Diabetes Patients Data Analysis and Predictive Model

```
#Import all neccessary libraries
import pandas as pd
import numpy as np
import seaborn as sns
from matplotlib import pyplot as plt
```

```
#load the dataset from csv file
df=pd.read_csv('diabetes.csv')
```

Understanding Data Anomilies:

- **Pregnancies:** This column represents the number of pregnancies the individual has had.
- **Glucose**: Represents the plasma glucose concentration (measured in milligrams per deciliter, mg/dL) in a 2-hour oral glucose tolerance test.
- **BloodPressure:** Represents the diastolic blood pressure (measured in mm Hg) of the individual.
- **SkinThickness:** Represents the thickness of the skinfold at the triceps (measured in millimeters, mm).
- **Insulin:** Represents the serum insulin level (measured in milli-international units per milliliter, mIU/mL).
- **BMI (Body Mass Index):** Represents the individual's body mass index, which is a measure of body fat based on height and weight (weight in kg / (height in meters)^2).
- **DiabetesPedigreeFunction:** Represents a diabetes pedigree function that quantifies the diabetes genetic risk score based on family history.
- **Age:** Represents the age of the individual in years.
- **Outcome:** This is the target variable and represents whether the individual has diabetes (1) or does not have diabetes (0)

```
#Display few records from dataset
df.head()
```

	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

```
#How many rows and Columns are there in dataset
df.shape
```

```
(768, 9)
```

```
#Checks if there any missing values (There is no missing or 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
```

```
#Understand Data Types of Columns
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
```

```
#Analyze data statistics by describing whole dataset
df.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	7(
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	
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	

- There are 768 Records in the dataset

Mean(Average) :

- On average, individuals in the dataset have approximately 3.85 pregnancies.
- The mean plasma glucose concentration is around 120.89 mg/dL.
- The mean diastolic blood pressure is approximately 69.11 mm Hg.
- The average skin thickness is roughly 20.54 mm.
- The average serum insulin level is about 79.80 mIU/mL.
- The mean BMI is approximately 31.99.
- The average diabetes pedigree function score is roughly 0.47.
- The average age of individuals in the dataset is around 33.24 years.
- The mean outcome is approximately 0.35, which suggests that, on average, about 35% of individuals in the dataset have diabetes (1) while the rest do not (0).

Minimum

- Most of data anomilies minimum values are zero (0)
- Minimum Age in records is 21 Year and she is youngest in whole dataset.

Maximum

- Maximum Pregnencies is about 17,
- Maximum Glucose values is 199,
- Maximum BloodPressure Values is 122
- Maximumn BMI is 67
- Maximum Age is 81

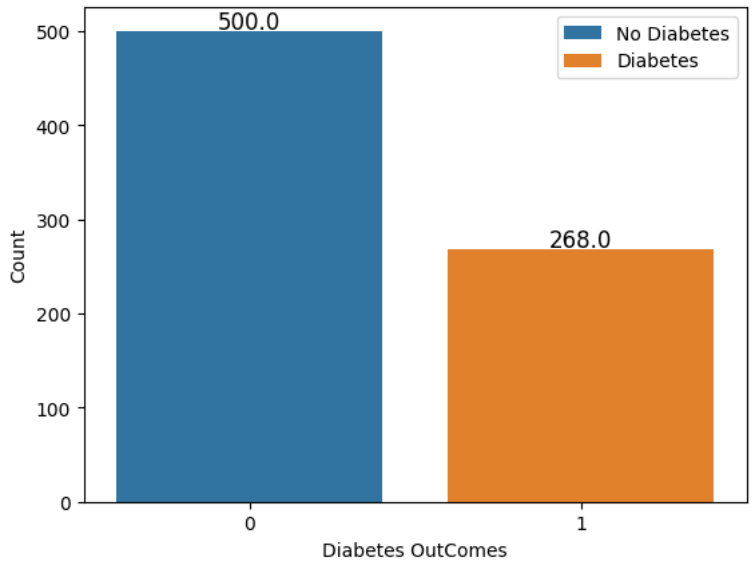
Percentile

- About 50% of Age is less than or equal to 29 year
- About 50% of records has BMI less than equal to 32

How is the data distribution of the target varibale (OutCome)?

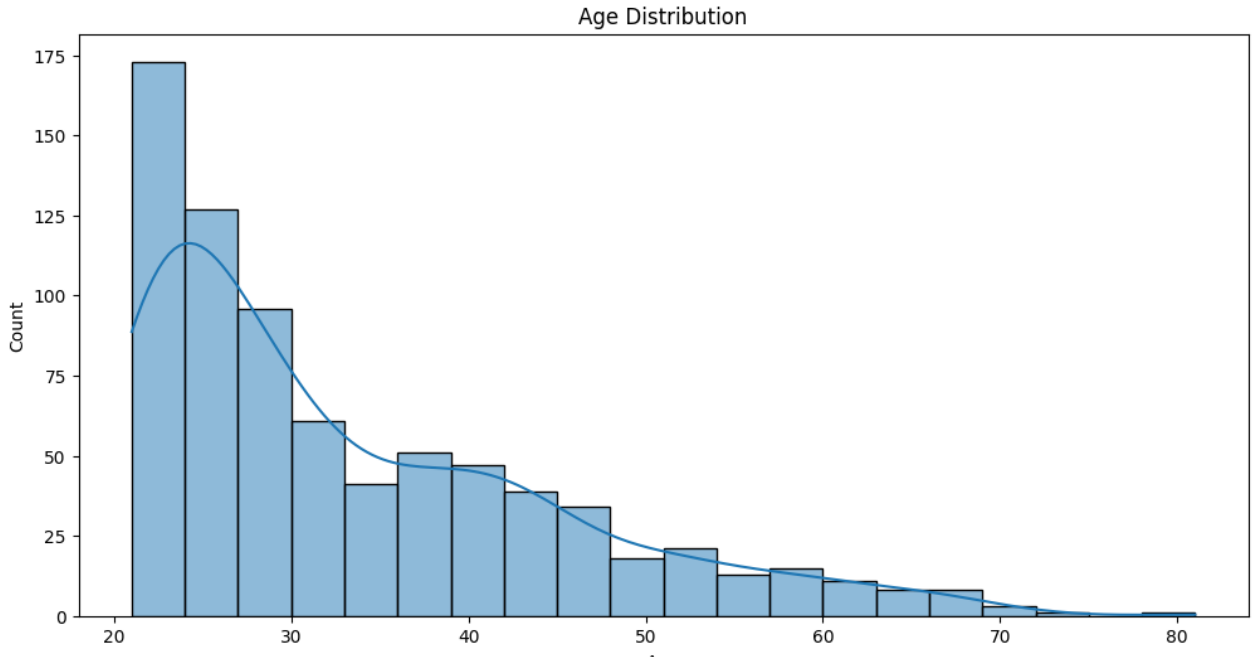
```
sns.countplot(x='Outcome',data=df,label=['No Diabetes','Diabetes'])
```

