CS6140 Project 3

Yihan Xu
NUID: 001566238
Jake Van Meter
NUID: 002965845

Overview:

In this project, we have done data pre-processing, data mining and visualization on a Heart Disease prediction dataset by plotting the **scatter matrix, heatmap** of the dataset, we also explored the dataset by running **lasso regularization, feature importance ranking** and **linear regression** on the dataset, performing **PCA** on the dataset and trying different **mathematical modifications** on the data to fully understand the dataset.

We also tried **5** different classifiers and calculated **accuracy**, **bias and variance, F1 score, confusion matrix, ROC** and **PRC** on each classifier to evaluate the model.

In the end, we tried **3** iterations on **2** top performed classifiers to improve the performance.

Extensions covered:

- We used **5** different classifiers on the dataset when.
- We have experimented with PCA, normalization, and mathematical transformation of the data set.
- We learned how to use lasso regularization for feature selection via identification of insignificant features.
- We learned how to use a random forest classifier to rank the importance of features.
- We carried out **3** iterations on our **2** top performing classifiers.
- We have generated **ROC** and calculated **AUC** for each classifier.

Task 1: Visualizing and pre-processing the data

In this section, we will walk through the steps of our data pre-processing and data mining process, and answer the questions in task 1 in the end.

Convert categorical data into numeric data

At first glance, the data set is heterogeneous with a mixture of categorical and numeric data, with the numerical features having different units.

To better analyze and utilize the dataset, we first converted the categorical features into numeric features by converting each unique value of the feature into its own column with binary values (i.e. 1 for true and 0 for false) (omitting the fourth unique value's column to avoid the dummy variable trap). For example, we converted the **ChestPainType** feature (which contains 4 possible values: ATA, NAP, ASY, TA) into 3 separate feature columns: ATA, NAP, TA, where, ATA=(1, 0, 0), NAP=(0, 1, 0), TA=(0, 0, 1), and ASY=(0, 0, 0).

We also converted the values of **ExerciseAngina** with string values for "Y"/"N" into having "1" for yes and "0" for no. We also converted the values of **Sex** from "F"/"M" into "1" for female and "0" for male.

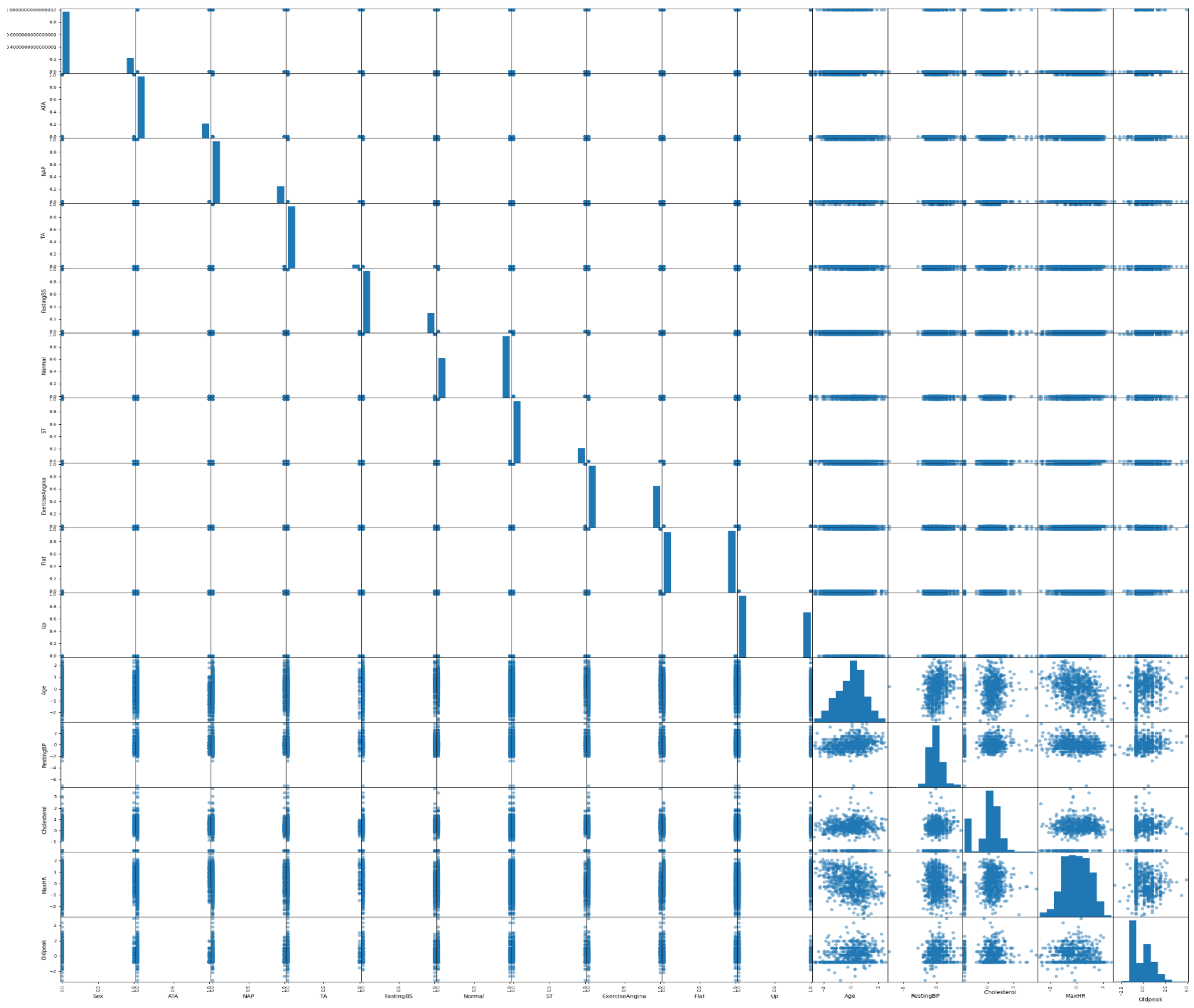
Normalize the dataset

To normalize the dataset we whitened the numeric features by dividing the original data's numeric features' columns by the std of all values in those columns.

Visualize the cleaned dataset

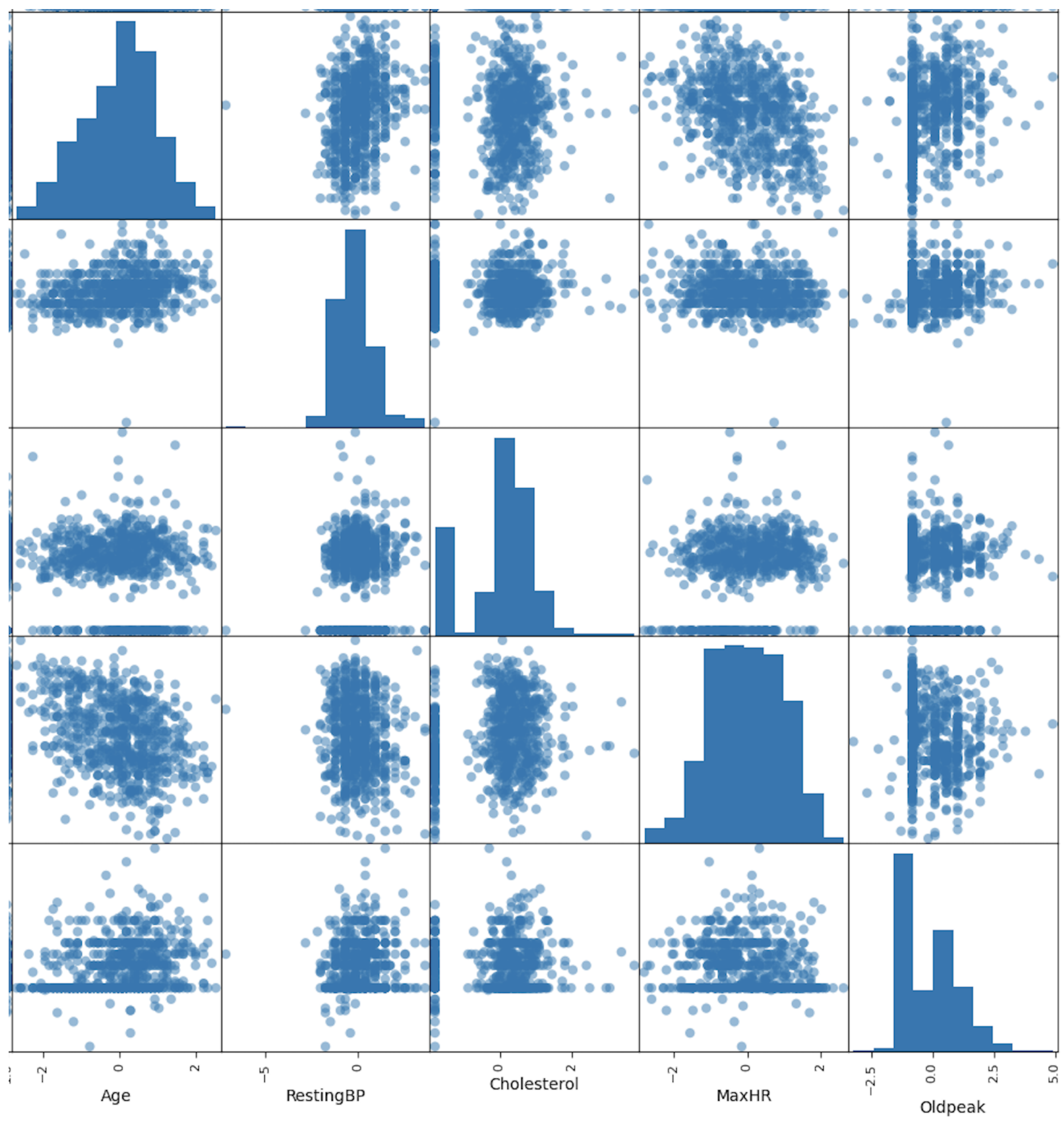
To better understand the dataset, we visualize the dataset by plotting the **scatter matrix** of all features, plotting the **heatmap** of all features to show their correlation, and plotting the **boxplot** of all features to see their distribution.

Scatter matrix:



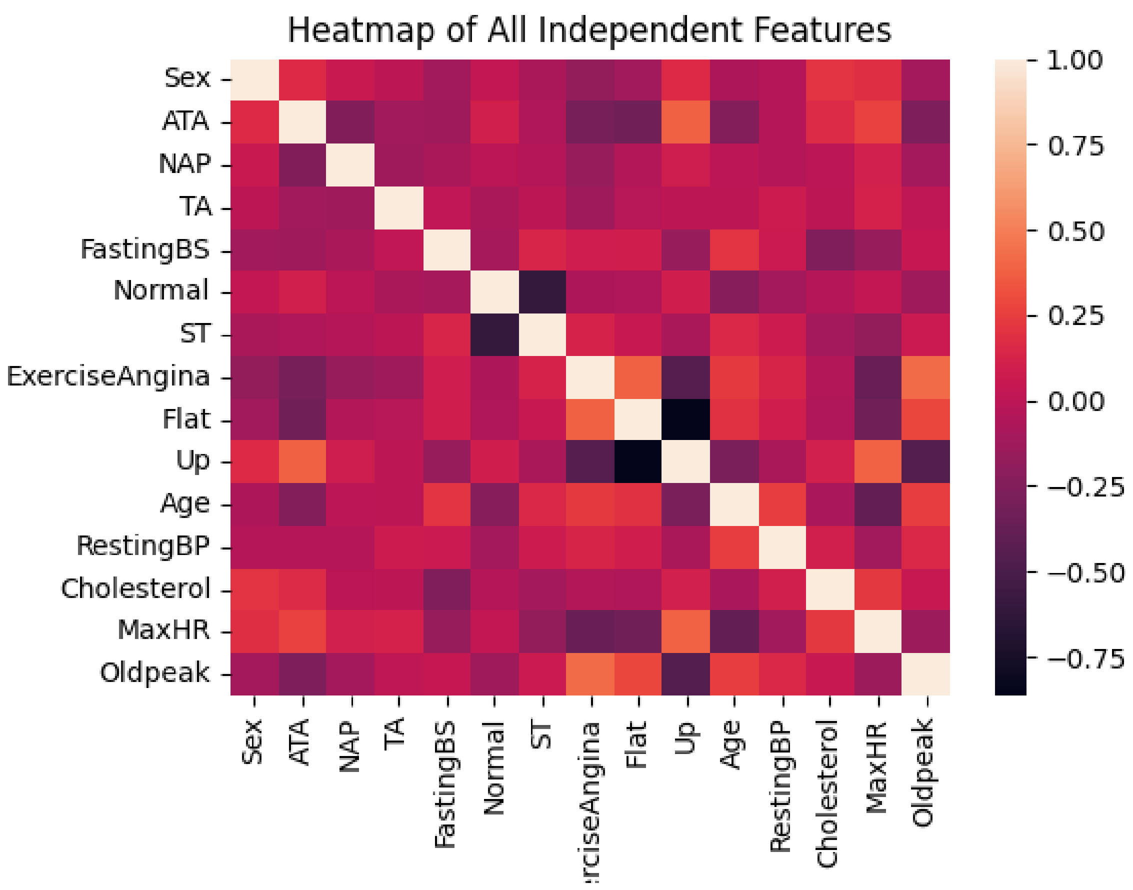
The above matrix shows the plot of all features in pairs, according to this matrix, since most of the features are categorical so we only see data points distributed in 2 edges or 4 points of a cell.

By looking at the histogram in the diagonal direction, we can see the distribution of the data in the feature: most categorical data are distributed not so evenly except for Flat and Up (ST_Slope), and for the numeric features, most data are distributed in the middle range.



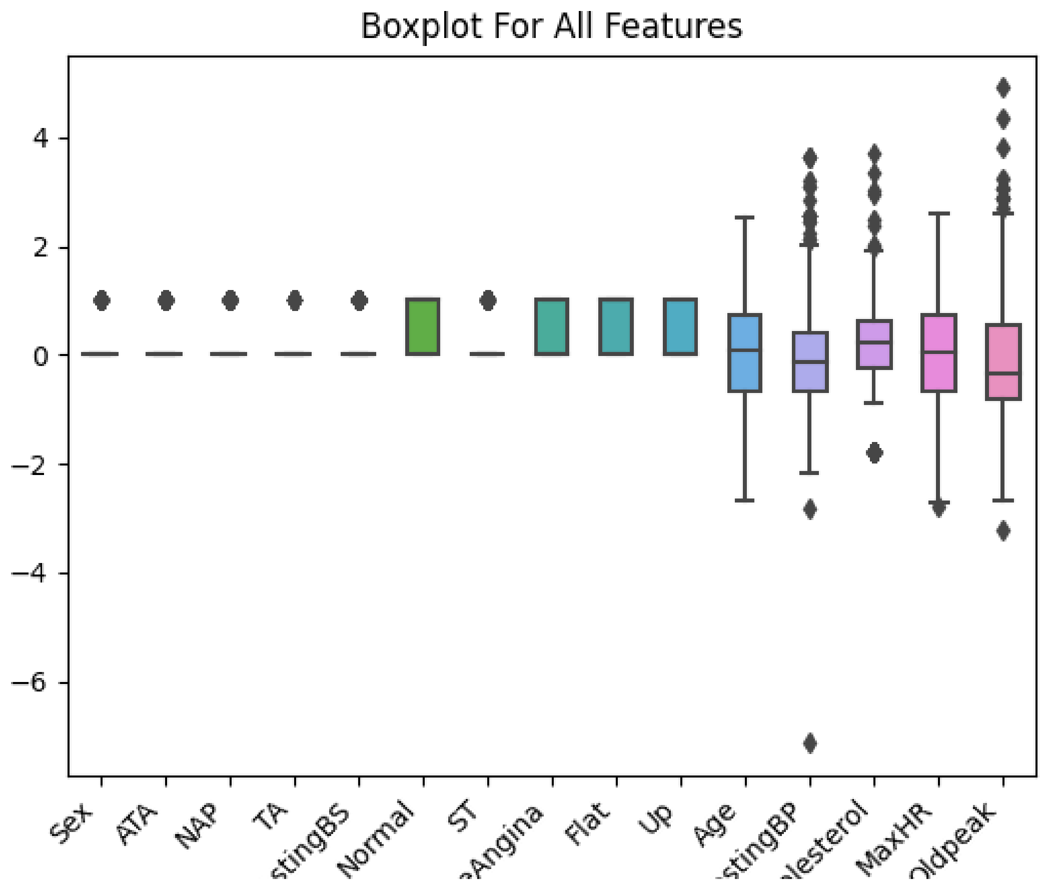
By looking at the plot matrix of all numeric data, We can see there are no obvious natural clusters in it, and most of them are linearly convergent.

Heatmap:



According to the heatmap, we can find the correlations of features: **Up** and **ATA**, **Oldpeak** and **ExerciseAngina**, **MaxHR** and **Up**, **Age** and **FastingBS** have relatively higher positive correlations, **Up** and **Oldpeak**, **Age** and **MaxHR**, **Flat** and **MaxHR**, **ExerciseAngina** and **Up** has relatively higher negative correlations.

Boxplot:



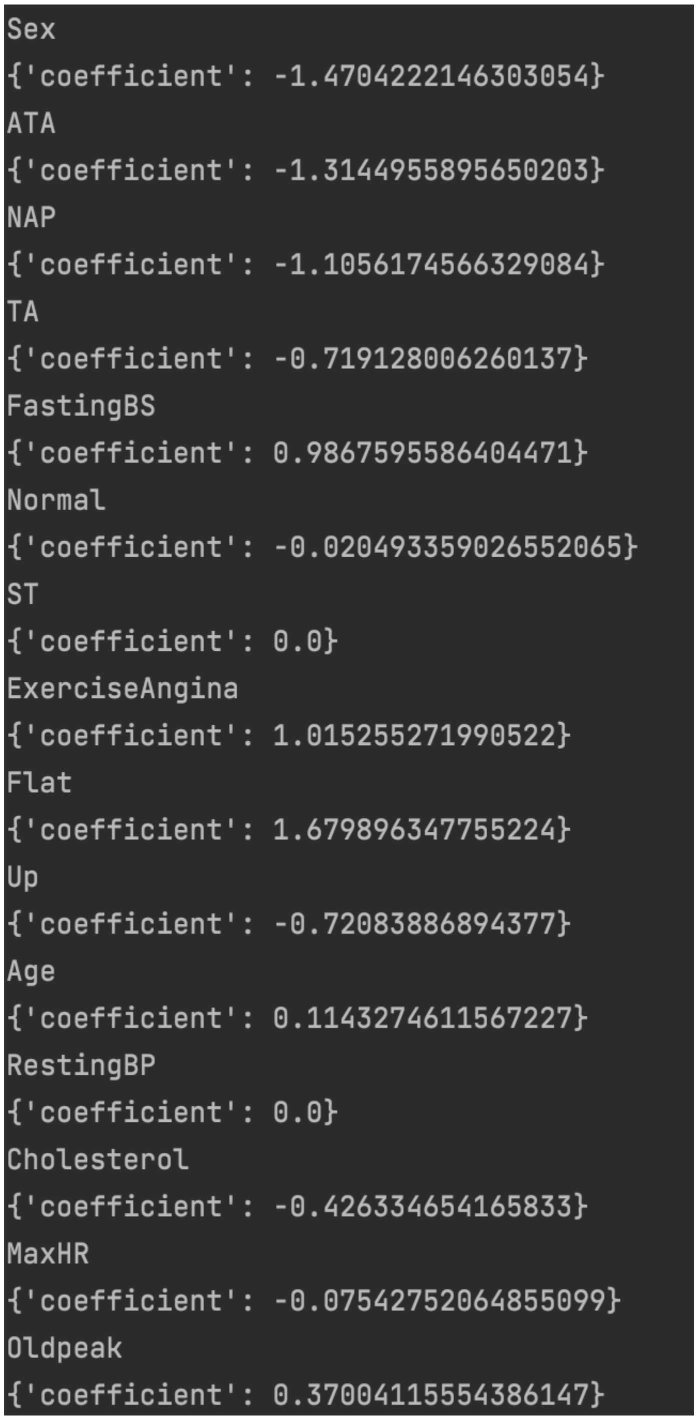
According to the boxplot, we can see the distributions of all features: for the categorical features, it is either 1 or 0, and for the numeric data, they are mostly distributed in the middle range, and convergent.

Analyze the dataset

Feature selection using Logistic Regression with Lasso Regularization:

To understand the feature significance, we used lasso regularization on the features and selected the ones whose coefficient was greater than 0.

Lasso Regularization Coefficients for all independent features:



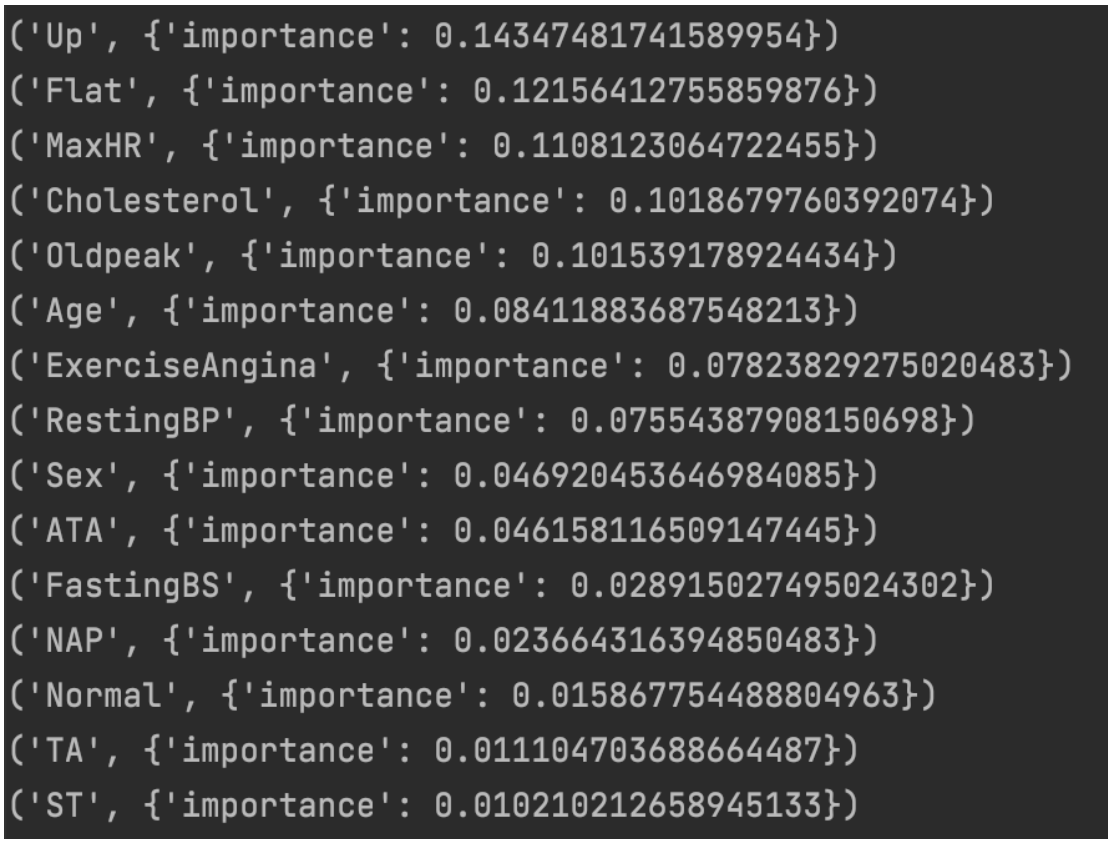
```
Sex
{'coefficient': -1.4704222146303054}
ATA
{'coefficient': -1.3144955895650203}
NAP
{'coefficient': -1.1056174566329084}
TA
{'coefficient': -0.719128006260137}
FastingBS
{'coefficient': 0.9867595586404471}
Normal
{'coefficient': -0.020493359026552065}
ST
{'coefficient': 0.0}
ExerciseAngina
{'coefficient': 1.015255271990522}
Flat
{'coefficient': 1.679896347755224}
Up
{'coefficient': -0.72083886894377}
Age
{'coefficient': 0.1143274611567227}
RestingBP
{'coefficient': 0.0}
Cholesterol
{'coefficient': -0.426334654165833}
MaxHR
{'coefficient': -0.07542752064855099}
Oldpeak
{'coefficient': 0.37004115554386147}
```

Above are the lasso regularization coefficients for all features. There are 2 features with 0 coefficient: **ST** and **RestingBP**. **ST** and **Normal** are dummy variables generated from the **RestingECG** categorical variable. Since **ST** was 0 and **Normal** was close to 0, we concluded that **RestingECG** was insignificant. We also concluded that **RestingBP** was insignificant because its coefficient was also 0.

