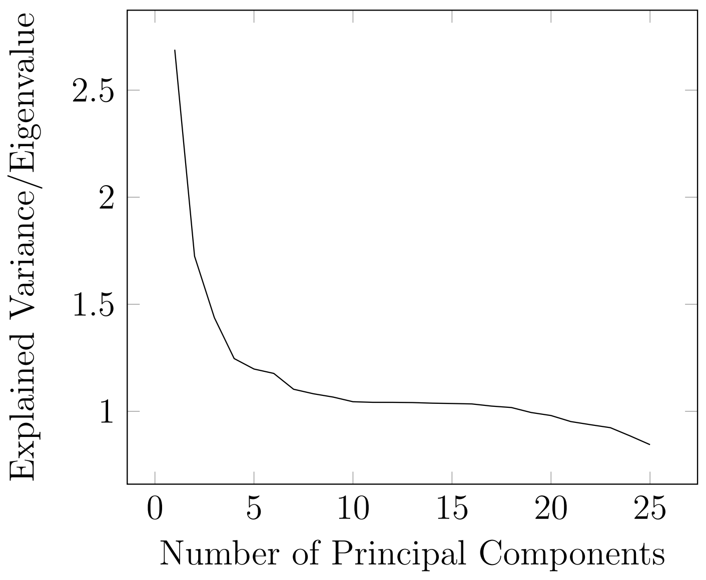


Figure 21: Scree plot from PCA on correlation matrix

In Figure 21, principal components up to the 25th were selected to plot, as beyond this number, the explained variance was negligible from the remaining PCs. The plot is useful in establishing an initial idea as for an appropriate number of principal components to retain in further analysis.

A criterion is then established to determine the number of PCs kept in the data. The requirement is to reduce dimensions of the data up to the point that the retained principal components explain 90% of the variance in the data. In order to determine the number of PCs that achieve this, figure 22 is consulted.

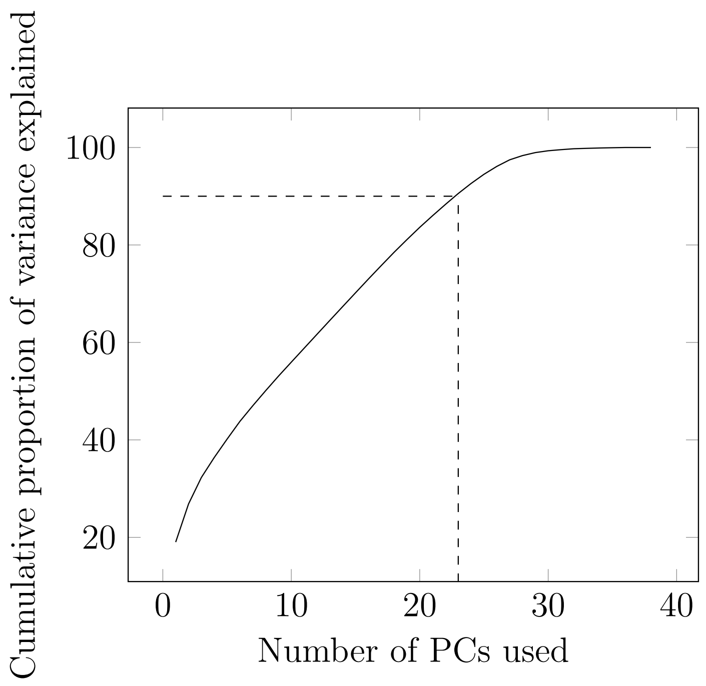


Figure 22: Cumulative proportion of variance explained

This figure provides a visual representation of the total amount of variance explained by the PCs as the number of PCs retained in the model increases. A cumulative measure is used to capture this notion. Therefore, in order to explain 90% of the variation in the data, 23 principal components must be kept.

The model is then trained on the first 23 PCs (as the PCs are given in descending order, with those having the highest eigenvalues ranking first, then the PC with the second highest eigenvalue ranking second, and so on). As will be explored in the Model Evaluation and Results section, whilst the dimensional reduction is successful by way of easing computational strain, the performance of the model does not see meaningful improvement in terms of evaluation on the training set. In the following portion of the study, the machine learning model is officially established, as well as the justifications for choosing the model.

3.3 Model Development

3.3.1 Random Forests for Regression

Random Forests consist of a multiple of decision trees, whose training algorithm applies the general technique of both bagging towards trees and feature bagging towards features (Mou et al., 2019) [9]. Since the goal of this section of the study is to predict the continuous delay time of a train, the regression method was used. For a regression task, the final result is returned by averaging the output of each regression tree. Such an advanced method can to some extent handle the overfitting problems in decision trees.

3.3.2 Reason for choosing Random Forest

According to Breiman (2001) [10], combining the output of a few classifiers outperforms using any separate one of them. Moreover, Random Forests have proven to be one of the most effective tools under this condition (Fernández-Delgado, 2014) [11]. They often behave well in high-dimensional settings. When it comes to this specific challenge on train-delay prediction, Random Forests are also the most commonly-used method in literature to handle such a problem, as reviewed by Tiong et al. (2023) [7]. An example might explain for the reason. A study of a specific train-delays prediction problem in Italy (Oneto et al., 2016) [12] demonstrated that the performance of Random Forests surpassed the other two methods (Extreme Learning Machines and Kernel Regularized Least Square) when regarding the train-delay prediction problem as a multivariate regression one.

3.3.3 Variable Importance

There is an important property in Random Forests called variable importance, which shows the impact of variables on predictions. Through this property, the importance of variables can be ranked. Accordingly, variable selection can be conducted based on variable importance and correlation (Gregorutti, 2017) [13]. For a regression task, the percentage of increase of Mean Square Error (%IncMSE) and increase in Node Purity (IncNodePurity) is the default implementation in `R`. The percentage of increase of Mean Square Error randomly assigns a value to each variable. If that variable is more important, then the error of the prediction of the model will increase when its value is randomly replaced. Increase in Node Purity is another measurement of variable importance, depended on the Gini impurity index used for split nodes in trees. All in all, the higher the value of Mean Decrease Accuracy or Mean Decrease Gini, the higher the importance of that variable to a model. In general, if there are a large number of categorical variables in the dataset, IncNodePurity might be more suitable; if the dataset mainly contains continuous variables, then %IncMSE might be more suitable. However, both measures have their limitations and cannot fully represent the influence of variables to the prediction.

