AI BASED DIABETES PREDICTION SYSTEM

PHASE 3 SUBMISSION

DATA LOADING & PREPROCESSING

SUBMITTED BY

KAVIYASRI S

21EC635

GOVERNMENT COLLEGE OF ENGINEERING TIRUNELVELI (9508)

- **INTRODUCTION:**

Loading and preprocessing data are crucial steps in building a diabetic prediction system. The accuracy and reliability of predictions heavily depend on the quality of data and how well it's prepared for analysis. In this phase we are going to start develop the diabetes prediction system by loading and preprocessing the dataset

- **DATA SOURCE:**

https://www.kaggle.com/datasets/mathchi/diabetes-data-set/

- **OBJECTIVE OF DATA SOURCE:**

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset.

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.

The Pima Indian Diabetes data set consists of

- → Pregnancies: Number of times pregnant
- → Glucose: Plasma glucose concentration a 2 hours in an oral glucose tolerance test
- → Blood Pressure: Diastolic blood pressure (mm Hg)
- → Skin Thickness: Triceps skin fold thickness (mm)
- → Insulin: 2-Hour serum insulin (mu U/ml)
- → BMI: Body mass index (weight in kg/(height in m)^2)
- → Diabetes Pedigree Function: Diabetes pedigree function
- → Age: Age (years)
- → Outcome: Class variable (0 or 1) 268 of 768 are 1, the others are 0

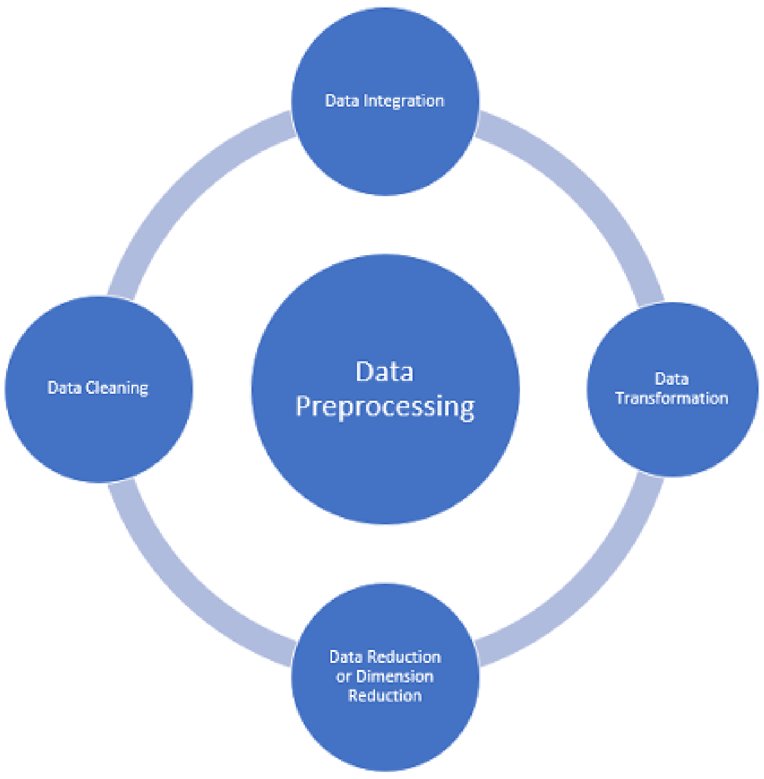
▪ WHAT IS DATA PREPROCESSING?

Data preprocessing is the process of transforming raw data into an understandable format. It is also an important step in data mining as we cannot work with raw data. The quality of the data should be checked before applying machine learning or data mining algorithms.

▪ MAJOR TASKS IN DATA PREPROCESSING

There are 4 major tasks in data preprocessing –

1. Data cleaning,
2. Data integration,
3. Data reduction
4. Data transformation.



- **Data Cleaning**

Data cleaning is the process of removing incorrect data, incomplete data, and inaccurate data from the datasets, and it also replaces the missing values

- **Data Integration**

The process of combining multiple sources into a single dataset. The Data integration process is one of the main components of data management.

- **Data Reduction**

This process helps in the reduction of the volume of the data, which makes the analysis easier yet produces the same or almost the same result. This reduction also helps to reduce storage space. Some of the data reduction techniques are dimensionality reduction, numerosity reduction, and data compression.

