**AI BASED DIABETIES PREDICTION SYSTEM**

Name: S.M.Jeens Ferroy

Reg.No: 511721106302

**Introduction:**

* Diabetes is a health condition that affects how your body turns food into energy. Most of the food you eat is broken down into sugar (also called glucose) and released into your bloodstream. When your blood sugar goes up, it signals your pancreas to release insulin.
* Without ongoing, careful management, diabetes can lead to a buildup of sugars in the blood, which can increase the risk of dangerous complications, including stroke and heart disease. So that i decide to predict using Machine Learning in Python.



**Installing Libraries**

In this first step I have imported most common libraries used in python for machine learning such as Pandas, Seaborn, Matplitlib etc.

I am using Python because if very flexible and effective programming language i ever used. I used Python in software development field too.

# Import libraries  
import numpy as np *# linear algebra*  
import pandas as pd *# data processing, CSV file I/O (e.g. pd.read\_csv)*  
import seaborn as sns *# for data visualization*  
import matplotlib.pyplot as plt *# to plot charts*  
from collections import Counter  
import os  
  
*# Modeling Libraries*  
from sklearn.preprocessing import QuantileTransformer  
from sklearn.metrics import confusion\_matrix, accuracy\_score, precision\_score  
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier, VotingClassifier  
from sklearn.linear\_model import LogisticRegression  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.svm import SVC  
from sklearn.model\_selection import GridSearchCV, cross\_val\_score, StratifiedKFold, learning\_curve, train\_test\_split

The sklearn library is very versatile and handy and serves real-world purposes. It provides wide range of ML algorithms and Models.

**Importing Data:**

In this project, i used Diabetes Database from Kaggle. This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases.

**Database Link: (**<https://www.kaggle.com/datasets/mathchi/diabetes-data-set/download?datasetVersionNumber=1>)

# Import dataset  
df = pd.read\_csv("../input/diabetes-database/diabetes.csv")

*# Get familier with dataset structure*  
df.info()

**Output:**

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

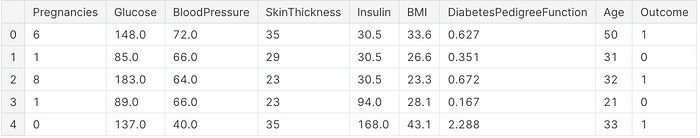
dtypes: float64(2), int64(7)

memory usage: 54.1 KB

Excepting BMI and DiabetesPedigreeFunction all the columns are integer. Outcome is the label containing 1 and 0 values. 1 means person has diabetes and 0 mean person is not diabetic.

*# Show top 5 rows*  
df.head()

**Output:**



**Missing Value Analysis**

Next, i will cleanup the dataset which is the important part of data science. Missing data can lead to wrong statistics during modeling and predictions.

*# Explore missing values*

df.isnull().sum()

**Output:**

Pregnancies 0

Glucose 5

BloodPressure 35

SkinThickness 227

Insulin 374

BMI 11

DiabetesPedigreeFunction 0

Age 0

Outcome 0

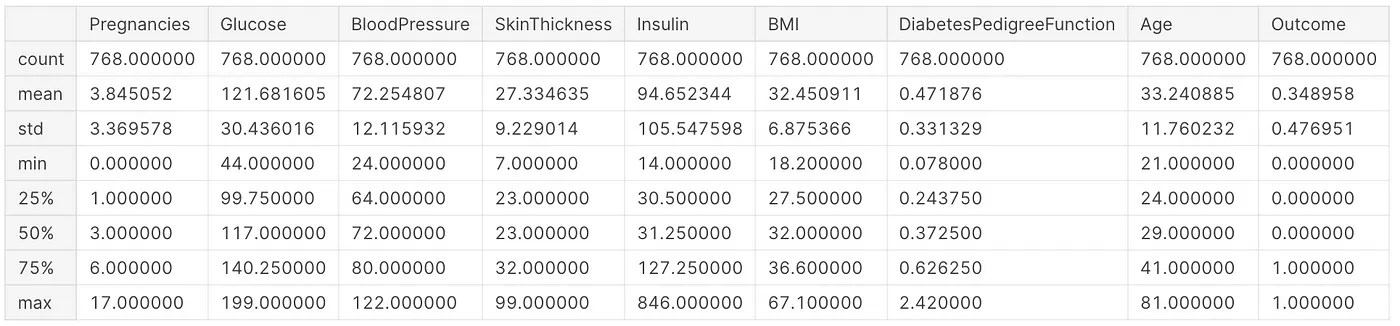
dtype: int64

I observed that there is no missing values in dataset however the features like Glucose, BloodPressure, Insulin, SkinThickness has 0 values which is not possible. We have to replace 0 values with either mean or median values of specific column.

