import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import statistics as s

sns.set\_style("whitegrid")

plt.style.use("fivethirtyeight")

df = pd.read\_csv(r"C:\Users\Pc\Downloads\heart.csv")

df.head()

df.info()

size = df.size

print("Size of dataset is :",size)

shape = df.shape

print("shape of datset is \n\n:",shape)

print(df.describe())

print(df.head())

print("Data Type for Each Columns are\n",df.dtypes.value\_counts())

df.dtypes == 'object'

n = df.columns[df.dtypes != 'object']

df[n]

print("",df[n].isnull())

df[n].isnull().sum().sort\_values(ascending=False)

df[n].isnull().sum().sort\_values(ascending=False)/len(df)

df['age']

average = s.mean(df['age'])

print("Average age : ",average)

print(df['age'])

print(df['sex'])

print(df['trestbps'])

print(df['chol'])

corr\_matrix = df.corr()

fig, ax = plt.subplots(figsize=(15, 15))

ax = sns.heatmap(corr\_matrix,

annot=True,

linewidths=0.5,

fmt=".2f",

cmap="YlGnBu");

bottom, top = ax.get\_ylim()

ax.set\_ylim(bottom + 0.5, top - 0.5)

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

def print\_score(clf, X\_train, y\_train, X\_test, y\_test, train=True):

if train:

pred = clf.predict(X\_train)

clf\_report = pd.DataFrame(classification\_report(y\_train, pred, output\_dict=True))

print("Train Result:\n================================================")

print("Accuracy Score: {accuracy\_score(y\_train, pred) \* 100:.2f}%")

print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

print("CLASSIFICATION REPORT:\n{clf\_report}")

print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

print("Confusion Matrix: \n {confusion\_matrix(y\_train, pred)}\n")

elif train==False:

pred = clf.predict(X\_test)

clf\_report = pd.DataFrame(classification\_report(y\_test, pred, output\_dict=True))

print("Test Result:\n================================================")

print("Accuracy Score: {accuracy\_score(y\_test, pred) \* 100:.2f}%")

print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

print("CLASSIFICATION REPORT:\n{clf\_report}")

print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

print("Confusion Matrix: \n {confusion\_matrix(y\_test, pred)}\n")

#Spliting the data into training and testing data

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

X = df.drop('target', axis=1)

X.shape

y = df.target

y.shape

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=56)

print(X\_train)

#Classification using Logistic Regression

from sklearn.linear\_model import LogisticRegression

lr\_clf = LogisticRegression(solver='liblinear')

lr\_clf.fit(X\_train, y\_train)

print\_score(lr\_clf, X\_train, y\_train, X\_test, y\_test, train=True)

print\_score(lr\_clf, X\_train, y\_train, X\_test, y\_test, train=False)

#Displaying Training and Testing Accuracy for Logistic Regression

test\_score = accuracy\_score(y\_test, lr\_clf.predict(X\_test)) \* 100

train\_score = accuracy\_score(y\_train, lr\_clf.predict(X\_train)) \* 100

results\_df = pd.DataFrame(data=[["Logistic Regression", train\_score, test\_score]],

columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])

results\_df

#Classification using K-nearest neighbors

from sklearn.neighbors import KNeighborsClassifier

knn\_clf = KNeighborsClassifier()

knn\_clf.fit(X\_train, y\_train)

print\_score(knn\_clf, X\_train, y\_train, X\_test, y\_test, train=True)

print\_score(knn\_clf, X\_train, y\_train, X\_test, y\_test, train=False)

#Displaying Training and Testing Accuracy for K-nearest neighbors

test\_score = accuracy\_score(y\_test, knn\_clf.predict(X\_test)) \* 100

train\_score = accuracy\_score(y\_train, knn\_clf.predict(X\_train)) \* 100

results\_df\_2 = pd.DataFrame(data=[["K-nearest neighbors", train\_score, test\_score]],

columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])

results\_df = results\_df.append(results\_df\_2, ignore\_index=True)

results\_df