AI-BASED DIABETES PREDICTION SYSTEM

PHASE-3 DOCUMENT SUBMISSION

Register Number: 963521104017

Name:GURU RAHUL RU

Project Title : DIABETES PREDICTION SYSTEM

- **INTRODUCTION**

In today's rapidly evolving world, healthcare stands on the cusp of a technological revolution. Artificial Intelligence (AI) is at the forefront of this revolution, empowering healthcare professionals with innovative tools to enhance diagnosis, treatment, and patient care. One significant application of AI in healthcare is the prediction and prevention of chronic diseases, such as diabetes, which affects millions of people worldwide.

Diabetes, a chronic metabolic disorder characterized by elevated blood sugar levels, has reached epidemic proportions, posing a significant challenge to global healthcare systems. Early detection and proactive management of diabetes can significantly improve patients' quality of life and reduce the burden on healthcare resources. This is where the integration of AI technologies becomes invaluable.

Diabetes, a chronic metabolic disorder characterized by elevated blood sugar levels, has reached epidemic proportions, posing a significant challenge to global healthcare systems. Early detection and proactive management of diabetes can significantly improve patients' quality of life and reduce the burden on healthcare resources. This is where the integration of AI technologies becomes invaluable.

- Given Data Set

```
In [3]:
data.head()  #displaying the head of dataset
Out[3]:
```

	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

- Necessary Steps to Follow
- Import necessary libraries

1 Start by importing the necessary libraries

Program:

import pandas as pd

import numpy as np

from sklearn.model_selection

import train_test_split from sklearn.preprocessing

import StandardScaler

2 Load the Dataset:

Load your dataset into a Pandas DataFrame. You can typically find diabetes patients symptoms datasets in CSV format, but you can adapt this code to other formats as needed.

Program:

df = pd.read_csv(' E:\USA_Diabetes.csv ')

Pd.read()

3. Exploratory Data Analysis (EDA): Perform EDA to understand your data better. This includes checking for missing values, exploring the data's statistics, and visualizing it to identify patterns.

Program:

# Check for missing values

print(df.isnull().sum())

# Explore statistics print(df.describe())

# Visualize the data (e.g., histograms, scatter plots, etc.)

4. Feature Engineering:

Depending on your dataset, you may need to create new features or transforming the existing ones this can involve in the one hot encoding the categorial variables, handling date / time data.or scaling the numerical features.

Importance of Loading and Processing the Data Set

1. Data Quality Assurance:

Identification of Errors: Loading data allows you to identify missing values, outliers, and other errors in the dataset. Addressing these issues is vital for accurate analysis.

Data Cleaning: Processing the data involves cleaning, where you handle missing or inconsistent data points, ensuring that the dataset is reliable and accurate.

2. Understanding the Data:

Exploratory Data Analysis (EDA): Loading data enables you to perform EDA. Understanding the distribution, relationships, and patterns in the data is critical for making informed decisions about the analysis or modeling techniques to use.

3. Feature Engineering:

Creation of Relevant Features: Processing the data allows you to create new features derived from the existing ones. Properly engineered features can significantly enhance the performance of machine learning models.

4. Data Transformation:

Normalization and Scaling: Data might need to be normalized or scaled to ensure that all features contribute equally to the analysis or modeling process.

Categorical Data Encoding: Processing includes converting categorical variables into a format suitable for machine learning algorithms, such as one-hot encoding or label encoding.

5. Model Performance:

Impact on Model Accuracy: The quality of the dataset directly impacts the accuracy and reliability of machine learning models. Well-preprocessed data often leads to better-performing models.

Reduction of Overfitting: Proper processing, such as feature selection and dimensionality reduction, can help prevent overfitting by removing noise from the data.

6. Computational Efficiency:

Reduced Processing Time: Large datasets, especially in big data scenarios, can be computationally intensive. Proper processing can reduce the time it takes to train models and perform analyses.

Optimized Memory Usage: Processing can involve optimizing data types and structures to minimize memory usage, enabling the analysis to be performed on a broader scale.

7. Decision Making:

Informed Decision-Making: Accurate and well-processed data ensures that the decisions made based on the analysis are reliable, leading to better business strategies, problem-solving, and outcomes.

