PROJECT REPORT ON

# Heart Disease Classification Using Supervised Machine Learning Algorithms

Team Members:

Mihir Mishra - 20UCS118

Lakshya Agrawal - 20UCC058

Raghav Mandhana - 20UCC084

Taj Basha - 20UCC109

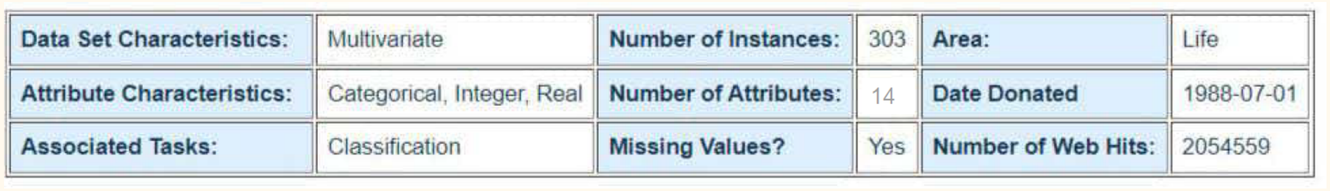
## Introduction

Supervised learning, also known as supervised machine learning, is a subcategory of machine learning and artificial intelligence. It is defined by its use of labeled datasets to train algorithms to classify data or predict outcomes accurately. As input data is fed into the model, it adjusts its weights until the model has been fitted appropriately, which occurs as part of the cross-validation process. Supervised learning helps organizations solve a variety of real-world problems at scale, such as classifying spam in a separate folder from your inbox.

In this project, we have used various machine learning algorithms, such as support vector machines (SVM), random forest, k-nearest neighbors (KNN), and AdaBoost, to classify whether or not a person has heart disease based on various characteristics, or features, of the person. The project is implemented in  python and it reads  a dataset containing the features and target variable, performs data exploration and visualization, and prepares the data for machine learning. The code then trains and evaluates the machine learning models using cross-validation and various evaluation metrics, and the results of these models are discussed in the report.

## Dataset

The dataset for this project is collected from the Heart Disease Data Set from the University of California-Irvine (UCI) Machine Learning Repository. (Heart Disease Cleveland UCI).



| | | | | | |
|---|---|---|---|---|---|
| **Data Set Characteristics:** | Multivariate | **Number of Instances:** | 303 | **Area:** | Life |
| **Attribute Characteristics:** | Categorical, Integer, Real | **Number of Attributes:** | 14 | **Date Donated** | 1988-07-01 |
| **Associated Tasks:** | Classification | **Missing Values?** | Yes | **Number of Web Hits:** | 2054559 |

### Attribute Information:

There are 14 attributes in the dataset.

The attributes are age, sex, chest pain(cp), resting blood pressure (trestbps) , serum cholestoral (chol), fasting blood sugar (fbs) , resting electrocardiographic results (restecg) , maximum heart rate achieved (thalach) , exercise induced angina (exang) , ST depression induced by exercise relative to rest (oldpeak) , the slope of the peak exercise ST segment (slope) , number of major vessels (ca) , thal, target.

## Attribute Description:

1. age: age in years
2. sex: sex (1 = male; 0 = female)
3. cp: chest pain type (Values= 0,1,2,3)

   Value 0 = typical angina; Value 1 = atypical angina;
   Value 2 = non-anginal pain ; Value3: asymptomatic
4. trestbps: resting blood pressure (in mm Hg on admission to the hospital)
5. chol: serum cholesterol in mg/dl
6. fbs: (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
7. restecg: resting electrocardiographic results (Values=0,1,2)

   Value 0: normal
   Value 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV)
   Value 2: showing probable or definite left ventricular hypertrophy by Estes' criteria
8. thalach: maximum heart rate achieved
9. exang: exercise induced angina (1 = yes; 0 = no)
10. oldpeak = ST depression induced by exercise relative to rest
11. slope: the slope of the peak exercise ST segment(Values=0,1,2)

    Value 0 = upsloping ; Value 1 = flat ; Value 2 = downsloping
12. ca: number of major vessels (0-3) colored by fluoroscopy
13. thal: 0 = normal; 1 = fixed defect; 2 = reversible defect and the label Target variable: condition: 0 = no disease, 1 = disease

14. Target: Condition (0 = no disease, 1 = disease)

### Categorical attributes are :

"cp","fbs", "restecg", "ca", "thal"

### Numeric attributes are :

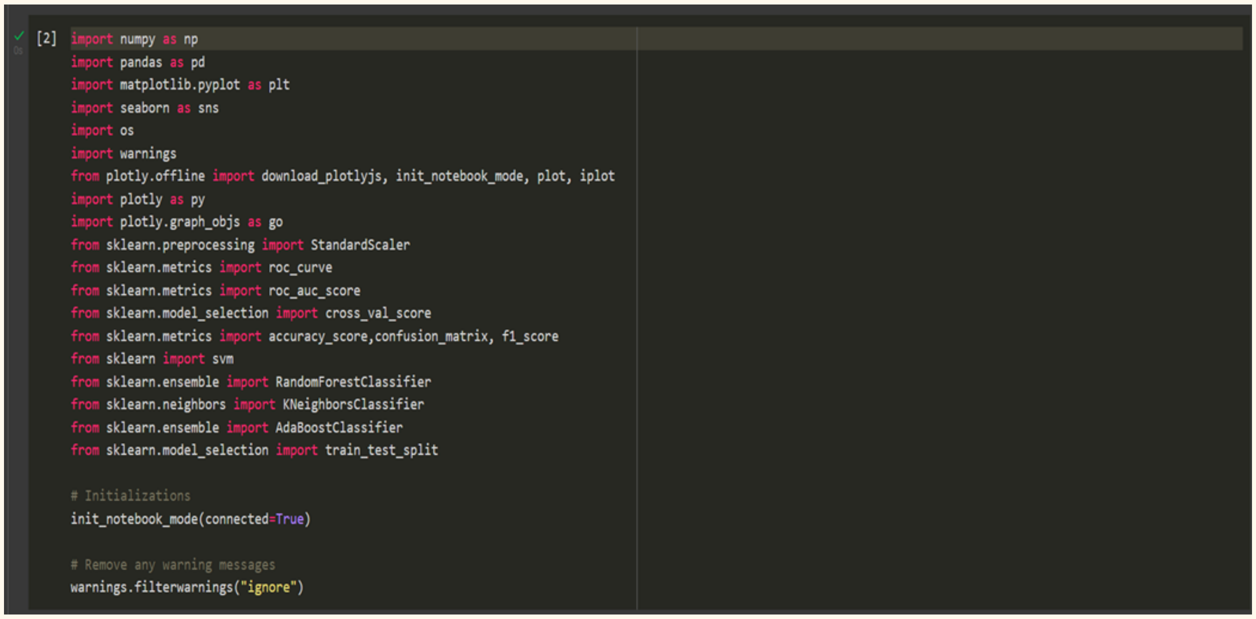
"age", "sex", "trestbps", "chol", "thalach", "exang", "oldpeak", "slope"

## Working of the Model

- The problem was understood thoroughly, and it was decided to choose the most suitable Classifier based on their F1-score and Cross Validation accuracy.
- The code imports several libraries, including NumPy, pandas, matplotlib, seaborn, and several modules from scikit-learn.
- The code then counts the number of observations in the "target" column that are equal to 0 or 1 and prints the results. This is being done to check the balance of the target classes in the dataset.
- As the values of "target" variable are split almost equally in the dataset ROC curves are used to summarize the trade-off between the true positive rate and false positive rate for a classifier.
- We have used "Label Encoding" to give descriptive labels to data.
- We have used several pre-processing techniques on the data, including scaling using the 'StandardScaler' class, feature selection using the SelectKBest() function and splitting the data set into testing and training dataset. All of this is done using the scikit-learn library.
- A custom function is defined that performs cross-validation on the machine learning model.
- Different Classifiers like K-Nearest Neighbour Classifier, Random Forest Classification and AdaBoost Classification algorithms, Support Vector Machine are used to fit and predict the model. Heat Map and ROC curves are printed along with the accuracy score for each classifier.

- Finally, different graphs are made based on Cross Validation Accuracy, F1 Score and AUC Score in order to compare different classifiers.
- Finally, the Classifier with the most F1 Score and Cross Validation accuracy is taken and thus the values are measured.

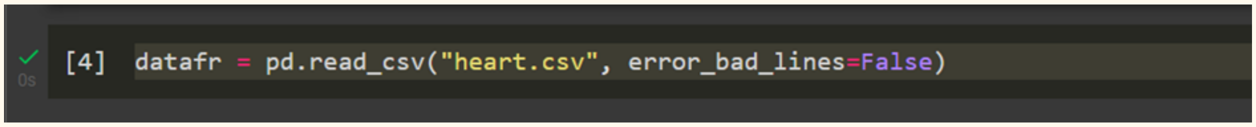
## Libraries Used



```
[2] import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    import seaborn as sns
    import os
    import warnings
    from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
    import plotly as py
    import plotly.graph_objs as go
    from sklearn.preprocessing import StandardScaler
    from sklearn.metrics import roc_curve
    from sklearn.metrics import roc_auc_score
    from sklearn.model_selection import cross_val_score
    from sklearn.metrics import accuracy_score,confusion_matrix, f1_score
    from sklearn import svm
    from sklearn.ensemble import RandomForestClassifier
    from sklearn.neighbors import KNeighborsClassifier
    from sklearn.ensemble import AdaBoostClassifier
    from sklearn.model_selection import train_test_split

    # Initializations
    init_notebook_mode(connected=True)

    # Remove any warning messages
    warnings.filterwarnings("ignore")
```

## Importing Dataset

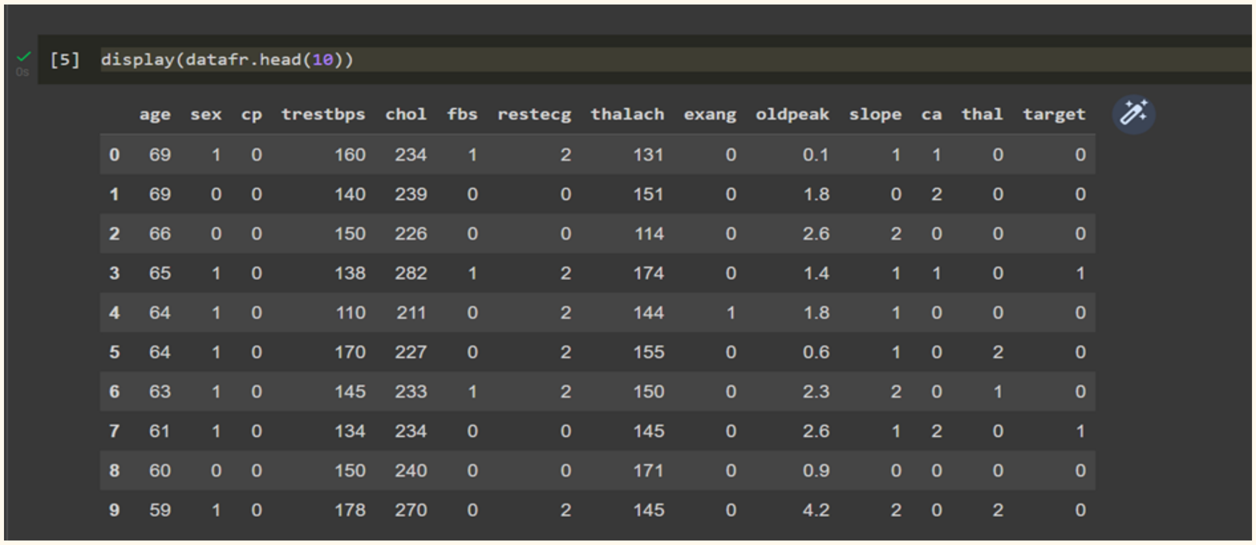
We have used the pandas (pd) library function to read the dataset.



```
[4] datafr = pd.read_csv("heart.csv", error_bad_lines=False)
```

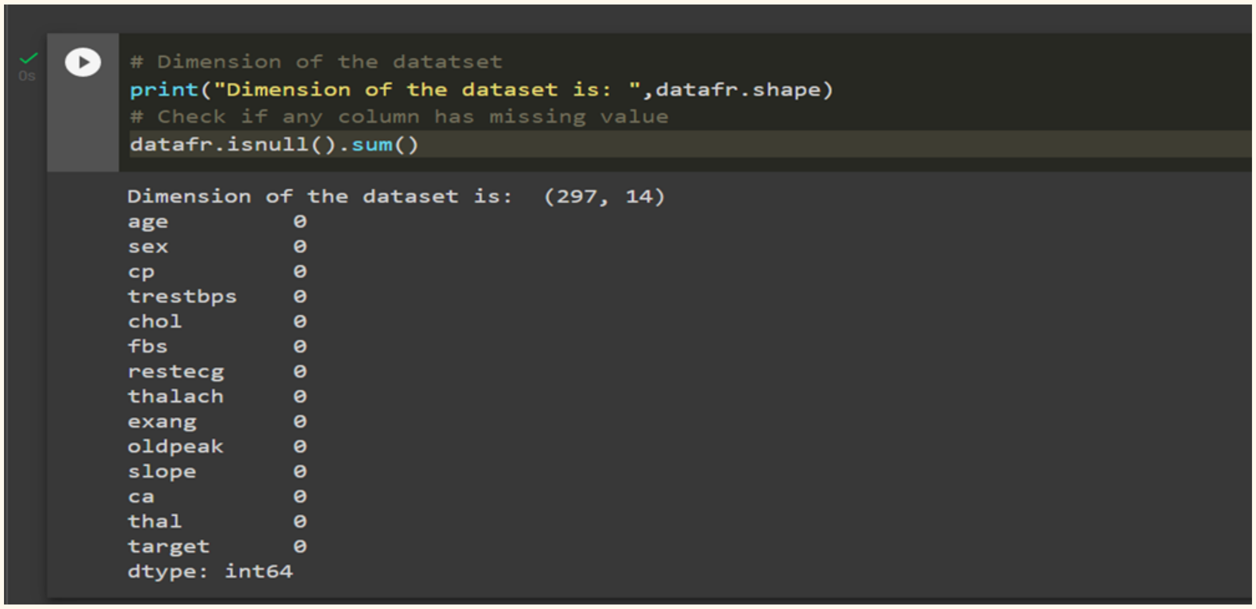
## Preview of the Data

### Dimension of the Dataset



```
[5] display(datafr.head(10))
```

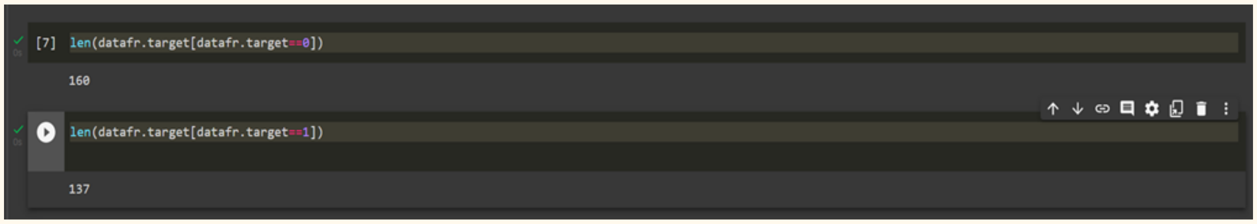
| | age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca | thal | target |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 69 | 1 | 0 | 160 | 234 | 1 | 2 | 131 | 0 | 0.1 | 1 | 1 | 0 | 0 |
| 1 | 69 | 0 | 0 | 140 | 239 | 0 | 0 | 151 | 0 | 1.8 | 0 | 2 | 0 | 0 |
| 2 | 66 | 0 | 0 | 150 | 226 | 0 | 0 | 114 | 0 | 2.6 | 2 | 0 | 0 | 0 |
| 3 | 65 | 1 | 0 | 138 | 282 | 1 | 2 | 174 | 0 | 1.4 | 1 | 1 | 0 | 1 |
| 4 | 64 | 1 | 0 | 110 | 211 | 0 | 2 | 144 | 1 | 1.8 | 1 | 0 | 0 | 0 |
| 5 | 64 | 1 | 0 | 170 | 227 | 0 | 2 | 155 | 0 | 0.6 | 1 | 0 | 2 | 0 |
| 6 | 63 | 1 | 0 | 145 | 233 | 1 | 2 | 150 | 0 | 2.3 | 2 | 0 | 1 | 0 |
| 7 | 61 | 1 | 0 | 134 | 234 | 0 | 0 | 145 | 0 | 2.6 | 1 | 2 | 0 | 1 |
| 8 | 60 | 0 | 0 | 150 | 240 | 0 | 0 | 171 | 0 | 0.9 | 0 | 0 | 0 | 0 |
| 9 | 59 | 1 | 0 | 178 | 270 | 0 | 2 | 145 | 0 | 4.2 | 2 | 0 | 2 | 0 |



```
# Dimension of the datatset
print("Dimension of the dataset is: ",datafr.shape)
# Check if any column has missing value
datafr.isnull().sum()
```

```
Dimension of the dataset is:  (297, 14)
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

## Observing the target column



```
[7] len(datafr.target[datafr.target==0])

160

len(datafr.target[datafr.target==1])

