Diabetes Prediction

Certainly! A diabetes prediction project typically involves using machine learning techniques to build a model that can predict whether an individual is likely to have diabetes based on certain features or attributes. Below is a detailed project description for a diabetes prediction project:

Project Title: Diabetes Prediction

Objective:

The main objective of this project is to develop a machine learning model that can predict whether a person is likely to have diabetes or not based on various health-related features. The project aims to assist healthcare professionals in early diagnosis and intervention for individuals at risk of diabetes.

Dataset:

The dataset used for this project is a collection of health-related information from individuals, including features such as pregnancies, glucose levels, blood pressure, skin thickness, insulin levels, BMI (Body Mass Index), diabetes pedigree function, and age. The dataset also includes an 'Outcome' column indicating whether the individual has diabetes (1) or not (0).

Project Workflow:

1. **Data Exploration and Preprocessing:**
   - Load the dataset and examine its structure, check for missing values.
   - Visualize the distribution of features and the target variable ('Outcome').
   - Handle missing values by replacing zeros with NaN in specific columns.
   - Visualize and analyze the distribution of NaN values.
   - Replace NaN values with the mean or median of the corresponding columns.
   - Visualize histograms and pair plots to understand the relationships between variables.
2. **Data Analysis:**
   - Compute descriptive statistics to understand the central tendency and spread of each feature.
   - Explore the correlation matrix to identify relationships between features and the target variable.
3. **Data Visualization:**
   - Visualize the distribution of the 'Outcome' variable using count plots and pie charts.
   - Create pair plots to visualize relationships between different features, considering the 'Outcome'.
4. **Data Modeling:**
   - Split the dataset into training and testing sets using the train_test_split method.
   - Choose a machine learning algorithm suitable for binary classification (e.g., Logistic Regression, Random Forest).
   - Train the selected model on the training set.
5. **Model Evaluation:**
   - Make predictions on the test set.
   - Evaluate the model's performance using metrics such as accuracy, precision, recall, and F1-score.
   - Display a confusion matrix to understand the model's performance in classifying true positives, true negatives, false positives, and false negatives.
6. **Results and Interpretation:**
   - Present the model's accuracy and performance metrics.
   - Interpret the results and discuss the implications for diabetes prediction.
   - Provide insights into features that contribute most to the model's predictions.

Tools and Libraries:

- Python (programming language)
- Pandas, NumPy (data manipulation)
- Matplotlib, Seaborn (data visualization)
- Scikit-learn (machine learning)

Deliverables:

- Jupyter Notebook or Python script containing the entire project code.
- A report documenting the project, including data exploration, methodology, results, and conclusions.

Future Enhancements:

- Explore the possibility of using different machine learning algorithms for comparison.
- Fine-tune hyperparameters to improve model performance.
- Investigate additional features or data sources for improved predictions.

This project aims to contribute to the field of healthcare by providing a tool for early diabetes prediction, enabling timely interventions and improving patient outcomes.

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import missingno as msno
```

In [2]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from pandas.plotting import scatter_matrix
```

In [3]:
```
from sklearn.metrics import precision_score, recall_score, f1_score
```

In [4]:
```
#Import Data
data = pd.read_csv("diabetes.csv")
data
```

Out[4]:

	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
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

In [5]:
```
#Names of Column available
data.columns
```

Out[5]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [6]:
```
data.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
```

In [7]:
```
data.describe()
```

Out[7]:

	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

In [8]:
```
data.std()
```

Out[8]:
```
Pregnancies                 3.369578
Glucose                    31.972618
BloodPressure              19.355807
SkinThickness              15.952218
Insulin                   115.244002
BMI                         7.884160
DiabetesPedigreeFunction    0.331329
Age                        11.760232
Outcome                     0.476951
dtype: float64
```

In [9]:
```
#Checking for Null Values or Not
data.isnull()
```

Out[9]:

	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
...	...	...	...	...	...	...	...	...	...
763	False	False	False	False	False	False	False	False	False
764	False	False	False	False	False	False	False	False	False
765	False	False	False	False	False	False	False	False	False
766	False	False	False	False	False	False	False	False	False
767	False	False	False	False	False	False	False	False	False

768 rows × 9 columns

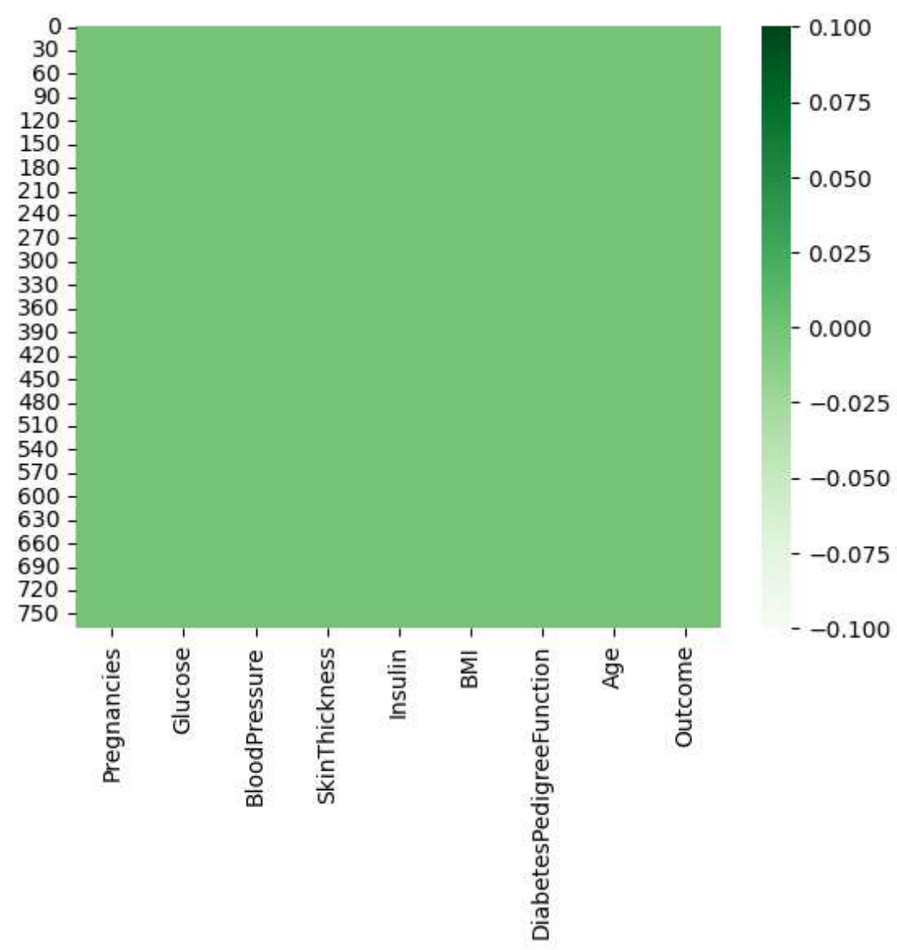
In [10]:
```
#Checking for total Null Values
data.isnull().sum()
```

Out[10]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

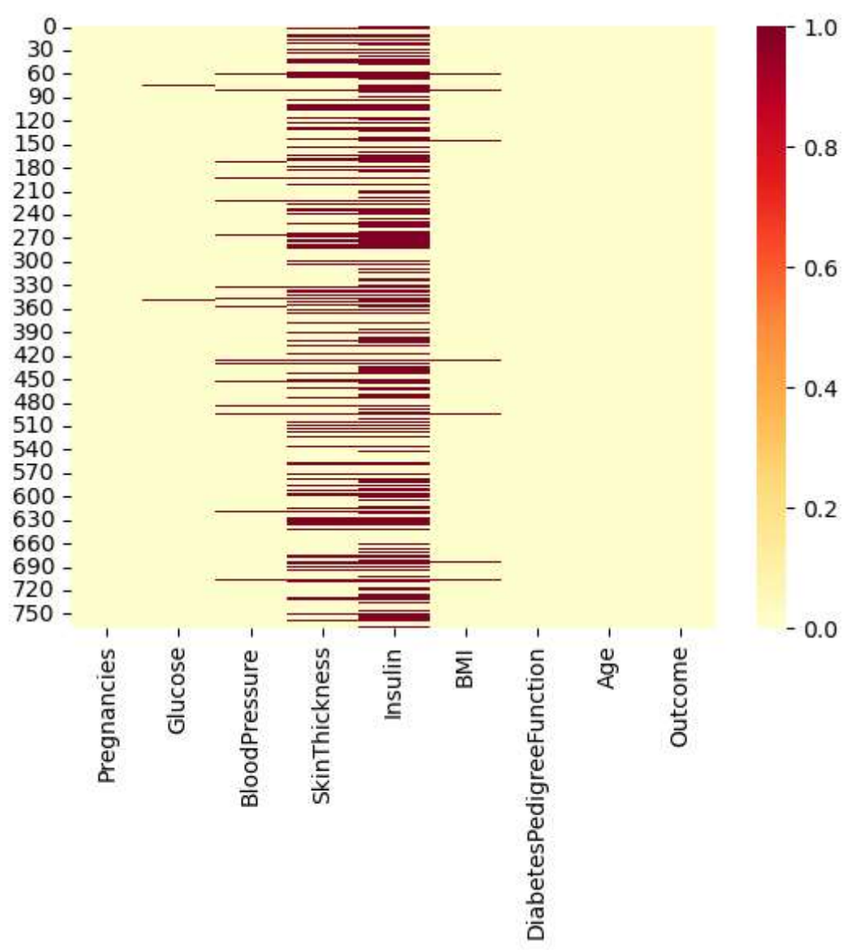
In [11]:
```
#Create a copy of DataFrame to avoid modifying the original data
data_c = data.copy(deep=True)
#Replace Zeroes with Nan in specific columns
cols_to_repl_zeroes =['Glucose','BloodPressure','SkinThickness','Insulin','BMI']
data_c[cols_to_repl_zeroes] = data_c[cols_to_repl_zeroes].replace(0, np.nan)
#Check missing values again after reploacement
print(data_c.isnull().sum())
```

```
Pregnancies                   0
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
Outcome                       0
dtype: int64
```

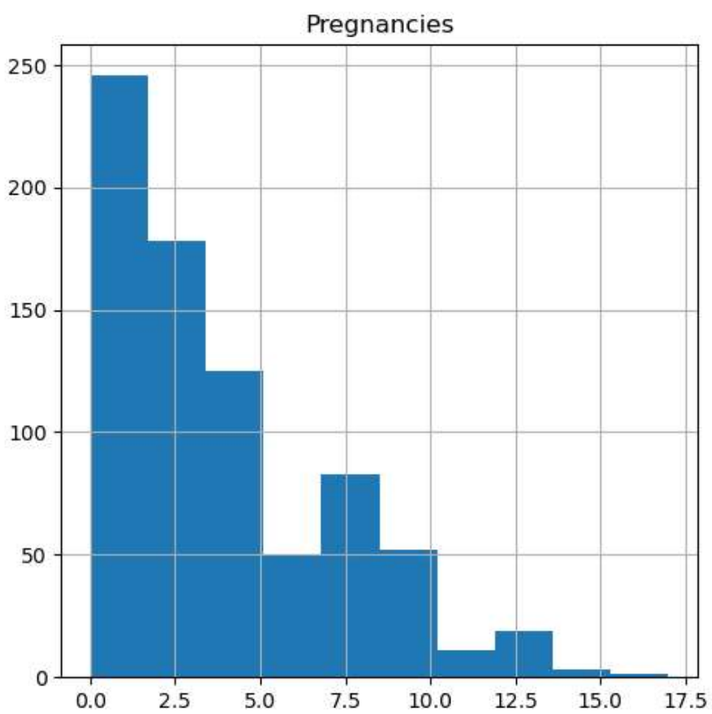
In [12]:
```
sns.heatmap(data.isnull(), cmap="Greens")
# Display the heatmap
plt.show()
```