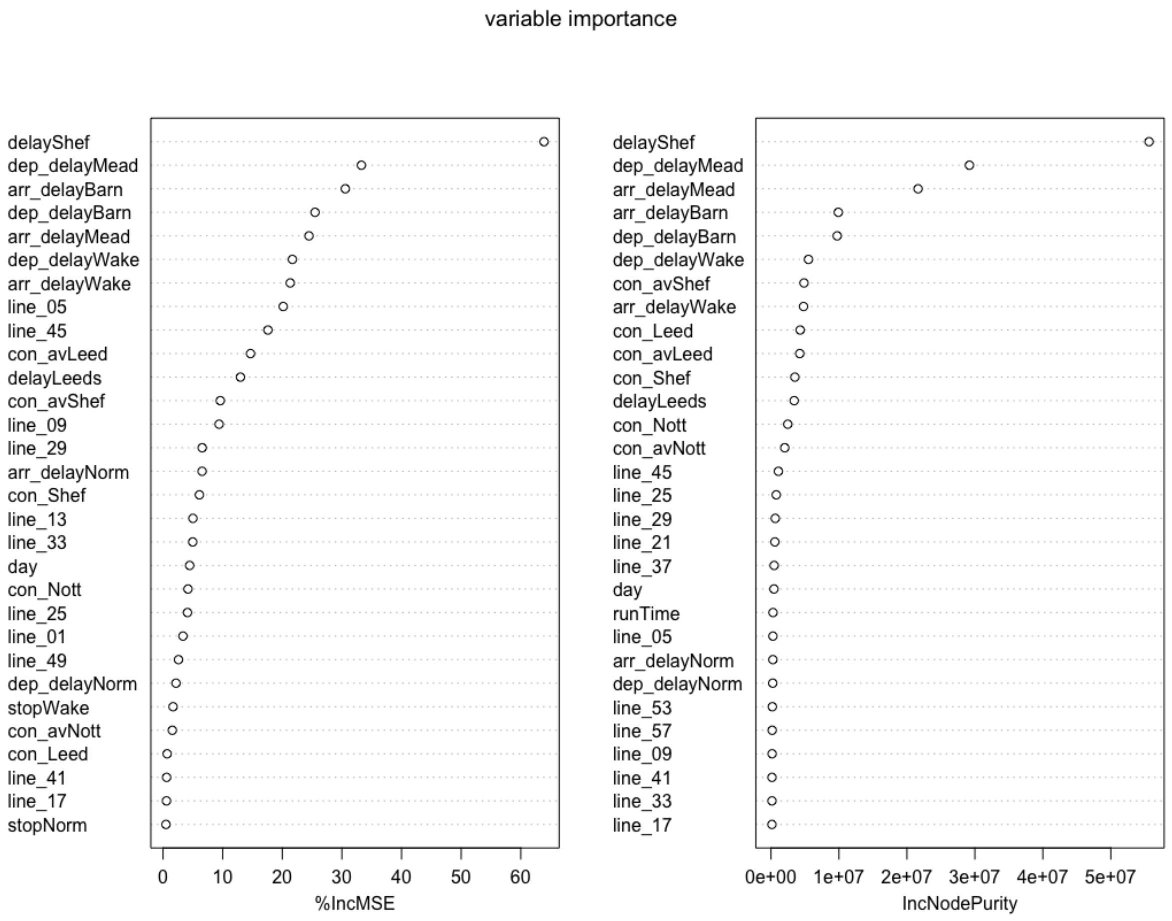


Figure 23: Figure for Variable Importance of Train - Delay Model

Using `importance()` and `varImpPlot()` function to our constructed model can get value of each variable importance and the figure for variable importance (Figure 23). Conclusions can be drawn that variables such as Train Code '1M01' and whether a train will stop at certain station have little impact on predictions, whereas the delay time in Sheffield has the biggest impact.

3.3.4 Cross Validation

There existed an overfitting problem in our Random Forest model. To improve this situation, turning one hyperparameter which controls the number of features selected randomly to train each tree was needed. `Mtry`, the number of variables to randomly sample as candidates at each split, is such kind of tuning parameter. `tuneRF()` function can tune `randomForest()` for the optimal `mtry` parameter automatically. However, this function might produce off-target results, possibly due to different response to `set.seed()` function for `tuneRF()` and `randomForest()`. Accordingly, code using `for` loop can be applied to adjust for a more suitable value of `mtry` and thus test the best performance. Additionally, a resampling technique called 'k - Fold Cross Validation' helps in comparing and selecting an optimized model by finding an appropriate value of `mtry` parameter, thus preventing the overfitting of the training dataset. As mentioned in the ECGs section, the value of k is optional, aiming to avoid high variance and bias. The choice of k is typically 5 or 10 in most cases when facing such kind of problems, and this report will use $k = 10$ because Kohavi (1995) [14] conducted a detailed survey on this field and finally recommended using 10 - fold Cross Validation. That study also pointed out that the variance of cross validation estimates would remain almost the same regardless of k values if the induction algorithm was stable for a given dataset. Cross validation was an essential component in detecting overfitting, which bears a potential impact on the model results. As such, the model evaluation and results are explored in the next section, highlighting the importance of cross validation.

3.4 Model Evaluation and Results

The model was studied and continuously evolved in complexity over a 5 - week period, with results computed at the end of each week. The primary evaluation metric used was the Mean Squared Error (MSE) of the predictions. Below in Table 4 is a summary of the results, demonstrating how the model performs on the test data:

Week	Model description	MSE
1	"Burn in" model	105,329
2	Base Random Forest	30,484
3	Random Forest with more features	30,821
4	Tuned Random Forest	29,803
5	Random Forest without outliers	31,219

Table 4: Week - by - week model evolution and performance on test data

Week 1 offered high error rates as the model was not properly constructed and calibrated, and so can be considered a 'burn in' week. In the second week, the model was constructed, with some preliminary basic features, as a random forest regression model. The model performance improved significantly, and the MSE reduced by 74,845. In week 3, further features were added to the Random Forest model. The model was tuned via hyperparameter optimisation, and overfitting to the training set was detected by cross - validation, and this improved performance in week 4. Finally, in week 5, the MSE rose slightly due to the removal of 3 outliers, which potentially may have been influential datum. Therefore, the removal of the outliers actually reduced model performance. In retrospect, utilising quantitative measures such as leverage, cook's distance and the hat matrix, for instance, would provide more substantive conclusions as for the determining the influence of these data points on the fitted model.

With regards to training data (with 3 outliers included) performance, the results are captured below in Table 5:

Model	MSE
Random Forest (final)	29,542
Random Forest (with PCA)	36,837
Support Vector Regression (SVR)	34,730

Table 5: Model performance on training data compared to benchmark model

The model had a relatively low MSE of 29, 542 in its final form, and whilst there was succesful dimensionality reduction via PCA, this did not actual improve the in - sample model performance, as the MSE increased by 7295. It was decided that PCA would not have been an approporiate measure to incorporate into the final model. In order to guage a suitable MSE quantity for this particular problem on this dataset, a Support Vector Regression (SVR) model is utilised as a benchmark model for comparison, and achieved a MSE of 34, 730 on the training data. The model overall performed worse, with regard to in - sample performance, when the 3 outliers were not included in the model, potentially due to their influence.

3.5 Trains: Conclusion and Discussion

In this portion of the study, a Random Forest machine learning model is presented to analyse the relationship between arrival delays to Nottingham station and numerous characteristics which affect this. The initial aim mentioned at the beginning of the study of identifying a candidate machine learning model was achieved, the model being a Random Forest regression model. The model was suitable in capturing the complexities of the underlying structure of the data, and this is evident from its comparative performance to the benchmark SVR model, achieving the second aim.

Upon reflection, challenges in pre - processing of the data was moderate, and largely computational and were to do with design matrix construction. As such, due to the large number of features identified, PCA was utilised, but proved to be unsuccessful in improving the performance of the model, and was not used in the final model. Outliers were also considered in the final model, as they were influential. The model evolved in complexity over a 5 week period, and to detect over fitting cross - validation was successfully utilised.

For future research endeavours, it could have significantly benefited the model development to have access to additional characteristics which affect delay times, to extract increasingly accurate feature variables to enabling most robust prediction results. Such characteristics might have included such as temperature, crew related information, maintenance related information and more. Additional potential factors are captured in Figure

25 in the Appendix.

With regard to future consequences and impact of the study, by modelling predictions of future train delays, commuters will adapt by taking earlier departures. Furthermore, train companies, particularly those operating around the East Midlands region, would obtain a chance to take measures in order to prevent delays as much as possible in high - risk periods, lowering costs for such companies.