8. Reproducibility and Collaboration:

Reproducibility: Detailed documentation of the data processing steps ensures that the analysis can be reproduced, verified, and validated by other researchers or team members.

Collaboration: Processed datasets facilitate collaboration among team members, as everyone is working with a consistent and well-understood set of data.

Challenges and Solutions involved in Preprocessing the Diabetes Prediction Dataset

Preprocessing the diabetes prediction dataset, like any other dataset, comes with its set of challenges. Here are some common challenges specifically associated with preprocessing the diabetes prediction dataset:

1. Missing Values:

Challenge: Datasets often have missing values, and dealing with them is crucial. In the diabetes prediction dataset, missing values can occur in various fields like glucose levels, blood pressure, or BMI.

Solution: Techniques such as imputation (filling missing values with estimated values) or removing rows/columns with missing values can be applied. The choice depends on the extent of missing data and its impact on the analysis.

2. Imbalanced Data:

Challenge: The diabetes prediction dataset might be imbalanced, i.e., one class (diabetic or non-diabetic) significantly outnumbers the other. This can lead to biased models.

Solution: Techniques like oversampling the minority class, undersampling the majority class, or using synthetic data generation methods (SMOTE) can balance the class distribution.

3. Outliers:

Challenge: Outliers in features such as glucose levels or BMI can skew the analysis and models.

Solution: Outliers can be detected using statistical methods (like Z-score) and removed or transformed (e.g., log transformation) to mitigate their impact on the analysis.

4. Feature Scaling:

Challenge: Features in the diabetes dataset may have different scales, affecting the performance of machine learning algorithms.

Solution: Features can be scaled using techniques like Min-Max scaling or standardization (Z-score normalization) to bring them to a similar scale, ensuring fair comparison and accurate modeling.

5. Categorical Data:

Challenge: The dataset might contain categorical variables (like gender) that need to be converted into a numerical format for machine learning algorithms.

Solution: Techniques like one-hot encoding or label encoding can be applied to convert categorical variables into a numerical format.

6. Feature Selection:

Challenge: The dataset may have irrelevant or redundant features, leading to overfitting or increased computational complexity.

Solution: Feature selection techniques like correlation analysis, recursive feature elimination, or feature importance from models can be used to select the most relevant features for prediction.

7. Data Standardization:

Challenge: Data might come from different sources or have varying formats, requiring standardization.

Solution: Standardizing data formats and units ensures consistency and accuracy in the analysis. For instance, ensuring consistent units for features like glucose levels and blood pressure is vital.

8. Data Leakage:

Challenge: Information from the test set accidentally influencing the preprocessing steps can lead to data leakage and overly optimistic model evaluations.

Solution: Ensure that preprocessing steps are applied separately to the training and test datasets to prevent data leakage. Techniques like cross-validation can help in evaluating the model's performance effectively.

1.Loading the dataset:

Loading the dataset using machine learning is the process of bringing the data into the machine learning environment so that it can be used to train and evaluate a model.

The specific steps involved in loading the dataset will vary depending on the machine learning library or framework that is being used. However, there are some general steps that are common to most machine learning frameworks:

a.Identify the dataset:

The first step is to identify the dataset that you want to load. This dataset may be stored in a local file, in a database, or in a cloud storage service.

b.Load the dataset:

Once you have identified the dataset, you need to load it into the machine learning environment. This may involve using a built-in function in the machine learning library, or it may involve writing your own code.

c.Preprocess the dataset:

Once the dataset is loaded into the machine learning environment, you may need to preprocess it before you can start training and evaluating your model. This may involve cleaning the data, transforming the data into a suitable format, and splitting the data into training and test sets. Preprocess the dataset Load the dataset Identify the dataset Loading the dataset

Here, how to load a dataset using machine learning in Python

Program:

```
import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score,
mean_absolute_error,mean_squared_error
from sklearn.linear_model import Linear Regression
from sklearn.linear_model
import Lasso from sklearn.ensemble
import RandomForestRegressor
from sklearn.svm import SVR
import xgboost as xg
%matplotlib inline import warnings
warnings.filterwarnings("ignore")
```

```
/opt/conda/lib/python3.10/sitepackages/scipy/__init__.py:146
: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is
required for this version of SciPy (detected version 1.23.5
warnings.warn(f"A NumPy version >={np_minversion} and
<{np_maxversion}"
```

