Predicting Heart Disease

Table of contents

Introduction -

Our data comes from a UCI dataset which corroborates studies from Hungarian Institute of Cardiology, University Hospital, Zurich, Switzerland, University Hospital, Basel, Switzerland, and V.A. Medical Center, Long Beach and Cleveland Clinic Foundation. It contains data on 303 patients with the following attributes:age

- sex
- chest pain type (4 values)
- resting blood pressure
- serum cholesterol in mg/dl
- fasting blood sugar > 120 mg/dl
- resting electrocardiographic results (values 0,1,2)
- maximum heart rate achieved
- exercise induced angina
- oldpeak = ST depression induced by exercise relative to rest
- the slope of the peak exercise ST segment
- number of major vessels (0-3) colored by fluoroscopy
- thal: 3 = normal; 6 = fixed defect; 7 = reversible defect

Citations -

We found the dataset on Kaggle - https://www.kaggle.com/ronitf/heart-disease-uci

What we did -

The main goal of our project was to predict whether or not a patient would have heart disease based on a multitude of attributes. In order to do this we first had to understand and analyze the dataset. This was followed by cleaning the data. First, we checked if any of the attributes contained null inputs. Luckily, our dataset did not have any null values. The next step was to analyze each attribute and the correlations with our target, whether or not the individual has heart disease. The next steps were to apply machine learning models to the data set. We began with a Neural Network to our data and got an accuracy of 85%. Next, we applied Logical Regression, getting an accuracy of 83%, a precision of 78% , and a f1 score of 85%. Finally, we applied a K-Nearest Neighbors model and got an accuracy of 85%, a precision score of 86%, and a f1 score of 78%. The last step was to visualize our results which we did through a series of graphs and plots.

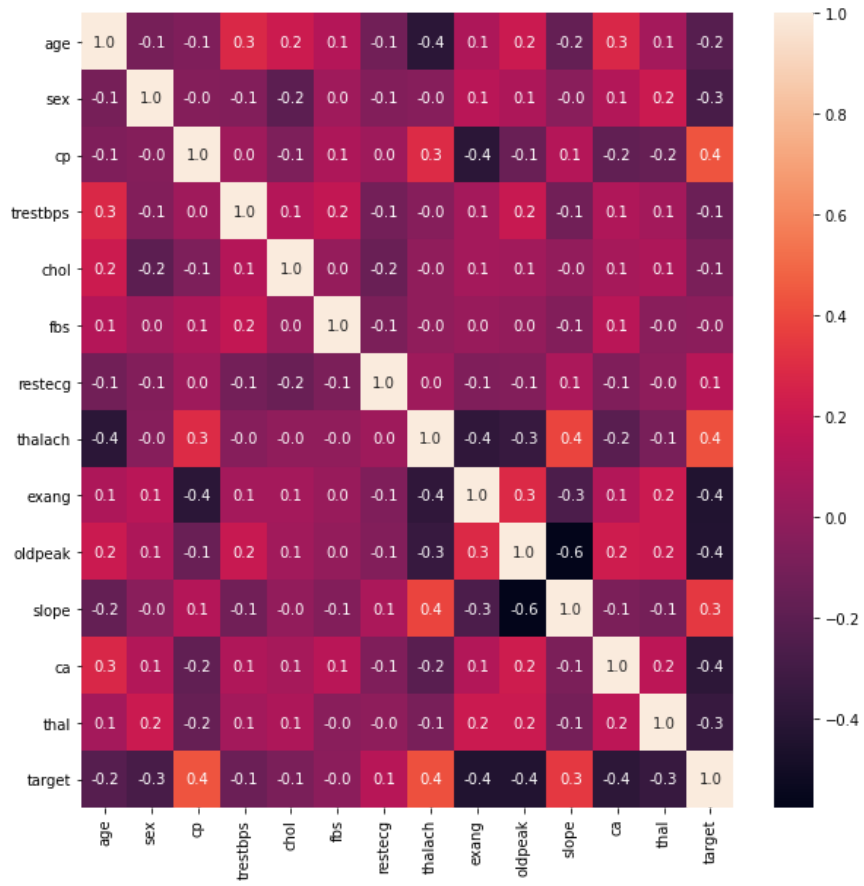
How it works-

Data Analysis & Feature Extraction

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

After doing df.describe(), the data looks relatively clean. All variables are numeric, and the data appears to be balanced.

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733	1.039604	1.399340	0.729373
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794	1.161075	0.616226	1.022606
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	1.000000	0.000000
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	0.800000	1.000000	0.000000
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.000000	1.600000	2.000000	1.000000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	4.000000

