IBM NAAN MUDHALVAN

ARTIFICIAL INTELLIGENCE-GROUP 2

PHASE -3 : DEVELOPMENT PART 1

PROBLEM STATEMENT:

The problem is to build an AI-powered diabetes prediction system that uses machine learning algorithms to analyze medical data and predict the likelihood of an individual developing diabetes. The system aims to provide early risk assessment and personalized preventive measures, allowing individuals to take proactive actions to manage their health.

EXPLANATION:

An AI-based diabetes prediction system is a computer program or application that uses artificial intelligence techniques, such as machine learning algorithms, to analyze data related to an individual's health, lifestyle, and medical history in order to predict the likelihood of them developing diabetes in the future. This system can help identify individuals at higher risk of diabetes, allowing for early intervention and preventive measures to manage or mitigate the disease.

DATASET DETAILS:

Context:

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective is to predict based on diagnostic measurements whether a patient has diabetes.

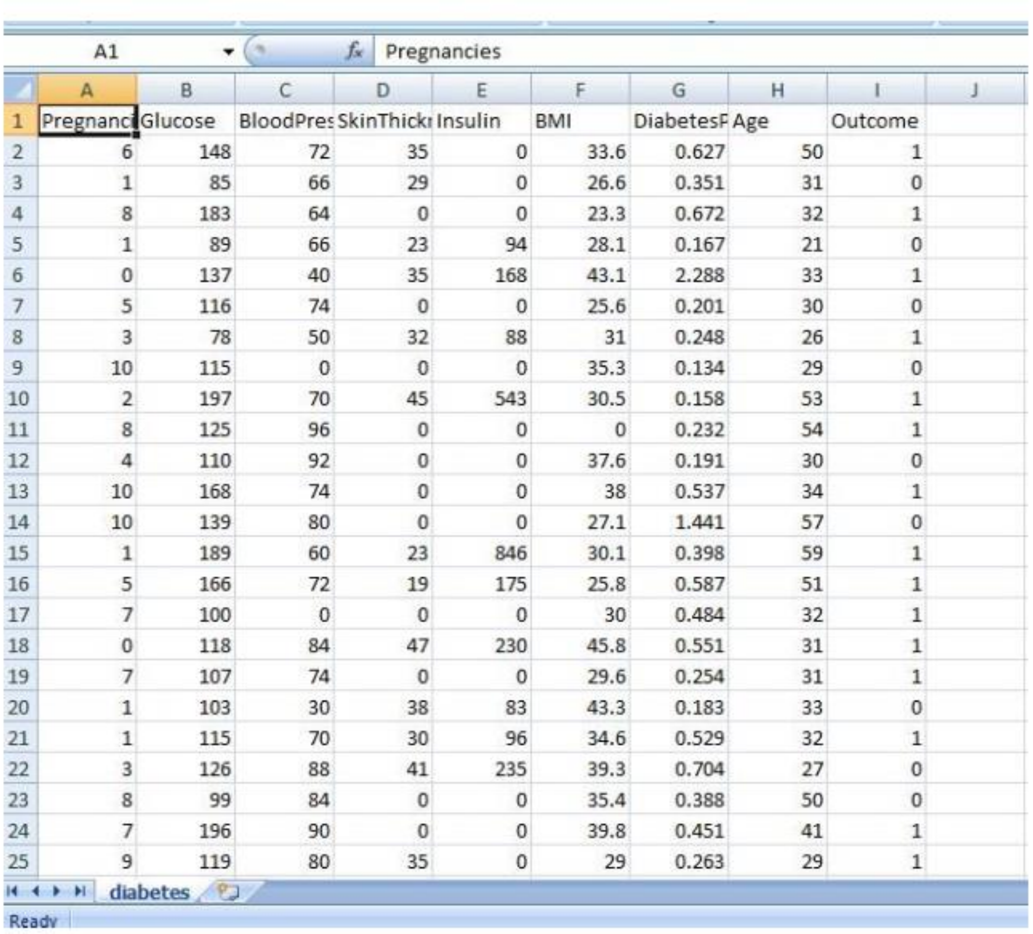
Content:

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.

- Pregnancies: Number of times pregnant
- Glucose: Plasma glucose concentration a 2 hours in an oral glucose tolerance test
- BloodPressure: Diastolic blood pressure (mm Hg)
- SkinThickness: Triceps skin fold thickness (mm)
- Insulin: 2-Hour serum insulin (mu U/ml)

- BMI: Body mass index (weight in kg/(height in m)^2)
- DiabetesPedigreeFunction: Diabetes pedigree function
- Age: Age (years)
- Outcome: Class variable (0 or 1)

GIVEN DATA SET:



	A	B	C	D	E	F	G	H	I	J
1	Pregnanci	Glucose	BloodPres	SkinThickn	Insulin	BMI	DiabetesP	Age	Outcome	
2	6	148	72	35	0	33.6	0.627	50	1	
3	1	85	66	29	0	26.6	0.351	31	0	
4	8	183	64	0	0	23.3	0.672	32	1	
5	1	89	66	23	94	28.1	0.167	21	0	
6	0	137	40	35	168	43.1	2.288	33	1	
7	5	116	74	0	0	25.6	0.201	30	0	
8	3	78	50	32	88	31	0.248	26	1	
9	10	115	0	0	0	35.3	0.134	29	0	
10	2	197	70	45	543	30.5	0.158	53	1	
11	8	125	96	0	0	0	0.232	54	1	
12	4	110	92	0	0	37.6	0.191	30	0	
13	10	168	74	0	0	38	0.537	34	1	
14	10	139	80	0	0	27.1	1.441	57	0	
15	1	189	60	23	846	30.1	0.398	59	1	
16	5	166	72	19	175	25.8	0.587	51	1	
17	7	100	0	0	0	30	0.484	32	1	
18	0	118	84	47	230	45.8	0.551	31	1	
19	7	107	74	0	0	29.6	0.254	31	1	
20	1	103	30	38	83	43.3	0.183	33	0	
21	1	115	70	30	96	34.6	0.529	32	1	
22	3	126	88	41	235	39.3	0.704	27	0	
23	8	99	84	0	0	35.4	0.388	50	0	
24	7	196	90	0	0	39.8	0.451	41	1	
25	9	119	80	35	0	29	0.263	29	1	

Necessary step to follow:

1.Import Libraries:

Start by importing the necessary libraries:

PROGRAM:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns
```

```
sns.set()

from pandas.plotting import scatter_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import classification_report
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

2.LOAD THE DATA

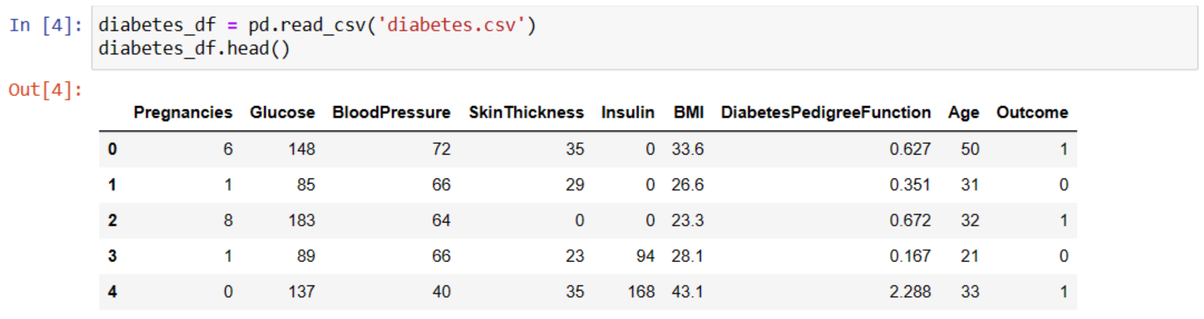
Load your dataset into a Pandas DataFrame. You can typically find

diabetes datasets in CSV format we can see the output below.

