Predicting Atrial Fibrillation Using Machine Learning

Antonio Duran[1], Graham Gee[1], Hoang-Nam Chu[1], and Matthew Lunn[1]

[1]Carleton University, Department of Mathematics and Statistics, Ottawa, Ontario

April 20, 2025

Abstract

Background: Atrial Fibrillation is a disease of age that is rapidly increasing in incidence as the average life expectancy increases. Detecting atrial fibrillation early is important so that medications and preventative care can be administered.

Objective: Given a dataset containing synthetic data from a cohort of 100, 000 patients from Calgary, unsupervised and supervised learning techniques were applied to classify patients as having or not having atrial fibrillation.

Methods: K-prototypes clustering, k-nearest neighbours, XGBoost, Random Forest, and a simple neural network was applied to an oversampled dataset.

Results: 3 clusters were found within the dataset using unsupervised learning. XGBoost resulted in the highest accuracy, recall, and AUC of .997, .9907, and 0.989, respectively.

Conclusion: Supervised learning techniques can be subject to overfitting without significant parameter tuning and cross-validation. Deep learning techniques should be tried on the dataset next to verify the results of the most recent papers.

Keywords: Atrial fibrillation • Machine Learning • Prediction

1 Introduction

The average life expectancy of a Canadian in 2011 was 81.7 years, an increase of 24.6 years from over a century ago. [1] Advances in medical technology, among other factors, have led to our increased longevity. As our quest for immortality continues, a surprising paradox juxtaposes this progress. Diseases of age, such as Atrial Fibrillation (AF), are on the rise. The global prevalence of AF has risen from 33.5 million to 59 million between 2010 and 2019 [22].

Atrial Fibrillation is a cardiac disorder marked by irregular heartbeat or rhythms in the upper chambers of the heart. It is most commonly diagnosed in older individuals with lifetime risk of developing AF increasing with age [9]. From the age of 75 to 80, the risk increases from 9 to 22 percent [24]. The consequences of AF are several; because blood is not moving properly throughout the body, the pooled blood can dislodge and cause blood clots leading to complications such as ischemic stroke and heart failure [2].

There are several manners to diagnose AF. An electrocardiogram (ECG) is a device used to monitor the speed and rhythm of the heart. An irregular heartbeat pattern detected by the ECG may indicate the presence of AF. Blood tests can also be used to identify root causes such as underlying health conditions or medications that may trigger AF [8]. Despite this, detecting AF can be difficult as the causes may be silent and the disease reoccurring [10].

Risk scores have been developed that use traditional statistics to identify patients at risk for AF. The C2HEST4, CHARGE-AF5, $CHADS_2$ are such models. These have shown to be weak or moderately predictive with concordance indices - a generalization of the area under the curve - of 0.59 - 0.73. Several limitations exist for these risk models; although simple, the studies that have used them work for certain ethnic populations, clinical settings, and symptomatic AF [5].

The increasing abundance of data storage, availability and governance, as well as increasing computational power, has allowed researchers and clinicians to leverage machine learning models to predict the incidence of AF. Machine learning leans on computational power to extract patterns from reams of data using statistical models. Some of the current machine learning models have achieved greater C-index scores of 0.78-0.82.7 [5]. Further, deep learning, a subset of machine learning that uses multiple layers of neural networks for its training, has achieved even higher scores [19].

Despite this, there is still a lack of research on generalizing to models which can predict AF with different inputs, under different clinical conditions, and with a variety of populations. The objective of this report is to explore the predictive capabilities of a range of unsupervised and supervised techniques on synthetic patient data provided by the Statistical Society of Canada's Case Study Competition (SSC).

2 Data

The data was retrieved from the Statistical Society of Canada's (SSC) annual case study competition, which gives the opportunity for university students to present solutions to real-world problems. The case study from which the data was obtained, *Prediction of New Onset Atrial Fibrillation Using Routinely Reported 12-Lead ECG Variables and Electronic Health Data*, contained synthetically generated patient data trained on a repository of around 100,000 patients from The Cardiovascular Imaging Registry of Calgary (CIROC). The data is of mixed type containing variables on patient demographics, patient cardiovascular health, prior health status, medication history and laboratory and ECG results. The data was prepared by Drs. James White, Dina Labib, and Jacqueline Flewitt of the Libin Cardiovascular Institute, University of Calgary [5].

2.1 Classes and Variables

The objective of the case study was to predict the onset of new atrial fibrillation (AF) from a cohort that had no prior history. Thus, the classes were segmented into those who had developed AF during the follow-up, encoded by the variable

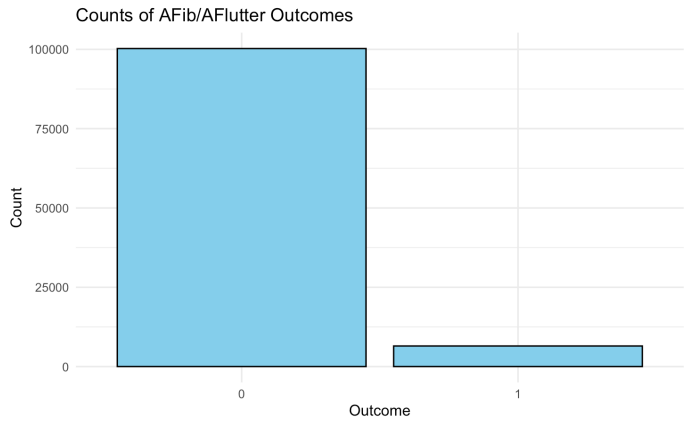

Figure 1: Severe class imbalances with a 1:15 ratio of non-AF to AF. There were 100, 056 of non-AF cases and 6441 cases of AF.

outcome_afib_aflutter_new_post. Those who developed AF during the follow-up were coded by a 1 and those who did not were coded by a 0.

2.2 Imbalances in the Data

There were severe imbalances in the data with 100,056 cases labelled "0" and 6,441 cases labelled "1." Figure 1 shows the huge discrepancy between the two classes. This level of imbalance can impede the results of classification tasks, as the data is heavily skewed toward the majority class. We therefore employed resampling methods (e.g. oversampling) to partially address this skew. Oversampling works by repeatedly sampling the minority class until the majority and minority classes are identical.

2.3 Data Imputation

Missing data is a frequently encountered issue in clinical health data. Subjects may omit parts of their medical history, or variables may not be fully captured [14]. The dataset contained 37 columns with at least any percentage of missing data. The mechanism by which data is missing must also be evaluated. Three general categories of missing-ness exist: 1) Missing completely at random (MCAR), 2) Missing at random (MAR), 3) Not Missing at Random (NMAR). Determining the mechanism behind the missing-ness can be difficult, however. If the cause of missing-ness is undetermined, the results of the regression can be severely biased [15].

The mechanism behind our dataset's missing values was tested using Little's MCAR test. A p-value of 1 indicated that the null hypothesis of the data being MCAR was failed to be rejected. Given this result we proceeded with Multiple Imputation by Chained Equations [4]. This imputation method replaces the missing values M times, creating M datasets in the process. The choice of imputation algorithm was predictive mean matching which works by predicting missing values based on observed ones. It was chosen due to its effectiveness in imputing mixed data and for its computational efficiency. 5–10 sets (M) are recommended in MICE but 3 datasets were chosen because of computational constraints. The machine learning algorithms were then run on each of the 3 datasets and the results pooled at the end [14].

2.4 Dimension Reduction

The curse of dimensionality refers to the sparseness of the distribution of points when datasets contain large numbers of dimensions (or features) [23]. This is commonly seen in clinical data which contain large amounts of variables on patient data [16]. One method to reduce the dimension of a dataset is called Principal Components Analysis (PCA). PCA was developed by Pearson and seeks to reduce a set of features to linear combinations of this set that maximally capture the variance of these features [17].

Sadegh et al. [26] show in their comparative analysis of dimensionality reduction techniques that PCA used with logistic regression, k-nearest neighbours, and support vector machines produced AUCs of 99.5, 98.1 and 99.1 respectively. We chose to reduce the dimensionality of our dataset from 139 to 62, capturing 70% of the variance in the model. Figure 2 shows the scree plot, a graphical tool used to identify cumulative variance, displaying 61 components capturing 70% of the variance in the data.

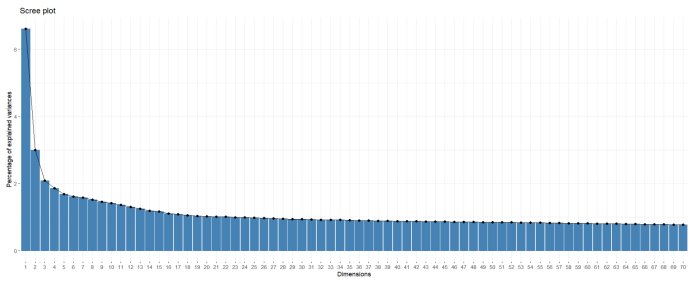

Figure 2: The scree plot is shown with the number of principal components along the x-axis and the percentage of explained variances along the y-axis. The scree plot shows the increase in percentage of variance explained by the model when that particular component is added to the model. With 61 principal components, the desired 70% of variance explained by model.

2.5 Data Reduction

The problem of too much data can bring unnecessary detail to a model. For example, a model may be able to represent the data with just the slope and the intercepts; this is a much cleaner representation of the original data [23]. Due to the size of our dataset, the Multivariate Adaptive Regression Splines (MARS) by Friedman was used. MARS is a data reduction technique that uses piecewise linear functions to build models similar to least squares [23]. MARS was applied to our dataset but had little to no effect on reducing the data.

2.6 Data Splitting

The data was split into training, validation, and test sets after data pre-processing to prevent data leakage. The *caret* package was used to split the data. Common split ratios are 80:20, 75:25, and Joseph [21] shows that a $\sqrt{p} : 1$ where p is the optimal number of parameters in a linear regression model that explains the data well, is optimal. However, we chose an 80:10:10, for the training set, validation set, and test set respectively, split for its simplicity and ubiquity. The training set was used to train the models, while the validation set was used to verify our results. Finally, the test set was used at the end of the model building process.

3 Methodology

Two main problems are available to solve with machine learning: prediction and classification. Prediction problems usually involve a machine learning model to predict a continuous outcome, while classification identifies pre-defined and discrete classes. Our problem is one of classification as our data is labeled as either AF or non-AF.