```
plt.legend()
plt.xlabel('Diabetes OutComes')
plt.ylabel('Count')
#Showing Annotation of value labels on Bars
ax = plt.gca()
for p in ax.patches:
    ax.annotate(f'{p.get_height()}', (p.get_x() + p.get_width() / 2., p.get_height()), ha='center', va='center', fontsize=12, color='blac
plt.show()
```



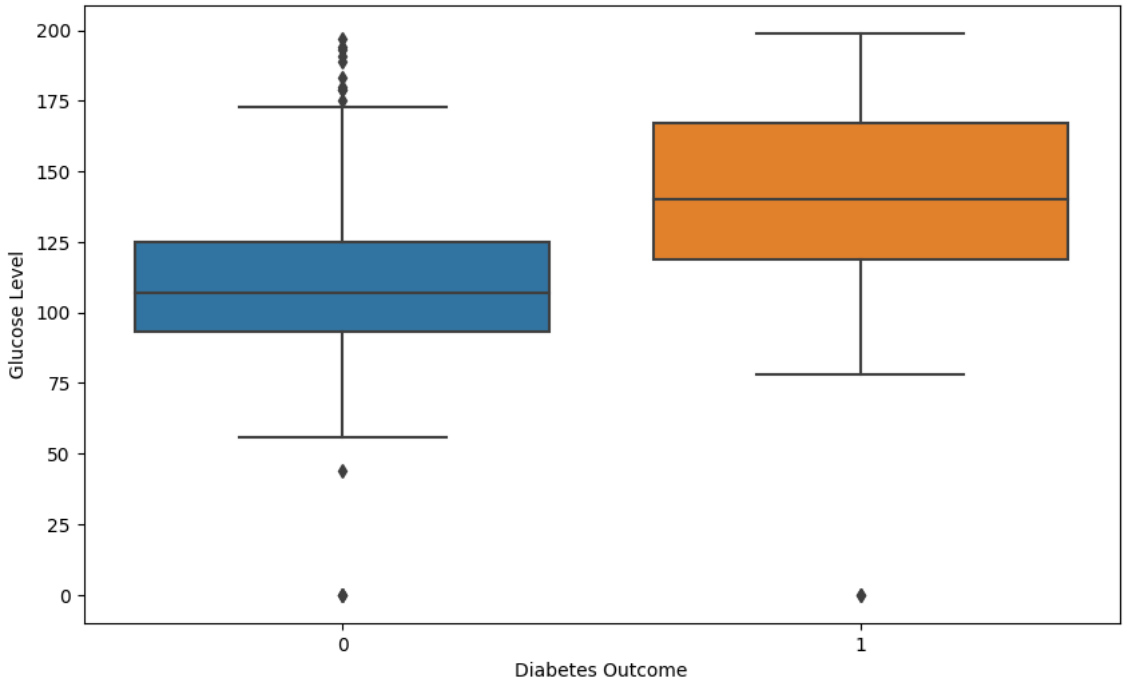
What is the distribution of Age and BMI?

