NAME – ANUSHKA GUPTA

BATCH- ML-MAJOR-JUNE

PROJECT NAME - This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

Do classification by at least 2 Algorithms

PROJECT :

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

sns.set\_style("whitegrid")

sns.set\_context("poster")

%matplotlib inline

# library imported

# Other Libraries

# ML Algorithms

from sklearn.linear\_model import LogisticRegression, LinearRegression

from sklearn.naive\_bayes import GaussianNB

from sklearn.ensemble import RandomForestClassifier

# For building models

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

from sklearn.preprocessing import StandardScaler

# For measuring performance

from sklearn import metrics

from sklearn.model\_selection import cross\_val\_score

# Ignore warnings

import warnings

warnings.filterwarnings('ignore')

# Load the Dataset

# importing Dataset from Local drive

from google.colab import files

Uploaded=files.upload()

df = pd.read\_csv("diabetes.csv")

# Check if the data is properly loaded

print("Size of the dataset:", df.shape)

df.head()

**output :** Size of the dataset: (768, 9)

# The shape of the loaded dataset is same as what is specified by the source, so we're good to go!

For further inspection, shown below is the list of columns of the data along with its count and type

df.info()

**output :** <class 'pandas.core.frame.DataFrame'>

RangeIndex: 768 entries, 0 to 767

Data columns (total 9 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Pregnancies 768 non-null int64

1 Glucose 768 non-null int64

2 BloodPressure 768 non-null int64

3 SkinThickness 768 non-null int64

4 Insulin 768 non-null int64

5 BMI 768 non-null float64

6 DiabetesPedigreeFunction 768 non-null float64

7 Age 768 non-null int64

8 Outcome 768 non-null int64

dtypes: float64(2), int64(7)

memory usage: 54.1 KB

# Explore the Dataset

df.describe()

**#** The table above summarizes the common statistics stuff we can compute from the data.

By observing the row count, we can confirm that there is no missing data.

But if we look at the min row, there are columns with zero values that are not expected to have zero values. These columns are Glucose, BloodPressure, SkinThickness and BMI. Imagine meeting someone having zero of these attributes, it means that person is a ghost.

# Display the number of zero values per columns

print("---Count zero values per column---")

for col in ["Glucose", "BloodPressure", "SkinThickness", "BMI"]:

    print("{}: {}".format( col, df[col].value\_counts()[0] ))

# Print the percentage of rows with zero values

print("\n---Rows with zero values in %---")

print("% of rows with zero values in all columns listed above:",

      (df[(df["Glucose"]==0) | (df["BloodPressure"]==0) |

          (df["BMI"]==0) | (df["SkinThickness"]==0)].shape[0] / df.shape[0]) \* 100)

print("% of rows with zero values in columns 'Glucose', 'BloodPressure' and 'BMI':",

      (df[(df["Glucose"]==0) | (df["BloodPressure"]==0) |

          (df["BMI"]==0)].shape[0] / df.shape[0]) \* 100)

Output : --Count zero values per column---

Glucose: 5

BloodPressure: 35

SkinThickness: 227

BMI: 11

---Rows with zero values in %---

% of rows with zero values in all columns listed above: 30.729166666666668

% of rows with zero values in columns 'Glucose', 'BloodPressure' and 'BMI': 5.729166666666666

# We can remove rows with zero values in columns Glucose, BloodPressure or BMI since these rows are just around 6% of the data. While, we can impute values for SkinThickness because we don't want 30% of our data to be thrown away.

Relationships Now, let's look at the correlation between the predictors.