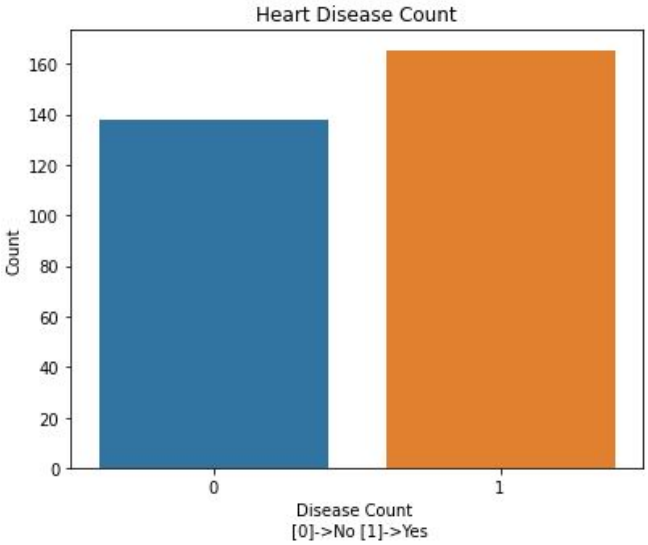


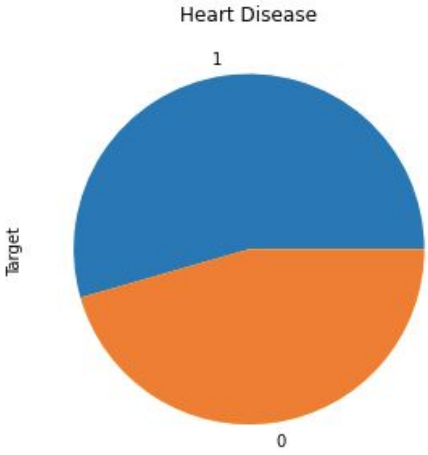
Correlation matrix

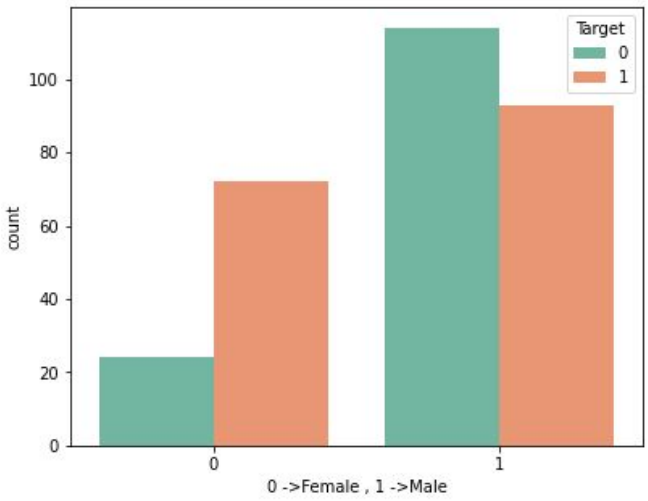
After analyzing the correlation matrix, it's clear that chest pain, maximum heart rate achieved, and the slope of the peak exercise ST segment are the highest positive correlated

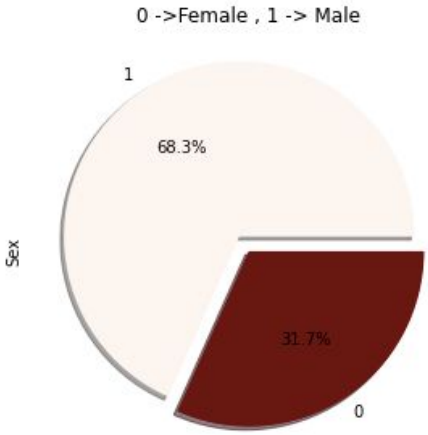
attributes. There are many negatively correlated variables, but fbs and chol have very low correlation scores. These should be dropped.

How many people are suffering from Heart Disease ?



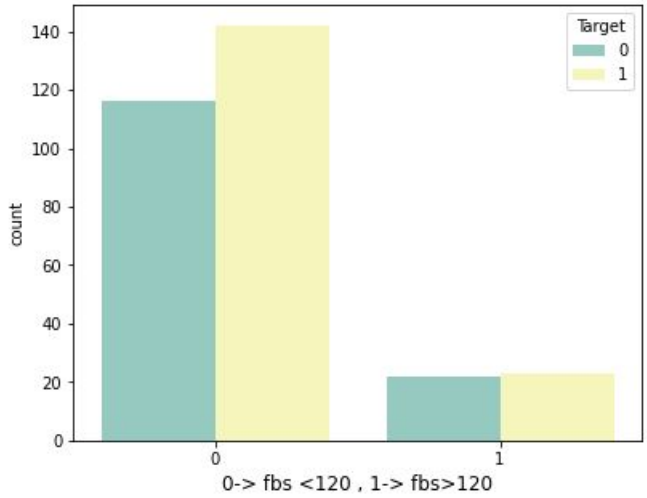


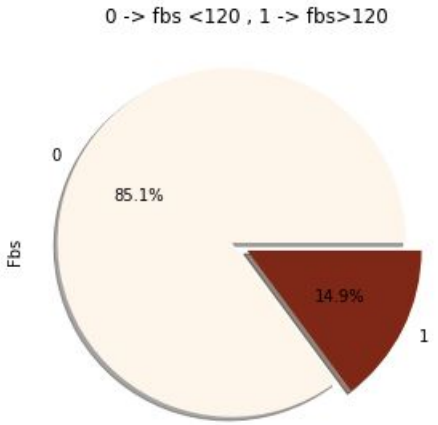




Number of Women suffering from Heart Disease are more than Men but Men population is more than Women. We will use these insights for our model development.

