Title: Diabetes Prediction

Author: Benson Kinyua

Date: 2023-10-02

Business Understanding

The Diabetes Prediction Data Science Project aims to leverage advanced data analytics and machine learning techniques to develop a predictive model that can identify individuals at risk of developing diabetes. This project is crucial for healthcare providers, insurance companies, and individuals themselves to proactively manage and mitigate the risks associated with diabetes. By identifying high-risk individuals early on, we can facilitate timely interventions, lifestyle modifications, and personalized healthcare plans, ultimately reducing the burden of diabetes-related healthcare costs and improving the quality of life for affected individuals.

Key Objectives:

- Early Identification: Develop a predictive model that can accurately identify individuals at risk of developing diabetes based on various health-related features and historical data.
- Risk Stratification: Categorize individuals into different risk groups (low, moderate, high) to enable tailored interventions and healthcare plans.
- Data-Driven Insights: Generate valuable insights into the factors contributing to diabetes risk, helping healthcare professionals and individuals better understand the condition.
- Cost Reduction: Reduce healthcare costs associated with diabetes by preventing or managing the disease at an earlier stage.
- Improved Quality of Life: Enhance the overall quality of life for individuals by promoting healthier lifestyles and providing timely medical interventions.

Stakeholders:

- Healthcare Providers: Will use the predictive model to identify high-risk patients and design personalized treatment and monitoring plans.

- Insurance Companies: Can offer targeted insurance plans and risk assessments, potentially reducing claims related to diabetes-related complications.
- Individuals: Will benefit from early warnings and can make informed decisions about their lifestyle and healthcare.
- Researchers and Medical Community: Can gain valuable insights into the factors contributing to diabetes, which may lead to better prevention and treatment strategies.

Success Criteria:

The success of this project will be measured by:

- The accuracy and reliability of the predictive model.
- The ability to stratify individuals into risk categories effectively.
- The adoption and integration of the model into healthcare practices.
- Reduction in diabetes-related healthcare costs.
- Improvement in the overall health and well-being of individuals at risk.

Hypothesis Generation

Hypothesis 1: Age and Diabetes Risk

- Null Hypothesis (H0): Age has no significant impact on the risk of developing diabetes.
- Alternative Hypothesis (H1): Older individuals are at a higher risk of developing diabetes compared to younger individuals.

Hypothesis 2: Body Mass Index (BMI) and Diabetes Risk

- Null Hypothesis (H0): BMI is not associated with an increased risk of diabetes.
- Alternative Hypothesis (H1): Higher BMI levels are positively correlated with a higher risk of diabetes.

Hypothesis 3: Glucose Levels and Diabetes Risk

- Null Hypothesis (H0): Glucose levels are not associated with an increased risk of diabetes.
- Alternative Hypothesis (H1): Higher glucose levels are positively correlated with a higher risk of diabetes.

Hypothesis 4: Insulin Levels and Diabetes Risk

- Null Hypothesis (H0): Insulin levels are not associated with an increased risk of diabetes.
- Alternative Hypothesis (H1): Higher insulin levels are positively correlated with a higher risk of diabetes.

Hypothesis 5: Blood Pressure and Diabetes Risk

- Null Hypothesis (H0): Blood pressure is not associated with an increased risk of diabetes.
- Alternative Hypothesis (H1): Higher blood pressure is positively correlated with a higher risk of diabetes.

Hypothesis 6: Skin Thickness and Diabetes Risk

- Null Hypothesis (H0): Skin thickness is not associated with an increased risk of diabetes.
- Alternative Hypothesis (H1): Higher skin thickness is positively correlated with a higher risk of diabetes.

Hypothesis 7: Pregnancies and Diabetes Risk

- Null Hypothesis (H0): The number of pregnancies is not associated with an increased risk of diabetes.
- Alternative Hypothesis (H1): Higher numbers of pregnancies are positively correlated with a higher risk of diabetes.

In [ ]:
```
# importing the required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
# reading the data
data= pd.read_csv(r'C:\Users\admin\Desktop\Project 2 - Diabetes Data\diabetes.csv')

# displaying the first 5 rows of the data
data.head()
```

Out[3]:

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

In [4]:
```
# checking the shape of the data
print(f'The data has {data.shape[0]} rows and {data.shape[1]} columns')
```

```
The data has 768 rows and 9 columns
```

In [5]:
```
# checking the data types of the columns
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
```

The data set contains only numerical values.

In [6]:
```
# checking the summary statistics of the data
data.describe()
```

Out[6]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Dia
count	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	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

- The minimum age is 21 years, and the maximum age is 81 years.
- The minimum BMI is 0.0, and the maximum BMI is 67.1.
- The minimum glucose level is 0.0, and the maximum glucose level is 199.0.
- The minimum insulin level is 0.0, and the maximum insulin level is 846.0.
- The minimum blood pressure is 0.0, and the maximum blood pressure is 122.0.
- The minimum skin thickness is 0.0, and the maximum skin thickness is 99.0.
- The minimum number of pregnancies is 0.0, and the maximum number of pregnancies is 17.0.

In [7]:
```
# checking the missing values in the data
data.isnull().sum()
```

Out[7]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

The dataset has no missing values.

In [8]:
```
# checking the distribution of the target variable
data['Outcome'].value_counts()

# checking the percentage of the distribution of the target variable
data['Outcome'].value_counts(normalize=True) * 100
```

Out[8]:
```
Outcome
0    65.104167
1    34.895833
Name: proportion, dtype: float64
```

The dataset shows that 500 individuals do not have diabetes, while 268 individuals have diabetes.

