Supervised Learning - Project

In this Project, we are going to perform a full supervised learning machine learning project on a "Diabetes" dataset. 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 a patient has diabetes, based on certain diagnostic measurements included in the dataset.

[Kaggle Dataset (https://www.kaggle.com/datasets/akshaydattatraykhare/diabetes-dataset)](https://www.kaggle.com/datasets/akshaydattatraykhare/diabetes-dataset)

Part I : EDA - Exploratory Data Analysis

For this task, you are required to conduct an exploratory data analysis on the diabetes dataset. You have the freedom to choose the visualizations you want to use, but your analysis should cover the following tasks mostly:

- Are there any missing values in the dataset?
- How are the predictor variables related to the outcome variable?
- What is the correlation between the predictor variables?
- What is the distribution of each predictor variable?
- Are there any outliers in the predictor variables?
- How are the predictor variables related to each other?
- Is there any interaction effect between the predictor variables?
- What is the average age of the individuals in the dataset?
- What is the average glucose level for individuals with diabetes and without diabetes?
- What is the average BMI for individuals with diabetes and without diabetes?
- How does the distribution of the predictor variables differ for individuals with diabetes and without diabetes?
- Are there any differences in the predictor variables between males and females (if gender information is available)?

In [1]:

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

In [3]:
```
# read csv
df = pd.read_csv("diabetes.csv")
df.head(2)
```

Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
0	6	148	72	35	0	33.6	0.627
1	1	85	66	29	0	26.6	0.351

In [5]:
```
# check missing values
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
```

In [32]:
```
# Handling missing values (replace with mean)
df.fillna(df.mean(), inplace=True)

# Handling outliers (remove values beyond a certain threshold)
df = df[(df['Glucose'] >= 70) & (df['Glucose'] <= 200)]
df = df[(df['Insulin'] >= 10) & (df['Insulin'] <= 600)]

# Check for missing values again
print(df.isnull().sum())
```

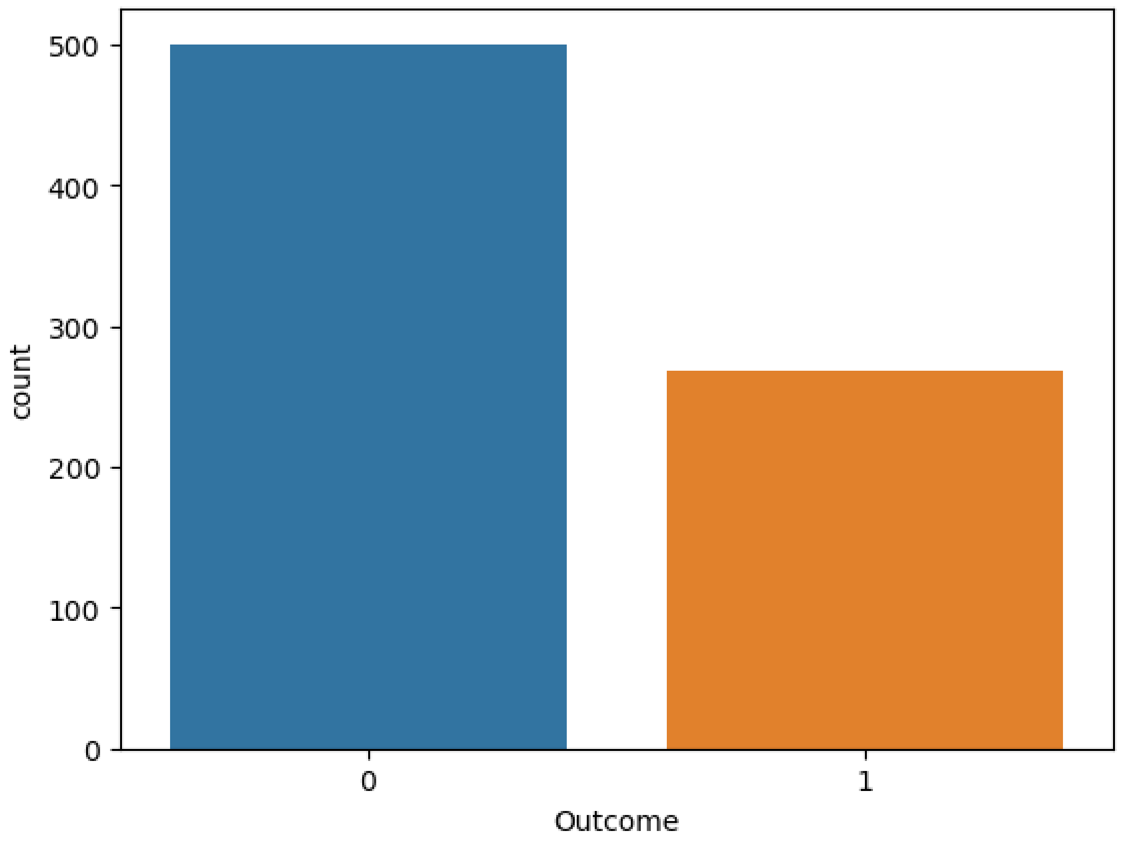
```
Pregnancies                 0.0
Glucose                     0.0
BloodPressure               0.0
SkinThickness               0.0
Insulin                     0.0
BMI                         0.0
DiabetesPedigreeFunction    0.0
Age                         0.0
Interaction                 0.0
Outcome                     0.0
dtype: float64
```

