```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import KFold, StratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files und

#import os
#for dirname, _, filenames in os.walk('/kaggle/input'):
    #for filename in filenames:
        #print(os.path.join(dirname, filename))
import os
print(os.listdir("../input"))
#lists all files available in ../input

# You can write up to 5GB to the current directory (/kaggle/working/) that gets preserved as
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the
```

```
data1= pd.read_csv("../input/HeartDate.csv")
data2= pd.read_csv("../input/HeartDate.csv")
```

```
data1.head()
```

```
sns.countplot(x="target",data= data1,palette= "bwr")
plt.show()
```

```
#we already performed data exploration in google colab, (see attached file), so we will not r
pd.crosstab(data1.age,data1.target).plot(kind="bar",figsize=(20,6))
plt.title('Heart Disease Frequency in terms of Age')
plt.xlabel('Age')
plt.ylabel('Frequency')
#plt.savefig('heartDiseaseAndAges.png')
plt.show()
```

```
#we see in the above plot that age 54 has the higest amount of confirmed heart disease cases.
#We now compare cholesterol levels with a scatter plot
plt.scatter(x=data1.age[data1.target==1], y=data1.chol[(data1.target==1)], c="red")
plt.scatter(x=data1.age[data1.target==0], y=data1.chol[(data1.target==0)])
plt.legend(["Have disease", "Do not have disease"])
plt.xlabel("Age")
plt.ylabel("Cholesterol levels")
plt.show()



#We now compare maximum heart rate achieved and age with a scatter plot
plt.scatter(x=data1.age[data1.target==1], y=data1.thalach[(data1.target==1)], c="red")
plt.scatter(x=data1.age[data1.target==0], y=data1.thalach[(data1.target==0)])
plt.legend(["Have disease", "Do not have disease"])
plt.xlabel("Age")
plt.ylabel("Maximum heart rate achieved")
plt.show()


pd.crosstab(data1.cp,data1.target).plot(kind="bar",figsize=(17,6),color=['#FFC300','#581845'
plt.title('Heart Disease Frequency According To Chest Pain Type')
plt.xlabel('Chest Pain Type')
plt.xticks(rotation = 0)
plt.ylabel('Frequency of Disease and No disease')
plt.show()


#creating dummy variables
a = pd.get_dummies(data1['cp'], prefix = "cp")
b = pd.get_dummies(data1['thal'], prefix = "thal")
c = pd.get_dummies(data1['slope'], prefix = "slope")


frames = [data1, a, b, c]
data1 = pd.concat(frames, axis = 1)
data1.head()


data1 = data1.drop(columns = ['cp', 'thal', 'slope'])
data1.head()


#logistic regression
y = data1.target.values
x_data = data1.drop(['target'], axis = 1)


# Normalize
x = (x_data - np.min(x_data)) / (np.max(x_data) - np.min(x_data)).values


x_train, x_test, y_train, y_test = train_test_split(x,y,test_size = 0.25,random_state=0)
x_train = x_train.T
y_train = y_train.T
x_test = x_test.T
y_test = y_test.T


#sklearn
accuracies = {}

lr = LogisticRegression()
```

```
lr.fit(x_train.T,y_train.T)
acc = lr.score(x_test.T,y_test.T)*100

accuracies['Logistic Regression'] = acc
print("Test Accuracy {:.2f}%".format(acc))


# KNN Model
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 2)  # n_neighbors means k
knn.fit(x_train.T, y_train.T)
prediction = knn.predict(x_test.T)

print("{} NN Score: {:.2f}%".format(2, knn.score(x_test.T, y_test.T)*100))


#Since the accuracy isn't the best with k=2, we will try to find the best k value.
scoreList = []
for i in range(1,20):
    knn2 = KNeighborsClassifier(n_neighbors = i)  # n_neighbors means k
    knn2.fit(x_train.T, y_train.T)
    scoreList.append(knn2.score(x_test.T, y_test.T))

plt.plot(range(1,20), scoreList)
plt.xticks(np.arange(1,20,1))
plt.xlabel("K value")
plt.ylabel("Score")
plt.show()

acc = max(scoreList)*100
accuracies['KNN'] = acc
print("Maximum KNN Score is {:.2f}%".format(acc))


#we obtain the best k values for 5 and 6


#Random forest
# Random Forest Classification
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators = 1000, random_state = 1)
rf.fit(x_train.T, y_train.T)

acc = rf.score(x_test.T,y_test.T)*100
accuracies['Random Forest'] = acc
print("Random Forest Algorithm Accuracy Score : {:.2f}%".format(acc))


#Naive Bayes
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(x_train.T, y_train.T)

acc = nb.score(x_test.T,y_test.T)*100
```

```
accuracies['Naive Bayes'] = acc
print("Accuracy of Naive Bayes: {:.2f}%".format(acc))


#tree
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier()
dtc.fit(x_train.T, y_train.T)

acc = dtc.score(x_test.T, y_test.T)*100
accuracies['Decision Tree'] = acc
print("Decision Tree Test Accuracy {:.2f}%".format(acc))


colors = ["purple", "green", "orange", "magenta","#CFC60E"]

sns.set_style("whitegrid")
plt.figure(figsize=(16,5))
plt.yticks(np.arange(0,100,10))
plt.ylabel("Accuracy % by using test set in sklearn library")
plt.xlabel("Models")
sns.barplot(x=list(accuracies.keys()), y=list(accuracies.values()), palette=colors)
plt.show()


#By using a 25% test set (using train-test split), we see that Random forest performs the bes


#K fold method
kf= KFold(n_splits=10, shuffle= True)
lr_reg=LogisticRegression()


scores=[]
ydash= data1.target
for i in range(10):
    result= next(kf.split(x_data), None)
    x_tr=x_data.iloc[result[0]]
    x_te=x_data.iloc[result[1]]
    y_tr=ydash.iloc[result[0]]
    y_te=ydash.iloc[result[1]]
    model=lr_reg.fit(x_tr, y_tr)
    predictions= lr_reg.predict(x_te)
    scores.append(model.score(x_te, y_te))
print('Score from each iteration: ', scores)
print('Average kfold score:' , np.mean(scores))


rf_reg= RandomForestClassifier(n_estimators = 1000, random_state = 1)
scores2=[]
for i in range(10):
    result= next(kf.split(x_data), None)
    x_tr1=x_data.iloc[result[0]]
    x_te1=x_data.iloc[result[1]]
    y_tr1=ydash.iloc[result[0]]
    y_te1=ydash.iloc[result[1]]
    model=rf_reg.fit(x_tr1, y_tr1)
    predictions= rf_reg.predict(x_te1)
    scores2.append(model.score(x_te1, y_te1))
print('Score from each iteration: ', scores2)
print('Average kfold score:' , np.mean(scores2))


nb_reg= GaussianNB()
scores3=[]
for i in range(10):
    result= next(kf.split(x_data), None)
    x_tr2=x_data.iloc[result[0]]
    x_te2=x_data.iloc[result[1]]
    y_tr2=ydash.iloc[result[0]]
    y_te2=ydash.iloc[result[1]]
    model=nb_reg.fit(x_tr2, y_tr2)
    predictions= nb_reg.predict(x_te2)
    scores3.append(model.score(x_te2, y_te2))
print('Score from each iteration: ', scores3)
print('Average kfold score:' , np.mean(scores3))


t_reg= DecisionTreeClassifier()
scores4=[]
for i in range(10):
    result= next(kf.split(x_data), None)
    x_tr3=x_data.iloc[result[0]]
    x_te3=x_data.iloc[result[1]]
    y_tr3=ydash.iloc[result[0]]
    y_te3=ydash.iloc[result[1]]
    model=t_reg.fit(x_tr3, y_tr3)
    predictions= t_reg.predict(x_te3)
    scores4.append(model.score(x_te3, y_te3))
print('Score from each iteration: ', scores4)
print('Average kfold score:' , np.mean(scores4))


#we use n neighbor=5 since that provided the ebst results.
k_reg= KNeighborsClassifier(n_neighbors = 6)
scores5=[]
for i in range(10):
    result= next(kf.split(x_data), None)
    x_tr4=x_data.iloc[result[0]]
    x_te4=x_data.iloc[result[1]]
    y_tr4=ydash.iloc[result[0]]
    y_te4=ydash.iloc[result[1]]
    model=k_reg.fit(x_tr4, y_tr4)
    predictions= k_reg.predict(x_te4)
    scores5.append(model.score(x_te4, y_te4))
print('Score from each iteration: ', scores5)
print('Average kfold score:' , np.mean(scores5))
```

```
kfoldAccuracies={}
kfoldAccuracies['KNN'] = 0.5935483870967742
kfoldAccuracies['Tree']= 0.7419354838709677
kfoldAccuracies['Naive Bayes']= 0.8193548387096774
kfoldAccuracies['Random Forest']= 0.8129032258064516
kfoldAccuracies['Logistic Regression']= 0.8483870967741935

colors = ["purple", "green", "orange", "magenta","#CFC60E"]

sns.set_style("whitegrid")
plt.figure(figsize=(16,5))
plt.yticks(np.arange(0,100,10))
plt.ylabel("Accuracy % by using K fold Accuracy")
plt.xlabel("Models")
sns.barplot(x=list(kfoldAccuracies.keys()), y=list(kfoldAccuracies.values()), palette=colors)
plt.show()


sns.heatmap(data2.corr(),annot=True,cmap='RdYlGn',linewidths=0.2) #data.corr()-->correlation
fig=plt.gcf()
fig.set_size_inches(20,12)
plt.show()


#kfolds stratified:


kf2= StratifiedKFold(n_splits=10, shuffle= True)


#knn
StratkfoldAcc={}
k_reg= KNeighborsClassifier(n_neighbors = 6)
scores6=[]
ydash= data1.target
i=1
for train_index,test_index in kf2.split(x_data,ydash):
    #print('{} of KFold {}'.format(i,kf2.n_splits))
    xtr,xvl = x_data.loc[train_index],x_data.loc[test_index]
    ytr,yvl = ydash.loc[train_index],ydash.loc[test_index]

    #model
    model=k_reg.fit(xtr,ytr)
    predictions= k_reg.predict(xvl)
    scores6.append(model.score(xvl, yvl))
print('Score from each iteration: ', scores6)
StratkfoldAcc['KNN'] =np.mean(scores6)
print('Average Stratified kfold score:' , np.mean(scores6))


#tree
t_reg= DecisionTreeClassifier()
scores7=[]
ydash= data1.target
```

```
i=1
for train_index,test_index in kf2.split(x_data,ydash):
    #print('{} of KFold {}'.format(i,kf2.n_splits))
    xtr,xvl = x_data.loc[train_index],x_data.loc[test_index]
    ytr,yvl = ydash.loc[train_index],ydash.loc[test_index]

    #model
    model=t_reg.fit(xtr,ytr)
    predictions= t_reg.predict(xvl)
    scores7.append(model.score(xvl, yvl))
print('Score from each iteration: ', scores7)
StratkfoldAcc['Tree'] =np.mean(scores7)
print('Average Stratified kfold score:' , np.mean(scores7))



nb_reg= GaussianNB()
scores8=[]
ydash= data1.target
i=1
for train_index,test_index in kf2.split(x_data,ydash):
    #print('{} of KFold {}'.format(i,kf2.n_splits))
    xtr,xvl = x_data.loc[train_index],x_data.loc[test_index]
    ytr,yvl = ydash.loc[train_index],ydash.loc[test_index]

    #model
    model=nb_reg.fit(xtr,ytr)
    predictions= nb_reg.predict(xvl)
    scores8.append(model.score(xvl, yvl))
print('Score from each iteration: ', scores8)
StratkfoldAcc['Naive Bayes'] =np.mean(scores8)
print('Average Stratified kfold score:' , np.mean(scores8))


rf_reg=RandomForestClassifier(n_estimators = 1000, random_state = 1)
scores9=[]
ydash= data1.target
i=1
for train_index,test_index in kf2.split(x_data,ydash):
    #print('{} of KFold {}'.format(i,kf2.n_splits))
    xtr,xvl = x_data.loc[train_index],x_data.loc[test_index]
    ytr,yvl = ydash.loc[train_index],ydash.loc[test_index]

    #model
    model=rf_reg.fit(xtr,ytr)
    predictions= rf_reg.predict(xvl)
    scores9.append(model.score(xvl, yvl))
print('Score from each iteration: ', scores9)
StratkfoldAcc['Random Forest'] =np.mean(scores9)
print('Average Stratified kfold score:' , np.mean(scores9))


lr_reg=LogisticRegression()
```

```
scores10=[]
ydash= data1.target
i=1
for train_index,test_index in kf2.split(x_data,ydash):
    #print('{} of KFold {}'.format(i,kf2.n_splits))
    xtr,xvl = x_data.loc[train_index],x_data.loc[test_index]
    ytr,yvl = ydash.loc[train_index],ydash.loc[test_index]

    #model
    model=lr_reg.fit(xtr,ytr)
    predictions= lr_reg.predict(xvl)
    scores10.append(model.score(xvl, yvl))
print('Score from each iteration: ', scores10)
StratkfoldAcc['Logistic Regression'] =np.mean(scores10)
print('Average Stratified kfold score:' , np.mean(scores10))


colors = ["purple", "green", "orange", "magenta","#CFC60E"]

sns.set_style("whitegrid")
plt.figure(figsize=(16,5))
plt.yticks(np.arange(0,100,10))
plt.ylabel("Accuracy % by using Stratified K fold Accuracy")
plt.xlabel("Models")
sns.barplot(x=list(StratkfoldAcc.keys()), y=list(StratkfoldAcc.values()), palette=colors)
plt.show()
```