Ravi, Sneha

Diabetes Insights: Data-Driven Approach to Early Detection

Analysis With ML TEchniques: PCA, Logistic Regression, Random forest classifier

**Background**

Diabetes is a widespread chronic condition that affects millions of people worldwide. Early detection and management are crucial to prevent complications and improve patients' quality of life. This project aims to develop a data-driven approach using machine learning techniques to predict early-stage diabetes risk based on various signs and symptoms. By leveraging advanced algorithms and feature importance analysis, the goal is to create an accurate and efficient model that can assist healthcare professionals in identifying potential diabetes cases early on, leading to timely interventions and better health outcomes.

**A Look into Statistical Methods**

Logistic Regression: This method is employed for binary classification tasks, fitting a linear model to the log odds of the response variable to predict probabilities of the target class. It provides insights into how each predictor influences the outcome.

Random Forest Classifier: An ensemble learning method that combines multiple decision trees to make predictions, offering robust and accurate results.

Principal Component Analysis (PCA): An unsupervised dimensionality reduction technique, PCA identifies the most significant components in the data that capture its variance and reduce complexity.

ROC Curve Analysis: This evaluation method plots the true positive rate (recall) against the false positive rate (specificity) at different probability thresholds, providing a comprehensive assessment of model performance.

**Data**

Dataset: Early-Stage Diabetes Risk Prediction

Size: 520 rows and 17 columns

Age: Age of the patient (numerical)

Gender: Gender of the patient (binary - 'Male' or 'Female')

Polyuria: Excessive urination (binary - 'Yes' or 'No')

Polydipsia: Excessive thirst (binary - 'Yes' or 'No')

Sudden Weight Loss: Sudden loss of weight (binary - 'Yes' or 'No')

Weakness: General weakness (binary - 'Yes' or 'No')

Polyphagia: Excessive hunger and food intake (binary - 'Yes' or 'No')

Genital Thrush: Presence of genital thrush infection (binary - 'Yes' or 'No')

Visual Blurring: Blurred vision (binary - 'Yes' or 'No')

Itching: Itching sensation (binary - 'Yes' or 'No')

Irritability: Irritability (binary - 'Yes' or 'No')

Delayed Healing: Delayed wound healing (binary - 'Yes' or 'No')

Partial Paresis: Partial paralysis (binary - 'Yes' or 'No')

Muscle Stiffness: Muscle stiffness (binary - 'Yes' or 'No')

Alopecia: Hair loss (binary - 'Yes' or 'No')

Obesity: Obesity condition (binary - 'Yes' or 'No')

Class: Target variable, indicating the presence of early-stage diabetes (binary - 'Positive' or 'Negative')

Fig 1.1:

A screenshot of a computer

Description automatically generated A blue screen with white text

Description automatically generated

**Data Exploring and Prep**

The dataset was inspected for missing values to identify any incomplete records. Basic information about the dataset, including the number of rows, columns, and data types, was obtained. Subsequently, histograms were plotted to visualize the distribution of the 'Age' attribute, providing valuable insights into the age distribution of individuals in the dataset. Categorical variables were encoded into numerical format, converting them into binary values for further analysis. To prepare for model training and evaluation, the dataset was split into training and testing sets with an 80:20 ratio. The dimensions of the training and testing sets were then verified to ensure the correct splitting process.

Distribution of the age variable which is the only continuous factor in analyzing the early-stage diabetes.

Fig 2.1:

A diagram of a distribution of age

Description automatically generated

**Data Analysis**

**Principal Component Analysis (PCA):**

PCA was utilized to reduce the dimensionality of the dataset and extract the most important features explaining the variance in the data. The cumulative explained variance ratio was examined to determine the number of principal components that capture a significant portion of the dataset's variability. PCA can be valuable in the real world to simplify complex datasets and facilitate data visualization, making it easier for healthcare professionals to interpret and communicate the patterns and relationships within the data.

By reducing the 16 attributes to 10 principal components, PCA captures 84.18% of the data variance, effectively simplifying the data. Removed Age column from the analysis as it is a continuous variable

Fig 3.1:

A graph with red lines and blue squares

Description automatically generated

PC-0: Polydipsia, Polyuria, and Gender. Explains 36.8% of the variance.