Loading Dataset:

```
dataset = pd.read_csv('E:/USA_Housing.csv')
```

Data Exploration:

DataSet :

OUTPUT

```
                                                       In [3]:
data.head()  #displaying the head of dataset
```

```
                                                       Out[3]:
```

	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

2.Preprocessing the dataset:

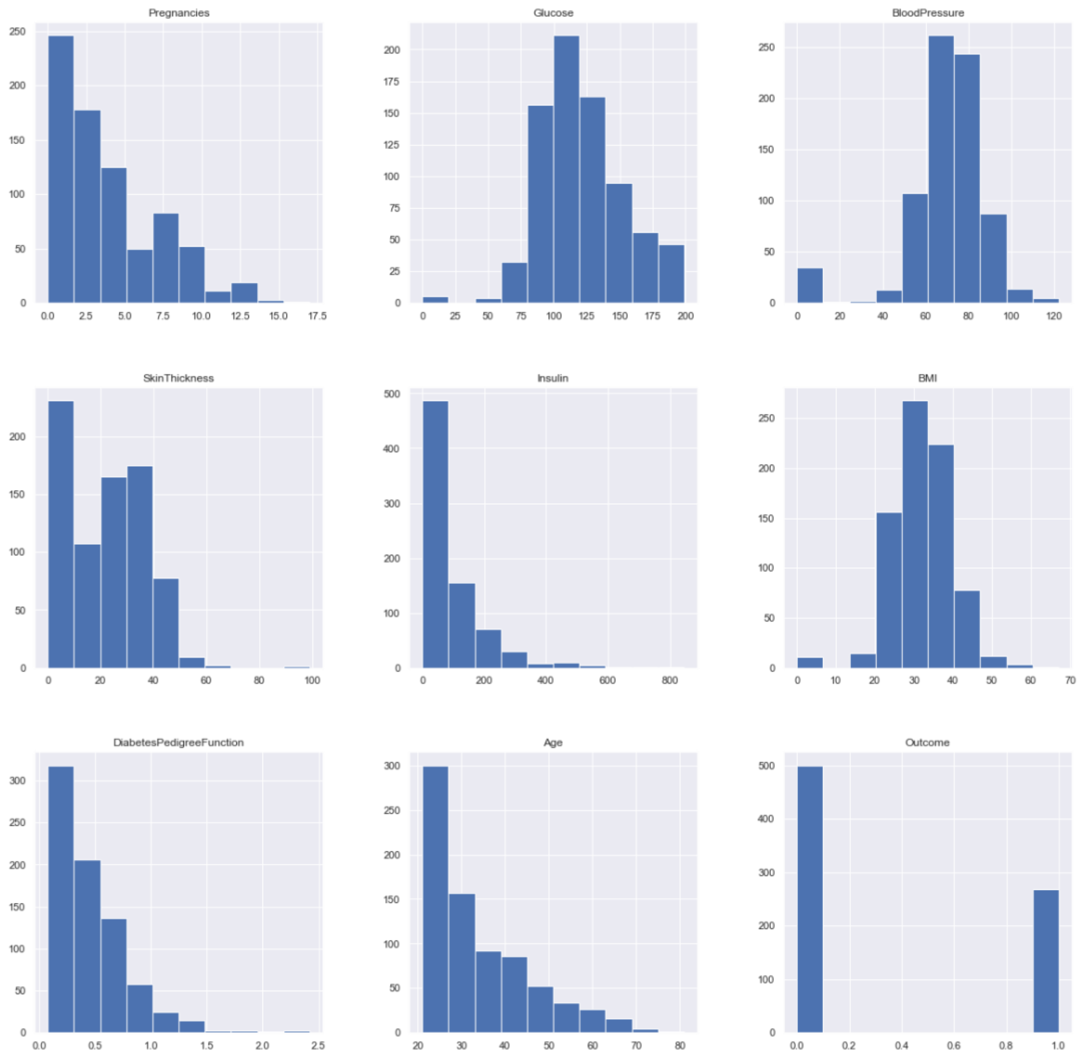
Data preprocessing is the process of cleaning, transforming, and integrating data in order to make it ready for analysis. This may involve removing errors and inconsistencies, handling missing values, transforming the data into a consistent format, and scaling the data to a suitable range.

Data Visualization

Plotting the data distribution plots before removing null values

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

Output:



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.

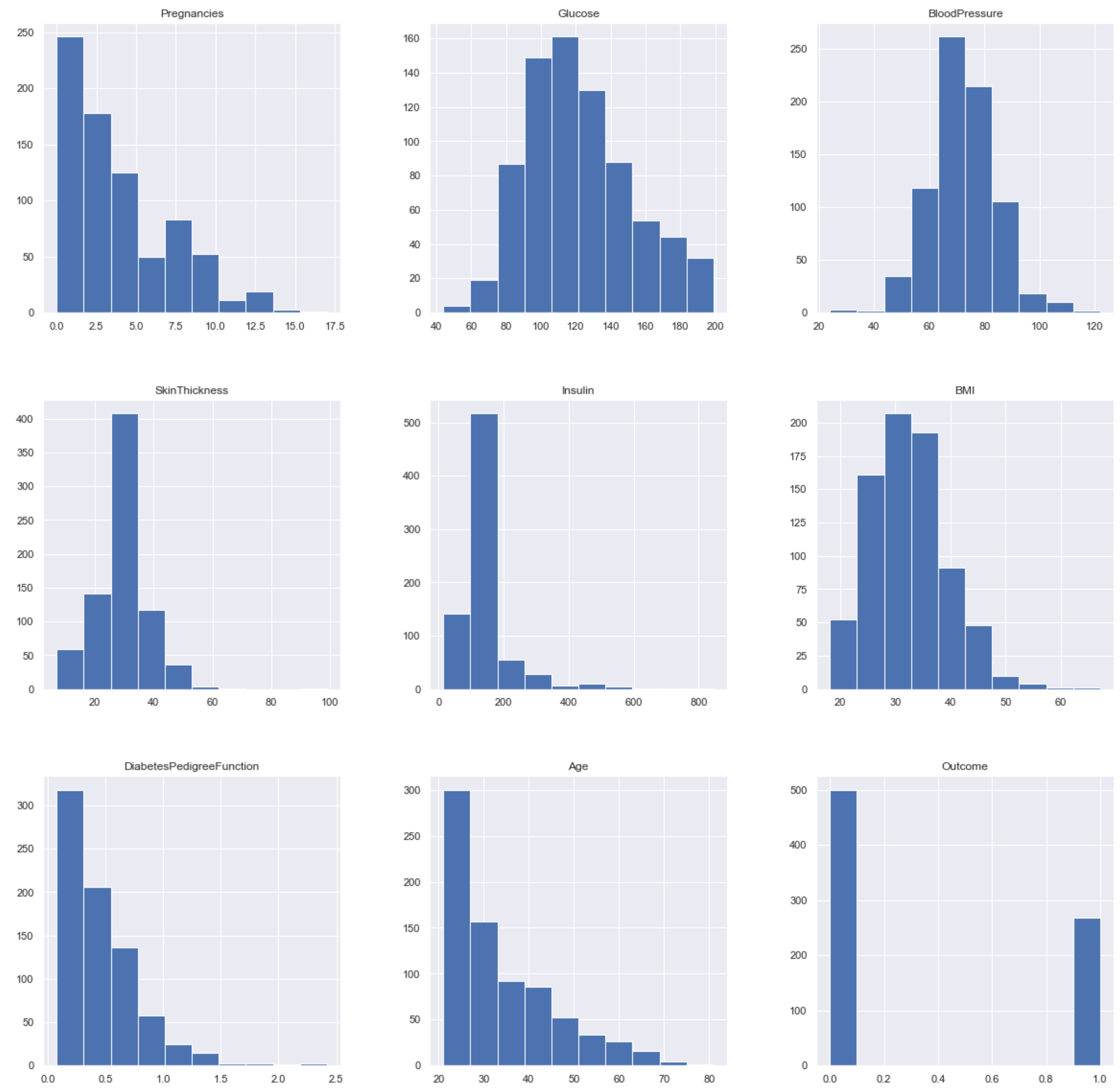
Now we will be imputing the mean value of the column to each missing value of that particular column.

```
diabetes_df_copy['Glucose'].fillna(diabetes_df_copy['Glucose'].
mean(), inplace = True)
diabetes_df_copy['BloodPressure'].fillna(diabetes_df_copy['Blo
odPressure'].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'].m
edian(), 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



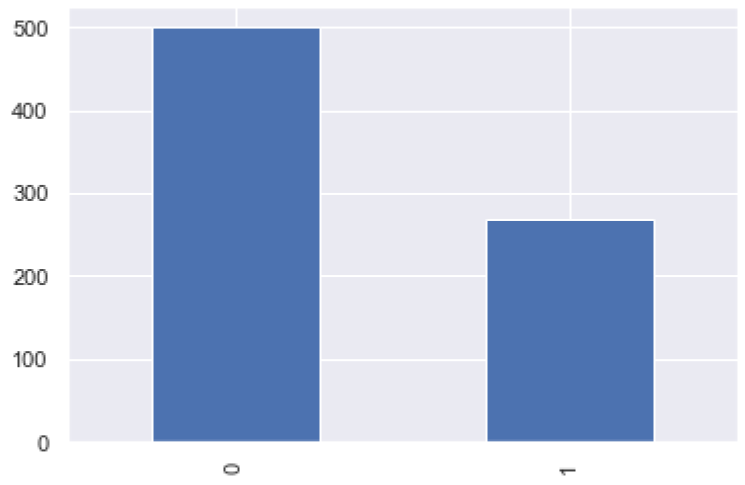
Plotting Null Count Analysis Plot

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:

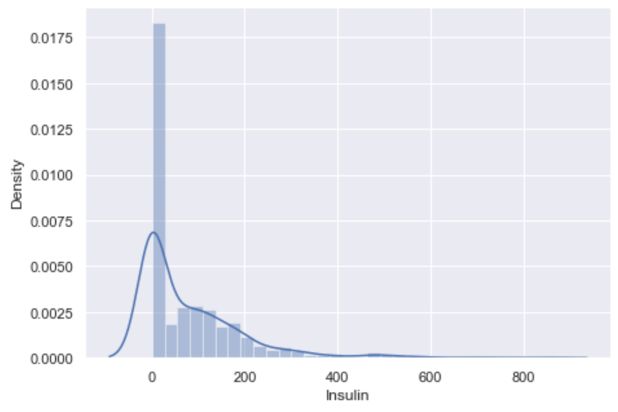
```
0    500
1    268
Name: Outcome, dtype: int64
```

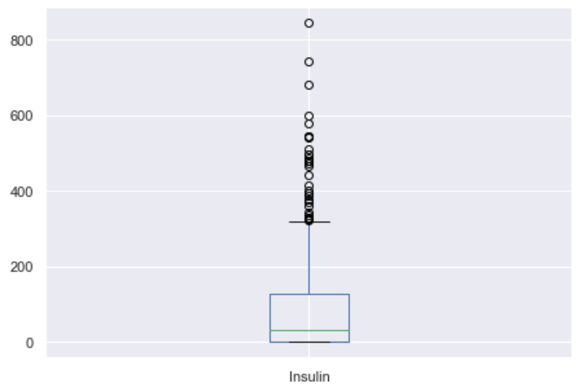


Inference: Here from the above visualization it is clearly visible that our **dataset is completely imbalanced** in fact the number of patients who are **diabetic is half of the patients who are non-diabetic.**