# Determine correlation between variables

df.corr()

# Visualize Correlation

## Generate a mask for the upper triangle

mask = np.zeros\_like(df.corr(), dtype=np.bool)

mask[np.triu\_indices\_from(mask)] = True

## Set up the matplotlib figure

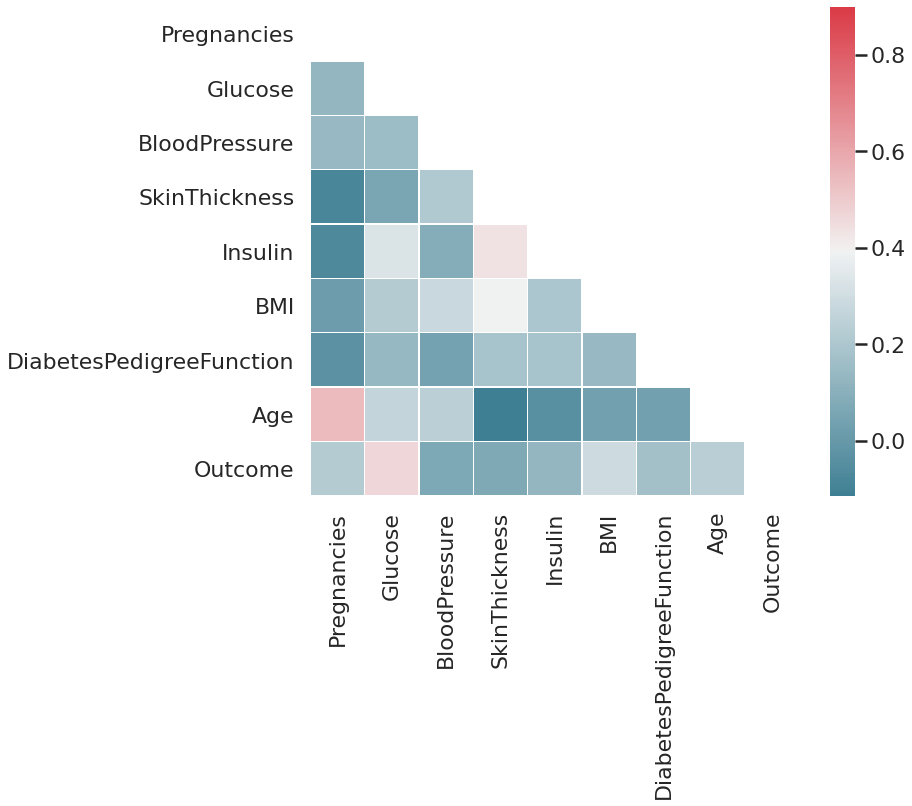
f, ax = plt.subplots(figsize=(11, 9))

## Generate a custom diverging colormap

cmap = sns.diverging\_palette(220, 10, as\_cmap=True)

## Draw the heatmap with the correct aspect ratio

sns.heatmap(df.corr(), mask=mask, cmap=cmap, vmax=.9, square=True, linewidths=.5, ax=ax)



# Glucose has the highest correlation with our target variable Outcome, followed by BMI. While, BloodPressure and SkinThickness has the lowest correlation.

We can look more closely on the relationship of Outcome with the predictors using histograms, as shown below. The first histogram of the cell denotes when Outcome==0 or non-diabetic while the other one represents when Outcome==1 or diabetic.

# Function to plot histogram

def histplt(col):

    print("----- Outcome vs {}-----".format(col))

    print(df[["Outcome", col]].groupby("Outcome").hist(figsize=(10,3)))

# Plot histogram for Outcome vs Pregnancies

histplt("Pregnancies")

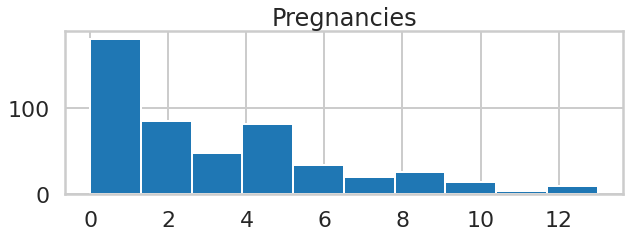
**output :** ----- Outcome vs Pregnancies-----

Outcome

0 [[AxesSubplot(0.125,0.125;0.775x0.755)]]