Thus, our methodology consists of unsupervised and supervised methods of classification. Each method will be discussed as follows.

3.1 Unsupervised Learning

Unsupervised learning machine learning algorithms group data into similar classes or clusters based on unlabeled data that is provided to it. There may or may not be any patterns which exist among the data. Unsupervised learning can be used in our context to verify whether or not there are any additional clusters of information which may be present aside from the pre-existing cluster of AF or no AF.

3.1.1 K-Prototypes

Several machine learning algorithms can be used to cluster data, specifically clinical health data. K-prototypes was chosen as the desired clustering algorithm as it performs well on mixed data types [20]. K-prototypes extends the K-means algorithm by partitioning numeric objects into K-clusters that minimize the within-group sum of squares. It uses the Euclidean distance as the distance measure. K-prototypes extends this notion further by using a dissimilarity measure on the categorical variable.

3.1.2 Davies–Bouldin

The choice of clusters can be chosen with a measure such as the Davies–Bouldin (DB) index. The DB index is the ratio of the sum of the intracluster distances over the intercluster distances. The minimum DB index is chosen corresponding to the number of clusters to be chosen [23].

3.2 Supervised Learning

Supervised learning involves the classification of data based on pre-existing labels. Supervised learning can also be thought of as humans supervising or labeling the data before it is provided to the algorithm[6]. We look at several supervised learning algorithms.

3.2.1 K-Nearest Neighbours (KNN)

The KNN classifier uses proximity as a tool to make classifications about the data. Developed by Fix, Hodges, and Cover, KNN relies on a majority voting scheme where the label most frequently represented by a data point is assigned its class [13]. Several distance metrics can be used as well, but we choose to use the Euclidean distance for consistency.

3.2.2 Classification Trees: Random Forest

Decision trees are a type of classifier introduced by Friedman and Breiman in the 1990s. They use the classification and regression trees algorithm which finds the best way to split a parent node into successive child nodes. The best way to split the data is evaluated by metrics such as the Gini index.

Random Forest extends CART and was trademarked by Breiman and Cutler. It is a part of ensemble learning methods called bootstrap aggregation (bagging), which aggregates the results of several decision trees, known collectively as a forest. Once the node size, number of trees, and number of features sampled is determined, RF can determine the appropriate predicted class through a majority vote - similar to KNN [12].

3.2.3 Boosting – XGBoost

Instead of bagging as the ensemble method, boosting relies on sequentially training of models rather than parallel. Each model's weakness is then used to improve the next iteration [11]. XGBoost, formally known as extreme gradient boosting, utilizes the gradient descent algorithm to generate these weak models. XGBoost is favoured for its high scalability and its high accuracy [7].

3.2.4 Neural Networks

A neural network mimics the architecture of the human brain by classifying data based on several neural layers. A simple neural network is composed of an input layer, several hidden layers, and an output layer. Weights are assigned to each input to produce a desired output [23].

3.3 Classification Measures

Three important performance metrics that explain the model's abilities are **Accuracy, Recall** and **AUC**. In order to derive these measures, a confusion matrix was produced. The confusion matrix is a 2 by 2 table that represents a model's predictive qualities. Table 1 describes the contents of each quadrant. Each quadrant can be described as one of four cases:

1. **True Positive (TP)**: The observation is actually positive and the model identifies it as such.
2. **False Negative (FN)**: The observation is actually positive but the model declares it as negative.
3. **True Negative (TN)**: The observation is actually negative and the model declares it as so.
4. **False Positive (FP)**: The observation is actually negative but the model declares it as positive.

These classifications are represented in the confusion matrix starting in the top left-hand quadrant and proceeding clockwise [25].

From these labels, we can derive two important measures: **Accuracy** and **Recall**. Accuracy measures the total number of true negatives and positives that the model predicts out of every single observation, that is,

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Recall measures the total number of true positives that were classified out of all positive observations:

$$\text{Recall} = \frac{TP}{TP + FN}$$

These two measures are important but can contrast each other for our dataset. For heavily imbalanced datasets, where the positive cases are very rare, recall is a more meaningful measure. For disease detection, correctly identifying positive cases as opposed to negative cases is vital [3].

AUC is a classification threshold which evaluates a model's performance over several thresholds. The closer the value of AUC to one, the greater the model's ability to correctly predict true positives and false negatives [3].

4 Results

The results were gathered from statistical analyses on the three datasets from the MICE imputations and the results were pooled together and are displayed in table 1 and table 2. The Random Forest algorithm was applied but failed to yield any results because of memory capacity issues. The neural network was run on the training set and not the validation set.

4.1 Visualizations

Several visualizations were performed before the statistical analyses to show the variation in the data.

Figures 3-5 describe demographic trends about the patients. Figure 3 shows that sex is equally balanced across classes, and figure 5 shows the same across the entire dataset. Figure 4 shows that age within the data is normally distributed around the age of 55.

Figures 6 and 7 paint a picture of the health status of patients. The top comorbidities associated with AF are known to be dyslipidemia and hypertension, while old age and low creatinine levels seems to be associated with death outcomes.

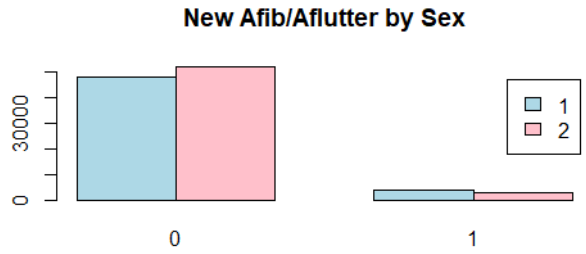

Figure 3: Classification of AF by sex.

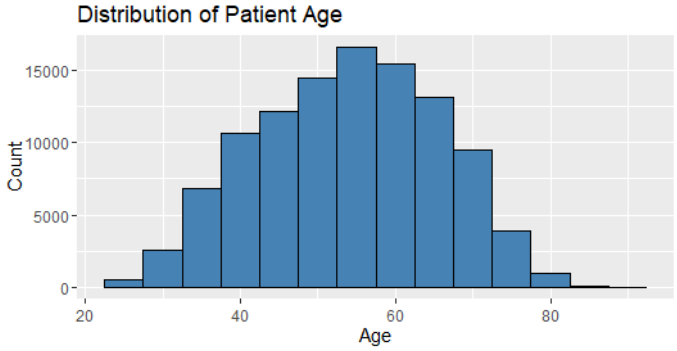

Figure 4: Distribution of patients by age. It is normally distributed.

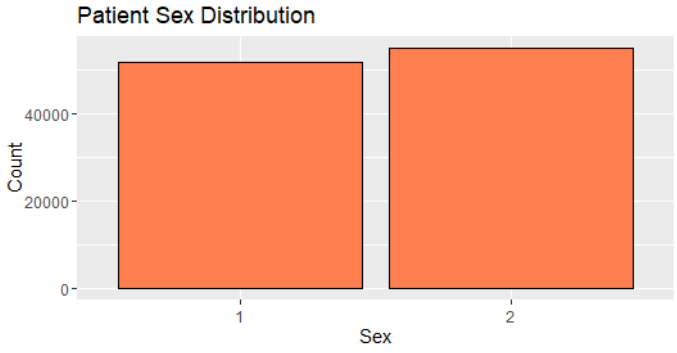

Figure 5: Patient sex distribution within the entire dataset.

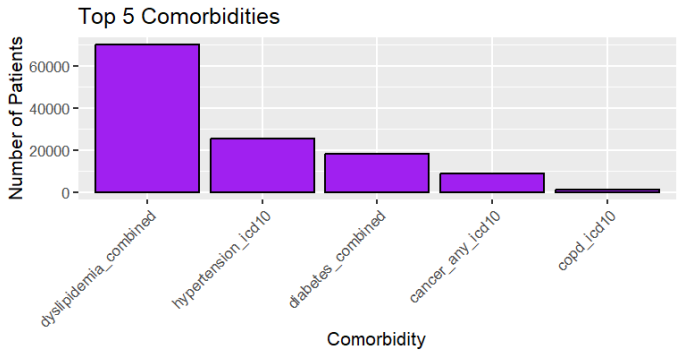

Figure 6: Top 5 comorbidities alongside AF.

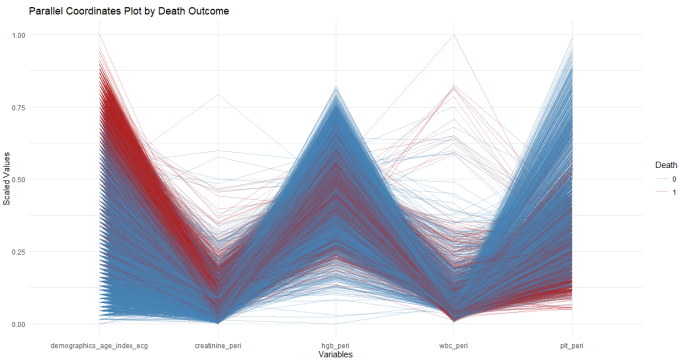

Figure 7: Parallel coordinates plot demonstrating outcomes of death and any associations.

4.2 Unsupervised Learning Results

Running the k-prototypes algorithm yielded a different number of desirable clusters from our labeled data. Using the within-cluster sum of squares and the DBI, figures 8 and 9 show respectively that the number of optimal clusters within the datasets was about 3. This lay in contrast to the 2 clusters that are apparent from our class labels. This suggests that the data may be categorized within another set of labels.

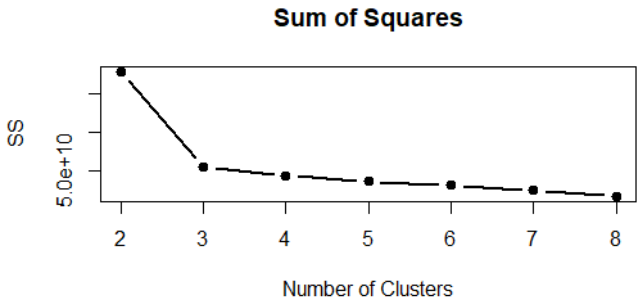

Figure 8: Total within-cluster sum of squares (SS) plotted against the number of clusters (k = 2 to 8). This figure immediately starts off with a steep drop between 2 and 3 clusters which suggests that adding more clusters initially improves this cluster compactness. After k = 4, the curve narrows out, indicating that adding anymore cluster may not be very beneficial. This suggests the optimal number of clusters is around 3.

