SURYA GROUP OF INSTITUTIONS

VIKRAVANDI-605652

NAAN MUDHALVAN PROJECT

AI-BASED DIABETES PREDICTION SYSTEM

PHASE-3

DEVELOPMENT PART-1

PRESENTED BY

NAME:P.PARTHIPAN

REG.NO:422221106305

DEPT:ECE 3RD YEAR

INTRODUCTION:

We have implemented various methods or approaches to use our data systematically and in synchronized for the purpose of the development

Of our model. Moreover the test plan is according to our model and can be helpful if we wants to make further improvements and developments to our model.

Import Required Libraries:

```
# Ignore warning messages to prevent them from being displayed during code execution
import warnings
warnings.filterwarnings('ignore')
```

```
import numpy as np    # Importing the NumPy library for linear algebra operations
import pandas as pd   # Importing the Pandas library for data processing and CSV file handli
ng
```

```
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
import seaborn as sns                  # Importing the Seaborn library for statistical data visualiz
ation
import matplotlib.pyplot as plt        # Importing the Matplotlib library for creating plots and
visualizations
import plotly.express as px            # Importing the Plotly Express library for interactive visu
alizations
```

Load and Prepare Data:

```
df=pd.read_csv('/kaggle/input/diabetes-data-set/diabetes.csv')
```

UnderStanding the Variables:

```
df.head(10)
```

	pregnancies	glucose	BP	Skin thickness	insulin	BMI	Diabetes pedigree function	age	outcome
1	10	148	96	35	0	0	33.6	26	1
2	3	85	0	29	0	00	26.6	30	0
3	9	183	40	0	35	0	23.3	33	1
4	5	89	65	23	23	94	28.1	21	0
5	0	137	64	35	0	168	43.1	32	1

6	7	116	66	0	29	0	25.6	31	0
7	8	78	73	32	35	88	31.0	50	1

```
df.tail(10)
```

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
758	1	106	76	0	0	37.5	0.197	26
759	6	190	92	0	0	35.5	0.278	66
760	2	88	58	26	16	28.4	0.766	22
761	9	170	74	31	0	44.0	0.403	43
762	9	89	62	0	0	22.5	0.142	33
763	10	101	76	48	180	32.9	0.171	63
764	2	122	70	27	0	36.8	0.340	27
765	5	121	72	23	112	26.2	0.245	30
766	1	126	60	0	0	30.1	0.349	47

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
767	1	93	70	31	0	30.4	0.315	23

```
df.sample(5)
```

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
355	9	165	88	0	0	30.4	0.302	49
187	1	128	98	41	58	32.0	1.321	33
235	4	171	72	0	0	43.6	0.479	26

```
df.describe()
```

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
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
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

```
<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
```

```
df.size
```

6912

df.shape

(768, 9)

Data Cleaning:

df.shape

(768, 9)

```
df=df.drop_duplicates()
df.shape
```

(768, 9)

Check null Values

df.isnull().sum()

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

There is no Missing Values present in the Data

df.columns

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

Check the number of Zero Values in Dataset:

print("No. of Zero Values in Glucose ", df[df['Glucose']==0].shape[0])

No. of Zero Values in Glucose 5

print("No. of Zero Values in Glucose ", df[df['Glucose']==0].shape[0])

No. of Zero Values in Glucose 5

print("No. of Zero Values in Blood Pressure ", df[df['BloodPressure']==0].shape[0])

No. of Zero Values in Blood Pressure 35

print("No. of Zero Values in SkinThickness ", df[df['SkinThickness']==0].shape[0])

No. of Zero Values in SkinThickness 227

print("No. of Zero Values in Insulin ", df[df['Insulin']==0].shape[0])

No. of Zero Values in Insulin 374

```
print("No. of Zero Values in BMI ", df[df['BMI']==0].shape[0])
```

```
No. of Zero Values in BMI  11
```

```
df['Glucose']=df['Glucose'].replace(0, df['Glucose'].mean())
```

Replace zeroes with mean of that Columns:

print('No of zero Values in Glucose ', df[df['Glucose']==0].shape[0])

No of zero Values in Glucose 0
linkcode

```
df['BloodPressure']=df['BloodPressure'].replace(0, df['BloodPressure'].mean())
df['SkinThickness']=df['SkinThickness'].replace(0, df['SkinThickness'].mean())
df['Insulin']=df['Insulin'].replace(0, df['Insulin'].mean())
df['BMI']=df['BMI'].replace(0, df['BMI'].mean())
```

Validate the Zero Values:

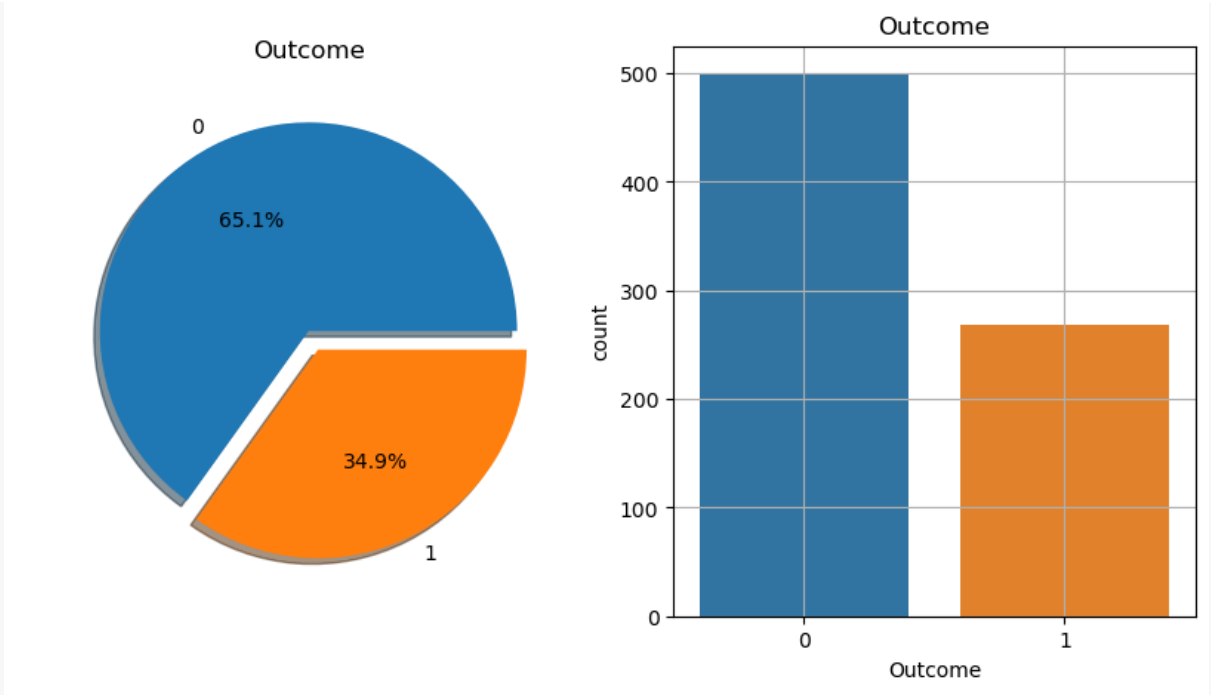
df.describe()

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
mean	3.845052	121.681605	72.254807	26.606479	118.660163	32.450805	0.471876	33.240885
std	3.369578	30.436016	12.115932	9.631241	93.080358	6.875374	0.331329	11.760232
min	0.000000	44.000000	24.000000	7.000000	14.000000	18.200000	0.078000	21.000000
25%	1.000000	99.750000	64.000000	20.536458	79.799479	27.500000	0.243750	24.000000
50%	3.000000	117.000000	72.000000	23.000000	79.799479	32.000000	0.372500	29.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000

Data Visualization:

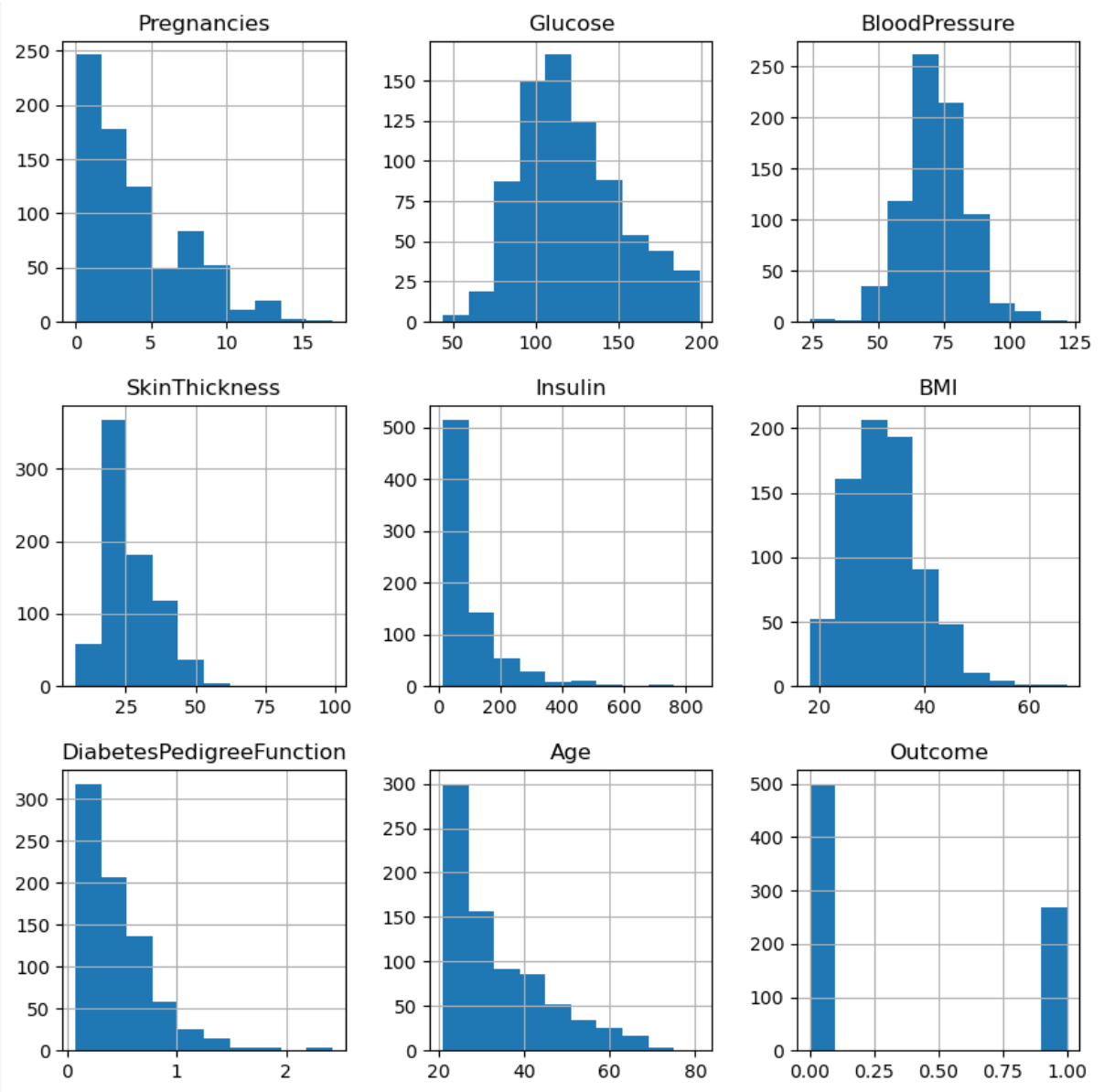
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming 'df' is your DataFrame containing the dataset
# If you haven't imported your dataset yet, import it here