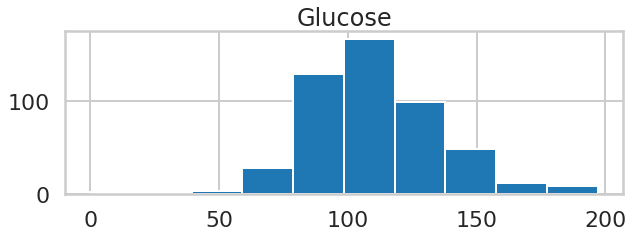
1 [[AxesSubplot(0.125,0.125;0.775x0.755)]]

dtype: object

****

# Plot histogram for Outcome vs Glucose

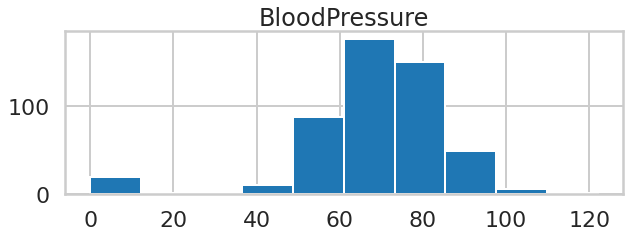
histplt("Glucose")



# As expected for Glucose, diabetic people has higher levels of it while non-diabetic people has the normal Glucose which is around 90-100.

# Plot histogram for Outcome vs BloodPressure

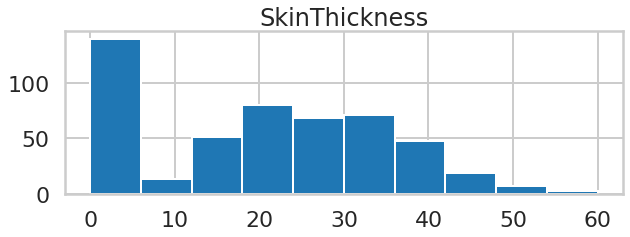
histplt("BloodPressure")



# The trends forBloodPressure look the same for diabetic and non-diabetic people.

# Plot histogram for Outcome vs SkinThickness

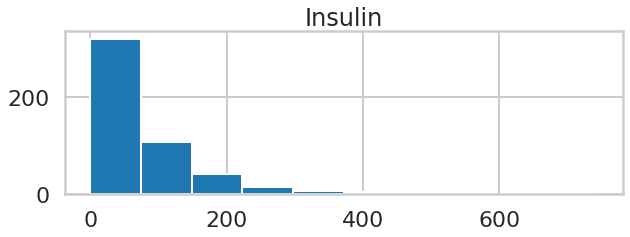
histplt("SkinThickness")



# The average SkinThickness of diabetic people looks slightly higher than non-diabetic people.

# Plot histogram for Outcome vs Insulin

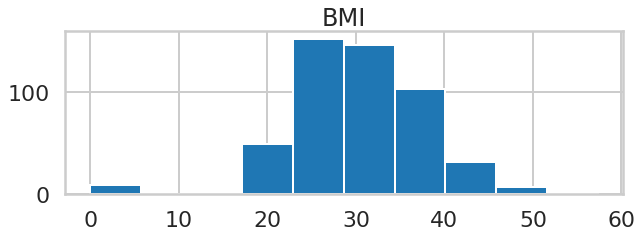
histplt("Insulin")



# Suprisingly, the insulin levels for both outcomes are pretty much the same, except that the range of values of insulin for non-diabetic people is smaller.

# Plot histogram for Outcome vs BMI

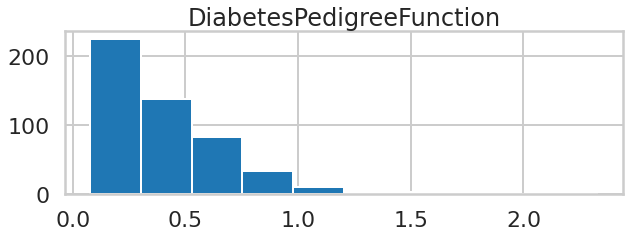
histplt("BMI")



# Same with SkinThickness, the average BMI of diabetic people looks slightly higher than non-diabetic people.

# Plot histogram for Outcome vs DiabetesPedigreeFunction

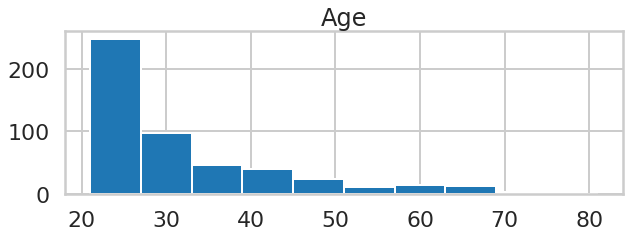
histplt("DiabetesPedigreeFunction")



# The max and average DiabetesPedigreeFunctionvalue of diabetic people is higher than that of the non-diabetics.

# Plot histogram for Outcome vs Age

histplt("Age")



# Most of the non-diabetic people are in their 20s. The distribution of diabetic people with Age within the 20-40 is almost uniform, and there are also many diabetic people aged 50 and above.

