2022

Pima Indians Diabetes Prediction

Diabetes Prediction Using AI

Prepared by,
Sugina Surendran MC
suginaponnu@gmail.com

INTRODUCTION

Diabetes mellitus, commonly referred to simply as diabetes, is a metabolic disease that causes high blood. The hormone insulin moves sugar from the blood into your cells to be stored or used for energy. With diabetes, your body either doesn't make enough insulin or can't effectively use the insulin it does make.

This project intends to analyze the PIMA Indian Diabetes dataset and create a model to predict a particular observation is at a risk of developing diabetes, And this project includes the methods followed by, Feature Engineering, Processing, and Exploratory Data Analysis(EDA) to build an effective model.

Steps included in the ML Project life Cycle

Step 1: Data Preparation

Step 2: Model Building

Step 3: Building the app using Flask

Step 4: HTML

Step 5: Code Push to GitHub

Step 6: Deploying the app using Heroku

UI For the App: https://diabeticheck.herokuapp.com

Pima Indians Diabetes Database

https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database

Dataset Details

1: Pregnancies: Number of times pregnant

2: Glucose: Plasma glucose concentration a 2hours in an oral glucose tolerance test

3: BloodPressure: Diastolic blood pressure (mm Hg)

4: SkinThickness: Triceps skinfold thickness (mm)

5: Insulin: 2-Hour serum insulin (mu U/ml)

6: BMI: Body mass index (weight in kg/(height in m)^2)

7: DiabetesPedigreeFunction: Diabetes pedigree function

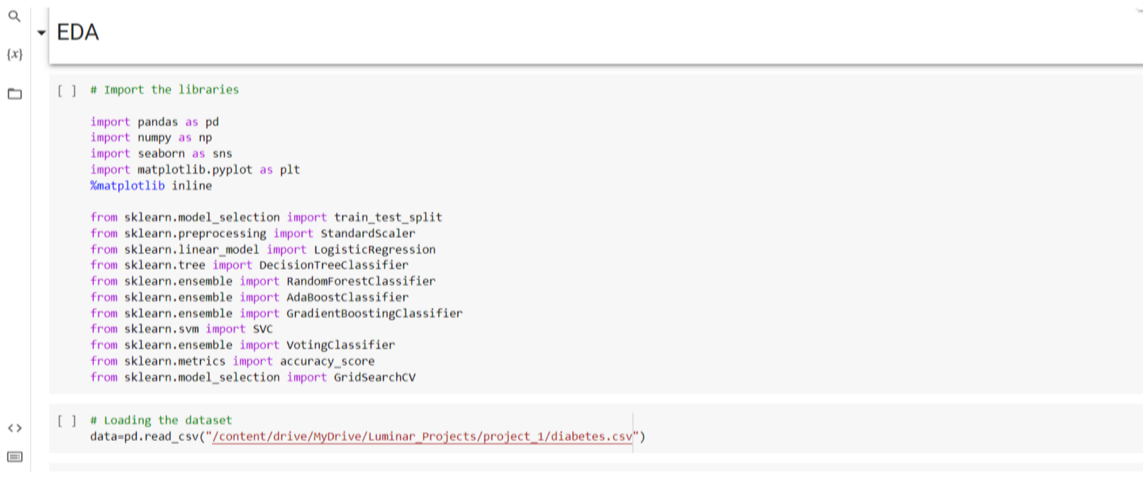
8: Age: Age (years)

9: Outcome: Class variable (0 or 1) 268 of 768 are 1, the others are 0

Step-1: Data Preparation

i. EDA

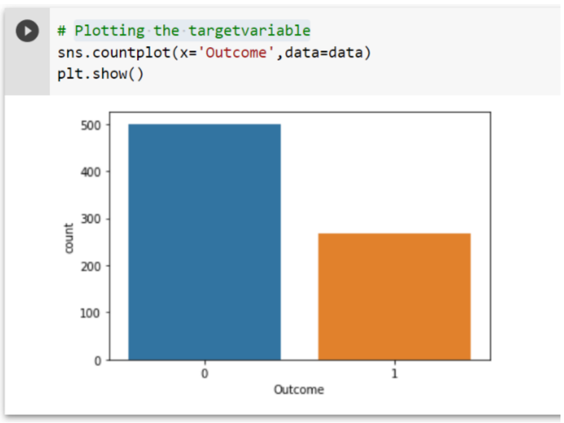
- Import libraries and loading dataset



```
# Import the libraries

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV
```

