DOMAIN NAME: ARTIFICIAL INTELLIGENCE

PROJECT NAME : DIABETES PREDICTION SYSTEM USING AI

PHASE 4: DEVELOPMENT PART 2

OVERVIEW:

we will be predicting that whether the patient has diabetes or not on the basis of the features we will provide to our machine learning model, and for that, we will be using the famous Pima Indians Diabetes Database.

1. *Data analysis: Here one will get to know about how the data analysis part is done in a data science life cycle.*
2. *Exploratory data analysis: EDA is one of the most important steps in the data science project life cycle and here one will need to know that how to make inferences from the visualizations and data analysis*
3. *Model building: Here we will be using 4 ML models and then we will choose the best performing model.*
4. *Saving model: Saving the best model using pickle to make the prediction from real data.*

Importing Libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as snssns.set()

from mlxtend.plotting import plot_decision_regions

import missingno as msno

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

Here we will be reading the dataset which is in the CSV format

diabetes_df = pd.read_csv('diabetes.csv')
diabetes_df.head()
Output:

Exploratory Data Analysis (EDA)

	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

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

diabetes_df.columns
Output:

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

Information about the dataset

diabetes_df.info()
Output:

```
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

To know more about the dataset

diabetes_df.describe()
Output:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	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	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

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

diabetes_df.describe().T
Output:

	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: tput:

	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:

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.

```
diabetes_df_copy = diabetes_df.copy(deep = True)

diabetes_df_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']]
=
diabetes_df_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']]
.replace(0,np.NaN)


# Showing the Count of NANs

print(diabetes_df_copy.isnull().sum())
```

Output:

```
Pregnancies               0

Glucose                   5

BloodPressure            35

SkinThickness           227

Insulin                 374

BMI                      11

DiabetesPedigreeFunction  0
```

Age 0

Outcome 0

dtype: int64

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.

Data Visualization

Plotting the data distribution plots before removing null values

output:

	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:

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.

```
diabetes_df_copy = diabetes_df.copy(deep = True)

diabetes_df_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']]
=
diabetes_df_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']]
.replace(0,np.NaN)
```

 # Showing the Count of NANs

print(diabetes_df_copy.isnull().sum())

Output:

Pregnancies 0

Glucose 5

BloodPressure 35

SkinThickness 227

Insulin 374

BMI 11

DiabetesPedigreeFunction 0

Age 0

Outcome 0

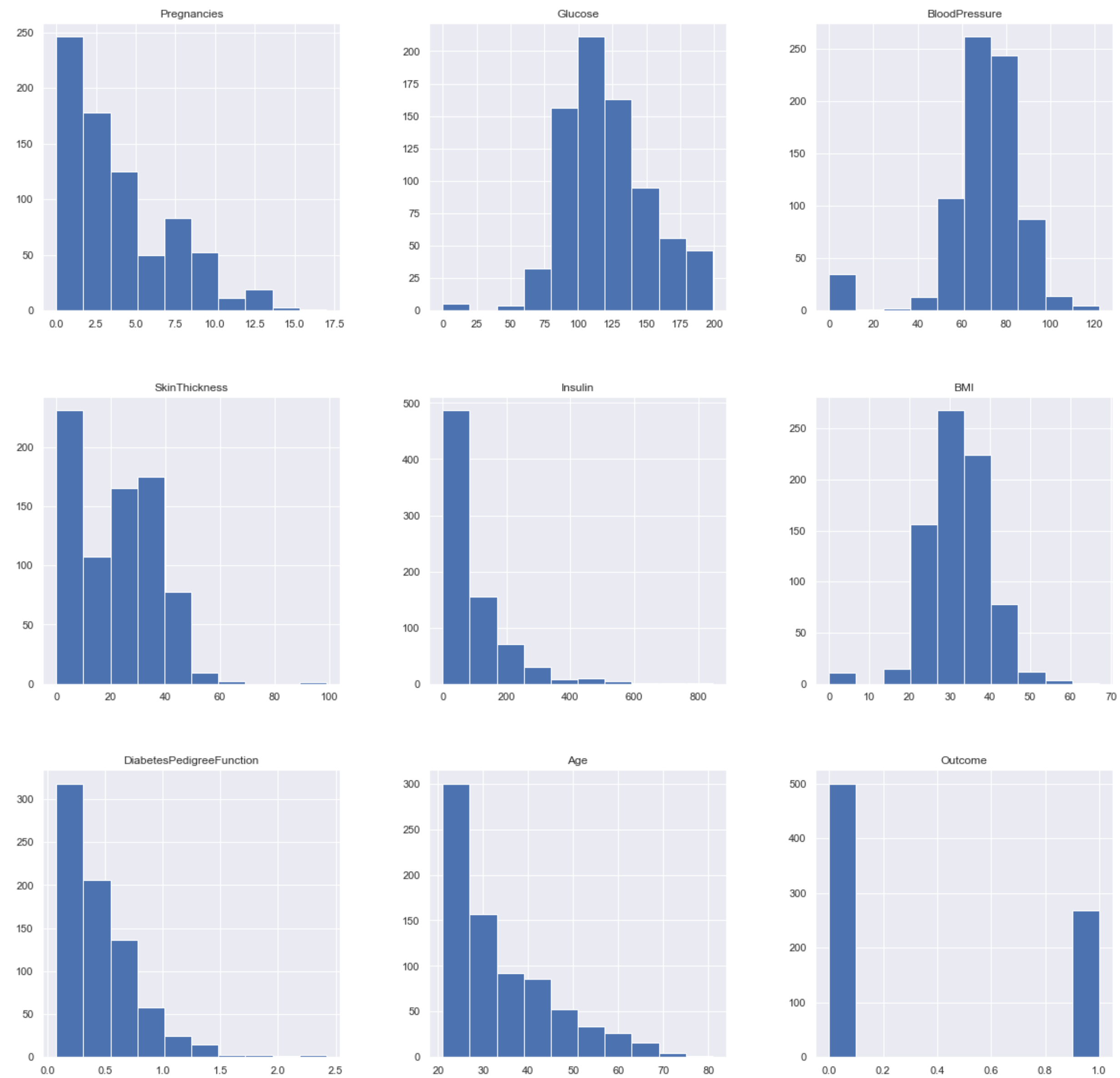
dtype: int64

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.

Data Visualization

Plotting the data distribution plots before removing null values

Output:


Pregnancies
250 200 150 100 50 0
0.0 2.5 5.0 7.5 10.0 12.5 15.0 17.5
Glucose
200 175 150 125 100 75 50 25 0
0 25 50 75 100 125 150 175 200
BloodPressure
250 200 150 100 50 0
0 20 40 60 80 100 120
SkinThickness
200 150 100 50 0
0 20 40 60 80 100
Insulin
500 400 300 200 100 0
0 200 400 600 800
BMI
250 200 150 100 50 0
0 10 20 30 40 50 60 70
DiabetesPedigreeFunction
300 250 200 150 100 50 0
0.0 0.5 1.0 1.5 2.0 2.5
Age
300 250 200 150 100 50 0
20 30 40 50 60 70 80
Outcome
500 400 300 200 100 0
0.0 0.2 0.4 0.6 0.8 1.0

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['BloodPressure'].mean(), inplace = True)

diabetes_df_copy['SkinThickness'].fillna(diabetes_df_copy['SkinThickness']. median(), inplace = True)

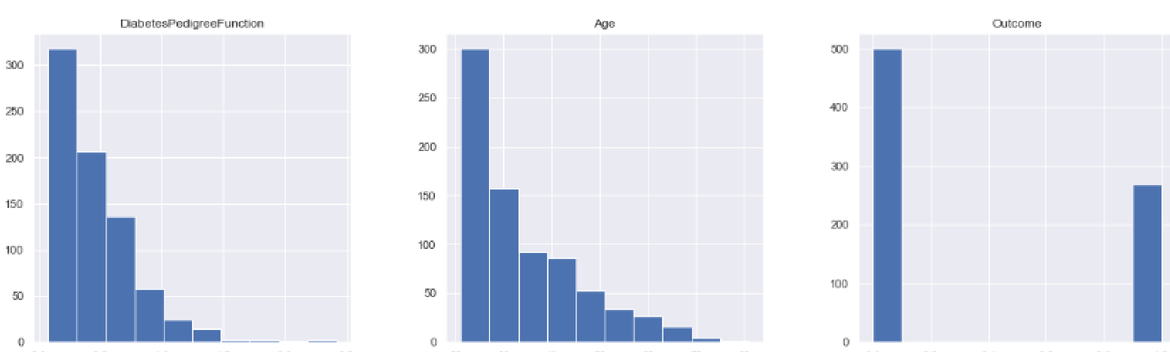
diabetes_df_copy['Insulin'].fillna(diabetes_df_copy['Insulin'].median(), inplace = True)

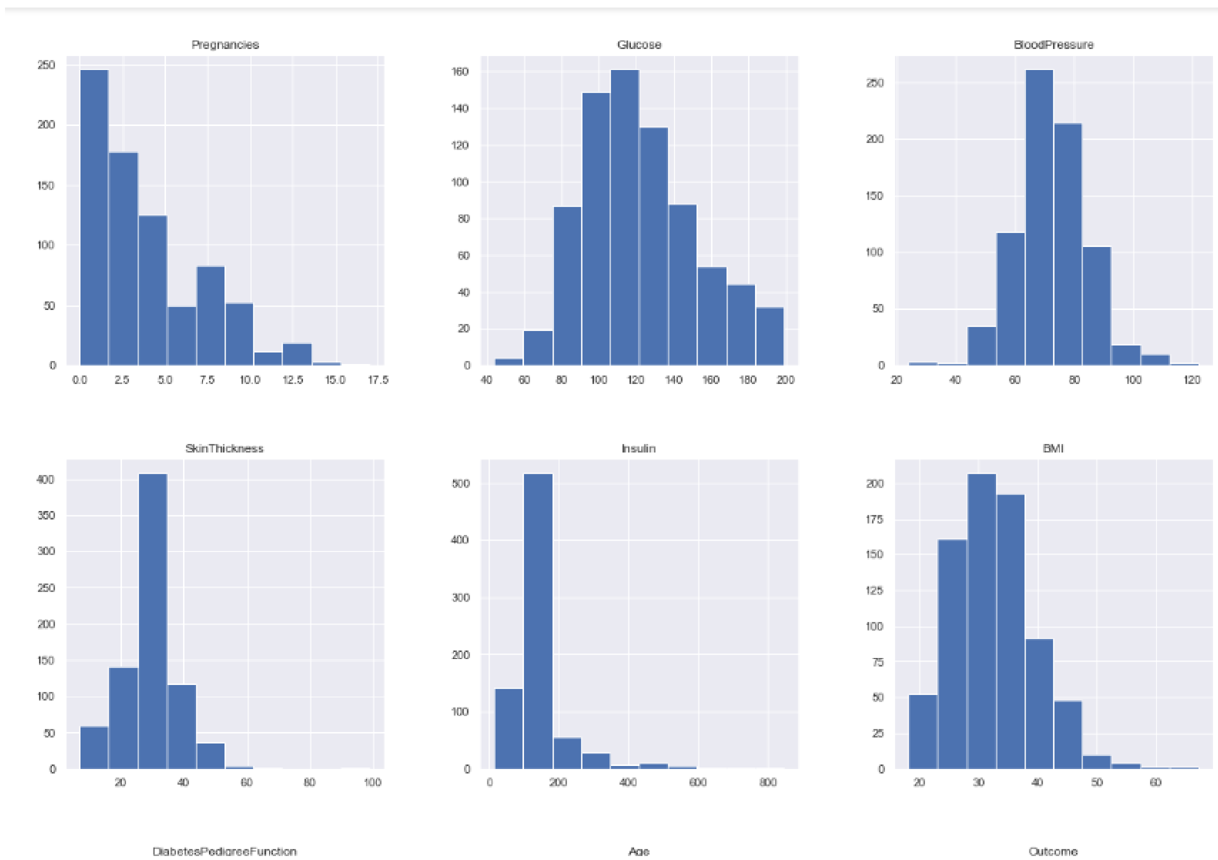
diabetes_df_copy['BMI'].fillna(diabetes_df_copy['BMI'].median(), inplace = True)

Plotting the distributions after removing the NAN values.

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

Output:



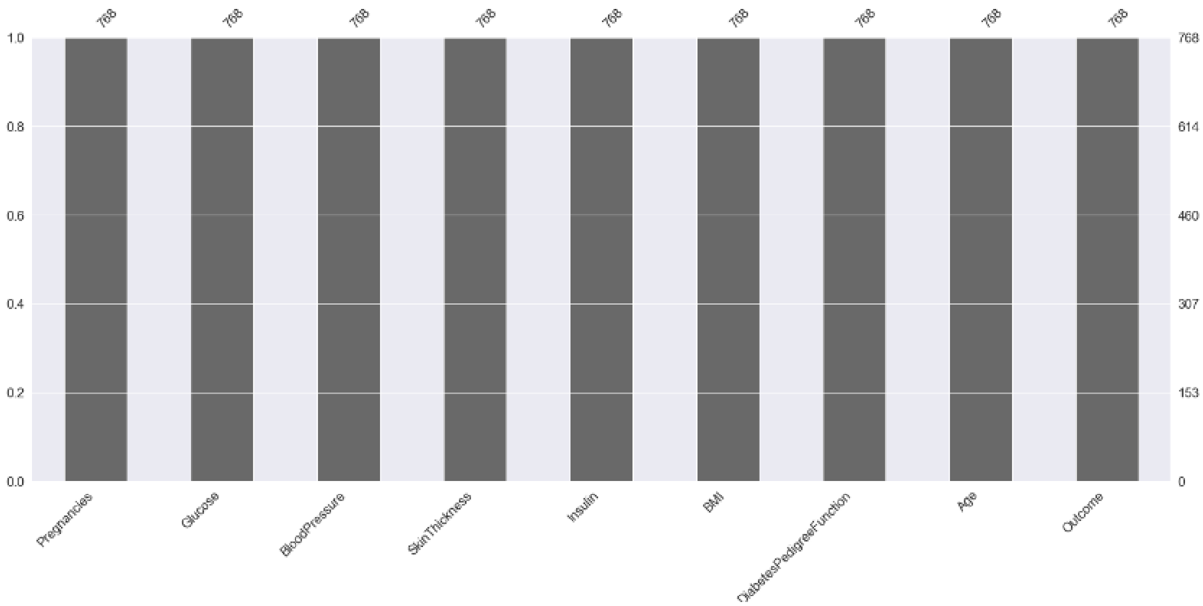


Inference: Here we are again using the hist plot to see the distribution of the dataset but this time we are using this visualization to see the changes that we can see after those null values are removed from the dataset and we can clearly see the difference for example – In age column after removal 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:



768
768
768
768
768
768
768
768
768
1.0
0.8
0.6
0.4
0.2
0.0
768
614
460
307
153
0
Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age
Outcome

Inference: Now in the above graph also we can clearly see that there are

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 +
print(diabetes_df.Outcome.value_counts())
p=diabetes_df.Outcome.value_counts().plot(kind="bar")
```

Output:

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

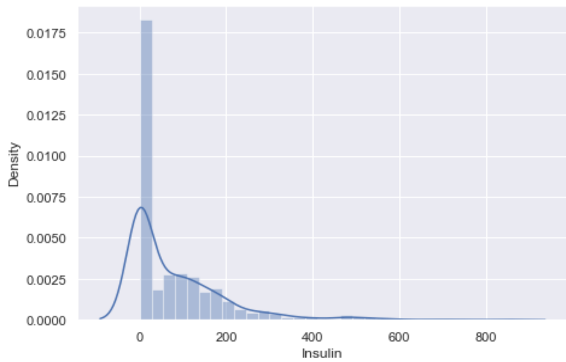
Inference: Here from the above visualization it is clearly visible that our who are **diabetic is half of the patients who are non-diabetic.**

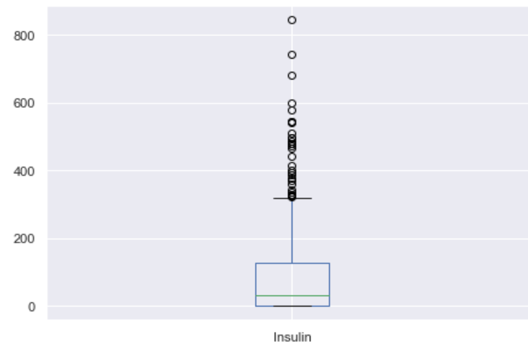
```
plt.subplot(121), sns.distplot(diabetes_df['Insulin'])
plt.subplot(122), diabetes_df['Insulin'].plot.box(figsize=(16,5))
plt.show()
```

Output:

Inference: That's how **Distplot** can be helpful where one will able to se **one can see the outliers in that column** and other information too whic

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.