```
plt.figure(figsize=(12, 6))
#Ploting Hisptogram with kde and bins of 20
sns.histplot(df['Age'], bins=20, kde=True)
plt.xlabel('Age')
plt.title('Age Distribution')
plt.show()

plt.figure(figsize=(12, 6))
sns.histplot(df['BMI'], bins=20, kde=True)
plt.xlabel('BMI')
plt.title('BMI Distribution')
plt.show()
```



How do Glucose levels vary with Outcome?

```
plt.figure(figsize=(10, 6))
sns.boxplot(x='Outcome', y='Glucose', data=df)
plt.xlabel('Diabetes Outcome')
plt.ylabel('Glucose Level')
plt.show()
```

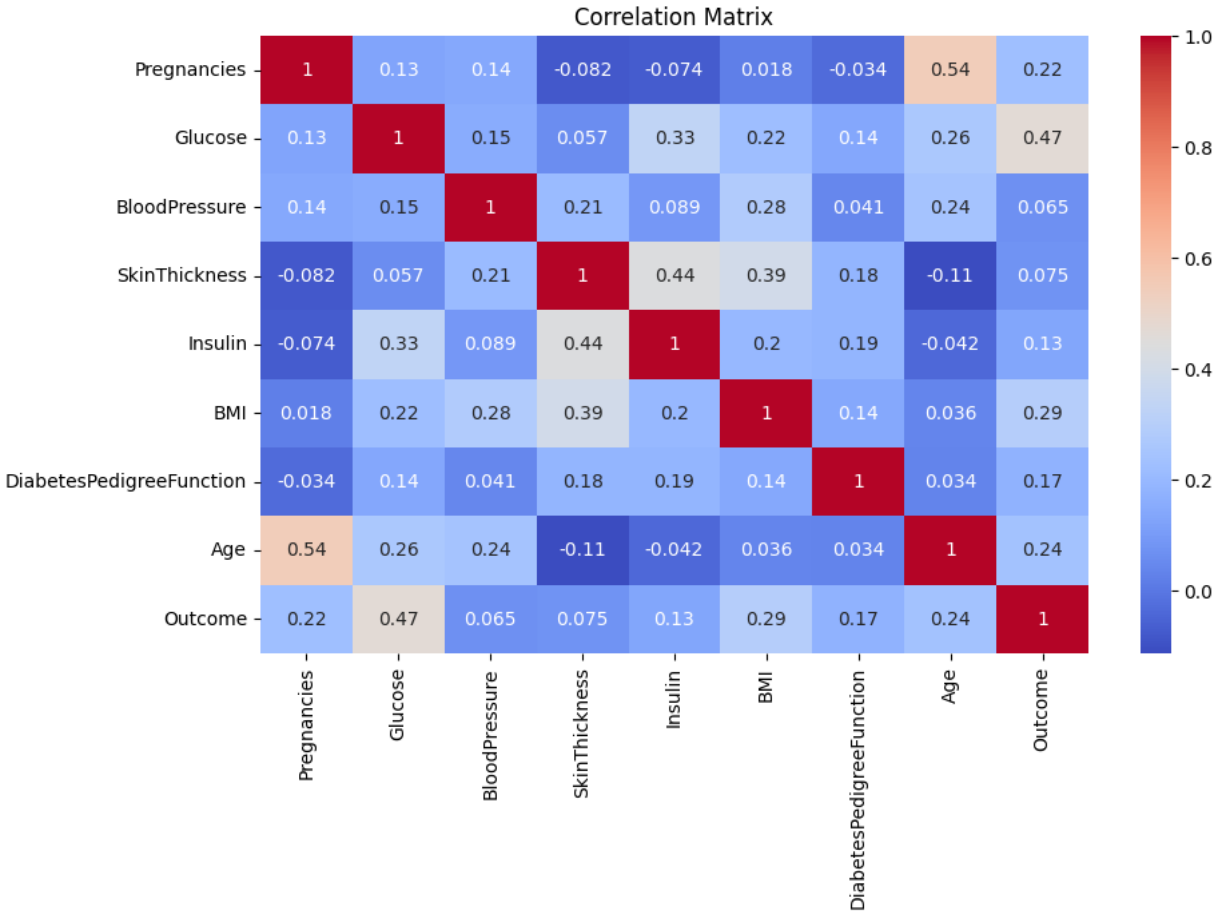