4 Conclusion

The two distinct studies presented in this report highlight the versatile application of machine learning methods in approaching solving different types of data - based problems. For analysis of electrocardiogram (ECG) readings to diagnose heart conditions, the implementation of a Random Forest model proved to be an effective measure in predicting patient outcomes, showcasing its strong utility in medical diagnostics. Similarly, in the study of modelling train delays on the Leeds to Nottingham line, the Random Forest regression model demonstrated its efficacy in capturing complex relationships between the numerous factors influencing arrival times. For both studies, challenges encountered in data pre-processing and model development were addressed thoroughly, via innovative approaches, such as feature extraction and outlier detection. Granted, while both studies yielded meaningful results, avenues for further improvement were identified, including the incorporation of additional relevant features and the exploration of different modelling techniques.

Looking ahead, there are plenty of opportunities for extension of the models far beyond their respective domains. As such, the models developed offer valuable tools for medical practitioners in diagnosing heart conditions and for transportation authorities in managing train schedules.

5 Appendices

5.1 ECG Study

5.1.1 Samples from ECG Code

Below is a selection of some of the code used in the ECG study. First is the function used to split the ECG into individual heartbeats after drift had been removed from the ECGs.

```
#loading the data with drift removed
train <- read.csv("ECG_Datatraining.csv")
test <- read.csv("ECG_Datatest.csv")


#function for splitting ECGs
split <- function(t){
  q <- length(x)
  inds <- which(x >= c(x[1],x[1:q-1]) & x > c(x[2:q], x[q]) & x > t)
  #Removing maximums close together
  for(i in 1:(length(inds)-1)){
    if(inds[i+1]-inds[i] < 100){
      inds[i] <- 0
    }
  }
  inds <- inds[inds != 0]
  #line below can be uncomented to see the maximums plotted
  #points(inds,x[inds],col=2)
  m <- length(inds)-1; # number of full heartbeat cycles
  p <- 200; # length we choose for the processed vectors
  X1.new <- matrix(NA,m,p);
  for (m_ in 1:m){
    t.old <- inds[m_]:inds[m_+1]
    x.old <- x[inds[m_]:inds[m_+1]]
    t.new <- seq(inds[m_],inds[m_+1],length=p)
```

```
        x.new <- approx(t.old,x.old,t.new)$y
        X1.new[m_,] <- x.new
    }
    return(X1.new)
}
```

The next section of code was used to create the first predictions based on the variance of the ECGs.

```
#loading data
load("ECG.RData")

#calculating variance for test and train sets
vars.train <- numeric(115)
for(i in 1:115){
    vars.train[i] <- var(X.train[i,])
}

vars.test <- numeric(100)
for(i in 1:100){
    vars.test[i] <- var(X.test[i,])
}

#creating model
model <- rpart(classification ~ data,
               data = data.frame(data = vars.train,
                                 classification = as.factor(y.train),
               method = "class" ))

#creating predictions
predictions <- predict(model,
                       data.frame(data = vars.test),
                                        type = "class")
```

Finally the code to create the final predictions is shown. The data frames loaded here have been created using the methods described in section 2.2.

```
#load data frames
data_df <- read.csv("train_df.csv")
test_df <- read.csv("test_df.csv")

#putting data frames in correct format


data_df2 <- data_df[, !names(data_df) %in% "X"]
data_df2$y.train <- as.factor(data_df2$y.train)
names(test_df)[2:4] <- names(data_df2)[2:4]



#creating model
model <- randomForest(y.train ~ qrsint.train + t_inversion.train,
                            data = data_df2,
                            ntree = 22)

#creating predictions
predictions <- predict(model,
                            newdata = test_df,
                            type = "class")
```

5.2 Trains Study

5.2.1 Exploratory Data Analysis (EDA)

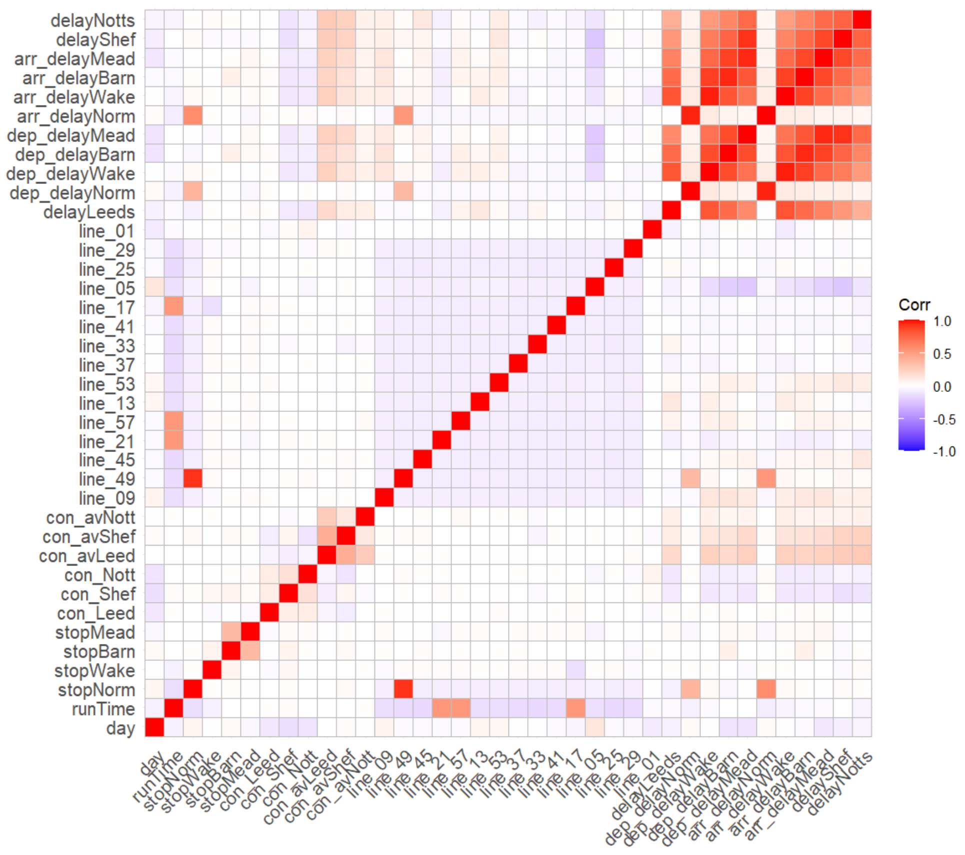


Figure 24: Correlation plot (heatmap) of features

5.2.2 Challenges with the data: the number of stations

```
load("trainsData.RData")
matrixData <- data.frame(
  day=rep("-", length(trainingData)),
  stringsAsFactors=FALSE
)
for (i in 1:length(trainingData)) {
  dummy <- trainingData[[i]]
  matrixData$day[i] <- length(dummy$timings$day.week)
}; head(matrixData)
max(matrixData$day)
min(matrixData$day)
```

5.2.3 Challenges with the data: Train Code

```
codes <- numeric(length=length(trainingData))
for (i in 1:length(trainingData)){
  codes[i] <- trainingData[[i]]$timings$train.code[1]
}
unique_training_trains <- unique(codes)

testcodes <- numeric(length=length(testData))
for (i in 1:length(testData)){
  testcodes[i] <- testData[[i]]$timings$train.code[1]
}
unique_test_trains <- unique(testcodes)
```

5.2.4 Principal Component Analysis, as an optimisation problem

Let $\mathbf{x}_1, \ldots, \mathbf{x}_n$ denote a sample of vectors in $\mathbb{R}^p$ with sample mean vector $\overline{\mathbf{x}}$ and sample covariance matrix $\mathbf{S}$. Suppose $\mathbf{S} = \frac{1}{n}\mathbf{X}^\top\mathbf{H}\mathbf{X}$ has spectral decomposition

$$\mathbf{S} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^\top = \sum_{j=1}^{p} \lambda_j \mathbf{v}_j \mathbf{v}_j^\top,$$

where the eigenvalues are $\lambda_1 \geq \lambda_2 \geq \lambda_p \geq 0$ with $\mathbf{\Lambda} = \operatorname{diag}\{\lambda_1, \ldots, \lambda_p\}$, and $\mathbf{V}$ contains the eigenvectors of $\mathbf{S}$. If $\mathbf{S}$ is of full rank, then $\lambda_p > 0$. If $\mathbf{S}$ is rank r, with $r < p$, then $\lambda_{r+1} = \ldots \lambda_p = 0$ and we can truncate $\mathbf{V}$ to consider just the first r columns.

