The Code of the Health care diabetes project

%matplotlib inline

### import libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from matplotlib import style

import seaborn as sns

healthcaredata = pd.read\_csv('health care diabetes.csv')

healthcaredata

**#Project Task: Week 1 Data Exploration:**

**# Perform descriptive analysis. Understand the variables and their corresponding values.**

**#On the columns below, a value of zero does not make sense and thus indicates missing value:**

#• Glucose

#• BloodPressure

#• SkinThickness

#• Insulin

#• BMI

healthcaredata.describe

healthcaredata.dtypes

# The health care diabetes data has mostly numerical data types integer,float values

healthcaredata.isnull().sum()

# There are no null values in the health care diabetes data.

healthcaredata.info()

Positive = healthcaredata[healthcaredata['Outcome']==1]

Positive

# We are looking only for the positive values,no values less than 0.

**#Visually explore these variables using histograms. Treat the missing values accordingly.**

plt.hist(healthcaredata['Glucose'])

plt.hist(healthcaredata['BloodPressure'])

plt.hist(healthcaredata['SkinThickness'])

plt.hist(healthcaredata['Insulin'])

plt.hist(healthcaredata['BMI'])

healthcaredata['Glucose'].value\_counts()

healthcaredata['BloodPressure'].value\_counts()

healthcaredata['SkinThickness'].value\_counts()

healthcaredata['Insulin'].value\_counts()

healthcaredata['BMI'].value\_counts()

healthcaredata.describe().transpose()

**#Project Task: Week 2 Data Exploration:**

**1.Check the balance of the data by plotting the count of outcomes by their value.**

**#Describe your findings and plan future course of action.**

plt.hist(Positive['BMI'],histtype='stepfilled',bins=20)

plt.hist(Positive['Glucose'],histtype='stepfilled',bins=20)

Positive['Glucose'].value\_counts()

plt.hist(Positive['BloodPressure'],histtype='stepfilled',bins=20)

plt.hist(Positive['SkinThickness'],histtype='stepfilled',bins=20)

plt.hist(Positive['Insulin'],histtype='stepfilled',bins=20)

Positive['Insulin'].value\_counts()

**#Create scatter charts between the pair of variables to understand the relationships. Describe your findings.**

BloodPressure = Positive['BloodPressure']

Glucose = Positive['Glucose']

SkinThickness = Positive['SkinThickness']

Insulin = Positive['Insulin']

BMI = Positive['BMI']

plt.scatter(BloodPressure, Glucose, color=['b'])

plt.xlabel('BloodPressure')

plt.ylabel('Glucose')

plt.title('BloodPressure & Glucose')

plt.show()

g =sns.scatterplot(x= "Glucose" ,y= "BloodPressure",

hue="Outcome",

data=healthcaredata);

B =sns.scatterplot(x= "BMI" ,y= "Insulin",

hue="Outcome",

data=healthcaredata);

S =sns.scatterplot(x= "SkinThickness" ,y= "Insulin",

hue="Outcome",

data=healthcaredata);

**#3.Perform correlation analysis. Visually explore it using a heat map.**

sns.heatmap(healthcaredata.corr())

plt.subplots(figsize=(8,8))

sns.heatmap(healthcaredata.corr(),annot=True,cmap='viridis')

plt.subplots(figsize=(8,8))

sns.heatmap(healthcaredata.corr(),annot=True)

**#Project Task: Week 3 Data Modeling:**

**#1. Devise strategies for model building. It is important to decide the right validation framework.**

**#Express your thought process.**

**#2. Apply an appropriate classification algorithm to build a model. Compare various models with the results from KNN algorithm.**

healthcaredata.head(5)

features = healthcaredata.iloc[:,[0,1,2,3,4,5,6,7]].values

label = healthcaredata.iloc[:,8].values

#Train test split

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

X\_train,X\_test,y\_train,y\_test = train\_test\_split(features,

label,

test\_size=0.2,

random\_state =10)

#Create model

from sklearn.linear\_model import LogisticRegression

model = LogisticRegression()

model.fit(X\_train,y\_train)

print(model.score(X\_train,y\_train))

print(model.score(X\_test,y\_test))

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(label,model.predict(features))

cm

#Applying Decission Tree Classifier

from sklearn.tree import DecisionTreeClassifier

model3 = DecisionTreeClassifier(max\_depth=5)

model3.fit(X\_train,y\_train)

model3.score(X\_train,y\_train)

model3.score(X\_test,y\_test)

#Applying Random Forest

from sklearn.ensemble import RandomForestClassifier

model4 = RandomForestClassifier(n\_estimators=11)

model4.fit(X\_train,y\_train)

model4.score(X\_train,y\_train)

model4.score(X\_test,y\_test)

#Support Vector Classifier

from sklearn.svm import SVC

model5 = SVC(kernel='rbf',

gamma='auto')

model5.fit(X\_train,y\_train)

model5.score(X\_test,y\_test)

#Applying K-NN

from sklearn.neighbors import KNeighborsClassifier

model2 = KNeighborsClassifier(n\_neighbors=7,

metric='minkowski',

p = 2)

model2.fit(X\_train,y\_train)

model2.score(X\_test,y\_test)

#Now we can compare the models with the results from KNN Algorithm by looking at the scores

# The KNN Algorithm has a score of 0.6948051948051948

# The Logistic Regression has a score of 0.7337662337662337

# The Decission Tree Classifier has a score of 0.8289902280130294

# The Random Forest has a score of 0.7272727272727273

# The Support Vector Classifier has a score of 0.6168831168831169

# From the results shown above we can say that the Decision Tree Classifier is the best model

**#Project Task: Week 4**

**#Data Modeling:**

**#1. Create a classification report by analyzing sensitivity, specificity, AUC (ROC curve), etc.**

**#Please be descriptive to explain what values of these parameter you have used.**

from sklearn.metrics import classification\_report

print(classification\_report(label,model.predict(features)))

#Preparing ROC Curve (Receiver Operating Characteristics Curve)

from sklearn.metrics import roc\_curve

from sklearn.metrics import roc\_auc\_score

# predict probabilities

probs = model.predict\_proba(features)

# keep probabilities for the positive outcome only

probs = probs[:, 1]

# calculate AUC

auc = roc\_auc\_score(label, probs)

print('AUC: %.3f' % auc)

# calculate roc curve

fpr, tpr, thresholds = roc\_curve(label, probs)

# plot no skill

plt.plot([0, 1], [0, 1], linestyle='--')

# plot the roc curve for the model

plt.plot(fpr, tpr, marker='.')

#Preparing ROC Curve (Receiver Operating Characteristics Curve)

from sklearn.metrics import roc\_curve

from sklearn.metrics import roc\_auc\_score

# predict probabilities

probs = model2.predict\_proba(features)

# keep probabilities for the positive outcome only

probs = probs[:, 1]

# calculate AUC

auc = roc\_auc\_score(label, probs)

print('AUC: %.3f' % auc)

# calculate roc curve

fpr, tpr, thresholds = roc\_curve(label, probs)

print("True Positive Rate - {}, False Positive Rate - {} Thresholds - {}".format(tpr,fpr,thresholds))

# plot no skill

plt.plot([0, 1], [0, 1], linestyle='--')

# plot the roc curve for the model

plt.plot(fpr, tpr, marker='.')

plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate")

#Precision Recall Curve for Logistic Regression

from sklearn.metrics import precision\_recall\_curve

from sklearn.metrics import f1\_score

from sklearn.metrics import auc

from sklearn.metrics import average\_precision\_score

# predict probabilities

probs = model.predict\_proba(features)

# keep probabilities for the positive outcome only

probs = probs[:, 1]

# predict class values

yhat = model.predict(features)

# calculate precision-recall curve

precision, recall, thresholds = precision\_recall\_curve(label, probs)

# calculate F1 score

f1 = f1\_score(label, yhat)

# calculate precision-recall AUC

auc = auc(recall, precision)

# calculate average precision score

ap = average\_precision\_score(label, probs)

print('f1=%.3f auc=%.3f ap=%.3f' % (f1, auc, ap))

# plot no skill

plt.plot([0, 1], [0.5, 0.5], linestyle='--')

# plot the precision-recall curve for the model

plt.plot(recall, precision, marker='.')

#Precision Recall Curve for KNN

from sklearn.metrics import precision\_recall\_curve

from sklearn.metrics import f1\_score

from sklearn.metrics import auc

from sklearn.metrics import average\_precision\_score

# predict probabilities

probs = model2.predict\_proba(features)

# keep probabilities for the positive outcome only

probs = probs[:, 1]

# predict class values

yhat = model2.predict(features)

# calculate precision-recall curve

precision, recall, thresholds = precision\_recall\_curve(label, probs)

# calculate F1 score

f1 = f1\_score(label, yhat)

# calculate precision-recall AUC

auc = auc(recall, precision)

# calculate average precision score

ap = average\_precision\_score(label, probs)

print('f1=%.3f auc=%.3f ap=%.3f' % (f1, auc, ap))

# plot no skill

plt.plot([0, 1], [0.5, 0.5], linestyle='--')

# plot the precision-recall curve for the model

plt.plot(recall, precision, marker='.')

#Precision Recall Curve for Decission Tree Classifier

from sklearn.metrics import precision\_recall\_curve

from sklearn.metrics import f1\_score

from sklearn.metrics import auc

from sklearn.metrics import average\_precision\_score

# predict probabilities

probs = model3.predict\_proba(features)

# keep probabilities for the positive outcome only

probs = probs[:, 1]

# predict class values

yhat = model3.predict(features)

# calculate precision-recall curve

precision, recall, thresholds = precision\_recall\_curve(label, probs)

# calculate F1 score

f1 = f1\_score(label, yhat)

# calculate precision-recall AUC

auc = auc(recall, precision)

# calculate average precision score

ap = average\_precision\_score(label, probs)

print('f1=%.3f auc=%.3f ap=%.3f' % (f1, auc, ap))

# plot no skill

plt.plot([0, 1], [0.5, 0.5], linestyle='--')

# plot the precision-recall curve for the model

plt.plot(recall, precision, marker='.')

#Precision Recall Curve for Random Forest

from sklearn.metrics import precision\_recall\_curve

from sklearn.metrics import f1\_score

from sklearn.metrics import auc

from sklearn.metrics import average\_precision\_score

# predict probabilities

probs = model4.predict\_proba(features)

# keep probabilities for the positive outcome only

probs = probs[:, 1]

# predict class values

yhat = model4.predict(features)

# calculate precision-recall curve

precision, recall, thresholds = precision\_recall\_curve(label, probs)

# calculate F1 score

f1 = f1\_score(label, yhat)

# calculate precision-recall AUC

auc = auc(recall, precision)

# calculate average precision score

ap = average\_precision\_score(label, probs)

print('f1=%.3f auc=%.3f ap=%.3f' % (f1, auc, ap))

# plot no skill

plt.plot([0, 1], [0.5, 0.5], linestyle='--')

# plot the precision-recall curve for the model

plt.plot(recall, precision, marker='.')