df['Glucose'] = df['Glucose'].replace(0, df['Glucose'].mean())*# Correcting missing values in blood pressure*  
df['BloodPressure'] = df['BloodPressure'].replace(0, df['BloodPressure'].mean()) # There are 35 records with 0 BloodPressure in dataset*# Correcting missing values in BMI*  
df['BMI'] = df['BMI'].replace(0, df['BMI'].median())*# Correct missing values in Insulin and SkinThickness*  
  
df['SkinThickness'] = df['SkinThickness'].replace(0, df['SkinThickness'].median())  
df['Insulin'] = df['Insulin'].replace(0, df['Insulin'].median())

Now, lets review the dataset statistics

*# Review dataset statistics*  
df.describe()



Now i have clean dataset without missing values in features which is good.

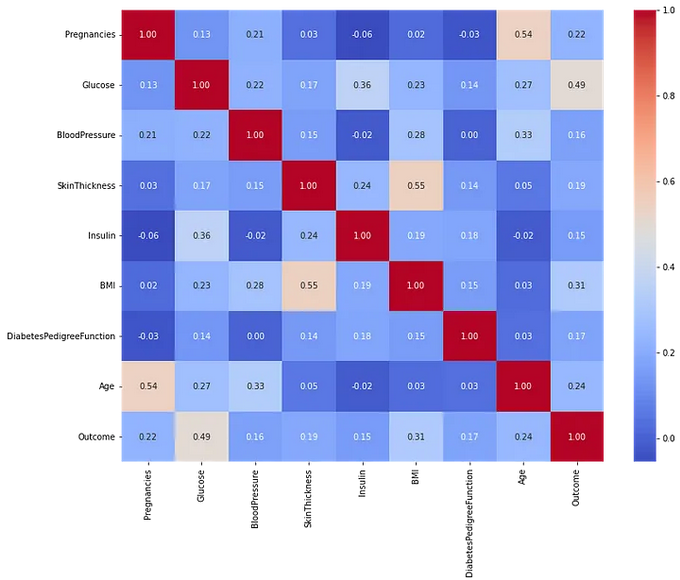
**Exploratory Data Analysis**

In this step, i showcased anlytics using GUI using Seaborn.

**Correlation**

Correlation is one or more variables are related to each other. It also helps to find the feature importance and clean the dataset before i start Modeling

plt.figure(figsize=(13,10))  
sns.heatmap(df.corr(),annot=True, fmt = ".2f", cmap = "coolwarm")

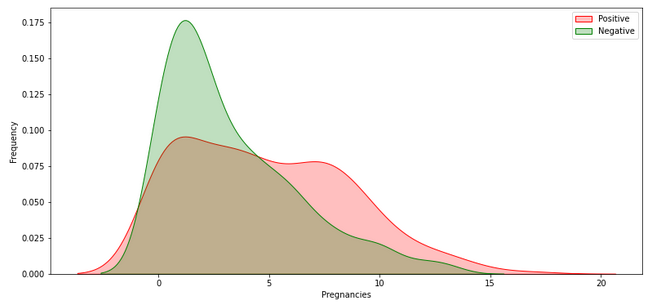


According to observation, features like Pregnancies, Gluecose, BMI, and Age is more correlated with Outcome. In next steps, i showcased details representation of these features**.**

**Pregnancies**

Women with diabetes can and do have healthy pregnancies and healthy babies. Managing diabetes can help reduce your risk for complications. Untreated diabetes increases your risk for pregnancy complications, like high blood pressure, depression, premature birth, birth defects and pregnancy loss.