PROGRAM:

```
diabetes_df = pd.read_csv('diabetes.csv')
diabetes_df.head()
```

OUTPUT



```
In [4]: diabetes_df = pd.read_csv('diabetes.csv')
        diabetes_df.head()

Out[4]:
```

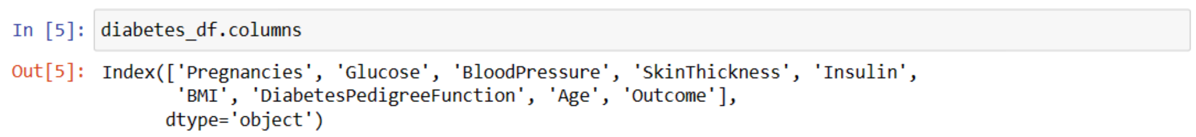
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

3.Exploratory Data Analysis (EDA):

Now let' see that what are columns available in our dataset.

```
diabetes_df.columns
```

OUTPUT :

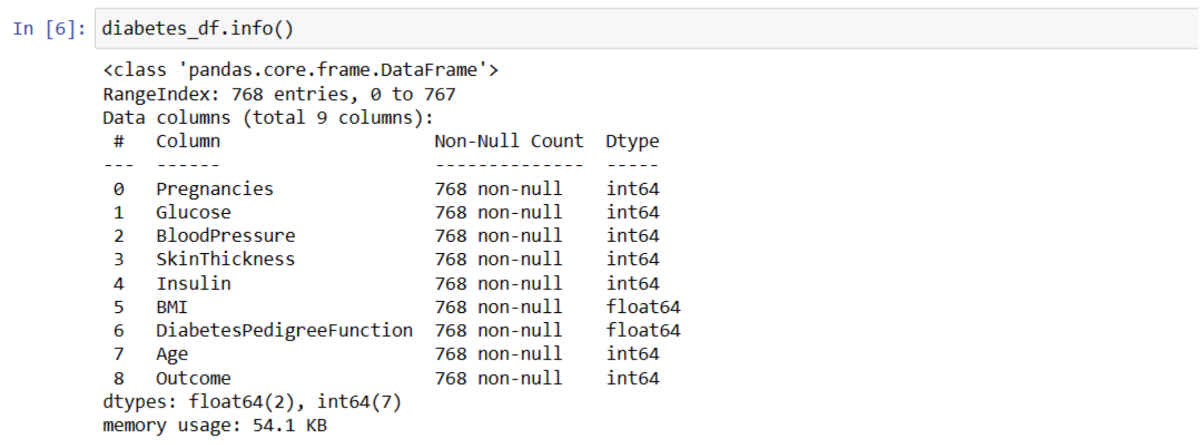


```
In [5]: diabetes_df.columns

Out[5]: Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
               'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
              dtype='object')
```

Information about the dataset

```
diabetes_df.info()
```

OUTPUT :

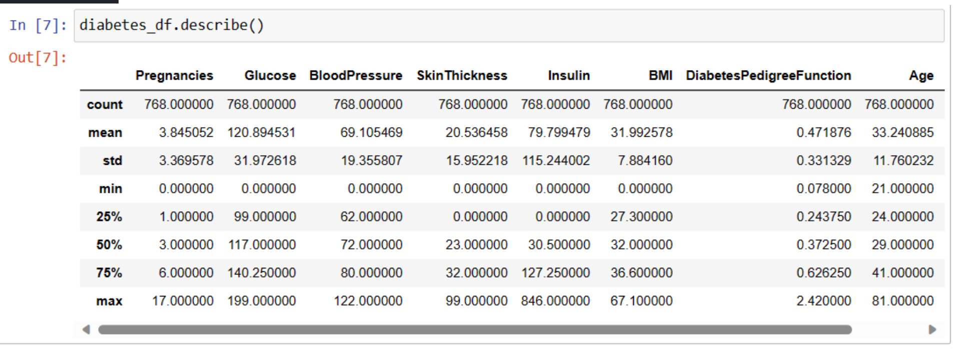


```
In [6]: diabetes_df.info()

        <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
```

To know more about the dataset

```
diabetes_df.describe()
```

OUTPUT :



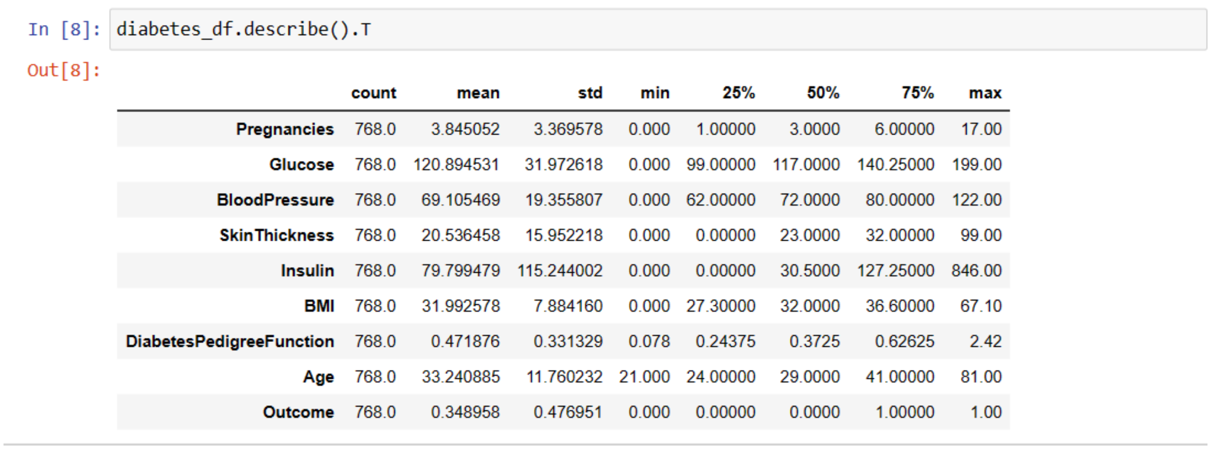
```
In [7]: diabetes_df.describe()

Out[7]:
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000

To know more about the dataset with transpose – here T is for the transpose

```
diabetes_df.describe().T
```

OUTPUT:



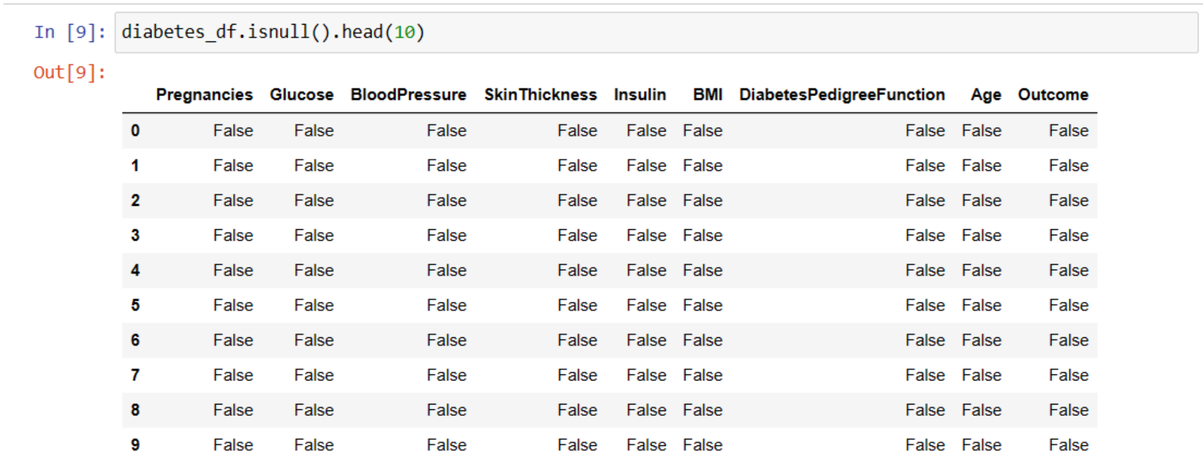
```
In [8]: diabetes_df.describe().T
Out[8]:
```

	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

Now let's check that if our dataset have null values or not

```
diabetes_df.isnull().head(10)
```

OUTPUT:



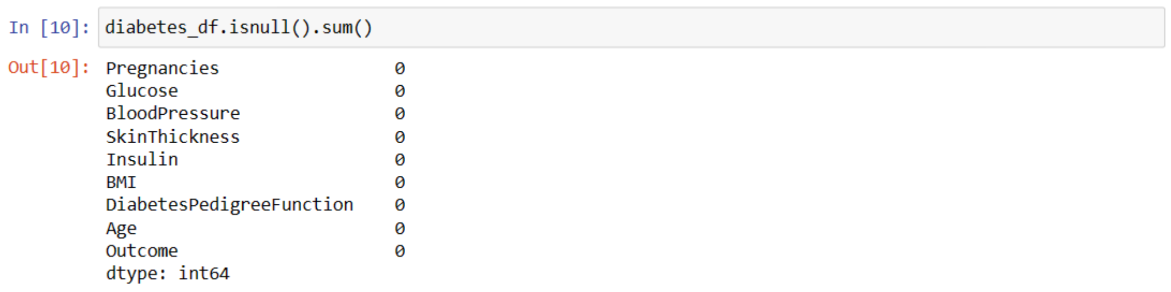
```
In [9]: diabetes_df.isnull().head(10)
Out[9]:
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False
5	False	False	False	False	False	False	False	False	False
6	False	False	False	False	False	False	False	False	False
7	False	False	False	False	False	False	False	False	False
8	False	False	False	False	False	False	False	False	False
9	False	False	False	False	False	False	False	False	False

Now let's check the number of null values our dataset has.

```
diabetes_df.isnull().sum()
```

OUTPUT :



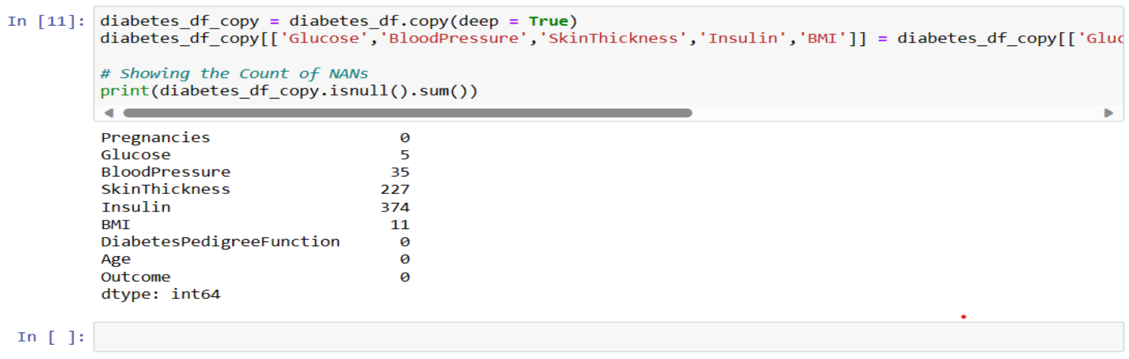
```
In [10]: diabetes_df.isnull().sum()

Out[10]: Pregnancies                 0
         Glucose                     0
         BloodPressure               0
         SkinThickness               0
         Insulin                     0
         BMI                         0
         DiabetesPedigreeFunction    0
         Age                         0
         Outcome                     0
         dtype: int64
```

Here from the above code we first checked that is there any null values from the IsNull() function then we are going to take the sum of all those missing values from the sum() function and the inference we now get is that there are no missing values but that is actually not a true story as in this particular dataset all the missing values were given the 0 as a value which is not good for the authenticity of the dataset. Hence we will first replace the 0 value with the NAN value then start the imputation process.

PROGRAM:

```
diabetes_df_copy = diabetes_df.copy(deep = True)
diabetes_df_copy[['Glucose','BloodPressure','SkinThickness','Insu-
lin','BMI']] = diabetes_df_copy[['Glucose','BloodPressure','Skin-
Thickness','Insulin','BMI']].replace(0,np.NaN)

# Showing the Count of NANs
print(diabetes_df_copy.isnull().sum())
```

OUTPUT :



```
In [11]: diabetes_df_copy = diabetes_df.copy(deep = True)
         diabetes_df_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']] = diabetes_df_copy[['Gluc

         # Showing the Count of NANs
         print(diabetes_df_copy.isnull().sum())

         Pregnancies                   0
         Glucose                       5
         BloodPressure                35
         SkinThickness               227
         Insulin                     374
         BMI                          11
         DiabetesPedigreeFunction      0
         Age                           0
         Outcome                       0
         dtype: int64

In [ ]:
```

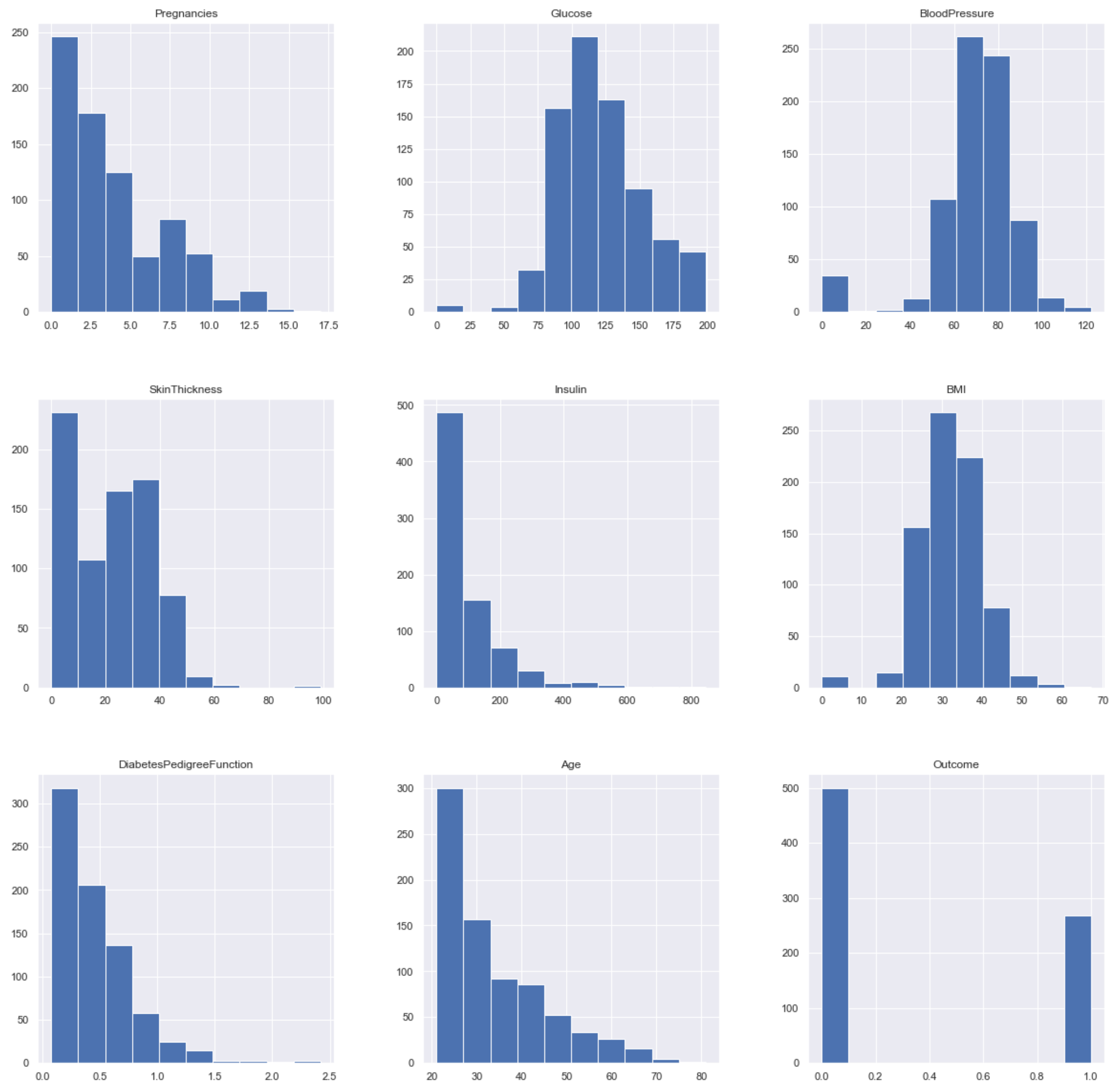
As mentioned above that now we will be replacing the zeros with the NAN values so that we can impute it later to maintain the authenticity of the dataset as well as trying to have a better Imputation approach i.e to apply mean values of each column to the null values of the respective columns.

<u>4.Data Visualization:</u>

Plotting the data distribution plots before removing null values

```
p = diabetes_df.hist(figsize = (20,20))
```

<u>Output:</u>



Inference: So here we have seen the distribution of each features whether it is dependent data or independent data and one thing which could always strike that why do we need to see the distribution of data? So the answer is simple it is the best way to start the analysis of the dataset as it shows the occurrence of every kind of value in the graphical structure which in turn lets us know the range of the data.

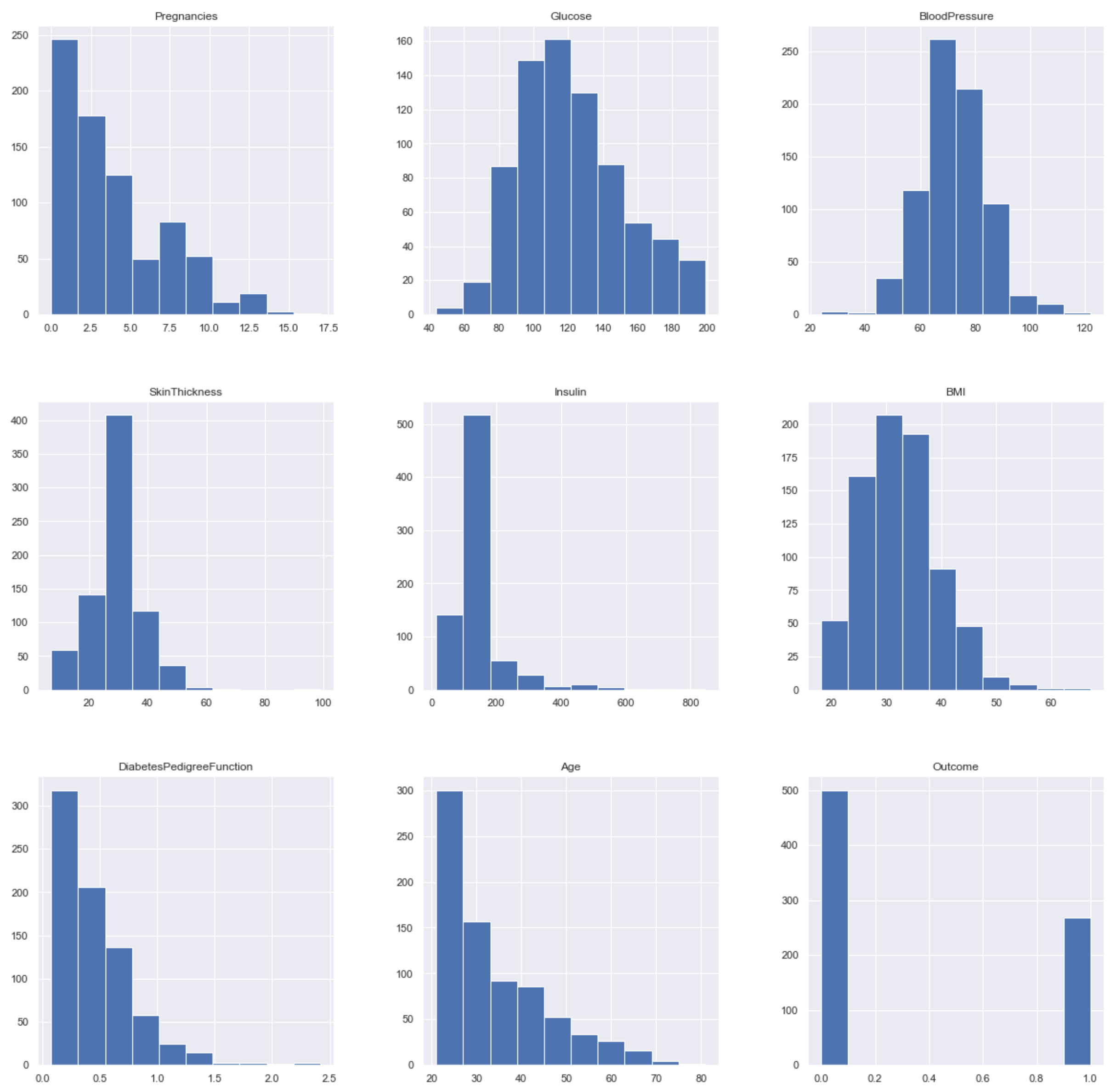
Now we will be imputing the mean value of the column to each missing value of that particular column.

```
diabetes_df_copy['Glucose'].fillna(diabetes_df_copy['Glu-
cose'].mean(), inplace = True)
diabetes_df_copy['BloodPressure'].fillna(diabetes_df_copy['Blood-
Pressure'].mean(), inplace = True)
diabetes_df_copy['SkinThickness'].fillna(diabetes_df_copy['Skin-
Thickness'].median(), inplace = True)
diabetes_df_copy['Insulin'].fillna(diabetes_df_copy['Insulin'].me-
dian(), inplace = True)
diabetes_df_copy['BMI'].fillna(diabetes_df_copy['BMI'].median(),
inplace = True)
```

Plotting the distributions after removing the NAN values.

```
p = diabetes_df_copy.hist(figsize = (20,20))
```

Output:

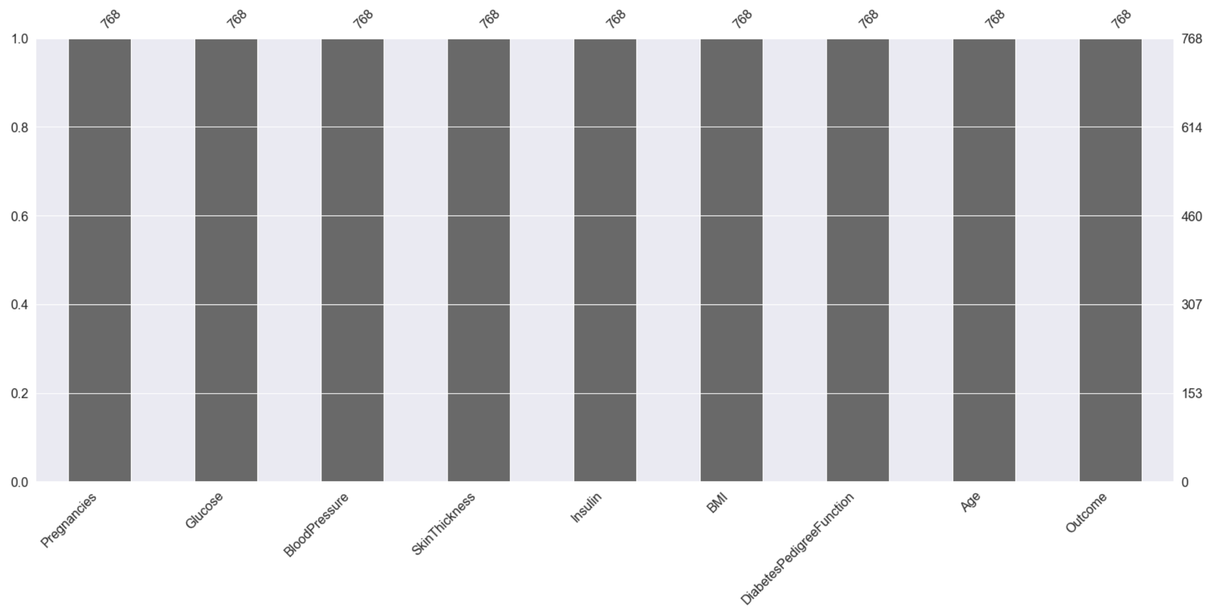


Inference: Here we are again using the hist plot to see the distribution of the dataset but this time we are using this visualization to see the changes that we can see after those null values are removed from the dataset and we can clearly see the difference for example – In age column after removal of the null values, we can see that there is a spike at the range of 50 to 100 which is quite logical as well.

Plotting Null Count Analysis Plot

```
p = msno.bar(diabetes_df)
```

Output:



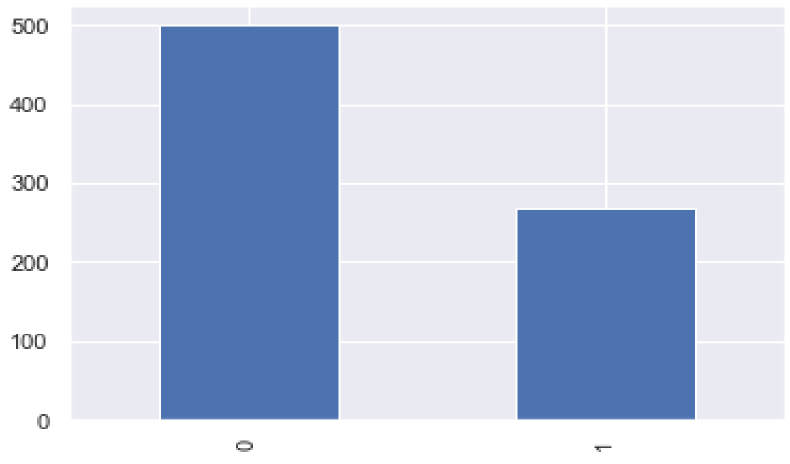
Inference: Now in the above graph also we can clearly see that there are no null values in the dataset.