137
```

They are roughly the same so we can use the ROC curve as it summarizes the trade-off between the true positive rate and false positive rate for a predictive model.

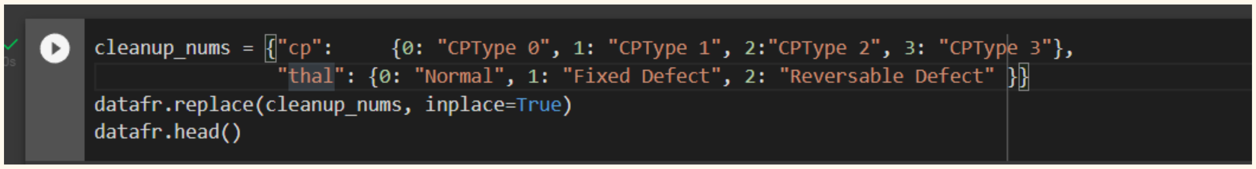
## Replacing Labels

In the cp column we replace:

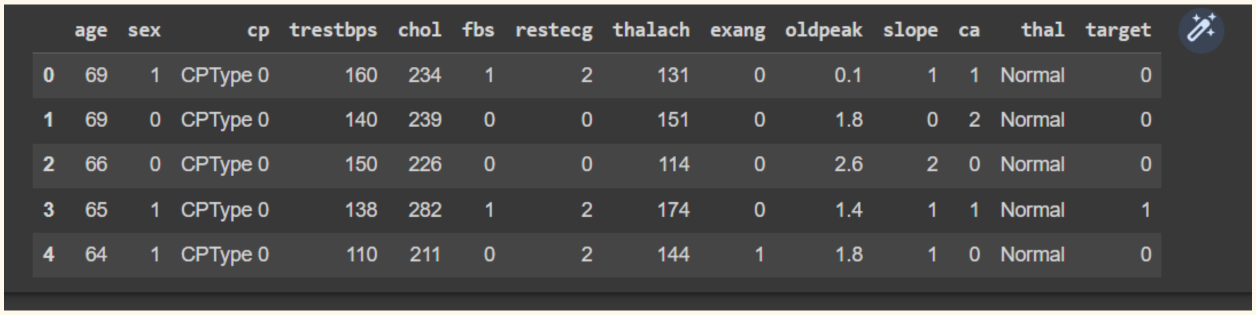
0 ="CPType 0", 1: "CPType 1", 2:"CPType 2", 3: "CPType  3"

In the thal column we replace:

0: "Normal", 1: "Fixed Defect", 2: "Reversable Defect"



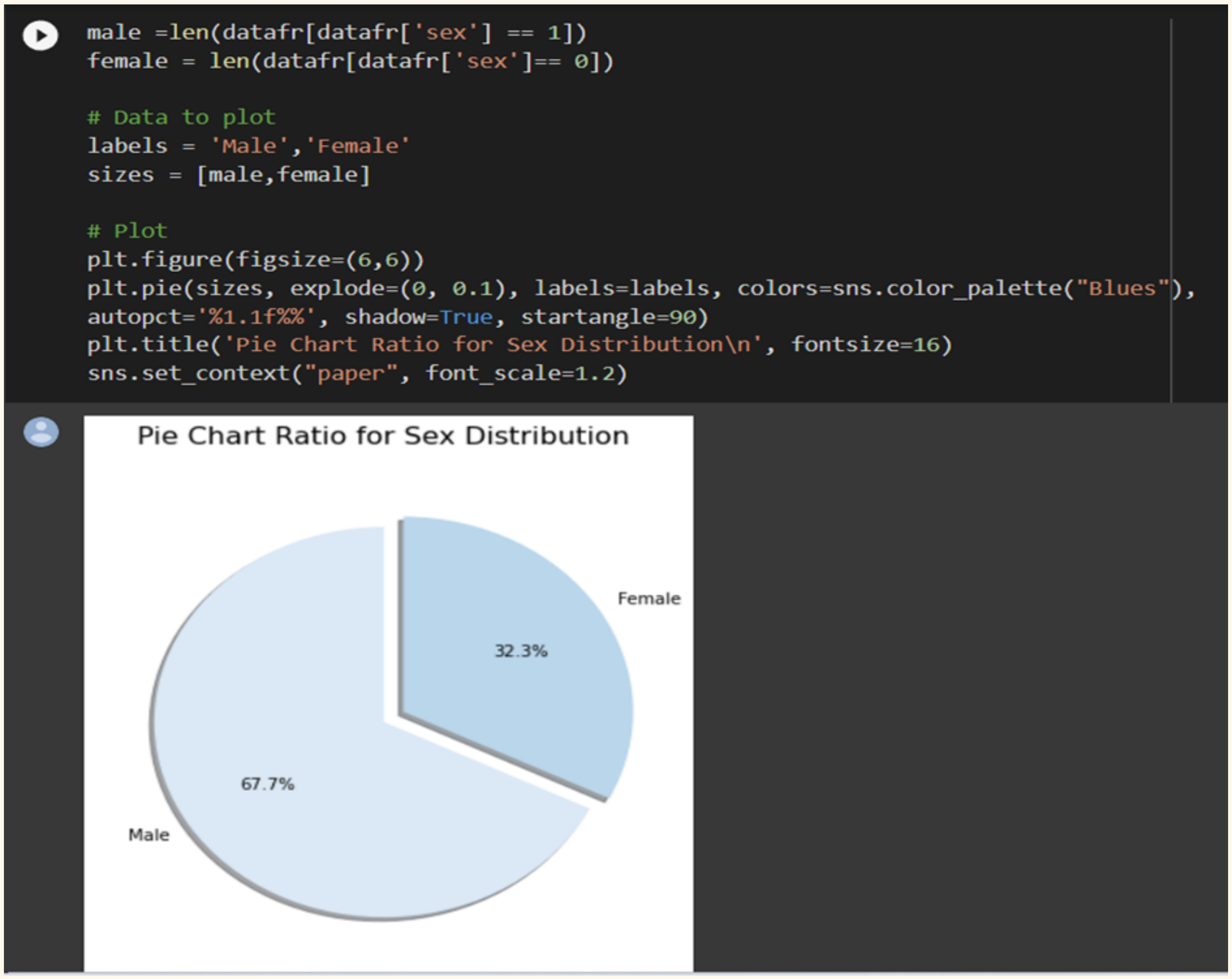
```
cleanup_nums = {"cp":     {0: "CPType 0", 1: "CPType 1", 2:"CPType 2", 3: "CPType 3"},
                "thal": {0: "Normal", 1: "Fixed Defect", 2: "Reversable Defect" }}
datafr.replace(cleanup_nums, inplace=True)
datafr.head()
```



| | age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca | thal | target |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 69 | 1 | CPType 0 | 160 | 234 | 1 | 2 | 131 | 0 | 0.1 | 1 | 1 | Normal | 0 |
| 1 | 69 | 0 | CPType 0 | 140 | 239 | 0 | 0 | 151 | 0 | 1.8 | 0 | 2 | Normal | 0 |
| 2 | 66 | 0 | CPType 0 | 150 | 226 | 0 | 0 | 114 | 0 | 2.6 | 2 | 0 | Normal | 0 |
| 3 | 65 | 1 | CPType 0 | 138 | 282 | 1 | 2 | 174 | 0 | 1.4 | 1 | 1 | Normal | 1 |
| 4 | 64 | 1 | CPType 0 | 110 | 211 | 0 | 2 | 144 | 1 | 1.8 | 1 | 0 | Normal | 0 |

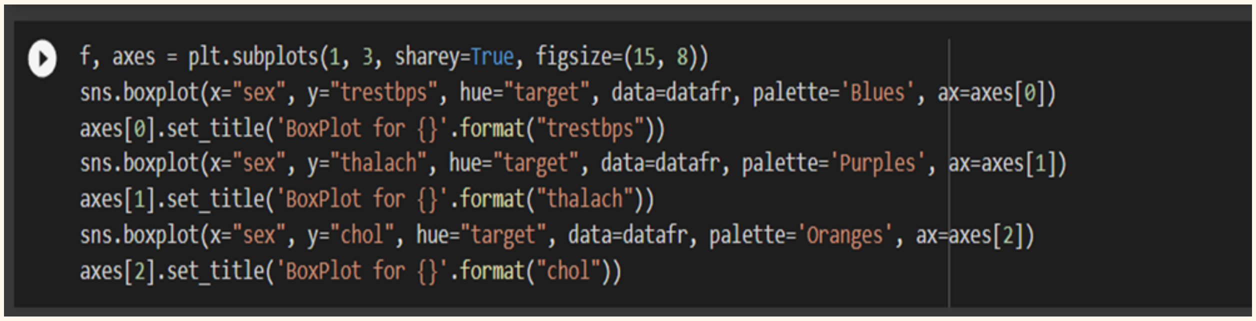
## Graphical Representation

### Pie Chart for sex distribution -

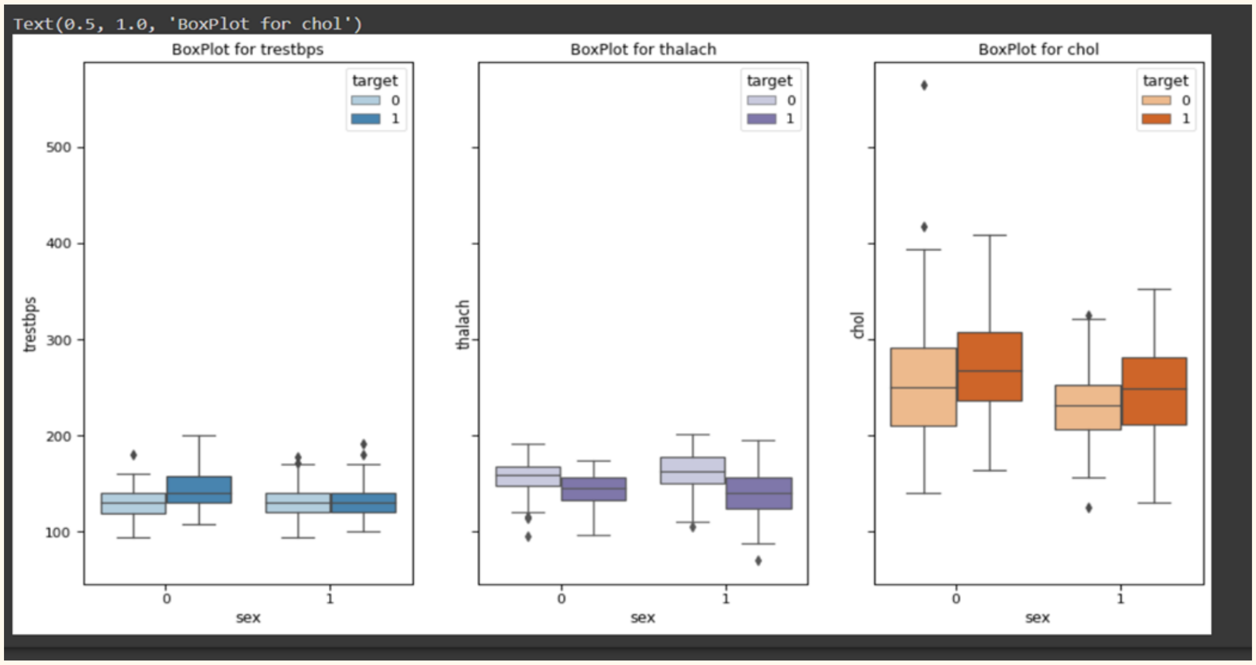


```
male =len(datafr[datafr['sex'] == 1])
female = len(datafr[datafr['sex']== 0])

# Data to plot
labels = 'Male','Female'
sizes = [male,female]

# Plot
plt.figure(figsize=(6,6))
plt.pie(sizes, explode=(0, 0.1), labels=labels, colors=sns.color_palette("Blues"),
autopct='%1.1f%%', shadow=True, startangle=90)
plt.title('Pie Chart Ratio for Sex Distribution\n', fontsize=16)
sns.set_context("paper", font_scale=1.2)
```

### Box plot for resting blood pressure, cholesterol, and maximum heart rate-

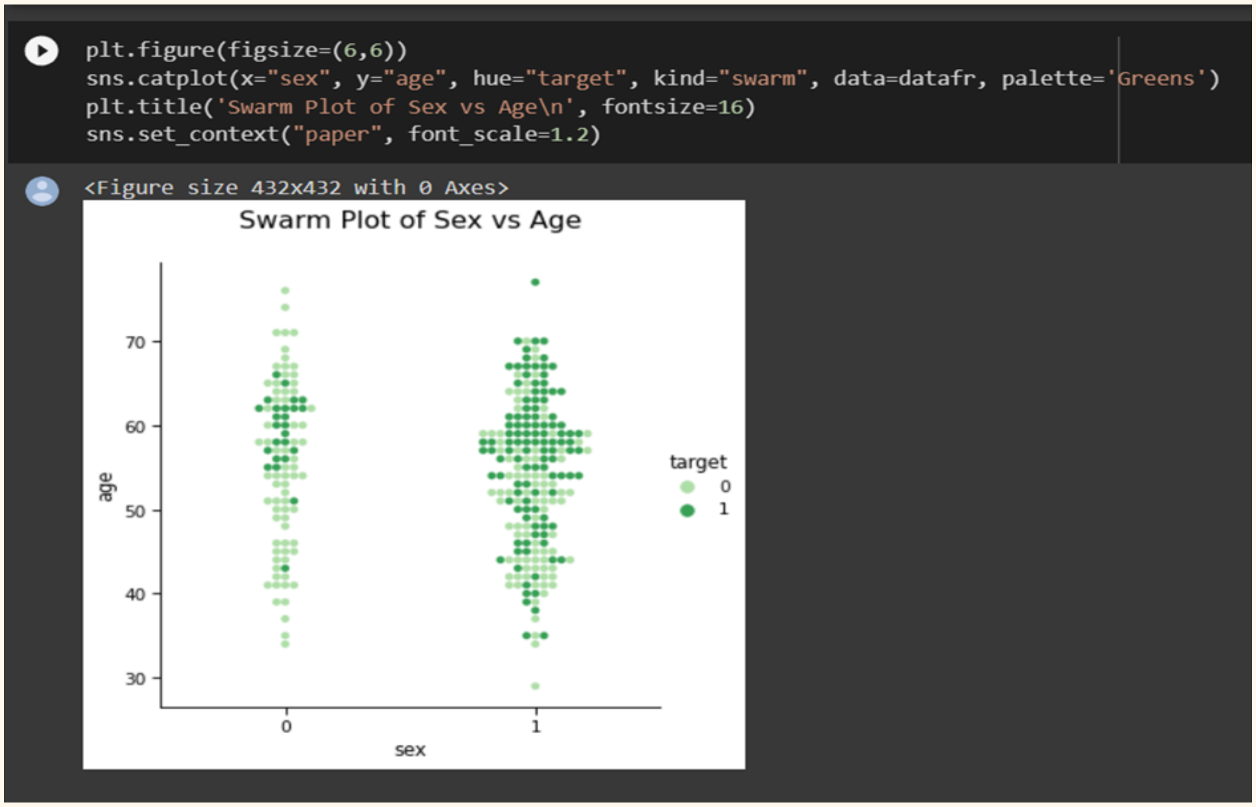


```
f, axes = plt.subplots(1, 3, sharey=True, figsize=(15, 8))
sns.boxplot(x="sex", y="trestbps", hue="target", data=datafr, palette='Blues', ax=axes[0])
axes[0].set_title('BoxPlot for {}'.format("trestbps"))
sns.boxplot(x="sex", y="thalach", hue="target", data=datafr, palette='Purples', ax=axes[1])
axes[1].set_title('BoxPlot for {}'.format("thalach"))
sns.boxplot(x="sex", y="chol", hue="target", data=datafr, palette='Oranges', ax=axes[2])
axes[2].set_title('BoxPlot for {}'.format("chol"))
```



