COMP 4254 – ASSIGNMENT 2

Jason Chow

A01069757

Introduction

The goal of this report is to describe logistic regression models that can predict if surveyors do not have diabetes or have either prediabetes or diabetes. The models are based on a clean dataset of 70,692 survey responses to the Centre of Disease and Control from 2015. It has an equal 50-50 split of respondents with no diabetes and with either prediabetes or diabetes. The target variable Diabetes_binary has 2 classes. 0 is for no diabetes, and 1 is for prediabetes or diabetes (prediabetes or diabetes will be referred to as diabetes in the rest of the report).

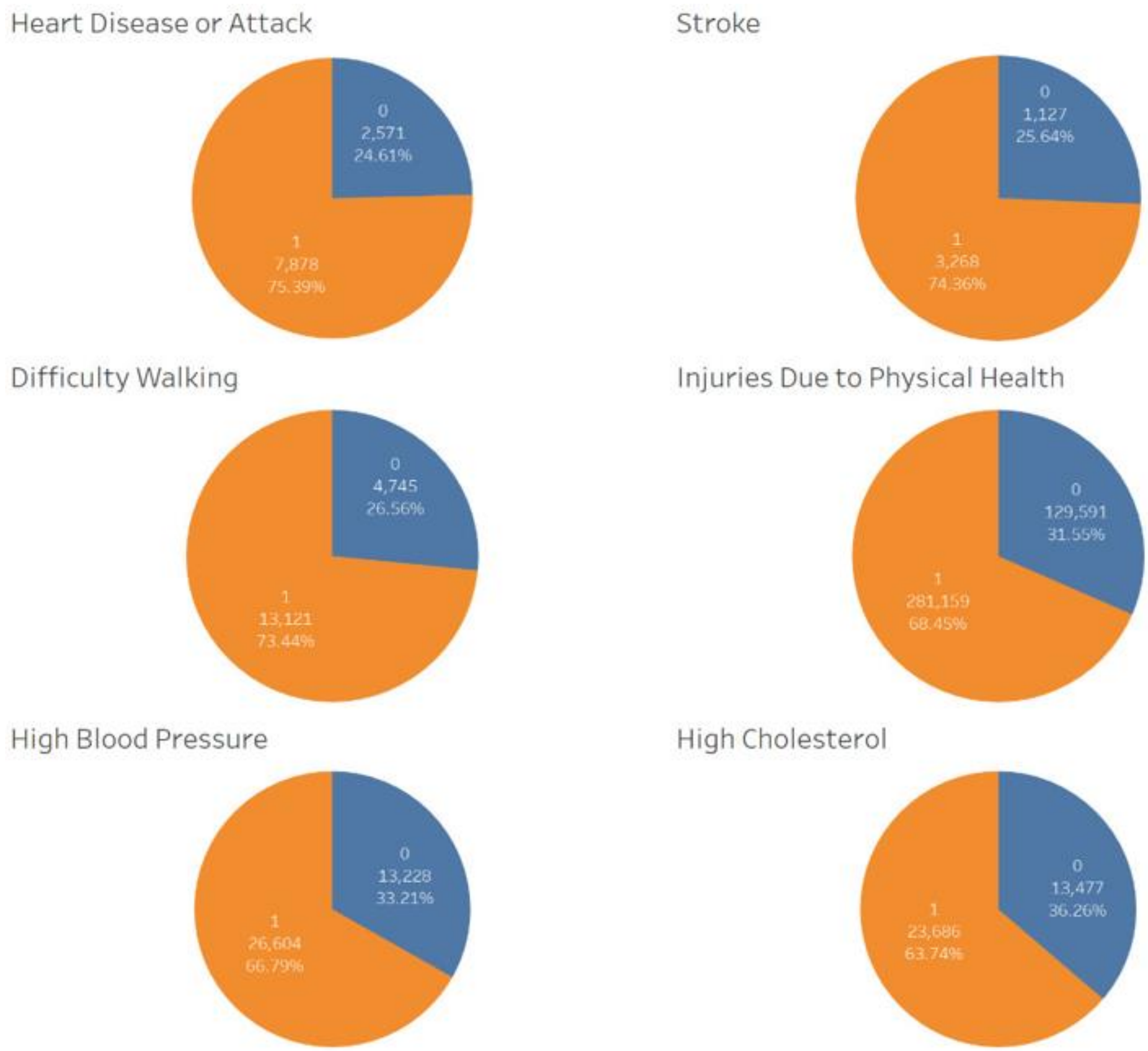
Data Exploration

Tableau is used to visualize the dataset. The following figure shows the feature variables (listed in absolute numbers and in percentages) that falls under class 0 or 1.

	Diabetes binary			
	0	1	0	1
Age	275,312	331,512	45.37%	54.63%
Any Healthcare	33,584	33,924	49.75%	50.25%
BMI	981,557	1,129,093	46.50%	53.50%
Chol Check	33,838	35,105	49.08%	50.92%
Diff Walk	4,745	13,121	26.56%	73.44%
Education	180,137	167,735	51.78%	48.22%
Fruits	22,556	20,693	52.15%	47.85%
Gen Hlth	84,236	116,323	42.00%	58.00%
Heart Diseaseor Attack	2,571	7,878	24.61%	75.39%
High BP	13,228	26,604	33.21%	66.79%
High Chol	13,477	23,686	36.26%	63.74%
Hvy Alcohol Consump	2,188	832	72.45%	27.55%
Income	218,669	184,156	54.28%	45.72%
Ment Hlth	107,532	157,707	40.54%	59.46%
No Docbc Cost	2,897	3,742	43.64%	56.36%
Phys Activity	27,412	22,287	55.16%	44.84%
Phys Hlth	129,591	281,159	31.55%	68.45%
Sex	15,371	16,935	47.58%	52.42%
Smoker	15,281	18,317	45.48%	54.52%
Stroke	1,127	3,268	25.64%	74.36%
Veggies	29,024	26,736	52.05%	47.95%

This figure highlights which feature is typically highly associated with having diabetes and which feature variables are least typically associated with having diabetes. Features that have a high percentage that fall under class 1 is highly associated with having diabetes. Features that have a high percentage that fall under class 0 is least associated with having diabetes.

The following figure shows which features are typically highly associated with having diabetes.