Now, let's check that how well our outcome column is balanced

```
color_wheel = {1: "#0392cf", 2: "#7bc043"}
colors = diabetes_df["Outcome"].map(lambda x: color_wheel.get(x +
1))
print(diabetes_df.Outcome.value_counts())
p=diabetes_df.Outcome.value_counts().plot(kind="bar"
```

Output:

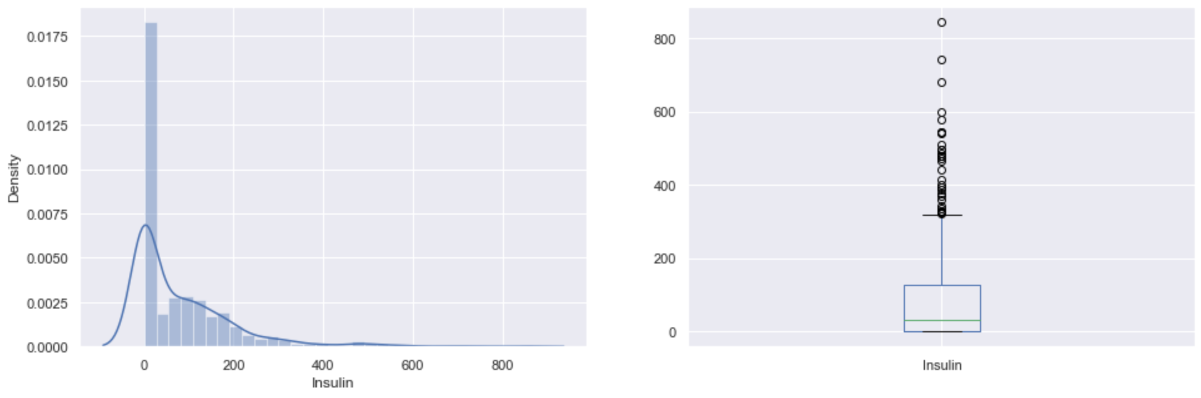
```
0    500
1    268
Name: Outcome, dtype: int64
```



Inference: Here from the above visualization it is clearly visible that our dataset is completely imbalanced in fact the number of patients who are diabetic is half of the patients who are non-diabetic.

```
plt.subplot(121), sns.distplot(diabetes_df['Insulin'])
plt.subplot(122), diabetes_df['Insulin'].plot.box(figsize=(16,5))
plt.show()
```

<u>Output:</u>



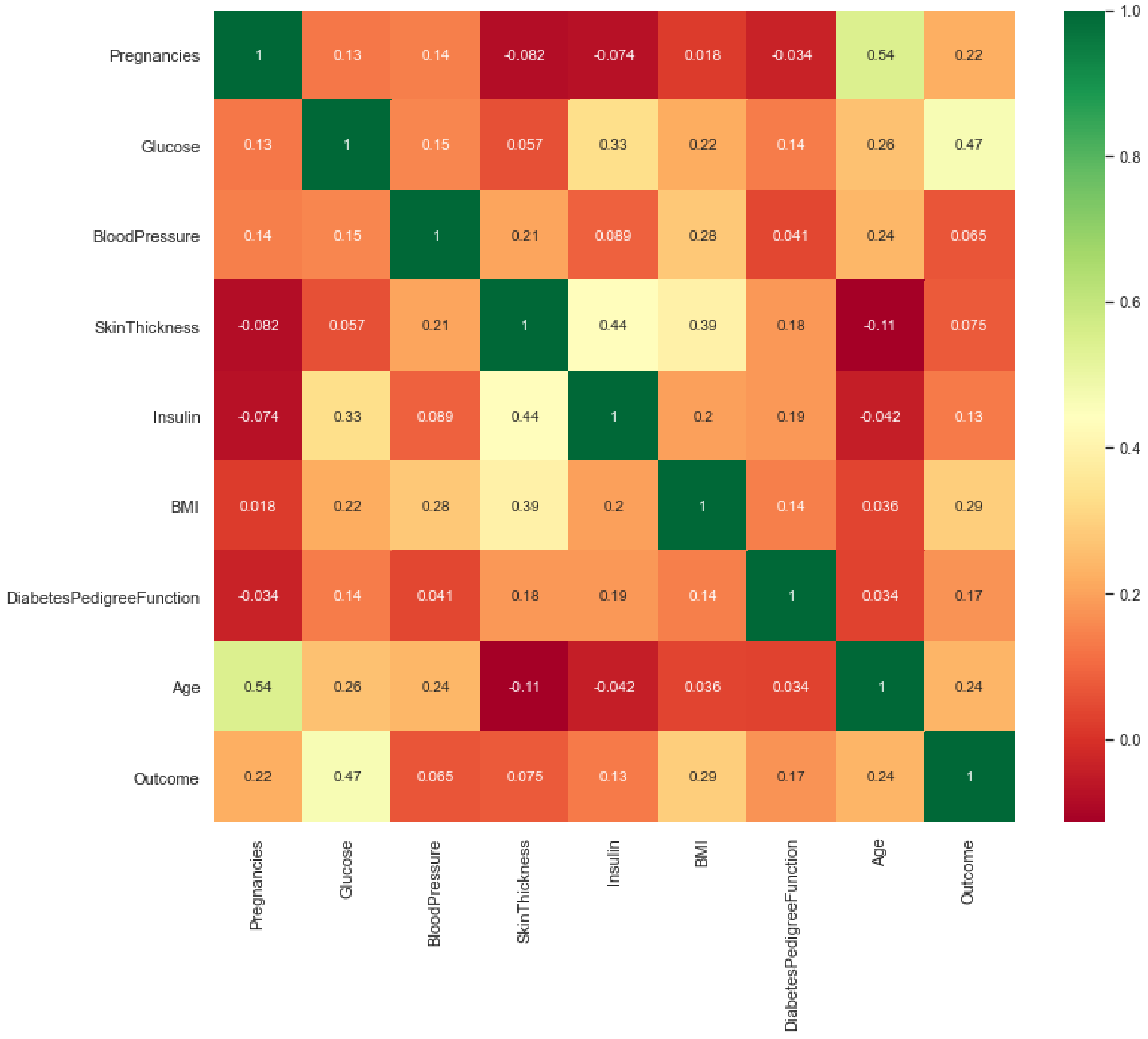
Inference: That's how Distplot can be helpful where one will able to see the distribution of the data as well as with the help of boxplot one can see the outliers in that column and other information too which can be derived by the box and whiskers plot.

<u>5.Correlation between all the features:</u>

Correlation between all the features before cleaning

```
plt.figure(figsize=(12,10))
# seaborn has an easy method to showcase heatmap
p = sns.heatmap(diabetes_df.corr(), annot=True,cmap ='RdYlGn')
```

Output:

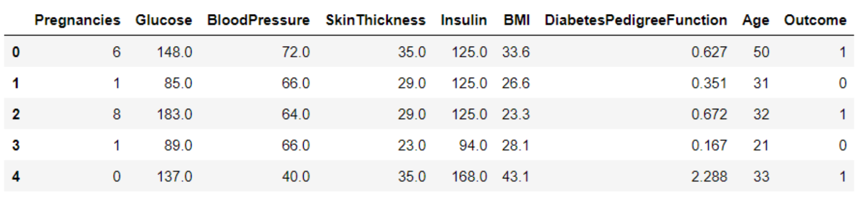


6.Scaling the Data:

Before scaling down the data let’s have a look into it

```
diabetes_df_copy.head()
```

Output:

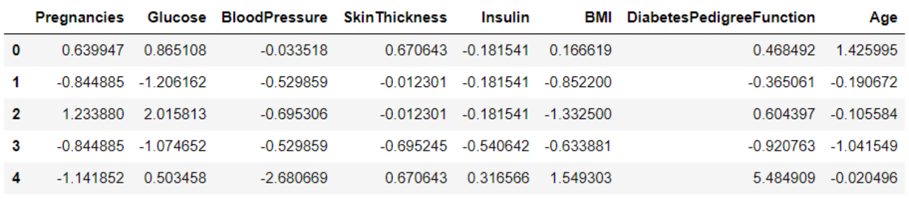


	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.0	72.0	35.0	125.0	33.6	0.627	50	1
1	1	85.0	66.0	29.0	125.0	26.6	0.351	31	0
2	8	183.0	64.0	29.0	125.0	23.3	0.672	32	1
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21	0
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33	1

After Standard scaling

```
sc_X = StandardScaler()
X =  pd.DataFrame(sc_X.fit_transform(diabetes_df_copy.drop(["Out-
come"],axis = 1),), columns=['Pregnancies',
'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'Dia-
betesPedigreeFunction', 'Age'])
X.head()
```

Output:



	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	0.639947	0.865108	-0.033518	0.670643	-0.181541	0.166619	0.468492	1.425995
1	-0.844885	-1.206162	-0.529859	-0.012301	-0.181541	-0.852200	-0.365061	-0.190672
2	1.233880	2.015813	-0.695306	-0.012301	-0.181541	-1.332500	0.604397	-0.105584
3	-0.844885	-1.074652	-0.529859	-0.695245	-0.540642	-0.633881	-0.920763	-1.041549
4	-1.141852	0.503458	-2.680669	0.670643	0.316566	1.549303	5.484909	-0.020496

That's how our dataset will be looking like when it is scaled down or we can see every value now is on the same scale which will help our ML model to give a better result.

Output:

```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

7.Model Building:

Splitting the dataset:

```
X = diabetes_df.drop('Outcome', axis=1)
y = diabetes_df['Outcome']
```

Now we will split the data into training and testing data using the train_test_split function

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,
test_size=0.33,random_state=7)
```

8.Random Forest:

Building the model using RandomForest

```
from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(n_estimators=200)
rfc.fit(X_train, y_train)
```

Now after building the model let's check the accuracy of the model on the training dataset.

```
rfc_train = rfc.predict(X_train)
from sklearn import metrics

print("Accuracy_Score =", format(metrics.accuracy_score(y_train,
rfc_train)))
```

Output: Accuracy = 1.0

So here we can see that on the training dataset our model is overfitted.

Getting the accuracy score for Random Forest

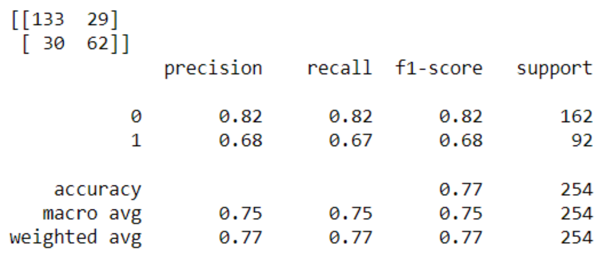
```
from sklearn import metrics

predictions = rfc.predict(X_test)
print("Accuracy_Score =", format(metrics.accuracy_score(y_test, pre-
dictions)))
```

Output:

```
Accuracy_Score = 0.7677165354330708
```

Classification report and confusion matrix of random forest model

Output:



```
[[133  29]
 [ 30  62]]
              precision    recall  f1-score   support

           0       0.82      0.82      0.82       162
           1       0.68      0.67      0.68        92

    accuracy                           0.77       254
   macro avg       0.75      0.75      0.75       254
weighted avg       0.77      0.77      0.77       254
```

9.Decision Tree:

Building the model using DecisionTree

```
from sklearn.tree import DecisionTreeClassifier

dtree = DecisionTreeClassifier()
dtree.fit(X_train, y_train)
```

Now we will be making the predictions on the **testing data** directly as it is of more importance.

Getting the accuracy score for Decision Tree

```
from sklearn import metrics

predictions = dtree.predict(X_test)
print("Accuracy Score =", format(metrics.accuracy_score(y_test,pre-
dictions)))
```

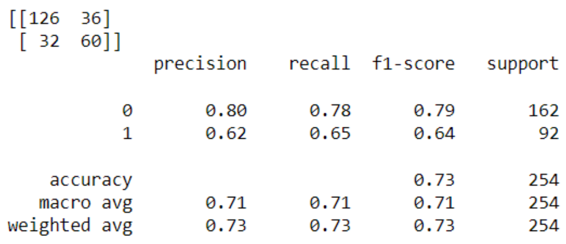
Output:

```
Accuracy Score = 0.7322834645669292
```

Classification report and confusion matrix of the decision tree model

```
from sklearn.metrics import classification_report, confusion_matrix


print(confusion_matrix(y_test, predictions))
print(classification_report(y_test,predictions))
```

Output:



```
[[126  36]
 [ 32  60]]
              precision    recall  f1-score   support

           0       0.80      0.78      0.79       162
           1       0.62      0.65      0.64        92

    accuracy                           0.73       254
   macro avg       0.71      0.71      0.71       254
weighted avg       0.73      0.73      0.73       254
```

10. XgBoost classifier:

Building model using XGBoost

```
from xgboost import XGBClassifier

xgb_model = XGBClassifier(gamma=0)
xgb_model.fit(X_train, y_train)
```

Output:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=4, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

Now we will be making the predictions on the testing data directly as it is of more importance.

Getting the accuracy score for the XgBoost classifier

```
from sklearn import metrics

xgb_pred = xgb_model.predict(X_test)
print("Accuracy Score =", format(metrics.accuracy_score(y_test,
xgb_pred)))
```

Output:

```
Accuracy Score = 0.7401574803149606
```

Classification report and confusion matrix of the XgBoost classifier

Output:

```
[[127  35]
 [ 31  61]]
              precision    recall  f1-score   support

           0       0.80      0.78      0.79       162
           1       0.64      0.66      0.65        92

    accuracy                           0.74       254
   macro avg       0.72      0.72      0.72       254
weighted avg       0.74      0.74      0.74       254
```

11.Support Vector Machine (SVM):

Building the model using Support Vector Machine (SVM)

```
from sklearn.svm import SVC
```

```
svc_model = SVC()
svc_model.fit(X_train, y_train)
```

Prediction from support vector machine model on the testing data

```
svc_pred = svc_model.predict(X_test)
```

Accuracy score for SVM

```
from sklearn import metrics

print("Accuracy Score =", format(metrics.accuracy_score(y_test,
svc_pred)))
```

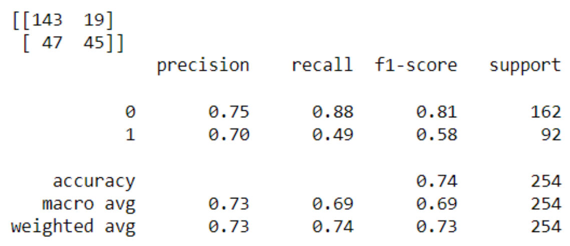
<u>Output:</u>

```
Accuracy Score = 0.7401574803149606
```

Classification report and confusion matrix of the SVM classifier

```
from sklearn.metrics import classification_report, confusion_matrix

print(confusion_matrix(y_test, svc_pred))
print(classification_report(y_test,svc_pred))
```

<u>Output:</u>



```
[[143  19]
 [ 47  45]]
              precision    recall  f1-score   support

           0       0.75      0.88      0.81       162
           1       0.70      0.49      0.58        92

    accuracy                           0.74       254
   macro avg       0.73      0.69      0.69       254
weighted avg       0.73      0.74      0.73       254
```

<u>The Conclusion from Model Building</u>

Therefore Random forest is the best model for this prediction since it has an accuracy_score of 0.76.

Feature Importance

Knowing about the feature importance is quite necessary as it shows that how much weightage each feature provides in the model building phase.

Getting feature importances

```
rfc.feature_importances_
```

Output:

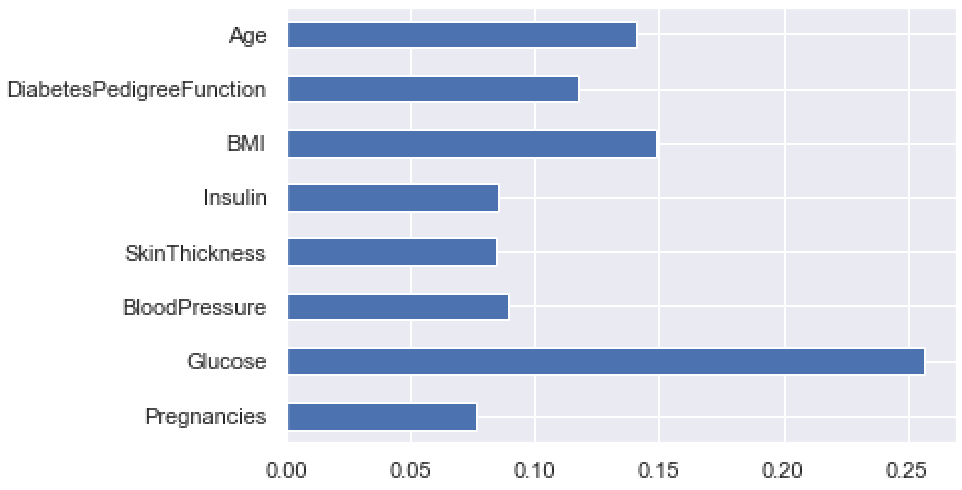
```
array([0.07684946, 0.25643635, 0.08952599, 0.08437176, 0.08552636,
       0.14911634, 0.11751284, 0.1406609 ])
```

From the above output, it is not much clear that which feature is important for that reason we will now make a visualization of the same.

Plotting feature importances

```
(pd.Series(rfc.feature_importances_, index=X.col-
umns).plot(kind='barh'))
```

Output:

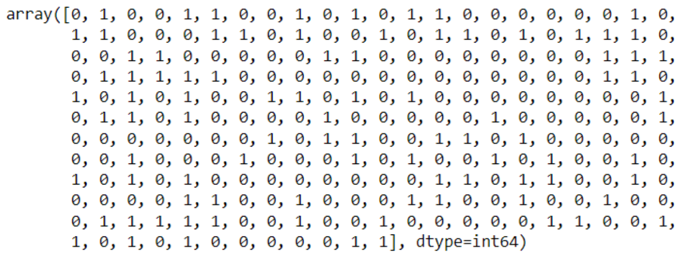


Here from the above graph, it is clearly visible that Glucose as a feature is the most important in this dataset.

Saving Model – Random Forest

```
import pickle

# Firstly we will be using the dump() function to save the model us-
ing pickle

saved_model = pickle.dumps(rfc)


# Then we will be loading that saved model

rfc_from_pickle = pickle.loads(saved_model)


# lastly, after loading that model we will use this to make predic-
tions

rfc_from_pickle.predict(X_test)
```

Output:

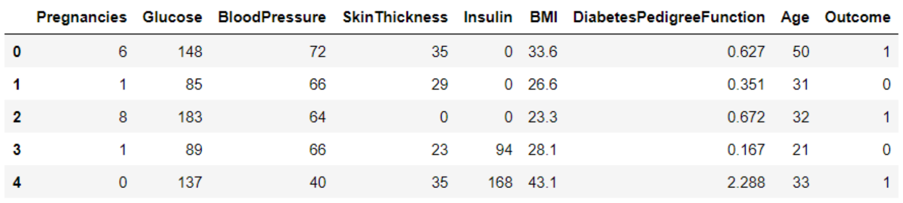


```
array([0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0,
       1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
       1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0,
       1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0,
       0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0,
       0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1,
       1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1], dtype=int64)
```

Now for the last time, I'll be looking at the head and tail of the dataset so that we can take any random set of features from both the head and tail of the data to test that if our model is good enough to give the right prediction.

```
diabetes_df.head()
```

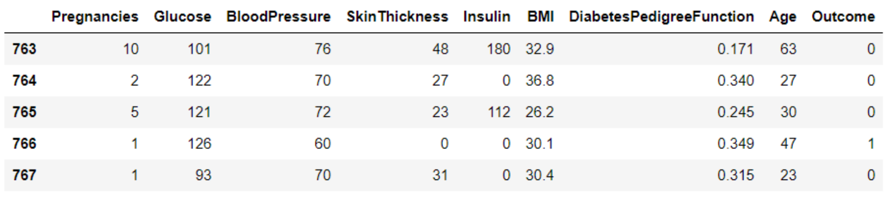
Output:



	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

```
diabetes_df.tail()
```

Output:



	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

Putting data points in the model will either return 0 or 1 i.e. person suffering from diabetes or not.

```
rfc.predict([[0,137,40,35,168,43.1,2.228,33]]) #4th patient
```

Output:

Another one

```
rfc.predict([[10,101,76,48,180,32.9,0.171,63]])  # 763 th patient
```

Output:

```
array([0], dtype=int64)
```

CONCLUSION:

Certainly! In conclusion, data preprocessing is a crucial step in the development of an AI-based diabetes prediction system. It plays a pivotal role in ensuring the quality, accuracy, and reliability of the predictive model. By carefully cleaning, transforming, and preparing the raw data, we can address issues such as missing values, outliers, and noise, ultimately enhancing the model's performance.

In summary, data preprocessing lays the foundation for building a robust and accurate diabetes prediction system. It ensures that the AI model can effectively learn from the data and make reliable predictions, ultimately contributing to improved healthcare outcomes and patient well-being.