- ### Data Transformation

The change made in the format or the structure of the data is called data transformation. This step can be simple or complex based on the requirements. There are some methods for data transformation.

Smoothing: With the help of algorithms, we can remove noise from the dataset, which helps in knowing the important features of the dataset. By smoothing, we can find even a simple change that helps in prediction.

Aggregation: In this method, the data is stored and presented in the form of a summary. The data set, which is from multiple sources, is integrated into with data analysis description. This is an important step since the accuracy of the data depends on the quantity and quality of the data. When the quality and the quantity of the data are good, the results are more relevant.

Discretization: The continuous data here is split into intervals. Discretization reduces the data size. For example, rather than specifying the class time, we can set an interval like (3 pm-5 pm, or 6 pm-8 pm).

Normalization: It is the method of scaling the data so that it can be represented in a smaller range. Example ranging from -1.0 to 1.0.

- **IMPLEMENTATION:**

1. import libraries

```
#import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

2. Load the dataset

```
#Reading the dataset
df = pd.read_csv("/kaggle/input/diabetes-predictionsys/diabetes.csv")
df.head()
```

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

3. Analysis the dataset

```
#columns available in our dataset
df.columns
```

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

```
#Information about the data
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
```

```
#more about the dataset

df.describe().T
```

```
                          count        mean         std     min       25%  \
Pregnancies               768.0    3.845052    3.369578   0.000   1.00000   
Glucose                   768.0  120.894531   31.972618   0.000  99.00000   
BloodPressure             768.0   69.105469   19.355807   0.000  62.00000   
SkinThickness             768.0   20.536458   15.952218   0.000   0.00000   
Insulin                   768.0   79.799479  115.244002   0.000   0.00000   
BMI                       768.0   31.992578    7.884160   0.000  27.30000   
DiabetesPedigreeFunction  768.0    0.471876    0.331329   0.078   0.24375   
Age                       768.0   33.240885   11.760232  21.000  24.00000   
Outcome                   768.0    0.348958    0.476951   0.000   0.00000   

                               50%        75%     max  
Pregnancies                 3.0000    6.00000   17.00  
Glucose                   117.0000  140.25000  199.00  
BloodPressure              72.0000   80.00000  122.00  
SkinThickness              23.0000   32.00000   99.00  
Insulin                    30.5000  127.25000  846.00  
BMI                        32.0000   36.60000   67.10  
DiabetesPedigreeFunction    0.3725    0.62625    2.42  
Age                        29.0000   41.00000   81.00  
Outcome                     0.0000    1.00000    1.00  
```

4. Clean the data

Clean the data after analyse the null , impossible value and duplicated value

```
#analyse the null and duplicated values
print(f'Duplicated rows are: \n { df.duplicated().sum()} \n \n \n Null
values per column are: \n {df.isnull().sum()}\n \n \n Zero values per
column are: \n {(df == 0).sum()} , \n \n \n data types of each column is:
\n {df.dtypes}  ')
```

```
Duplicated rows are: 
 0 


 Null values per column are: 
 Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
```

```
dtype: int64


 Zero values per column are:
 Pregnancies                 111
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
Outcome                     500
dtype: int64 ,


 data types of each column is:
 Pregnancies                   int64
Glucose                       int64
BloodPressure                 int64
SkinThickness                 int64
Insulin                       int64
BMI                         float64
DiabetesPedigreeFunction    float64
Age                           int64
Outcome                       int64
dtype: object
```

```python
#analyse the impossible values
impossible_values = (df["Glucose"] == 0) | (df["BloodPressure"] == 0) |
(df["SkinThickness"] == 0) | (df["Insulin"] == 0) | (df["BMI"] == 0)
impossible_values.sum()
```

