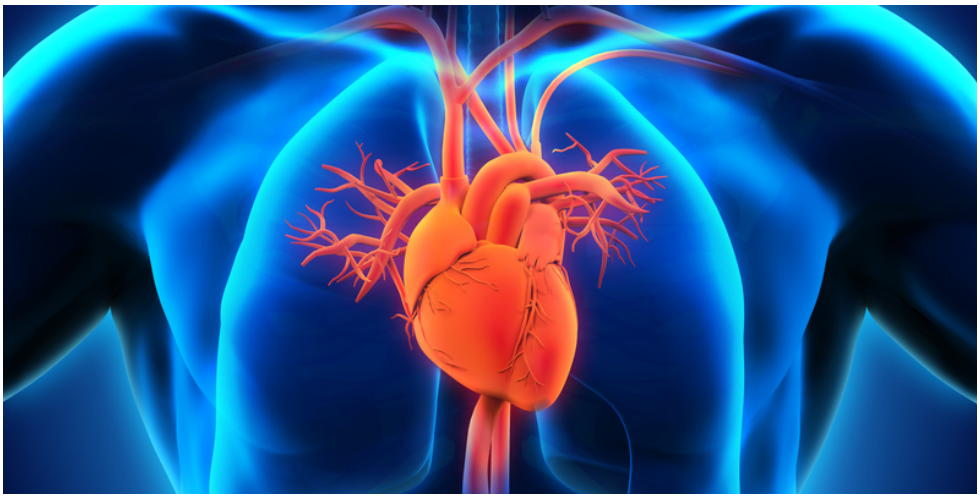
MA 5790 PREDICTIVE MODELING FINAL PROJECT

Michigan Tech

“Predicting Heart Disease”



Group 6

Bhavani Chalamalla

Emily Wood

Under the Guidance of

Qiuying Sha

Submission Date: December 2023

Introduction **3**
Variable Description 3
Pre-processing Data **4**
Categorical Predictor Pre-processing 5
Continuous Predictor Pre-processing 6
Correlation 8
After Pre-Processing 9
Splitting & Spending Data **10**
Model Building **10**
Linear Models 11
Non-Linear Models 12
Conclusion **15**
References **15**
Appendix 1: Linear Model Outputs **17**
Appendix 2: Non-Linear Model Outputs **19**
Appendix 3: R Code **22**

Introduction

According to the Centers for Disease Control and Prevention (CDC), heart disease is the leading cause of death for men and women in the United States, with one out of five deaths caused by heart disease - that is 695,000 deaths in 2021 [1]. Along with the large number of deaths, heart disease also costs the United States roughly $239.9 billion through health care and medicines.

There are different types of heart disease, with the most common being coronary artery disease and heart attacks. However, there are known medical conditions or lifestyles that are contributing factors to heart disease. Some of these factors are high blood pressure, high blood cholesterol, and smoking, while the lifestyle choices are diabetes, being overweight/obese, an unhealthy diet, physical inactivity, and excessive alcohol use. Because there are distinct factors and lifestyle choices that can lead to heart disease, our group is trying to predict whether someone has heart disease by looking at different predictors having to do with these prediction factors.

We chose the dataset “Heart Failure Prediction” from Kaggle, however, the dataset is a combination of 5 different heart datasets that have never been combined before [2]. The five different datasets come from the UCI Machine Learning Repository and are named as follows: Cleveland (303 samples), Hungarian (294 samples), Switzerland (123 samples), Long Beach VA (200 samples), and Stalog (Heart) Data Set (270 samples). This leaves us with a total of 1190 samples, however there were 272 duplicates that the authors removed. This leaves the final dataset with 918 samples for our group to analyze.

Variable Description

When downloaded from Kaggle, the dataset had a total of 12 columns, 1 of which was to be used as predictors and one as the response variable (HeartDisease). The table below contains the descriptions seen from the Kaggle website it was downloaded from.

Variable Name	Variable Description From Kaggle
Age	age of patient [years]
Sex	sex of patient [M: male, F: female]
ChesPainType	chest pain type [TA: typical angina, ATA: atypical angina, NAP: non-anginal pain, ASY: asymptomatic]
RestingBP	resting blood pressure [mm Hg]
Cholesterol	serum cholesterol [mm/dl]
FastingBS	fasting blood sugar [1: if FastingBS > 120 mg/dl, 0: otherwise]

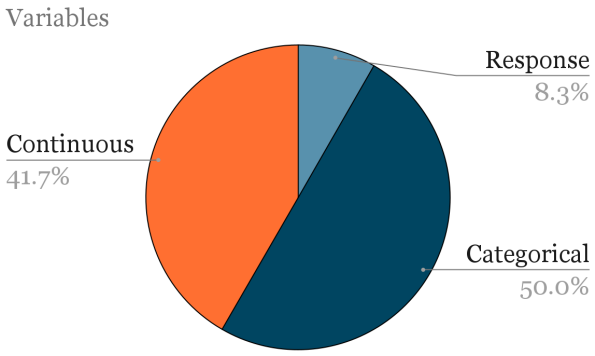
RestingECG	resting electrocardiogram results [Normal: normal, ST: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV), LVH: showing probable or definite left ventricular hypertrophy by Estes' criteria]
MaxHR	maximum heart rate achieved [Numeric value between 60 and 202]
ExerciseAngina	exercise-induced angina [Y: yes, N: no]
Oldpeak	oldpeak = ST [numeric value measure in depression]
ST_Slope	the slope of the peak exercise ST segment [Up: upsloping, Flat: flat, Down: downsloping]
HeartDisease	output class [1: heart disease, 0: normal]

Pre-processing Data

The table below contains a screenshot of what the data looked like. There were 12 variables in total, 11 predictors, and 918 observations.

	A	B	C	D	E	F	G	H	I	J	K	L
1	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
2	40	M	ATA	140	289	0	Normal	172	N	0	Up	0
3	49	F	NAP	160	180	0	Normal	156	N	1	Flat	1
4	37	M	ATA	130	283	0	ST	98	N	0	Up	0
5	48	F	ASY	138	214	0	Normal	108	Y	1.5	Flat	1
6	54	M	NAP	150	195	0	Normal	122	N	0	Up	0
7	39	M	NAP	120	339	0	Normal	170	N	0	Up	0

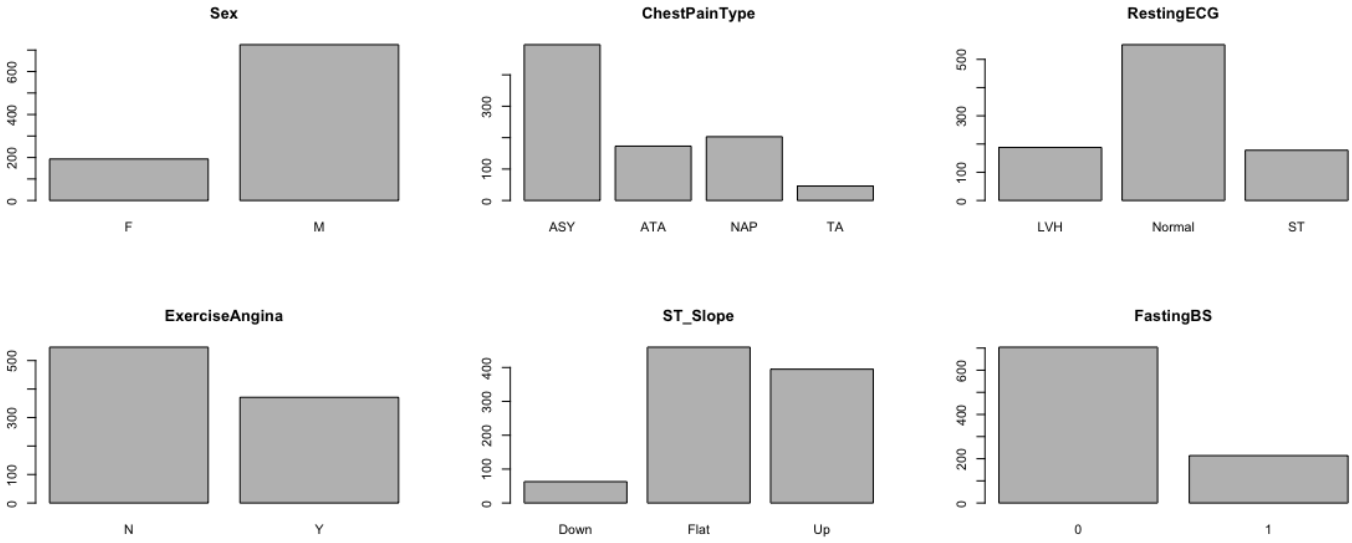
There are 3 variable types: categorical, continuous, and the response variable. Once split, there were 6 categorical variables (Sex, ChestPainType, FastingBS, RestingECG, ExerciseAngina, and ST_Slope), and 5 continuous predictors (Age, RestingBP, Cholesterol, MaxHR, and Oldpeak), and 1 response variable (HeartDisease).



Once establishing the characteristics of the variables, exploratory data analysis was performed. There were no missing values or duplicate rows found throughout the data.

