Heart Disease Prediction

IMPORTING LIBRARIES

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import rcParams
from matplotlib.cm import rainbow
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn import *

from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
import pickle
```

LOADING DATASET & Preprocess the data

```
data = pd.read_csv("/content/heart.csv")
```

```
data.describe()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
count	1025.000000	1025.000000	1025.000000	1025.000000	1025.00000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000
mean	54.434146	0.695610	0.942439	131.611707	246.00000	0.149268	0.529756	149.114146	0.336585	1.071512	1.385366	0.754146	2.323902	0.513171
std	9.072290	0.460373	1.029641	17.516718	51.59251	0.356527	0.527878	23.005724	0.472772	1.175053	0.617755	1.030798	0.620660	0.500070
min	29.000000	0.000000	0.000000	94.000000	126.00000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	48.000000	0.000000	0.000000	120.000000	211.00000	0.000000	0.000000	132.000000	0.000000	0.000000	1.000000	0.000000	2.000000	0.000000
50%	56.000000	1.000000	1.000000	130.000000	240.00000	0.000000	1.000000	152.000000	0.000000	0.800000	1.000000	0.000000	2.000000	1.000000
75%	61.000000	1.000000	2.000000	140.000000	275.00000	0.000000	1.000000	166.000000	1.000000	1.800000	2.000000	1.000000	3.000000	1.000000

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       1025 non-null   int64  
 1   sex       1025 non-null   int64  
 2   cp        1025 non-null   int64  
 3   trestbps  1025 non-null   int64  
 4   chol      1025 non-null   int64  
 5   fbs       1025 non-null   int64  
 6   restecg   1025 non-null   int64  
 7   thalach   1025 non-null   int64  
 8   exang     1025 non-null   int64  
 9   oldpeak   1025 non-null   float64
 10  slope     1025 non-null   int64  
 11  ca        1025 non-null   int64  
 12  thal      1025 non-null   int64  
 13  target    1025 non-null   int64  
dtypes: float64(1), int64(13)
memory usage: 112.2 KB
```

```
data.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0

```
data.tail()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
1020	59	1	1	140	221	0	1	164	1	0.0	2	0	2	1
1021	60	1	0	125	258	0	0	141	1	2.8	1	1	3	0
1022	47	1	0	110	275	0	0	118	1	1.0	1	1	2	0
1023	50	0	0	110	254	0	0	159	0	0.0	2	0	2	1

Total missing percent of data

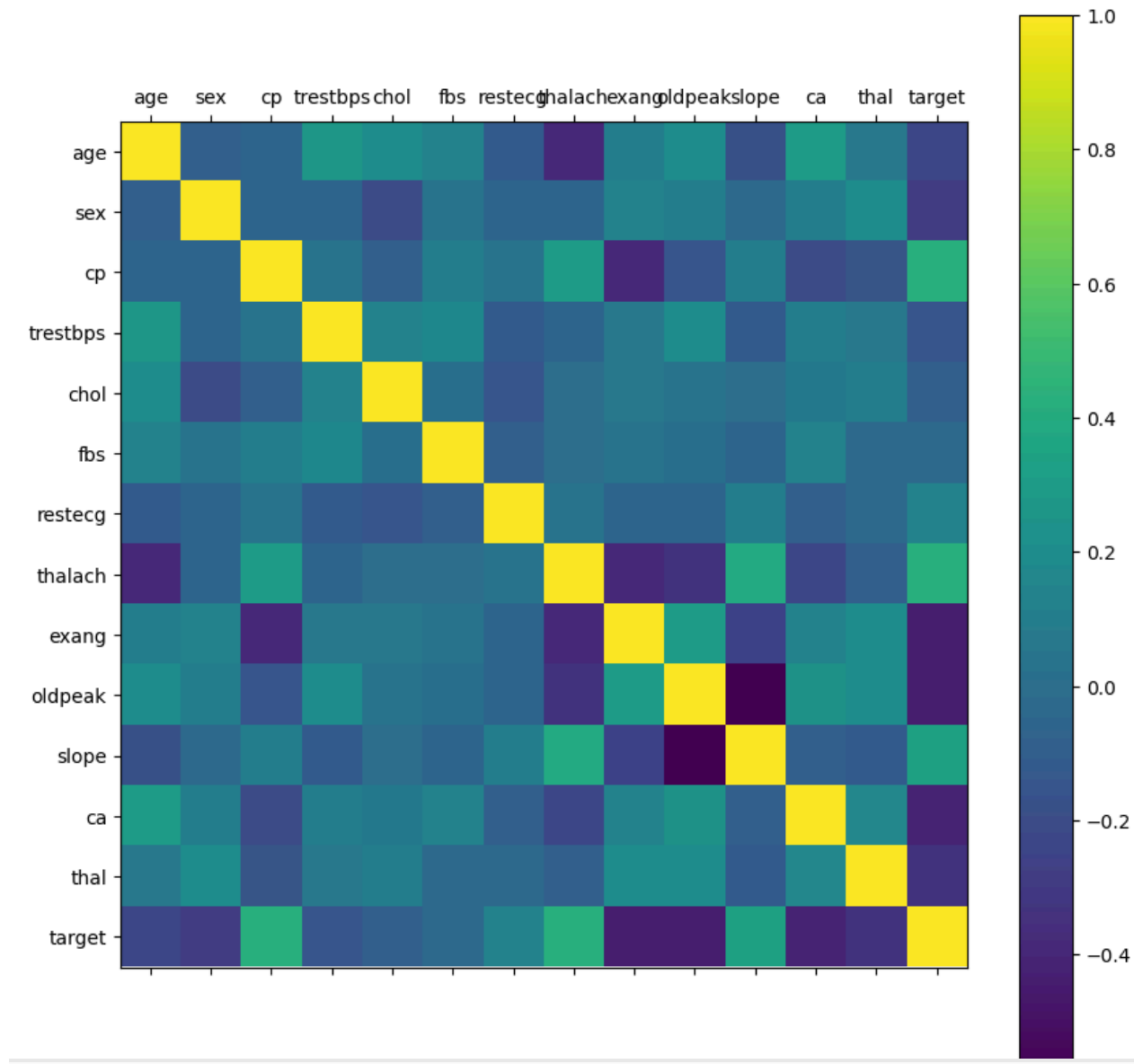
```
missing_data = data.isnull().sum()
total_percentage = (missing_data.sum()/data.shape[0]) * 100
```

```
print(f'Total percentage of missing data is {round(total_percentage,2)}%')
duplicate = data[data.duplicated()]
print("Duplicate rows:")
duplicate
#drop duplicate rows
data = data.drop_duplicates()
```