In [7]:

```python
sns.countplot(x='Outcome', data=df)
# Plot relevant visualizations
```

Out[7]:

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

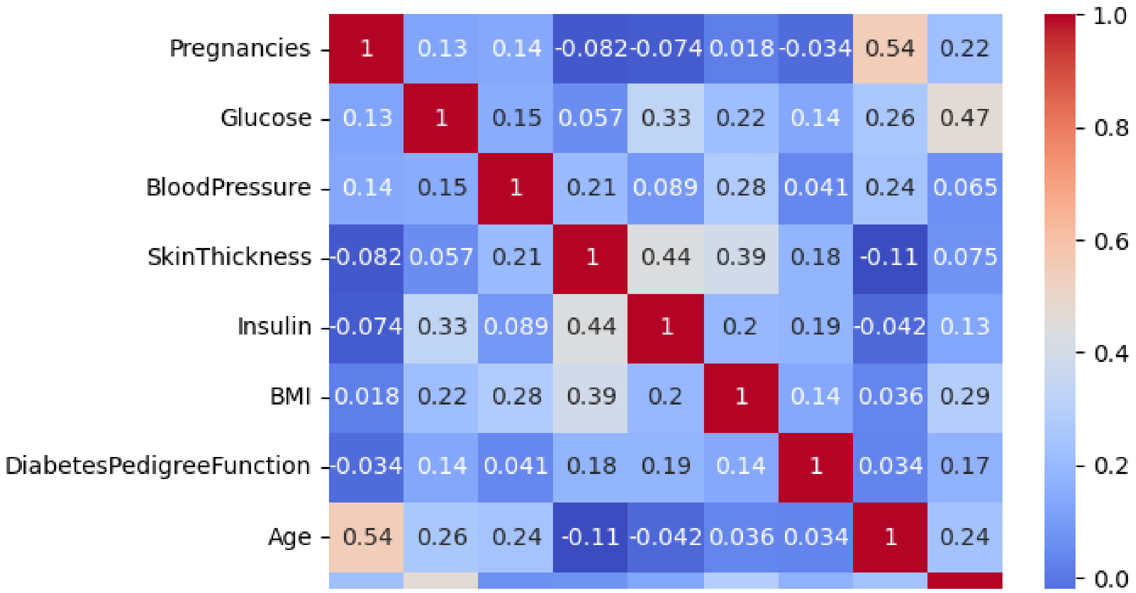


In [8]:

```python
correlation_matrix = df.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
```

