Data Science Capstone project – Healthcare

Problem Statement:

NIDDK (National Institute of Diabetes and Digestive and Kidney Diseases) research creates knowledge about and treatments for the most chronic, costly, and consequential diseases.

•The dataset used in this project is originally from NIDDK. The objective is to predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset.

•Build a model to accurately predict whether the patients in the dataset have diabetes or not.

Dataset Description

The datasets consists of several medical predictor variables and one target variable (Outcome). Predictor variables includes the number of pregnancies the patient has had, their BMI, insulin level, age, and more.

Variables	Description
Pregnancies	Number of times pregnant
Glucose	Plasma glucose concentration in an oral glucose tolerance test
Blood Pressure	Diastolic blood pressure (mm Hg)
Skin Thickness	Triceps skin fold thickness (mm)
Insulin	Two hour serum insulin
BMI	Body Mass Index
DiabetesPedigreeFunction	Diabetes pedigree function
Age	Age in years
Outcome	Class variable (either 0 or 1). 268 of 768 values are 1, and the others are 0

Solution

1. Data Exploration

- Perform descriptive analysis. Understand the variables and their corresponding values. On the columns below, a value of zero does not make sense and thus indicates missing value:

  •Glucose

  •BloodPressure

  •SkinThickness

  •Insulin

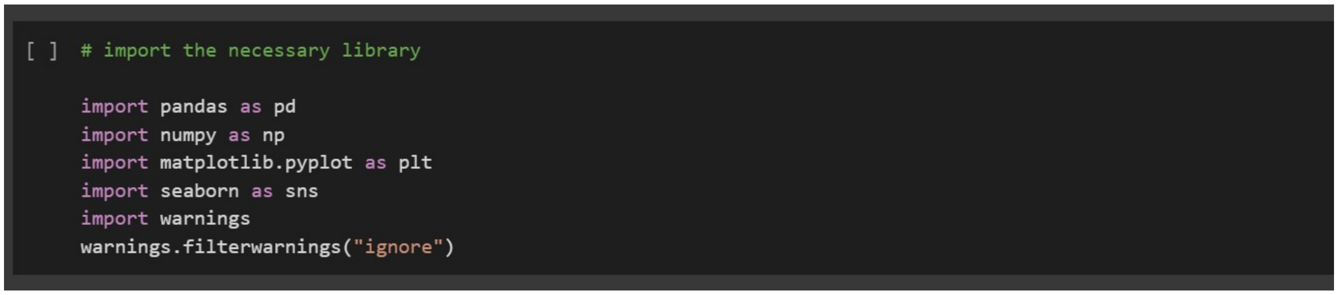
  •BMI

•Visually explore these variables using histograms. Treat the missing values accordingly.

•There are integer and float data type variables in this dataset. Create a count (frequency) plot describing the data types and the count of variables.

1.1 Import the necessary library

We'll import the necessary library

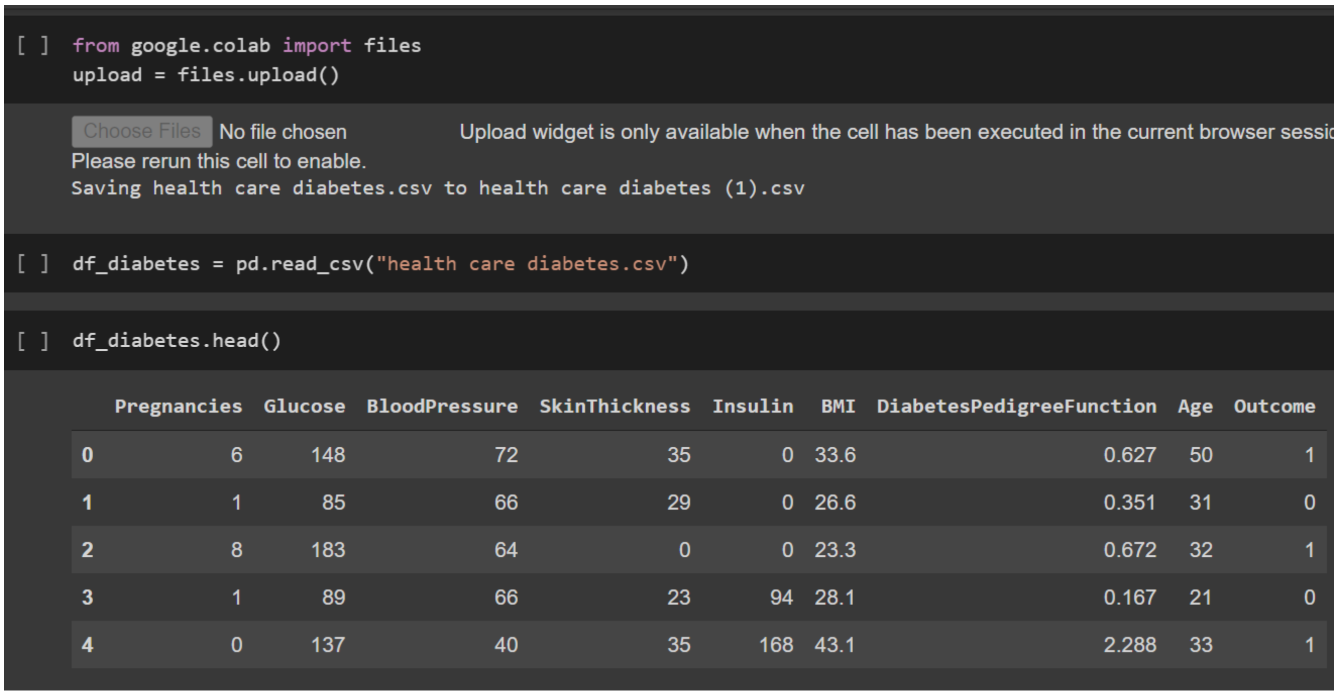


```
[ ]  # import the necessary library

     import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     import seaborn as sns
     import warnings
     warnings.filterwarnings("ignore")
```

we'll import more library as required in the subsequent section

1.2 Importing the required dataset

We'll import the "Health care diabetes" dataset in Google colab



```
[ ] from google.colab import files
    upload = files.upload()
```

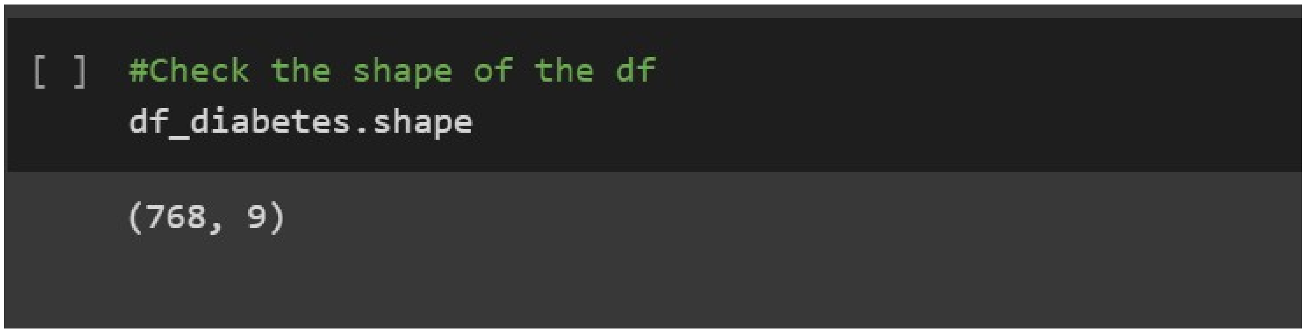
```
Upload widget is only available when the cell has been executed in the current browser sessi
Please rerun this cell to enable.
Saving health care diabetes.csv to health care diabetes (1).csv
```

```
[ ] df_diabetes = pd.read_csv("health care diabetes.csv")
```

```
[ ] df_diabetes.head()
```

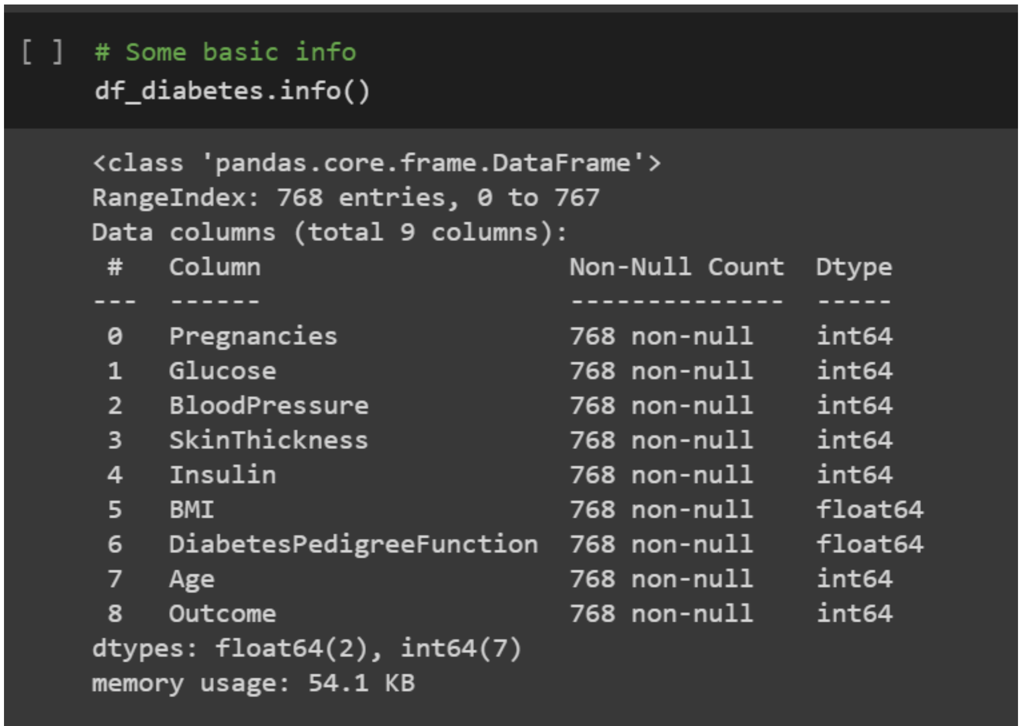
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

1.3 Basic information



```
[ ] #Check the shape of the df
    df_diabetes.shape

    (768, 9)
```

So the dataset has 768 records spread around 9 columns



```
[ ] # Some basic info
    df_diabetes.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
```

•7 columns are integers i.e. Pregnancies, Glucose, BP, Skinthickness, Insuling, Age, outcome

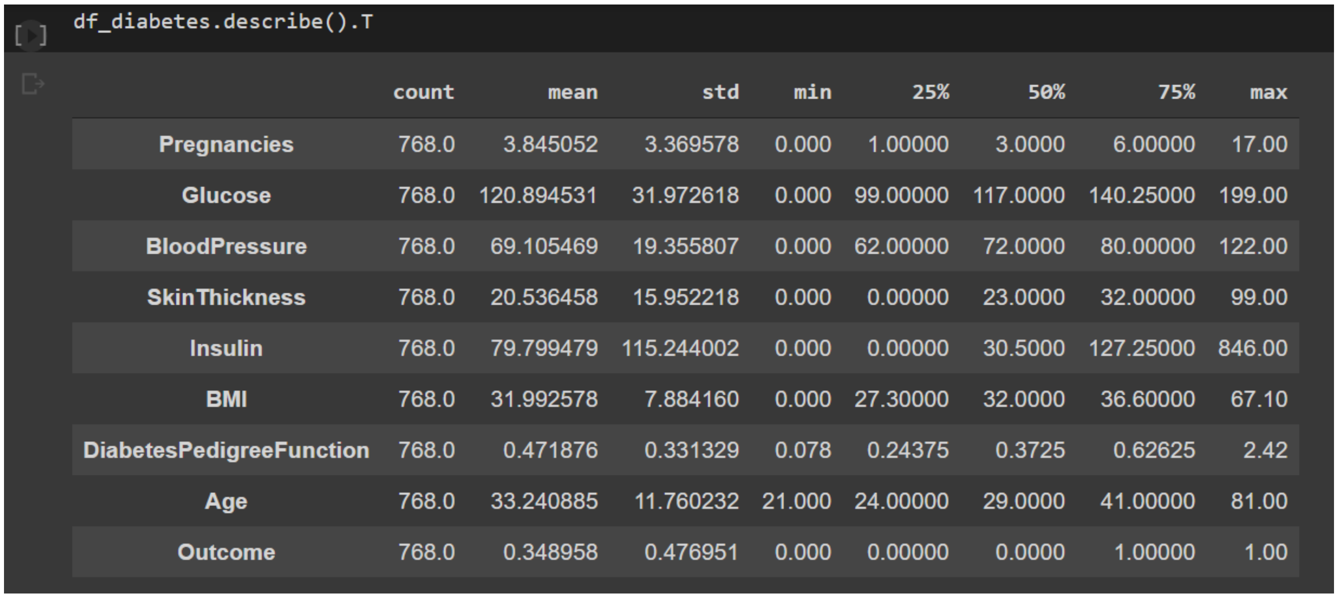
•2 columns are float i.e. BMI(body mass index), DiabetesPedigreeFunction(a Function which calculates the likelihood of diabetes based on family history)

•None of the columns have null values as shown

•it seems that outcome is the predictor variable which show if the person have diabetes or not

- •0 indicates non-Diabetic
- •1 indicates Diabetic

1.4 Descriptive Analysis:



```
df_diabetes.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Pregnancies	768.0	3.845052	3.369578	0.000	1.00000	3.0000	6.00000	17.00
Glucose	768.0	120.894531	31.972618	0.000	99.00000	117.0000	140.25000	199.00
BloodPressure	768.0	69.105469	19.355807	0.000	62.00000	72.0000	80.00000	122.00
SkinThickness	768.0	20.536458	15.952218	0.000	0.00000	23.0000	32.00000	99.00
Insulin	768.0	79.799479	115.244002	0.000	0.00000	30.5000	127.25000	846.00
BMI	768.0	31.992578	7.884160	0.000	27.30000	32.0000	36.60000	67.10
DiabetesPedigreeFunction	768.0	0.471876	0.331329	0.078	0.24375	0.3725	0.62625	2.42
Age	768.0	33.240885	11.760232	21.000	24.00000	29.0000	41.00000	81.00
Outcome	768.0	0.348958	0.476951	0.000	0.00000	0.0000	1.00000	1.00

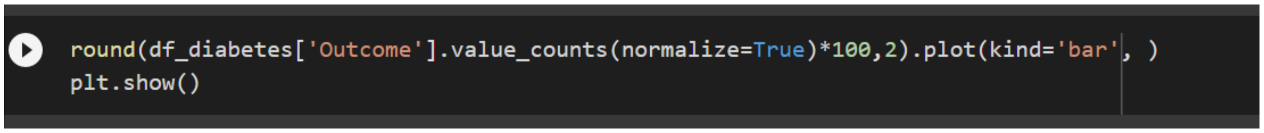
above we see the basic statistics about the diabetes dataframe some point to note

•For Pregnancy we have min of 0 and max of 17 with a mean of 3

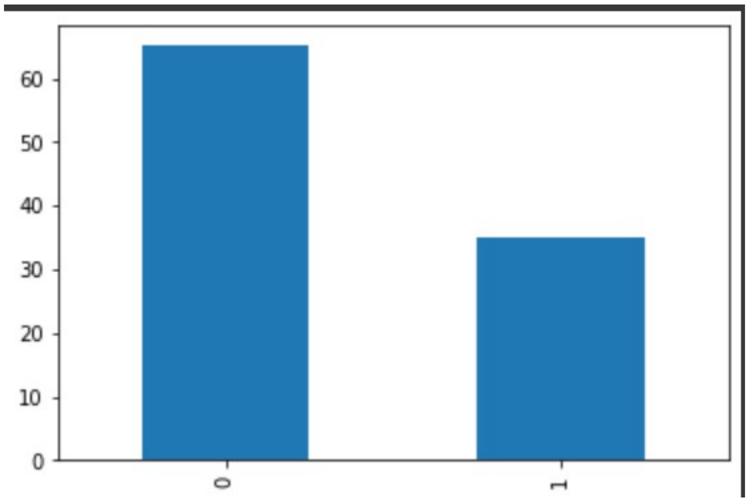
•one thing to note is that for many of the predictors (Glucose, BP, Skinthickness, Insulin, BMI) we have min, reading of 0 which is not possible, so it is most like be the missing values which needs to be imputes based on the distribution of these predictors

- •For age values ranges from 11-81 with a means of 33 which is somewhat normal distributes

1.5 Proportion of the outcome:



```
round(df_diabetes['Outcome'].value_counts(normalize=True)*100,2).plot(kind='bar', )
plt.show()
```



So we have about 65% non-diabetic and 35% diabetic

Now we have some idea about the dataset

Now lets deal with the missing values. Although in isna we dont have any missing values. we saw above that we have lot of zero values in many of the predictors (Glucose, BP, Skinthickness, Insulin, BMI) which is not possible. So we have to treat these zero as missing values and impute them in some manner.

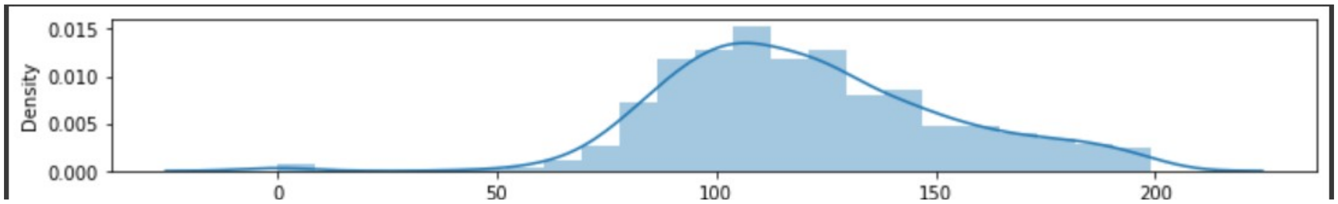
1.6 Missing Values:

1.5.1 Let's Check the distribution of variables:

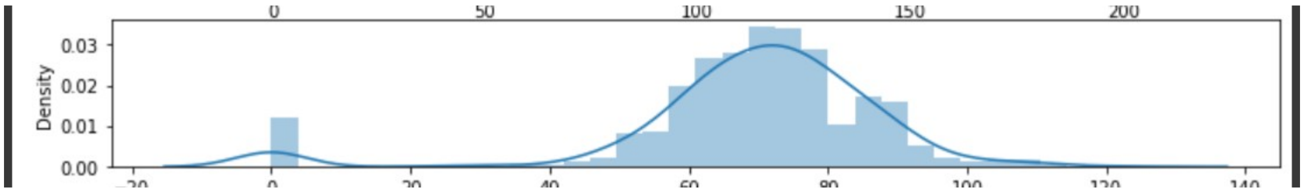
Before we impute these zero values it is advisable to see the distribution of the missing values of see most efficient strategy i.e. mean, median to deal with the missing values

- •if the distribution of the predictors it skewed(right or left) it is advisable to impute it with median
- •if the distribution of the predictors it normal then we can impute it with mean.

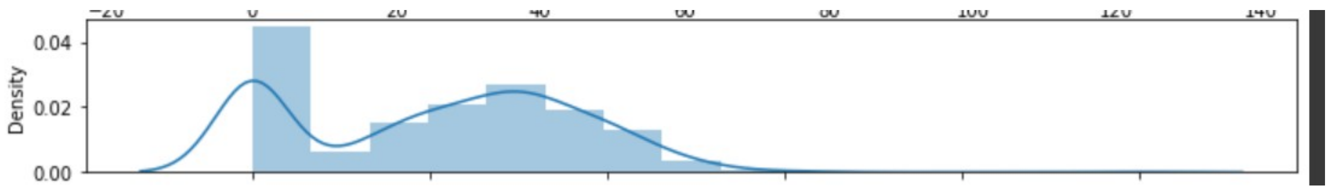
Glucose



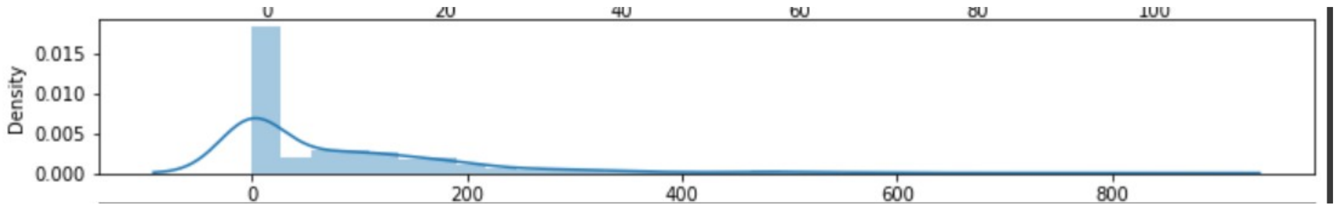
Blood Pressure



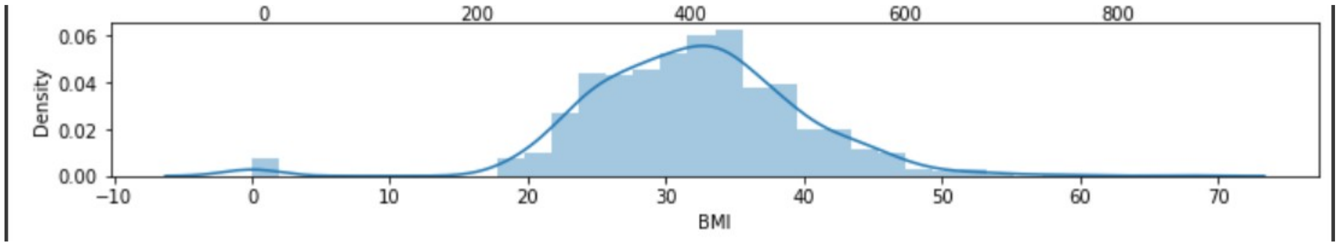
Skin Thickness



Insulin



BMI

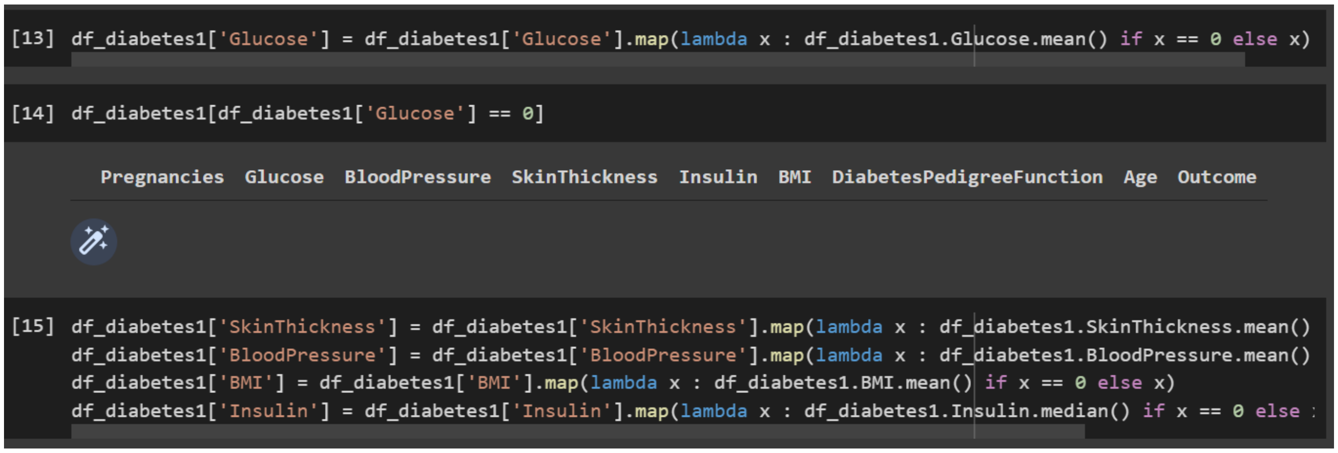


1. So from the above frequency plot we know that Glucose, BP, skinthickness and BMI are somewhat normally distributes.

2. Insulin is Right skewed so for insuling we can impute its values with median and rest we can impute

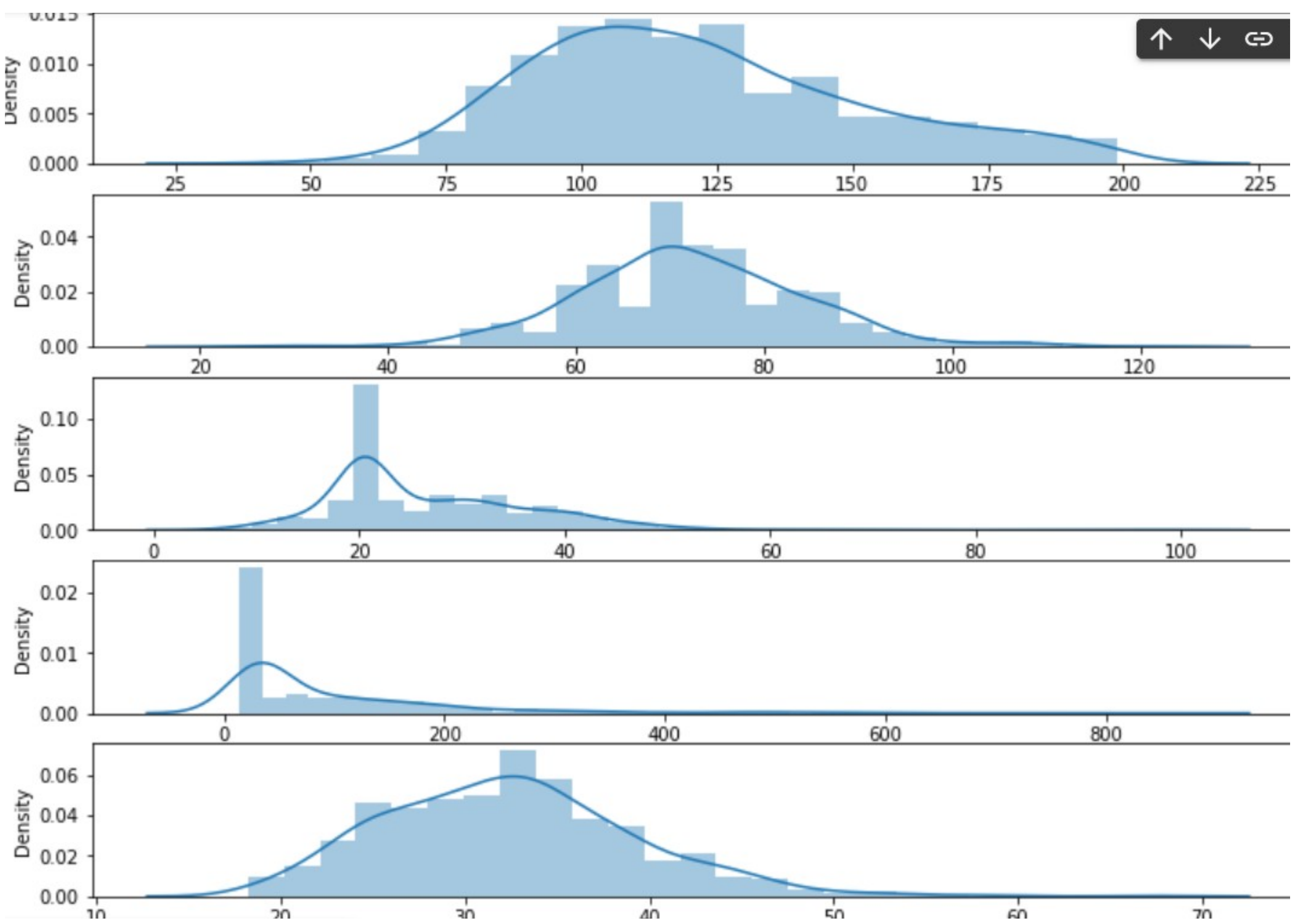
with means

1.5.2 Treat Missing Values



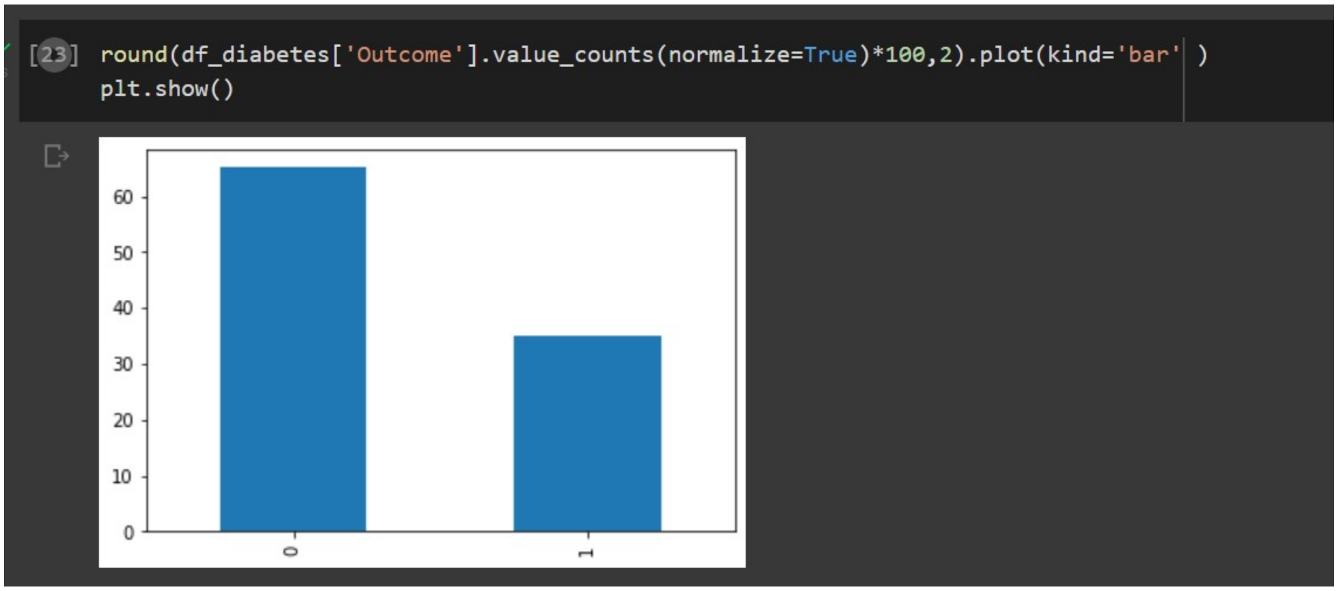
```
[13] df_diabetes1['Glucose'] = df_diabetes1['Glucose'].map(lambda x : df_diabetes1.Glucose.mean() if x == 0 else x)

[14] df_diabetes1[df_diabetes1['Glucose'] == 0]

     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin  BMI  DiabetesPedigreeFunction  Age  Outcome

[15] df_diabetes1['SkinThickness'] = df_diabetes1['SkinThickness'].map(lambda x : df_diabetes1.SkinThickness.mean()
     df_diabetes1['BloodPressure'] = df_diabetes1['BloodPressure'].map(lambda x : df_diabetes1.BloodPressure.mean()
     df_diabetes1['BMI'] = df_diabetes1['BMI'].map(lambda x : df_diabetes1.BMI.mean() if x == 0 else x)
     df_diabetes1['Insulin'] = df_diabetes1['Insulin'].map(lambda x : df_diabetes1.Insulin.median() if x == 0 else :
```

1.5.3 Checking the Distribution after imputation



2. Exploratory Data Analysis(EDA)

2.1 Checking the count plot of the outcome



```
[23] round(df_diabetes['Outcome'].value_counts(normalize=True)*100,2).plot(kind='bar' )
     plt.show()
```

•So here we have an imbalanced dataset where non-diabetic are represented more than the diabetic. This could pose the problem while trying to classify those people who have diabetic.

•This results in models that have poor predictive performance, specifically for the minority class i.e. Diabetic

•here we are talking about the correct prediction of diabetic person so the correct prediction of the minority class is more important i.e. it is important that we classify all the diabetic even if some non-diabetic could also be classify as diabetic not the other way around

•Some of the approaches to deal with this sort of problem could be

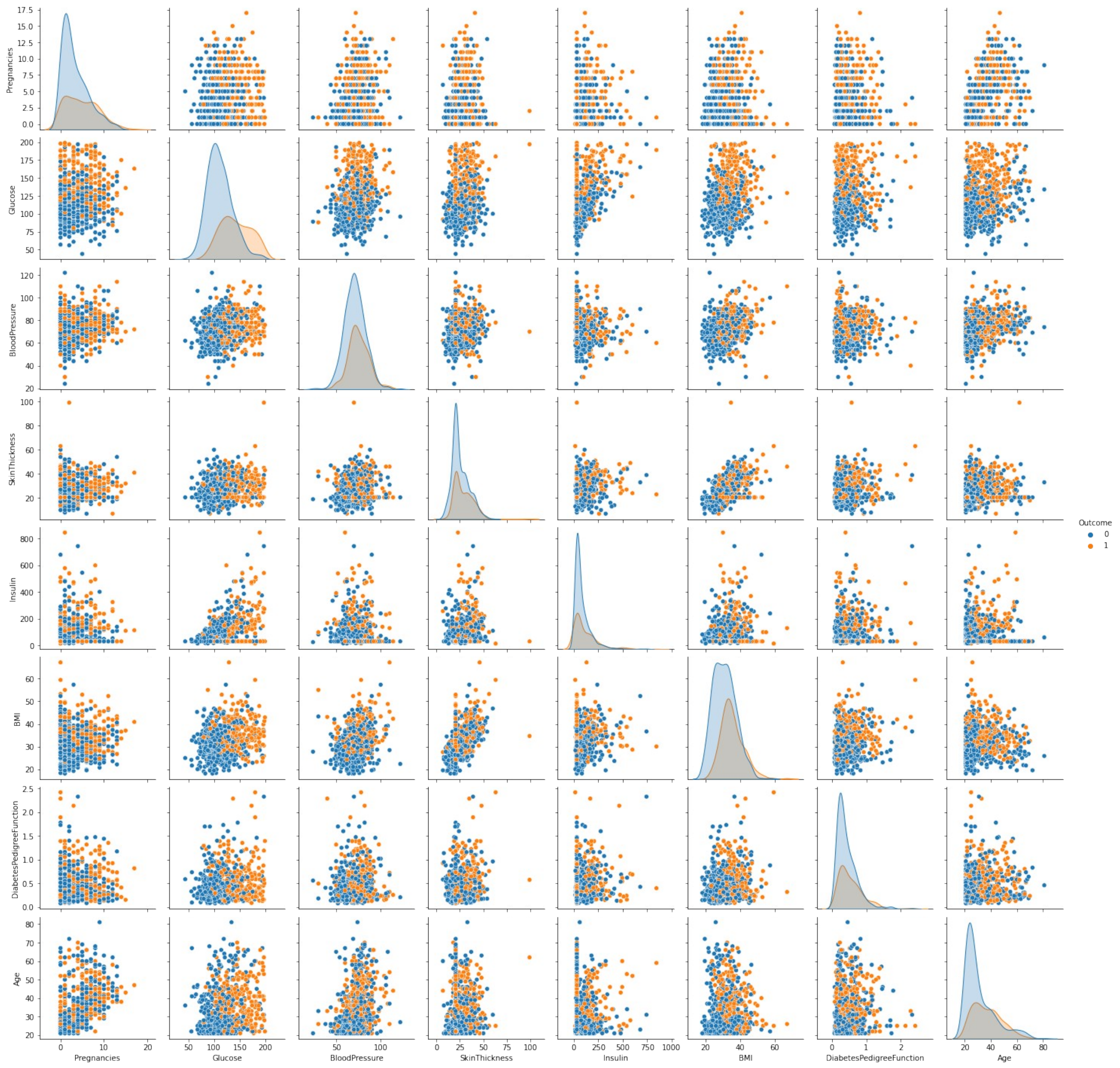
1.Try to see the precision and recall values with the original data

2.Do Stratified sampling so the equal number of classes get represented in train/test split

3.if Nothing work we can always use Oversampling/Under-sampling algo like SMOTE to better represent the data

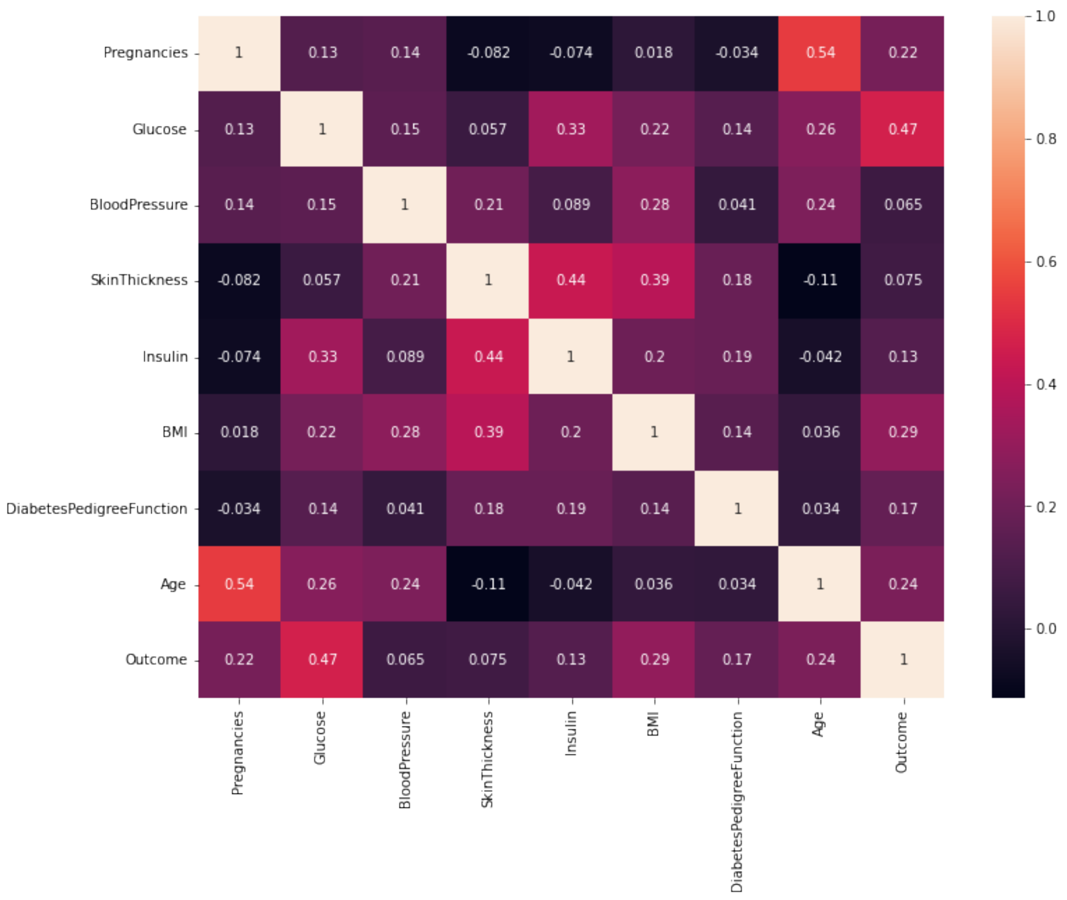
2.2 Creating the Scatter plot

For this let’s use the Seaborn library Pairplot method with hue = Outcome



2.3 Correlation matrix:

Here we’ll be doing the correlation analysis between to see the relationship between the predictor and outcome. Also see if there is a problem of multi-collinearity between variables



As Such we do not see Strong correlation between any predictor variable when look form the outcome point of view there are some what positive corr with Glucose level, BMI, Age, Preg in that order.

So now we can move onto the model building part.

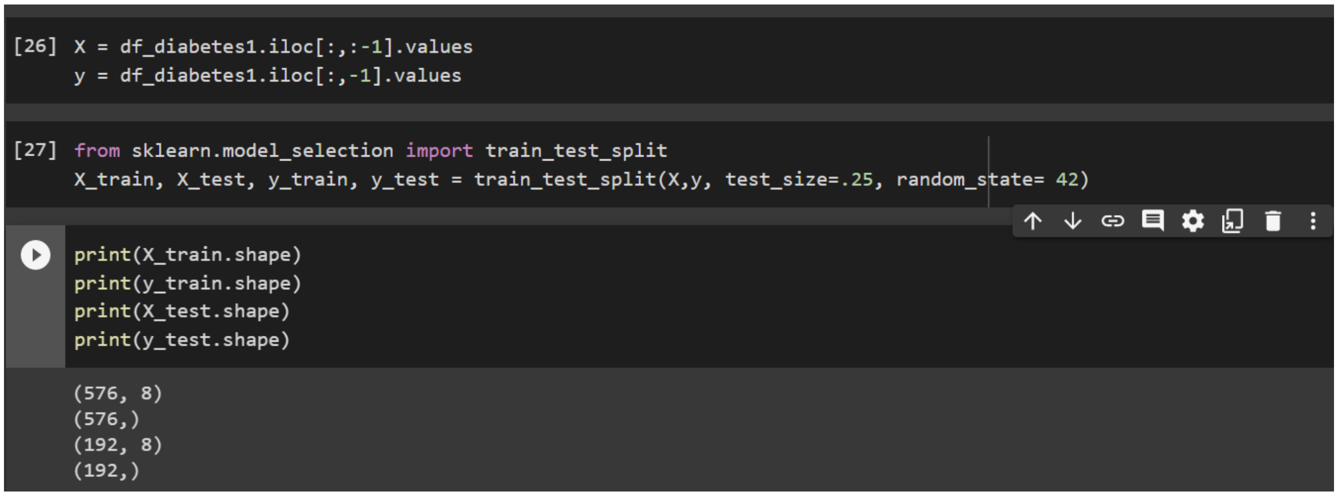
3. Data Modeling:

3.1 Data Preprocessing:

3.1.1 Train/Test Split

Before we move on to the model building and comparing various model with KNN, we need to preprocess the data

First do the test train split



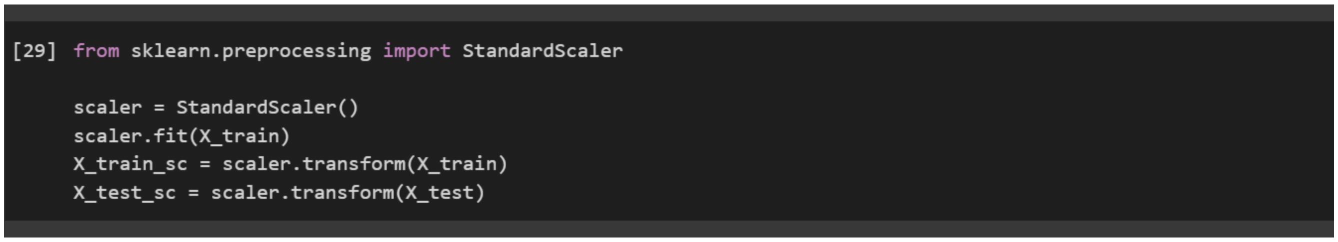
```
[26] X = df_diabetes1.iloc[:,:-1].values
     y = df_diabetes1.iloc[:,-1].values
```

```
[27] from sklearn.model_selection import train_test_split
     X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=.25, random_state= 42)
```

```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)

(576, 8)
(576,)
(192, 8)
(192,)
```

So now the train set have 576 records and test set have 192 records

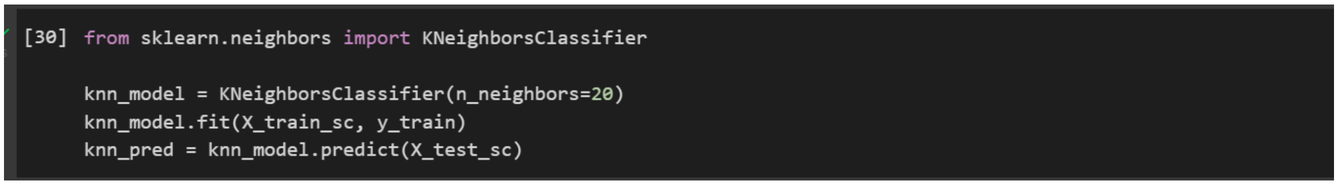
3.1.2 Standard Scaling



```
[29] from sklearn.preprocessing import StandardScaler

     scaler = StandardScaler()
     scaler.fit(X_train)
     X_train_sc = scaler.transform(X_train)
     X_test_sc = scaler.transform(X_test)
```

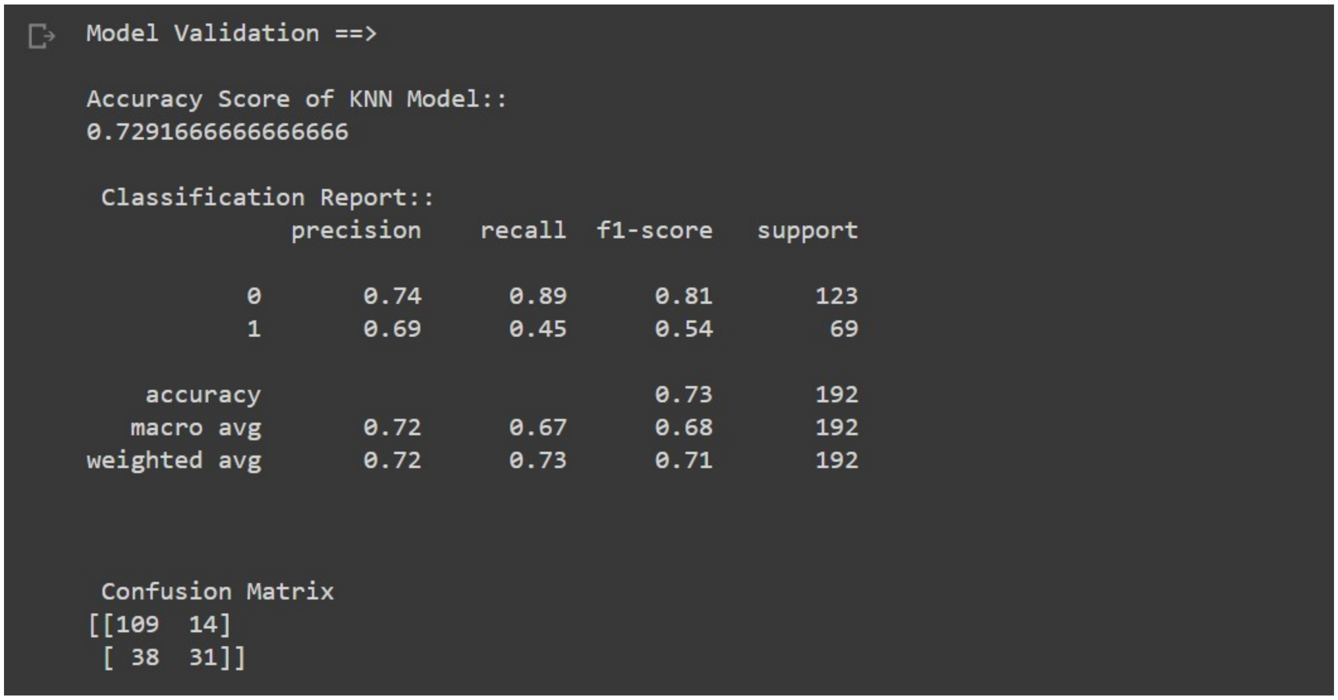
3.2 Data Modeling

3.2.1 K-Nearest Neighbors:



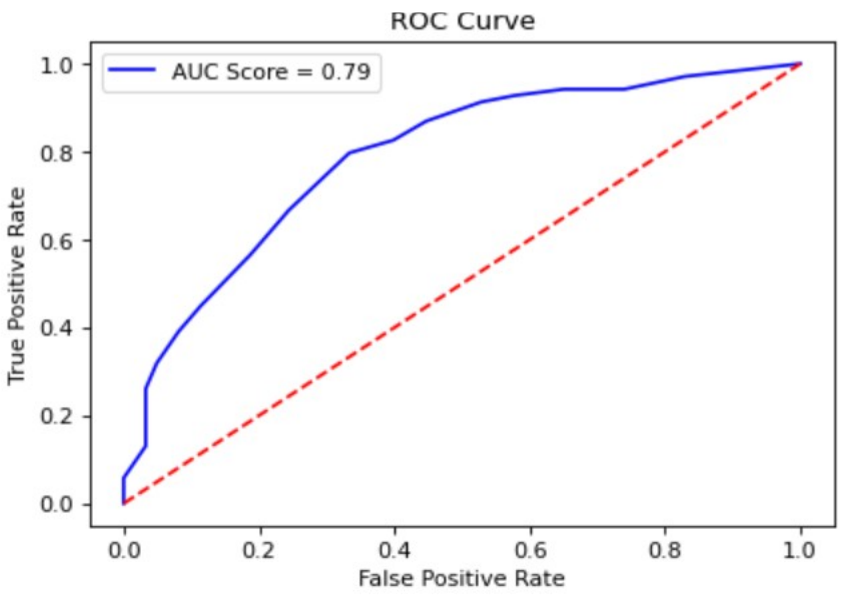
```
[30] from sklearn.neighbors import KNeighborsClassifier

     knn_model = KNeighborsClassifier(n_neighbors=20)
     knn_model.fit(X_train_sc, y_train)
     knn_pred = knn_model.predict(X_test_sc)
```

So model is trained on the X_train data and we have predicted the outcome based on the test predictors now lets evaluate



```
Model Validation ==>

Accuracy Score of KNN Model::
0.7291666666666666

 Classification Report::
              precision    recall  f1-score   support

           0       0.74      0.89      0.81       123
           1       0.69      0.45      0.54        69

    accuracy                           0.73       192
   macro avg       0.72      0.67      0.68       192
weighted avg       0.72      0.73      0.71       192



 Confusion Matrix
[[109  14]
 [ 38  31]]
```

•

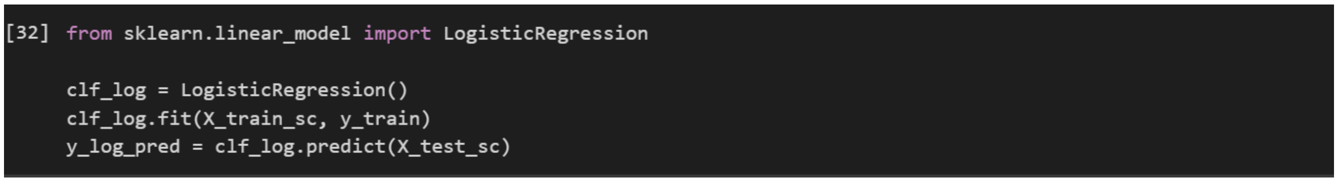


Here as we see that the precision for both positive and negative is okay i.e. how many correctly predicted turned out to be positive.

- but when we look at the recall values which is how many actual positive (on both diabetic and non diabetic) we are able to predict correctly. Here recall values is very low esp. for the diabetic class. Out of the total diabetic we are able to predict only 46% of the time
- Also the AOC score which is how much better we are able to differentiate between diabetic and non-diabetic class is 79% which is okay.

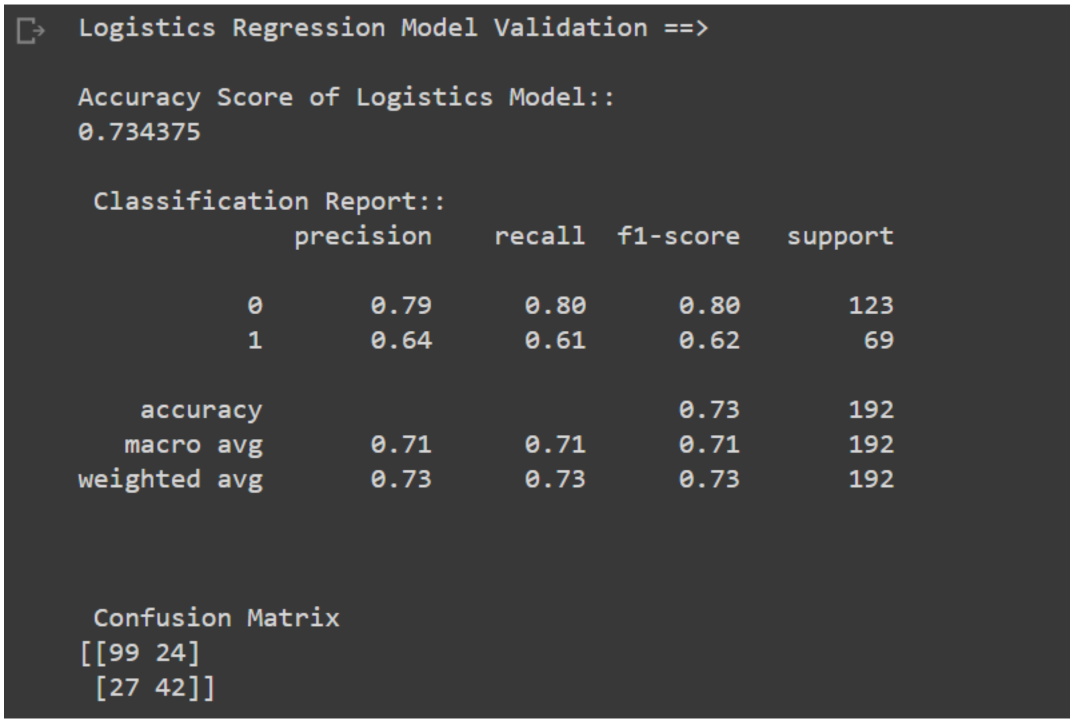
Now lets train other classifier to see if the position improve

3.2.2. Logistic Regression

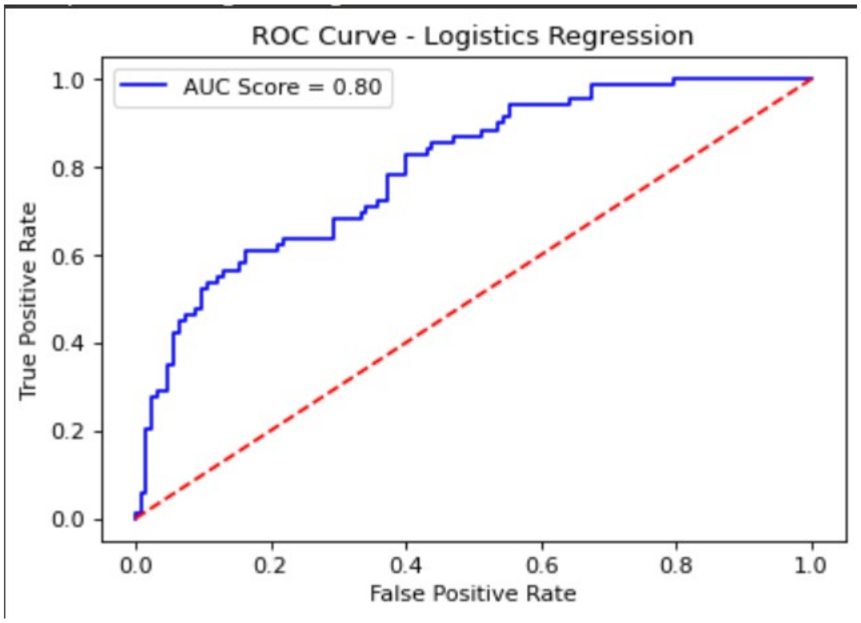


```
[32] from sklearn.linear_model import LogisticRegression

     clf_log = LogisticRegression()
     clf_log.fit(X_train_sc, y_train)
     y_log_pred = clf_log.predict(X_test_sc)
```

Evaluation

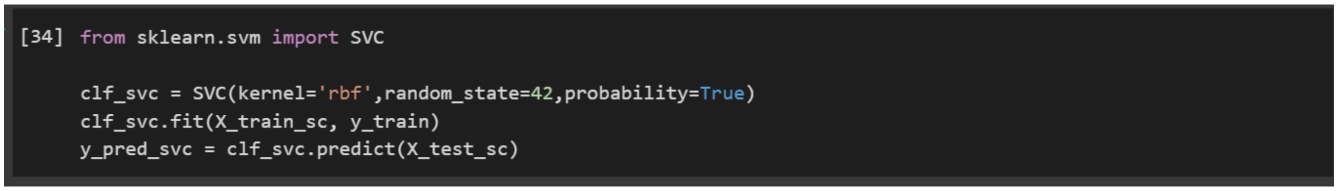


```
Logistics Regression Model Validation ==>

Accuracy Score of Logistics Model::
0.734375

 Classification Report::
              precision    recall  f1-score   support

           0       0.79      0.80      0.80       123
           1       0.64      0.61      0.62        69

    accuracy                           0.73       192
   macro avg       0.71      0.71      0.71       192
weighted avg       0.73      0.73      0.73       192



 Confusion Matrix
[[99 24]
 [27 42]]
```



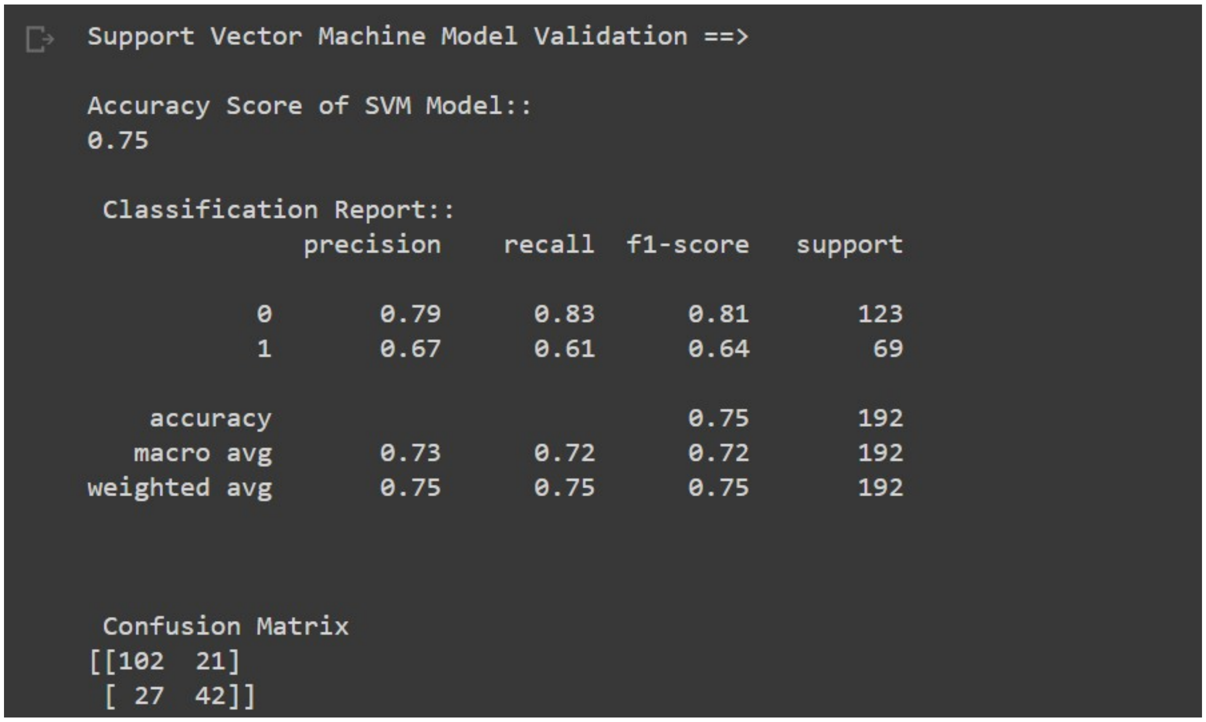
- its performance is almost same as that of the basic KNN classifier . Some small improvement are shown
- accuracy has only improved by about 0.6 percent point
- Precision has decrease for positive class by about 5% point
- but Good news is that the recall value have increase as compared to the basic KNN model by 15% point which is a good
- Also the Ability to differentiate has only change by about 2%

3.2.3 Support Vector Machine

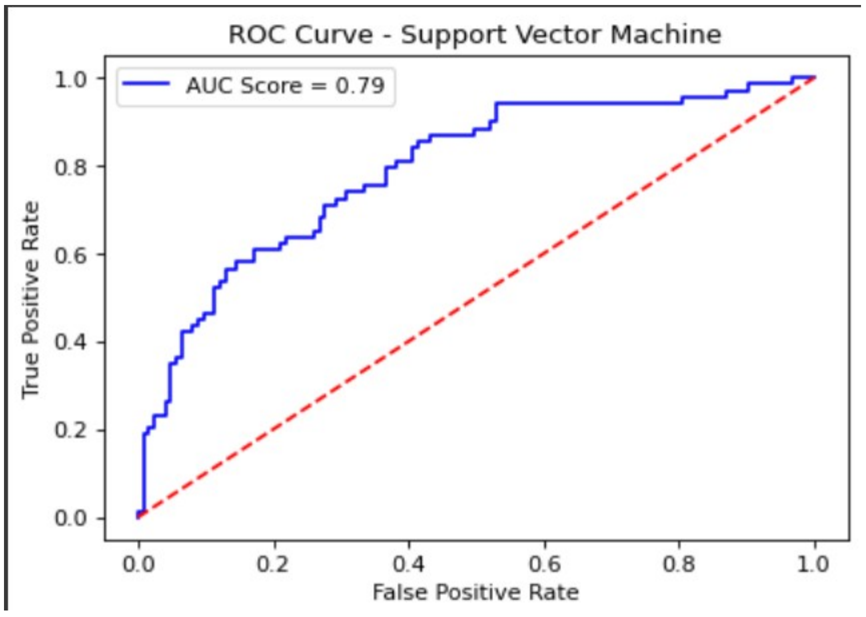


```
[34] from sklearn.svm import SVC

     clf_svc = SVC(kernel='rbf',random_state=42,probability=True)
     clf_svc.fit(X_train_sc, y_train)
     y_pred_svc = clf_svc.predict(X_test_sc)
```

Evaluation



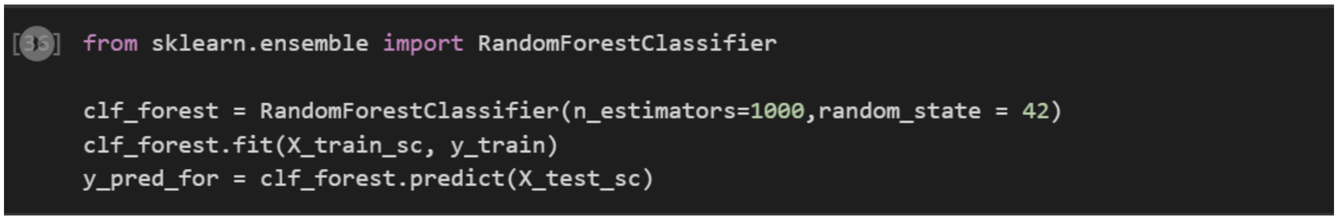
```
Support Vector Machine Model Validation ==>

Accuracy Score of SVM Model::
0.75

 Classification Report::
              precision    recall  f1-score   support

           0       0.79      0.83      0.81       123
           1       0.67      0.61      0.64        69

    accuracy                           0.75       192
   macro avg       0.73      0.72      0.72       192
weighted avg       0.75      0.75      0.75       192



 Confusion Matrix
[[102  21]
 [ 27  42]]
```



Not much has change with the using SVM model some parameter like accracy has increase but not much improvement

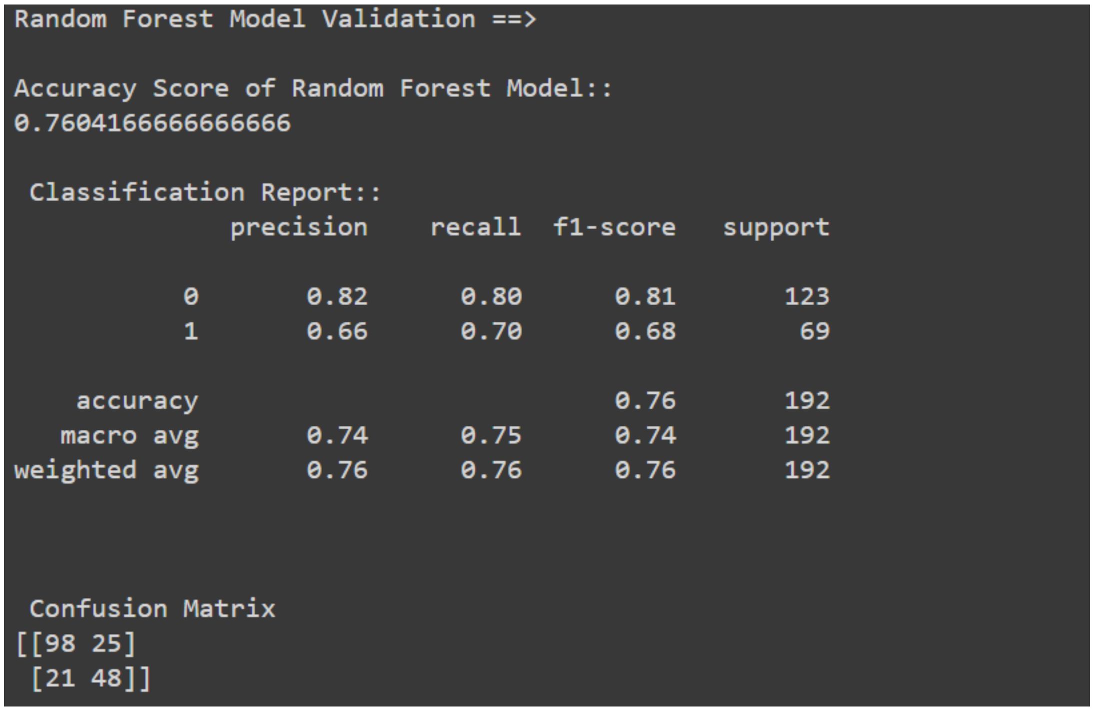
Lets see the Random Forest model

3.2.4 Ensemble Model – Random Forest

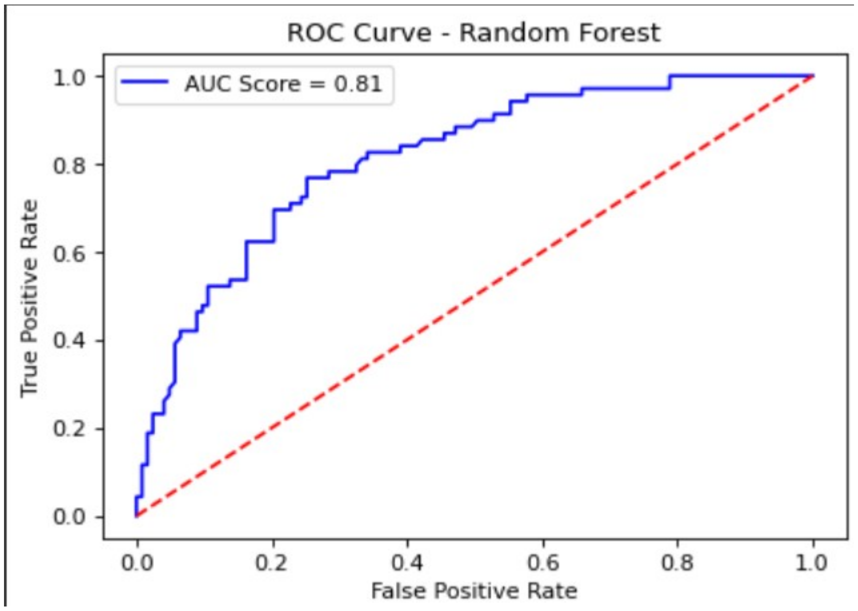


```
[35] from sklearn.ensemble import RandomForestClassifier

     clf_forest = RandomForestClassifier(n_estimators=1000,random_state = 42)
     clf_forest.fit(X_train_sc, y_train)
     y_pred_for = clf_forest.predict(X_test_sc)
```

Evaluation



```
Random Forest Model Validation ==>

Accuracy Score of Random Forest Model::
0.7604166666666666

 Classification Report::
              precision    recall  f1-score   support

           0       0.82      0.80      0.81       123
           1       0.66      0.70      0.68        69

    accuracy                           0.76       192
   macro avg       0.74      0.75      0.74       192
weighted avg       0.76      0.76      0.76       192



 Confusion Matrix
[[98 25]
 [21 48]]
```



As compared to the Basic KNN model we see following improvement

- Accuracy has increase from 73% to 76%
- for non-diabetic it is showing much improvement from 72% to 82%
- but for diabetic it has declined from 69 to 66% which means out of 100 total diabetic that it predict 66 are actually diabetic
- for non-diabetic it is showing the decline from 89% to 80% which means earlier out of total 100 non diabetic it is able to predict 89 correctly but now only 80 are correctly diagnosed as Non-diabetic

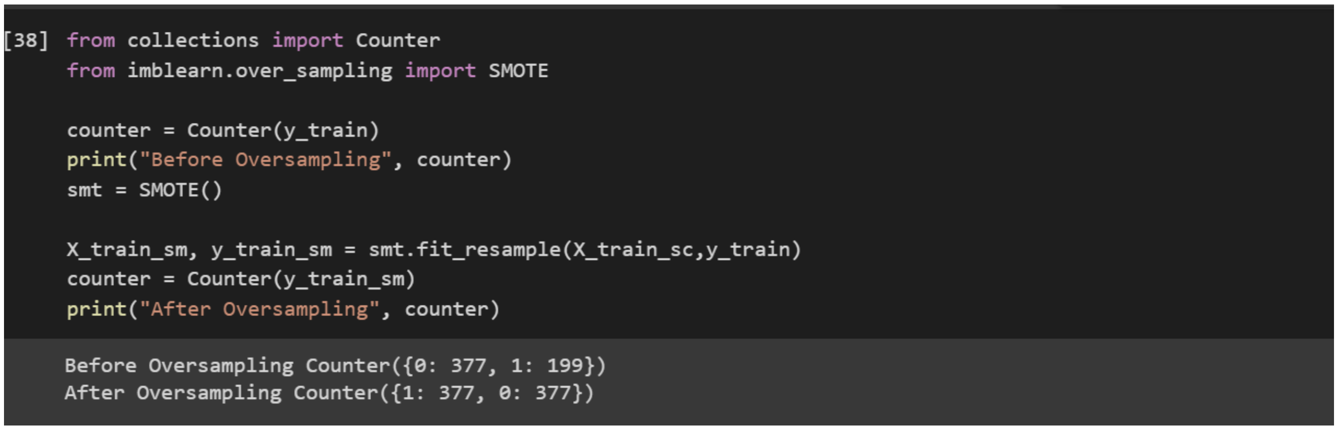
- But Good news is that for diabetic values has increase from 45% to 70% which means earlier out of 100 diabetic KNN model was able to do correct prediction for only 45 person which has now increase to 70 people
- AOC (ROC) score has measly increase from 79% to 81%

3.3 Data Modeling with oversampling

Earlier we saw that we have an imbalance data i.e. 65% of cases are of non-diabetic class and 35% are for the diabetic class. This is the big reason for not able to correctly predict the diabetic classification as compared to the Non diabetic classification

To solve this we will apply Oversampling techniques like SMOTE (Synthetic minority oversamping technique)

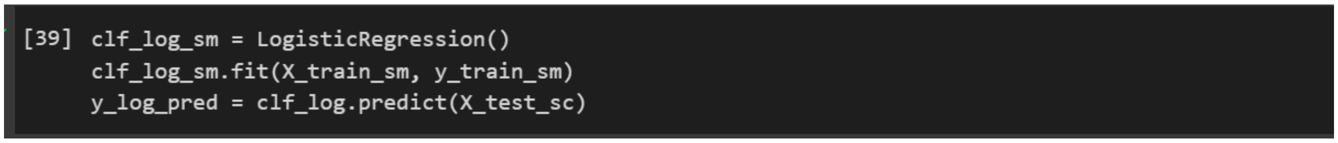
SMOTE is an oversampling technique where the synthetic samples are generated for the minority class. This algorithm helps to overcome the overfitting problem posed by random oversampling. It focuses on the feature space to generate new instances with the help of interpolation between the positive instances that lie together



```
[38] from collections import Counter
     from imblearn.over_sampling import SMOTE

     counter = Counter(y_train)
     print("Before Oversampling", counter)
     smt = SMOTE()

     X_train_sm, y_train_sm = smt.fit_resample(X_train_sc,y_train)
     counter = Counter(y_train_sm)
     print("After Oversampling", counter)
```

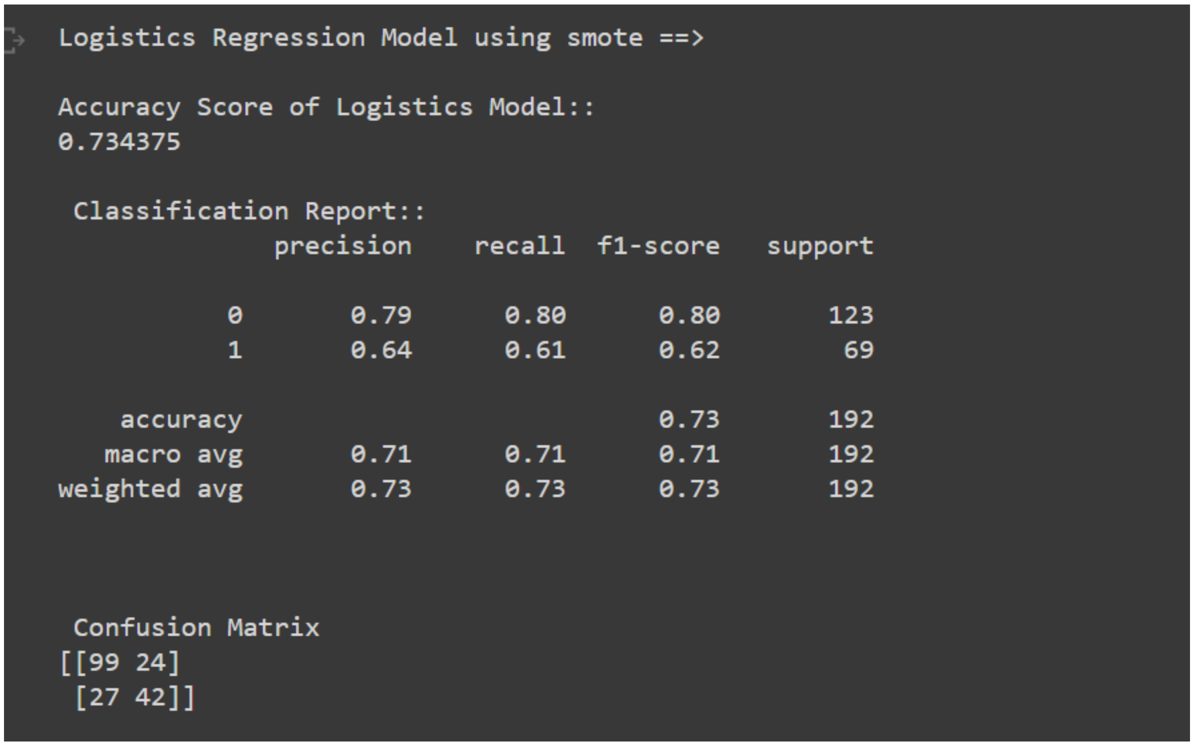
```
Before Oversampling Counter({0: 377, 1: 199})
After Oversampling Counter({1: 377, 0: 377})
```

3.3.1. Logistics Regression

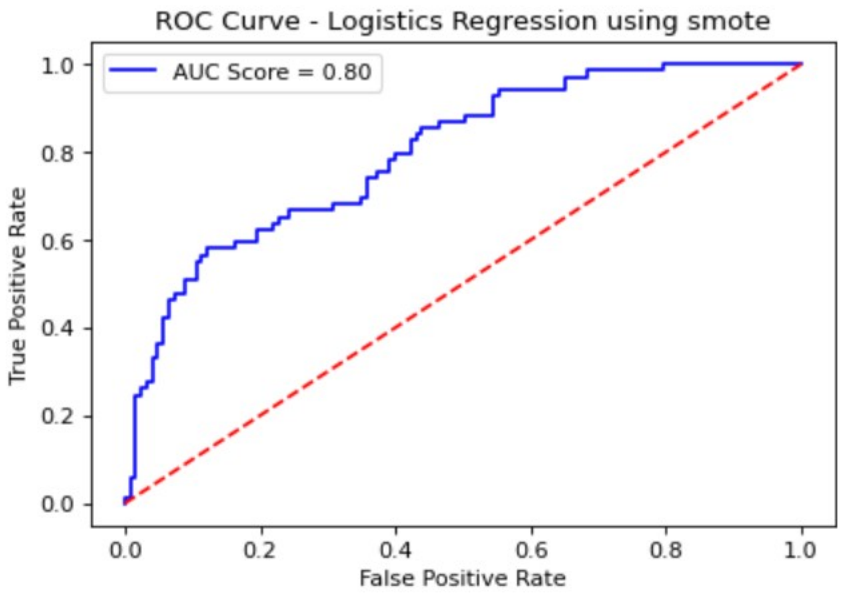


```
[39] clf_log_sm = LogisticRegression()
     clf_log_sm.fit(X_train_sm, y_train_sm)
     y_log_pred = clf_log.predict(X_test_sc)
```

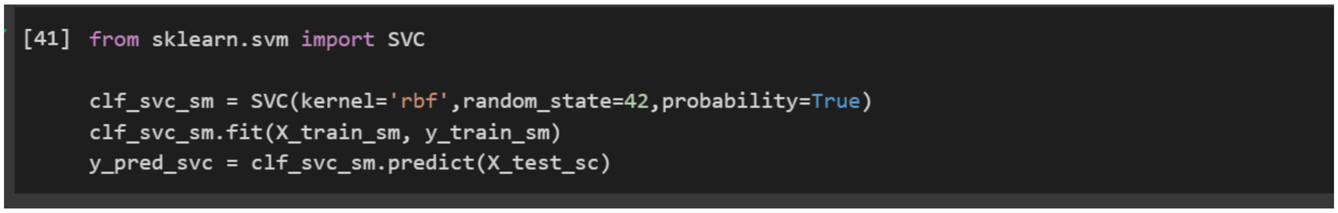
Evaluation



```
Logistics Regression Model using smote ==>

Accuracy Score of Logistics Model::
0.734375

 Classification Report::
              precision    recall  f1-score   support

           0       0.79      0.80      0.80       123
           1       0.64      0.61      0.62        69

    accuracy                           0.73       192
   macro avg       0.71      0.71      0.71       192
weighted avg       0.73      0.73      0.73       192



 Confusion Matrix
[[99 24]
 [27 42]]
```

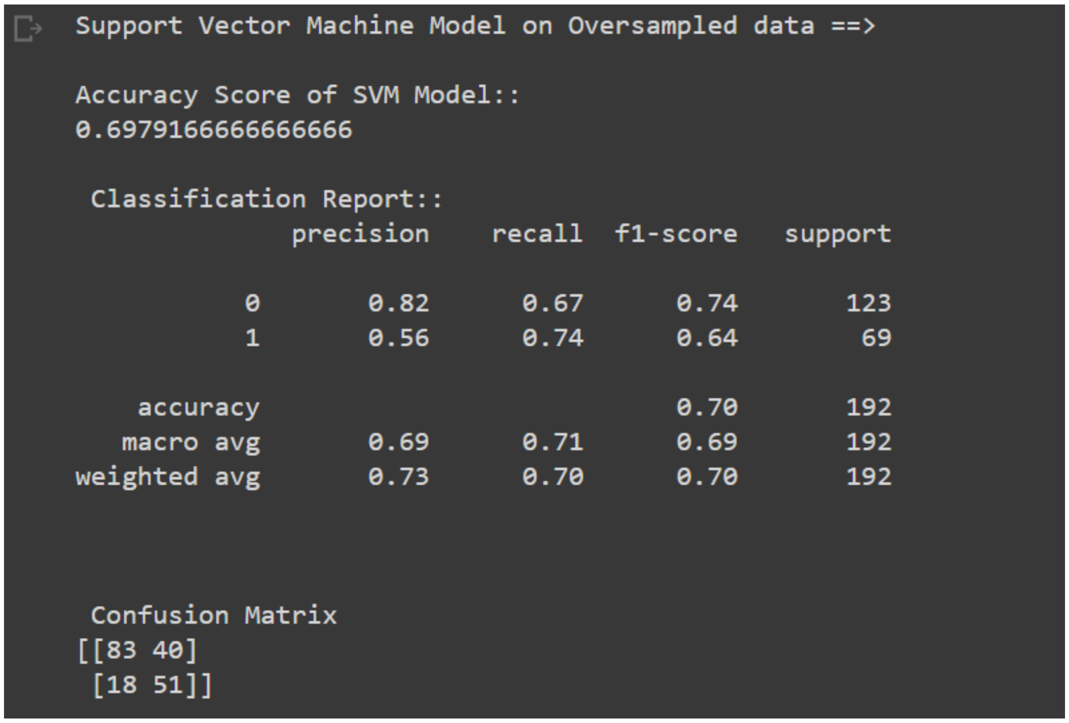


3.3.2 Support Vector Machine:

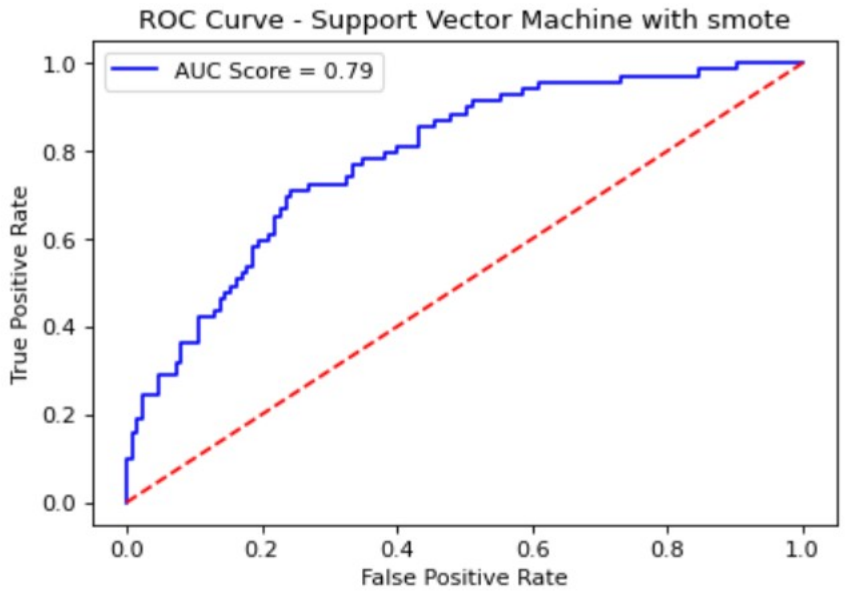


```
[41] from sklearn.svm import SVC

     clf_svc_sm = SVC(kernel='rbf',random_state=42,probability=True)
     clf_svc_sm.fit(X_train_sm, y_train_sm)
     y_pred_svc = clf_svc_sm.predict(X_test_sc)
```

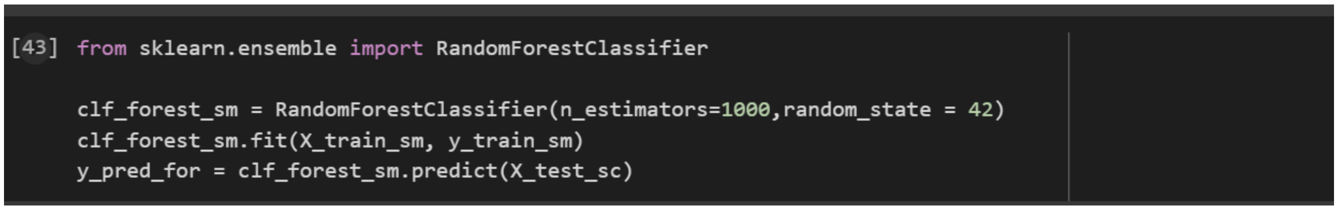
Evaluation



```
Support Vector Machine Model on Oversampled data ==>

Accuracy Score of SVM Model::
0.6979166666666666

 Classification Report::
              precision    recall  f1-score   support

           0       0.82      0.67      0.74       123
           1       0.56      0.74      0.64        69

    accuracy                           0.70       192
   macro avg       0.69      0.71      0.69       192
weighted avg       0.73      0.70      0.70       192



 Confusion Matrix
[[83 40]
 [18 51]]
```

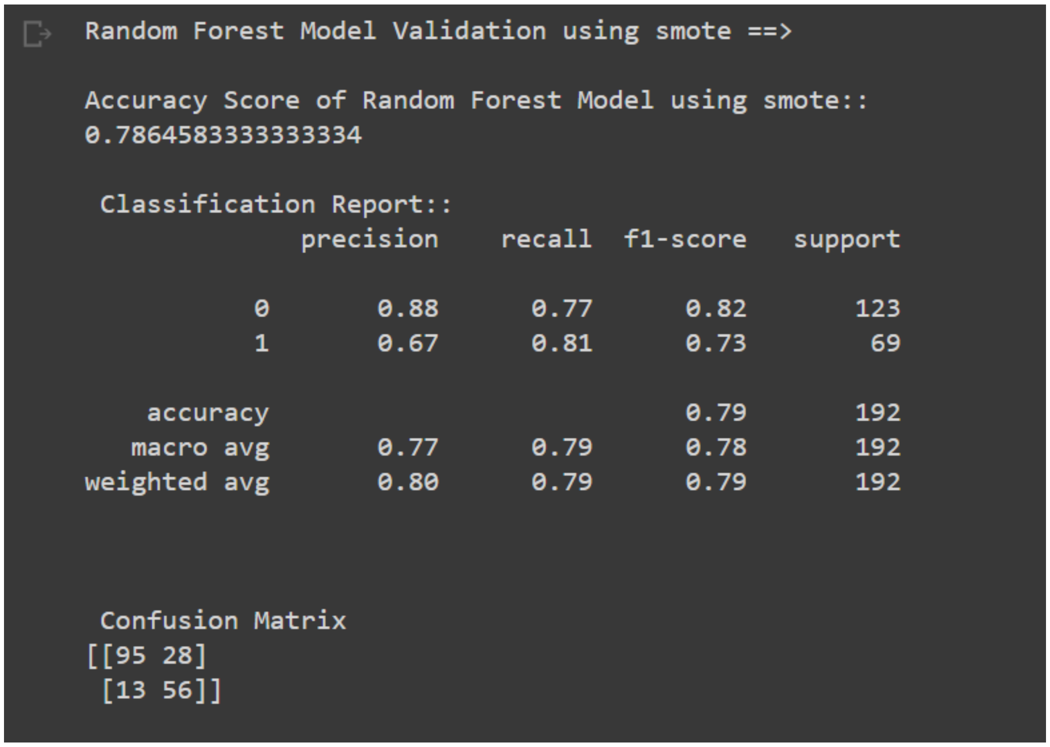


3.3.3 Ensemble Learning – Random Forest Classifier

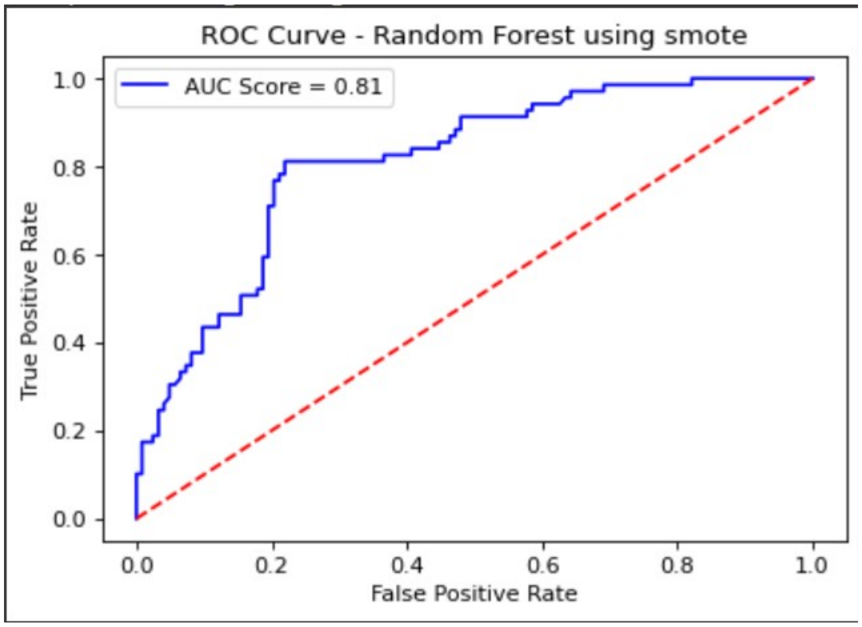


```
[43] from sklearn.ensemble import RandomForestClassifier

     clf_forest_sm = RandomForestClassifier(n_estimators=1000,random_state = 42)
     clf_forest_sm.fit(X_train_sm, y_train_sm)
     y_pred_for = clf_forest_sm.predict(X_test_sc)
```

Evaluation



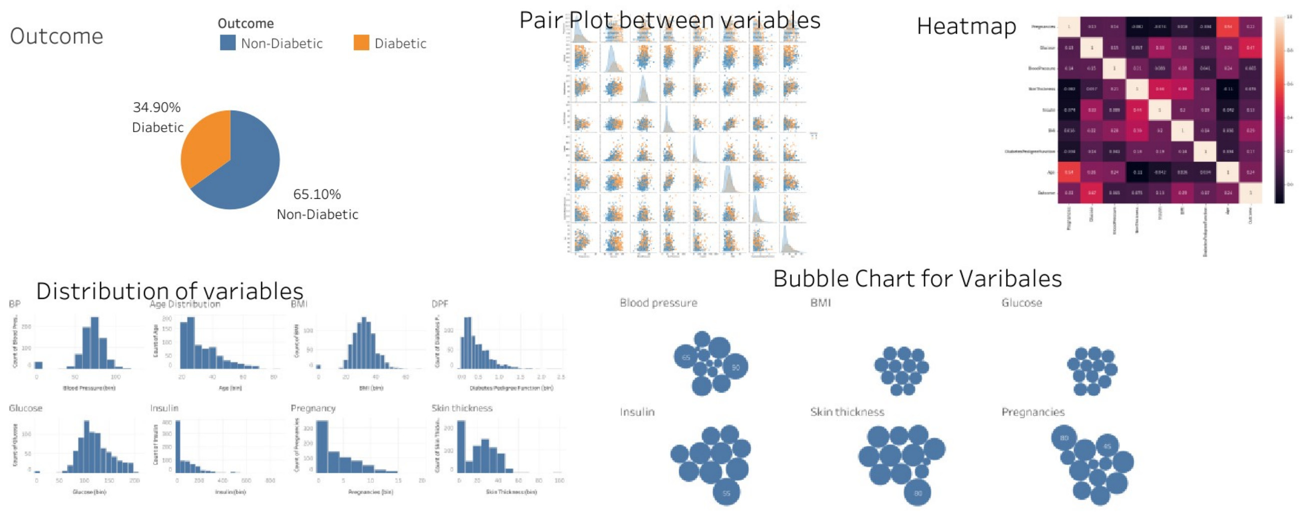
```
Random Forest Model Validation using smote ==>

Accuracy Score of Random Forest Model using smote::
0.7864583333333334

 Classification Report::
              precision    recall  f1-score   support

           0       0.88      0.77      0.82       123
           1       0.67      0.81      0.73        69

    accuracy                           0.79       192
   macro avg       0.77      0.79      0.78       192
weighted avg       0.80      0.79      0.79       192



 Confusion Matrix
[[95 28]
 [13 56]]
```



AS we can see that with using the over-sampled data we are able to increase the recall values 70% to 80% at the cost of 3% point decrease in the accuracy.

That means that we are able to better identify the diabetic person at the cost of few non-diabetic that could be misclassified as diabetic

4. Data Reporting:



For more details of the dashboard check-out

[Tableau public link]