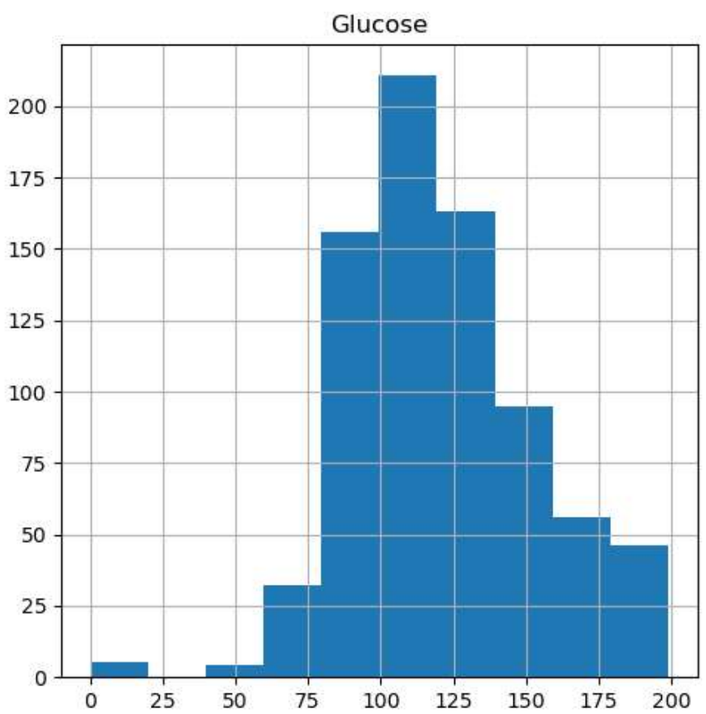


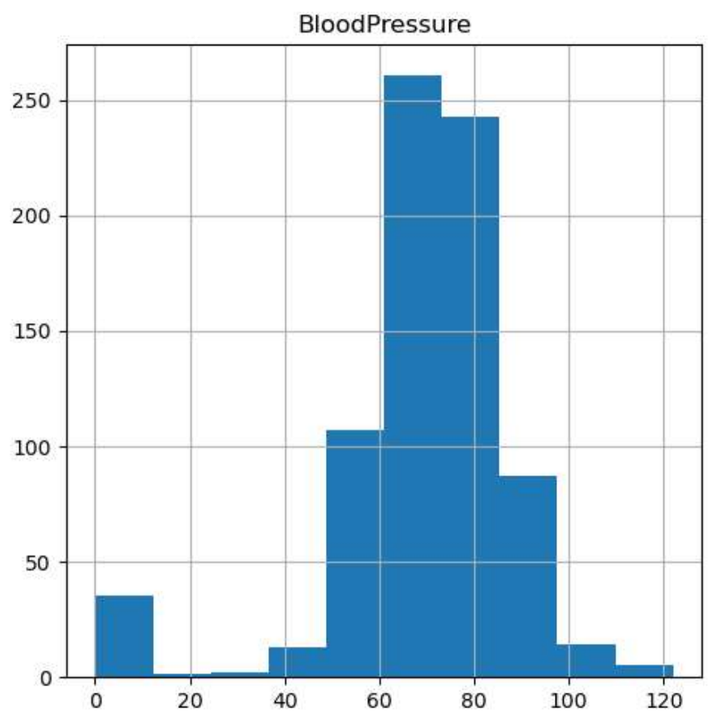
In [13]:
```
#HeatMap after Replacement
sns.heatmap(data_c.isnull(), cmap="YlOrRd")  # Example using "YlOrRd"
plt.show()
```

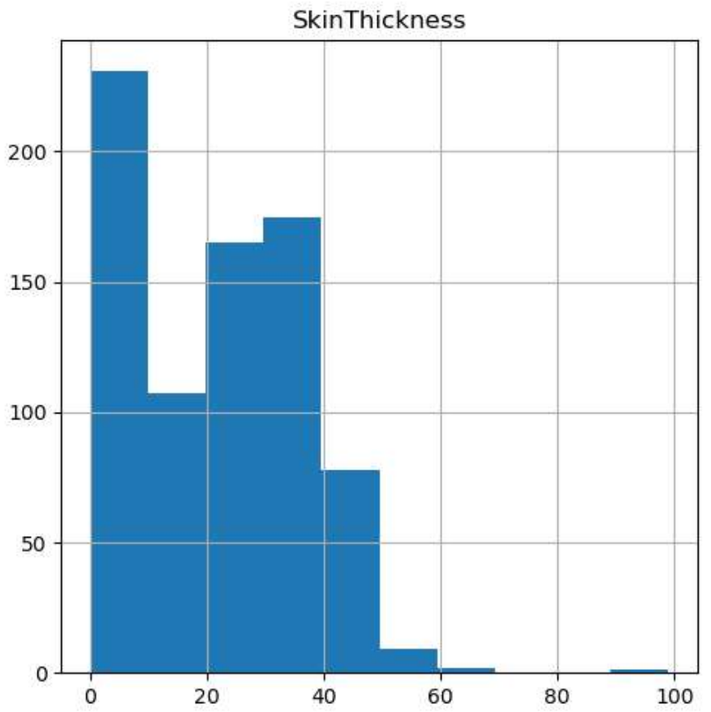


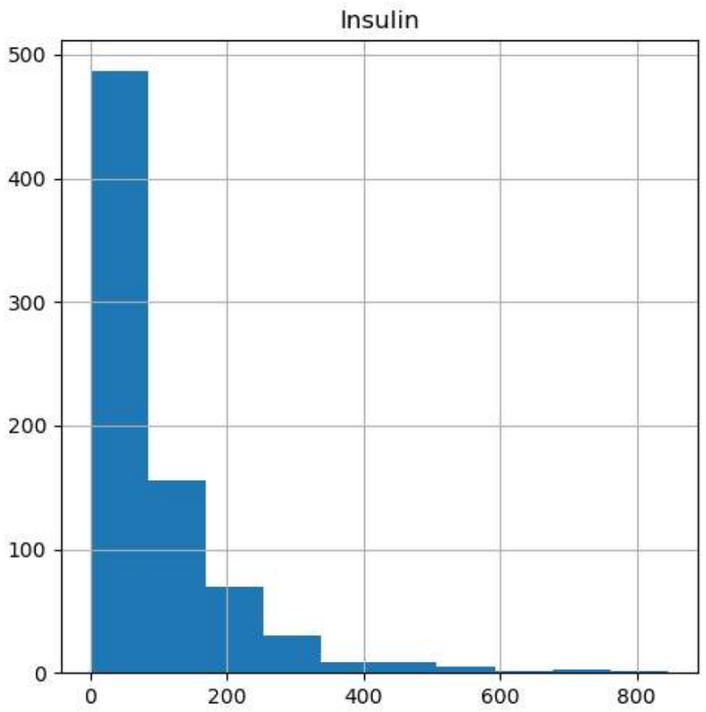
In [14]:
```
#Visualization the histograms before Replacement
data.hist(figsize=(20,20))
plt.show()
```

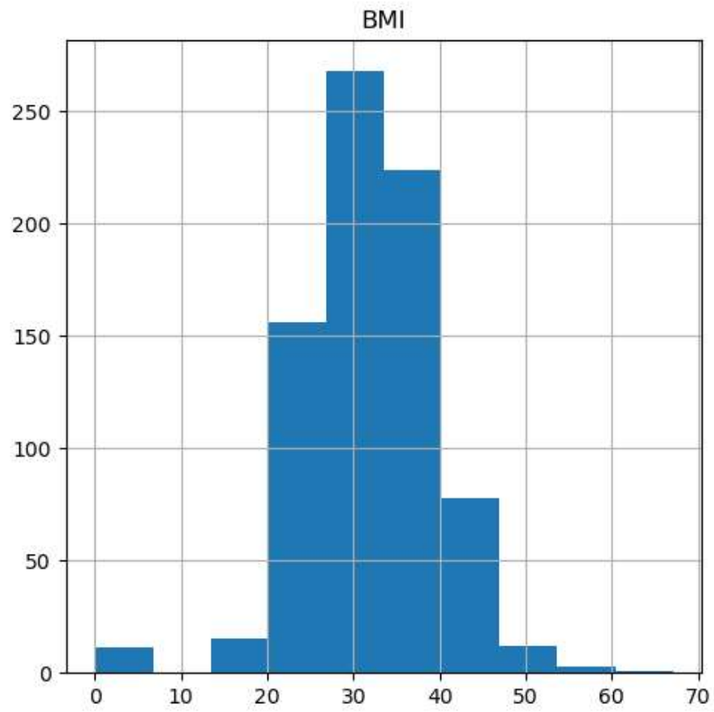

Pregnancies
250
200
150
100
50
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