This figure shows visually which features are found in the highest percentage in diabetics compared to non-diabetics. The lay person would commonly associate these features to diabetes.

The following figure shows which features are typically least associated with having diabetes.

This figure shows visually which features are found in the least percentages in diabetics compared to non-diabetics. The lay person would not commonly associate these features to diabetes. The only surprising result is for heavy alcohol use. Most people would not associate heavy alcohol use to lower chances of having diabetes.

Identifying Significant Features

The significant features are shown in the pie charts previously shown. These features are described in detail below:

HeartDiseaseorAttack - coronary heart disease (CHD) or myocardial infarction (MI). 0 = no. 1 = yes.

Stroke - (Ever told) you had a stroke. 0 = no. 1 = yes.

DiffWalk - Do you have serious difficulty walking or climbing stairs? 0 = no. 1 = yes.

PhysHlth – How many physical illness or injury days have you experienced in the past 30 days? The results are 1-30 days.

HighBP – Do you have high blood pressure? 0 = no. 1 = yes.

HighChol – Do you have high cholesterol? 0 = no. 1 = yes.

HvyAlcoholConsump – Do you consume alcohol greater or equal to 14 drinks per week (for adult men) or greater or equal to 7 drinks per week (for adult women). 0 = no. 1 = yes.

PhysActivity – Have you done physical activity in past 30 days, not including job. 0 = no. 1 = yes.

Income – Personal income from a scale of 1 to 8. 1 = less than $10,000. 5 = less than $35,000. 8 = $75,000 or more.

Fruits – Do you consume fruit more than 1 or more times per day. 0 = no. 1 = yes.

Veggies - Do you consume veggies more than 1 or more times per day. 0 = no. 1 = yes.

Education – Education level from a scale of 1 to 6. 1 = never attended school or only kindergarten. 6 = college 4 years or more (College graduate).

Imputing and Variable Creation

The dataset is very clean. The only features that were created is related to BMI. The BMI results are originally listed in absolute numbers. These results are binned according to the categorizes that the World Health Organization provides. These categorizes are Underweight < 18.5, Normal Range = 18 .5 - 24.9, Overweight = 25.0 - 29.9, Obese Class 1 >= 30.0 – 34.9, Obese Class 2 >= 35.0 - 39.9, Obese Class >= 40.

Data Modelling

Four models are created. The first, second, and third model are a logistic regression model that predicts using cross fold validation. Cross fold validation is splitting data into random test and training set over several runs to produce more accurate results. Each model is created identically except that the first model includes all features that are significant, according to the chi test, the second model list features that are typically highly associated with having diabetes, and the third model list features that are typically least associated with having diabetes. The fourth model is a logistic regression model that predicts using stacking. Stacking is the procedure of building a model with the output from multiple models. It includes all features that are significant, according to the chi test.

Model Evaluation

The accuracy, precision, recall, f1 scores, and ROC plot for all models are listed below.

```
Accuracy and Standard Deviation For All Folds:
********************************************
Average Accuracy:
0.7497172970677171
Accuracy SD:
0.004982024806612728
Average Precision:
0.7391629664828998
Precision SD:
0.005614349890527581
Average Recall:
0.7718258903096247
Recall SD:
0.004018384909742144
Average F1:
0.7551353687967807
F1 SD:
0.004458060416573516
AUC scores
0.81777053519882 0.7451742464642558
```

Model 1 Results

```
Accuracy and Standard Deviation For All Folds:
*********************************************
Average Accuracy:
0.7068552064084233
Accuracy SD:
0.003066174325343591
Average Precision:
0.7121365909645712
Precision SD:
0.007697546514127863
Average Recall:
0.6944817504914873
Recall SD:
0.004830260793146434
Average F1:
0.7031633813994118
F1 SD:
0.00402807823226244
AUC scores
0.7677557908882348 0.7207889763391301
```

Model 2 Results

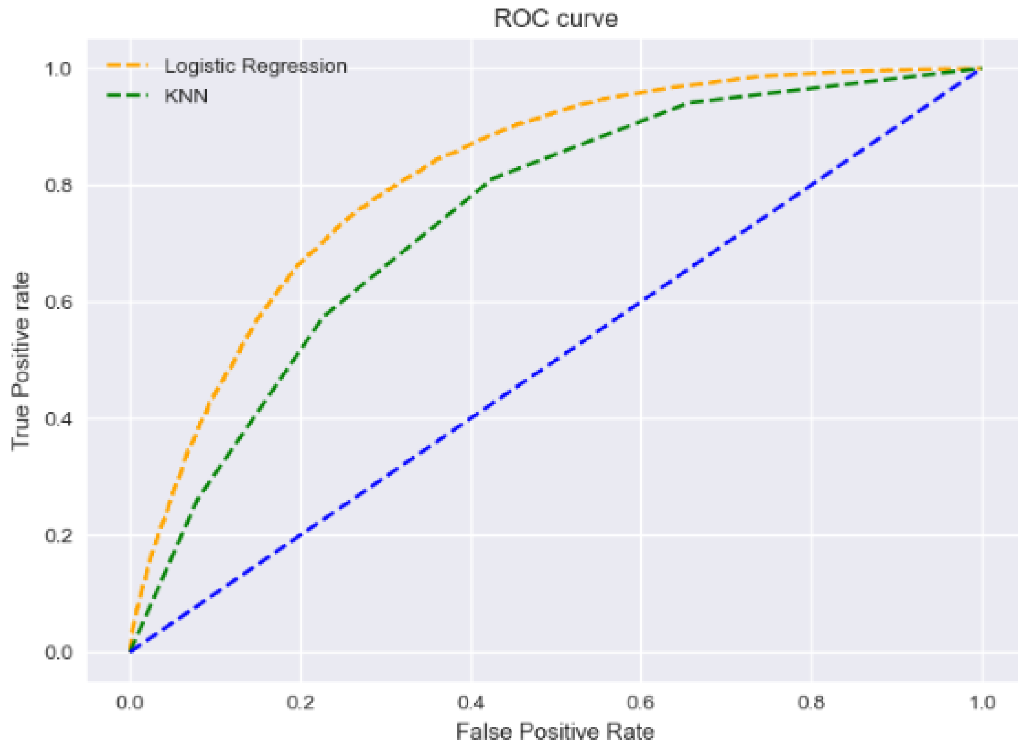
```
Accuracy and Standard Deviation For All Folds:
*******************************************
Average Accuracy:
0.6172409046099385
Accuracy SD:
0.0034113112162165913
Average Precision:
0.6267903157258626
Precision SD:
0.004118845128004356
Average Recall:
0.5795100404938446
Recall SD:
0.007221740620551153
Average F1:
0.6022091008110751
F1 SD:
0.005300966518584348
AUC scores
0.6634776703423656 0.550629449284036
```

