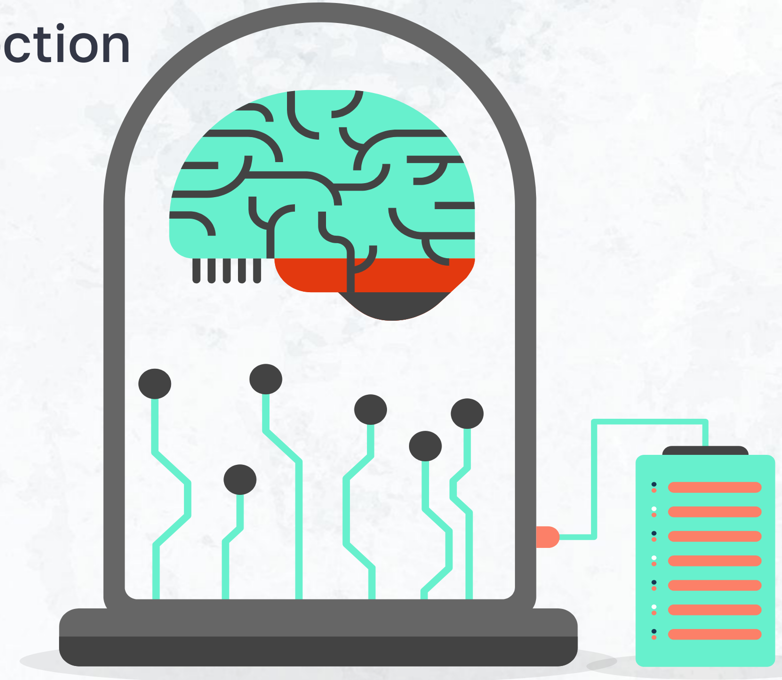
Machine Learning - Final project -

EGEI 2023

Iulia Florea
Guido Pistorius
Jairo Ramirez Torres

Table of contents

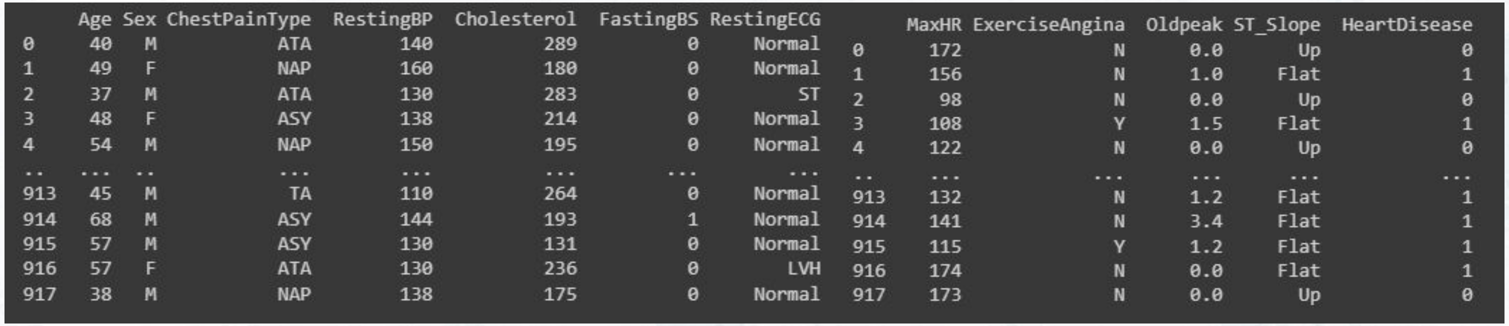


01 →

Data selection and cleaning

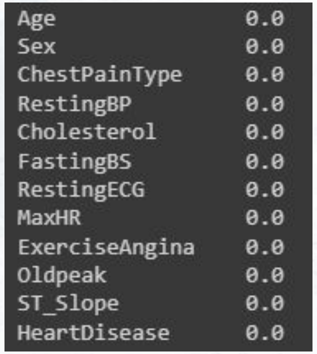
- Dataset: Heart Failure Prediction
- Variables: 5 numerical and 7 categorical
  - Target variable: HeartDisease (0 or 1)

Missing values



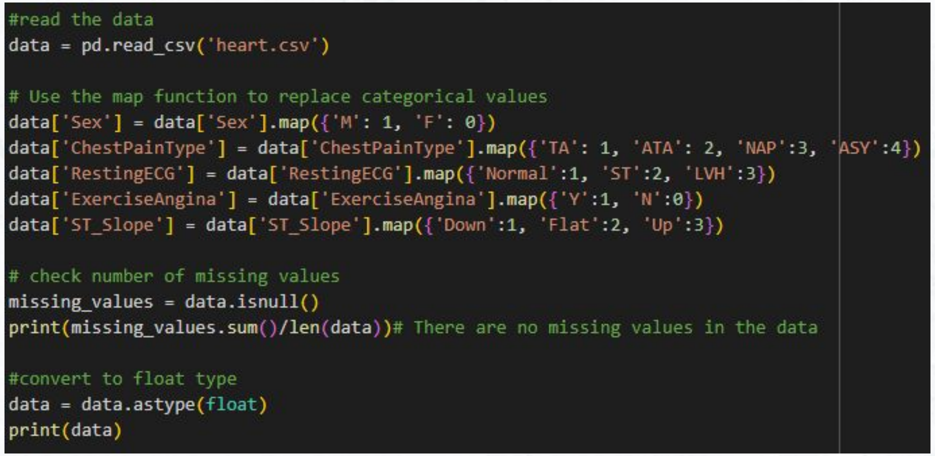
```
     Age Sex ChestPainType  RestingBP  Cholesterol  FastingBS RestingECG
0     40   M           ATA        140          289          0     Normal
1     49   F           NAP        160          180          0     Normal
2     37   M           ATA        130          283          0         ST
3     48   F           ASY        138          214          0     Normal
4     54   M           NAP        150          195          0     Normal
..   ...  ..           ...        ...          ...        ...        ...
913   45   M            TA        110          264          0     Normal
914   68   M           ASY        144          193          1     Normal
915   57   M           ASY        130          131          0     Normal
916   57   F           ATA        130          236          0        LVH
917   38   M           NAP        138          175          0     Normal
```

```
     MaxHR ExerciseAngina  Oldpeak ST_Slope  HeartDisease
0      172              N      0.0       Up             0
1      156              N      1.0     Flat             1
2       98              N      0.0       Up             0
3      108              Y      1.5     Flat             1
4      122              N      0.0       Up             0
..     ...            ...      ...      ...           ...
913    132              N      1.2     Flat             1
914    141              N      3.4     Flat             1
915    115              Y      1.2     Flat             1
916    174              N      0.0     Flat             1
917    173              N      0.0       Up             0
```