The principal components of $\mathbf{X}$ are defined sequentially. If $\mathbf{v}_k$ is the value of $\mathbf{u}$ that maximises the objective for the k^{th} problem (for $k < j$), then the j^{th} principal component is the solution to the following optimisation problem:

$$\max_{\mathbf{u}:\|\mathbf{u}\|=1} \mathbf{u}^\top \mathbf{S} \mathbf{u}$$

subject to

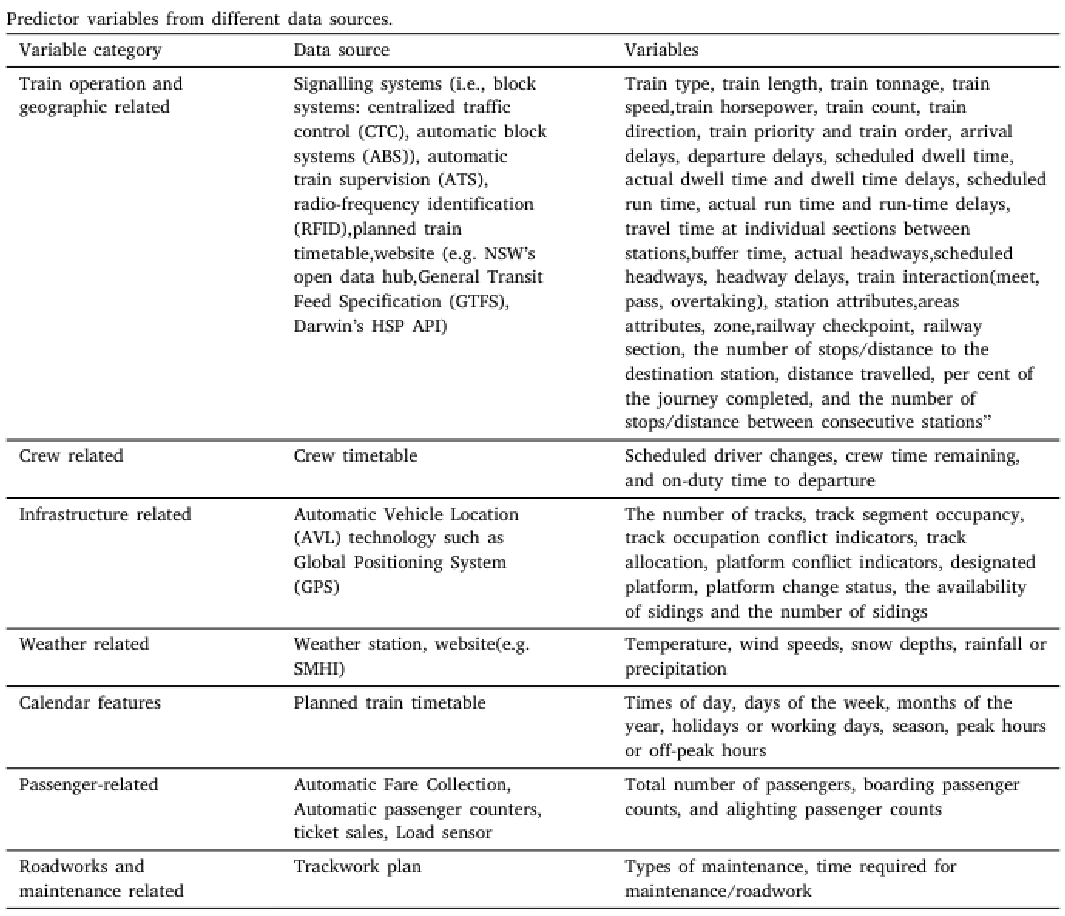
$$\mathbf{v}_k^\top \mathbf{u} = 0, \quad k = 1, \ldots, j-1.$$

(for $j = 1$ there does not exist a orthogonality constraint).

5.2.5 Feature extraction

Features			
day	con_avShef	line_33	dep_delayBarn
runTime	con_avNott	line_41	dep_delayMead
stopNorm	line_09	line_17	arr_delayNorm
stopWake	line_49	line_05	arr_delayWake
stopBarn	line_45	line_25	arr_delayBarn
stopMead	line_21	line_29	arr_delayMead
con_Leed	line_57	line_01	delayShef
con_Shef	line_13	delayLeeds	delayNotts
con_Nott	line_53	dep_delayNorm	
con_avLeed	line_37	dep_delayWake	

Table 6: Features (38) used to train RF model on trains data



Predictor variables from different data sources.

Variable category	Data source	Variables
Train operation and geographic related	Signalling systems (i.e., block systems: centralized traffic control (CTC), automatic block systems (ABS)), automatic train supervision (ATS), radio-frequency identification (RFID),planned train timetable,website (e.g. NSW's open data hub,General Transit Feed Specification (GTFS), Darwin's HSP API)	Train type, train length, train tonnage, train speed,train horsepower, train count, train direction, train priority and train order, arrival delays, departure delays, scheduled dwell time, actual dwell time and dwell time delays, scheduled run time, actual run time and run-time delays, travel time at individual sections between stations,buffer time, actual headways,scheduled headways, headway delays, train interaction(meet, pass, overtaking), station attributes,areas attributes, zone,railway checkpoint, railway section, the number of stops/distance to the destination station, distance travelled, per cent of the journey completed, and the number of stops/distance between consecutive stations"
Crew related	Crew timetable	Scheduled driver changes, crew time remaining, and on-duty time to departure
Infrastructure related	Automatic Vehicle Location (AVL) technology such as Global Positioning System (GPS)	The number of tracks, track segment occupancy, track occupation conflict indicators, track allocation, platform conflict indicators, designated platform, platform change status, the availability of sidings and the number of sidings
Weather related	Weather station, website(e.g. SMHI)	Temperature, wind speeds, snow depths, rainfall or precipitation
Calendar features	Planned train timetable	Times of day, days of the week, months of the year, holidays or working days, season, peak hours or off-peak hours
Passenger-related	Automatic Fare Collection, Automatic passenger counters, ticket sales, Load sensor	Total number of passengers, boarding passenger counts, and alighting passenger counts
Roadworks and maintenance related	Trackwork plan	Types of maintenance, time required for maintenance/roadwork

Figure 25: Predictor variables from different data sources (Tiong, Ma, Palmqvist, 2023) [7]