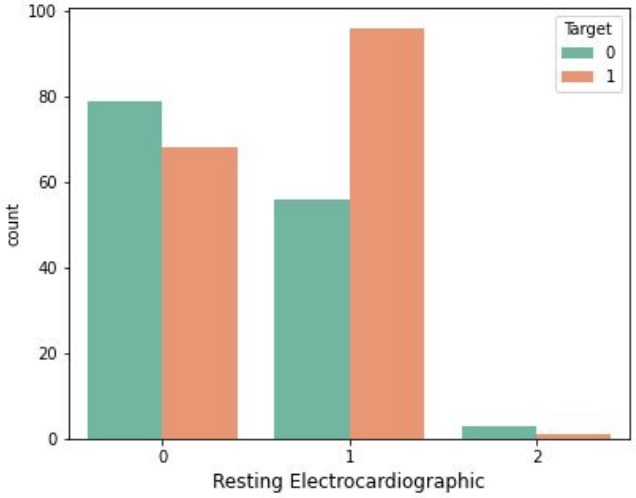
Fasting Blood Sugar (FBS)

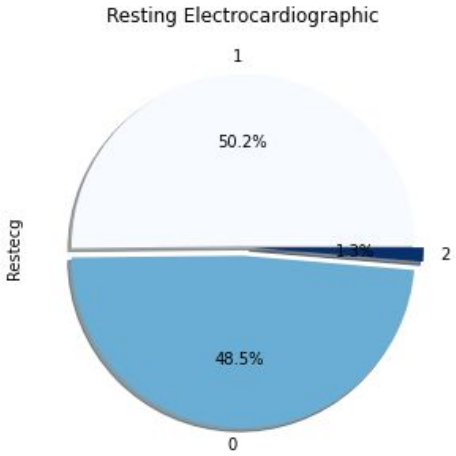




People having fps < 120 have more chance of having Heart Disease than people having fps >120

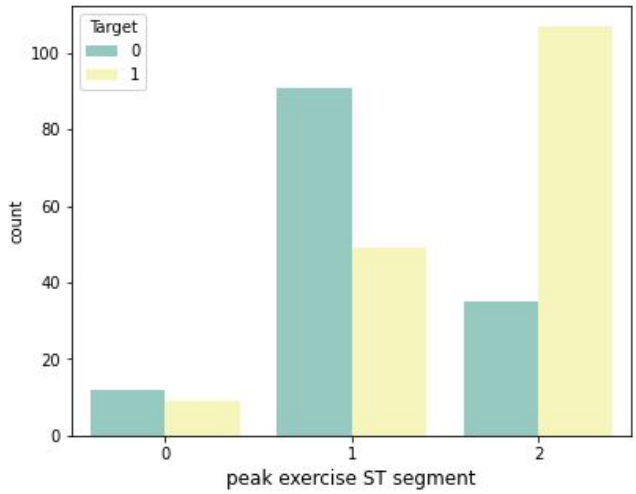
Resting ECG

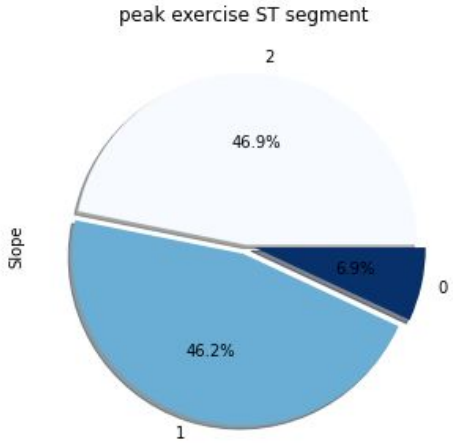




If resting electrocardiographic is 1 then person have more chances of suffering from Heart Disease

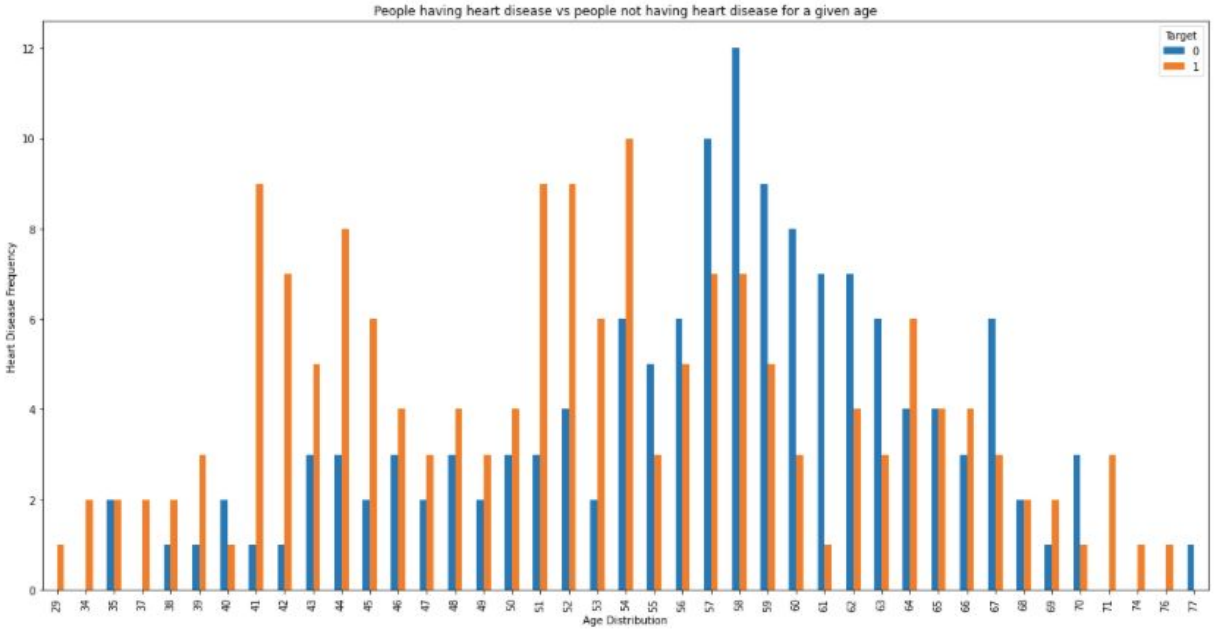
Peak Exercise ST Segment





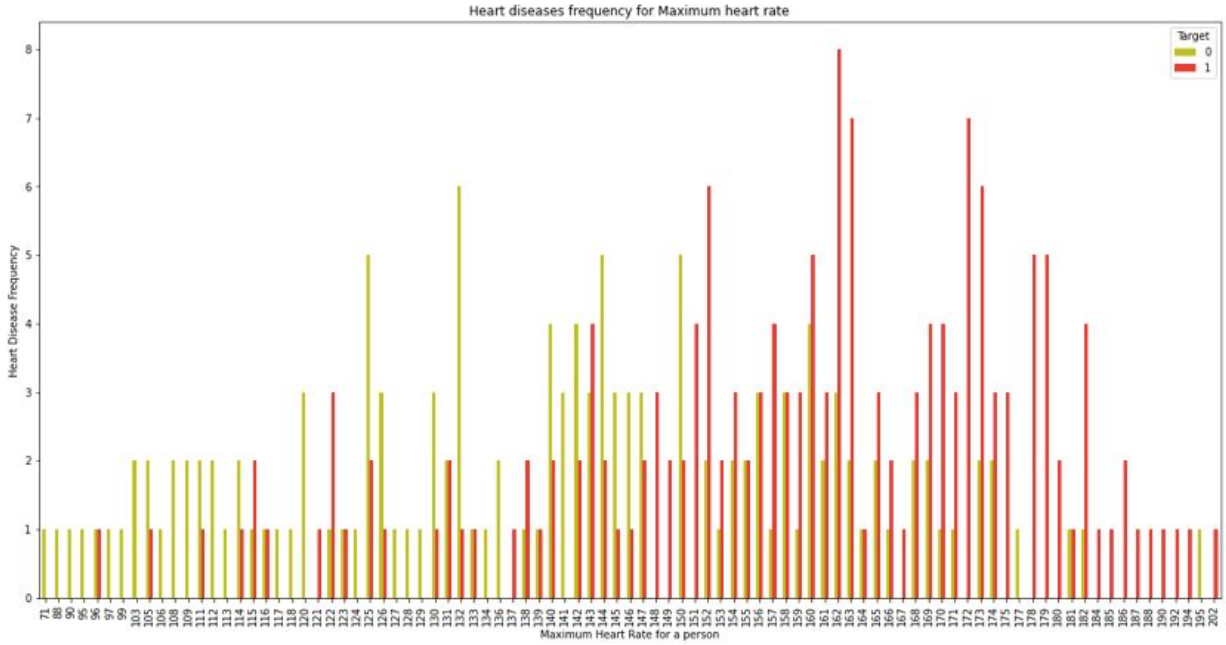
Feature (the peak exercise ST segment slope) has three symbolic values (flat, up sloping, downsloping). Therefore People having up sloping are more prone to Heart Disease than flat and downsloping.

Analyzing heart disease over age distribution



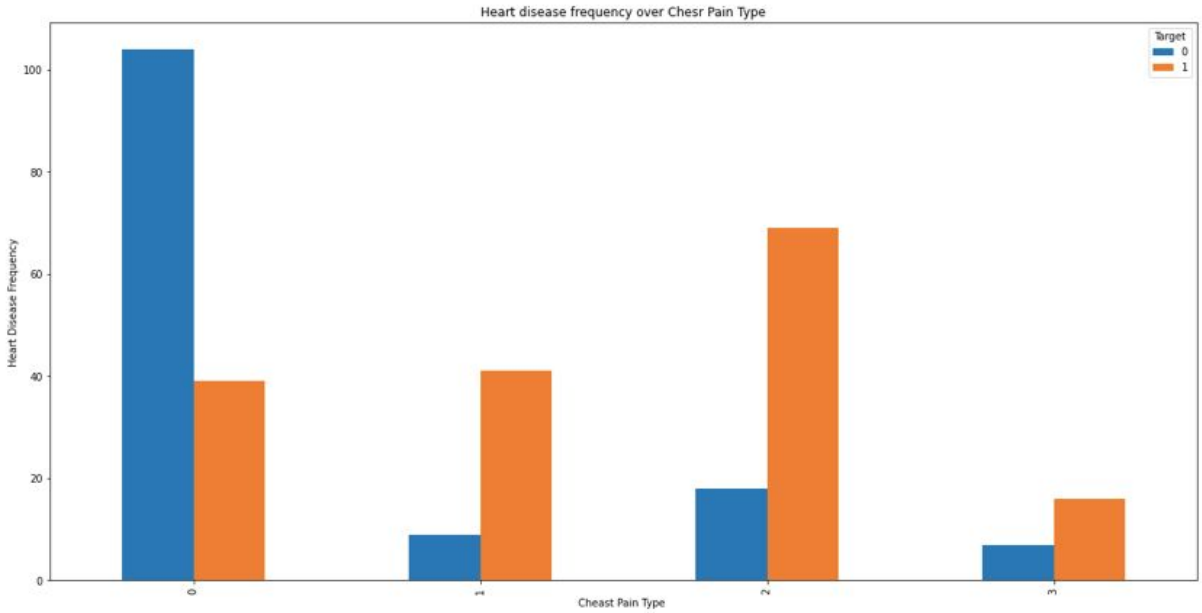
We could infer from the chart above that age is not a huge influencing factor for heart disease

Analyzing heart disease over Maximum Heart Rate



We could infer from the above chart that people who have higher heart rate has a higher probability of having a heart disease

Analyzing heart disease frequency over chest pain type



Chest pain type:
1 --> Typical angina 2 --> Atypical angina 3 --> Non-anginal pain 4 --> Asymptomatic
We could infer from the above chart that people who have Atypical angina or non-anginal pain have a higher probability of having heart disease.

Modeling -

Neural Networks - Binary Classification

Three layers were used for the sequential model.

```
model = models.Sequential()
model.add(layers.Dense(16,activation='relu',
input_shape=(trainAttr.shape[1],)))
model.add(layers.Dense(16,activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

We also used binary_crossentropy, as it was ideal for binary classification.

```
model.compile(optimizer='rmsprop',loss='binary_crossentropy',
metrics=['accuracy'])
```

```
history = model.fit(trainAttr, trainTarget, epochs=8, batch_size=16)
```

```
Epoch 1/8
16/16 [==============================] - 0s 1ms/step - loss: 0.6632 - accuracy: 0.6446
Epoch 2/8
16/16 [==============================] - 0s 1ms/step - loss: 0.6051 - accuracy: 0.7769
Epoch 3/8
16/16 [==============================] - 0s 1ms/step - loss: 0.5531 - accuracy: 0.8140
Epoch 4/8
16/16 [==============================] - 0s 2ms/step - loss: 0.5099 - accuracy: 0.8264
Epoch 5/8
16/16 [==============================] - 0s 1ms/step - loss: 0.4780 - accuracy: 0.8223
Epoch 6/8
16/16 [==============================] - 0s 1ms/step - loss: 0.4514 - accuracy: 0.8223
Epoch 7/8
16/16 [==============================] - 0s 1ms/step - loss: 0.4329 - accuracy: 0.8182
Epoch 8/8
16/16 [==============================] - 0s 1ms/step - loss: 0.4202 - accuracy: 0.8182
```

```
results = model.evaluate(testAttr, testTarget)
results
```

```
2/2 [==============================] - 0s 4ms/step - loss: 0.3730 - accuracy: 0.8525
[0.3730427622795105, 0.8524590134620667]
```

KNN

K-Nearest Neighbors is used to classify a value by looking at all cases and classifying the value by selecting a certain number of cases based on similarity.

First, we have to split the data into training and testing sets.

```
Xtrain,Xtest,Ytrain,Ytest = train_test_split(data_scaled, data_label,
test_size=0.20, random_state=217)
```

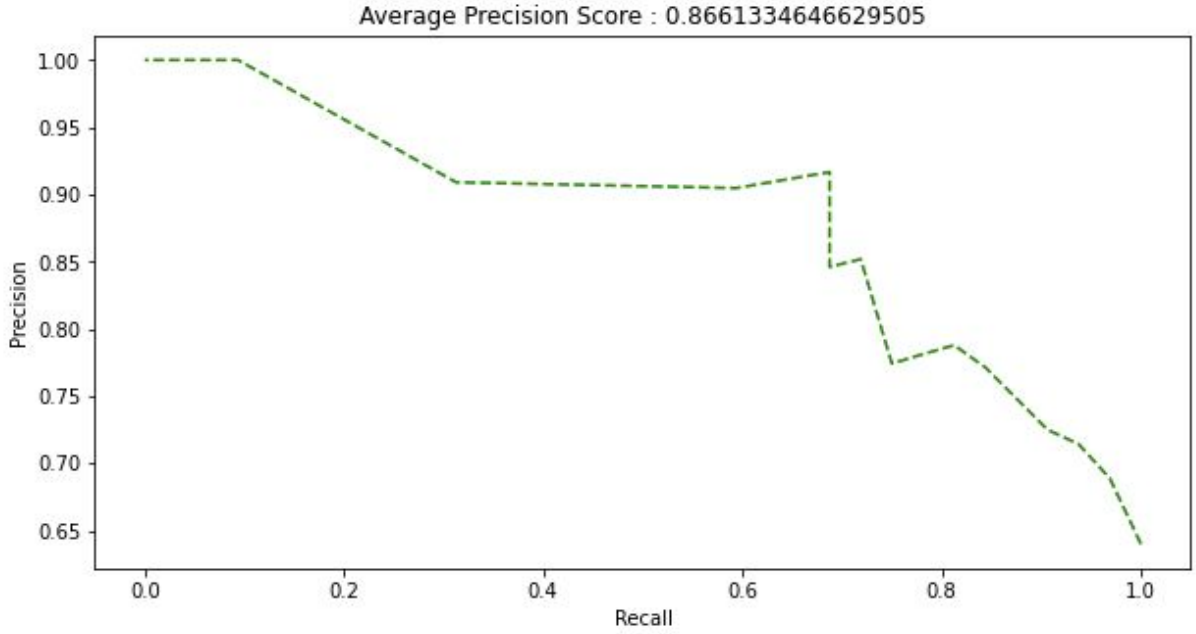
Then we have to pick a k value that gives us the highest accuracy. In this case, our highest accuracy k value is 17.

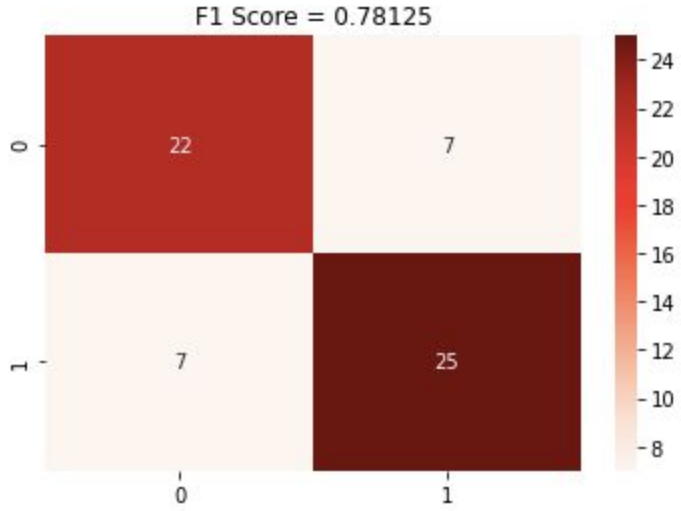
```
for i in range(1, 20):
  k=KNeighborsClassifier(algorithm='auto',n_neighbors= i)
  score_k=CrossVal(Xtrain, Ytrain.values.ravel(), k)
  print('{} : {}'.format(i, score_k))
```

Next we apply our KNN using this k value, and fit the value.

```
k=KNeighborsClassifier(algorithm='auto',n_neighbors= 17)
k.fit(Xtrain,Ytrain)
```

Finally, using our model, we can get our accuracy, precision, recall, and f1 scores.


Average Precision Score : 0.8661334646629505
Precision
Recall
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.0
0.2
0.4
0.6
0.8
1.0


F1 Score = 0.78125
22
7
7
25
0
1
0
1
24
22
20
18
16
14
12
10
8

Logistic Regression

Logistic regression is used when trying to predict a categorical value rather than a discrete one. In this project, our goal is to predict whether or not a certain patient has heart disease - a categorical variable. It works similarly to linear regression, but uses the value extracted from the linear regression and is transformed into a categorical variable. This is done by a decision boundary - discrete values will be assigned into categorical values by which side of the decision boundary they fall on. It uses gradient descent rather than least squared as its cost function.

First, the fbs and restecg are dropped since they have low correlation and may hurt the model.

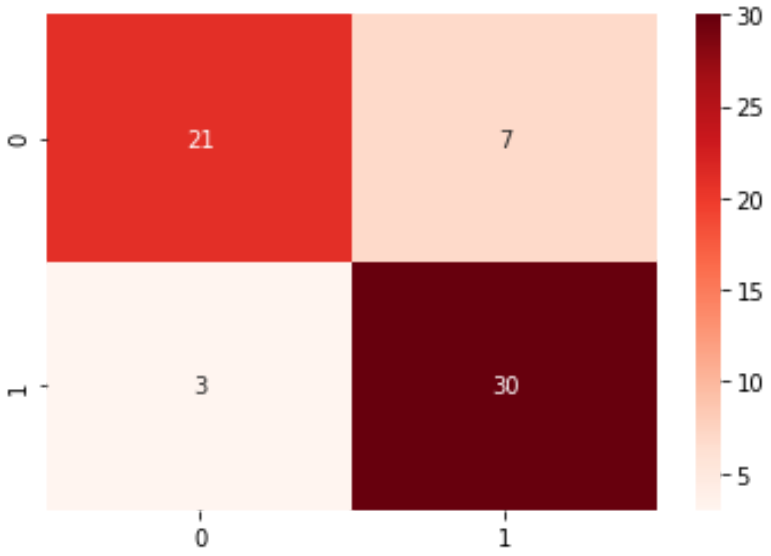
```
x = np.array(data.drop(['Target', 'Fbs', 'Chol'], 1))
y = np.array(data['Target'])
```

Then, the data is split into testing and training

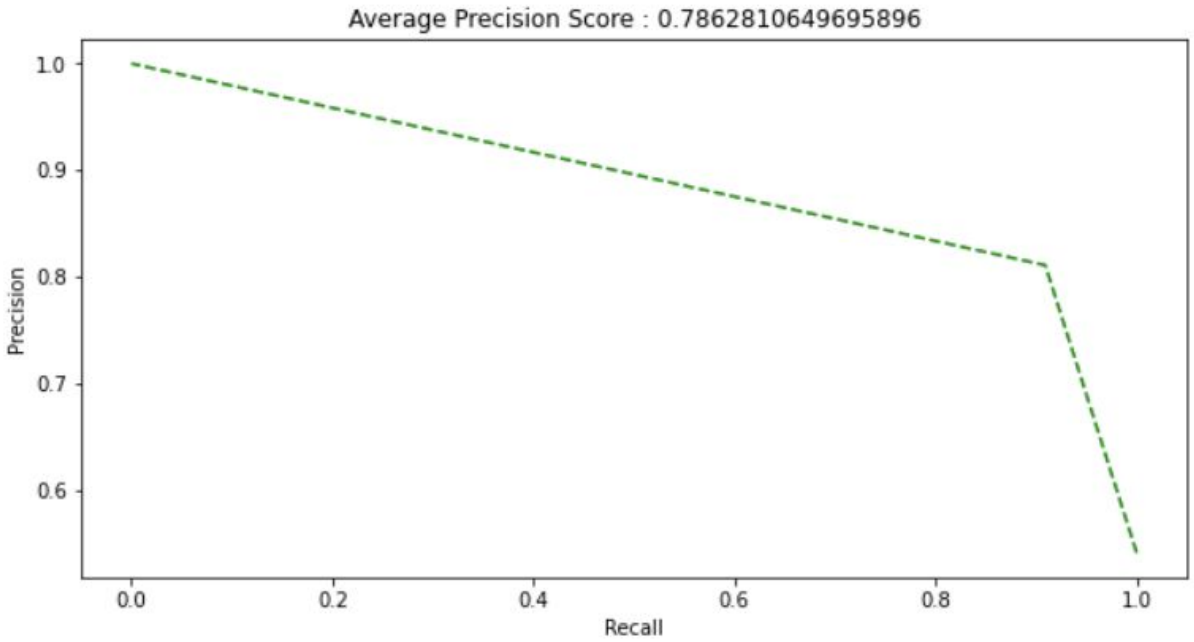
```
x_train, x_test, y_train, y_test = model_selection.train_test_split(x, y, stratify=y,
random_state=42, test_size = 0.2)
```

Finally, we make the logistic regression model and predict.

```
model = make_pipeline(LogisticRegression(fit_intercept=False))
model.fit(x_train, y_train)
predict = model.predict(x_test)
```

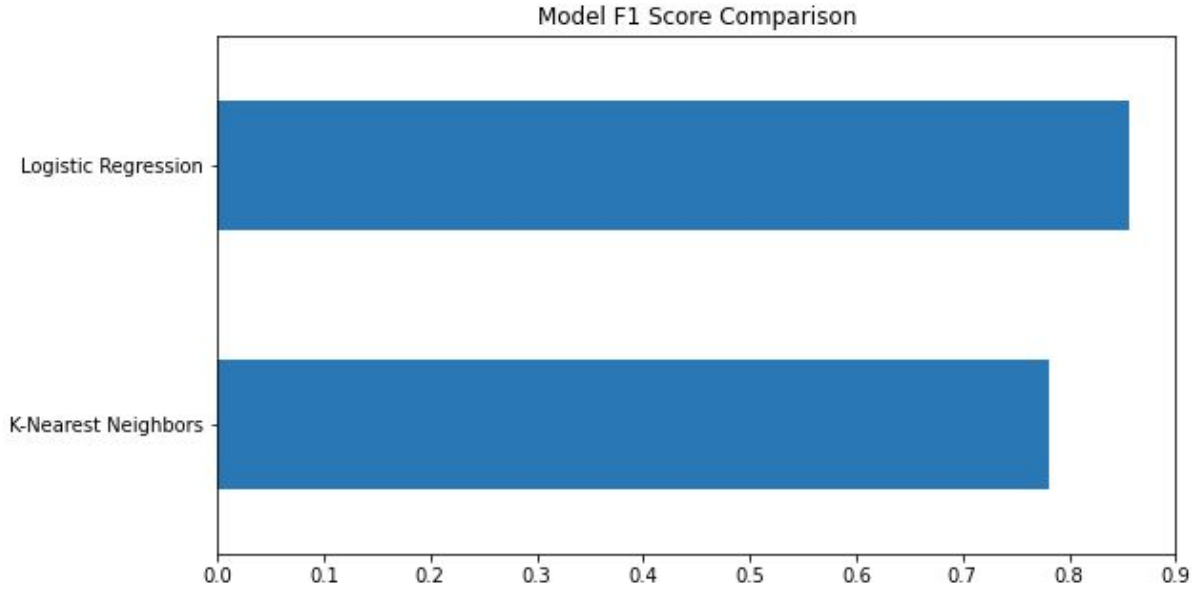


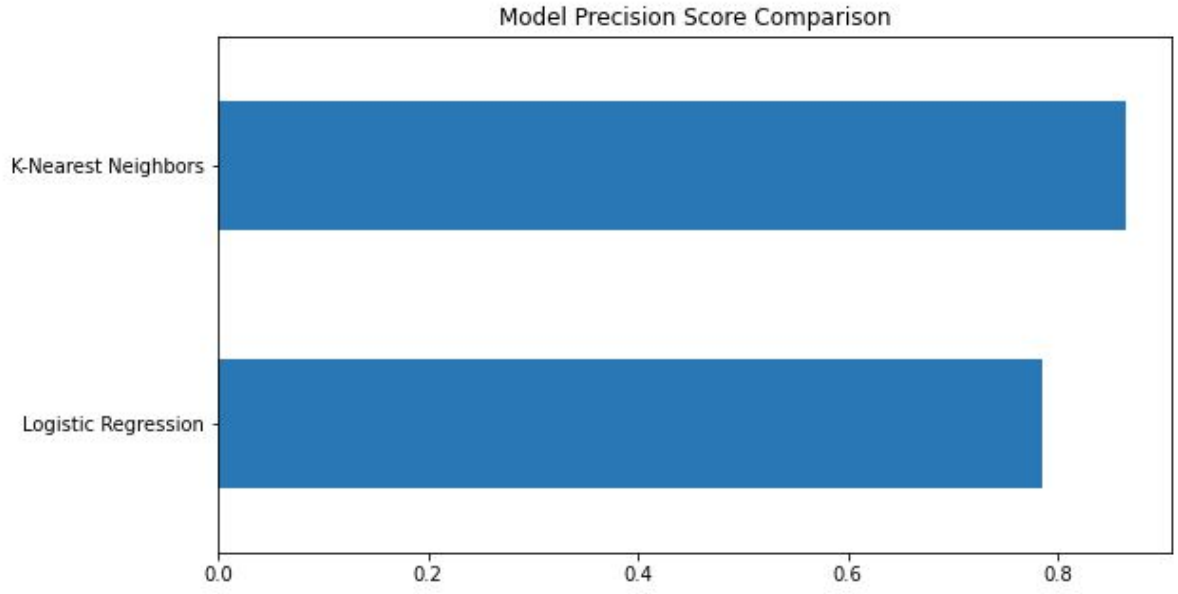
Confusion matrix

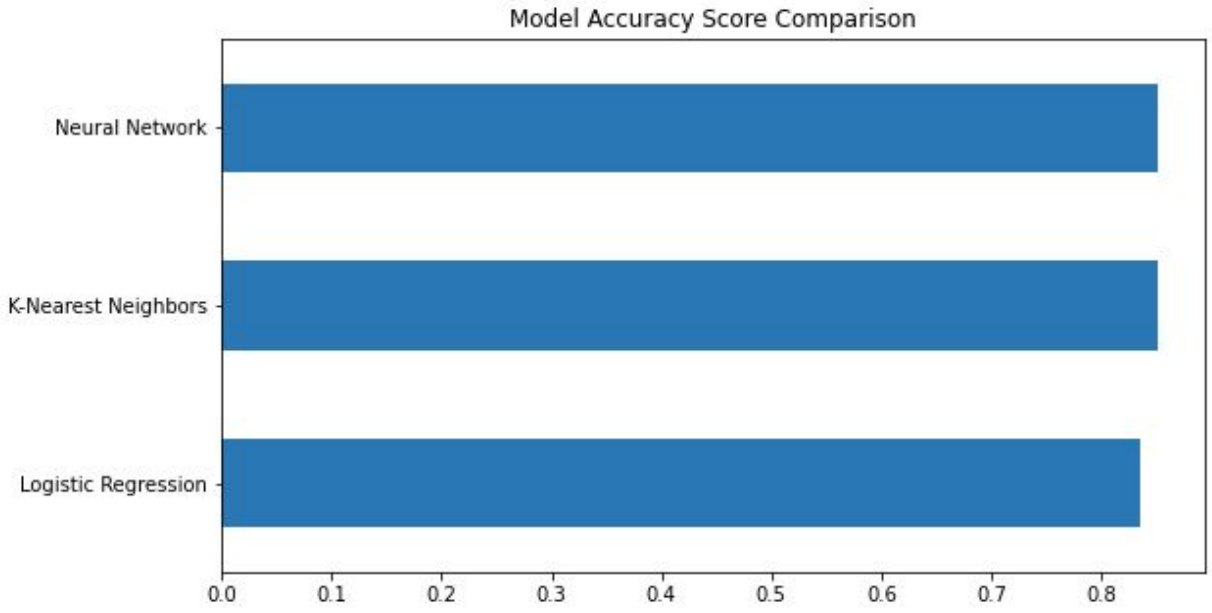


Results

The results are best described through the following graphs:







- Our conclusions are dependant on which evaluation metric is most sought after
- For accuracy, the neural network or KNN is the best
- For Precision, KNN has the highest value
- For F1 Score, Logical Regression is the best bet
- Overall, it seems that KNN gives the best results but all three machine learning models offer benefits