```
#Finding People Counts whos Glucos is 180
df[(df['Glucose']>180) & (df['Outcome']==1) &(df['Insulin']>0)].count()
```

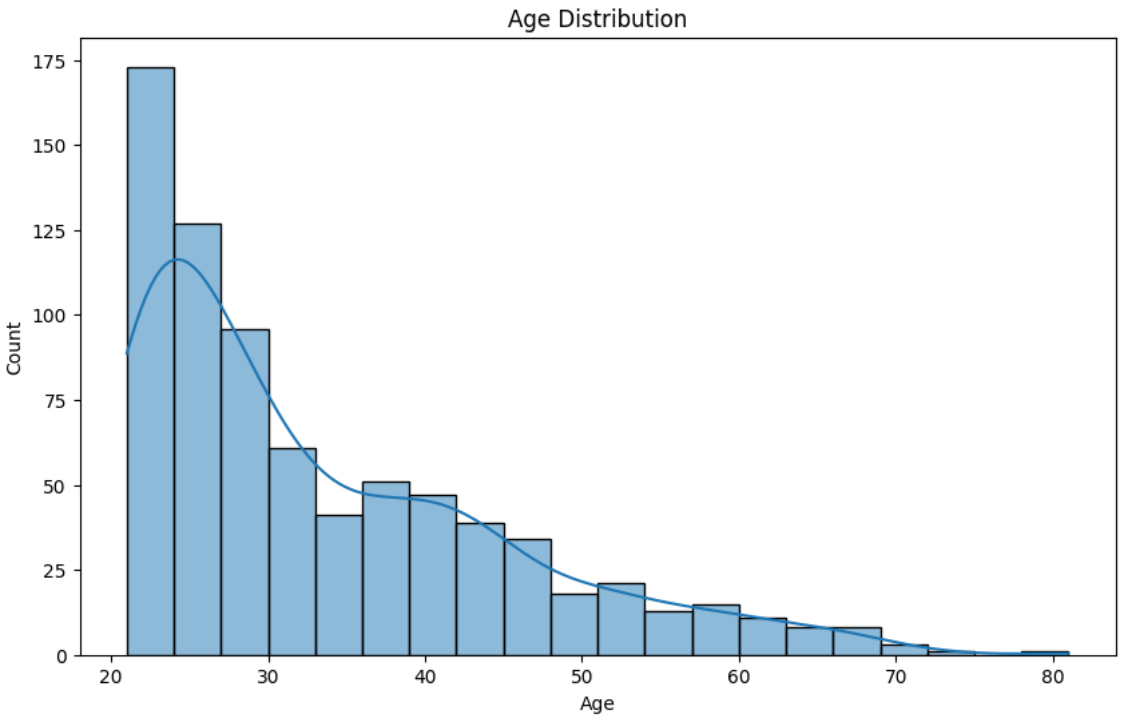
```
Pregnancies                 19
Glucose                     19
BloodPressure               19
SkinThickness               19
Insulin                     19
BMI                         19
DiabetesPedigreeFunction    19
Age                         19
Outcome                     19
dtype: int64
```

```
#Finding Corelations
correlation_matrix = df.corr()
```

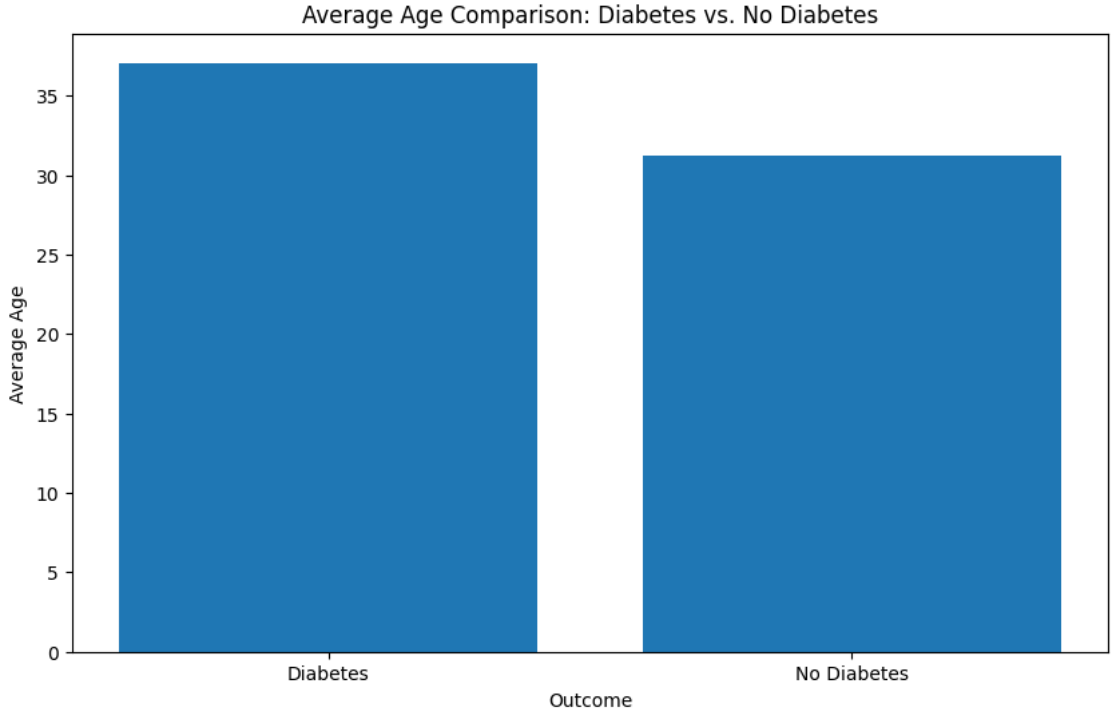
```
plt.figure(figsize=(10, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```