5.3 Entire code for predicting train delays

5.3.1 Matrix for training data

```
# Load data
load("trainsData.RData")


# Function to convert text time to seconds
textToSeconds <- function(textTime) {
  if (textTime == "0"){
    x = 0
  }
  else {
    seconds <- as.numeric(strsplit(textTime, split = ":")[[1]]) %*% c(60*60, 60, 1)
    x = as.numeric(seconds)
  }
  return(x)
}


# Function to categorize day into weekday or weekend (binary)
dayConvert <- function(day){
  y = 0
  # 1 if the day is weekday
  if (day == "Monday" | day == "Tuesday" | day == "Wednesday" | day == "Thursday" | day == "Friday"){
    y = 1
  }
  return(y)
}


# Function to categorize run time of a train into peak or off-peak (binary)
runTimeConvert <- function(time){
  z = 0
  # 1 if run time is peak
  if ((time>=10 & time<=11) | (time>=20 & time<=21)){
    z = 1
  }
  return(z)
}


# Functions to convert the stops - it will check if a train stops at a particular stop.
NormStopConvert <- function(stops){
  x1 = 0
  if ("NORMNTN" %in% stops){
    x1 = 1
  }
  return(x1)
}

WakeStopConvert <- function(stops){
  x2 = 0
  if ("WKFLDKG" %in% stops){
    x2 = 1
  }
  return(x2)
}

BarnStopConvert <- function(stops){
  x3 = 0
  if ("BNSLY" %in% stops){
    x3 = 1
  }
  return(x3)
}

MeadStopConvert <- function(stops){
  x4 = 0
  if ("MEADWHL" %in% stops){
    x4 = 1
  }
  return(x4)
}


# Exploring train codes:
codes <- numeric(length=length(trainingData))
for (i in 1:length(trainingData)){
  codes[i] <- trainingData[[i]]$timings$train.code[1]
}

unique_training_trains <- unique(codes)


# Functions that returns 1 or 0 to train depending on the line it runs on
code_09 <- function(i){
  g1 <- 0
  tr_line <- trainingData[[i]]$timings$train.code[1]
  if (tr_line == "1Y09"){
    g1 <- 1
  }
  return(g1)
}

code_49 <- function(i){
  g2 <- 0
  tr_line <- trainingData[[i]]$timings$train.code[1]
  if (tr_line == "1Y49"){
    g2 <- 1
  }
  return(g2)
}

code_45 <- function(i){
  g3 <- 0
  tr_line <- trainingData[[i]]$timings$train.code[1]
  if (tr_line == "1Y45"){
    g3 <- 1
  }
  return(g3)
}

code_21 <- function(i){
  g4 <- 0
  tr_line <- trainingData[[i]]$timings$train.code[1]
  if (tr_line == "1Y21"){
    g4 <- 1
  }
  return(g4)
}
```

```
code_57 <- function(i){
  g5 <- 0
  tr_line <- trainingData[[i]]$timings$train.code[1]
  if (tr_line == "1Y57"){
    g5 <- 1
  }
  return(g5)
}

code_13 <- function(i){
  g6 <- 0
  tr_line <- trainingData[[i]]$timings$train.code[1]
  if (tr_line == "1Y13"){
    g6 <- 1
  }
  return(g6)
}

code_53 <- function(i){
  g7 <- 0
  tr_line <- trainingData[[i]]$timings$train.code[1]
  if (tr_line == "1Y53"){
    g7 <- 1
  }
  return(g7)

}

code_37 <- function(i){
  g8 <- 0
  tr_line <- trainingData[[i]]$timings$train.code[1]
  if (tr_line == "1Y37"){
    g8 <- 1
  }
  return(g8)
}

code_33 <- function(i){
  g9 <- 0
  tr_line <- trainingData[[i]]$timings$train.code[1]
  if (tr_line == "1Y33"){
    g9 <- 1
  }
  return(g9)
}

code_41 <- function(i){
  g10 <- 0
  tr_line <- trainingData[[i]]$timings$train.code[1]
  if (tr_line == "1Y41"){
    g10 <- 1
  }
  return(g10)
}

code_17 <- function(i){
  g11 <- 0
  tr_line <- trainingData[[i]]$timings$train.code[1]
  if (tr_line == "1Y17"){
    g11 <- 1
  }
  return(g11)
```

```
}

code_05 <- function(i){
  g12 <- 0
  tr_line <- trainingData[[i]]$timings$train.code[1]
  if (tr_line == "1Y05"){
    g12 <- 1
  }
  return(g12)
}

code_25 <- function(i){
  g13 <- 0
  tr_line <- trainingData[[i]]$timings$train.code[1]
  if (tr_line == "1Y25"){
    g13 <- 1
  }
  return(g13)
}

code_29 <- function(i){
  g14 <- 0
  tr_line <- trainingData[[i]]$timings$train.code[1]
  if (tr_line == "1Y29"){
    g14 <- 1
  }
  return(g14)
}

code_01 <- function(i){
  g15 <- 0
  tr_line <- trainingData[[i]]$timings$train.code[1]
  if (tr_line == "1M01"){
    g15 <- 1
  }
  return(g15)
}


# This is the initialisation of the design matrix
matrixData <- data.frame(
  # Day
  day=rep(0, length(trainingData)),

  # Run time
  runTime=rep(0, length(trainingData)),

  # Stops
  stopNorm=rep(0, length(trainingData)),
  stopWake=rep(0, length(trainingData)),
  stopBarn=rep(0, length(trainingData)),
  stopMead=rep(0, length(trainingData)),

  # Congestions
  con_Leed=rep(0, length(trainingData)),
  con_Shef=rep(0, length(trainingData)),
  con_Nott=rep(0, length(trainingData)),
  con_avLeed=rep(0, length(trainingData)),
  con_avShef=rep(0, length(trainingData)),
  con_avNott=rep(0, length(trainingData)),

  # Train codes
  line_09 = rep(0, length(trainingData)),
  line_49 = rep(0, length(trainingData)),
  line_45 = rep(0, length(trainingData)),
  line_21 = rep(0, length(trainingData)),
  line_57 = rep(0, length(trainingData)),
  line_13 = rep(0, length(trainingData)),
  line_53 = rep(0, length(trainingData)),
  line_37 = rep(0, length(trainingData)),
  line_33 = rep(0, length(trainingData)),
  line_41 = rep(0, length(trainingData)),
  line_17 = rep(0, length(trainingData)),
  line_05 = rep(0, length(trainingData)),
  line_25 = rep(0, length(trainingData)),
  line_29 = rep(0, length(trainingData)),
  line_01 = rep(0, length(trainingData)),

  # Departure delays
  delayLeeds=rep(0, length(trainingData)),
  dep_delayNorm=rep(0, length(trainingData)),
  dep_delayWake=rep(0, length(trainingData)),
  dep_delayBarn=rep(0, length(trainingData)),
  dep_delayMead=rep(0, length(trainingData)),

  # Arrival delays
  arr_delayNorm=rep(0, length(trainingData)),
  arr_delayWake=rep(0, length(trainingData)),
  arr_delayBarn=rep(0, length(trainingData)),