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

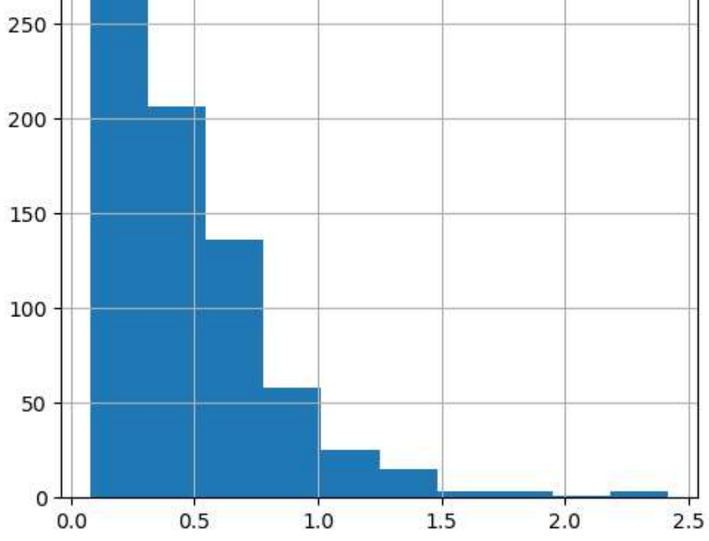

Insulin
500
400
300
200
100
0
0
200
400
600
800

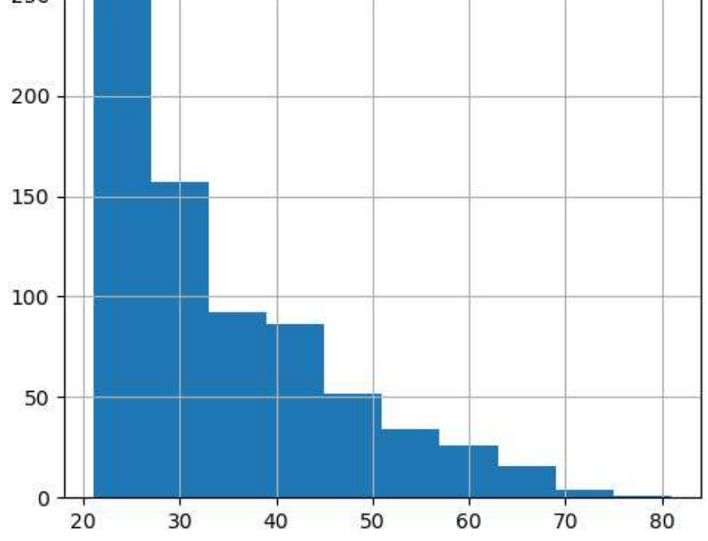

BMI
250
200
150
100
50
0
0
10
20
30
40
50
60
70

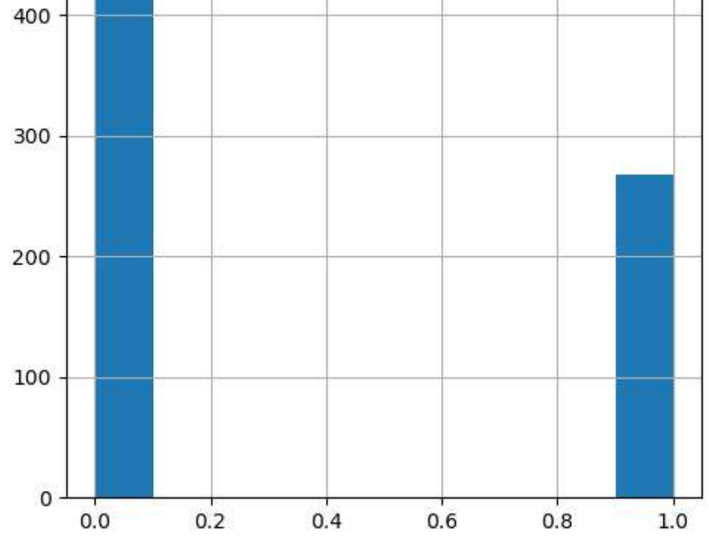
DiabetesPedigreeFunction
300
0

Age
300
250

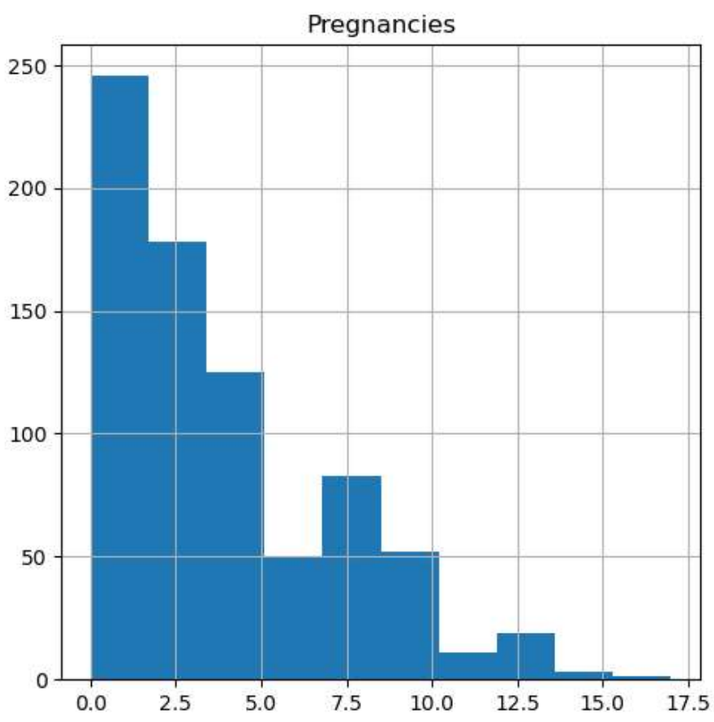
Outcome
500

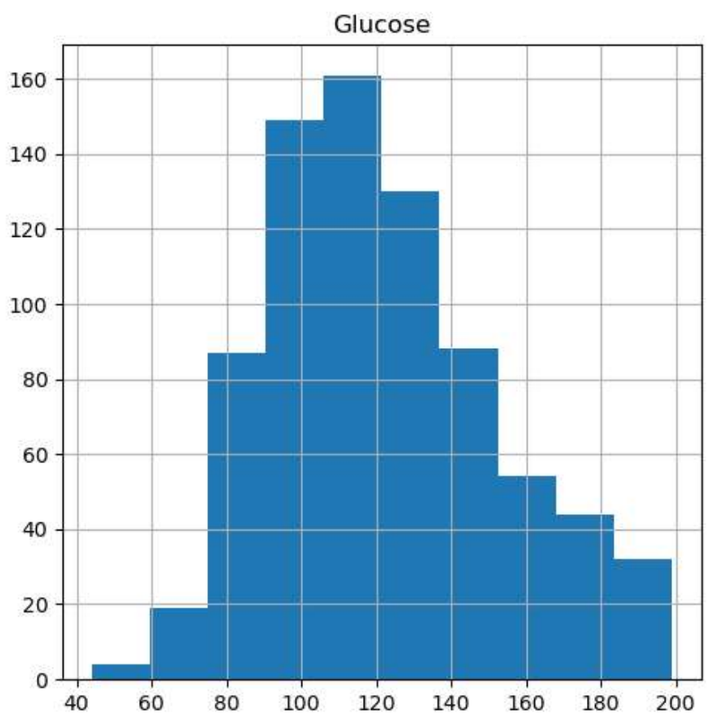


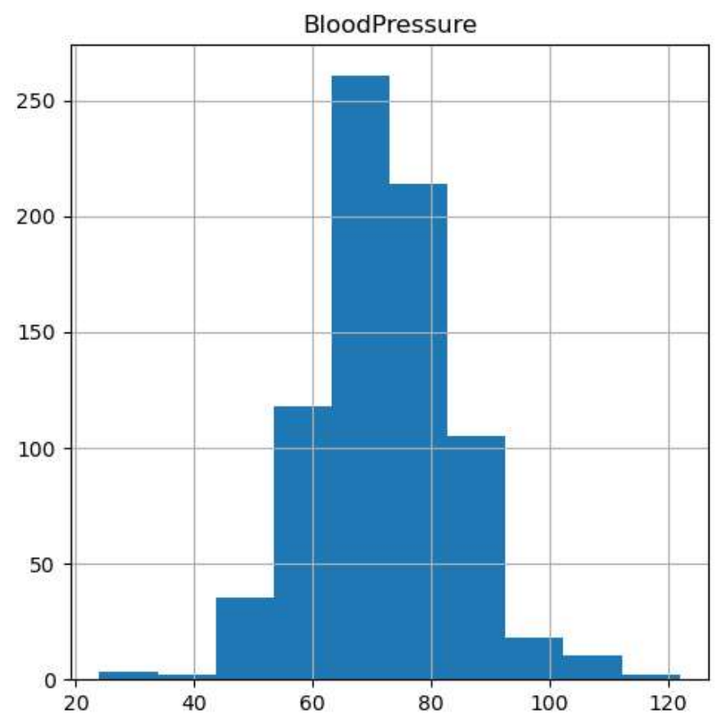


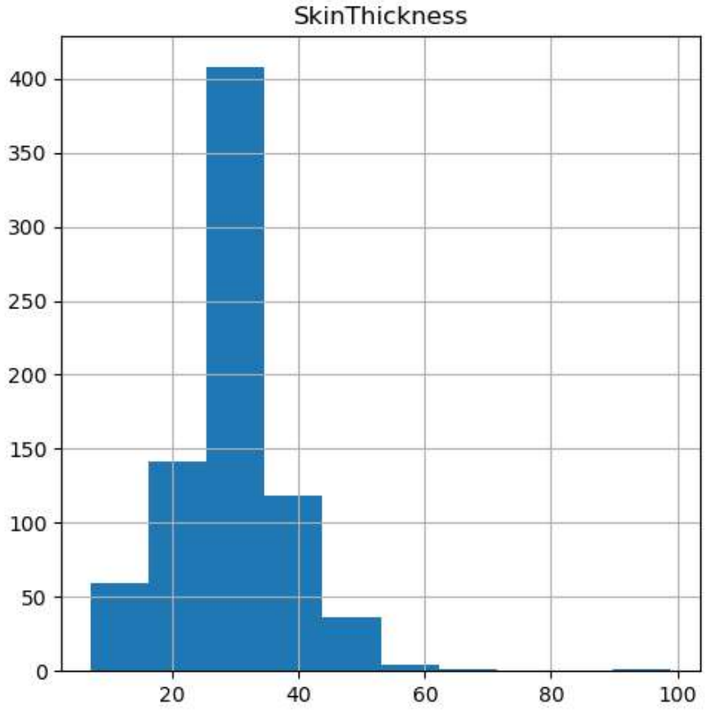


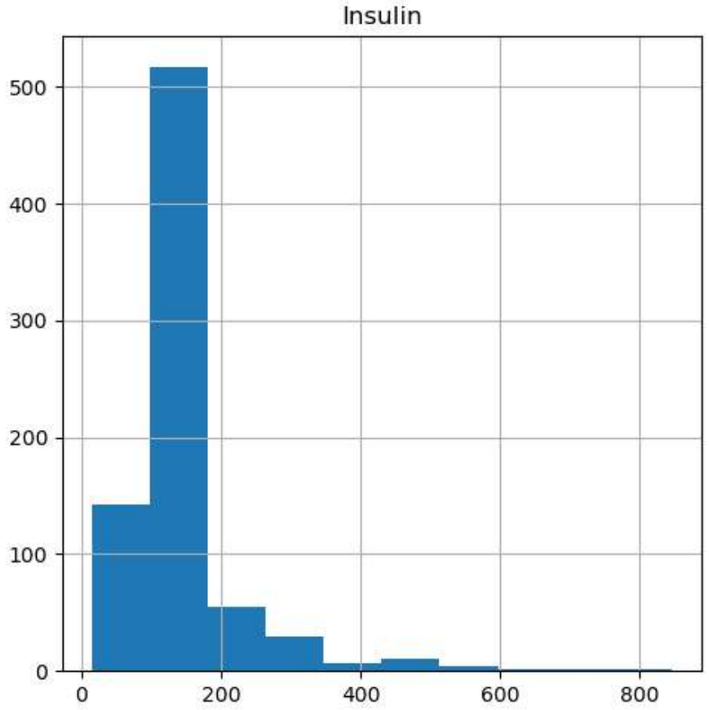
In [15]:
```
#Replace NaN Values with mean or median of the corresponding columns
data_c['Glucose'].fillna(data_c['Glucose'].mean(),inplace=True)
data_c['BloodPressure'].fillna(data_c['BloodPressure'].mean(),inplace=True)
data_c['SkinThickness'].fillna(data_c['SkinThickness'].median(),inplace=True)
data_c['Insulin'].fillna(data_c['Insulin'].median(),inplace=True)
data_c['BMI'].fillna(data_c['BMI'].median(),inplace=True)
#Visualization the histograms after Replacement
p=data_c.hist(figsize=(20,20))
```