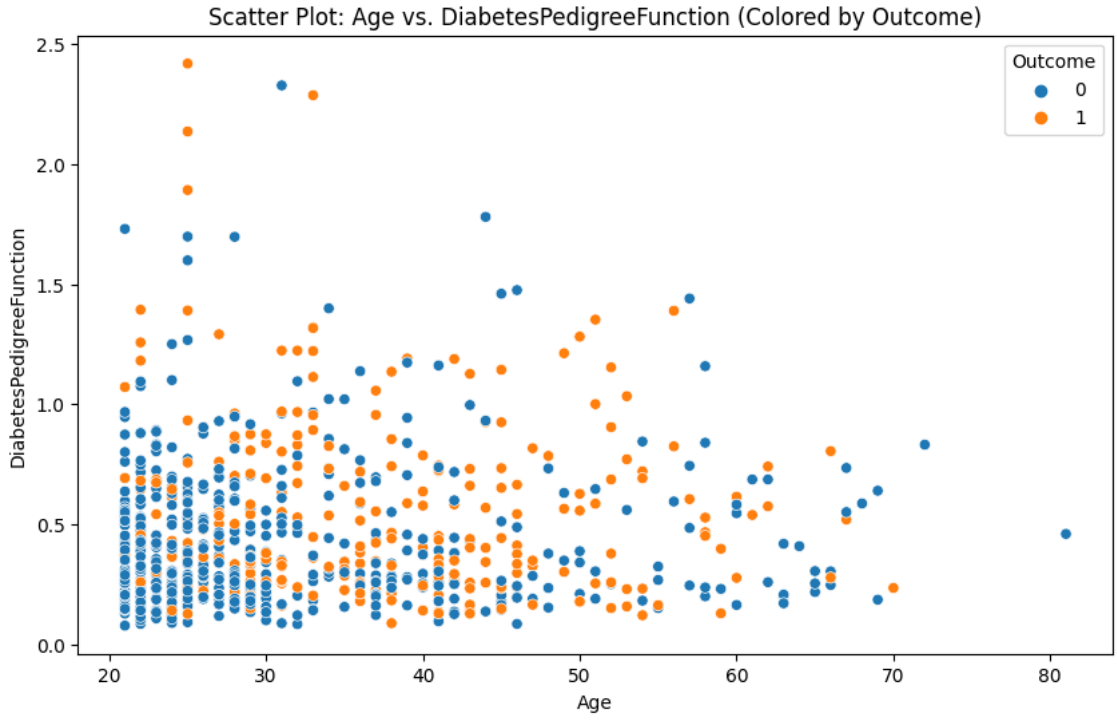

```
# Univariate analysis for 'Age'
plt.figure(figsize=(10, 6))
sns.histplot(df['Age'], bins=20, kde=True)
plt.xlabel('Age')
plt.title('Age Distribution')
plt.show()
```



```
# Diabetes vs Non Diabetes Ages Group
# Separate the data into two groups: Diabetes and No Diabetes
diabetes_group = df[df['Outcome'] == 1]
no_diabetes_group = df[df['Outcome'] == 0]
plt.figure(figsize=(10, 6))
plt.bar(['Diabetes', 'No Diabetes'], [diabetes_group['Age'].mean(), no_diabetes_group['Age'].mean()])
plt.xlabel('Outcome')
plt.ylabel('Average Age')
plt.title('Average Age Comparison: Diabetes vs. No Diabetes')
plt.show()
```

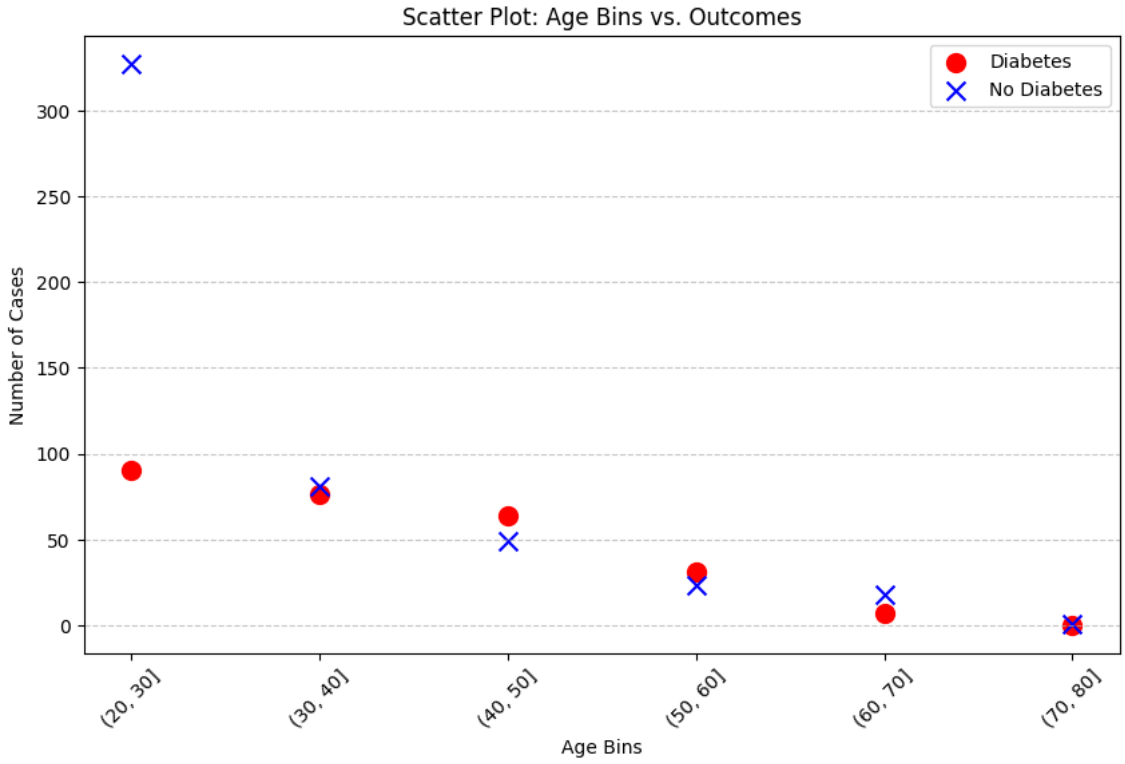



```
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='Age', y='DiabetesPedigreeFunction', hue='Outcome')
plt.xlabel('Age')
plt.ylabel('DiabetesPedigreeFunction')
plt.title('Scatter Plot: Age vs. DiabetesPedigreeFunction (Colored by Outcome)')
plt.show()
```



