Heart Disease Prediction using machine learning

Table of Contents

Packages Required

In [1]:
```
#loading dataset
import pandas as pd
import numpy as np
#visualisation
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
#EDA
from collections import Counter
import ydata_profiling as pp
# data preprocessing
from sklearn.preprocessing import StandardScaler, LabelEncoder
# data splitting
from sklearn.model_selection import train_test_split
# data modeling
from sklearn.metrics import confusion_matrix,accuracy_score,roc_curve,classificatio
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.decomposition import PCA
from sklearn.svm import SVC
```

```
#ensembling
from mlxtend.classifier import StackingCVClassifier
```

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	ta
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	

In [3]:
```
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 Preprocessing

In [4]:
```
# Checking for missing values
print(data.isnull().sum())

# Dropping duplicate rows
data = data.drop_duplicates()
```

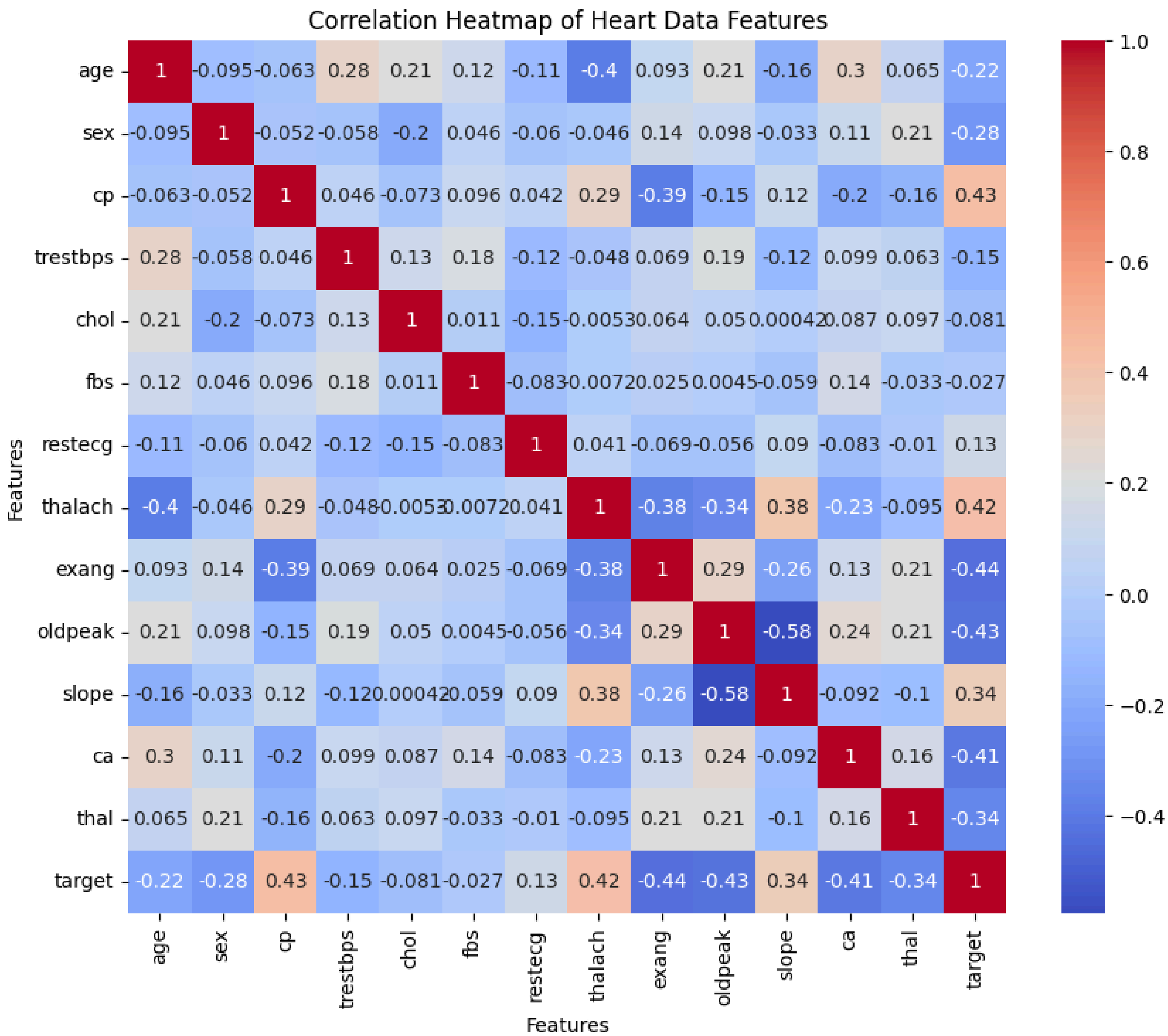
```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

Encoding Categorical Data

In [5]:
```
# Encoding categorical features
categorical_columns = ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca', 'thal
for col in categorical_columns:
    le = LabelEncoder()
    data[col] = le.fit_transform(data[col])
```

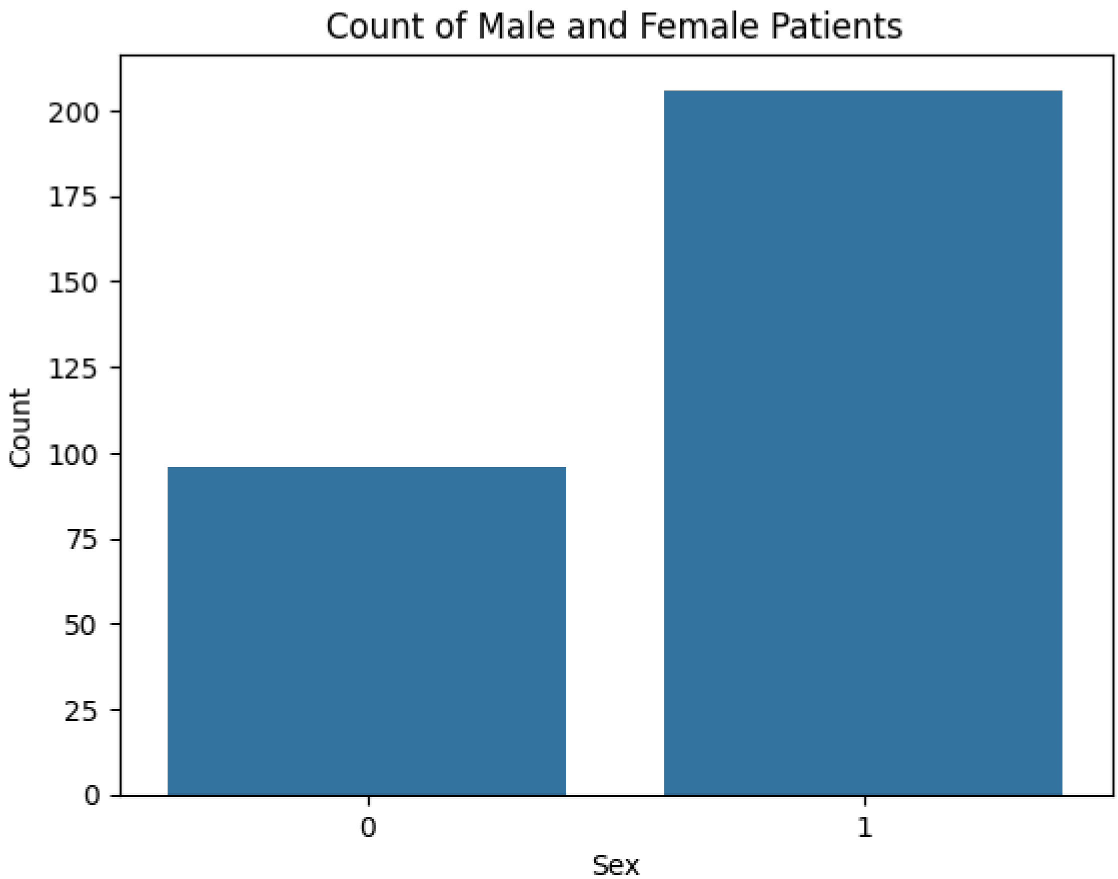
EDA

In [6]:
```
# Correlation heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(data.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap of Heart Data Features')
plt.xlabel('Features')
plt.ylabel('Features')
plt.show()
```



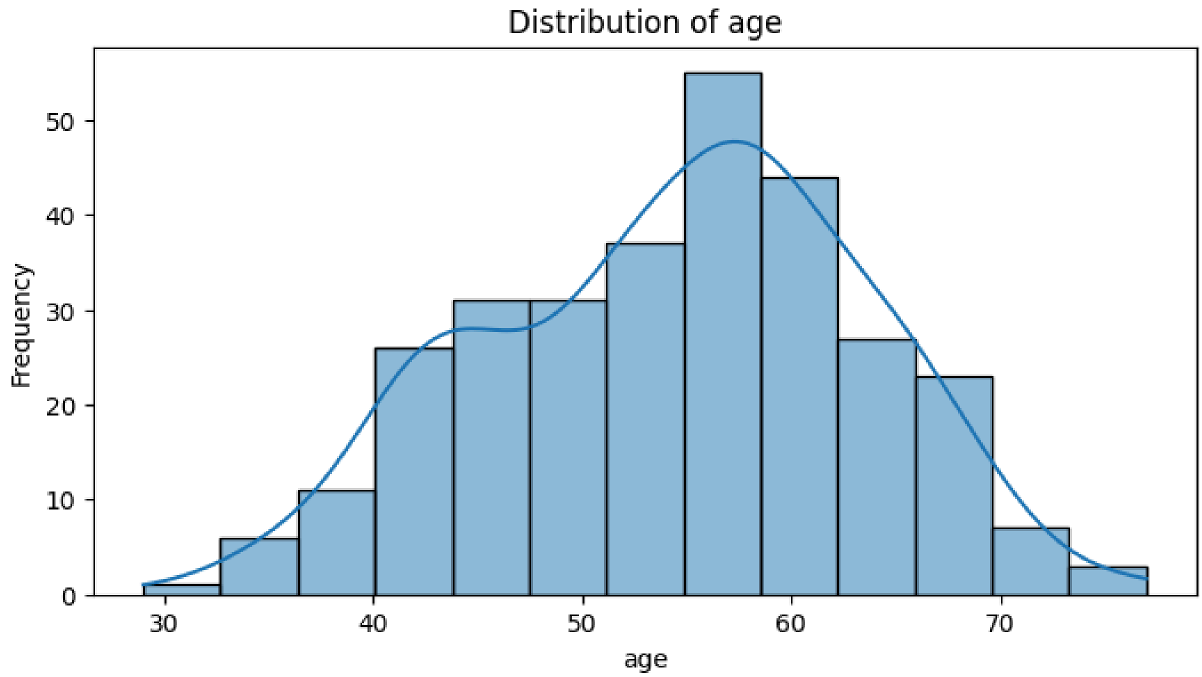
In [7]:

```
# Countplot for the 'sex' feature (without palette)
sns.countplot(x='sex', data=data)
plt.title('Count of Male and Female Patients')
plt.xlabel('Sex')
plt.ylabel('Count')
plt.show()
```

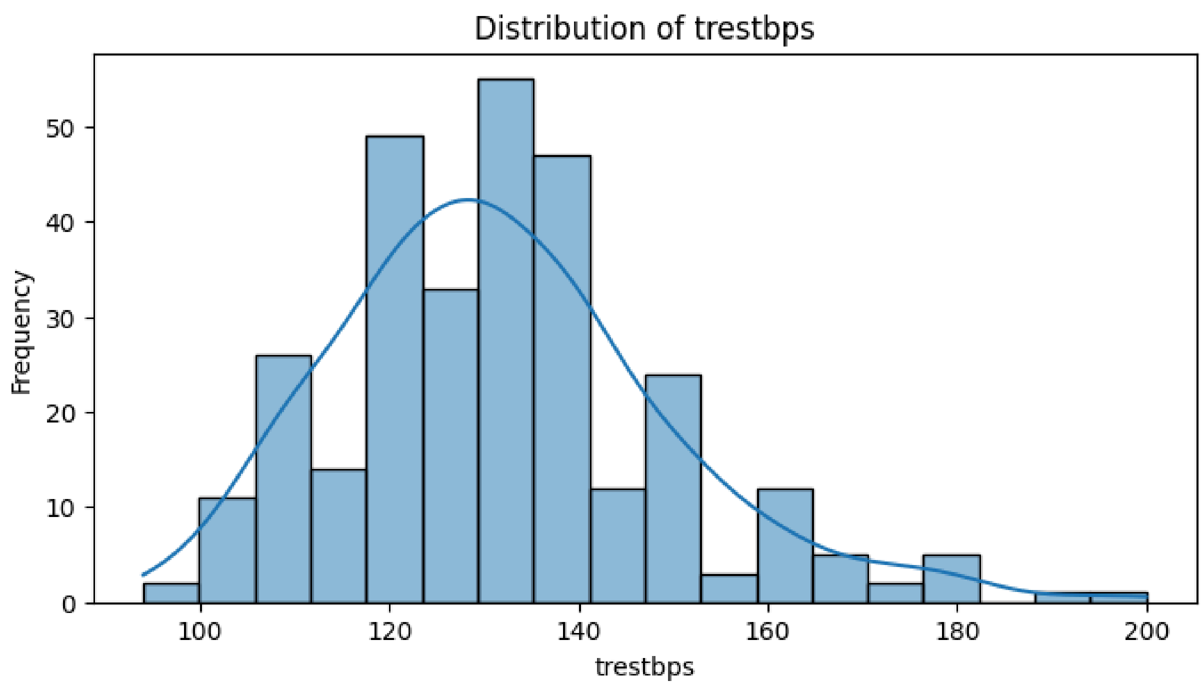


In [8]:

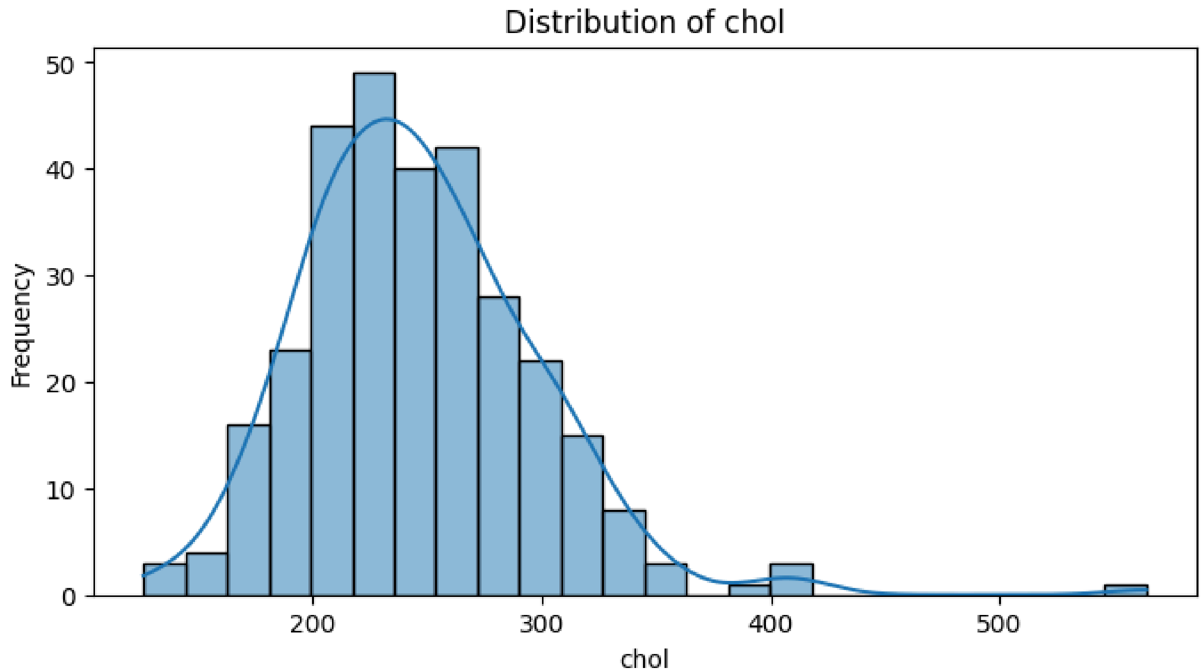
```
# Histogram for feature distribution
numerical_cols = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
for col in numerical_cols:
 plt.figure(figsize=(8, 4))
 sns.histplot(data[col], kde=True)
 plt.title(f'Distribution of {col}')
 plt.xlabel(f'{col}')
 plt.ylabel('Frequency')
 plt.show()
```

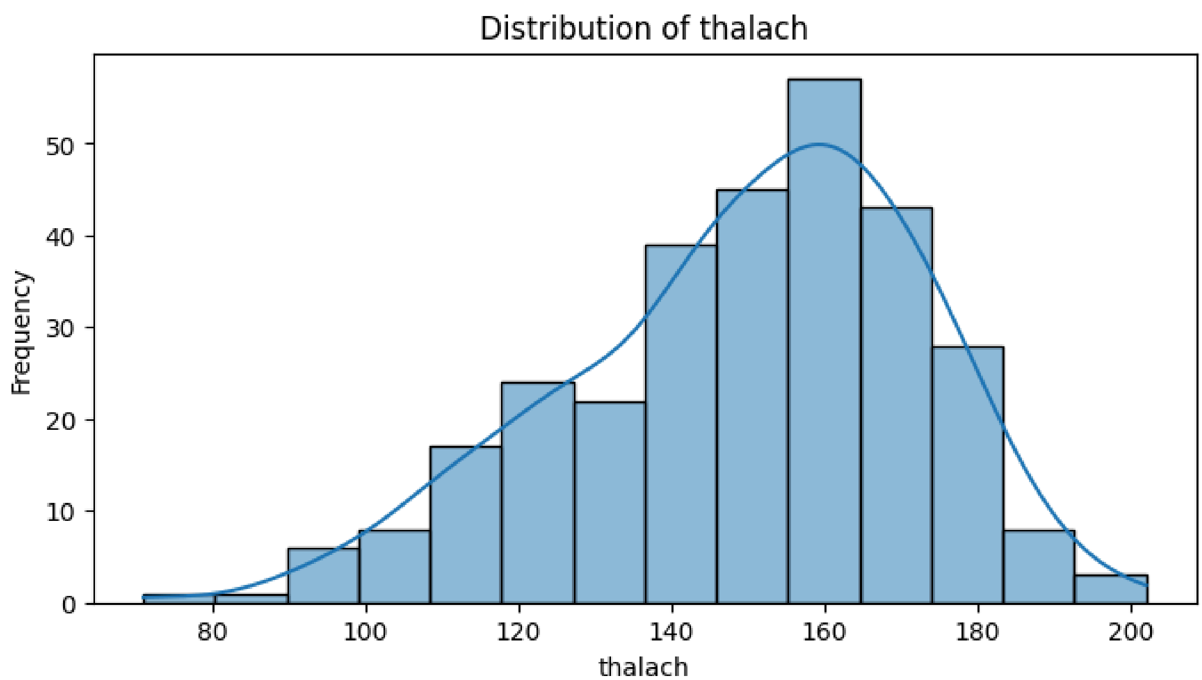


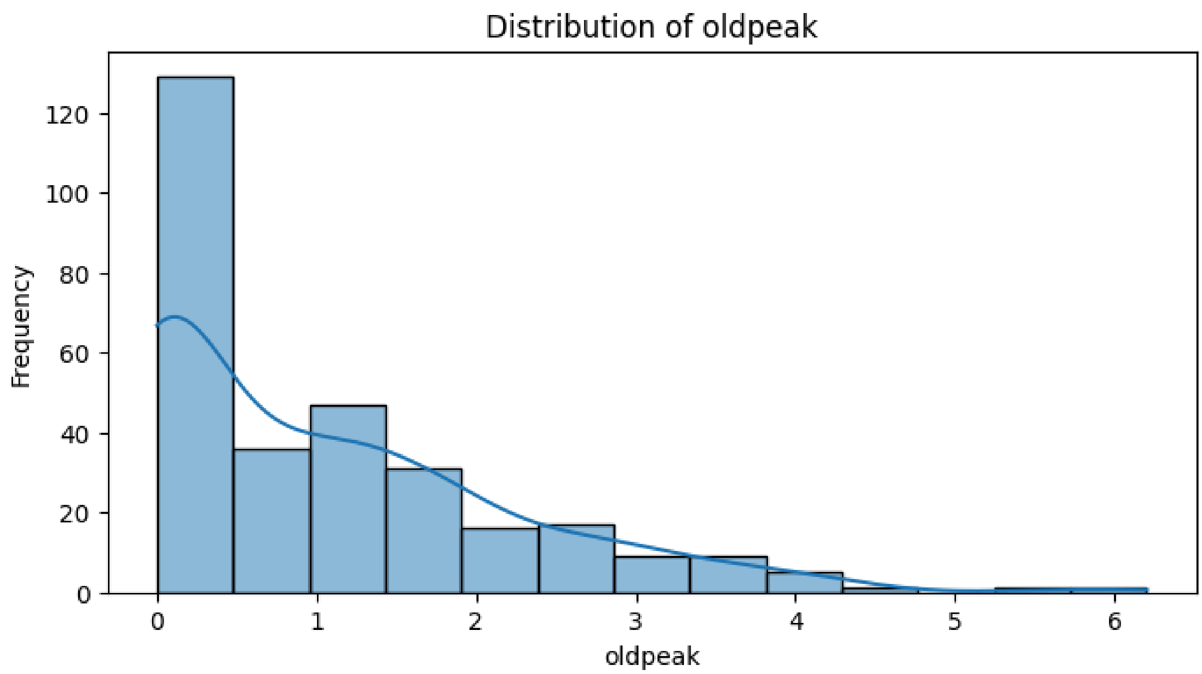
Distribution of age
Frequency
age
0
10
20
30
40
50
30
40
50
60
70



Distribution of trestbps
Frequency
trestbps
0
10
20
30
40
50
100
120
140
160
180
200


Distribution of chol
Frequency
chol
0
10
20
30
40
50
200
300
400
500


Distribution of thalach
Frequency
thalach
0
10
20
30
40
50
80
100
120
140
160
180
200



Model prepration

In [9]:
```
y = data["target"]
X = data.drop('target',axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_st
```

Before applying algorithm we should check whether the data is equally splitted or not, because if data is not splitted equally it will cause for data imbalacing problem

In [10]:
```
print(y_test.unique())
Counter(y_train)
```

```
[1 0]
```

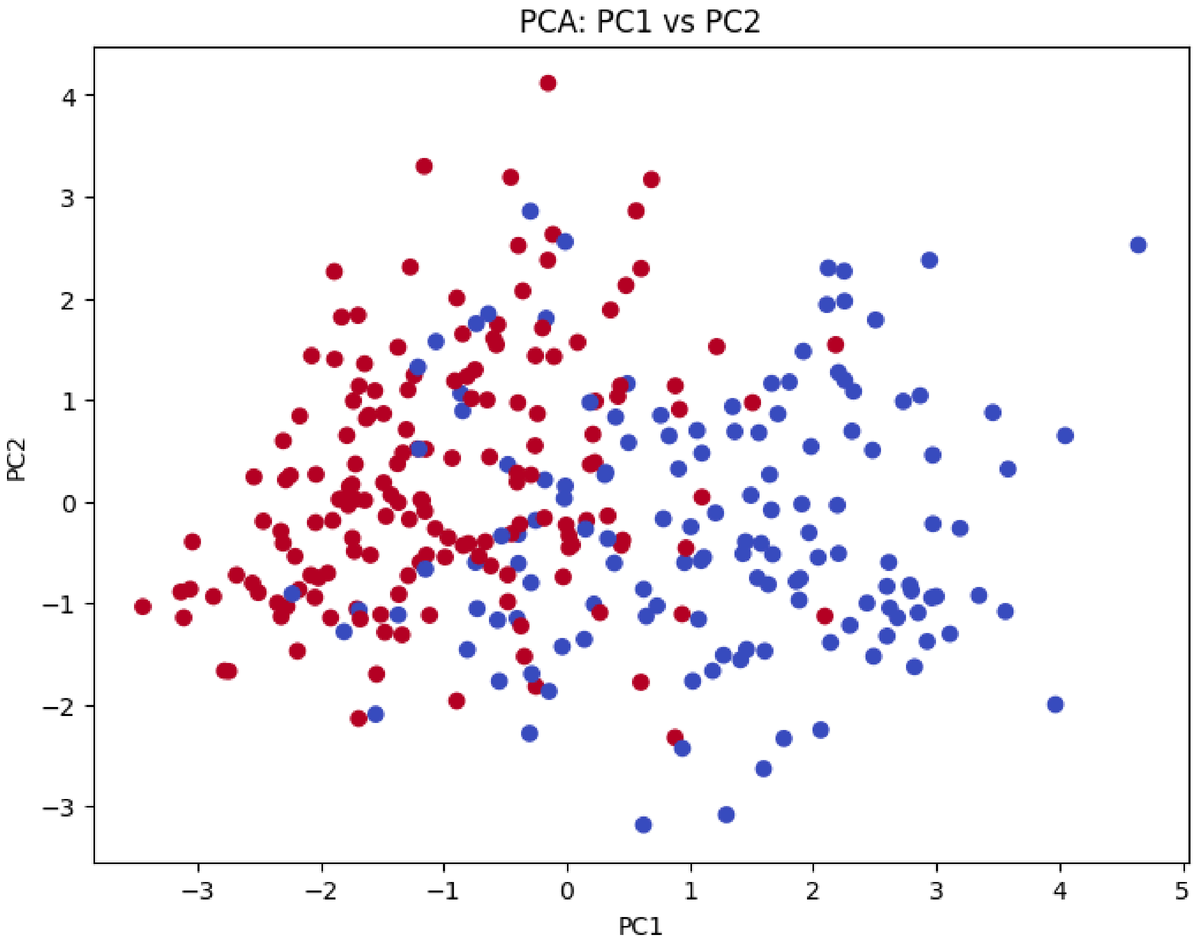
Out[10]:
```
Counter({1: 128, 0: 113})
```

In [11]:
```
scaler = StandardScaler()
scaled_data = scaler.fit_transform(data.drop('target', axis=1))
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Principal Component Analysis (PCA)

In [12]:
```
# Applying PCA and selecting PC1 and PC2
pca = PCA(n_components=2)
X_pca = pca.fit_transform(scaled_data)

# Visualizing PCA results
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='coolwarm')
plt.xlabel('PC1')
```

```
plt.ylabel('PC2')
plt.title('PCA: PC1 vs PC2')
plt.show()
```



ML models

Here I take different machine learning algorithm and try to find algorithm which predict accurately.

1. Logistic Regression
2. Naive Bayes
3. Random Forest Classifier
4. Extreme Gradient Boost
5. K-Nearest Neighbour
6. Decision Tree
7. Support Vector Machine

In [13]:

```
m1 = 'Logistic Regression'
lr = LogisticRegression()
model = lr.fit(X_train, y_train)
lr_predict = lr.predict(X_test)
lr_conf_matrix = confusion_matrix(y_test, lr_predict)
lr_acc_score = accuracy_score(y_test, lr_predict)
```

```
print("confussion matrix")
print(lr_conf_matrix)
print("\n")
print("Accuracy of Logistic Regression:",lr_acc_score*100,'\n')
print(classification_report(y_test,lr_predict))
```

```
confussion matrix
[[20  5]
 [ 5 31]]


Accuracy of Logistic Regression: 83.60655737704919

              precision    recall  f1-score   support

           0       0.80      0.80      0.80        25
           1       0.86      0.86      0.86        36

    accuracy                           0.84        61
   macro avg       0.83      0.83      0.83        61
weighted avg       0.84      0.84      0.84        61
```

In [14]:

```
m2 = 'Naive Bayes'
nb = GaussianNB()
nb.fit(X_train,y_train)
nbpred = nb.predict(X_test)
nb_conf_matrix = confusion_matrix(y_test, nbpred)
nb_acc_score = accuracy_score(y_test, nbpred)
print("confussion matrix")
print(nb_conf_matrix)
print("\n")
print("Accuracy of Naive Bayes model:",nb_acc_score*100,'\n')
print(classification_report(y_test,nbpred))
```

```
confussion matrix
[[20  5]
 [ 7 29]]


Accuracy of Naive Bayes model: 80.32786885245902

              precision    recall  f1-score   support

           0       0.74      0.80      0.77        25
           1       0.85      0.81      0.83        36

    accuracy                           0.80        61
   macro avg       0.80      0.80      0.80        61
weighted avg       0.81      0.80      0.80        61
```

In [15]:

```
m3 = 'Random Forest Classfier'
rf = RandomForestClassifier(n_estimators=20, random_state=12,max_depth=5)
rf.fit(X_train,y_train)
rf_predicted = rf.predict(X_test)
rf_conf_matrix = confusion_matrix(y_test, rf_predicted)
```

```
rf_acc_score = accuracy_score(y_test, rf_predicted)
print("confussion matrix")
print(rf_conf_matrix)
print("\n")
print("Accuracy of Random Forest:",rf_acc_score*100,'\n')
print(classification_report(y_test,rf_predicted))
```

```
confussion matrix
[[18  7]
 [ 5 31]]


Accuracy of Random Forest: 80.32786885245902

              precision    recall  f1-score   support

           0       0.78      0.72      0.75        25
           1       0.82      0.86      0.84        36

    accuracy                           0.80        61
   macro avg       0.80      0.79      0.79        61
weighted avg       0.80      0.80      0.80        61
```

In [16]:
```
m4 = 'Extreme Gradient Boost'
xgb = XGBClassifier(learning_rate=0.01, n_estimators=25, max_depth=15,gamma=0.6, su
                    reg_lambda=2, booster='dart', colsample_bylevel=0.6, colsample_
xgb.fit(X_train, y_train)
xgb_predicted = xgb.predict(X_test)
xgb_conf_matrix = confusion_matrix(y_test, xgb_predicted)
xgb_acc_score = accuracy_score(y_test, xgb_predicted)
print("confussion matrix")
print(xgb_conf_matrix)
print("\n")
print("Accuracy of Extreme Gradient Boost:",xgb_acc_score*100,'\n')
print(classification_report(y_test,xgb_predicted))
```

```
confussion matrix
[[13 12]
 [ 0 36]]


Accuracy of Extreme Gradient Boost: 80.32786885245902

              precision    recall  f1-score   support

           0       1.00      0.52      0.68        25
           1       0.75      1.00      0.86        36

    accuracy                           0.80        61
   macro avg       0.88      0.76      0.77        61
weighted avg       0.85      0.80      0.79        61
```

In [17]:
```
m5 = 'K-NeighborsClassifier'
knn = KNeighborsClassifier(n_neighbors=10)
knn.fit(X_train, y_train)
```

```python
knn_predicted = knn.predict(X_test)
knn_conf_matrix = confusion_matrix(y_test, knn_predicted)
knn_acc_score = accuracy_score(y_test, knn_predicted)
print("confussion matrix")
print(knn_conf_matrix)
print("\n")
print("Accuracy of K-NeighborsClassifier:",knn_acc_score*100,'\n')
print(classification_report(y_test,knn_predicted))
```

```
confussion matrix
[[19  6]
 [ 5 31]]


Accuracy of K-NeighborsClassifier: 81.9672131147541 

              precision    recall  f1-score   support

           0       0.79      0.76      0.78        25
           1       0.84      0.86      0.85        36

    accuracy                           0.82        61
   macro avg       0.81      0.81      0.81        61
weighted avg       0.82      0.82      0.82        61
```

In [18]:

```python
m6 = 'DecisionTreeClassifier'
dt = DecisionTreeClassifier(criterion = 'entropy',random_state=0,max_depth = 6)
dt.fit(X_train, y_train)
dt_predicted = dt.predict(X_test)
dt_conf_matrix = confusion_matrix(y_test, dt_predicted)
dt_acc_score = accuracy_score(y_test, dt_predicted)
print("confussion matrix")
print(dt_conf_matrix)
print("\n")
print("Accuracy of DecisionTreeClassifier:",dt_acc_score*100,'\n')
print(classification_report(y_test,dt_predicted))
```

```
confussion matrix
[[18  7]
 [ 6 30]]


Accuracy of DecisionTreeClassifier: 78.68852459016394 

              precision    recall  f1-score   support

           0       0.75      0.72      0.73        25
           1       0.81      0.83      0.82        36

    accuracy                           0.79        61
   macro avg       0.78      0.78      0.78        61
weighted avg       0.79      0.79      0.79        61
```

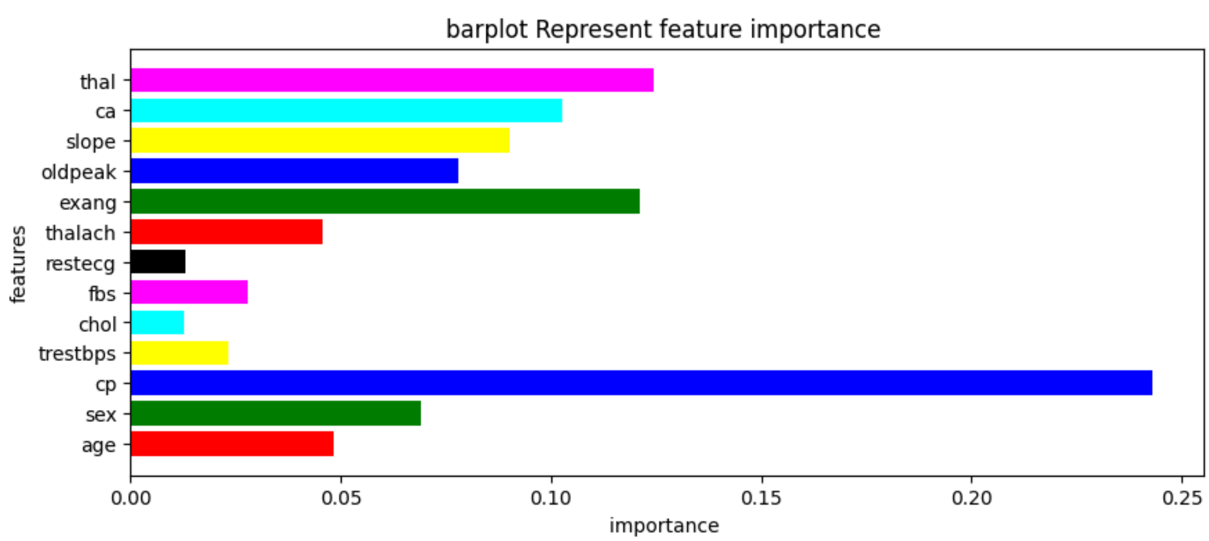
In [19]:

```python
m7 = 'Support Vector Classifier'
svc =  SVC(kernel='rbf', C=2)
```

```python
svc.fit(X_train, y_train)
svc_predicted = svc.predict(X_test)
svc_conf_matrix = confusion_matrix(y_test, svc_predicted)
svc_acc_score = accuracy_score(y_test, svc_predicted)
print("confussion matrix")
print(svc_conf_matrix)
print("\n")
print("Accuracy of Support Vector Classifier:",svc_acc_score*100,'\n')
print(classification_report(y_test,svc_predicted))
```

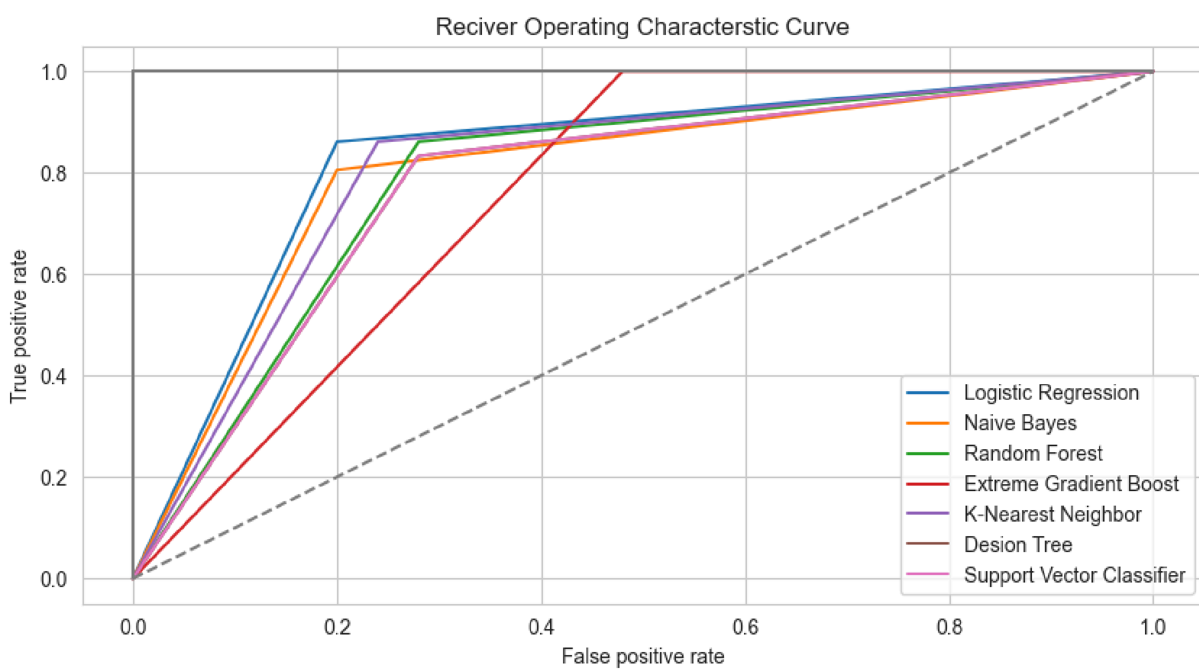
```
confussion matrix
[[18  7]
 [ 6 30]]


Accuracy of Support Vector Classifier: 78.68852459016394

              precision    recall  f1-score   support

           0       0.75      0.72      0.73        25
           1       0.81      0.83      0.82        36

    accuracy                           0.79        61
   macro avg       0.78      0.78      0.78        61
weighted avg       0.79      0.79      0.79        61
```

In [20]:
```python
imp_feature = pd.DataFrame({'Feature': ['age', 'sex', 'cp', 'trestbps', 'chol', 'fb
       'exang', 'oldpeak', 'slope', 'ca', 'thal'], 'Importance': xgb.feature_import
plt.figure(figsize=(10,4))
plt.title("barplot Represent feature importance ")
plt.xlabel("importance ")
plt.ylabel("features")
plt.barh(imp_feature['Feature'], imp_feature['Importance'], color=['red', 'green',
plt.show()
```



In [21]:
```python
lr_false_positive_rate,lr_true_positive_rate,lr_threshold = roc_curve(y_test,lr_pre
nb_false_positive_rate,nb_true_positive_rate,nb_threshold = roc_curve(y_test,nbpred
rf_false_positive_rate,rf_true_positive_rate,rf_threshold = roc_curve(y_test,rf_pre
xgb_false_positive_rate,xgb_true_positive_rate,xgb_threshold = roc_curve(y_test,xgb
```

```
knn_false_positive_rate,knn_true_positive_rate,knn_threshold = roc_curve(y_test,knn
dt_false_positive_rate,dt_true_positive_rate,dt_threshold = roc_curve(y_test,dt_pre
svc_false_positive_rate,svc_true_positive_rate,svc_threshold = roc_curve(y_test,svc


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 Boo
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 Class
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()
```



Model Evaluation

In [22]:

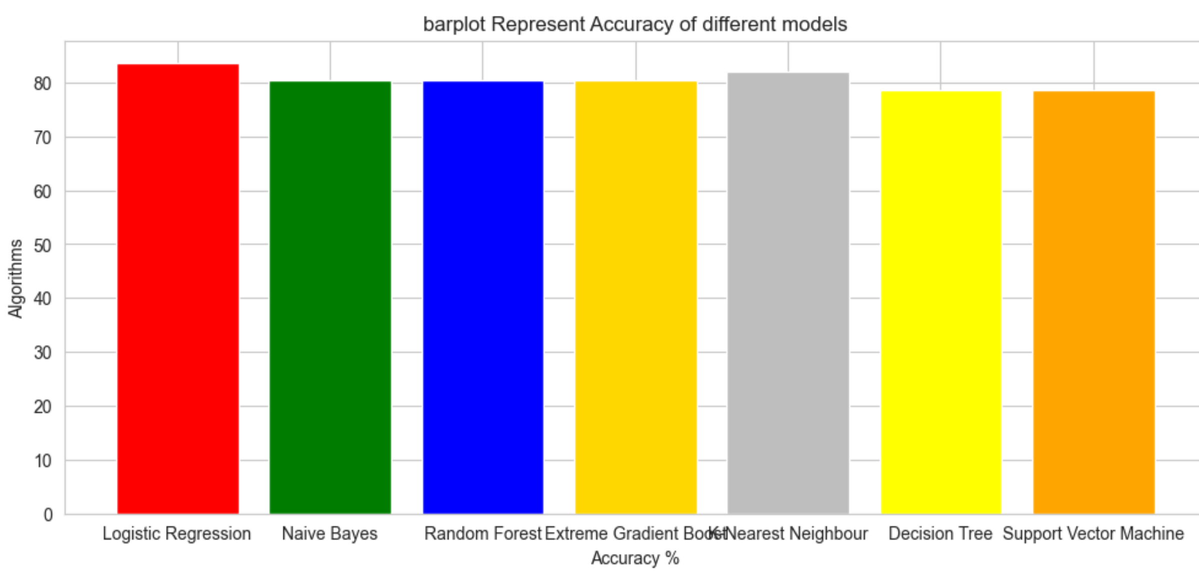
```
model_ev = pd.DataFrame({'Model': ['Logistic Regression','Naive Bayes','Random Fore
                    'K-Nearest Neighbour','Decision Tree','Support Vector Machine']
                    nb_acc_score*100,rf_acc_score*100,xgb_acc_score*100,knn_acc_sco
model_ev
```

Out[22]:

	Model	Accuracy
0	Logistic Regression	83.606557
1	Naive Bayes	80.327869
2	Random Forest	80.327869
3	Extreme Gradient Boost	80.327869
4	K-Nearest Neighbour	81.967213
5	Decision Tree	78.688525
6	Support Vector Machine	78.688525

In [23]:

```
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_ev['Model'],model_ev['Accuracy'],color = colors)
plt.show()
```



Conclusion

In this project, we applied various machine learning models to predict heart disease based on a dataset containing patient information. After performing data preprocessing, including scaling and encoding categorical variables, we utilized **Principal Component Analysis (PCA)** to reduce the dimensionality and visualize the data using the first two principal components (PC1 and PC2).

Several machine learning models were trained and evaluated, including:

- Logistic Regression

- Naive Bayes
- Random Forest Classifier
- Extreme Gradient Boosting (XGBoost)
- K-Nearest Neighbors (KNN)
- Decision Tree Classifier
- Support Vector Machine (SVM)

Model Performance:

The accuracy results of the models were as follows:

1. **Logistic Regression**: 83.61%
2. **Naive Bayes**: 80.33%
3. **Random Forest Classifier**: 80.33%
4. **Extreme Gradient Boosting**: 80.33%
5. **K-Nearest Neighbors (KNN)**: 81.97%
6. **Decision Tree Classifier**: 78.69%
7. **Support Vector Machine (SVM)**: 78.69%

Insights:

- Logistic Regression emerged as the most accurate model, closely followed by K-Nearest Neighbors.
- All models performed similarly, with accuracies between 78% and 83%, indicating consistent predictive power across different techniques.
- The models demonstrated that features like chest pain type (cp), maximum heart rate achieved (thalach), and ST depression (oldpeak) are important predictors for heart disease.
- PCA helped to visualize the separation between classes, confirming that the data contains patterns that machine learning models can exploit effectively.