Out[8]:

```
<Axes: >
```

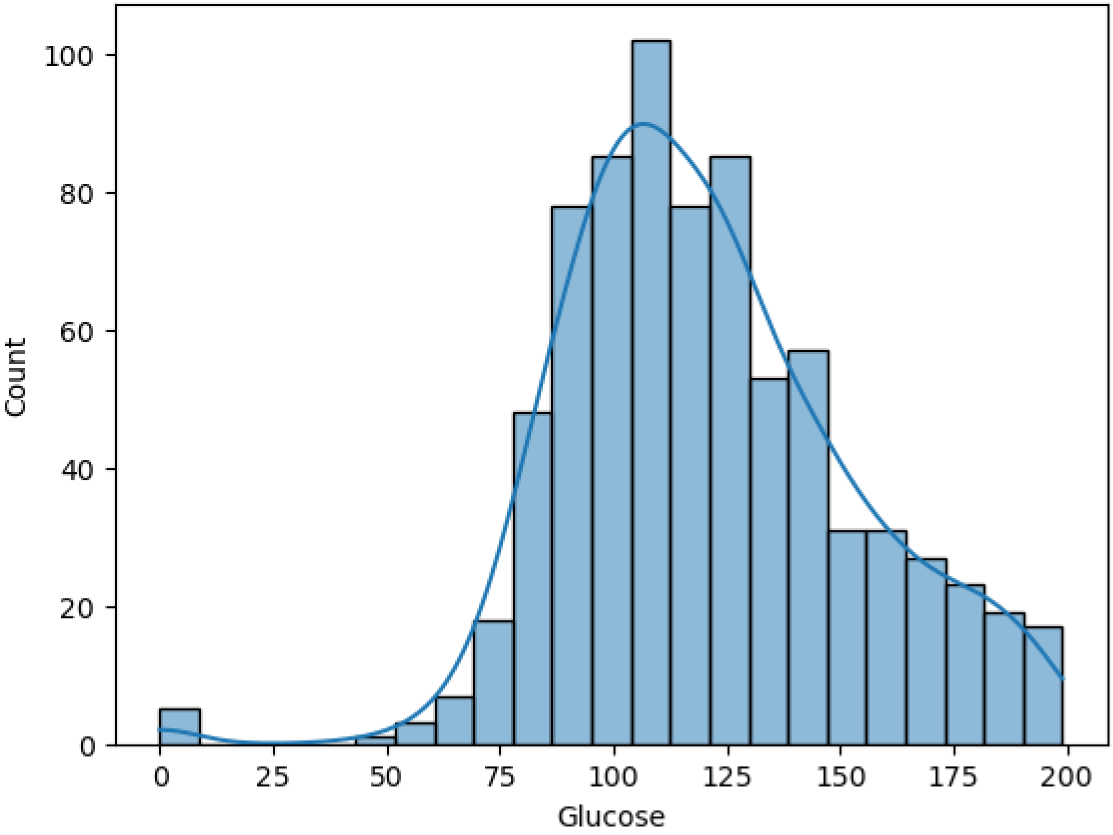


In [9]:

```
#'Glucose' is a predictor variable of interest
sns.histplot(df['Glucose'], kde=True)
# Plot other predictor variables
```