```
Text(0.5, 1.0, 'BoxPlot for chol')
```

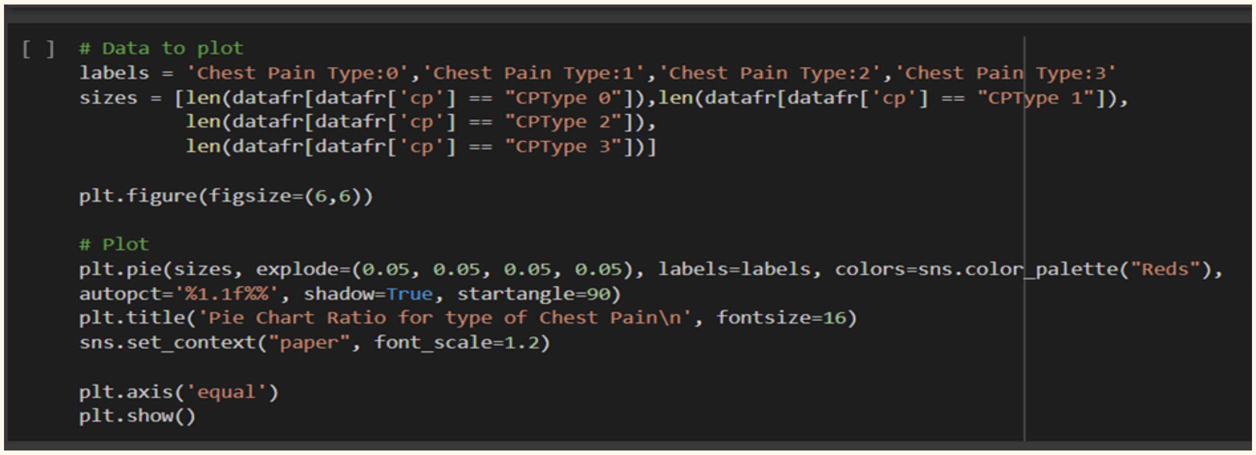
## Plot displaying male and female with potential heart disease based on age



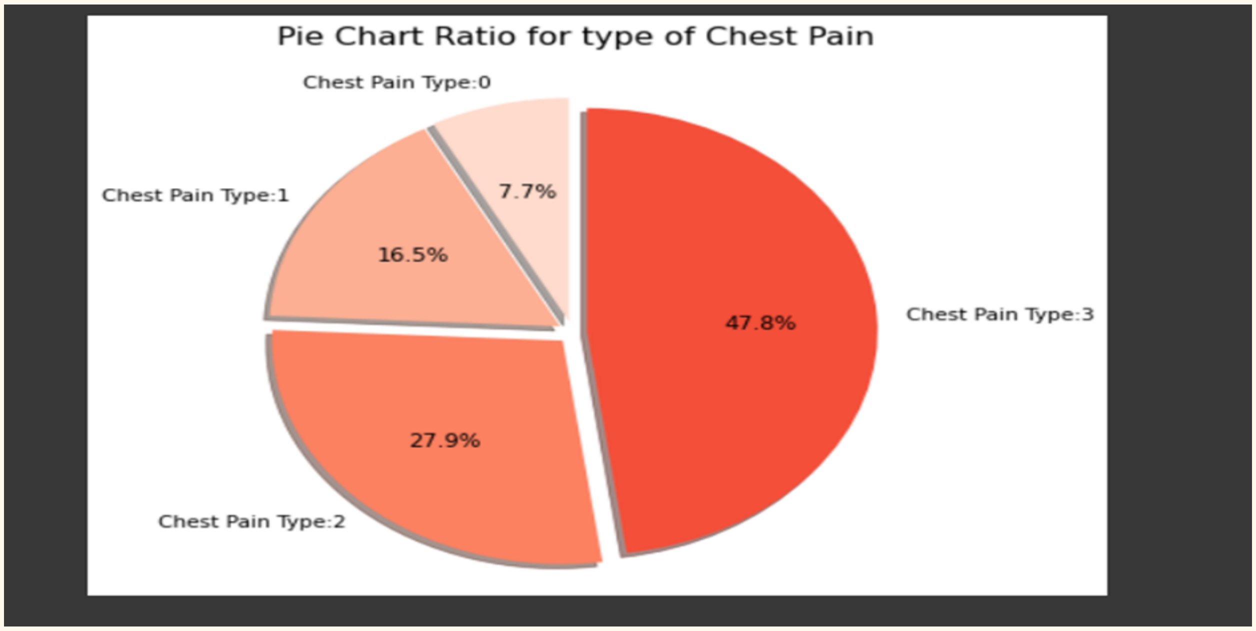
```
plt.figure(figsize=(6,6))
sns.catplot(x="sex", y="age", hue="target", kind="swarm", data=datafr, palette='Greens')
plt.title('Swarm Plot of Sex vs Age\n', fontsize=16)
sns.set_context("paper", font_scale=1.2)
```

```
<Figure size 432x432 with 0 Axes>
```

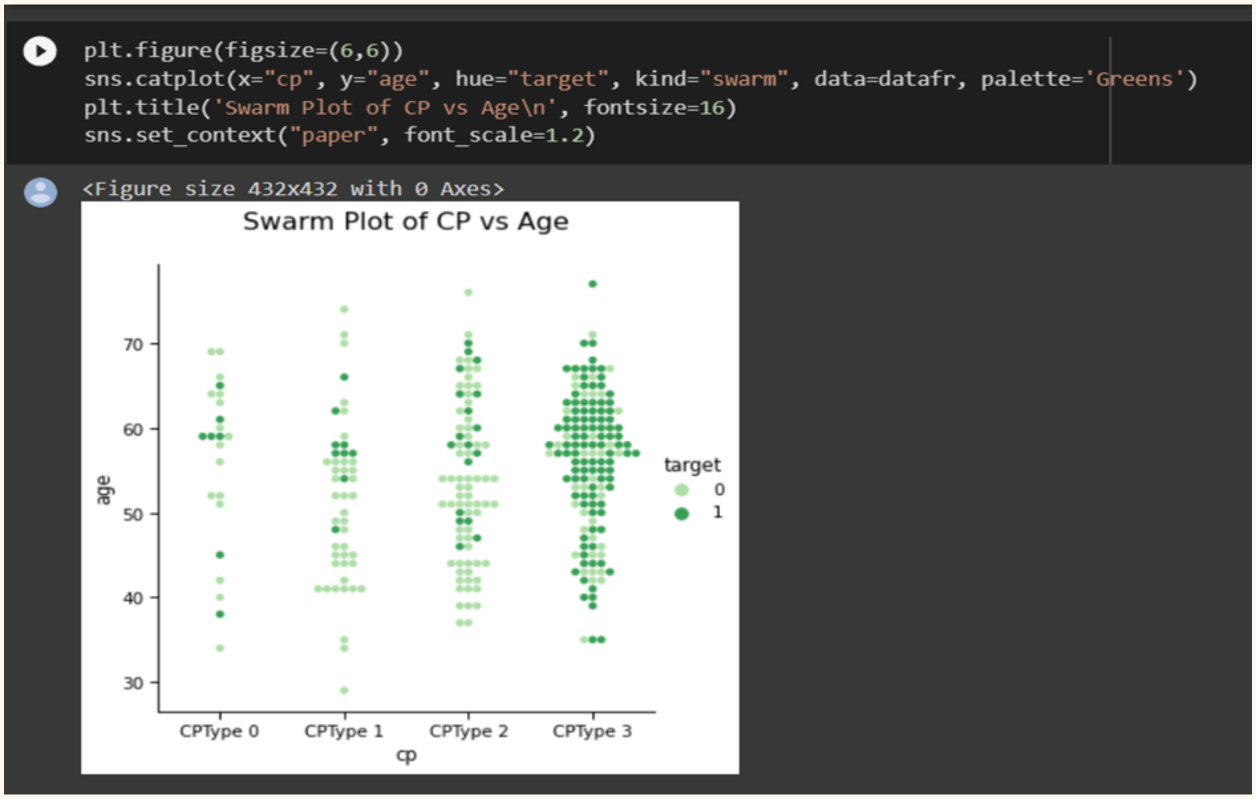
## Pie chart for different chest pain types in the dataset-



```
[ ] # Data to plot
    labels = 'Chest Pain Type:0','Chest Pain Type:1','Chest Pain Type:2','Chest Pain Type:3'
    sizes = [len(datafr[datafr['cp'] == "CPType 0"]),len(datafr[datafr['cp'] == "CPType 1"]),
             len(datafr[datafr['cp'] == "CPType 2"]),
             len(datafr[datafr['cp'] == "CPType 3"])]

    plt.figure(figsize=(6,6))

    # Plot
    plt.pie(sizes, explode=(0.05, 0.05, 0.05, 0.05), labels=labels, colors=sns.color_palette("Reds"),
    autopct='%1.1f%%', shadow=True, startangle=90)
    plt.title('Pie Chart Ratio for type of Chest Pain\n', fontsize=16)
    sns.set_context("paper", font_scale=1.2)

    plt.axis('equal')
    plt.show()
```

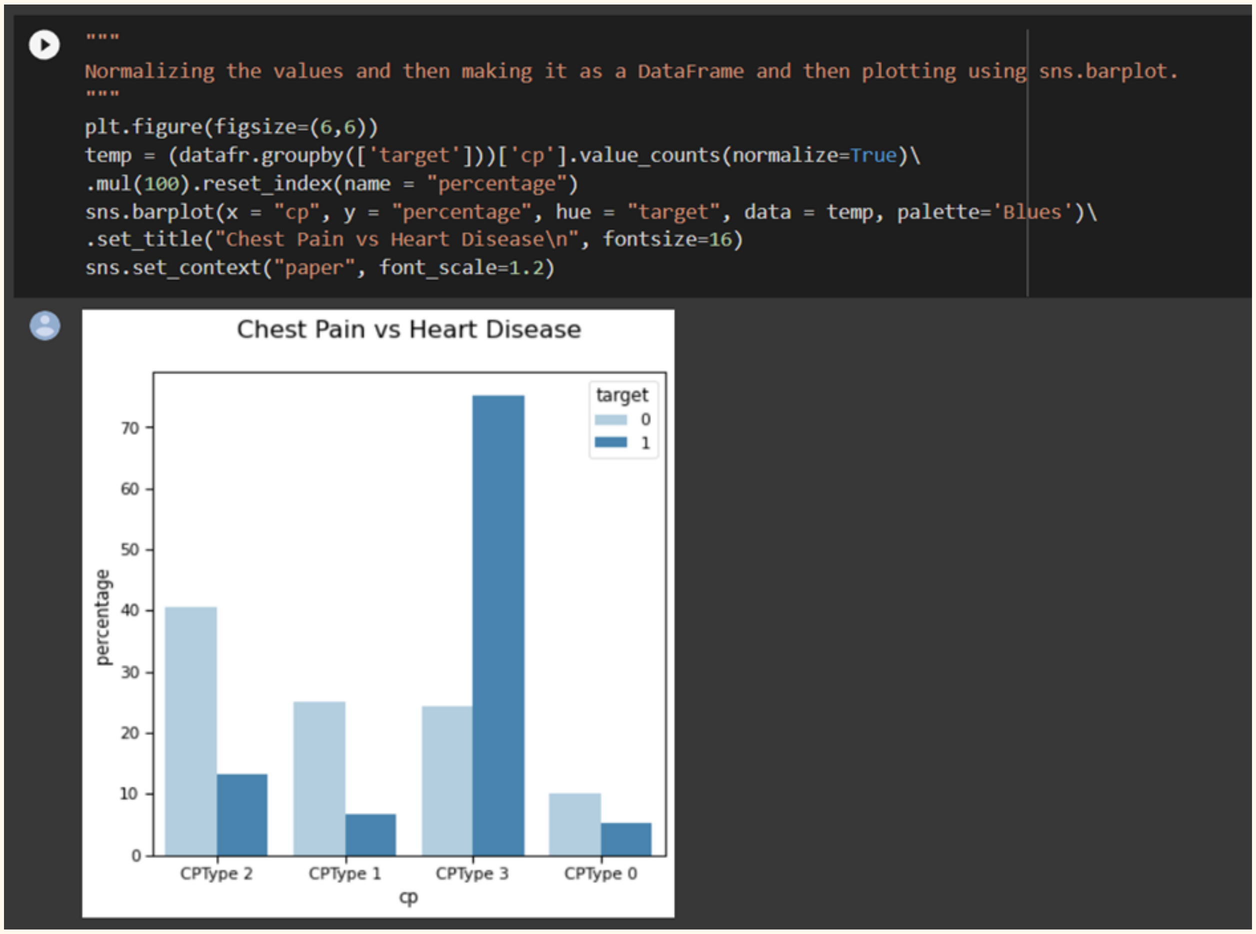


**Plot displaying potential of heart disease based on various levels of chest pain types by age-**



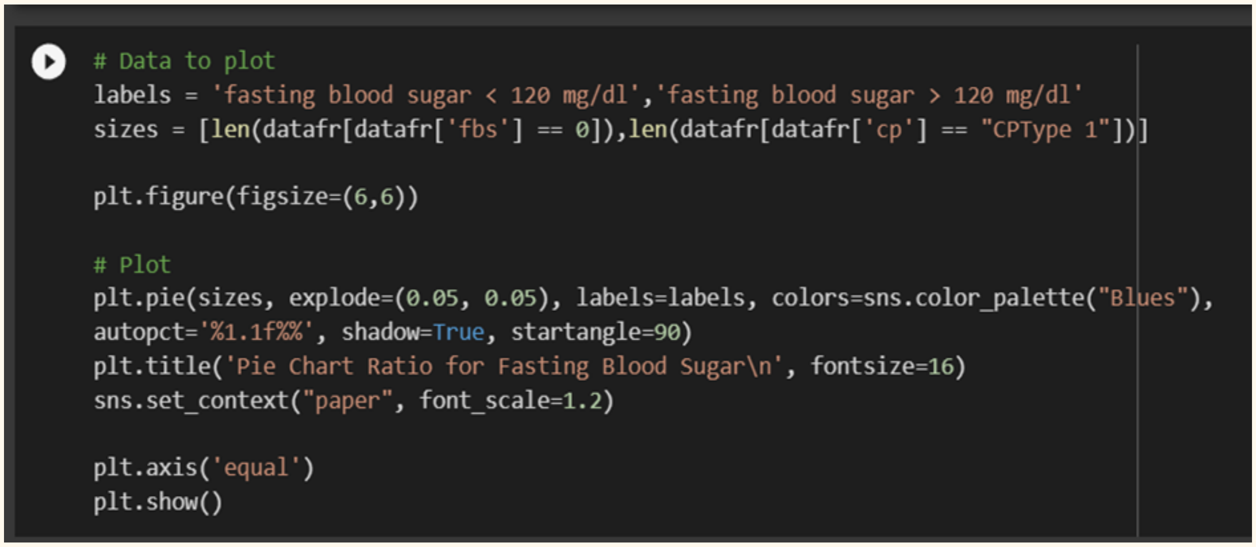
```
plt.figure(figsize=(6,6))
sns.catplot(x="cp", y="age", hue="target", kind="swarm", data=datafr, palette='Greens')
plt.title('Swarm Plot of CP vs Age\n', fontsize=16)
sns.set_context("paper", font_scale=1.2)
```

```
<Figure size 432x432 with 0 Axes>
```

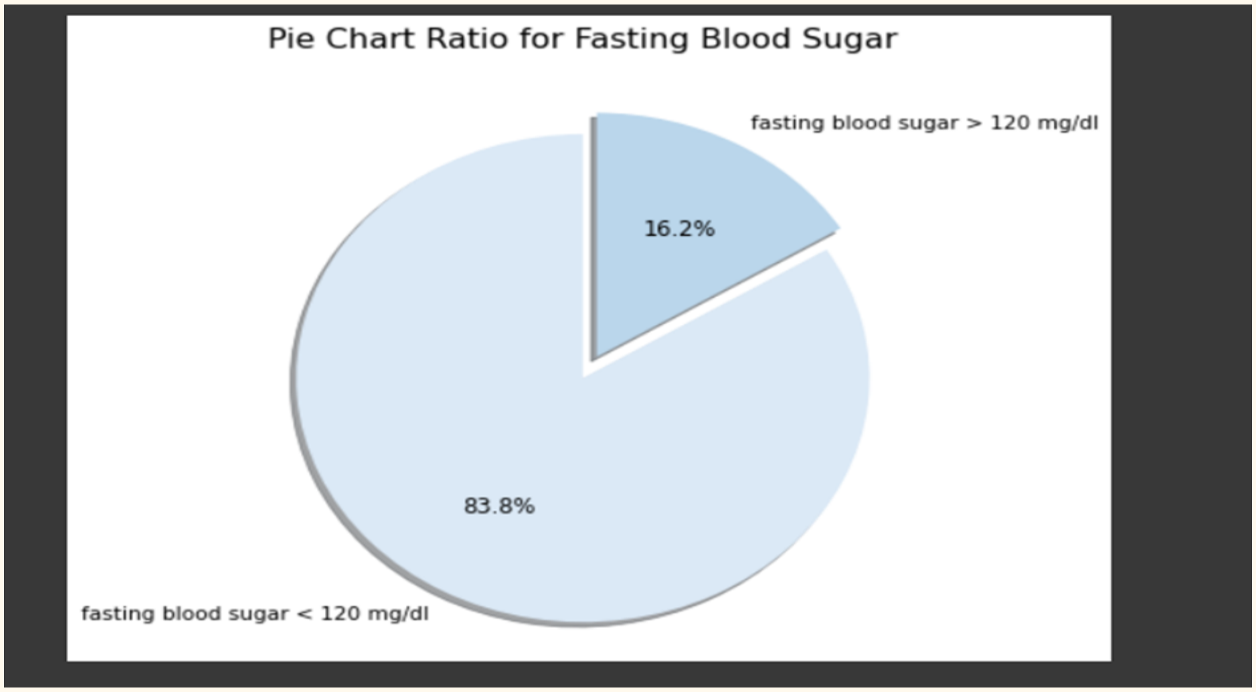