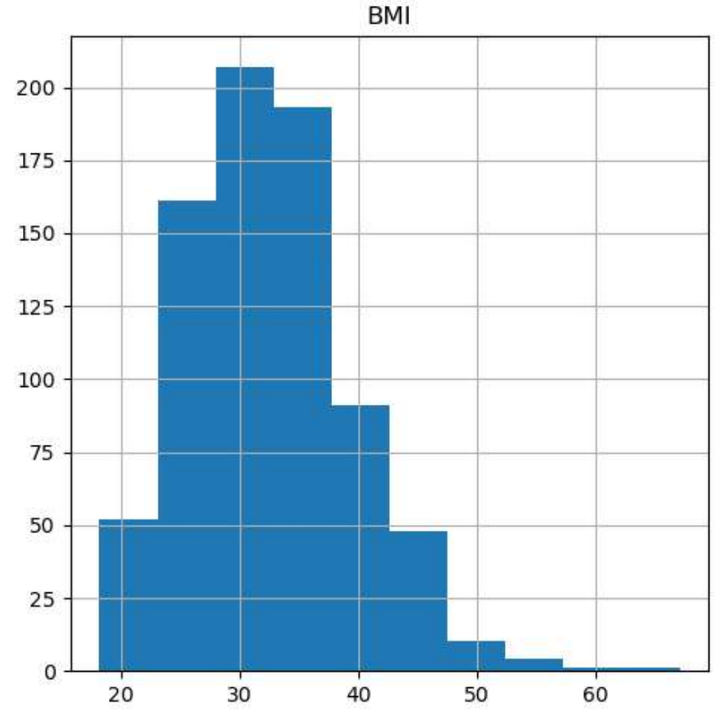

Pregnancies
0
50
100
150
200
250
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Glucose
0
20
40
60
80
100
120
140
160
40
60
80
100
120
140
160
180
200