```
376
```

Replace the null value with median value

```python
#replace the null value with median value

lst=['Glucose','BloodPressure','SkinThickness','Insulin','BMI']
for i in lst:
    df[i].replace(0, np.nan, inplace=True)
    df[i].fillna(df[i].median(), inplace=True)
df.head()
```

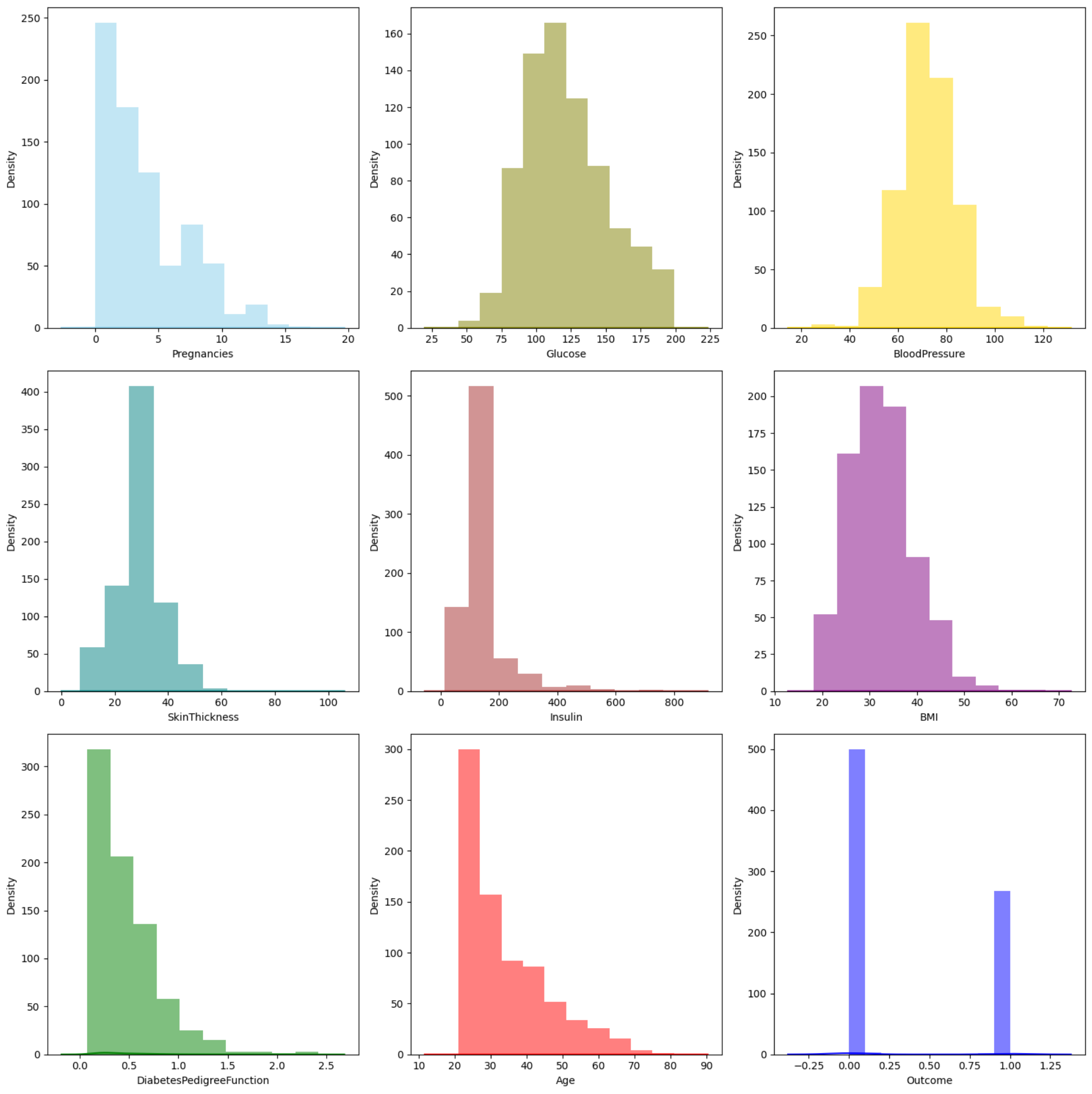
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6    148.0           72.0           35.0    125.0  33.6
1            1     85.0           66.0           29.0    125.0  26.6
2            8    183.0           64.0           29.0    125.0  23.3
3            1     89.0           66.0           23.0     94.0  28.1
4            0    137.0           40.0           35.0    168.0  43.1
```

```
   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

5.Visualize the dataset

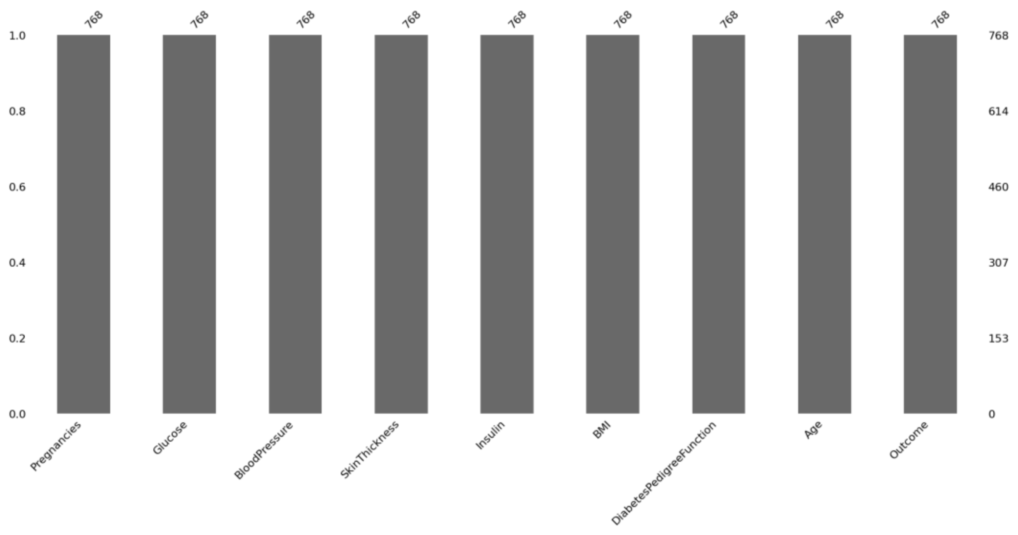
After clean the data visualize the data distribution by plotting it as histogram

