1. import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

import os

print(os.listdir())

import warnings

warnings.filterwarnings('ignore')

1. dataset = pd.read\_csv("dataset.csv")
2. type(dataset)
3. dataset.shape
4. dataset.head(5)
5. dataset.sample(5)
6. dataset.describe()
7. dataset.info()
8. info = ["age","1: male, 0: female","chest pain type, 1: typical angina, 2: atypical angina, 3: non-anginal pain, 4: asymptomatic","resting blood pressure"," serum cholestoral in mg/dl","fasting blood sugar > 120 mg/dl","resting electrocardiographic results (values 0,1,2)"," maximum heart rate achieved","exercise induced angina","oldpeak = ST depression induced by exercise relative to rest","the slope of the peak exercise ST segment","number of major vessels (0-3) colored by flourosopy","thal: 3 = normal; 6 = fixed defect; 7 = reversable defect"]

for i in range(len(info)):

    print(dataset.columns[i]+":\t\t\t"+info[i])

1. dataset["target"].describe()
2. dataset["target"].unique()
3. print(dataset.corr()["target"].abs().sort\_values(ascending=False))
4. sns.countplot(data=dataset, x="target")

# Calculate and print the count of each unique value in the "target" column

target\_temp = dataset["target"].value\_counts()

print(target\_temp)

1. print("Percentage of patience without heart problems: "+str(round(target\_temp[0]\*100/303,2)))

print("Percentage of patience with heart problems: "+str(round(target\_temp[1]\*100/303,2)))

1. from sklearn.model\_selection import train\_test\_split

predictors = dataset.drop("target",axis=1)

target = dataset["target"]

X\_train,X\_test,Y\_train,Y\_test = train\_test\_split(predictors,target,test\_size=0.20,random\_state=0)

1. X\_train.shape
2. X\_test.shape
3. Y\_train.shape
4. Y\_test.shape
5. from sklearn.metrics import accuracy\_score
6. from sklearn.linear\_model import LogisticRegression

lr = LogisticRegression()

lr.fit(X\_train,Y\_train)

Y\_pred\_lr = lr.predict(X\_test)

1. Y\_pred\_lr.shape
2. score\_lr = round(accuracy\_score(Y\_pred\_lr,Y\_test)\*100,2)

print("The accuracy score achieved using Logistic Regression is: "+str(score\_lr)+" %")

1. from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n\_neighbors=7)

knn.fit(X\_train,Y\_train)

Y\_pred\_knn=knn.predict(X\_test)

1. Y\_pred\_knn.shape
2. score\_knn = round(accuracy\_score(Y\_pred\_knn,Y\_test)\*100,2)

print("The accuracy score achieved using KNN is: "+str(score\_knn)+" %")

1. from sklearn.ensemble import RandomForestClassifier

max\_accuracy = 0

for x in range(2000):

    rf = RandomForestClassifier(random\_state=x)

    rf.fit(X\_train,Y\_train)

    Y\_pred\_rf = rf.predict(X\_test)

    current\_accuracy = round(accuracy\_score(Y\_pred\_rf,Y\_test)\*100,2)

    if(current\_accuracy>max\_accuracy):

        max\_accuracy = current\_accuracy

        best\_x = x

#print(max\_accuracy)

#print(best\_x)

rf = RandomForestClassifier(random\_state=best\_x)

rf.fit(X\_train,Y\_train)

Y\_pred\_rf = rf.predict(X\_test)

1. Y\_pred\_rf.shape
2. score\_rf = round(accuracy\_score(Y\_pred\_rf,Y\_test)\*100,2)

print("The accuracy score achieved using Decision Tree is: "+str(score\_rf)+" %")

1. from sklearn.naive\_bayes import GaussianNB

nb = GaussianNB()

nb.fit(X\_train,Y\_train)

Y\_pred\_nb = nb.predict(X\_test)

1. Y\_pred\_nb.shape
2. score\_nb = round(accuracy\_score(Y\_pred\_nb,Y\_test)\*100,2)

print("The accuracy score achieved using Naive Bayes is: "+str(score\_nb)+" %")

1. from sklearn.tree import DecisionTreeClassifier

max\_accuracy = 0

for x in range(200):

    dt = DecisionTreeClassifier(random\_state=x)

    dt.fit(X\_train,Y\_train)

    Y\_pred\_dt = dt.predict(X\_test)

    current\_accuracy = round(accuracy\_score(Y\_pred\_dt,Y\_test)\*100,2)

    if(current\_accuracy>max\_accuracy):

        max\_accuracy = current\_accuracy

        best\_x = x

#print(max\_accuracy)

#print(best\_x)

dt = DecisionTreeClassifier(random\_state=best\_x)

dt.fit(X\_train,Y\_train)

Y\_pred\_dt = dt.predict(X\_test)

1. print(Y\_pred\_dt.shape)

score\_dt = round(accuracy\_score(Y\_pred\_dt,Y\_test)\*100,2)

print("The accuracy score achieved using Decision Tree is: "+str(score\_dt)+" %")

1. scores = [score\_lr,score\_nb,score\_knn,score\_dt,score\_rf]

algorithms = ["Logistic Regression","Naive Bayes","K-Nearest Neighbors","Decision Tree","Random Forest"]

for i in range(len(algorithms)):

    print("The accuracy score achieved using "+algorithms[i]+" is: "+str(scores[i])+" %")

37)sns.barplot(x=algorithms, y=scores)

plt.gca().set\_yticklabels(['{:.0f}%'.format(x) for x in plt.gca().get\_yticks()])

plt.xlabel("Algorithms")

plt.ylabel("Accuracy Score (%)")

plt.title("Accuracy Scores of Different Algorithms")

plt.xticks(rotation=45)

plt.show()

38)def plot\_confusion\_matrix(conf\_matrix, algo\_name):

plt.figure(figsize=(6, 4))

sns.heatmap(conf\_matrix, annot=True, cmap='Blues', fmt='d', cbar=False)

plt.title("Confusion Matrix - " + algo\_name)

plt.xlabel("Predicted label")

plt.ylabel("True label")

plt.show()

39)conf\_matrix\_lr = confusion\_matrix(Y\_test, Y\_pred\_lr)

plot\_confusion\_matrix(conf\_matrix\_lr, "Logistic Regression")

40)precision\_lr = precision\_score(Y\_test, Y\_pred\_lr)

recall\_lr = recall\_score(Y\_test, Y\_pred\_lr)

f1\_lr = f1\_score(Y\_test, Y\_pred\_lr)

print("Validation for Logistic Regression:")

print("Precision:", precision\_lr)

print("Recall:", recall\_lr)

print("F1-score:", f1\_lr)

41)conf\_matrix\_knn = confusion\_matrix(Y\_test, Y\_pred\_knn)

plot\_confusion\_matrix(conf\_matrix\_knn, "K-Nearest Neighbors")

42)precision\_knn = precision\_score(Y\_test, Y\_pred\_knn)

recall\_knn = recall\_score(Y\_test, Y\_pred\_knn)

f1\_knn = f1\_score(Y\_test, Y\_pred\_knn)

print("\nValidation for K-Nearest Neighbors:")

print("Precision:", precision\_knn)

print("Recall:", recall\_knn)

print("F1-score:", f1\_knn)

43)conf\_matrix\_rf = confusion\_matrix(Y\_test, Y\_pred\_rf)

plot\_confusion\_matrix(conf\_matrix\_rf, "Random Forest")

44)precision\_rf = precision\_score(Y\_test, Y\_pred\_rf)

recall\_rf = recall\_score(Y\_test, Y\_pred\_rf)

f1\_rf = f1\_score(Y\_test, Y\_pred\_rf)

print("\nValidation for Random Forest:")

print("Precision:", precision\_rf)

print("Recall:", recall\_rf)

print("F1-score:", f1\_rf)

45)conf\_matrix\_nb = confusion\_matrix(Y\_test, Y\_pred\_nb)

plot\_confusion\_matrix(conf\_matrix\_nb, "Naive Bayes")

46)precision\_nb = precision\_score(Y\_test, Y\_pred\_nb)

recall\_nb = recall\_score(Y\_test, Y\_pred\_nb)

f1\_nb = f1\_score(Y\_test, Y\_pred\_nb)

print("\nValidation for Naive Bayes:")

print("Precision:", precision\_nb)

print("Recall:", recall\_nb)

print("F1-score:", f1\_nb)

47)conf\_matrix\_dt = confusion\_matrix(Y\_test, Y\_pred\_dt)

plot\_confusion\_matrix(conf\_matrix\_dt, "Decision Tree")

48)precision\_dt = precision\_score(Y\_test, Y\_pred\_dt)

recall\_dt = recall\_score(Y\_test, Y\_pred\_dt)

f1\_dt = f1\_score(Y\_test, Y\_pred\_dt)

print("\nValidation for Decision Tree:")

print("Precision:", precision\_dt)

print("Recall:", recall\_dt)

print("F1-score:", f1\_dt)

49)algorithms = ["Logistic Regression", "K-Nearest Neighbors", "Random Forest", "Naive Bayes", "Decision Tree"]

precision = [precision\_lr, precision\_knn, precision\_rf, precision\_nb, precision\_dt]

recall = [recall\_lr, recall\_knn, recall\_rf, recall\_nb, recall\_dt]

f1\_score = [f1\_lr, f1\_knn, f1\_rf, f1\_nb, f1\_dt]

comparison\_df = pd.DataFrame({

"Precision": precision,

"Recall": recall,

"F1-score": f1\_score

}, index=algorithms)

comparison\_df.plot(kind='bar', figsize=(10, 6))

plt.title('Comparison of Precision, Recall, and F1-score for Different Algorithms')

plt.xlabel('Algorithms')

plt.ylabel('Score')

plt.xticks(rotation=45)

plt.legend(title='Metrics')

plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.tight\_layout()

plt.show()