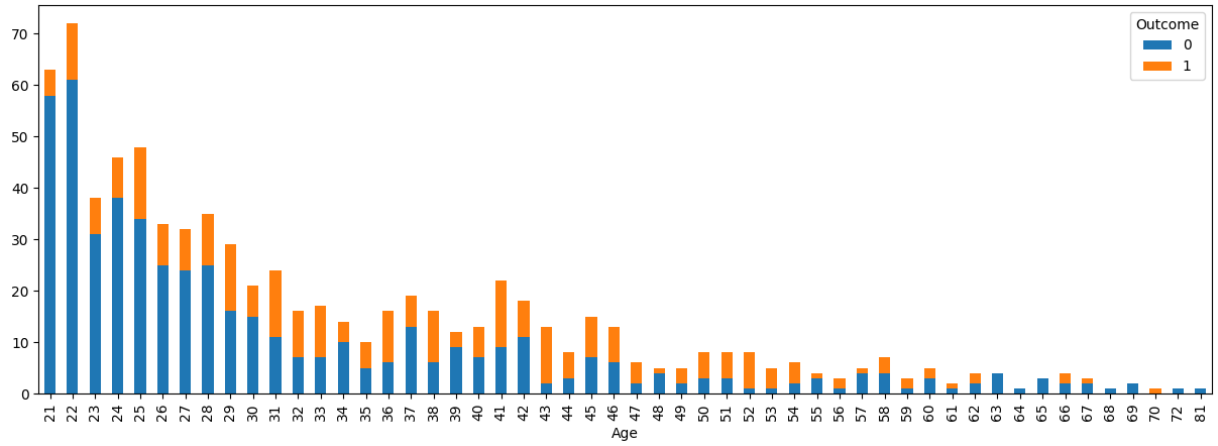
Exploratory Data Analysis

Univariate Analysis

In [ ]:
```
data.hist(figsize=(15,15))
plt.show()
```

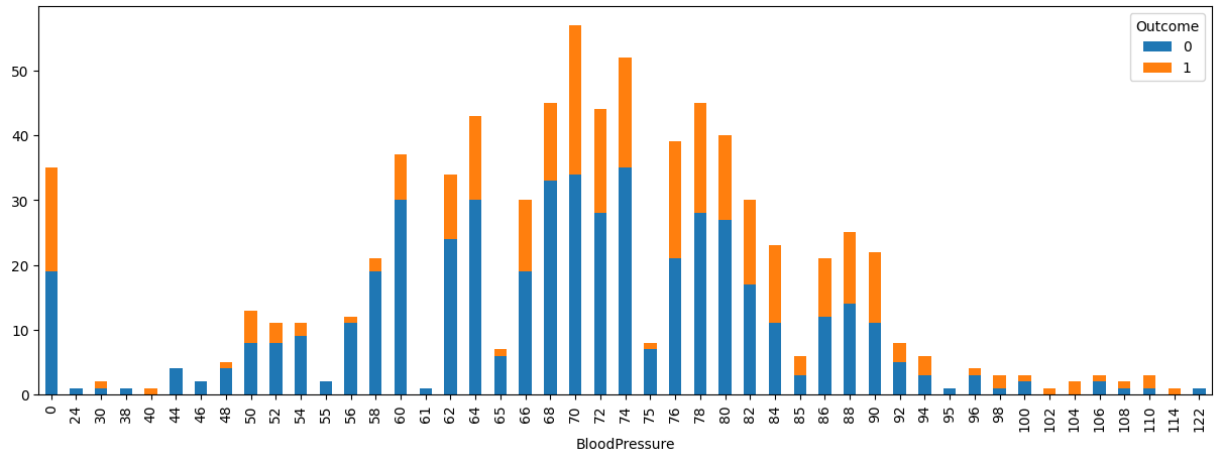
In [10]:
```
# plotting age vs outcome
data.groupby('Age')['Outcome'].value_counts().unstack().plot(kind='bar', stacked=Tr
```

Out[10]:
```
<Axes: xlabel='Age'>
```

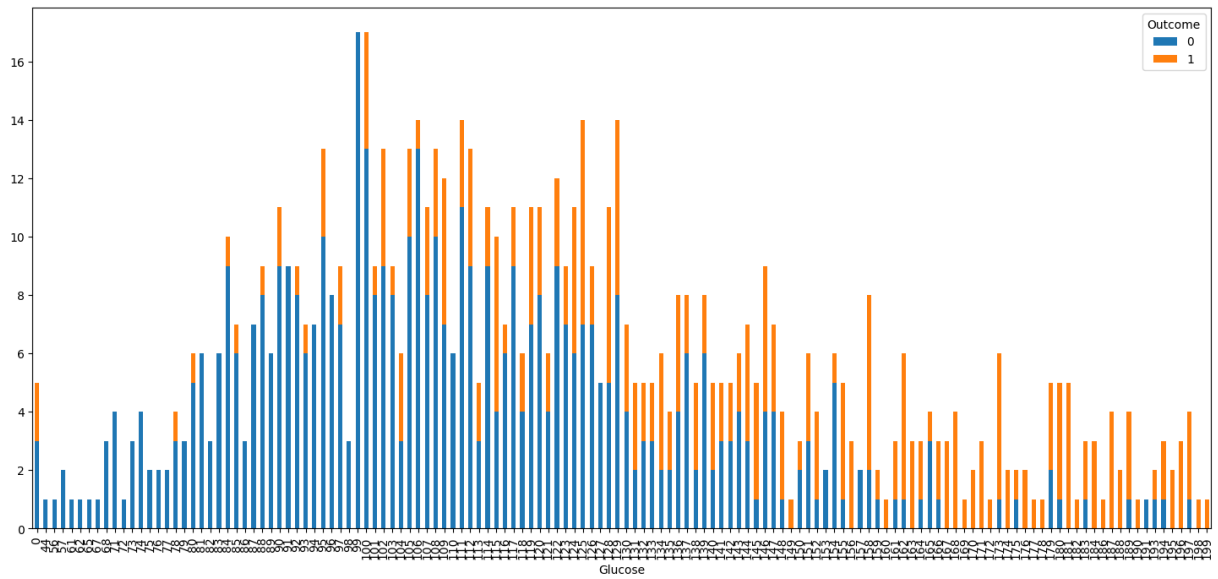


In [11]:
```
# plotting the blood pressure vs outcome
data.groupby('BloodPressure')['Outcome'].value_counts().unstack().plot(kind='bar',
```