Model 3 Results

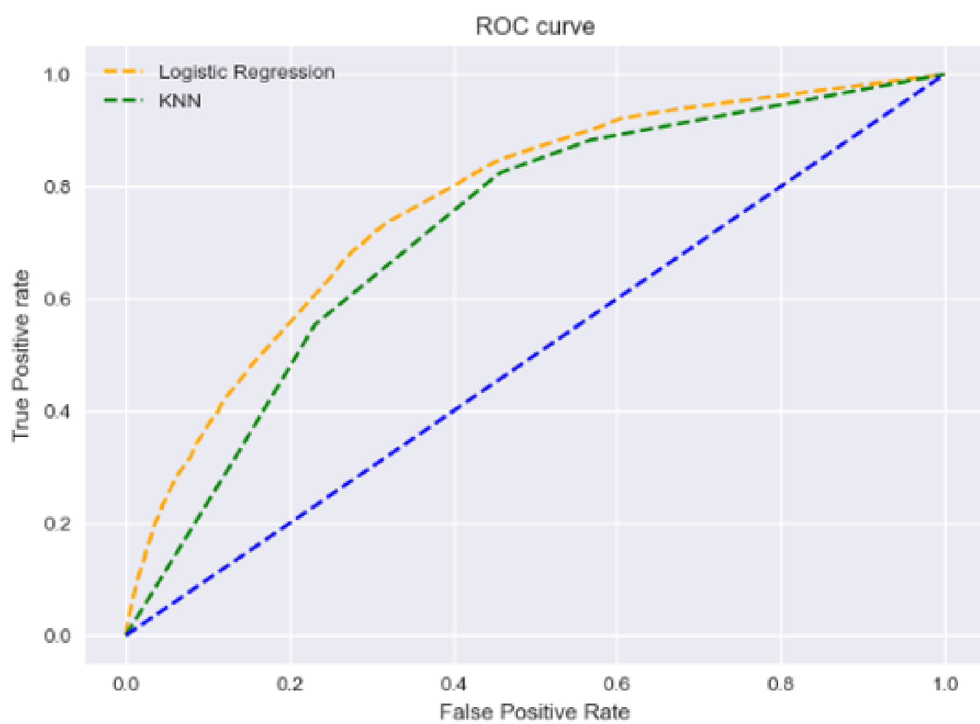
```
** Evaluate Base Models **
Precision:0.75 Recall:0.76 F1:0.75 Accuracy:0.75   LogisticRegression
Precision:0.66 Recall:0.65 F1:0.65 Accuracy:0.66   DecisionTreeClassifier
Precision:0.75 Recall:0.77 F1:0.76 Accuracy:0.75   AdaBoostClassifier
Precision:0.73 Recall:0.68 F1:0.71 Accuracy:0.71   RandomForestClassifier

** Evaluate Stacked Model **
Precision:0.75 Recall:0.76 F1:0.76 Accuracy:0.75   LogisticRegression
AUC scores
0.817909396207008 0.7453792074111758
```