```
lst1=['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFunction','Age','Outcome']
lst_col=['skyblue','olive','gold','teal','brown','purple','green','red','blue']
f, axes = plt.subplots(3, 3, figsize=(15, 15), sharex=False) # Set up the matplotlib figure
axes = axes.flatten()  # Plot a simple histogram with binsize determined automatically
for ax,k,m in zip(axes,lst1,lst_col):
    ax.hist(df[k], color=m, bins=10, alpha=0.5)
    sns.distplot(df[k], color=m, ax=ax)
plt.tight_layout()
```



Plotting Null Count Analysis Plot

```python
import missingno as msno
p = msno.bar(df)
```



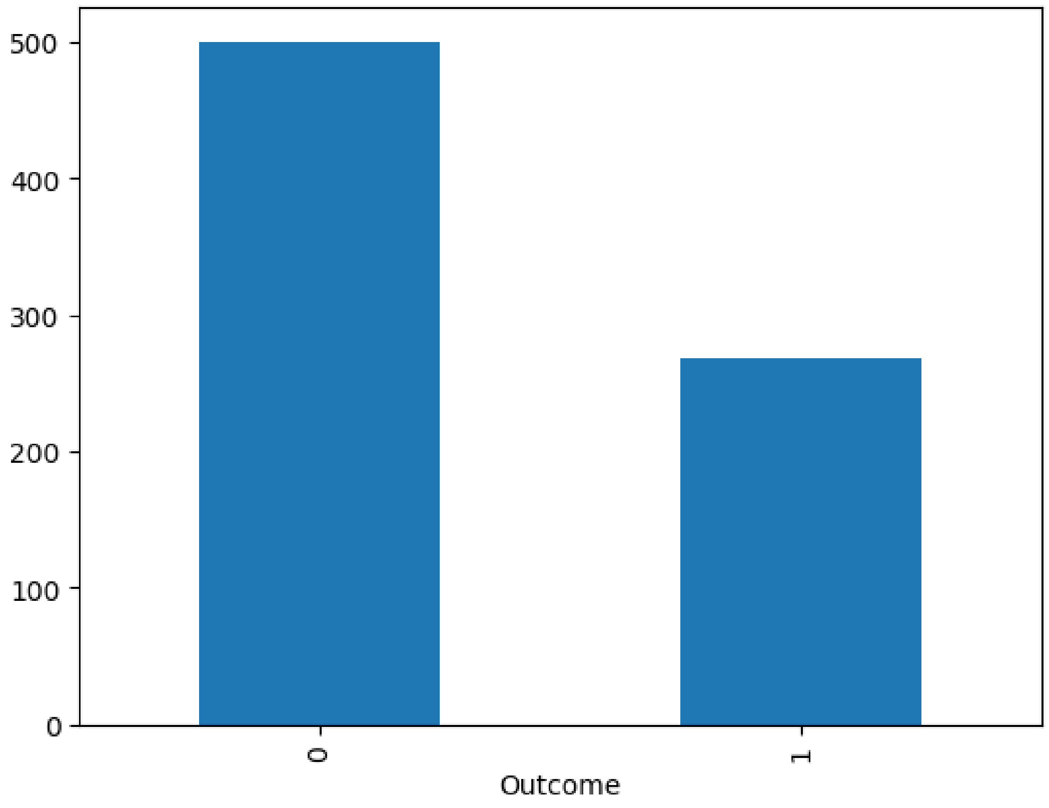