BloodPressure
0
50
100
150
200
250
20
40
60
80
100
120


SkinThickness
0
50
100
150
200
250
300
350
400
20
40
60
80
100

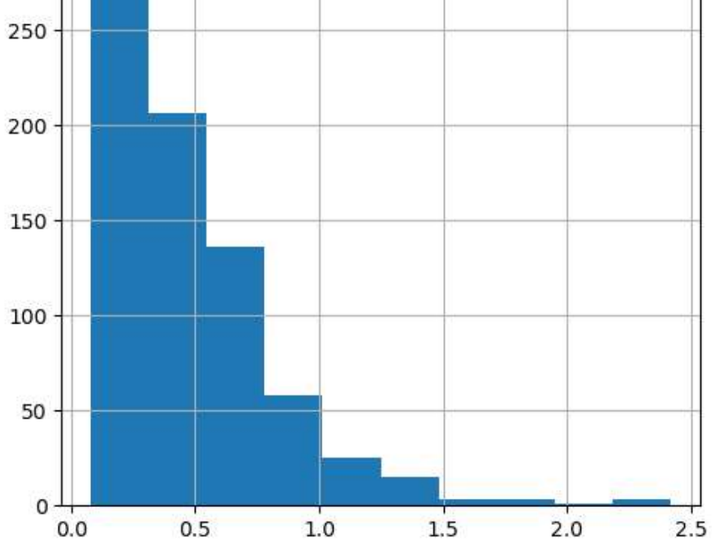

Insulin
0
100
200
300
400
500
0
200
400
600
800

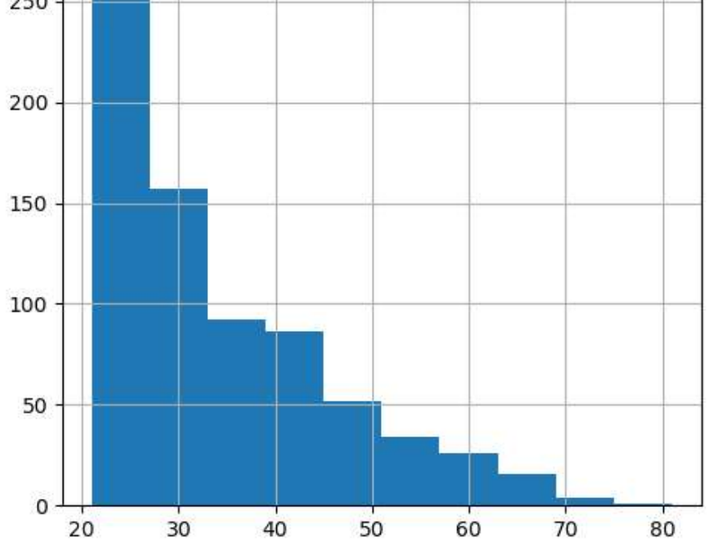

BMI
0
25
50
75
100
125
150
175
200
20
30
40
50
60

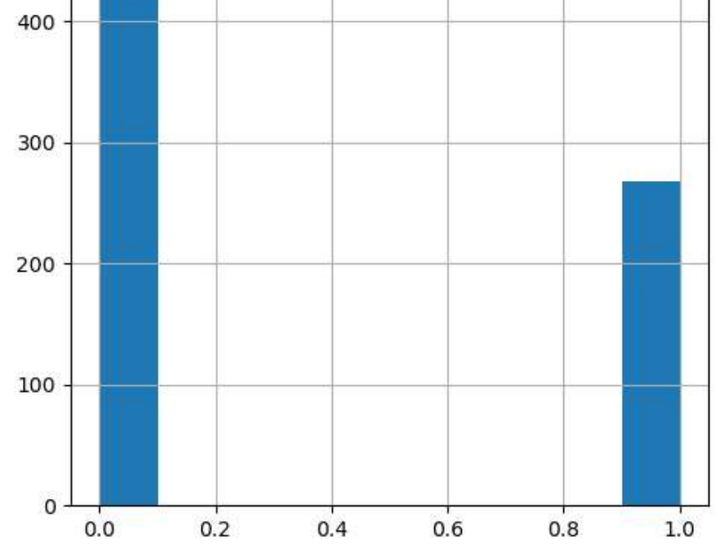
DiabetesPedigreeFunction
300

Age
300

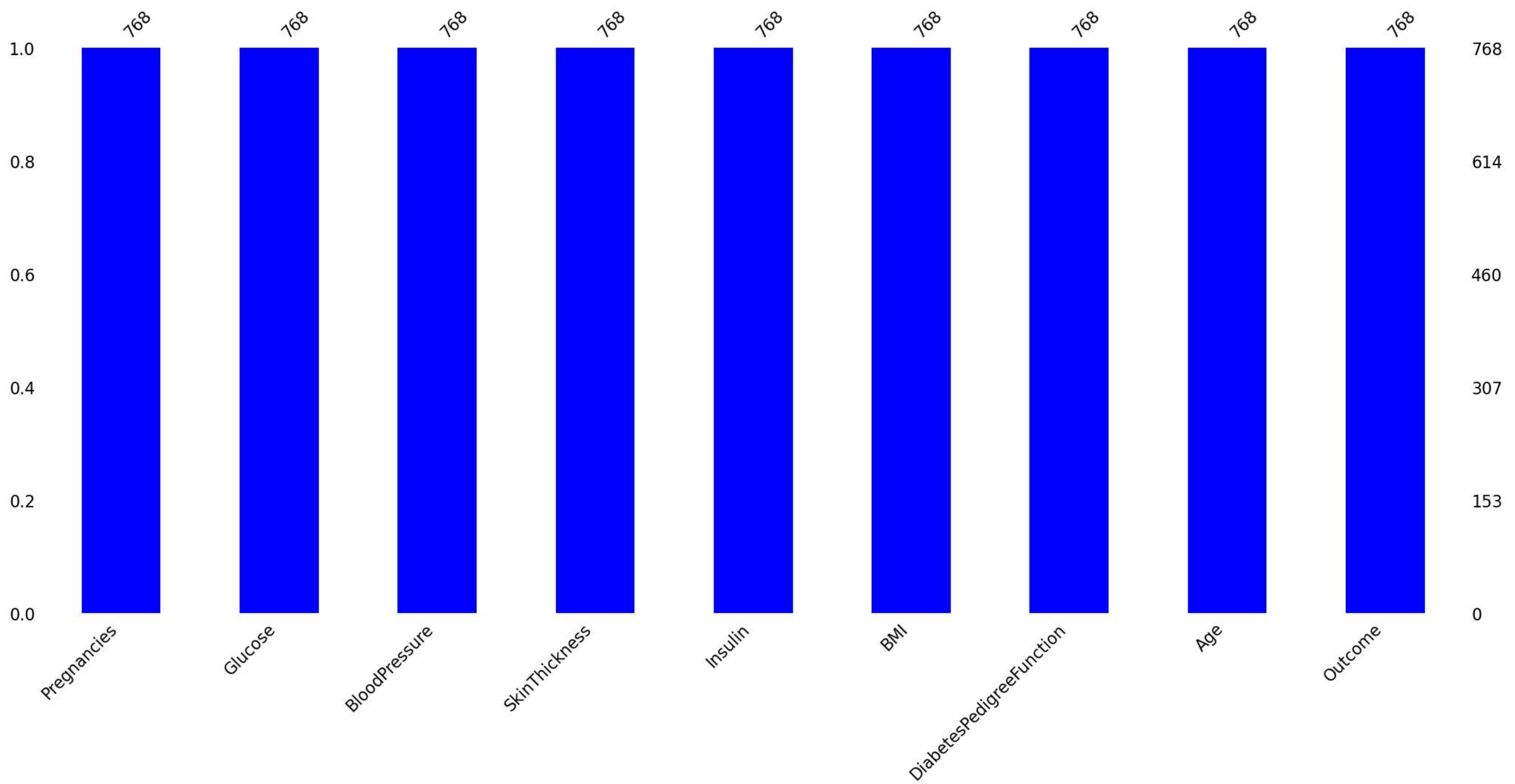
Outcome
500



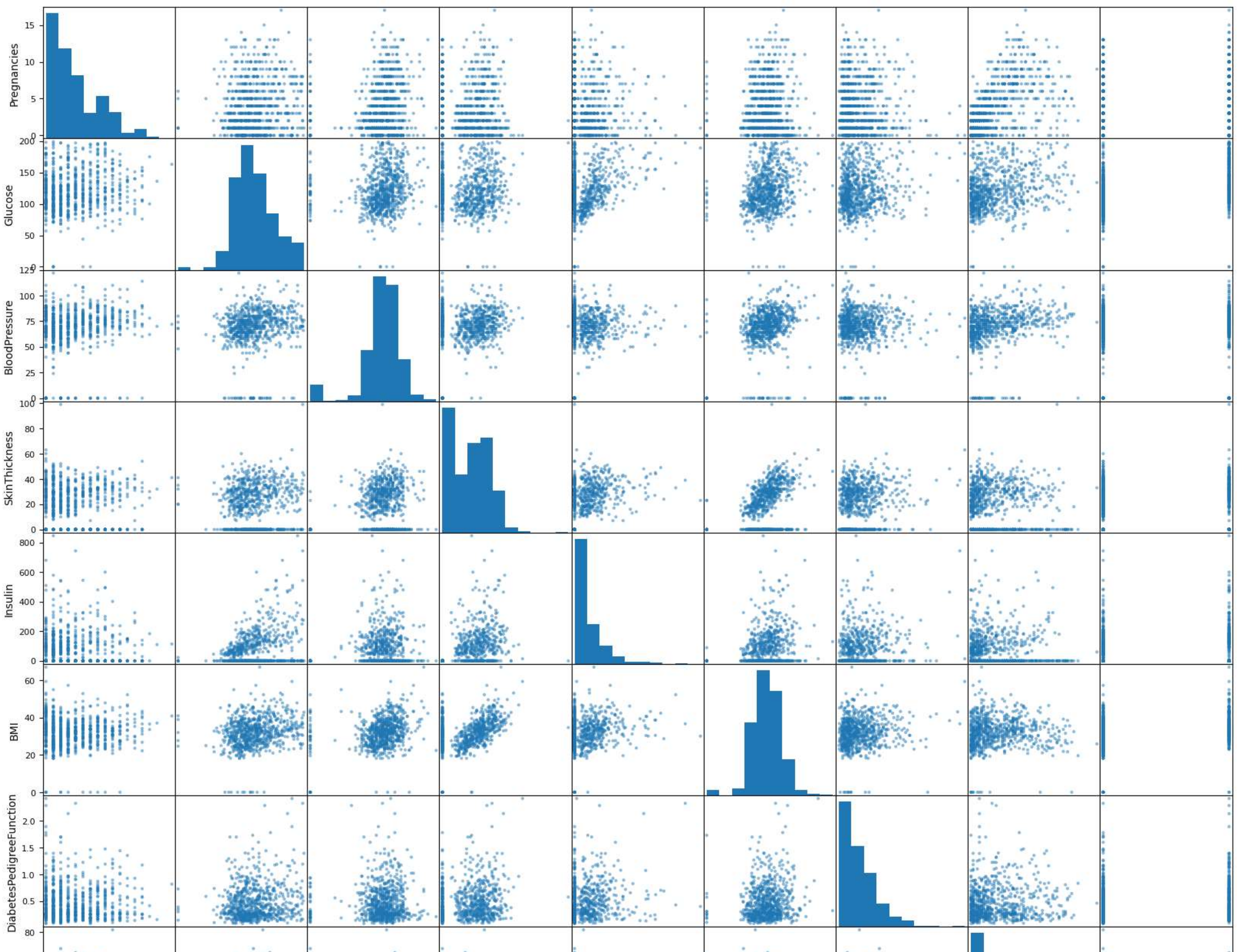


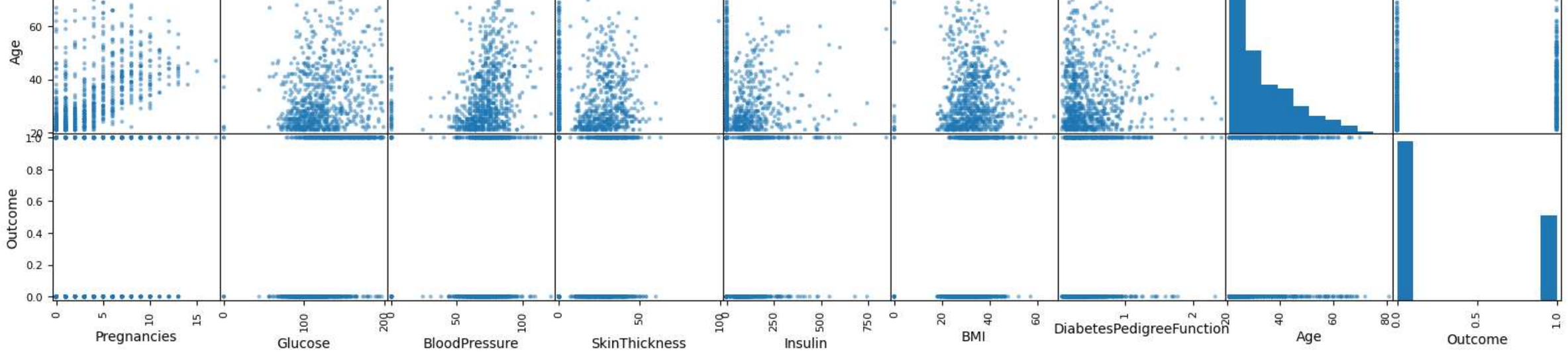


In [16]:
```
#plotting null count analysis plot
p = msno.bar(data_c, color="blue")
# Optional: Set background color to white for better contrast
plt.gca().set_facecolor("white")
# Display the chart
plt.show()
```



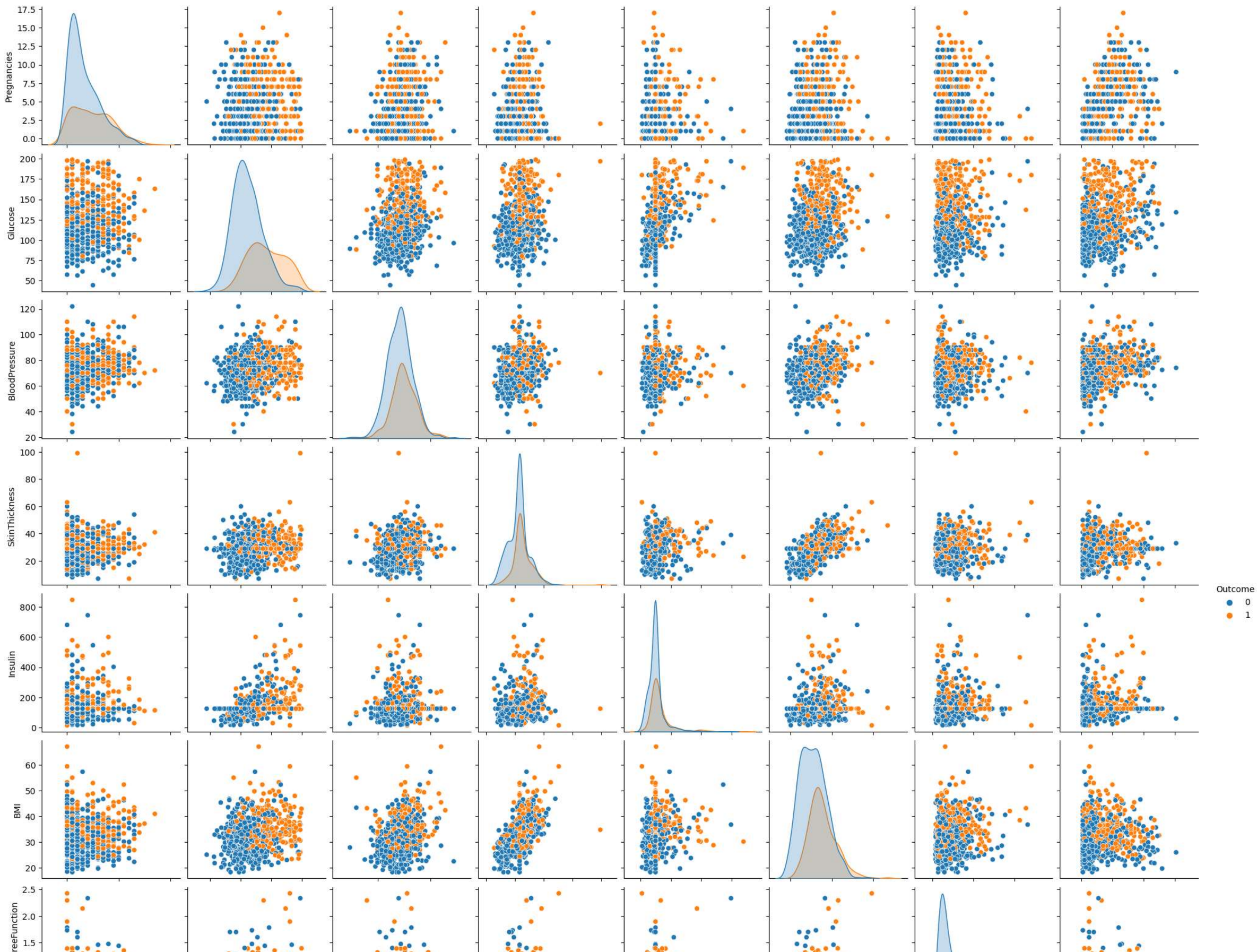
In [17]:
```
#Plot scatter matrix of uncleand data
P= scatter_matrix(data,figsize=(20,20))
```

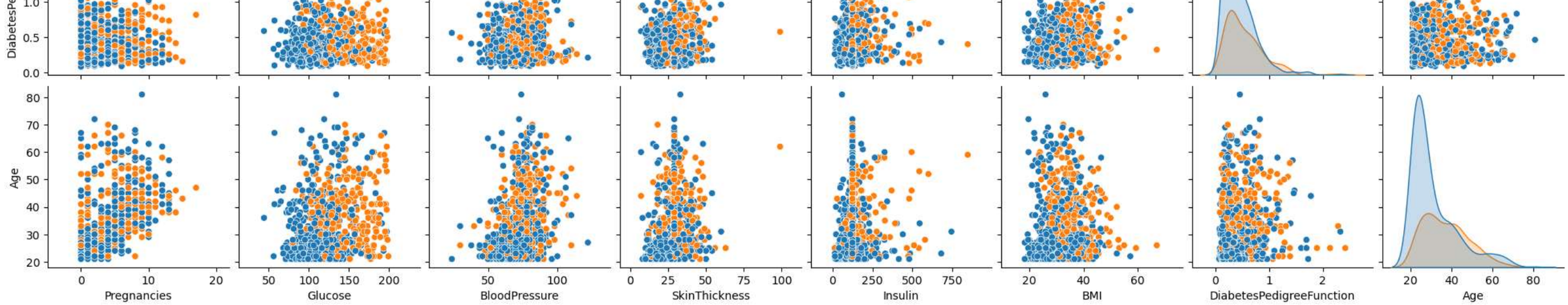

Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction



In [18]:
```
#plotting pair plots for the data
#plotting pair plots for the data
sns.pairplot(data_c,hue='Outcome')
```