Lastly, this data is imbalanced as usual. We can resample, but we will not do it for now. The value counts for each outcome is shown below

# Check how balanced / imbalanced the data is

df["Outcome"].value\_counts()

**output :** 0 500

1 268

Name: Outcome, dtype: int64

# Prepare the Data for Modelling

Handling Zero Values Remove rows As stated above, we can remove rows with zero values in columns Glucose, BloodPressure and BMI.

# Create new dataframe wherein the unwanted rows are not included

df\_rem = df[ (df["Glucose"]!=0) & (df["BloodPressure"]!=0) & (df["BMI"]!=0) ]

# Check the new dataframe

print("Size of dataframe:", df\_rem.shape)

df\_rem.head()

**output :** Size of dataframe: (724, 9)

# Check minimum values of the new dataframe

df\_rem.describe().loc["min"]

**output :** Pregnancies 0.000

Glucose 44.000

BloodPressure 24.000

SkinThickness 0.000

Insulin 0.000

BMI 18.200

DiabetesPedigreeFunction 0.078

Age 21.000

Outcome 0.000

Name: min, dtype: float64

# Impute Since there are many rows with zero values in SkinThickness, we will use Linear Regression to change those values to non-zeroes.

# Separate rows that have zero value in SkinThickness from the rows that have value > 0

df\_impute = df\_rem[df\_rem["SkinThickness"]!=0]

df\_0 = df\_rem[df\_rem["SkinThickness"]==0]

# Use Linear Regression for imputation

## Instantiate the Linear Regression Algorithm

linreg = LinearRegression()

# Fit the dataframe with SkinThickness > 0 on linreg

linreg.fit(df\_impute.drop(["SkinThickness", "Outcome"], axis=1), df\_impute["SkinThickness"])

# Get the new values of SkinThickness

df\_0["SkinThickness"] = linreg.predict(df\_0.drop(["SkinThickness","Outcome"], axis=1))

# Merge the imputed datas, then check

df\_impute = df\_impute.append(df\_0)

df\_impute.describe()

**#** Train-Test Split

# Seaprate the predictors from the target variable

X = df\_impute.drop(["Outcome"], axis=1)

y = df\_impute["Outcome"]

print("Size of x (predictors):\t{}\nSize of y (target):\t{}".format(X.shape, y.shape))

**output :** Size of x (predictors): (724, 8)

Size of y (target): (724,)

# Split the dataset into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, stratify=y, random\_state=1)

# Check shape to make sure it is all in order

print("Size of x\_train: {} \t Size of x\_test: {} \nSize of y\_train: {} \t Size of y\_test: {}".format(

    X\_train.shape, X\_test.shape, y\_train.shape, y\_test.shape))

**Output :** Size of x\_train: (506, 8) Size of x\_test: (218, 8)

Size of y\_train: (506,) Size of y\_test: (218,)

print(y\_train.value\_counts(), '\n', y\_test.value\_counts())

**output :** 0 332

1 174

Name: Outcome, dtype: int64

0 143

1 75

Name: Outcome, dtype: int64

# Standard Scaler

# Instantiate the Standard Scaler

scaler = StandardScaler()

# Fit the scaler to the training set

scaler.fit(X\_train)

# Transform the training set

X\_train\_scaled = scaler.transform(X\_train)

# Transform the test set

X\_test\_scaled = scaler.transform(X\_test)

# Change to Pandas dataframe for easier viewing and manipulation of the data

X\_train\_sdf = pd.DataFrame(X\_train\_scaled, index=X\_train.index, columns=X\_train.columns)

X\_test\_sdf = pd.DataFrame(X\_test\_scaled, index=X\_test.index, columns=X\_test.columns)

**#** Build the Models

# Initialized for easy plotting of confusion matrix

def confmatrix(y\_pred, title):

    cm = metrics.confusion\_matrix(y\_test, y\_pred)

    df\_cm = pd.DataFrame(cm, columns=np.unique(y\_test), index = np.unique(y\_test))

    df\_cm.index.name = 'Actual'

    df\_cm.columns.name = 'Predicted'

    plt.figure(figsize = (10,7))

    plt.title(title)

    sns.set(font\_scale=1.4) # For label size

    sns.heatmap(df\_cm, cmap="Blues", annot=True,annot\_kws={"size": 16})

CLASSIFICATION ALGORITHMS

Logistic Regression

Build/Train the Model

# Instantiate the Algorithm

logreg = LogisticRegression()

# Train/Fit the model

logreg.fit(X\_train\_scaled, y\_train)

**output :** LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True,

intercept\_scaling=1, l1\_ratio=None, max\_iter=100,

multi\_class='auto', n\_jobs=None, penalty='l2',

random\_state=None, solver='lbfgs', tol=0.0001, verbose=0,

warm\_start=False)

# Validate the Model

# Predict on the test set

logreg\_pred = logreg.predict(X\_test\_scaled)

**#** Classification Report

# Get performance metrics

logreg\_score = metrics.accuracy\_score(y\_test, logreg\_pred) \* 100

# Print classification report

print("Classification report for {}:\n{}".format(logreg, metrics.classification\_report(y\_test, logreg\_pred)))

print("Accuracy score:", logreg\_score)

**output :**

Classification report for LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True,

intercept\_scaling=1, l1\_ratio=None, max\_iter=100,

multi\_class='auto', n\_jobs=None, penalty='l2',

random\_state=None, solver='lbfgs', tol=0.0001, verbose=0,

warm\_start=False):

precision recall f1-score support

0 0.82 0.89 0.85 143

1 0.75 0.63 0.68 75

accuracy 0.80 218

macro avg 0.78 0.76 0.77 218

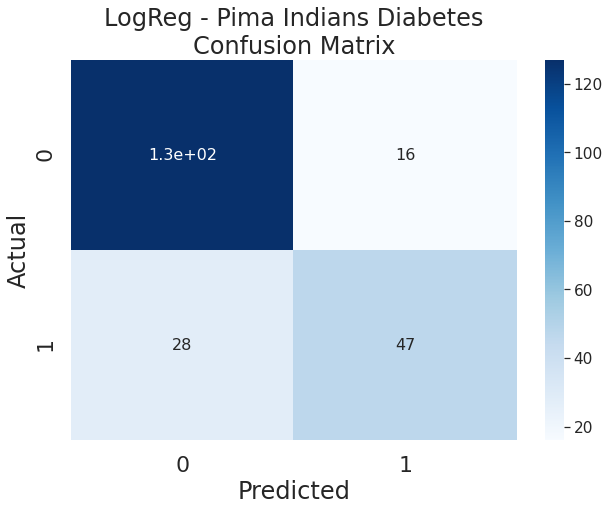
weighted avg 0.79 0.80 0.79 218

Accuracy score: 79.81651376146789

# Confusion Matrix

# Plot the confusion matrix

confmatrix(logreg\_pred, "LogReg - Pima Indians Diabetes\nConfusion Matrix")

****

**#** Random Forest Classifier

Build/Train the Model

# Instantiate algorithm

rf = RandomForestClassifier()

# Fit the model to the data

rf.fit(X\_train\_scaled, y\_train)

**output :**

RandomForestClassifier(bootstrap=True, ccp\_alpha=0.0, class\_weight=None,

criterion='gini', max\_depth=None, max\_features='auto',

max\_leaf\_nodes=None, max\_samples=None,

min\_impurity\_decrease=0.0, min\_impurity\_split=None,

min\_samples\_leaf=1, min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0, n\_estimators=100,

n\_jobs=None, oob\_score=False, random\_state=None,

verbose=0, warm\_start=False)

# Validate the Model

# Predict on the test set

rf\_pred = rf.predict(X\_test\_scaled)

**#** Classification Report

# Get performance metrics

rf\_score = metrics.accuracy\_score(y\_test, rf\_pred) \* 100

# Print classification report

print("Classification report for {}:\n{}".format(rf, metrics.classification\_report(y\_test, rf\_pred)))

print("Accuracy score:", rf\_score)

**output :**

Classification report for RandomForestClassifier(bootstrap=True, ccp\_alpha=0.0, class\_weight=None,

criterion='gini', max\_depth=None, max\_features='auto',

max\_leaf\_nodes=None, max\_samples=None,

min\_impurity\_decrease=0.0, min\_impurity\_split=None,

min\_samples\_leaf=1, min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0, n\_estimators=100,

n\_jobs=None, oob\_score=False, random\_state=None,

verbose=0, warm\_start=False):

precision recall f1-score support

0 0.83 0.90 0.86 143

1 0.76 0.64 0.70 75

accuracy 0.81 218

macro avg 0.79 0.77 0.78 218

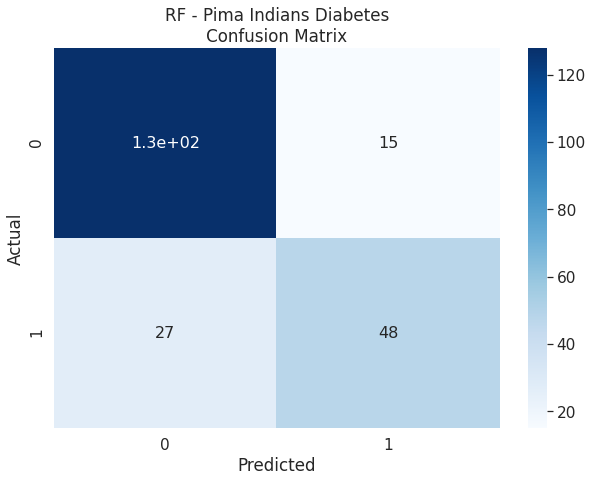
weighted avg 0.80 0.81 0.80 218

Accuracy score: 80.73394495412845

# Confusion Matrix

# Plot the confusion matrix

confmatrix(rf\_pred, "RF - Pima Indians Diabetes\nConfusion Matrix")

****

# Cross-Validation

# Perform cross-validation then get the mean

rf\_cv = np.mean(cross\_val\_score(rf, X, y, cv=10) \* 100)

print("10-Fold Cross-Validation score for KNN fit in Regular Training Set:", rf\_cv)

**output :** 10-Fold Cross-Validation score for KNN fit in Regular Training Set: 77.35350076103501

# Result

df\_results = pd.DataFrame.from\_dict({

    'Accuracy Score':{'Logistic Regression':logreg\_score,  'Random Forest':rf\_score},

    })

df\_results

**output :**

Accuracy Score

Logistic Regression 79.816514

Random Forest 80.733945

TOOLS:

* **LOGISTIC CLASSIFICATION ALGORITHM :**

Logistic Regression is a ‘Statistical Learning’ technique categorized in ‘Supervised’ Machine Learning (ML) methods dedicated to ‘Classification’ tasks. It has gained a tremendous reputation for last two decades especially in financial sector due to its prominent ability of detecting defaulters

* **RANDOM FOREST CLASSIFICATION** :

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of **ensemble learning,** which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

As the name suggests, ***"Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset."*** Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

**The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.**

ALGOTITHMS :

1. **LOGISTIC CLASSIFICATION ALGORITHM**
2. **RANDOM FOREST CLASSIFICATION**

CONCLUSION :

Within two methods, number 1 is more acceptable because it takes less computational power and gives us a much higher accuracy. But the Second method is easy to implement. By this two method, we can see that how we can train a model that can predict someone diabetes by taking some input feature like Glucose, BMI, Age.

For future work it is necessary to make a local dataset from hospital. Food habit of every country is different from each other. So it is important to gather a local dataset

Logistic classification algorithm calculate the accuracy - 79.816514

Random forest classification algorithm calculate the accuracy - 80.733945