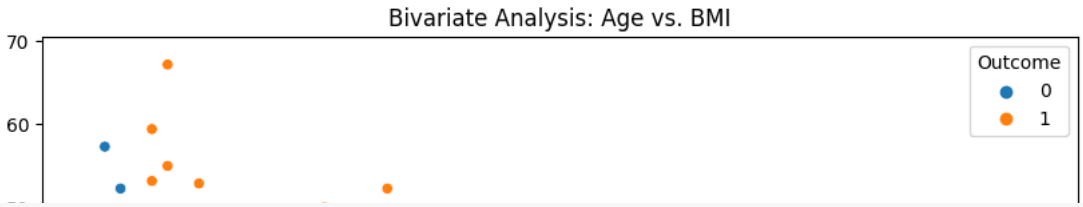

```
#Number of Cases According to How many Diabetes Patients are there
age_bins = [20, 30, 40, 50, 60, 70, 80]
# Group data into age bins and count outcomes
```

```
age_grouped = df.groupby(pd.cut(df['Age'], bins=age_bins))['Outcome'].value_counts().unstack().fillna(0)

# Convert age bin labels to strings
age_grouped.index = age_grouped.index.astype(str)

# Create a scatter plot to visualize age bins vs. outcomes
plt.figure(figsize=(10, 6))
plt.scatter(age_grouped.index, age_grouped[1], color='red', label='Diabetes', s=100, marker='o')
plt.scatter(age_grouped.index, age_grouped[0], color='blue', label='No Diabetes', s=100, marker='x')
plt.xlabel('Age Bins')
plt.ylabel('Number of Cases')
plt.xticks(rotation=45)
plt.legend()
plt.title('Scatter Plot: Age Bins vs. Outcomes')
plt.grid(True, axis='y', linestyle='--', alpha=0.7)
plt.show()
```