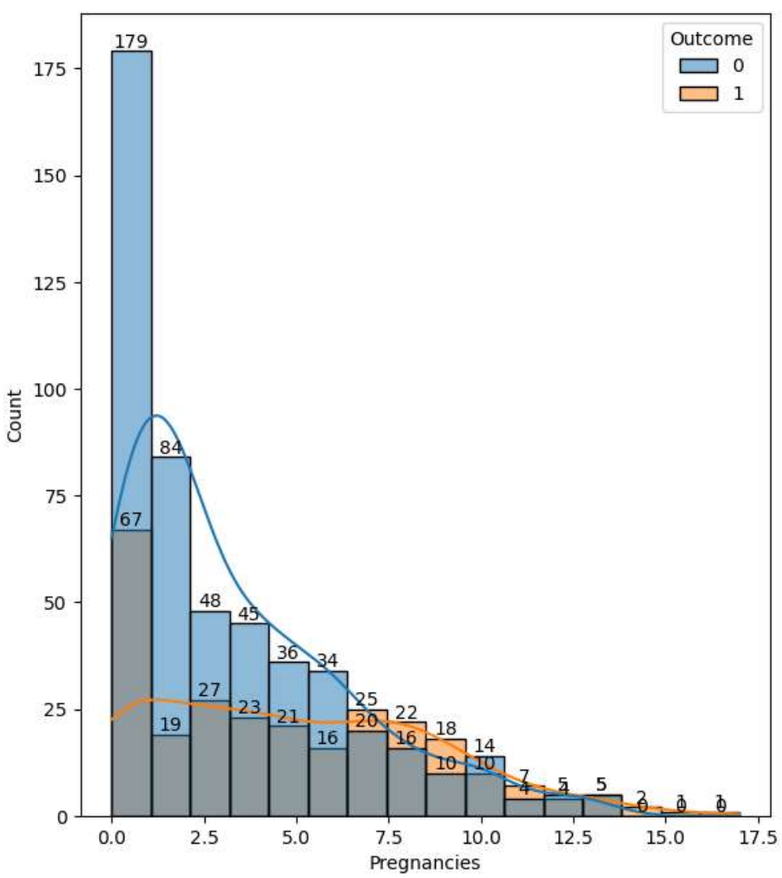
Out[18]:
```
<seaborn.axisgrid.PairGrid at 0x1e6e7745ab0>
```

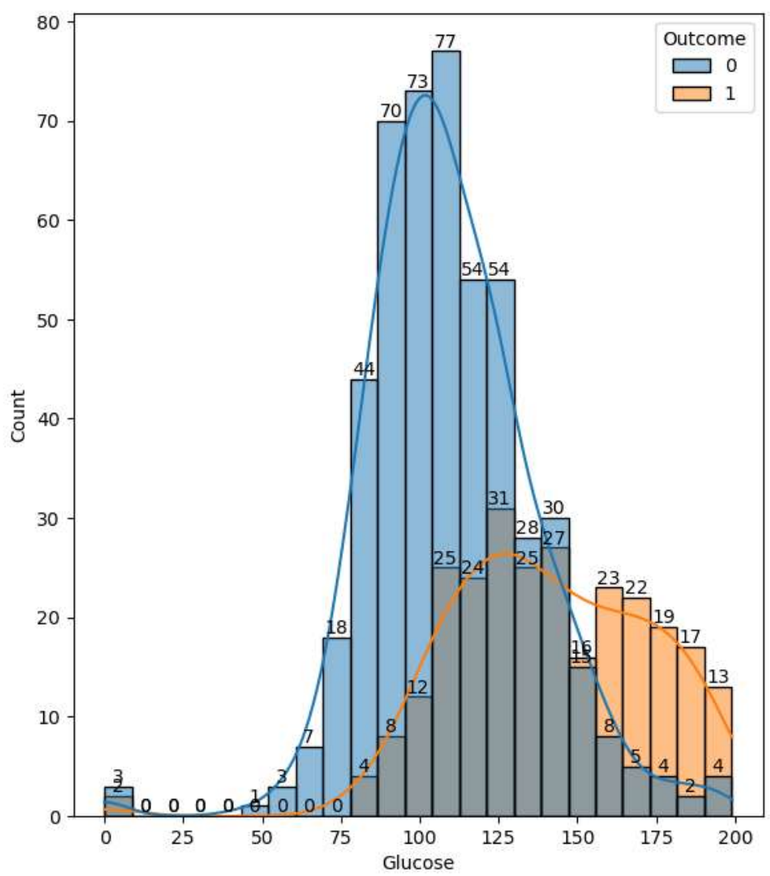

Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Outcome
0
1

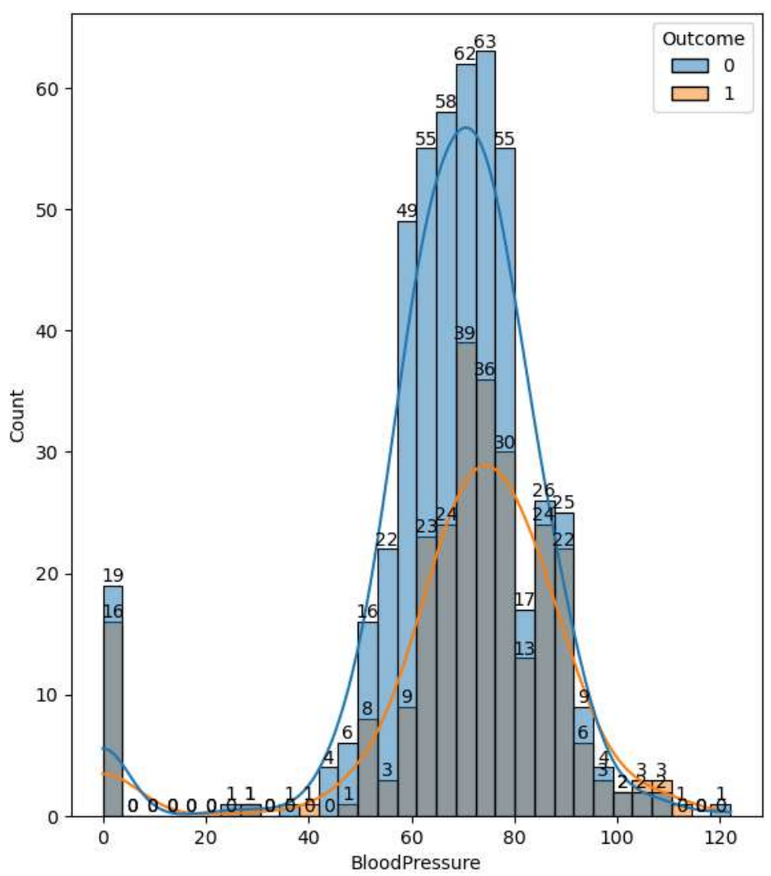


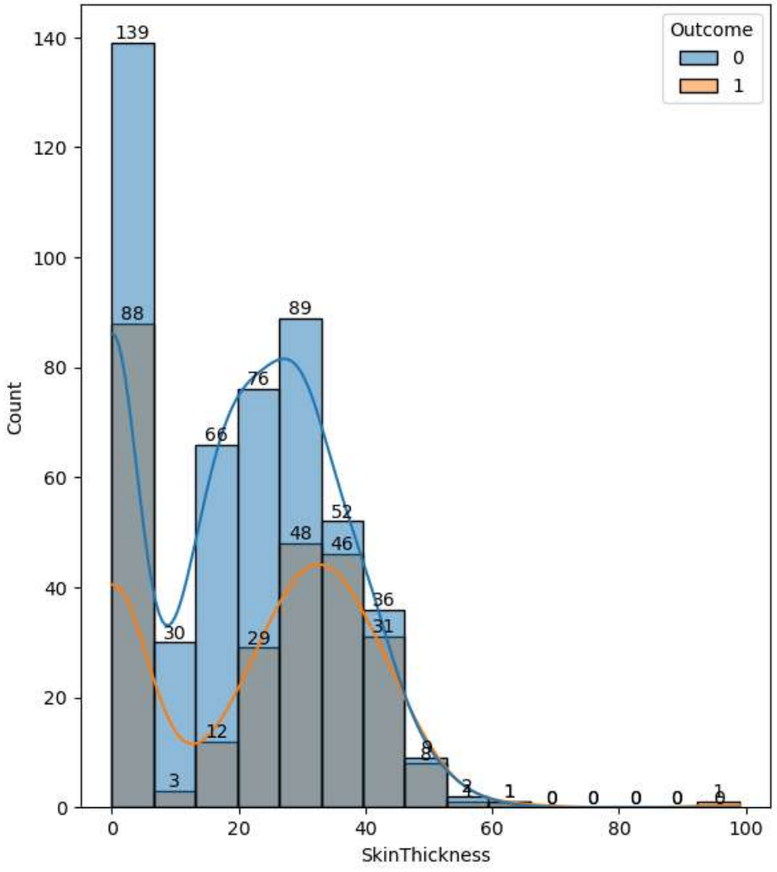
In [19]:

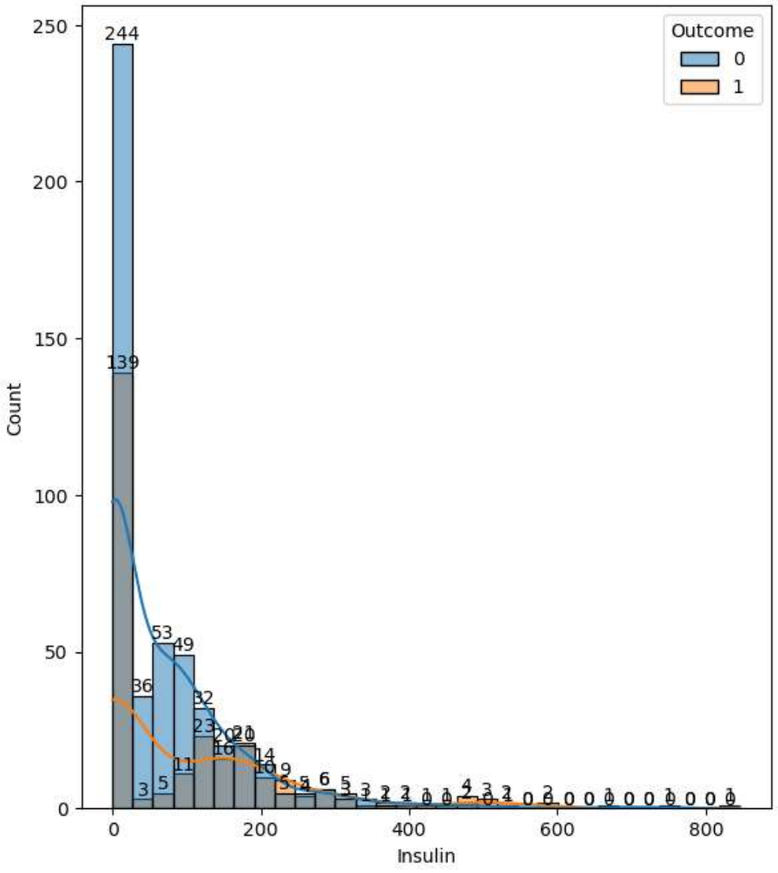
```
#Histplot of Dataset Variables relation with Outcome
pno = 1
plt.figure(figsize=(18, 20))

for i in data.columns:
    if pno < 9:
        plt.subplot(3, 3, pno)
        ax = sns.histplot(data=data, x=i, hue="Outcome", kde=True)  # Adjust hue to match your column name
        plt.xlabel(i)
        pno += 1

        # Add bar labels for readability
        for container in ax.containers:
            ax.bar_label(container)

plt.tight_layout()  # Adjust spacing for better visibility
plt.show()
```

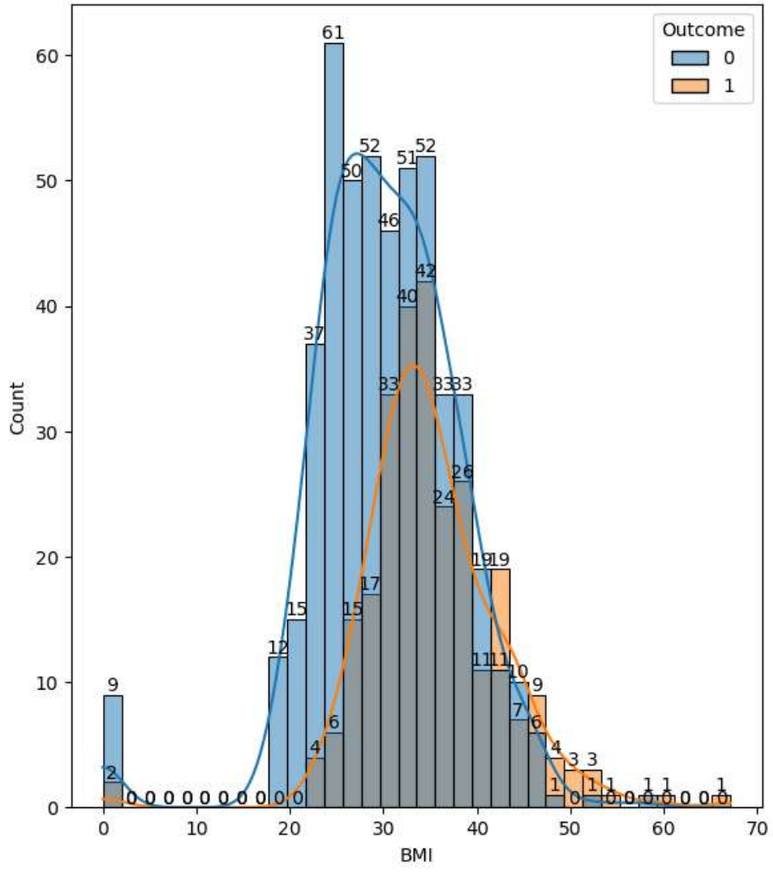

Outcome
0
1
Count
Pregnancies


Outcome
0
1
Count
Glucose


Outcome
0
1
Count
BloodPressure

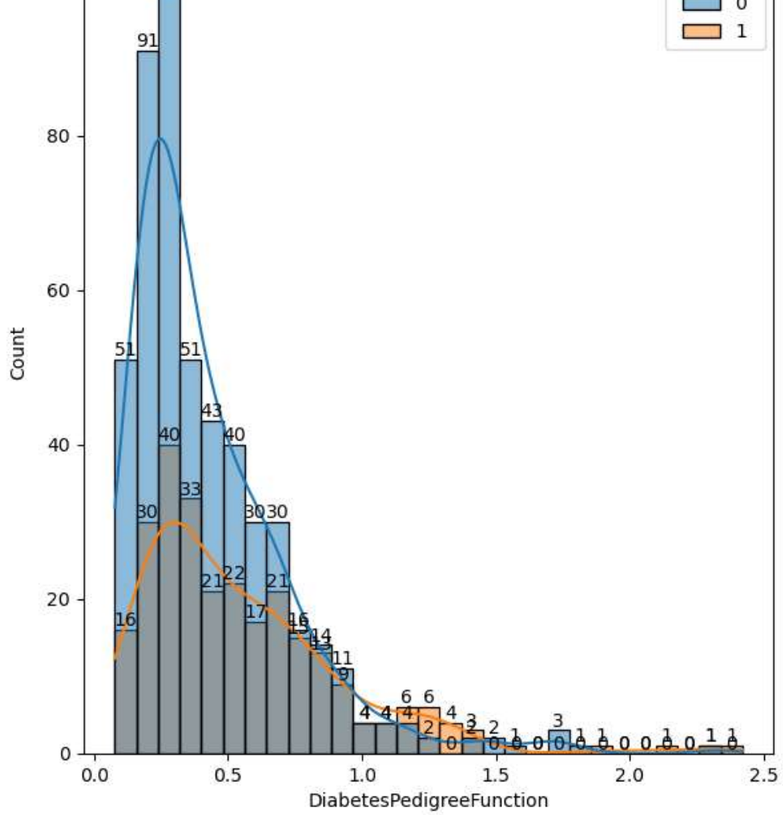

Outcome
0
1
Count
SkinThickness

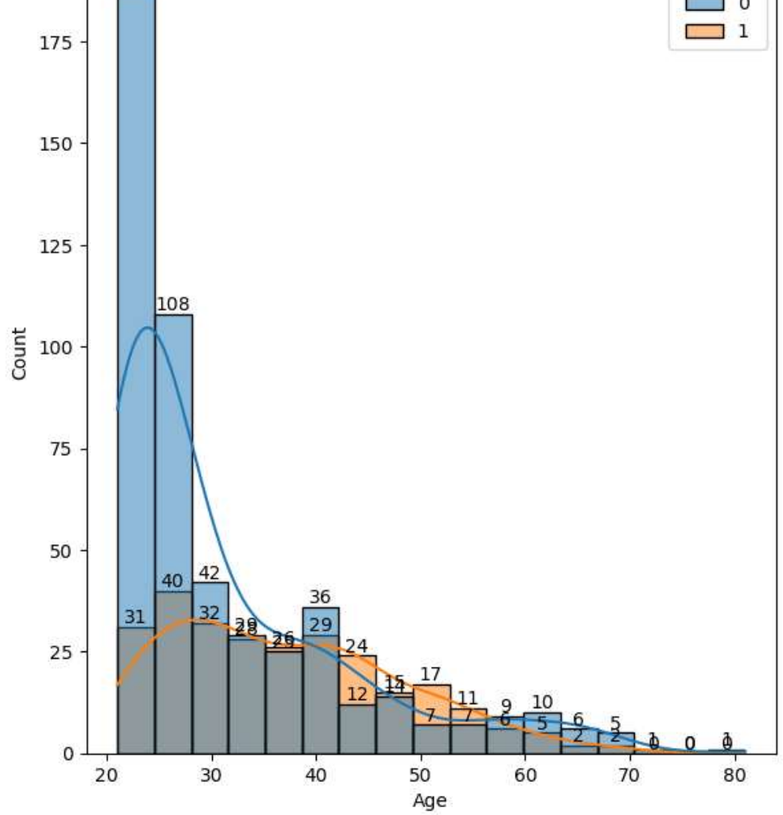

Outcome
0
1
Count
Insulin


Outcome
0
1
Count
BMI

Outcome
0
1
Count
DiabetesPedigreeFunction

Outcome
0
1
Count
Age

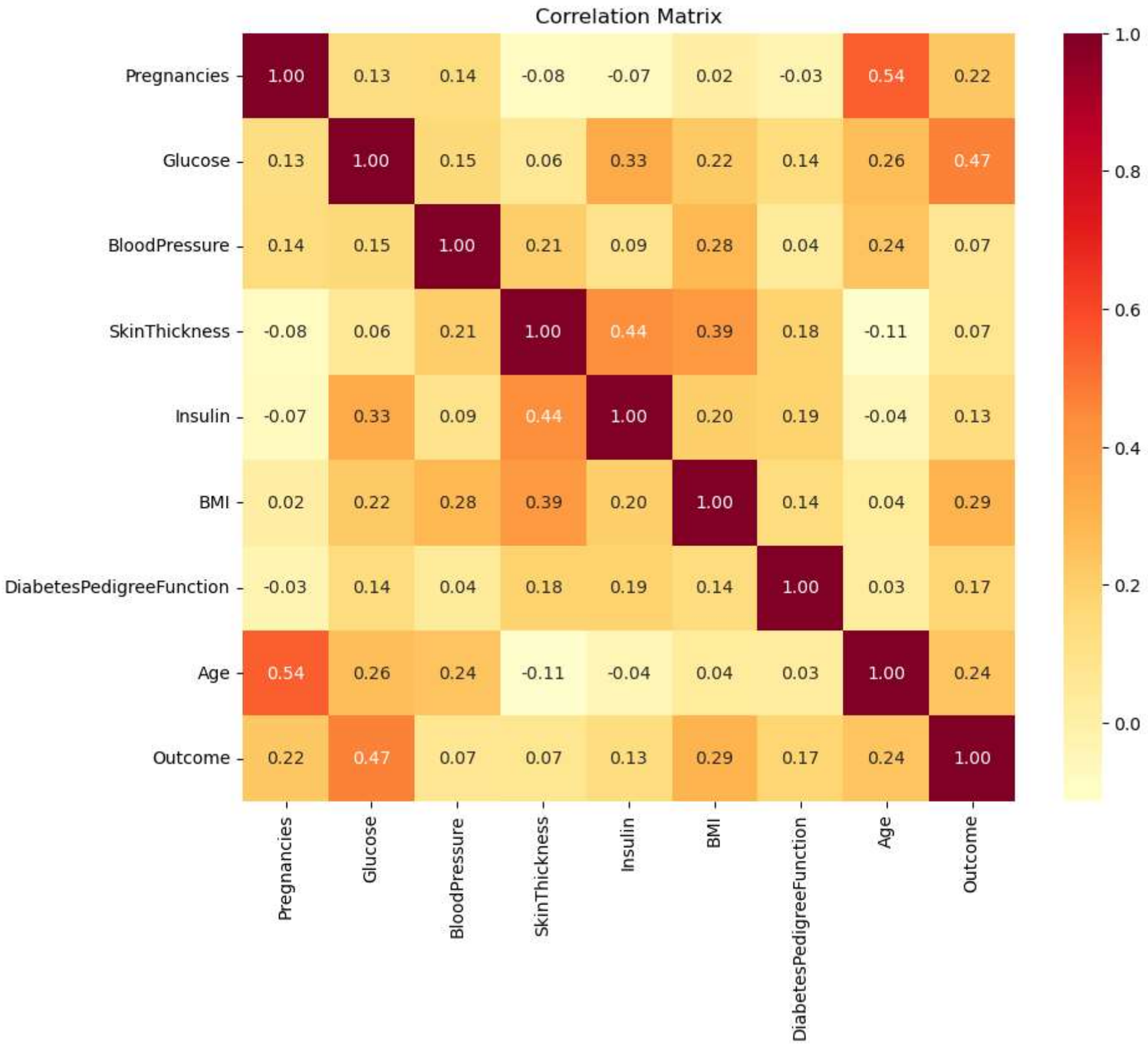




In [20]:
```
correlation = data.corr()
print(correlation)
```

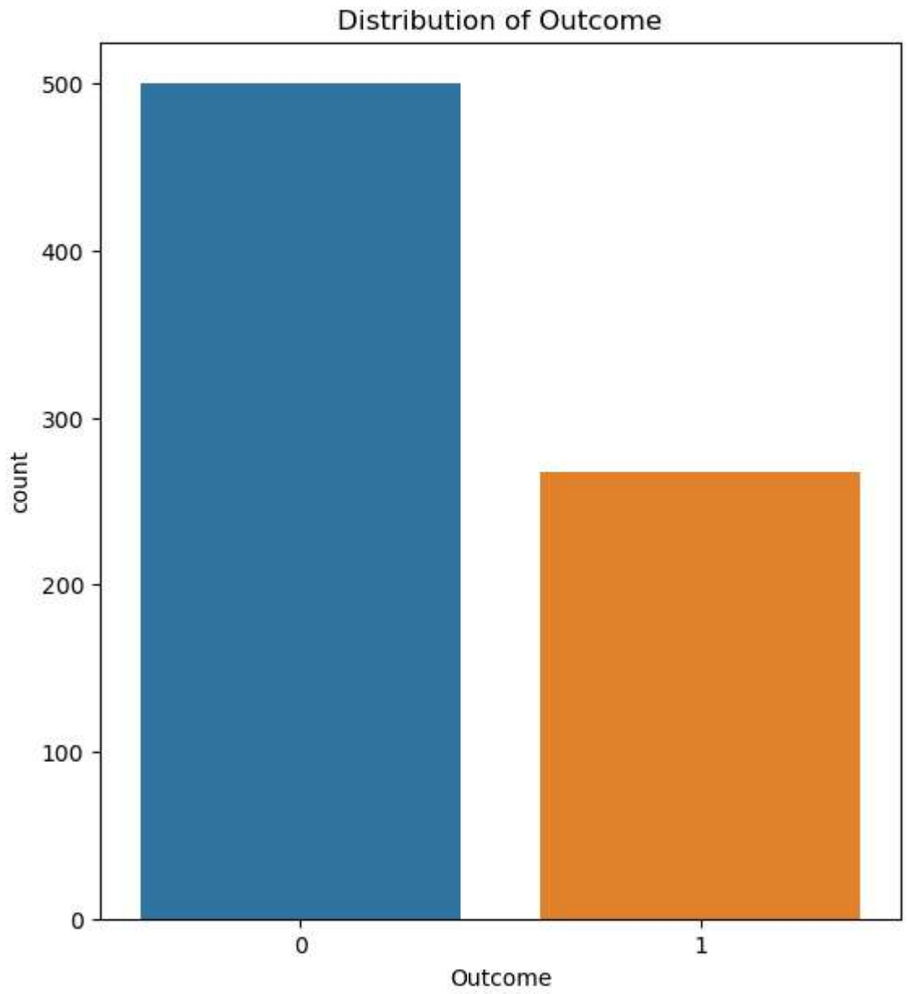
```
                          Pregnancies   Glucose  BloodPressure  SkinThickness  \
Pregnancies                  1.000000  0.129459       0.141282      -0.081672
Glucose                      0.129459  1.000000       0.152590       0.057328
BloodPressure                0.141282  0.152590       1.000000       0.207371
SkinThickness               -0.081672  0.057328       0.207371       1.000000
Insulin                     -0.073535  0.331357       0.088933       0.436783
BMI                          0.017683  0.221071       0.281805       0.392573
DiabetesPedigreeFunction    -0.033523  0.137337       0.041265       0.183928
Age                          0.544341  0.263514       0.239528      -0.113970
Outcome                      0.221898  0.466581       0.065068       0.074752

                           Insulin       BMI  DiabetesPedigreeFunction  \
Pregnancies              -0.073535  0.017683                 -0.033523
Glucose                   0.331357  0.221071                  0.137337
BloodPressure             0.088933  0.281805                  0.041265
SkinThickness             0.436783  0.392573                  0.183928
Insulin                   1.000000  0.197859                  0.185071
BMI                       0.197859  1.000000                  0.140647
DiabetesPedigreeFunction  0.185071  0.140647                  1.000000
Age                      -0.042163  0.036242                  0.033561
Outcome                   0.130548  0.292695                  0.173844

                               Age   Outcome
Pregnancies               0.544341  0.221898
Glucose                   0.263514  0.466581
BloodPressure             0.239528  0.065068
SkinThickness            -0.113970  0.074752
Insulin                  -0.042163  0.130548
BMI                       0.036242  0.292695
DiabetesPedigreeFunction  0.033561  0.173844
Age                       1.000000  0.238356
Outcome                   0.238356  1.000000
```

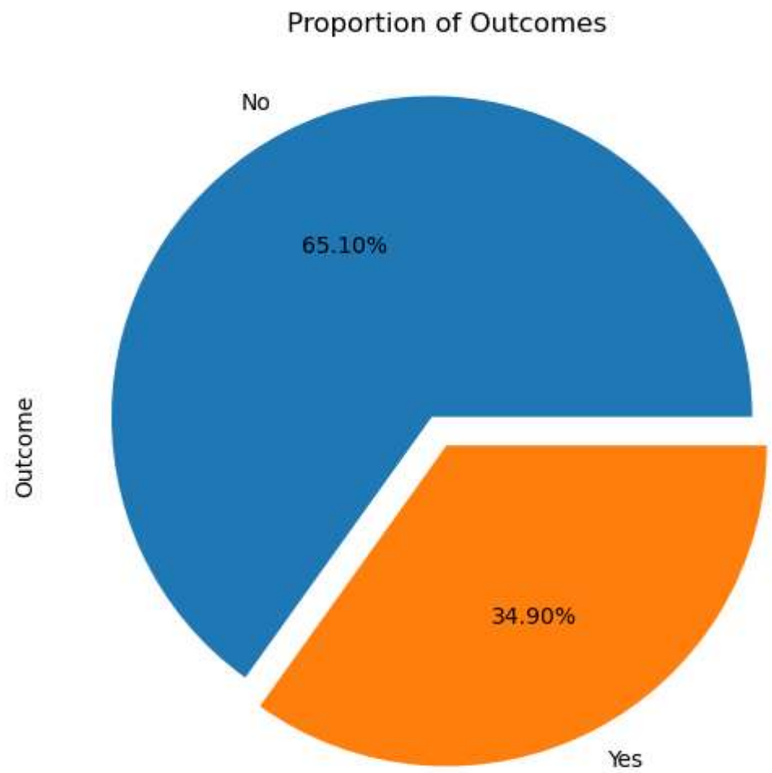
In [21]:
```
plt.figure(figsize=(10, 8))
sns.heatmap(correlation, annot=True, cmap="YlOrRd",fmt=".2f")
plt.title('Correlation Matrix')

# Display the heatmap
plt.show()
```



In [22]:
```
ig, ax = plt.subplots(1, 2, figsize=(14, 7))

# Countplot for "Outcome" distribution
sns.countplot(data=data, x="Outcome", ax=ax[0])
ax[0].set_title("Distribution of Outcome")

# Pie chart for "Outcome" proportions
data["Outcome"].value_counts().plot.pie(
```

```
    ax=ax[1],
    explode=[0.1, 0],  # Adjust for better visualization
    autopct="%1.2F%%",
    labels=["No", "Yes"],  # Replace with actual outcome labels
)
ax[1].set_title("Proportion of Outcomes")

plt.show()
```





In [23]:

```
# Assuming you have already loaded your data into a DataFrame called `data`

# Check if "Outcome" is a valid column name
if "Outcome" not in data.columns:
    print("Error: 'Outcome' column not found in the DataFrame.")
    exit()

# Define features (X) and target variable (y)
X = data.drop("Outcome", axis=1)  # Features without the outcome column