```
"""
Normalizing the values and then making it as a DataFrame and then plotting using sns.barplot.
"""
plt.figure(figsize=(6,6))
temp = (datafr.groupby(['target']))['cp'].value_counts(normalize=True)\
.mul(100).reset_index(name = "percentage")
sns.barplot(x = "cp", y = "percentage", hue = "target", data = temp, palette='Blues')\
.set_title("Chest Pain vs Heart Disease\n", fontsize=16)
sns.set_context("paper", font_scale=1.2)
```

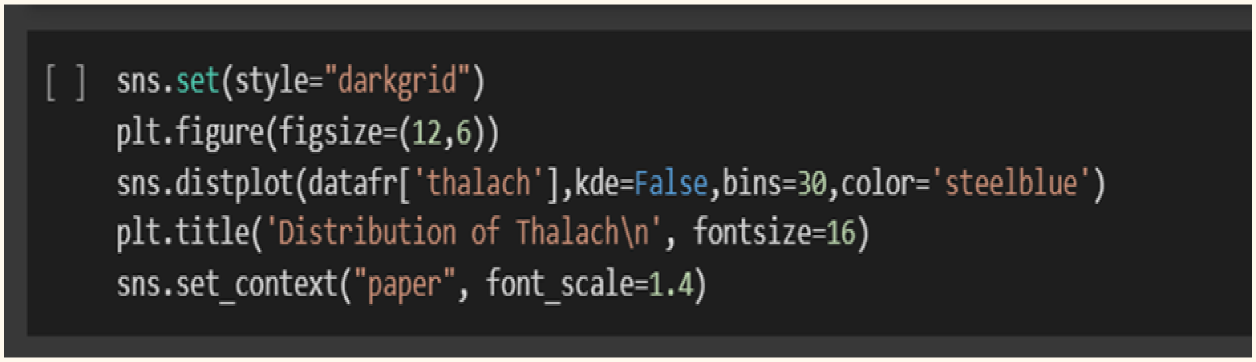
## Visualizing the ratio of dataset based on (fasting blood sugar > 120 mg/dl)-



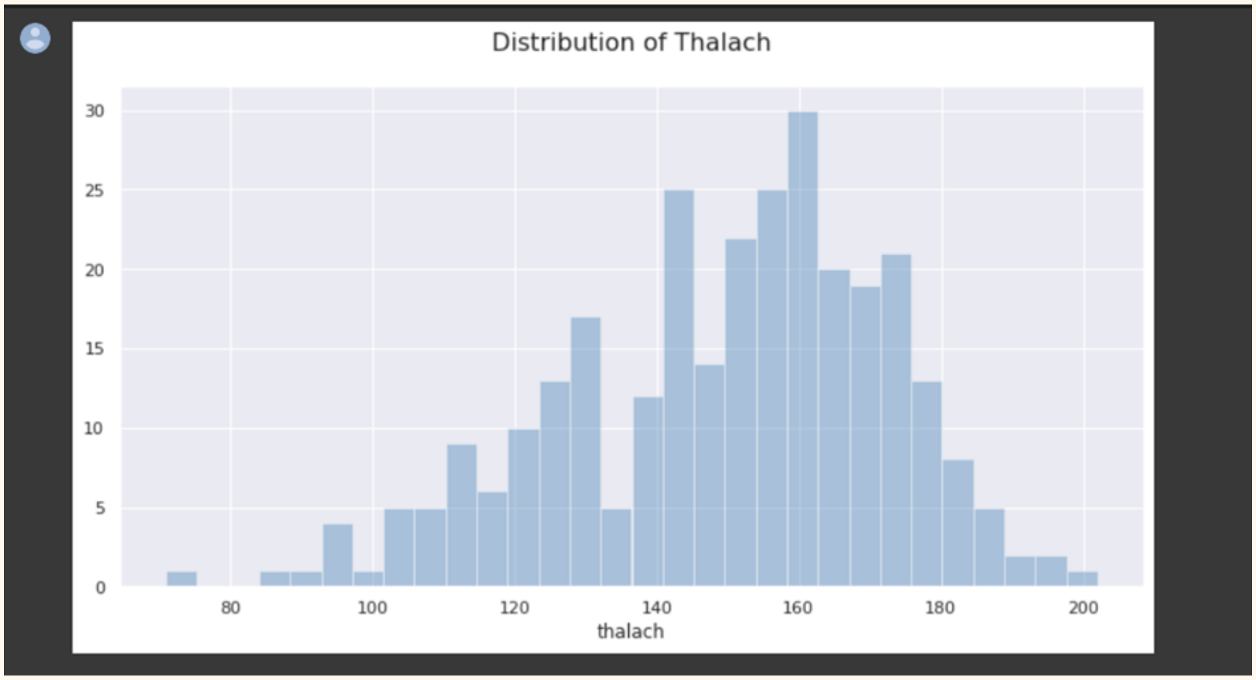
```
# Data to plot
labels = 'fasting blood sugar < 120 mg/dl','fasting blood sugar > 120 mg/dl'
sizes = [len(datafr[datafr['fbs'] == 0]),len(datafr[datafr['cp'] == "CPType 1"])]

plt.figure(figsize=(6,6))

# Plot
plt.pie(sizes, explode=(0.05, 0.05), labels=labels, colors=sns.color_palette("Blues"),
autopct='%1.1f%%', shadow=True, startangle=90)
plt.title('Pie Chart Ratio for Fasting Blood Sugar\n', fontsize=16)
sns.set_context("paper", font_scale=1.2)

plt.axis('equal')
plt.show()
```



## Checking the distribution of feature 'thalach: maximum heart rate achieved'

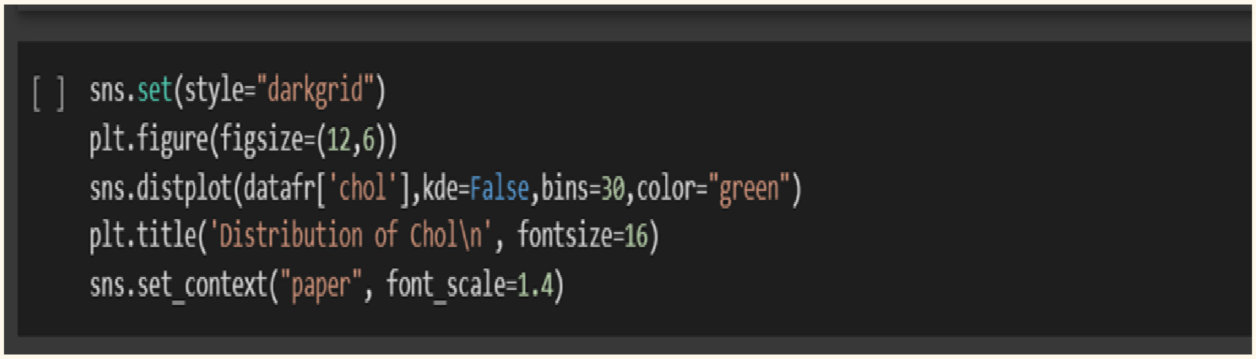


```
[ ]  sns.set(style="darkgrid")
     plt.figure(figsize=(12,6))
     sns.distplot(datafr['thalach'],kde=False,bins=30,color='steelblue')
     plt.title('Distribution of Thalach\n', fontsize=16)
     sns.set_context("paper", font_scale=1.4)
```

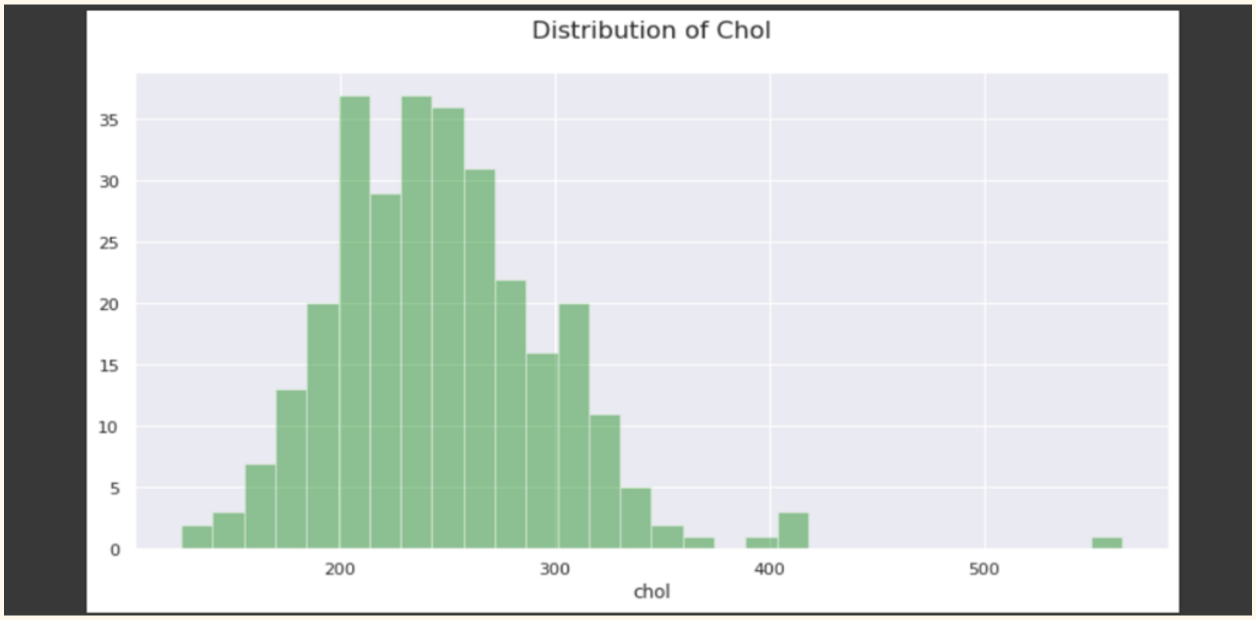


The distribution is normal indicating most populations fall between 140-180 with some left tail indicating few outliers which we can validate once we create a scatter plot with residuals, leverage and cook's distance.

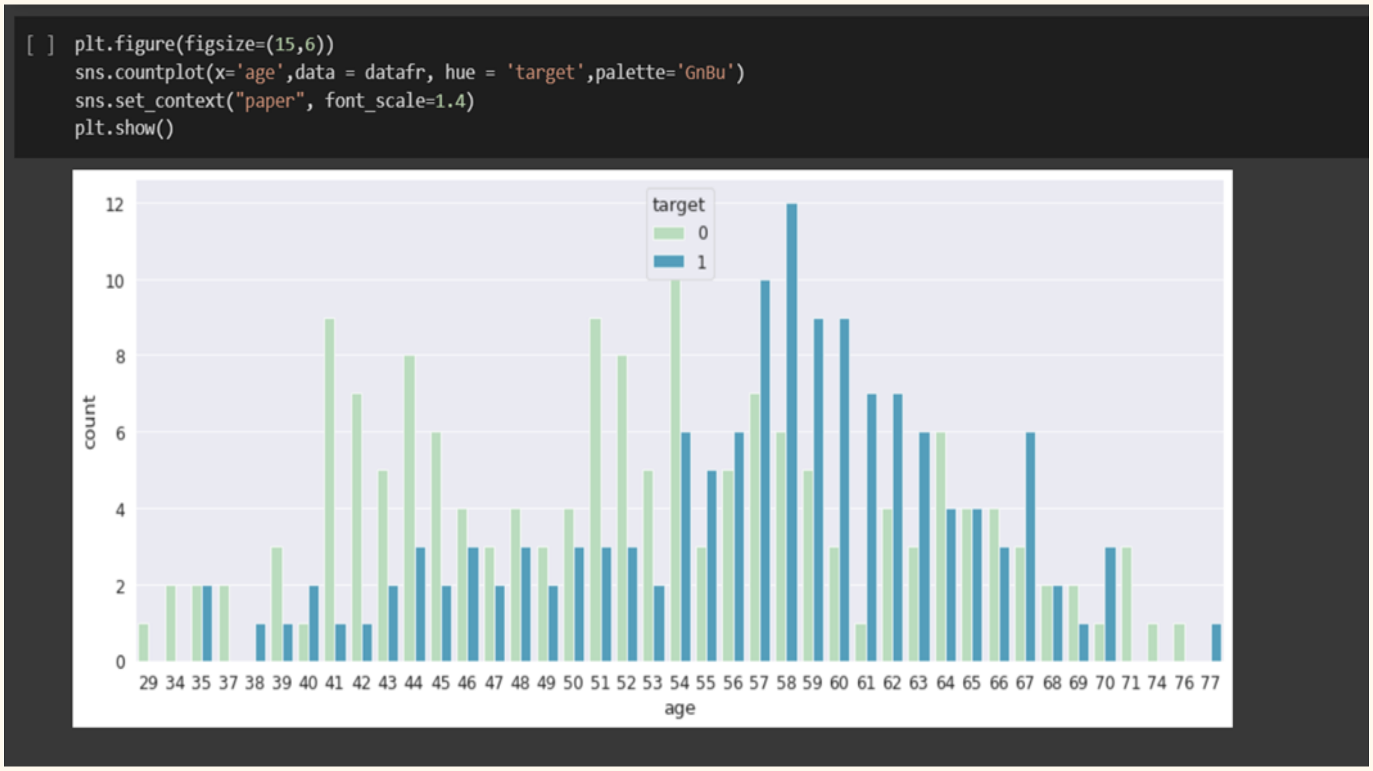
## Checking the distribution of feature 'chol: serum cholesterol in mg/dl'



```
[ ] sns.set(style="darkgrid")
    plt.figure(figsize=(12,6))
    sns.distplot(datafr['chol'],kde=False,bins=30,color="green")
    plt.title('Distribution of Chol\n', fontsize=16)
    sns.set_context("paper", font_scale=1.4)
```



## Visualizing the distribution of people having heart disease based on age-

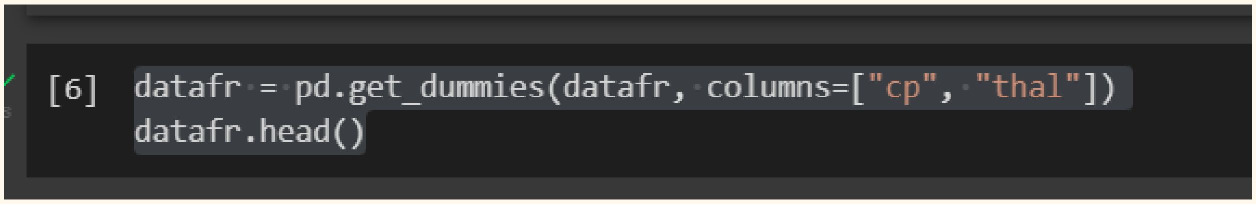


```
[ ] plt.figure(figsize=(15,6))
    sns.countplot(x='age',data = datafr, hue = 'target',palette='GnBu')
    sns.set_context("paper", font_scale=1.4)
    plt.show()
```

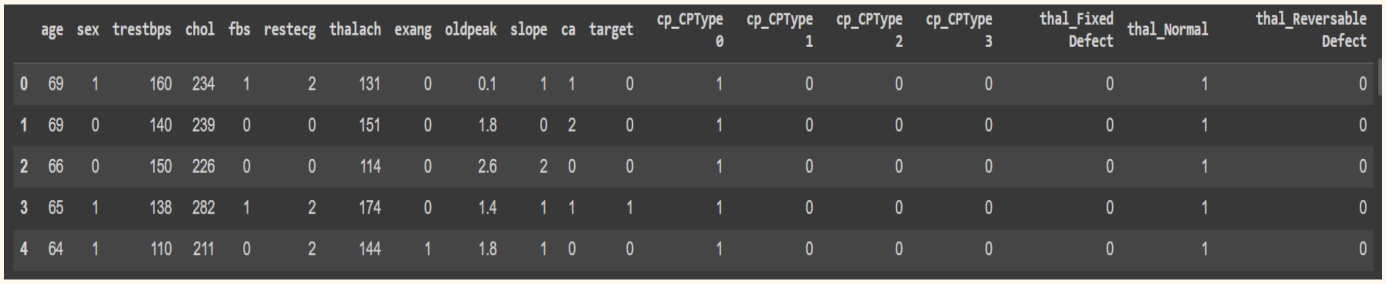
## Data Preparation

### Data Manipulation:

We use data manipulation to convert categorical data into numeric data as the machine learning algorithm can only work with numeric values. The get_dummies method in pandas is used to convert categorical variables into dummy variables, also known as indicator variables.



```
[6]  datafr = pd.get_dummies(datafr, columns=["cp", "thal"])
     datafr.head()
```



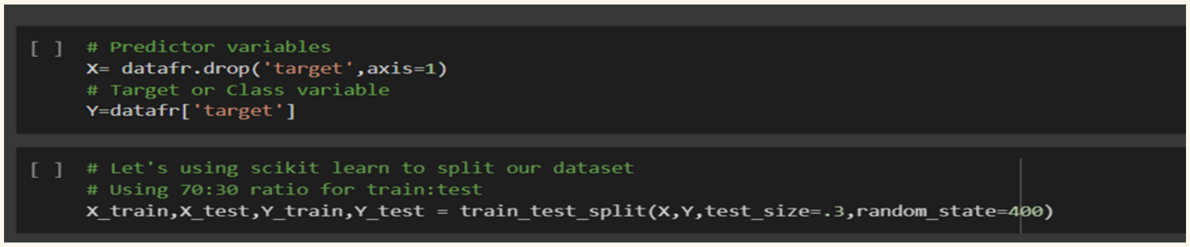
|   | age | sex | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca | target | cp_CPType 0 | cp_CPType 1 | cp_CPType 2 | cp_CPType 3 | thal_Fixed Defect | thal_Normal | thal_Reversable Defect |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 69 | 1 | 160 | 234 | 1 | 2 | 131 | 0 | 0.1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 69 | 0 | 140 | 239 | 0 | 0 | 151 | 0 | 1.8 | 0 | 2 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 2 | 66 | 0 | 150 | 226 | 0 | 0 | 114 | 0 | 2.6 | 2 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 3 | 65 | 1 | 138 | 282 | 1 | 2 | 174 | 0 | 1.4 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 4 | 64 | 1 | 110 | 211 | 0 | 2 | 144 | 1 | 1.8 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |

### Splitting data into test data and training data:

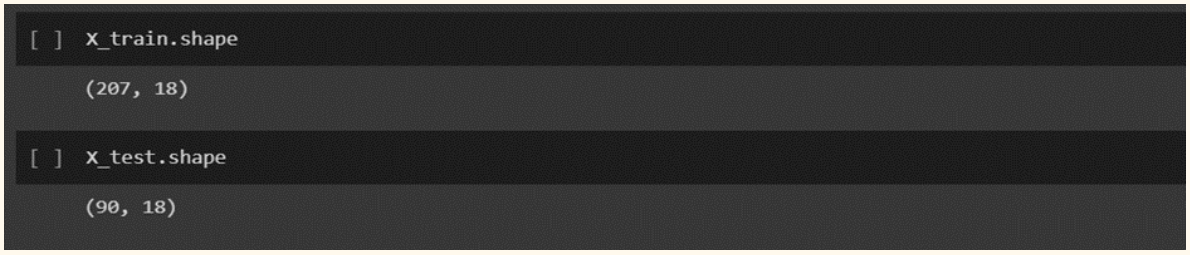
We divided the entire dataset into two parts in this step: the training dataset and the testing dataset.

Training dataset - It is deployed for making learning model based on the corresponding algorithm

Testing dataset - It is used to check the efficiency of the learned model by comparing the predicted result and original result. Based on this accuracy and f measure are calculated.



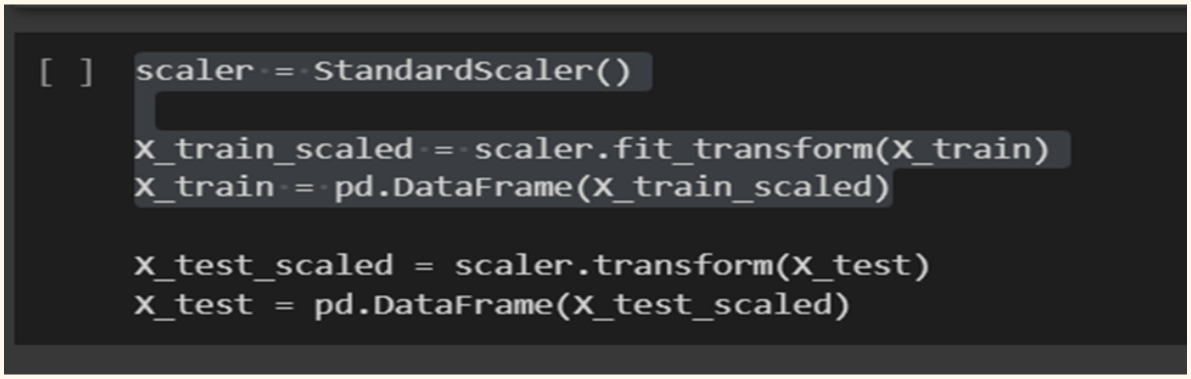
```
[ ]  # Predictor variables
     X= datafr.drop('target',axis=1)
     # Target or Class variable
     Y=datafr['target']

[ ]  # Let's using scikit learn to split our dataset
     # Using 70:30 ratio for train:test
     X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=.3,random_state=400)
```



```
[ ]  X_train.shape

     (207, 18)

[ ]  X_test.shape

     (90, 18)
```

## Pre-processing and cleaning

We use standardization to preprocess the data. Standardization is a common preprocessing technique in machine learning, where the goal is to transform the features of a dataset so that they have zero mean and unit variance. This is often done by subtracting the mean of each feature from the feature's values and dividing the result by the standard deviation of the feature. In our machine learning algorithm we assume that the input features are normally distributed or at least have a similar scale.



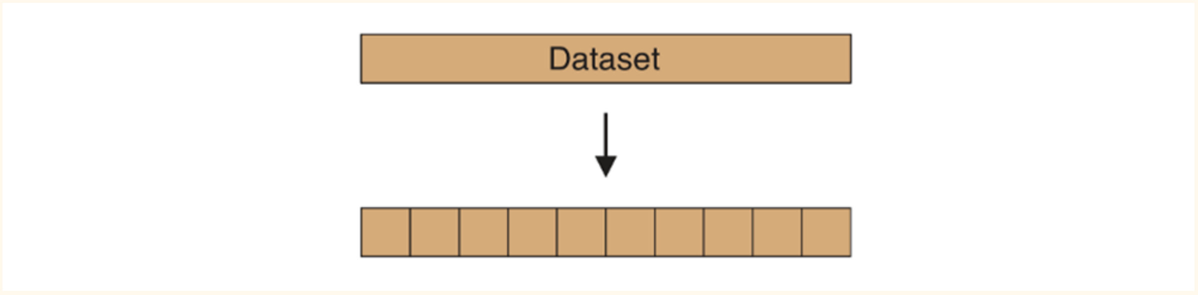
```
[ ]  scaler = StandardScaler()

     X_train_scaled = scaler.fit_transform(X_train)
     X_train = pd.DataFrame(X_train_scaled)

     X_test_scaled = scaler.transform(X_test)
     X_test = pd.DataFrame(X_test_scaled)
```

In this code, StandardScaler is a preprocessing function that is used to standardize the features of a dataset by subtracting the mean and scaling to unit variance.

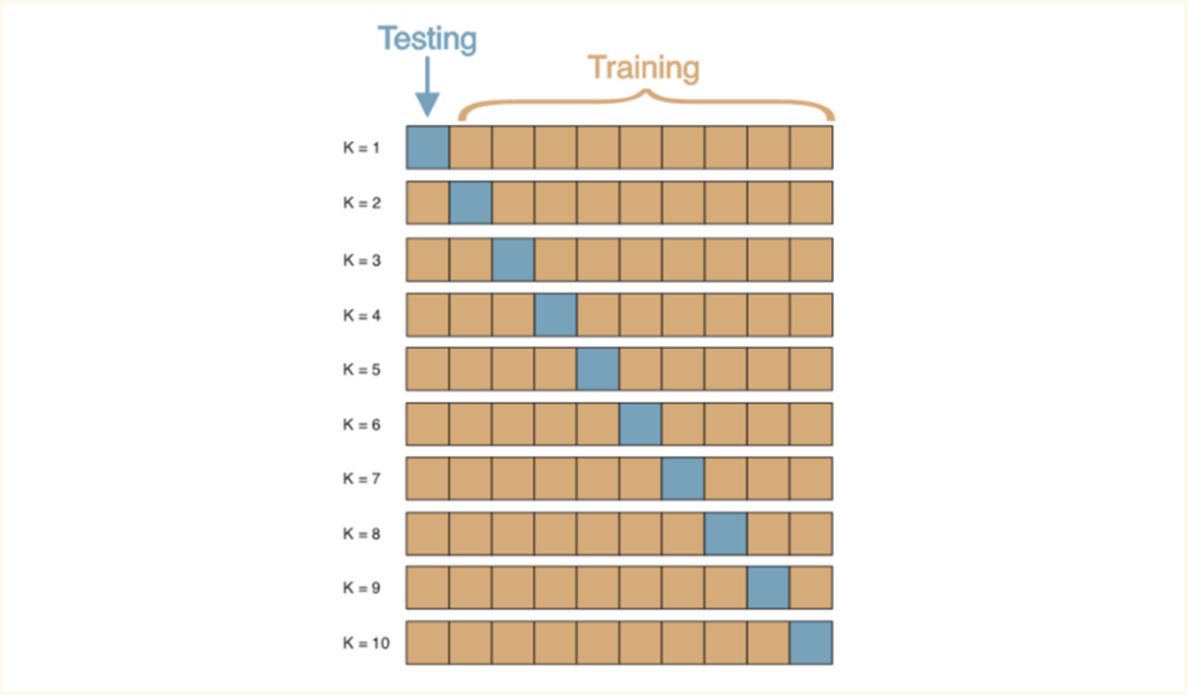
## Evaluating ML Model

Cross-validation is a technique for evaluating ML models by training several ML models on subsets of the available input data and evaluating them on the complementary subset of the data.

In this analysis, we'll use the 10-fold cross-validation. So, the first step is to shuffle and split our dataset into 10 folds.

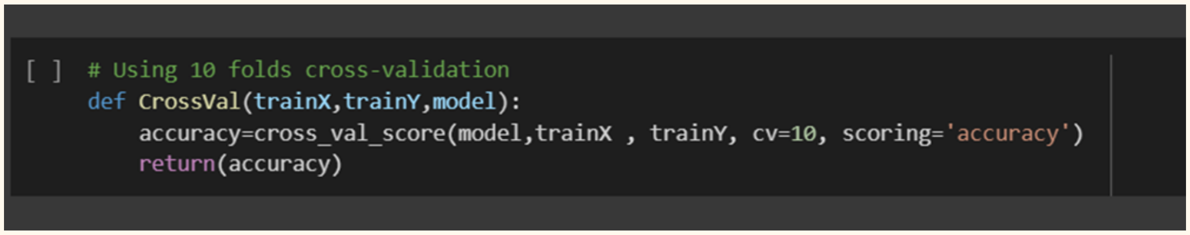


Then, we'll use one fold for testing and computing the empirical square loss and the remaining 9 other ones for training our model in each *k* interaction. By doing this, each time we begin a new interaction we use a different fold for testing. This way, we guarantee that every *k* part is used one time for testing.



In the end, we'll have 10 values for the empirical square loss, one for each interaction. The final empirical square loss will be the average of these values.

### Code



```
[ ]  # Using 10 folds cross-validation
     def CrossVal(trainX,trainY,model):
         accuracy=cross_val_score(model,trainX , trainY, cv=10, scoring='accuracy')
         return(accuracy)
```

## Applying ML Algorithms

We have used four machine learning algorithms in this project. Simple Vector Machine, Random Forest Classification, KNN Classifier, and AdaBoost Classification algorithms. Various parameters, such as test accuracy, F1 score, Confusion Matrix, and Area under the ROC curve(AUC) score, have been compared to determine which model is the best for our dataset.

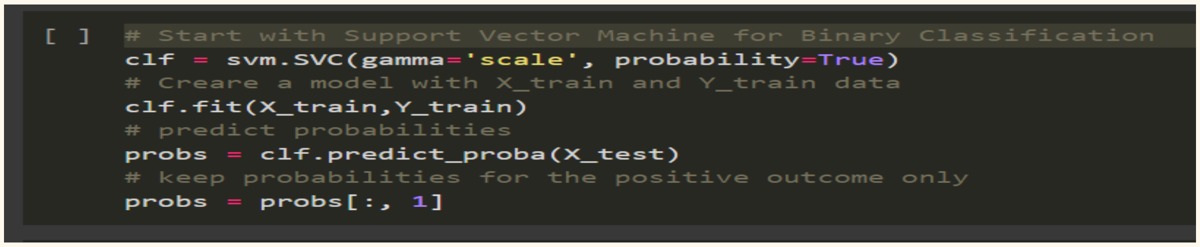
**The different Algorithms used:**

### Support Vector Machine (SVM)

Support Vector Machine or SVM is a supervised and linear Machine Learning algorithm primarily used for solving classification problems as well as regression problems and is also referred to as Support Vector Classification. SVM also supports the kernel method called the kernel SVM, which allows us to tackle non-linearity. SVM chooses the extreme points/vectors that help create a hyperplane to differentiate between the different categories.

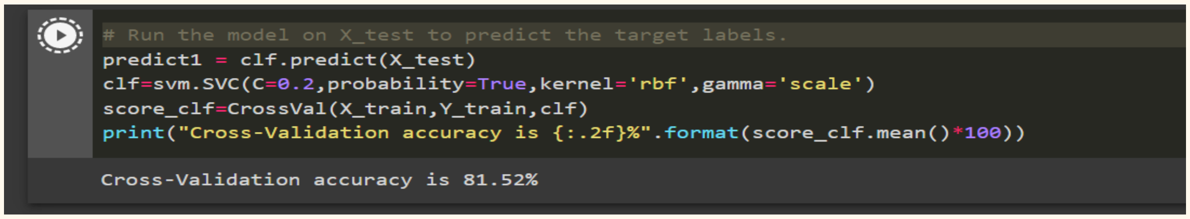
SVM algorithms can be used for Face detection, image classification, text categorization, etc.

## Code Implementation of SVM:



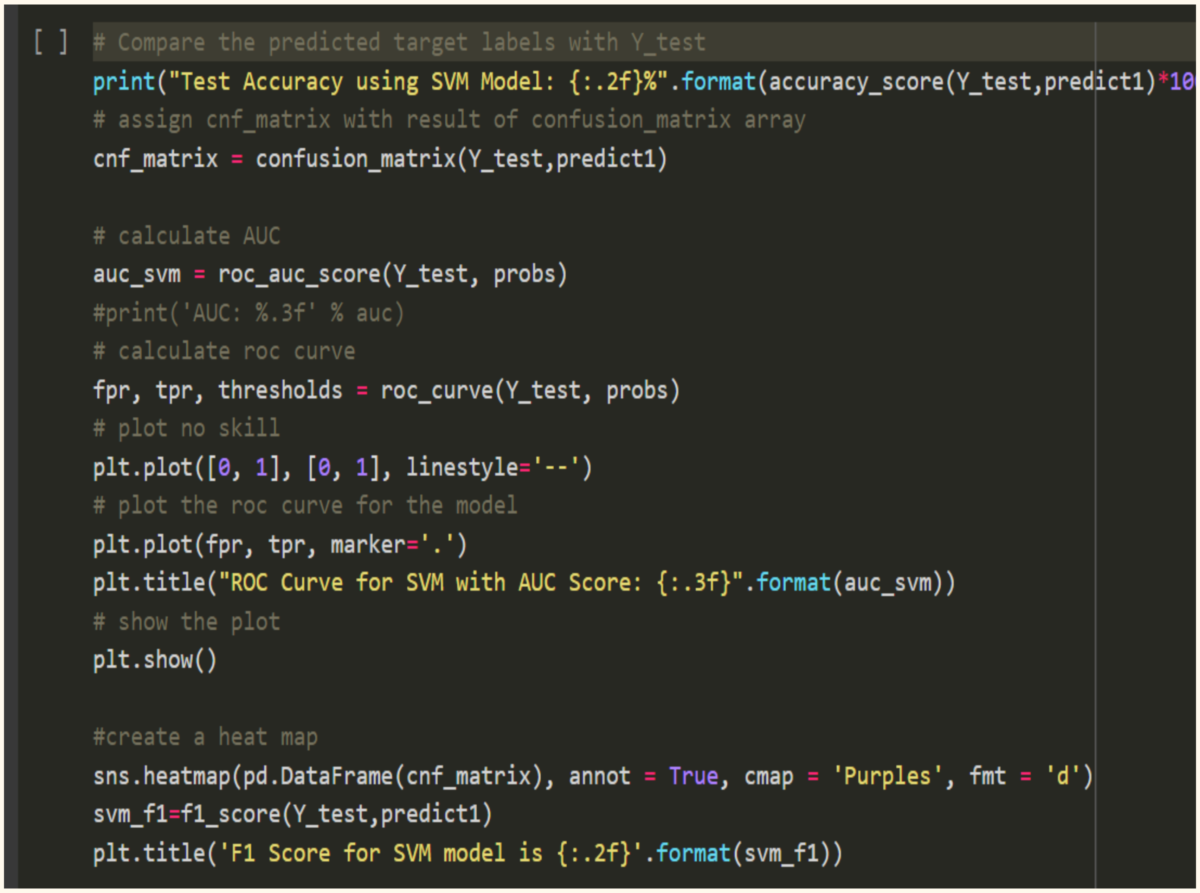
```
[ ]  # Start with Support Vector Machine for Binary Classification
     clf = svm.SVC(gamma='scale', probability=True)
     # Creare a model with X_train and Y_train data
     clf.fit(X_train,Y_train)
     # predict probabilities
     probs = clf.predict_proba(X_test)
     # keep probabilities for the positive outcome only
     probs = probs[:, 1]
```

## Accuracy of the model:

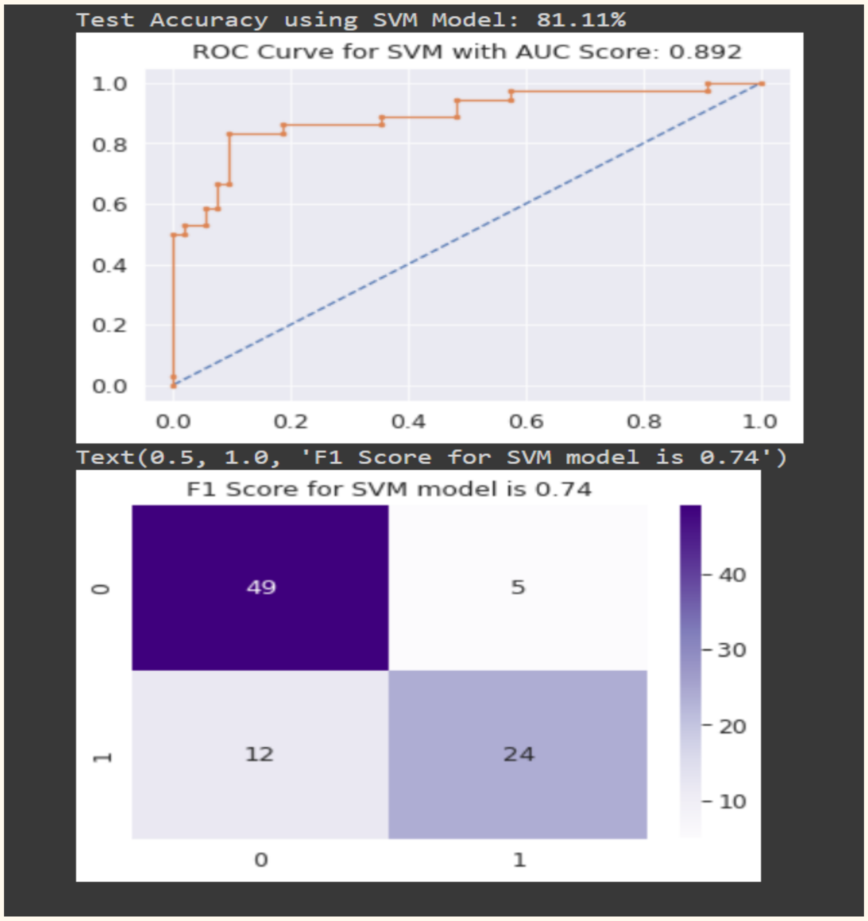