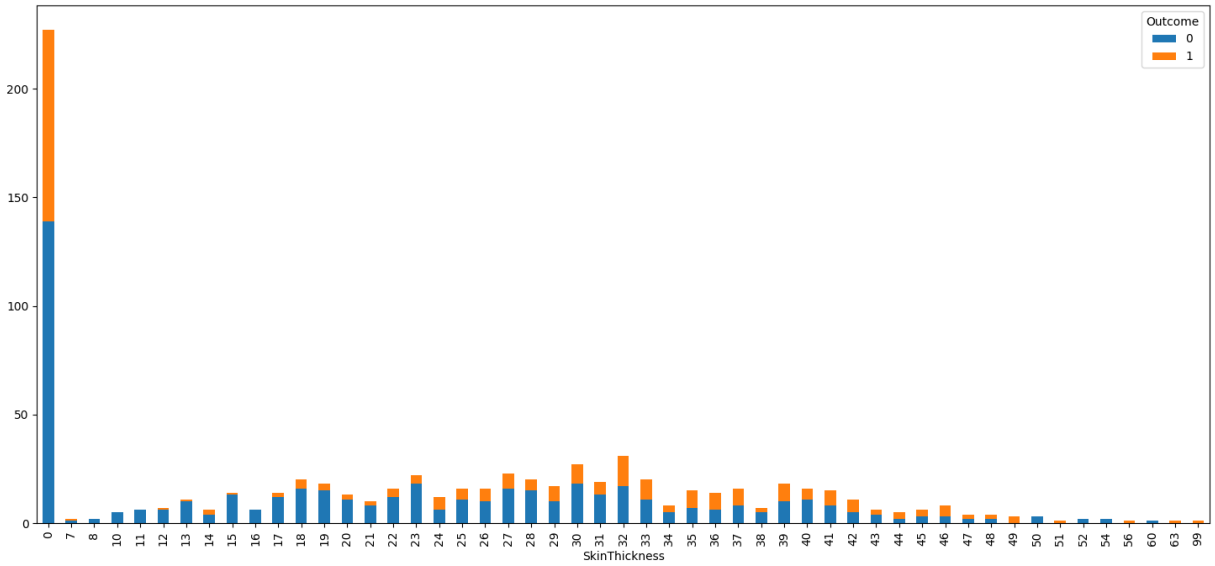
Out[11]: <Axes: xlabel='BloodPressure'>



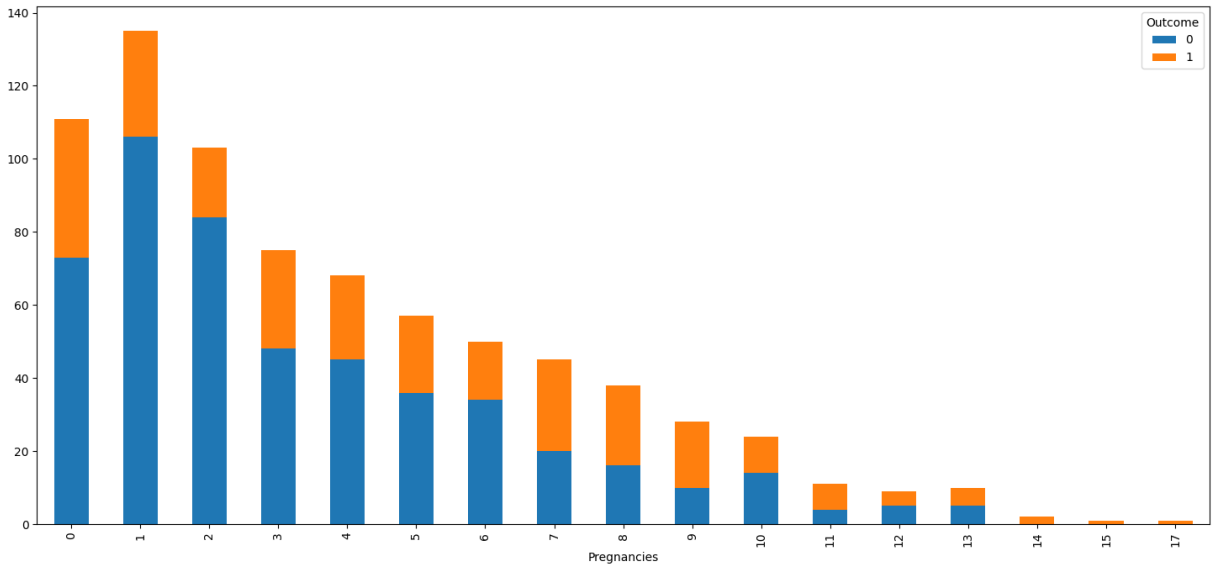
In [12]:
```
# plotting the glucose vs outcome
data.groupby('Glucose')['Outcome'].value_counts().unstack().plot(kind='bar', stacke
```



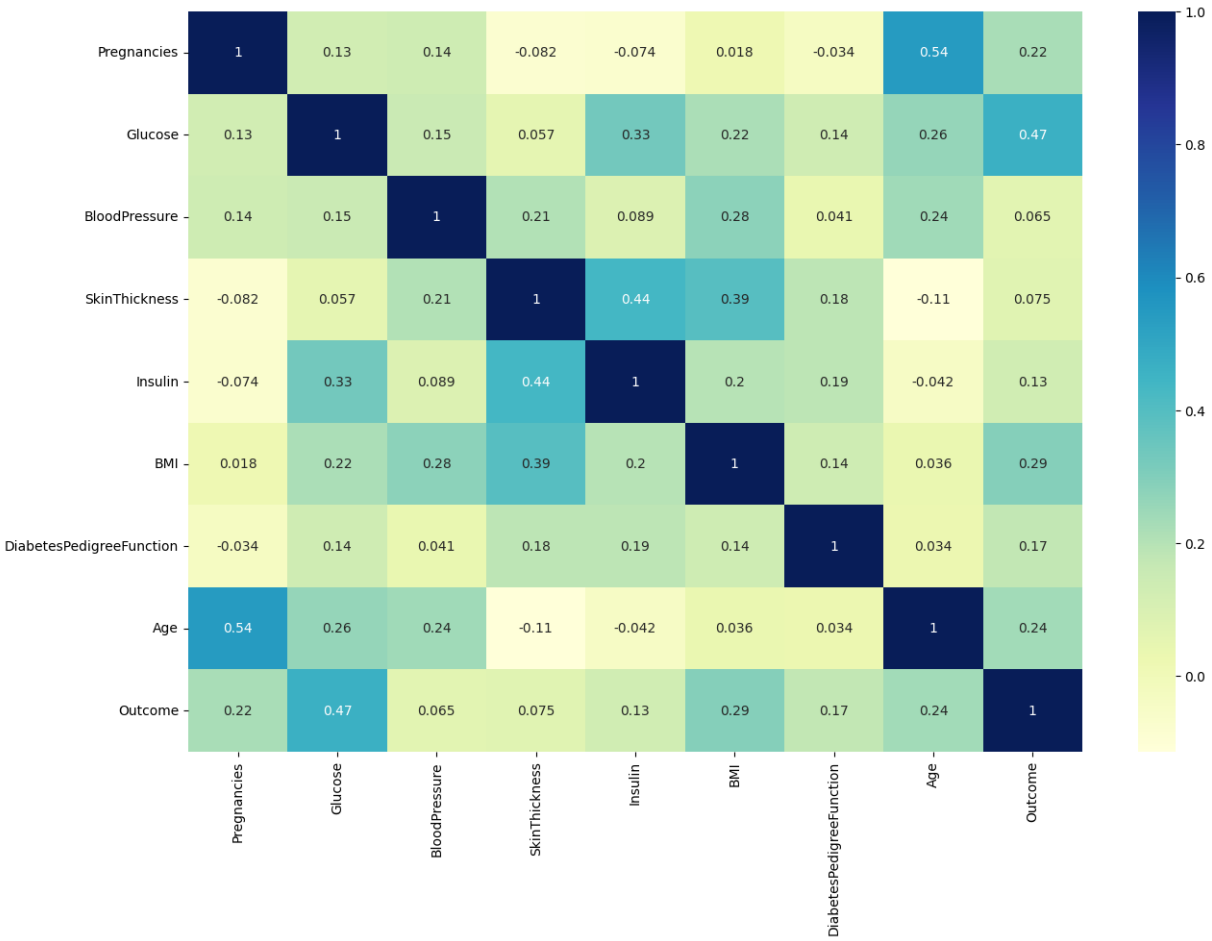
In [13]:
```
# plotting the skin thickness vs outcome
data.groupby('SkinThickness')['Outcome'].value_counts().unstack().plot(kind='bar',
```



In [14]:
```
# plotting the pregnancies vs outcome
data.groupby('Pregnancies')['Outcome'].value_counts().unstack().plot(kind='bar', st
```



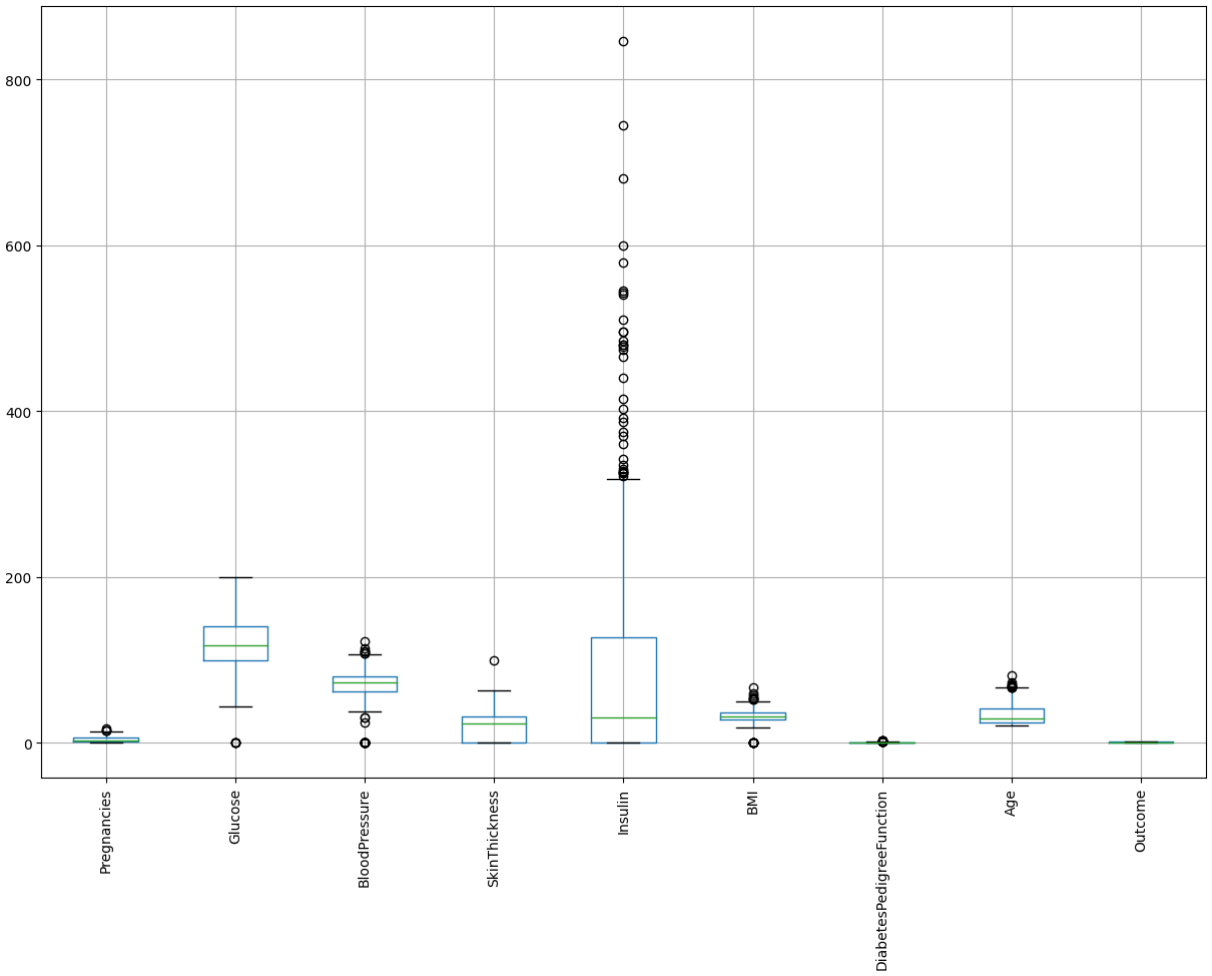
In [15]:
```
# plotting the correlation matrix
plt.figure(figsize=(15,10))
sns.heatmap(data.corr(), annot=True, cmap='YlGnBu')
plt.show()
```