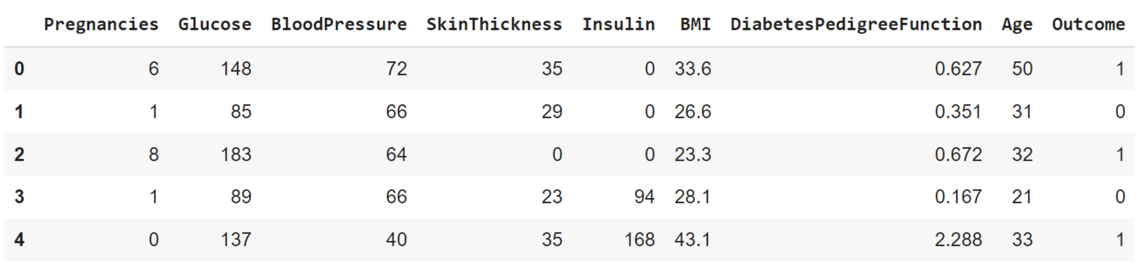
```
# Loading the dataset
data=pd.read_csv("/content/drive/MyDrive/Luminar_Projects/project_1/diabetes.csv")
```

- Checking the unique values in the Target variable
- Plotting the Target variable



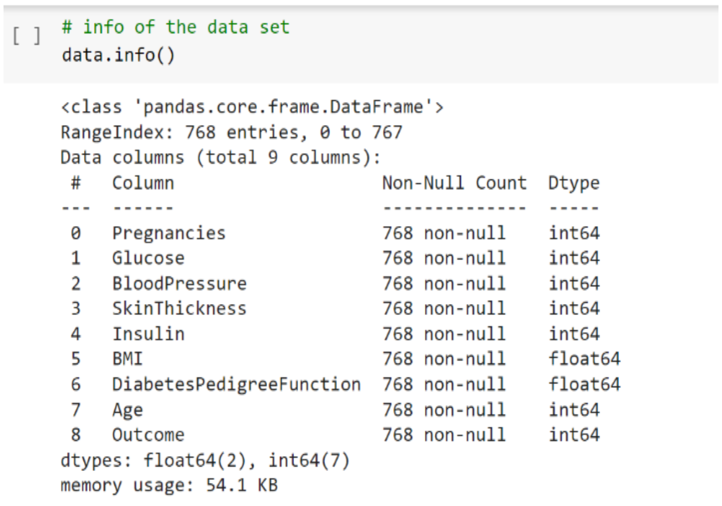
```
# Plotting the targetvariable
sns.countplot(x='Outcome',data=data)
plt.show()
```

- Preview Data- head(),tail(),info(),describe()..



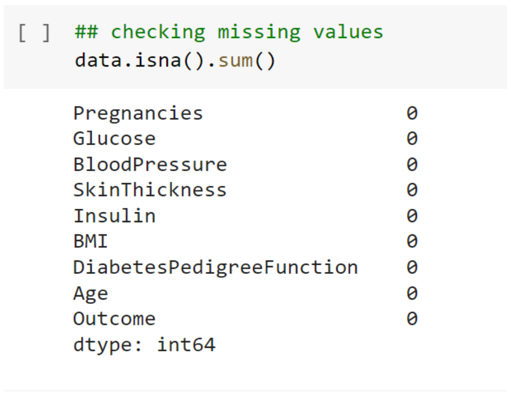
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