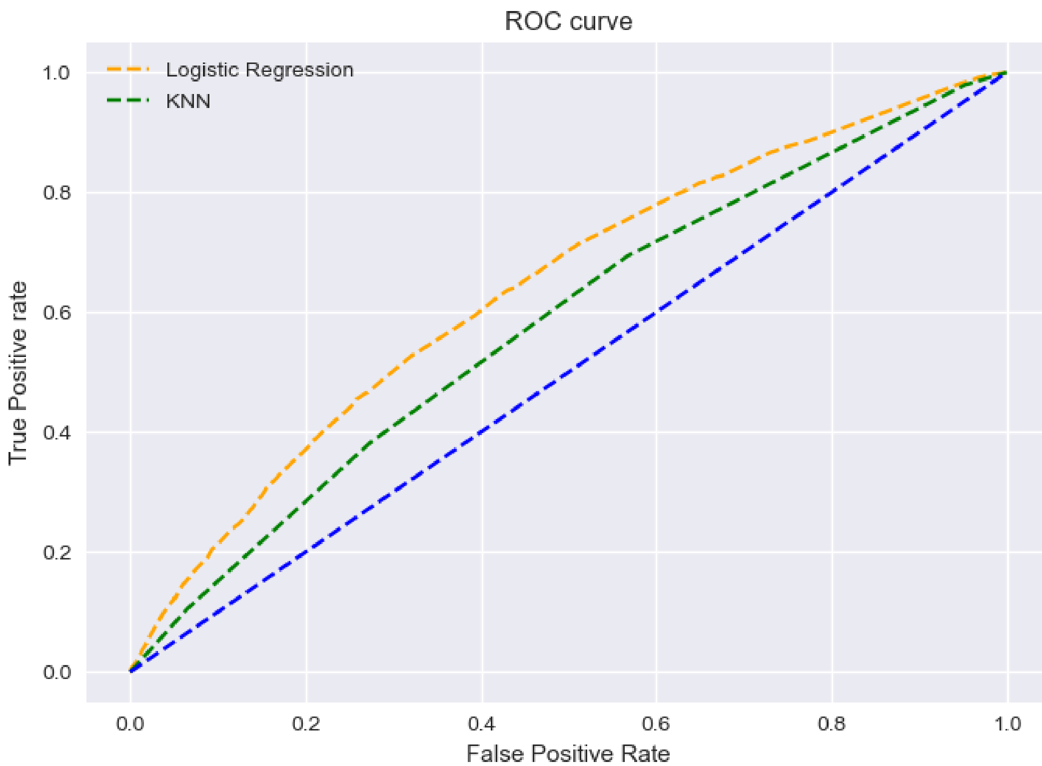
Model 4 Results



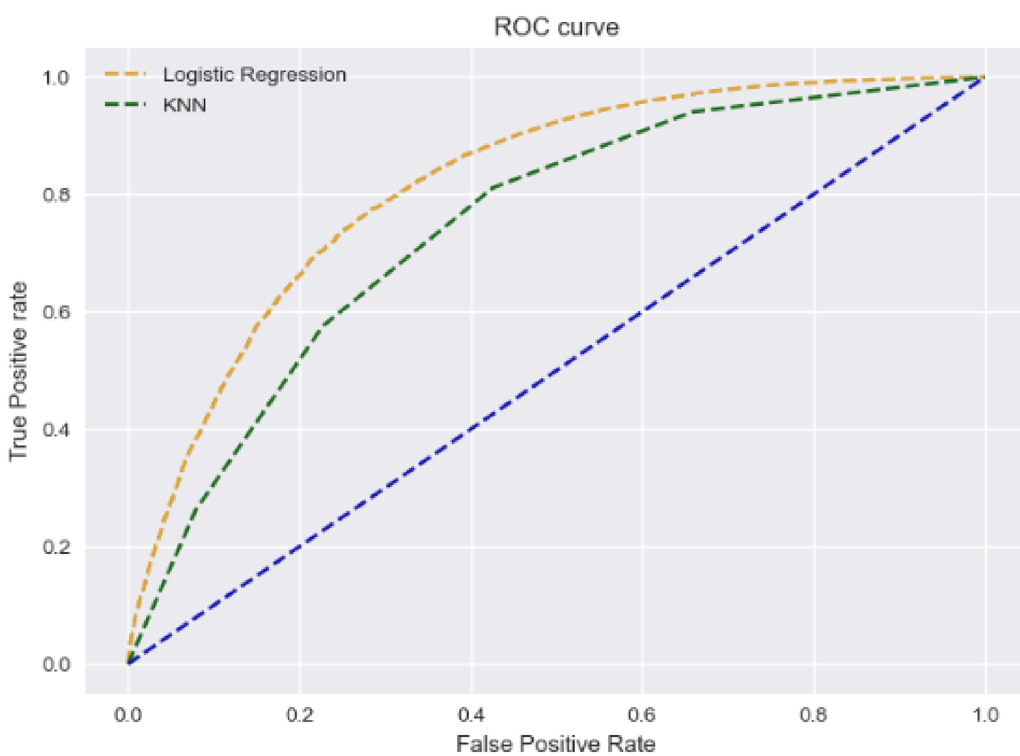
Model 1 ROC



Model 2 ROC



Model 3 ROC



Model 4 ROC

The accuracy percentage shows how accurately the model predicts. The precision percentage shows how relevant are the model's results. The recall percentage show how many of the correct prediction are found. The F-1 score reflects the model's overall accuracy. The results from Models 1 and 4 are nearly identical and outperforms the others. This result is not surprising as the more significant factors, there are in a model, the more accurate it is. Model 2 outperforms Model 3. This result is due to Model 2's features are more statistically significant to the dependant variable that Model 3's features.

The scores overall for each model is above average. The model performs well as it uses very clean data and the data is balanced. Furthermore, the results for Model 1, 2, 3 performs cross fold validation with averages 8 outputs and Model 4 performs stacking, which builds a model with the output from multiple models. The more outputs that are averaged, the more accurate the results.

Similarly, the ROC curves from Model 1 and 4 are nearly identical and outperforms the others. Model 2's ROC curve outperforms Model 3's. ROC curves are frequently used to show in a graphical way the connection/trade-off between the true positive and false positive rates of a model using different probability thresholds. The further left the logistic regression and KNN lines are from the middle line, the more likely the model predicts correctly. The AUC score shows the model's ability to distinguish between classes. It is a summary of the ROC curve. Each ROC curve and AUC score shows that its respective model predicts accurately except for Model 3's. Model's 3 logistic regression and KNN are fairly close to the middle line which shows the model only slightly predicts better than a 50/50 guess.

The advantages of using Model 1 and 4 are they are the most accurate models. The advantage of using model 2 is that only 6 variables are needed to produce a relatively reliable model. There is no advantage of using model 3.

Conclusion

The results from Model 1 and 4 are nearly identical. Just slightly, the AUC scores from the ROC curves are higher in Model 4. Furthermore, the coding necessary to create Model 4 is slightly quicker to produce for most. Therefore, Model 4 is the top model. This model can be improved if more models are averaged to produce the output.

Appendix

Logistic Regression Model Python Script for Model 1

```
import pandas                  as pd
from sklearn import preprocessing
from sklearn.linear_model import LogisticRegression
import numpy as np
from sklearn import metrics


PATH     = "C:\\datasets\\"                # Windows
CSV_DATA = "diabetes.csv"
df       = pd.read_csv(PATH + CSV_DATA, sep=',')

# Show all columns.
pd.set_option('display.max_columns', None)
pd.set_option('display.width', 1000)

print(df.describe())

print("Value counts: " + str(df['Diabetes_binary'].value_counts()))
le = preprocessing.LabelEncoder()

df['y'] = le.fit_transform(df['Diabetes_binary'])
print(df.tail())
y = df[['y']]


predictorVariables = list(df.keys())
predictorVariables.remove('Diabetes_binary')
predictorVariables.remove('y')

X = df[predictorVariables]
X = X.copy()

y = df['y']


X['bmiBin']  = pd.cut(x=X['BMI'], bins=[0, 18.49, 24.9, 29.9, 34.9, 39.9,
99])

tempDf  = X[['bmiBin']]                 # Isolate columns
dummyDf = pd.get_dummies(tempDf, columns=['bmiBin'])
X      = pd.concat(([X, dummyDf]), axis=1)      # Join dummy df with original
del X['bmiBin']

X.rename(columns={'bmiBin_(0.0, 18.49]':'BMI - Underweight','bmiBin_(18.49,
24.9]':'BMI - Normal', 'bmiBin_(24.9, 29.9]':'BMI - Overweight',
'bmiBin_(29.9, 34.9]':'BMI - Obese Class 1', 'bmiBin_(34.9, 39.9]':'BMI -
Obese Class 2', 'bmiBin_(39.9, 99.0]':'BMI - Obese Class 3' }, inplace=True)

print(X.head())
```

```python
from sklearn.preprocessing import MinMaxScaler
sc_x    = MinMaxScaler()
X_Scale = sc_x.fit_transform(X)


from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

test      = SelectKBest(score_func=chi2, k=26)
chiScores = test.fit(X, y) # Summarize scores
np.set_printoptions(precision=3)

print("\nPredictor variables: " + str(list(X.keys())))
print("Predictor Chi-Square Scores: " + str(chiScores.scores_))


cols = chiScores.get_support(indices=True)
print("\nSignificant columns after chi-square test")
print(cols)
features = X.columns[cols]
print("Significant column names after chi-square test")
print(np.array(features))


from   sklearn.model_selection import train_test_split
from   sklearn.linear_model    import LogisticRegression

X = X[['HighBP', 'HighChol', 'CholCheck', 'BMI', 'Smoker', 'Stroke',
 'HeartDiseaseorAttack', 'PhysActivity', 'Fruits', 'Veggies',
 'HvyAlcoholConsump', 'NoDocbcCost', 'GenHlth', 'MentHlth',
 'PhysHlth', 'DiffWalk', 'Sex', 'Age', 'Education', 'Income',
 'BMI - Underweight', 'BMI - Normal', 'BMI - Overweight',
 'BMI - Obese Class 1', 'BMI - Obese Class 2', 'BMI - Obese Class 3']]

X_Scale = sc_x.fit_transform(X)

# Split data.
X_train,X_test,y_train,y_test = train_test_split(X_Scale, y, test_size=0.25,
                                                 random_state=0)

# Build logistic regression model and make predictions.
logisticModel = LogisticRegression(fit_intercept=True, solver='liblinear',
                                   random_state=0)
logisticModel.fit(X_train,y_train)
y_pred=logisticModel.predict(X_test)
print("\nPredictions from logistic model")
print(y_pred)


# enumerate splits - returns train and test arrays of indexes.
# scikit-learn k-fold cross-validation
from sklearn.model_selection import KFold
#kfold = KFold(3, True)
kfold = KFold(n_splits=8, shuffle=True)
count = 0
```

```python
accuracyList = []
precisionList = []
recallList = []
f1List = []

for train_index, test_index in kfold.split(X_Scale):

    X_train, X_test = X_Scale[train_index], X_Scale[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Perform logistic regression.
    logisticModel = LogisticRegression(fit_intercept=True,
                                       solver='liblinear')
    # Fit the model.
    logisticModel.fit(X_train, np.ravel(y_train))

    y_pred = logisticModel.predict(X_test)
    y_prob = logisticModel.predict_proba(X_test)

    # Show confusion matrix and accuracy scores.
    cm = pd.crosstab(y_test, y_pred,
                     rownames=['Actual'],
                     colnames=['Predicted'])
    count += 1
    print("\n***K-fold: " + str(count))

    # Calculate accuracy and precision scores and add to the list.
    accuracy = metrics.accuracy_score(y_test, y_pred)
    precision = metrics.precision_score(y_test, y_pred)
    recall = metrics.recall_score(y_test, y_pred)
    f1 = metrics.f1_score(y_test, y_pred)

    print('\nAccuracy: ', accuracy)
    accuracyList.append(accuracy)
    print("\nPrecision: ", precision)
    precisionList.append(precision)
    print("\nRecall: ", recall)
    recallList.append(recall)
    print("\nF1: ", f1)
    f1List.append(f1)
    print("\nConfusion Matrix")
    print(cm)

print("\nAccuracy and Standard Deviation For All Folds:")
print("*********************************************")
print("Average Accuracy: ")
print(np.mean(accuracyList))
print("Accuracy SD: ")
print(np.std(accuracyList))
print("Average Precision: ")
print(np.mean(precisionList))
print("Precision SD: ")
print(np.std(precisionList))
print("Average Recall: ")
print(np.mean(recallList))
print("Recall SD: ")
print(np.std(recallList))
```

```
print("Average F1: ")
print(np.mean(f1List))
print("F1 SD: ")
print(np.std(f1List))


from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split


# split into train-test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=27)

# train models
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier

# logistic regression
model1 = LogisticRegression()
# knn
model2 = KNeighborsClassifier(n_neighbors=4)

# fit model
model1.fit(X_train, y_train)
model2.fit(X_train, y_train)

# predict probabilities
pred_prob1 = model1.predict_proba(X_test)
pred_prob2 = model2.predict_proba(X_test)

from sklearn.metrics import roc_curve

# roc curve for models
fpr1, tpr1, thresh1 = roc_curve(y_test, pred_prob1[:,1], pos_label=1)
fpr2, tpr2, thresh2 = roc_curve(y_test, pred_prob2[:,1], pos_label=1)

# roc curve for tpr = fpr
random_probs = [0 for i in range(len(y_test))]
p_fpr, p_tpr, _ = roc_curve(y_test, random_probs, pos_label=1)

# Compute AUC score.
from sklearn.metrics import roc_auc_score

# auc scores
auc_score1 = roc_auc_score(y_test, pred_prob1[:,1])
auc_score2 = roc_auc_score(y_test, pred_prob2[:,1])

print("AUC scores")
print(auc_score1, auc_score2)

# matplotlib
import matplotlib.pyplot as plt
plt.style.use('seaborn')

# plot roc curves
plt.plot(fpr1, tpr1, linestyle='--',color='orange', label='Logistic
```

```
Regression')
plt.plot(fpr2, tpr2, linestyle='--',color='green', label='KNN')
plt.plot(p_fpr, p_tpr, linestyle='--', color='blue')
# title
plt.title('ROC curve')
# x label
plt.xlabel('False Positive Rate')
# y label
plt.ylabel('True Positive rate')

plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show();
```

Logistic Regression Model Python Script for Model 2

```
import pandas                  as pd
from sklearn import preprocessing
from sklearn.linear_model import LogisticRegression
import numpy as np
from sklearn import metrics


PATH     = "C:\\datasets\\"                # Windows
CSV_DATA = "diabetes.csv"
df       = pd.read_csv(PATH + CSV_DATA, sep=',')

# Show all columns.
pd.set_option('display.max_columns', None)
pd.set_option('display.width', 1000)

print(df.describe())

print("Value counts: " + str(df['Diabetes_binary'].value_counts()))
le = preprocessing.LabelEncoder()

df['y'] = le.fit_transform(df['Diabetes_binary'])
print(df.tail())
y = df[['y']]


predictorVariables = list(df.keys())
predictorVariables.remove('Diabetes_binary')
predictorVariables.remove('y')

X = df[predictorVariables]
X = X.copy()

y = df['y']


X['bmiBin']  = pd.cut(x=X['BMI'], bins=[0, 18.49, 24.9, 29.9, 34.9, 39.9,
99])

tempDf  = X[['bmiBin']]                  # Isolate columns
```

```python
dummyDf = pd.get_dummies(tempDf, columns=['bmiBin'])
X       = pd.concat(([X, dummyDf]), axis=1)      # Join dummy df with original
del X['bmiBin']

X.rename(columns={'bmiBin_(0.0, 18.49]':'BMI - Underweight','bmiBin_(18.49, 24.9]':'BMI - Normal', 'bmiBin_(24.9, 29.9]':'BMI - Overweight', 'bmiBin_(29.9, 34.9]':'BMI - Obese Class 1', 'bmiBin_(34.9, 39.9]':'BMI - Obese Class 2', 'bmiBin_(39.9, 99.0]':'BMI - Obese Class 3' }, inplace=True)

print(X.head())


from sklearn.preprocessing import MinMaxScaler
sc_x    = MinMaxScaler()
X_Scale = sc_x.fit_transform(X)


from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

test      = SelectKBest(score_func=chi2, k=26)
chiScores = test.fit(X, y) # Summarize scores
np.set_printoptions(precision=3)

print("\nPredictor variables: " + str(list(X.keys())))
print("Predictor Chi-Square Scores: " + str(chiScores.scores_))


cols = chiScores.get_support(indices=True)
print("\nSignificant columns after chi-square test")
print(cols)
features = X.columns[cols]
print("Significant column names after chi-square test")
print(np.array(features))


from   sklearn.model_selection import train_test_split
from   sklearn.linear_model    import LogisticRegression

X = X[['HighBP', 'HighChol', 'Stroke',
 'HeartDiseaseorAttack', 'PhysHlth', 'DiffWalk']]

X_Scale = sc_x.fit_transform(X)

# Split data.
X_train,X_test,y_train,y_test = train_test_split(X_Scale, y, test_size=0.25,
                                                 random_state=0)

# Build logistic regression model and make predictions.
logisticModel = LogisticRegression(fit_intercept=True, solver='liblinear',
                                   random_state=0)
logisticModel.fit(X_train,y_train)
y_pred=logisticModel.predict(X_test)
print("\nPredictions from logistic model")
print(y_pred)
```

```python
# enumerate splits - returns train and test arrays of indexes.
# scikit-learn k-fold cross-validation
from sklearn.model_selection import KFold
#kfold = KFold(3, True)
kfold = KFold(n_splits=8, shuffle=True)
count = 0

accuracyList = []
precisionList = []
recallList = []
f1List = []

for train_index, test_index in kfold.split(X_Scale):

    X_train, X_test = X_Scale[train_index], X_Scale[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Perform logistic regression.
    logisticModel = LogisticRegression(fit_intercept=True,
                                       solver='liblinear')
    # Fit the model.
    logisticModel.fit(X_train, np.ravel(y_train))

    y_pred = logisticModel.predict(X_test)
    y_prob = logisticModel.predict_proba(X_test)

    # Show confusion matrix and accuracy scores.
    cm = pd.crosstab(y_test, y_pred,
                     rownames=['Actual'],
                     colnames=['Predicted'])
    count += 1
    print("\n***K-fold: " + str(count))

    # Calculate accuracy and precision scores and add to the list.
    accuracy = metrics.accuracy_score(y_test, y_pred)
    precision = metrics.precision_score(y_test, y_pred)
    recall = metrics.recall_score(y_test, y_pred)
    f1 = metrics.f1_score(y_test, y_pred)

    print('\nAccuracy: ', accuracy)
    accuracyList.append(accuracy)
    print("\nPrecision: ", precision)
    precisionList.append(precision)
    print("\nRecall: ", recall)
    recallList.append(recall)
    print("\nF1: ", f1)
    f1List.append(f1)
    print("\nConfusion Matrix")
    print(cm)

print("\nAccuracy and Standard Deviation For All Folds:")
print("*********************************************")
print("Average Accuracy: ")
print(np.mean(accuracyList))
print("Accuracy SD: ")
print(np.std(accuracyList))
print("Average Precision: ")
```

```python
print(np.mean(precisionList))
print("Precision SD: ")
print(np.std(precisionList))
print("Average Recall: ")
print(np.mean(recallList))
print("Recall SD: ")
print(np.std(recallList))
print("Average F1: ")
print(np.mean(f1List))
print("F1 SD: ")
print(np.std(f1List))


from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split


# split into train-test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=27)

# train models
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier

# logistic regression
model1 = LogisticRegression()
# knn
model2 = KNeighborsClassifier(n_neighbors=4)

# fit model
model1.fit(X_train, y_train)
model2.fit(X_train, y_train)

# predict probabilities
pred_prob1 = model1.predict_proba(X_test)
pred_prob2 = model2.predict_proba(X_test)

from sklearn.metrics import roc_curve

# roc curve for models
fpr1, tpr1, thresh1 = roc_curve(y_test, pred_prob1[:,1], pos_label=1)
fpr2, tpr2, thresh2 = roc_curve(y_test, pred_prob2[:,1], pos_label=1)

# roc curve for tpr = fpr
random_probs = [0 for i in range(len(y_test))]
p_fpr, p_tpr, _ = roc_curve(y_test, random_probs, pos_label=1)

# Compute AUC score.
from sklearn.metrics import roc_auc_score

# auc scores
auc_score1 = roc_auc_score(y_test, pred_prob1[:,1])
auc_score2 = roc_auc_score(y_test, pred_prob2[:,1])

print("AUC scores")
print(auc_score1, auc_score2)
```

```
# matplotlib
import matplotlib.pyplot as plt
plt.style.use('seaborn')

# plot roc curves
plt.plot(fpr1, tpr1, linestyle='--',color='orange', label='Logistic
Regression')
plt.plot(fpr2, tpr2, linestyle='--',color='green', label='KNN')
plt.plot(p_fpr, p_tpr, linestyle='--', color='blue')
# title
plt.title('ROC curve')
# x label
plt.xlabel('False Positive Rate')
# y label
plt.ylabel('True Positive rate')

plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show();
```

Logistic Regression Model Python Script for Model 3

```
import pandas                  as pd
from sklearn import preprocessing
from sklearn.linear_model import LogisticRegression
import numpy as np
from sklearn import metrics


PATH     = "C:\\datasets\\"                # Windows
CSV_DATA = "diabetes.csv"
df       = pd.read_csv(PATH + CSV_DATA, sep=',')

# Show all columns.
pd.set_option('display.max_columns', None)
pd.set_option('display.width', 1000)

print(df.describe())

print("Value counts: " + str(df['Diabetes_binary'].value_counts()))
le = preprocessing.LabelEncoder()

df['y'] = le.fit_transform(df['Diabetes_binary'])
print(df.tail())
y = df[['y']]


predictorVariables = list(df.keys())
predictorVariables.remove('Diabetes_binary')
predictorVariables.remove('y')

X = df[predictorVariables]
X = X.copy()
```

```
y = df['y']


X['bmiBin']  = pd.cut(x=X['BMI'], bins=[0, 18.49, 24.9, 29.9, 34.9, 39.9,
99])

tempDf  = X[['bmiBin']]                  # Isolate columns
dummyDf = pd.get_dummies(tempDf, columns=['bmiBin'])
X       = pd.concat(([X, dummyDf]), axis=1)      # Join dummy df with original
del X['bmiBin']

X.rename(columns={'bmiBin_(0.0, 18.49]':'BMI - Underweight','bmiBin_(18.49,
24.9]':'BMI - Normal', 'bmiBin_(24.9, 29.9]':'BMI - Overweight',
'bmiBin_(29.9, 34.9]':'BMI - Obese Class 1', 'bmiBin_(34.9, 39.9]':'BMI -
Obese Class 2', 'bmiBin_(39.9, 99.0]':'BMI - Obese Class 3' }, inplace=True)

print(X.head())


from sklearn.preprocessing import MinMaxScaler
sc_x    = MinMaxScaler()
X_Scale = sc_x.fit_transform(X)


from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

test      = SelectKBest(score_func=chi2, k=26)
chiScores = test.fit(X, y) # Summarize scores
np.set_printoptions(precision=3)

print("\nPredictor variables: " + str(list(X.keys())))
print("Predictor Chi-Square Scores: " + str(chiScores.scores_))


cols = chiScores.get_support(indices=True)
print("\nSignificant columns after chi-square test")
print(cols)
features = X.columns[cols]
print("Significant column names after chi-square test")
print(np.array(features))


from   sklearn.model_selection import train_test_split
from   sklearn.linear_model    import LogisticRegression

X = X[['PhysActivity', 'Fruits', 'Veggies',
 'HvyAlcoholConsump', 'Education', 'Income']]

X_Scale = sc_x.fit_transform(X)

# Split data.
X_train,X_test,y_train,y_test = train_test_split(X_Scale, y, test_size=0.25,
                                                 random_state=0)

# Build logistic regression model and make predictions.
logisticModel = LogisticRegression(fit_intercept=True, solver='liblinear',
```

```
                                    random_state=0)
logisticModel.fit(X_train,y_train)
y_pred=logisticModel.predict(X_test)
print("\nPredictions from logistic model")
print(y_pred)


# enumerate splits - returns train and test arrays of indexes.
# scikit-learn k-fold cross-validation
from sklearn.model_selection import KFold
#kfold = KFold(3, True)
kfold = KFold(n_splits=8, shuffle=True)
count = 0

accuracyList = []
precisionList = []
recallList = []
f1List = []

for train_index, test_index in kfold.split(X_Scale):

    X_train, X_test = X_Scale[train_index], X_Scale[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Perform logistic regression.
    logisticModel = LogisticRegression(fit_intercept=True,
                                       solver='liblinear')
    # Fit the model.
    logisticModel.fit(X_train, np.ravel(y_train))

    y_pred = logisticModel.predict(X_test)
    y_prob = logisticModel.predict_proba(X_test)

    # Show confusion matrix and accuracy scores.
    cm = pd.crosstab(y_test, y_pred,
                     rownames=['Actual'],
                     colnames=['Predicted'])
    count += 1
    print("\n***K-fold: " + str(count))

    # Calculate accuracy and precision scores and add to the list.
    accuracy = metrics.accuracy_score(y_test, y_pred)
    precision = metrics.precision_score(y_test, y_pred)
    recall = metrics.recall_score(y_test, y_pred)
    f1 = metrics.f1_score(y_test, y_pred)

    print('\nAccuracy: ', accuracy)
    accuracyList.append(accuracy)
    print("\nPrecision: ", precision)
    precisionList.append(precision)
    print("\nRecall: ", recall)
    recallList.append(recall)
    print("\nF1: ", f1)
    f1List.append(f1)
    print("\nConfusion Matrix")
    print(cm)
```

```
print("\nAccuracy and Standard Deviation For All Folds:")
print("**********************************************")
print("Average Accuracy: ")
print(np.mean(accuracyList))
print("Accuracy SD: ")
print(np.std(accuracyList))
print("Average Precision: ")
print(np.mean(precisionList))
print("Precision SD: ")
print(np.std(precisionList))
print("Average Recall: ")
print(np.mean(recallList))
print("Recall SD: ")
print(np.std(recallList))
print("Average F1: ")
print(np.mean(f1List))
print("F1 SD: ")
print(np.std(f1List))


from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split


# split into train-test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=27)

# train models
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier

# logistic regression
model1 = LogisticRegression()
# knn
model2 = KNeighborsClassifier(n_neighbors=4)

# fit model
model1.fit(X_train, y_train)
model2.fit(X_train, y_train)

# predict probabilities
pred_prob1 = model1.predict_proba(X_test)
pred_prob2 = model2.predict_proba(X_test)

from sklearn.metrics import roc_curve

# roc curve for models
fpr1, tpr1, thresh1 = roc_curve(y_test, pred_prob1[:,1], pos_label=1)
fpr2, tpr2, thresh2 = roc_curve(y_test, pred_prob2[:,1], pos_label=1)

# roc curve for tpr = fpr
random_probs = [0 for i in range(len(y_test))]
p_fpr, p_tpr, _ = roc_curve(y_test, random_probs, pos_label=1)

# Compute AUC score.
from sklearn.metrics import roc_auc_score
```

```
# auc scores
auc_score1 = roc_auc_score(y_test, pred_prob1[:,1])
auc_score2 = roc_auc_score(y_test, pred_prob2[:,1])

print("AUC scores")
print(auc_score1, auc_score2)

# matplotlib
import matplotlib.pyplot as plt
plt.style.use('seaborn')

# plot roc curves
plt.plot(fpr1, tpr1, linestyle='--',color='orange', label='Logistic
Regression')
plt.plot(fpr2, tpr2, linestyle='--',color='green', label='KNN')
plt.plot(p_fpr, p_tpr, linestyle='--', color='blue')
# title
plt.title('ROC curve')
# x label
plt.xlabel('False Positive Rate')
# y label
plt.ylabel('True Positive rate')

plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show();
```

Logistic Regression Model Python Script for Model 4

```
from sklearn.linear_model    import LogisticRegression
from sklearn.tree            import DecisionTreeClassifier
from sklearn.ensemble        import AdaBoostClassifier
from sklearn.ensemble        import RandomForestClassifier
from sklearn.metrics         import precision_score, recall_score, f1_score,
accuracy_score
from sklearn.model_selection import train_test_split
import numpy  as np
import pandas as pd

# Show all columns.
pd.set_option('display.max_columns', None)
pd.set_option('display.width', 1000)

# Prepare the data.
PATH     = "C:\\datasets\\"
CSV_DATA = "diabetes.csv"

df       = pd.read_csv(PATH + CSV_DATA)
df['bmiBin']  = pd.cut(x=df['BMI'], bins=[0, 18.49, 24.9, 29.9, 34.9, 39.9,
99])

tempDf  = df[['bmiBin']]                  # Isolate columns
dummyDf = pd.get_dummies(tempDf, columns=['bmiBin'])
X      = pd.concat(([df, dummyDf]), axis=1)      # Join dummy df with original
```

```python
del X['bmiBin']

X.rename(columns={'bmiBin_(0.0, 18.49]':'BMI - Underweight','bmiBin_(18.49, 24.9]':'BMI - Normal', 'bmiBin_(24.9, 29.9]':'BMI - Overweight', 'bmiBin_(29.9, 34.9]':'BMI - Obese Class 1', 'bmiBin_(34.9, 39.9]':'BMI - Obese Class 2', 'bmiBin_(39.9, 99.0]':'BMI - Obese Class 3' }, inplace=True)

X = X.copy()
del X['Diabetes_binary']
y = df['Diabetes_binary']

print(X.head())

def getUnfitModels():
    models = list()
    models.append(LogisticRegression())
    models.append(DecisionTreeClassifier())
    models.append(AdaBoostClassifier())
    models.append(RandomForestClassifier(n_estimators=10))
    return models

def evaluateModel(y_test, predictions, model):
    precision = round(precision_score(y_test, predictions),2)
    recall    = round(recall_score(y_test, predictions), 2)
    f1        = round(f1_score(y_test, predictions), 2)
    accuracy  = round(accuracy_score(y_test, predictions), 2)

    print("Precision:" + str(precision) + " Recall:" + str(recall) +\
          " F1:" + str(f1) + " Accuracy:" + str(accuracy) +\
          "   " + model.__class__.__name__)

def fitBaseModels(X_train, y_train, X_test, models):
    dfPredictions = pd.DataFrame()

    # Fit base model and store its predictions in dataframe.
    for i in range(0, len(models)):
        models[i].fit(X_train, y_train)
        predictions = models[i].predict(X_test)
        colName = str(i)
        dfPredictions[colName] = predictions
    return dfPredictions, models

def fitStackedModel(X, y):
    model = LogisticRegression()
    model.fit(X, y)
    return model

# Split data into train, test and validation sets.
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.70)
X_test, X_val, y_test, y_val = train_test_split(X_temp, y_temp,
test_size=0.50)

# Get base models.
unfitModels = getUnfitModels()

# Fit base and stacked models.
dfPredictions, models = fitBaseModels(X_train, y_train, X_test, unfitModels)
```

```
stackedModel          = fitStackedModel(dfPredictions, y_test)

# Evaluate base models with validation data.
print("\n** Evaluate Base Models **")
dfValidationPredictions = pd.DataFrame()
for i in range(0, len(models)):
    predictions = models[i].predict(X_val)
    colName = str(i)
    dfValidationPredictions[colName] = predictions
    evaluateModel(y_val, predictions, models[i])

# Evaluate stacked model with validation data.
stackedPredictions = stackedModel.predict(dfValidationPredictions)
print("\n** Evaluate Stacked Model **")
evaluateModel(y_val, stackedPredictions, stackedModel)


from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split


# split into train-test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=27)

# train models
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier

# logistic regression
model1 = LogisticRegression()
# knn
model2 = KNeighborsClassifier(n_neighbors=4)

# fit model
model1.fit(X_train, y_train)
model2.fit(X_train, y_train)

# predict probabilities
pred_prob1 = model1.predict_proba(X_test)
pred_prob2 = model2.predict_proba(X_test)

from sklearn.metrics import roc_curve

# roc curve for models
fpr1, tpr1, thresh1 = roc_curve(y_test, pred_prob1[:,1], pos_label=1)
fpr2, tpr2, thresh2 = roc_curve(y_test, pred_prob2[:,1], pos_label=1)

# roc curve for tpr = fpr
random_probs = [0 for i in range(len(y_test))]
p_fpr, p_tpr, _ = roc_curve(y_test, random_probs, pos_label=1)

# Compute AUC score.
from sklearn.metrics import roc_auc_score

# auc scores
auc_score1 = roc_auc_score(y_test, pred_prob1[:,1])
```

```python
auc_score2 = roc_auc_score(y_test, pred_prob2[:,1])
print("AUC scores")
print(auc_score1, auc_score2)

# matplotlib
import matplotlib.pyplot as plt
plt.style.use('seaborn')

# plot roc curves
plt.plot(fpr1, tpr1, linestyle='--',color='orange', label='Logistic
Regression')
plt.plot(fpr2, tpr2, linestyle='--',color='green', label='KNN')
plt.plot(p_fpr, p_tpr, linestyle='--', color='blue')
# title
plt.title('ROC curve')
# x label
plt.xlabel('False Positive Rate')
# y label
plt.ylabel('True Positive rate')

plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show()
```