Categorical Predictor Pre-processing

Without preprocessing, the categorical variable distribution is seen below in bar plots. There are 6 categorical predictors.



For the categorical predictors, near zero variance was assessed using the "nearZeroVar" function. **The function found that there were no near-zero variance predictors.**

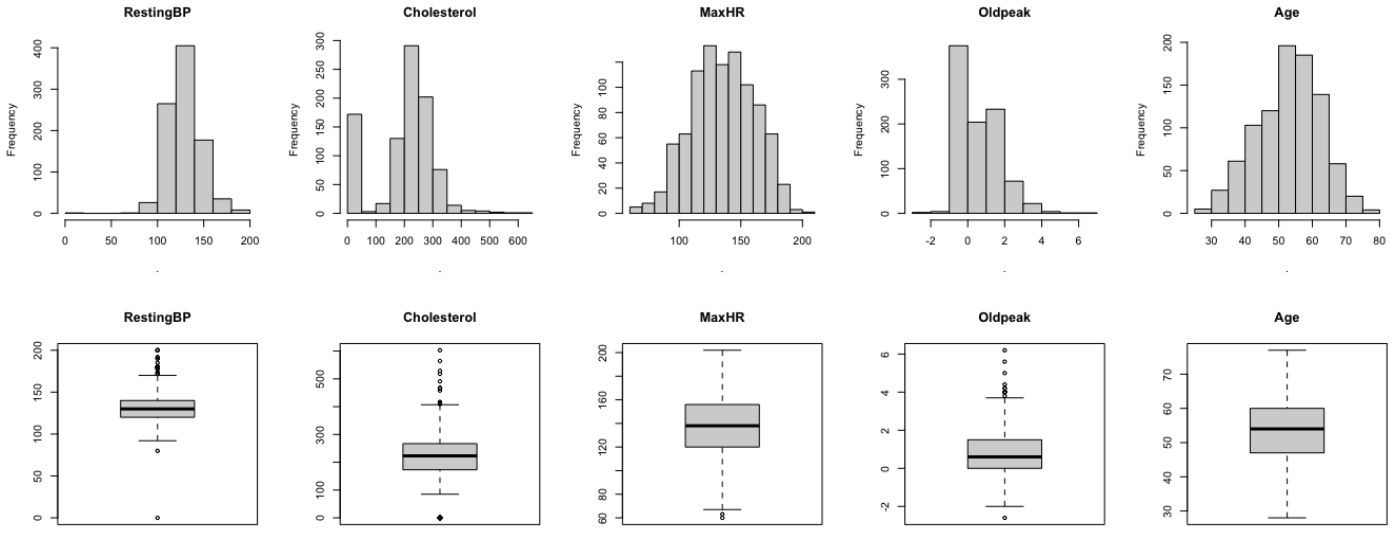
To continue exploratory data analysis, dummy variables were created for the categorical predictors and combined with the original dataset. The original categorical predictors, as well as binary predictors that had overlap (for example, SexF was deleted while SexM was kept) were deleted.

After preprocessing and deletion of duplicates, there were a total of 10 categorical columns.

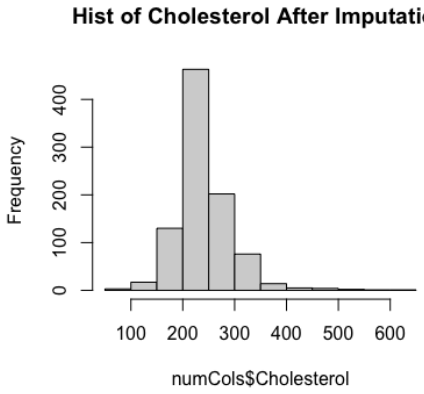
Continuous Predictor Pre-processing

The distributions of the continuous predictors were assessed using boxplots and histograms, seen below. The predictors MaxHR and Age appeared to have normal distributions with little to no outliers seen in the boxplots, however other predictors such as Cholesterol had odd histograms with multiple peaks. Resting BP and Oldepeak both had an intense skew that would need to also be dealt with. The skew of Oldpeak is the worst, also indicating that some sort of transformation will be needed, with Cholesterol to follow.

Skewness Before Pre-Processing				
RestingBP	Cholesterol	MaxHR	Oldpeak	Age
0.1795453	-0.6090891	-0.1441234	1.0211999	-0.1956127



In the histogram plot of Cholesterol there are two peaks, with one centered around 0. A cholesterol of 0 is not possible, leading us to believe that cholesterol values that were originally missing were instead filled in with 0. After deleting the 0s seen in cholesterol, a mean imputation was used to input the missing number. Below we can see the histograms, boxplots, and skew after imputation.



Skewness After Imputation
Cholesterol
1.371151

After imputation, the skew of cholesterol increased. This predictor will also need a transformation.

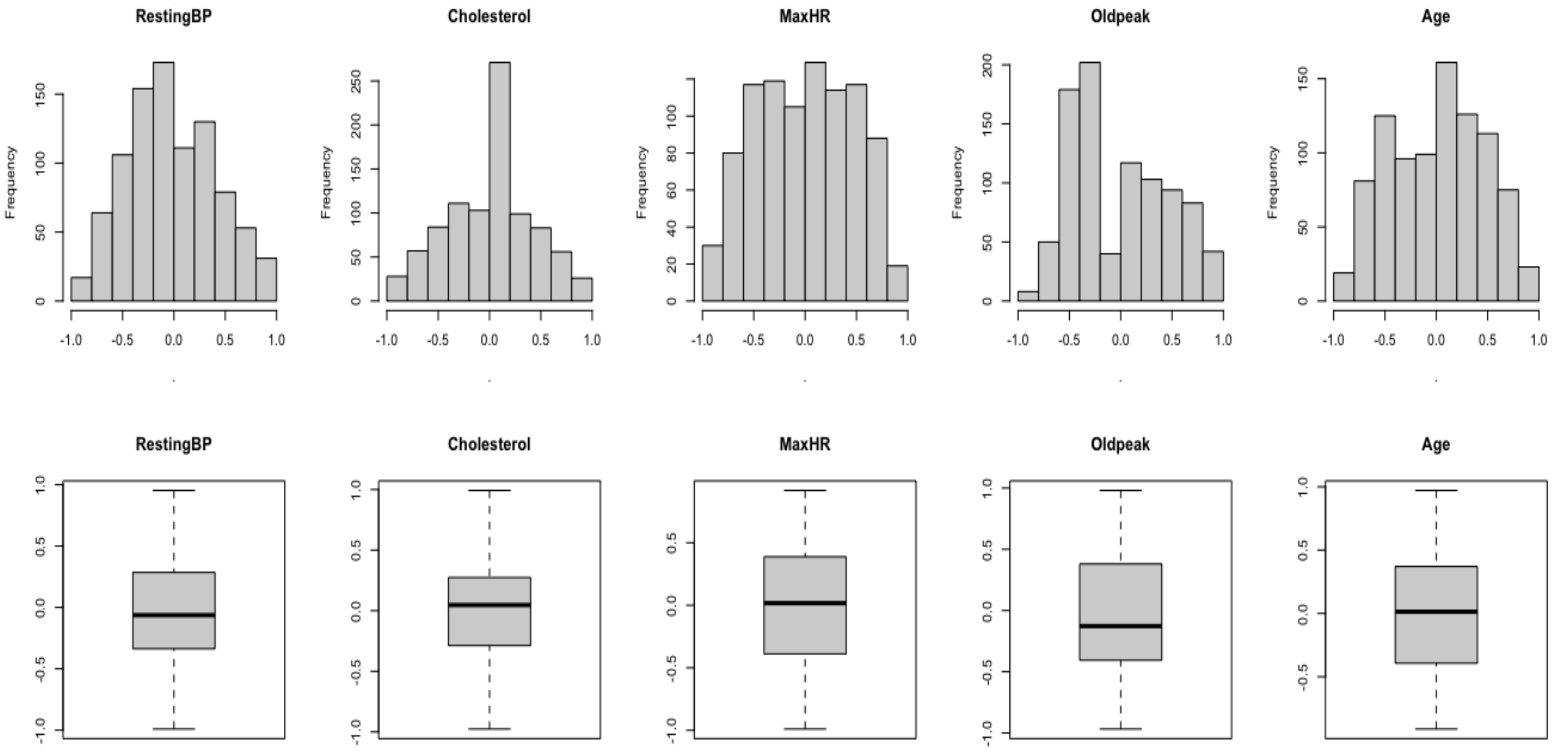
To conduct BoxCox transformations, there cannot be negative values within the dataset. Oldpeak is the only continuous variable that contains negative values, with a minimum value of 2.6. A constant value of 2.7 was added to Oldpeak so that BoxCox transformations could be done on the dataset.

While performing the BoxCox transformation, all continuous variables were also centered and scaled. The table below contains the BoxCox lambdas for transformation.

	RestingBP	Cholesterol	MaxHR	Oldpeak	Age
BoxCox Lambda	-	0.1	1.2	0.3	1.4

Finally, spatial sign for outliers is conducted. The following boxplots and histogram show the final continuous dataset that was used in model prediction.