```
# Run the model on X_test to predict the target labels.
predict1 = clf.predict(X_test)
clf=svm.SVC(C=0.2,probability=True,kernel='rbf',gamma='scale')
score_clf=CrossVal(X_train,Y_train,clf)
print("Cross-Validation accuracy is {:.2f}%".format(score_clf.mean()*100))

Cross-Validation accuracy is 81.52%
```

## Heat Map:



```
[ ]  # Compare the predicted target labels with Y_test
     print("Test Accuracy using SVM Model: {:.2f}%".format(accuracy_score(Y_test,predict1)*10
     # assign cnf_matrix with result of confusion_matrix array
     cnf_matrix = confusion_matrix(Y_test,predict1)

     # calculate AUC
     auc_svm = roc_auc_score(Y_test, probs)
     #print('AUC: %.3f' % auc)
     # calculate roc curve
     fpr, tpr, thresholds = roc_curve(Y_test, probs)
     # plot no skill
     plt.plot([0, 1], [0, 1], linestyle='--')
     # plot the roc curve for the model
     plt.plot(fpr, tpr, marker='.')
     plt.title("ROC Curve for SVM with AUC Score: {:.3f}".format(auc_svm))
     # show the plot
     plt.show()

     #create a heat map
     sns.heatmap(pd.DataFrame(cnf_matrix), annot = True, cmap = 'Purples', fmt = 'd')
     svm_f1=f1_score(Y_test,predict1)
     plt.title('F1 Score for SVM model is {:.2f}'.format(svm_f1))
```



```
Test Accuracy using SVM Model: 81.11%
```

ROC Curve for SVM with AUC Score: 0.892
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

```
Text(0.5, 1.0, 'F1 Score for SVM model is 0.74')
```

F1 Score for SVM model is 0.74
0
1
49
5
12
24
0
1
40
30
20
10

## Random Forest Classifier

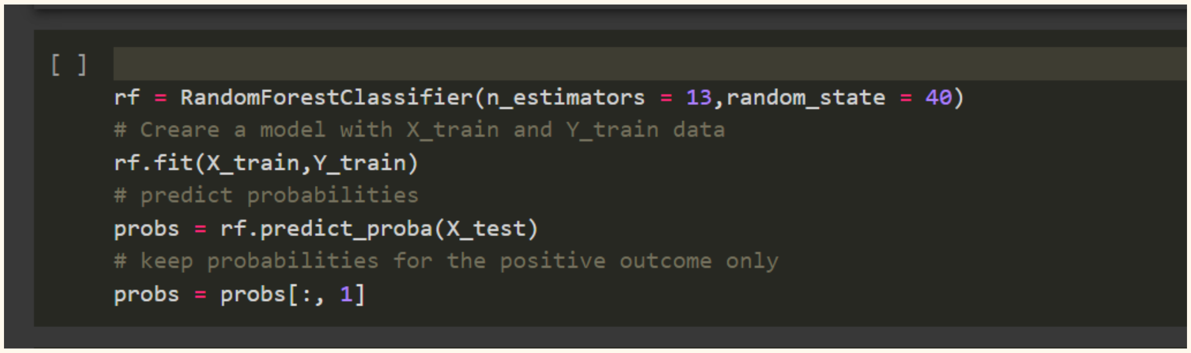
Random forest is a supervised learning algorithm. It can be used for classification and regression problems and is based on ensemble learning. It is also a flexible and easy-to-use algorithm. A forest consists of trees. It is said that the more trees there are, the more robust a forest is. Random forests create decision trees on randomly selected data samples, get predictions from each tree and choose the best solution using voting. It also provides a pretty good indicator of the feature's importance.

We have used this model because:

- This approach makes it easy to estimate variable importance, or contribution, to a model.
- When there are many decision trees in a random forest, the classifier will not overfit the model because averaging the uncorrelated trees lowers the overall variance and prediction error.

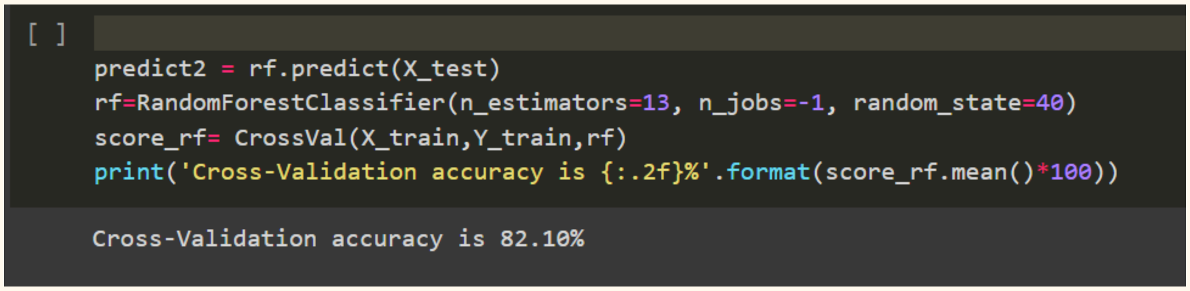
Random forest algorithms can be used for disease trends, land use, marketing, banking, etc.

## Code Implementation of Random Forest:



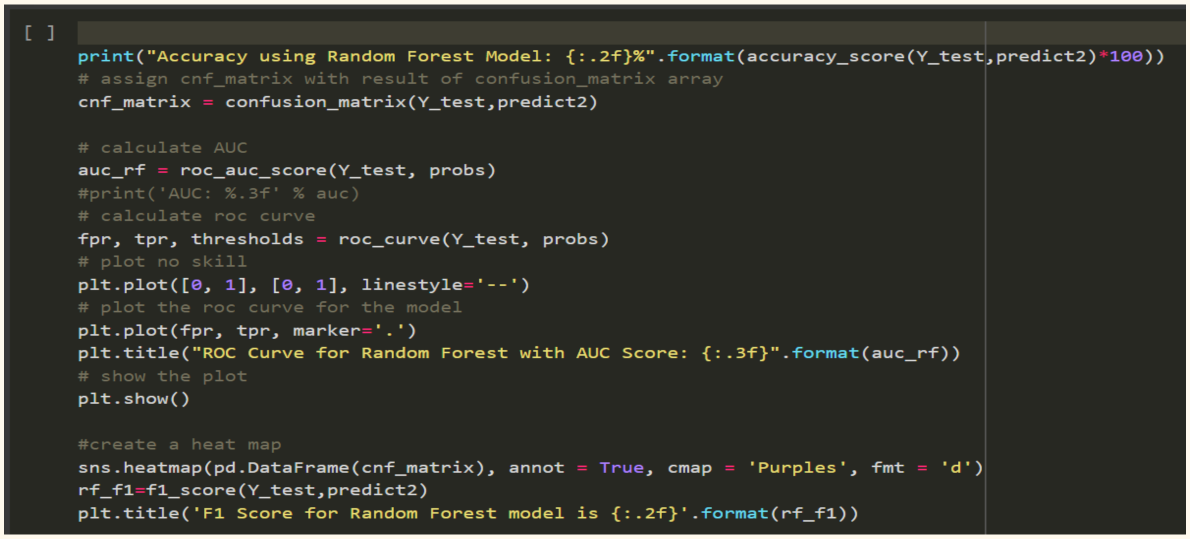
```
[ ]
    rf = RandomForestClassifier(n_estimators = 13,random_state = 40)
    # Creare a model with X_train and Y_train data
    rf.fit(X_train,Y_train)
    # predict probabilities
    probs = rf.predict_proba(X_test)
    # keep probabilities for the positive outcome only
    probs = probs[:, 1]
```

## Accuracy of the model:

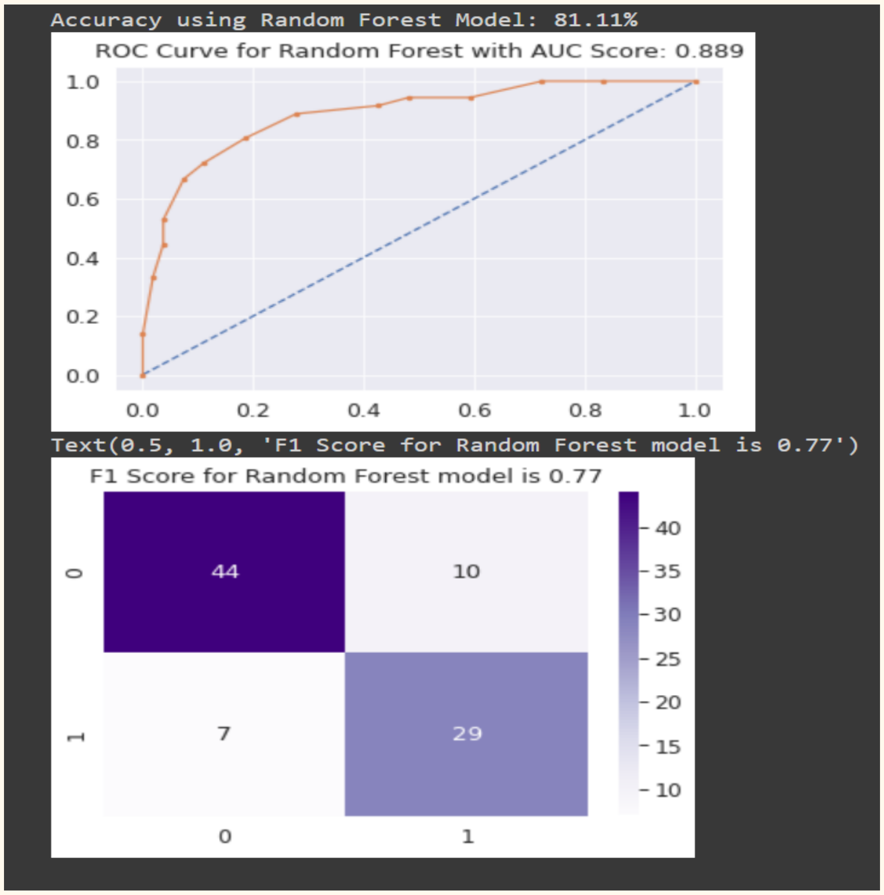


```
[ ]
    predict2 = rf.predict(X_test)
    rf=RandomForestClassifier(n_estimators=13, n_jobs=-1, random_state=40)
    score_rf= CrossVal(X_train,Y_train,rf)
    print('Cross-Validation accuracy is {:.2f}%'.format(score_rf.mean()*100))

    Cross-Validation accuracy is 82.10%
```

## Heat Map:



```
[ ]
    print("Accuracy using Random Forest Model: {:.2f}%".format(accuracy_score(Y_test,predict2)*100))
    # assign cnf_matrix with result of confusion_matrix array
    cnf_matrix = confusion_matrix(Y_test,predict2)

    # calculate AUC
    auc_rf = roc_auc_score(Y_test, probs)
    #print('AUC: %.3f' % auc)
    # calculate roc curve
    fpr, tpr, thresholds = roc_curve(Y_test, probs)
    # plot no skill
    plt.plot([0, 1], [0, 1], linestyle='--')
    # plot the roc curve for the model
    plt.plot(fpr, tpr, marker='.')
    plt.title("ROC Curve for Random Forest with AUC Score: {:.3f}".format(auc_rf))
    # show the plot
    plt.show()

    #create a heat map
    sns.heatmap(pd.DataFrame(cnf_matrix), annot = True, cmap = 'Purples', fmt = 'd')
    rf_f1=f1_score(Y_test,predict2)
    plt.title('F1 Score for Random Forest model is {:.2f}'.format(rf_f1))
```



```
Accuracy using Random Forest Model: 81.11%
```

```
Text(0.5, 1.0, 'F1 Score for Random Forest model is 0.77')
```

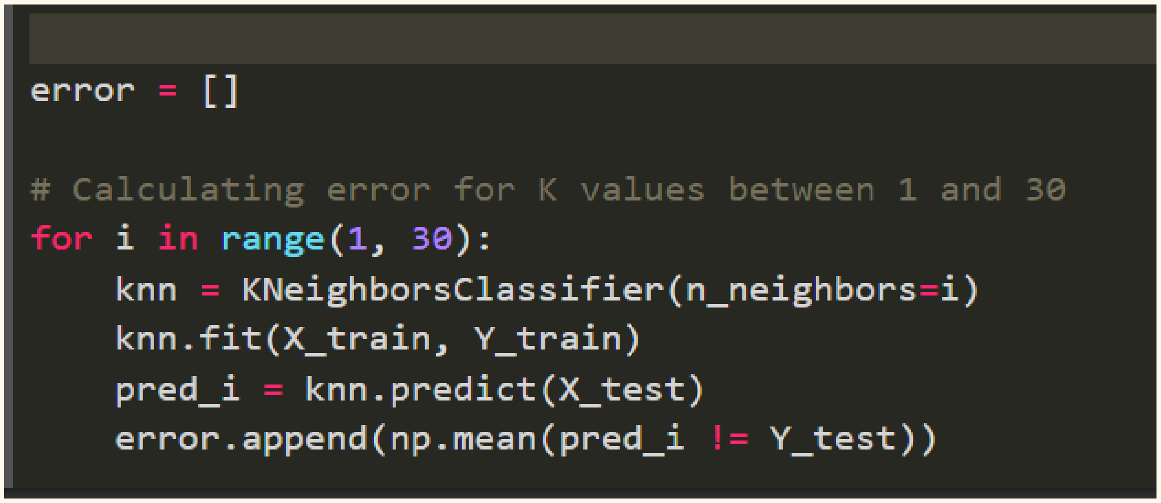
## K-Nearest Neighbour Classifier (KNN)

K-Nearest Neighbour is a simple Machine Learning algorithm based on Supervised Learning.  It is mainly used for Classification problems but can also be used for Regression. KNN algorithm assumes the similarity between the new case/data and available cases and puts the new case into the category that is most similar to the available categories. It is also called a lazy learner algorithm because it does not learn from the training set immediately; instead, it stores the dataset and acts on the dataset at the time of classification. K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data.

KNN algorithms can be used for handwriting detection, image recognition, video recognition, etc. KNN is most useful when labeled data is too expensive or impossible to obtain, and it can achieve high accuracy in various prediction-type problems.

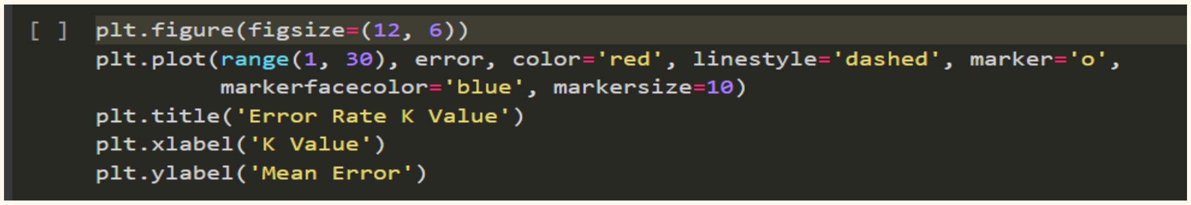
## Code Implementation of KNN:

We first create an empty list called "error" to hold error values. We then use a loop to iterate over values of k from 1 to 30 and calculate the error for each value of k. The error is calculated by fitting a KNN model with the current value of k to the training data, making predictions on the test data, and taking the mean of the prediction errors.



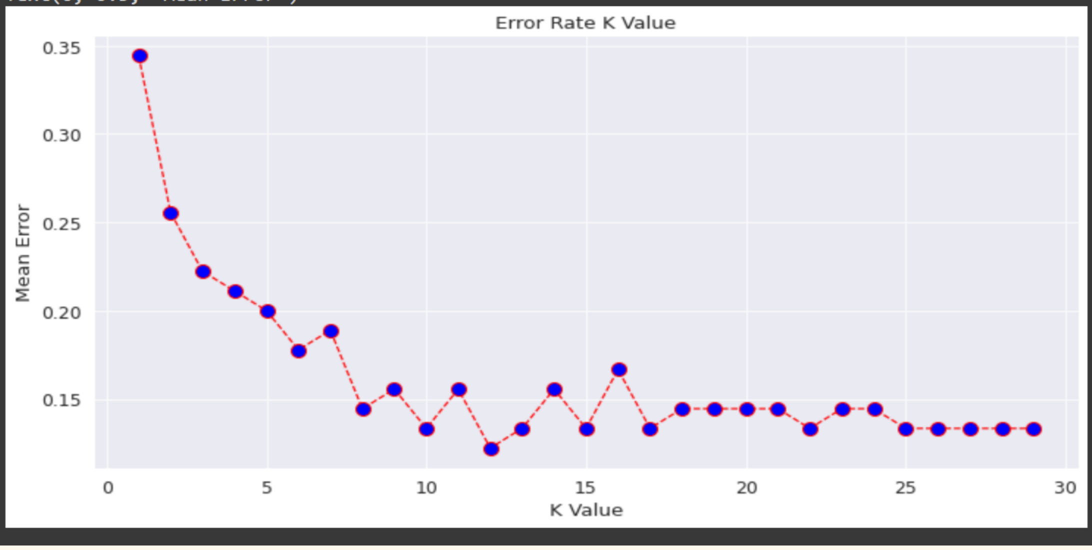
```
error = []

# Calculating error for K values between 1 and 30
for i in range(1, 30):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, Y_train)
    pred_i = knn.predict(X_test)
    error.append(np.mean(pred_i != Y_test))
```

The code then plots the error values as a function of k.



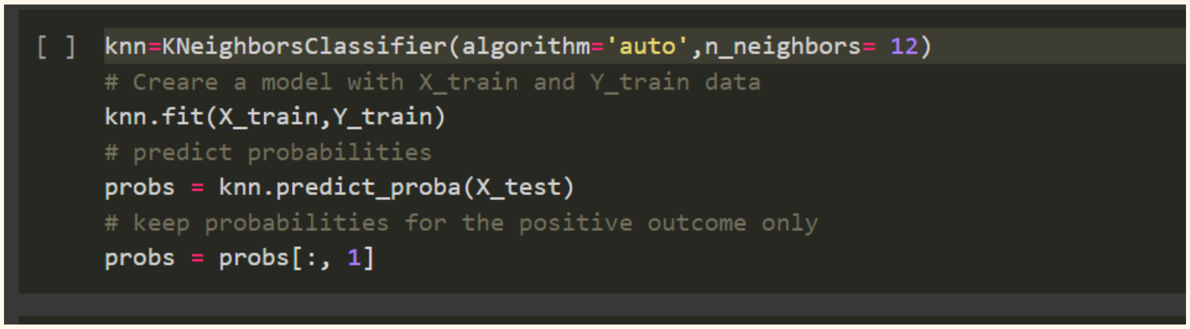
```
[ ]  plt.figure(figsize=(12, 6))
     plt.plot(range(1, 30), error, color='red', linestyle='dashed', marker='o',
              markerfacecolor='blue', markersize=10)
     plt.title('Error Rate K Value')
     plt.xlabel('K Value')
     plt.ylabel('Mean Error')
```

```
Text(0, 0.5, 'Mean Error')
```



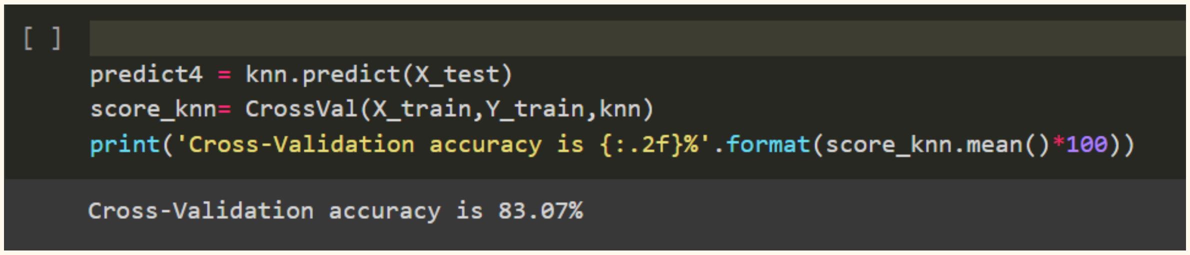
**We can see that the error is minimum for k = 12.**

**We will create a KNN model with k=12 and fit it to the training data.**



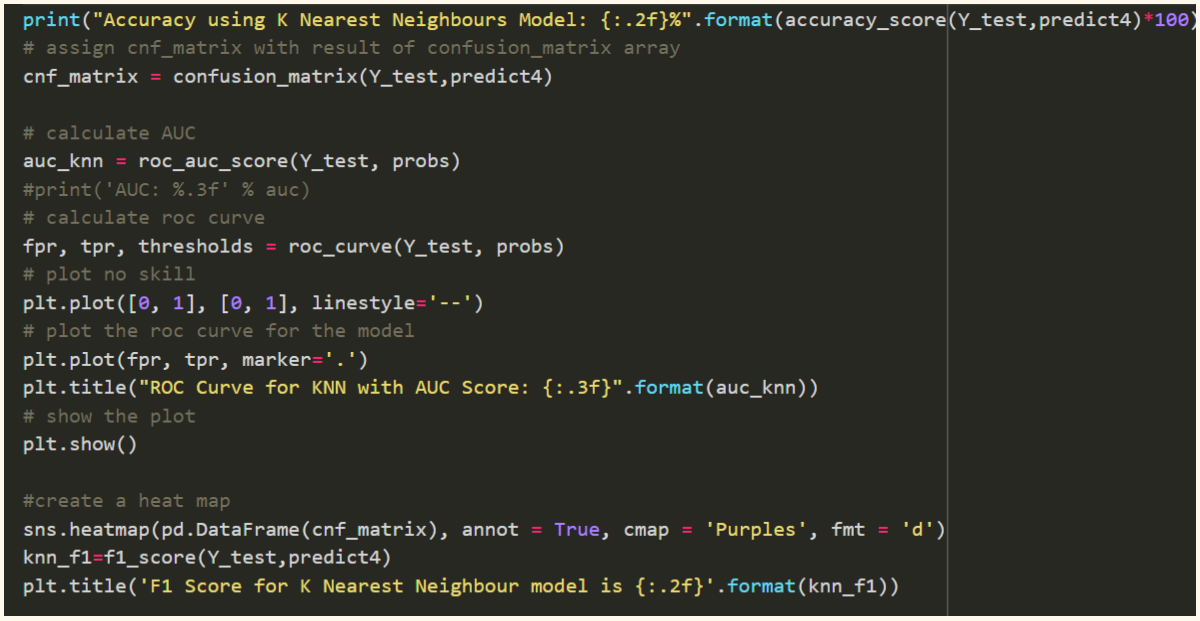
```
[ ]  knn=KNeighborsClassifier(algorithm='auto',n_neighbors= 12)
     # Creare a model with X_train and Y_train data
     knn.fit(X_train,Y_train)
     # predict probabilities
     probs = knn.predict_proba(X_test)
     # keep probabilities for the positive outcome only
     probs = probs[:, 1]
```

## Accuracy of the Model:

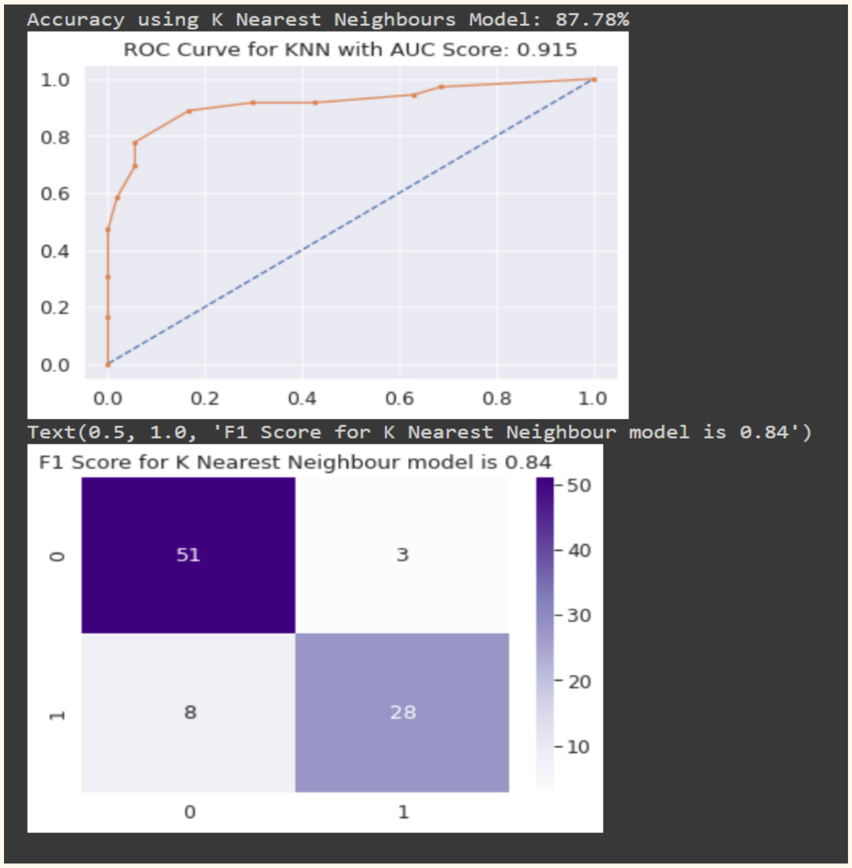


```
[ ]
    predict4 = knn.predict(X_test)
    score_knn= CrossVal(X_train,Y_train,knn)
    print('Cross-Validation accuracy is {:.2f}%'.format(score_knn.mean()*100))

    Cross-Validation accuracy is 83.07%
```

## Heat Map



```
print("Accuracy using K Nearest Neighbours Model: {:.2f}%".format(accuracy_score(Y_test,predict4)*100
# assign cnf_matrix with result of confusion_matrix array
cnf_matrix = confusion_matrix(Y_test,predict4)

# calculate AUC
auc_knn = roc_auc_score(Y_test, probs)
#print('AUC: %.3f' % auc)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(Y_test, probs)
# plot no skill
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(fpr, tpr, marker='.')
plt.title("ROC Curve for KNN with AUC Score: {:.3f}".format(auc_knn))
# show the plot
plt.show()

#create a heat map
sns.heatmap(pd.DataFrame(cnf_matrix), annot = True, cmap = 'Purples', fmt = 'd')
knn_f1=f1_score(Y_test,predict4)
plt.title('F1 Score for K Nearest Neighbour model is {:.2f}'.format(knn_f1))
```



```
Accuracy using K Nearest Neighbours Model: 87.78%
```

ROC Curve for KNN with AUC Score: 0.915
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

```
Text(0.5, 1.0, 'F1 Score for K Nearest Neighbour model is 0.84')
```

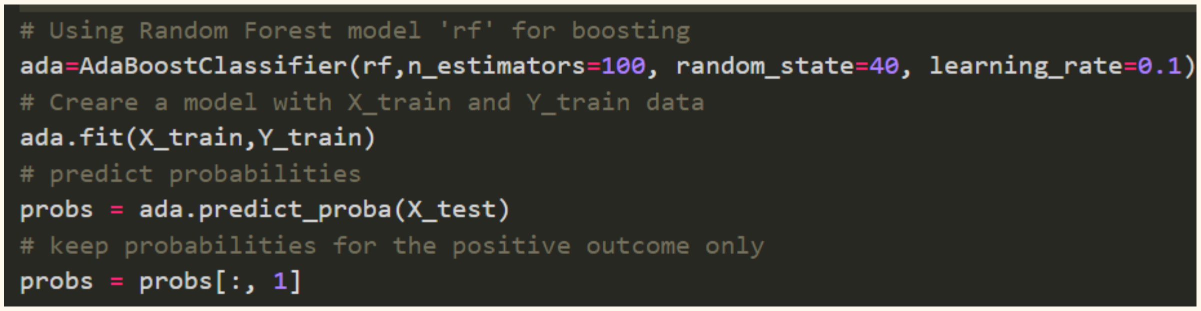
F1 Score for K Nearest Neighbour model is 0.84
51
3
8
28
0
1
0
1
50
40
30
20
10

## AdaBoost Classifier

AdaBoost, also called Adaptive Boosting, is a technique in Machine Learning used as an Ensemble Method. Again AdaBoost is a technique that works on both classification and regression problems. The most common algorithm used with AdaBoost is decision trees with one level. What this algorithm does is that it builds a model and gives equal weights to all the data points. It then assigns higher weights to points that are wrongly classified. Now all the points with higher weights are given more importance in the next model. It will keep training models until and unless a lower error is received.

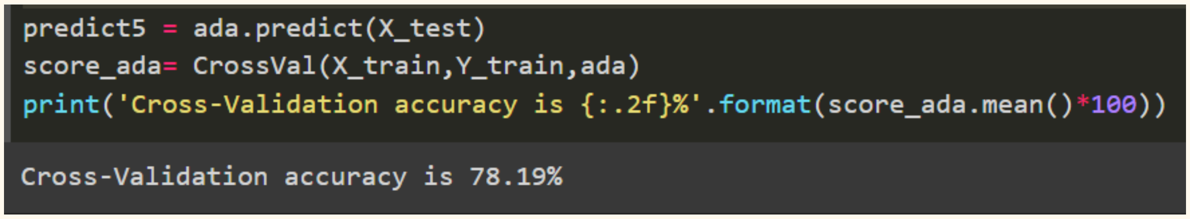
AdaBoost helps you combine multiple "weak classifiers" into a single "strong classifier."

### Code Implementation of AdaBoost:



```
# Using Random Forest model 'rf' for boosting
ada=AdaBoostClassifier(rf,n_estimators=100, random_state=40, learning_rate=0.1)
# Creare a model with X_train and Y_train data
ada.fit(X_train,Y_train)
# predict probabilities
probs = ada.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
```

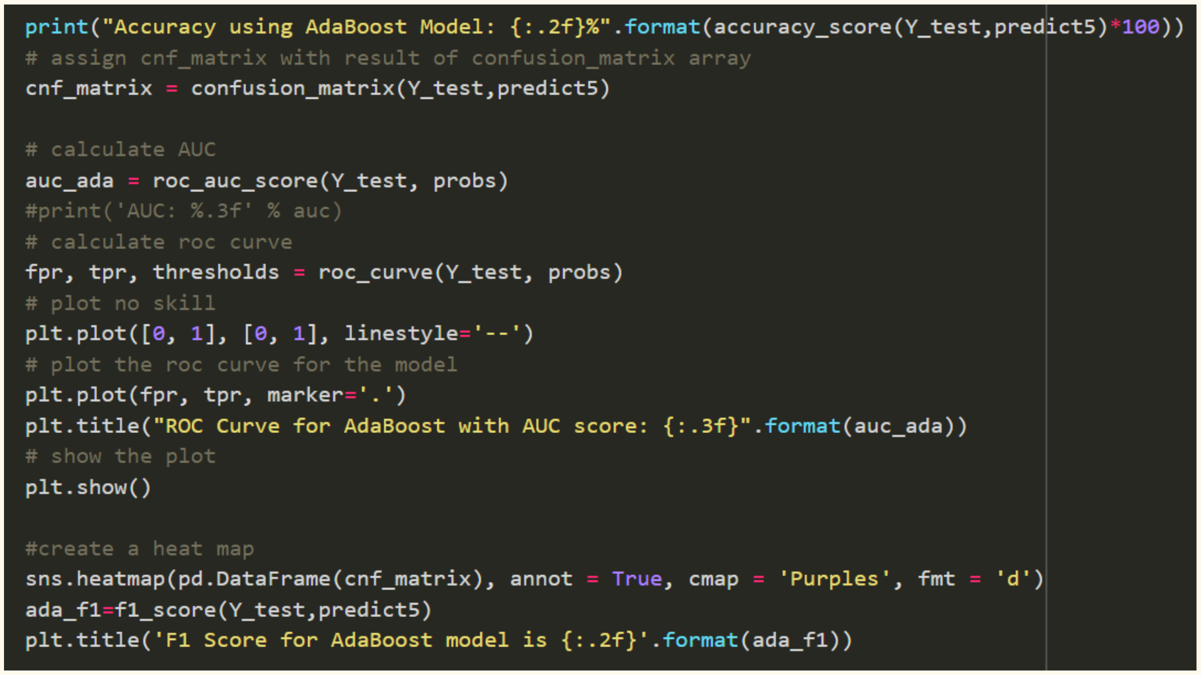
## Accuracy of the Model :



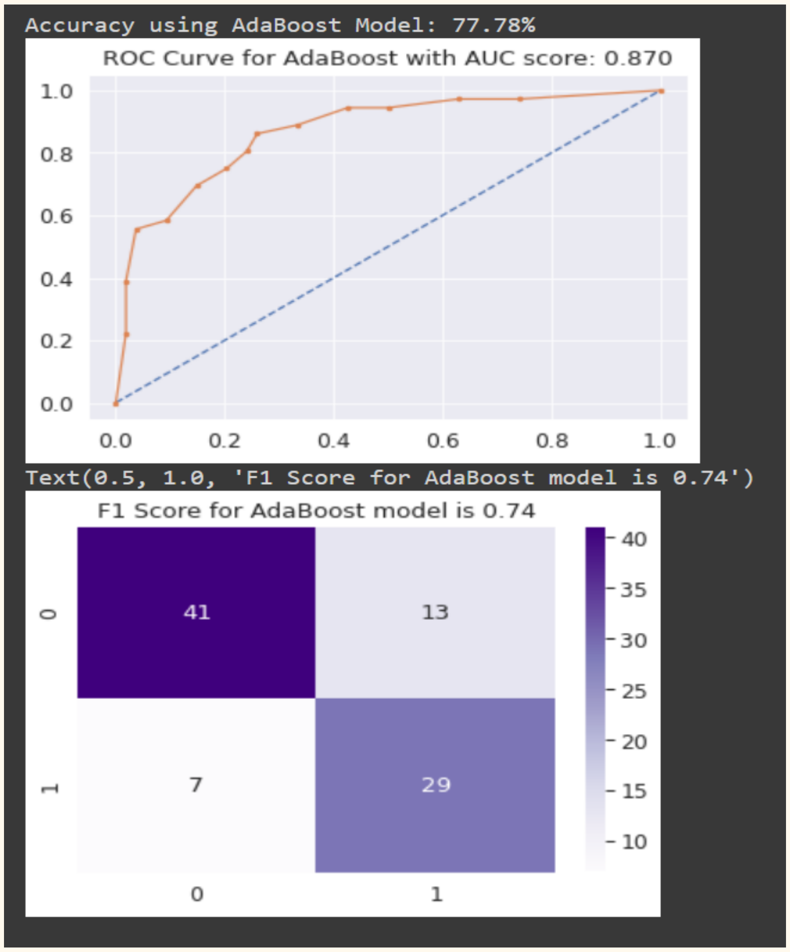
```
predict5 = ada.predict(X_test)
score_ada= CrossVal(X_train,Y_train,ada)
print('Cross-Validation accuracy is {:.2f}%'.format(score_ada.mean()*100))
```