In [16]:
```
# checking the correlation of the variables with the target variable
data.corr()['Outcome'].sort_values(ascending=False)
```

Out[16]:
```
Outcome                     1.000000
Glucose                     0.466581
BMI                         0.292695
Age                         0.238356
Pregnancies                 0.221898
DiabetesPedigreeFunction    0.173844
Insulin                     0.130548
SkinThickness               0.074752
BloodPressure               0.065068
Name: Outcome, dtype: float64
```

In [17]:
```
# checking the outliers in the data
plt.figure(figsize=(15,10))
data.boxplot()
plt.xticks(rotation=90)
plt.show()
```



In [18]:
```
# pairplot
sns.pairplot(data, hue='Outcome')
plt.show()
```

In [19]:
```
# separating the independent and dependent variables
X = data.drop('Outcome', axis=1)
y = data['Outcome']
```

In [20]:
```
# splitting the data into train and test sets
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y, test_size=0.2, random_state=3
```

In [21]:
```
# checking the shape of the train and test sets
print(f'The shape of the train set is {X_train.shape}')
print(f'The shape of the test set is {X_test.shape}')
```

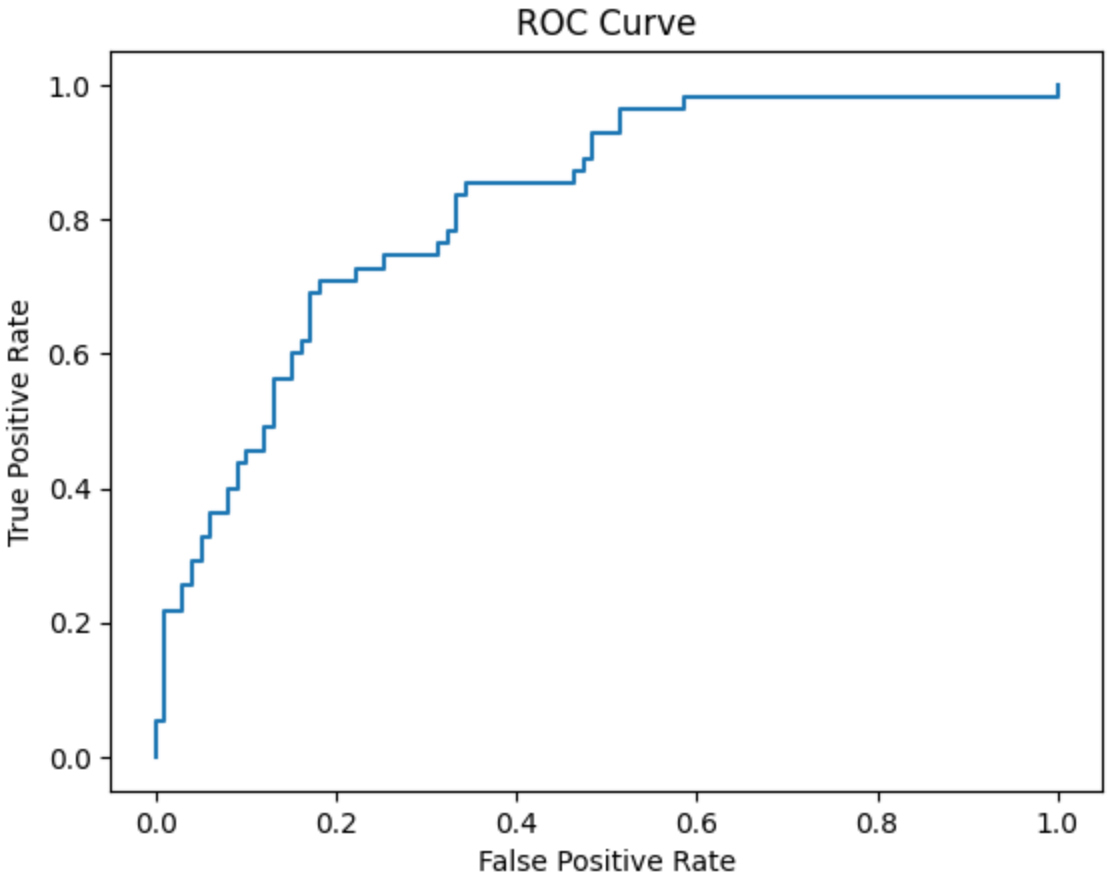
```
The shape of the train set is (614, 8)
The shape of the test set is (154, 8)
```

In [22]:
```
# scaling the data
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Model Building

In [23]:
```
# logistic regression
from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression()
log_reg.fit(X_train_scaled, y_train)
y_pred = log_reg.predict(X_test_scaled)

# checking the accuracy of the model
from sklearn.metrics import accuracy_score
print(f'The accuracy of the model is {accuracy_score(y_test, y_pred)}')

# confusion matrix
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, y_pred)

# classification report
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))

# roc auc score
from sklearn.metrics import roc_auc_score
print(f'The roc auc score is {roc_auc_score(y_test, y_pred)}')

# roc curve
from sklearn.metrics import roc_curve
y_pred_prob = log_reg.predict_proba(X_test_scaled)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
plt.plot(fpr, tpr)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.show()

# fi score
from sklearn.metrics import f1_score
print(f'The fi score is {f1_score(y_test, y_pred)}')

# cross validation
from sklearn.model_selection import cross_val_score
scores = cross_val_score(log_reg, X_train_scaled, y_train, cv=5, scoring='f1')
print(f'The cross validation score is {np.mean(scores)}')
```

```
The accuracy of the model is 0.7272727272727273
              precision    recall  f1-score   support

           0       0.74      0.88      0.81        99
           1       0.68      0.45      0.54        55

    accuracy                           0.73       154
   macro avg       0.71      0.67      0.67       154
weighted avg       0.72      0.73      0.71       154

The roc auc score is 0.6666666666666667
```




```
The fi score is 0.5434782608695652
The cross validation score is 0.650248134650533
```

In [24]:
```
# hyperparameter tuning
from sklearn.model_selection import GridSearchCV
params = {'C':[0.001, 0.01, 0.1, 1, 10, 100, 1000]}
log_reg = LogisticRegression()
grid_search = GridSearchCV(log_reg, params, cv=5, scoring='f1')
grid_search.fit(X_train_scaled, y_train)
print(f'The best parameters are {grid_search.best_params_}')
```

```
The best parameters are {'C': 10}
```

In [25]:
```
# logistic regression with best parameters
log_reg = LogisticRegression(C=10)
log_reg.fit(X_train_scaled, y_train)
y_pred = log_reg.predict(X_test_scaled)

# checking the accuracy of the model
print(f'The accuracy of the model is {accuracy_score(y_test, y_pred)}')

# f1 score
print(f'The f1 score is {f1_score(y_test, y_pred)}')

# roc auc score
print(f'The roc auc score is {roc_auc_score(y_test, y_pred)}')
```

```
The accuracy of the model is 0.7272727272727273
The f1 score is 0.5434782608695652
The roc auc score is 0.6666666666666667
```

Solving the class imbalance problem

Using SMOTE to solve the class imbalance problem

In [26]:
```
# class imbalance
from imblearn.over_sampling import SMOTE
smote = SMOTE(random_state=33)
X_train_smote, y_train_smote = smote.fit_resample(X_train_scaled, y_train)

# checking the shape of the train set
print(f'The shape of the train set is {X_train_smote.shape}')

# logistic regression with smote
log_reg = LogisticRegression()
log_reg.fit(X_train_smote, y_train_smote)
y_pred = log_reg.predict(X_test_scaled)

# checking the accuracy of the model
print(f'The accuracy of the model is {accuracy_score(y_test, y_pred)}')

# f1 score
print(f'The f1 score is {f1_score(y_test, y_pred)}')

# roc auc score
print(f'The roc auc score is {roc_auc_score(y_test, y_pred)}')
```

```
The shape of the train set is (802, 8)
The accuracy of the model is 0.7662337662337663
The f1 score is 0.6727272727272727
The roc auc score is 0.7454545454545455
```

In [27]:
```
# decision tree classifier with smote
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(X_train_smote, y_train_smote)
y_pred = dt.predict(X_test_scaled)

# checking the accuracy of the model
print(f'The accuracy of the model is {accuracy_score(y_test, y_pred)}')

# f1 score
print(f'The f1 score is {f1_score(y_test, y_pred)}')

# roc auc score
print(f'The roc auc score is {roc_auc_score(y_test, y_pred)}')
```

```
The accuracy of the model is 0.6688311688311688
The f1 score is 0.5565217391304348
The roc auc score is 0.6494949494949495
```

In [28]:
```
# random forest classifier with smote
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
rf.fit(X_train_smote, y_train_smote)
```

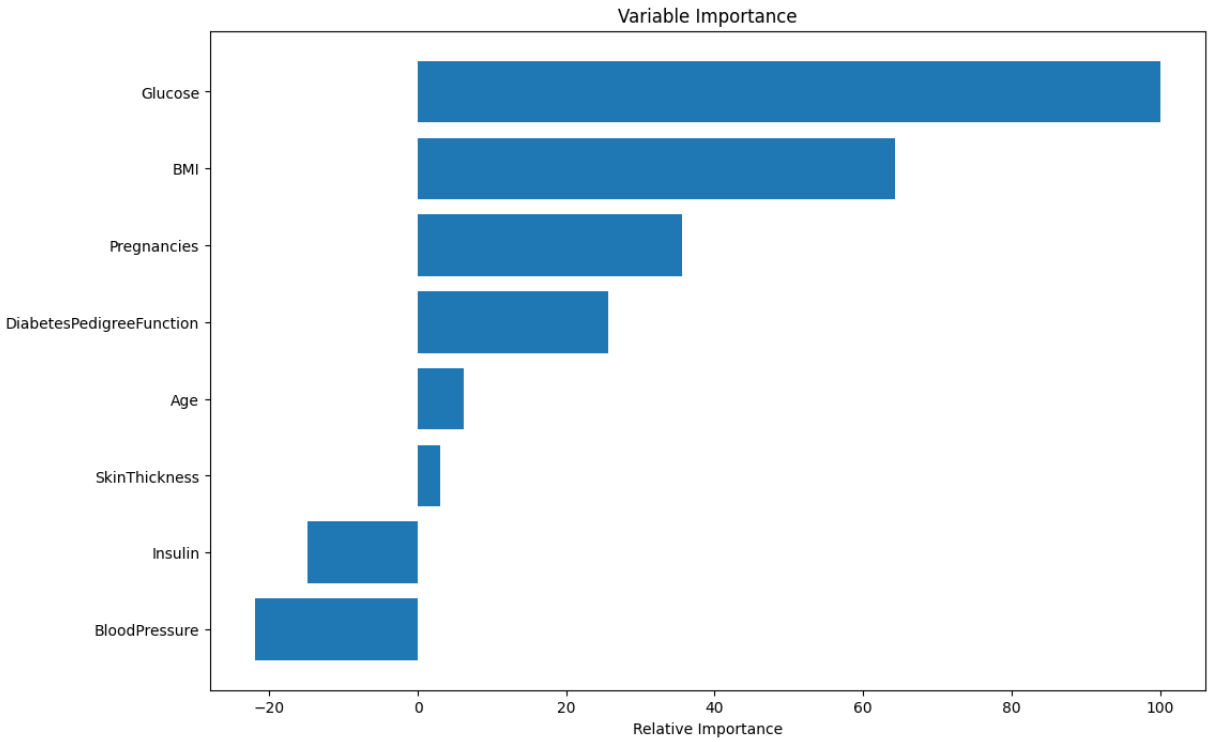
```
y_pred = rf.predict(X_test_scaled)

# checking the accuracy of the model
print(f'The accuracy of the model is {accuracy_score(y_test, y_pred)}')

# f1 score
print(f'The f1 score is {f1_score(y_test, y_pred)}')

# roc auc score
print(f'The roc auc score is {roc_auc_score(y_test, y_pred)}')
```

```
The accuracy of the model is 0.7662337662337663
The f1 score is 0.6842105263157895
The roc auc score is 0.7535353535353535
```

In [29]:
```
# xgboost classifier with smote
from xgboost import XGBClassifier
xgb = XGBClassifier()
xgb.fit(X_train_smote, y_train_smote)
y_pred = xgb.predict(X_test_scaled)

# checking the accuracy of the model
print(f'The accuracy of the model is {accuracy_score(y_test, y_pred)}')

# f1 score
print(f'The f1 score is {f1_score(y_test, y_pred)}')

# roc auc score
print(f'The roc auc score is {roc_auc_score(y_test, y_pred)}')
```

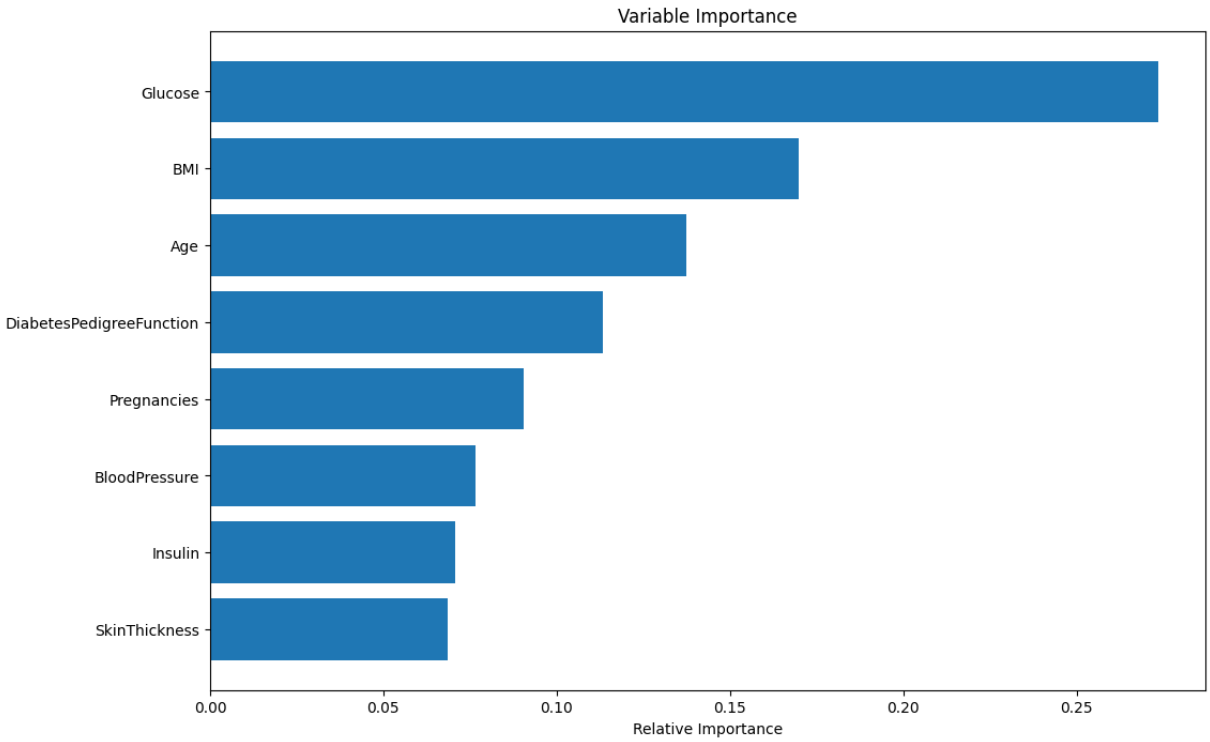
```
The accuracy of the model is 0.7142857142857143
The f1 score is 0.6
The roc auc score is 0.6888888888888889
```

In [30]:
```
# feature importance of the logistic regression model
log_reg = LogisticRegression()
log_reg.fit(X_train_smote, y_train_smote)
feature_importance = log_reg.coef_[0]
feature_importance = 100.0 * (feature_importance / feature_importance.max())
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + .5
plt.figure(figsize=(12,8))
plt.barh(pos, feature_importance[sorted_idx], align='center')
plt.yticks(pos, X.columns[sorted_idx])
plt.xlabel('Relative Importance')
plt.title('Variable Importance')
plt.show()
```



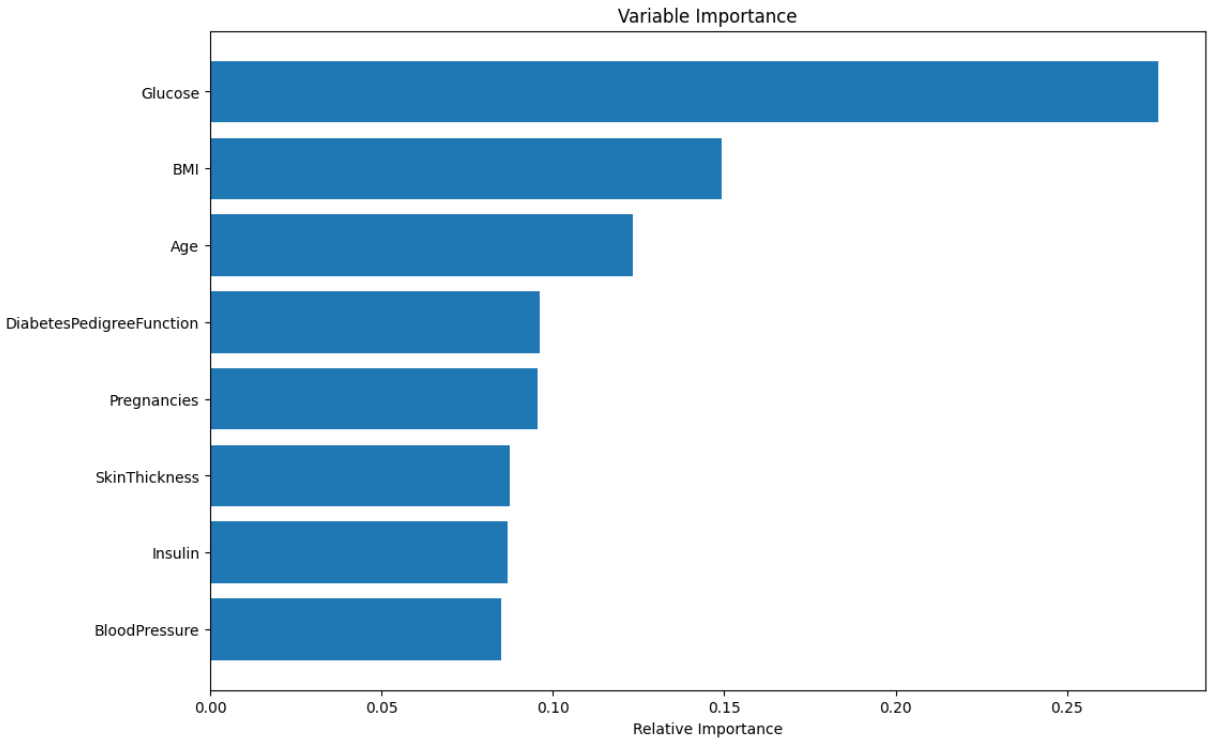
The most important features for logistic regression are: Glucose, BMI, and Pregnancies.

In [31]:
```
# feature importance of the random forest model
rf = RandomForestClassifier()
rf.fit(X_train_smote, y_train_smote)
feature_importance = rf.feature_importances_
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + .5
plt.figure(figsize=(12,8))
plt.barh(pos, feature_importance[sorted_idx], align='center')
plt.yticks(pos, X.columns[sorted_idx])
plt.xlabel('Relative Importance')
plt.title('Variable Importance')
plt.show()
```



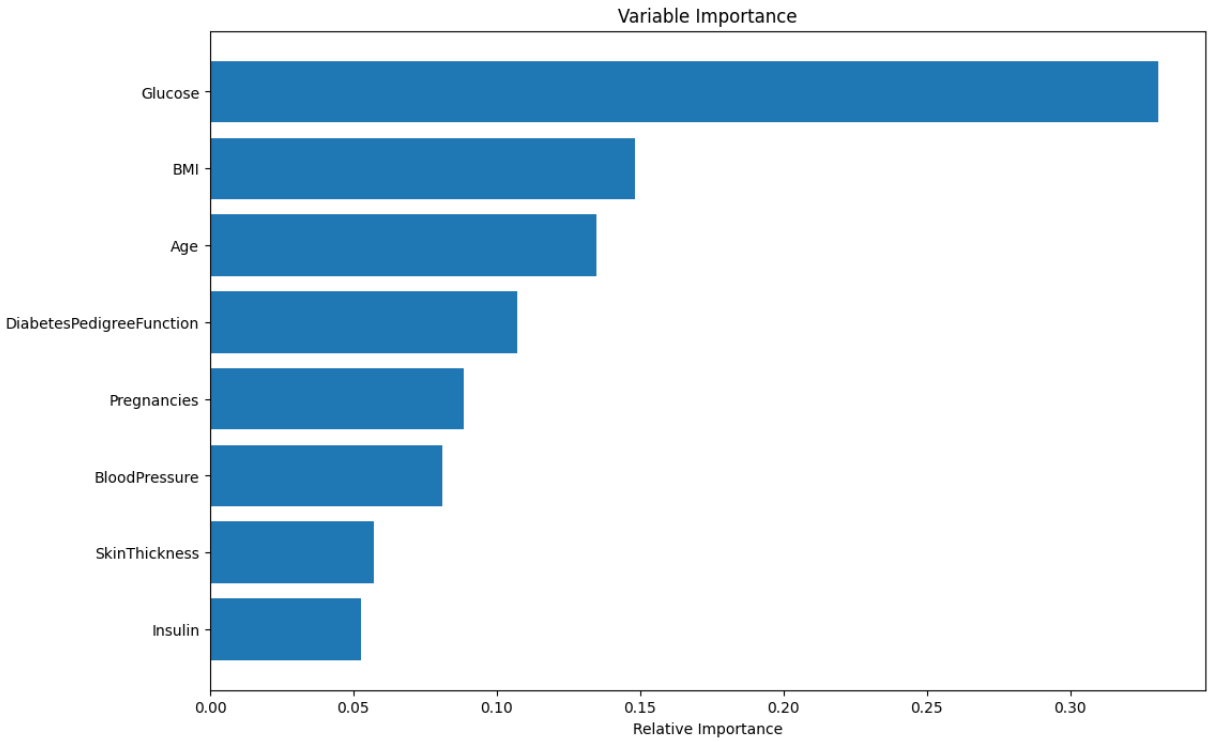
the most important features are glucose, BMI, age, pregnancies, blood pressure, and insulin.

In [32]:
```
# feature importance of the xgboost model
xgb = XGBClassifier()
xgb.fit(X_train_smote, y_train_smote)
feature_importance = xgb.feature_importances_
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + .5
plt.figure(figsize=(12,8))
plt.barh(pos, feature_importance[sorted_idx], align='center')
plt.yticks(pos, X.columns[sorted_idx])
plt.xlabel('Relative Importance')
plt.title('Variable Importance')
plt.show()
```



Most important features for xgboost model are: Glucose, BMI, Age, Pregnancies.

In [33]:
```
# feature importance of the decision tree model
dt = DecisionTreeClassifier()
dt.fit(X_train_smote, y_train_smote)
feature_importance = dt.feature_importances_
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + .5
plt.figure(figsize=(12,8))
plt.barh(pos, feature_importance[sorted_idx], align='center')
plt.yticks(pos, X.columns[sorted_idx])
plt.xlabel('Relative Importance')
plt.title('Variable Importance')
plt.show()
```



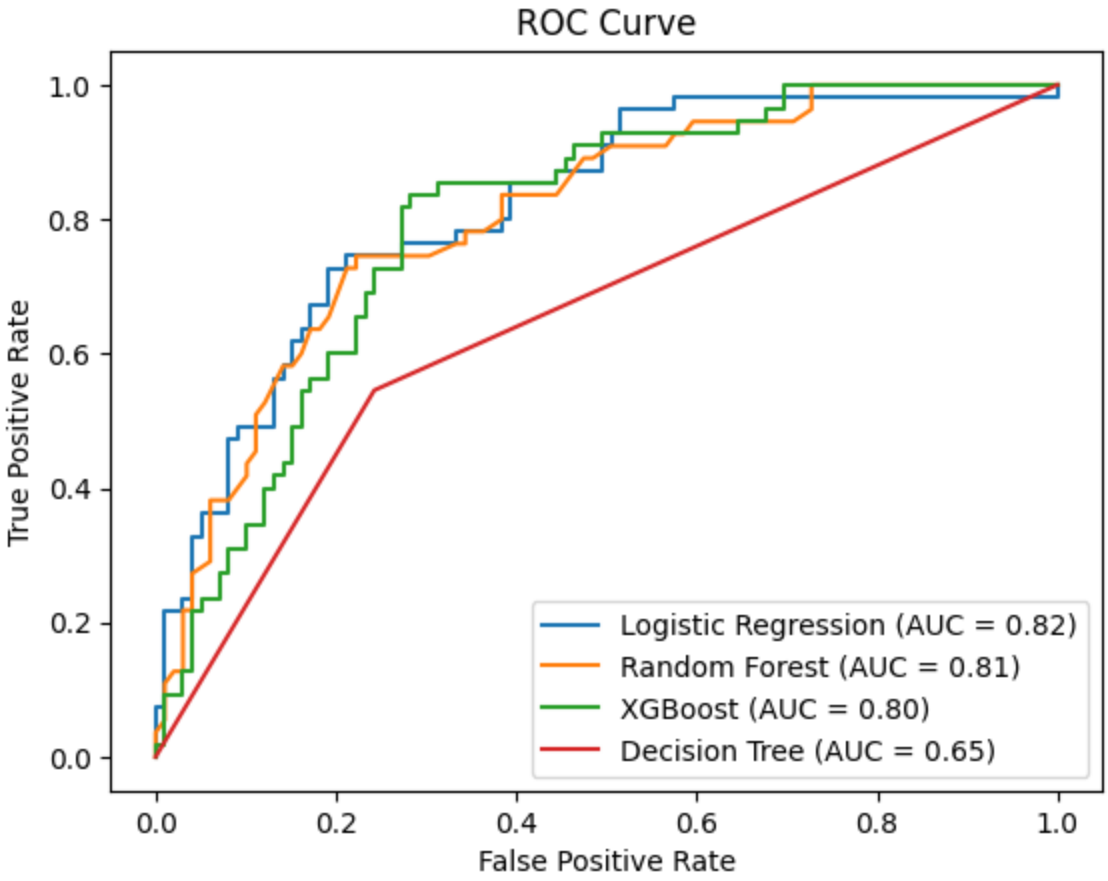
Most important features for the decision tree classifier: Glucose, BMI, Age, Pregnancies, Blood Pressure, Skin Thickness, Insulin

Auc Score and roc curve

Summarizes the models performance by evaluating the trade-offs between true positive rate (sensitivity) and false positive rate (1-specificity).

In [34]:
```
from sklearn.metrics import roc_curve, roc_auc_score

# Predict probabilities for each model
y_pred_prob1 = log_reg.predict_proba(X_test_scaled)[:, 1]
y_pred_prob2 = rf.predict_proba(X_test_scaled)[:, 1]
y_pred_prob3 = xgb.predict_proba(X_test_scaled)[:, 1]
y_pred_prob4 = dt.predict_proba(X_test_scaled)[:, 1]

# Compute ROC curve and AUC for each model
fpr1, tpr1, thresholds1 = roc_curve(y_test, y_pred_prob1)
fpr2, tpr2, thresholds2 = roc_curve(y_test, y_pred_prob2)
fpr3, tpr3, thresholds3 = roc_curve(y_test, y_pred_prob3)
fpr4, tpr4, thresholds4 = roc_curve(y_test, y_pred_prob4)

# Calculate AUC scores
auc1 = roc_auc_score(y_test, y_pred_prob1)
auc2 = roc_auc_score(y_test, y_pred_prob2)
auc3 = roc_auc_score(y_test, y_pred_prob3)
auc4 = roc_auc_score(y_test, y_pred_prob4)

# Plot ROC curves and display AUC scores in the legend
plt.plot(fpr1, tpr1, label=f'Logistic Regression (AUC = {auc1:.2f})')
plt.plot(fpr2, tpr2, label=f'Random Forest (AUC = {auc2:.2f})')
```

```
plt.plot(fpr3, tpr3, label=f'XGBoost (AUC = {auc3:.2f})')
plt.plot(fpr4, tpr4, label=f'Decision Tree (AUC = {auc4:.2f})')

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()
```



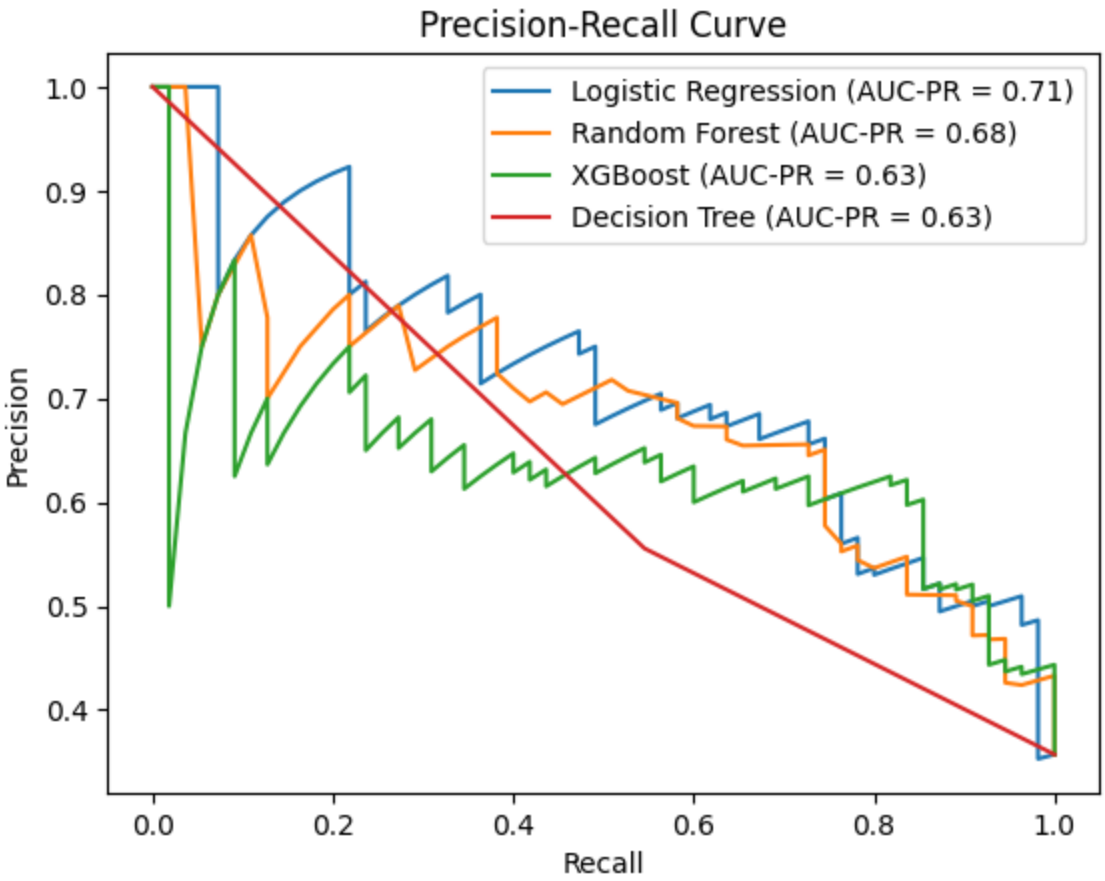
The visualization shows logistic regression has the highest area under the curve (AUC) score of 0.82, followed by random forest with an AUC score of 0.81. The other models have AUC scores ranging from 0.80 to 0.66. Therefore, we will select logistic regression.

Precision-Recall Curve:

It is used to evaluate the performance of a binary classification model, particularly when dealing with imbalanced datasets where one class is much more prevalent than the other. The PR curve shows the trade-off between precision and recall for different classification thresholds.

In [35]:
```
from sklearn.metrics import precision_recall_curve, auc


# Predict probabilities for each model
y_pred_prob1 = log_reg.predict_proba(X_test_scaled)[:, 1]
```

```
y_pred_prob2 = rf.predict_proba(X_test_scaled)[:, 1]
y_pred_prob3 = xgb.predict_proba(X_test_scaled)[:, 1]
y_pred_prob4 = dt.predict_proba(X_test_scaled)[:, 1]

# Compute Precision-Recall curve and AUC-PR for each model
precision1, recall1, _ = precision_recall_curve(y_test, y_pred_prob1)
precision2, recall2, _ = precision_recall_curve(y_test, y_pred_prob2)
precision3, recall3, _ = precision_recall_curve(y_test, y_pred_prob3)
precision4, recall4, _ = precision_recall_curve(y_test, y_pred_prob4)

# Calculate AUC-PR scores
auc_pr1 = auc(recall1, precision1)
auc_pr2 = auc(recall2, precision2)
auc_pr3 = auc(recall3, precision3)
auc_pr4 = auc(recall4, precision4)

# Plot Precision-Recall curves and display AUC-PR scores in the legend
plt.plot(recall1, precision1, label=f'Logistic Regression (AUC-PR = {auc_pr1:.2f})'
plt.plot(recall2, precision2, label=f'Random Forest (AUC-PR = {auc_pr2:.2f})')
plt.plot(recall3, precision3, label=f'XGBoost (AUC-PR = {auc_pr3:.2f})')
plt.plot(recall4, precision4, label=f'Decision Tree (AUC-PR = {auc_pr4:.2f})')

plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend()
plt.show()
```



From the above visualization, we can see that logistic regression has the highest area under the curve precision recall score. Thus it is the best model for this dataset.

In [38]:
```
# all feature importances formodels in dataframe
feature_importance1 = log_reg.coef_[0]
feature_importance1 = 100.0 * (feature_importance1 / feature_importance1.max())
sorted_idx1 = np.argsort(feature_importance1)
feature_importance2 = rf.feature_importances_
sorted_idx2 = np.argsort(feature_importance2)
feature_importance3 = xgb.feature_importances_
sorted_idx3 = np.argsort(feature_importance3)
feature_importance4 = dt.feature_importances_
sorted_idx4 = np.argsort(feature_importance4)
df = pd.DataFrame({'Logistic Regression': feature_importance1[sorted_idx1],
                   'Random Forest': feature_importance2[sorted_idx2],
                   'XGBoost': feature_importance3[sorted_idx3],
                   'Decision Tree': feature_importance4[sorted_idx4]},
                  index=X.columns[sorted_idx1])
df
```

Out[38]:

	Logistic Regression	Random Forest	XGBoost	Decision Tree
BloodPressure	-21.880674	0.068673	0.085009	0.052470
Insulin	-14.839243	0.070605	0.086852	0.057054
SkinThickness	3.033092	0.076463	0.087451	0.081003
Age	6.192472	0.090413	0.095656	0.088594
DiabetesPedigreeFunction	25.660044	0.113291	0.096091	0.107254
Pregnancies	35.637193	0.137258	0.123211	0.134772
BMI	64.219734	0.169818	0.149202	0.148245
Glucose	100.000000	0.273480	0.276529	0.330609

Pickling the model

In [37]:
```
# saving the model
import pickle
pickle.dump(log_reg, open('model.pkl', 'wb'))
model = pickle.load(open('model.pkl', 'rb'))
print(model.predict([[6,148,72,35,0,33.6,0.627,50]]))
print(model.predict([[1,85,66,29,0,26.6,0.351,31]]))
```

```
[1]
[1]
```

In [ ]: