HEART DISEASE PREDICTION

In [2]:
```python
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')
```

----------------------------------Reading The DataSet--------------------------------------

In [3]:
```python
df = pd.read_csv(r"C:\Users\my\Desktop\Heart_dataset.csv")
```

In [4]:
```python
df
```

Out[4]:

	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

303 rows × 14 columns

In [5]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       303 non-null    int64
 1   sex       303 non-null    int64
 2   cp        303 non-null    int64
 3   trestbps  303 non-null    int64
 4   chol      303 non-null    int64
 5   fbs       303 non-null    int64
 6   restecg   303 non-null    int64
 7   thalach   303 non-null    int64
 8   exang     303 non-null    int64
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64
 11  ca        303 non-null    int64
 12  thal      303 non-null    int64
 13  target    303 non-null    int64
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

In [6]:
```python
df.describe()
```

Out[6]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
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	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	2.313531	0.544554
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	0.612277	0.498835
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	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	2.000000	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	2.000000	1.000000
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	3.000000	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	3.000000	1.000000

In [7]:
```python
df.shape
```

Out[7]:
```
(303, 14)
```

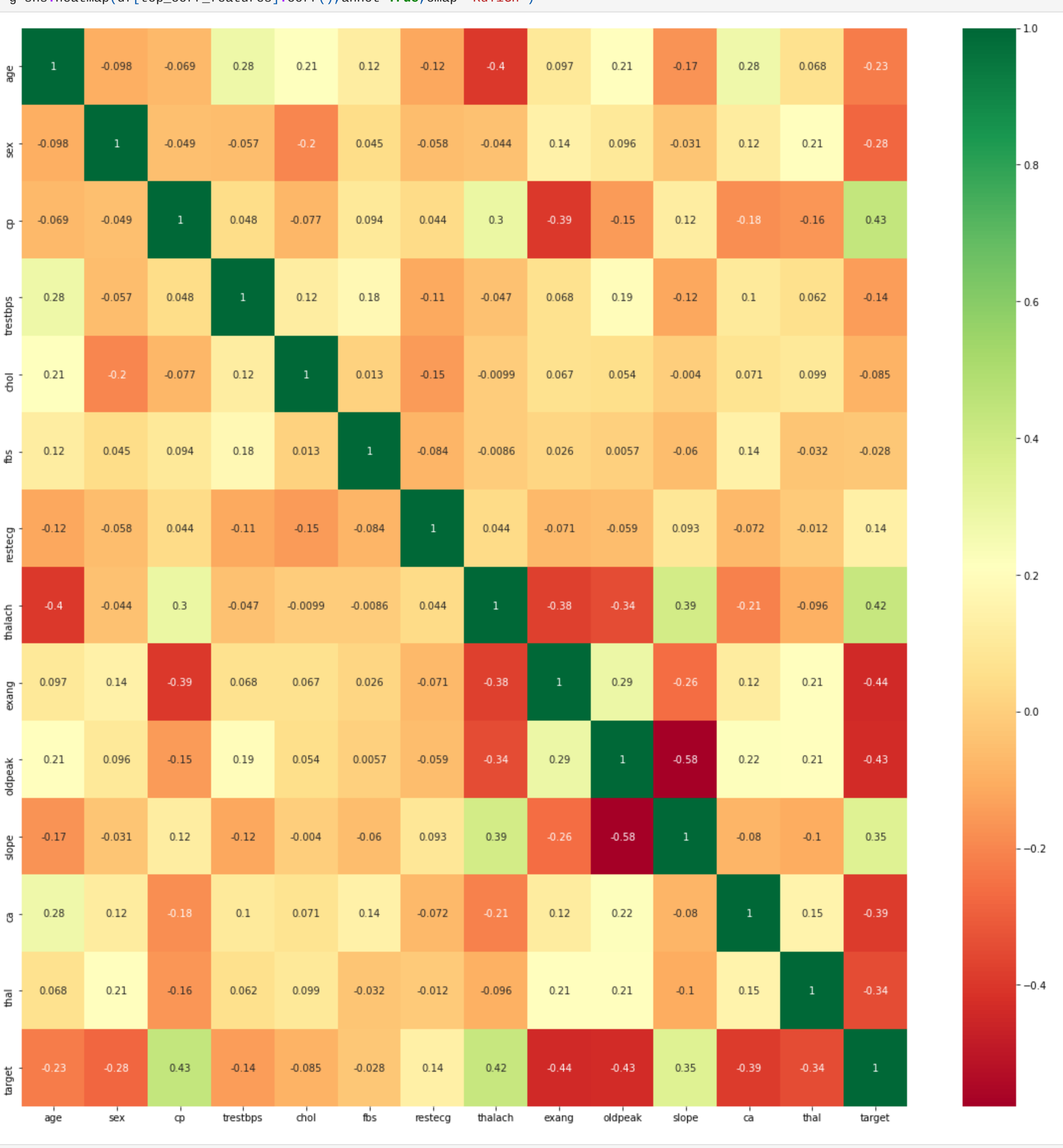
In [8]:
```python
#checking the NA value

df.isna().sum()
```