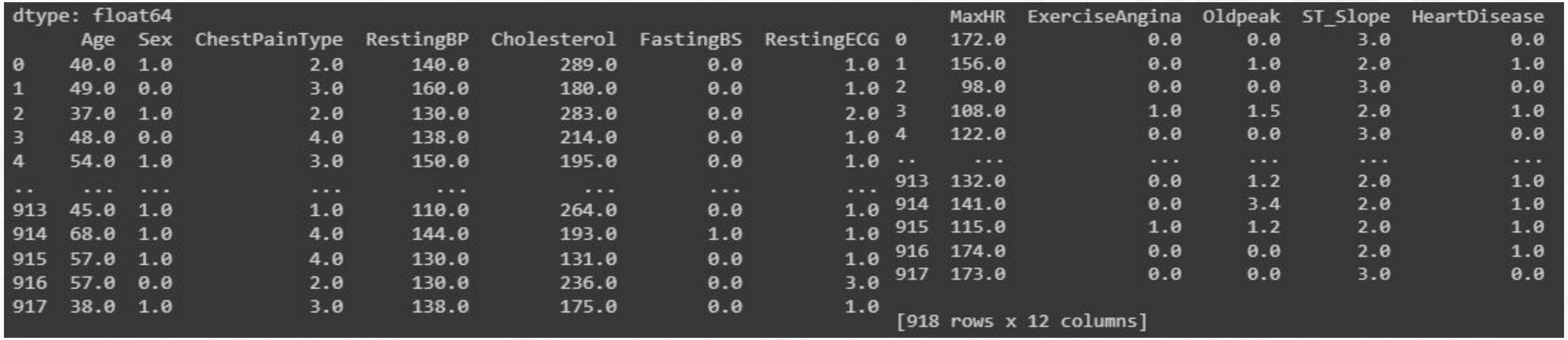
```
Age               0.0
Sex               0.0
ChestPainType     0.0
RestingBP         0.0
Cholesterol       0.0
FastingBS         0.0
RestingECG        0.0
MaxHR             0.0
ExerciseAngina    0.0
Oldpeak           0.0
ST_Slope          0.0
HeartDisease      0.0
```

- Data cleaning
  - Encoding categorical variables
  - Checking for missing values
  - Converting the values to *float* type



```
#read the data
data = pd.read_csv('heart.csv')

# Use the map function to replace categorical values
data['Sex'] = data['Sex'].map({'M': 1, 'F': 0})
data['ChestPainType'] = data['ChestPainType'].map({'TA': 1, 'ATA': 2, 'NAP':3, 'ASY':4})
data['RestingECG'] = data['RestingECG'].map({'Normal':1, 'ST':2, 'LVH':3})
data['ExerciseAngina'] = data['ExerciseAngina'].map({'Y':1, 'N':0})
data['ST_Slope'] = data['ST_Slope'].map({'Down':1, 'Flat':2, 'Up':3})

# check number of missing values
missing_values = data.isnull()
print(missing_values.sum()/len(data))# There are no missing values in the data

#convert to float type
data = data.astype(float)
print(data)
```

Data after encoding

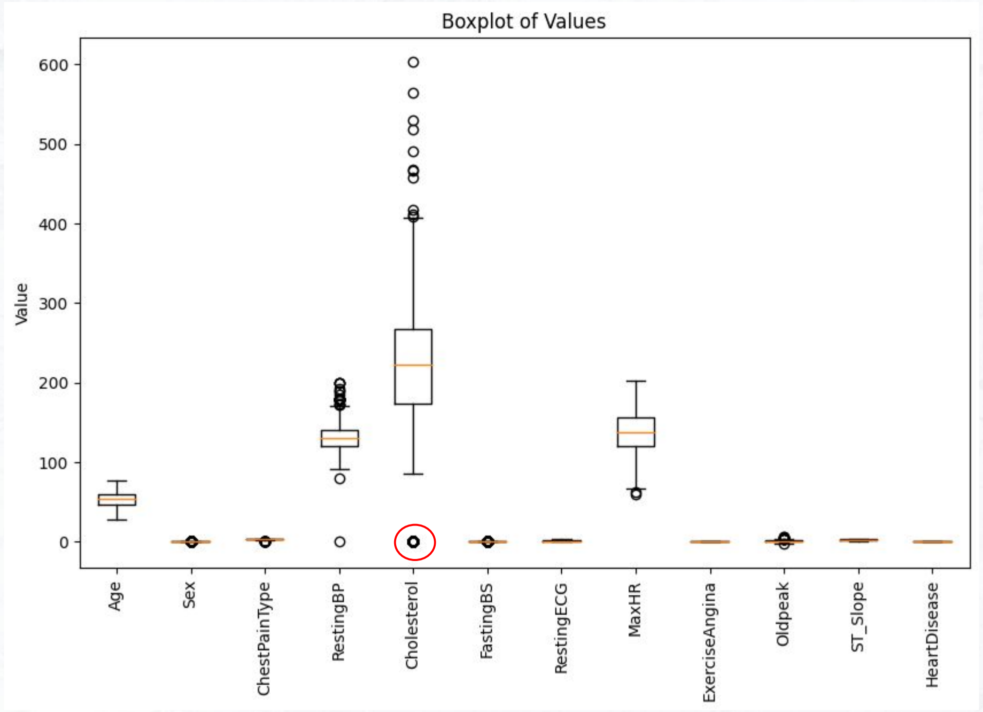


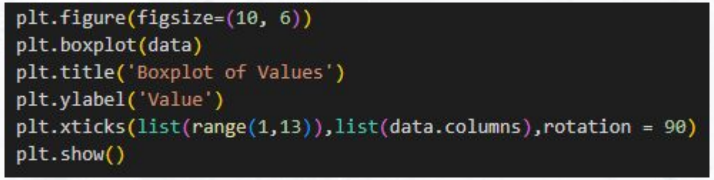
```
dtype: float64                                                                    MaxHR  ExerciseAngina  Oldpeak  ST_Slope  HeartDisease
      Age  Sex  ChestPainType  RestingBP  Cholesterol  FastingBS  RestingECG 0    172.0             0.0      0.0       3.0           0.0
0    40.0  1.0            2.0      140.0        289.0        0.0         1.0 1    156.0             0.0      1.0       2.0           1.0
1    49.0  0.0            3.0      160.0        180.0        0.0         1.0 2     98.0             0.0      0.0       3.0           0.0
2    37.0  1.0            2.0      130.0        283.0        0.0         2.0 3    108.0             1.0      1.5       2.0           1.0
3    48.0  0.0            4.0      138.0        214.0        0.0         1.0 4    122.0             0.0      0.0       3.0           0.0
4    54.0  1.0            3.0      150.0        195.0        0.0         1.0 ..     ...             ...      ...       ...           ...
..    ...  ...            ...        ...          ...        ...         ... 913  132.0             0.0      1.2       2.0           1.0
913  45.0  1.0            1.0      110.0        264.0        0.0         1.0 914  141.0             0.0      3.4       2.0           1.0
914  68.0  1.0            4.0      144.0        193.0        1.0         1.0 915  115.0             1.0      1.2       2.0           1.0
915  57.0  1.0            4.0      130.0        131.0        0.0         1.0 916  174.0             0.0      0.0       2.0           1.0
916  57.0  0.0            2.0      130.0        236.0        0.0         3.0 917  173.0             0.0      0.0       3.0           0.0
917  38.0  1.0            3.0      138.0        175.0        0.0         1.0
                                                                              [918 rows x 12 columns]
```

