Heart Attack Prediction Using Different ML Models

Importing necessary libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

Importing and reading the data.

In [2]:
```
data = pd.read_csv('heart.csv')
data.head()
```

Out[2]:

	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

In [3]:
```
data.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 [4]:
```
data.shape
```

Out[4]: (303, 14)

In [5]:
```
data.describe()
```

Out[5]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach
count	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
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000

Information about the data set we got.

Attribute Information

1) age

2) sex

3) chest pain type (4 values)

4) resting blood pressure

5) serum cholestoral in mg/dl

6) fasting blood sugar > 120 mg/dl

7) resting electrocardiographic results (values 0,1,2)

8) maximum heart rate achieved

9) exercise induced angina

10) oldpeak = ST depression induced by exercise relative to rest

11) the slope of the peak exercise ST segment

12) number of major vessels (0-3) colored by flourosopy

13) thal: 0 = normal; 1 = fixed defect; 2 = reversable defect

14) target: 0= less chance of heart attack 1= more chance of heart attack

EDA

Finding null values

In [6]:

```
print(data.isnull().sum())
# sum will sums up the number of True values (i.e., missing values) along each co
#the count shows how much missing values are there
```

```
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
```

In [7]:
```
print(data.isnull())
#True means have null values and False means no null values
```

```
       age    sex     cp  trestbps   chol    fbs  restecg  thalach  exang  \
0    False  False  False     False  False  False    False    False  False   
1    False  False  False     False  False  False    False    False  False   
2    False  False  False     False  False  False    False    False  False   
3    False  False  False     False  False  False    False    False  False   
4    False  False  False     False  False  False    False    False  False   
..     ...    ...    ...       ...    ...    ...      ...      ...    ...   
298  False  False  False     False  False  False    False    False  False   
299  False  False  False     False  False  False    False    False  False   
300  False  False  False     False  False  False    False    False  False   
301  False  False  False     False  False  False    False    False  False   
302  False  False  False     False  False  False    False    False  False   

     oldpeak  slope     ca   thal  target  
0      False  False  False  False   False  
1      False  False  False  False   False  
2      False  False  False  False   False  
3      False  False  False  False   False  
4      False  False  False  False   False  
..       ...    ...    ...    ...     ...  
298    False  False  False  False   False  
299    False  False  False  False   False  
300    False  False  False  False   False  
301    False  False  False  False   False  
302    False  False  False  False   False  

[303 rows x 14 columns]
```

In [8]:
```
print(data['age'].isnull().sum())

```

```
0
```

In [9]:
```
print(data['sex'].isnull().sum())
```

```
0
```

Finding Duplicate values

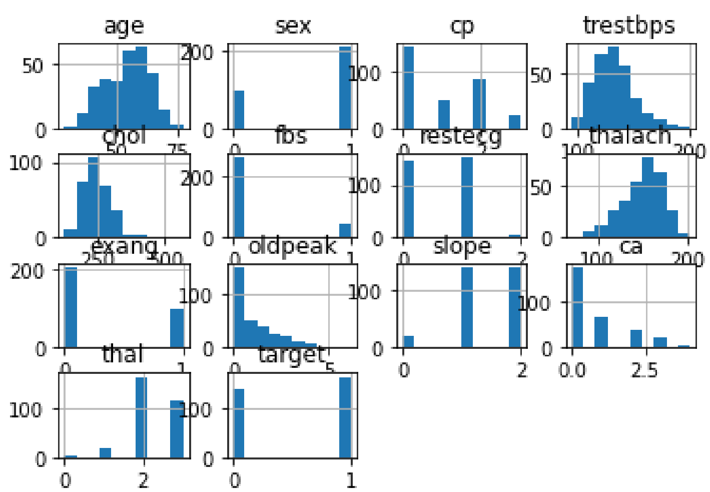
In [10]:
```
# Check for duplicate rows in the entire DataFrame
duplicate_rows = data[data.duplicated()]
print("Duplicate rows:")
print(duplicate_rows)
```

```
Duplicate rows:
     age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
164   38    1   2       138   175    0        1      173      0      0.0   

     slope  ca  thal  target  
164      2   4     2       1  
```

Understanding data with the help of visualization.

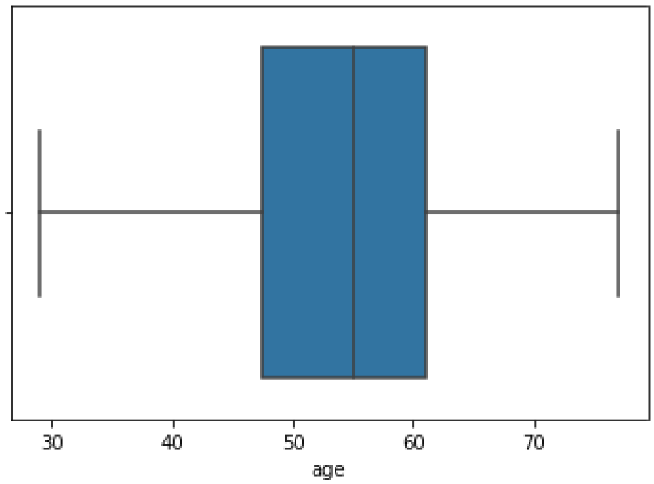
In [11]:
```
#histogram
data.hist()
```

Out[11]:
```
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)
```



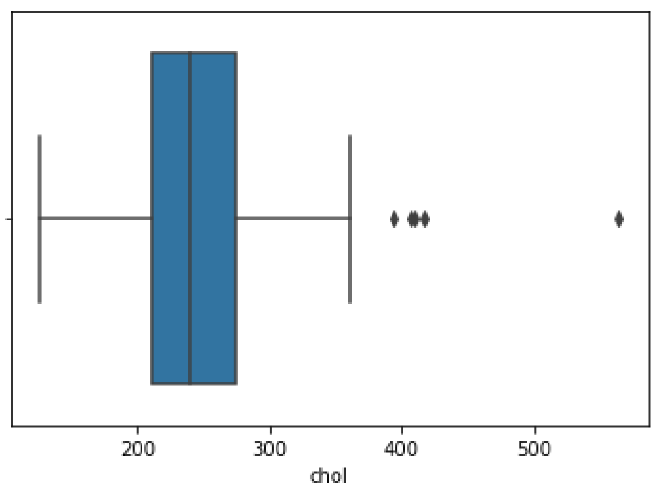
In [12]:
```
#boxplot
sns.boxplot(x='age', data=data)
```

Out[12]: <AxesSubplot:xlabel='age'>



In [13]:
```
sns.boxplot(x='chol', data=data)
```

Out[13]: <AxesSubplot:xlabel='chol'>



In [14]:
```
#cooreLation
data.corr()
```

cp	-0.068653	-0.049353	1.000000	0.047608	-0.076904	0.094444	0.044421	0.295762	-0.39428
trestbps	0.279351	-0.056769	0.047608	1.000000	0.123174	0.177531	-0.114103	-0.046698	0.06761
chol	0.213678	-0.197912	-0.076904	0.123174	1.000000	0.013294	-0.151040	-0.009940	0.06702
fbs	0.121308	0.045032	0.094444	0.177531	0.013294	1.000000	-0.084189	-0.008567	0.02566
restecg	-0.116211	-0.058196	0.044421	-0.114103	-0.151040	-0.084189	1.000000	0.044123	-0.07073
thalach	-0.398522	-0.044020	0.295762	-0.046698	-0.009940	-0.008567	0.044123	1.000000	-0.37881
exang	0.096801	0.141664	-0.394280	0.067616	0.067023	0.025665	-0.070733	-0.378812	1.00000
oldpeak	0.210013	0.096093	-0.149230	0.193216	0.053952	0.005747	-0.058770	-0.344187	0.28822
slope	-0.168814	-0.030711	0.119717	-0.121475	-0.004038	-0.059894	0.093045	0.386784	-0.25774
ca	0.276326	0.118261	-0.181053	0.101389	0.070511	0.137979	-0.072042	-0.213177	0.11573
thal	0.068001	0.210041	-0.161736	0.062210	0.098803	-0.032019	-0.011981	-0.096439	0.20675
target	-0.225439	-0.280937	0.433798	-0.144931	-0.085239	-0.028046	0.137230	0.421741	-0.43675

Important insights we got from correlation.

There are moderate positive correlations between the target variable (target) and features such as cp (chest pain type), thalach (maximum heart rate achieved), and slope (slope of the peak exercise ST segment).

Moderate negative correlations exist between the target variable and features like exang (exercise-induced angina) and oldpeak (ST depression induced by exercise relative to rest).

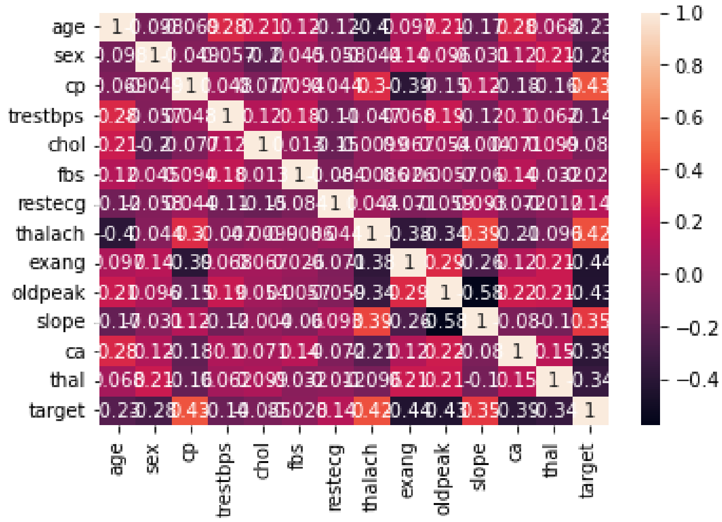
Some features exhibit correlations among themselves, such as age with trestbps (resting blood pressure) and chol (serum cholesterol levels).

Features with weak correlations with the target variable may be considered less influential in predicting the target variable.

In [15]:

```
#heatmap
sns.heatmap(data.corr(), annot=True)
```

Out[15]: <AxesSubplot:>

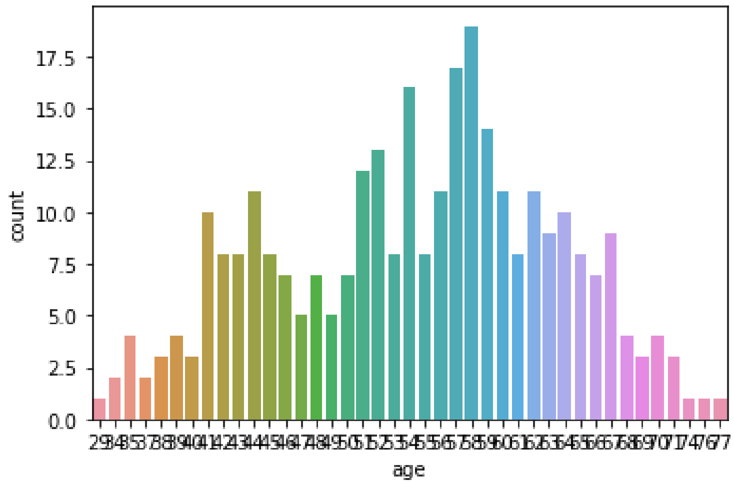


In [16]:
```
#frequency count
data['age'].value_counts()
```

Out[16]:
```
58    19
57    17
54    16
59    14
52    13
51    12
62    11
60    11
44    11
56    11
64    10
41    10
63     9
67     9
65     8
43     8
45     8
55     8
42     8
61     8
53     8
46     7
48     7
66     7
50     7
49     5
47     5
70     4
39     4
35     4
68     4
38     3
71     3
40     3
69     3
34     2
37     2
29     1
74     1
76     1
77     1
Name: age, dtype: int64
```

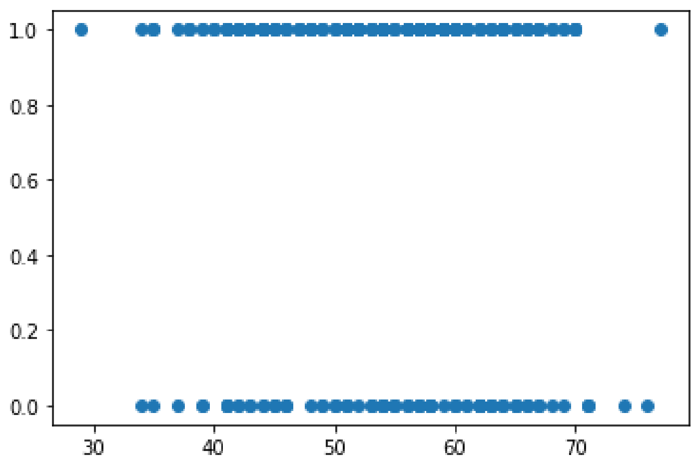
In [17]:
```
#barplot
sns.countplot(x='age', data=data)
```

Out[17]: <AxesSubplot:xlabel='age', ylabel='count'>



In [18]:
```
#scatterplot
plt.scatter(x='age', y='sex', data=data)
```

Out[18]: <matplotlib.collections.PathCollection at 0x1ac7a95ce80>



In [19]:

```
#pairplot
sns.pairplot(data)
```

Out[19]: <seaborn.axisgrid.PairGrid at 0x1ac7a990d30>

Pandas Profiling for Automated EDA

In [20]:
```
pip install pandas-profiling
```

```
Requirement already satisfied: pandas-profiling in c:\users\lenova\anaconda3\lib\sit
e-packages (3.6.6)
Requirement already satisfied: ydata-profiling in c:\users\lenova\anaconda3\lib\site
-packages (from pandas-profiling) (4.1.2)
Requirement already satisfied: scipy<1.10,>=1.4.1 in c:\users\lenova\anaconda3\lib\s
ite-packages (from ydata-profiling->pandas-profiling) (1.7.1)
Requirement already satisfied: pandas!=1.4.0,<1.6,>1.1 in c:\users\lenova\anaconda3
\lib\site-packages (from ydata-profiling->pandas-profiling) (1.3.4)
Requirement already satisfied: matplotlib<3.7,>=3.2 in c:\users\lenova\anaconda3\lib
\site-packages (from ydata-profiling->pandas-profiling) (3.4.3)
Requirement already satisfied: pydantic<1.11,>=1.8.1 in c:\users\lenova\anaconda3\li
b\site-packages (from ydata-profiling->pandas-profiling) (1.10.7)
Requirement already satisfied: PyYAML<6.1,>=5.0.0 in c:\users\lenova\anaconda3\lib\s
ite-packages (from ydata-profiling->pandas-profiling) (6.0)
Requirement already satisfied: jinja2<3.2,>=2.11.1 in c:\users\lenova\anaconda3\lib
\site-packages (from ydata-profiling->pandas-profiling) (2.11.3)Note: you may need t
o restart the kernel to use updated packages.

WARNING: visions 0.7.5 does not provide the extra 'type-image-path'
DEPRECATION: pyodbc 4.0.0-unsupported has a non-standard version number. pip 24.0 wi
ll enforce this behaviour change. A possible replacement is to upgrade to a newer ve
rsion of pyodbc or contact the author to suggest that they release a version with a
conforming version number. Discussion can be found at https://github.com/pypa/pip/is
sues/12063 (https://github.com/pypa/pip/issues/12063)

[notice] A new release of pip is available: 23.3.2 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
Requirement already satisfied: visions==0.7.5 in c:\users\lenova\anaconda3\lib\site-
packages (from visions[type_image_path]==0.7.5->ydata-profiling->pandas-profiling)
(0.7.5)
Requirement already satisfied: numpy<1.24,>=1.16.0 in c:\users\lenova\anaconda3\lib
\site-packages (from ydata-profiling->pandas-profiling) (1.22.4)
Requirement already satisfied: htmlmin==0.1.12 in c:\users\lenova\anaconda3\lib\site
-packages (from ydata-profiling->pandas-profiling) (0.1.12)
Requirement already satisfied: phik<0.13,>=0.11.1 in c:\users\lenova\anaconda3\lib\s
ite-packages (from ydata-profiling->pandas-profiling) (0.12.3)
Requirement already satisfied: requests<2.29,>=2.24.0 in c:\users\lenova\anaconda3\l
ib\site-packages (from ydata-profiling->pandas-profiling) (2.28.2)
Requirement already satisfied: tqdm<4.65,>=4.48.2 in c:\users\lenova\anaconda3\lib\s
ite-packages (from ydata-profiling->pandas-profiling) (4.64.1)
Requirement already satisfied: seaborn<0.13,>=0.10.1 in c:\users\lenova\anaconda3\li
b\site-packages (from ydata-profiling->pandas-profiling) (0.11.2)
Requirement already satisfied: multimethod<1.10,>=1.4 in c:\users\lenova\anaconda3\l
ib\site-packages (from ydata-profiling->pandas-profiling) (1.9.1)
Requirement already satisfied: statsmodels<0.14,>=0.13.2 in c:\users\lenova\anaconda
3\lib\site-packages (from ydata-profiling->pandas-profiling) (0.13.5)
Requirement already satisfied: typeguard<2.14,>=2.13.2 in c:\users\lenova\anaconda3
\lib\site-packages (from ydata-profiling->pandas-profiling) (2.13.3)
Requirement already satisfied: imagehash==4.3.1 in c:\users\lenova\anaconda3\lib\sit
e-packages (from ydata-profiling->pandas-profiling) (4.3.1)
Requirement already satisfied: PyWavelets in c:\users\lenova\anaconda3\lib\site-pack
ages (from imagehash==4.3.1->ydata-profiling->pandas-profiling) (1.1.1)
Requirement already satisfied: pillow in c:\users\lenova\anaconda3\lib\site-packages
(from imagehash==4.3.1->ydata-profiling->pandas-profiling) (8.4.0)
Requirement already satisfied: attrs>=19.3.0 in c:\users\lenova\anaconda3\lib\site-p
ackages (from visions==0.7.5->visions[type_image_path]==0.7.5->ydata-profiling->pand
as-profiling) (21.2.0)
Requirement already satisfied: networkx>=2.4 in c:\users\lenova\anaconda3\lib\site-p
ackages (from visions==0.7.5->visions[type_image_path]==0.7.5->ydata-profiling->pand
as-profiling) (2.6.3)
Requirement already satisfied: tangled-up-in-unicode>=0.0.4 in c:\users\lenova\anaco
nda3\lib\site-packages (from visions==0.7.5->visions[type_image_path]==0.7.5->ydata-
profiling->pandas-profiling) (0.2.0)
Requirement already satisfied: MarkupSafe>=0.23 in c:\users\lenova\anaconda3\lib\sit
e-packages (from jinja2<3.2,>=2.11.1->ydata-profiling->pandas-profiling) (1.1.1)
Requirement already satisfied: cycler>=0.10 in c:\users\lenova\anaconda3\lib\site-pa
ckages (from matplotlib<3.7,>=3.2->ydata-profiling->pandas-profiling) (0.10.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\lenova\anaconda3\lib\si
te-packages (from matplotlib<3.7,>=3.2->ydata-profiling->pandas-profiling) (1.3.1)
Requirement already satisfied: pyparsing>=2.2.1 in c:\users\lenova\anaconda3\lib\sit
e-packages (from matplotlib<3.7,>=3.2->ydata-profiling->pandas-profiling) (3.0.4)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\lenova\anaconda3\lib
\site-packages (from matplotlib<3.7,>=3.2->ydata-profiling->pandas-profiling) (2.8.
2)
Requirement already satisfied: pytz>=2017.3 in c:\users\lenova\anaconda3\lib\site-pa
ckages (from pandas!=1.4.0,<1.6,>1.1->ydata-profiling->pandas-profiling) (2021.3)
Requirement already satisfied: joblib>=0.14.1 in c:\users\lenova\anaconda3\lib\site-
packages (from phik<0.13,>=0.11.1->ydata-profiling->pandas-profiling) (1.3.2)
Requirement already satisfied: typing-extensions>=4.2.0 in c:\users\lenova\anaconda3
\lib\site-packages (from pydantic<1.11,>=1.8.1->ydata-profiling->pandas-profiling)
(4.9.0)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\lenova\anaconda3
\lib\site-packages (from requests<2.29,>=2.24.0->ydata-profiling->pandas-profiling)
(2.0.4)
Requirement already satisfied: idna<4,>=2.5 in c:\users\lenova\anaconda3\lib\site-pa
ckages (from requests<2.29,>=2.24.0->ydata-profiling->pandas-profiling) (3.2)
```

```
Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\users\lenova\anaconda3\li
b\site-packages (from requests<2.29,>=2.24.0->ydata-profiling->pandas-profiling) (1.
26.7)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\lenova\anaconda3\lib\s
ite-packages (from requests<2.29,>=2.24.0->ydata-profiling->pandas-profiling) (2021.
10.8)
Requirement already satisfied: patsy>=0.5.2 in c:\users\lenova\anaconda3\lib\site-pa
ckages (from statsmodels<0.14,>=0.13.2->ydata-profiling->pandas-profiling) (0.5.2)
Requirement already satisfied: packaging>=21.3 in c:\users\lenova\anaconda3\lib\site
-packages (from statsmodels<0.14,>=0.13.2->ydata-profiling->pandas-profiling) (23.1)
Requirement already satisfied: colorama in c:\users\lenova\anaconda3\lib\site-packag
es (from tqdm<4.65,>=4.48.2->ydata-profiling->pandas-profiling) (0.4.4)
Requirement already satisfied: six in c:\users\lenova\anaconda3\lib\site-packages (f
rom cycler>=0.10->matplotlib<3.7,>=3.2->ydata-profiling->pandas-profiling) (1.15.0)
```

In [21]:

```python
import pandas_profiling as pp
```

```
C:\Users\lenova\AppData\Local\Temp/ipykernel_8964/1872674328.py:1: DeprecationWarnin
g: `import pandas_profiling` is going to be deprecated by April 1st. Please use `imp
ort ydata_profiling` instead.
  import pandas_profiling as pp
```

In [22]:
```
pp.ProfileReport(data)
```

Summarize dataset: 100% 48/48 [00:10<00:00, 3.55it/s, Completed]

Generate report structure: 100% 1/1 [00:08<00:00, 8.20s/it]

Render HTML: 100% 1/1 [00:02<00:00, 2.45s/it]

Overview

Dataset statistics

Number of variables	14
Number of observations	303
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	1
Duplicate rows (%)	0.3%
Total size in memory	33.3 KiB
Average record size in memory	112.4 B

Variable types

Numeric	5
Categorical	9

Alerts

Dataset has 1 (0.3%) duplicate rows	**Duplicates**
`cp` is highly overall correlated with `target`	**High correlation**
`thal` is highly overall correlated with `target`	**High correlation**

Out[22]:

Preparing the data

In [23]:
```
x = data.drop('target',axis=1)
y = data["target"]
```

Splitting the data.

In [24]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20, random_
```

Feature scaling

In [25]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

Fitting and transforming the data.

In [26]:
```
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

Applying different ML algorithms to find best algorithm with higher prediction

Logist regression.

In [27]:
```
from sklearn.linear_model import LogisticRegression
LR = LogisticRegression()
```

In [28]:
```
#fiting the model
model = LR.fit(x_train, y_train)
```

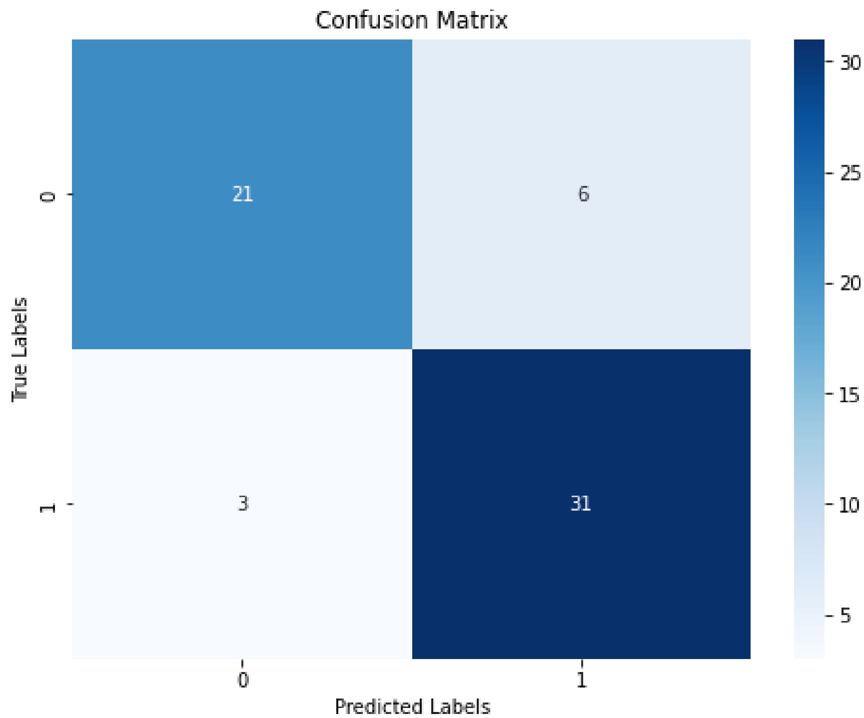
In [29]:
```
#predictions
LR_predict = LR.predict(x_test)
```

In [30]:
```
from sklearn.metrics import confusion_matrix,accuracy_score,roc_curve,classificat
```

In [31]:
```
#confusion matrix
LR_conf_matrix = confusion_matrix(y_test, LR_predict)
print("confussion matrix")
print(LR_conf_matrix)
```

```
confussion matrix
[[21  6]
 [ 3 31]]
```

In [32]:
```
plt.figure(figsize=(8, 6))
sns.heatmap(LR_conf_matrix, annot=True, cmap='Blues', fmt='g')
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```



In [33]:
```
LR_acc_score = accuracy_score(y_test, LR_predict)
print("Accuracy of Logistic Regression:",LR_acc_score*100,'\n')
```

```
Accuracy of Logistic Regression: 85.24590163934425
```

In [34]:
```
print(classification_report(y_test,LR_predict))
```

```
              precision    recall  f1-score   support

           0       0.88      0.78      0.82        27
           1       0.84      0.91      0.87        34

    accuracy                           0.85        61
   macro avg       0.86      0.84      0.85        61
weighted avg       0.85      0.85      0.85        61
```

Gaussian Naive Bayes

In [35]:
```
from sklearn.naive_bayes import GaussianNB
NB = GaussianNB()
```

In [36]:
```
#fitting the model
NB.fit(x_train,y_train)
```

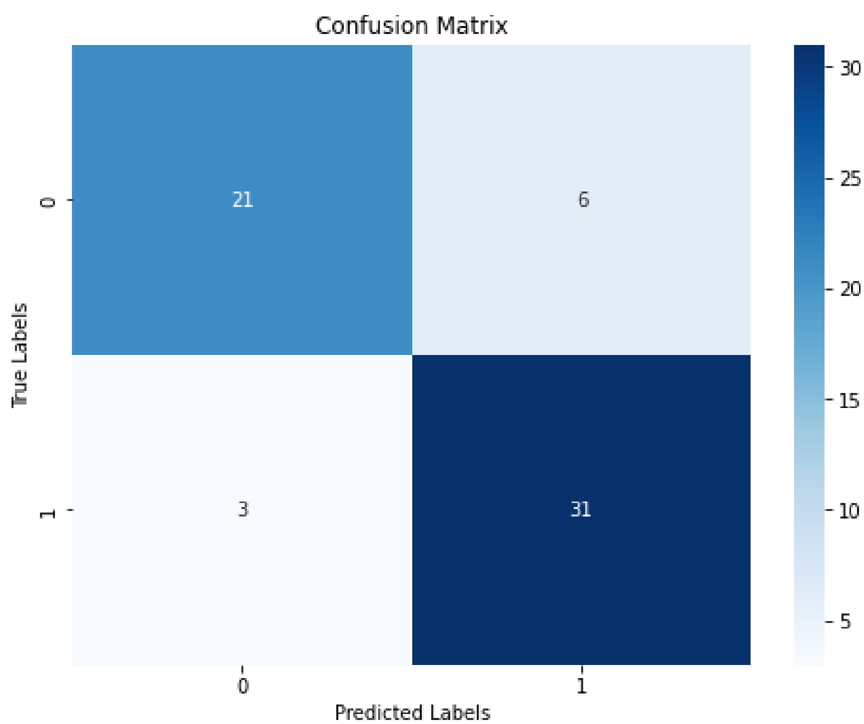
Out[36]:
```
▾ GaussianNB ⓘ ?
GaussianNB()
```
(https://scikit-learn.org/1.4/modules/generated/sklearn.naive_bayes.GaussianNB.html)