```
Cross-Validation accuracy is 78.19%
```

## Heat Map



```
print("Accuracy using AdaBoost Model: {:.2f}%".format(accuracy_score(Y_test,predict5)*100))
# assign cnf_matrix with result of confusion_matrix array
cnf_matrix = confusion_matrix(Y_test,predict5)

# calculate AUC
auc_ada = roc_auc_score(Y_test, probs)
#print('AUC: %.3f' % auc)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(Y_test, probs)
# plot no skill
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(fpr, tpr, marker='.')
plt.title("ROC Curve for AdaBoost with AUC score: {:.3f}".format(auc_ada))
# show the plot
plt.show()

#create a heat map
sns.heatmap(pd.DataFrame(cnf_matrix), annot = True, cmap = 'Purples', fmt = 'd')
ada_f1=f1_score(Y_test,predict5)
plt.title('F1 Score for AdaBoost model is {:.2f}'.format(ada_f1))
```



```
Accuracy using AdaBoost Model: 77.78%
```

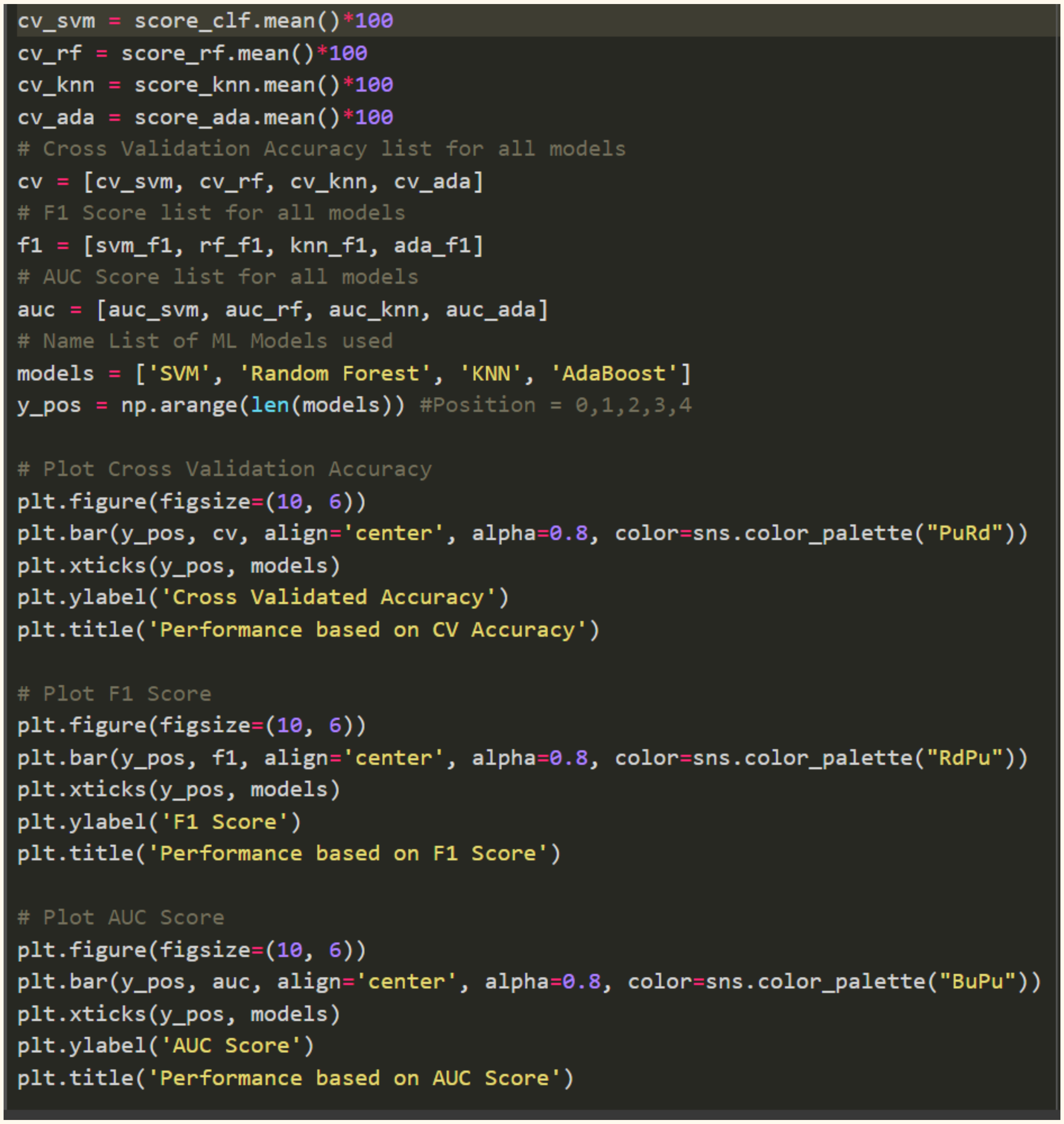
```
Text(0.5, 1.0, 'F1 Score for AdaBoost model is 0.74')
```

## Conclusion

In this project, we first structured the Heart Disease dataset, graphically represented all the information presented to us, cleaned and processed our data, and finally showed the primary way of classifying all this info using some basic ML algorithms using all the knowledge taught to us during the course of our IDS classes.

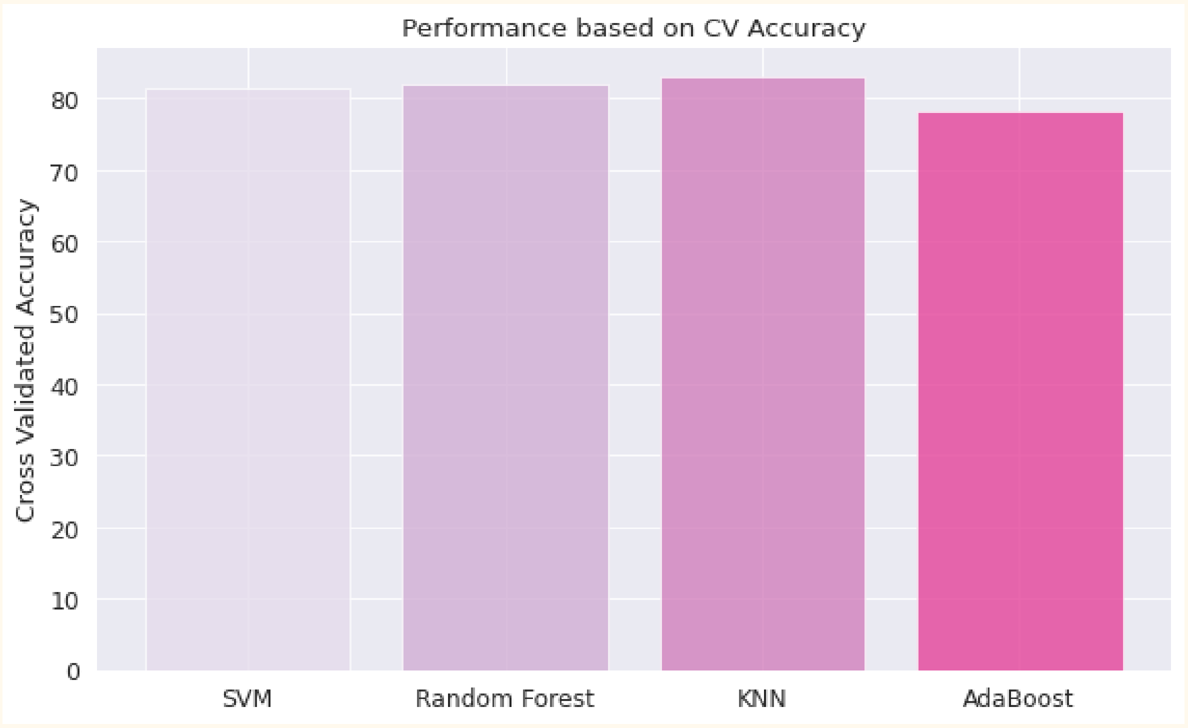
We graphically represented our data using bar charts, box plots, pie charts, etc. After a few more steps, this data was then modeled and used for our different ML algorithms, the accuracy comparison of which can be seen below in our bar graphs of CV accuracy, F1 score and AUC score.

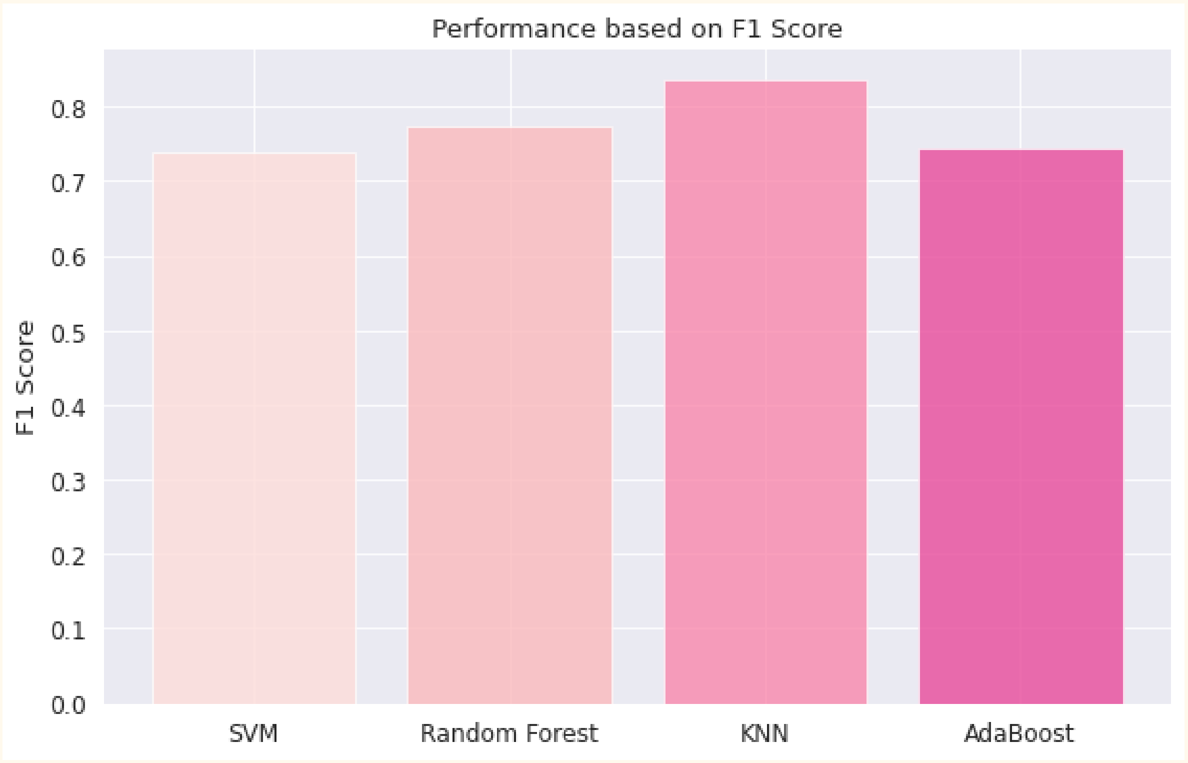
## Code

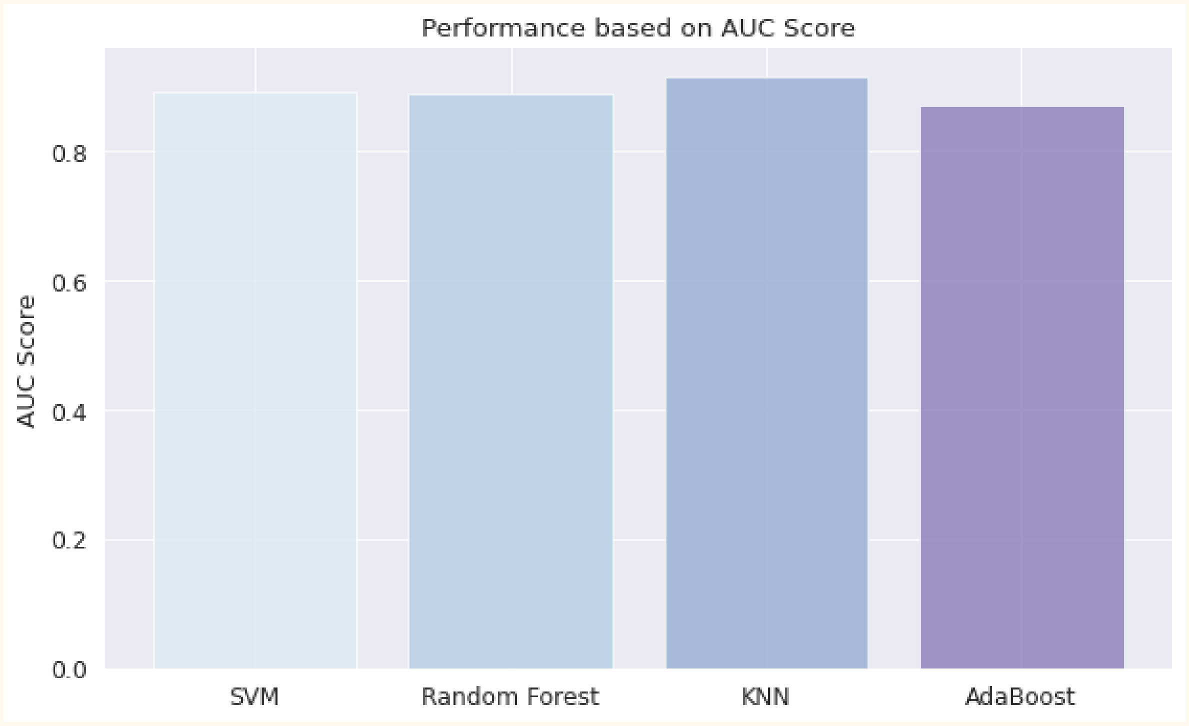


```python
cv_svm = score_clf.mean()*100
cv_rf = score_rf.mean()*100
cv_knn = score_knn.mean()*100
cv_ada = score_ada.mean()*100
# Cross Validation Accuracy list for all models
cv = [cv_svm, cv_rf, cv_knn, cv_ada]
# F1 Score list for all models
f1 = [svm_f1, rf_f1, knn_f1, ada_f1]
# AUC Score list for all models
auc = [auc_svm, auc_rf, auc_knn, auc_ada]
# Name List of ML Models used
models = ['SVM', 'Random Forest', 'KNN', 'AdaBoost']
y_pos = np.arange(len(models)) #Position = 0,1,2,3,4

# Plot Cross Validation Accuracy
plt.figure(figsize=(10, 6))
plt.bar(y_pos, cv, align='center', alpha=0.8, color=sns.color_palette("PuRd"))
plt.xticks(y_pos, models)
plt.ylabel('Cross Validated Accuracy')
plt.title('Performance based on CV Accuracy')

# Plot F1 Score
plt.figure(figsize=(10, 6))
plt.bar(y_pos, f1, align='center', alpha=0.8, color=sns.color_palette("RdPu"))
plt.xticks(y_pos, models)
plt.ylabel('F1 Score')
plt.title('Performance based on F1 Score')

# Plot AUC Score
plt.figure(figsize=(10, 6))
plt.bar(y_pos, auc, align='center', alpha=0.8, color=sns.color_palette("BuPu"))
plt.xticks(y_pos, models)
plt.ylabel('AUC Score')
plt.title('Performance based on AUC Score')
```

## Graph


Performance based on CV Accuracy
Cross Validated Accuracy
0
10
20
30
40
50
60
70
80
SVM
Random Forest
KNN
AdaBoost


Performance based on F1 Score
F1 Score
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
SVM
Random Forest
KNN
AdaBoost



Based on the above comparison plots: Cross-Validated Accuracy and F1 Score, K Nearest Neighbours (KNN) has an ideal balance between underfitting and overfitting along with a high F1 Score. KNN also outperforms the competition with the highest test accuracy and AUC Score.

**Thus we can safely conclude that KNN is the best classifier for the Heart Disease dataset we took.**

## Code Folder

Google Drive

It contains -

the Data set file (heart.csv), The Code file( ids_project.py), The Jupyter NoteBook file (ids_project.ipynb), and this report in pdf format.

Google Colaboratory