02 →

Data analysis and Variable selection

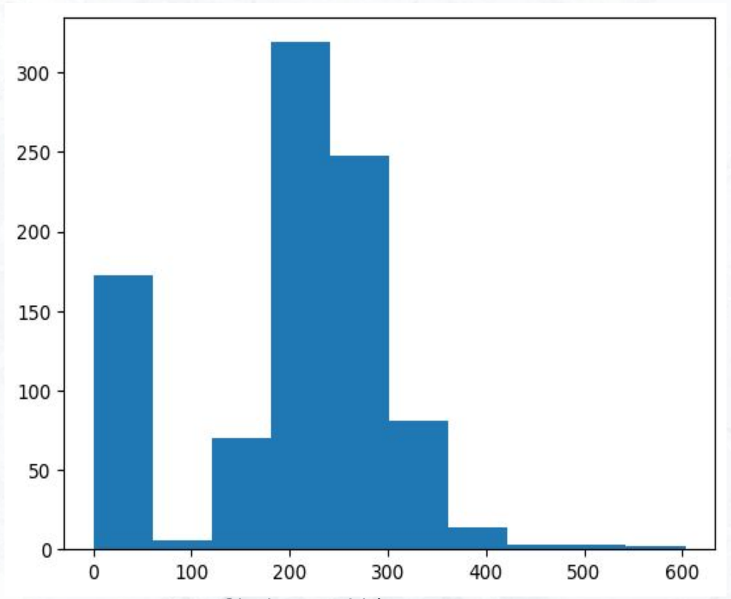
Data analysis



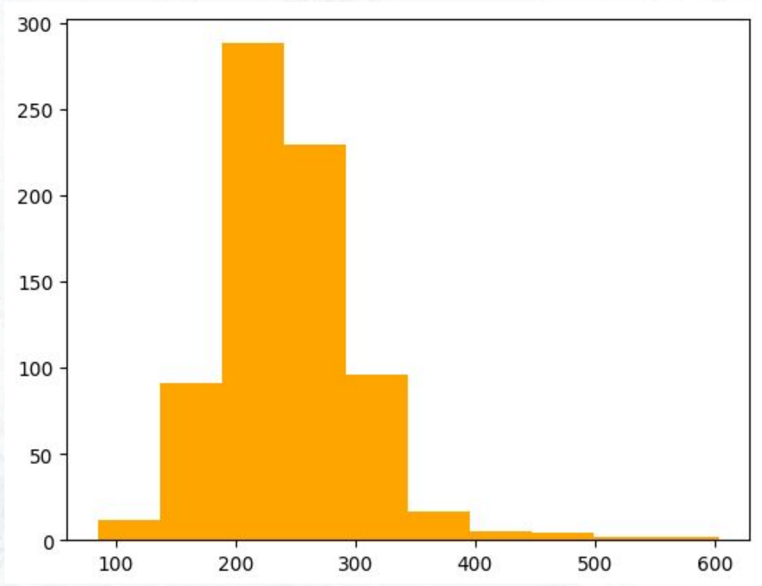


```
plt.figure(figsize=(10, 6))
plt.boxplot(data)
plt.title('Boxplot of Values')
plt.ylabel('Value')
plt.xticks(list(range(1,13)),list(data.columns),rotation = 90)
plt.show()
```

- Boxplot to observe outliers
- The boxplot shows the value 0 for cholesterol, which is <u>not possible</u> → Missing values

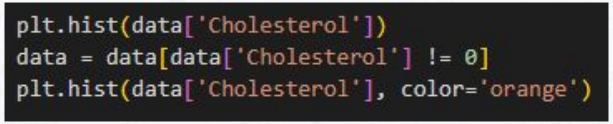


Cholesterol histogram



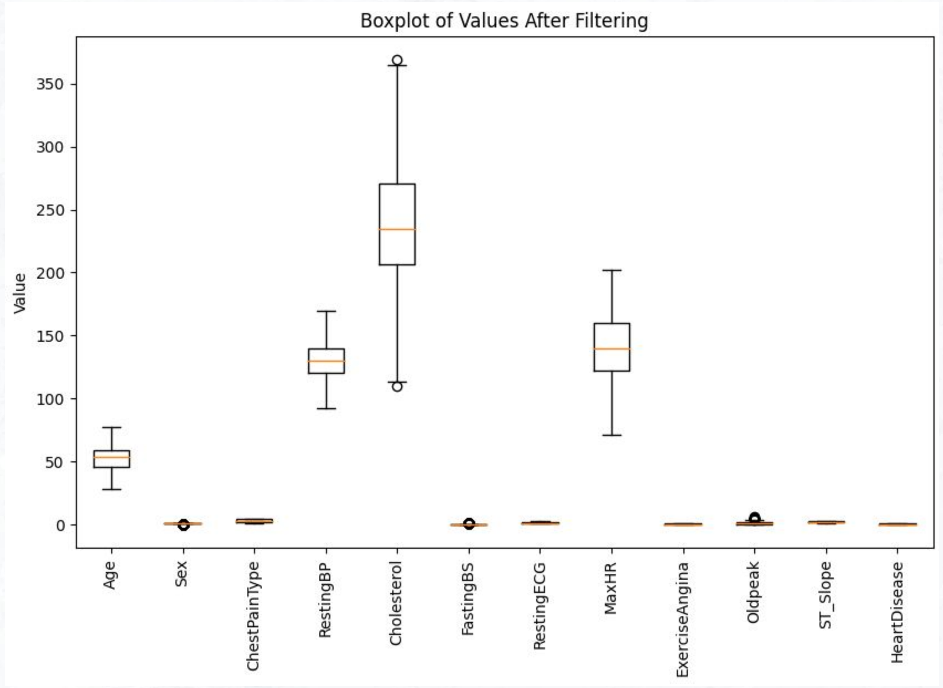
Cholesterol histogram without 0 values

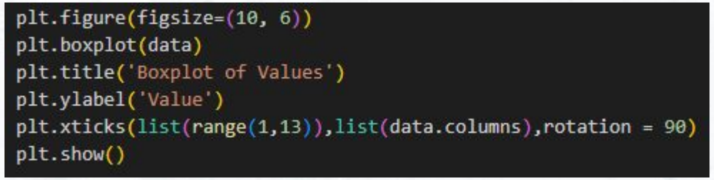
- The histogram shows how the values for cholesterol are distributed → many 0 values
- 0 values = missing values → biased results
- We eliminate the observations that have the value 0 for cholesterol



```
plt.hist(data['Cholesterol'])
data = data[data['Cholesterol'] != 0]
plt.hist(data['Cholesterol'], color='orange')
```