```
Total percentage of missing data is 0.0%
Duplicate rows:
```

```
rcParams['figure.figsize'] = 10,10
plt.matshow(data.corr())
plt.yticks(np.arange(data.shape[1]), data.columns)
plt.xticks(np.arange(data.shape[1]), data.columns)
plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar at 0x7dc7f2dc62c0>
```



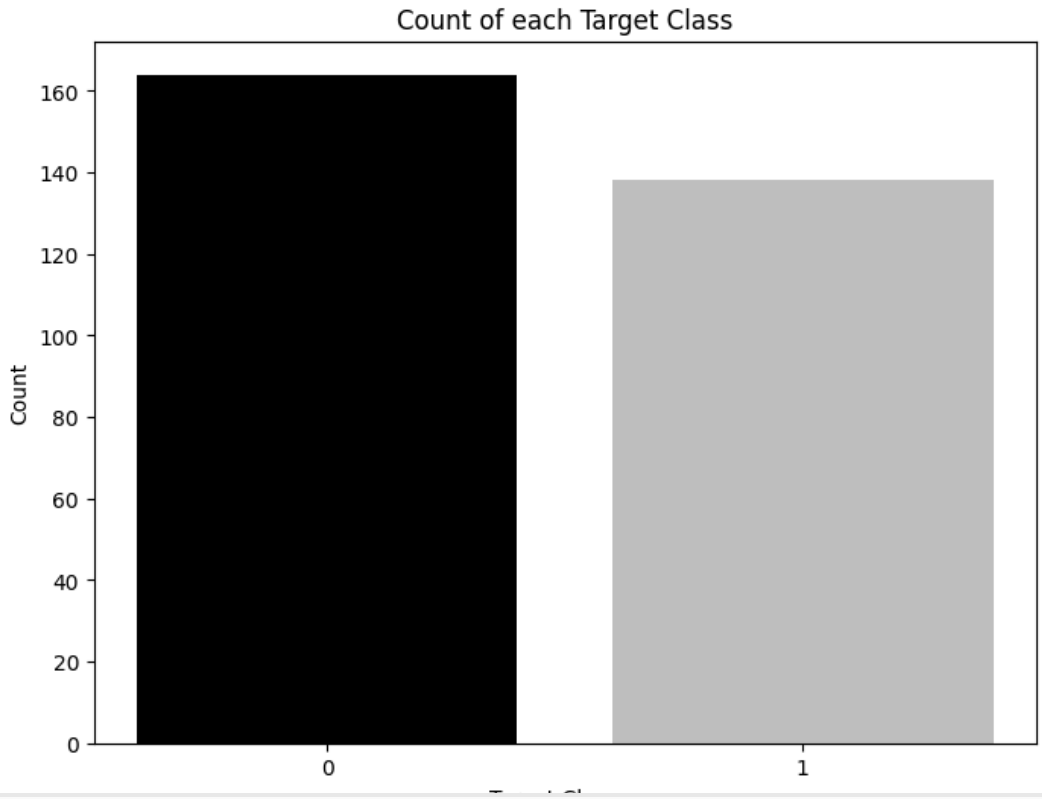

```
corr = data.corr()
corr.style.background_gradient(cmap= 'coolwarm')
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
age	1.000000	-0.094962	-0.063107	0.283121	0.207216	0.119492	-0.111590	-0.395235	0.093216	0.206040	-0.164124	0.302261	0.065317	-0.221476
sex	-0.094962	1.000000	-0.051740	-0.057647	-0.195571	0.046022	-0.060351	-0.046439	0.143460	0.098322	-0.032990	0.113060	0.211452	-0.283609
cp	-0.063107	-0.051740	1.000000	0.046486	-0.072682	0.096018	0.041561	0.293367	-0.392937	-0.146692	0.116854	-0.195356	-0.160370	0.432080
trestbps	0.283121	-0.057647	0.046486	1.000000	0.125256	0.178125	-0.115367	-0.048023	0.068526	0.194600	-0.122873	0.099248	0.062870	-0.146269
chol	0.207216	-0.195571	-0.072682	0.125256	1.000000	0.011428	-0.147602	-0.005308	0.064099	0.050086	0.000417	0.086878	0.096810	-0.081437