Out[8]:
```
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
```

Feature Selection

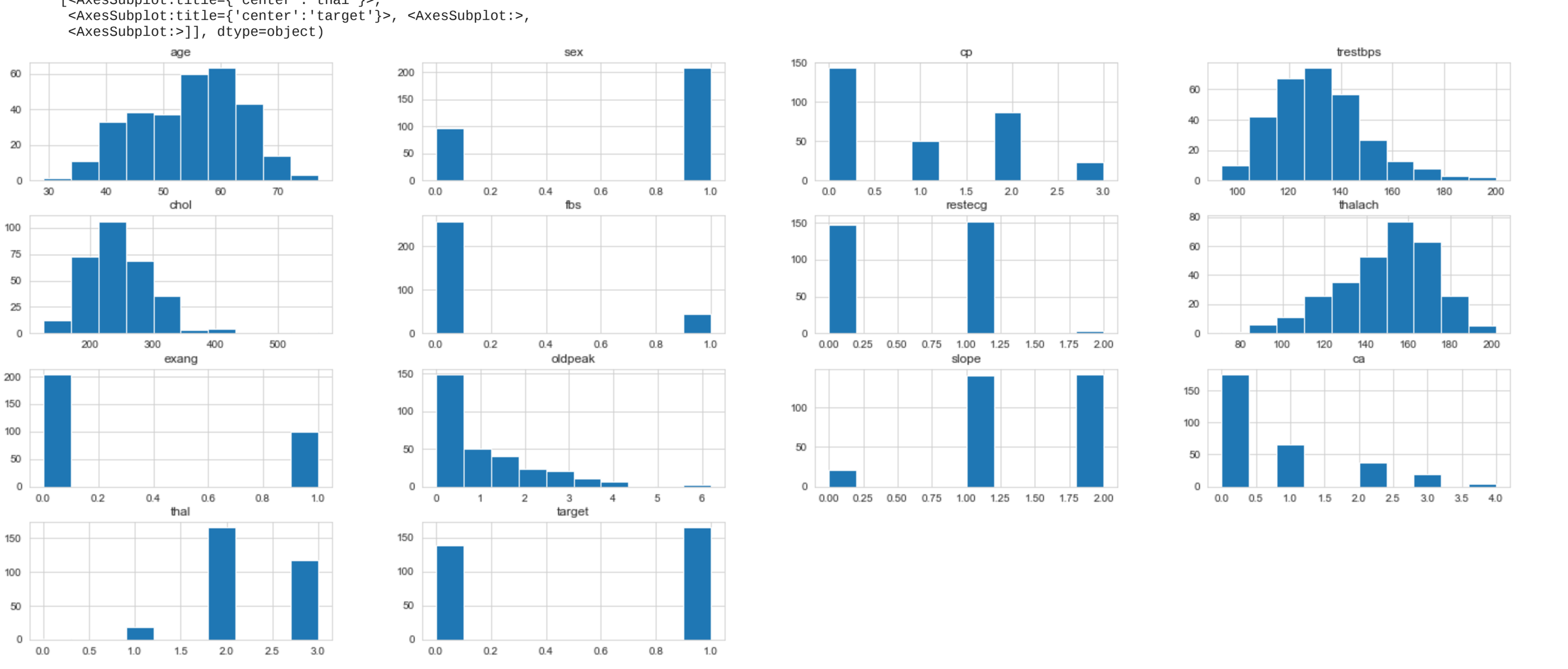
To find out features are positively correlated or negatively correlated