- Checking total number of entries and column types



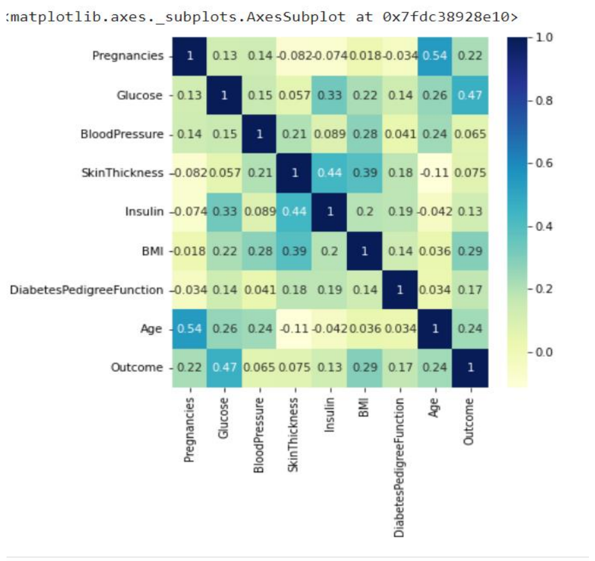
```
[ ]  # info of the data set
     data.info()

     <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
```

- Checking missing values



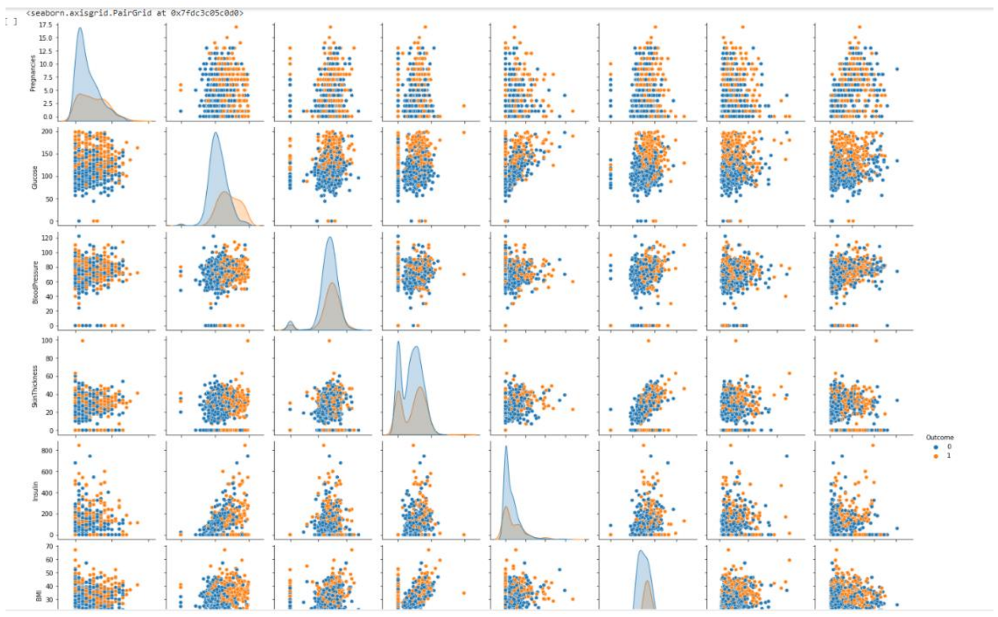
```
[ ]  ## checking missing values
     data.isna().sum()

     Pregnancies                 0
     Glucose                     0
     BloodPressure               0
     SkinThickness               0
     Insulin                     0
     BMI                         0
     DiabetesPedigreeFunction    0
     Age                         0
     Outcome                     0
     dtype: int64
```

- Checking outlier values(Box-plot)
- Correlation of data based on target values
- Correlation map using heat-map



```
:matplotlib.axes._subplots.AxesSubplot at 0x7fdc38928e10>
```

- Drawing Pair-plot(seaborn)