  arr_delayMead=rep(0, length(trainingData)),
  delayShef=rep(0, length(trainingData)),
  delayNotts=rep(0, length(trainingData)),

  stringsAsFactors=FALSE
)


# Stop Condition Converter:
# This function returns 0 for the departure/arrival time if the train does not stop at the particular station.
stopCondConv <- function(t){
  if (identical(t, character(0)) == TRUE){
    y = 0
  } else {
    y = t
  }
  return(y)
}

for (i in 1:length(trainingData)) {
  dp <- trainingData[[i]]

  # DAY
  matrixData$day[i] <- dayConvert(dp$timings$day.week[1])


  # RUN TIME
  matrixData$runTime[i] <- runTimeConvert(dp$congestion$hour)


  # CHECKING THE STOPS
  matrixData$stopNorm[i] <- NormStopConvert(dp$timings$departure.from)
  matrixData$stopWake[i] <- WakeStopConvert(dp$timings$departure.from)
  matrixData$stopBarn[i] <- BarnStopConvert(dp$timings$departure.from)
  matrixData$stopMead[i] <- MeadStopConvert(dp$timings$departure.from)
```

```
# CONGESTIONS
for (j in 1:length(historicalCongestion$Day)) {
  if ((historicalCongestion$Day[j]==dp$congestion$week.day) & (historicalCongestion$Hour[j]==dp$congestion$hour)) {
    matrixData$con_Leed[i] <- -historicalCongestion$Leeds.trains[j] + dp$congestion$Leeds.trains
    matrixData$con_Shef[i] <- -historicalCongestion$Sheffield.trains[j] + dp$congestion$Sheffield.trains
    matrixData$con_Nott[i] <- -historicalCongestion$Nottingham.trains[j] + dp$congestion$Nottingham.trains
    matrixData$con_avLeed[i] <- -historicalCongestion$Leeds.av.delay[j] + dp$congestion$Leeds.av.delay
    matrixData$con_avShef[i] <- -historicalCongestion$Sheffield.av.delay[j] + dp$congestion$Sheffield.av.delay
    matrixData$con_avNott[i] <- -historicalCongestion$Nottingham.av.delay[j] + dp$congestion$Nottingham.av.delay
    break
  }
}


# TRAIN CODES
matrixData$line_09[i] <- code_09(i)
matrixData$line_49[i] <- code_49(i)
matrixData$line_45[i] <- code_45(i)
matrixData$line_21[i] <- code_21(i)
matrixData$line_57[i] <- code_57(i)
matrixData$line_13[i] <- code_13(i)
matrixData$line_53[i] <- code_53(i)
matrixData$line_37[i] <- code_37(i)
matrixData$line_33[i] <- code_33(i)
matrixData$line_41[i] <- code_41(i)
matrixData$line_17[i] <- code_17(i)
matrixData$line_05[i] <- code_05(i)
matrixData$line_25[i] <- code_25(i)
matrixData$line_29[i] <- code_29(i)
matrixData$line_01[i] <- code_01(i)


# Departure delay (for Leeds)
depLeeds<- textToSeconds(head(dp$timings$departure.time,1))
schLeeds <- textToSeconds(head(dp$timings$departure.schedule,1))
matrixData$delayLeeds[i] <- depLeeds - schLeeds

# Arrival Delay (for Sheffield)
arrSheffield <- textToSeconds(tail(dp$timings$arrival.time,1))
schSheffield <- textToSeconds(tail(dp$timings$arrival.schedule,1))
matrixData$delayShef[i] <- arrSheffield - schSheffield

# Arrival Delay (for Notts)
matrixData$delayNotts[i] <- dp$arrival$delay.secs[1]

# DEPARTURES

# Actual departure times
dept_Norm <- dp$timings$departure.time[dp$timings$departure.from == "NORMNTN"]
dept_Wake <- dp$timings$departure.time[dp$timings$departure.from == "WKFLDKG"]
dept_Barn <- dp$timings$departure.time[dp$timings$departure.from == "BNSLY"]
dept_Mead <- dp$timings$departure.time[dp$timings$departure.from == "MEADWHL"]

dept <- c(stopCondConv(dept_Norm), stopCondConv(dept_Wake), stopCondConv(dept_Barn), stopCondConv(dept_Mead))
dept_seconds <- unlist(lapply(dept, textToSeconds))

# Scheduled departure times
schNorm <- dp$timings$departure.schedule[dp$timings$departure.from == "NORMNTN"]
schWake <- dp$timings$departure.schedule[dp$timings$departure.from == "WKFLDKG"]
schBarn <- dp$timings$departure.schedule[dp$timings$departure.from == "BNSLY"]
```

```
  schMead <- dp$timings$departure.schedule[dp$timings$departure.from == "MEADWHL"]

  sch <- c(stopCondConv(schNorm), stopCondConv(schWake), stopCondConv(schBarn), stopCondConv(schMead))
  sch_seconds <- unlist(lapply(sch, textToSeconds))

  # Departure delays
  matrixData$dep_delayNorm[i] <- dept_seconds[1] - sch_seconds[1]
  matrixData$dep_delayWake[i] <- dept_seconds[2] - sch_seconds[2]
  matrixData$dep_delayBarn[i] <- dept_seconds[3] - sch_seconds[3]
  matrixData$dep_delayMead[i] <- dept_seconds[4] - sch_seconds[4]


  # ARRIVALS

  # Actual arrival times
  arrt_Norm <- dp$timings$arrival.time[dp$timings$arrival.to == "NORMNTN"]
  arrt_Wake <- dp$timings$arrival.time[dp$timings$arrival.to == "WKFLDKG"]
  arrt_Barn <- dp$timings$arrival.time[dp$timings$arrival.to == "BNSLY"]
  arrt_Mead <- dp$timings$arrival.time[dp$timings$arrival.to == "MEADWHL"]

  arrt <- c(stopCondConv(arrt_Norm), stopCondConv(arrt_Wake), stopCondConv(arrt_Barn), stopCondConv(arrt_Mead))
  arrt_seconds <- unlist(lapply(arrt, textToSeconds))

  # Scheduled arrival times
  scha_Norm <- dp$timings$arrival.schedule[dp$timings$arrival.to == "NORMNTN"]
  scha_Wake <- dp$timings$arrival.schedule[dp$timings$arrival.to == "WKFLDKG"]
  scha_Barn <- dp$timings$arrival.schedule[dp$timings$arrival.to == "BNSLY"]
  scha_Mead <- dp$timings$arrival.schedule[dp$timings$arrival.to == "MEADWHL"]

  scha <- c(stopCondConv(scha_Norm),  stopCondConv(scha_Wake), stopCondConv(scha_Barn), stopCondConv(scha_Mead))
  scha_seconds <- unlist(lapply(scha, textToSeconds))

  # Arrival delays
  matrixData$arr_delayNorm[i] <- arrt_seconds[1] - scha_seconds[1]
  matrixData$arr_delayWake[i] <- arrt_seconds[2] - scha_seconds[2]
  matrixData$arr_delayBarn[i] <- arrt_seconds[3] - scha_seconds[3]
  matrixData$arr_delayMead[i] <- arrt_seconds[4] - scha_seconds[4]
}


# Save the design matrix to a file
save(matrixData, file="DesignMatrix.RData")
```