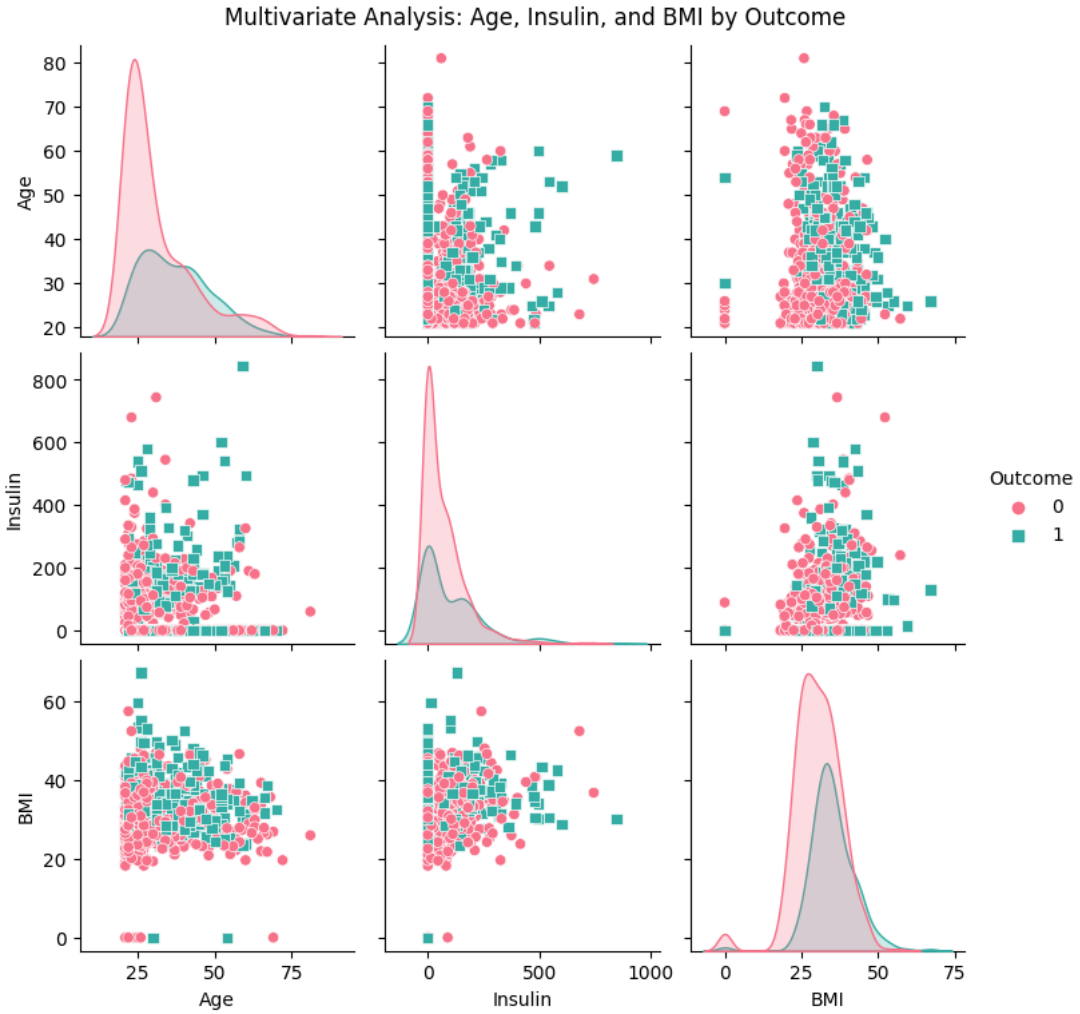


Moslty are between Age group of 20 to 30

```
# Example: Bivariate analysis between 'Age' and 'BMI'
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Age', y='BMI', data=df, hue='Outcome')
plt.xlabel('Age')
plt.ylabel('BMI')
plt.title('Bivariate Analysis: Age vs. BMI')
plt.show()
```




```
# Create a pairplot for multivariate analysis
sns.pairplot(data=df, vars=['Age', 'Insulin', 'BMI'], hue='Outcome', markers=["o", "s"], diag_kind='kde', palette='husl')
plt.suptitle("Multivariate Analysis: Age, Insulin, and BMI by Outcome", y=1.02)
plt.show()
```



Data Preprocessing

- Before building the model, we should preprocess the data. This includes handling missing values, encoding categorical variables (if any), and splitting the dataset into features (X) and the target variable (y).
- But in Our Dataset , there is no missing or null values so we direclty went on data preprocessing

```
# Define independent variables (features) and the target variable
# Since Outcomes in 0 and 1 , We will perform Logistic Regression Analysis
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score
X = df.drop('Outcome', axis=1) #Include all df columns except Outcome
y = df['Outcome'] #Only Outcome colum

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Creating Logistic Regression ( 0, 1)
model = LogisticRegression()
```

```
# Fitting into Model
model.fit(X_train, y_train)

# Prediction on Test
y_pred = model.predict(X_test)

# Evaluating model  performance report
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print(f'Accuracy: {accuracy:.2f}')
print(report)
```

```
Accuracy: 0.75
              precision    recall  f1-score   support

           0       0.81      0.79      0.80        99
           1       0.64      0.67      0.65        55

    accuracy                           0.75       154
   macro avg       0.73      0.73      0.73       154
weighted avg       0.75      0.75      0.75       154
```

Accuracy: Accuracy is a measure of how well the model correctly predicted both classes (0 and 1). In this case, the model achieved an accuracy of 0.75, which means it correctly classified 75% of the total samples.

Precision: Precision is a measure of the model's ability to correctly identify positive cases (1) among all the cases it predicted as positive. Precision for class 0 is 0.81, which means that out of all the cases the model predicted as class 0, 81% were correctly classified. Precision for class 1 is 0.64, indicating that 64% of the cases predicted as class 1 were correct.

Recall (Sensitivity): Recall measures the model's ability to correctly identify positive cases (1) out of all actual positive cases. Recall for class 0 is 0.79, meaning the model correctly identified 79% of actual class 0 cases. Recall for class 1 is 0.67, indicating that 67% of actual class 1 cases were correctly identified.

F1-Score: The F1-score is the harmonic mean of precision and recall. It provides a balanced measure of the model's performance. The F1-score for class 0 is 0.80, and for class 1, it's 0.65.

Support: Support indicates the number of samples in each class.

Macro Avg: The macro average calculates the average of precision, recall, and F1-score for both classes, giving equal weight to each class. In this case, the macro average F1-score is 0.73.

Weighted Avg: The weighted average calculates the average of precision, recall, and F1-score for both classes, but it gives more weight to the class with a larger number of samples. In this case, the weighted average F1-score is 0.75.

From this classification report, you can conclude the following:

The model's overall accuracy is 75%, which means it is correct in predicting diabetes outcomes for 75% of the samples.

Precision and recall values provide insights into how well the model performs for each class. In this case, class 0 (no diabetes) has higher precision and recall than class 1 (diabetes), indicating better performance for class 0.