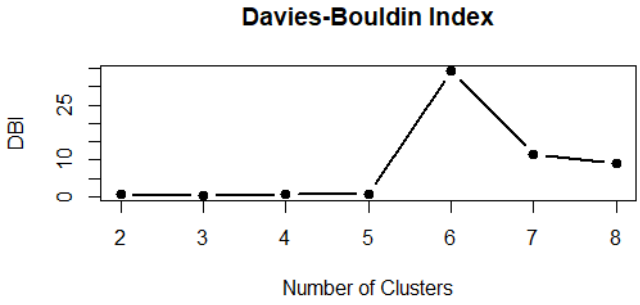

Figure 9: Davies-Bouldin Index (DBI) across cluster numbers (k = 2 to 8). DBI is lowest between 2 and 5 clusters, suggesting better clustering within that range. The spike at k = 6 shows poor separation and should not be considered. Overall, 3 clusters seem to optimal number of clusters.

4.3 Supervised Learning Results

Tables 1 and 2 summarize the supervised learning results. Table 1 shows the confusion matrix that was run for all four models, and the table 2 displays their performance metrics. We start by contrasting accuracy and recall. The highest accuracy came from XGBoost while the lowest accuracy came from the neural network. The highest recall

came from XGBoost and the lowest recall came from the neural network. We can see that models have recall values that are similar to each other. XGBoost could representing overfitting of the data even though it was validated using the validation set. AUC values were highest for XGBoost as well.

Since detection of AF is vital so that preventative measures can take place, a high recall value is important. Due to its overfitting, K-nearest neighbours may be a good option as it had both high accuracy and recall. However, if further cross-validation was performed on tuning XGBoost's parameters, it may serve as the best option. For the 647 patients that had AF in the validation set, it was able to correctly classify 641 of them. A low false negative number is valuable as we want the most amount of patients to receive help as possible.

5 Conclusion

Several machine learning techniques, both unsupervised and supervised, were applied to the SSC's dataset to classify patient data as having atrial fibrillation or no atrial fibrillation. After applying a range of data-reduction techniques, as well as pre-processing the data, four supervised learning and one unsupervised techniques were applied to the dataset.

The results show that had the highest scores for Recall, Accuracy and AUC was from XGBoost. From our discussion in the theoretical portion, a model that can identify true positives is vital in disease detection.

Some weaknesses that exist for our methodological approach include a lack of cross-validation or other validation strategies for parameter tuning. This lack of justification leaves our models prone to overfitting. In addition, leveraging greater computational power—such as using cloud computing or hardware upgrades—could have enabled us to tune parameters more frequently. Varying the kinds of datasets on which the models were trained could also have provided a clearer picture of our results. Finally, experimenting with more machine learning algorithms and with deep learning could help us discover more accurate models [18].

Table 1: Confusion matrices for Models 1–4

Model 1: K-nearest Neighbours

	Actual: Atrial Fibrillation	**Actual: No Atrial Fibrillation**
Predicted: Atrial Fibrillation	599	64
Predicted: No Atrial Fibrillation	48	9964

Model 2: XGBoost

	Actual: Atrial Fibrillation	**Actual: No Atrial Fibrillation**
Predicted: Atrial Fibrillation	641	22
Predicted: No Atrial Fibrillation	6	10006

Model 3: Random Forest

	Actual: Atrial Fibrillation	**Actual: No Atrial Fibrillation**
Predicted: Atrial Fibrillation	0	0
Predicted: No Atrial Fibrillation	0	0

Model 4: Neural Network

	Actual: Atrial Fibrillation	**Actual: No Atrial Fibrillation**
Predicted: Atrial Fibrillation	68334	7266
Predicted: No Atrial Fibrillation	12105	72921

Table 2: Performance metrics for classification methods

Model	**Accuracy**	**Recall**	**AUC**
K-Nearest Neighbours	0.9899	0.9258	0.9502
XGBoost	0.997	0.9907	0.989
Random Forest	NA	NA	NA
Neural Network	0.8794	0.8495	0.6617

References

[1] https://www150.statcan.gc.ca/n1/pub/82-624-x/2014001/article/14009-eng.htm. Accessed: 2025-4-19.

[2] Atrial fibrillation. https://www.heartandstroke.ca/heart-disease/conditions/atrial-fibrillation. Accessed: 2025-4-19.

[3] Classification: ROC and AUC. https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc. Accessed: 2025-4-19.

[4] No title. https://stefvanbuuren.name/fimd/. Accessed: 2025-4-19.

[5] Prediction of new onset atrial fibrillation using routinely reported 12-lead ECG variables and electronic health data. https://ssc.ca/en/case-study/prediction-new-onset-atrial-fibrillation-using-routinely-reported-12-lead-ecg-variables. Accessed: 2025-4-19.

[6] What is supervised learning? https://cloud.google.com/discover/what-is-supervised-learning?hl=en. Accessed: 2025-4-19.

[7] XGBoost. https://www.nvidia.com/en-us/glossary/xgboost/. Accessed: 2025-4-19.

[8] How atrial fibrillation is diagnosed. https://www.webmd.com/heart-disease/atrial-fibrillation/afib-diagnosis, June 2011. Accessed: 2025-4-19.

[9] What is afib? https://www.hopkinsmedicine.org/health/conditions-and-diseases/atrial-fibrillation, April 2023. Accessed: 2025-4-19.

[10] Data science centre at the university of ottawa heart institute finds common language across data. https://www.ottawaheart.ca/news/data-science-centre-the-university-ottawa-heart-institute-finds-common-language-across-d March 2024. Accessed: 2025-4-19.

[11] What is boosting? https://www.ibm.com/think/topics/boosting, December 2024. Accessed: 2025-4-19.

[12] What is random forest? https://www.ibm.com/think/topics/random-forest, April 2025. Accessed: 2025-4-19.

[13] What is the k-nearest neighbors algorithm? https://www.ibm.com/think/topics/knn, April 2025. Accessed: 2025-4-19.

[14] Peter C Austin, Ian R White, Douglas S Lee, and Stef van Buuren. Missing data in clinical research: A tutorial on multiple imputation. *Can. J. Cardiol.*, 37(9):1322–1331, September 2021.

[15] D A Bennett. How can I deal with missing data in my study? *Aust. N. Z. J. Public Health*, 25(5):464–469, October 2001.

[16] Visar Berisha, Chelsea Krantsevich, P Richard Hahn, Shira Hahn, Gautam Dasarathy, Pavan Turaga, and Julie Liss. Digital medicine and the curse of dimensionality. *NPJ Digit. Med.*, 4(1):153, October 2021.

[17] Michael Greenacre, Patrick J F Groenen, Trevor Hastie, Alfonso Iodice D'Enza, Angelos Markos, and Elena Tuzhilina. Principal component analysis. *Nat. Rev. Methods Primers*, 2(1):1–21, December 2022.

[18] Youfu He, Yu Zhou, Yu Qian, Jingjie Liu, Jinyan Zhang, Debin Liu, and Qiang Wu. Cardioattentionnet: advancing ECG beat characterization with a high-accuracy and portable deep learning model. *Front. Cardiovasc. Med.*, 11:1473482, 2024.

[19] Jim Holdsworth and Mark Scapicchio. What is deep learning? `https://www.ibm.com/think/topics/deep-learning`, April 2025. Accessed: 2025-4-19.

[20] Zhexue Huang. Extensions to the k-means algorithm for clustering large data sets with categorical values. `https://cse.hkust.edu.hk/~qyang/537/Papers/huang98extensions.pdf`, 1998. Accessed: 2025-4-19.

[21] V Roshan Joseph. Optimal ratio for data splitting. *Stat. Anal. Data Min.*, 15(4):531–538, August 2022.

[22] Dominik Linz, Monika Gawalko, Konstanze Betz, Jeroen M Hendriks, Gregory Y H Lip, Nicklas Vinter, Yutao Guo, and Søren Johnsen. Atrial fibrillation: epidemiology, screening and digital health. *Lancet Reg. Health Eur.*, 37(100786):100786, February 2024.

[23] Shirley Mills. Lecture notes for stat4601, April 2025.

[24] Zeid Nesheiwat, Amandeep Goyal, and Mandar Jagtap. Atrial fibrillation. In *StatPearls*. StatPearls Publishing, Treasure Island (FL), January 2025.

[25] Anna Magdalena Ogłoszka and Łukasz Smaga. Classification methods in the diagnosis of breast cancer. *Biom. Lett.*, 59(2):99–126, December 2022.

[26] Seyed-Ali Sadegh-Zadeh, Nasrin Sadeghzadeh, Ommolbanin Soleimani, Saeed Shiry Ghidary, Sobhan Movahedi, and Seyed-Yaser Mousavi. Comparative analysis of dimensionality reduction techniques for EEG-based emotional state classification. *Am. J. Neurodegener. Dis.*, 13(4):23–33, October 2024.