fbs	0.119492	0.046022	0.096018	0.178125	0.011428	1.000000	-0.083081	-0.007169	0.024729	0.004514	-0.058654	0.144935	-0.032752	-0.026826
restecg	-0.111590	-0.060351	0.041561	-0.115367	-0.147602	-0.083081	1.000000	0.041210	-0.068807	-0.056251	0.090402	-0.083112	-0.010473	0.134874
thalach	-0.395235	-0.046439	0.293367	-0.048023	-0.005308	-0.007169	0.041210	1.000000	-0.377411	-0.342201	0.384754	-0.228311	-0.094910	0.419955
exang	0.093216	0.143460	-0.392937	0.068526	0.064099	0.024729	-0.068807	-0.377411	1.000000	0.286766	-0.256106	0.125377	0.205826	-0.435601
oldpeak	0.206040	0.098322	-0.146692	0.194600	0.050086	0.004514	-0.056251	-0.342201	0.286766	1.000000	-0.576314	0.236560	0.209090	-0.429146
slope	-0.164124	-0.032990	0.116854	-0.122873	0.000417	-0.058654	0.090402	0.384754	-0.256106	-0.576314	1.000000	-0.092236	-0.103314	0.343940
ca	0.302261	0.113060	-0.195356	0.099248	0.086878	0.144935	-0.083112	-0.228311	0.125377	0.236560	-0.092236	1.000000	0.160085	-0.408992
thal	0.065317	0.211452	-0.160370	0.062870	0.096810	-0.032752	-0.010473	-0.094910	0.205826	0.209090	-0.103314	0.160085	1.000000	-0.343101

Count of each Target Class

```
rcParams['figure.figsize'] = 8,6
plt.bar(data['target'].unique(), data['target'].value_counts(), color = ['black', 'silver'])
plt.xticks([0, 1])
plt.xlabel('Target Classes')
plt.ylabel('Count')
plt.title('Count of each Target Class')
```

```
Text(0.5, 1.0, 'Count of each Target Class')
```



Divide data into training & testing classes

```
X = data.drop(['target'], axis=1)
y = data['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=0)
print("XTrain->", X_train.shape[0], "XTest->", X_test.shape[0], "YTrain->", y_train.shape[0], "YTest->", y_test.shape[0])
```

```
XTrain-> 211 XTest-> 91 YTrain-> 211 YTest-> 91
```

Model Building

KNN Algorithm

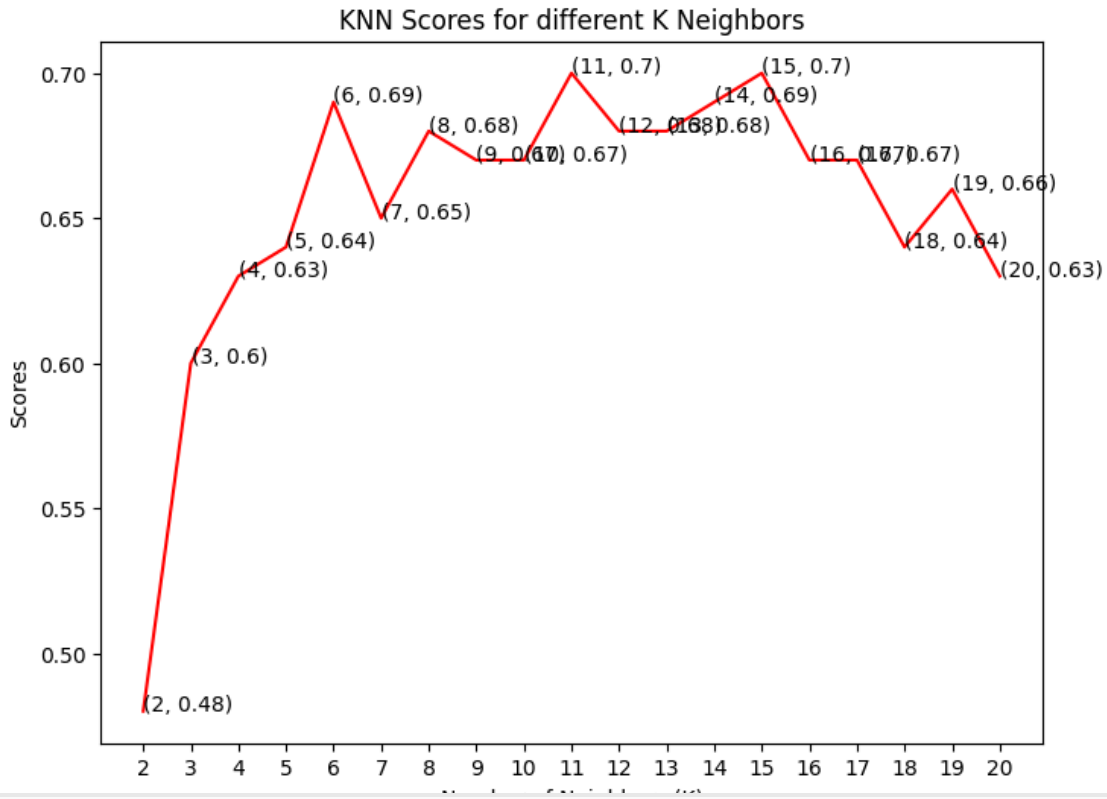
```
knn_scores = []
for k in range(2,21):
  knn_classifier = KNeighborsClassifier(n_neighbors = k)
  knn_classifier.fit(X_train.values, y_train.values)
  knn_score = round(knn_classifier.score(X_test.values, y_test.values),2)
  knn_scores.append(knn_score)


knn_classifier = KNeighborsClassifier(n_neighbors= 5)
knn_classifier.fit(X_train, y_train)
knn_score = knn_classifier.predict(X_test)
print(classification_report(y_test,knn_score))
```

```
              precision    recall  f1-score   support

           0       0.62      0.49      0.55        41
           1       0.64      0.76      0.70        50

    accuracy                           0.64        91
   macro avg       0.63      0.62      0.62        91
weighted avg       0.64      0.64      0.63        91
```

KNN Scores of different K neighbors

```
plt.plot([k for k in range(2, 21)], knn_scores, color = 'red')
for i in range(2,21):
  plt.text(i, knn_scores[i-2], (i, knn_scores[i-2]))
plt.xticks([i for i in range(2,21)])
plt.xlabel('Number of Neighbors (K)')
plt.ylabel('Scores')
plt.title('KNN Scores for different K Neighbors')
```

```
Text(0.5, 1.0, 'KNN Scores for different K Neighbors')
```



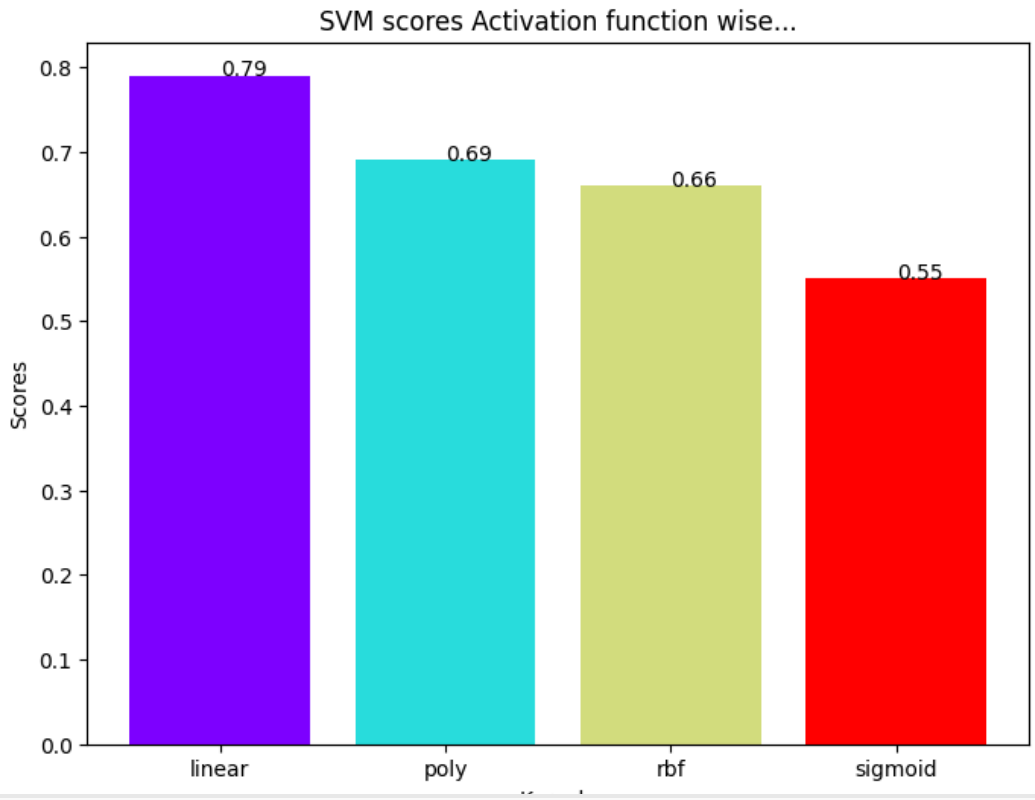
Support Vector Machine

```
from sklearn.metrics import accuracy_score

svc_scores = []
kernels = ['linear', 'poly', 'rbf', 'sigmoid']
for i in range(len(kernels)):
  svc_classifier = SVC(kernel = kernels[i])
  svc_classifier.fit(X_train.values, y_train.values)
  svc_scores.append(round(svc_classifier.score(X_test.values, y_test.values),2))

svc_classifier = SVC(kernel = kernels[0])
svc_classifier.fit(X_train.values, y_train.values)
svc_prediction_result = svc_classifier.predict(X_test.values)
#print(svc_prediction_result)
print(accuracy_score(y_test.values,svc_prediction_result))
```

```
0.7912087912087912
```

```
colors = rainbow(np.linspace(0, 1, len(kernels)))
plt.bar(kernels, svc_scores, color = colors)
for i in range(len(kernels)):
  plt.text(i, svc_scores[i], svc_scores[i])
plt.xlabel('Kernels')
plt.ylabel('Scores')
plt.title('SVM scores Activation function wise...')
```

```
Text(0.5, 1.0, 'SVM scores Activation function wise...')
```



Decision Tree

```
dt_scores = []
for i in range(1, len(X.columns) + 1):
  dt_classifier = DecisionTreeClassifier(max_features = i, random_state = 0)
  dt_classifier.fit(X_train.values, y_train.values)
  dt_scores.append(round(dt_classifier.score(X_test.values, y_test.values),2))
print("Done")
```

```
Done
```

```
print(dt_scores)
```

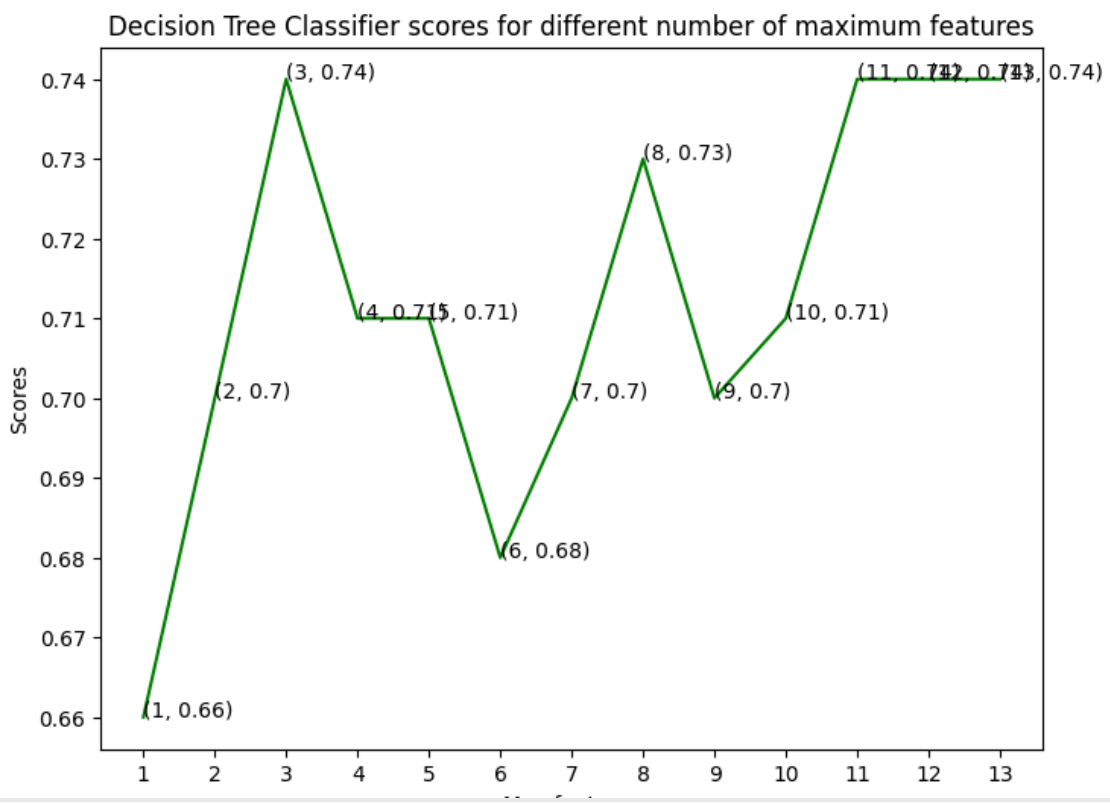
```
[0.66, 0.7, 0.74, 0.71, 0.71, 0.68, 0.7, 0.73, 0.7, 0.71, 0.74, 0.74, 0.74]
```

```
dt_classifier = DecisionTreeClassifier(max_features = 13, random_state = 0)
dt_classifier.fit(X_train.values, y_train.values)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(max_features=13, random_state=0)
```

```
plt.plot([i for i in range(1, len(X.columns) + 1)], dt_scores, color = 'green')
for i in range(1,len(X.columns) + 1):
  plt.text(i, dt_scores[i-1], (i, dt_scores[i-1]))
plt.xticks([i for i in range(1, len(X.columns) + 1)])
plt.xlabel('Max features')
plt.ylabel('Scores')
plt.title('Decision Tree Classifier scores for different number of maximum features')
```

```
Text(0.5, 1.0, 'Decision Tree Classifier scores for different number of maximum features')
```