PC-1: Itching, Irritability, and Obesity. Explains 24.3% of variance

PC-2: Genital thrush, muscle stiffness, and partial paresis. Explains 13.4% of the variance.

PC-3: Polyuria, Polydipsia, and partial paresis. Explains 9.2% of the variance

PC-4: Polyphagia, Genital thrush, and visual blurring. Explains approximately 5.9% of the data variance

Fig 3.2:

A chart with many squares and numbers

Description automatically generated

PC-0: "Age" has a dominant role in this principal component, explaining about 97.7% of the variance in the data. Removing age will help with identifying actual trends of the variables that result in prediction of early diabetes.

Fig 3.3:

A graph of different colored squares

Description automatically generated

**Random Forest Classifier:**

Random Forest Classifier is an ensemble learning method that combines multiple decision trees to make predictions, offering robust and accurate results. It reduces overfitting by averaging the predictions of multiple trees, thereby improving generalization to new, unseen data. The model can handle large datasets with numerous features and is less sensitive to outliers

Random Forest Gini Model Results:

* Training Accuracy Score: 98.798%
* Test Accuracy Score: 96.154%
* Precision: 91.667%
* Recall: 97.059%
* Specificity: 95.714%

Strength:

* High Recall (97.059%) shows a strong ability to correctly identify positive cases of diabetes, which is crucial for early diabetes detection. The high recall ensures that a significant number of true positive cases are captured, reducing the risk of missing individuals with early-stage diabetes.

Limitations:

* Moderate Precision (91.667%): While the model's precision is relatively high, it means that around 8% of the cases classified as positive (diabetic) are false positives.

Fig 3.4:

A blue squares with white text

Description automatically generated

Fig 3.5:

A blue background with white text

Description automatically generated

A ROC curve towards the top-left corner suggests a strong and accurate model with good sensitivity and specificity.

Fig 3.6:

A graph of a curve

Description automatically generated with medium confidence

**Logistic Regression:**

Logistic Regression was applied to model the probability of diabetes occurrence based on multiple predictor variables. The model was fitted using the training dataset and evaluated on the testing dataset. The Logistic Regression model achieved high accuracy on both training and testing data, indicating its effectiveness in predicting diabetes outcomes.

Logistic Regression Model Results:

* Training Accuracy Score: 92.788%
* Logistic Regression Model Testing Accuracy Score: 91.346%
* Precision: 82.051
* Recall: 94.118
* Specificity: 90.0

Strength:

* High Recall (94.118%): The model has a strong ability to correctly identify positive cases of diabetes, reducing the risk of missing true diabetic patients.

Limitation:

* Precision (82.051%): The model's precision indicates that there is a significant number of false positives, potentially leading to unnecessary stress and follow-up tests for non-diabetic patients.

Fig 3.7:

A diagram of a logistic regression

Description automatically generated

Fig 3.8:

A screen shot of a computer

Description automatically generated

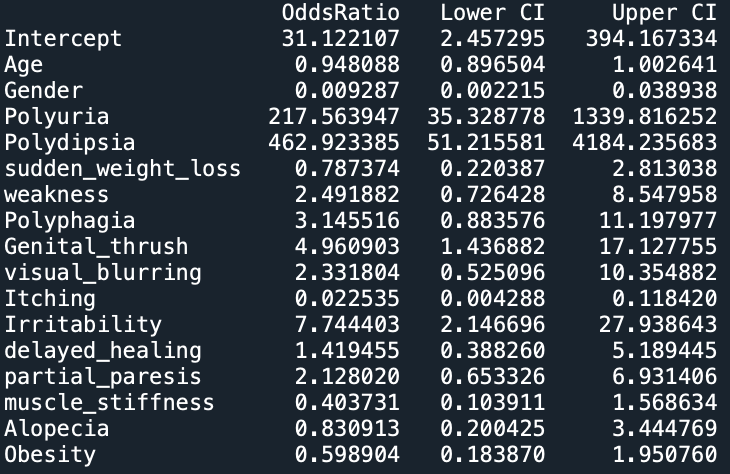
Age: The confidence interval for age includes 1 so that not a great predictor of early-stage diabetes

Males are associated with a significantly lower likelihood of having diabetes compared to females. Females are 99.07% likely to get diabetes at an early stage.

