AI BASED DIABETES PREDICTION SYSTEM USING MACHINE LEARNING TECHNIQUES

PHASE 3 (DEVELOPMENT PART -1)

TEAM MEMBERS :

- **GOPINATH N** (*TEAM LEADER*)
- **HARSHA VARTHAN M**
- **ARUN KUMAR M**
- **ARUL SELVAN V**
- **HARI HARAN R**

INTRODUCTION:

Data preprocessing is a data mining technique that involves transforming raw data into an understandable format. Real-world data is often incomplete, inconsistent, and/or lacking in certain behaviors or trends, and is likely to contain many errors. Data preprocessing is a proven method of resolving such issues. Data preprocessing prepares raw data for further processing.

WHY PREPROCESSING?

Real-world data are generally:

- **Incomplete**: Lacking attribute values, lacking certain attributes of interest, or containing only aggregate data
- **Noisy**: Containing errors or outliers
- **Inconsistent**: Containing discrepancies in codes or names

TASKS IN DATA PREPROCESSING:

- **Data cleaning**: Fill in missing values, smooth noisy data, identify or remove outliers, and resolve inconsistencies.
- **Data integration**: Using multiple databases, data cubes, or files.

- **Data transformation**: Normalization and aggregation.
- **Data reduction**: Reducing the volume but producing the same or similar analytical results.
- **Data discretization**: Part of data reduction, replacing numerical attributes with nominal ones.

WHAT IS EXPLORATORY DATA ANALYSIS?

In statistics, exploratory data analysis (EDA) is an approach to analyzing data sets to summarize their main characteristics, often with visual methods. A statistical model can be used or not, but primarily EDA is for seeing what the data can tell us beyond the formal modeling or hypothesis testing task.

DATA CLEANING :

Data cleaning is the process of cleaning / standardising the data to make it ready for analysis. Most of times, there will be discrepancies in the captured data such as incorrect data formats, missing data, errors while capturing the data. This is an important step in any given data science project because the accuracy of the results depends heavily on the data we use.

DATA INTEGERATION:

Data integration is the process of combining data from multiple sources into a unified view. This can be done for a variety of reasons, such as:

- To improve data quality and consistency
- To gain insights from multiple data sources
- To avoid data duplication
- To improve data accessibility and governance

DATA TRAFORMATION:

Data transformation is the process of converting raw data into a format that is suitable for analysis. This may involve cleaning the data, converting data types, and merging data from multiple sources. Normalization and aggregation are two common data transformation techniques.

Normalization is the process of transforming the values of a dataset into a common scale. This is important to ensure that all of the features in the dataset are treated equally by machine learning algorithms.

PROGRAMS

DATA CLEANING:

```
#importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

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

EDA

```
df.head()
```

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	
0	6	148	72	35	0	33.6	0.627	50	1

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	
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

```
df.columns
```

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

```
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  
```

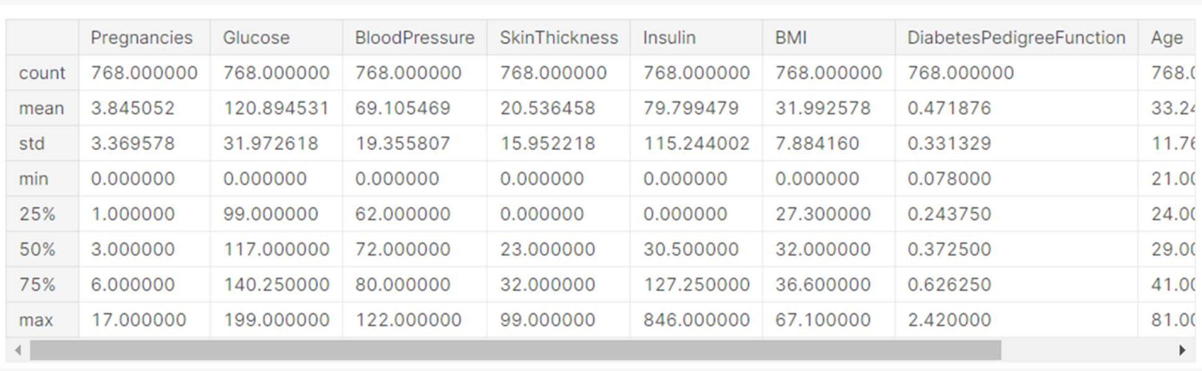
```
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

```python
df.shape
```

```
(768, 9)
```

```
In [1]:
```

```python
linkcode
df.describe()
```



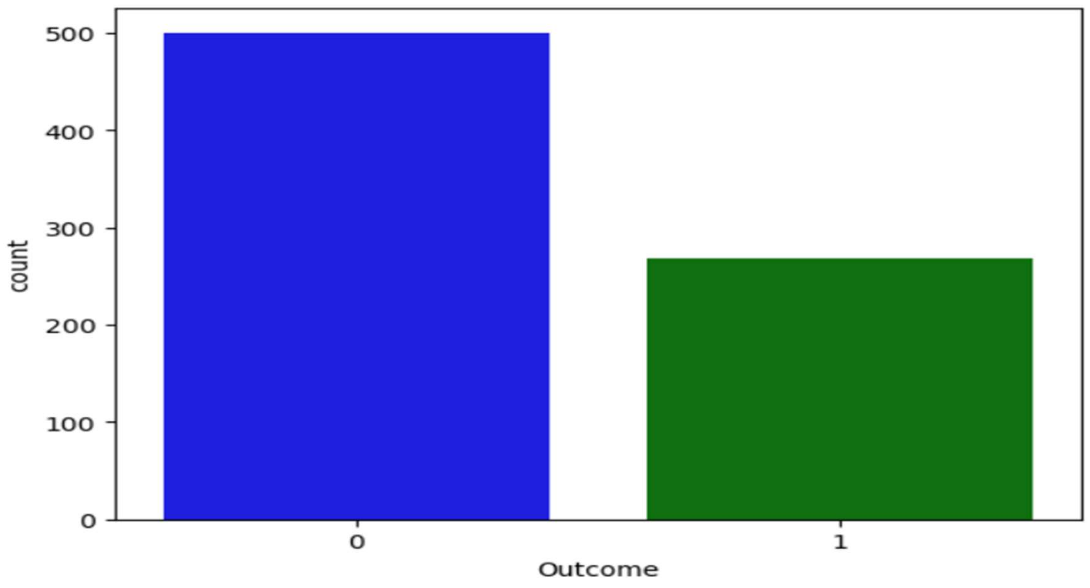
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.2
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.7
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.0
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.0
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.0
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.0
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.0

```python
#checking null values
df.isnull().sum()
```

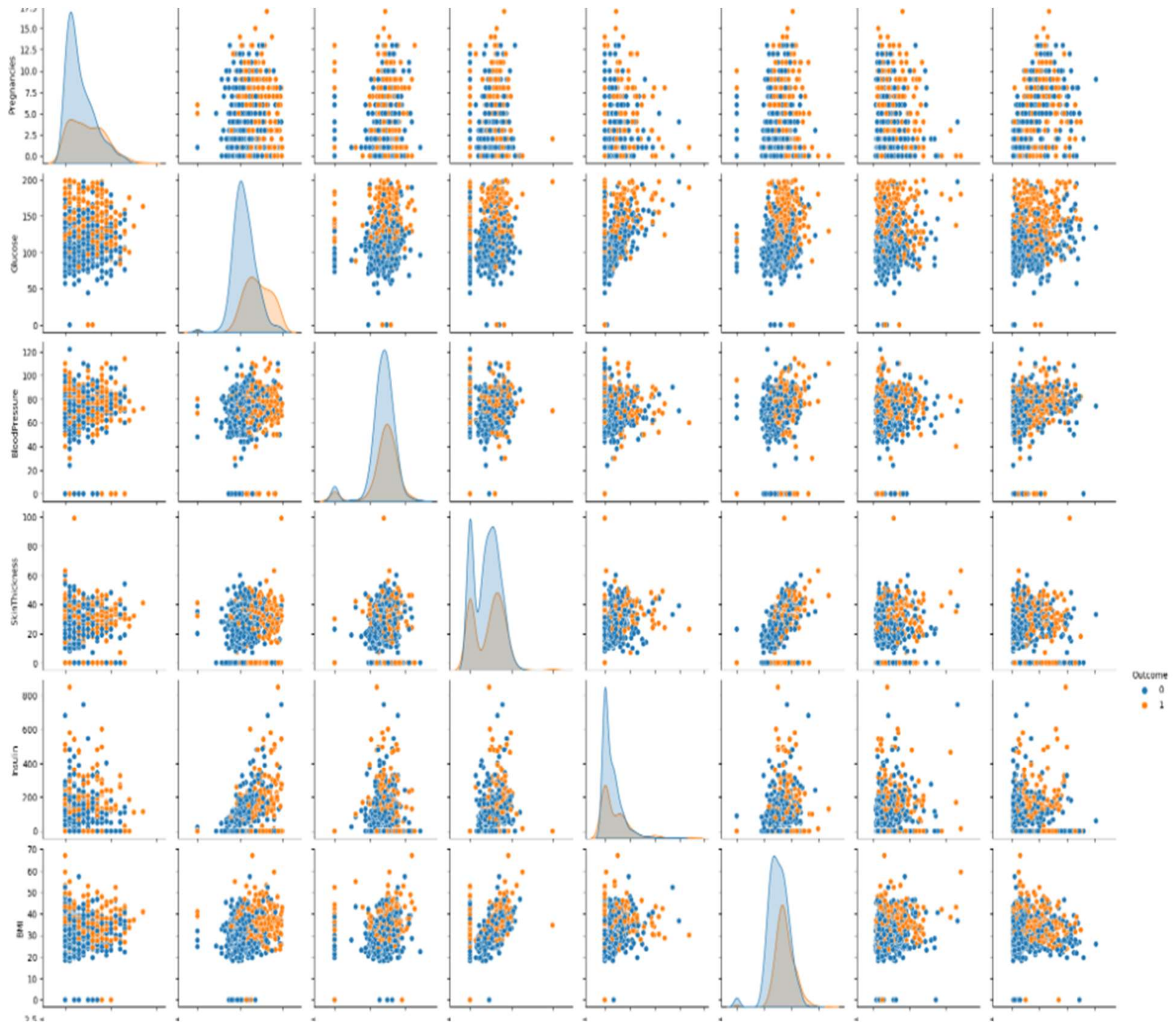
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

```python
#countplot
sns.countplot(x='Outcome',data=df,palette=['b','g'])
```

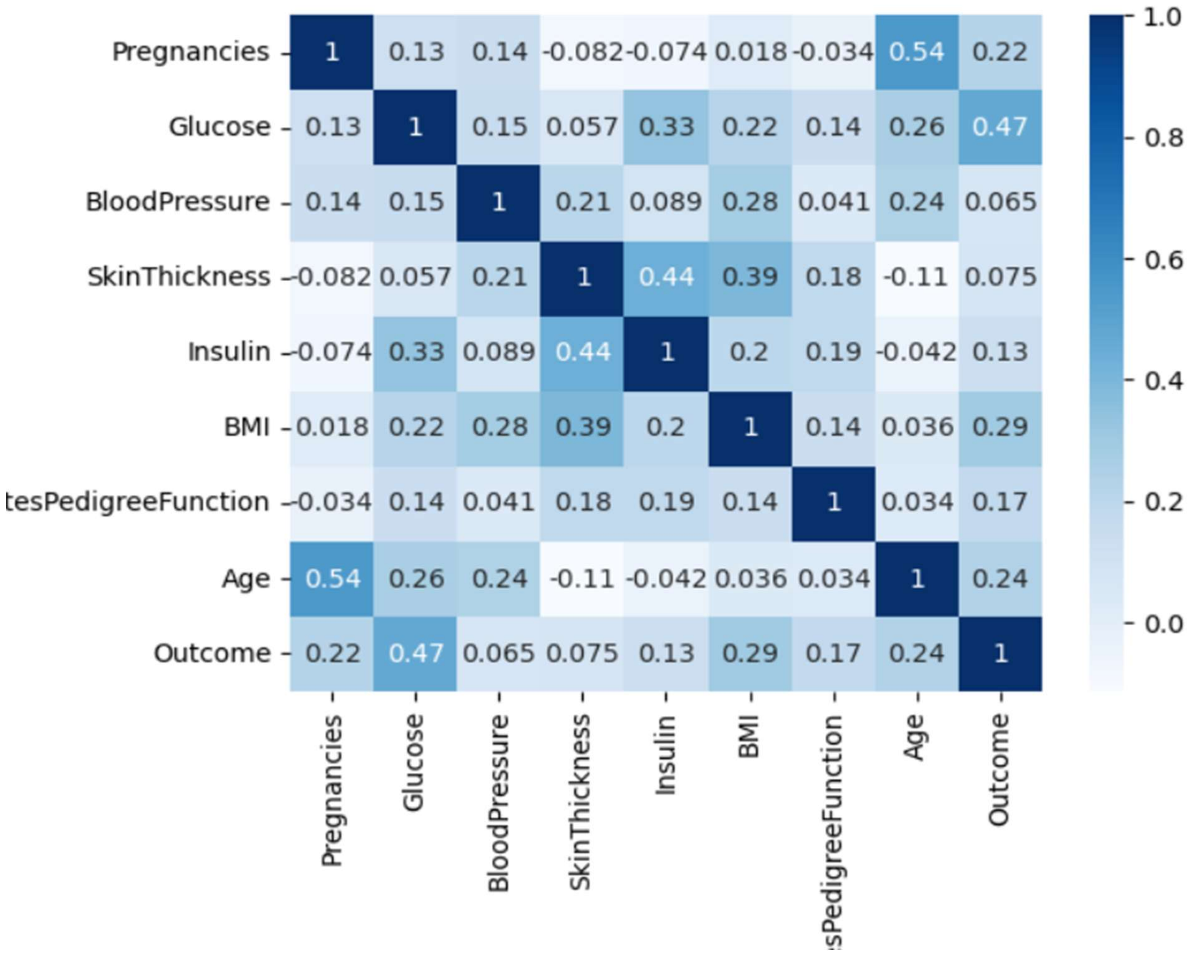
```
<AxesSubplot:xlabel='Outcome', ylabel='count'>
```



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



```
#correlation heatmap
sns.heatmap(df.corr(),annot=True,cmap='Blues')
plt.show()
```



```
#replacing zero value with NaN
df_new=df
df_new[["Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI"]] = df_new[["Gl
ucose", "BloodPressure", "SkinThickness", "Insulin", "BMI"]].replace(0, np.NaN)
```

In [3]:

```
#count of Nan
df_new.isnull().sum()
```

Out[13]:

```
Pregnancies                   0
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
Outcome                       0
```

```
dtype: int64
#replacing NaN with mean values
df_new["Glucose"].fillna(df_new["Glucose"].mean(), inplace = True)
df_new["BloodPressure"].fillna(df_new["BloodPressure"].mean(), inplace = True)
df_new["SkinThickness"].fillna(df_new["SkinThickness"].mean(), inplace = True)
df_new["Insulin"].fillna(df_new["Insulin"].mean(), inplace = True)
df_new["BMI"].fillna(df_new["BMI"].mean(), inplace = True)
```

```
In [4]:
#checking null values
df_new.isnull().sum()
```

```
Out[15]:
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
y=df_new['Outcome']
X=df_new.drop('Outcome',axis=1)
```

```
In [17]:
linkcode
#spliting X and y
from sklearn.model_selection import train_test_split

X_train,X_test,Y_train,Y_test=train_test_split(X,y,test_size=0.20,random_state=0,stra
tify=df_new['Outcome'])
```

```
from sklearn.linear_model import LogisticRegression

model=LogisticRegression()
model.fit(X_train,Y_train)
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:818: Con
vergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
LogisticRegression()
In [19]:
#get prediction
y_predict=model.predict(X_test)
y_predict
```

```
Out[19]:
```

```
array([0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0,
       1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0])
```

In [20]:

```
linkcode
#confusion matrix
from sklearn.metrics import confusion_matrix

cm=confusion_matrix(Y_test,y_predict)
cm
```

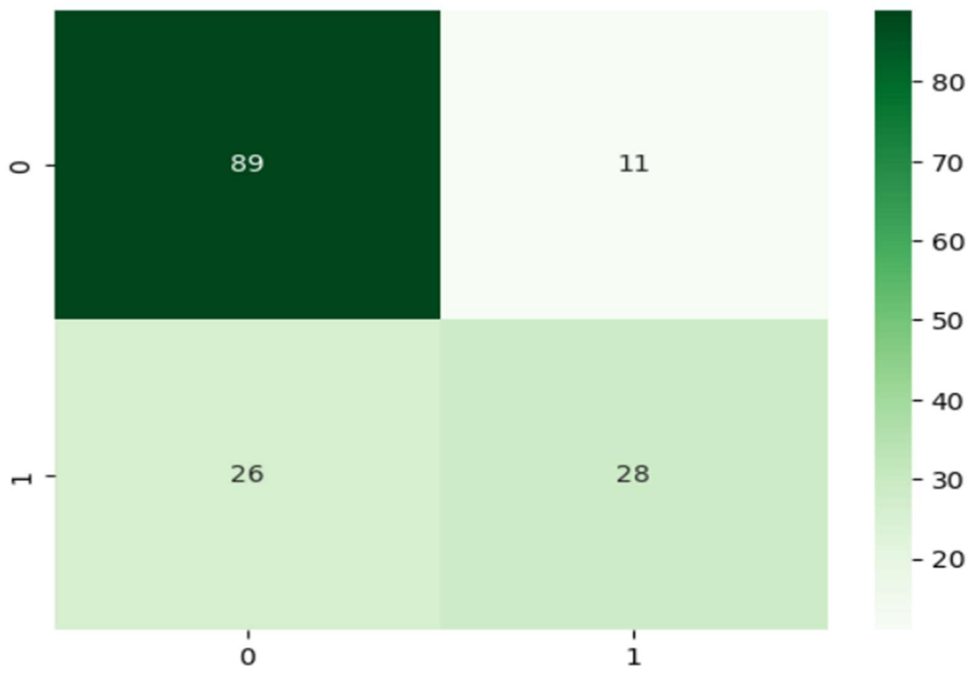
Out[20]:

```
array([[89, 11],
       [26, 28]])
```

In [21]:

```
linkcode
#heatmap of confusion matrix
sns.heatmap(pd.DataFrame(cm),annot=True, cmap="Greens")
```

```
<AxesSubplot:>
```



```
#accuracy score
```

```
from sklearn.metrics import accuracy_score

accuracy=accuracy_score(Y_test,y_predict)
print("Accuracy : ",round(accuracy,2)*100,'%')
```

ACCURACY:

83.0%

CONCLUSION:

In this phase we have used some techniques based on data preprocessing and loading the above code is one of the technique used and got the accuracy of about 83.0% with around run time of 67.5s.