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

Output:





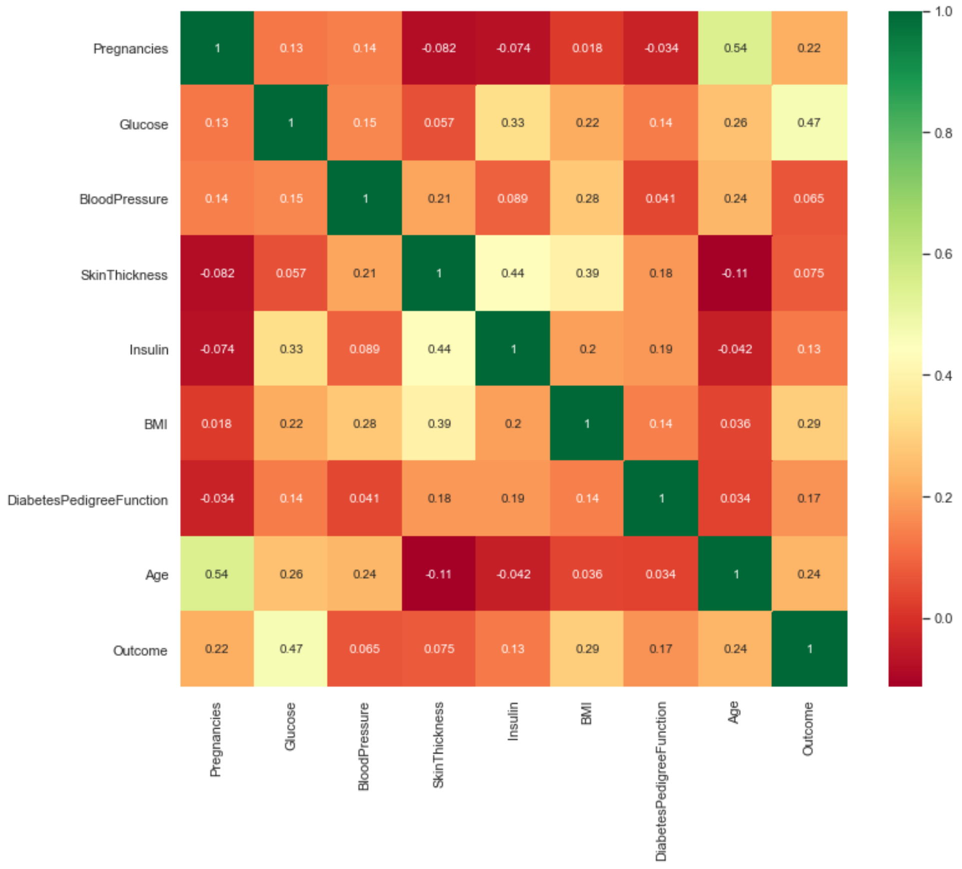
Inference: That's how **Distplot** can be helpful where one will able to see the distribution of the data as well as with the help of **boxplot one can see the outliers in that column** and other information too which can be derived by the **box and whiskers plot.**

Correlation between all the features

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:

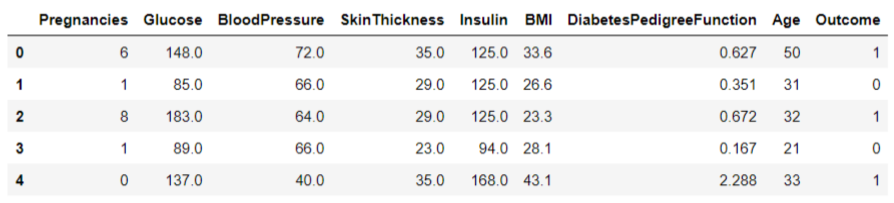


Scaling the Data

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