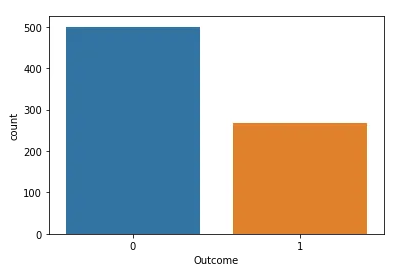
*# Explore* Pregnancies *vs Outcome*plt.figure(figsize=(13,6))  
g = sns.kdeplot(df["Pregnancies"][df["Outcome"] == 1],   
 color="Red", shade = True)  
g = sns.kdeplot(df["Pregnancies"][df["Outcome"] == 0],   
 ax =g, color="Green", shade= True)g.set\_xlabel("Pregnancies")  
g.set\_ylabel("Frequency")  
g.legend(["Positive","Negative"])



**Outcome**

Outcome has 1 and 0 values where 1 indicates that person has diabetes and 0 shows person has no diabetes. This is my label column in dataset.

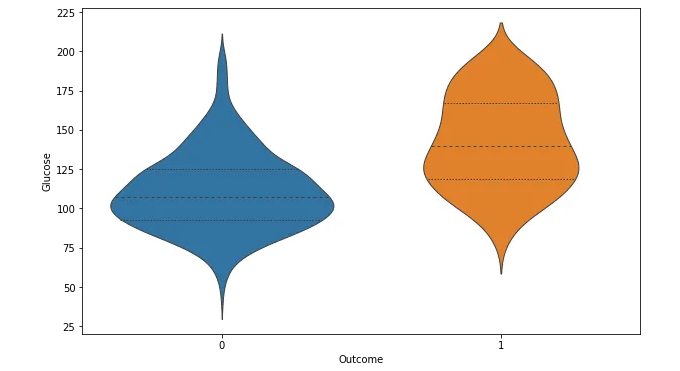
sns.countplot('Outcome', data = df)



It indicates, There are more people who do not have diabetes in dataset which is around 65% and 35% people has diabetes.

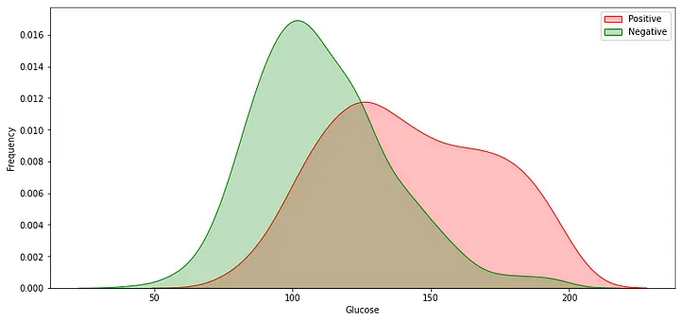
**Glucose**

# Explore Gluecose vs Outcomeplt.figure(figsize=(10,6))  
sns.violinplot(data=df, x="Outcome", y="Glucose",  
 split=True, inner="quart", linewidth=1)



The chances of diabetes is gradually increasing with level of Glucose.

*# Explore Glucose vs Outcome*  
  
plt.figure(figsize=(13,6))  
g = sns.kdeplot(df["Glucose"][df["Outcome"] == 1], color="Red", shade = True)  
g = sns.kdeplot(df["Glucose"][df["Outcome"] == 0], ax =g, color="Green", shade= True)  
g.set\_xlabel("Glucose")  
g.set\_ylabel("Frequency")  
g.legend(["Positive","Negative"])



**Feature Engineering**

Till now, i explored the dataset, did missing value corrections and data visualization. Next, i have started feature engineering. Feature engineering is useful to improve the performance of machine learning algorithms and is often considered as applied machine learning.

Selecting the important features and reducing the size of the feature set makes computation in machine learning and data analytic algorithms more feasible.

**Outlier Detection**

In this part i removed all the records outlined in dataset. Outliers impacts Model accuracy. I used Tukey Method used for outlier detection.

def detect\_outliers(df,n,features):  
 outlier\_indices = []  
 *"""*  
 *Detect outliers from given list of features. It returns a list of the indices*  
 *according to the observations containing more than n outliers according*  
 *to the Tukey method*  
 *"""*  
 *# iterate over features(columns)*  
 for col **in** features:  
 Q1 = np.percentile(df[col], 25)  
 Q3 = np.percentile(df[col],75)  
 IQR = Q3 - Q1  
   
 *# outlier step*  
 outlier\_step = 1.5 \* IQR  
   
 *# Determine a list of indices of outliers for feature col*  
 outlier\_list\_col = df[(df[col] < Q1 - outlier\_step) | (df[col] > Q3 + outlier\_step )].index  
   
 *# append the found outlier indices for col to the list of outlier indices*   
 outlier\_indices.extend(outlier\_list\_col)  
   
 *# select observations containing more than 2 outliers*  
 outlier\_indices = Counter(outlier\_indices)  
 multiple\_outliers = list( k for k, v **in** outlier\_indices.items() if v > n )  
   
 return multiple\_outliers   
  
*# detect outliers from numeric features*  
outliers\_to\_drop = detect\_outliers(df, 2 ,["Pregnancies", 'Glucose', 'BloodPressure', 'BMI', 'DiabetesPedigreeFunction', 'SkinThickness', 'Insulin', 'Age'])

Here, I find outliers from all the features such as Pregnancies, Glucose, BloodPressure, BMI, DiabetesPedigreeFunction, SkinThickness, Insulin, and Age.

df.drop(df.loc[outliers\_to\_drop].index, inplace=True)

I have successfully removed all outliers from dataset now. The next step is to split the dataset in train and test and proceed the modeling.

**Modeling**

In this sections, i tried different models and compare the accuracy for each. Then, i performed Hyperparameter Tuning on Models that has high accuracy.

Before i split the dataset i need to transform the data into quantile using

sklearn.preprocessing .

# Data Transformation  
q = QuantileTransformer()  
X = q.fit\_transform(df)  
transformedDF = q.transform(X)  
transformedDF = pd.DataFrame(X)  
transformedDF.columns =['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']# Show top 5 rows  
transformedDF.head()



**Data Splitting**

Next, i split data in test and train dataset. Train dataset will be used in Model training and evaluation and test dataset will be used in prediction. Before i predict the test data, i performed cross validation for various models.

features = df.drop(["Outcome"], axis=1)  
labels = df["Outcome"]x\_train, x\_test, y\_train, y\_test = train\_test\_split(features, labels, test\_size=0.30, random\_state=7)

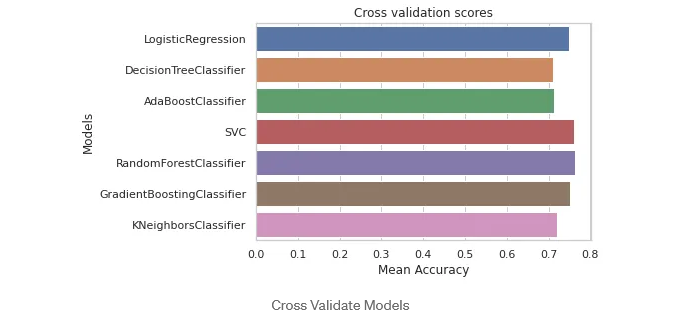
Above code splits dataset into train (70%) and test (30%) dataset.

**Cross Validate Models**

def evaluate\_model(models):  
 *"""*  
 *Takes a list of models and returns chart of cross validation scores using mean accuracy*  
 *"""*  
   
 *# Cross validate model with Kfold stratified cross val*  
 kfold = StratifiedKFold(n\_splits = 10)  
   
 result = []  
 for model **in** models :  
 result.append(cross\_val\_score(estimator = model, X = x\_train, y = y\_train, scoring = "accuracy", cv = kfold, n\_jobs=4))  
  
 cv\_means = []  
 cv\_std = []  
 for cv\_result **in** result:  
 cv\_means.append(cv\_result.mean())  
 cv\_std.append(cv\_result.std())  
  
 result\_df = pd.DataFrame({  
 "CrossValMeans":cv\_means,  
 "CrossValerrors": cv\_std,  
 "Models":[  
 "LogisticRegression",  
 "DecisionTreeClassifier",  
 "AdaBoostClassifier",  
 "SVC",  
 "RandomForestClassifier",  
 "GradientBoostingClassifier",  
 "KNeighborsClassifier"  
 ]  
 })  
  
 *# Generate chart*  
 bar = sns.barplot(x = "CrossValMeans", y = "Models", data = result\_df, orient = "h")  
 bar.set\_xlabel("Mean Accuracy")  
 bar.set\_title("Cross validation scores")  
 return result\_df

Method `evaluate\_model` takes a list of models and returns chart of cross validation scores using mean accuracy.

*# Modeling step Test differents algorithms*   
random\_state = 30  
models = [  
 LogisticRegression(random\_state = random\_state, solver='liblinear'),  
 DecisionTreeClassifier(random\_state = random\_state),  
 AdaBoostClassifier(DecisionTreeClassifier(random\_state = random\_state), random\_state = random\_state, learning\_rate = 0.2),  
 SVC(random\_state = random\_state),  
 RandomForestClassifier(random\_state = random\_state),  
 GradientBoostingClassifier(random\_state = random\_state),  
 KNeighborsClassifier(),  
]  
evaluate\_model(models)



As per above observation, i found that SVC, RandomForestClassifier, and LogisticRegression model has more accuracy. Next, i will do hyper parameter tuning on three models.

**Hyperparameter Tuning**

Hyperparameter tuning is choosing a set of optimal hyperparameters for a learning algorithm. A hyperparameter is a model argument whose value is set before the learning process begins. The key to machine learning algorithms is hyperparameter tuning.

I have done tuning process for SVC, RandomForestClassifier, and LogisticRegression models one by one.

# Import libraries  
from sklearn.model\_selection import GridSearchCV  
from sklearn.metrics import classification\_reportdef analyze\_grid\_result(grid\_result):  
 '''  
 Analysis of GridCV result and predicting with test dataset  
 Show classification report at last  
 ''' # Best parameters and accuracy  
 print("Tuned hyperparameters: (best parameters) ", grid\_result.best\_params\_)  
 print("Accuracy :", grid\_result.best\_score\_)  
   
 means = grid\_result.cv\_results\_["mean\_test\_score"]  
 stds = grid\_result.cv\_results\_["std\_test\_score"]  
 for mean, std, params in zip(means, stds, grid\_result.cv\_results\_["params"]):  
 print("%0.3f (+/-%0.03f) for %r" % (mean, std \* 2, params))  
 print() print("Detailed classification report:")  
 y\_true, y\_pred = y\_test, grid\_result.predict(x\_test)  
 print(classification\_report(y\_true, y\_pred))  
 print()

First of all i have imported GridSearchCV and classification\_report from sklearn library. Then, i have defined `analyze\_grid\_result` method which will show prediction result. I called this method for each Model used in SearchCV. In next step, i will perform tuning for each model.

**Logistic Regression**

# Define models and parameters for LogisticRegression  
model = LogisticRegression(solver='liblinear')  
solvers = ['newton-cg', 'liblinear']  
penalty = ['l2']  
c\_values = [100, 10, 1.0, 0.1, 0.01]# Define grid search  
grid = dict(solver = solvers, penalty = penalty, C = c\_values)  
cv = StratifiedKFold(n\_splits = 50, random\_state = 1, shuffle = True)  
grid\_search = GridSearchCV(estimator = model, param\_grid = grid, cv = cv, scoring = 'accuracy', error\_score = 0)  
logi\_result = grid\_search.fit(x\_train, y\_train)# Logistic Regression Hyperparameter Result  
analyze\_grid\_result(logi\_result)

Output:

Tuned hyperparameters: (best parameters) {'C': 10, 'penalty': 'l2', 'solver': 'liblinear'}  
Accuracy : 0.7883636363636363  
Detailed classification report:  
  
 precision recall f1-score support  
  
 0 0.78 0.84 0.81 147  
 1 0.68 0.58 0.62 83  
  
 accuracy 0.75 230  
 macro avg 0.73 0.71 0.72 230  
weighted avg 0.74 0.75 0.74 230

As per my observation, in LogisticRegression it returned best score 0.78 with `{‘C’: 10, ‘penalty’: ‘l2’, ‘solver’: ‘liblinear’}` parameters. Next i will perform tuning for other models.

**SVC**

# Define models and parameters for LogisticRegression  
model = SVC()# Define grid search  
tuned\_parameters = [  
 {"kernel": ["rbf"], "gamma": [1e-3, 1e-4], "C": [1, 10, 100, 1000]},  
 {"kernel": ["linear"], "C": [1, 10, 100, 1000]},  
]  
cv = StratifiedKFold(n\_splits = 2, random\_state = 1, shuffle = True)  
grid\_search = GridSearchCV(estimator = model, param\_grid = tuned\_parameters, cv = cv, scoring = 'accuracy', error\_score = 0)  
scv\_result = grid\_search.fit(x\_train, y\_train)# SVC Hyperparameter Result  
analyze\_grid\_result(scv\_result)

Output:

Tuned hyperparameters: (best parameters) {'C': 10, 'kernel': 'linear'}  
Accuracy : 0.7775797976410084  
Detailed classification report:  
  
 precision recall f1-score support  
  
 0 0.78 0.84 0.81 147  
 1 0.67 0.57 0.61 83  
  
 accuracy 0.74 230  
 macro avg 0.72 0.70 0.71 230  
weighted avg 0.74 0.74 0.74 230

SVC Model gave max 0.77 accuracy which is bit less than LogisticRegression. I will not use this model anymore.

**Random Forest Classifier**

# Define models and parameters for LogisticRegression  
model = RandomForestClassifier(random\_state=42)# Define grid search  
tuned\_parameters = {   
 'n\_estimators': [200, 500],  
 'max\_features': ['auto', 'sqrt', 'log2'],  
 'max\_depth' : [4,5,6,7,8],  
 'criterion' :['gini', 'entropy']  
}  
cv = StratifiedKFold(n\_splits = 2, random\_state = 1, shuffle = True)  
grid\_search = GridSearchCV(estimator = model, param\_grid = tuned\_parameters, cv = cv, scoring = 'accuracy', error\_score = 0)  
grid\_result = grid\_search.fit(x\_train, y\_train)# SVC Hyperparameter Result  
analyze\_grid\_result(grid\_result)

Output:

Tuned hyperparameters: (best parameters) {'criterion': 'entropy', 'max\_depth': 5, 'max\_features': 'log2', 'n\_estimators': 200}  
Accuracy : 0.7663648051875454  
Detailed classification report:  
  
 precision recall f1-score support  
  
 0 0.78 0.83 0.80 147  
 1 0.66 0.58 0.62 83  
  
 accuracy 0.74 230  
 macro avg 0.72 0.70 0.71 230  
weighted avg 0.73 0.74 0.74 230

Randomforest model gave max 0.76% accuracy which is not best comparing to other model. So i decided to use LogisticRegression Model for prediction.

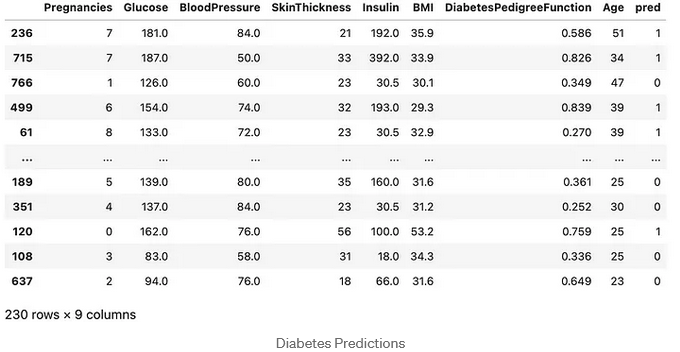
**Prediction**

Till now, i worked on EDA, Feature Engineering, Cross Validation of Models, and Hyperparameter Tuning and find the best working Model for my dataset. Next, I did prediction from my test dataset and storing the result in CSV.

# Test predictions  
y\_pred = logi\_result.predict(x\_test)  
print(classification\_report(y\_test, y\_pred))  
# output  
 precision recall f1-score support  
  
 0 0.78 0.84 0.81 147  
 1 0.68 0.58 0.62 83  
  
 accuracy 0.75 230  
 macro avg 0.73 0.71 0.72 230  
weighted avg 0.74 0.75 0.74 230

Finally append new feature column in test dataset called Prediction and print the dataset.

x\_test['pred'] = y\_pred  
print(x\_test)



I will perform feature importance in separate article for more understanding the impact of feature after modeling.

**Conclusion**

* Diabetes is one of the ricks during Pregnancy. It has to be treat to avoid complications.
* BMI index can help to avoid complications of diabetes a way before
* Diabetes start showing in age of 35 – 40 and increase with person age.