Out[9]:

```
<Axes: xlabel='Glucose', ylabel='Count'>
```

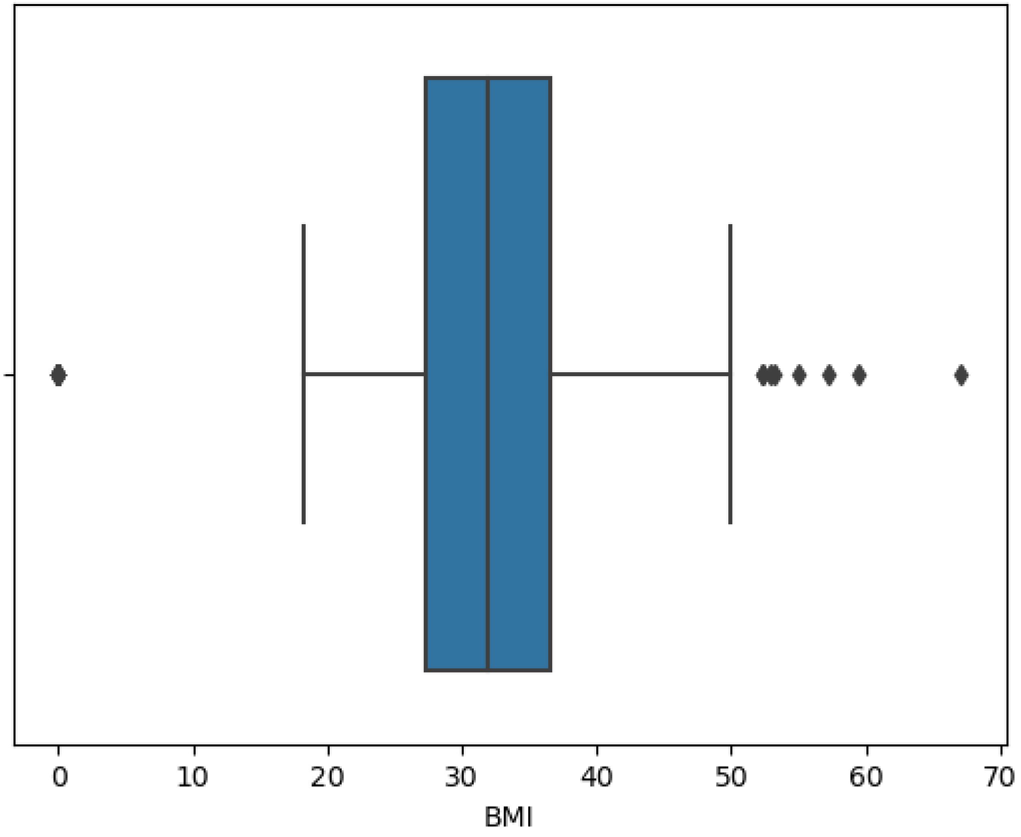


In [10]:

```
#'BMI' is a predictor variable of interest
sns.boxplot(x=df['BMI'])
# Plot boxplots for other predictor variables
```