y = data["Outcome"]  # Target variable

# Split data into training and testing sets using train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Now you can use X_train, X_test, y_train, and y_test for training your model

# Example: training a Random Forest classifier
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier()
```

```
model.fit(X_train, y_train)

# Make predictions on the test set
predictions = model.predict(X_test)

# Evaluate the model's performance on the test set (e.g., accuracy, precision, recall, etc.)
```

In [24]:
```
from sklearn.linear_model import LogisticRegression

# Ensure you have already split your data into X_train, y_train, X_test, and y_test

# Create a LogisticRegression model with max_iter=768
model = LogisticRegression(max_iter=768)

# Fit the model to the training data
model.fit(X_train, y_train)

# Now the model is trained and ready for making predictions on new data
```

Out[24]:
```
LogisticRegression
LogisticRegression(max_iter=768)
```

In [25]:
```
# Ensure you have already trained your model and imported necessary libraries

# Make predictions on the test set
predictions = model.predict(X_test)

# (Optional) Print the predicted labels for comparison
print("Predicted labels:", predictions)
```

```
Predicted labels: [1 0 1 0 0 1 1 0 0 1 0 0 1 0 1 0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 1 0 0 1 0 0 0
 0 0 0 0 0 1 0 1 1 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0
 0 0 1 1 0 1 1 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 1 1 0 0 1 0 0 1 0 0 0
 1 0 0 0 0 1 1 0 0 0 0 0 1 1 0 1 1 1 0 0 1 0 0 1 0 0 0 0 0 1 0 0 0 1 0 1 0
 1 0 0 0 0 0]
```

In [26]:
```
# Calculate accuracy using accuracy_score from sklearn.metrics
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, predictions)

# Print the accuracy with formatting
print(f'Accuracy: {accuracy * 100:.2f}%')

# (Optional) Print the actual labels for comparison
print("Actual labels:", y_test)

# (Optional) Calculate other evaluation metrics as needed
```

```
Accuracy: 72.08%
Actual labels: 230    1
309    1
220    1
704    0
191    0
      ..
509    0
157    0
613    0
18     0
570    0
Name: Outcome, Length: 154, dtype: int64
```

In [27]:
```
# Ensure you have already made predictions and have the actual labels

# Calculate precision, recall, and F1-score
precision = precision_score(y_test, predictions)
recall = recall_score(y_test, predictions)
```

```python
f1 = f1_score(y_test, predictions)

# Print the results
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-score: {f1:.4f}")

# (Optional) Calculate macro and weighted versions for multi-class problems
if len(set(y_test)) > 2:
    precision_macro = precision_score(y_test, predictions, average="macro")
    recall_macro = recall_score(y_test, predictions, average="macro")
    f1_macro = f1_score(y_test, predictions, average="macro")

    precision_weighted = precision_score(y_test, predictions, average="weighted")
    recall_weighted = recall_score(y_test, predictions, average="weighted")
    f1_weighted = f1_score(y_test, predictions, average="weighted")

    print(f"\nMacro-averaged precision: {precision_macro:.4f}")
    print(f"Macro-averaged recall: {recall_macro:.4f}")
    print(f"Macro-averaged F1-score: {f1_macro:.4f}")

    print(f"\nWeighted precision: {precision_weighted:.4f}")
    print(f"Weighted recall: {recall_weighted:.4f}")
    print(f"Weighted F1-score: {f1_weighted:.4f}")
```

```
Precision: 0.6739
Recall: 0.5254
F1-score: 0.5905
```

In [ ]: