%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

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

data.head()

data.isnull().any()

data.info()

Positive = data[data['Outcome']==1]

Positive.head(5)

data['Glucose'].value\_counts().head(7)

plt.hist(data['Glucose'])

data['BloodPressure'].value\_counts().head(7)

plt.hist(data['BloodPressure'])

data['SkinThickness'].value\_counts().head(7)

plt.hist(data['SkinThickness'])

data['Insulin'].value\_counts().head(7)

plt.hist(data['Insulin'])

data['BMI'].value\_counts().head(7)

plt.hist(data['BMI'])

data.describe().transpose()

# Week 2

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

Positive['BMI'].value\_counts().head(7)

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

Positive['Glucose'].value\_counts().head(7)

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

Positive['BloodPressure'].value\_counts().head(7)

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

Positive['SkinThickness'].value\_counts().head(7)

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

Positive['Insulin'].value\_counts().head(7)

#Scatter plot

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=data);

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

hue="Outcome",

data=data);

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

hue="Outcome",

data=data);

### correlation matrix

data.corr()

### create correlation heat map

sns.heatmap(data.corr())

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

sns.heatmap(data.corr(),annot=True,cmap='viridis') ### gives correlation value

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

sns.heatmap(data.corr(),annot=True) ### gives correlation value

# Logistic Regreation and model building

data.head(5)

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

label = data.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

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='.')

#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)

model5model.score(X\_test,y\_test).score(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)

#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='.')