The F1-score balances precision and recall and provides a single metric to evaluate the model's performance. The weighted average F1-score of 0.75 suggests a reasonable overall performance of the model.

```
# Make Prediction:  input features for prediction
#create a new input dataframe that to be predict
new_data = pd.DataFrame({
    'Pregnancies': [0],
    'Glucose': [118],
    'BloodPressure': [72],
    'SkinThickness': [35],
    'Insulin': [0],
    'BMI': [25.6],
    'DiabetesPedigreeFunction': [0.407],
    'Age': [16]
})


predictions = model.predict(new_data)
print("Predicted Outcome (0=No Diabetes, 1=Diabetes):", predictions)
```

```
Predicted Outcome (0=No Diabetes, 1=Diabetes): [0]
```

```
#Adding GUI To Model Install neccessery modules
#!pip install dash
#!pip install plotly
```

```
import pandas as pd
import numpy as np
import dash
import dash_core_components as dcc
import dash_html_components as html
from dash.dependencies import Input, Output
import plotly.express as px
from sklearn.linear_model import LogisticRegression

# Load the dataset from the CSV file
df = pd.read_csv('diabetes.csv')

# Create a logistic regression model
model = LogisticRegression()
X = df.drop('Outcome', axis=1)
y = df['Outcome']
model.fit(X, y)

# Initialize the Dash app
app = dash.Dash(__name__)

# Define the layout of the dashboard
app.layout = html.Div([
    html.H1("Diabetes Prediction Dashboard"),

    # Input fields for user to enter data
    html.Label("Enter Values for Prediction:"),
    dcc.Input(id='pregnancies', type='number', placeholder='Pregnancies'),
    dcc.Input(id='glucose', type='number', placeholder='Glucose'),
    dcc.Input(id='bloodpressure', type='number', placeholder='BloodPressure'),
    dcc.Input(id='skinthickness', type='number', placeholder='SkinThickness'),
    dcc.Input(id='insulin', type='number', placeholder='Insulin'),
    dcc.Input(id='bmi', type='number', placeholder='BMI'),
    dcc.Input(id='diabetespedigreefunction', type='number', placeholder='DiabetesPedigreeFunction'),
    dcc.Input(id='age', type='number', placeholder='Age'),

    # Display the prediction result
    html.Div(id='prediction-result'),

    # Scatter plots for Age vs. BMI and Insulin vs. BloodPressure
    dcc.Graph(id='age-bmi-scatter'),
    dcc.Graph(id='insulin-bp-scatter')
])

# Define callback function to make predictions and update scatter plots
@app.callback(
    [Output('prediction-result', 'children'),
     Output('age-bmi-scatter', 'figure'),
     Output('insulin-bp-scatter', 'figure')],
    [Input('pregnancies', 'value'),
     Input('glucose', 'value'),
     Input('bloodpressure', 'value'),
     Input('skinthickness', 'value'),
     Input('insulin', 'value'),
     Input('bmi', 'value'),
     Input('diabetespedigreefunction', 'value'),
     Input('age', 'value')]
)
def predict_diabetes(pregnancies, glucose, bloodpressure, skinthickness, insulin, bmi, dpf, age):
    # Create a new input dataframe for prediction
    new_data = pd.DataFrame({
        'Pregnancies': [pregnancies],
        'Glucose': [glucose],
        'BloodPressure': [bloodpressure],
        'SkinThickness': [skinthickness],
        'Insulin': [insulin],
        'BMI': [bmi],
        'DiabetesPedigreeFunction': [dpf],
        'Age': [age]
    })

    # Make predictions
```

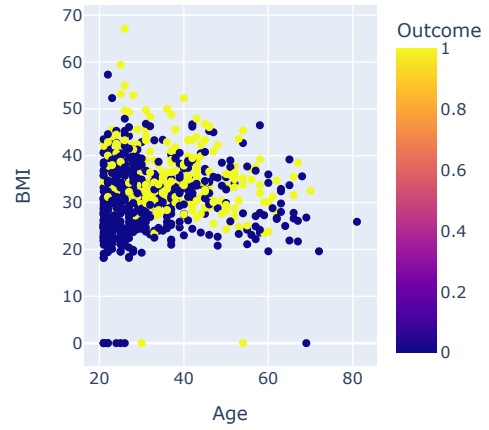
```
    predictions = model.predict(new_data)

    # Convert predictions to human-readable text
    result_text = "Predicted Outcome: " + ("Yes" if predictions[0] == 1 else "No")

    # Create scatter plots
    scatter_age_bmi = px.scatter(df, x='Age', y='BMI', color='Outcome', title='Age vs. BMI')
    scatter_insulin_bp = px.scatter(df, x='Insulin', y='BloodPressure', color='Outcome', title='Insulin vs. BloodPressure')

    return result_text, scatter_age_bmi, scatter_insulin_bp

# Run the app
if __name__ == '__main__':
    app.run_server(debug=True)
```

Diabetes Prediction Dashboard

Enter Values for Prediction:

0	148	110
36	35	36
0.7	66	

Predicted Outcome: Yes



0s completed at 6:13 PM