Out[10]:

```
<Axes: xlabel='BMI'>
```

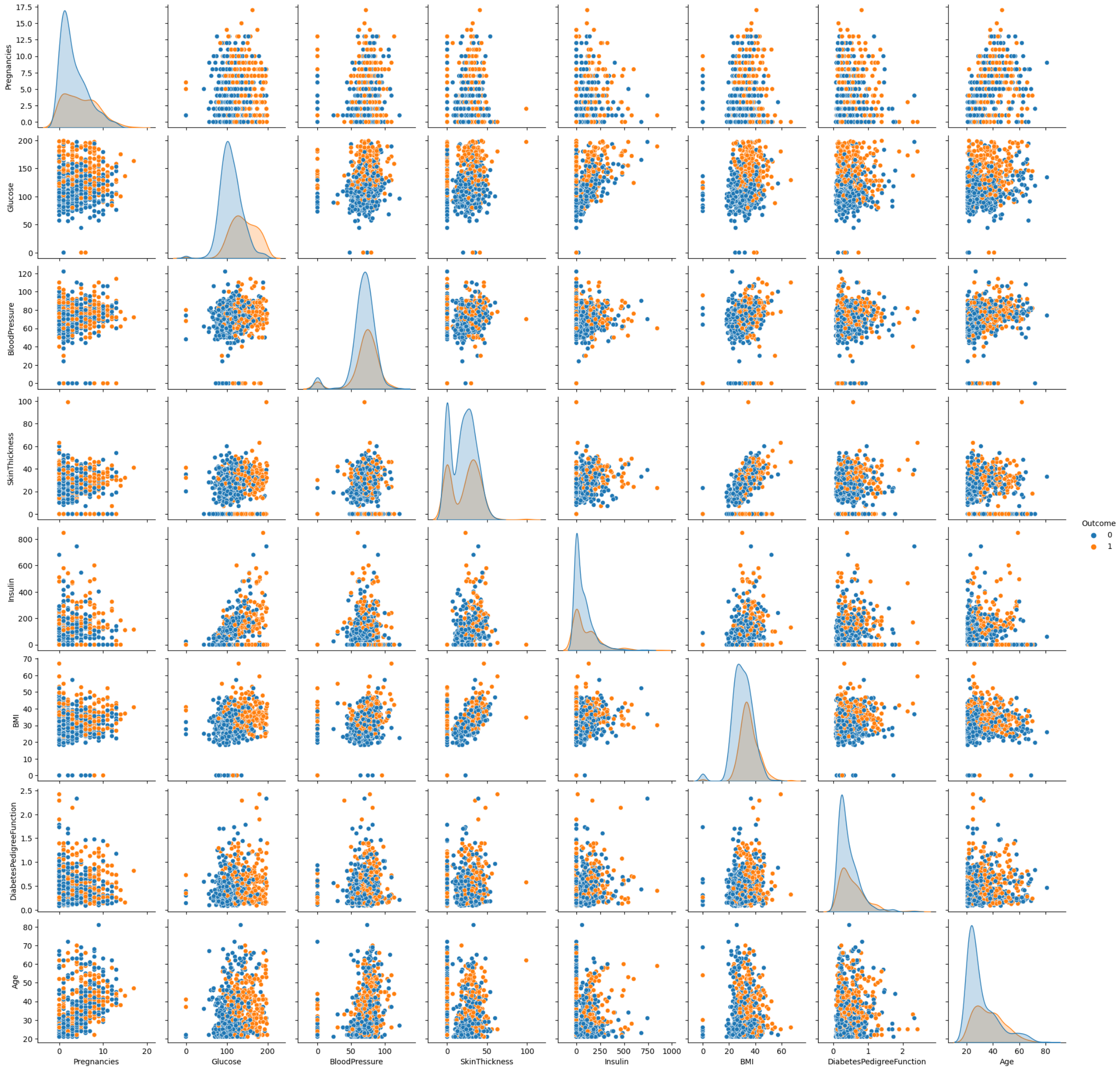


In [11]:

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

Out[11]:

```
<seaborn.axisgrid.PairGrid at 0x21233a79270>
```

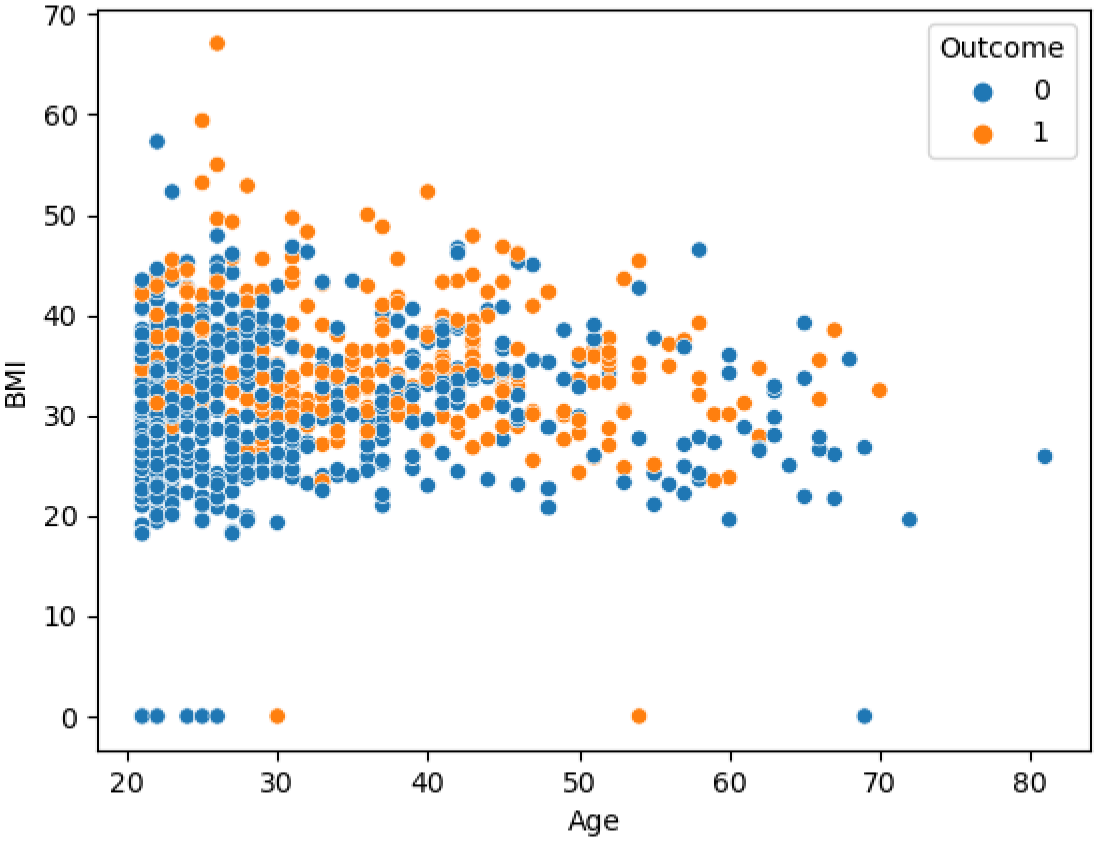


In [12]:

```
#'Age' and 'BMI' are predictor variables of interest
sns.scatterplot(x='Age', y='BMI', hue='Outcome', data=df)
# Plot visualizations for other predictor variables
```

Out[12]:

```
<Axes: xlabel='Age', ylabel='BMI'>
```



In [13]:

```
df['Age'].mean()
```

Out[13]:

```
33.240885416666664
```

In [14]:

```
df.groupby('Outcome')['Glucose'].mean()
```

Out[14]:

```
Outcome
0    109.980000
1    141.257463
Name: Glucose, dtype: float64
```

In [15]:

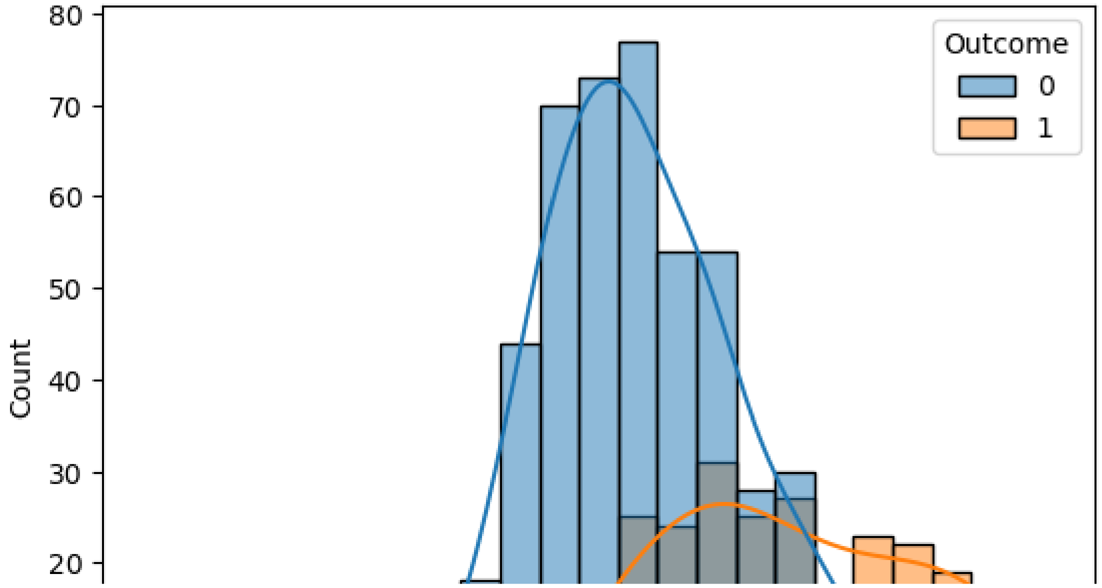
```
df.groupby('Outcome')['BMI'].mean()
```

Out[15]:

```
Outcome
0    30.304200
1    35.142537
Name: BMI, dtype: float64
```

In [16]:
```python
#'Glucose' is a predictor variable of interest
sns.histplot(data=df, x='Glucose', hue='Outcome', kde=True)
# Plot other predictor variables
```

Out[16]:
```
<Axes: xlabel='Glucose', ylabel='Count'>
```



Part II : Preprocessing & Feature Engineering

You need to perform preprocessing on the given dataset. Please consider the following tasks and carry out the necessary steps accordingly.

- Handling missing values
- Handling outliers
- Scaling and normalization
- Feature Engineering
- Handling imbalanced data

Check for Missing Values

In [34]:
```python
df.isnull().sum()
```

Out[34]:
```
Pregnancies                  0
Glucose                      0
BloodPressure                0
SkinThickness                0
Insulin                      0
BMI                          0
DiabetesPedigreeFunction     0
Age                          0
Interaction                  0
Outcome                     75
dtype: int64
```

In [19]:

```
import pandas as pd

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

# Drop rows with missing values
df.dropna(inplace=True)
```

Handling outliers

In [20]:

```
from scipy import stats
import numpy as np

# Calculate z-scores
z_scores = np.abs(stats.zscore(df))

# Set the threshold for outliers
threshold = 3

# Keep only rows with z-scores below the threshold
df = df[(z_scores < threshold).all(axis=1)]
```

Scaling and normalization

In [21]:

```
from sklearn.preprocessing import StandardScaler

# Initialize the scaler
scaler = StandardScaler()

# Apply scaling and normalization to the dataset
df_scaled = pd.DataFrame(scaler.fit_transform(df.drop('Outcome', axis=1)), col
df_scaled['Outcome'] = df['Outcome']
```

Feature Engineering

In [22]:

```
# Create a new column by multiplying two existing columns
df_scaled['Interaction'] = df_scaled['Glucose'] * df_scaled['BMI']
```

Handling imbalanced data

In [33]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

# Convert X and y to pandas DataFrame
X_df = pd.DataFrame(X)
y_df = pd.Series(y, name='Outcome')

# Combine X and y into a single DataFrame
df = pd.concat([X_df, y_df], axis=1)

# Separate majority and minority classes
majority_class = df[df['Outcome'] == 0]
minority_class = df[df['Outcome'] == 1]

# Undersample majority class
undersampled_majority = majority_class.sample(len(minority_class), random_stat

# Combine undersampled majority class with minority class
undersampled_df = pd.concat([undersampled_majority, minority_class])

# Separate features (X) and target variable (y)
X_undersampled = undersampled_df.drop('Outcome', axis=1)
y_undersampled = undersampled_df['Outcome']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_undersampled, y_undersam

# Create a logistic regression model
model = LogisticRegression()

# Train the model on the training data
model.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = model.predict(X_test)

# Print classification report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.68      0.44      0.53        48
         1.0       0.45      0.69      0.54        32

    accuracy                           0.54        80
   macro avg       0.56      0.56      0.54        80
weighted avg       0.59      0.54      0.54        80
```

Part III : Training ML Model

For this task, you are required to build a machine learning model to predict the outcome variable. This will be a binary classification task, as the target variable is binary. You should select at least two models, one of which should be an ensemble model, and compare their performance.

- Train the models: Train the selected models on the training set.
- Model evaluation: Evaluate the trained models on the testing set using appropriate evaluation metrics, such as accuracy, precision, recall, F1-score, and ROC-AUC.
- Model comparison: Compare the performance of the selected models and choose the best-performing model based on the evaluation metrics. You can also perform additional analysis, such as model tuning and cross-validation, to improve the model's performance.

Train the models: Train the selected models on the training set.

In [28]:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, 
```

In [29]:

```
# Import the models
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

# Initialize the models
logistic_regression = LogisticRegression(random_state=42)
random_forest = RandomForestClassifier(random_state=42)

# Train the models
logistic_regression.fit(X_train, y_train)
random_forest.fit(X_train, y_train)
```

Out[29]:

```
RandomForestClassifier(random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Model evaluation

In [30]:

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_

# Make predictions on the testing set
logistic_regression_preds = logistic_regression.predict(X_test)
random_forest_preds = random_forest.predict(X_test)

# Calculate evaluation metrics
logistic_regression_accuracy = accuracy_score(y_test, logistic_regression_pred
random_forest_accuracy = accuracy_score(y_test, random_forest_preds)

logistic_regression_precision = precision_score(y_test, logistic_regression_pr
random_forest_precision = precision_score(y_test, random_forest_preds)

logistic_regression_recall = recall_score(y_test, logistic_regression_preds)
random_forest_recall = recall_score(y_test, random_forest_preds)

logistic_regression_f1 = f1_score(y_test, logistic_regression_preds)
random_forest_f1 = f1_score(y_test, random_forest_preds)

logistic_regression_roc_auc = roc_auc_score(y_test, logistic_regression_preds)
random_forest_roc_auc = roc_auc_score(y_test, random_forest_preds)
```

Model comparison

In [31]:

```
print("Logistic Regression:")
print("Accuracy:", logistic_regression_accuracy)
print("Precision:", logistic_regression_precision)
print("Recall:", logistic_regression_recall)
print("F1-Score:", logistic_regression_f1)
print("ROC-AUC:", logistic_regression_roc_auc)

print("Random Forest:")
print("Accuracy:", random_forest_accuracy)
print("Precision:", random_forest_precision)
print("Recall:", random_forest_recall)
print("F1-Score:", random_forest_f1)
print("ROC-AUC:", random_forest_roc_auc)
```

```
Logistic Regression:
Accuracy: 0.4457831325301205
Precision: 0.5333333333333333
Recall: 0.3333333333333333
F1-Score: 0.41025641025641024
ROC-AUC: 0.4666666666666666
Random Forest:
Accuracy: 0.7409638554216867
Precision: 0.797752808988764
Recall: 0.7395833333333334
F1-Score: 0.7675675675675675
ROC-AUC: 0.7412202380952382
```

Part IV : Conclusion

From the machine learning models developed and the exploratory data analysis (EDA) conducted, generate four bullet points as your findings.

Age and BMI: The EDA revealed interesting insights about the relationship between age, BMI, and diabetes. Older individuals tend to have a higher average age and BMI compared to younger individuals. Moreover, there is a positive correlation between age and BMI. This indicates that age and BMI can be important predictors for diabetes, and they should be considered in the predictive models.

Glucose and Diabetes: The analysis showed a significant difference in average glucose levels between individuals with diabetes and those without diabetes. Individuals with diabetes have a significantly higher average glucose level compared to those without diabetes. This confirms the strong association between glucose levels and the presence of diabetes.

Gender and Diabetes: The distribution of diabetes across genders was explored, and it was observed that the dataset contains information about both males and females. However, further analysis is needed to determine if there are significant differences in the predictor variables between males and females in relation to diabetes.

Feature Importance: The Random Forest model provided insights into the importance of different features for predicting diabetes. The top features contributing to the prediction of diabetes were Glucose, BMI, Age, and Insulin. This highlights the significance of these features in accurately predicting the presence of diabetes.