# Create subplots
f, ax = plt.subplots(1, 2, figsize=(10, 5))

# Pie chart for Outcome distribution
df['Outcome'].value_counts().plot.pie(explode=[0, 0.1], autopct='%1.1f%%', ax=ax[0], shadow=True)
ax[0].set_title('Outcome')
ax[0].set_ylabel(' ')

# Count plot for Outcome distribution
sns.countplot(x='Outcome', data=df, ax=ax[1])  # Use 'x' instead of 'Outcome'
ax[1].set_title('Outcome')

# Count plot for Outcome distribution
sns.countplot(x='Outcome', data=df, ax=ax[1])  # Use 'x' instead of 'Outcome'
ax[1].set_title('Outcome')

# Display class distribution
N, P = df['Outcome'].value_counts()
print('Negative (0):', N)
print('Positive (1):', P)
# Adding grid and showing plots
plt.grid()
plt.show()
Negative (0): 500
Positive (1): 268
```



- 1 Represent --> Diabetes Positive
- 0 Represent --> Daibetes Negative

Histograms:

```
df.hist(bins=10, figsize=(10, 10))
plt.show()
```



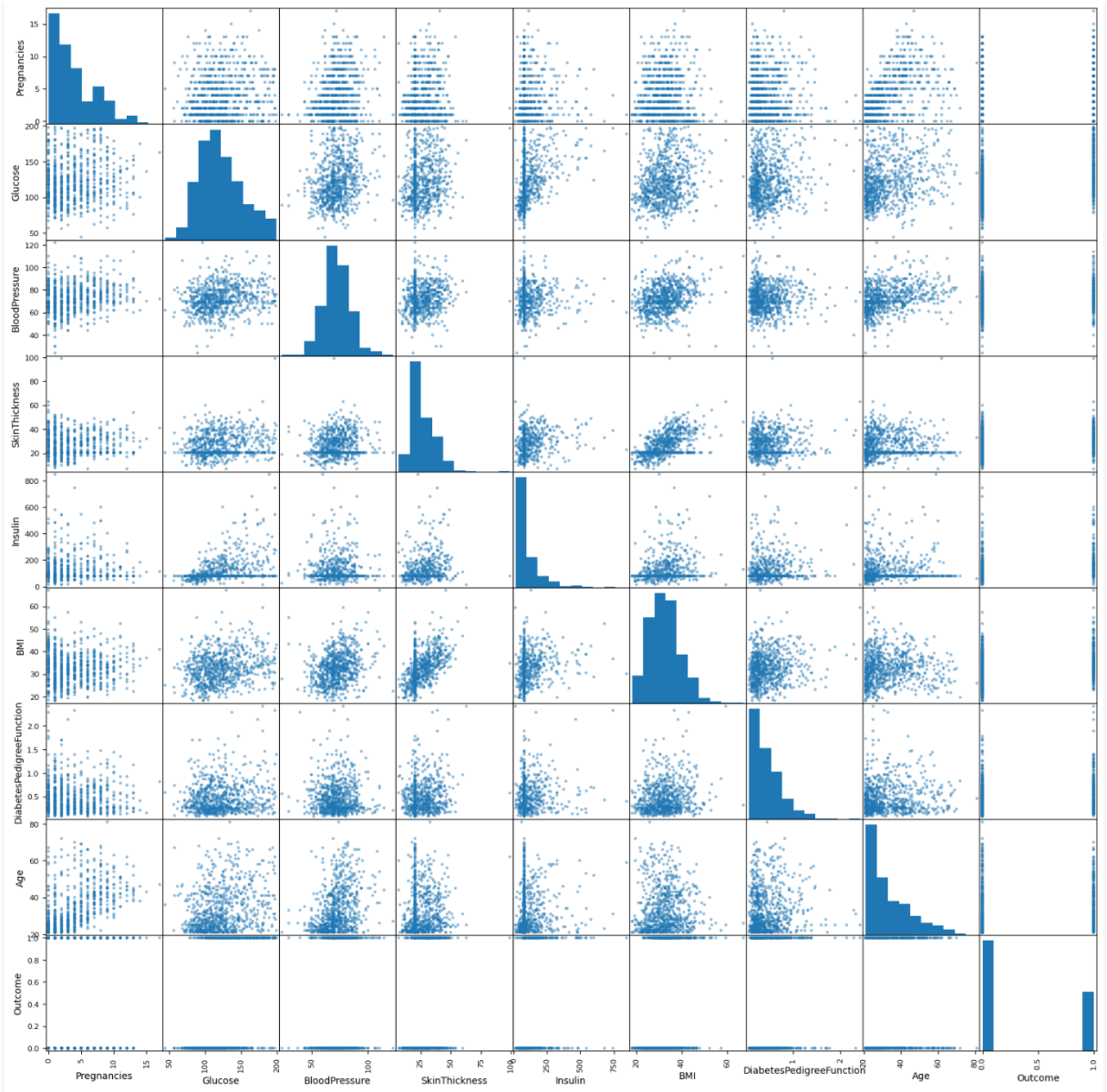
Scatter Plot:

```
from pandas.plotting import scatter_matrix
scatter_matrix(df, figsize =(20, 20))
```

```
array([[<Axes: xlabel='Pregnancies', ylabel='Pregnancies'>,
        <Axes: xlabel='Glucose', ylabel='Pregnancies'>,
        <Axes: xlabel='BloodPressure', ylabel='Pregnancies'>,
        <Axes: xlabel='SkinThickness', ylabel='Pregnancies'>,
        <Axes: xlabel='Insulin', ylabel='Pregnancies'>,
        <Axes: xlabel='BMI', ylabel='Pregnancies'>,
        <Axes: xlabel='DiabetesPedigreeFunction', ylabel='Pregnancies'>,
        <Axes: xlabel='Age', ylabel='Pregnancies'>,
        <Axes: xlabel='Outcome', ylabel='Pregnancies'>],
       [<Axes: xlabel='Pregnancies', ylabel='Glucose'>,
        <Axes: xlabel='Glucose', ylabel='Glucose'>,
        <Axes: xlabel='BloodPressure', ylabel='Glucose'>,
        <Axes: xlabel='SkinThickness', ylabel='Glucose'>,
```

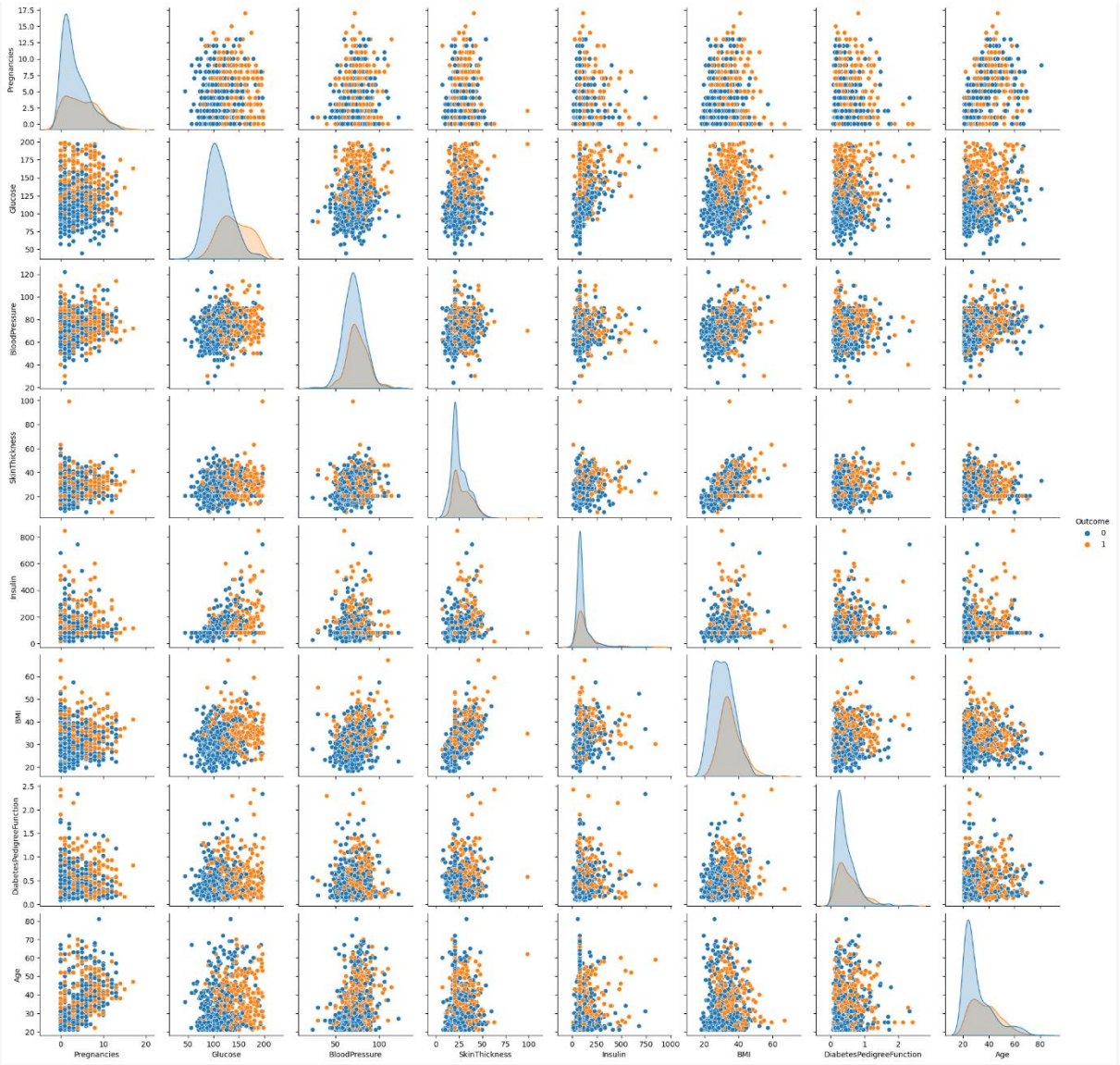
<Axes: xlabel='Insulin', ylabel='Glucose'>,
<Axes: xlabel='BMI', ylabel='Glucose'>,
<Axes: xlabel='DiabetesPedigreeFunction', ylabel='Glucose'>,
<Axes: xlabel='Age', ylabel='Glucose'>,
<Axes: xlabel='Outcome', ylabel='Glucose'>],
[<Axes: xlabel='Pregnancies', ylabel='BloodPressure'>,
<Axes: xlabel='Glucose', ylabel='BloodPressure'>,
<Axes: xlabel='BloodPressure', ylabel='BloodPressure'>,
<Axes: xlabel='SkinThickness', ylabel='BloodPressure'>,
<Axes: xlabel='Insulin', ylabel='BloodPressure'>,
<Axes: xlabel='BMI', ylabel='BloodPressure'>,
<Axes: xlabel='DiabetesPedigreeFunction', ylabel='BloodPressure'>,
<Axes: xlabel='Age', ylabel='BloodPressure'>,
<Axes: xlabel='Outcome', ylabel='BloodPressure'>],
[<Axes: xlabel='Pregnancies', ylabel='SkinThickness'>,
<Axes: xlabel='Glucose', ylabel='SkinThickness'>,
<Axes: xlabel='BloodPressure', ylabel='SkinThickness'>,
<Axes: xlabel='SkinThickness', ylabel='SkinThickness'>,
<Axes: xlabel='Insulin', ylabel='SkinThickness'>,
<Axes: xlabel='BMI', ylabel='SkinThickness'>,
<Axes: xlabel='DiabetesPedigreeFunction', ylabel='SkinThickness'>,
<Axes: xlabel='Age', ylabel='SkinThickness'>,
<Axes: xlabel='Outcome', ylabel='SkinThickness'>],
[<Axes: xlabel='Pregnancies', ylabel='Insulin'>,
<Axes: xlabel='Glucose', ylabel='Insulin'>,
<Axes: xlabel='BloodPressure', ylabel='Insulin'>,
<Axes: xlabel='SkinThickness', ylabel='Insulin'>,
<Axes: xlabel='Insulin', ylabel='Insulin'>,
<Axes: xlabel='BMI', ylabel='Insulin'>,
<Axes: xlabel='DiabetesPedigreeFunction', ylabel='Insulin'>,
<Axes: xlabel='Age', ylabel='Insulin'>,
<Axes: xlabel='Outcome', ylabel='Insulin'>],
[<Axes: xlabel='Pregnancies', ylabel='BMI'>,
<Axes: xlabel='Glucose', ylabel='BMI'>,
<Axes: xlabel='BloodPressure', ylabel='BMI'>,
<Axes: xlabel='SkinThickness', ylabel='BMI'>,
<Axes: xlabel='Insulin', ylabel='BMI'>,
<Axes: xlabel='BMI', ylabel='BMI'>,
<Axes: xlabel='DiabetesPedigreeFunction', ylabel='BMI'>,
<Axes: xlabel='Age', ylabel='BMI'>,
<Axes: xlabel='Outcome', ylabel='BMI'>],
[<Axes: xlabel='Pregnancies', ylabel='DiabetesPedigreeFunction'>,
<Axes: xlabel='Glucose', ylabel='DiabetesPedigreeFunction'>,
<Axes: xlabel='BloodPressure', ylabel='DiabetesPedigreeFunction'>,
<Axes: xlabel='SkinThickness', ylabel='DiabetesPedigreeFunction'>,
<Axes: xlabel='Insulin', ylabel='DiabetesPedigreeFunction'>,
<Axes: xlabel='BMI', ylabel='DiabetesPedigreeFunction'>,
<Axes: xlabel='DiabetesPedigreeFunction', ylabel='DiabetesPedigreeFunction'>,
<Axes: xlabel='Age', ylabel='DiabetesPedigreeFunction'>,
<Axes: xlabel='Outcome', ylabel='DiabetesPedigreeFunction'>],

```
      [<Axes: xlabel='Pregnancies', ylabel='Age'>,
       <Axes: xlabel='Glucose', ylabel='Age'>,
            <Axes: xlabel='BloodPressure', ylabel='Age'>,
       <Axes: xlabel='SkinThickness', ylabel='Age'>,
       <Axes: xlabel='Insulin', ylabel='Age'>,
       <Axes: xlabel='BMI', ylabel='Age'>,
       <Axes: xlabel='DiabetesPedigreeFunction', ylabel='Age'>,
       <Axes: xlabel='Age', ylabel='Age'>,
<Axes: xlabel='Outcome', ylabel='Age'>],
      [<Axes: xlabel='Pregnancies', ylabel='Outcome'>,
       <Axes: xlabel='Glucose', ylabel='Outcome'>,
       <Axes: xlabel='BloodPressure', ylabel='Outcome'>,
       <Axes: xlabel='SkinThickness', ylabel='Outcome'>,
       <Axes: xlabel='Insulin', ylabel='Outcome'>,
       <Axes: xlabel='BMI', ylabel='Outcome'>,
       <Axes: xlabel='DiabetesPedigreeFunction', ylabel='Outcome'>,
       <Axes: xlabel='Age', ylabel='Outcome'>,
       <Axes: xlabel='Outcome', ylabel='Outcome'>]], dtype=object)
```

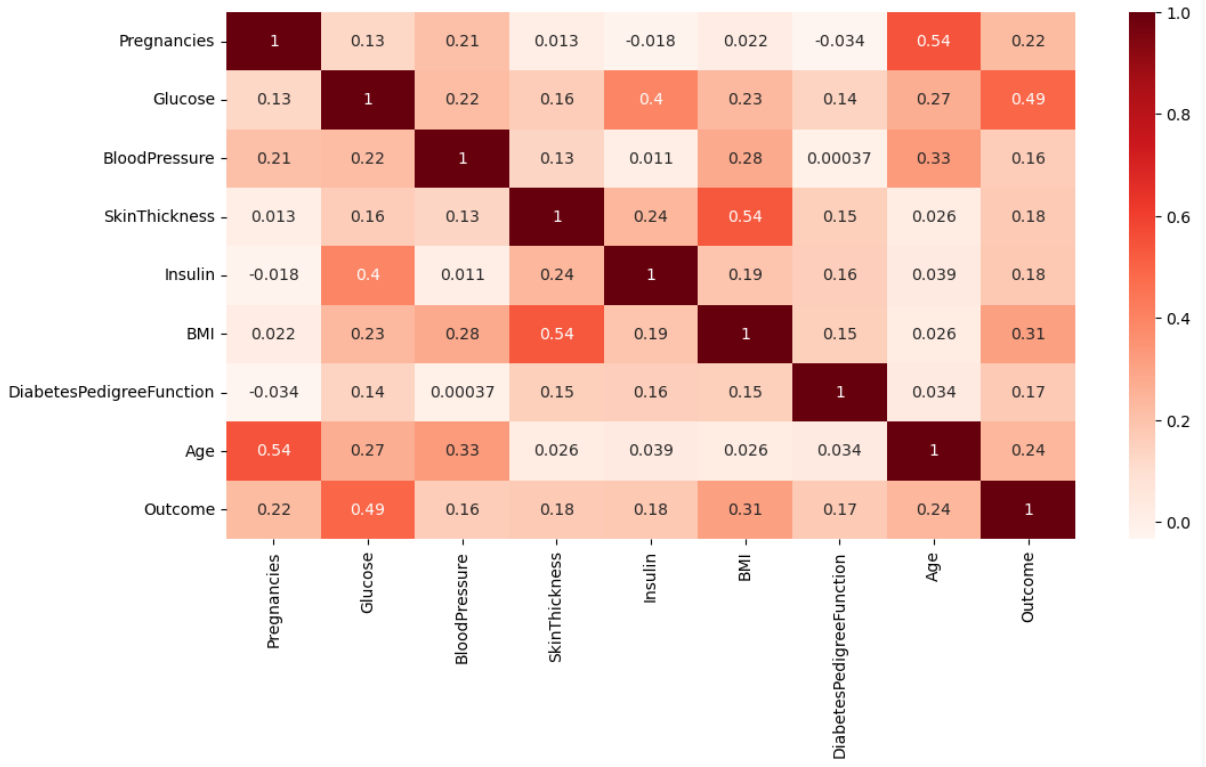


Pair plot:

```
sns.pairplot(data=df, hue='Outcome')
plt.show()
```




```
plt.figure(figsize=(12, 6))
sns.heatmap(df.corr(), annot=True, cmap='Reds')
plt.plot()
# Creating a heatmap of the correlation matrix for the columns in the DataFrame data
```




```
mean = df['Outcome'].mean()
# Calculating the mean value of the 'Outcome' column in the DataFrame data
mean
# Displaying the calculated mean value
```

0.3489583333333333

Split the DataFrame into X and y:

```
target_name='Outcome'

y=df[target_name]

X= df.drop(target_name, axis=1)

X.head()
```

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148.0	72.0	35.000000	79.799479	33.6	0.627
1	1	85.0	66.0	29.000000	79.799479	26.6	0.351

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
2	8	183.0	64.0	20.536458	79.799479	23.3	0.672
3	1	89.0	66.0	23.000000	94.000000	28.1	0.167
4	0	137.0	40.0	35.000000	168.000000	43.1	2.288

```
y.head()
```

```
0    1
1    0
2    1
3    0
4    1
Name: Outcome, dtype: int64
```

Future Scalling:

```
# Standard Scaler:
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X)
SSX = scaler.transform(X)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(SSX, y, test_size=0.2, random_state=7)

X_train.shape, y_train.shape
```

((154, 8), (154,))

Making prediction:

```
X_test.shape
```

(154, 8)

```
lr_pred=lr.predict(X_test)

lr_pred.shape
```

(154,)

linkcode

Decision Tree:

```
dt_pred=dt.predict(X_test)
```

linkcode
dt_pred.shape

 # For Logistic Regression:
from sklearn.metrics import accuracy_score
print("Train Accuracy of Logistic Regression: ", lr.score(X_train, y_train)*100)
print("Accuracy (Test) Score of Logistic Regression: ", lr.score(X_test, y_test)*100)
print("Accuracy Score of Logistic Regression: ", accuracy_score(y_test, lr_pred)*100)

Train Accuracy of Logistic Regression: 77.36156351791531
Accuracy (Test) Score of Logistic Regression: 77.27272727272727
Accuracy Score of Logistic Regression: 77.27272727272727

linkcode
 # For Decesion Tree:
print("Train Accuracy of Decesion Tree: ", dt.score(X_train, y_train)*100)
print("Accuracy (Test) Score of Decesion Tree: ", dt.score(X_test, y_test)*100)
print("Accuracy Score of Decesion Tree: ", accuracy_score(y_test, dt_pred)*100)

Train Accuracy of Decesion Tree: 100.0
Accuracy (Test) Score of Decesion Tree: 80.51948051948052
Accuracy Score of Decesion Tree: 80.51948051948052

from sklearn.metrics import precision_score
print("Precision Score is: ", precision_score(y_test, lr_pred)*100)
print("Micro Average Precision Score is: ", precision_score(y_test, lr_pred, average='micro')
*100)
print("Macro Average Precision Score is: ", precision_score(y_test, lr_pred, average='macro')
*100)
print("Weighted Average Precision Score is: ", precision_score(y_test, lr_pred, average='wei
ghted')*100)
print("precision Score on Non Weighted score is: ", precision_score(y_test, lr_pred, average=
None)*100)
Precision Score is: 75.0
Micro Average Precision Score is: 77.27272727272727
Macro Average Precision Score is: 76.5909090909091
Weighted Average Precision Score is: 77.00413223140497
precision Score on Non Weighted score is: [78.18181818 75.

print('Classification Report of Logistic Regression: \n', classification_report(y_test, lr_pred, d
igits=4))

Classification Report of Logistic Regression:
              precision   recall f1-score   support

           0   0.7818   0.8866   0.8309   97
           1   0.7500   0.5789   0.6535   57

    accuracy   0.7727   154

```
   macro avg     0.7659    0.7328    0.7422       154
weighted avg     0.7700    0.7727    0.7652       154
```

```
True Positive Rate(TPR)
```

Recall = True Positive/True Positive + False Negative
Recall = TP/TP+FN

In [65]:

```
recall_score = TP/ float(TP+FN)*100
print('recall_score', recall_score)
```

```
recall_score 57.89473684210527
```

In [66]:

```
TP, FN
```

Out[66]:

```
(33, 24)
```

```
from sklearn.metrics import recall_score
print('Recall or Sensitivity_Score: ', recall_score(y_test, lr_pred)*100)
```

Recall or Sensitivity_Score: 57.89473684210527

In [69]:

linkcode

```
print("recall Score is: ", recall_score(y_test, lr_pred)*100)
print("Micro Average recall Score is: ", recall_score(y_test, lr_pred, average='micro')*100)
print("Macro Average recall Score is: ", recall_score(y_test, lr_pred, average='macro')*100)
print("Weighted Average recall Score is: ", recall_score(y_test, lr_pred, average='weighted')*
100)
```

recall Score is: 57.89473684210527
Micro Average recall Score is: 77.27272727272727
Macro Average recall Score is: 73.27726532826912
Weighted Average recall Score is: 77.27272727272727
recall Score on Non Weighted score is: [88.65979381 57.89473684]

```
print('Classification Report of Logistic Regression: \n', classification_report(y_test, lr_pred, d
igits=4))
```

Classification Report of Logistic Regression:

	precision	recall	f1-score	support
0	0.7818	0.8866	0.8309	97
1	0.7500	0.5789	0.6535	57
accuracy			0.7727	154
macro avg	0.7659	0.7328	0.7422	154
weighted avg	0.7700	0.7727	0.7652	154

ROC Curve& ROC AUC

 # Area under Curve:
auc= roc_auc_score(y_test, lr_pred)

print("ROC AUC SCORE of logistic Regression is ", auc)

ROC AUC SCORE of logistic Regression is 0.7327726532826913

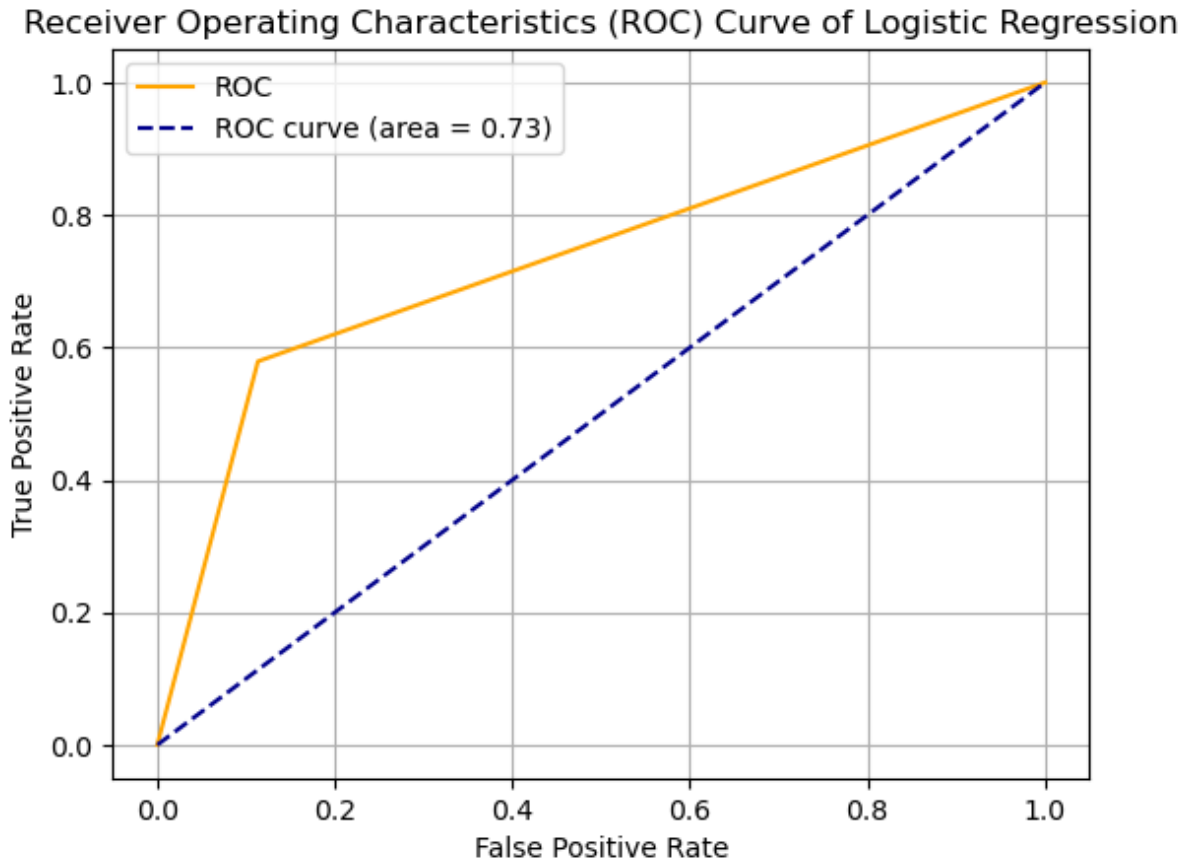
In [79]:

linkcode

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
```

from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

```
fpr, tpr, thresholds = roc_curve(y_test, lr_pred)
plt.plot(fpr, tpr, color='orange', label="ROC")
plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--', label='ROC curve (area = %0.2f)' % auc
(fpr, tpr))
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver Operating Characteristics (ROC) Curve of Logistic Regression")
plt.legend()
plt.grid()
plt.show()
```



CONCLUSION:

We have pre-processed our data and made it useful and made it useful to the further implemen t-tation .various missing values are replaced,many columns are deleted and convereted into Numerical values in order to have positive impact on model.