Inference: Now in the above graph also we can clearly see that there are no null values in the dataset.

check that how well our outcome column is balanced

dataset is completely imbalanced in fact the number of patients who are diabetic is half of the patients who are non-diabetic

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

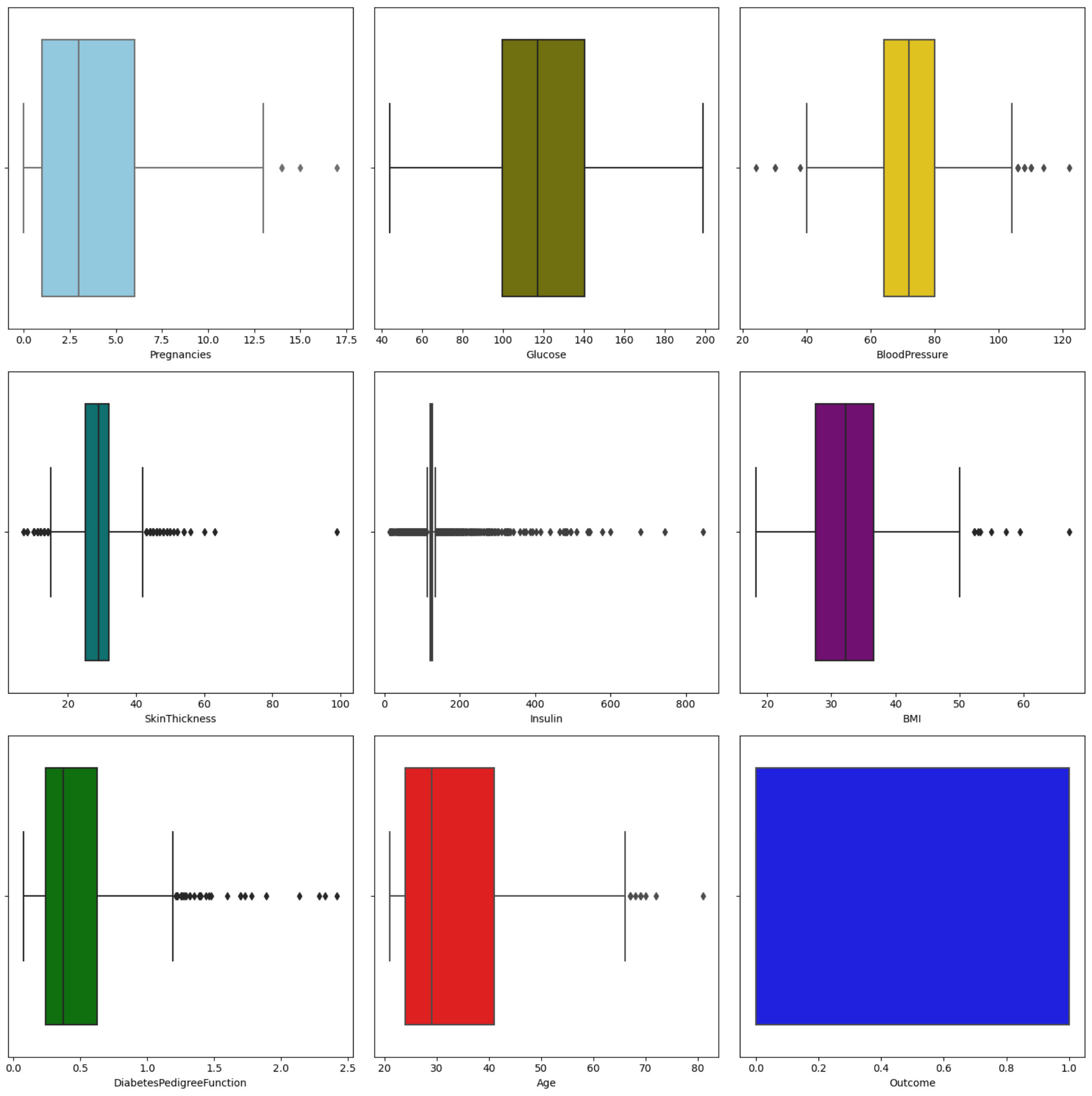
```
Outcome
0    500
1    268
Name: count, dtype: int64
```



Interference:dataset is completely imbalanced in fact the number of patients who are diabetic is half of the patients who are non-diabetic

- ## Visualization of Outliers

```
# Generate a box plot for each feature
lst1=['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI',
'DiabetesPedigreeFunction','Age','Outcome']
lst_col=['skyblue','olive','gold','teal','brown','purple','green','red','blue'
]
f, axes = plt.subplots(3, 3, figsize=(15, 15), sharex=False) # Set up the
matplotlib figure
axes = axes.flatten()  # Plot a simple histogram with binsize determined
automatically
for ax,k,m in zip(axes,lst1,lst_col):
    sns.boxplot(data=df, x=k, color=m, ax=ax)
plt.tight_layout()
plt.show()
```

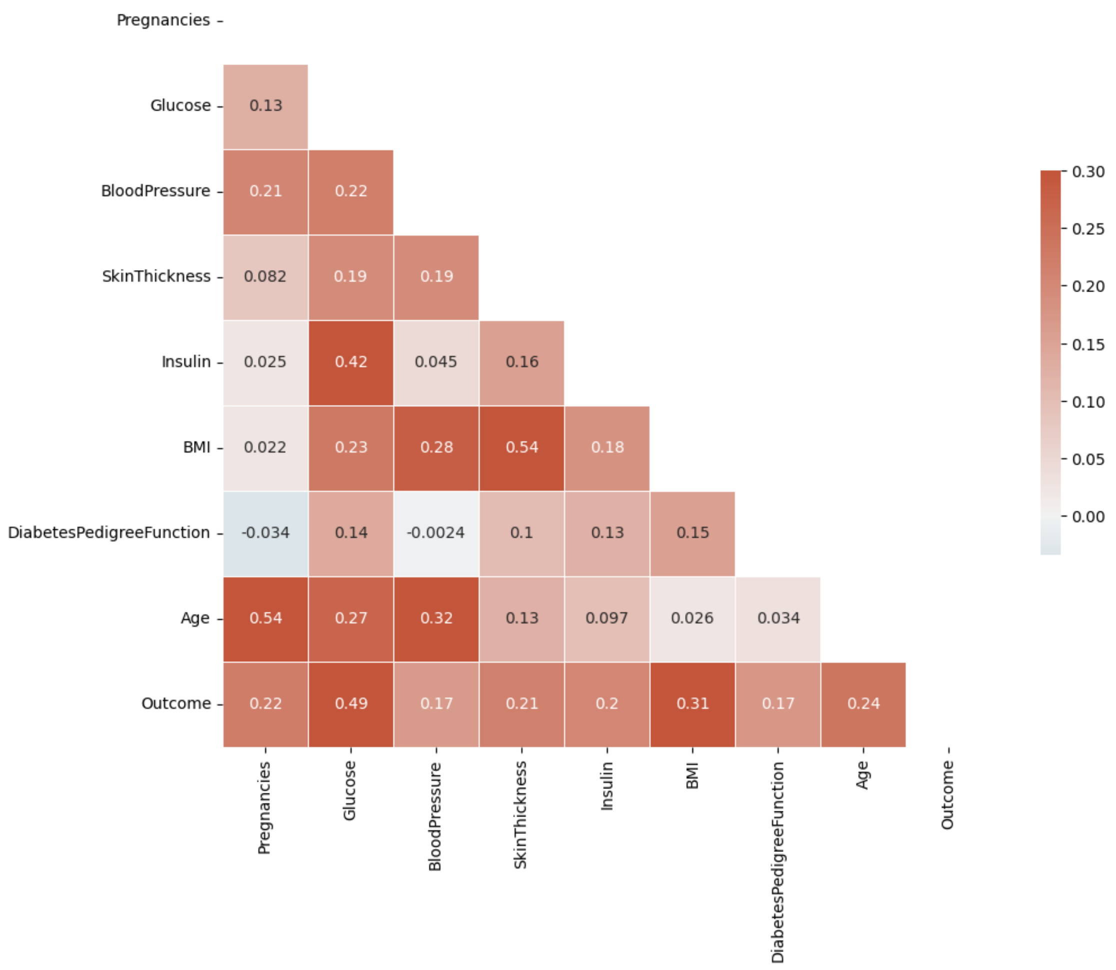


- ## Check the Relationship Between Variables

Then we wanna check the relationships between the different variables(columns). This can provide insights into which variables are strongly or weakly associated with each other.

```
corr = df.corr() # Compute the correlation matrix
mask = np.triu(np.ones_like(corr, dtype=bool)) # Generate a mask for the
```

```
upper triangle
f, ax = plt.subplots(figsize=(11, 9)) # Set up the matplotlib figure
cmap = sns.diverging_palette(230, 20, as_cmap=True) # Generate a custom
diverging colormap
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=0.3, center=0, square=True,
linewidths=.5, cbar_kws={"shrink": .5}, annot=True) # Draw the heatmap with
the mask and correct aspect ratio
plt.tight_layout()
```



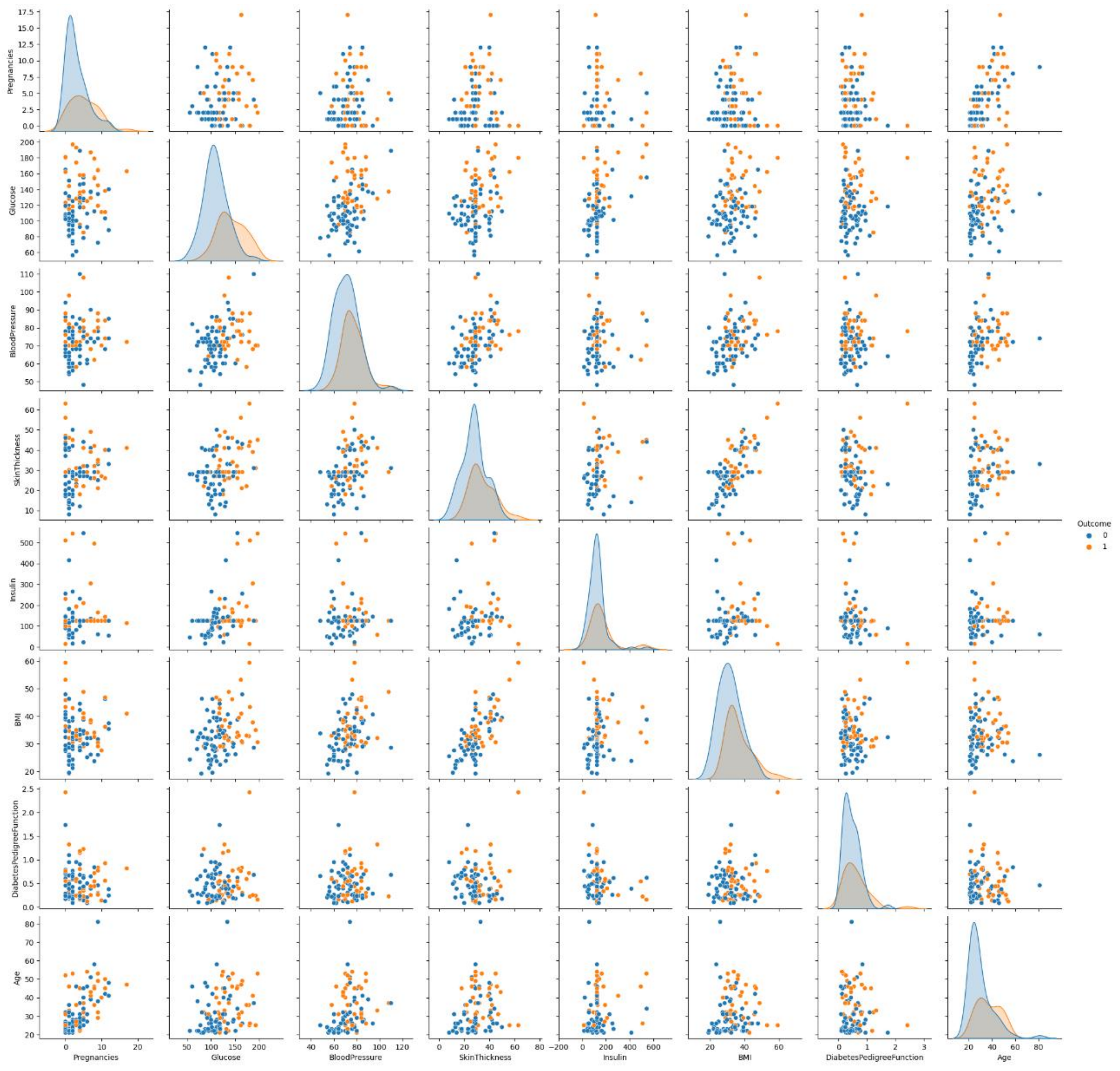
Interference: The correlation coefficient values range from -1 to 1. If the correlation coefficient is close to 1, it means that there is a strong positive correlation between the two variables. When it is close to -1, the variables have a strong negative correlation. Glucose, Age and BMI are moderately correlated with Outcome. Pregnancies and Age show a strong correlation.

- ## Pairwise Variable Relationship

This can be very helpful to understand how the variables interact with each other and identify any potential patterns or trends in the data.

```
# We'll use a sample of the data to make the pairplot faster to generate
df_sample = df.sample(100, random_state=1)
```

```
# Create a pairplot
sns.pairplot(df_sample, hue="Outcome")
plt.show()
```



6.Split the data

we need to split the data into a training set and a test set. This allows us to evaluate how well our model generalizes to unseen data. We'll use 80% of the data for training and 20% for testing.

```
from sklearn.model_selection import train_test_split
X = df.drop("Outcome", axis=1)
y = df["Outcome"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)# Split the data into a training set and a test set
X_train.shape, X_test.shape
```

```
((614, 8), (154, 8))
```

7.Data Transformation

Transform data as needed, for example, scaling

a)Before Scaling

```
df.head()
```

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6    148.0           72.0           35.0    125.0  33.6
1            1     85.0           66.0           29.0    125.0  26.6
2            8    183.0           64.0           29.0    125.0  23.3
3            1     89.0           66.0           23.0     94.0  28.1
4            0    137.0           40.0           35.0    168.0  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

b)After Scaling

Scaled it by trained model

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler() # Initialize the scaler
scaler.fit(X_train) # Fit the scaler to the training data
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
pd.DataFrame(X_train_scaled, columns=['Pregnancies',
'Glucose', 'BloodPressure','SkinThickness', 'Insulin', 'BMI',
'DiabetesPedigreeFunction', 'Age']).head()
```

```
   Pregnancies   Glucose  BloodPressure  SkinThickness   Insulin       BMI  \
0    -0.526397 -1.256881      -0.018995       0.034298 -0.175620 -0.007450
1     1.588046 -0.326051       0.808174      -0.560583 -0.175620 -0.599092
2    -0.828460  0.571536      -2.169636      -1.155463 -0.652193 -0.526941
3    -1.130523  1.302903      -1.838768       0.034298 -0.175620 -1.508200
4     0.681856  0.405316       0.642740       0.986106  2.604392  1.998360

   DiabetesPedigreeFunction       Age
0                 -0.490735 -1.035940
1                  2.415030  1.487101
2                  0.549161 -0.948939
```

```
3                 -0.639291  2.792122
4                 -0.686829  1.139095
```

Scaled it by test model

```
X_test_scaled = scaler.transform(X_test)
pd.DataFrame(X_test_scaled, columns=['Pregnancies',
'Glucose', 'BloodPressure','SkinThickness', 'Insulin', 'BMI',
'DiabetesPedigreeFunction', 'Age']).head()
```

```
   Pregnancies   Glucose  BloodPressure  SkinThickness   Insulin       BMI  \
0     0.681856 -0.791466      -1.177033       0.510202  0.561935  0.237865
1    -0.526397 -0.326051       0.229156       0.391226 -0.175620  0.483180
2    -0.526397 -0.459026      -0.680731       0.034298 -0.175620 -0.223904
3     1.285983 -0.492270       0.642740       0.034298 -0.175620 -1.118582
4     0.983919  0.471804       1.469910       0.034298 -0.175620 -0.353777

   DiabetesPedigreeFunction       Age
0                 -0.116372  0.878091
1                 -0.954231 -1.035940
2                 -0.924520 -1.035940
3                  1.149329  0.095078
4                 -0.770021  1.487101
```

- **CONCLUSION:**

Loading and preprocessing the dataset are crucial steps in building a prediction system. The use of Pandas for data loading and scikit-learn for preprocessing ensures efficiency and consistency in handling the data

These steps set the foundation for subsequent stages, such as model training and evaluation, in the development of a diabetes prediction system.