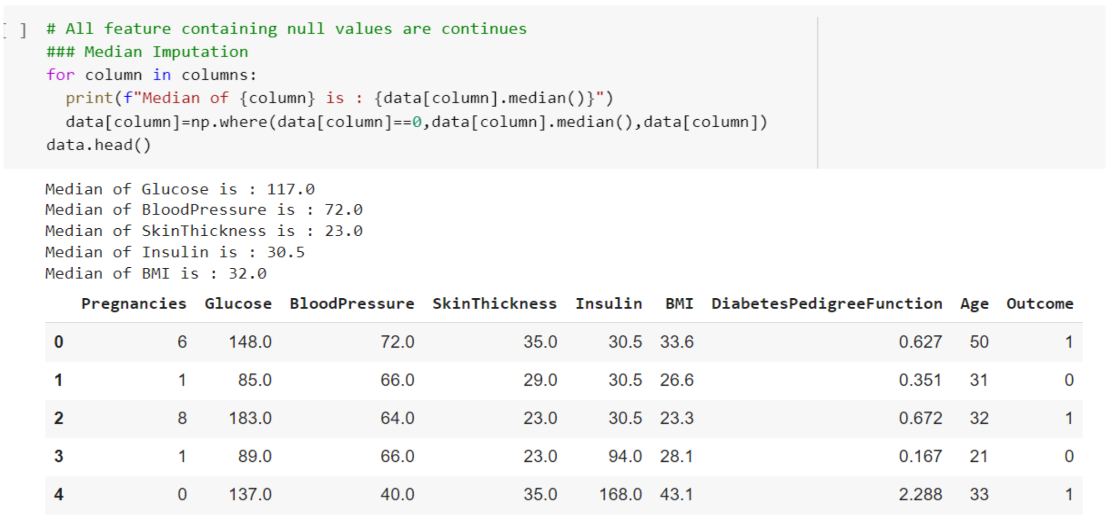


```
<seaborn.axisgrid.PairGrid at 0x7fdc3c05c0d0>
```

ii. <u>Missing Value Imputation (Median Imputation)</u>

- Here missing values are present in the form of number 0
- columns which zero value present

  columns=['Glucose','BloodPressure','SkinThickness','Insulin','BMI']
- <u>Median imputation</u> (All columns are continuous)

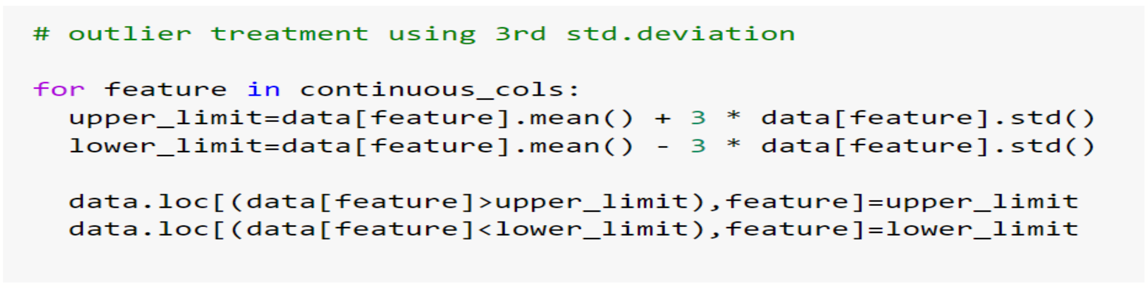


```
[ ]  # All feature containing null values are continues
     ### Median Imputation
     for column in columns:
       print(f"Median of {column} is : {data[column].median()}")
       data[column]=np.where(data[column]==0,data[column].median(),data[column])
     data.head()
```

```
Median of Glucose is : 117.0
Median of BloodPressure is : 72.0
Median of SkinThickness is : 23.0
Median of Insulin is : 30.5
Median of BMI is : 32.0
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.0	72.0	35.0	30.5	33.6	0.627	50	1
1	1	85.0	66.0	29.0	30.5	26.6	0.351	31	0
2	8	183.0	64.0	23.0	30.5	23.3	0.672	32	1
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21	0
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33	1

iii. <u>Outlier Treatment (using Third Standard Deviation)</u>



```
# outlier treatment using 3rd std.deviation

for feature in continuous_cols:
  upper_limit=data[feature].mean() + 3 * data[feature].std()
  lower_limit=data[feature].mean() - 3 * data[feature].std()

  data.loc[(data[feature]>upper_limit),feature]=upper_limit
  data.loc[(data[feature]<lower_limit),feature]=lower_limit
```

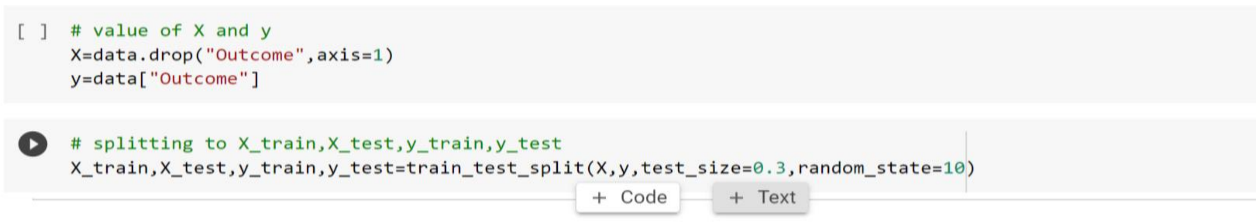
iv. <u>Handling Imbalance Data (SMOTE)</u>

- Model performance is less after doing smote. We choose the dataset without smote

v. <u>Categorical Encoding</u>

- No categorical variable in the Dataset

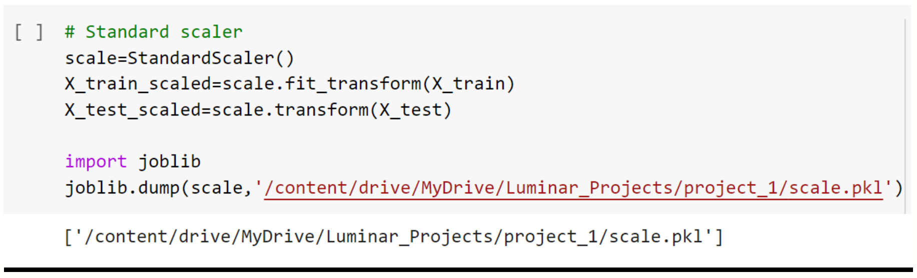
vi. <u>Train Test Split</u>



```
[ ]  # value of X and y
     X=data.drop("Outcome",axis=1)
     y=data["Outcome"]
```

```
     # splitting to X_train,X_test,y_train,y_test
     X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=10)
```

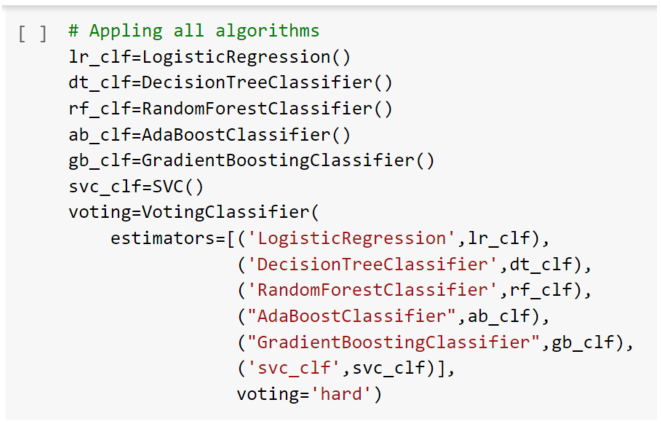
vii. Scaling Down (Standard Scaler)



```
[ ]  # Standard scaler
     scale=StandardScaler()
     X_train_scaled=scale.fit_transform(X_train)
     X_test_scaled=scale.transform(X_test)

     import joblib
     joblib.dump(scale,'/content/drive/MyDrive/Luminar_Projects/project_1/scale.pkl')

     ['/content/drive/MyDrive/Luminar_Projects/project_1/scale.pkl']
```

Step-2: Model Building

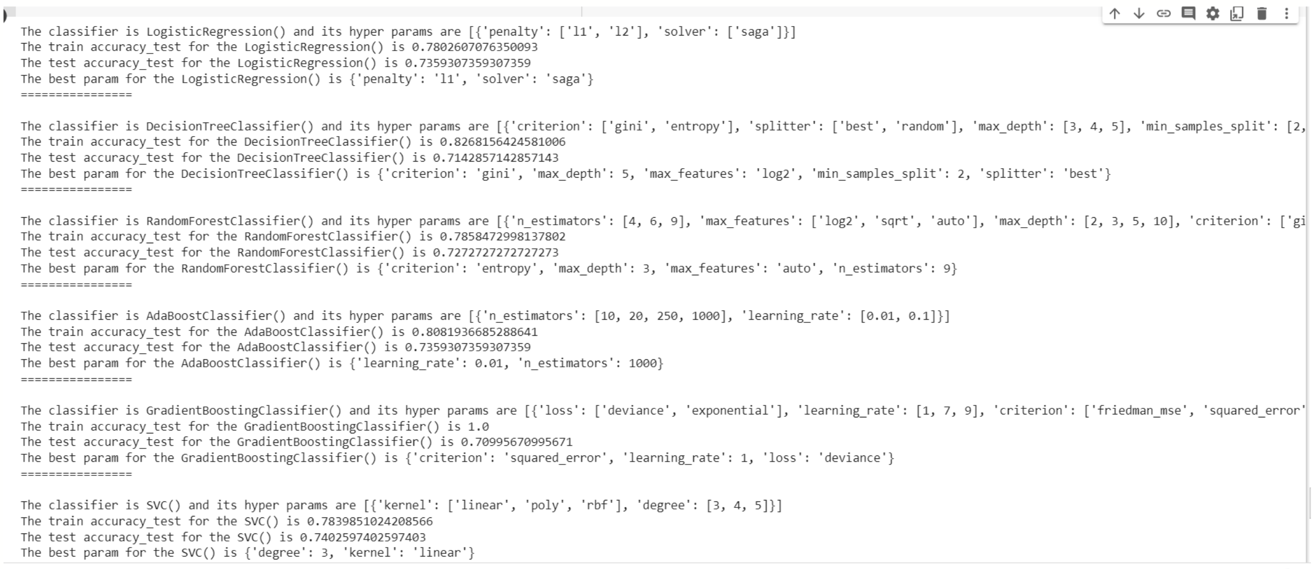
i. Applying classification algorithms in Voting Classifier



```
[ ]  # Appling all algorithms
     lr_clf=LogisticRegression()
     dt_clf=DecisionTreeClassifier()
     rf_clf=RandomForestClassifier()
     ab_clf=AdaBoostClassifier()
     gb_clf=GradientBoostingClassifier()
     svc_clf=SVC()
     voting=VotingClassifier(
         estimators=[('LogisticRegression',lr_clf),
                     ('DecisionTreeClassifier',dt_clf),
                     ('RandomForestClassifier',rf_clf),
                     ("AdaBoostClassifier",ab_clf),
                     ("GradientBoostingClassifier",gb_clf),
                     ('svc_clf',svc_clf)],
                     voting='hard')
```

ii. Fitting the Algorithms

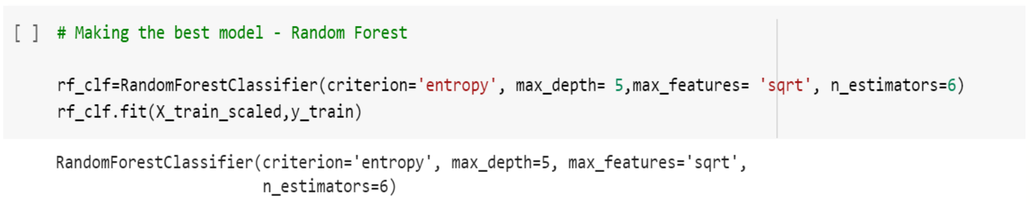
iii. Hyperparameter Tuning



```
The classifier is LogisticRegression() and its hyper params are [{'penalty': ['l1', 'l2'], 'solver': ['saga']}]
The train accuracy_test for the LogisticRegression() is 0.7802607076350093
The test accuracy_test for the LogisticRegression() is 0.7359307359307359
The best param for the LogisticRegression() is {'penalty': 'l1', 'solver': 'saga'}
================

The classifier is DecisionTreeClassifier() and its hyper params are [{'criterion': ['gini', 'entropy'], 'splitter': ['best', 'random'], 'max_depth': [3, 4, 5], 'min_samples_split': [2,
The train accuracy_test for the DecisionTreeClassifier() is 0.8268156424581006
The test accuracy_test for the DecisionTreeClassifier() is 0.7142857142857143
The best param for the DecisionTreeClassifier() is {'criterion': 'gini', 'max_depth': 5, 'max_features': 'log2', 'min_samples_split': 2, 'splitter': 'best'}
================

The classifier is RandomForestClassifier() and its hyper params are [{'n_estimators': [4, 6, 9], 'max_features': ['log2', 'sqrt', 'auto'], 'max_depth': [2, 3, 5, 10], 'criterion': ['gi
The train accuracy_test for the RandomForestClassifier() is 0.7858472998137802
The test accuracy_test for the RandomForestClassifier() is 0.7272727272727273
The best param for the RandomForestClassifier() is {'criterion': 'entropy', 'max_depth': 3, 'max_features': 'auto', 'n_estimators': 9}
================

The classifier is AdaBoostClassifier() and its hyper params are [{'n_estimators': [10, 20, 250, 1000], 'learning_rate': [0.01, 0.1]}]
The train accuracy_test for the AdaBoostClassifier() is 0.8081936685288641
The test accuracy_test for the AdaBoostClassifier() is 0.7359307359307359
The best param for the AdaBoostClassifier() is {'learning_rate': 0.01, 'n_estimators': 1000}
================

The classifier is GradientBoostingClassifier() and its hyper params are [{'loss': ['deviance', 'exponential'], 'learning_rate': [1, 7, 9], 'criterion': ['friedman_mse', 'squared_error'
The train accuracy_test for the GradientBoostingClassifier() is 1.0
The test accuracy_test for the GradientBoostingClassifier() is 0.70995670995671
The best param for the GradientBoostingClassifier() is {'criterion': 'squared_error', 'learning_rate': 1, 'loss': 'deviance'}
================

The classifier is SVC() and its hyper params are [{'kernel': ['linear', 'poly', 'rbf'], 'degree': [3, 4, 5]}]
The train accuracy_test for the SVC() is 0.7839851024208566
The test accuracy_test for the SVC() is 0.7402597402597403
The best param for the SVC() is {'degree': 3, 'kernel': 'linear'}
```

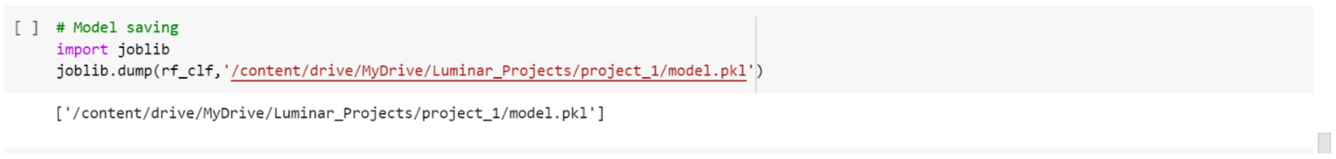
- Random Forest Classifier gives better accuracy

iv. Making The Best Model (Random Forest)



```
[ ] # Making the best model - Random Forest

    rf_clf=RandomForestClassifier(criterion='entropy', max_depth= 5,max_features= 'sqrt', n_estimators=6)
    rf_clf.fit(X_train_scaled,y_train)

    RandomForestClassifier(criterion='entropy', max_depth=5, max_features='sqrt',
                           n_estimators=6)
```

v. Model Saving



```
[ ] # Model saving
    import joblib
    joblib.dump(rf_clf,'/content/drive/MyDrive/Luminar_Projects/project_1/model.pkl')

    ['/content/drive/MyDrive/Luminar_Projects/project_1/model.pkl']
```

Step-3: Building the app using Flask

Codes: https://github.com/Sugina99/Pima_Indians_Diabetes_Project.git

Step-4: HTML

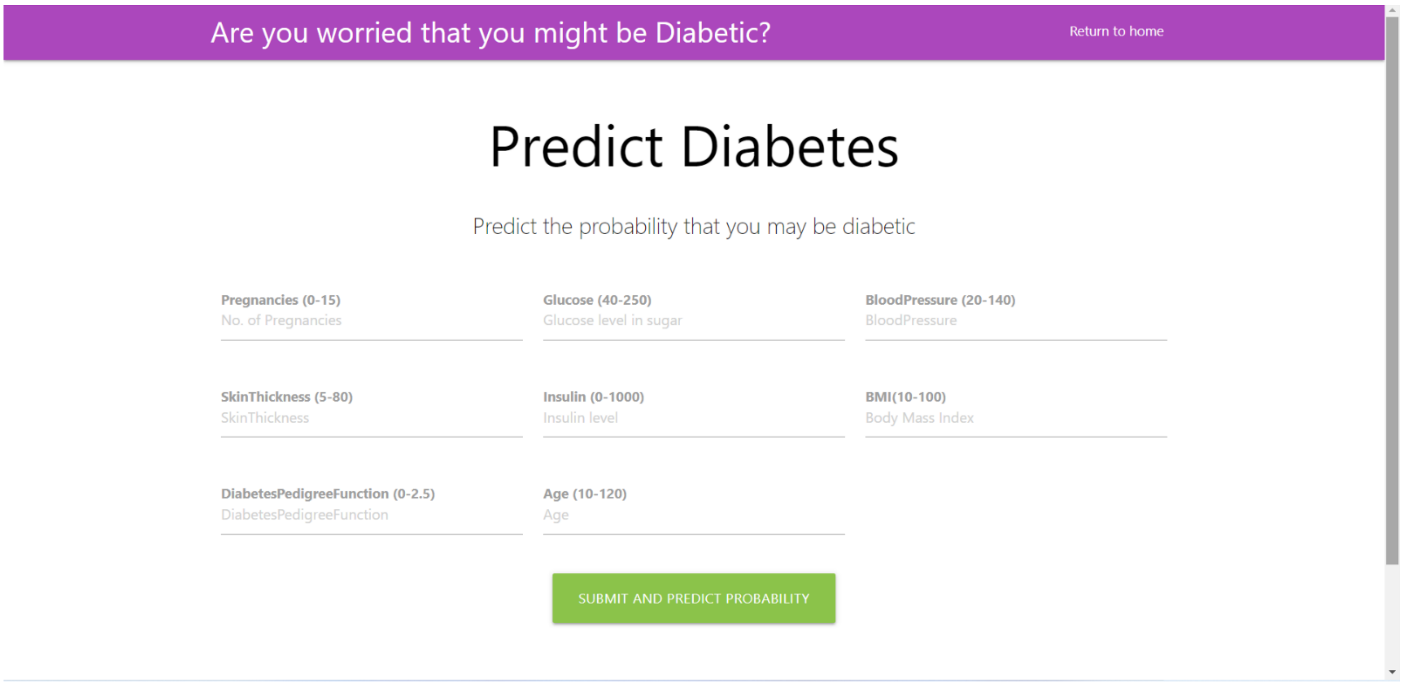
Codes: https://github.com/Sugina99/Pima_Indians_Diabetes_Project.git

Step 5: Code push to GitHub

Codes: https://github.com/Sugina99/Pima_Indians_Diabetes_Project.git

Step 6: Deploying the app using Heroku

CONCLUSION



Are you worried that you might be Diabetic?

Return to home

Predict Diabetes

Predict the probability that you may be diabetic

Pregnancies (0-15) No. of Pregnancies	Glucose (40-250) Glucose level in sugar	BloodPressure (20-140) BloodPressure
SkinThickness (5-80) SkinThickness	Insulin (0-1000) Insulin level	BMI(10-100) Body Mass Index
DiabetesPedigreeFunction (0-2.5) DiabetesPedigreeFunction	Age (10-120) Age	

SUBMIT AND PREDICT PROBABILITY