Cardiovascular Disease;

A Review of Machine Learning Models for Predicting Cardiovascular Diseases

Amr Ibrahim & Vlad Yashaev

SUNY Old Westbury

Fall 2022

Abstract

Cardiovascular diseases (CVD) are the leading cause of death globally with around 17.9 million lives per year. Heart attacks and strokes account for more than 4/5 CVD deaths, and a third of those deaths occur in people under 70 years old [2]. Identifying people with CVD and those who are at high cardiovascular risk can prevent premature deaths. This project is motivated by the desire to improve outcomes for patients at risk for CVD by using machine learning models to refine early detection of potential any heart disease in patients. The first step to achieve such a goal is to review if machine learning models can even be used for the purposes of predicting potential CVD. For that, we review various available open-source machine learning algorithms, including, Logistic Regression, Support Vector Machine (SVM), Random Forest, k-Nearest Neighbor and Decision Tree. In addition, "Heart Failure Prediction Dataset," from Kaggle will be used to train the models on predetermined cases of CVD [1]. Once trained, the models will be run, and their predictions will be evaluated for accuracy, precision, recall, and F-scores as appropriate. The goal is to achieve at least 80% accuracy with the prediction models. A model with high predictive accuracy can help with early detection of CVD and as a result, may be determined as a useful tool in helping to lower rates of premature deaths due to heart disease.

Introduction

The hypothesis is that we could build a supervised predictive model that predicts heart disease events with 70-80% accuracy. To go about building the models, we utilize various classification tools including K-Nearest Neighbors (KNN), Decision Tree, Random Forest, Logistic Regression, and Support Vector Machines (SVM). A determination of how to split the data is needed. Based on that, the data is split into either training/validation/test partitions or training/validation partitions. If the former, the split would be 50%/30%/20%, or 60%/40% if the latter. The splitting of the data is done by using *sklearn.model_selection*'s *train_test_split* method. To evaluate the models, we extract the performance metrics for each model using *classification_report* from *sklearn.metrics*. The performance metrics print out each model's accuracy, recall, precision, and F1-score which we use to evaluate and compare models.

One of the models that is tested for accuracy and precision is the k-nearest neighbors (kNN) model. This supervised learning method uses a user-defined k value to compare k number of values closest to the observation for which the model is meant to make a prediction for. If the majority of k-number of values lean towards a particular output, the model returns that output as the prediction for the new observation.

The decision tree classification model uses a set of questions about the different features of a dataset to predict a classification value of a new observation. Another classification method that is used as a prediction model is the random forest method. Like the decision tree model, this method also uses a yes/no set of questions to derive at a predicted classification value. However, the main difference is that instead of using one tree with a single root node, the random forest method compiles several decision trees with random cases from the training data. When predicting a classification for a test subject, the random forest model output prediction is equivalent to the result found by the majority of all the decision trees.

In a logistic regression, we forecast the likelihood of an outcome rather than the actual continuous value. This produces a value in the range between 0 and 1. Then, based on the use case, we choose a line. Any data point with a probability value over the line is put into class 1 and data below the line are categorized into class 0 [4].

SVM outputs an optimal line of separation among classes. A "hyperplane" is the term used to describe this line of separation in a multidimensional world. To create this separating hyperplane, SVM takes into account outliers that are quite near to a differing class. Any new point that must be predicted after this hyperplane has been built into the model is checked to determine which side it is on [4].

Data Description

The dataset, “Heart Failure Prediction Dataset,” from Kaggle, is used to train and test the predictive abilities of various machine learning models [1]. This dataset is formatted as a comma-separated values (CSV) file. The dataset includes 12 features and has 918 observations (i.e., 918 rows by 12 columns). As this is a raw dataset, a thorough review of the data is performed. This includes data cleansing and preparation tasks such as ensuring all data is numerical, dealing with outliers, and dimensionality reduction. The data preparation steps prior to setting up predictive models ensure greater accuracy and precision of the models.

The first of the data preparation steps taken is changing all non-numerical features to numerical features. This is accomplished using Panda’s *get_dummies* method. The *get_dummies* method divides string values in a feature into separate columns. As an example, the *RestingECG* column starts out with 3 unique string values. Using the *get_dummies* method on this feature results in each unique value having its own column (i.e., 3 new features are added to the dataset). Each observation that meets the condition of a particular dummy feature has a value of 1 to represent True, otherwise 0 to represent False. As with the *RestingECG* feature example, similar dummy features are created for other categorical features.

Once all the features are changed to numerical values, each feature is inspected for outliers. For ease of visualizing outliers, each feature is first plotted as a box plot. For each feature with outliers, a determination is made on how to handle the outliers.

Lastly, based on correlation data, a decision is made on dimensionality reduction. For features that have correlations greater than 0.7, only one of the pair features is required to remain in the dataset. Furthermore, factor analysis and the use of a scree plot help determine if any features get combined, thus reducing the number of features with minimal impact on the predictive models.

Exploratory Data Analysis: Visualization

For exploratory data analysis, columns were separated into two categories: numerical and categorical. Numerical columns were graphed using Seaborn’s *histplot*, while categorical columns were graphed using Seaborn’s *countplot* with the *HeartDisease* feature as the hue.

Numerical Data

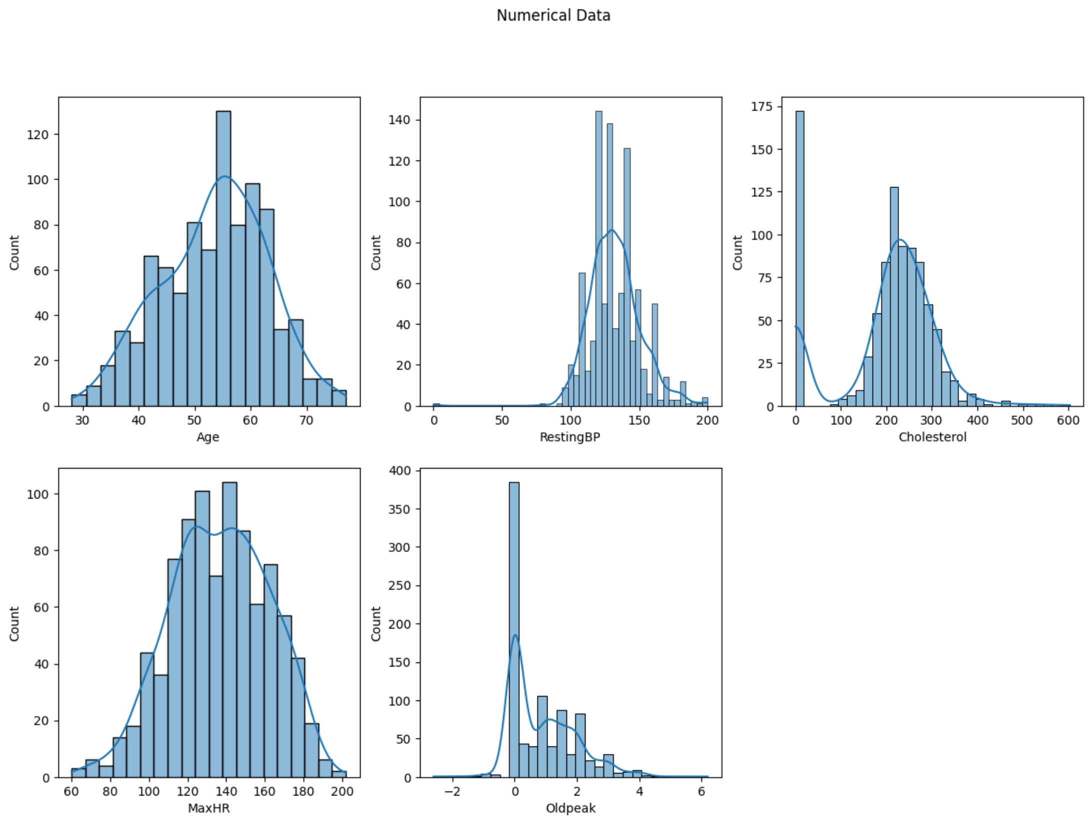


Fig 1. Numerical data visualization using *seaborn.histplot*

Approximately all numerical data have normal distributions except for *Oldpeak*. *Oldpeak* has a positively skewed distribution. *RestingBP* and *Cholesterol* have outliers around 0. Descriptive statistics for each numerical variable are as follows:

	Age	***RestingBP***	***Cholesterol***	***MaxHR***	***Oldpeak***
mean	53.51	132.4	198.8	136.81	0.89
std	9.43	18.51	109.38	25.46	1.07
25%	47.0	120.0	173.25	120.0	0.0
75%	60.0	140.0	267.0	156.0	1.5

Table 1. Numerical Features Descriptive Statistics

Categorical Data

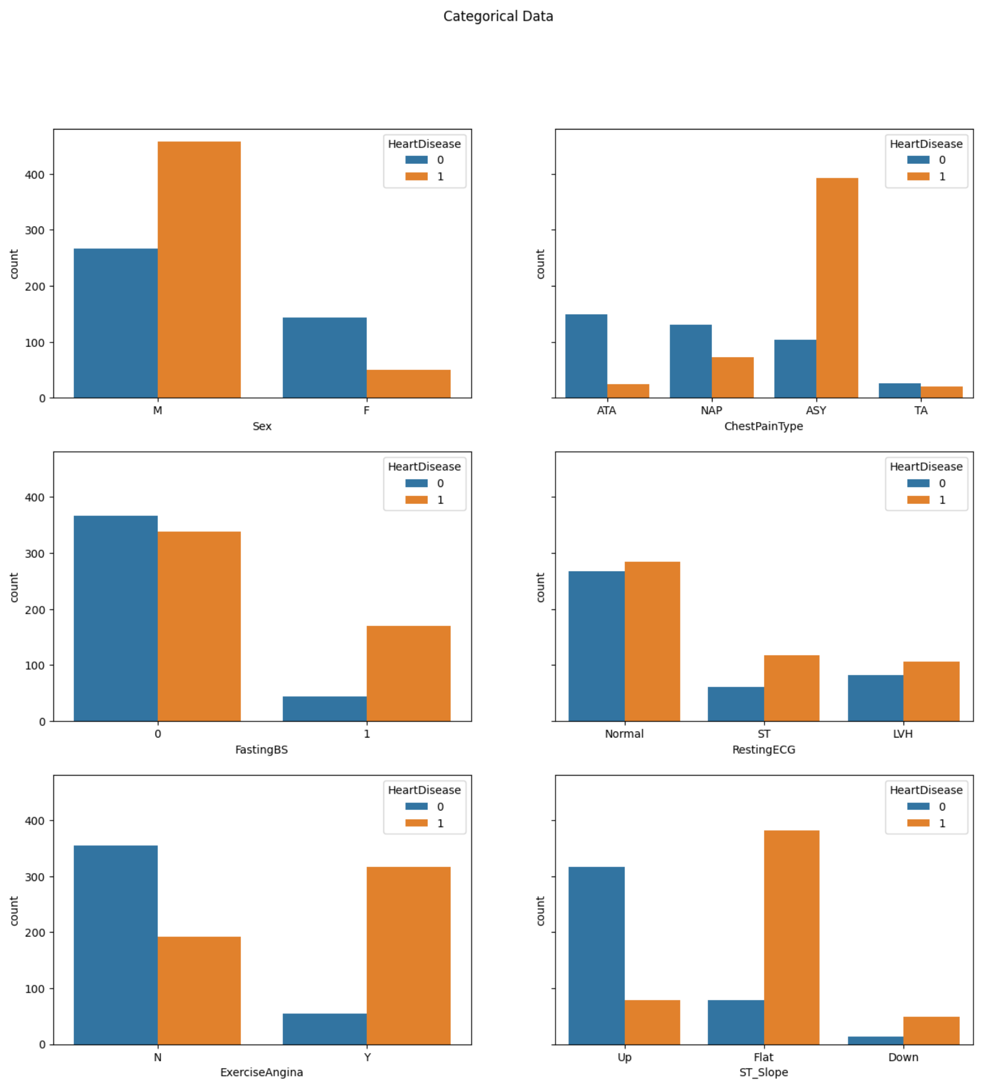


Fig 2. Categorical data visualization using *seaborn.countplot* with the *HeartDisease* feature as the hue. Blue boxes represent subjects with no heart disease and orange boxes represent subjects with heart disease

HeartDisease is the target data where there is no significant difference between subjects with heart disease and subjects without heart disease with ratio of 0.81, no heart disease/heart disease counts. Therefore, this dataset can be treated as balanced. Each categorical variable has a category with highest occurrence of heart disease as follows:

- Sex: Male
- ChestPainType: ASY (Atypical Angina)
- FastingBS: 1 (Fasting Blood Sugar > 120 mg/dl)
- RestingECG: ST

- ExerciseAngina: Y (subjects with exercise-induced angina)
- ST_Slope: Flat

Pair plot & Correlation

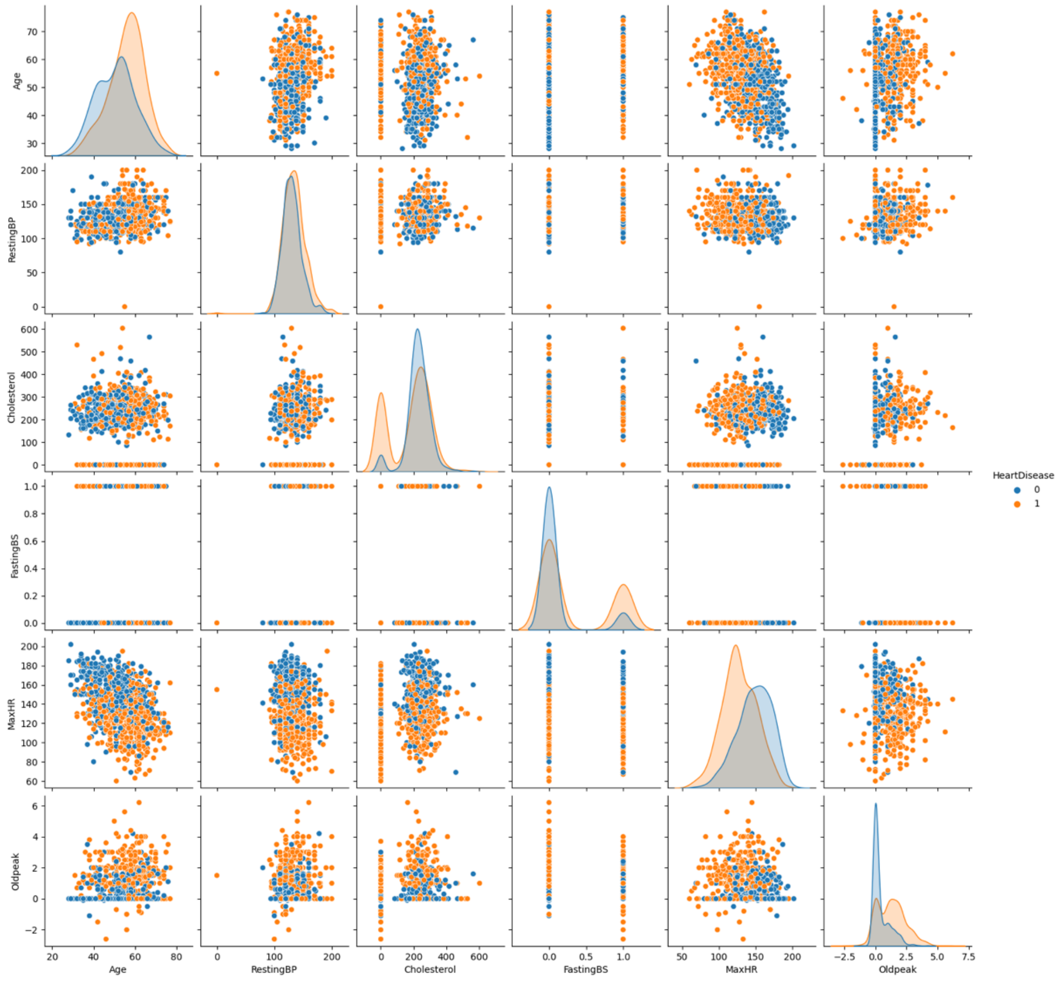


Fig 3. Pairplot with the *HeartDisease* feature as the hue. Orange represents subjects with heart disease and blue represents subject without heart disease

As shown in Fig. 3, heart disease seems to occur more frequently at higher ages compared to lower ages.

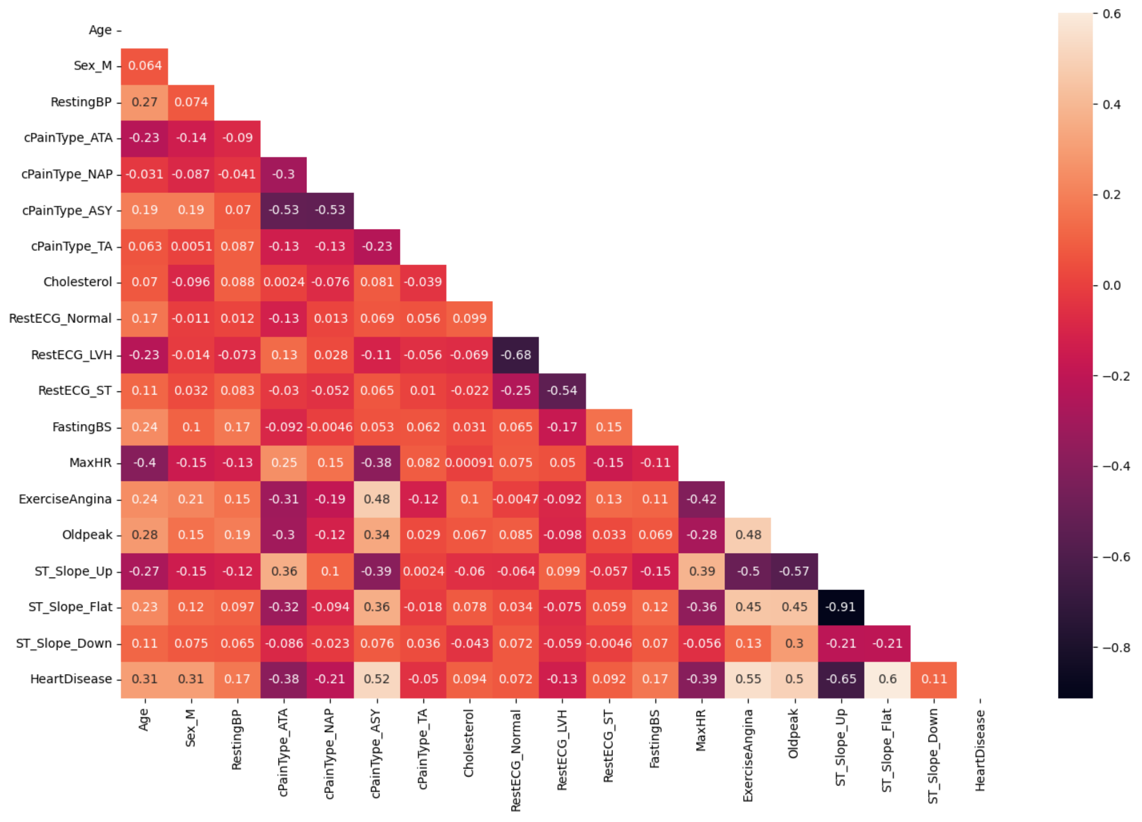


Fig 4. Correlation plot

Approximately all correlations are under 0.8 except for *ST_Slope_UP* and *ST_Slope_Flat* with 0.91 correlation. From Fig.4, heart disease is estimated to be most influenced by ASY chest pain type, exercise angina, old peak, and flat ST slope.

Data Cleaning and Preprocessing

The data cleansing portion of the project includes reviewing datatypes, null values, and converting categorical features to numerical features, as well as finding and handling outliers. The *.info()* method was used for an overview of all the features. While this method also shows null values, it was primarily used to review whether any of the features' datatypes would need to be adjusted. No features needed datatype adjustments. Similarly, using the *.isna()* method revealed there are no missing values in the dataset. Prior to handling outliers, conversion of categorical features to numerical features is required.

Non-numerical features were converted to numerical ones using Panda's *get_dummies* method. This method is used to divide unique string values of a feature into separate columns. Features such as the *RestingECG* column have 3 unique values. The *get_dummies* method converts this feature into 3 new dummy features, one for each unique value in the feature. If an observation/row of the dummy feature meets the feature's conditions, a value of 1 is assigned to that row. This essentially converts categorical features to numerical features. Next, the outliers are reviewed and handled.

An *Outliers()* class was built to find and handle all features with outlier observations. This class has a data frame as a required parameter, and a particular feature's name as an optional parameter. The class has several methods defined, one of which is, *.OutlierInfoDF()*. This method is designed to return a dictionary showing the passed-in data frame's features with outliers. The returned value is a dictionary composed of features with outliers, showing each qualifying feature's Mean, Q1, Q3, and IQR values. There are two additional methods that return boxplots. One is to show the boxplot of a particular feature (assuming a feature name was passed to create an object instance for that feature). The other method is, *.dfOutliersBoxplot()*. This method loops through the dataset and returns a boxplot for each feature with outlier observations. Using such visualizations as the boxplots as well as Seaborn's *.pairplot()* method provides a visual representation of potential features with outliers. Once outliers have been identified, a decision was initially made to remove them from the dataset. The removal was done using yet another method in the *Outliers()* class; *.removeNamedOutliers()*. This outlier removal method returns a data frame with the outliers filtered out of the specified feature. Using the class and its methods, 3 features with outliers were identified. The feature, *RestingBP*, had 28 outliers; *Cholesterol* had 177 outliers; *OldPeak* had 12 outliers. Some of the outliers had outlier observations across more than one of these features. After removing all the outliers, the dataset remains with 701 observations to work with. The predictive models were run, first, on the data without outliers. Then, a copy of the same dataset was scaled to reduce the effects of outliers, but no outliers were actually removed. Running the models on the scaled data with the outliers in place resulted in higher accuracy metrics. As a result, we kept the outliers in the dataset and scaled the dataset.

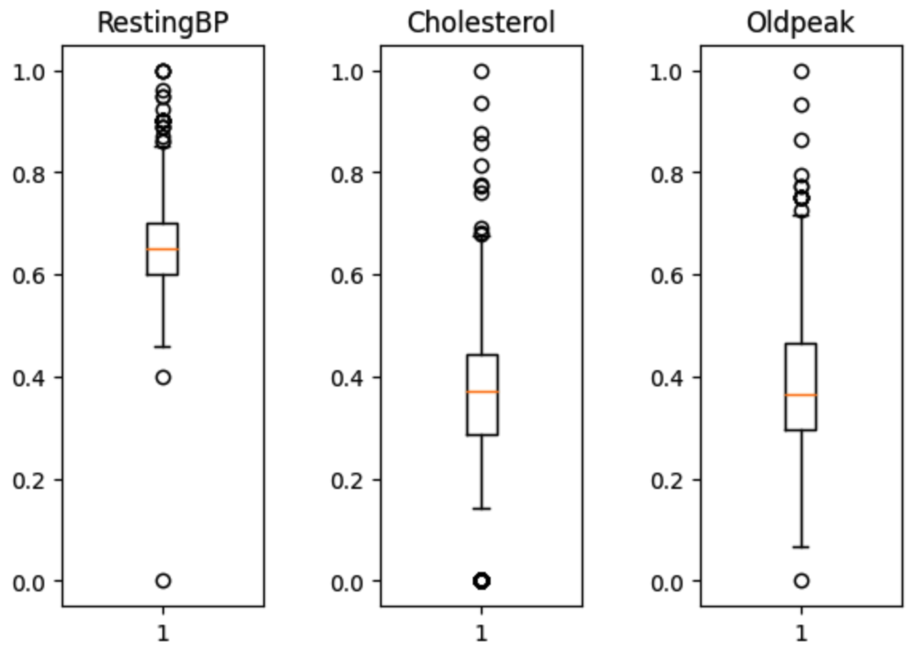


Fig 5. Boxplots of features with outliers

The major data preprocessing step taken for this project is dimensionality reduction. To achieve a desired reduction, factor analysis was performed using a scree plot and evaluating factor eigenvalues and how they relate to the dataset's total variability.

To begin with, the target feature of the dataset (*HeartDisease*) was separated into a data frame of its own. This was done so that *FactorAnalyzer()* could be used to model a factor analysis on the remaining data frame's features. At this point, no number of factors has been specified as more information was needed to determine how many factors are required for minimal information loss. To find this information, a scree plot was used to plot *n* number of factors equivalent to the current number of features in the dataset (18 features without *HeartDisease*). The relative eigenvalue of each factor is returned in the form of a plot (figure 6).

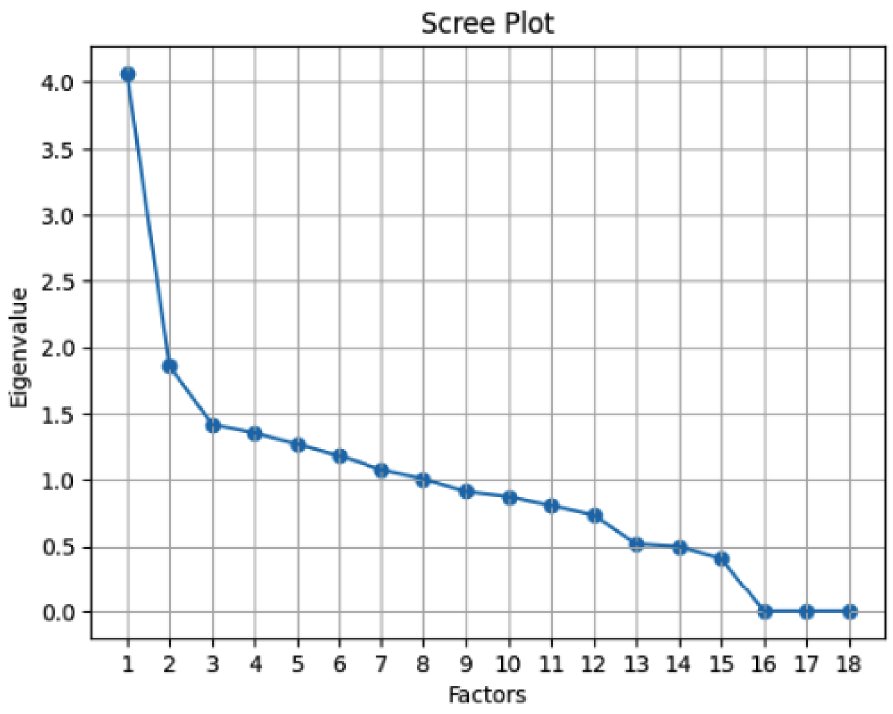


Fig 6. Scree Plot

Note in Fig 6, there are 8 factors with an eigenvalue of 1 or above. Such factors, that have an eigenvalue equal to or greater than 1, are the ideal number of factors to keep post dimensionality reduction. However, each factor's eigenvalue divided by the total number of factors (18 in this case), shows the proportion of the total variability of the dataset accounted for by that factor. For example, factor 1 has an eigenvalue of 4.067. Dividing 4.067 by 18 gives a proportion of 0.2259. This proportion means that 22.59% of the total variability of the dataset is represented by factor 1. Calculating the variability of the recommended first 8 factors with the highest eigenvalues accounts for only 73.55% of the dataset's total variability. As a result, an initial decision was made to include all factors with eigenvalues greater than 0.5. Setting this threshold returns a suggested number of factors of 14, which accounts for 97.74% of the dataset's total variability. However, when we later ran the predictive models, we decided that, based on the resultant accuracies and feature importance values, it is best to keep all features other than those with a high correlation (greater than 0.7). The correlation between *ST_Slope_Flat* and *ST_Slope_Up* is 0.91. *ST_Sloe_Flat* was removed as it came up with a smaller importance level than *ST_Slope_Up* in initial model runs (figure 7).

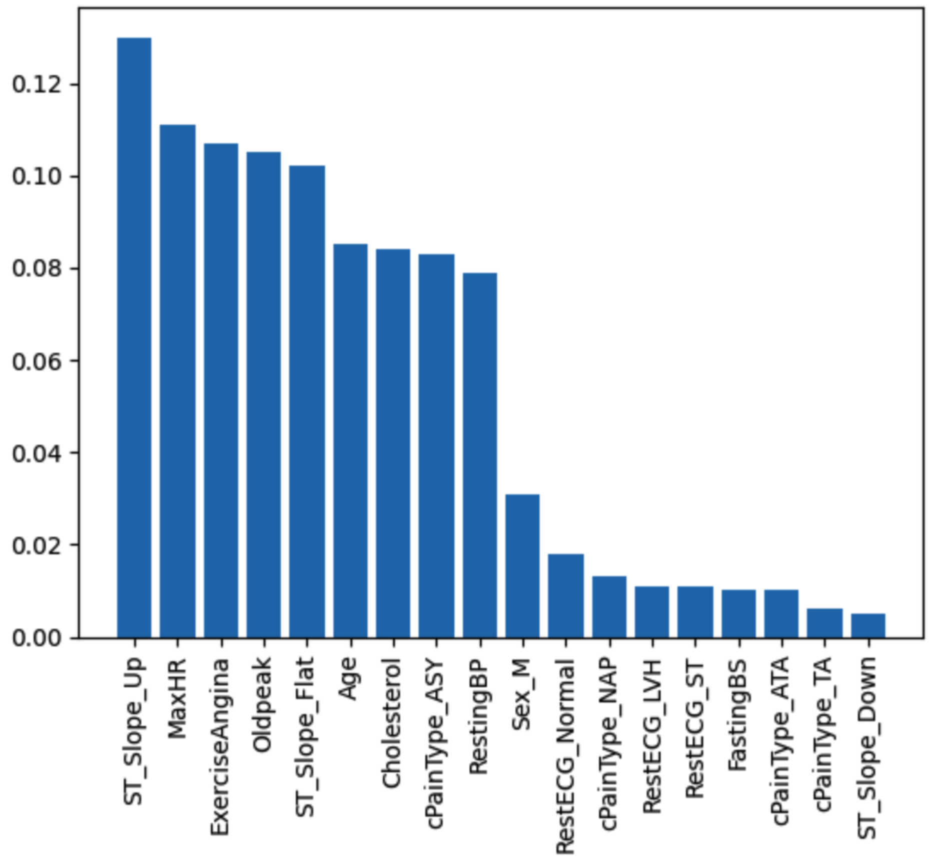
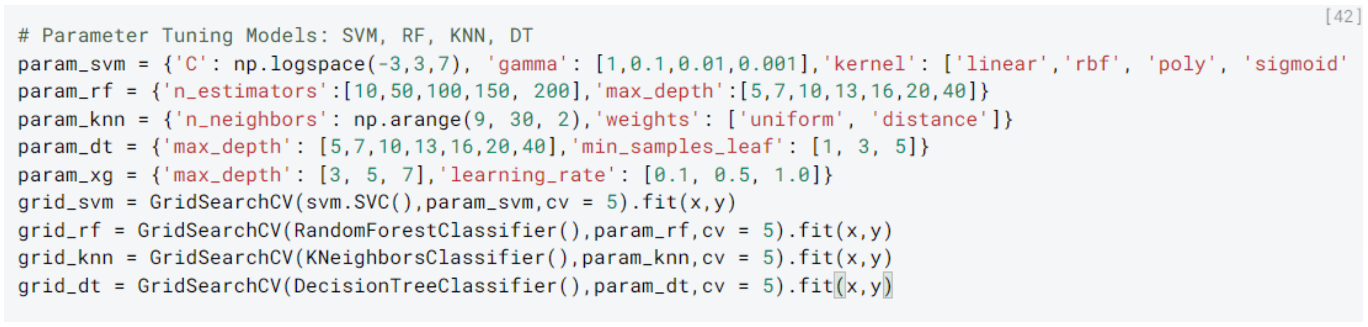


Fig 7. Random Forest feature importance

Data Modeling

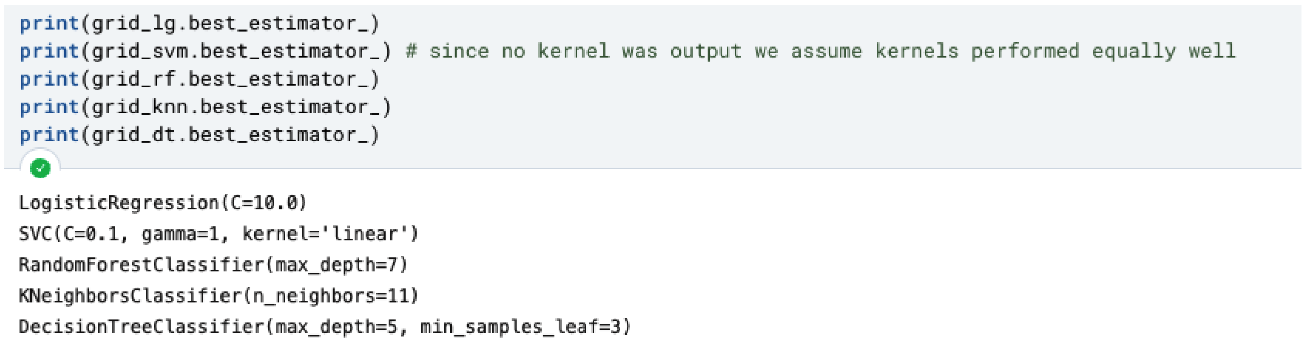
The predictive machine learning models used include Logistic Regression, Support Vector Machine, Random Forest, K Nearest Neighbor, and Decision Tree. Before running any model, the cleaned and preprocessed dataset is split into training and testing data. The variable x is assigned all the features other than the target feature. The target feature, assigned to variable y, is “*HeartDisease*.” This initial split is used to fit *GridSearchCV*, which is used to help determine how it may be best to tune the hyperparameters of each model.

Figure 8 shows the parameters entered when running the *GridSearchCV* method. Figure 9 displays the suggested hyperparameters. After testing some of these hyperparameter suggestions, we determined to follow some of the suggestions but not all. For the random forest model, as an example, *GridSeachCV* recommended we only define a *max_depth*, with a value of 7. However, the best results came when we set the *max_depth* parameter to 17, and *n_estimators* to 200.



```
# Parameter Tuning Models: SVM, RF, KNN, DT
param_svm = {'C': np.logspace(-3,3,7), 'gamma': [1,0.1,0.01,0.001],'kernel': ['linear','rbf', 'poly', 'sigmoid'
param_rf = {'n_estimators':[10,50,100,150, 200],'max_depth':[5,7,10,13,16,20,40]}
param_knn = {'n_neighbors': np.arange(9, 30, 2),'weights': ['uniform', 'distance']}
param_dt = {'max_depth': [5,7,10,13,16,20,40],'min_samples_leaf': [1, 3, 5]}
param_xg = {'max_depth': [3, 5, 7],'learning_rate': [0.1, 0.5, 1.0]}
grid_svm = GridSearchCV(svm.SVC(),param_svm,cv = 5).fit(x,y)
grid_rf = GridSearchCV(RandomForestClassifier(),param_rf,cv = 5).fit(x,y)
grid_knn = GridSearchCV(KNeighborsClassifier(),param_knn,cv = 5).fit(x,y)
grid_dt = GridSearchCV(DecisionTreeClassifier(),param_dt,cv = 5).fit(x,y)
```

Fig 8. *GridSearchCV* code

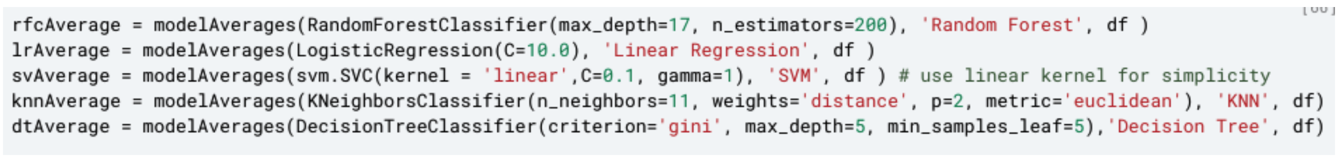


```
print(grid_lg.best_estimator_)
print(grid_svm.best_estimator_) # since no kernel was output we assume kernels performed equally well
print(grid_rf.best_estimator_)
print(grid_knn.best_estimator_)
print(grid_dt.best_estimator_)
```

```
LogisticRegression(C=10.0)
SVC(C=0.1, gamma=1, kernel='linear')
RandomForestClassifier(max_depth=7)
KNeighborsClassifier(n_neighbors=11)
DecisionTreeClassifier(max_depth=5, min_samples_leaf=3)
```

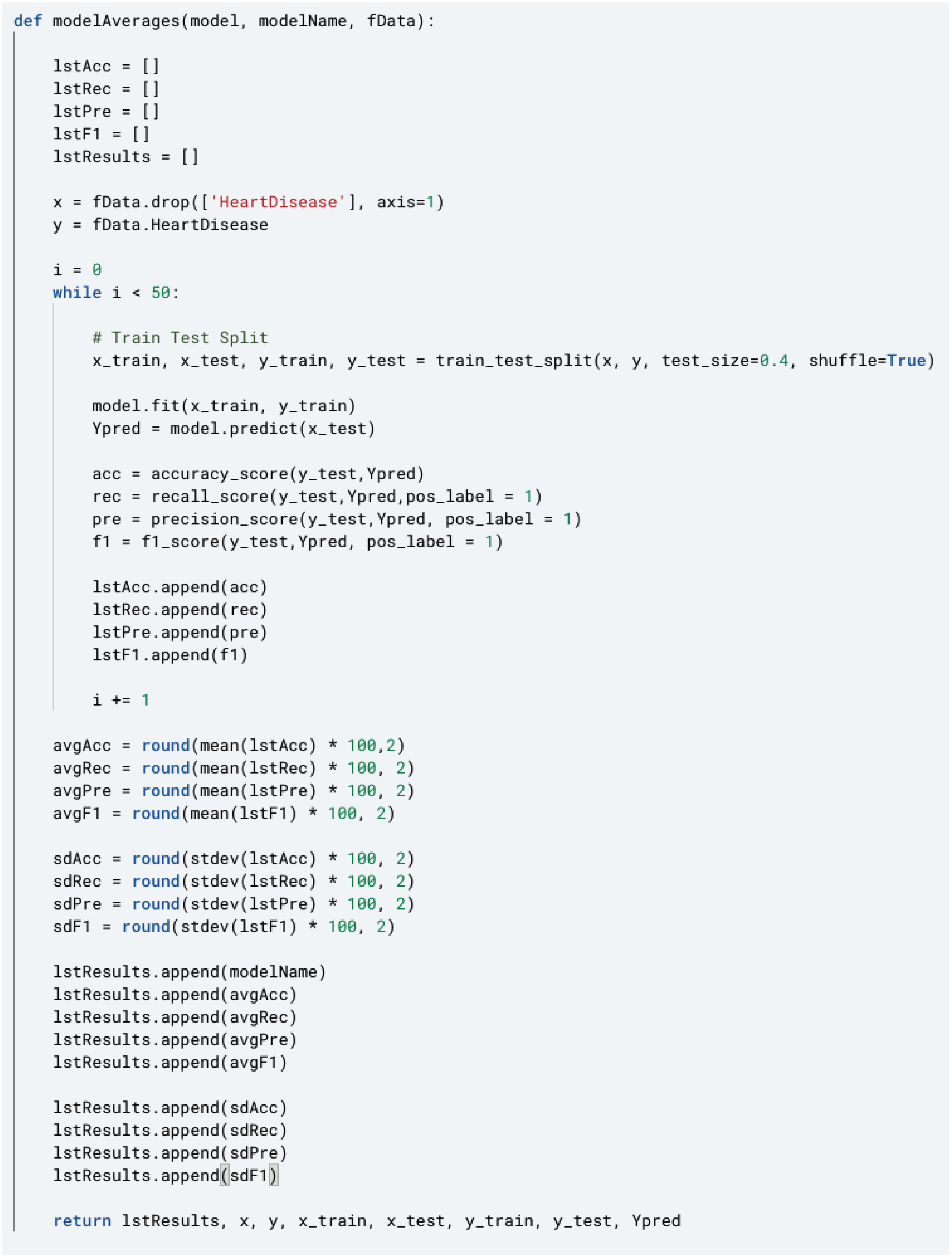
Fig 9. *GridSearchCV* output

Due to the technological limitations of the coding notebook used, Deepnote, the entire notebook had to be rerun often. This caused a large range of possible resultant predictive accuracy levels from run to run. As a result, it was determined to run the models through a function that would split, fit and run the training/testing data 50 times for each model it is called on. The function would then collect the performance results of each run and average them out, returning the mean and standard deviation results of accuracy, recall, precision, and F1 score. Figure 10 shows the function call for each model and the hyperparameter settings used. Figure 11 is of the function itself.



```
rfcAverage = modelAverages(RandomForestClassifier(max_depth=17, n_estimators=200), 'Random Forest', df )
lrAverage = modelAverages(LogisticRegression(C=10.0), 'Linear Regression', df )
svAverage = modelAverages(svm.SVC(kernel = 'linear',C=0.1, gamma=1), 'SVM', df ) # use linear kernel for simplicity
knnAverage = modelAverages(KNeighborsClassifier(n_neighbors=11, weights='distance', p=2, metric='euclidean'), 'KNN', df)
dtAverage = modelAverages(DecisionTreeClassifier(criterion='gini', max_depth=5, min_samples_leaf=5),'Decision Tree', df)
```

Fig 10. Models with Hyperparameter settings

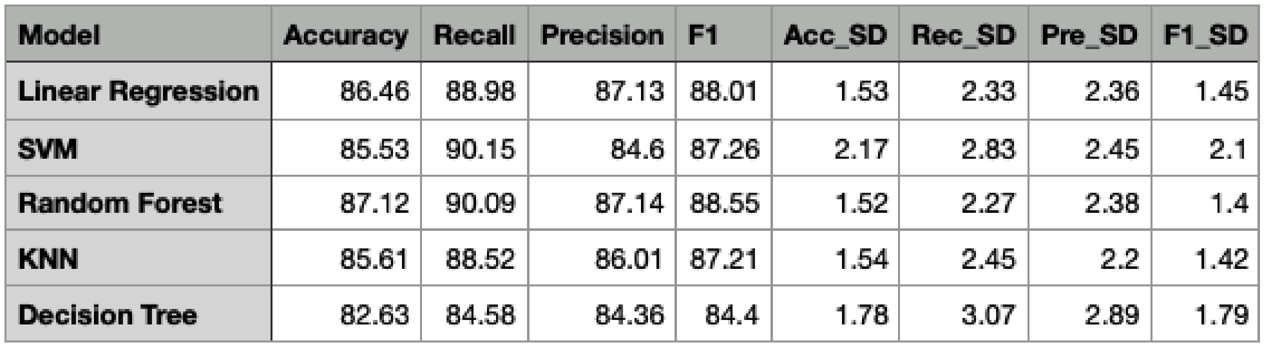


```python
def modelAverages(model, modelName, fData):

    lstAcc = []
    lstRec = []
    lstPre = []
    lstF1 = []
    lstResults = []

    x = fData.drop(['HeartDisease'], axis=1)
    y = fData.HeartDisease

    i = 0
    while i < 50:

        # Train Test Split
        x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.4, shuffle=True)

        model.fit(x_train, y_train)
        Ypred = model.predict(x_test)

        acc = accuracy_score(y_test,Ypred)
        rec = recall_score(y_test,Ypred,pos_label = 1)
        pre = precision_score(y_test,Ypred, pos_label = 1)
        f1 = f1_score(y_test,Ypred, pos_label = 1)

        lstAcc.append(acc)
        lstRec.append(rec)
        lstPre.append(pre)
        lstF1.append(f1)

        i += 1

    avgAcc = round(mean(lstAcc) * 100,2)
    avgRec = round(mean(lstRec) * 100, 2)
    avgPre = round(mean(lstPre) * 100, 2)
    avgF1 = round(mean(lstF1) * 100, 2)

    sdAcc = round(stdev(lstAcc) * 100, 2)
    sdRec = round(stdev(lstRec) * 100, 2)
    sdPre = round(stdev(lstPre) * 100, 2)
    sdF1 = round(stdev(lstF1) * 100, 2)

    lstResults.append(modelName)
    lstResults.append(avgAcc)
    lstResults.append(avgRec)
    lstResults.append(avgPre)
    lstResults.append(avgF1)

    lstResults.append(sdAcc)
    lstResults.append(sdRec)
    lstResults.append(sdPre)
    lstResults.append(sdF1)

    return lstResults, x, y, x_train, x_test, y_train, y_test, Ypred
```

Fig 11. The function that runs each model. Note the *train_test_split* function is set with shuffle = True. Also note returned values include the last version of train/test data.

Evaluation

Performance metrics were applied to all models to produce accuracy, recall, precision, F1 score, and their respective standard deviations shown in table 1.



Model	Accuracy	Recall	Precision	F1	Acc_SD	Rec_SD	Pre_SD	F1_SD
Linear Regression	86.46	88.98	87.13	88.01	1.53	2.33	2.36	1.45
SVM	85.53	90.15	84.6	87.26	2.17	2.83	2.45	2.1
Random Forest	87.12	90.09	87.14	88.55	1.52	2.27	2.38	1.4
KNN	85.61	88.52	86.01	87.21	1.54	2.45	2.2	1.42
Decision Tree	82.63	84.58	84.36	84.4	1.78	3.07	2.89	1.79

Table 2. Model evaluation dataframe

Our model of preference is the model with highest recall value which in this case SVM. In this case, predicting true negatives and limiting false negatives is very important. For example, if a patient was predicted positive, they would schedule a doctor checkup and if it was the case that the prediction was a false positive there was no harm done. However, if the patient was predicted a false negative they would be in danger since they would not need to schedule a doctor checkup according to the prediction.

The Confusion matrix corresponding to the model evaluation table shown in figure 12.

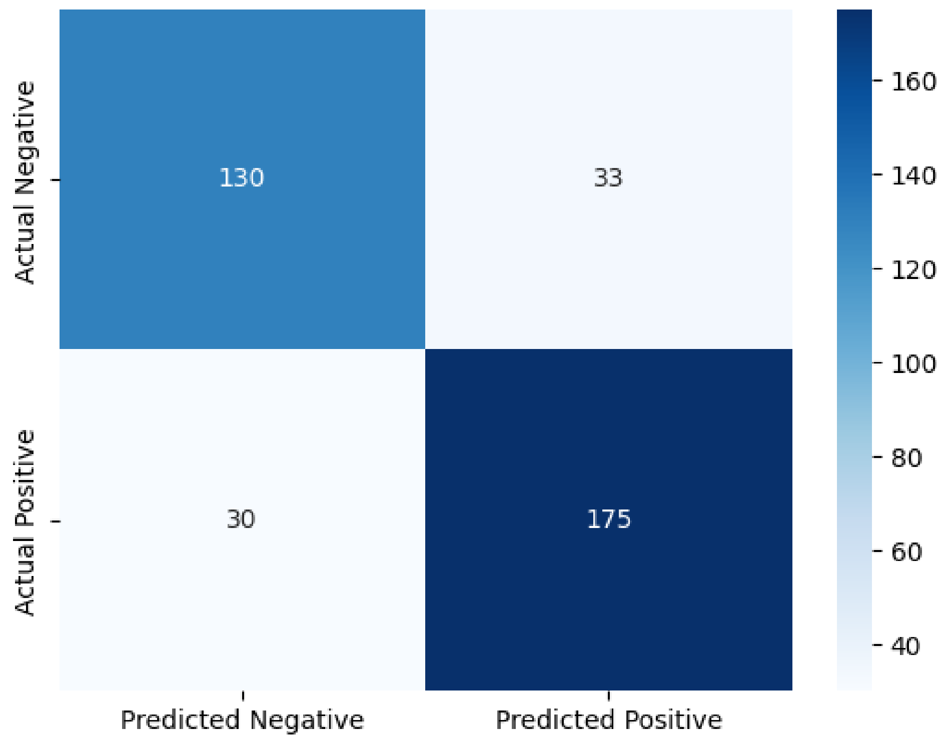


Fig 12. SVM Confusion Matrix

SVM was run once again to show the importance of features shown in figure 13.

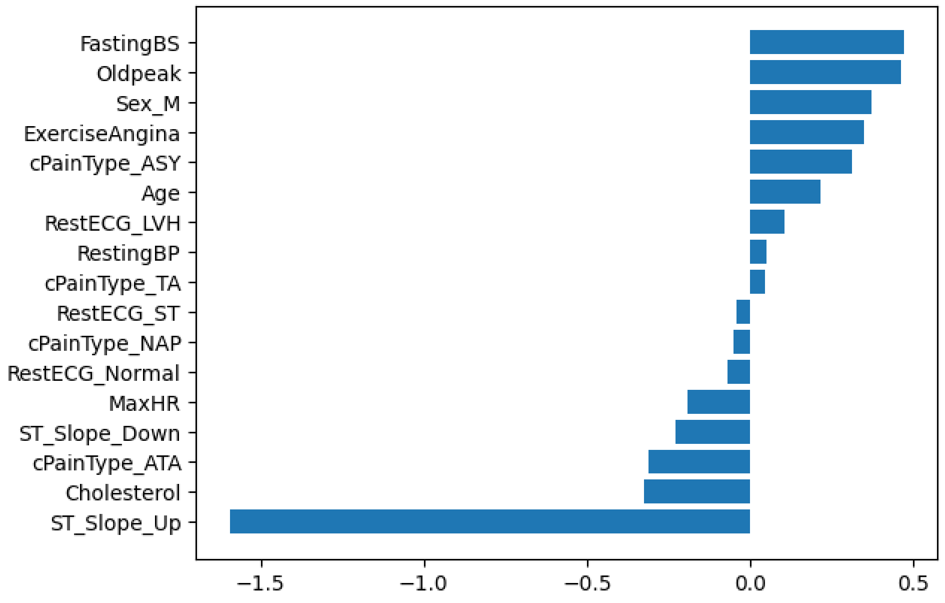


Fig 13. SVM Coefficient Feature Importance

Discussion

The decision to retain the outliers in the dataset when running the models was because with them in place, the performance metrics scored higher. The other reason is that the outlier observations account for around 20% of the entire dataset. However, there was a drawback to this decision. The feature with the most outliers is Cholesterol. It is well known that cholesterol is a major cause of heart disease [5]. However, these particular outliers had values of 0, likely in place of "Null." This caused the Cholesterol feature to play a smaller role in helping the models predict potential CVD than it would have otherwise, in the real world. Thus, it is important to note that these models, while performing well with this dataset, may be biased and not work as well with new observations. Nevertheless, even with this specific dataset, we were able to demonstrate that machine learning could be a useful tool in helping with early detection of CVD in patients.

Conclusion

Machine learning is becoming a mainstream tool in many industries including healthcare. The limitless potential uses for this technology make it primarily only limited by one's own imagination. The ease of access to such open-source technologies can be used to exponentially improve society on many fronts, one of which is healthcare. As has been demonstrated by this project, with a relatively small training dataset, a machine learning model can be trained to help in the early detection of cardiovascular diseases in patients. While further experimentation and analysis is needed, this project provides a clear path forward in using machine learning in healthcare.

One area of further analysis we would recommend is to do a thorough statistical review related to the variations in results based on the training data. For example, every time the Deepnote notebook had to be rerun, the different models would be refit/retrained and would return different accuracy measurements. It would be of benefit to perform statistical hypothesis testing to review the potential benefits of keeping track of each iteration's returned accuracy metrics and averaging them out.

References

[1] fedesoriano. (September 2021). Heart Failure Prediction Dataset. Retrieved 11/05/2022 from https://www.kaggle.com/fedesoriano/heart-failure-prediction.

[2] World Health Organization. (n.d.). *Cardiovascular diseases*. World Health Organization. Retrieved November 5, 2022, from https://www.who.int/health-topics/cardiovascular-diseases#tab=tab_1

[3] zyBooks, a Wiley brand. (2020). CS 6010: Algorithms and Programming Techniques. https://www.zybooks.com/catalog/linear-algebra/ (accessed 2022).

[4] *Learn classification algorithms using Python and scikit-learn*. IBM developer. (n.d.). Retrieved November 6, 2022, from https://developer.ibm.com/tutorials/learn-classification-algorithms-using-python-and-scikit-learn/

[5] Mayo Clinic Staff. (2021, July 20). *Blood cholesterol. Natuonal heart, lung, and blood institute. HTTPS://www.nhlbi.nih.gov/health-topics/blood-cholesterol*. Retrieved December 15, 2022, from https://www.mayoclinic.org/diseases-conditions/high-blood-cholesterol/symptoms-causes/syc-20350800#:~:text=Overview,deposits%20in%20your%20blood%20vessels