Polyuria: Individuals with Polyuria are approximately 217.56 times as likely to have early diabetes compared to those without Polyuria.

Polydipsia: Individuals with Polydipsia are approximately 462.92 times as likely to have early diabetes compared to those without Polydipsia.

Fig 3.9



A ROC curve towards the top-left corner suggests a strong and accurate model with good sensitivity and specificity.

Fig 4.0:

A graph of a logistic regression roc curve

Description automatically generated

**Feature Importance:**

Helps in understanding which variables contribute significantly to the overall trends. The analysis of feature importance revealed crucial variables for detecting diabetes early on.

The top three most helpful features are:

* Polydipsia (excessive thirst)
* Polyuria (frequent urination)
* Gender

Fig 4.1:

A graph of a number of diabetes

Description automatically generated with medium confidence

**Conclusion**

Best-performing model: The Random Forest Classifier exhibited exceptional performance, achieving 96.154% testing accuracy and 97.059% recall.

Model Reliability: The Random Forest Classifier is very reliable with the combination of 97.059% recall and 91.667% precision, indicating its reliability in diagnosing people with diabetes and avoiding misdiagnosis. This is essential in the early detection of diabetes, as misdiagnoses can lead to unnecessary treatments.

Feature Importance: The top three most influential features for detecting potential diabetes cases are Polydipsia, Polyuria, Gender. PCA analysis confirmed that these variables significantly contribute to PC-0, explaining about 36.8% of the data's variance. These symptoms are highly indicative of early-stage diabetes.

**Appendix**

Data Source: <https://archive.ics.uci.edu/dataset/529/early+stage+diabetes+risk+prediction+dataset>

Code used for analysis:

import os

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

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

from sklearn.preprocessing import LabelEncoder

import statsmodels.api as sm

import statsmodels.formula.api as smf

from sklearn.metrics import (confusion\_matrix, accuracy\_score)

import dmba as dmba

from sklearn.decomposition import PCA

from sklearn.ensemble import RandomForestClassifier

from collections import defaultdict

from sklearn import metrics

from sklearn.metrics import confusion\_matrix, precision\_recall\_fscore\_support

from sklearn.metrics import roc\_curve, accuracy\_score, roc\_auc\_score

from xgboost import XGBClassifier

os.chdir(r'/Users/sneharavi/Desktop/Quantitaive\_Methods/Final Project')

os.getcwd()

#https://archive.ics.uci.edu/dataset/529/early+stage+diabetes+risk+prediction+dataset

df = pd.read\_csv('diabetes\_data.csv')

df.isna().sum()

df.info()

sns.histplot(data=df, x='Age',palette="dark")

plt.title('Distribution of Age')

# Male = 1, Female = 0

gender = LabelEncoder()

df['Gender'] = gender.fit\_transform(df['Gender'])

# Yes = 1, No = 0

polyuria = LabelEncoder()

df['Polyuria']= polyuria.fit\_transform(df['Polyuria'])

# Yes = 1, No = 0

polydipsia = LabelEncoder()

df['Polydipsia']= polydipsia.fit\_transform(df['Polydipsia'])

# Yes = 1, No = 0

df.rename(columns = {'sudden weight loss':'sudden\_weight\_loss'}, inplace= True)

sudden\_weight\_loss = LabelEncoder()

df['sudden\_weight\_loss']= sudden\_weight\_loss.fit\_transform(df['sudden\_weight\_loss'])

# Yes = 1, No = 0

weakness = LabelEncoder()

df['weakness']= weakness.fit\_transform(df['weakness'])

# Yes = 1, No = 0

polyphagia = LabelEncoder()

df['Polyphagia']= polyphagia.fit\_transform(df['Polyphagia'])

# Yes = 1, No = 0

df.rename(columns = {'Genital thrush':'Genital\_thrush'}, inplace= True)

genital\_thrush = LabelEncoder()

df['Genital\_thrush']= genital\_thrush.fit\_transform(df['Genital\_thrush'])

# Yes = 1, No = 0

df.rename(columns = {'visual blurring':'visual\_blurring'}, inplace= True)

visual\_blurring = LabelEncoder()

df['visual\_blurring']= visual\_blurring.fit\_transform(df['visual\_blurring'])

# Yes = 1, No = 0

Itching = LabelEncoder()

df['Itching']= Itching.fit\_transform(df['Itching'])

# Yes = 1, No = 0

Irritability = LabelEncoder()

df['Irritability']= Irritability.fit\_transform(df['Irritability'])

# Yes = 1, No = 0

df.rename(columns = {'delayed healing':'delayed\_healing'}, inplace= True)

delayed\_healing = LabelEncoder()

df['delayed\_healing']= delayed\_healing.fit\_transform(df['delayed\_healing'])

# Yes = 1, No = 0

df.rename(columns = {'partial paresis':'partial\_paresis'}, inplace= True)

partial\_paresis = LabelEncoder()

df['partial\_paresis']= partial\_paresis.fit\_transform(df['partial\_paresis'])

# Yes = 1, No = 0

df.rename(columns = {'muscle stiffness':'muscle\_stiffness'}, inplace= True)

muscle\_stiffness = LabelEncoder()

df['muscle\_stiffness']= muscle\_stiffness.fit\_transform(df['muscle\_stiffness'])

# Yes = 1, No = 0

alopecia = LabelEncoder()

df['Alopecia']= alopecia.fit\_transform(df['Alopecia'])

# Yes = 1, No = 0

obesity = LabelEncoder()

df['Obesity']= obesity.fit\_transform(df['Obesity'])

# Yes = 1, No = 0

df.rename(columns = {'class':'Diabetes'}, inplace= True)

output = LabelEncoder()

df['Diabetes']= output.fit\_transform(df['Diabetes'])

X = df.drop(columns='Diabetes')

Y = df['Diabetes']

X.columns

df.columns

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.2, shuffle=True, random\_state=100 )

print('X Train Shape:',X\_train.shape)

print('Y Train Shape:',y\_train.shape)

print('X Test Shape:',X\_test.shape)

print('Y Test Shape:',y\_test.shape)

#-------------------------------------------------------------------------------------------------------------------------------------

# Random Forest

model\_gini = RandomForestClassifier(n\_estimators=35, criterion='gini', max\_depth=5, oob\_score=True)

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

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

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

print('Random Forest Gini Model Training Accuracy Score:' ,round(model\_gini.score(X\_train, y\_train)\*100,3))

print('Random Forest Gini Model Test Accuracy Score:', round(model\_gini.score(X\_test, y\_test)\*100,3))

# RandomForest: Accuracy

pred\_gini = model\_gini.predict(X\_test)

pred\_gini

cm\_gini = confusion\_matrix(y\_test, pred\_gini)

cm\_gini

print('Random Forest Gini Model Training Accuracy Score:' ,round(model\_gini.score(X\_train, y\_train)\*100,3))

print('Random Forest Gini Model Test Accuracy Score:', round(model\_gini.score(X\_test, y\_test)\*100,3))

print('Precision', round((cm\_gini[0, 0] / sum(cm\_gini[:, 0]))\*100,3))

print('Recall', round((cm\_gini[0, 0] / sum(cm\_gini[0, :]))\*100,3))

print('Specificity', round((cm\_gini[1, 1] / sum(cm\_gini[1, :]))\*100,3))

confusion\_gini = dmba.classificationSummary(y\_test, pred\_gini, class\_names=model\_gini.classes\_)

labels = ([['True Negative: %s'%cm\_gini[0,0], 'False Positive: %s'%cm\_gini[0,1]],['False Negative: %s'%cm\_gini[1,0], 'True Positive: %s'%cm\_gini[1,1]]])

fig, ax = plt.subplots()

sns.heatmap(cm\_gini, annot= labels, fmt = '', cmap= 'Blues')

plt.title('Confusion Matrix: Random Forest Classifier (Gini)')

pred\_prob\_gin = model\_gini.predict(X\_test)

# Calculate the ROC curve

fpr, tpr, thresholds = roc\_curve(y\_test, pred\_prob)

auc\_score = roc\_auc\_score(y\_test, pred\_gini)

# Plot the ROC curve

plt.title('Random Forest Gini Model ROC Curve')

plt.plot(fpr, tpr, color='purple', label=f'AUC = {auc\_score:.2f}')

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

plt.xlim([0, 1])

plt.ylim([0, 1])

plt.ylabel('Recall (True Positive Rate)')

plt.xlabel('1 - Specificity (False Positive Rate)')

plt.legend(loc='lower right')

plt.show()

#-------------------------------------------------------------------------------------------------------------------------------------

# Feature Importance Gini

X = df.drop(columns='Diabetes')

Y = df['Diabetes']

scores\_gini = defaultdict(list)

for \_ in range(3):

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.20, random\_state=42)

model\_gini = RandomForestClassifier(criterion='gini')

model\_gini.fit(x\_train, y\_train)

acc = metrics.accuracy\_score(y\_test, model\_gini.predict(x\_test))

for column in X.columns:

X\_t = x\_test.copy()

X\_t[column] = np.random.permutation(X\_t[column].values)

shuff\_acc = metrics.accuracy\_score(y\_test, model\_gini.predict(X\_t))

scores\_gini[column].append((acc - shuff\_acc) / acc)

print('Gini Features sorted by their score:')

print(sorted([(round(np.mean(score), 4), feat) for feat, score in scores\_gini.items()], reverse=True))

importances\_gini = model\_gini.feature\_importances\_

df\_gini = pd.DataFrame({'feature': X.columns, 'Accuracy decrease': [np.mean(scores\_gini[column]) for column in X.columns], 'Gini decrease': importances\_gini})

# Sorting DataFrames based on Accuracy Decrease

df\_gini = df\_gini.sort\_values('Accuracy decrease', ascending=False)

# Plotting the Feature Importance

plt.figure(figsize=(8, 5))

plt.barh(df\_gini['feature'], df\_gini['Gini decrease'], color='lightblue')

plt.xlabel('Gini Importance')

plt.ylabel('Features')

plt.title('Gini Feature Importance for Early Diabetes Detection')

plt.gca().invert\_yaxis()

plt.tight\_layout()

plt.show()

#-------------------------------------------------------------------------------------------------------------------------------------

# PCA

sp\_pca = PCA()

sp\_pca.fit(df)

sp\_pca.components\_

df\_components\_ = pd.DataFrame(sp\_pca.components\_,columns=df.columns,index = ['PC-'+str(i) for i in range(sp\_pca.n\_components\_)])

sp\_pca.explained\_variance\_

sp\_pca.explained\_variance\_ratio\_

explained\_variance = pd.DataFrame(sp\_pca.explained\_variance\_ratio\_)

explained\_variance.plot()

cum\_sum\_explained\_variance = np.cumsum(sp\_pca.explained\_variance\_ratio\_)

cum\_sum\_explained\_variance

cum\_sum\_explained\_variance\_df = pd.DataFrame(cum\_sum\_explained\_variance)

cum\_sum\_explained\_variance\_df.rename(columns = {'0':'Cumulative Exp\_Variance'}, inplace= True)

num\_of\_steps = range(0,len(cum\_sum\_explained\_variance\_df))

%matplotlib inline

ax = explained\_variance.head(20).plot.bar(legend=False, figsize=(12, 10))

ax.set\_xlabel('Principal Component', size = 15)

ax.set\_ylabel('Explained Variance Ratio', size=15)

plt.step(num\_of\_steps, cum\_sum\_explained\_variance\_df, where='mid',label='Cumulative explained variance', color='red')

plt.title('Explained Variance Ratio by Principal Components', size=20)

plt.show()

loadings = pd.DataFrame(sp\_pca.components\_[0:5, :], columns=df.columns)

print(loadings)

maxPC = 1.01 \* loadings.loc[0:5, :].abs().to\_numpy().max() # this is for automatically setting the y-axis scale, so that you can always zoom into the principal components no matter how small their values are

f, axes = plt.subplots(5, 1, figsize=(10, 9), sharex=True)

for i, ax in enumerate(axes):

pc\_loadings = loadings.loc[i, :]

colors = ['C0' if l > 0 else 'C1' for l in pc\_loadings]

ax.axhline(color='#888888')

pc\_loadings.plot.bar(ax=ax, color=colors)

ax.set\_ylabel(f'PC{i}')

ax.set\_ylim(-maxPC, maxPC)

plt.tight\_layout()

plt.show()

# PCA

df1 = df.drop(columns='Age')

sp\_pca = PCA(n\_components=10)

sp\_pca.fit(df1)

sp\_pca.components\_

df\_components\_ = pd.DataFrame(sp\_pca.components\_,columns=df1.columns,index = ['PC-'+str(i) for i in range(sp\_pca.n\_components\_)])

sp\_pca.explained\_variance\_

sp\_pca.explained\_variance\_ratio\_

explained\_variance = pd.DataFrame(sp\_pca.explained\_variance\_ratio\_)

explained\_variance.plot()

cum\_sum\_explained\_variance = np.cumsum(sp\_pca.explained\_variance\_ratio\_)

cum\_sum\_explained\_variance

cum\_sum\_explained\_variance\_df = pd.DataFrame(cum\_sum\_explained\_variance)

cum\_sum\_explained\_variance\_df.rename(columns = {'0':'Cumulative Exp\_Variance'}, inplace= True)

num\_of\_steps = range(0,len(cum\_sum\_explained\_variance\_df))

%matplotlib inline

ax = explained\_variance.head(20).plot.bar(legend=False, figsize=(12, 10))

ax.set\_xlabel('Principal Component', size = 15)

ax.set\_ylabel('Explained Variance Ratio', size=15)

plt.step(num\_of\_steps, cum\_sum\_explained\_variance\_df, where='mid',label='Cumulative explained variance', color='red')

plt.title('Explained Variance Ratio by Principal Components', size=20)

plt.show()

loadings = pd.DataFrame(sp\_pca.components\_[0:10, :], columns=df1.columns)

print(loadings)

maxPC = 1.01 \* loadings.loc[0:10, :].abs().to\_numpy().max() # this is for automatically setting the y-axis scale, so that you can always zoom into the principal components no matter how small their values are

f, axes = plt.subplots(10, 1, figsize=(10, 10.5), sharex=True)

for i, ax in enumerate(axes):

pc\_loadings = loadings.loc[i, :]

colors = ['C0' if l > 0 else 'C1' for l in pc\_loadings]

ax.axhline(color='#888888')

pc\_loadings.plot.bar(ax=ax, color=colors)

ax.set\_ylabel(f'PC{i}')

ax.set\_ylim(-maxPC, maxPC)

plt.tight\_layout()

plt.show()

#-------------------------------------------------------------------------------------------------------------------------------------

# XGBoost Classifier

X\_train.info()

y\_train.info()

xgb = XGBClassifier(objective='binary:logistic', use\_label\_encoder=False, subsample = 1, max\_depth = 2,eval\_metric = 'error', random\_state=100)

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

print('XG Boost Model Training Accuracy Score:',round((xgb.score(X\_train, y\_train))\*100,3))

print('XG Boost Model Testing Accuracy Score:',round((xgb.score(X\_test, y\_test))\*100,3))

xgb\_predict = xgb.predict(X\_test)

xgb\_predict

cm\_xgb = confusion\_matrix(y\_test, xgb\_predict)

cm\_xgb

print('XG Boost Model Training Accuracy Score:',round((xgb.score(X\_train, y\_train))\*100,3))

print('XG Boost Model Testing Accuracy Score:',round((xgb.score(X\_test, y\_test))\*100,3))

print('Precision:', round((cm\_xgb[0, 0] / sum(cm\_xgb[:, 0]))\*100,3))

print('Recall:', round((cm\_xgb[0, 0] / sum(cm\_xgb[0, :]))\*100,3))

print('Specificity:', round((cm\_xgb[1, 1] / sum(cm\_xgb[1, :]))\*100,3))

%matplotlib inline

confusion\_xg = dmba.classificationSummary(y\_test, xgb\_predict, class\_names=xgb.classes\_)

labels = ([['True Negative: %s'%cm\_xgb[0,0], 'False Positive: %s'%cm\_xgb[0,1]],['False Negative: %s'%cm\_xgb[1,0], 'True Positive: %s'%cm\_xgb[1,1]]])

fig, ax = plt.subplots()

sns.heatmap(cm\_xgb, annot= labels, fmt = '', cmap="BuPu")

plt.title('Confusion Matrix: XGB Classifier')

fpr, tpr, thresholds = roc\_curve(y\_test, xgb.predict\_proba(X\_test)[:, 0])

auc\_score = roc\_auc\_score(y\_test, xgb\_predict)

# Plot the ROC curve

plt.title('XG Boost ROC Curve')

plt.plot(fpr, tpr, color='blue', label=f'AUC = {auc\_score:.2f}')

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

plt.xlim([0, 1])

plt.ylim([0, 1])

plt.ylabel('Recall (True Positive Rate)')

plt.xlabel('1 - Specificity (False Positive Rate)')

plt.legend(loc='upper left')

plt.show()

#-------------------------------------------------------------------------------------------------------------------------------------

# Logistic Regression

logit\_model = smf.logit(formula=('Diabetes ~ Age + Gender + Polyuria + Polydipsia + sudden\_weight\_loss + weakness + Polyphagia + Genital\_thrush + visual\_blurring + Itching + Irritability + delayed\_healing +partial\_paresis + muscle\_stiffness+ Alopecia + Obesity'),data=pd.concat([X\_train, y\_train], axis=1)).fit()

logit\_model.summary()

yhat = round(logit\_model.predict(X\_train[['Age', 'Gender', 'Polyuria', 'Polydipsia', 'sudden\_weight\_loss',

'weakness', 'Polyphagia', 'Genital\_thrush', 'visual\_blurring',

'Itching', 'Irritability', 'delayed\_healing', 'partial\_paresis',

'muscle\_stiffness', 'Alopecia', 'Obesity']]))

pred = round(logit\_model.predict(X\_test[['Age', 'Gender', 'Polyuria', 'Polydipsia', 'sudden\_weight\_loss',

'weakness', 'Polyphagia', 'Genital\_thrush', 'visual\_blurring',

'Itching', 'Irritability', 'delayed\_healing', 'partial\_paresis',

'muscle\_stiffness', 'Alopecia', 'Obesity']]))

cm = confusion\_matrix(y\_test,pred)

accuracy\_score(yhat, y\_train)

accuracy\_score(pred, y\_test)

print('Logistic Regression Model Training Accuracy Score:',round(accuracy\_score(yhat, y\_train)\*100,3))

print('Logistic Regression Model Testing Accuracy Score:',round(accuracy\_score(pred, y\_test)\*100,3))

print('Precision', round((cm[0, 0] / sum(cm[:, 0]))\*100,3))

print('Recall', round((cm[0, 0] / sum(cm[0, :]))\*100,3))

print('Specificity', round((cm[1, 1] / sum(cm[1, :]))\*100,3))

%matplotlib inline

labels = ([['True Negative: %s'%cm[0,0], 'False Positive: %s'%cm[0,1]],['False Negative: %s'%cm[1,0], 'True Positive: %s'%cm[1,1]]])

fig, ax = plt.subplots()

sns.heatmap(cm, annot= labels, fmt = '', cmap='BuPu')

plt.title('Confusion Matrix: Logistic Regression')

plt.show()

pred\_prob = logit\_model.predict(X\_test)

# Calculate the ROC curve

fpr, tpr, thresholds = roc\_curve(y\_test, pred\_prob)

auc\_score = roc\_auc\_score(y\_test, pred)

# Plot the ROC curve

plt.title('Logistic Regression ROC Curve')

plt.plot(fpr, tpr, color='green', label=f'AUC = {auc\_score:.2f}')

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

plt.xlim([0, 1])

plt.ylim([0, 1])

plt.ylabel('Recall (True Positive Rate)')

plt.xlabel('1 - Specificity (False Positive Rate)')

plt.legend(loc='lower right')

plt.show()