Removing outliers





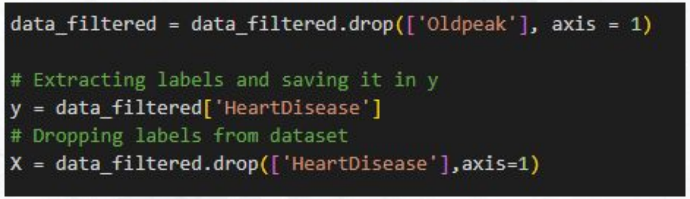
```
plt.figure(figsize=(10, 6))
plt.boxplot(data)
plt.title('Boxplot of Values')
plt.ylabel('Value')
plt.xticks(list(range(1,13)),list(data.columns),rotation = 90)
plt.show()
```

- We filtered the data to remove the outliers for the variables:
  - Age
  - Cholesterol
  - Resting Blood Pressure (Resting BP)
  - Maximum Heart Rate Achieved (MaxHR)

- After removing the outliers we have 704 observations

More data cleaning

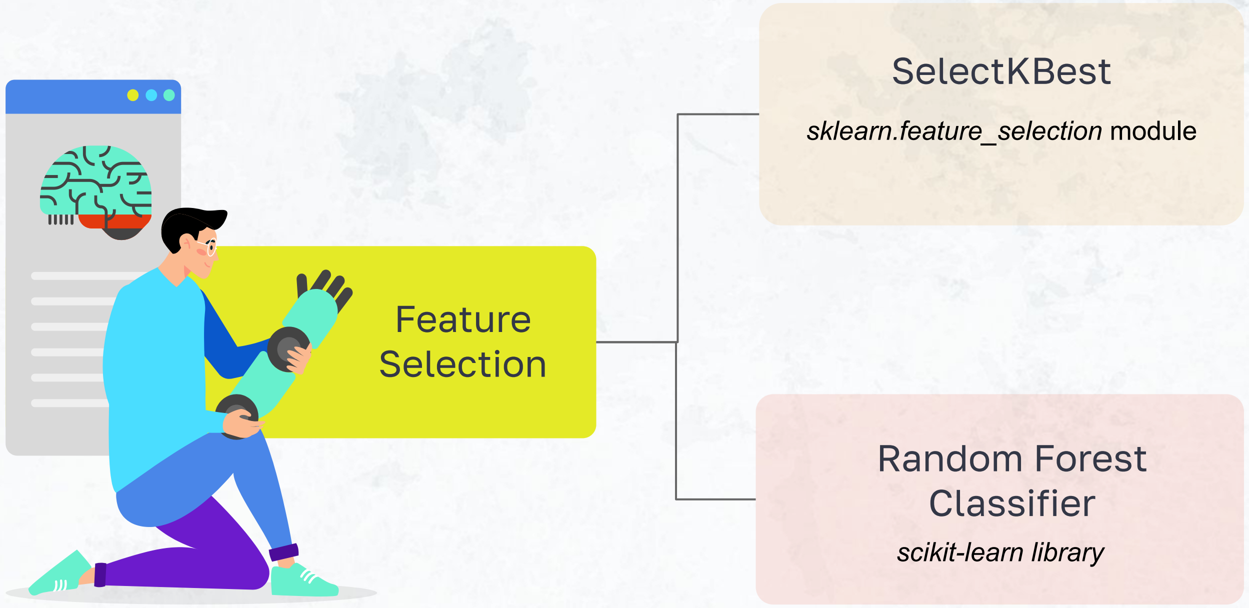
- Eliminate the OldPeak variable, since some models do not accept negative values
- Select the target variable: HeartDisease (0 or 1)



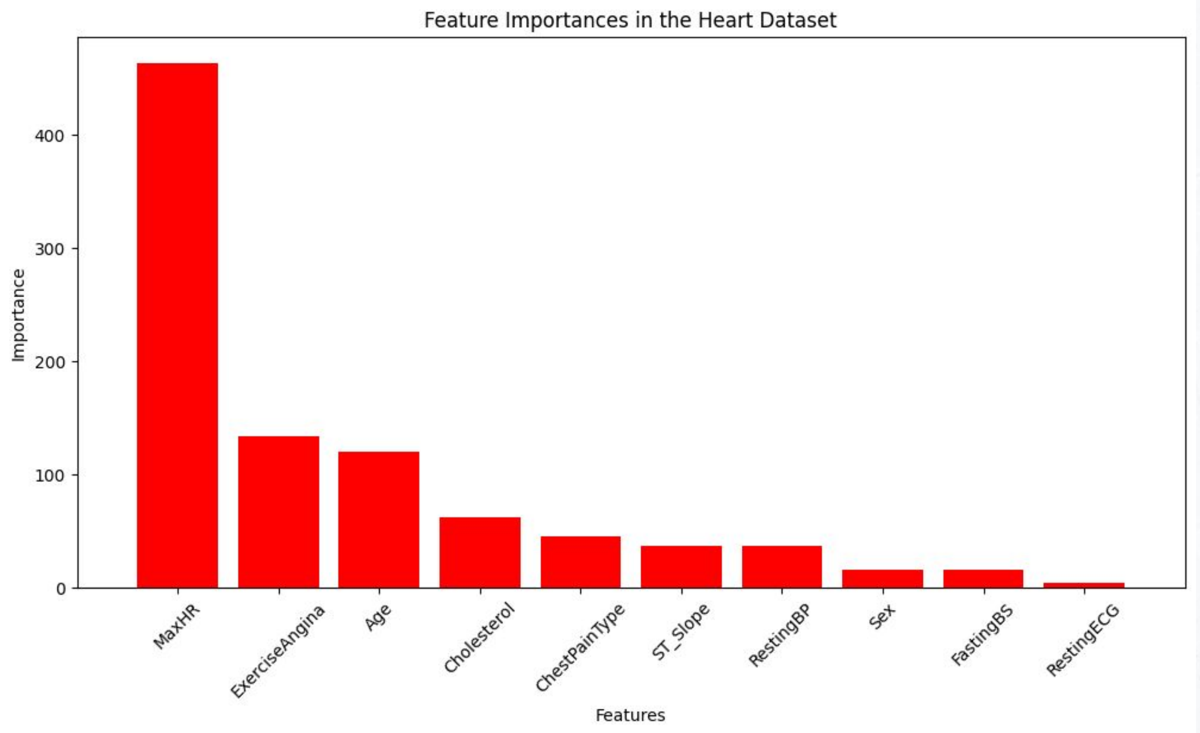
```
data_filtered = data_filtered.drop(['Oldpeak'], axis = 1)

# Extracting labels and saving it in y
y = data_filtered['HeartDisease']
# Dropping labels from dataset
X = data_filtered.drop(['HeartDisease'],axis=1)
```

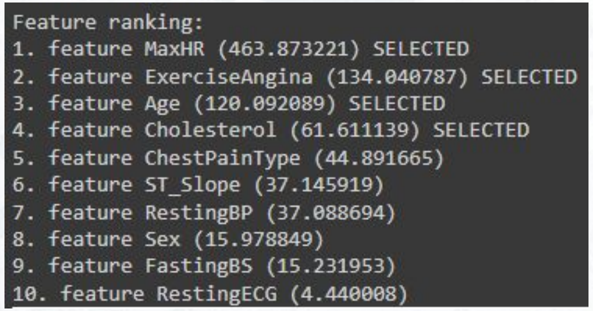
Feature selection

- Choosing the most relevant features for our models is important, as they will contribute to the predictive power
- We are using 2 methods to identify the features and we combine the top 2 from each of them, to be used in the models

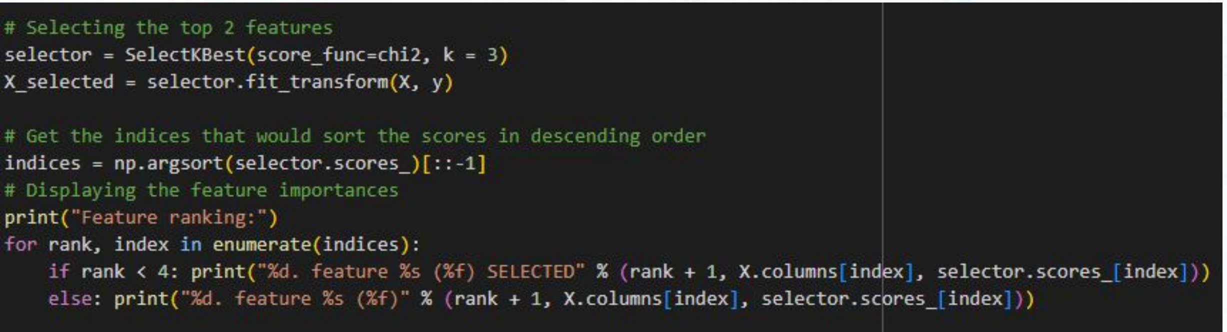


Select KBest Model



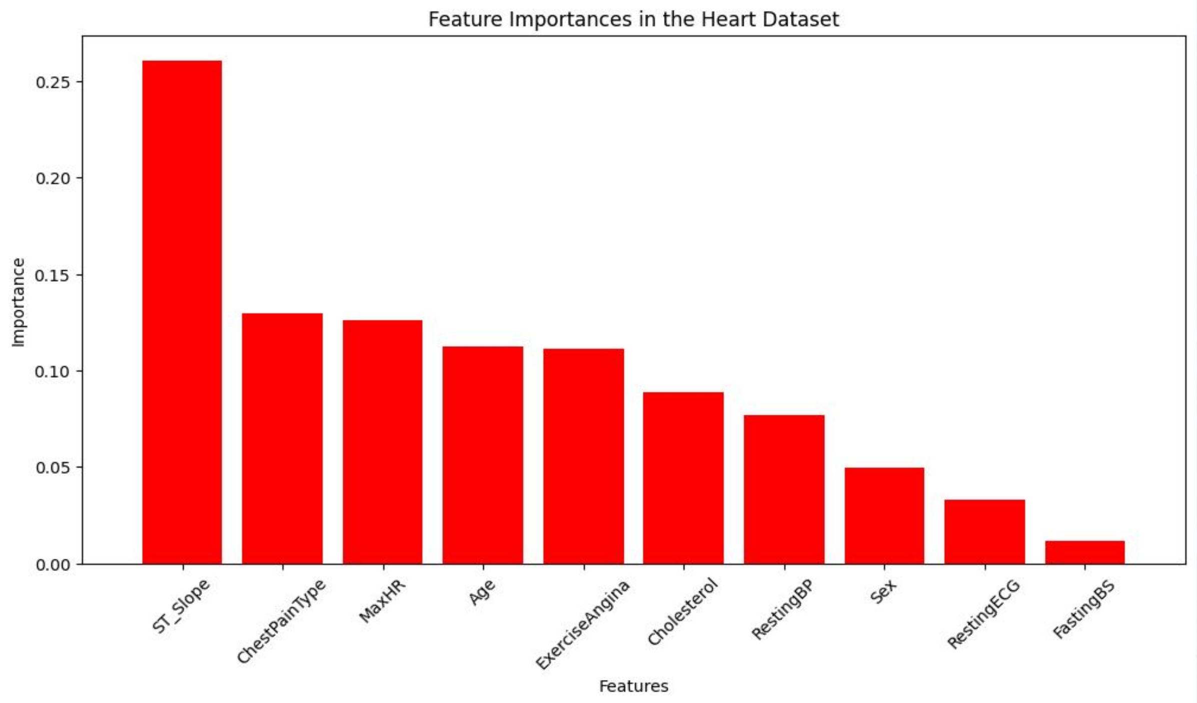


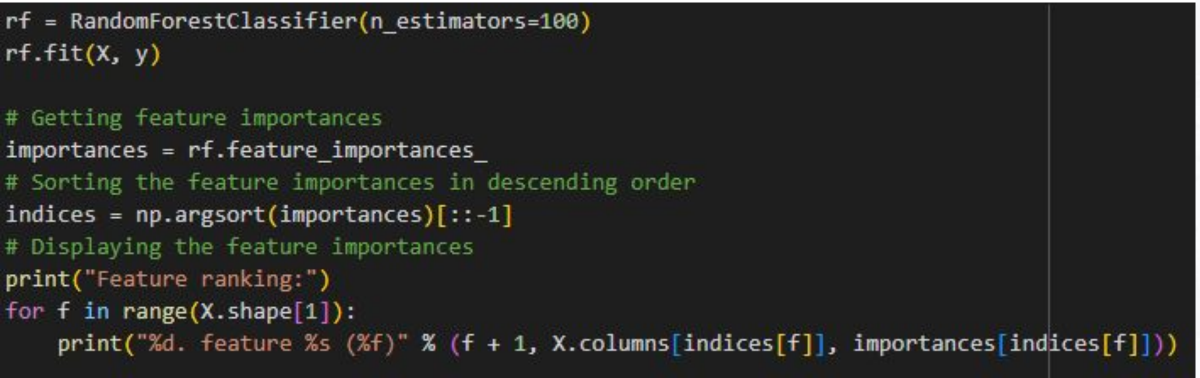
```
Feature ranking:
1. feature MaxHR (463.873221) SELECTED
2. feature ExerciseAngina (134.040787) SELECTED
3. feature Age (120.092089) SELECTED
4. feature Cholesterol (61.611139) SELECTED
5. feature ChestPainType (44.891665)
6. feature ST_Slope (37.145919)
7. feature RestingBP (37.088694)
8. feature Sex (15.978849)
9. feature FastingBS (15.231953)
10. feature RestingECG (4.440008)
```



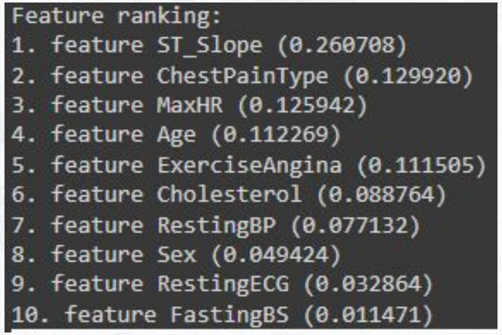
```
# Selecting the top 2 features
selector = SelectKBest(score_func=chi2, k = 3)
X_selected = selector.fit_transform(X, y)

# Get the indices that would sort the scores in descending order
indices = np.argsort(selector.scores_)[::-1]
# Displaying the feature importances
print("Feature ranking:")
for rank, index in enumerate(indices):
    if rank < 4: print("%d. feature %s (%f) SELECTED" % (rank + 1, X.columns[index], selector.scores_[index]))
    else: print("%d. feature %s (%f)" % (rank + 1, X.columns[index], selector.scores_[index]))
```

Random Forest Classifier





```
rf = RandomForestClassifier(n_estimators=100)
rf.fit(X, y)

# Getting feature importances
importances = rf.feature_importances_
# Sorting the feature importances in descending order
indices = np.argsort(importances)[::-1]
# Displaying the feature importances
print("Feature ranking:")
for f in range(X.shape[1]):
    print("%d. feature %s (%f)" % (f + 1, X.columns[indices[f]], importances[indices[f]]))
```



```
Feature ranking:
1. feature ST_Slope (0.260708)
2. feature ChestPainType (0.129920)
3. feature MaxHR (0.125942)
4. feature Age (0.112269)
5. feature ExerciseAngina (0.111505)
6. feature Cholesterol (0.088764)
7. feature RestingBP (0.077132)
8. feature Sex (0.049424)
9. feature RestingECG (0.032864)
10. feature FastingBS (0.011471)
```

Final Feature Selection

1. Maximum heart rate achieved
2. Exercise-induced angina
3. The slope of the peak exercise ST segment
4. Chest Pain Type

Select KBest

1. Maximum heart rate achieved
2. Exercise-induced angina
3. Age
4. Cholesterol

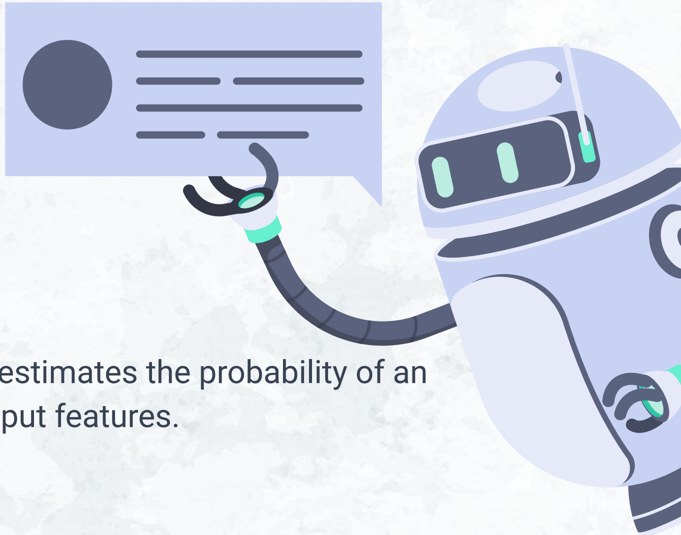
Random Forest Classifier

1. The slope of the peak exercise ST segment
2. Chest Pain Type
3. Maximum heart rate achieved
4. Age

03 →

ML Models: Logistic, KNN, Decision Tree, Random Forest

Types of models



(a) Logistic Regression

Logistic Regression is a statistical model used for binary classification that estimates the probability of an instance belonging to a **particular class based on a linear combination** of input features.

(b) K-Nearest Neighbors (KNN)

K-Nearest Neighbors is a non-parametric and instance-based machine learning algorithm that **classifies a data point based on the majority class of its k nearest neighbors** in the feature space.

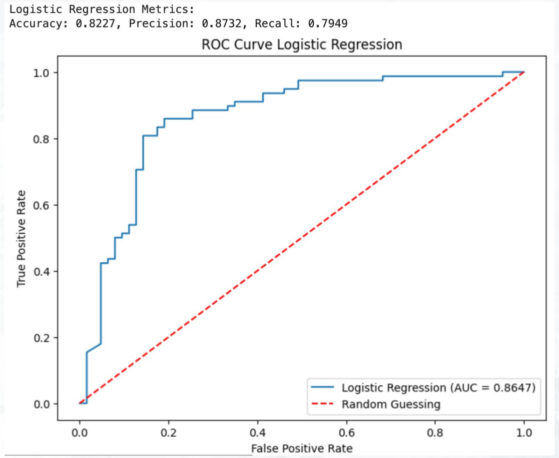
(c) Decision Tree

A Decision Tree is a **tree-like model** where **internal nodes** represent **decisions** based on features, branches represent possible outcomes, and leaf nodes represent the final classification or regression result.

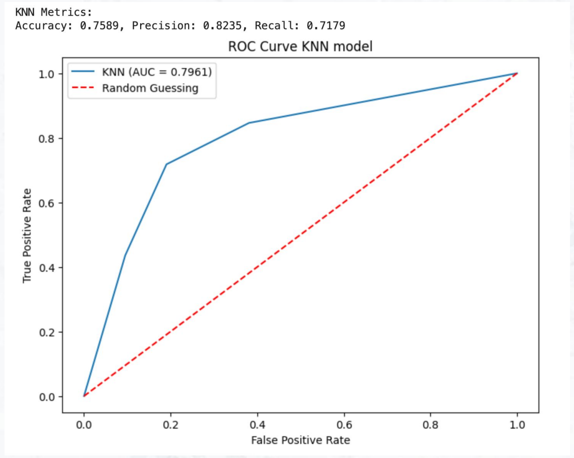
(d) Random Forest

Random Forest is an ensemble learning method that builds **multiple decision trees** during training and outputs the mode of the classes (classification) or the mean prediction (regression) of the individual trees as the final result.

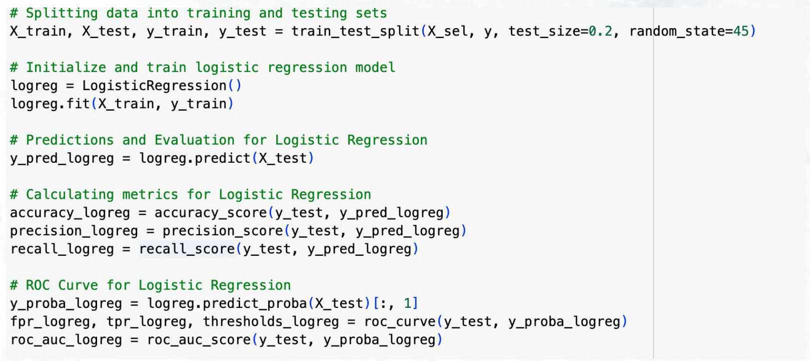
Logistic regression & KNN



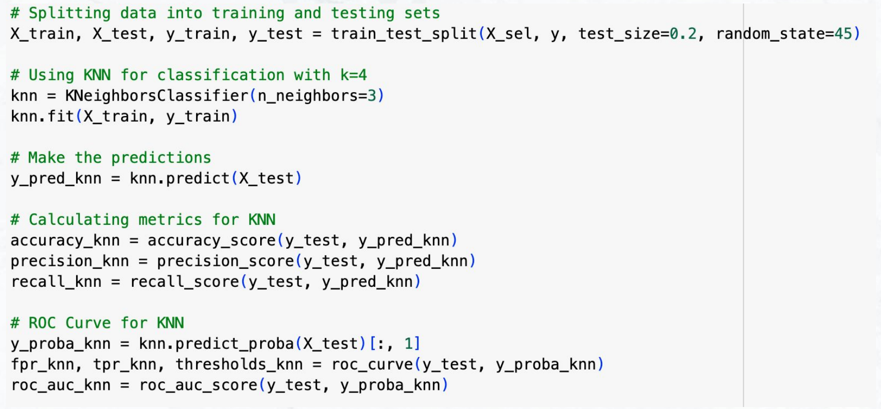
```
Logistic Regression Metrics:
Accuracy: 0.8227, Precision: 0.8732, Recall: 0.7949
```



```
KNN Metrics:
Accuracy: 0.7589, Precision: 0.8235, Recall: 0.7179
```

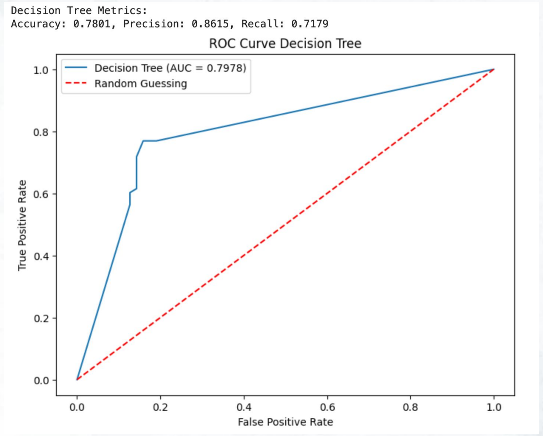


```
# Splitting data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_sel, y, test_size=0.2, random_state=45)

# Initialize and train logistic regression model
logreg = LogisticRegression()
logreg.fit(X_train, y_train)

# Predictions and Evaluation for Logistic Regression
y_pred_logreg = logreg.predict(X_test)

# Calculating metrics for Logistic Regression
accuracy_logreg = accuracy_score(y_test, y_pred_logreg)
precision_logreg = precision_score(y_test, y_pred_logreg)
recall_logreg = recall_score(y_test, y_pred_logreg)

# ROC Curve for Logistic Regression
y_proba_logreg = logreg.predict_proba(X_test)[:, 1]
fpr_logreg, tpr_logreg, thresholds_logreg = roc_curve(y_test, y_proba_logreg)
roc_auc_logreg = roc_auc_score(y_test, y_proba_logreg)
```

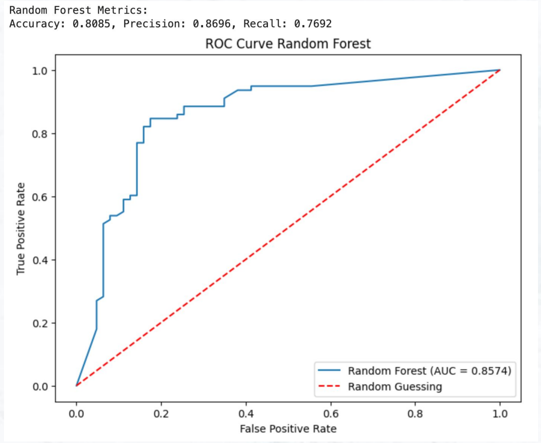


```
# Splitting data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_sel, y, test_size=0.2, random_state=45)

# Using KNN for classification with k=4
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

# Make the predictions
y_pred_knn = knn.predict(X_test)

# Calculating metrics for KNN
accuracy_knn = accuracy_score(y_test, y_pred_knn)
precision_knn = precision_score(y_test, y_pred_knn)
recall_knn = recall_score(y_test, y_pred_knn)

# ROC Curve for KNN
y_proba_knn = knn.predict_proba(X_test)[:, 1]
fpr_knn, tpr_knn, thresholds_knn = roc_curve(y_test, y_proba_knn)
roc_auc_knn = roc_auc_score(y_test, y_proba_knn)
```

Decision Tree & Random Forest



```
Decision Tree Metrics:
Accuracy: 0.7801, Precision: 0.8615, Recall: 0.7179
```



```
Random Forest Metrics:
Accuracy: 0.8085, Precision: 0.8696, Recall: 0.7692
```

```
# Initialize and train the decision tree classifier
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)

# Predictions and Evaluation for Decision Tree
y_pred_dt = clf.predict(X_test)

# Calculating metrics for Decision Tree
accuracy_dt = accuracy_score(y_test, y_pred_dt)
precision_dt = precision_score(y_test, y_pred_dt)
recall_dt = recall_score(y_test, y_pred_dt)

# ROC Curve for Decision Tree
y_proba_dt = clf.predict_proba(X_test)[:, 1]
fpr_dt, tpr_dt, thresholds_dt = roc_curve(y_test, y_proba_dt)
roc_auc_dt = roc_auc_score(y_test, y_proba_dt)
```

```
# Initialize and train the Random Forest classifier
rf_clf = RandomForestClassifier(random_state=42)
rf_clf.fit(X_train, y_train)

# Predictions and Evaluation for Random Forest
y_pred_rf = rf_clf.predict(X_test)

# Calculating metrics for Random Forest
accuracy_rf = accuracy_score(y_test, y_pred_rf)
precision_rf = precision_score(y_test, y_pred_rf)
recall_rf = recall_score(y_test, y_pred_rf)

# ROC Curve for Random Forest
y_proba_rf = rf_clf.predict_proba(X_test)[:, 1]
fpr_rf, tpr_rf, thresholds_rf = roc_curve(y_test, y_proba_rf)
roc_auc_rf = roc_auc_score(y_test, y_proba_rf)
```

Final Comparison Table

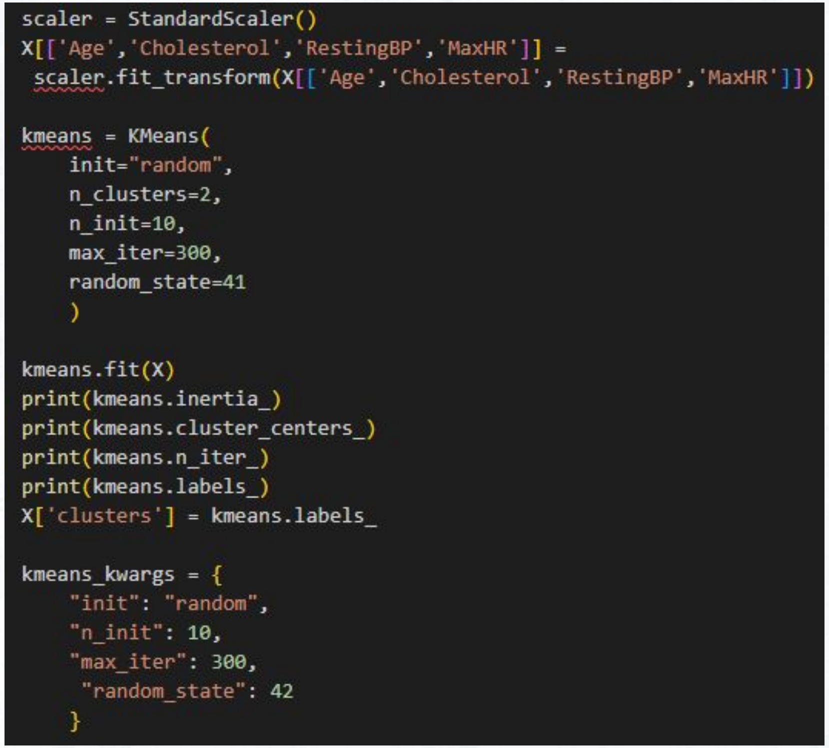
Model	*Accuracy*	*Precision*	*Recall*	*ROC-AUC*
Logistic regression	**0.8227**	**0.8732**	**0.7949**	**0.8647**
KNN	0.7801	0.8235	0.7179	0.7961
Decision Tree	0.7801	0.8615	0.7179	0.7978
Random Forest	0.8085	0.8696	0.7692	0.8574

- The **logistic regression** outperforms the other models taking into account all the test statistics

04 →

K-means clustering

Algorithm

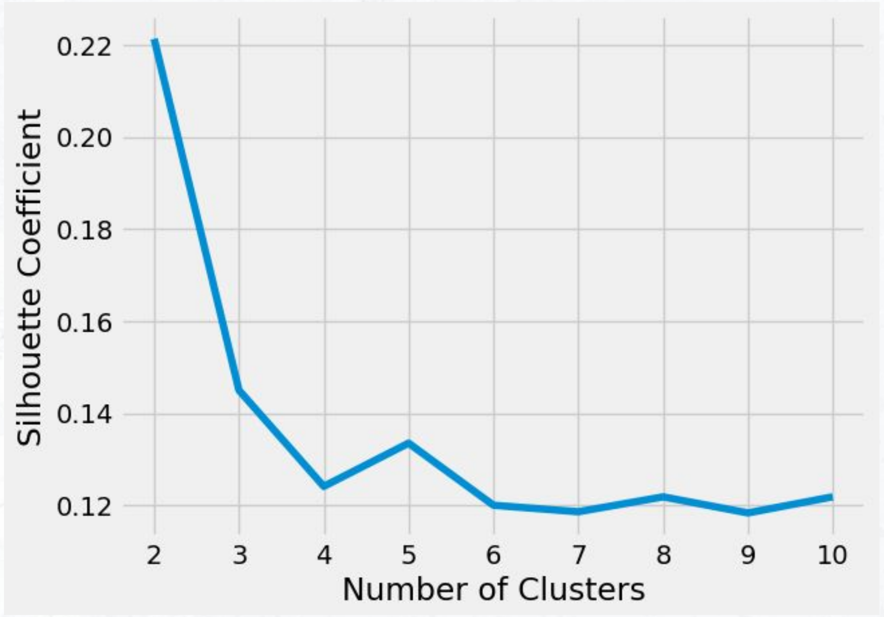


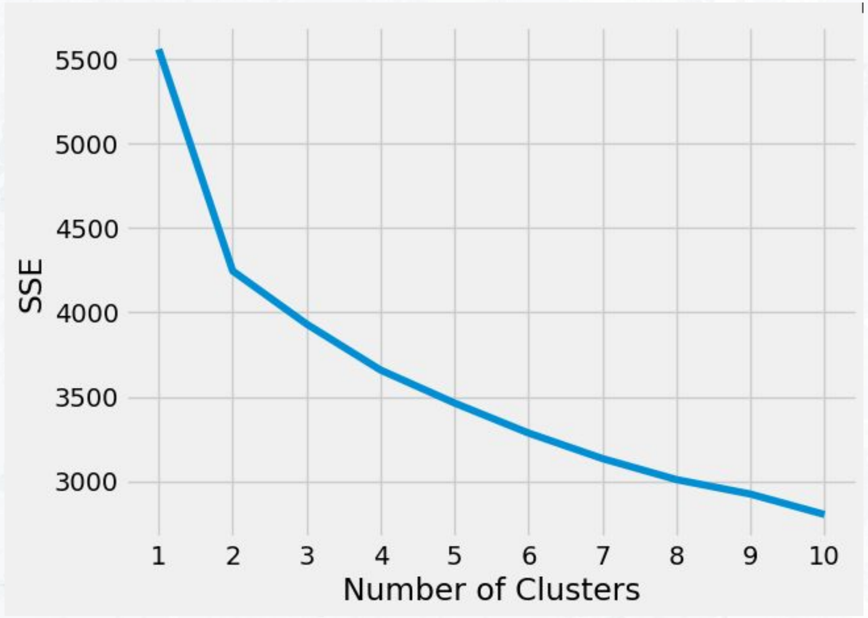
```
scaler = StandardScaler()
X[['Age','Cholesterol','RestingBP','MaxHR']] =
 scaler.fit_transform(X[['Age','Cholesterol','RestingBP','MaxHR']])

kmeans = KMeans(
    init="random",
    n_clusters=2,
    n_init=10,
    max_iter=300,
    random_state=41
    )

kmeans.fit(X)
print(kmeans.inertia_)
print(kmeans.cluster_centers_)
print(kmeans.n_iter_)
print(kmeans.labels_)
X['clusters'] = kmeans.labels_

kmeans_kwargs = {
    "init": "random",
    "n_init": 10,
    "max_iter": 300,
     "random_state": 42
    }
```

- Step 1: Choose number of K
- Step 2: Randomly initialize centroids
- Steps 3-4: Iterative process called expectation-maximization
    - Expectation assigns each point to closest centroid
    - Maximization computes the new centroid mean of each cluster
    - Repeat until centroid positions do not change

Choosing K

- To choose the optimal numbers of K, we used Silhouette and Elbow Method.
- In both cases, K=2 is optimal





Cluster analysis

Cluster 0 is composed of 295 men and 58 women that are old on average, with higher than average Resting BP, Cholesterol, Fasting BS, Resting ECG, and Exercise Angina leading to large proportion of them having heart disease.

Cluster 1, on the other hand, are a younger group of 243 men and 108 women with lower than average Resting BP, Cholesterol, Fasting BS, Resting ECG, and Exercise Angina leading to a smaller proportion of them having heart disease.

Mean	Age	Sex	Chest Pain Type	Resting BP	Cholesterol	Fasting BS	Resting ECG	Max HR	Exercise Angina	ST Slope	**Heart Disease**
X (704)	0	0.764	3.149	0	0	0.161	1.634	0	0.381	2.420	**0.472**
Cluster 0 (353)	0.507	0.836	3.737	0.238	0.080	0.227	1.782	-0.665	0.657	2.108	**0.751**
Cluster 1 (351)	-0.509	0.692	2.558	-0.239	-0.080	0.094	1.484	0.669	0.103	2.735	**0.191**

Thank you!