6 Code Appendix

```r
rm(list=ls())
setwd("C:/Users/Owner/Documents/University/Winter 2025/STAT 4601/Final Project")
getwd()

#Download data set
dataset<-read.csv("synthetic_data_stats_competition_2025_final 1(in).csv")

#Set seed
set.seed(123)


#Basic visualizations & Pre-Processing


#Get # rows
xx<-1:as.integer(length(dataset$hypertension_icd10))
#Making all NA's zero so that it will plot without issue
dataset <- dataset %>% mutate(across(everything(), ~ ifelse(is.na(.), 0, .)))
#Getting names of columns
names<-colnames(dataset)

#3x3 grid of plots
par(mfrow=c(1,1))

#Plotting each grid
for(i in 1:137){

  plot(x=xx,y=dataset[,i], xlab="datapoints",ylab=names[i],main=paste("Plot of",names[i
    ],sep=" "))
}


#Download MICE package to use MICE imputation
library(mice)
#library(openxlsx)

#Fill these 2 columns with missing data with 0s.
for(i in 1:dim(dataset)[1]){
  if(dataset[i,136]==0){
    dataset[i,137]<-0
  }
  if(dataset[i,134]==0){
    dataset[i,135]<-0
  }
}

#Boolean data set where TRUE if data is missing, FALSE if it isn't
miss_data<-is.na(dataset)

#calculate percentage of missingness in each column
missing<-c()
for(i in 1:dim(dataset)[2]){
  num<-0
  for(j in 1:dim(dataset)[1]){
    if(miss_data[j,i]==TRUE){
      num<-num+1
    }
```

```
  }
  p<-num/dim(dataset)[1]
  missing[i]<-p
}

#Delete columns missing at least 70% of data
for(i in length(missing):1){
  if(missing[i]>0.7){
    dataset<-dataset[,-i]
  }
}

#Remove columns with only 1 level/factor
rem_cols<-sapply(dataset,function(x)length(unique(x))==1)
dataset<-dataset[,!rem_cols]

#Impute the missing data using MICE
#WARNING: This function takes a while to run
imputed_data<-mice(dataset,m=3,maxit=5,method='pmm',print=FALSE,seed=500)

#Get the complete imputated data sets from the MICE function
imp_dat1<-complete(imputed_data,1)
imp_dat2<-complete(imputed_data,2)
imp_dat3<-complete(imputed_data,3)

#Export datasets with imputation
write.xlsx(imp_dat1, "C:/Users/Owner/Documents/University/Winter 2025/STAT 4601/Final
    Project/imputed_data_1.xlsx")
write.xlsx(imp_dat2, "C:/Users/Owner/Documents/University/Winter 2025/STAT 4601/Final
    Project/imputed_data_2.xlsx")
write.xlsx(imp_dat3, "C:/Users/Owner/Documents/University/Winter 2025/STAT 4601/Final
    Project/imputed_data_3.xlsx")

#Download imputed data sets so we don't have to repeatedly run MICE function
imputed_data1<-read.csv("imputed_data_1.csv")
imputed_data2<-read.csv("imputed_data_2.csv")
imputed_data3<-read.csv("imputed_data_3.csv")

#Remove the id column from each dataset
imputed_data1<-imputed_data1[,-1]
imputed_data2<-imputed_data2[,-1]
imputed_data3<-imputed_data3[,-1]

#Download package that will be used to split data into train/test/validation sets
library(caret)

#Split the first imputed data set into training, testing and validation sets
train_indices1<-createDataPartition(imputed_data1$outcome_afib_aflutter_new_post,p=0.8,
    list=FALSE,times=1)
train_data1<-imputed_data1[train_indices1,]
not_train_data1<-imputed_data1[-train_indices1,]
valid_indices1<-createDataPartition(not_train_data1$outcome_afib_aflutter_new_post,p
    =0.5,list=FALSE,times=1)
valid_data1<-not_train_data1[valid_indices1,]
test_data1<-not_train_data1[-valid_indices1,]

#Split the second imputed data set into training, testing and validation sets
train_indices2<-createDataPartition(imputed_data2$outcome_afib_aflutter_new_post,p=0.8,
    list=FALSE,times=1)
train_data2<-imputed_data2[train_indices2,]
```

```
not_train_data2<-imputed_data2[-train_indices2,]
valid_indices2<-createDataPartition(not_train_data2$outcome_afib_aflutter_new_post,p=0.5,list=FALSE,times=1)
valid_data2<-not_train_data2[valid_indices2,]
test_data2<-not_train_data2[-valid_indices2,]

#Split the third imputed data set into training, testing and validation sets
train_indices3<-createDataPartition(imputed_data3$outcome_afib_aflutter_new_post,p=0.8,list=FALSE,times=1)
train_data3<-imputed_data3[train_indices3,]
not_train_data3<-imputed_data3[-train_indices3,]
valid_indices3<-createDataPartition(not_train_data3$outcome_afib_aflutter_new_post,p=0.5,list=FALSE,times=1)
valid_data3<-not_train_data3[valid_indices3,]
test_data3<-not_train_data3[-valid_indices3,]

library(profvis)
library(ROSE)

#Over-sample the data set
over_train_data1<-ovun.sample(outcome_afib_aflutter_new_post~.,data=train_data1,method="over",p=0.5,seed=2)$data
over_train_data2<-ovun.sample(outcome_afib_aflutter_new_post~.,data=train_data2,method="over",p=0.5,seed=2)$data
over_train_data3<-ovun.sample(outcome_afib_aflutter_new_post~.,data=train_data3,method="over",p=0.5,seed=2)$data

#Under-sample the data set
under_train_data1<-ovun.sample(outcome_afib_aflutter_new_post~.,data=train_data1,method="under",p=0.5,seed=2)$data
under_train_data2<-ovun.sample(outcome_afib_aflutter_new_post~.,data=train_data2,method="under",p=0.5,seed=2)$data
under_train_data3<-ovun.sample(outcome_afib_aflutter_new_post~.,data=train_data3,method="under",p=0.5,seed=2)$data

#Store the response variable of the oversampled and undersampled data
over_response1<-over_train_data1$outcome_afib_aflutter_new_post
over_response2<-over_train_data2$outcome_afib_aflutter_new_post
over_response3<-over_train_data3$outcome_afib_aflutter_new_post

under_response1<-under_train_data1$outcome_afib_aflutter_new_post
under_response2<-under_train_data2$outcome_afib_aflutter_new_post
under_response3<-under_train_data3$outcome_afib_aflutter_new_post

#Get the principle components for each training set

#PCA on over and under sampled data first as they depend on the training sets
over_pca1<-prcomp(over_train_data1, center = TRUE, scale.=TRUE)
fviz_eig(over_pca1,ncp=70)
over_pca2<-prcomp(over_train_data2, center = TRUE, scale.=TRUE)
fviz_eig(over_pca2,ncp=70)
over_pca3<-prcomp(over_train_data3, center = TRUE, scale.=TRUE)
fviz_eig(over_pca3,ncp=70)

#Error scaling constant/zero column to unit variance, seems to be different each time, run under sampled ones 1 at a time!
under_pca1<-prcomp(under_train_data1, center = TRUE, scale.=TRUE)
fviz_eig(under_pca1,ncp=70)
under_pca2<-prcomp(under_train_data2, center = TRUE, scale.=TRUE)
fviz_eig(under_pca2,ncp=70)
```

```
under_pca3<-prcomp(under_train_data3, center = TRUE, scale.=TRUE)
fviz_eig(under_pca3,ncp=70)

#PCA on training sets
train_pca1<-prcomp(train_data1, center = TRUE, scale.=TRUE)
fviz_eig(train_pca1,ncp=70)
train_pca2<-prcomp(train_data2, center = TRUE, scale.=TRUE)
fviz_eig(train_pca2,ncp=70)
train_pca3<-prcomp(train_data3, center = TRUE, scale.=TRUE)
fviz_eig(train_pca3,ncp=70)

#Create data sets of the pcas with a cumulative variance proportion of 70%
summary(over_pca1)
over_reduced_data1<-over_pca1$x[,1:61]
summary(under_pca1)

reduce_datasets <- function(dat, pca_model, variables_kept) {
  #Get the absolute values of the loadings from the rotation matrix
  absolute_loadings<-abs(pca_model$rotation)
  #Sum the loadings across the principal components
  variable_importance<-rowSums(absolute_loadings)

  #Extract the
  important_variables<-names(sort(variable_importance, decreasing = TRUE))[1:variables_
    kept]

  return(dat[, important_variables])
}

#Create data sets of the pcas with a cumulative variance proportion of 70% , then
    reduced original datasets using the pca's.
#I chose to use 62 variables as it matches the number of principal components (62)
    necessary to get 70% variance
#If there are better methods of choosing an amount of variables to keep we should
    explore those, as the reasoning here is fairly loose and not mathematical.

#Over and undersampled training sets
pca_over_reduced_data1<-over_pca1$x[,1:61]
pca_over_reduced_data2<-over_pca2$x[,1:61]
pca_over_reduced_data3<-over_pca3$x[,1:61]

over_reduced_data1<-pca_over_reduced_data1

reduced_over_data1<-data.frame(reduce_datasets(over_train_data1,over_pca1,62),outcome_
    afib_aflutter_new_post=over_response1)
reduced_over_data2<-data.frame(reduce_datasets(over_train_data2,over_pca2,62),outcome_
    afib_aflutter_new_post=over_response2)
reduced_over_data3<-data.frame(reduce_datasets(over_train_data3,over_pca3,62),outcome_
    afib_aflutter_new_post=over_response3)
#Note these lines is where I chose 62 variables, not the above:      here ^^ sets it to
     62 most important variables according to pca in dataset.

#These may crash on one, just keep running next lines
pca_under_reduced_data1<-under_pca1$x[,1:60]
pca_under_reduced_data2<-under_pca2$x[,1:60]
pca_under_reduced_data3<-under_pca3$x[,1:60]

reduced_under_data1<-data.frame(reduce_datasets(under_train_data1,under_pca1,62),
    outcome_afib_aflutter_new_post=under_response1)
reduced_under_data2<-data.frame(reduce_datasets(under_train_data2,under_pca2,62),
```

```
    outcome_afib_aflutter_new_post=under_response2)
reduced_under_data3<-data.frame(reduce_datasets(under_train_data3,under_pca3,62),
    outcome_afib_aflutter_new_post=under_response3)

#Training sets
pca_reduced_train_data1<-train_pca1$x[,1:62]
pca_reduced_train_data2<-train_pca2$x[,1:62]
pca_reduced_train_data3<-train_pca3$x[,1:62]

#Include response since it was removed
reduced_train_data1<-data.frame(reduce_datasets(train_data1,train_pca1,62),outcome_afib
    _aflutter_new_post=train_data1$outcome_afib_aflutter_new_post)
reduced_train_data2<-data.frame(reduce_datasets(train_data2,train_pca2,62),outcome_afib
    _aflutter_new_post=train_data2$outcome_afib_aflutter_new_post)
reduced_train_data3<-data.frame(reduce_datasets(train_data3,train_pca3,62),outcome_afib
    _aflutter_new_post=train_data3$outcome_afib_aflutter_new_post)

#Test sets
reduced_test_data1<-data.frame(reduce_datasets(test_data1,train_pca1,62),outcome_afib_
    aflutter_new_post=test_data1$outcome_afib_aflutter_new_post)
reduced_test_data2<-data.frame(reduce_datasets(test_data2,train_pca2,62),outcome_afib_
    aflutter_new_post=test_data2$outcome_afib_aflutter_new_post)
reduced_test_data3<-data.frame(reduce_datasets(test_data3,train_pca3,62),outcome_afib_
    aflutter_new_post=test_data3$outcome_afib_aflutter_new_post)

#Clean data and get imputed data with factors
dat<-imputed_data1
#Factor all columns
dat[] <- lapply(dat, function(col) {
  if (all(col %in% c(0, 1)) && is.numeric(col)) {
    factor(col)
  } else {
    col
  }
})

#Remove the time column for the response as they are too related
dat <- dat %>% select(-time_to_outcome_afib_aflutter_new_post)

dat1<-imputed_data2
#Factor all columns
dat1[] <- lapply(dat1, function(col) {
  if (all(col %in% c(0, 1)) && is.numeric(col)) {
    factor(col)
  } else {
    col
  }
})

#Remove the time column for the response as they are too related
dat1 <- dat1 %>% select(-time_to_outcome_afib_aflutter_new_post)

dat2<-imputed_data3
#Factor all columns
dat2[] <- lapply(dat2, function(col) {
  if (all(col %in% c(0, 1)) && is.numeric(col)) {
    factor(col)
  } else {
    col
  }
```

```
})

#Remove the time column for the response as they are too related
dat2 <- dat2 %>% select(-time_to_outcome_afib_aflutter_new_post)


###MARS



#Must load files imputed_data1, imputed_data2, imputed_data3 before running
#We will do this using the caret package

#Run MARS regression
MARS_model1<-train(outcome_afib_aflutter_new_post ~ ., data = dat, method = "earth",
    trControl = trainControl(method = "cv", number = 2),tuneLength = 2)
MARS_model2<-train(outcome_afib_aflutter_new_post ~ ., data = dat1, method = "earth",
    trControl = trainControl(method = "cv", number = 2),tuneLength = 2)
MARS_model3<-train(outcome_afib_aflutter_new_post ~ ., data = dat2, method = "earth",
    trControl = trainControl(method = "cv", number = 2),tuneLength = 2)


#Summary of mars
summary(MARS_model1$finalModel)
summary(MARS_model2$finalModel)
summary(MARS_model3$finalModel)

#Get coefficients
MARS_coefs<-coef(MARS_model1$finalModel)
MARS_coefs1<-coef(MARS_model2$finalModel)
MARS_coefs2<-coef(MARS_model3$finalModel)


#Model selected different combinations of
#ecg_resting_qtc - 2 behaviours
#inr_peri - 10 behaviours
#albumin_peri - 2 bahaviours
#bilirubin_total_peri - 7 behaviours
#urea_peri - 2 behaviours
#creatinine_peri - 2 behaviours
#ferritin_peri - 4 behaviours

#Mars works by running regression of the form Y = B0 + B1*h(a-X1) + B2*h(b-X2) + ...
#Each hinge function is unique, and is of the form h(a-x) = max(0,a-x) or h(x-a) = max
    (0,x-a)
#Each hinge function acts on different variables, but can act on the same one multiple
    times
#For ex, inr_peri is acted on 9 times, creating 9 threshold points with 10 different
    regions of behaviour for inr_peri
#So in that regard, it can be said inr_peri has 10 different coefficients corresponding
     to the different regions identified by the model.

new_dat<-data.frame(imputed_data1$ecg_resting_qtc,imputed_data1$inr_peri,imputed_data1$
    albumin_peri,imputed_data1$bilirubin_total_peri,imputed_data1$urea_peri,imputed_
    data1$creatinine_peri,imputed_data1$ferritin_peri)
unique_mars_values<-sapply(new_dat,unique)

new_dat1<-data.frame(imputed_data2$ecg_resting_qtc,imputed_data2$inr_peri,imputed_data2
    $albumin_peri,imputed_data2$bilirubin_total_peri,imputed_data2$urea_peri,imputed_
    data2$creatinine_peri,imputed_data2$ferritin_peri)
```

```
unique_mars_values<-sapply(new_dat,unique)

new_dat2<-data.frame(imputed_data3$ecg_resting_qtc,imputed_data3$inr_peri,imputed_data3$albumin_peri,imputed_data3$bilirubin_total_peri,imputed_data3$urea_peri,imputed_data3$creatinine_peri,imputed_data3$ferritin_peri)
unique_mars_values<-sapply(new_dat,unique)

pred_probs <- predict(MARS_model1, newdata = dat, type = "prob")[,2]
actual <- as.numeric(dat$outcome_afib_aflutter_new_post) - 1
residuals_raw <- actual - pred_probs
threshold <- quantile(abs(residuals_raw), 0.9)
reduced_data <- new_dat[abs(residuals_raw) > threshold, ]

pred_probs1 <- predict(MARS_model2, newdata = dat, type = "prob")[,2]
actual1 <- as.numeric(dat1$outcome_afib_aflutter_new_post) - 1
residuals_raw1 <- actual1 - pred_probs1
threshold1 <- quantile(abs(residuals_raw1), 0.9)
reduced_data1 <- new_dat1[abs(residuals_raw1) > threshold1, ]

pred_probs2 <- predict(MARS_model3, newdata = dat, type = "prob")[,2]
actual2 <- as.numeric(dat2$outcome_afib_aflutter_new_post) - 1
residuals_raw2 <- actual2 - pred_probs2
threshold2 <- quantile(abs(residuals_raw2), 0.9)
reduced_data2 <- new_dat2[abs(residuals_raw2) > threshold2, ]

#Final note for this is that I don't believe MARS does a good job at reducing the data, so will look to try other methods



### Generalized Additive Model (GAM)



#Using gam from the mgcv package
library(mgcv)
train_data1_reduced<-data.frame(train_data1)

#name of the columns for the model
response_var <- "outcome_afib_aflutter_new_post"
predictor_vars <- setdiff(colnames(train_data1_reduced), response_var)

#Run the Generalized Additive Model
gam_model <- gam(as.formula(paste(response_var, "~", paste(predictor_vars, collapse = " + "))), data=train_data1_reduced, family = binomial(link = "logit"), method = "REML")

#Get residuals and fitted values
train_data1_reduced$fitted <- predict(gam_model, type = "response")
train_data1_reduced$residuals <- residuals(gam_model, type = "deviance")

#Number of rows to keep, will have to justify this number!
top_n <- 20000

#Rank the rows and keep the 20000 most influencial ones(Explain this better)
train_data1_reduced <- train_data1_reduced %>% mutate(residual_strength = abs(residuals)) %>% arrange(desc(residual_strength)) %>% slice(1:top_n)

#Remove them(residuals,fitted columns that were added) at the end
train_data1_reduced<-train_data1_reduced[,c(-120,-119,-118)]
```

```r
library(stats)
library(cluster)
library(clustMixType)
library(dplyr)
library(Rtsne)
library(ggplot2)
library(tidyr)
library(pROC)

###Training Sets
#Re factors the factored rows and prepare test set
train_data1$demographics_birth_sex<-train_data1$demographics_birth_sex-1
train_data2$demographics_birth_sex<-train_data2$demographics_birth_sex-1
train_data3$demographics_birth_sex<-train_data3$demographics_birth_sex-1

#Find columns that are factors
binary_cols_train <- sapply(train_data1, function(x) all(x %in% c(0, 1)) & is.numeric(x))
#Factor these columns
train_data1[binary_cols_train] <- lapply(train_data1[binary_cols_train], as.factor)
train_data2[binary_cols_train] <- lapply(train_data2[binary_cols_train], as.factor)
train_data3[binary_cols_train] <- lapply(train_data3[binary_cols_train], as.factor)

#Removing time variable
train_data1 <- train_data1[, !(names(train_data1) %in% c("time_to_outcome_afib_aflutter_new_post"))]
train_data2 <- train_data2[, !(names(train_data2) %in% c("time_to_outcome_afib_aflutter_new_post"))]
train_data3 <- train_data3[, !(names(train_data3) %in% c("time_to_outcome_afib_aflutter_new_post"))]

###Test sets
test_data1$demographics_birth_sex<-test_data1$demographics_birth_sex-1
test_data1$demographics_birth_sex<-test_data1$demographics_birth_sex-1
test_data1$demographics_birth_sex<-test_data1$demographics_birth_sex-1

binary_cols_test <- sapply(test_data1, function(x) all(x %in% c(0, 1)) & is.numeric(x))

test_data1[binary_cols_test] <- lapply(test_data1[binary_cols_test], as.factor)
test_data2[binary_cols_test] <- lapply(test_data2[binary_cols_test], as.factor)
test_data3[binary_cols_test] <- lapply(test_data3[binary_cols_test], as.factor)

#Storing test response
test1_response<-test_data1$outcome_afib_aflutter_new_post
test2_response<-test_data2$outcome_afib_aflutter_new_post
test3_response<-test_data3$outcome_afib_aflutter_new_post

#Removing Response and time response from test set.
test_data1<-test_data1[, !(names(test_data1) %in% c("outcome_afib_aflutter_new_post", "time_to_outcome_afib_aflutter_new_post"))]
test_data2<-test_data2[, !(names(test_data2) %in% c("outcome_afib_aflutter_new_post", "time_to_outcome_afib_aflutter_new_post"))]
test_data3<-test_data3[, !(names(test_data3) %in% c("outcome_afib_aflutter_new_post", "time_to_outcome_afib_aflutter_new_post"))]

#Reduced sets
```

```
binary_cols_train_reduced <- sapply(reduced_train_data1, function(x) all(x %in% c(0, 1)) & is.numeric(x))
#Factor these columns
reduced_train_data1[binary_cols_train_reduced] <- lapply(reduced_train_data1[binary_cols_train_reduced], as.factor)
reduced_train_data2[binary_cols_train_reduced] <- lapply(reduced_train_data2[binary_cols_train_reduced], as.factor)
reduced_train_data3[binary_cols_train_reduced] <- lapply(reduced_train_data3[binary_cols_train_reduced], as.factor)

reduced_train_data1 <- reduced_train_data1[, !(names(reduced_train_data1) %in% c("time_to_outcome_afib_aflutter_new_post"))]
reduced_train_data2 <- reduced_train_data2[, !(names(reduced_train_data2) %in% c("time_to_outcome_afib_aflutter_new_post"))]
reduced_train_data3 <- reduced_train_data3[, !(names(reduced_train_data3) %in% c("time_to_outcome_afib_aflutter_new_post"))]

binary_cols_test_reduced <- sapply(reduced_test_data1, function(x) all(x %in% c(0, 1)) & is.numeric(x))

reduced_test_data1[binary_cols_test_reduced] <- lapply(reduced_test_data1[binary_cols_test_reduced], as.factor)
reduced_test_data2[binary_cols_test_reduced] <- lapply(reduced_test_data2[binary_cols_test_reduced], as.factor)
reduced_test_data3[binary_cols_test_reduced] <- lapply(reduced_test_data3[binary_cols_test_reduced], as.factor)

reduced_test_data1 <- reduced_test_data1[, !(names(reduced_test_data1) %in% c("time_to_outcome_afib_aflutter_new_post"))]
reduced_test_data2 <- reduced_test_data2[, !(names(reduced_test_data2) %in% c("time_to_outcome_afib_aflutter_new_post"))]
reduced_test_data3 <- reduced_test_data3[, !(names(reduced_test_data3) %in% c("time_to_outcome_afib_aflutter_new_post"))]

reduced_test1_response<-reduced_test_data1$outcome_afib_aflutter_new_post
reduced_test2_response<-reduced_test_data2$outcome_afib_aflutter_new_post
reduced_test3_response<-reduced_test_data3$outcome_afib_aflutter_new_post

binary_cols_under_reduced <- sapply(reduced_under_data2, function(x) all(x %in% c(0, 1)) & is.numeric(x))

#reduced_under_data1[binary_cols_under_reduced] <- lapply(reduced_under_data1[binary_cols_under_reduced], as.factor)
reduced_under_data2[binary_cols_under_reduced] <- lapply(reduced_under_data2[binary_cols_under_reduced], as.factor)
reduced_under_data3[binary_cols_under_reduced] <- lapply(reduced_under_data3[binary_cols_under_reduced], as.factor)

#reduced_under_data1 <- reduced_under_data1[, !(names(reduced_under_data1) %in% c("time_to_outcome_afib_aflutter_new_post"))]
reduced_under_data2 <- reduced_under_data2[, !(names(reduced_under_data2) %in% c("time_to_outcome_afib_aflutter_new_post"))]
reduced_under_data3 <- reduced_under_data3[, !(names(reduced_under_data3) %in% c("time_to_outcome_afib_aflutter_new_post"))]

binary_cols_over_reduced <- sapply(reduced_over_data1, function(x) all(x %in% c(0, 1)) & is.numeric(x))

reduced_over_data1[binary_cols_over_reduced] <- lapply(reduced_over_data1[binary_cols_over_reduced], as.factor)
```

```r
reduced_over_data2[binary_cols_over_reduced] <- lapply(reduced_over_data2[binary_cols_over_reduced], as.factor)
reduced_over_data3[binary_cols_over_reduced] <- lapply(reduced_over_data3[binary_cols_over_reduced], as.factor)

reduced_over_data1 <- reduced_over_data1[, !(names(reduced_over_data1) %in% c("time_to_outcome_afib_aflutter_new_post"))]
reduced_over_data2 <- reduced_over_data2[, !(names(reduced_over_data2) %in% c("time_to_outcome_afib_aflutter_new_post"))]
reduced_over_data3 <- reduced_over_data3[, !(names(reduced_over_data3) %in% c("time_to_outcome_afib_aflutter_new_post"))]

###Validation sets
# Re factors the factored rows and prepare validation set
valid_data1$demographics_birth_sex <- valid_data1$demographics_birth_sex - 1
valid_data2$demographics_birth_sex <- valid_data2$demographics_birth_sex - 1
valid_data3$demographics_birth_sex <- valid_data3$demographics_birth_sex - 1

# Find columns that are factors
binary_cols_valid <- sapply(valid_data1, function(x) all(x %in% c(0, 1)) & is.numeric(x))

# Factor these columns
valid_data1[binary_cols_valid] <- lapply(valid_data1[binary_cols_valid], as.factor)
valid_data2[binary_cols_valid] <- lapply(valid_data2[binary_cols_valid], as.factor)
valid_data3[binary_cols_valid] <- lapply(valid_data3[binary_cols_valid], as.factor)

# Removing time variable
valid_data1 <- valid_data1[, !(names(valid_data1) %in% c("time_to_outcome_afib_aflutter_new_post"))]
valid_data2 <- valid_data2[, !(names(valid_data2) %in% c("time_to_outcome_afib_aflutter_new_post"))]
valid_data3 <- valid_data3[, !(names(valid_data3) %in% c("time_to_outcome_afib_aflutter_new_post"))]



### K-Prototypes ###


#For DBI Index to find optimal # of clusters
Davies.Bouldin <- function(A, SS, m) {
  # A - the centres of the clusters
  # SS - the within sum of squares
  # m - the sizes of the clusters
  N <- nrow(A) # number of clusters
  # intercluster distance
  S <- sqrt(SS/m)
  # Get the distances between centres
  M <- as.matrix(dist(A))
  # Get the ratio of intercluster/centre.dist
  R <- matrix(0, N, N)
  for (i in 1:(N-1)) {
    for (j in (i+1):N) {
      R[i,j] <- (S[i] + S[j])/M[i,j]
      R[j,i] <- R[i,j]
    }
  }
  return(mean(apply(R, 1, max)))
}
```

```r

colours <- rainbow(14)

#Creating vectors for the errors and DBI
errs <- numeric(7)
DBI <- numeric(7)

#Running K-Prototypes on 2 to 8 clusters
for (i in 2:8) {
  KP <- kproto(train_data1, i, iter.max=15)
  errs[i-1] <- sum(KP$withinss)
  DBI[i-1] <- Davies.Bouldin(KP$centers, KP$withinss,KP$size)
  print(KP$centers)

}

#Plotting sum of squares
plot(2:8, errs, type = "b", pch = 20, cex = 1.5, lwd = 2, main = "Sum of Squares", xlab
     = "Number of Clusters", ylab = "SS")

#Plotting DBI
plot(2:8, DBI, type = "b", pch = 20, cex = 1.5, lwd = 2, main = "Davies-Bouldin Index",
     xlab = "Number of Clusters", ylab = "DBI")

#Kproto for 2 clusters
kproto2_train_data1<-kproto(train_data1, 2, iter.max=15)
kproto3_train_data1<-kproto(train_data1, 3, iter.max=15)
kproto6_train_data1<-kproto(train_data1, 6, iter.max=15)


###Function to summarize the clusters###
summarize_clusters<-function(kproto_dat,dat){

  #Label the clusters in the data
  cluster_labels <- kproto_dat$cluster
  dat$cluster <- cluster_labels

  #mean_test<-aggregate(. ~ cluster, data = train_data1, FUN = function(x) if(is.
    numeric(x)) mean(x) else NA)
  #This will either take the mean of numeric columns, or sum the factors by cluster
  mean_test <- dat %>%
    group_by(cluster) %>%
    summarise(across(
      everything(),
      ~ if (is.numeric(.x)) {
        mean(.x, na.rm = TRUE)
      } else if (all(levels(as.factor(.x)) %in% c("0", "1"))) {
        paste0("0: ", sum(.x == 0), ", 1: ", sum(.x == 1))
      } else {
        NA_character_
      },
      .names = "{.col}"
    ))


  #Here will identify the variables with highest difference in mean
  numeric_means <- mean_test %>%
    select(where(is.numeric)) %>%
    summarise(across(everything(), ~ max(.x) - min(.x)))

```

```r
  # Create a "row" with range values, labeled as 'range' in the cluster column
  range_row <- numeric_means
  range_row$cluster <- "range"
  range_row <- range_row %>% select(cluster, everything())

  # Add to the bottom of the summary
  mean_test$cluster <- as.character(mean_test$cluster)
  mean_test <- bind_rows(mean_test, range_row)


  return(mean_test)
}

#Summarize findings
kproto2_train_data1_Summary<-summarize_clusters(kproto2_train_data1,train_data1)
kproto3_train_data1_Summary<-summarize_clusters(kproto3_train_data1,train_data1)
kproto6_train_data1_Summary<-summarize_clusters(kproto6_train_data1,train_data1)



### GLM Logistic Regression ###



#Function to run Logistic Regression
run_GLM<-function(train_set,test_set_no_response,test_response){

  log_model<-glm(outcome_afib_aflutter_new_post ~ .,data=train_set, quasibinomial(link=
    "logit"))
  print(summary(log_model))

  #Get the 10 largest coefficients
  train_coefs<-coef(log_model)
  #Remove intercept
  train_coefs<-train_coefs[names(train_coefs) != "(Intercept)"]
  top10_train <- sort(abs(train_coefs), decreasing = TRUE)[1:10]
  top10_named_train <- train_coefs[names(top10_train)]
  print("10 Largest coefficients in training model:")
  print(top10_named_train)

  #Plotting
  # Fix: Strip dummy suffixes like '1' from factor names
  top_var <- names(top10_named_train)[1]
  clean_var <- gsub("1$", "", top_var)  # removes trailing '1' only

  # Check if it's in the original data
  if (!(clean_var %in% colnames(train_set))) {
    stop(paste("Cleaned variable", clean_var, "not found in dataset."))
  }


  # Add predictions to training set
  train_set$predicted_prob <- predict(log_model, type = "response")
  train_set$predicted_class <- ifelse(train_set$predicted_prob > 0.5, 1, 0)
  train_set$correct <- ifelse(train_set$predicted_class == train_set$outcome_afib_
    aflutter_new_post, "Correct", "Incorrect")

  # Plot
  ggplot(train_set, aes(x = !!sym(clean_var), y = outcome_afib_aflutter_new_post, color
    = correct)) +
```

```
    geom_jitter(width = 0.2, height = 0.05, alpha = 0.7) +
    geom_smooth(method = "glm", method.args = list(family = "binomial"), se = FALSE,
   color = "black") +
    labs(
      title = paste("Response vs", clean_var, "colored by prediction accuracy"),
      x = clean_var,
      y = "Outcome",
      color = "Prediction"
    ) +
    theme_minimal()

  # ---- Confusion Matrix ----
  cat("\nTraining Confusion Matrix:\n")
  cm_train <- confusionMatrix(factor(train_set$predicted_class), factor(train_set$
    outcome_afib_aflutter_new_post))
  print(cm_train)

  # ---- AUC ----
  roc_train <- roc(train_set$outcome_afib_aflutter_new_post, train_set$predicted_prob)
  auc_train <- auc(roc_train)
  cat("\nTraining AUC: ", round(auc_train, 4), "\n")

}

#Running Logistic Regression on training sets & test sets & validation sets
run_GLM(train_data1,test_data1,test1_response)
run_GLM(train_data2,test_data2,test2_response)
run_GLM(train_data3,test_data3,test3_response)

run_GLM(train_data1, valid_data1, valid1_response)
run_GLM(train_data2, valid_data2, valid2_response)
run_GLM(train_data3, valid_data3, valid3_response)

run_GLM(reduced_train_data1,reduced_test_data1,reduced_test1_response)
run_GLM(reduced_train_data2,reduced_test_data2,reduced_test2_response)
run_GLM(reduced_train_data3,reduced_test_data3,reduced_test3_response)

#library(randomForest)
#resp_col<-over_train_data1$outcome_afib_aflutter_new_post
#over_reduced_data1<-as.data.frame(over_reduced_data1)
#RF_model<-randomForest(resp_col~.,data=over_reduced_data1,ntree=100,importance = TRUE,
   proximity=TRUE)

library(class)

#turns every column in dataset to numeric
numeric_cols_train <- sapply(over_train_data1, is.numeric)
#takes every column that isn't of 1 class
non_constant_cols_train <- sapply(over_train_data1[, numeric_cols_train], function(x)
   length(unique(x)) > 1)
#Takes columns that are both numeric and non-constant from train dataset
train_selected <- names(over_train_data1[, numeric_cols_train][, non_constant_cols_
   train])

#Dataset of the numerical data
numeric_cols_test <- sapply(valid_data1, is.numeric)
#Takes columns that are both numeric and non-constant from validation dataset
test_selected <- names(valid_data1[, numeric_cols_test])
#Creates list of variables that are in both
common_cols <- intersect(train_selected, test_selected)
```

```r

#Makes dataset of selected columns from training dataset
train_input <- over_train_data1[, common_cols]
#Takes our repsonse variable from training dataset as vector and factoring it
train.class <- factor(over_train_data1$outcome_afib_aflutter_new_post)

#Gets principle components to reduce number of columns
pca_model <- prcomp(train_input, center = TRUE, scale. = TRUE)
#Take the first 61 components (70% cumulative variance)
train.data <- pca_model$x[, 1:61]

#Makes dataset of selected columns from validation dataset
valid_input <- valid_data1[, common_cols]
#Take the first 61 components of validation pca dataset
valid.data <- predict(pca_model, newdata = valid_input)[, 1:61]
#Takes our repsonse variable from validation dataset as vector
test.class <- factor(valid_data1$outcome_afib_aflutter_new_post)

#Run KNN to classify the data into the 2 classes
knn_pred<-knn(train=train.data,test=valid.data,cl=train.class,k=5,prob=TRUE)

#Create the confusion matrix from the knn classification model
confusionMatrix(knn_pred,test.class)

library(pROC)

#Calculate the AUC of our KNN model
roc_obj<-roc(test.class,attributes(knn_pred)$prob)
(auc_val<-auc(roc_obj))

library(xgboost)

#subtract from train and testing/validation response varaibles to ensure the classes
    are either 0 or 1
train.class<-as.numeric(train.class)-1
test.class<-as.numeric(test.class)-1

#Get the XGBoost training and validation dataset
data_train<-xgb.DMatrix(data=as.matrix(train.data),label=train.class)
data_valid<-xgb.DMatrix(data=as.matrix(valid.data),label=test.class)

#Create the parameters for the xgb model
parameters<-list(
  objective="multi:softprob",
  num_class=2,
  eval_metric="mlogloss"
)

#Create the XGBoost model with ur training and testing dataset
xgb_model<-xgb.train(params=parameters,data=data_train,nrounds=50,watchlist=list(train=
    data_train,eval=data_valid),verbose=0)

#Get the prediction probabilities of the model and create the prediction matrix
pred_probs<-predict(xgb_model,data_valid)
pred_matrix<-matrix(pred_probs, ncol=2, byrow = TRUE)
pred_col<-max.col(pred_matrix)-1

#Create the confusion matrix from the XGBoost classification model
confusionMatrix(factor(pred_col),factor(test.class))

```

```r
#Calculate the AUC from the XGBoost model
roc_obj <- roc(test.class,pred_col)
(auc_val <- auc(roc_obj))


library(dplyr)
library(ggplot2)
library(nnet)
library(tensorflow)
library(pROC)


#Basic preparing for neural net, all values must be integers, and factors must all be zeroes and ones "dummy variables".
#Training sets
NN_train_data1 <- train_data1[, !(names(train_data1) %in% c("outcome_afib_aflutter_new_post", "time_to_outcome_afib_aflutter_new_post"))]
NN_train_data2 <- train_data2[, !(names(train_data2) %in% c("outcome_afib_aflutter_new_post", "time_to_outcome_afib_aflutter_new_post"))]
NN_train_data3 <- train_data3[, !(names(train_data3) %in% c("outcome_afib_aflutter_new_post", "time_to_outcome_afib_aflutter_new_post"))]


NN_train_data1 <- as.data.frame(lapply(NN_train_data1, function(x) as.numeric(as.character(x))))
NN_train_data2 <- as.data.frame(lapply(NN_train_data2, function(x) as.numeric(as.character(x))))
NN_train_data3 <- as.data.frame(lapply(NN_train_data3, function(x) as.numeric(as.character(x))))

NN_train_data1$demographics_birth_sex<-NN_train_data1$demographics_birth_sex-1
NN_train_data2$demographics_birth_sex<-NN_train_data2$demographics_birth_sex-1
NN_train_data3$demographics_birth_sex<-NN_train_data3$demographics_birth_sex-1

NN_train1_output<-as.numeric(train_data1$outcome_afib_aflutter_new_post)
NN_train2_output<-as.numeric(train_data2$outcome_afib_aflutter_new_post)
NN_train3_output<-as.numeric(train_data3$outcome_afib_aflutter_new_post)


#Test sets
NN_test_data1 <- test_data1[, !(names(test_data1) %in% c("outcome_afib_aflutter_new_post", "time_to_outcome_afib_aflutter_new_post"))]
NN_test_data2 <- test_data2[, !(names(test_data2) %in% c("outcome_afib_aflutter_new_post", "time_to_outcome_afib_aflutter_new_post"))]
NN_test_data3 <- test_data3[, !(names(test_data3) %in% c("outcome_afib_aflutter_new_post", "time_to_outcome_afib_aflutter_new_post"))]


NN_test_data1 <- as.data.frame(lapply(NN_test_data1, function(x) as.numeric(as.character(x))))
NN_test_data2 <- as.data.frame(lapply(NN_test_data2, function(x) as.numeric(as.character(x))))
NN_test_data3 <- as.data.frame(lapply(NN_test_data3, function(x) as.numeric(as.character(x))))

NN_test_data1$demographics_birth_sex<-NN_test_data1$demographics_birth_sex-1
NN_test_data2$demographics_birth_sex<-NN_test_data2$demographics_birth_sex-1
NN_test_data3$demographics_birth_sex<-NN_test_data3$demographics_birth_sex-1

NN_test1_output<-as.numeric(test_data1$outcome_afib_aflutter_new_post)
```

```
NN_test2_output<-as.numeric(test_data2$outcome_afib_aflutter_new_post)
NN_test3_output<-as.numeric(test_data3$outcome_afib_aflutter_new_post)

###Reduced sets
#Training sets
NN_reduced_train_data1 <- reduced_train_data1[, !(names(reduced_train_data1) %in% c("
    outcome_afib_aflutter_new_post", "time_to_outcome_afib_aflutter_new_post"))]
NN_reduced_train_data2 <- reduced_train_data2[, !(names(reduced_train_data2) %in% c("
    outcome_afib_aflutter_new_post", "time_to_outcome_afib_aflutter_new_post"))]
NN_reduced_train_data3 <- reduced_train_data3[, !(names(reduced_train_data3) %in% c("
    outcome_afib_aflutter_new_post", "time_to_outcome_afib_aflutter_new_post"))]

NN_reduced_train_data1 <- as.data.frame(lapply(NN_reduced_train_data1, function(x) as.
    numeric(as.character(x))))
NN_reduced_train_data2 <- as.data.frame(lapply(NN_reduced_train_data2, function(x) as.
    numeric(as.character(x))))
NN_reduced_train_data3 <- as.data.frame(lapply(NN_reduced_train_data3, function(x) as.
    numeric(as.character(x))))

NN_reduced_train1_output<-as.numeric(reduced_train_data1$outcome_afib_aflutter_new_post
    )
NN_reduced_train2_output<-as.numeric(reduced_train_data2$outcome_afib_aflutter_new_post
    )
NN_reduced_train3_output<-as.numeric(reduced_train_data3$outcome_afib_aflutter_new_post
    )

#Test sets
NN_reduced_test_data1 <- reduced_test_data1[, !(names(reduced_test_data1) %in% c("
    outcome_afib_aflutter_new_post", "time_to_outcome_afib_aflutter_new_post"))]
NN_reduced_test_data2 <- reduced_test_data2[, !(names(reduced_test_data2) %in% c("
    outcome_afib_aflutter_new_post", "time_to_outcome_afib_aflutter_new_post"))]
NN_reduced_test_data3 <- reduced_test_data3[, !(names(reduced_test_data3) %in% c("
    outcome_afib_aflutter_new_post", "time_to_outcome_afib_aflutter_new_post"))]

NN_reduced_test_data1 <- as.data.frame(lapply(NN_reduced_test_data1, function(x) as.
    numeric(as.character(x))))
NN_reduced_test_data2 <- as.data.frame(lapply(NN_reduced_test_data2, function(x) as.
    numeric(as.character(x))))
NN_reduced_test_data3 <- as.data.frame(lapply(NN_reduced_test_data3, function(x) as.
    numeric(as.character(x))))

NN_reduced_test1_output<-as.numeric(reduced_test_data1$outcome_afib_aflutter_new_post)
NN_reduced_test2_output<-as.numeric(reduced_test_data2$outcome_afib_aflutter_new_post)
NN_reduced_test3_output<-as.numeric(reduced_test_data3$outcome_afib_aflutter_new_post)

#Under and over sampled sets
#Oversampled
NN_reduced_over_data1 <- reduced_over_data1[, !(names(reduced_over_data1) %in% c("
    outcome_afib_aflutter_new_post", "time_to_outcome_afib_aflutter_new_post"))]
NN_reduced_over_data2 <- reduced_over_data2[, !(names(reduced_over_data2) %in% c("
    outcome_afib_aflutter_new_post", "time_to_outcome_afib_aflutter_new_post"))]
NN_reduced_over_data3 <- reduced_over_data3[, !(names(reduced_over_data3) %in% c("
    outcome_afib_aflutter_new_post", "time_to_outcome_afib_aflutter_new_post"))]

NN_reduced_over_data1 <- as.data.frame(lapply(NN_reduced_over_data1, function(x) as.
    numeric(as.character(x))))
NN_reduced_over_data2 <- as.data.frame(lapply(NN_reduced_over_data2, function(x) as.
    numeric(as.character(x))))
NN_reduced_over_data3 <- as.data.frame(lapply(NN_reduced_over_data3, function(x) as.
    numeric(as.character(x))))
```

```
821
822 NN_reduced_over1_output<-as.numeric(reduced_over_data1$outcome_afib_aflutter_new_post)
823 NN_reduced_over2_output<-as.numeric(reduced_over_data2$outcome_afib_aflutter_new_post)
824 NN_reduced_over3_output<-as.numeric(reduced_over_data3$outcome_afib_aflutter_new_post)
825
826 #Undersampled
827 #One of these (either under 1 2 or 3) may decide not to work! If that is the case just
    run line by line, the error seems to randomly choose one of the three datasets not
    to work under pca.
828 #NN_reduced_under_data1 <- reduced_under_data1[, !(names(reduced_under_data1) %in% c("
    outcome_afib_aflutter_new_post", "time_to_outcome_afib_aflutter_new_post"))]
829 NN_reduced_under_data2 <- reduced_under_data2[, !(names(reduced_under_data2) %in% c("
    outcome_afib_aflutter_new_post", "time_to_outcome_aflutter_new_post"))]
830 NN_reduced_under_data3 <- reduced_under_data3[, !(names(reduced_under_data3) %in% c("
    outcome_afib_aflutter_new_post", "time_to_outcome_aflutter_new_post"))]
831
832 #NN_reduced_under_data1 <- as.data.frame(lapply(NN_reduced_under_data1, function(x) as.
    numeric(as.character(x))))
833 NN_reduced_under_data2 <- as.data.frame(lapply(NN_reduced_under_data2, function(x) as.
    numeric(as.character(x))))
834 NN_reduced_under_data3 <- as.data.frame(lapply(NN_reduced_under_data3, function(x) as.
    numeric(as.character(x))))
835
836 #NN_reduced_under1_output <- as.numeric(reduced_under_data1$outcome_afib_aflutter_new_
    post)
837 NN_reduced_under2_output <- as.numeric(reduced_under_data2$outcome_afib_aflutter_new_
    post)
838 NN_reduced_under3_output <- as.numeric(reduced_under_data3$outcome_afib_aflutter_new_
    post)
839
840 ###Validation sets
841 NN_valid_data1 <- valid_data1[, !(names(valid_data1) %in% c("outcome_afib_aflutter_new_
    post", "time_to_outcome_afib_aflutter_new_post"))]
842 NN_valid_data2 <- valid_data2[, !(names(valid_data2) %in% c("outcome_afib_aflutter_new_
    post", "time_to_outcome_aflutter_new_post"))]
843 NN_valid_data3 <- valid_data3[, !(names(valid_data3) %in% c("outcome_afib_aflutter_new_
    post", "time_to_outcome_aflutter_new_post"))]
844
845 NN_valid_data1 <- as.data.frame(lapply(NN_valid_data1, function(x) as.numeric(as.
    character(x))))
846 NN_valid_data2 <- as.data.frame(lapply(NN_valid_data2, function(x) as.numeric(as.
    character(x))))
847 NN_valid_data3 <- as.data.frame(lapply(NN_valid_data3, function(x) as.numeric(as.
    character(x))))
848
849 NN_valid_data1$demographics_birth_sex <- NN_valid_data1$demographics_birth_sex - 1
850 NN_valid_data2$demographics_birth_sex <- NN_valid_data2$demographics_birth_sex - 1
851 NN_valid_data3$demographics_birth_sex <- NN_valid_data3$demographics_birth_sex - 1
852
853 NN_valid1_output <- as.numeric(valid_data1$outcome_afib_aflutter_new_post)
854 NN_valid2_output <- as.numeric(valid_data2$outcome_afib_aflutter_new_post)
855 NN_valid3_output <- as.numeric(valid_data3$outcome_afib_aflutter_new_post)
856
857
858
859 ###Function to run Neural Net###
860 run_NN<-function(dat_no_response,response,test_no_response=NA,test_response=NA) {
861
862   #Run Neural Net
863   NN<-nnet(dat_no_response, response, size=0, linout=TRUE, skip=TRUE, maxit=1000, trace
```

```
    =FALSE)

  ##Training set
  # Make predictions
  predictions <- predict(NN, dat_no_response)

  # Convert probabilities to binary output
  predicted_classes <- ifelse(predictions > 0.5, 1, 0)

  ##Test set
  # Make predictions
  predictions1 <- predict(NN, test_no_response)

  # Convert probabilities to binary output
  predicted_classes1 <- ifelse(predictions1 > 0.5, 1, 0)

  #How many of the predicted are correct
  cat("\nNeural Net training Accuracy: ",sum(response==predicted_classes)/length(
    response)*100,"%",sep="")
  cat("\nNeural Net test Accuracy: ",sum(test_response==predicted_classes1)/length(test
    _response)*100,"%\n",sep="")

  #Storing and displaying the resulted predictions of neural net
  dat_with_predictions<-data.frame(dat_no_response, Response=response, Predicted=
    predicted_classes)
  test_dat_with_predictions<-data.frame(test_no_response, Response=test_response,
    Predicted=predicted_classes1)
  View(dat_with_predictions)
  View(test_dat_with_predictions)

  print(confusionMatrix(factor(predicted_classes), factor(response)))

  #AUC training set
  roc_train<-roc(response, predictions)
  auc_train<-auc(roc_train)
  cat("\nNeural Net training AUC: ", round(auc_train, 4), "\n")

  #AUC test set
  roc_test <- roc(test_response, predictions1)
  auc_test <- auc(roc_test)
  cat("Neural Net test AUC: ", round(auc_test, 4), "\n")

  #Return the neural net
  return(NN)

}

###Running function on all data.
NN_train1<-run_NN(NN_train_data1,NN_train1_output,NN_test_data1,NN_test1_output)
NN_train2<-run_NN(NN_train_data2,NN_train1_output,NN_test_data2,NN_test2_output)
NN_train3<-run_NN(NN_train_data3,NN_train1_output,NN_test_data3,NN_test3_output)

#Reduced training data tested with reduced test dimension data.
NN_reduced_train1<-run_NN(NN_reduced_train_data1,NN_reduced_train1_output,NN_reduced_
    test_data1,NN_reduced_test1_output)
NN_reduced_train2<-run_NN(NN_reduced_train_data2,NN_reduced_train2_output,NN_reduced_
    test_data2,NN_reduced_test2_output)
NN_reduced_train3<-run_NN(NN_reduced_train_data3,NN_reduced_train3_output,NN_reduced_
    test_data3,NN_reduced_test3_output)

```

```
#Reduced training data tested with non reduced test dimension data.
NN_reduced1_train1<-run_NN(NN_reduced_train_data1,NN_reduced_train1_output,NN_test_
    data1,NN_test1_output)
NN_reduced1_train2<-run_NN(NN_reduced_train_data2,NN_reduced_train2_output,NN_test_
    data2,NN_test2_output)
NN_reduced1_train3<-run_NN(NN_reduced_train_data3,NN_reduced_train3_output,NN_test_
    data3,NN_test3_output)

#Testing over and under sampled on reduced test data.
NN_reduced_over1<-run_NN(NN_reduced_over_data1,NN_reduced_over1_output,NN_reduced_test_
    data1,NN_reduced_test1_output)
NN_reduced_over2<-run_NN(NN_reduced_over_data2,NN_reduced_over2_output,NN_reduced_test_
    data2,NN_reduced_test2_output)
NN_reduced_over3<-run_NN(NN_reduced_over_data3,NN_reduced_over3_output,NN_reduced_test_
    data3,NN_reduced_test3_output)

#Run these three 1 at a time
#NN_reduced_under1 <- run_NN(NN_reduced_under_data1, NN_reduced_under1_output, NN_
    reduced_test_data1, NN_reduced_test1_output)
NN_reduced_under2 <- run_NN(NN_reduced_under_data2, NN_reduced_under2_output, NN_
    reduced_test_data2, NN_reduced_test2_output)
NN_reduced_under3 <- run_NN(NN_reduced_under_data3, NN_reduced_under3_output, NN_
    reduced_test_data3, NN_reduced_test3_output)

#Testing over and undersampled data on normal test data.
#Same as noted above, one of these six may decide not to work! If that is the case just
     run line by line, the error seems to randomly choose one of the three datasets not
    to work under pca.
NN_reduced1_over1<-run_NN(NN_reduced_over_data1,NN_reduced_over1_output,NN_test_data1,
    NN_test1_output)
NN_reduced1_over2<-run_NN(NN_reduced_over_data2,NN_reduced_over2_output,NN_test_data2,
    NN_test2_output)
NN_reduced1_over3<-run_NN(NN_reduced_over_data3,NN_reduced_over3_output,NN_test_data3,
    NN_test3_output)

#Run these three 1 at a time
#NN_reduced1_under1 <- run_NN(NN_reduced_under_data1, NN_reduced_under1_output, NN_test
    _data1, NN_test1_output)
NN_reduced1_under2 <- run_NN(NN_reduced_under_data2, NN_reduced_under2_output, NN_test_
    data2, NN_test2_output)
NN_reduced1_under3 <- run_NN(NN_reduced_under_data3, NN_reduced_under3_output, NN_test_
    data3, NN_test3_output)

#Validation sets
NN_train1V <- run_NN(NN_train_data1, NN_train1_output, NN_valid_data1, NN_valid1_output
    )
NN_train2V <- run_NN(NN_train_data2, NN_train2_output, NN_valid_data2, NN_valid2_output
    )
NN_train3V <- run_NN(NN_train_data3, NN_train3_output, NN_valid_data3, NN_valid3_output
    )
```