```
diabetes_df_copy.head()
```

Output:

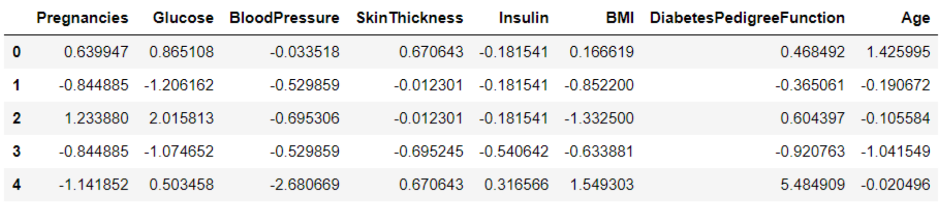


	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.0	72.0	35.0	125.0	33.6	0.627	50	1
1	1	85.0	66.0	29.0	125.0	26.6	0.351	31	0
2	8	183.0	64.0	29.0	125.0	23.3	0.672	32	1
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21	0
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33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

After Standard scaling

```
sc_X = StandardScaler()
X =
pd.DataFrame(sc_X.fit_transform(diabetes_df_copy.drop(["Outcome"],axis =
1),), columns=['Pregnancies',
'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI',
'DiabetesPedigreeFunction', '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.**

Let's explore our target column

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
```

Program

Algorithm 1:

Diabetes Prediction using various machine learning algorithms Generate training set and test set randomly. Specify algorithms that are used in model

```
mn=[KNN(       ),DTC(      ),GaussianNB(  ),
LDA(),SVC(),LinearSVC(),AdaBoost(),
RandomForestClassifier(),Perceptron(),
ExtraTreeClassifier(), Bagging(),
 LogisticRegression(), GradientBoostClassifier()]
 for(i=0; i<13; i++) do Model= mn[i];
Model.fit();
model.predict();
print(Accuracy(i),confusion_matrix, classification_report);
```

Algorithm 2:

Diabetes Prediction using pipeline

Step1: Import required libraries.

Step2: Import diabetes dataset.

Step3: Create pipeline for algorithms giving highest accuracy.
Step4: Add theses pipeline to a dictionary where all pipelines will be stored.

Step5: Fit the pipelines in training dataset.

Step6: Compare accuracies of all pipelines added.

Step7: Prediction and identification of the most accurate model will be done on test data. Pipelines work by allowing for a linear sequence of data transforms to be chained together culminating in a modelling process that can be evaluated. The goal is to ensure that all of the steps in the pipeline are constrained to the data available for the evaluation, such as the training dataset or each fold of the cross-validation procedure.

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, predictions)))
```

Output:

```
Accuracy_Score = 0.7677165354330708
```

Output:

```
  weighted avg       0.77      0.77      0.77       254
```

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,prediction
s)))
```

Output:

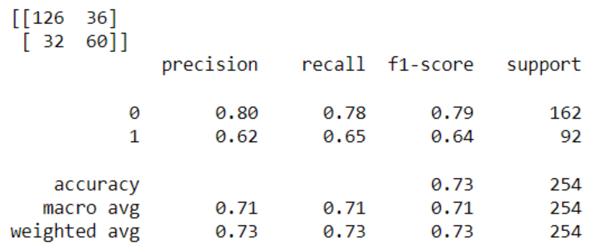
```
Accuracy Score = 0.7322834645669292
```

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
```

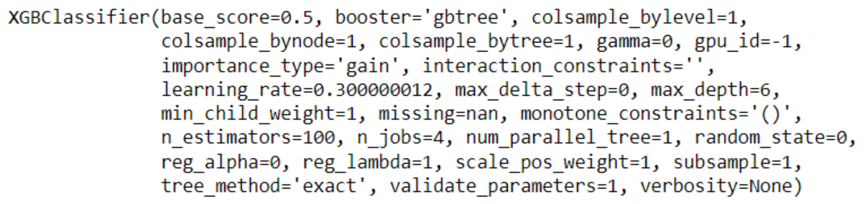
XgBoost classifier

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
```

Support Vector Machine (SVM)

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)))

Output:

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))
```

Output:

```
from sklearn.metrics import classification_report,
confusion_matrix
```

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

Output:

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)))
```

Output:

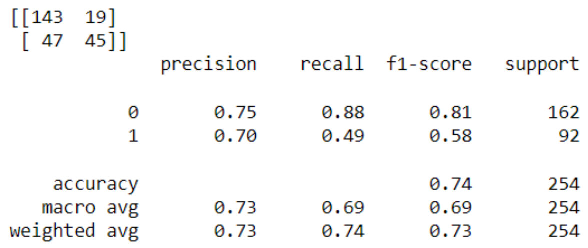
```
Accuracy Score = 0.7401574803149606
```

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))
```

Output:



```
[[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
```

CONCLUSION

In summary, diabetes prediction through machine learning offers a promising avenue for early detection and intervention.

By analyzing relevant data, these models can identify individuals at risk, enabling timely preventive measures. With ongoing advancements, this approach holds the potential to significantly improve healthcare outcomes, making a positive impact on public health by effectively managing diabetes and reducing its associated complications.