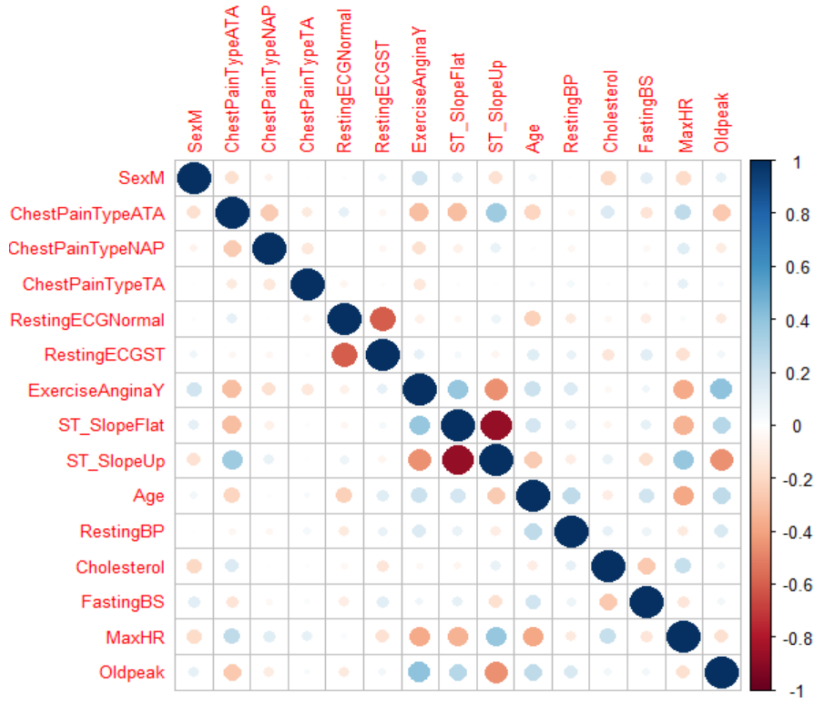
Skew After Pre-Processing				
RestingBP	Cholesterol	MaxHR	Oldpeak	Age
0.16871322	-0.06582053	-0.02827240	0.31156797	-0.02879606



After preprocessing, all predictors appear to have a more normal distribution with zero outliers. The skewness values are all below 0.35, and show that the distributions no longer have extreme skews.

Correlation

With the combination of dummy variables and deletion of duplicate predictors and all the preprocessed data, we then checked the correlation between all predictor columns, as seen below.



There are no predictors that have a correlation of greater than 0.8, so there are no multicollinearity issues.

After Pre-Processing

After preprocessing our dataset is left with **15 predictor columns, 10 of which are categorical and 5 continuous.** The names are listed below.

```
 [1] "SexM"
 [2] "ChestPainTypeATA"
 [3] "ChestPainTypeNAP"
 [4] "ChestPainTypeTA"
 [5] "RestingECGNormal"
 [6] "RestingECGST"
 [7] "ExerciseAnginaY"
 [8] "ST_SlopeFlat"
 [9] "ST_SlopeUp"
[10] "Age"
[11] "RestingBP"
[12] "Cholesterol"
[13] "FastingBS"
[14] "MaxHR"
[15] "Oldpeak"
```

Splitting & Spending Data

We split the data using a stratified split into 80% training and 20% testing data. The reason we used stratified was to ensure that the same number of classes were in testing and training. This left 735 samples to be used for training and 183 samples to be used for testing.

When training the data, we used a 10-fold cross-validation to create more robust models and training data.

Model Building

Categorical Outcome (Heart Disease)

The response variable (Heart Disease) in our heart failure prediction dataset is binary, taking on values of 0 and 1. Specifically, the response variable represents the presence or absence of heart disease, with 1 indicating the presence of heart disease and 0 indicating the absence. Given the binary nature of the response variable, our predictive modeling task is a classification problem. In classification, the goal is to develop a model that can accurately predict the class labels (in our case, the presence or absence of heart disease) based on the given set of predictors. So we chose to perform all Linear and Non-Linear Classification Models on our data.

Classification Models Performed	
Linear Models	**Non Linear Models**
• Logistic Regression	• Nonlinear Discriminant Analysis (MDA)
• Linear Discriminant Analysis (LDA)	• Regularized Discriminant Analysis (RDA)
• Partial Least Squares Discriminant Analysis (PLSDA)	• Neural Networks
• Penalized Model	• Flexible Discriminant Analysis (FDA)
• Nearest Shrunken Centroids	• Support Vector Machines (SVM)
	• K - Nearest Neighbors (KNN)
	• Naive Bayes

Linear Models

Below is a table of summaries for all models trained with a binary outcome. All parameters were tuned using 10-fold cross-validation. The resulting ROC, sensitivity, and specificity values were recorded from predicting the training set. The top (best-performing) model is highlighted and will be further explored. It's important to note that the optimistic results observed during the training set prediction might be indicative of the model's performance but should be validated on an independent test set to ensure robust generalization.

Linear Classification Models				
Model	**ROC**	**Sensitivity**	**Specificity**	**Best Tuning Parameter**
Logistic	0.9168953	0.8125	0.8756098	NA
LDA	0.918429	0.8094697	0.8829268	NA
PLSDA	0.9195607	0.8156250	0.8829268	ncomp = 4
Penalized	0.9209765	0.8032197	0.8853659	alpha = 0.2 lambda = 0.03111111
Nearest Shrunken Centroids	0.9098901	0.7693182	0.8902439	threshold = 0

Among the linear classification models evaluated for our heart failure prediction dataset, the penalized model stands out as the most promising, achieving a robust **ROC of 0.9210.** This model strikes a balance between **sensitivity (0.8032) and specificity (0.8854),** showcasing strong discriminative power in identifying both positive and negative cases. The model's optimization involves setting **alpha to 0.2 and lambda to 0.0311.** While other models like PLSDA and LDA also performed well with ROC values of 0.9196 and 0.9184, respectively, the penalized model demonstrates superior overall performance.

Non-Linear Models

The following are the nonlinear models that were trained on the same training split as the above models. The table below holds the metrics determining how well they performed.

Non-Linear Classification Models				
Model	**ROC**	**Sensitivity**	**Specificity**	**Best Tuning Parameters**
MDA	0.9184290	0.8094697	0.8829268	subclasses = 1
RDA	0.9070307	0.800094697	0.86585366	gamma=1, lambda=0
Neural Networks	0.9147866	0.8186553	0.8902439	size = 1 decay = 0.1, bag = T
FDA	0.9199545	0.8062500	0.8804878	degree = , nprune = 12
Support Vector Machines (SVM)	0.9244157	0.8034091	0.8780488	sigma = 0.046, c = 0.5
K - Nearest Neighbors (KNN)	0.9147889	0.8001894	0.8902439	k = 14
Naive Bayes	0.913447	0.7569129	0.9146341	NA

From the above table showing the performance of all non-linear models, the Support Vector Machine (SVM) emerges as the top-performing model with a high **ROC of 0.9244,** showcasing strong discriminative power. The **sensitivity and specificity values are 0.8034 and 0.8780,** respectively, reflecting a balanced ability to correctly identify both positive and negative cases. The SVM model is optimized with a **sigma value of 0.0460 and a cost parameter (c) of 0.5.** While the Flexible Discriminant Analysis (FDA) also demonstrates competitive performance with an ROC of 0.9199, its sensitivity and specificity are slightly lower at 0.8063 and 0.8805. Neural Networks, K-Nearest Neighbors (KNN), and Naive Bayes exhibit respectable but comparatively lower ROC values. Notably, the SVM model, with its superior performance metrics, stands out as the most promising choice in Non-Linear Models.

Model Evaluation and Comparison:

From Model fitting, we can clearly see that the top two best-performing models are Penalized and SVM. The top two models have then been used to predict on the test set and below are the resulting matrices.

1. Penalized

Confusion matrix of Penalized		
Predicted	**Actual**	
	No	**Yes**
No	75	10
Yes	10	88

The confusion matrix indicates that out of 183 instances, the GLM model correctly classified 75 instances as "No" and 88 instances as "Yes." It made 10 false positive predictions and 10 false negative predictions. The overall accuracy of the model is 89.07%, surpassing the No Information Rate significantly, with a statistically significant p-value. The Kappa value of 0.7803 suggests substantial agreement beyond chance. Sensitivity and specificity are 88.24% and 89.80%, respectively, demonstrating a balanced performance in correctly identifying both positive and negative cases. The model's overall performance is robust, as reflected in the balanced accuracy (Area Under the Curve) of 0.8902.

2. SVM

Confusion matrix of SVM		
Predicted	**Actual**	
	No	**Yes**
No	74	9
Yes	11	89

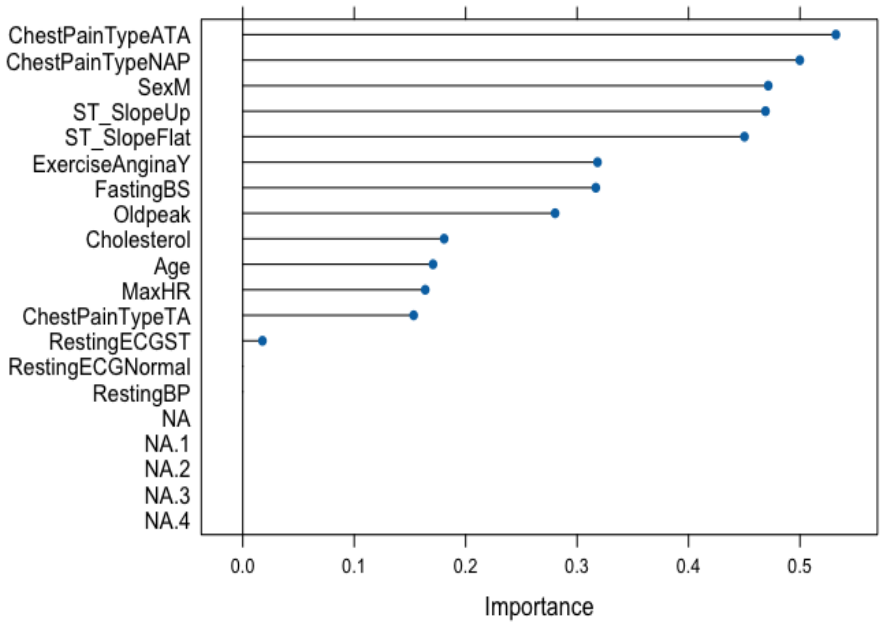
The confusion matrix indicates that out of 182 instances, the SVM model correctly classified 74 instances as "No" and 89 instances as "Yes." It made 11 false positive predictions and 9 false negative predictions. The overall accuracy of the model is 89.07%, exceeding the No Information Rate significantly, with a statistically significant p-value. The Kappa value of 0.78 suggests substantial agreement beyond chance. Sensitivity and specificity are 87.06% and

90.82%, respectively, demonstrating a balanced performance in correctly identifying both positive and negative cases. The model's overall performance is robust, as reflected in the balanced accuracy (Area Under the Curve) of 0.8894.

Top Two Models					
Model	**AUC**	**Sensitivity**	**Specificity**	**Accuracy**	**Kappa**
Penalized	0.8902	0.8824	0.8980	0.8907	0.7803
SVM	0.8894	0.8706	0.9082	0.8907	0.78

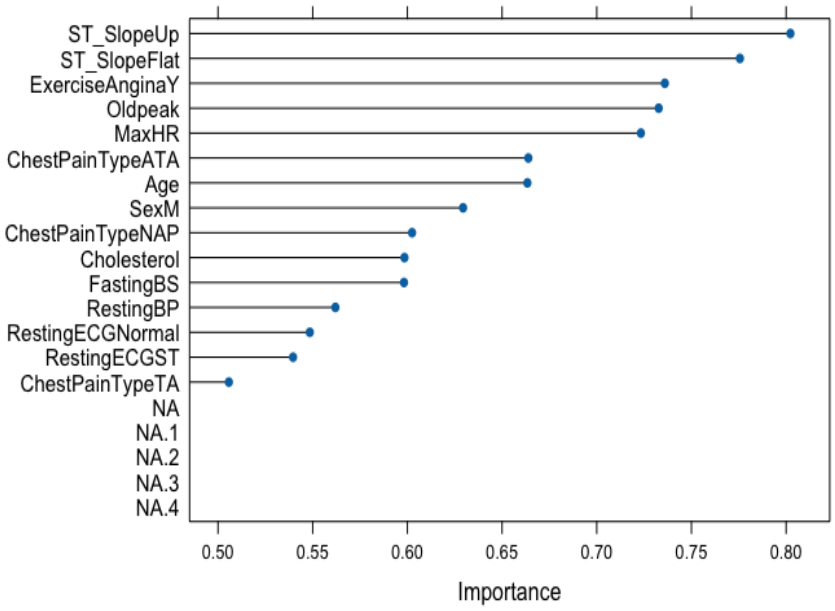
Variable Importance

1. Penalized



In the penalized model, the most crucial predictors for heart disease prediction are the type of chest pain (especially atypical angina and non-anginal pain **(100)**), gender (male) with **93.919**, the slope of the peak exercise ST segment (upward and flat)with **88.586 and 88.131**, exercise-induced angina, fasting blood sugar, and the presence of oldpeak. These factors contribute significantly to the model's ability to distinguish between individuals with and without heart disease. On the other hand, variables like resting ECG status, resting blood pressure, and certain chest pain types have comparatively lower importance in the model's predictions.

2. SVM



The most influential variables, according to their importance scores, include 'ST_SlopeUp' and 'ST_SlopeFlat' with scores of **100.00 and 91.01**, respectively, emphasizing the significance of the slope of the peak exercise ST segment. 'ExerciseAnginaY' and 'Oldpeak' follow closely with importance scores of **77.64 and 76.55**, underlining the impact of exercise-induced angina and ST depression during exercise on the model's predictions. 'MaxHR' (maximum heart rate achieved) and 'Age' contribute significantly with scores of **73.38 and 53.16**, respectively. Other predictors, such as chest pain types ('ChestPainTypeATA' and 'ChestPainTypeNAP'), gender ('SexM'), and cholesterol, also play substantial roles. Notably, 'ChestPainTypeTA' holds a zero importance score, indicating its minimal impact on the SVM model's predictive performance.

Conclusion

Comparing the confusion matrices and metrics of the Support Vector Machine (SVM) and Penalized (GLM), both models exhibit strong predictive performance for heart failure prediction. However, the GLM model slightly outperforms the SVM in several key metrics. The GLM model achieves a **higher sensitivity (87.06% for SVM vs. 88.24% for GLM)** and a **comparable specificity (90.82% for SVM vs. 89.80% for GLM)**, resulting in a **more balanced ability to correctly identify both positive and negative cases.** Additionally penalized model has a higher AUC **(0.8894 for SVM vs. 0.8902 for GLM)** than SVM. The GLM's superior sensitivity suggests that it has a stronger capacity to correctly identify individuals with heart disease, making it the preferred choice for heart failure prediction in this dataset.

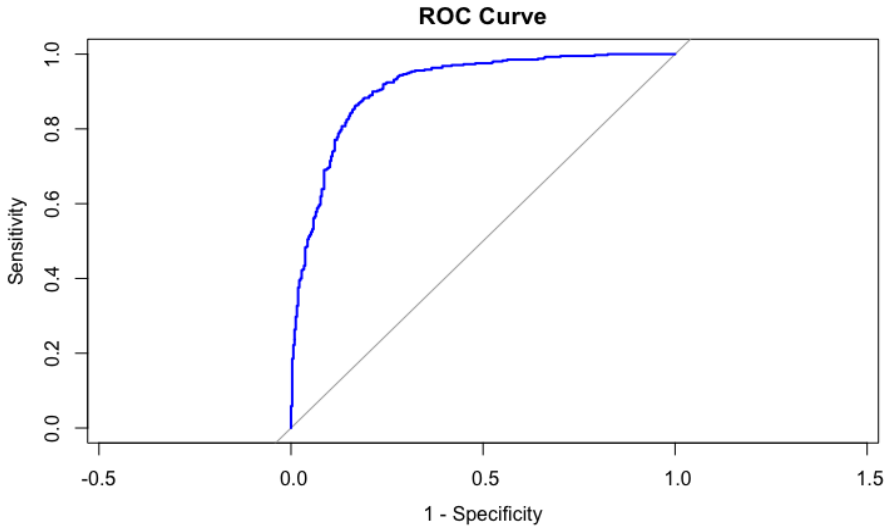
References

[1]CDC, “Heart Disease Facts,” *Centers for Disease Control and Prevention*, May 15, 2023. https://www.cdc.gov/heartdisease/facts.htm#:~:text=Heart%20disease%20is%20the%20leading

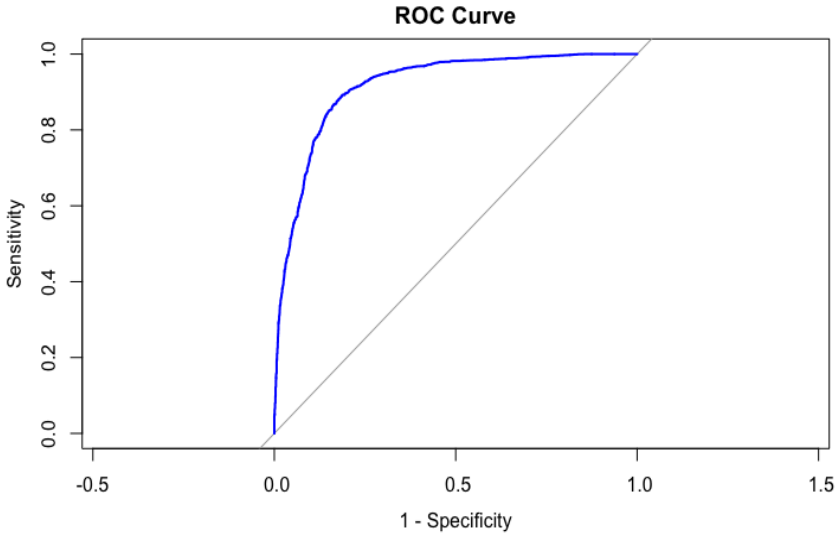
[2]]FEDESORIANO, “Heart Failure Prediction Dataset,” *www.kaggle.com*, 2021. https://www.kaggle.com/datasets/fedesoriano/heart-failure-prediction

Appendix 1: Linear Model Outputs

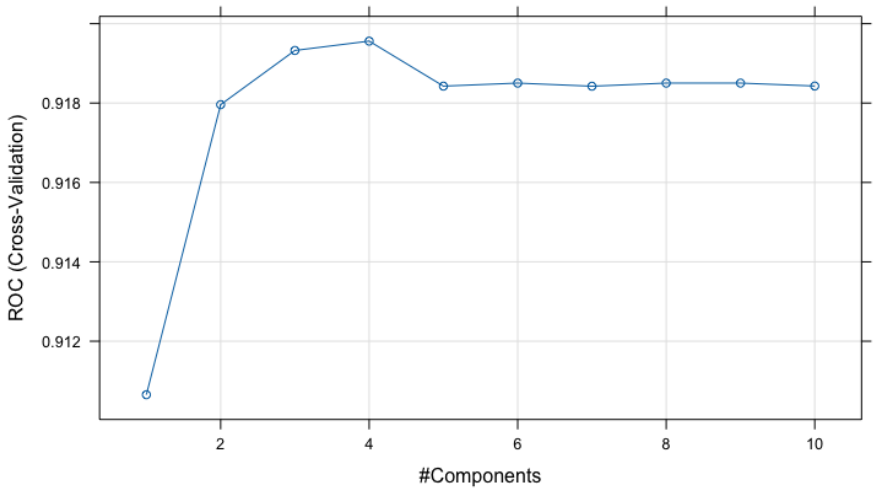
1) Logistic Regression
   ROC curve produced from logistic regression. The AUC for this model is roughly 0.917.



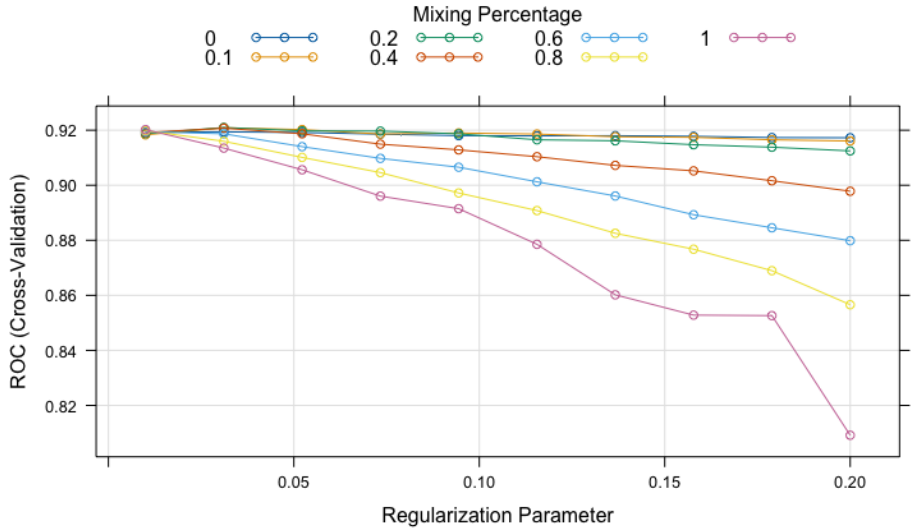
2) Linear Discriminant Analysis (LDA)
   ROC curve produced from Linear Discriminant Analysis (LDA). The AUC for this model is roughly 0.918.



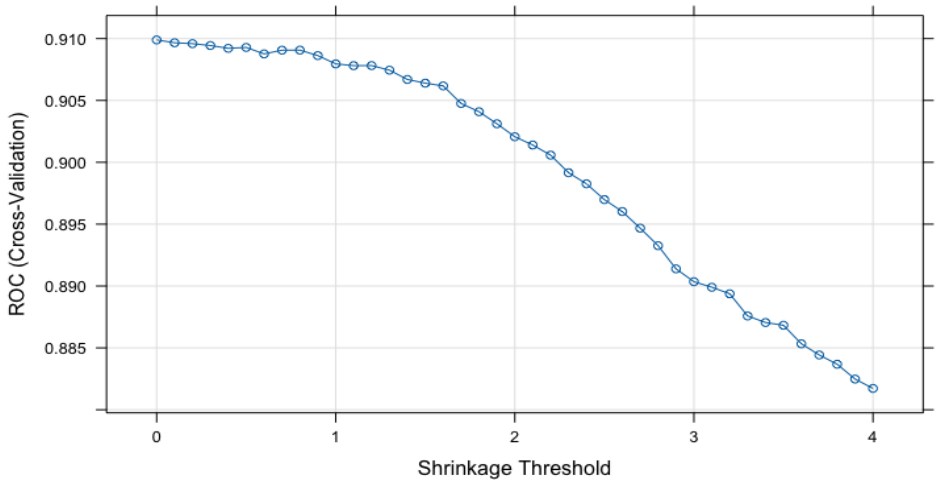
3) Partial Least Squares Discriminant Analysis (PLSDA)
The only tuning parameter for PLSDA is the number of components to be used. In this case, the best ncomp is 4, with an area under the curve of 0.921.



4) Penalized Models
The optimal tuning parameters for the penalized model were an alpha of 0.2 and lambda of 0.03111111, with an area under the curve of 0.921

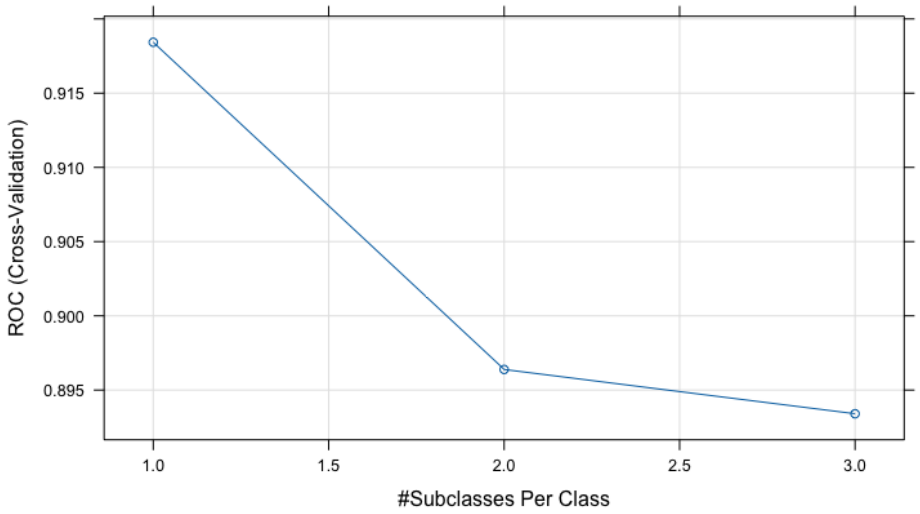


5) Nearest Shrunken Centroids
The best tuning parameters for nearest shrunken centroids were with a threshold held at 0. The area under the curve was 0.9099.

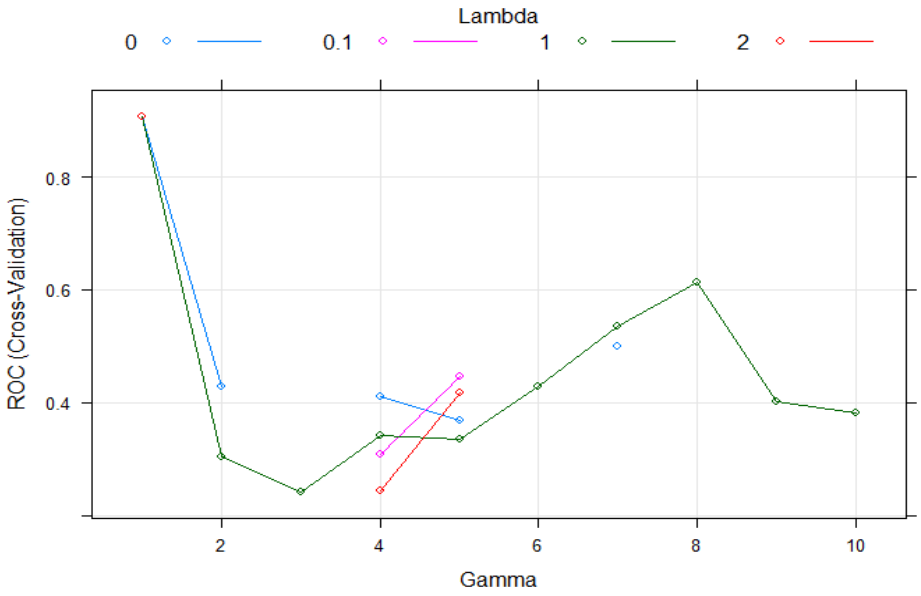


Appendix 2: Non-Linear Model Outputs

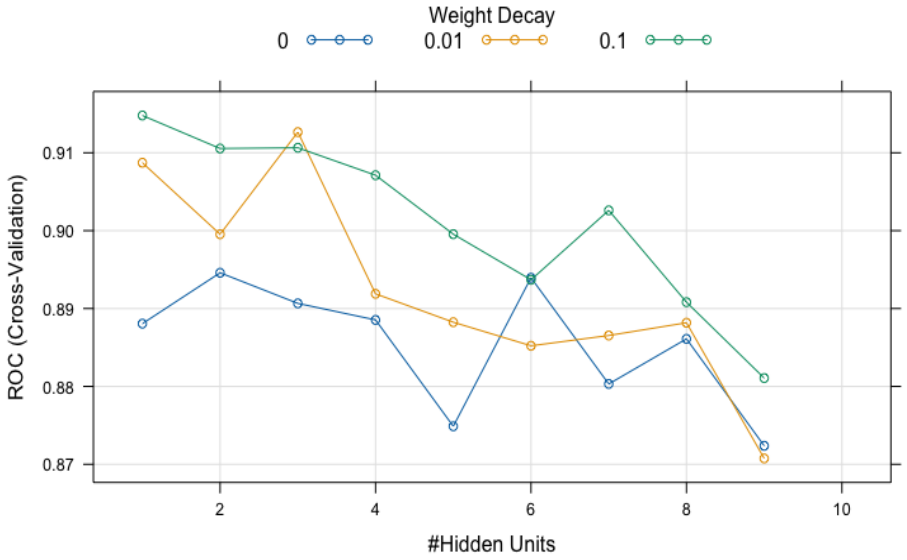
1) Nonlinear Discriminant Analysis (MDA)
The best tuning parameters for nonlinear discriminant analysis (MDA) were with subclasses equal to 1. The AUC for this model was 0.918.



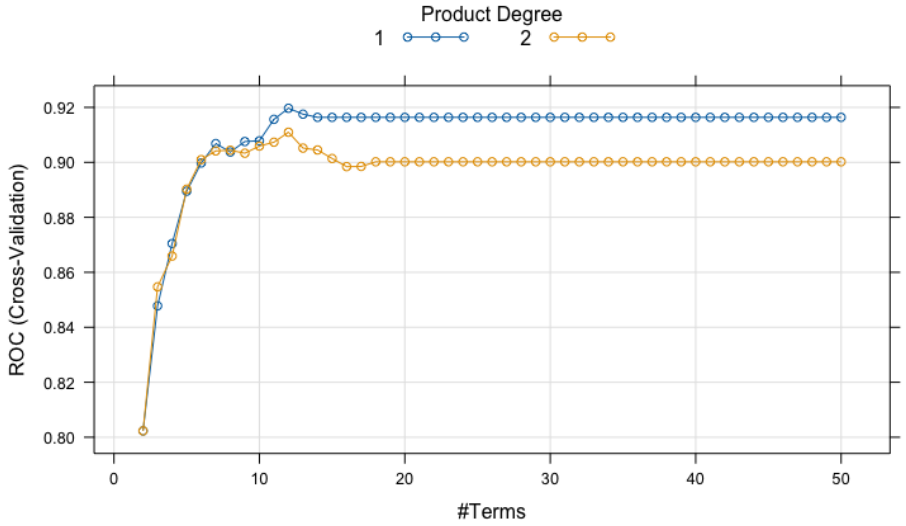
2) Regularized Discriminant Analysis (RDA)
The RDA had the best tuning parameters when the model had a gamma of 1, lambda of 0.



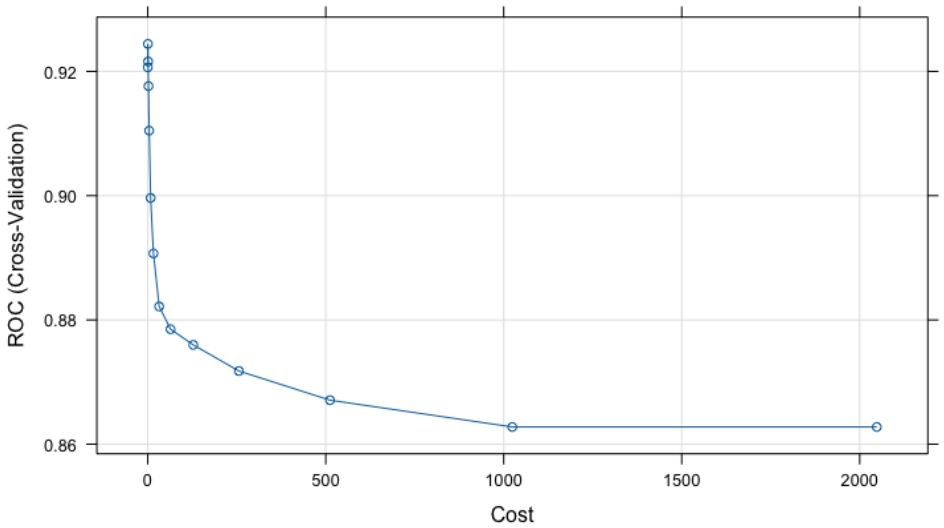
3) Neural Network
The neural network had the best tuning parameters when the model had a size of 1, decay of 0.1, and bag set equal to true. Area under the curve is equal to 0.915.



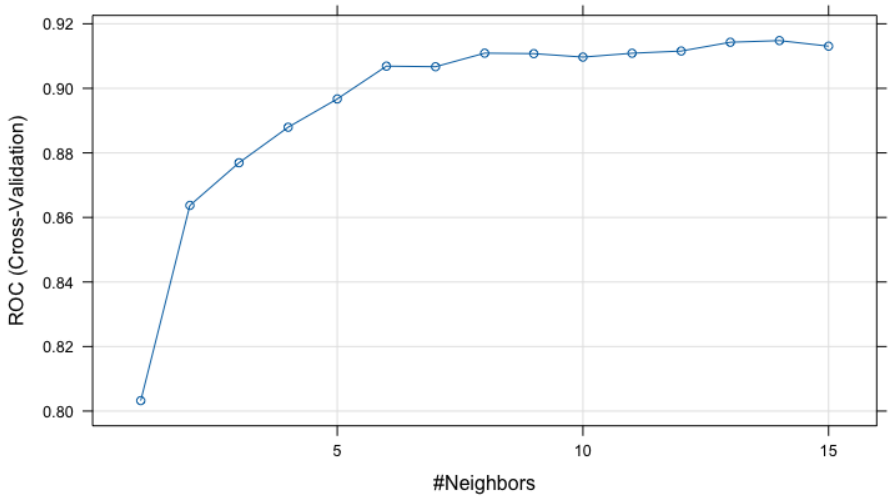
4) Flexible Discriminant Analysis
The best tuning parameters for flexible discriminant analysis were an nprune of 12 and a degree of 1. Area under the curve was equal to 0.9199.



5) Support Vector Machines (SVM)
Support vector machine had the results when tuning parameters were set to a sigma of 0.04603174 and c equal to 0.5. The area under the curve was equal to 0.924.



6) K Nearest Neighbors
The best tuning parameters found for K nearest neighbors was a k of 14. The area under the curve was equal to 0.915



Appendix 3: R Code

```
library('readr')

library('dplyr')

library('e1071')

library('corrplot')

library(caret)



#import data

heart_data<- read_csv('C:/Users/91630/Downloads/heart.csv')
```

```
summary(heart_data)


# Check for missing values in the entire dataset

missing_values <- sum(is.na(heart_data))

# Display the number of missing values

cat("Number of missing values in the dataset:", missing_values, "\n")

##### No missing values



# Check for duplicate rows in the entire dataset

duplicate_rows <- heart_data[duplicated(heart_data), ]

print(duplicate_rows)

###### no duplicates



# Separating Data ----------------------------------------------------

#goal variable

Response<- heart_data$HeartDisease

#categorical variables

categorical_Cols<-
heart_data[c("Sex","ChestPainType","RestingECG","ExerciseAngina","ST_Slope","FastingBS")
]

#### Numerical Cols

numCols<- heart_data[c("RestingBP","Cholesterol","MaxHR","Oldpeak","Age")]
```

```
####### Categorical columns

#create barplot for categorical columns

par(mfrow= c(2,4))

for (col in c(names(categorical_Cols))){

  categorical_Cols %>% pull(col) %>% table %>% barplot(main= col)

}


##### barplot for response variable as it is also categorical

barplot(table(Response), main="HeartDisease")




##### create  dummies

dummy <- dummyVars("~Sex+ChestPainType+RestingECG+ExerciseAngina+ST_Slope", data
=heart_data,fullRank = TRUE)

catDummies <- data.frame(predict(dummy, newdata = heart_data))

Cols<- c("Sex","ChestPainType","RestingECG","ExerciseAngina","ST_Slope", "HeartDisease")

heart <-cbind(catDummies,heart_data)

heart<-heart[, -which(names(heart) %in% Cols)]


head(heart)

dim(heart)


# Check for near-zero variance

library(caret)

nearzero_var <- nearZeroVar(heart, saveMetrics = TRUE)
```

```
# Display the results

print(nearzero_var)

#### No nearzero variance predictors




######## Numerical columns

#### checking for highly correlated variables

# Calculate correlation matrix

library(corrplot)

cor_matrix <- cor(heart)

highcorr<-findCorrelation(cor_matrix)

highcorr

#### no highly correlated ariables


# Create a correlation plot

#corrplot(cor_matrix, method = "circle", type = "full", title = "Correlation Plot of Heart Data", tl.col = "black", tl.srt = 45)

dev.new()

corrplot(cor_matrix, method = "circle", diag = TRUE, tl.cex = 0.8)


####(OR)


# Calculate correlation matrix

cor_matrix <- cor(heart)
```

```
dev.off()

# Create a heatmap

heatmap(cor_matrix,

        col = colorRampPalette(c("blue", "white", "red"))(100),

        main = "Correlation Heatmap",

        margins = c(10, 10))



# Numerical Data Plots ----------------------------------------------------

par(mar = c(1, 1, 1, 1), oma = c(1,1,1,1))

dev.off()

#create histogram and boxplot for each numerical column

par(mfrow= c(2,5))

for (col in c(names(numCols))){

  numCols %>% pull(col) %>% hist(main= col)

}

for (col in c(names(numCols))){

  numCols %>% pull(col) %>% boxplot(main= col)

}



#From above plots we can see some skewness in Cholesterol and Oldpeak columns.

#### So we want to check the accurate skewness
value----------------------------------------------------------

apply(numCols, 2, skewness, na.rm=TRUE)
```

As expected those 2 columns have skewness greater than 0.5 which means they are skewed and need transformation

So when trying to do boxcox transformation we are getting errors saying boxcox transformation can be done only to positive values i.e >0

so now we want to inspect those 2 columns carefully if they contain zeros or negative values

```
# Check if "Cholesterol" column contains zeros

zero_count <- sum(numCols$Cholesterol == 0)


# Check if "Cholesterol" column contains negative values

negative_count <- sum(numCols$Cholesterol < 0)


# Print the counts

cat("Count of zeros in Cholesterol column:", zero_count, "\n")

cat("Count of negative values in Cholesterol column:", negative_count, "\n")
```

from above output we can see Cholesterol contains 172 zeros in it

But it is impossible for a person to have zero cholesterol so i'll replace all zeroz with mean imputation

mean imputation on Cholesterol

```
library(caret)

# Calculate the mean excluding zeros

mean_chol <- mean(numCols$Cholesterol[numCols$Cholesterol != 0])
```

```
# Replace zeros with mean imputation
numCols$Cholesterol[numCols$Cholesterol == 0] <- mean_chol

# Check if "Cholesterol" column contains zeros
zero_count <- sum(numCols$Cholesterol == 0)
zero_count

##### Oldpeak
# Check if "Oldpeak" column contains zeros
zero_count <- sum(numCols$Oldpeak == 0)

# Check if "Oldpeak" column contains negative values
negative_count <- sum(numCols$Oldpeak < 0)


# Print the counts
cat("Count of zeros in Oldpeak column:", zero_count, "\n")
cat("Count of negative values in Oldpeak column:", negative_count, "\n")



##### from above output we can see that Oldpeak column contains both zeroz and negatives
#### It is possible that Oldpeak can be zero. So we thought of adding a constant to Oldpeak
### to choose a constant we checked the minimum value of Oldpeak
```

```
min(numCols$Oldpeak)


##### as the minimum value is 2.6 we are adding a constant of 2.7 to oldpeak as we want to remove zeros also

install.packages("moments")

library(moments)

numCols$Oldpeak <- numCols$Oldpeak+2.7




# Check if "Oldpeak" column contains zeros

zero_count <- sum(numCols$Oldpeak == 0)

zero_count


# Check if "Oldpeak" column contains negative values

negative_count <- sum(numCols$Oldpeak < 0)

negative_count



#### checking the plots and skewness after imputations

par(mar = c(1, 1, 1, 1), oma = c(1,1,1,1))

dev.off()

#create histogram and boxplot for each numerical column

par(mfrow= c(2,5))

for (col in c(names(numCols))){

  numCols %>% pull(col) %>% hist(main= col)

}
```

```
for (col in c(names(numCols))){
  numCols %>% pull(col) %>% boxplot(main= col)
}


#### skewness
apply(numCols, 2, skewness, na.rm=TRUE)


###### The skewness of Cholesterol is increased and still there are outliers in many columns
##### so now i want to do transformations
temp<- as.data.frame(numCols)
pre<- preProcess(temp, method = c("BoxCox", "center", "scale"))
numTrans<- predict(pre, temp)

print(pre)

#####Checking histograms for skewness before and after transformation
par(mfrow= c(2,5))
for (col in c(names(numCols))){
  numCols %>% pull(col) %>% hist(main= col)
}
for (col in c(names(numTrans))){
  numTrans %>% pull(col) %>% hist(main= col)
}
```

```
#### Histograms are appearing to be normally distributed now
##### So checking skewness
apply(numTrans, 2, skewness, na.rm=TRUE)
### yes now after center, scale and boxcox, the skewness of all columns got decreased



##### checking boxpolots for outliers before and after transformations
par(mfrow= c(2,5))
for (col in c(names(numCols))){
  numCols %>% pull(col) %>% boxplot(main= col)
}
for (col in c(names(numTrans))){
  numTrans %>% pull(col) %>% boxplot(main= col)
}



##### There are still outliers in our data
#### so we'll perform spatial sign
# Perform spatial sign transformation
spatSign <- spatialSign(numTrans)

# Convert the result to a data frame
spatSign <- as.data.frame(spatSign)
```

```
##### now i want to check the boxplots if they still consist of outliers even after performing spatial sign

par(mfrow= c(2,5))

for (col in c(names(numTrans))){

  numTrans %>% pull(col) %>% boxplot(main= col)

}

for (col in c(names(spatSign))){

  spatSign %>% pull(col) %>% boxplot(main= col)

}



###### now all the outliers are removed and we want to check if the skewness is increased

apply(numTrans, 2, skewness, na.rm=TRUE)


apply(spatSign, 2, skewness, na.rm=TRUE)

##### By observing the output, the skewness also got reduced.


######Now our data is all set for model selection



# Remove original numerical columns from heart_data

ContinuousCols<- c("RestingBP","Cholesterol","MaxHR","Oldpeak","Age")

heart<-heart[, -which(names(heart) %in% ContinuousCols)]
```

```
dim(heart)

# Append spatSign(Numerical variables df) dataframe to heart_data
heart <- cbind(heart, spatSign)
dim(heart)

# For reproducibility
set.seed(123)

splitIndex <- createDataPartition(heart_data$HeartDisease, p = 0.8, list = FALSE, times = 1)
# Create training and testing datasets based on the split
train_data <- heart[splitIndex,]
test_data <- heart[-splitIndex,]


# Include the "HeartDisease" response variable in the training and testing datasets
train_response <- heart_data$HeartDisease[splitIndex]
test_response <- heart_data$HeartDisease[-splitIndex]


# Define a resampling method
#ctrl <- trainControl(method = "cv", number = 5)


# Check the data type and levels of the response variable
```

```
str(train_response)
str(test_response)

# Convert response variable to a factor with two levels
train_response <- as.factor(train_response)
test_response <- as.factor(test_response)

# Check the levels of your factor variable
levels(train_response)


# Change levels from "0" to "No" and from "1" to "Yes"
levels(train_response) <- c("No", "Yes")
levels(test_response) <- c("No", "Yes")

# Verify that the levels have been changed
levels(train_response)
levels(test_response)


################### Models building
####### Logistic Regression

ctrl <- trainControl(method = "cv", number= 10,
                     summaryFunction = twoClassSummary,
```

```
                    classProbs = TRUE,
                    savePredictions = TRUE)
set.seed(123)
lrFull <- train(x= train_data,
                y = train_response,
                method = "glm",
                preProc = c("center", "scale"),
                family = "binomial",
                metric = "ROC" ,
                trControl = ctrl)
lrFull
plot(lrFull)
summary(lrFull)



lrPred <- predict(lrFull,newdata = test_data)
confusionMatrix(lrPred,test_response)



library(pROC)
FullRoc <- roc(lrFull$pred$obs,lrFull$pred$Yes)
plot(FullRoc, legacy.axes = TRUE, col = "blue", main = "ROC Curve")
auc(FullRoc)
```

```
####### LDA
## Using train function, should add pre-processing
## SET SEED

ctrl <- trainControl(method = "cv", number = 10,
                     summaryFunction = twoClassSummary,
                     classProbs = TRUE,
                     ##index = list(simulatedTest[,1:4]),
                     savePredictions = TRUE)

set.seed(123)
LDAFull <- train(x = train_data,
                 y = train_response,
                 method = "lda",
                 preProc = c("center", "scale"),
                 metric = "ROC",
                 trControl = ctrl)
LDAFull
summary(LDAFull)

ldaPred <- predict(LDAFull,newdata = test_data)
confusionMatrix(ldaPred,test_response)
```

```
library(pROC)

FullRoc <- roc(LDAFull$pred$obs,LDAFull$pred$Yes)

plot(FullRoc, legacy.axes = TRUE, col = "blue", main = "ROC Curve")

auc(FullRoc)
```

```
#####SVM

# Set up the training control with ROC as the summary function

ctrl <- trainControl(method = "cv", number= 10, summaryFunction = twoClassSummary,
classProbs = TRUE)


set.seed(123)

svm_model <- train(x = train_data,

                   y = train_response,

                   method = "svmRadial",

                   metric = "ROC",

                   preProc = c("center", "scale"),

                   tuneLength = 14,

                   trControl = ctrl)


svm_model

plot(svm_model)

ggplot(svm_model)+coord_trans(x='log2')
```

```
svmRpred <- predict(svm_model, newdata = test_data)
confusionMatrix(svmRpred,test_response)

svmRaccuracy <- data.frame(obs = test_response , pred = svmRpred)
defaultSummary(svmRaccuracy)



# Make predictions on the test data
svmRpred <- predict(svm_model, newdata = test_data)

# Evaluate the model using confusion matrix and other metrics
confusionMatrix(svmRpred, test_response)

# Create ROC curve
svm_probs <- predict(svm_model, newdata = test_data, type = "prob")[, "Yes"]
FullRoc <- roc(test_response, svm_probs)

# Plot the ROC curve
plot(FullRoc, legacy.axes = TRUE, col = "blue", main = "ROC Curve")

# Print AUC
auc_value <- auc(FullRoc)
cat("AUC:", auc_value, "\n")
```

######KNN

```
ctrl<- trainControl(method = "cv", number = 10, classProbs = TRUE, summaryFunction = twoClassSummary)

set.seed(123)

knnTune <- train(x = train_data,

                 y = train_response,

                 method = "knn",

                 metric = "Kappa",

                 # Center and scaling will occur for new predictions too

                 preProc = c("center", "scale"),

                 tuneGrid = data.frame(.k = 1:15),

                 trControl = ctrl)

knnTune

plot(knnTune)



knnpred <- predict(knnTune, newdata = test_data)

confusionMatrix(knnpred,test_response)

knnaccuracy <- data.frame(obs = test_response , pred = knnpred)

defaultSummary(knnaccuracy)

library(pROC)

FullRoc <- roc(knnTune$pred$obs,knnTune$pred$Yes)
```

```
plot(FullRoc, legacy.axes = TRUE, col = "blue", main = "ROC Curve")

auc(FullRoc)




###### Neural Networks

nnetGrid <- expand.grid(.decay = c(0, 0.01, .1),

                        .size = c(1:10),

                        ## The next option is to use bagging (see the

                        ## next chapter) instead of different random

                        ## seeds.

                        .bag = T)


ctrl <- trainControl(method = "cv", number = 10, classProbs = T, summaryFunction =
twoClassSummary)


set.seed(123)

nnetTune <- train(train_data, train_response,

                  method = "avNNet",

                  metric = "Kappa",

                  tuneGrid = nnetGrid,

                  trControl = ctrl,

                  ## Automatically standardize data prior to modeling

                  ## and prediction

                  preProc = c("center", "scale"),

                  linout = TRUE,
```

```
                trace = FALSE,
                MaxNWts = 10 * (ncol(train_data) + 1) + 10 + 1,
                maxit = 500)



nnetTune
plot(nnetTune)



######PLSDA

ctrl <- trainControl(method= "cv", number= 10, summaryFunction = twoClassSummary,
                     classProbs = TRUE)

## caret contains a built-in function called twoClassSummary that calculates the
## area under the ROC curve, the sensitivity, and the specificity.
set.seed(123)
plsFit2 <- train(x = train_data,
                 y = train_response,
                 method = "pls",
                 tuneGrid = expand.grid(.ncomp = 1:10),
                 preProc = c("center","scale"),
                 metric = "Kappa",
                 trControl = ctrl)
```

```
plsFit2
plot(plsFit2)




#####Penalized
ctrl <- trainControl(method = "cv", number= 10,
                     summaryFunction = twoClassSummary,
                     classProbs = TRUE,
                     ##index = list(simulatedTest[,1:4]),
                     savePredictions = TRUE)

glmnGrid <- expand.grid(.alpha = c(0, .1, .2, .4, .6, .8, 1),
                        .lambda = seq(.01, .2, length = 10))
set.seed(123)
glmnTuned <- train(x=train_data,
                   y = train_response,
                   method = "glmnet",
                   tuneGrid = glmnGrid,
                   preProc = c("center", "scale"),
                   metric = "Kappa",
                   trControl = ctrl)
glmnTuned
plot(glmnTuned)
```

```
glmnpred <- predict(glmnTuned, newdata = test_data)

confusionMatrix(glmnpred,test_response)



############ Multivariate Adaptive Regression Splines

# Fix the seed so that the results can be reproduced


## marsTuned <- train(solTrainXtrans, solTrainY,

# Explicitly declare the candidate models to test

ctrl = trainControl(method = "cv", number= 10, classProbs = T, summaryFunction =
twoClassSummary)


marsGrid <- expand.grid(.degree = 1:2, .nprune = 2:50)  ## Change 38 to 50


set.seed(123)

marsTuned <- train(x=train_data,

                   y = train_response,

                   method = "earth",

                   metric = "Kappa",

                   preProc = c("center", "scale"),

                   # Explicitly declare the candidate models to test

                   tuneGrid = marsGrid,

                   trControl = ctrl)


marsTuned
```

```
plot(marsTuned)



########### Nearest Shrunken Centroids
ctrl <- trainControl(method= "cv", number= 10, summaryFunction = twoClassSummary,
                     classProbs = TRUE)

## nscGrid <- data.frame(.threshold = 0:4)
nscGrid <- data.frame(.threshold = seq(0,4, by=0.1))

set.seed(123)
nscTuned <- train(x=train_data,
                  y = train_response,
                  method = "pam",
                  preProc = c("center", "scale"),
                  tuneGrid = nscGrid,
                  metric = "Kappa",
                  trControl = ctrl)

nscTuned
plot(nscTuned)



####### Nonlinear Discriminant Analysis
```

```
library(caret)
ctrl <- trainControl(method= "cv", number= 10, summaryFunction = twoClassSummary,
                     classProbs = TRUE)

set.seed(123)
mdaFit <- train(x=train_data,
                y = train_response,
                method = "mda",
                metric = "Kappa",
                preProc = c("center", "scale"),
                tuneGrid = expand.grid(.subclasses = 1:3),
                trControl = ctrl)
mdaFit
plot(mdaFit)



########## Flexible Discriminant Analysis
marsGrid <- expand.grid(.degree = 1:2, .nprune = 2:38)

ctrl<- trainControl(method = "cv", number = 10, summaryFunction =twoClassSummary,
classProbs = T )

set.seed(123)
fdaTuned <- train(x=train_data,
                  y = train_response,
                  method = "fda",
```

```
                metric = "Kappa",
                preProc = c("center", "scale"),
                # Explicitly declare the candidate models to test
                tuneGrid = marsGrid,
                trControl = ctrl)

fdaTuned
plot(fdaTuned)
plot(fdaTuned,main="FDA, degree = 1 and nprune = 6")
fdaPred <- predict(fdaTuned, newdata = simulatedTest[,1:4])
confusionMatrix(data = fdaPred,reference =simulatedTest[,6])




########## Naive Bayes
install.packages("klaR")
library(klaR)

ctrl<- trainControl(method = "cv", number = 10, summaryFunction = twoClassSummary,
classProbs = T)

set.seed(123)
nbFit <- train( x=train_data,
                y = train_response,
```

```
            method = "nb",
            metric = "Kappa",
            preProc = c("center", "scale"),
            ##tuneGrid = data.frame(.k = c(4*(0:5)+1, 20*(1:5)+1, 50*(2:9)+1)), ## 21 is the best
            tuneGrid = data.frame(.fL = 2,.usekernel = TRUE,.adjust = TRUE),
            trControl = ctrl)

nbFit
plot(nbFit)



############ RDA
rdaGrid <- expand.grid(.gamma= 1:10, .lambda = c(0, .1, 1, 2))

set.seed(123)
rdaFit <- train(x=train_data,
            y = train_response,
            method = "rda",
            metric = "ROC",
            tuneGrid = rdaGrid,
            trControl = ctrl)
rdaFit
plot(rdaFit)
rda_Pred<- predict(rdaFit, newdata=test_data)
confusionMatrix(data=rda_Pred, reference =test_response)
```

```
########## Calculating AUC for best models
pred <- predict(glmnTuned, newdata= test_data, type="raw")
roc(test_response, as.numeric(pred))

pred <- predict(svm_model, newdata= test_data, type="raw")
roc(test_response, as.numeric(pred))

######## Variable importance of best models
imp_predictors <- varImp(glmnTuned)
imp_predictors
plot(imp_predictors, top = 5)

imp_predictors <- varImp(svm_model)
imp_predictors
plot(imp_predictors)
```