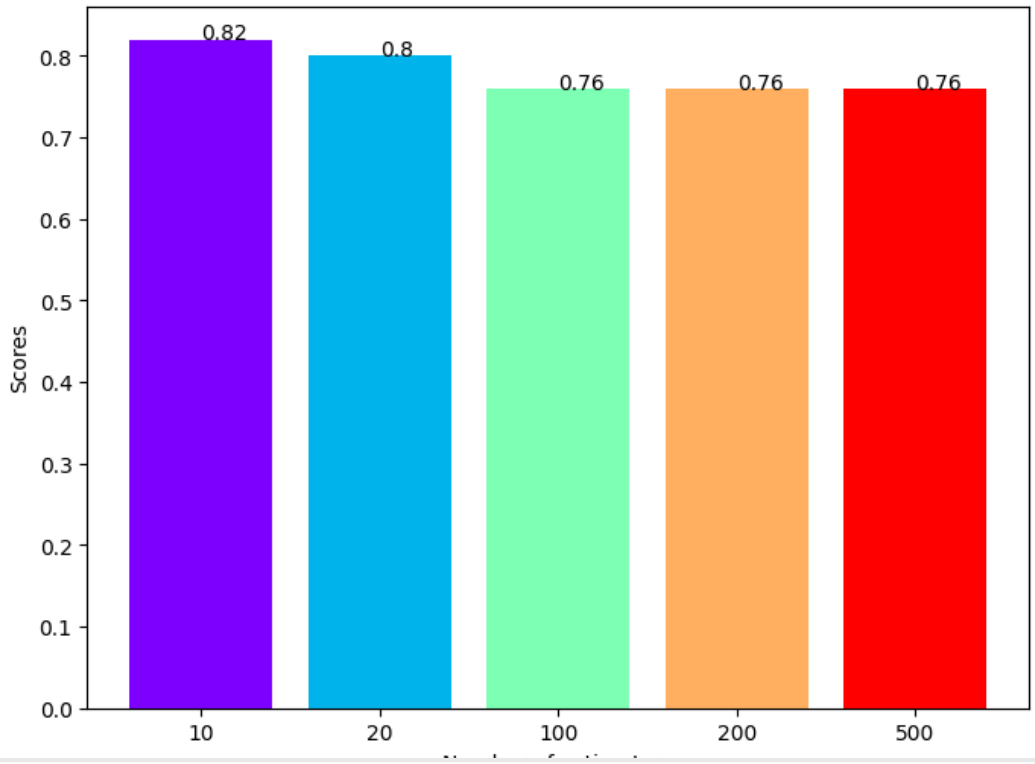


Random Forest

```
rf_model = RandomForestClassifier(n_estimators=100, random_state=0)
rf_scores = []
estimators = [10, 20, 100, 200, 500]
for i in estimators:
  rf_classifier = RandomForestClassifier(n_estimators = i, random_state = 0)
  rf_classifier.fit(X_train.values, y_train.values)
  rf_scores.append(round(rf_classifier.score(X_test.values, y_test.values),2))
```

```
colors = rainbow(np.linspace(0, 1, len(estimators)))
plt.bar([i for i in range(len(estimators))], rf_scores, color = colors, width = 0.8)
for i in range(len(estimators)):
  plt.text(i, rf_scores[i], rf_scores[i])
plt.xticks(ticks = [i for i in range(len(estimators))], labels = [str(estimator) for estimator in estimators])
plt.xlabel('Number of estimators')
plt.ylabel('Scores')
plt.title('Random Forest Classifier scores for different number of estimators')
```

```
Text(0.5, 1.0, 'Random Forest Classifier scores for different number of estimators')
```



Logistic Regression

```
logistic_model = LogisticRegression()
logistic_model.fit(X_train.values, y_train.values)
logistic_model_prediction = logistic_model.predict(X_test.values)
```

```
print(accuracy_score(y_test.values, logistic_model_prediction))
print(classification_report(y_test.values, logistic_model_prediction))
```

```
0.7912087912087912
                precision    recall  f1-score   support

             0       0.84      0.66      0.74        41
             1       0.76      0.90      0.83        50

      accuracy                           0.79        91
     macro avg       0.80      0.78      0.78        91
  weighted avg       0.80      0.79      0.79        91
```

Save Trained Models

```
import pickle
all_models = [rf_model,logistic_model,dt_classifier,svc_classifier,knn_classifier]
with open("models.pkl", 'wb') as files:
  pickle.dump(all_models, files)
print("Done")
```

```
Done
```

```
open_file = open("models.pkl", "rb")
loaded_list = pickle.load(open_file)
print(loaded_list)
open_file.close()
print("Done")
```

```
[RandomForestClassifier(random_state=0), LogisticRegression(), DecisionTreeClassifier(max_features=13, random_state=0), SVC(kernel='linear'), KNeighborsClassifier()]
Done
```

Predict Yes or NO

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import joblib
import pandas as pd
from google.colab import files

# Step 1: Upload your dataset
print("Please upload your dataset with 13 input features and a binary target column (0 or 1).")
uploaded = files.upload()  # Upload file through Google Colab's file uploader

# Step 2: Load the dataset
# Replace 'your_dataset.csv' with the name of your uploaded file if necessary
filename = list(uploaded.keys())[0]
data = pd.read_csv(filename)
X = data.iloc[:, :-1]
y = data.iloc[:, -1]

if X.shape[1] != 13:
    raise ValueError("The input dataset must have exactly 13 features.")

X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.25, random_state=42)
knn_model = KNeighborsClassifier(n_neighbors=5)
```