5.3.2 Matrix for test data

```
# Load data
#load("trainsData.RData")



# Function to convert text time to seconds
textToSeconds <- function(textTime) {
  if (textTime == "0"){
    x = 0
  }
  else {
    seconds <- as.numeric(strsplit(textTime, split = ":")[[1]]) %*% c(60*60, 60, 1)
    x = as.numeric(seconds)
  }
  return(x)
}
```

```
# Function to categorize day into weekday or weekend (binary)
dayConvert <- function(day){
  y = 0
  # 1 if the day is weekday
  if (day == "Monday" | day == "Tuesday" | day == "Wednesday" | day == "Thursday" | day == "Friday"){
    y = 1
  }
  return(y)
}


# Function to categorize run time of a train into peak or off-peak (binary)
runTimeConvert <- function(time){
  z = 0
  # 1 if run time is peak
  if ((time>=10 & time<=11) | (time>=20 & time<=21)){
    z = 1
  }
  return(z)
}


# Functions to convert the stops - it will check if a train stops at a particular stop.
NormStopConvert <- function(stops){
  x1 = 0
  if ("NORMNTN" %in% stops){
    x1 = 1
  }
  return(x1)
}

WakeStopConvert <- function(stops){
  x2 = 0
  if ("WKFLDKG" %in% stops){
    x2 = 1
  }
  return(x2)
}

BarnStopConvert <- function(stops){
  x3 = 0
  if ("BNSLY" %in% stops){
    x3 = 1
  }
  return(x3)
}

MeadStopConvert <- function(stops){
  x4 = 0
  if ("MEADWHL" %in% stops){
    x4 = 1
  }
  return(x4)
}


# Exploring train codes:
testcodes <- numeric(length=length(testData))
for (i in 1:length(testData)){
  testcodes[i] <- testData[[i]]$timings$train.code[1]
}
```

```
unique_test_trains <- unique(testcodes)


# Functions that returns 1 or 0 to train depending on the line it runs on
code_09 <- function(i){
  g1 <- 0
  tr_line <- testData[[i]]$timings$train.code[1]
  if (tr_line == "1Y09"){
    g1 <- 1
  }
  return(g1)
}

code_49 <- function(i){
  g2 <- 0
  tr_line <- testData[[i]]$timings$train.code[1]
  if (tr_line == "1Y49"){
    g2 <- 1
  }
  return(g2)
}

code_45 <- function(i){
  g3 <- 0
  tr_line <- testData[[i]]$timings$train.code[1]
  if (tr_line == "1Y45"){
    g3 <- 1
  }
  return(g3)
}

code_21 <- function(i){
  g4 <- 0
  tr_line <- testData[[i]]$timings$train.code[1]
  if (tr_line == "1Y21"){
    g4 <- 1
  }
  return(g4)
}

code_57 <- function(i){
  g5 <- 0
  tr_line <- testData[[i]]$timings$train.code[1]
  if (tr_line == "1Y57"){
    g5 <- 1
  }
  return(g5)
}

code_13 <- function(i){
  g6 <- 0
  tr_line <- testData[[i]]$timings$train.code[1]
  if (tr_line == "1Y13"){
    g6 <- 1
  }
  return(g6)
}

code_53 <- function(i){
  g7 <- 0
  tr_line <- testData[[i]]$timings$train.code[1]
  if (tr_line == "1Y53"){
    g7 <- 1
  }
  return(g7)
}

code_37 <- function(i){
  g8 <- 0
  tr_line <- testData[[i]]$timings$train.code[1]
  if (tr_line == "1Y37"){
    g8 <- 1
  }
  return(g8)
}

code_33 <- function(i){
  g9 <- 0
  tr_line <- testData[[i]]$timings$train.code[1]
  if (tr_line == "1Y33"){
    g9 <- 1
  }
  return(g9)
}

code_41 <- function(i){
  g10 <- 0
  tr_line <- testData[[i]]$timings$train.code[1]
  if (tr_line == "1Y41"){
    g10 <- 1
  }
  return(g10)
}

code_17 <- function(i){
  g11 <- 0
  tr_line <- testData[[i]]$timings$train.code[1]
  if (tr_line == "1Y17"){
    g11 <- 1
  }
  return(g11)
}

code_05 <- function(i){
  g12 <- 0
  tr_line <- testData[[i]]$timings$train.code[1]
  if (tr_line == "1Y05"){
    g12 <- 1
  }
  return(g12)
}

code_25 <- function(i){
  g13 <- 0
  tr_line <- testData[[i]]$timings$train.code[1]
  if (tr_line == "1Y25"){
    g13 <- 1
  }
  return(g13)
}

code_29 <- function(i){
  g14 <- 0
  tr_line <- testData[[i]]$timings$train.code[1]
  if (tr_line == "1Y29"){
    g14 <- 1
  }
  return(g14)
}

code_01 <- function(i){
  g15 <- 0
  tr_line <- testData[[i]]$timings$train.code[1]
  if (tr_line == "1G01"){
    g15 <- 1
  }
  return(g15)
}


# This is the initialisation of the design matrix
data_test <- data.frame(
  # Day
  day=rep(0, length(testData)),

  # Run time
  runTime=rep(0, length(testData)),

  # Stops
  stopNorm=rep(0, length(testData)),
  stopWake=rep(0, length(testData)),
  stopBarn=rep(0, length(testData)),
  stopMead=rep(0, length(testData)),

  # Congestions
  con_Leed=rep(0, length(testData)),
  con_Shef=rep(0, length(testData)),
  con_Nott=rep(0, length(testData)),
  con_avLeed=rep(0, length(testData)),
  con_avShef=rep(0, length(testData)),
  con_avNott=rep(0, length(testData)),

  # Train codes
  line_09 = rep(0, length(testData)),
  line_49 = rep(0, length(testData)),
  line_45 = rep(0, length(testData)),
  line_21 = rep(0, length(testData)),
  line_57 = rep(0, length(testData)),
  line_13 = rep(0, length(testData)),
  line_53 = rep(0, length(testData)),
  line_37 = rep(0, length(testData)),
  line_33 = rep(0, length(testData)),
  line_41 = rep(0, length(testData)),
  line_17 = rep(0, length(testData)),
  line_05 = rep(0, length(testData)),
  line_25 = rep(0, length(testData)),
  line_29 = rep(0, length(testData)),
  line_01 = rep(0, length(testData)),

  # Departure delays
  delayLeeds=rep(0, length(testData)),
  dep_delayNorm=rep(0, length(testData)),
  dep_delayWake=rep(0, length(testData)),
  dep_delayBarn=rep(0, length(testData)),
  dep_delayMead=rep(0, length(testData)),
  # Arrival delays
  arr_delayNorm=rep(0, length(testData)),
  arr_delayWake=rep(0, length(testData)),
  arr_delayBarn=rep(0, length(testData)),
  arr_delayMead=rep(0, length(testData)),
  delayShef=rep(0, length(testData)),

  stringsAsFactors=FALSE
)


# Stop Condition Converter:
# This function returns 0 for the departure/arrival time if the train does not stop at the particular station.
stopCondConv <- function(t){
  if (identical(t, character(0)) == TRUE){
    y = 0
  } else {
    y = t
  }
  return(y)
}

for (i in 1:length(testData)) {
  tp <- testData[[i]]

  # DAY
  data_test$day[i] <- dayConvert(tp$timings$day.week[1])


  # RUN TIME
  data_test$runTime[i] <- runTimeConvert(tp$congestion$hour)


  # CHECKING THE STOPS
  data_test$stopNorm[i] <- NormStopConvert(tp$timings$departure.from)
  data_test$stopWake[i] <- WakeStopConvert(tp$timings$departure.from)
  data_test$stopBarn[i] <- BarnStopConvert(tp$timings$departure.from)
  data_test$stopMead[i] <- MeadStopConvert(tp$timings$departure.from)


  # CONGESTIONS
  for (j in 1:length(historicalCongestion$Day)) {
    if ((historicalCongestion$Day[j]==tp$congestion$week.day) & (historicalCongestion$Hour[j]==tp$congestion$hour)) {
      data_test$con_Leed[i] <- -historicalCongestion$Leeds.trains[j] + tp$congestion$Leeds.trains
      data_test$con_Shef[i] <- -historicalCongestion$Sheffield.trains[j] + tp$congestion$Sheffield.trains
      data_test$con_Nott[i] <- -historicalCongestion$Nottingham.trains[j] + tp$congestion$Nottingham.trains
      data_test$con_avLeed[i] <- -historicalCongestion$Leeds.av.delay[j] + tp$congestion$Leeds.av.delay
      data_test$con_avShef[i] <- -historicalCongestion$Sheffield.av.delay[j] + tp$congestion$Sheffield.av.delay
      data_test$con_avNott[i] <- -historicalCongestion$Nottingham.av.delay[j] + tp$congestion$Nottingham.av.delay
      break
    }
  }


  # TRAIN CODES
  data_test$line_09[i] <- code_09(i)
  data_test$line_49[i] <- code_49(i)
  data_test$line_45[i] <- code_45(i)
  data_test$line_21[i] <- code_21(i)
  data_test$line_57[i] <- code_57(i)
  data_test$line_13[i] <- code_13(i)
  data_test$line_53[i] <- code_53(i)
```

```
data_test$line_37[i] <- code_37(i)
data_test$line_33[i] <- code_33(i)
data_test$line_41[i] <- code_41(i)
data_test$line_17[i] <- code_17(i)
data_test$line_05[i] <- code_05(i)
data_test$line_25[i] <- code_25(i)
data_test$line_29[i] <- code_29(i)
data_test$line_01[i] <- code_01(i)


# Departure delay (for Leeds)
depLeeds<- textToSeconds(head(tp$timings$departure.time,1))
schLeeds <- textToSeconds(head(tp$timings$departure.schedule,1))
data_test$delayLeeds[i] <- depLeeds - schLeeds

# Arrival Delay (for Sheffield)
arrSheffield <- textToSeconds(tail(tp$timings$arrival.time,1))
schSheffield <- textToSeconds(tail(tp$timings$arrival.schedule,1))
data_test$delayShef[i] <- arrSheffield - schSheffield


# DEPARTURES

# Actual departure times
dept_Norm <- tp$timings$departure.time[tp$timings$departure.from == "NORMNTN"]
dept_Wake <- tp$timings$departure.time[tp$timings$departure.from == "WKFLDKG"]
dept_Barn <- tp$timings$departure.time[tp$timings$departure.from == "BNSLY"]
dept_Mead <- tp$timings$departure.time[tp$timings$departure.from == "MEADWHL"]

dept <- c(stopCondConv(dept_Norm), stopCondConv(dept_Wake), stopCondConv(dept_Barn), stopCondConv(dept_Mead))
dept_seconds <- unlist(lapply(dept, textToSeconds))

# Scheduled departure times
schNorm <- tp$timings$departure.schedule[tp$timings$departure.from == "NORMNTN"]
schWake <- tp$timings$departure.schedule[tp$timings$departure.from == "WKFLDKG"]
schBarn <- tp$timings$departure.schedule[tp$timings$departure.from == "BNSLY"]
schMead <- tp$timings$departure.schedule[tp$timings$departure.from == "MEADWHL"]

sch <- c(stopCondConv(schNorm), stopCondConv(schWake), stopCondConv(schBarn), stopCondConv(schMead))
sch_seconds <- unlist(lapply(sch, textToSeconds))

# Departure delays
data_test$dep_delayNorm[i] <- dept_seconds[1] - sch_seconds[1]
data_test$dep_delayWake[i] <- dept_seconds[2] - sch_seconds[2]
data_test$dep_delayBarn[i] <- dept_seconds[3] - sch_seconds[3]
data_test$dep_delayMead[i] <- dept_seconds[4] - sch_seconds[4]


# ARRIVALS

# Actual arrival times
arrt_Norm <- tp$timings$arrival.time[tp$timings$arrival.to == "NORMNTN"]
arrt_Wake <- tp$timings$arrival.time[tp$timings$arrival.to == "WKFLDKG"]
arrt_Barn <- tp$timings$arrival.time[tp$timings$arrival.to == "BNSLY"]
arrt_Mead <- tp$timings$arrival.time[tp$timings$arrival.to == "MEADWHL"]

arrt <- c(stopCondConv(arrt_Norm), stopCondConv(arrt_Wake), stopCondConv(arrt_Barn), stopCondConv(arrt_Mead))
arrt_seconds <- unlist(lapply(arrt, textToSeconds))

# Scheduled arrival times
scha_Norm <- tp$timings$arrival.schedule[tp$timings$arrival.to == "NORMNTN"]
scha_Wake <- tp$timings$arrival.schedule[tp$timings$arrival.to == "WKFLDKG"]
```

```
  scha_Barn <- tp$timings$arrival.schedule[tp$timings$arrival.to == "BNSLY"]
  scha_Mead <- tp$timings$arrival.schedule[tp$timings$arrival.to == "MEADWHL"]

  scha <- c(stopCondConv(scha_Norm), stopCondConv(scha_Wake), stopCondConv(scha_Barn), stopCondConv(scha_Mead))
  scha_seconds <- unlist(lapply(scha, textToSeconds))

  # Arrival delays
  data_test$arr_delayNorm[i] <- arrt_seconds[1] - scha_seconds[1]
  data_test$arr_delayWake[i] <- arrt_seconds[2] - scha_seconds[2]
  data_test$arr_delayBarn[i] <- arrt_seconds[3] - scha_seconds[3]
  data_test$arr_delayMead[i] <- arrt_seconds[4] - scha_seconds[4]
}


# Save the design matrix to a file
save(data_test, file="test_Matrix.RData")
```