Feature importance ranking using Random Forest Classifier

We also took advantage of the feature importance ranking feature in **RandomForestClassifier**, below is the feature importance ranking:

Feature importance ranking



```
('Up', {'importance': 0.14347481741589954})
('Flat', {'importance': 0.12156412755859876})
('MaxHR', {'importance': 0.1108123064722455})
('Cholesterol', {'importance': 0.1018679760392074})
('Oldpeak', {'importance': 0.101539178924434})
('Age', {'importance': 0.08411883687548213})
('ExerciseAngina', {'importance': 0.07823829275020483})
('RestingBP', {'importance': 0.07554387908150698})
('Sex', {'importance': 0.04692045364698408 5})
('ATA', {'importance': 0.046158116509147445})
('FastingBS', {'importance': 0.028915027495024302})
('NAP', {'importance': 0.023664316394850483})
('Normal', {'importance': 0.015867754488804963})
('TA', {'importance': 0.011104703688664487})
('ST', {'importance': 0.010210212658945133})
```

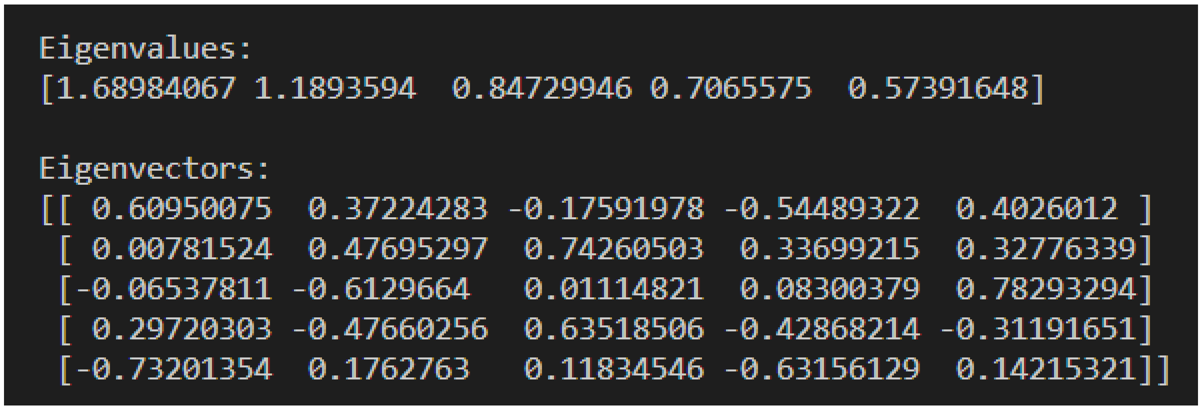
According to the feature importance ranking above, we can see that **ST_Slope** (which is categorized by **Flat** and **Up**) had the highest importance, and going forward we see **MaxHR, Cholesterol, Oldpeak** and so on.

RestingBP had an average importance, **RestingECG** (which is categorized by **ST** and **Normal**) had relatively low importance, and **ChestPainType** (which is categorized by **ATA, NAP, TA** and **ASY**) also had relatively low importance.

Perform PCA on the numeric features

We also performed PCA on the numeric features to find significant features and their correlations.

Eigenvalues and Eigenvectors of performing PCA on whitened numeric data:

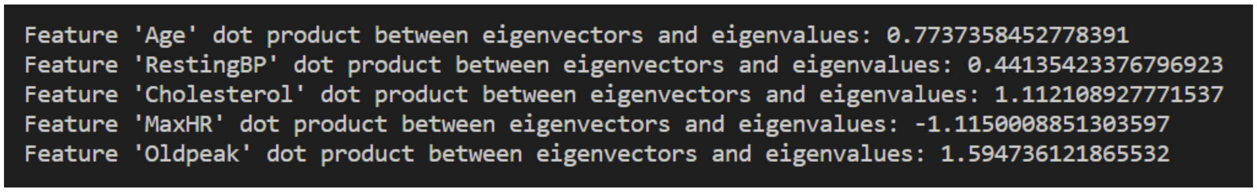


```
Eigenvalues:
[1.68984067 1.1893594  0.84729946 0.7065575  0.57391648]

Eigenvectors:
[[ 0.60950075  0.37224283 -0.17591978 -0.54489322  0.4026012 ]
 [ 0.00781524  0.47695297  0.74260503  0.33699215  0.32776339]
 [-0.06537811 -0.6129664   0.01114821  0.08300379  0.78293294]
 [ 0.29720303 -0.47660256  0.63518506 -0.42868214 -0.31191651]
 [-0.73201354  0.1762763   0.11834546 -0.63156129  0.14215321]]
```

The above image shows the results of performing PCA on the whitened training data's numerical features. The ordering of the features in the eigenvectors from left to right are: **Age**, **RestingBP**, **Cholesterol**, **MaxHR**, and **Oldpeak**. The eigenvalues from largest to smallest are 1.6898, 1.1894, 0.8473, 0.7066, and 0.5739, which together sum to about 5.007. In order to capture about **89%** of the variance in the data, the **first 4 dimensions** must be kept.

According to the coefficient in the first eigenvector, we can see that **Age** (0.609) and **MaxHR** (-0.544) have negative correlations, **Age** (0.609) and **Oldpeak** (0.402) have positive correlations. According to the second eigenvector, **MaxHR** (0.336) and **Oldpeak** (0.327) have positive correlations. According to the third eigenvector, **RestingBP** (-0.612) and **Oldpeak** (0.782) have negative correlation.

Dot product between eigenvalues and eigenvectors for all features:



```
Feature 'Age' dot product between eigenvectors and eigenvalues: 0.7737358452778391
Feature 'RestingBP' dot product between eigenvectors and eigenvalues: 0.44135423376796923
Feature 'Cholesterol' dot product between eigenvectors and eigenvalues: 1.112108927771537
Feature 'MaxHR' dot product between eigenvectors and eigenvalues: -1.1150008851303597
Feature 'Oldpeak' dot product between eigenvectors and eigenvalues: 1.594736121865532
```

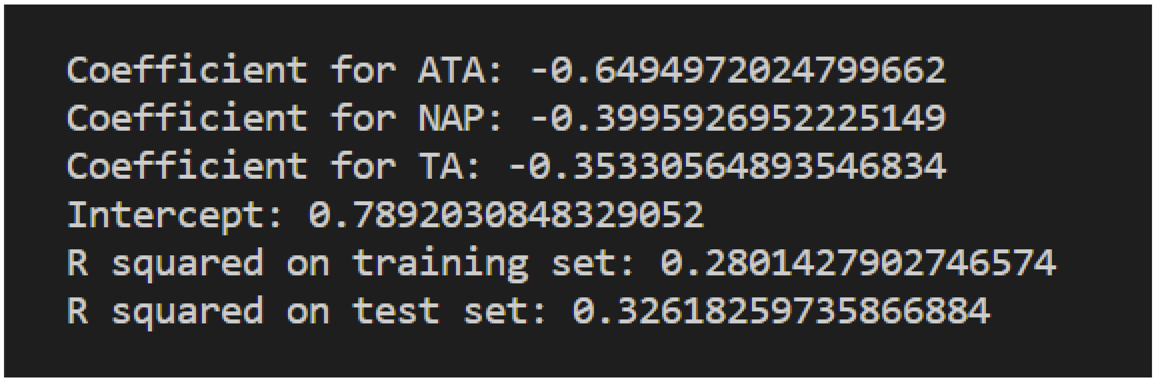
According to the dot product between eigenvalues and eigenvectors for all along the columns axis, we see that **RestingBP** has a relatively low product, which makes it less significant.

Perform Linear Regression on the independent features

We performed linear regression individually on the independent features to see their correlation with the dependent variable and the performance of each independent feature.

ChestPainType, **ST_Slope**, and **ExerciseAngina** were the features whose linear regression models obtained the top three highest r-squared scores on the test data. Linear regression models trained on those non-transformed features had r square scores of: 0.3262, 0.3123, and 0.2707 respectively.

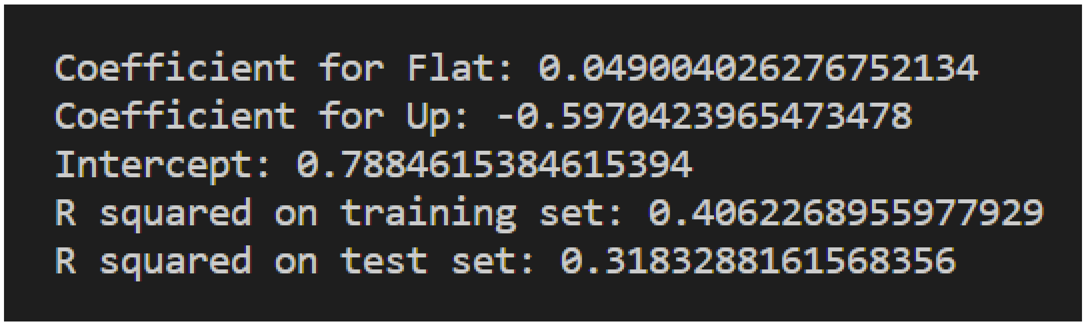
Simple Linear Regression with ChestPainType as predictor:



```
Coefficient for ATA: -0.6494972024799662
Coefficient for NAP: -0.3995926952225149
Coefficient for TA: -0.35330564893546834
Intercept: 0.7892030848329052
R squared on training set: 0.2801427902746574
R squared on test set: 0.32618259735866884
```

This relatively high r-squared score combined with the large lasso regularization coefficient values (**ATA** = -1.3145, **NAP** = -1.1056, **TA** = -0.7191), implies that **ChestPainType** is worth using as an input feature despite its relatively low importance scores from the random forest classifier (**ATA** = 0.0411, **NAP** = 0.0213, **TA** = 0.0097).

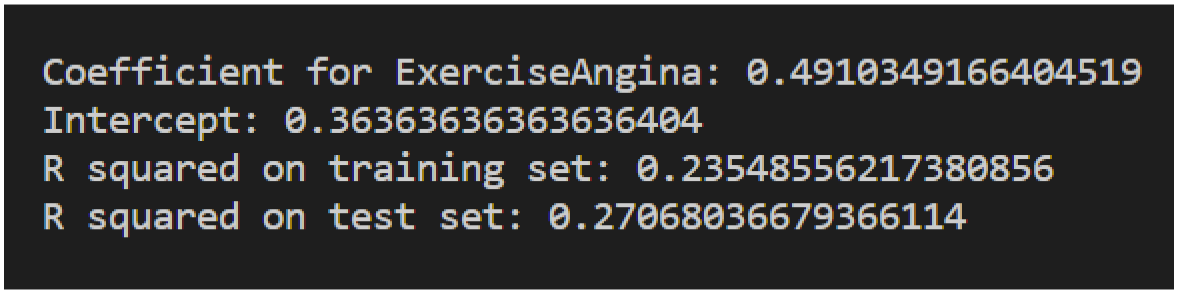
Simple Linear Regression with ST_Slope as predictor:



```
Coefficient for Flat: 0.049004026276752134
Coefficient for Up: -0.5970423965473478
Intercept: 0.7884615384615394
R squared on training set: 0.4062268955977929
R squared on test set: 0.3183288161568356
```

Along with this relatively high r-squared score, **ST_Slope** had large lasso regularization coefficient values (**Flat** = 1.6799, **Up** = -0.7208) and somewhat large random forest classifier importance values (**Flat** = 0.1054, **Up** = 0.1696) implies that **ST_Slope** is worth using as an input feature.

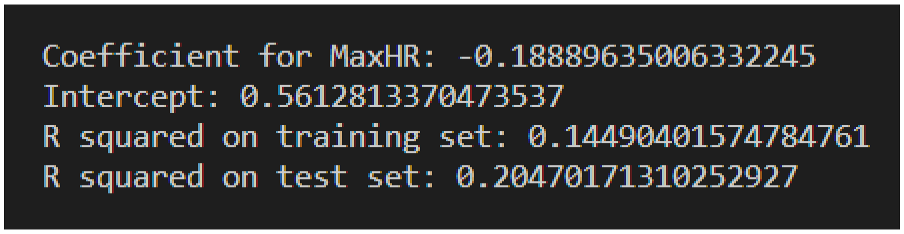
Simple Linear Regression with ExerciseAngina as predictor:



```
Coefficient for ExerciseAngina: 0.4910349166404519
Intercept: 0.3636363636363636404
R squared on training set: 0.23548556217380856
R squared on test set: 0.27068036679366114
```

Similar to **ST_Slope**, **ExerciseAngina** had a relatively large lasso regularization coefficient value (1.0153) and random forest classifier importance ranking (0.071).

Although **MaxHR** had one of the lowest lasso regression coefficients, it was not 0 and there are three main reasons why we are using it as an independent feature. Firstly, the linear regression model that used **MaxHR** as the predictor term obtained the fourth highest r-squared score on the test data.

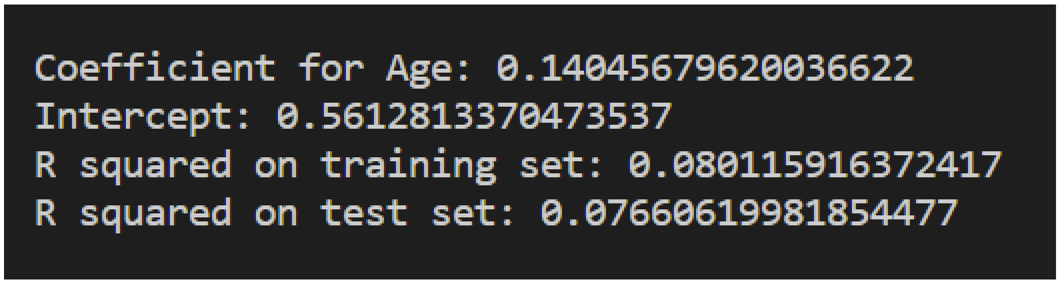


```
Coefficient for MaxHR: -0.18889635006332245
Intercept: 0.5612813370473537
R squared on training set: 0.14490401574784761
R squared on test set: 0.20470171310252927
```

Secondly, It had one of the highest importance scores as determined by the aforementioned RFC classifier with a value of 0.1108.

Finally, **MaxHR** was one of the largest contributors to variance in the data as shown by its eigenvalue-eigenvector dot product (-1.115).

Simple Linear Regression with Age as predictor:

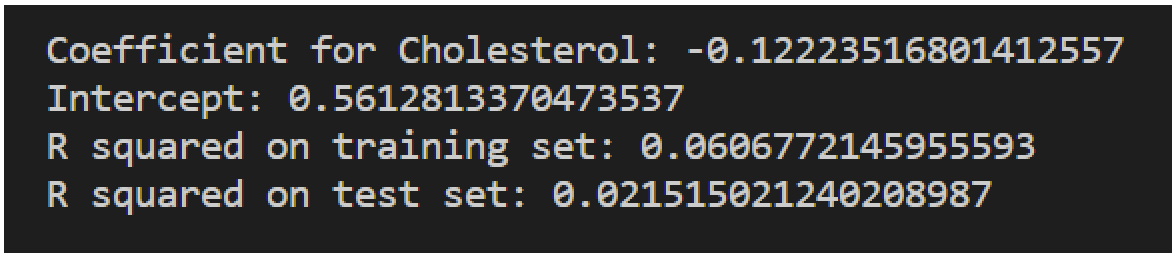


```
Coefficient for Age: 0.14045679620036622
Intercept: 0.5612813370473537
R squared on training set: 0.080115916372417
R squared on test set: 0.07660619981854477
```

The **Age** feature had a relatively low r-squared score (0.0766) for simple linear regression.

As will be discussed in the derived features section, this score was slightly improved by using the square root of the input values, it did not improve it by much. **Age** also had a relatively low lasso regression coefficient value (0.1143). Nevertheless, it had a mid-sized importance value (0.0803, 6th largest) and was not the lowest contributor to the variation as seen by its eigenvector-eigenvalue dot product on the whitened data (0.7737, 4th largest). Therefore, we decided that it was worth keeping as an input feature.

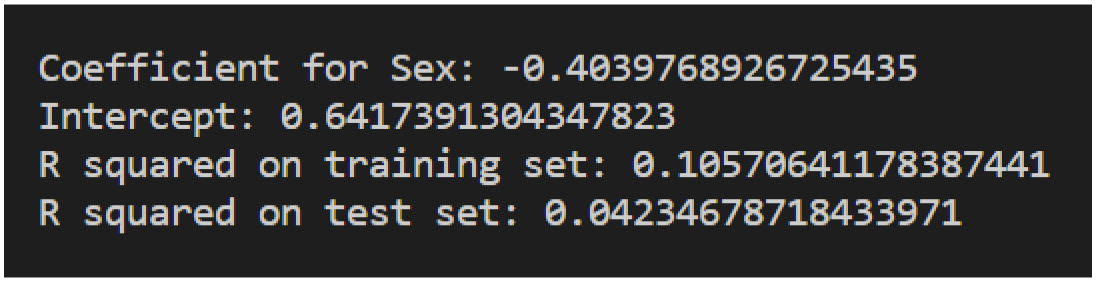
Simple Linear Regression with Cholesterol as predictor:



```
Coefficient for Cholesterol: -0.12223516801412557
Intercept: 0.5612813370473537
R squared on training set: 0.0606772145955593
R squared on test set: 0.021515021240208987
```

The **Cholesterol** feature had a low r-squared (0.0215) for simple linear regression. However, as will be discussed in the derived features section, this score was greatly improved by using the squared values as input. Moreover, **Cholesterol** had one of the largest random forest classifier importance values (0.1093, 3rd largest) and was one of the largest contributors to the variance in the data as shown by its eigenvector-eigenvalue dot product on the whitened data (1.1121, 3rd largest but not much difference from 2nd largest).

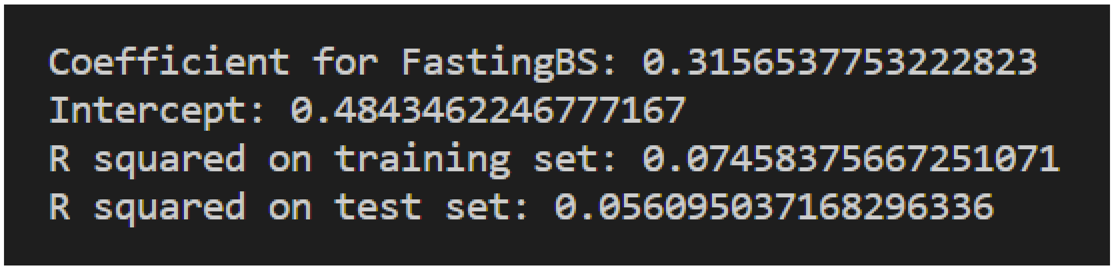
Simple Linear Regression with Cholesterol as predictor:



```
Coefficient for Sex: -0.4039768926725435
Intercept: 0.6417391304347823
R squared on training set: 0.10570641178387441
R squared on test set: 0.04234678718433971
```

The **Sex** feature had a low r-squared (0.0424) for simple linear regression. It also had a relatively low random forest classifier importance value (0.0448, 9th largest). But it had a very large lasso normalization coefficient value (-1.4704) which is why we used it as an input feature.

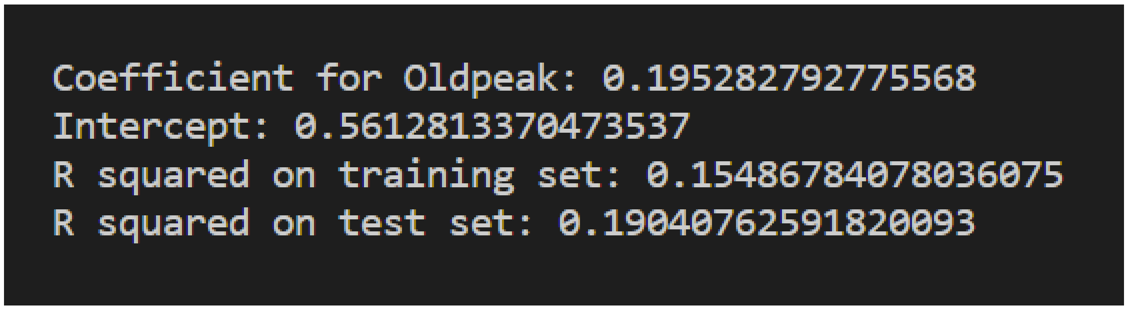
Simple Linear Regression with FastingBS as predictor:



```
Coefficient for FastingBS: 0.3156537753222823
Intercept: 0.4843462246777167
R squared on training set: 0.07458375667251071
R squared on test set: 0.056095037168296336
```

The **FastingBS** feature had a low r-squared (0.0561) for simple linear regression. It also had a relatively low random forest classifier importance value (0.0289, 11th largest). However, it had a somewhat large lasso normalization coefficient (0.9868) implying that it was worth using as an input feature.

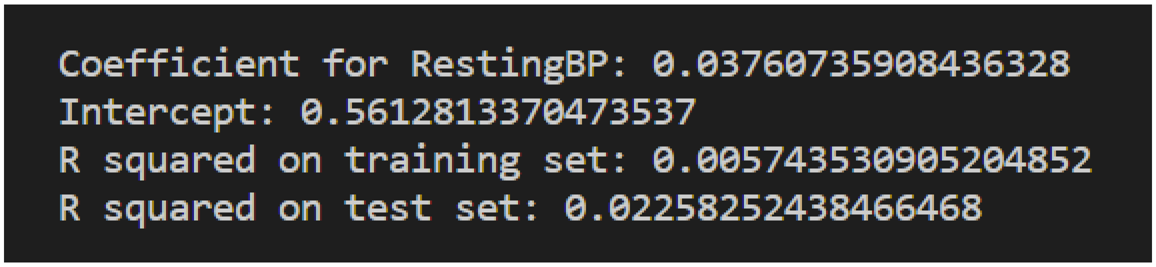
Simple Linear Regression with Oldpeak as predictor:



```
Coefficient for Oldpeak: 0.195282792775568
Intercept: 0.5612813370473537
R squared on training set: 0.15486784078036075
R squared on test set: 0.19040762591820093
```

The **Oldpeak** feature had a relatively mediocre r-squared (0.1904) for simple linear regression. This r-square was slightly improved by taking the square root of the input values. It also had a large random forest classifier importance value (0.1062) and was shown to contribute greatly to the variance in the data by the eigenvector-eigenvalue dot product (1.5947). Therefore, despite its relatively small lasso regression coefficient (0.37), we decided to use it as an input feature.

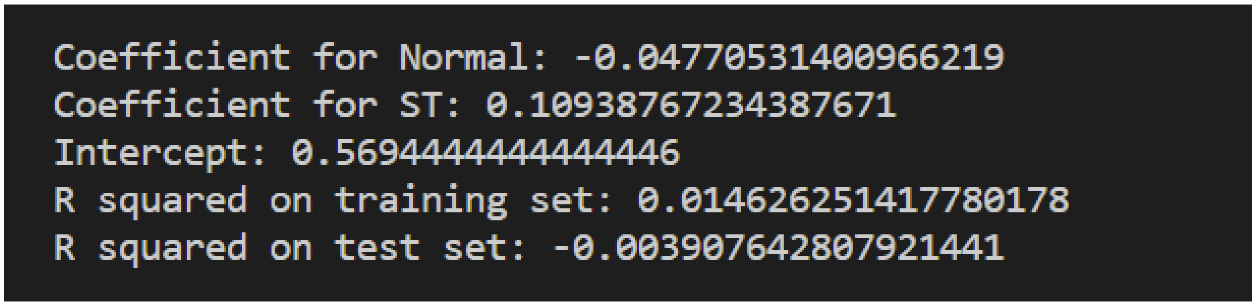
Simple Linear Regression with RestingBP as predictor:



```
Coefficient for RestingBP: 0.03760735908436328
Intercept: 0.5612813370473537
R squared on training set: 0.005743530905204852
R squared on test set: 0.02258252438466468
```

The **RestingBP** feature had a low r-squared (0.0226) for simple linear regression. It also had the lasso regression coefficient of 0 which is why we decided to not use it as an input feature.

Simple Linear Regression with RestingECG as predictor:



```
Coefficient for Normal: -0.04770531400966219
Coefficient for ST: 0.10938767234387671
Intercept: 0.5694444444444446
R squared on training set: 0.014626251417780178
R squared on test set: -0.003907642807921441
```

The **RestingECG** feature had the lowest r-squared (-0.0039) of all features for simple linear regression. It also had some of the lowest random forest classifier importance values (**Normal** = 0.0171, **ST** = 0.0102) and insignificant lasso regression coefficients (**Normal** = -0.0205, **ST** = 0). Therefore, all signals implied that we should not use **RestingECG** as an input feature.

Experiencing Mathematical modification on the Numeric features:

We also made some mathematical modifications on the input features before running linear regression to see if there would be any improvement on the model's performance. Please find details in 1.E.

Summary Answers to Task 1 Questions

1.A What variables do you plan to use as the input features?

According to the analysis on the dataset, we plan to used a total of 9 variables as input features:

1. ChestPainType
2. ST_Slope
3. ExerciseAngina
4. MaxHR
5. Age
6. Cholesterol
7. Sex
8. FastingBS
9. OldPeak

All the above features have relatively high score in feature importance, and have non-zero coefficient in logistic regression with lasso regularization, also performed well in linear regression.

The two variables we omitted based on their insignificance are (according to **lasso regularization, feature importance ranking** and **PCA** analysis):

1. RestingBP
2. RestingECG

1.B What pre-processing (if any) did you execute on the variables?

- We converted categorical features into binary features.
- We whitened the numeric data by subtracting each value by the column mean and dividing by the standard deviation.
- We performed PCA on the whitened numeric data.

1.C Which independent variables are strongly correlated (positively or negatively)?

According to the heatmap:

Up and **ATA**, **Oldpeak** and **ExerciseAngina**, **MaxHR** and **Up**, **Age** and **FastingBS** have relatively strong positive correlations, **Up** and **Oldpeak**, **Age** and **MaxHR**, **Flat** and **MaxHR**, **ExerciseAngina** and **Up** have relatively strong negative correlations.

According to the PCA eigenvectors:

Age and **MaxHR** are negatively correlated: in the first eigenvector **Age** has a value of 0.609 and **MaxHR** has a value of -0.544. **Age** and **Oldpeak** are positively correlated: in the first eigenvector **Age** has a value of 0.609 and **Oldpeak** has a value of 0.403. **MaxHR** and **Oldpeak** are negatively correlated: in the first eigenvector **MaxHR** has a value of -0.544 and **Oldpeak** has a value of 0.403. **RestingBP** and **Oldpeak** are positively correlated: in the first eigenvector **RestingBP** has a value of 0.372 and **Oldpeak** has a value of 0.403.

1.D How many significant signals exist in the independent variables?

There were significant signals in the:
- **RandomForestClassifier**'s feature importance ranking,
- The lasso regression coefficients of all features,
- The PCA eigenvalues and feature values in the eigenvectors.

The significant signals are: **ST_Slope, MaxHR, Cholesterol, Oldpeak, Age**.

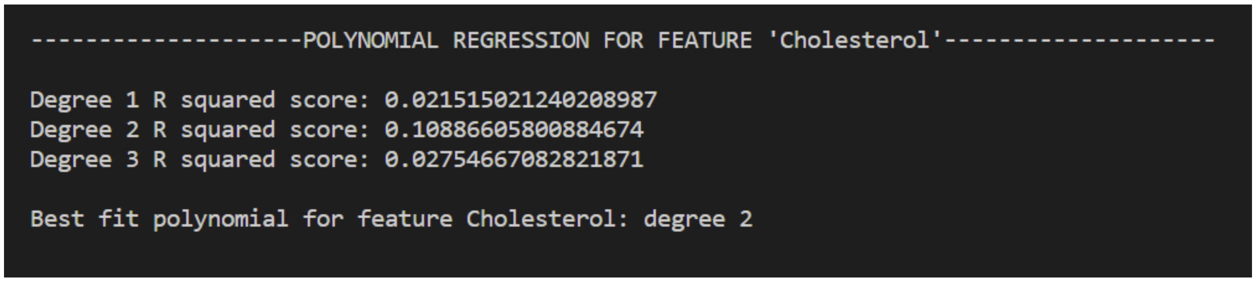
1.E What derived or alternative features might be useful for analysis (e.g. polynomial features)?

We experimented with four types of transformations on the numerical input variables before running linear regression on them: taking the square of the input features, taking the cube of the input features, taking the square root of the input features, and taking the cosine of the input features.

Polynomial Regression

After transforming the input features by squaring and cubing them, the only feature whose linear regression r-squared score improved was **Cholesterol**.

Polynomial regression with Cholesterol as predictor:

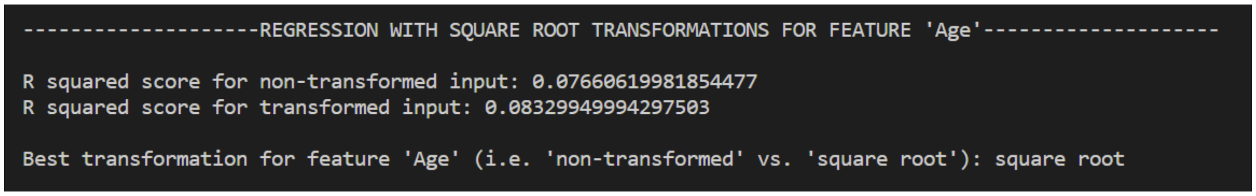

```
--------------------POLYNOMIAL REGRESSION FOR FEATURE 'Cholesterol'--------------------

Degree 1 R squared score: 0.021515021240208987
Degree 2 R squared score: 0.10886605800884674
Degree 3 R squared score: 0.02754667082821871

Best fit polynomial for feature Cholesterol: degree 2
```

Here we can see that the r-squared scores for the polynomial transformations of **Cholesterol** were about 0.0215 for non-transformed, 0.1089 for squared transformation, and 0.0275 for cubed transformation. This shows that by squaring the input values of the **Cholesterol** feature before running linear regression improves the coefficient of determination by almost 500%. Therefore, it may be useful to transform the **Cholesterol** feature by squaring the values for analysis.

Square Root Transformations

After transforming the input features by taking their square root, the linear regression models' for **Age**, **RestingBP**, and **Oldpeak** had improved r-squared scores.

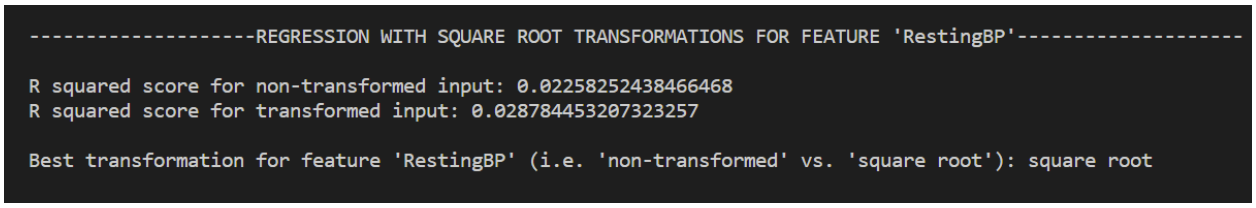
Regression with square root of Age as predictor:



```
--------------------REGRESSION WITH SQUARE ROOT TRANSFORMATIONS FOR FEATURE 'Age'--------------------

R squared score for non-transformed input: 0.07660619981854477
R squared score for transformed input: 0.08329949994297503

Best transformation for feature 'Age' (i.e. 'non-transformed' vs. 'square root'): square root
```

Here we see that using the square root of the **Age** feature's values as input improved its linear regression model's r-squared score from 0.0766 to 0.0833, an improvement of 0.0067.

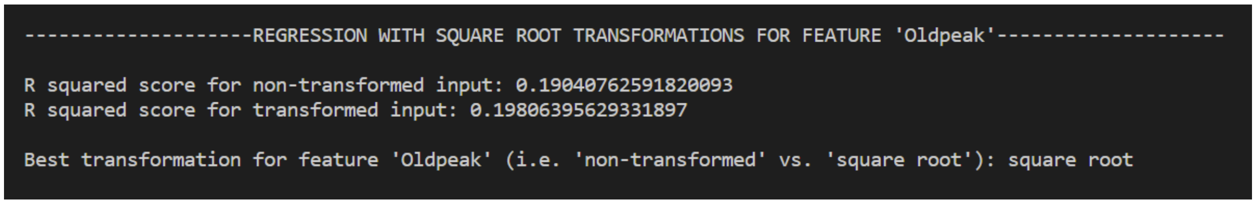
Regression with square root of RestingBP as predictor:



```
--------------------REGRESSION WITH SQUARE ROOT TRANSFORMATIONS FOR FEATURE 'RestingBP'--------------------

R squared score for non-transformed input: 0.02258252438466468
R squared score for transformed input: 0.028784453207323257

Best transformation for feature 'RestingBP' (i.e. 'non-transformed' vs. 'square root'): square root
```

This image shows that using the square root of the **RestingBP** feature's values as input improved its linear regression model's r-squared score from 0.0226 to 0.0288, an improvement of 0.0062.

Regression with square root of Oldpeak as predictor:



```
--------------------REGRESSION WITH SQUARE ROOT TRANSFORMATIONS FOR FEATURE 'Oldpeak'--------------------

R squared score for non-transformed input: 0.19040762591820093
R squared score for transformed input: 0.19806395629331897

Best transformation for feature 'Oldpeak' (i.e. 'non-transformed' vs. 'square root'): square root
```

Here we see that using the square root of the “Oldpeak” feature’s values as input improved the linear regression model’s r-squared score from 0.1904 to 0.1981, an improvement of about 0.0077.

Task 2, 3: Apply and Evaluate classifiers

We tried 5 different types of classifiers on the data:

1) K Nearest Neighbors
2) Logistic Regression
3) Support Vector machine
4) Naive Bayes
5) Decision tree

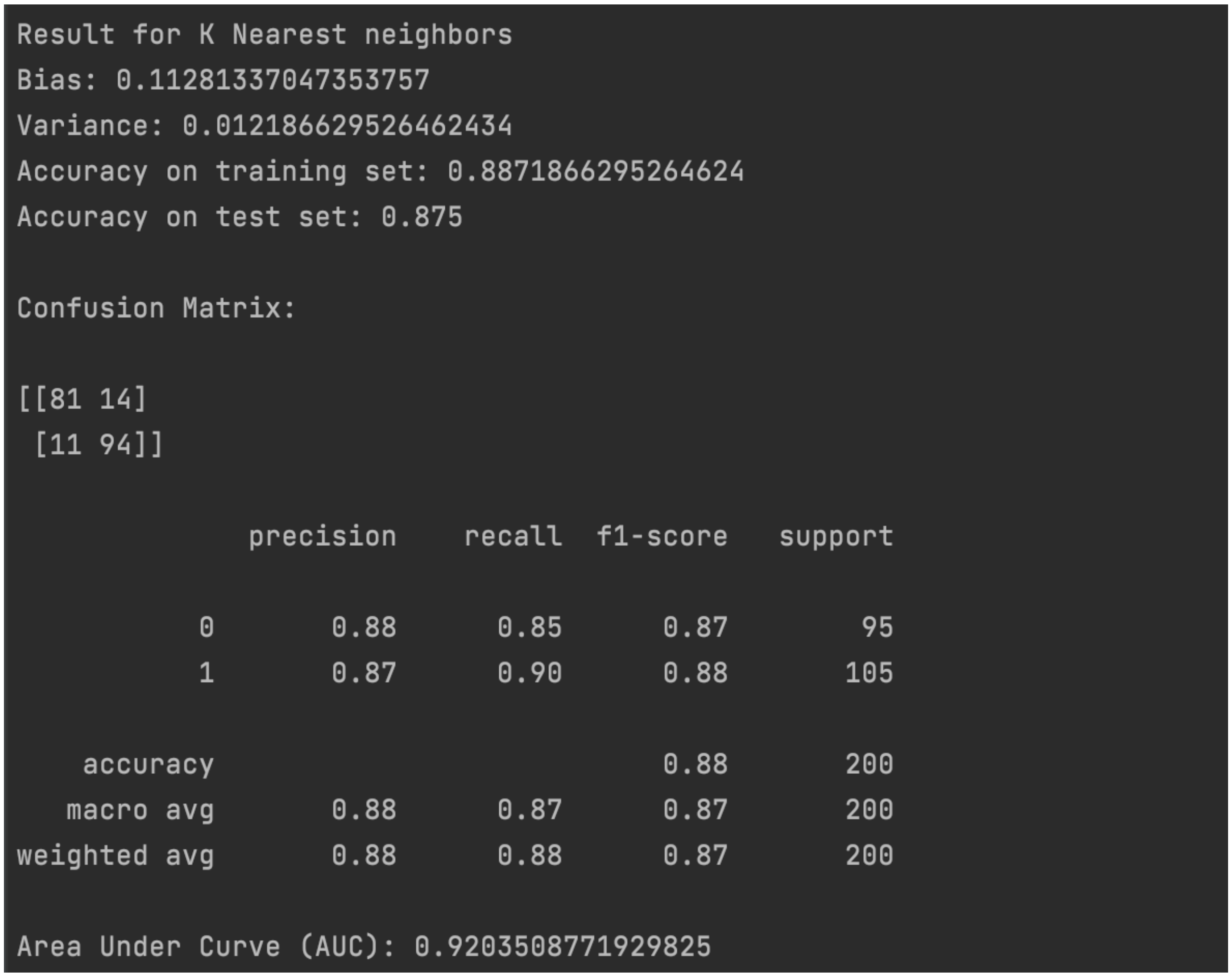
We will show the descriptions of parameters and evaluations on each classifier as follows.

K Nearest Neighbors

We used the sklearn.neighbors.KNeighborsClassifier with n_neighbors=10 and the other metaparameters set to their default values.

For the input data we used the preprocessed dataset with numeric data normalized and the two insignificant features (**RestingBP**, **RestingECG**) dropped.

Result of running KNN on the dataset:



```
Result for K Nearest neighbors
Bias: 0.11281337047353757
Variance: 0.012186629526462434
Accuracy on training set: 0.8871866295264624
Accuracy on test set: 0.875

Confusion Matrix:

[[81 14]
 [11 94]]

              precision    recall  f1-score   support

           0       0.88      0.85      0.87        95
           1       0.87      0.90      0.88       105

    accuracy                           0.88       200
   macro avg       0.88      0.87      0.87       200
weighted avg       0.88      0.88      0.87       200

Area Under Curve (AUC): 0.9203508771929825
```

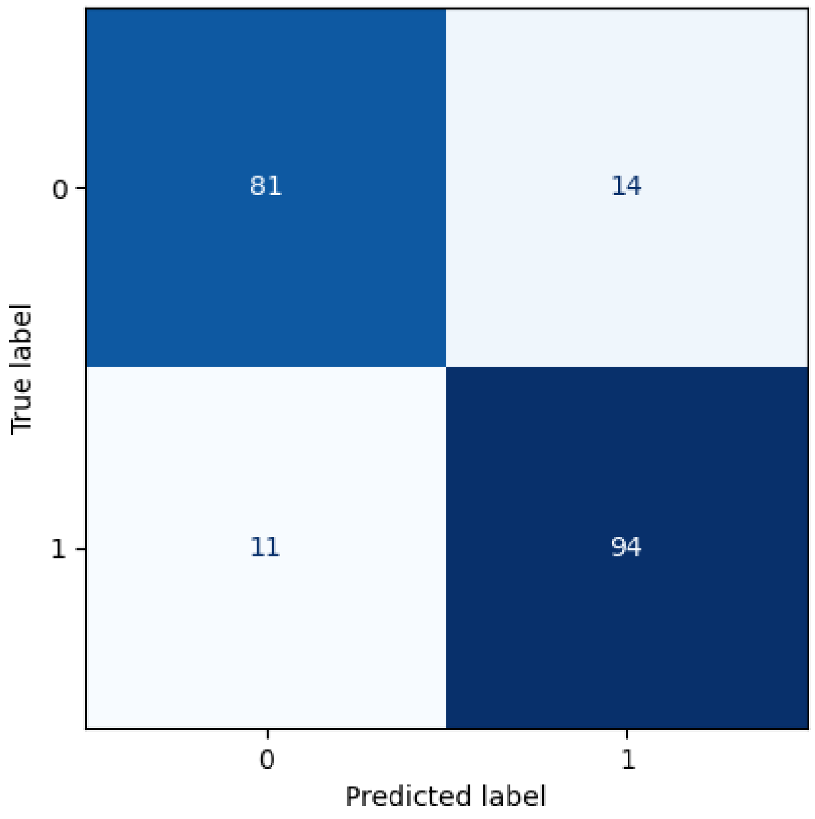
Bias and Variance:
According to the bias and variance of KNN the classifier with k=10, the bias was relatively high and variance was relatively low. To improve this, we need a more powerful model, we could:
1. Lower the number of k to prevent overfitting
2. Use more features (add the 2 dropped features back)
3. Decrease regularization

Accuracy:
Both training and test set had good accuracy and F1 score which is around 0.88 with **HeartDisease**=1 as the positive label.

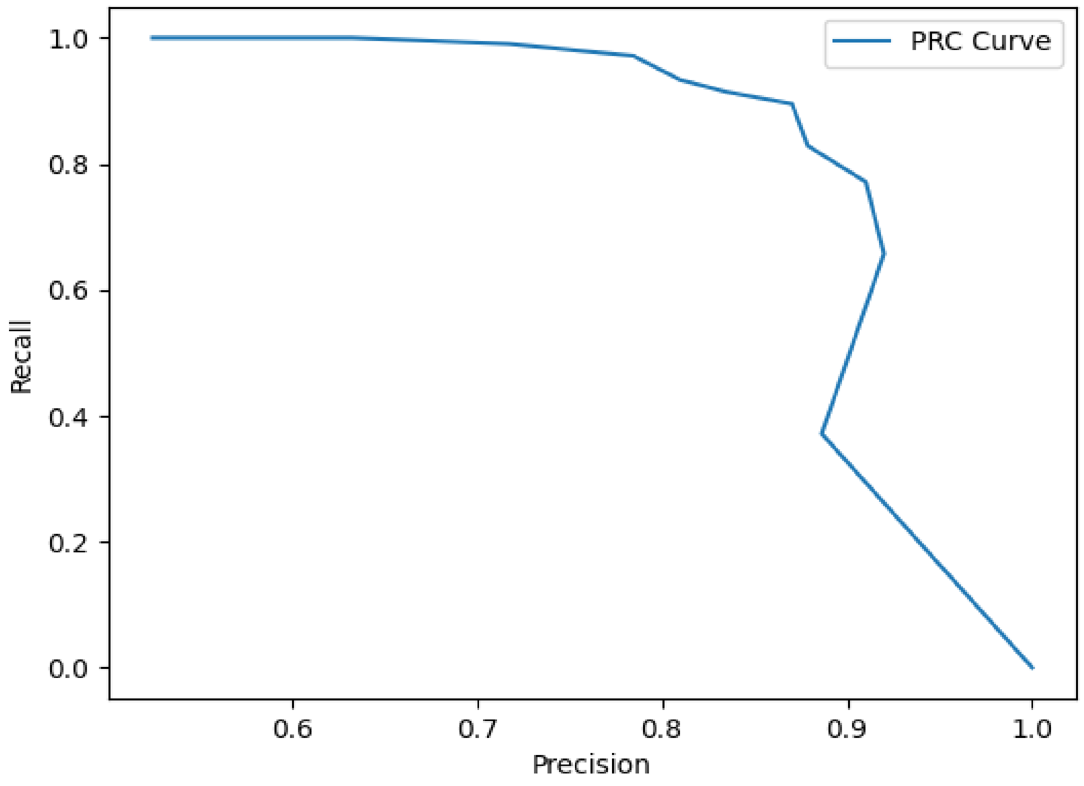
Confusion Matrix:
According to the confusion matrix, we have 81 true positives and 94 true negatives, 14 false positives and 11 false negatives, it is a relatively balanced ratio. Since this is a medical related dataset, it is better to have higher false positives, because it's safer for the patients.

AUC:
We have 0.92 under ROC, which is a relatively good area.

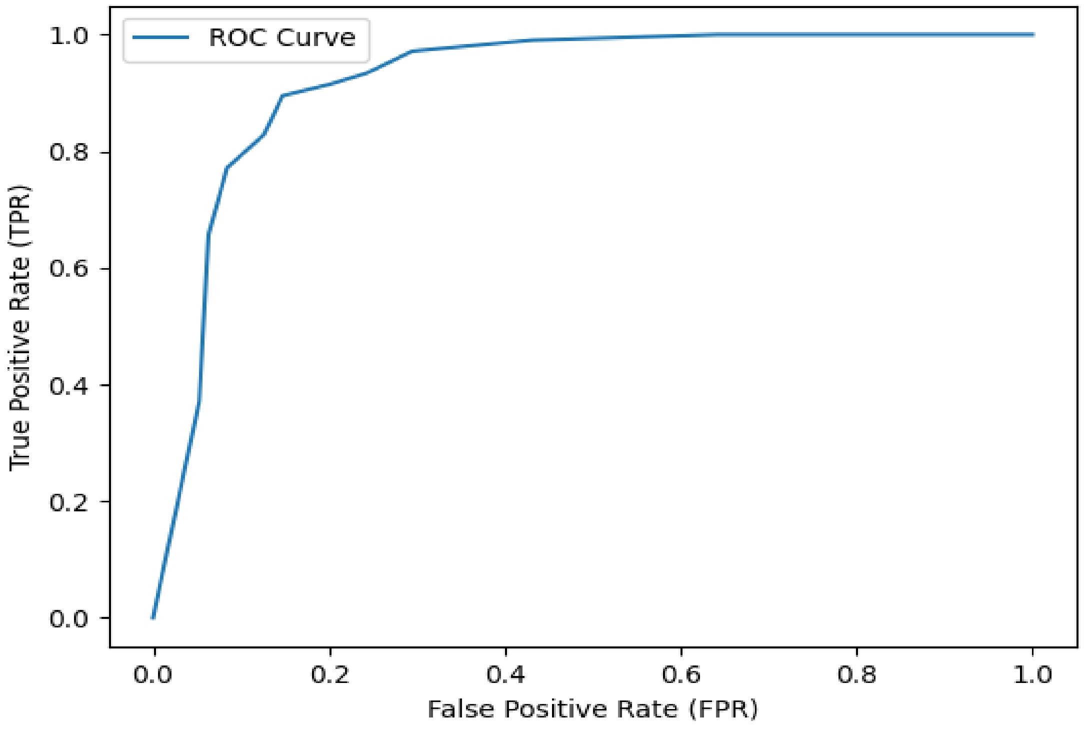
Confusion matrix of running KNN on the dataset:



PRC of running KNN on the dataset:



ROC of running KNN on the dataset:

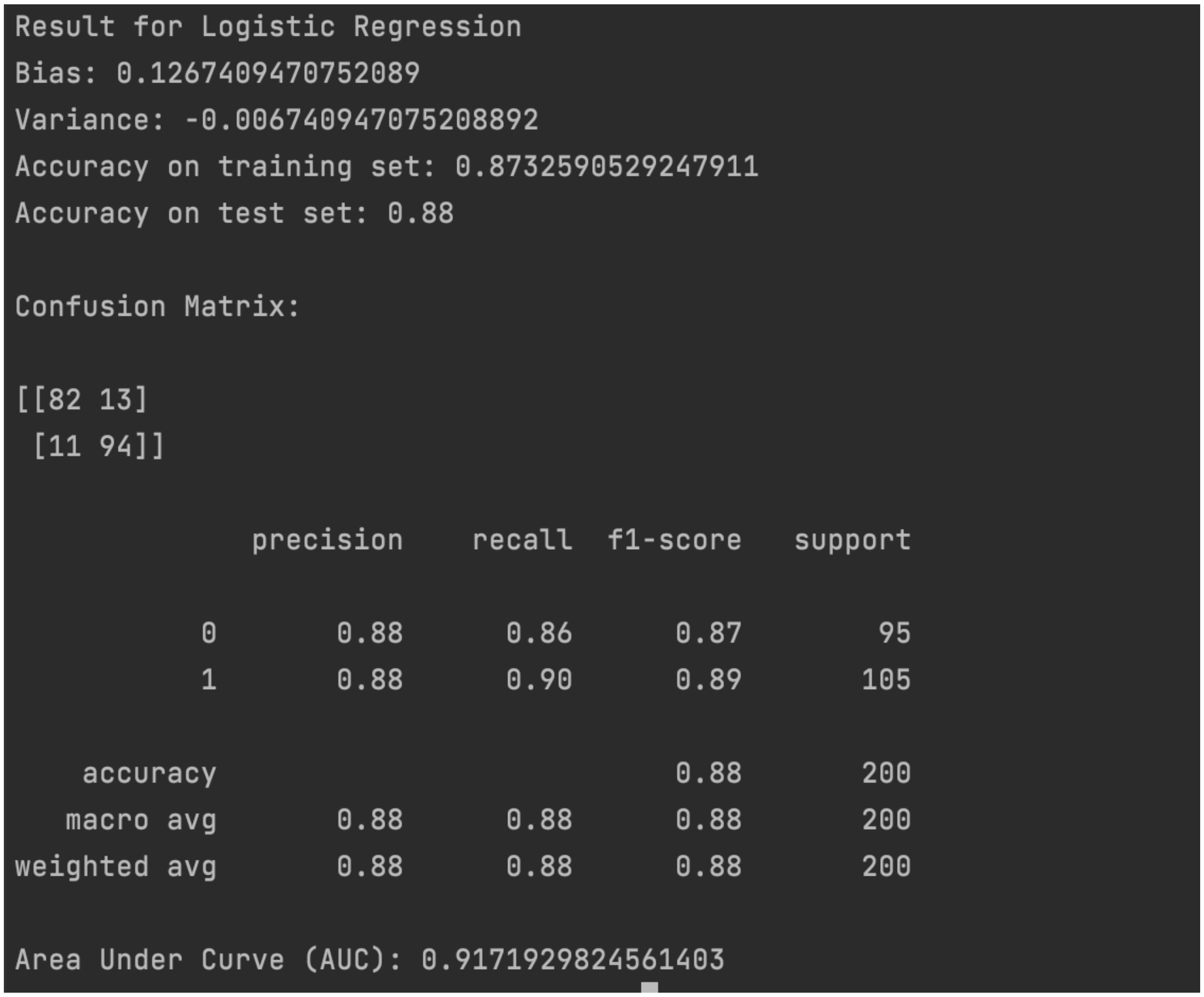


Logistic Regression

We used the sklearn.linear_model.LogisticRegression classifier with `C=0.5, solver='liblinear'` and the other parameters set to their default values.

For the input data we used the preprocessed dataset with numeric data normalized and the two insignificant features (**RestingBP**, **RestingECG**) dropped.

Result of running Logistic Regression on the dataset:



```
Result for Logistic Regression
Bias: 0.1267409470752089
Variance: -0.006740947075208892
Accuracy on training set: 0.8732590529247911
Accuracy on test set: 0.88

Confusion Matrix:

[[82 13]
 [11 94]]

              precision    recall  f1-score   support

           0       0.88      0.86      0.87        95
           1       0.88      0.90      0.89       105

    accuracy                           0.88       200
   macro avg       0.88      0.88      0.88       200
weighted avg       0.88      0.88      0.88       200

Area Under Curve (AUC): 0.9171929824561403
```

Bias and Variance:
According to the bias and variance of the Logistic Regression classifier, the bias was relatively high and variance was relatively low. To improve this, we need a more powerful model, we could:
1. Try to increase order on the polynomial or add terms
2. Add more features by adding the dropped features back.

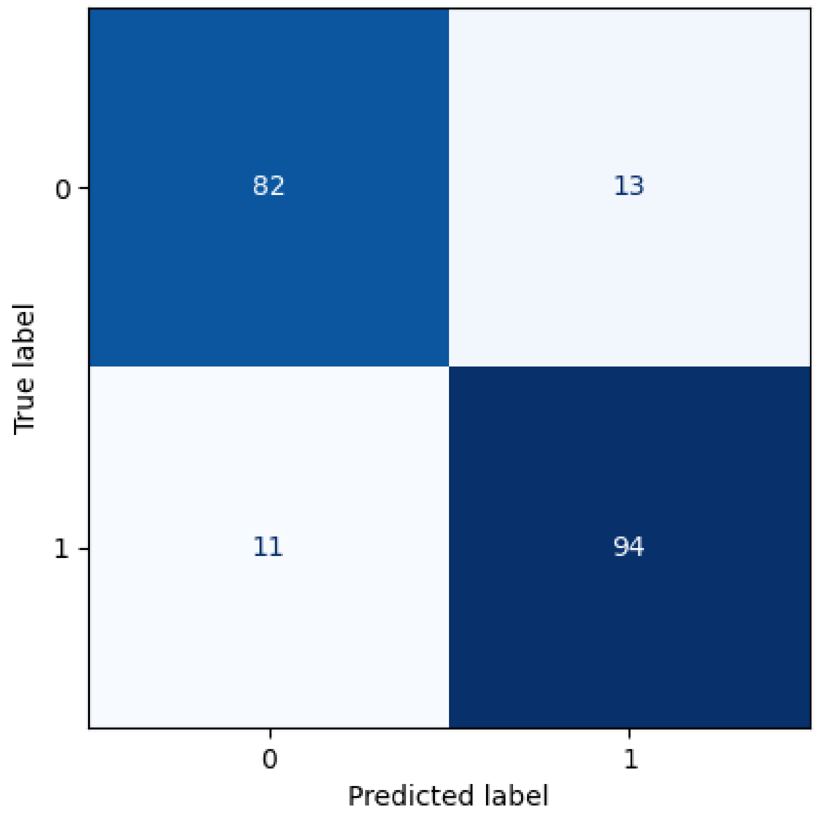
3. Decrease regularization

Accuracy:
Both training and test set had a good accuracy and F1 score which was around 0.88 with **HeatDisease**=1 as the positive label.

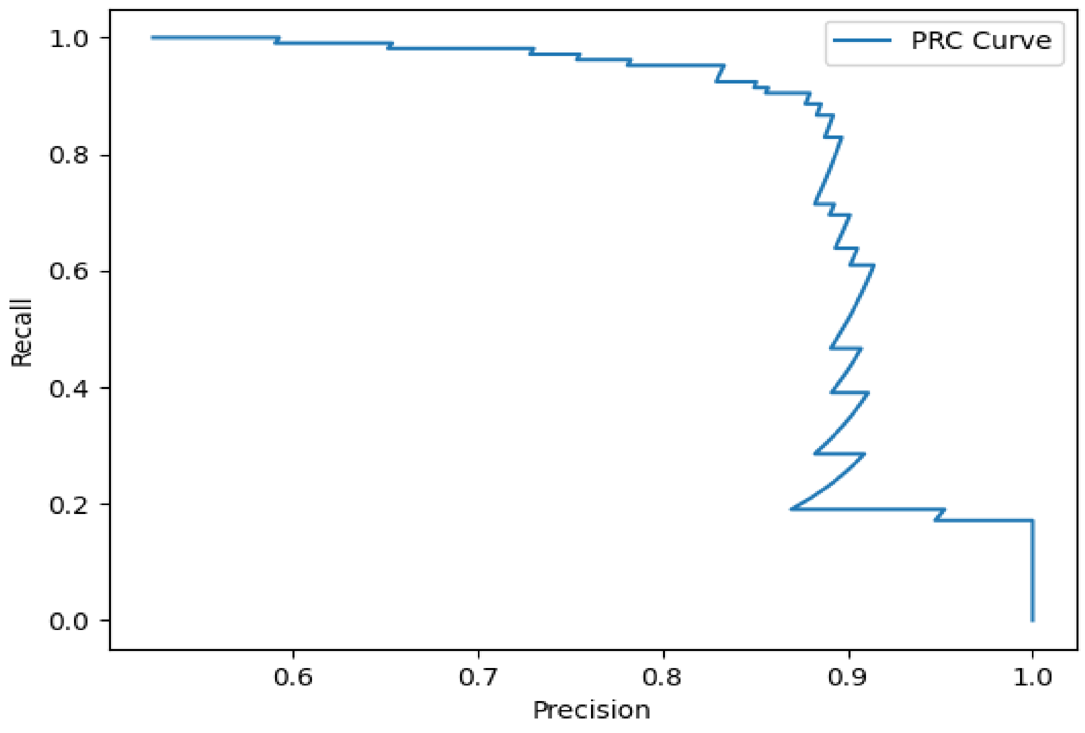
Confusion Matrix:
According to the confusion matrix, we have 82 true positives and 94 true negatives, 13 false positives and 11 false negatives, it is a relatively good ratio. Since this is a medical related dataset, it is better to have higher false positives, because it’s safer for the patients.

AUC:
We have around 0.92 under ROC, which is a relatively good area.

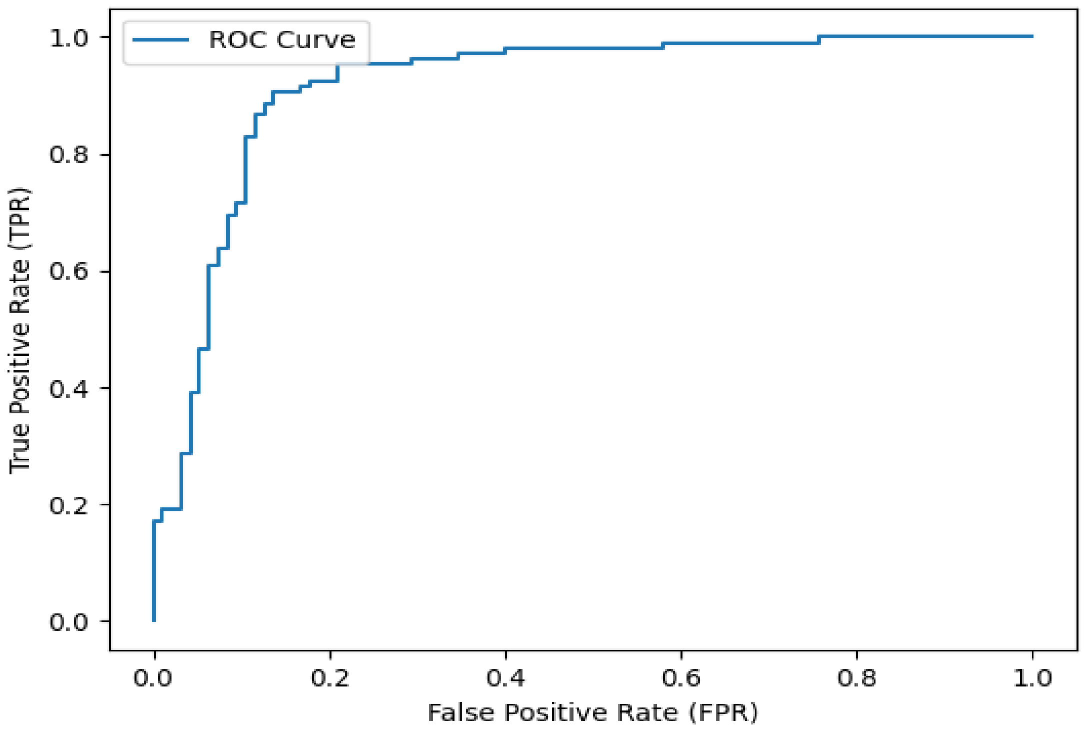
Confusion matrix of running Logistic Regression on the dataset:



PRC of running Logistic Regression on the dataset:



ROC of running Logistic Regression on the dataset:

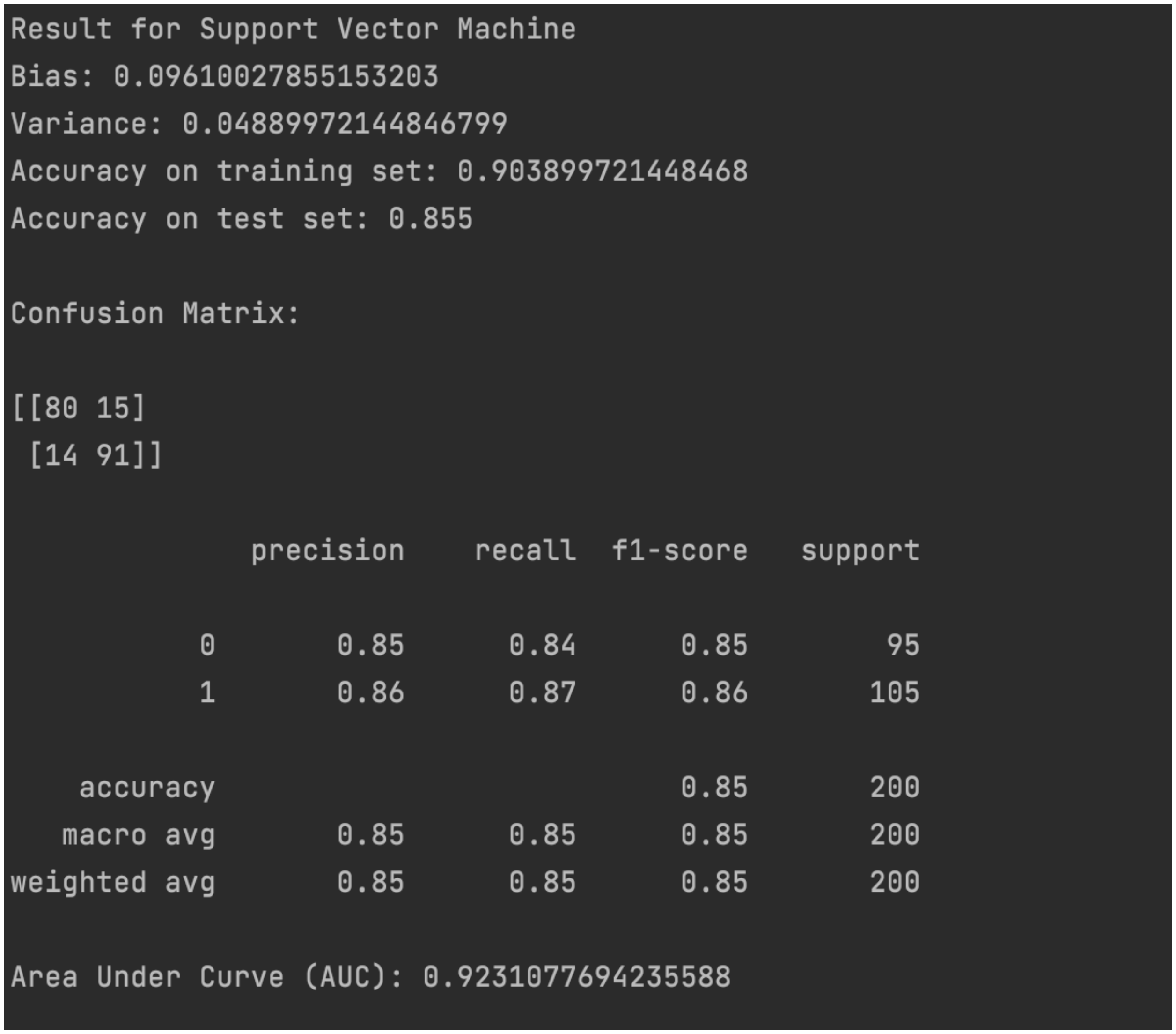


Support Vector Machine

We used the sklearn.svm.SVC classifier with `probability=True` and other metaparameters set to their default values.

For the input data we used the preprocessed dataset with numeric data normalized and the two insignificant features (**RestingBP**, **RestingECG**) dropped.

Result of running SVM on the dataset:



```
Result for Support Vector Machine
Bias: 0.09610027855153203
Variance: 0.04889972144846799
Accuracy on training set: 0.903899721448468
Accuracy on test set: 0.855

Confusion Matrix:

[[80 15]
 [14 91]]

              precision    recall  f1-score   support

           0       0.85      0.84      0.85        95
           1       0.86      0.87      0.86       105

    accuracy                           0.85       200
   macro avg       0.85      0.85      0.85       200
weighted avg       0.85      0.85      0.85       200

Area Under Curve (AUC): 0.9231077694235588
```

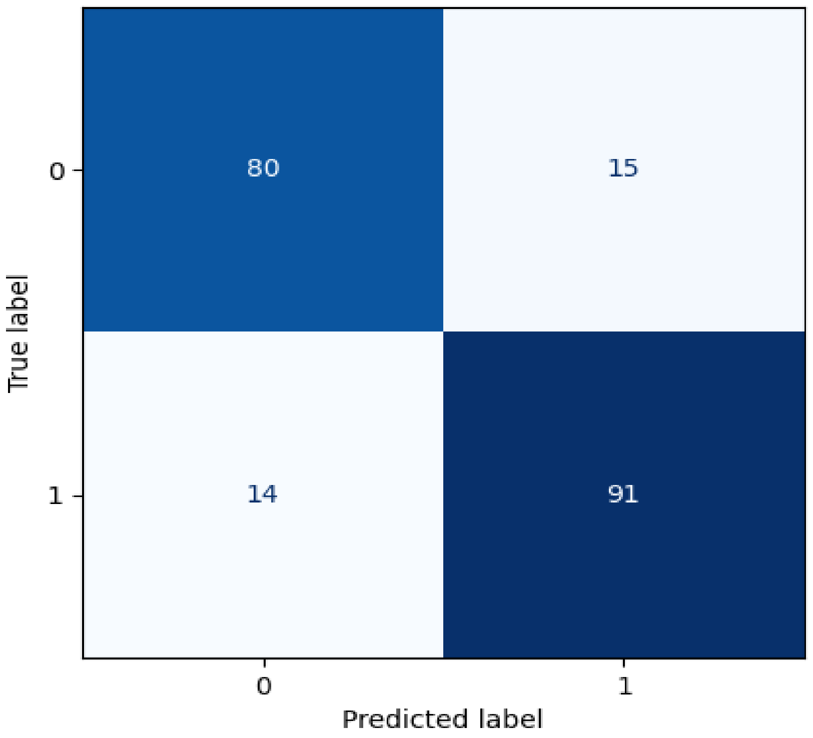
Bias and Variance:
According to the bias and variance of the Support Vector Machine classifier, the bias was relatively high and variance was relatively low. To improve this, we need a more powerful model. We could:
1. Increase the value of the C parameter: Increasing its value allows the model to better fit the training data, potentially reducing bias.
2. Use a more complex kernel: Using a more complex kernel function, such as a polynomial or Gaussian kernel, can increase the model's capacity to fit the data, potentially reducing bias.
3. Add more features to the data (add the dropped features back to the dataset).
4. Decrease regularization.

Accuracy:
Both training and test set had a good accuracy and G1 score which is around 0.86 with **HeartDisease**=1 as the positive label.

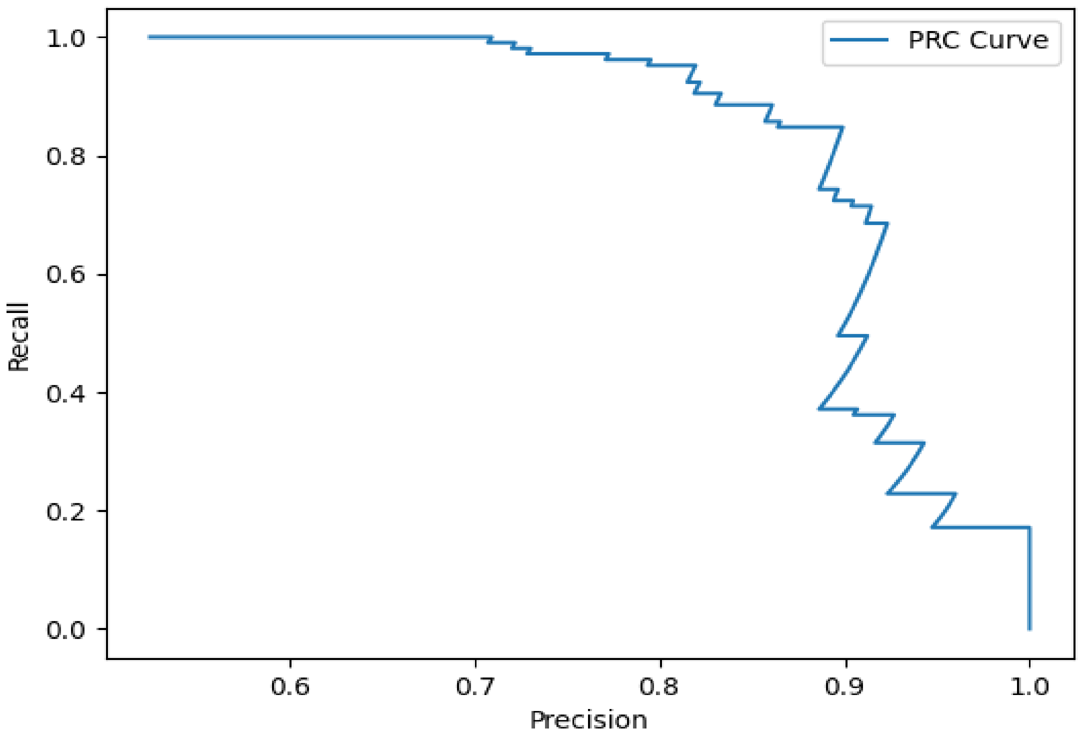
Confusion Matrix:
According to the confusion matrix, we had 80 true positives and 91 true negatives, 15 false positives and 14 false negatives. This is a relatively good ratio. Since this was a medical related dataset, it is better to have higher false positives, because it’s safer for the patients.

AUC:
We have around 0.92 under ROC, which is a relatively good area.

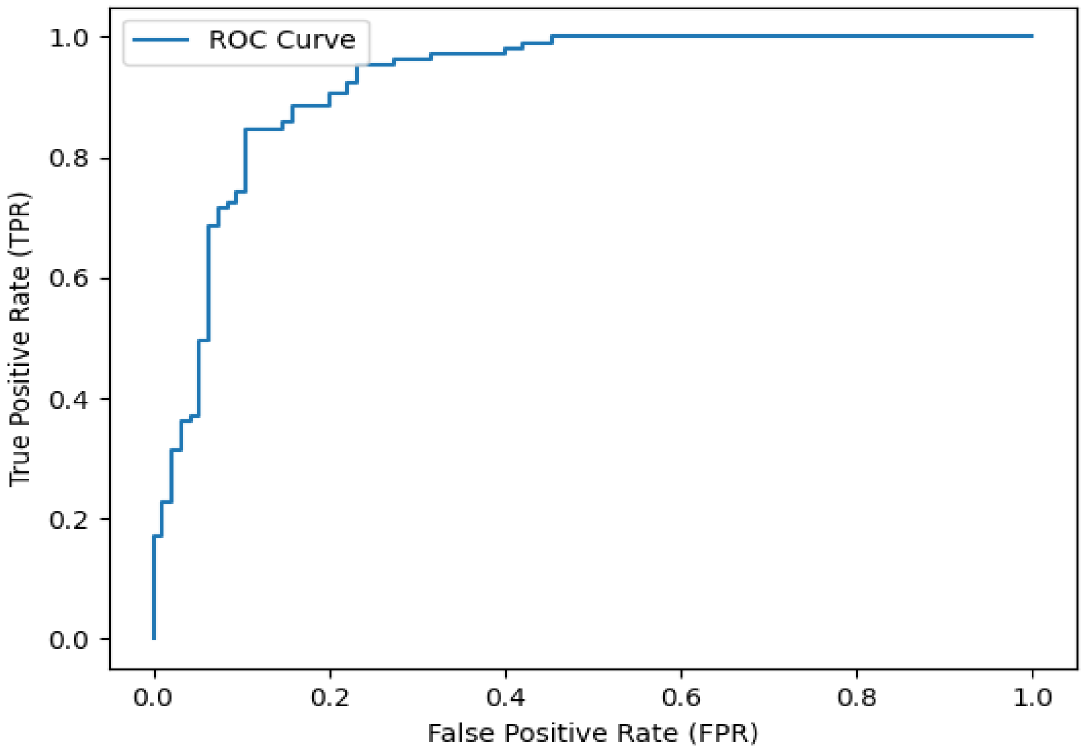
Confusion matrix of running SVM on the dataset:



PRC of running SVM on the dataset:



ROC of running SVM on the dataset:



Naive Bayes

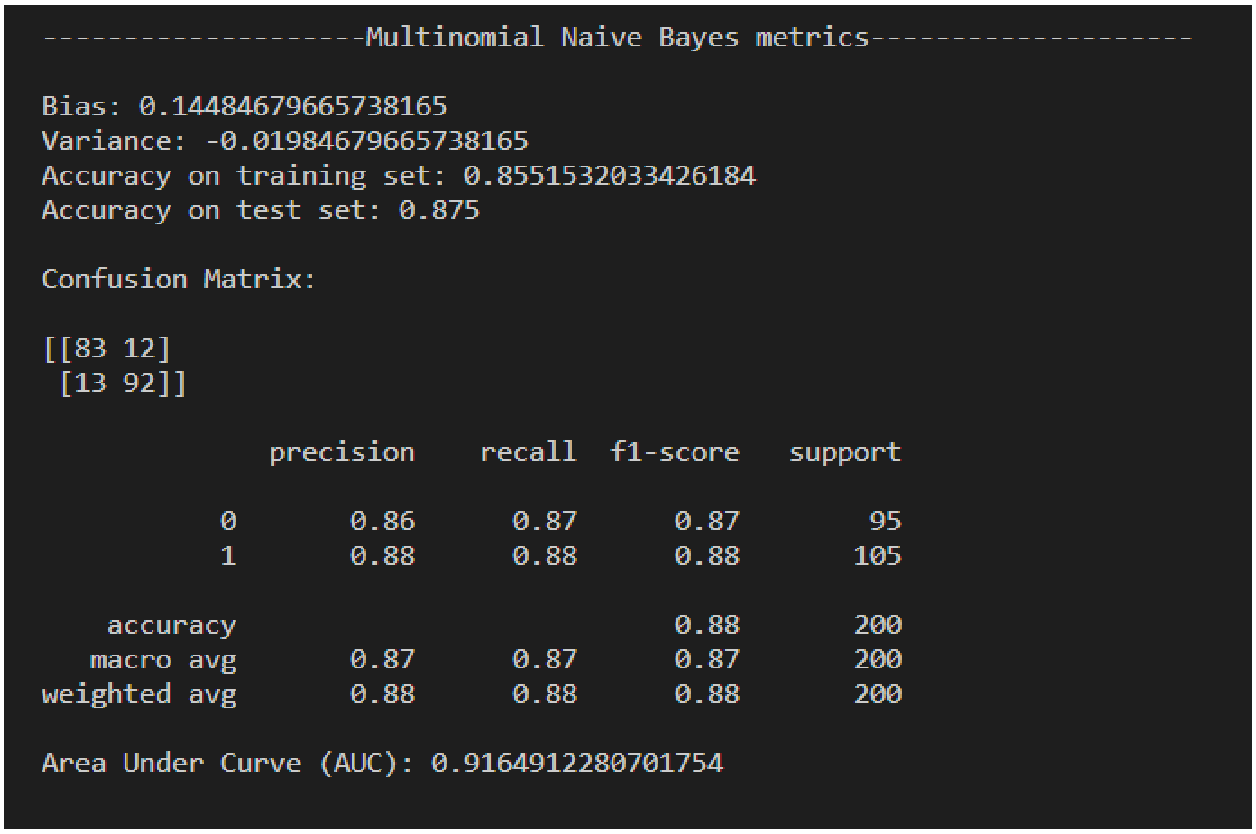
The raw input data for our classification study was heterogeneous in terms of data type: there was a mixture of categorical and numerical input features. During preprocessing we converted the categorical features into dummy variables which allowed us to treat them as discrete numerical features. However, since the other numerical features were normalized by converting their units into standard deviations, the data set's numerical features were split between discrete and continuous. This complicated the choice of naive bayes models because Multinomial is designed for discrete data, Bernoulli is designed for binary data and Gaussian is designed for continuous data. Given this complication, we decided to train each of these three naive bayes models to examine which had the best performance.

Multinomial Naive Bayes (MNB)

MNB is designed to use discrete numerical features. As such, before training or testing the MNB classifier, an additional preprocessing step was needed to ensure that all numeric values of the data set were greater than 0. This was necessary because the continuous numeric input features (i.e. **Cholesterol**, **Age**, **MaxHR**, and **Oldpeak**) had some negative values. Therefore, we replaced those negative values with 0. This preprocessing step was unique to the input data for the Multinomial Naive Bayes (MNB) classifier meaning that the input data to other models was not regularized in this way.

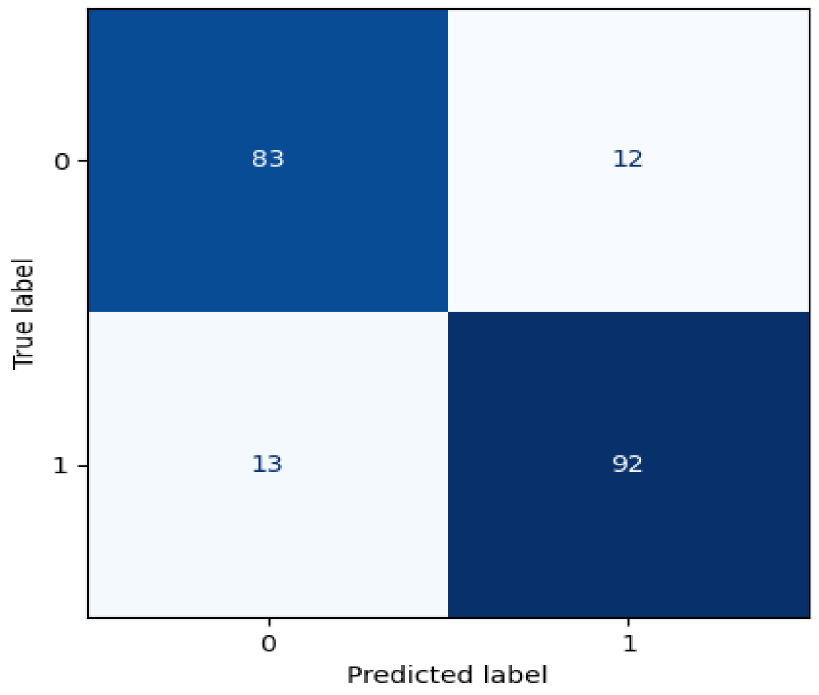
We used the sklearn.naive_bayes.MultinomialNB classifier with the metaparameters set to their default values.

Multinomial Naive Bayes metrics:



```
--------------------Multinomial Naive Bayes metrics--------------------

Bias: 0.14484679665738165
Variance: -0.01984679665738165
Accuracy on training set: 0.8551532033426184
Accuracy on test set: 0.875

Confusion Matrix:

[[83 12]
 [13 92]]

              precision    recall  f1-score   support

           0       0.86      0.87      0.87        95
           1       0.88      0.88      0.88       105

    accuracy                           0.88       200
   macro avg       0.87      0.87      0.87       200
weighted avg       0.88      0.88      0.88       200

Area Under Curve (AUC): 0.9164912280701754
```

Confusion matrix for Multinomial Naive Bayes classification of test data:



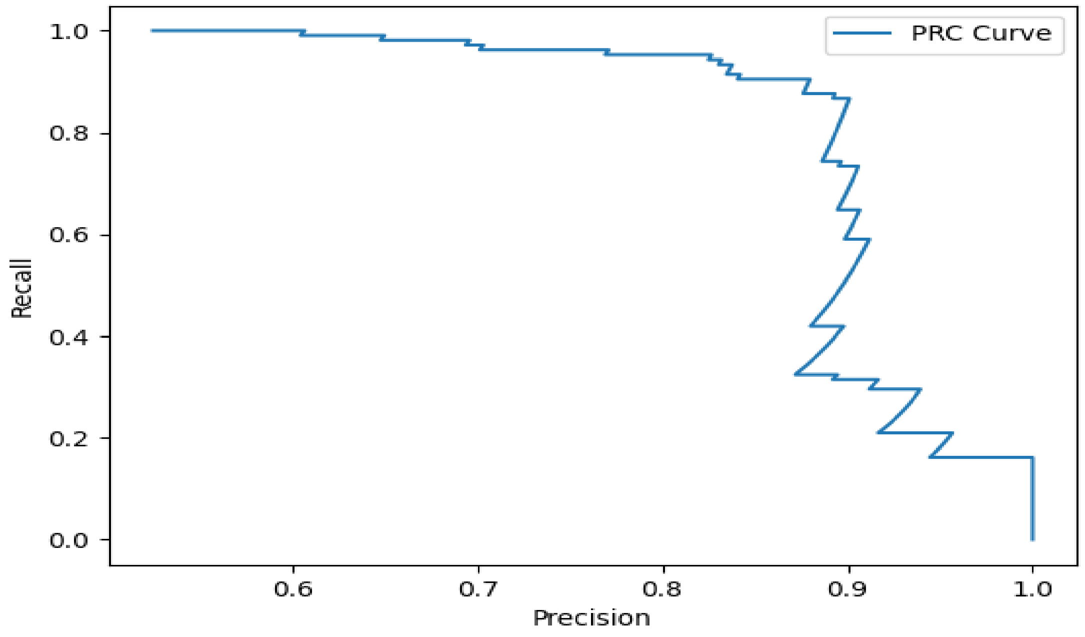
From the above two images, we see that the MNB model had a bias of about 0.143 and a variance of about -0.02. It had an accuracy of about 0.86 on the training set and an accuracy of about 0.88 on the test set.

These values show that the error rate on the test set is lower than the error rate on the training set meaning that the model is training poorly but generalizing well. This is likely due to the fact that the MNB model is designed to work with discrete data, but our input data was a mixture of continuous and discrete data.

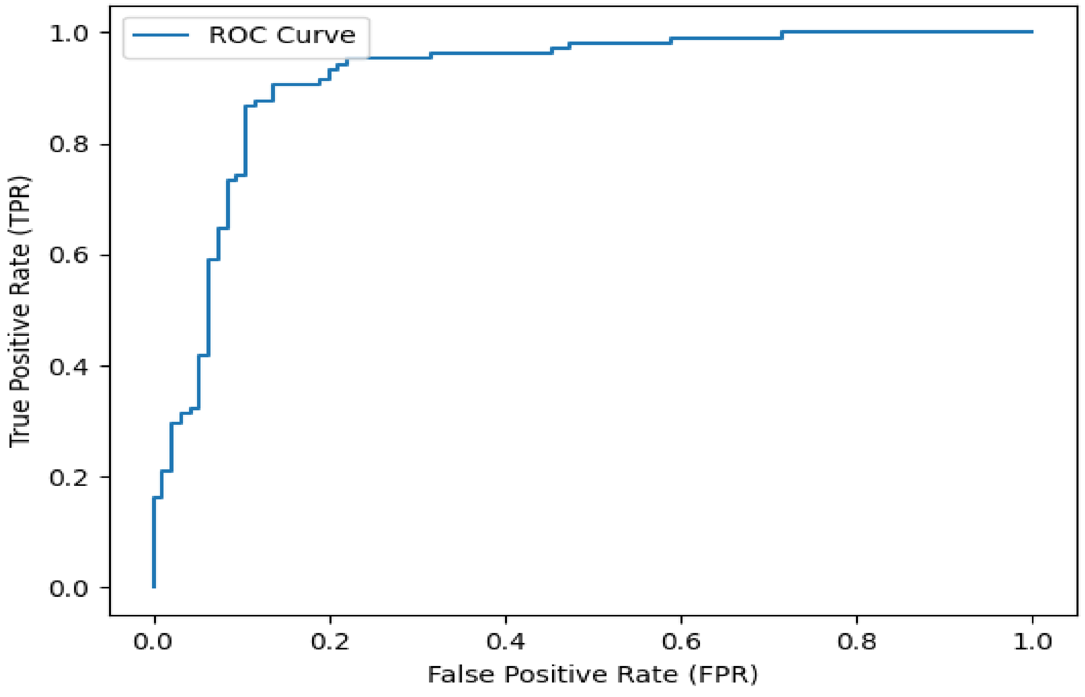
In order to improve the model we would need to use a more complex probability distribution.

Despite the poor training, the model's accuracy on both the training and test sets was above 80%. Moreover, the confusion matrix shows us that the type I errors and type II errors are relatively balanced: 12 and 13 respectively. However, these statistics could be due to the relative imbalance of the dependent variable.

PRC for Multinomial Naive Bayes classifier on data set:



ROC for Multinomial Naive Bayes classifier on data set:



The above are images of the MNB classifier’s precision recall curve and receiver operating characteristics curve. The ROC curve has an AUC of 0.92 and the F1 score with **HeartDisease**=1 as the positive label is about 0.88.

Viewing the ROC curve’s plot shows us that to achieve 100% recall, the FPR would have to be greater than 0.7. To achieve at least 95% recall, the FPR would have to be greater than about 0.2. To achieve at least 90% recall, the FPR would have to be greater than about 0.1. These suggest that the optimal operating point will have an FPR greater than 0.2.

When we look at the PRC curve we see that to obtain a recall of about 95%, our precision will be less than about 0.8.

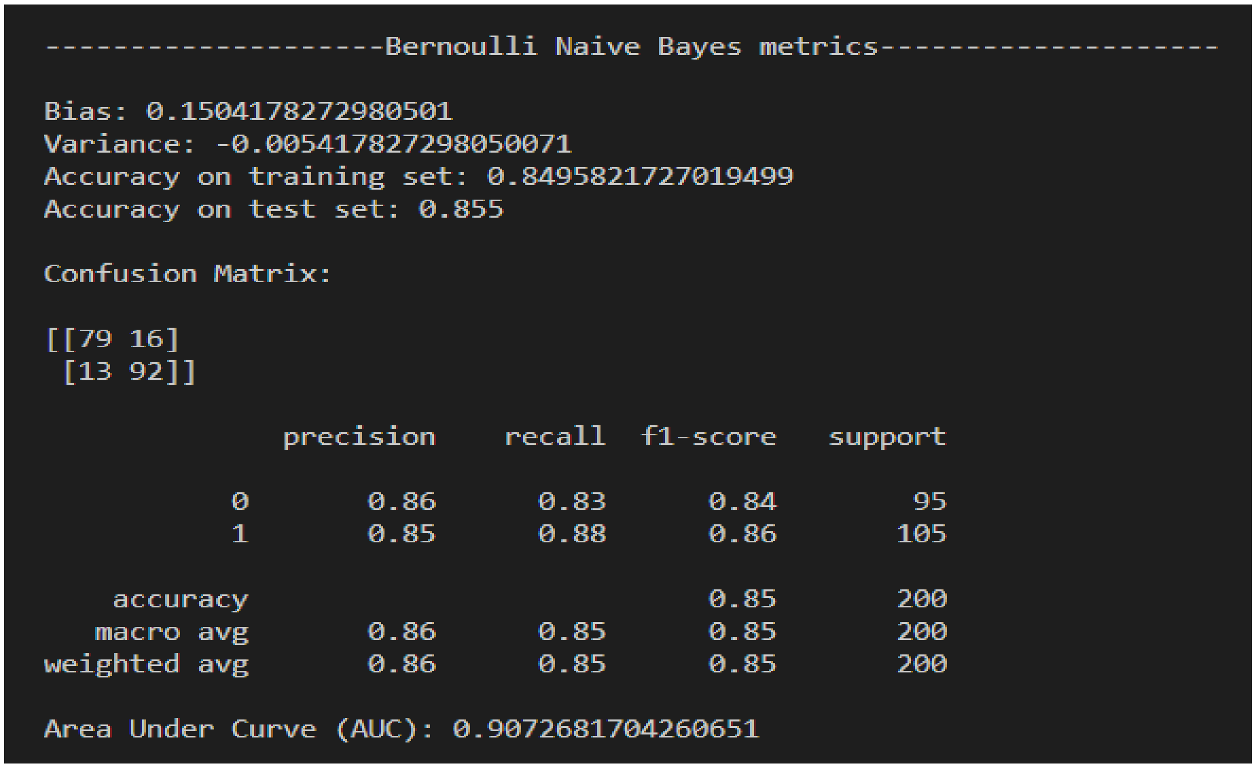
Bernoulli Naive Bayes (BNB)

BNB is designed to use binary features. Since our input data contains continuous features, the BNB model will have some of the same issues that the MNB model had. However, the negative values for the input data did not need to be removed for BNB.

For the input data we used the preprocessed dataset with numeric data normalized and the two insignificant features (**RestingBP**, **RestingECG**) dropped.

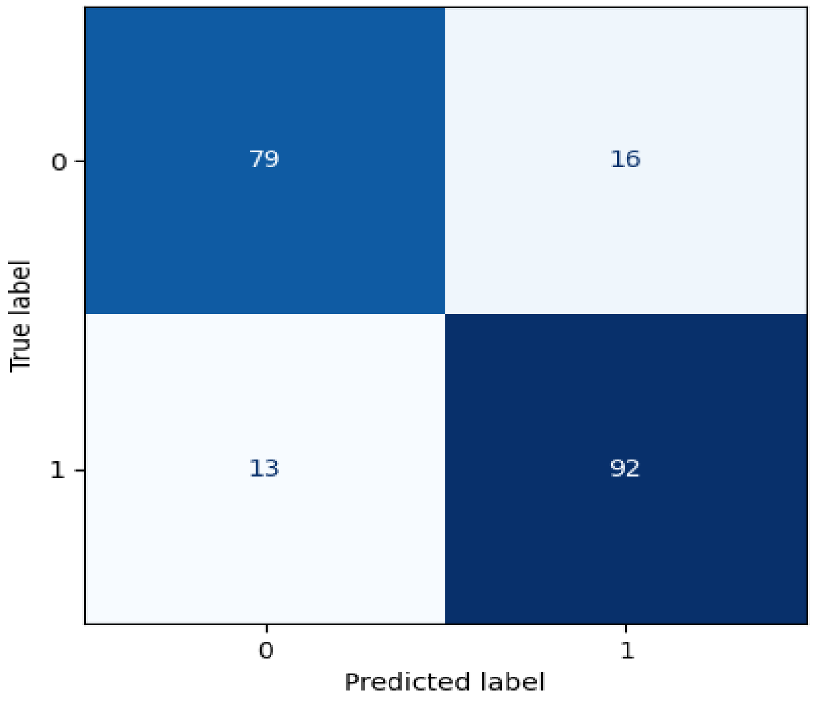
We used the sklearn.naive_bayes.BernoulliNB classifier with the metaparameters set to their default values.

Bernoulli Naive Bayes metrics:



```
--------------------Bernoulli Naive Bayes metrics--------------------

Bias: 0.15041782729805014
Variance: -0.005417827298050071
Accuracy on training set: 0.8495821727019499
Accuracy on test set: 0.855

Confusion Matrix:

[[79 16]
 [13 92]]

              precision    recall  f1-score   support

           0       0.86      0.83      0.84        95
           1       0.85      0.88      0.86       105

    accuracy                           0.85       200
   macro avg       0.86      0.85      0.85       200
weighted avg       0.86      0.85      0.85       200

Area Under Curve (AUC): 0.9072681704260651
```

Confusion matrix for Bernoulli Naive Bayes classification of test data:

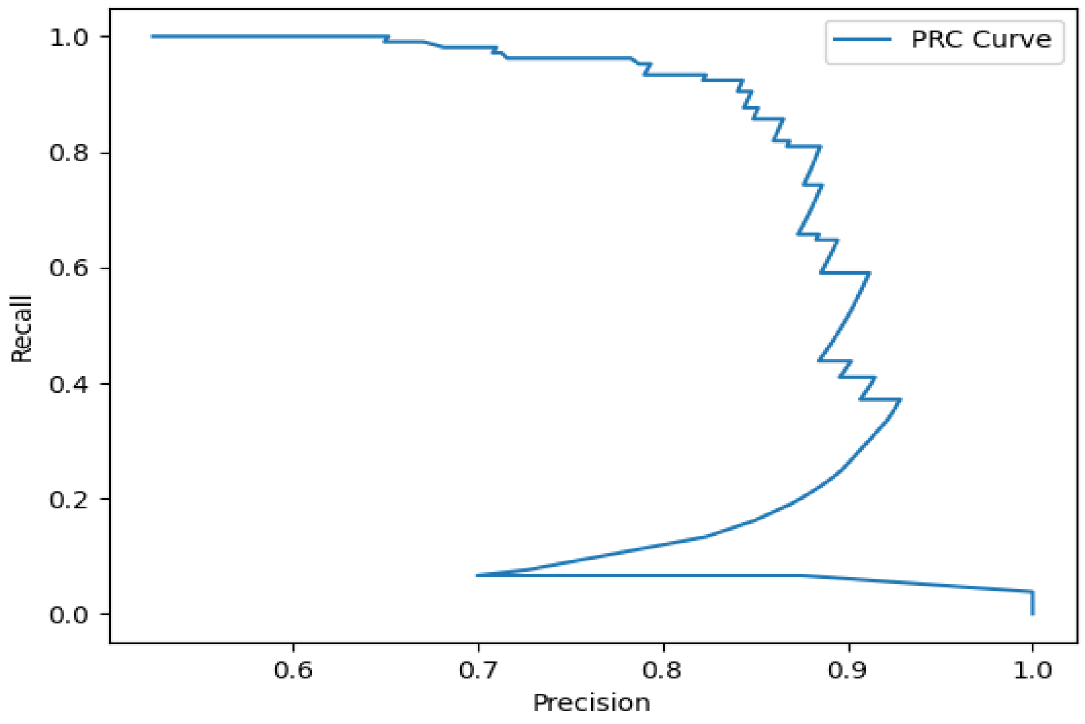


From the above two images we see that the BNB model had a bias of about 0.15 and a variance of about -0.005. It had an accuracy of about 0.85 on the training set and an accuracy of about 0.88 on the test set.

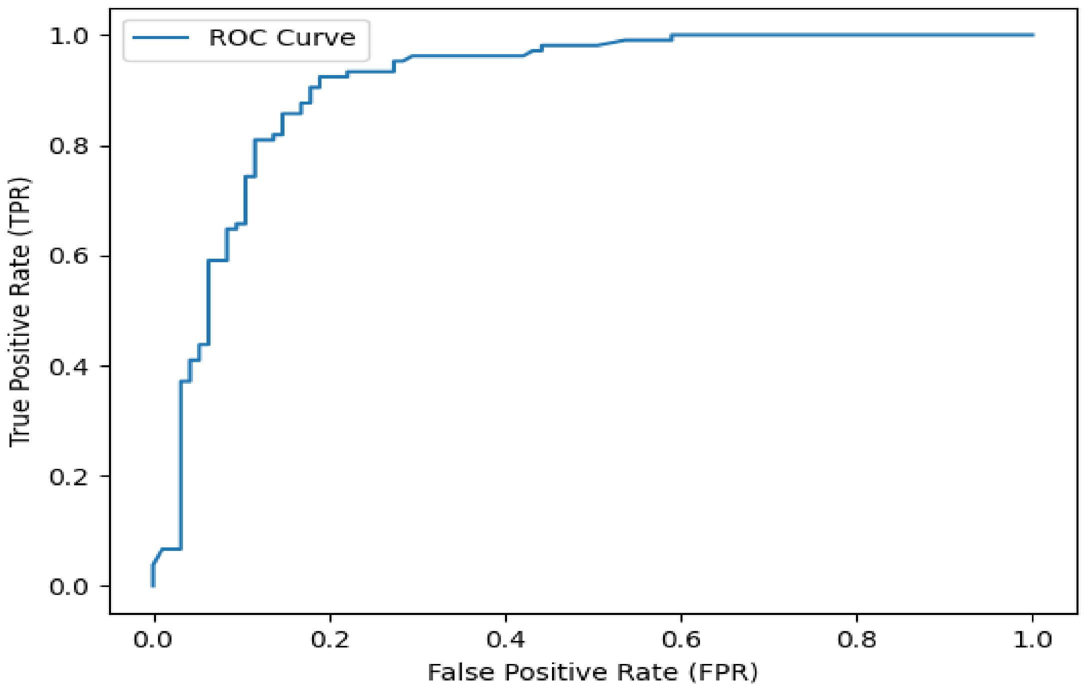
These values show that the error rate on the test set is lower than the error rate on the training set meaning that the model is training poorly but generalizing well. This is likely due to the fact that the BNB is designed to work with binary data, but our input data contained continuous features.

In order to reduce the error rate on the training set, we would need to use a more complex probability distribution or remove the non-binary features.

Despite the poor training, the model's accuracy on both the training and test sets was above 80%. Moreover, the confusion matrix shows us that the type I errors and type II errors are relatively balanced: 16 and 13 respectively. However, these statistics could be due to the relative imbalance of the dependent variable.
PRC for Bernoulli Naive Bayes classifier on data set:



ROC for Multinomial Naive Bayes classifier on data set:



The above are images of the BNB classifier's precision recall curve and receiver operating characteristics curve. The ROC curve has an AUC of about 91 and the F1 score with **HeatDisease**=1 as the positive label is about 86.

Looking at the ROC curve's plot shows us that to achieve 100% recall, the FPR would have to be greater than 0.62. To achieve at least 95% recall, the FPR would have to be greater than 0.3. To achieve at least 90% recall, the FPR would have to be greater than 0.2. These suggest that the optimal operating point will have an FPR greater than 0.3.

Looking at the PRC curve, we see that to obtain a recall of about 95%, our precision will be less than 0.8.

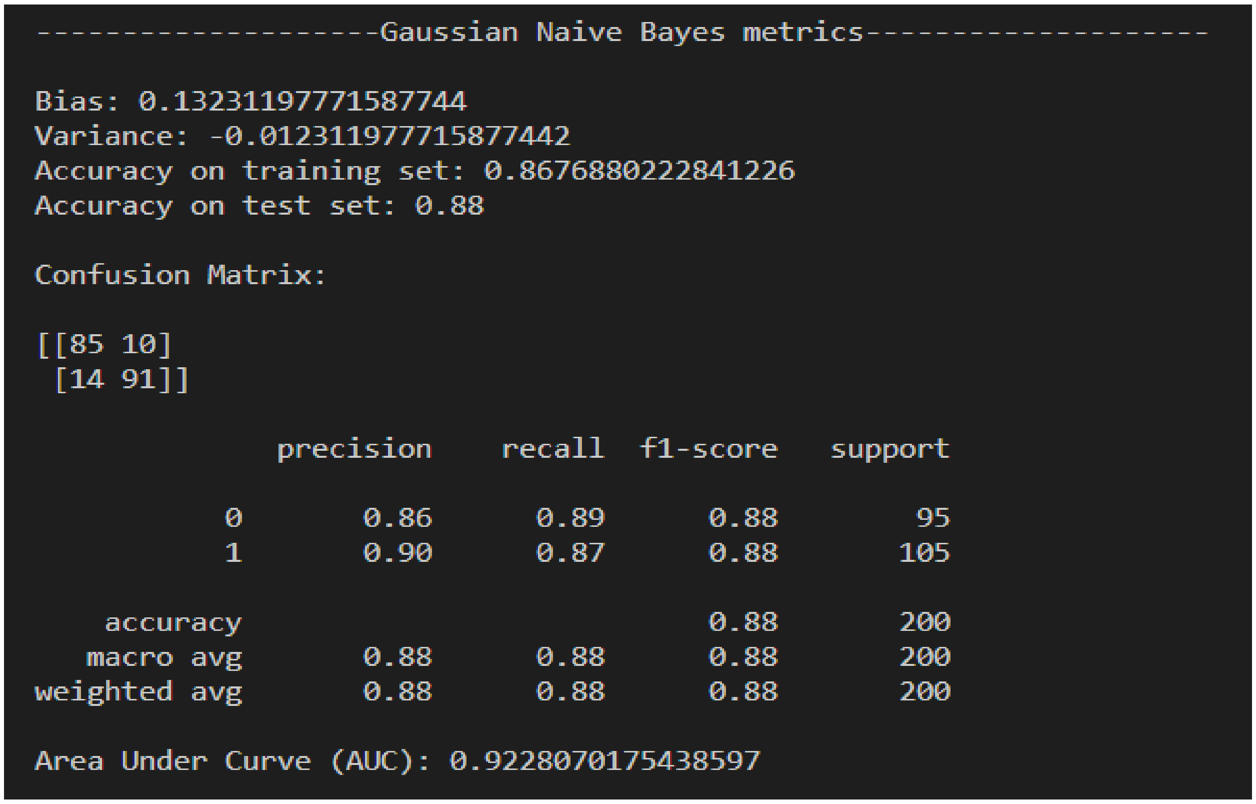
Gaussian Naive Bayes (GNB)

GNB is designed to use continuous features. Since our data contains binary features, the GNB model will have similar issues to those the MNB and BNB models had. However, since it expects continuous features, the negative values that our continuous features have are acceptable.

For the input data we used the preprocessed dataset with numeric data normalized and the two insignificant features (**RestingBP**, **RestingECG**) dropped.

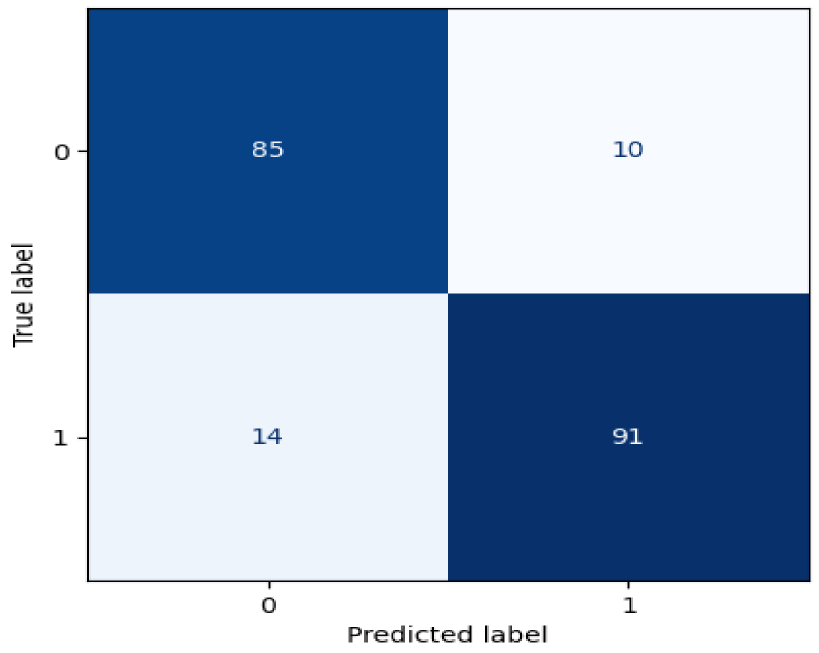
We used the sklearn.naive_bayes.GaussianNB classifier with the metaparameters set to their default values.

Gaussian Naive Bayes metrics:



```
--------------------Gaussian Naive Bayes metrics--------------------

Bias: 0.13231197771587744
Variance: -0.012311977715877442
Accuracy on training set: 0.8676880222841226
Accuracy on test set: 0.88

Confusion Matrix:

[[85 10]
 [14 91]]

              precision    recall  f1-score   support

           0       0.86      0.89      0.88        95
           1       0.90      0.87      0.88       105

    accuracy                           0.88       200
   macro avg       0.88      0.88      0.88       200
weighted avg       0.88      0.88      0.88       200

Area Under Curve (AUC): 0.9228070175438597
```

Confusion matrix for Gaussian Naive Bayes classification of test data:



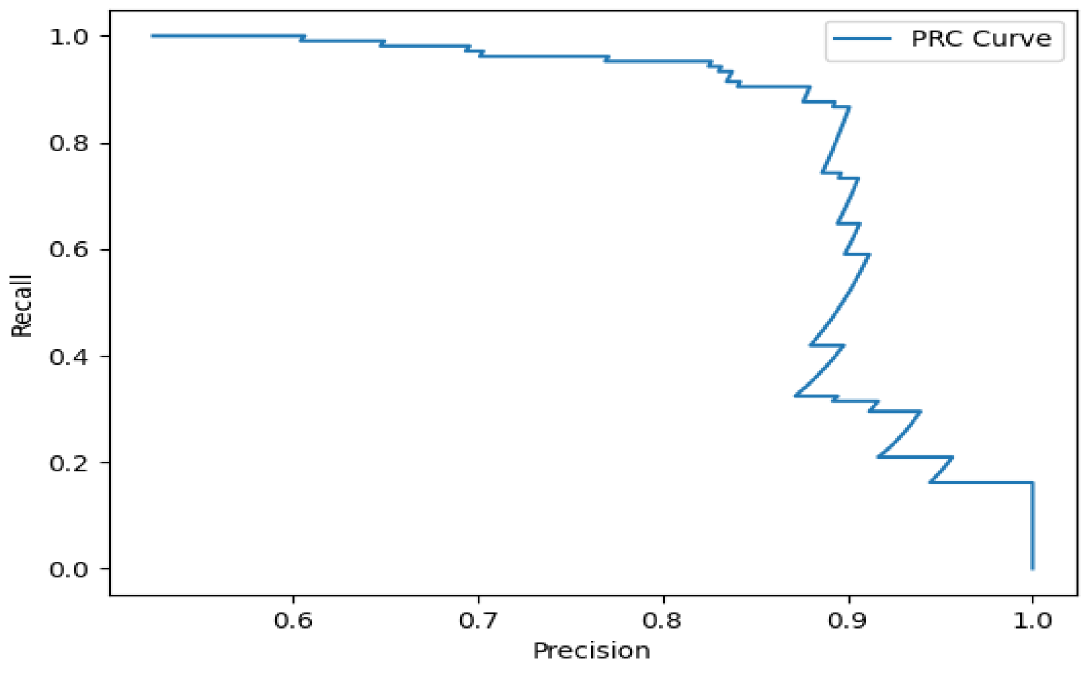
The above two images show that GNB had a bias of about 0.132 and a variance of about -0.012. It had an accuracy of about 0.87 on the training set and an accuracy of about 0.88 on the test set.

These values show that the error rate on the test set is lower than the error rate on the training set meaning that the model is training poorly but generalizing well. This is likely due to the fact that the GNB is designed to work with continuous data, but our input contained binary features.

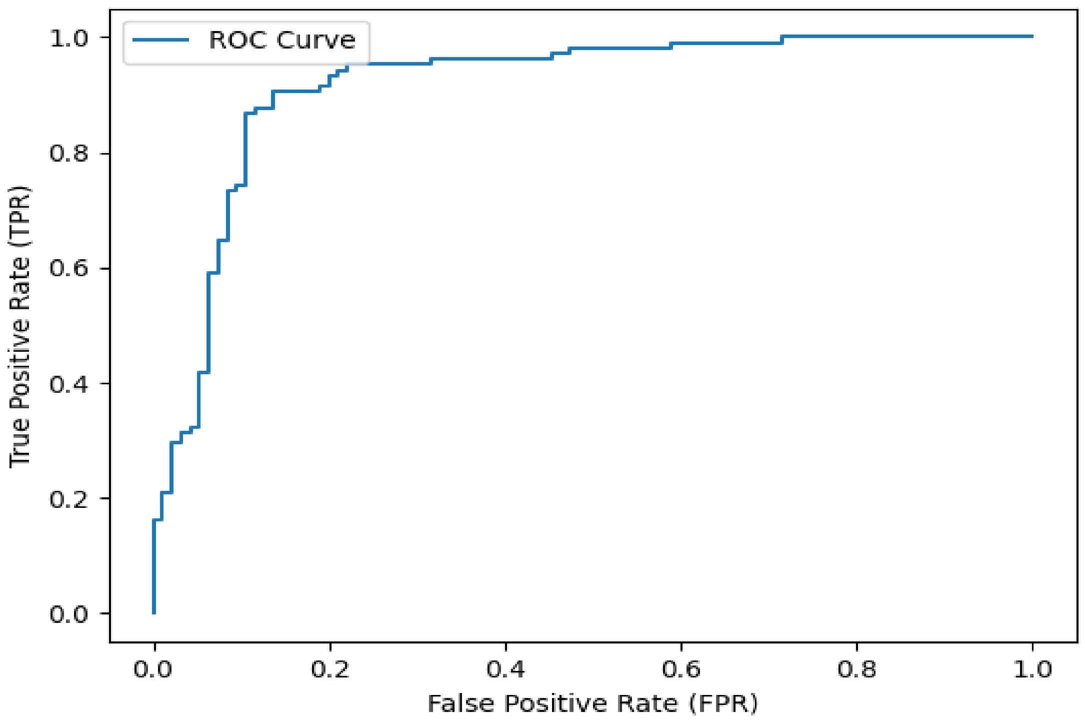
In order to reduce bias, we need to reduce the error rate on the training set. This can be done by using a more complex probability distribution (e.g. a mixture of Gaussian distributions) or by removing the non-continuous features.

Despite the poor training, the model's accuracy on both the training and test sets was above 85%. Moreover, the confusion matrix shows us that the type I errors and type II errors are relatively balanced: 10 and 14 respectively. However, these statistics could be due to the relative imbalance of the dependent variable.

PRC for Gaussian Naive Bayes classifier on data set:



ROC for Gaussian Naive Bayes classifier on data set:



The above are images of the GNB classifier’s precision recall curve and receiver operating characteristics curve. The ROC curve has an AUC of about 92 and the F1 score with **HeartDisease**=1 as the positive label is about 88.

Looking at the ROC curve’s plot shows us that to achieve 100% recall, the FPR would have to be greater than 0.7. To achieve at least 95% recall, the FPR would have to be greater than 0.2. To achieve at least 90% recall, the FPR would have to be greater than 0.1. These suggest that the optimal operating point will have an FPR greater than 0.2.

Looking at the PRC curve, we see that to obtain a recall of about 95%, our precision will be less than 0.7.

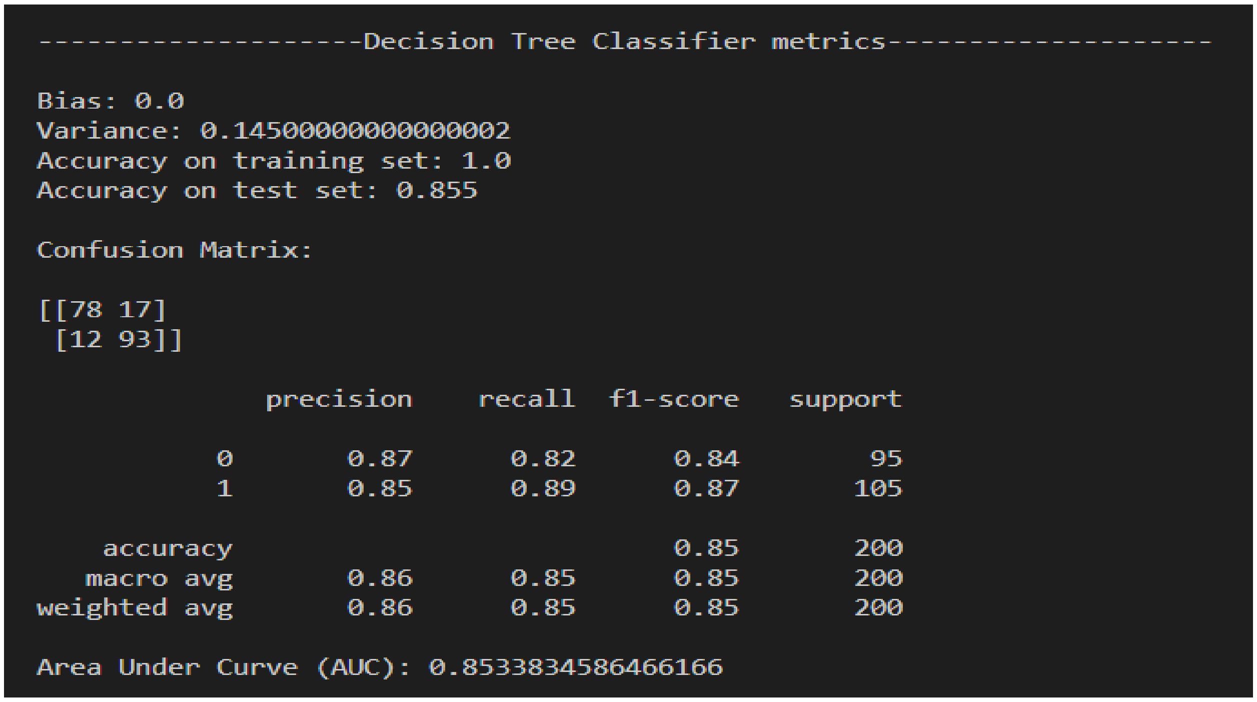
Decision Tree

The decision tree is designed to handle a mixture of categorical and numerical data, so the mixture of binary and continuous input data should not negatively affect it.

For the input data we used the preprocessed dataset with numeric data normalized and the two insignificant features (**RestingBP**, **RestingECG**) dropped.

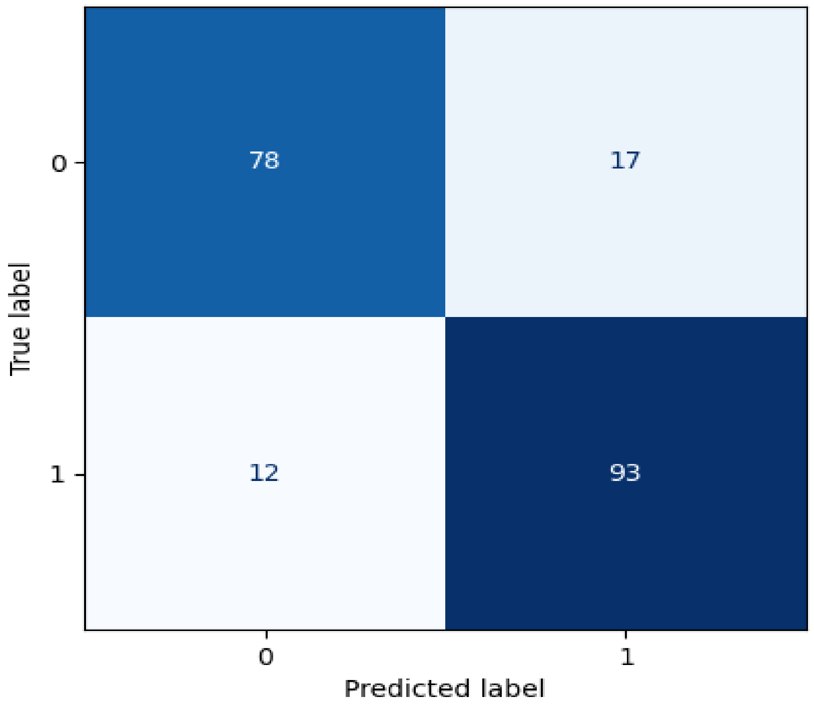
We used the sklearn.tree.DecisionTreeClassifier classifier with the metaparameters set to their default values.

Decision Tree classifier metrics:



```
--------------------Decision Tree Classifier metrics--------------------

Bias: 0.0
Variance: 0.14500000000000002
Accuracy on training set: 1.0
Accuracy on test set: 0.855

Confusion Matrix:

[[78 17]
 [12 93]]

              precision    recall  f1-score   support

           0       0.87      0.82      0.84        95
           1       0.85      0.89      0.87       105

    accuracy                           0.85       200
   macro avg       0.86      0.85      0.85       200
weighted avg       0.86      0.85      0.85       200

Area Under Curve (AUC): 0.8533834586466166
```

Confusion matrix for Decision Tree classification of test data:

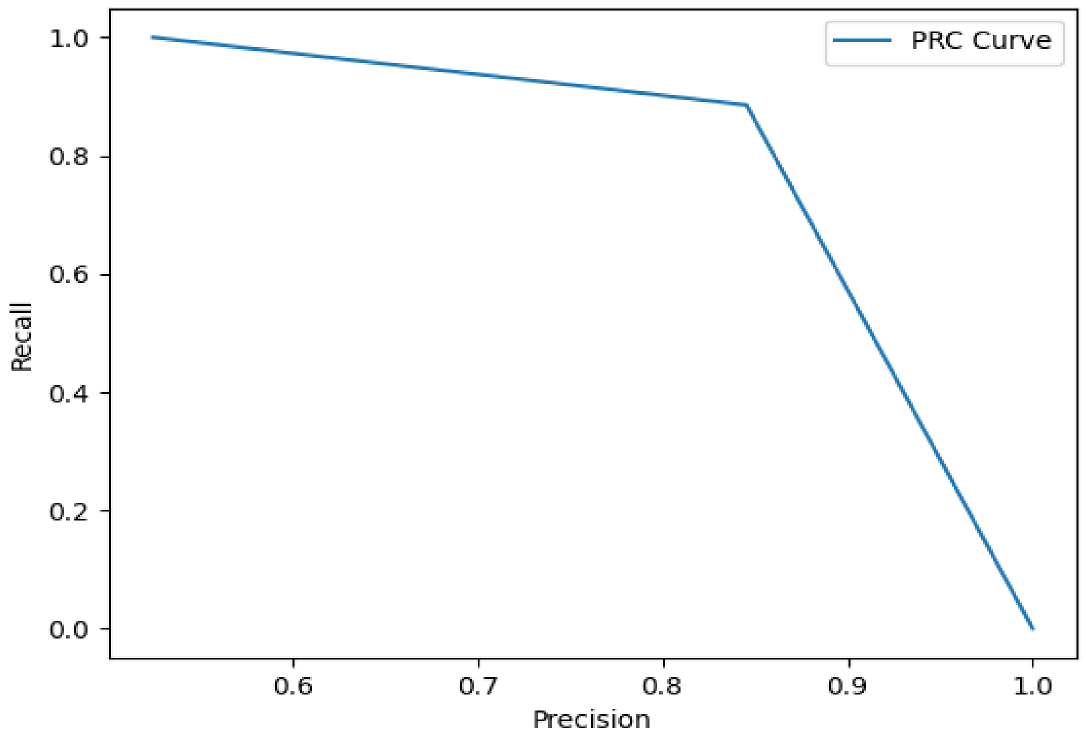


From the above images we see that the decision tree model had a bias of 0 and a variance of about 0.145. It had perfect accuracy on the training set (i.e. 1.0) and an accuracy of about 0.86 on the test set.

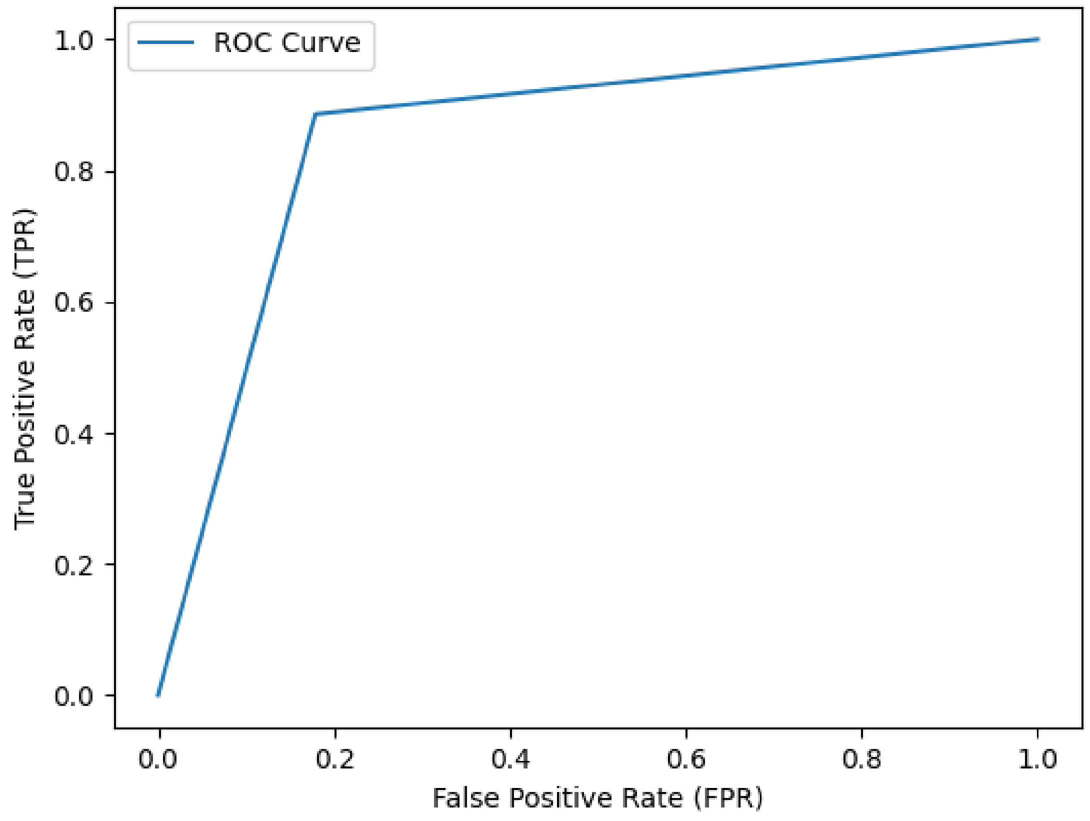
The error rate of 0 on the training set is expected for this simple decision tree model because there was no max depth specified so the tree is as complex as was required to fit the training data perfectly. In other words, the model is overfitting the training data. This also explains why the variance is much higher: because the accuracy on the test set is much lower meaning the error rate on the test set is much higher.

In order to reduce the error rate on the test set, we would need to specify a max depth metaparameter. Altering this metaparameter will decrease the complexity of the model and prevent it from overfitting the training data. This will increase the bias and decrease the variance.
Despite the overfitting, the model's accuracy on the test set was still above 85%. However, the confusion matrix shows us that type I errors are slightly less frequent than type II errors: 12 and 17 respectively.

PRC for Decision Tree classifier on data set:



ROC for Decision Tree classifier on data set:



The decision tree's ROC curve has an AUC of 0.85 and the F1 score with **HeartDisease**=1 as the positive label is about 0.87.

Looking at the ROC curve's plot shows us that to achieve 100% recall, the FPR would also have to be 1. To achieve at least 95% recall, the FPR would have to be greater than about 0.8. To achieve at least 90% recall, the FPR would have to be greater than about 0.4. These suggest that the optimal operating point will have an FPR greater than 0.8.

Looking at the PRC curve, we see that to obtain a recall of about 95%, our precision will be less than about 0.6.

Summary on the 5 classifiers:

Since our **HeartDisease** data was imbalanced, with there being more **HeartDisease**=1 than **HeartDisease**=0 values, we gave more weight to AUC scores than accuracy scores.

Overall, we see that the Support Vector Machine (SVM) and Gaussian Naive Bayes (GNB) models had the best overall performance. The GNB classifier had slightly higher accuracy (0.88 vs. 0.86). However, they had an equivalent AUC value implying that they both be improved a similar amount. Therefore, even though Logistic Regression had a higher accuracy than SVM we decided that SVM was a better model since it had slightly better room for improvement than Logistic Regression.

Task 4: Iteration

Support Vector Machine

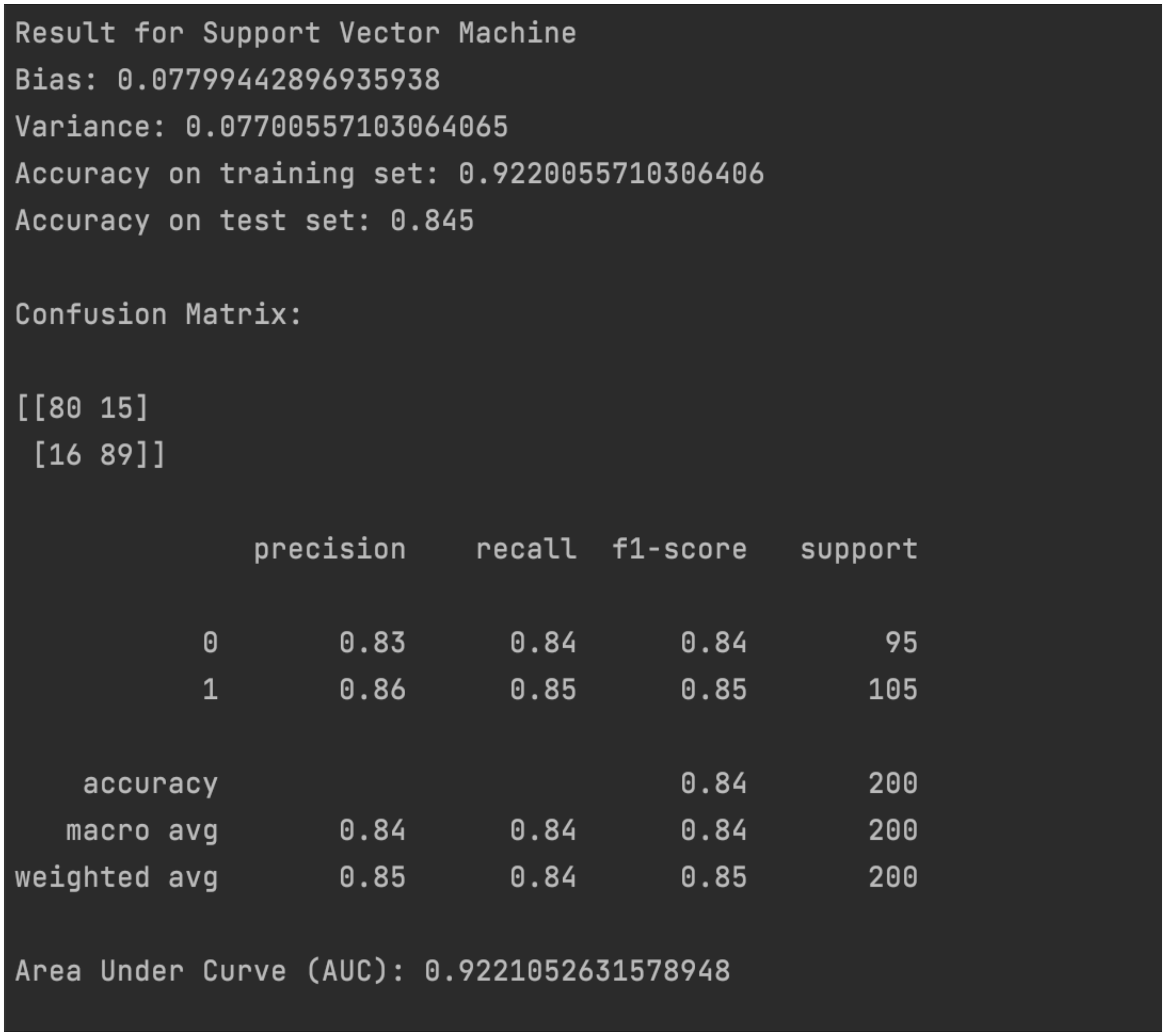
As we discuss earlier, in order to improve the performance of the current SVM model, we could try the following:
1. Increase the value of the C parameter: Increasing its value allows the model to better fit the training data, potentially reducing bias.
2. Use a more complex kernel: Using a more complex kernel function, such as a polynomial or Gaussian kernel, can increase the model's capacity to fit the data, potentially reducing bias.
3. Add more features to the data (add the dropped features back to the dataset).
4. Decrease regularization.

In the following iterations we will try 3 different modifications.

Iteration 1: increase value of the C parameter

The C parameter tells the SVM optimization how much you want to avoid misclassifying each training example. For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassified more points. For very tiny values of C, you should get misclassified examples, often even if your training data is linearly separable.
For this iteration, we used the sklearn.svm.SVC classifier with `probability=True, C=2.0` and other parameters set to their default values.
For the input data we used the preprocessed dataset with numeric data normalized and the two insignificant features (**RestingBP**, **RestingECG**) dropped.

Result of running SVM iteration 1 on the dataset:



```
Result for Support Vector Machine
Bias: 0.07799442896935938
Variance: 0.07700557103064065
Accuracy on training set: 0.9220055710306406
Accuracy on test set: 0.845

Confusion Matrix:

[[80 15]
 [16 89]]

              precision    recall  f1-score   support

           0       0.83      0.84      0.84        95
           1       0.86      0.85      0.85       105

    accuracy                           0.84       200
   macro avg       0.84      0.84      0.84       200
weighted avg       0.85      0.84      0.85       200

Area Under Curve (AUC): 0.9221052631578948
```

Bias and variance:
According to the result above, by increasing parameter C, we lower the number of misclassified data and build a better fit on the training data, however, by increasing parameter C, we chose a smaller-margin hyperplane, in this case, the bias decreased by 0.02 but variance increased by 0.02. However, in this case, bias and variance are more balanced and pretty much the same.

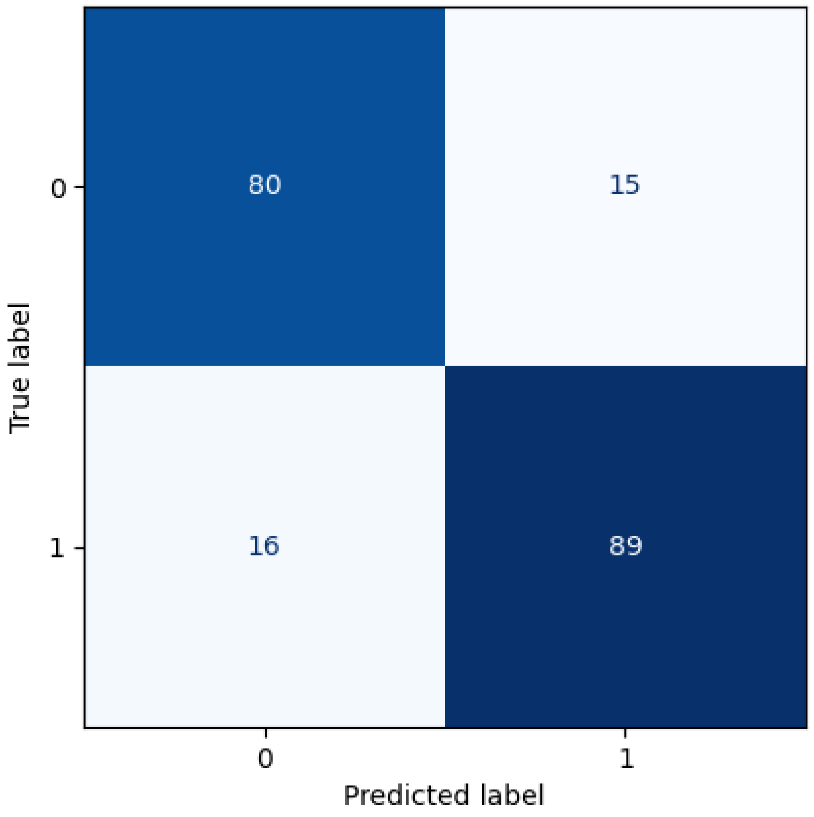
Accuracy:
In terms of accuracy, the accuracy on the training set improved by 0.02 and accuracy on the testing set decreased by 0.01.

F-1:
F-1 score is pretty much the same as the original model, at around 0.85.

AUC:
AUC is pretty much the same as the original model, at around 0.92.

Overall the model improves by decreasing bias and providing a more balanced bias and variance, and overall accuracy improves by 0.01.

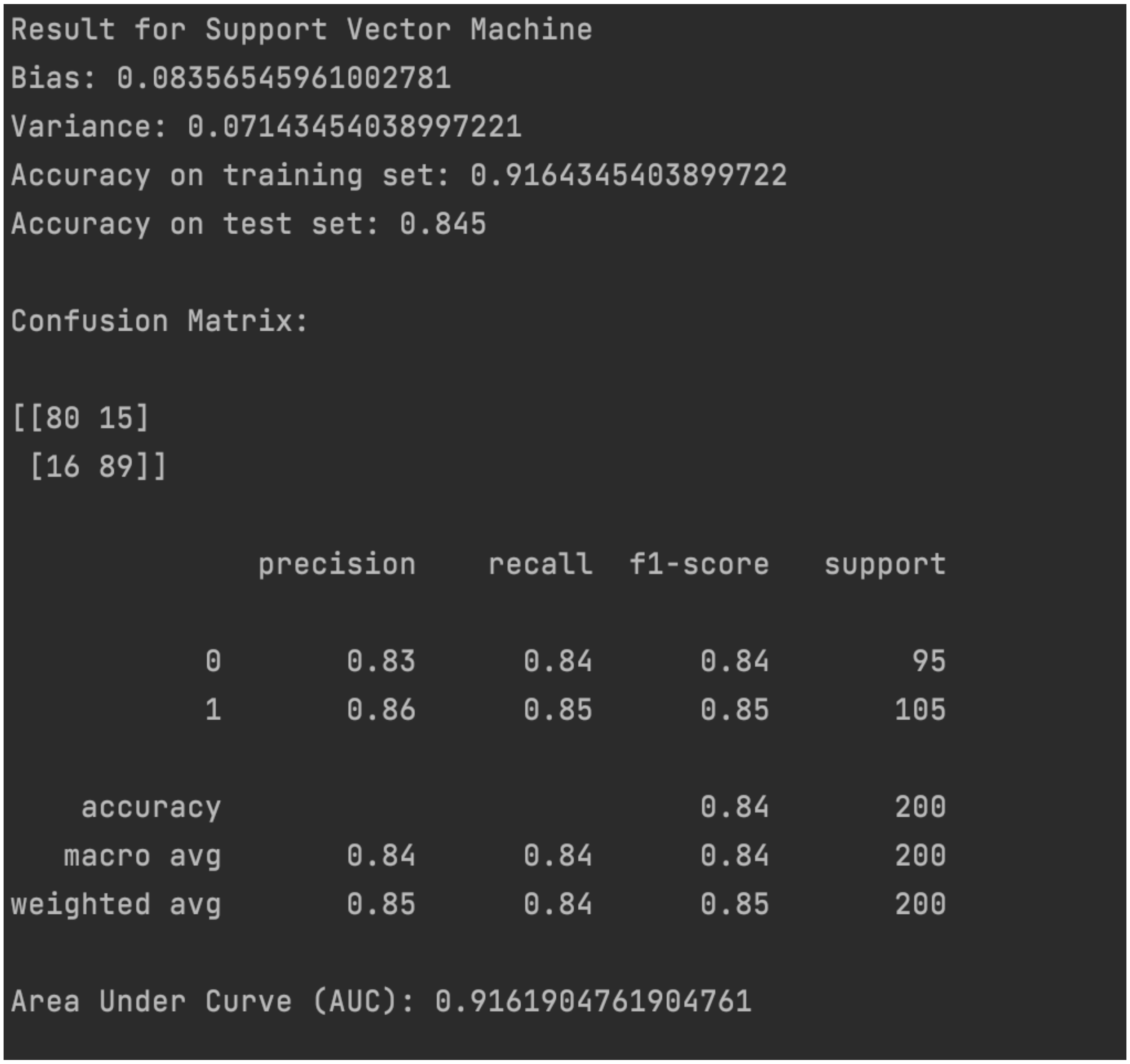
Confusion matrix of running SVM iteration 1 on the dataset:



Iteration 2: Use a more complex kernel

Using a more complex kernel function, such as a polynomial or Gaussian kernel, can increase the model's capacity to fit the data, potentially reducing bias.
For this iteration, we used the sklearn.svm.SVC classifier with `probability=True, kernel='poly'` and other parameters set to their default values.
For the input data we used the preprocessed dataset with numeric data normalized and the two insignificant features (**RestingBP**, **RestingECG**) dropped.

Result of running SVM iteration 2 on the dataset:



```
Result for Support Vector Machine
Bias: 0.08356545961002781
Variance: 0.07143454038997221
Accuracy on training set: 0.9164345403899722
Accuracy on test set: 0.845

Confusion Matrix:

[[80 15]
 [16 89]]

              precision    recall  f1-score   support

           0       0.83      0.84      0.84        95
           1       0.86      0.85      0.85       105

    accuracy                           0.84       200
   macro avg       0.84      0.84      0.84       200
weighted avg       0.85      0.84      0.85       200

Area Under Curve (AUC): 0.9161904761904761
```

Bias and variance:
According to the result above, by Using a more complex kernel function, we increased the model's capability to fit the training set's data, in this case, the bias decreased by around 0.02 but variance increased by around 0.02. However, in this case, bias and variance are more balanced and pretty much the same.

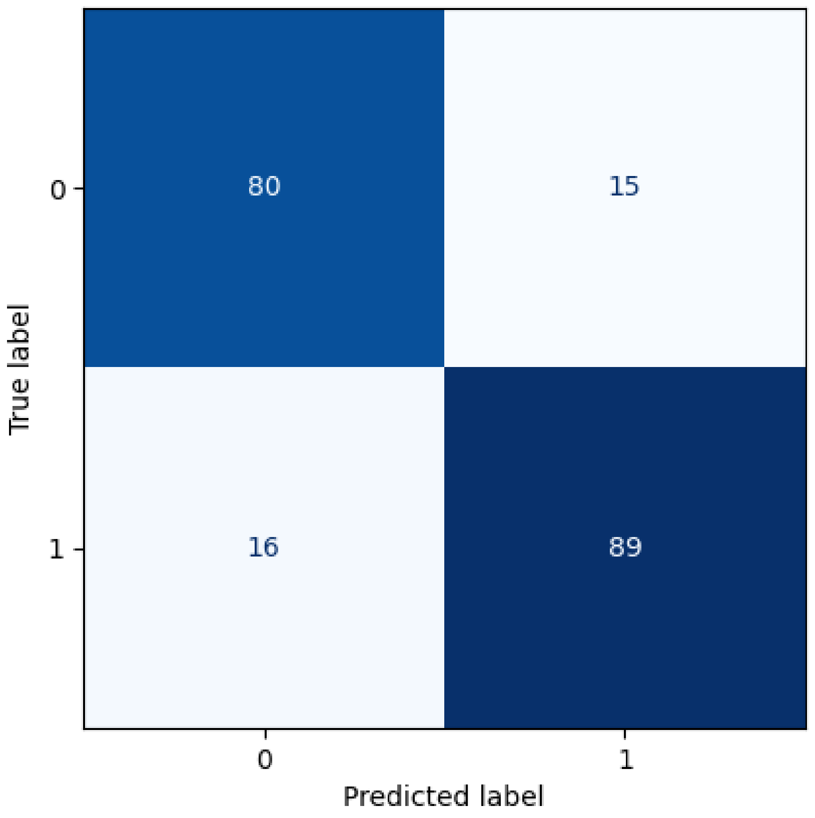
Accuracy:
In terms of accuracy, the accuracy on the training set improved by 0.02 and accuracy on the testing set decreased by 0.01.

F-1:
F-1 score is pretty much the same as the original model, at around 0.85.

AUC:
AUC is slightly smaller than the original model, at around 0.916.

Overall the model improves by decreasing bias and providing a more balanced bias and variance, and overall accuracy improves by 0.01.

Confusion matrix of running SVM iteration 2 on the dataset:

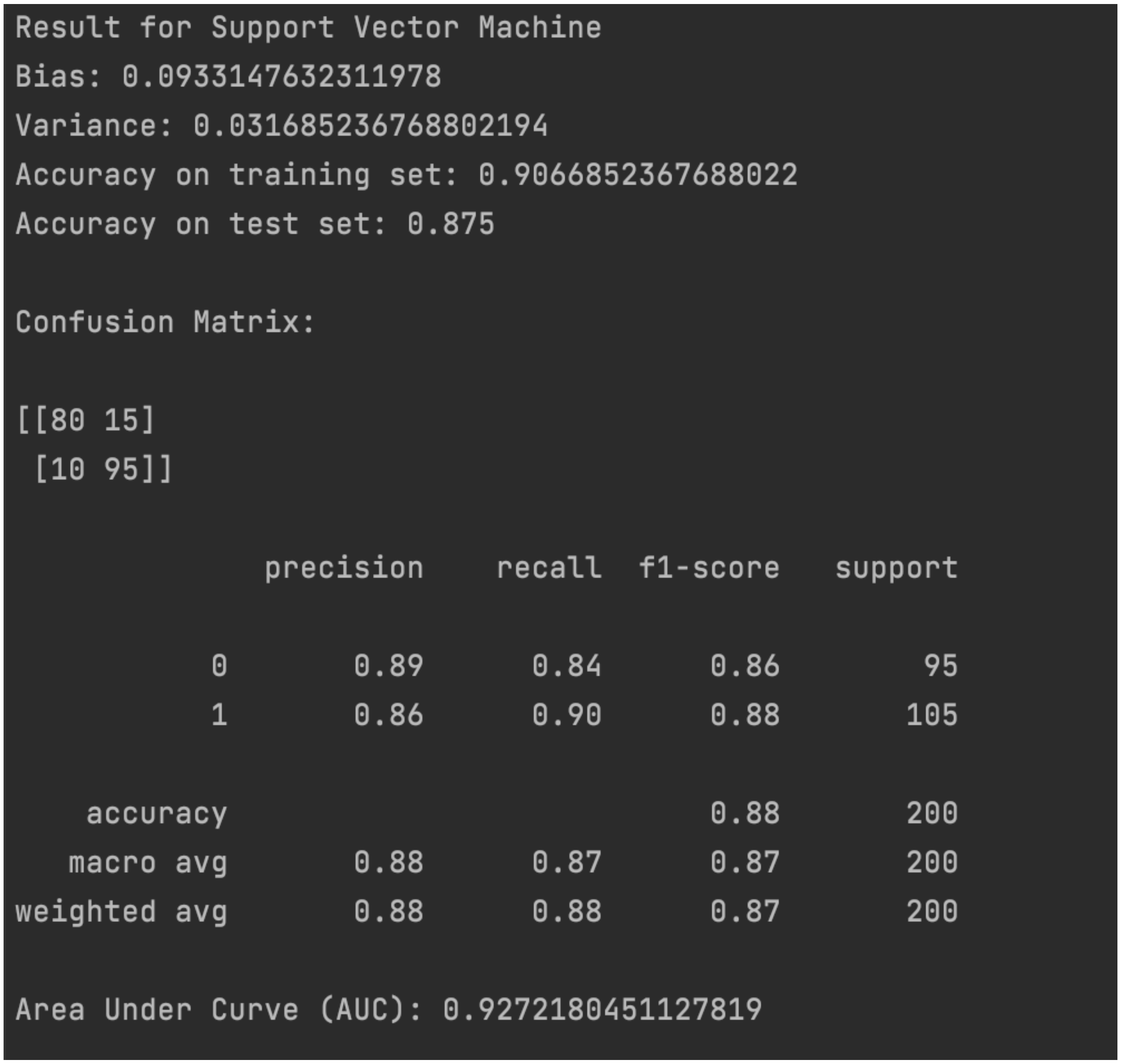


Iteration 3 (Best performance so far): Add more features to the data (add the dropped features back to the dataset)
By adding more features to the data, we can make the data and model more complex, potentially improving the performance.

For this iteration, we used the sklearn.svm.SVC classifier with `probability=True` and other parameters set to their default values.

For the input data we used the preprocessed dataset with numeric data normalized and the no features dropped.

Result of running SVM iteration 3 on the dataset:



```
Result for Support Vector Machine
Bias: 0.0933147632311978
Variance: 0.031685236768802194
Accuracy on training set: 0.9066852367688022
Accuracy on test set: 0.875

Confusion Matrix:

[[80 15]
 [10 95]]

              precision    recall  f1-score   support

           0       0.89      0.84      0.86        95
           1       0.86      0.90      0.88       105

    accuracy                           0.88       200
   macro avg       0.88      0.87      0.87       200
weighted avg       0.88      0.88      0.87       200

Area Under Curve (AUC): 0.9272180451127819
```

Bias and variance:
According to the result above, by adding more features to the dataset, we created a model with more complex data and more context in the dataset, in this case, the bias and variance both decreases, the bias decreased by 0.003 and variance decreased by 0.017

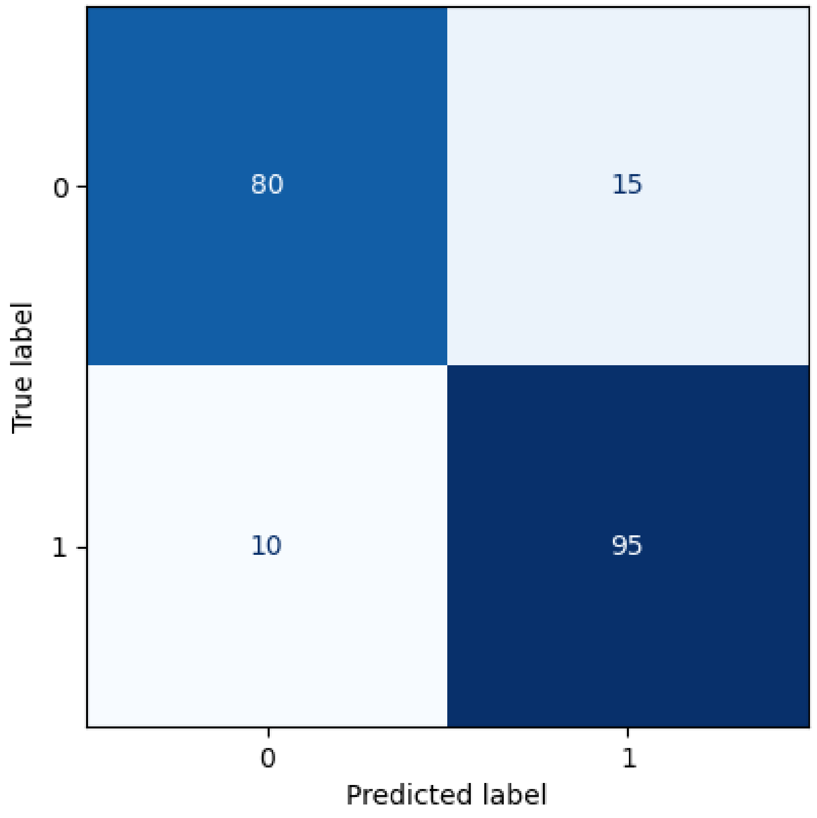
Accuracy:
In terms of accuracy, the accuracy on the training set improved by 0.003 and accuracy on the testing set improved by 0.02

F-1:
F-1 score increased from around 0.85 to around 0.87

AUC:
AUC also increased by 0.004

Overall the model improves at every statistic, the bias and variance both decreased, and accuracy for both training set and testing set improves, F-1 score and AUC also improves. **This is the best performed iteration on SVM so far.**

Confusion matrix of running SVM iteration 3 on the dataset:



Conclusion

From these 3 iterations, we understand that by increasing parameter C or applying a more complex kernel function, we are able to make the model fit the data better and decrease the bias, however, as a trade off, the variance increased a little bit as this model is more specific to the training data and might not fit other dataset well. The overall accuracy could also be slightly improved.

The best performing iteration is to add more features to the dataset, which can improve the complexity of the dataset and provide more context. In the 3rd iteration, every statistics and dimension is improved.

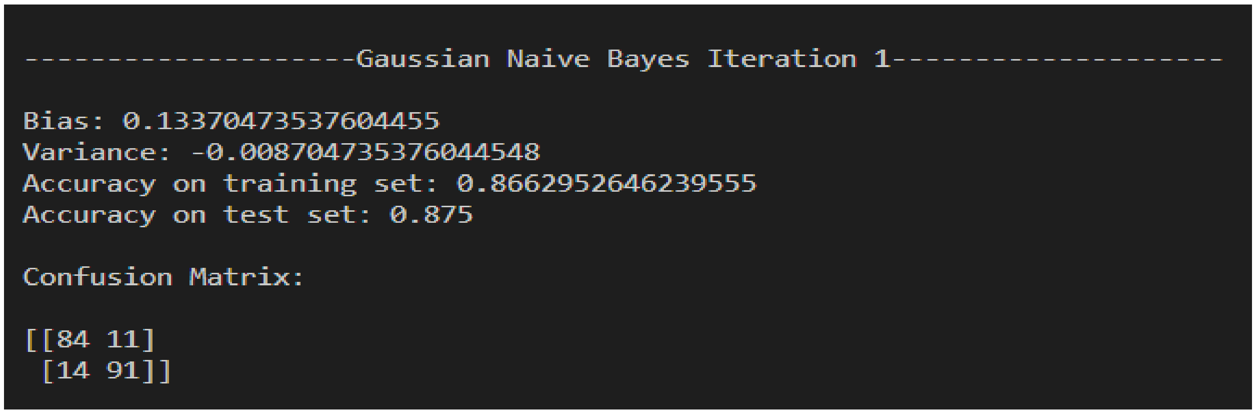
Gaussian Naive Bayes

For our GNB iterations we attempted to increase the variance and decrease the bias.

Iteration 1

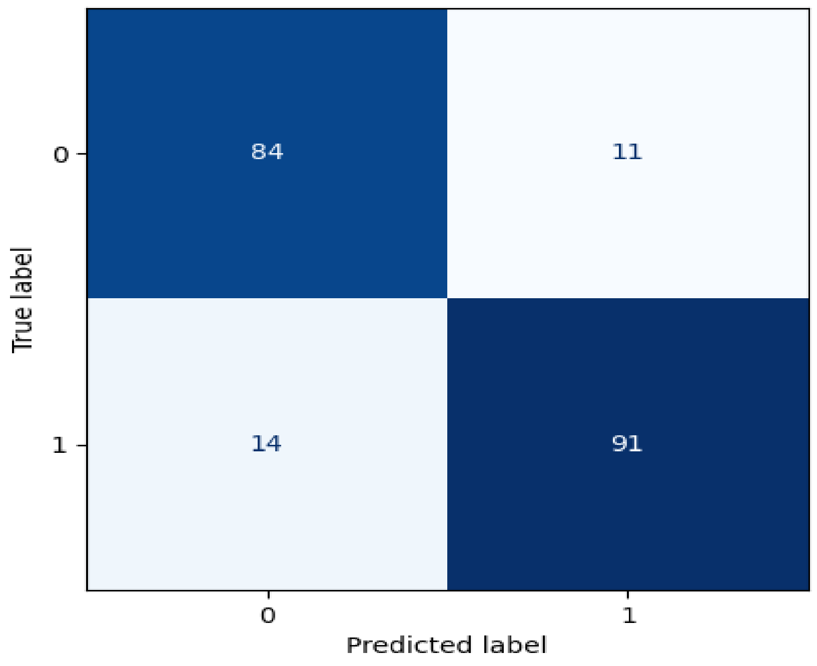
For iteration one, we modified the prior probability values from 0.5 for **HeartDisease**=0 and 0.5 for **HeartDisease**=1 to 0.4 for **HeartDisease**=0 and 0.6 for **HeartDisease**=1.

Gaussian Naive Bayes iteration 1 metrics:



```
--------------------Gaussian Naive Bayes Iteration 1--------------------

Bias: 0.13370473537604455
Variance: -0.008704735376044548
Accuracy on training set: 0.8662952646239555
Accuracy on test set: 0.875

Confusion Matrix:

[[84 11]
 [14 91]]
```

Gaussian Naive Bayes iteration 1 metric confusion matrix:

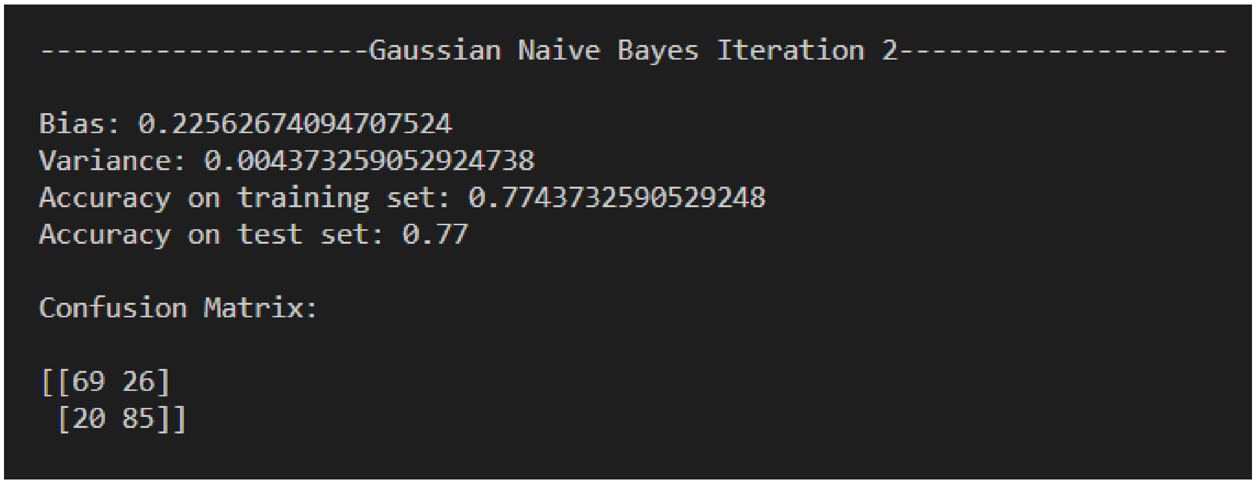


We see that this had the effect of increasing the variance (from -0.012 to -0.009) and increased the bias (from 0.132 to 0.134). It had a minor negative effect on the training and test set accuracy (-0.01 and -0.005 respectively). Therefore, this iteration had a positive effect on the variance, but a negative effect on the bias. This increase in variance is due to the larger increased error rate on the training set relative to the increased error rate on the training set. That is, accuracy on the training set went down, so variance went up.

Iteration 2

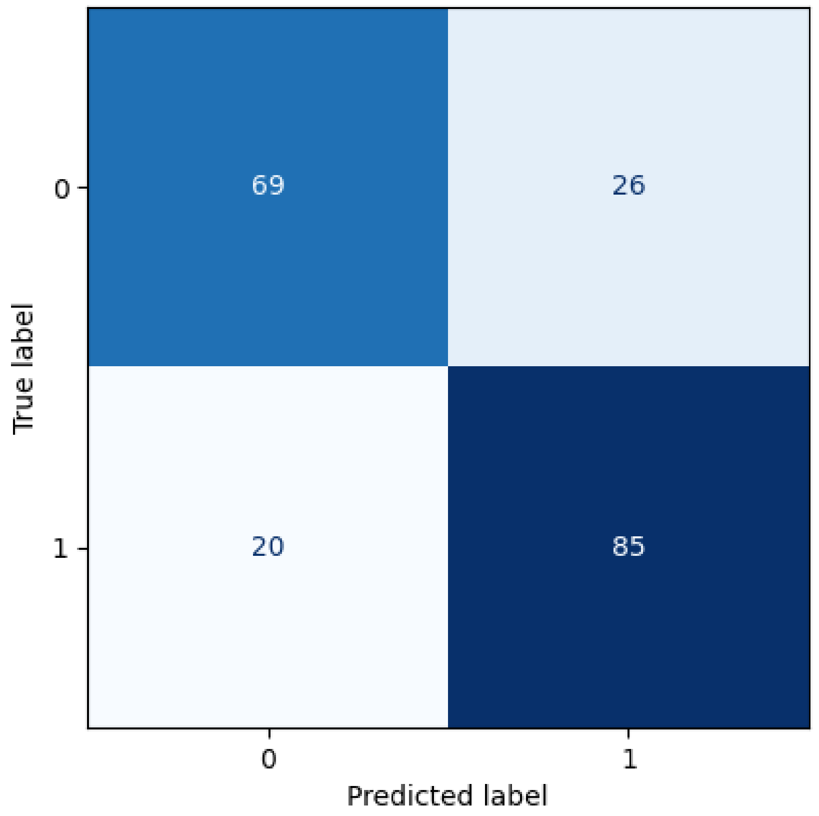
For this iteration we experimented with using only continuous input variables (i.e. **Age**, **Cholesterol**, **MaxHR**, **Oldpeak**). We kept the prior probability metaparameters the same as the previous iteration.

Gaussian Naive Bayes iteration 2 metrics:



```
--------------------Gaussian Naive Bayes Iteration 2--------------------

Bias: 0.22562674094707524
Variance: 0.004373259052924738
Accuracy on training set: 0.7743732590529248
Accuracy on test set: 0.77

Confusion Matrix:

[[69 26]
 [20 85]]
```

Gaussian Naive Bayes iteration 2 metric confusion matrix:

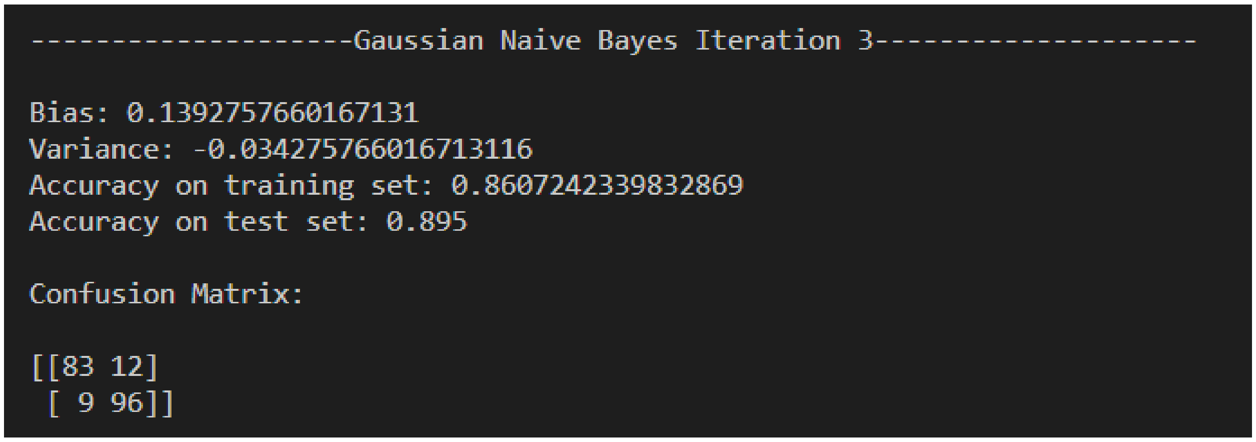


We see that this had the effect of increasing the variance (from -0.009 to 0.004) and increasing the bias (from 0.134 to 0.226). It had a major negative effect on the training and test set accuracy (-0.10 and -0.11 respectively). Similar to the last iteration this iteration had a positive effect on the variance, but a negative effect on the bias it also had a very large negative effect on the accuracy implying that removing the input variables in this way degraded the models performance. However, we see that it positively affected the ratio of type I errors to type II errors: there are now more type I errors than type II errors.

Iteration 3

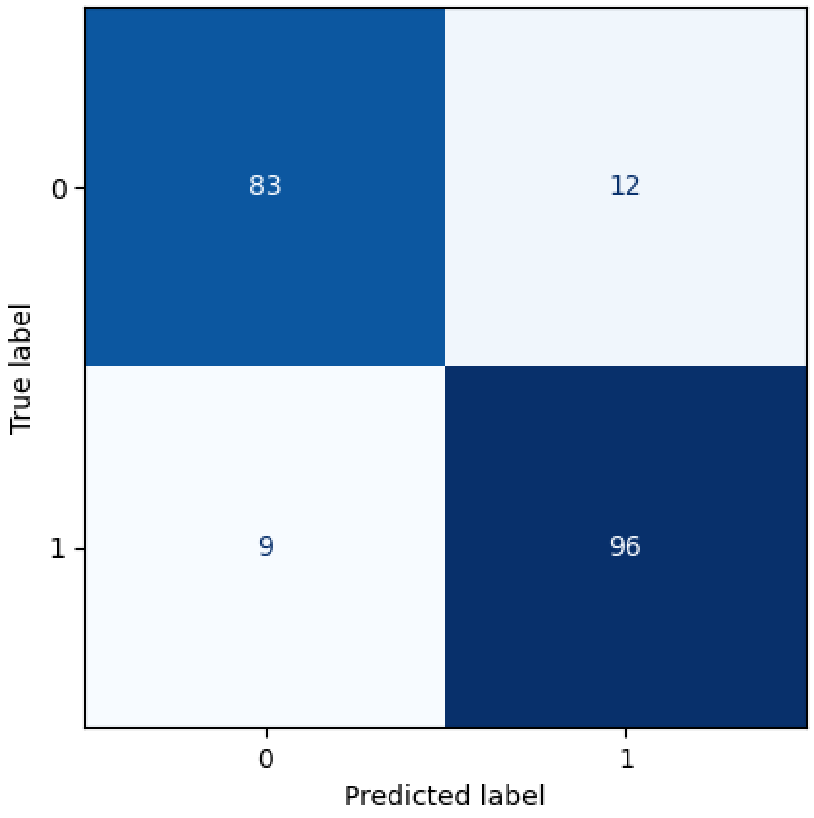
For this iteration, we returned to using the original input features meaning we threw out the model from iteration two. We also used the prior probabilities of iteration one. Instead, we experimented with increasing the var_smooting metaparameter from its default value of 1e-9 to 0.1

Gaussian Naive Bayes iteration 3 metrics:



```
--------------------Gaussian Naive Bayes Iteration 3--------------------

Bias: 0.1392757660167131
Variance: -0.034275766016713116
Accuracy on training set: 0.8607242339832869
Accuracy on test set: 0.895

Confusion Matrix:

[[83 12]
 [ 9 96]]
```

Gaussian Naive Bayes iteration 3 metric confusion matrix:



Since iteration 2 had such a negative impact on accuracy of the model, we will compare this iteration with iteration 1. We see that altering the var_smooting metaparameter had a negative effect on the training set accuracy and positive effect on the test set accuracy (-0.005 and +0.02 respectively). It decreased the variance (from -0.009 to -0.034) and increased the bias (from 0.134 to 0.139). It positively affected the ratio of type I errors to type II errors: there are now more type I errors than type II errors.

Conclusion

From these three iterations on the GNB model we can conclude that it is possible to increase the accuracy on the test set by altering the prior probability values from [0.5, 0.5] to [0.4, 0.6] and by altering the var_smoothing from 1e-9 to 0.1. However, we were unable to increase the variance without increasing the bias. Iteration 2 had the largest positive effect on the variance but it greatly negatively affected the model's performance. Therefore, we can conclude that in order to improve the model's bias-variance ratio while maintaining the model's accuracy would be to get more data.

Reflection

We learned how to combine various preprocessing techniques to conduct normalization, regularization, and feature selection on a heterogeneous data set.

- We learned how to visualize and understand the distributions and correlations of features by plotting the scatter matrix, heatmap and boxplot.
- We learned how to use lasso regularization coefficients to find and remove insignificant features.
- We learned how to use PCA to determine relations between independent features.
- We learned how to use a random forest classifier to obtain feature importance rankings.

We learned how to train and test the K Nearest Neighbors, Logistic Regression, Support Vector Machine, various Naive Bayes, and Decision Tree models provided by the sklearn library. We also learned how to evaluate their performance by looking at their accuracy, bias, variance, F1 score, and (AUC).

We learned how to balance the precision and recall statistics in certain ways by examining their PRC and ROC curves.

We learned how the SVM and Gaussian Naive Bayes models are affected by altering the various metaparameters.

Overall, Gaussian Naive Bayes proved to be the top performing model on our data set in terms of accuracy on the test set and AUC. It had an equivalent AUC to the Support Vector Machine classifier, but had a higher accuracy (especially on iteration 3).