In [37]:
```
#predictions
NBpred = NB.predict(x_test)
```

In [38]:
```
#confusion matrix
NB_conf_matrix = confusion_matrix(y_test, NBpred)
print("confussion matrix")
print(NB_conf_matrix)
```

```
confussion matrix
[[21  6]
 [ 3 31]]
```

In [39]:
```
plt.figure(figsize=(8, 6))
sns.heatmap(NB_conf_matrix, annot=True, cmap='Blues', fmt='g')
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```



In [40]:
```
#accuracy score
NB_acc_score = accuracy_score(y_test, NBpred)
print("Accuracy of Naive Bayes model:",NB_acc_score*100,'\n')
```

```
Accuracy of Naive Bayes model: 85.24590163934425
```

In [41]:
```
print(classification_report(y_test,NBpred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.78      0.82        27
           1       0.84      0.91      0.87        34

    accuracy                           0.85        61
   macro avg       0.86      0.84      0.85        61
weighted avg       0.85      0.85      0.85        61
```

Extreme Gradient Boost

In [42]:
```
from xgboost import XGBClassifier
xgb = XGBClassifier(learning_rate=0.01, n_estimators=25, max_depth=15,gamma=0.6, 
                    reg_lambda=2, booster='dart', colsample_bylevel=0.6, colsampl
```

In [43]:
```
#fitting the data
xgb.fit(x_train, y_train)
```

Out[43]:

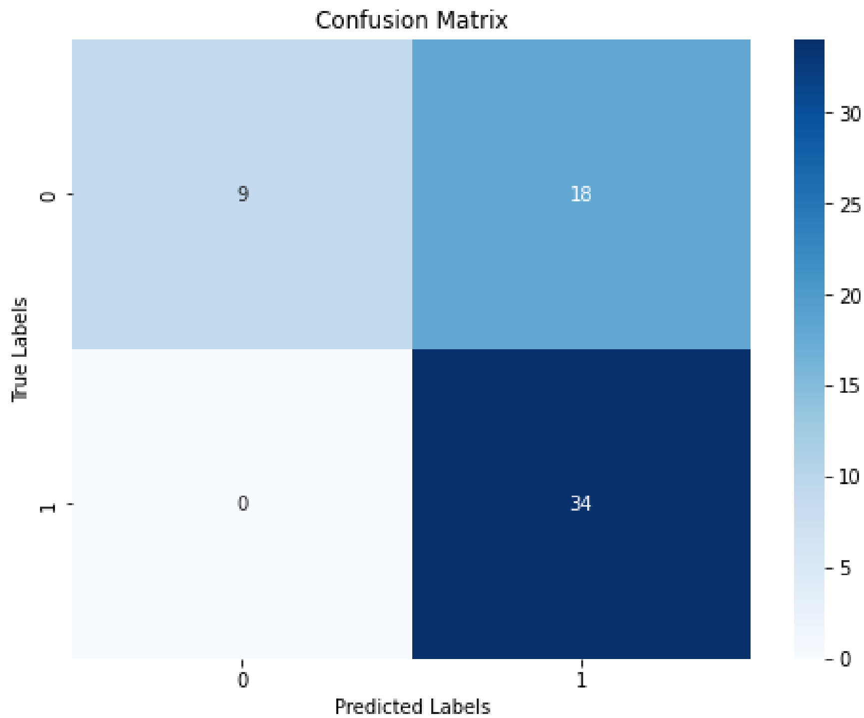
```
XGBClassifier

XGBClassifier(base_score=None, booster='dart', callbacks=None,
              colsample_bylevel=0.6, colsample_bynode=0.5, colsample_bytree=0.6,
              device=None, early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, feature_types=None, gamma=0.6, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=0.01, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=15,
              max_leaves=None, min_child_weight=None, missing=nan,
              monotone_constraints=None, multi_strategy=None, n_estimators=25,
              n_jobs=None, num_parallel_tree=None, random_state=None, ...)
```

In [44]:
```
#prediction
xgb_predicted = xgb.predict(x_test)
```

In [45]:
```
#confusion matrix
xgb_conf_matrix = confusion_matrix(y_test, xgb_predicted)
print("confussion matrix")
print(xgb_conf_matrix)
```

```
confussion matrix
[[ 9 18]
 [ 0 34]]
```

In [46]:
```
plt.figure(figsize=(8, 6))
sns.heatmap(xgb_conf_matrix, annot=True, cmap='Blues', fmt='g')
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```



In [47]:
```
#accuracy score
xgb_acc_score = accuracy_score(y_test, xgb_predicted)
print("Accuracy of Extreme Gradient Boost:",xgb_acc_score*100,'\n')
```

```
Accuracy of Extreme Gradient Boost: 70.49180327868852
```

In [48]:
```
print(classification_report(y_test,xgb_predicted))
```

```
              precision    recall  f1-score   support

           0       1.00      0.33      0.50        27
           1       0.65      1.00      0.79        34

    accuracy                           0.70        61
   macro avg       0.83      0.67      0.65        61
weighted avg       0.81      0.70      0.66        61
```

Random Forest

In [49]:
```
from sklearn.ensemble import RandomForestClassifier
RF = RandomForestClassifier(n_estimators=20, random_state=12,max_depth=5)
```

In [50]:
```
#fittig the data
RF.fit(x_train,y_train)
```

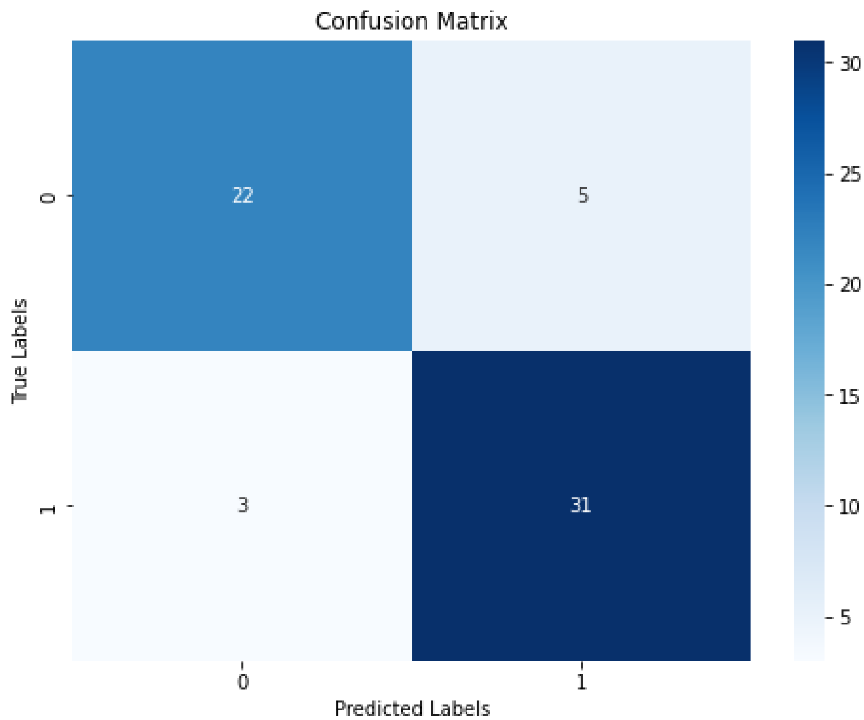
Out[50]:
```
RandomForestClassifier
RandomForestClassifier(max_depth=5, n_estimators=20, random_state=12)
```
(https://scikit-learn.org/1.4/modules/

In [51]:
```
#prediction
RF_predicted = RF.predict(x_test)
```

In [52]:
```
#confusion matrix
RF_conf_matrix = confusion_matrix(y_test, RF_predicted)
print("confussion matrix")
print(RF_conf_matrix)
```

```
confussion matrix
[[22  5]
 [ 3 31]]
```

In [53]:
```
plt.figure(figsize=(8, 6))
sns.heatmap(RF_conf_matrix, annot=True, cmap='Blues', fmt='g')
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```



In [54]:
```
#accuracy score
RF_acc_score = accuracy_score(y_test, RF_predicted)
print("Accuracy of Random Forest:",RF_acc_score*100,'\n')
```

```
Accuracy of Random Forest: 86.88524590163934
```

In [55]:
```
#classification report
print(classification_report(y_test,RF_predicted))
```

```
              precision    recall  f1-score   support

           0       0.88      0.81      0.85        27
           1       0.86      0.91      0.89        34

    accuracy                           0.87        61
   macro avg       0.87      0.86      0.87        61
weighted avg       0.87      0.87      0.87        61
```

Decision Tree

In [56]:
```
from sklearn.tree import DecisionTreeClassifier
DT = DecisionTreeClassifier(criterion = 'entropy',random_state=0,max_depth = 6)
```

In [57]:
```
#fitting the data
DT.fit(x_train, y_train)
```

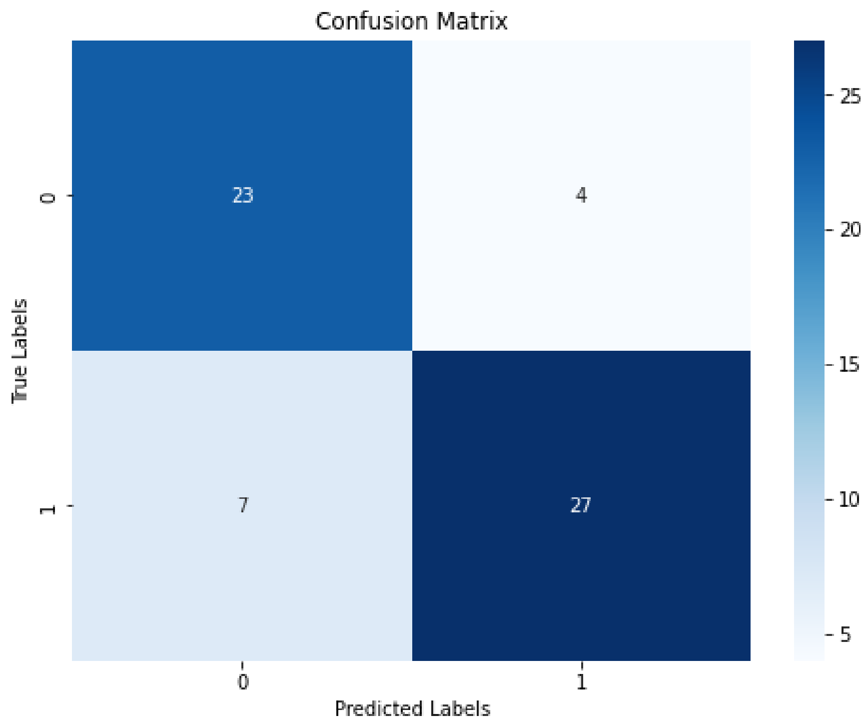
Out[57]:
```
DecisionTreeClassifier

DecisionTreeClassifier(criterion='entropy', max_depth=6, random_state=0)
```
(https://scikit-learn.org/1.4/modu

In [58]:
```
#prediction
DT_predicted = DT.predict(x_test)
```

In [59]:
```
#confusion matrix
DT_conf_matrix = confusion_matrix(y_test, DT_predicted)
print("confussion matrix")
print(DT_conf_matrix)
```

```
confussion matrix
[[23  4]
 [ 7 27]]
```

In [60]:
```
plt.figure(figsize=(8, 6))
sns.heatmap(DT_conf_matrix, annot=True, cmap='Blues', fmt='g')
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```



In [61]:
```
#accuracy score
DT_acc_score = accuracy_score(y_test, DT_predicted)
print("Accuracy of DecisionTreeClassifier:",DT_acc_score*100,'\n')
```

```
Accuracy of DecisionTreeClassifier: 81.9672131147541
```

In [62]:
```
print(classification_report(y_test,DT_predicted))
```

```
              precision    recall  f1-score   support

           0       0.77      0.85      0.81        27
           1       0.87      0.79      0.83        34

    accuracy                           0.82        61
   macro avg       0.82      0.82      0.82        61
weighted avg       0.82      0.82      0.82        61
```

Support Vector Machine

In [63]:
```
from sklearn.svm import SVC
svc =  SVC(kernel='rbf', C=2)
```

In [64]:
```
# fitting the data
svc.fit(x_train, y_train)
```

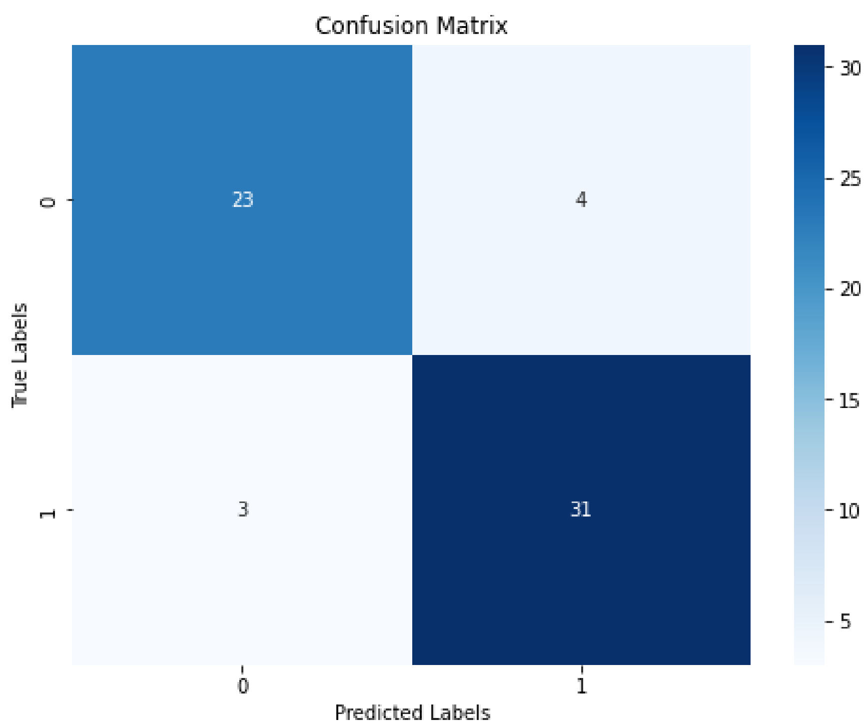
Out[64]:
```
▾ SVC ⓘ ?
(https://scikit-learn.org/1.4/modules/generated/sklearn.svm.SVC.html)
SVC(C=2)
```

In [65]:
```
#prediction
svc_predicted = svc.predict(x_test)
```

In [66]:
```
#confusion matrix
svc_conf_matrix = confusion_matrix(y_test, svc_predicted)
print("confussion matrix")
print(svc_conf_matrix)
```

```
confussion matrix
[[23  4]
 [ 3 31]]
```

In [67]:
```
plt.figure(figsize=(8, 6))
sns.heatmap(svc_conf_matrix, annot=True, cmap='Blues', fmt='g')
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```



In [68]:
```
#accuracy score
svc_acc_score = accuracy_score(y_test, svc_predicted)
print("Accuracy of Support Vector Classifier:",svc_acc_score*100,'\n')
```

```
Accuracy of Support Vector Classifier: 88.52459016393442
```

In [69]:
```
#classification report
print(classification_report(y_test,svc_predicted))
```

```
              precision    recall  f1-score   support

           0       0.88      0.85      0.87        27
           1       0.89      0.91      0.90        34

    accuracy                           0.89        61
   macro avg       0.89      0.88      0.88        61
weighted avg       0.89      0.89      0.88        61
```

K-NeighborsClassifier

In [70]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=10)
```

In [71]:
```
#fitting the data
knn.fit(x_train, y_train)
```

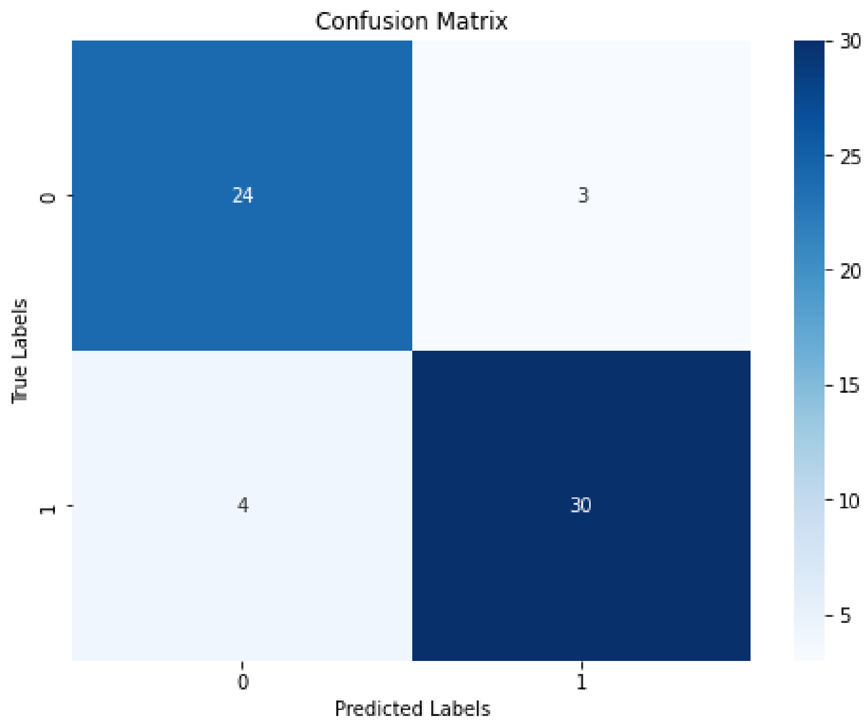
Out[71]:
```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=10)
```
(https://scikit-learn.org/1.4/modules/generated/sklearn.neighbors.KNeighborsC

In [72]:
```
#prediction
knn_predicted = knn.predict(x_test)
```

In [73]:
```
#confusion matrix
knn_conf_matrix = confusion_matrix(y_test, knn_predicted)
print("confussion matrix")
print(knn_conf_matrix)
```

```
confussion matrix
[[24  3]
 [ 4 30]]
```

In [74]:
```
plt.figure(figsize=(8, 6))
sns.heatmap(knn_conf_matrix, annot=True, cmap='Blues', fmt='g')
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```



In [75]:
```
#accuracy score
knn_acc_score = accuracy_score(y_test, knn_predicted)
print("Accuracy of K-NeighborsClassifier:",knn_acc_score*100,'\n')
```

```
Accuracy of K-NeighborsClassifier: 88.52459016393442
```

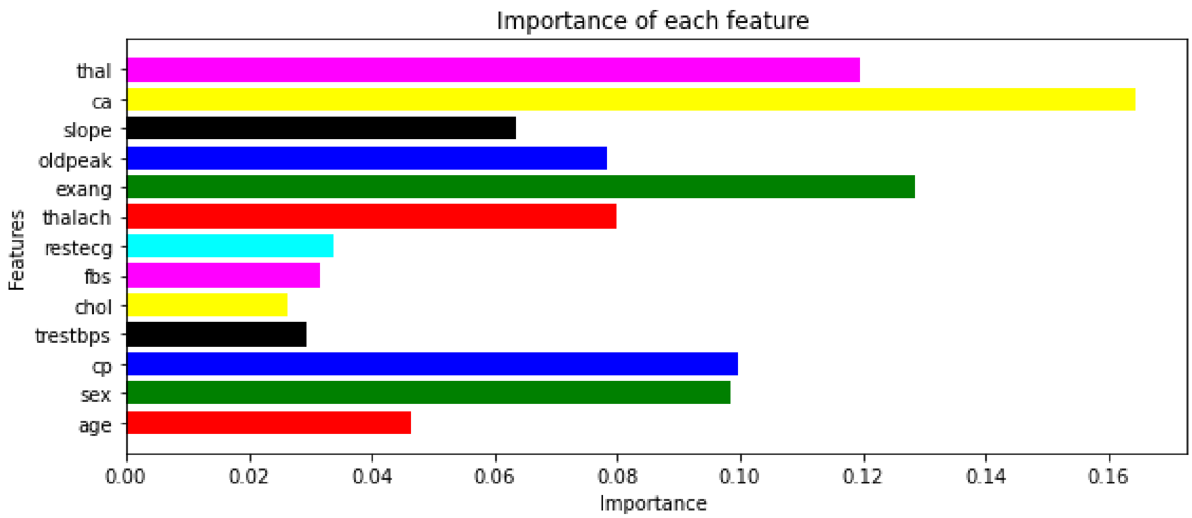
In [76]:
```
#classification report
print(classification_report(y_test,knn_predicted))
```

```
              precision    recall  f1-score   support

           0       0.86      0.89      0.87        27
           1       0.91      0.88      0.90        34

    accuracy                           0.89        61
   macro avg       0.88      0.89      0.88        61
weighted avg       0.89      0.89      0.89        61
```

Identifying the importance of each feature.

In [77]:

```
#using xgb.feature_importances_ feature
colors = ['red', 'green', 'blue', 'black', 'yellow', 'magenta', 'cyan']
important_features = pd.DataFrame({'Features': ['age', 'sex', 'cp', 'trestbps', '
        'exang', 'oldpeak', 'slope', 'ca', 'thal'], 'Importance': xgb.feature_impo
plt.figure(figsize=(10,4))
plt.title("Importance of each feature ")
plt.xlabel("Importance ")
plt.ylabel("Features")
plt.barh(important_features['Features'],important_features['Importance'],color = 
plt.show()
```



"ca" appears to be the most important feature, suggesting that it strongly influences the model's predictions.

Conversely, "chol" has the lowest importance, implying that it contributes less to the model's predictions compared to other features.

ROC Curve

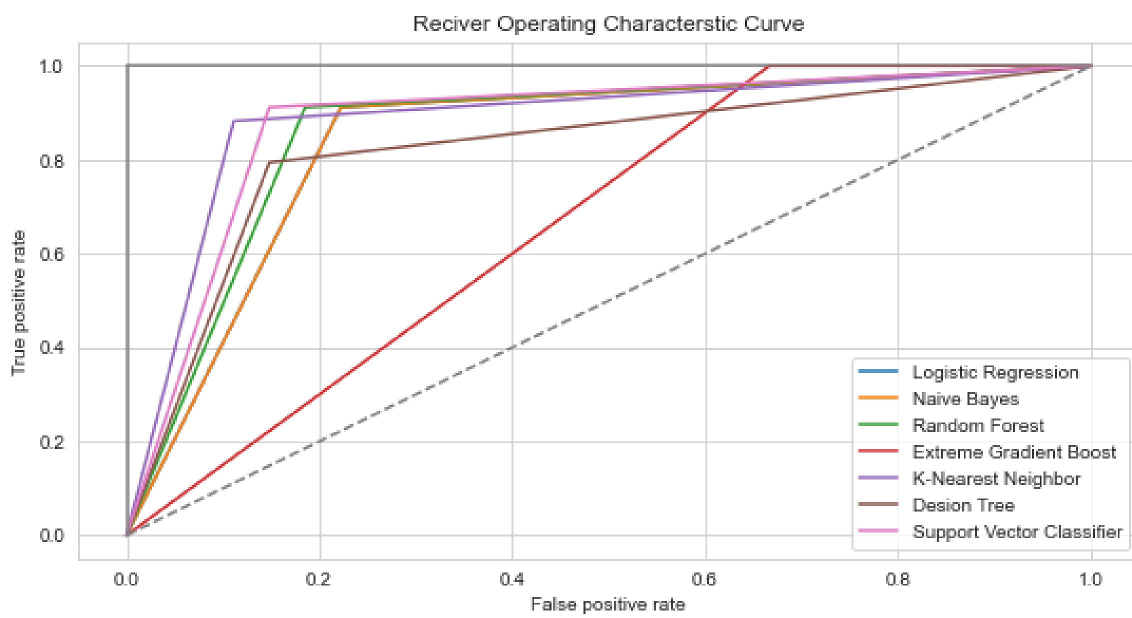
Finding the false positive rate, true positive rate, the threshold value

In [78]:

```
LR_false_positive_rate,LR_true_positive_rate,LR_threshold = roc_curve(y_test, LR_
NB_false_positive_rate,NB_true_positive_rate,NB_threshold = roc_curve(y_test,NBpr
RF_false_positive_rate,RF_true_positive_rate,RF_threshold = roc_curve(y_test,RF_p
xgb_false_positive_rate,xgb_true_positive_rate,xgb_threshold = roc_curve(y_test,x
knn_false_positive_rate,knn_true_positive_rate,knn_threshold = roc_curve(y_test,k
DT_false_positive_rate,DT_true_positive_rate,DT_threshold = roc_curve(y_test,DT_p
svc_false_positive_rate,svc_true_positive_rate,svc_threshold = roc_curve(y_test,s
```

In [79]:

```
sns.set_style('whitegrid')
plt.figure(figsize=(10,5))
plt.title('Reciver Operating Characterstic Curve')
plt.plot(LR_false_positive_rate,LR_true_positive_rate,label='Logistic Regression'
plt.plot(NB_false_positive_rate,NB_true_positive_rate,label='Naive Bayes')
plt.plot(RF_false_positive_rate,RF_true_positive_rate,label='Random Forest')
plt.plot(xgb_false_positive_rate,xgb_true_positive_rate,label='Extreme Gradient Bo
plt.plot(knn_false_positive_rate,knn_true_positive_rate,label='K-Nearest Neighbor
plt.plot(DT_false_positive_rate,DT_true_positive_rate,label='Desion Tree')
plt.plot(svc_false_positive_rate,svc_true_positive_rate,label='Support Vector Cla
plt.plot([0,1],ls='--')
plt.plot([0,0],[1,0],c='.5')
plt.plot([1,1],c='.5')
plt.ylabel('True positive rate')
plt.xlabel('False positive rate')
plt.legend()
plt.show()
```



The results suggest that KNN and SVC may be preferred choices for this classification task due to their superior performance compared to other classifiers.

In [80]:
```
model_evaluation = pd.DataFrame({'Model': ['Logistic Regression','Naive Bayes','R
                    'K-Nearest Neighbour','Decision Tree','Support Vector Machine
                    NB_acc_score*100,RF_acc_score*100,xgb_acc_score*100,knn_acc_s
model_evaluation
```

Out[80]:

	Model	Accuracy
0	Logistic Regression	85.245902
1	Naive Bayes	85.245902
2	Random Forest	86.885246
3	Extreme Gradient Boost	70.491803
4	K-Nearest Neighbour	88.524590
5	Decision Tree	81.967213
6	Support Vector Machine	88.524590

In [81]:
```
model_evaluation_sorted = model_evaluation.sort_values(by='Accuracy', ascending=F
model_evaluation_sorted
```

Out[81]:

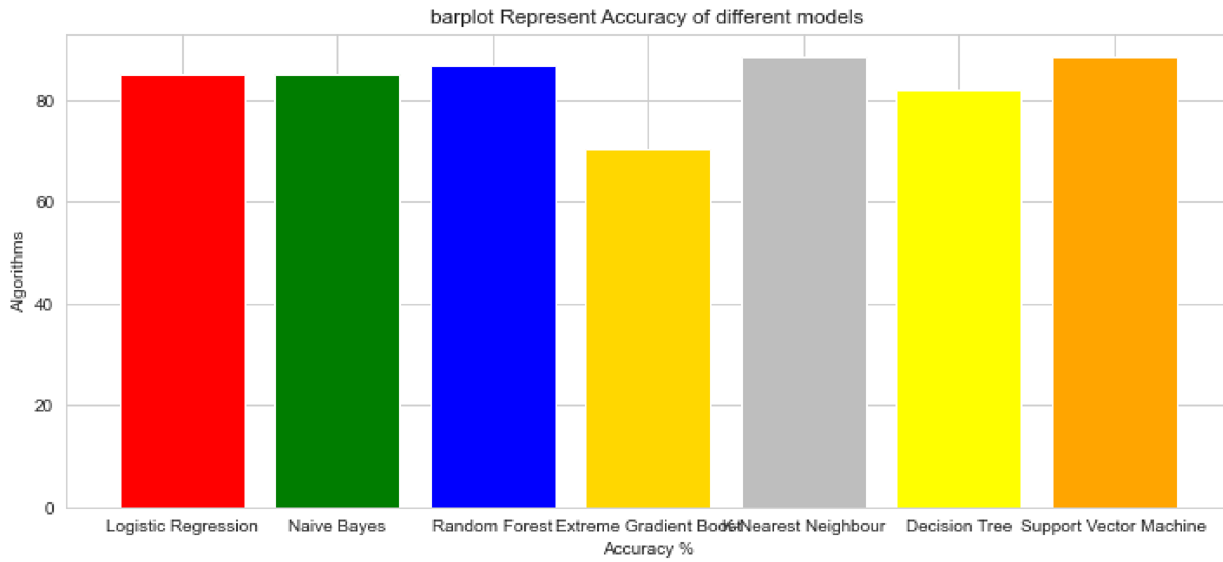
	Model	Accuracy
4	K-Nearest Neighbour	88.524590
6	Support Vector Machine	88.524590
2	Random Forest	86.885246
0	Logistic Regression	85.245902
1	Naive Bayes	85.245902
5	Decision Tree	81.967213
3	Extreme Gradient Boost	70.491803

KNN and SVM stand out as the top-performing models with the highest accuracy, while Extreme Gradient Boost lags behind with the lowest accuracy.

This analysis provides insights into the comparative performance of different models, guiding the selection of the most suitable model for the classification task at hand.

Graphically representing the performance of different models.

In [82]:
```
colors = ['red','green','blue','gold','silver','yellow','orange',]
plt.figure(figsize=(12,5))
plt.title("barplot Represent Accuracy of different models")
plt.xlabel("Accuracy %")
plt.ylabel("Algorithms")
plt.bar(model_evaluation['Model'],model_evaluation['Accuracy'],color = colors)
plt.show()
```



Using ensemble learning method in order to try to enhance the performance and accuracy of the model

In [83]:
```
pip install mlxtend
```

```
Requirement already satisfied: mlxtend in c:\users\lenova\anaconda3\lib\site-package
s (0.23.1)
Requirement already satisfied: scipy>=1.2.1 in c:\users\lenova\anaconda3\lib\site-pa
ckages (from mlxtend) (1.7.1)
Requirement already satisfied: numpy>=1.16.2 in c:\users\lenova\anaconda3\lib\site-p
ackages (from mlxtend) (1.22.4)
Requirement already satisfied: pandas>=0.24.2 in c:\users\lenova\anaconda3\lib\site-
packages (from mlxtend) (1.3.4)
Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: scikit-learn>=1.0.2 in c:\users\lenova\anaconda3\lib
\site-packages (from mlxtend) (1.4.1.post1)

DEPRECATION: pyodbc 4.0.0-unsupported has a non-standard version number. pip 24.0 wi
ll enforce this behaviour change. A possible replacement is to upgrade to a newer ve
rsion of pyodbc or contact the author to suggest that they release a version with a
conforming version number. Discussion can be found at https://github.com/pypa/pip/is
sues/12063 (https://github.com/pypa/pip/issues/12063)

[notice] A new release of pip is available: 23.3.2 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip


Requirement already satisfied: matplotlib>=3.0.0 in c:\users\lenova\anaconda3\lib\si
te-packages (from mlxtend) (3.4.3)
Requirement already satisfied: joblib>=0.13.2 in c:\users\lenova\anaconda3\lib\site-
packages (from mlxtend) (1.3.2)
Requirement already satisfied: cycler>=0.10 in c:\users\lenova\anaconda3\lib\site-pa
ckages (from matplotlib>=3.0.0->mlxtend) (0.10.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\lenova\anaconda3\lib\si
te-packages (from matplotlib>=3.0.0->mlxtend) (1.3.1)
Requirement already satisfied: pillow>=6.2.0 in c:\users\lenova\anaconda3\lib\site-p
ackages (from matplotlib>=3.0.0->mlxtend) (8.4.0)
Requirement already satisfied: pyparsing>=2.2.1 in c:\users\lenova\anaconda3\lib\sit
e-packages (from matplotlib>=3.0.0->mlxtend) (3.0.4)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\lenova\anaconda3\lib
\site-packages (from matplotlib>=3.0.0->mlxtend) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in c:\users\lenova\anaconda3\lib\site-pa
ckages (from pandas>=0.24.2->mlxtend) (2021.3)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\lenova\anaconda3\lib
\site-packages (from scikit-learn>=1.0.2->mlxtend) (2.2.0)
Requirement already satisfied: six in c:\users\lenova\anaconda3\lib\site-packages (f
rom cycler>=0.10->matplotlib>=3.0.0->mlxtend) (1.15.0)
```

Using the stacking technique.

In [84]:
```
from mlxtend.classifier import StackingCVClassifier
SCV=StackingCVClassifier(classifiers=[xgb,knn,svc],meta_classifier= svc,random_st
```

In [85]:
```
#fitting the data
SCV.fit(x_train,y_train)
```

Out[85]:
```
StackingCVClassifier
meta_classifier: SVC
SVC
(https://scikit-learn.org/1.4/modules/generated/sklearn.svm.SVC.html)
```

In [86]:
```
#pridiction
SCV_predicted = SCV.predict(x_test)
```

In [87]:
```
SCV_conf_matrix = confusion_matrix(y_test, SCV_predicted)
print("confussion matrix")
print(SCV_conf_matrix)
```

```
confussion matrix
[[24  3]
 [ 5 29]]
```

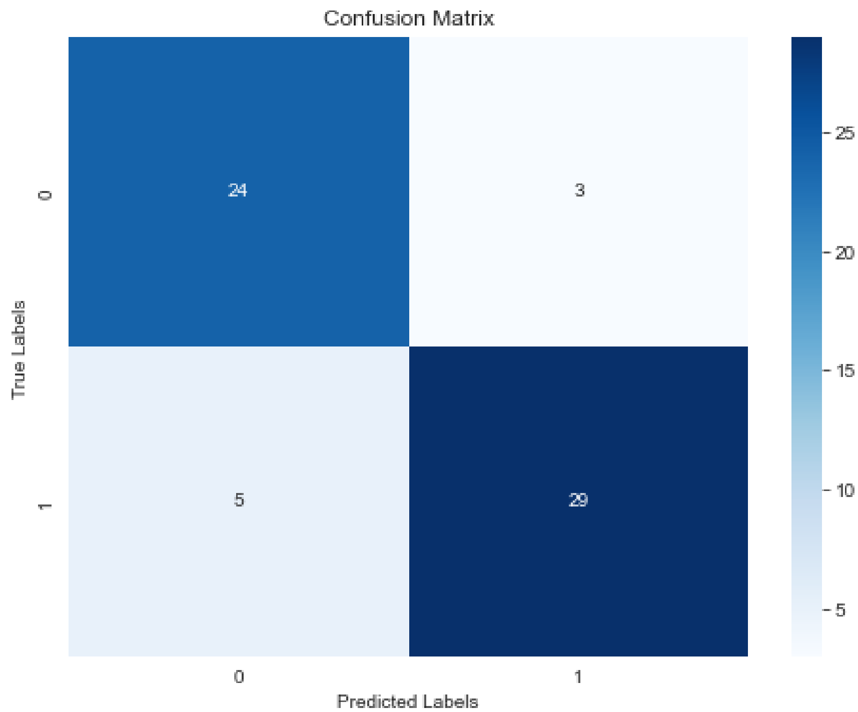
In [88]:
```
#accuracy score
SCV_acc_score = accuracy_score(y_test, SCV_predicted)
print("Accuracy of StackingCVClassifier:",SCV_acc_score*100,'\n')

```

```
Accuracy of StackingCVClassifier: 86.88524590163934
```

In [89]:
```
#classification report
print(classification_report(y_test,SCV_predicted))
```

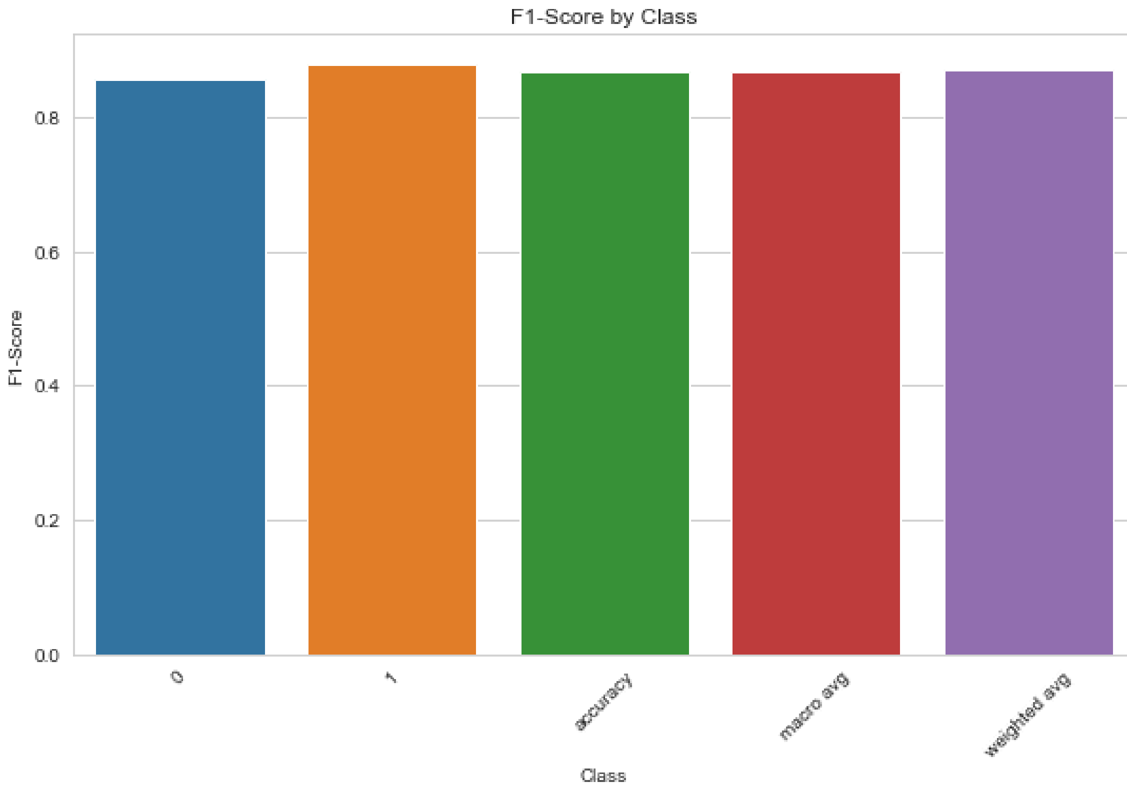
```
              precision    recall  f1-score   support

           0       0.83      0.89      0.86        27
           1       0.91      0.85      0.88        34

    accuracy                           0.87        61
   macro avg       0.87      0.87      0.87        61
weighted avg       0.87      0.87      0.87        61
```

In [90]:
```
plt.figure(figsize=(8, 6))
sns.heatmap(SCV_conf_matrix, annot=True, cmap='Blues', fmt='g')
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```



In [91]:
```
#clssification report
SCV_report = classification_report(y_test, SCV_predicted, output_dict=True)
df_report = pd.DataFrame(SCV_report).transpose()
```

In [92]:
```
plt.figure(figsize=(10, 6))
sns.barplot(x=df_report.index, y=df_report['f1-score'])
plt.title('F1-Score by Class')
plt.xlabel('Class')
plt.ylabel('F1-Score')
plt.xticks(rotation=45)
plt.show()
```

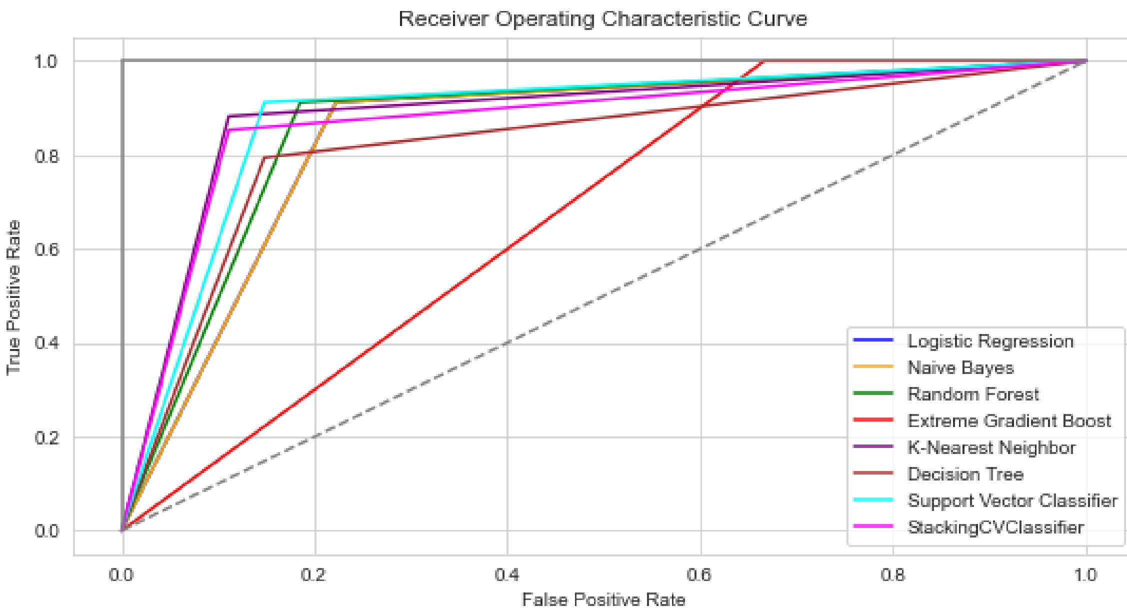


Representing the roc curve.

In [93]:
```
# Calculate ROC curves for all classifiers
classifiers = {
    'Logistic Regression': (LR_predict, 'blue'),
    'Naive Bayes': (NBpred, 'orange'),
    'Random Forest': (RF_predicted, 'green'),
    'Extreme Gradient Boost': (xgb_predicted, 'red'),
    'K-Nearest Neighbor': (knn_predicted, 'purple'),
    'Decision Tree': (DT_predicted, 'brown'),
    'Support Vector Classifier': (svc_predicted, 'cyan'),
    'StackingCVClassifier': (SCV_predicted, 'magenta')
}
```

In [94]:

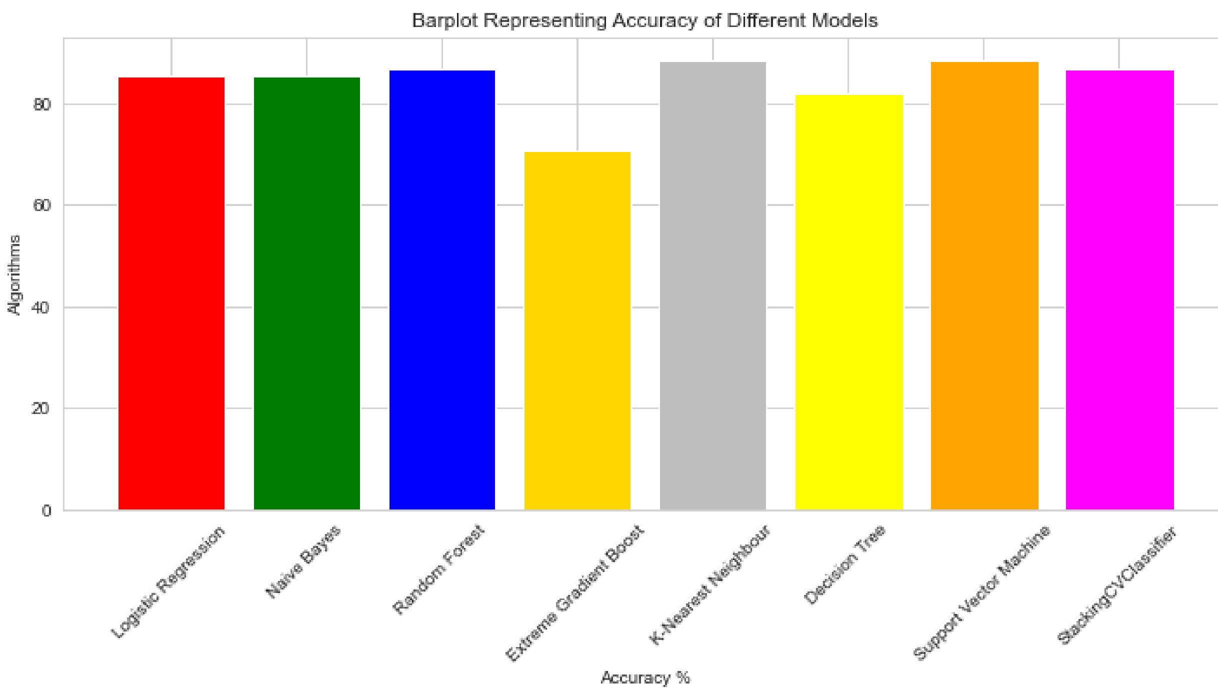
```
plt.figure(figsize=(10, 5))
plt.title('Receiver Operating Characteristic Curve')
for clf_name, (y_pred, color) in classifiers.items():
    false_positive_rate, true_positive_rate, _ = roc_curve(y_test, y_pred)
    plt.plot(false_positive_rate, true_positive_rate, label=clf_name, color=color

plt.plot([0, 1], ls='--', color='gray')
plt.plot([0, 0], [1, 0], c='.5')
plt.plot([1, 1], c='.5')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend()
plt.show()
```



visualizing the accuracy of different models.

In [96]:

```
# Define the colors for the bar plot
colors = ['red', 'green', 'blue', 'gold', 'silver', 'yellow', 'orange', 'magenta'

# Add the ensemble method result to the model_evaluation DataFrame
model_evaluation.loc[len(model_evaluation)] = ['StackingCVClassifier', SCV_acc_sc

# Plot the bar plot
plt.figure(figsize=(12, 5))
plt.title("Barplot Representing Accuracy of Different Models")
plt.xlabel("Accuracy %")
plt.ylabel("Algorithms")
plt.bar(model_evaluation['Model'], model_evaluation['Accuracy'], color=colors)
plt.xticks(rotation=45)
plt.show()
```



In [98]:

```
model_evaluation = pd.DataFrame({
    'Model': ['Logistic Regression', 'Naive Bayes', 'Random Forest', 'Extreme Gra
              'K-Nearest Neighbour', 'Decision Tree', 'Support Vector Machine', '
    'Accuracy': [LR_acc_score * 100, NB_acc_score * 100, RF_acc_score * 100, xgb_
                 knn_acc_score * 100, DT_acc_score * 100, svc_acc_score * 100, SC
})

# Display the model evaluation DataFrame
model_evaluation
```

Out[98]:

	Model	Accuracy
0	Logistic Regression	85.245902
1	Naive Bayes	85.245902
2	Random Forest	86.885246
3	Extreme Gradient Boost	70.491803
4	K-Nearest Neighbour	88.524590
5	Decision Tree	81.967213
6	Support Vector Machine	88.524590
7	StackingCVClassifier	86.885246

In [99]:

```
#soretd
model_evaluation_sorted = model_evaluation.sort_values(by='Accuracy', ascending=F
model_evaluation_sorted
```

Out[99]:

	Model	Accuracy
4	K-Nearest Neighbour	88.524590
6	Support Vector Machine	88.524590
2	Random Forest	86.885246
7	StackingCVClassifier	86.885246
0	Logistic Regression	85.245902
1	Naive Bayes	85.245902
5	Decision Tree	81.967213
3	Extreme Gradient Boost	70.491803

Based on the accuracy scores alone, K-Nearest Neighbour and Support Vector Machine appear to be the top-performing models

In [ ]:

```

```

In [ ]:

```

```