5.3.3 Model construction to make prediction

```
library(randomForest)
library(caret)

# Load the training data
load("DesignMatrix.RData")

# Reset data with the 3 outliers removed
matrixData <- matrixData[matrixData$delayNotts < 2000, ]

# Set.seed to Reproduce the same result
set.seed(9088)

# Create the Random Forest model and Use 10-Fold Cross-Validation to Optimize the model
rf.fit <- train(delayNotts ~ ., data=matrixData, method="rf", trControl=trainControl(method="cv", number=10))

# Make predictions on test data
load("test_Matrix.RData")
pred_test <- predict(rf.fit, newdata = data_test)

write.csv(pred_test, file = "trains_group_D.csv", row.names=FALSE)
```

References

[1] Mathew Jackson. “How to Read an ECG”. In: *GEEKY MEDICS* (2011). URL: https://geekymedics.com/how-to-read-an-ecg/.

[2] “QRS complex”. In: *Wikipedia* (). URL: https://en.wikipedia.org/wiki/QRS_complex.

[3] “Dicision trees”. In: *IBM* (). URL: https://www.ibm.com/cn-zh/topics/decision-trees/.

[4] “Random forest - Random forest”. In: (). URL: https://easyai.tech/ai-definition/random-forest/.

[5] “Relationship between decision trees and random forests”. In: *CSDN* (). URL: https://blog.csdn.net/qq_39777550/article/details/107312048/.

[6] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning*. Second. Springer Series in Statistics. Data mining, inference, and prediction. Springer, New York, 2009, pp. xxii+745, 337–340. ISBN: 978-0-387-84857-0. DOI: 10.1007/978-0-387-84858-7. URL: https://doi.org/10.1007/978-0-387-84858-7.

[7] Kah Yong Tiong, Zhenliang Ma, and Carl-William Palmqvist. “A review of data-driven approaches to predict train delays”. In: *Transportation Research Part C: Emerging Technologies* 148 (2023), p. 104027. ISSN: 0968-090X. DOI: https://doi.org/10.1016/j.trc.2023.104027. URL: https://www.sciencedirect.com/science/article/pii/S0968090X23000165.

[8] Dewei Li, Winnie Daamen, and Rob MP Goverde. “Estimation of train dwell time at short stops based on track occupation event data: A study at a Dutch railway station”. In: *Journal of Advanced Transportation* 50.5 (2016), pp. 877–896.

[9] Weiwei Mou, Zhaolan Cheng, and Chao Wen. “Predictive model of train delays in a railway system”. In: *RailNorrköping 2019. 8th International Conference on Railway Operations Modelling and Analysis (ICROMA)*. 2019, pp. 913–929.

[10] Leo Breiman. “Random forests”. In: *Machine learning* 45 (2001), pp. 5–32.

[11] Manuel Fernández-Delgado et al. “Do we need hundreds of classifiers to solve real world classification problems?” In: *The journal of machine learning research* 15.1 (2014), pp. 3133–3181.

[12] Luca Oneto et al. “Advanced analytics for train delay prediction systems by including exogenous weather data”. In: *2016 IEEE international conference on data science and advanced analytics (DSAA)*. IEEE. 2016, pp. 458–467.

[13] Baptiste Gregorutti, Bertrand Michel, and Philippe Saint-Pierre. “Correlation and variable importance in random forests”. In: *Statistics and Computing* 27 (2017), pp. 659–678.

[14] Ron Kohavi et al. “A study of cross-validation and bootstrap for accuracy estimation and model selection”. In: *Ijcai*. Vol. 14. 2. Montreal, Canada. 1995, pp. 1137–1145.