In [9]:
```python
import seaborn as sns
#get correlations of each features in dataset
corrmat = df.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(20,20))
#plot heat map
g=sns.heatmap(df[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```



In [49]:
```python
#Histogram to tell how each feature is distributes

df.hist()
```

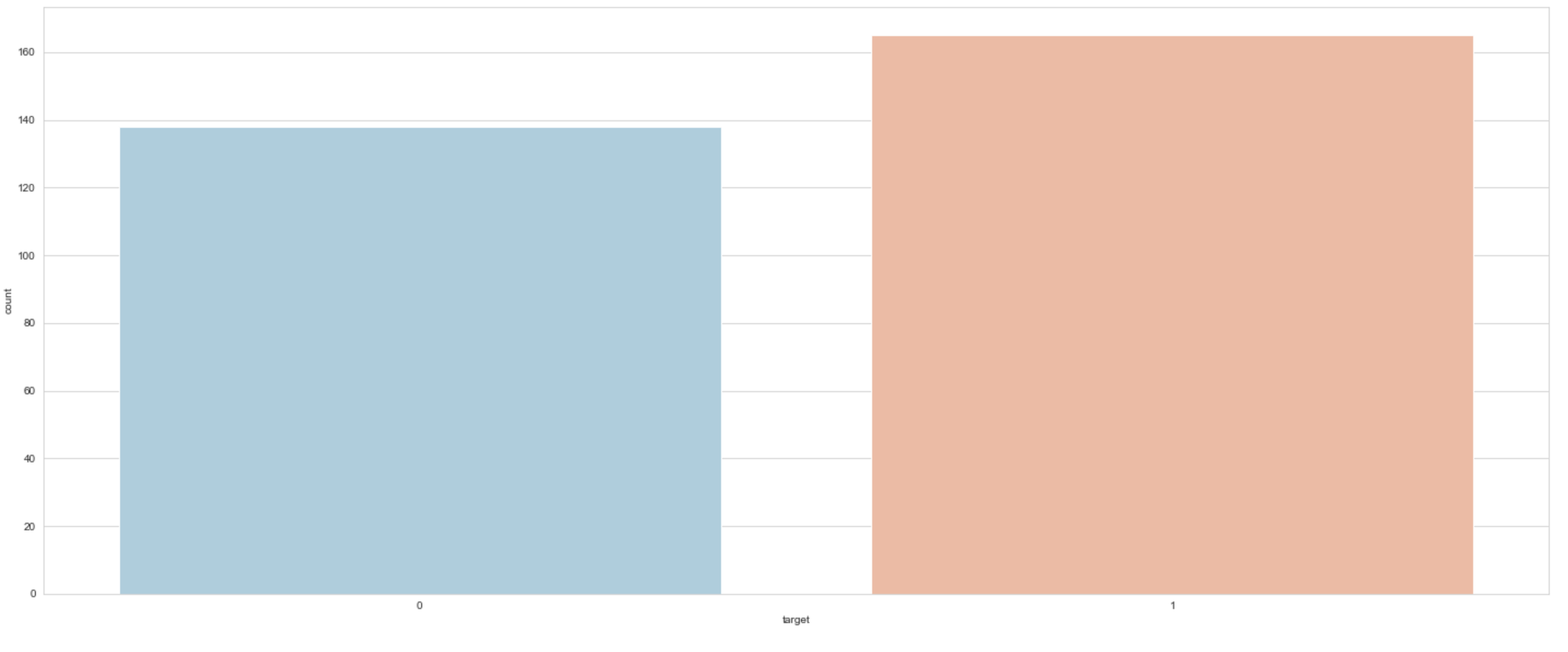
Out[49]:
```
array([[<AxesSubplot:title={'center':'age'}>,
        <AxesSubplot:title={'center':'sex'}>,
        <AxesSubplot:title={'center':'cp'}>,
        <AxesSubplot:title={'center':'trestbps'}>],
       [<AxesSubplot:title={'center':'chol'}>,
        <AxesSubplot:title={'center':'fbs'}>,
        <AxesSubplot:title={'center':'restecg'}>,
        <AxesSubplot:title={'center':'thalach'}>],
       [<AxesSubplot:title={'center':'exang'}>,
        <AxesSubplot:title={'center':'oldpeak'}>,
        <AxesSubplot:title={'center':'slope'}>,
        <AxesSubplot:title={'center':'ca'}>],
       [<AxesSubplot:title={'center':'thal'}>,
        <AxesSubplot:title={'center':'target'}>, <AxesSubplot:>,
        <AxesSubplot:>]], dtype=object)
```



TO FIND THE DATA SET IS BALANCED OR NOT

In [50]:
```python
# Here, 0 is False and 1 means True
sns.set_style('whitegrid')
sns.countplot(x='target',data=df,palette='RdBu_r')
```

Out[50]:
```
<AxesSubplot:xlabel='target', ylabel='count'>
```



------------------------------------ DATA PROCESSING--------------------------------------

Some of the attributes in the DataSet are of categorical variable and convert them into dummy data and to do this get_dummies function is used

In [12]:
```python
dataset = pd.get_dummies(df, columns = ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca', 'thal'])
```

Doing StandardScaling as some of the variables have many variations and different units and this will help to scale down the values within the standard distribution

In [13]:
```python
from sklearn.preprocessing import StandardScaler
standardScaler = StandardScaler()
columns_to_scale = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
dataset[columns_to_scale] = standardScaler.fit_transform(dataset[columns_to_scale])
```

In [14]:
```python
dataset.head()
```

Out[14]:

	age	trestbps	chol	thalach	oldpeak	target	sex_0	sex_1	cp_0	cp_1	...	slope_2	ca_0	ca_1	ca_2	ca_3	ca_4	thal_0	thal_1	thal_2	thal_3
0	0.952197	0.763956	-0.256334	0.015443	1.087338	1	0	1	0	0	...	0	1	0	0	0	0	0	1	0	0
1	-1.915313	-0.092738	0.072199	1.633471	2.122573	1	0	1	0	0	...	0	1	0	0	0	0	0	0	1	0
2	-1.474158	-0.092738	-0.816773	0.977514	0.310912	1	1	0	0	1	...	1	1	0	0	0	0	0	0	1	0
3	0.180175	-0.663867	-0.198357	1.239897	-0.206705	1	0	1	0	1	...	1	1	0	0	0	0	0	0	1	0
4	0.290464	-0.663867	2.082050	0.583939	-0.379244	1	1	0	1	0	...	1	1	0	0	0	0	0	0	1	0

5 rows × 31 columns

Splitting the dataset

In [15]:
```python
from sklearn.model_selection import train_test_split
```

creating the dependent and independent features

In [16]:
```python
Y_train = dataset['target']
X_train = dataset.drop(['target'], axis = 1)
```

In [17]:
```python
# to apply cross validation

from sklearn.model_selection import cross_val_score
```

In [18]:
```python
X_train.head()
```

Out[18]:

	age	trestbps	chol	thalach	oldpeak	sex_0	sex_1	cp_0	cp_1	cp_2	...	slope_2	ca_0	ca_1	ca_2	ca_3	ca_4	thal_0	thal_1	thal_2	thal_3
0	0.952197	0.763956	-0.256334	0.015443	1.087338	0	1	0	0	0	...	0	1	0	0	0	0	0	1	0	0
1	-1.915313	-0.092738	0.072199	1.633471	2.122573	0	1	0	0	1	...	0	1	0	0	0	0	0	0	1	0
2	-1.474158	-0.092738	-0.816773	0.977514	0.310912	1	0	0	1	0	...	1	1	0	0	0	0	0	0	1	0
3	0.180175	-0.663867	-0.198357	1.239897	-0.206705	0	1	0	1	0	...	1	1	0	0	0	0	0	0	1	0
4	0.290464	-0.663867	2.082050	0.583939	-0.379244	1	0	1	0	0	...	1	1	0	0	0	0	0	0	1	0

5 rows × 30 columns

In [19]:
```python
Y_train.head()
```

Out[19]:
```
0    1
1    1
2    1
3    1
4    1
Name: target, dtype: int64
```

In [20]:
```python
x_train, x_test, y_train, y_test = train_test_split(X_train, Y_train, test_size=0.3)
```

APPLYING DIFFERENT MACHINE LEARNING TECHNIQUES

-----Support Vector Machine

In [21]:
```python
from sklearn.svm import SVC
```

In [22]:
```python
sv=SVC(kernel='linear')
sv.fit(X_train,Y_train)
```

Out[22]:
```
SVC(kernel='linear')
```

In [23]:
```python
pred = sv.predict(x_test)
pred
```

Out[23]:
```
array([1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0,
       1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0,
       1, 0, 0], dtype=int64)
```

In [24]:
```python
from sklearn.metrics import accuracy_score
score = accuracy_score(y_test,pred)
print("Testing Accuracy of SVM model is :",score)
```

```
Testing Accuracy of SVM model is : 0.8791208791208791
```

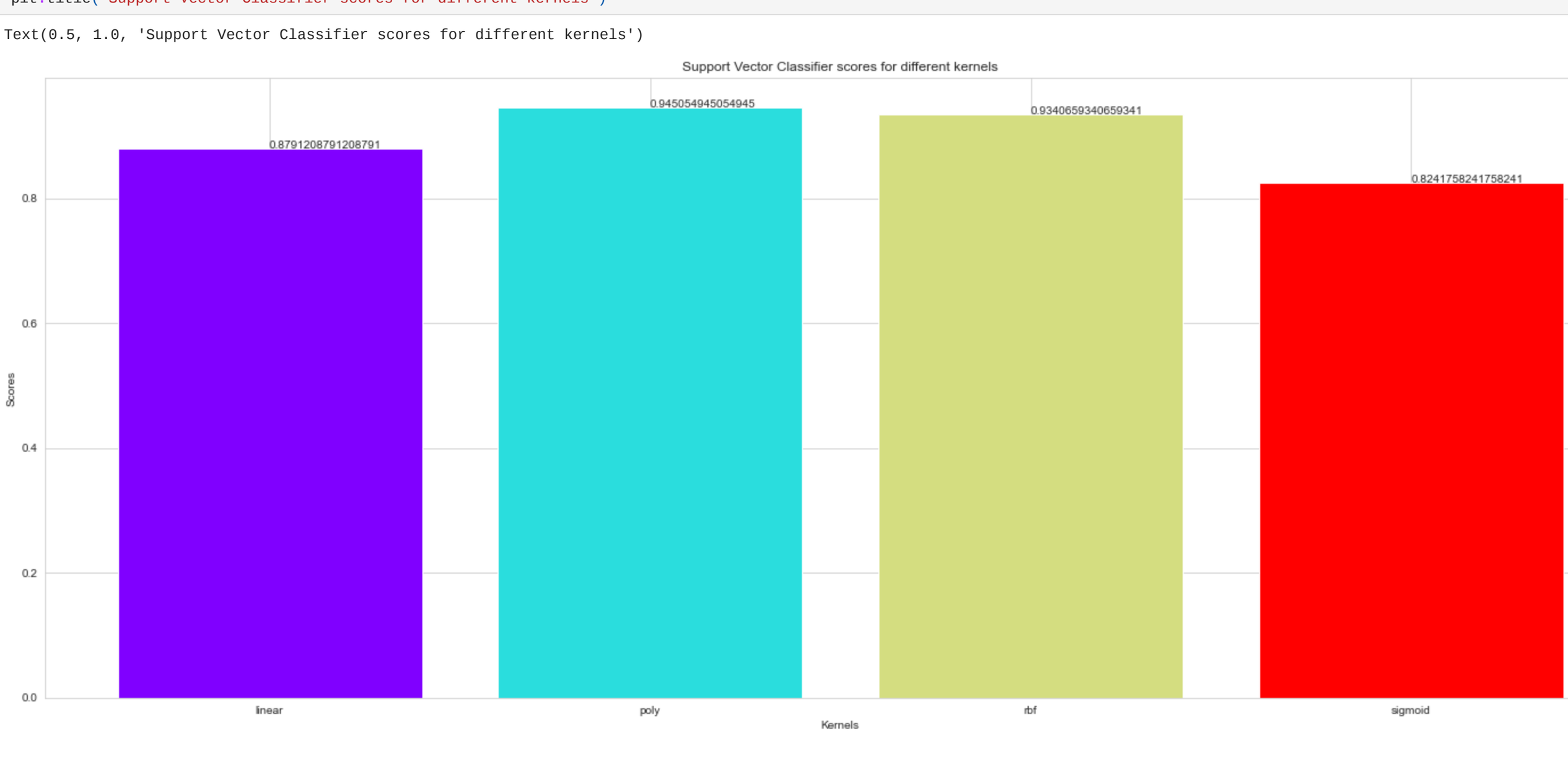
In [25]:
```python
from sklearn.metrics import classification_report

print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.83      0.85      0.84        34
           1       0.91      0.89      0.90        57

    accuracy                           0.88        91
   macro avg       0.87      0.87      0.87        91
weighted avg       0.88      0.88      0.88        91
```

In [26]:
```python
svc_scores = []
kernels = ['linear', 'poly', 'rbf', 'sigmoid']
for i in range(len(kernels)):
    svc_classifier = SVC(kernel = kernels[i])
    svc_classifier.fit(X_train, Y_train)
    svc_scores.append(svc_classifier.score(x_test, y_test))
```

In [51]:
```python
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('Support Vector Classifier scores for different kernels')
```

Out[51]:
```
Text(0.5, 1.0, 'Support Vector Classifier scores for different kernels')
```



-----KNeighborsClassifier

In [28]:
```python
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2)
classifier.fit(x_train, y_train)
```

Out[28]:
```
KNeighborsClassifier()
```

In [29]:
```python
pred = classifier.predict(x_test)
pred
```

Out[29]:
```
array([1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
       1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0,
       1, 0, 0], dtype=int64)
```

In [30]:
```python
acc=accuracy_score(y_test,pred)
print("Testing Accuracy of SVM model is :",acc)
```

```
Testing Accuracy of SVM model is : 0.8241758241758241
```

In [31]:
```python
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,pred)
```

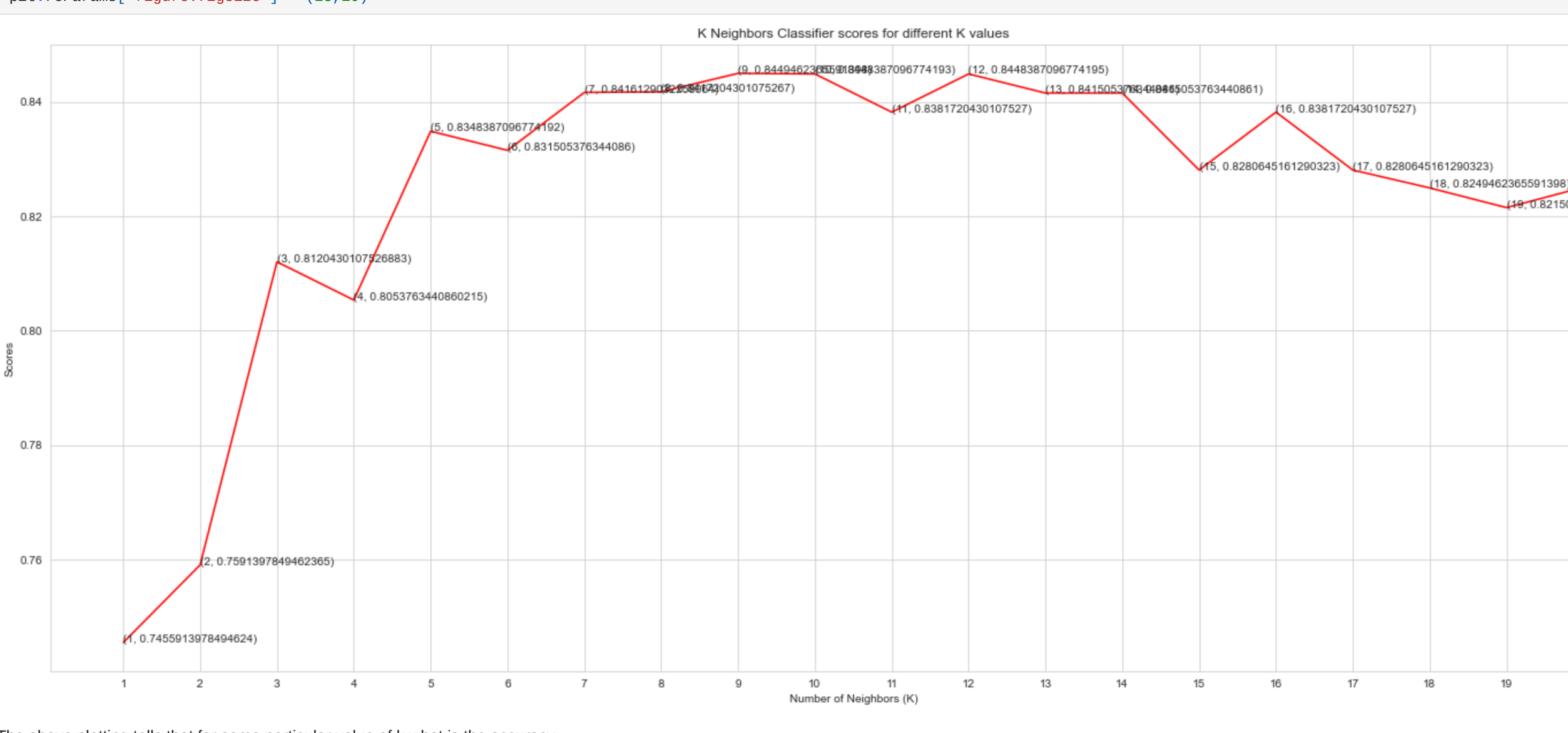
Out[31]:
```
array([[28,  6],
       [10, 47]], dtype=int64)
```

In [32]:
```python
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.74      0.82      0.78        34
           1       0.89      0.82      0.85        57

    accuracy                           0.82        91
   macro avg       0.81      0.82      0.82        91
weighted avg       0.83      0.82      0.83        91
```

In [33]:
```python
from sklearn.model_selection import cross_val_score
knn_scores = []
for k in range(1,21):
    knn_classifier = KNeighborsClassifier(n_neighbors = k)
    score=cross_val_score(knn_classifier,X_train,Y_train,cv=10)
    knn_scores.append(score.mean())
```

In [48]:
```python
plt.plot([k for k in range(1, 21)], knn_scores, color = 'red')
for i in range(1,21):
    plt.text(i, knn_scores[i-1], (i, knn_scores[i-1]))
plt.xticks([i for i in range(1, 21)])
plt.xlabel('Number of Neighbors (K)')
plt.ylabel('Scores')
plt.title('K Neighbors Classifier scores for different K values')
plt.rcParams['figure.figsize'] = (25,10)
```



The above plotting tells that for some particular value of k what is the accuracy

------DecisionTreeClassifier

In [35]:
```python
from sklearn.tree import DecisionTreeClassifier
dc= DecisionTreeClassifier()
dc.fit(x_train,y_train)
```

Out[35]:
```
DecisionTreeClassifier()
```

In [36]:
```python
pred=dc.predict(x_test)
pred
```

Out[36]:
```
array([1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0,
       1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1,
       0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0,
       1, 0, 0], dtype=int64)
```

In [37]:
```python
acc=accuracy_score(y_test,pred)
print("Testing Accuracy of SVM model is :",acc)
```

```
Testing Accuracy of SVM model is : 0.7582417582417582
```

In [38]:
```python
confusion_matrix(y_test,pred)
```

Out[38]:
```
array([[26,  8],
       [14, 43]], dtype=int64)
```

In [39]:
```python
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.65      0.76      0.70        34
           1       0.84      0.75      0.80        57

    accuracy                           0.76        91
   macro avg       0.75      0.76      0.75        91
weighted avg       0.77      0.76      0.76        91
```

In [40]:
```python
dc.feature_importances_
```

Out[40]:
```
array([0.15302667, 0.07150374, 0.06475051, 0.10087257, 0.12189157,
       0.00393222, 0.        , 0.06087073, 0.        , 0.00943732,
       0.02657263, 0.0125831 , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.01993191, 0.        , 0.01927117,
       0.02683041, 0.        , 0.01481249, 0.        , 0.        ,
       0.        , 0.        , 0.0138986 , 0.27981437, 0.        ])
```

------ RandomForestClassifier

In [41]:
```python
from sklearn.ensemble import RandomForestClassifier
rf= RandomForestClassifier(n_estimators=10 ,criterion='entropy',random_state=0)
rf.fit(x_train,y_train)
```

Out[41]:
```
RandomForestClassifier(criterion='entropy', n_estimators=10, random_state=0)
```

In [42]:
```python
 pred=rf.predict(x_test)
pred
```

Out[42]:
```
array([1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1,
       0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0,
       1, 0, 0], dtype=int64)
```

In [43]:
```python
acc=accuracy_score(y_test,pred)
print("Testing Accuracy of SVM model is :",acc)
```

```
Testing Accuracy of SVM model is : 0.7802197802197802
```

In [44]:
```python
confusion_matrix(y_test,pred)
```

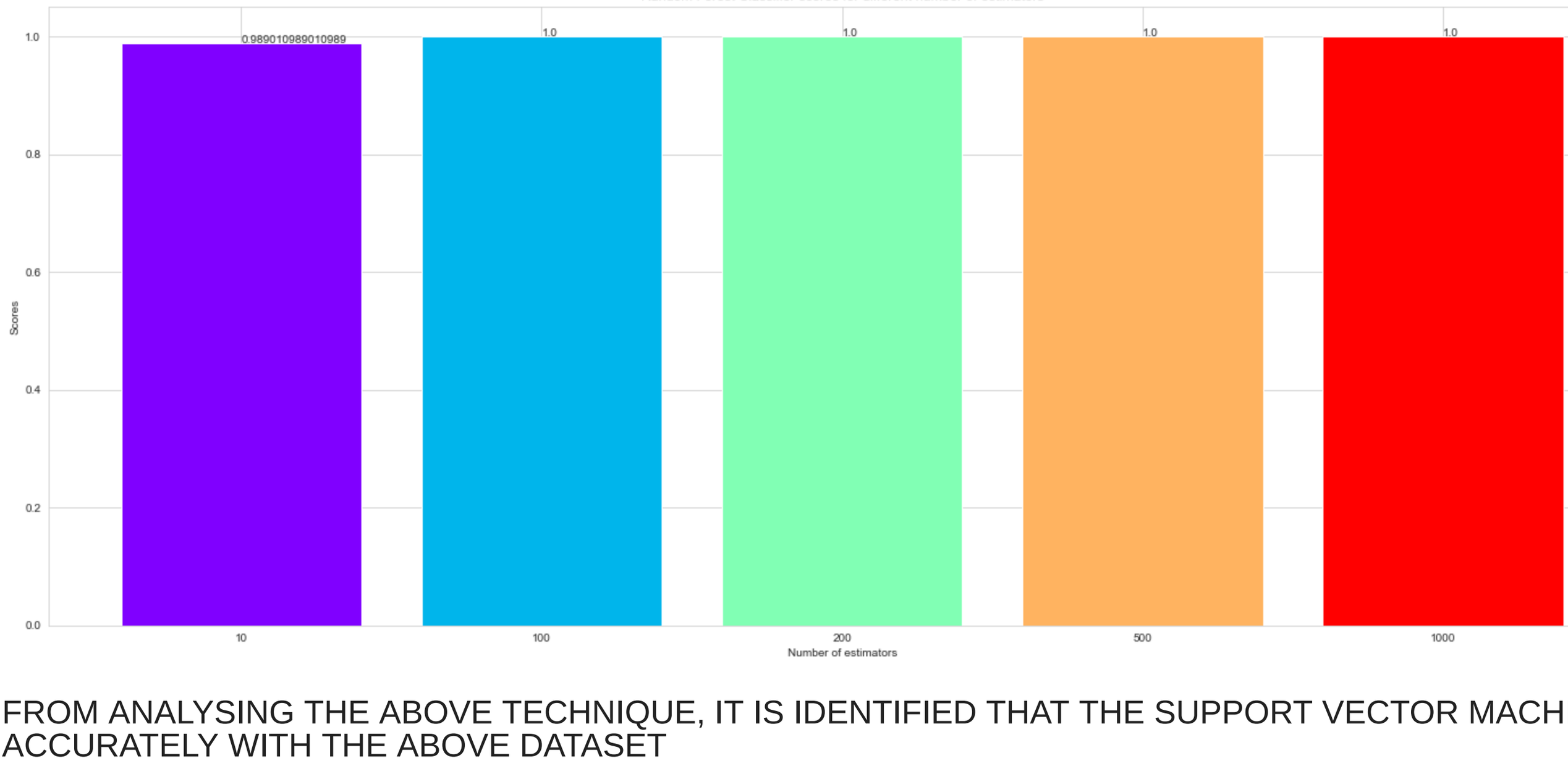
Out[44]:
```
array([[30,  4],
       [16, 41]], dtype=int64)
```

In [45]:
```python
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.65      0.88      0.75        34
           1       0.91      0.72      0.80        57

    accuracy                           0.78        91
   macro avg       0.78      0.80      0.78        91
weighted avg       0.81      0.78      0.78        91
```

In [46]:
```python
rf_scores = []
estimators = [10, 100, 200, 500, 1000]
for i in estimators:
    rf_classifier = RandomForestClassifier(n_estimators = i, random_state = 0)
    rf_classifier.fit(X_train, Y_train)
    rf_scores.append(rf_classifier.score(x_test, y_test))
```

In [47]:
```python
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')
plt.rcParams['figure.figsize'] = (25,10)
```



FROM ANALYSING THE ABOVE TECHNIQUE, IT IS IDENTIFIED THAT THE SUPPORT VECTOR MACHINE WORKS MORE ACCURATELY WITH THE ABOVE DATASET

In [ ]: