diabeticpredictiveanalysis-4

January 28, 2024

1 Revolutionizing Healthcare with Predictive Analysis: Uncovering Patterns for Enhanced Diabetes Prediction

1.0.1 INTRODUCTION

In the realm of healthcare, the application of Data Science and Machine Learning holds immense potential for improving patient outcomes and disease detection. This study focuses on the comprehensive analysis of diabetic patient data with the primary goal of predicting the presence of diabetes while minimizing the occurrence of false negatives. The significance lies in identifying actual diabetes cases and avoiding oversight, prioritizing recall as a crucial metric for model evaluation.

Through a systematic approach, encompassing data collection, preprocessing, feature engineering, model selection, training, and evaluation, we aim to develop a robust predictive model. The emphasis on minimizing false negatives is paramount, as overlooking true cases of diabetes can have severe consequences.

This research not only delves into the technical aspects of model building but also considers ethical considerations, interpretability, and ongoing monitoring for long-term efficacy in a healthcare context.

1.0.2 DATA SET

This dataset originates from the National Institute of Diabetes and Digestive and Kidney Diseases and was specifically curated to achieve the objective of diagnostically predicting the presence of diabetes in patients.

The dataset includes diagnostic measurements, and the selection of instances adheres to specific constraints imposed during the extraction from a more extensive database.

Notably, all individuals in this dataset are females aged at least 21 years and belong to the Pima Indian heritage. The focus on a subset of the population with these characteristics ensures a targeted analysis of diabetes prediction within a well-defined demographic context.

1.0.3 DATA EXPLORATION AND PREPROCESSING

Upon careful examination, it became evident that the dataset in question is well-suited for predictive analysis.

Predictive analysis, in this context, involves deciphering probable future trends and behaviors based on historical data.

In my analytical approach, I partitioned the dataset into independent and dependent variables to facilitate the derivation of predictions.

The independent variables under consideration encompass crucial health indicators, including Pregnancies, Glucose levels, Blood Pressure, Skin Thickness, Insulin levels, BMI (Body Mass Index), Diabetes Pedigree Function (assessing the likelihood of diabetes based on family history), and Age.

In contrast, the variable 'Outcome' assumes the role of the dependent variable, serving as the focal point for the predictive analysis.

Within the dataset, the numerical values 0 and 1 are assigned to represent the absence or presence of diabetes, respectively. Specifically, a value of 0 signifies that the individual is not diabetic, while a value of 1 indicates the presence of diabetes.

This dichotomous representation facilitates the development of a predictive model to discern the likelihood of diabetes based on the specified diagnostic measurements. G In the realm of feature engineering, the process involved handling missing data, visually inspecting outliers, and normalizing features to enhance the quality of the datase..

In the preliminary stages of data exploration, our analysis revealed a class imbalance with a ratio of 0.54, signifying an uneven distribution in the target variable.

To rectify this, a focused effor such as SMOTE(Synthetic Minority Over-sampling Technique), is used to address the issue of class imbalance in a dataset, particularly in binary classification problems.tIto address theiimbalance and ensure a more equitable representation of classes in the datase.

After training the model, predictions are made using the test data, which comprises 20% of the total dataset.

The accuracy of the model is then calculated and determined.

1.0.4 MODEL EVALUATION

In the realm of modeling and evaluation, two distinct approaches were employed to predict diabetes: K-Nearest Neighbors (KNN) and Logistic Regression.

While KNN exhibited an accuracy rate of 69.5%, it faced challenges with low recall and precision metrics, indicating limitations in effectively capturing true instances of diabetes and minimizing false positives.

On the other hand, Logistic Regression demonstrated a more balanced performance, achieving an overall accuracy of 79.9%.

The precision of 79.4% signifies a model that is adept at correctly identifying individuals with diabetes, mitigating the occurrence of false alarms. Moreover, the recall of 56% implies a commendable ability to capture a substantial proportion of actual diabetes cases.

The F1-score, a valuable metric for striking a balance between minimizing false negatives and avoiding false positives, reached 63.8%.

In essence, these evaluation metrics shed light on the trade-offs inherent in the models: a delicate equilibrium between accurately identifying diabetes cases and avoiding spurious predictions, showcasing the nuanced performance of each model in the context of diabetes prediction.

1.0.5 MODEL COMPARISON AND SELECTION:

In the process of comparing and selecting models, KNN, despite achieving an accuracy rate of 69.5%, Logistic Regression ultimately emerged as the favored model.

This decision was steered by the balanced F1-score of Logistic Regression, which serves as a comprehensive metric indicative of superior overall performance.

The significance of the balanced F1-score lies in its capacity to strike a crucial balance between minimizing false negatives, particularly crucial in the context of predicting diabetes.

Despite KNN's higher accuracy, Logistic Regression's ability to achieve a more harmonious equilibrium between precision and recall makes it the preferred choice for robust and reliable diabetes prediction.

1.0.6 CONCLUSION

In conclusion, the preferred model for accurate diabetes prediction is Logistic Regression, striking a balanced trade-off between recall, precision, and accuracy.

Notably, the DiabetesPedigreeFunction emerged as the most influential feature in the logistic regression model, underscoring its significance in making accurate predictions.

As a recommendation, Logistic Regression stands out as the optimal choice for robust and reliable diabetes prediction, offering a well-balanced performance across key evaluation metrics

1.0.7 FUTURE DIRECTIONS:

Delve deeper into advanced methodologies, including Random Forest, to augment predictive capabilities.

Implement a regimen of ongoing monitoring and refinement for the model, ensuring its adaptability to the dynamic landscape of evolving healthcare data. ### ACHIEVEMENTS:

Effectively managed class imbalance and fine-tuned features for optimal performance.

Logistic Regression attained an accuracy of 79.9%, showcasing significant potential for impactful healthcare predictions

By utilizing these insights, the project not only adds value to predictive analytics in healthcare but also underscores the importance of balanced model evaluation for practical applications.

m.

!pip install imbalanced-learn

[103]:
```
# Data Manipulation and analysis
#Pandas: Data analysis and manipulation library for working with structured␣
 ↪data using Data Frame and Series.
import pandas as pd

#NumPy: Numerical computing library supporting large, multi-dimensional arrays␣
 ↪and matrices, with high-level mathematical functions.
import numpy as np
```

```
# Data visualization
#Matplotlib: Comprehensive plotting library providing interface for creating␣
 ↪various plots like line, scatter, bar, and histograms.
import matplotlib.pyplot as plt

#Seaborn: Statistical data visualization library for creating attractive and␣
 ↪informative graphics, based on Matplotlib.
import seaborn as sns
```

[104]:
```
# Ignore warnings
import warnings
warnings.filterwarnings('ignore')
```

[105]:
```
#Machine Learning libraries
from sklearn import datasets
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler

#Technique for splitting data into training and testing sets to assess model␣
 ↪performance.
from sklearn.model_selection import train_test_split


#Logistic Regression: Method for predicting the probability of a binary outcome␣
 ↪using the logistic function.
from sklearn.linear_model import LogisticRegression

#Sklearn: Python's Scikit-learn, a powerful machine learning library providing␣
 ↪tools for data analysis and model building.
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier

#Accuracy: Metric measuring the proportion of correctly classified instances in␣
 ↪a classification model.
from sklearn.metrics import accuracy_score,␣
 ↪classification_report,confusion_matrix



#SMOTE: synthetic oversampling on the training set. This ensures that the model␣
 ↪is trained on a more balanced dataset,
#reducing the risk of bias and improving its ability to generalize to both␣
 ↪classes.
from imblearn.over_sampling import SMOTE
```

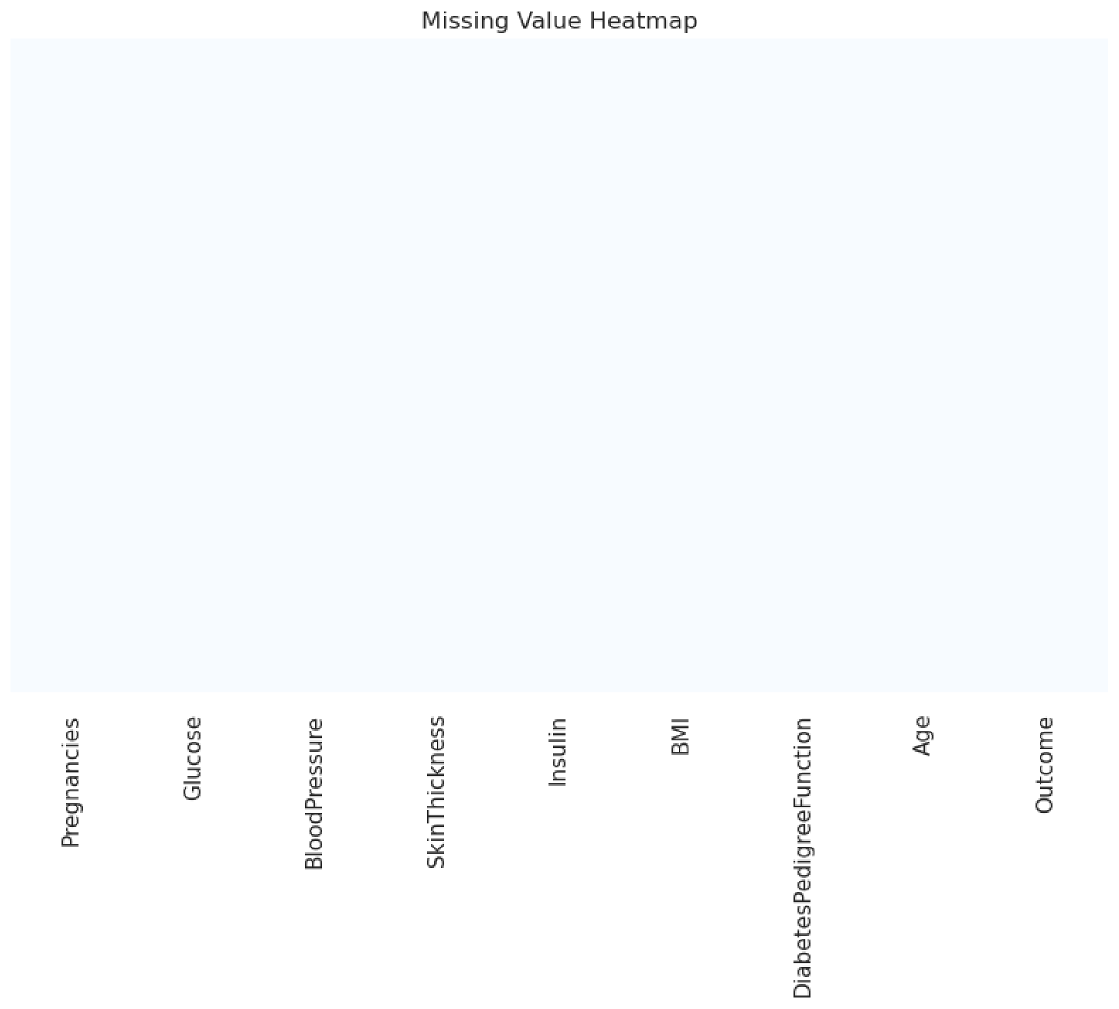
[106]:
```
df=pd.read_csv('diabetes.csv')
df
```

[106]:
```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0              6      148             72             35        0  33.6
1              1       85             66             29        0  26.6
2              8      183             64              0        0  23.3
3              1       89             66             23       94  28.1
4              0      137             40             35      168  43.1
..           …        …             …             …       …    …
763           10      101             76             48      180  32.9
764            2      122             70             27        0  36.8
765            5      121             72             23      112  26.2
766            1      126             60              0        0  30.1
767            1       93             70             31        0  30.4

     DiabetesPedigreeFunction  Age  Outcome
0                       0.627   50        1
1                       0.351   31        0
2                       0.672   32        1
3                       0.167   21        0
4                       2.288   33        1
..                        …  …      …
763                     0.171   63        0
764                     0.340   27        0
765                     0.245   30        0
766                     0.349   47        1
767                     0.315   23        0

[768 rows x 9 columns]
```

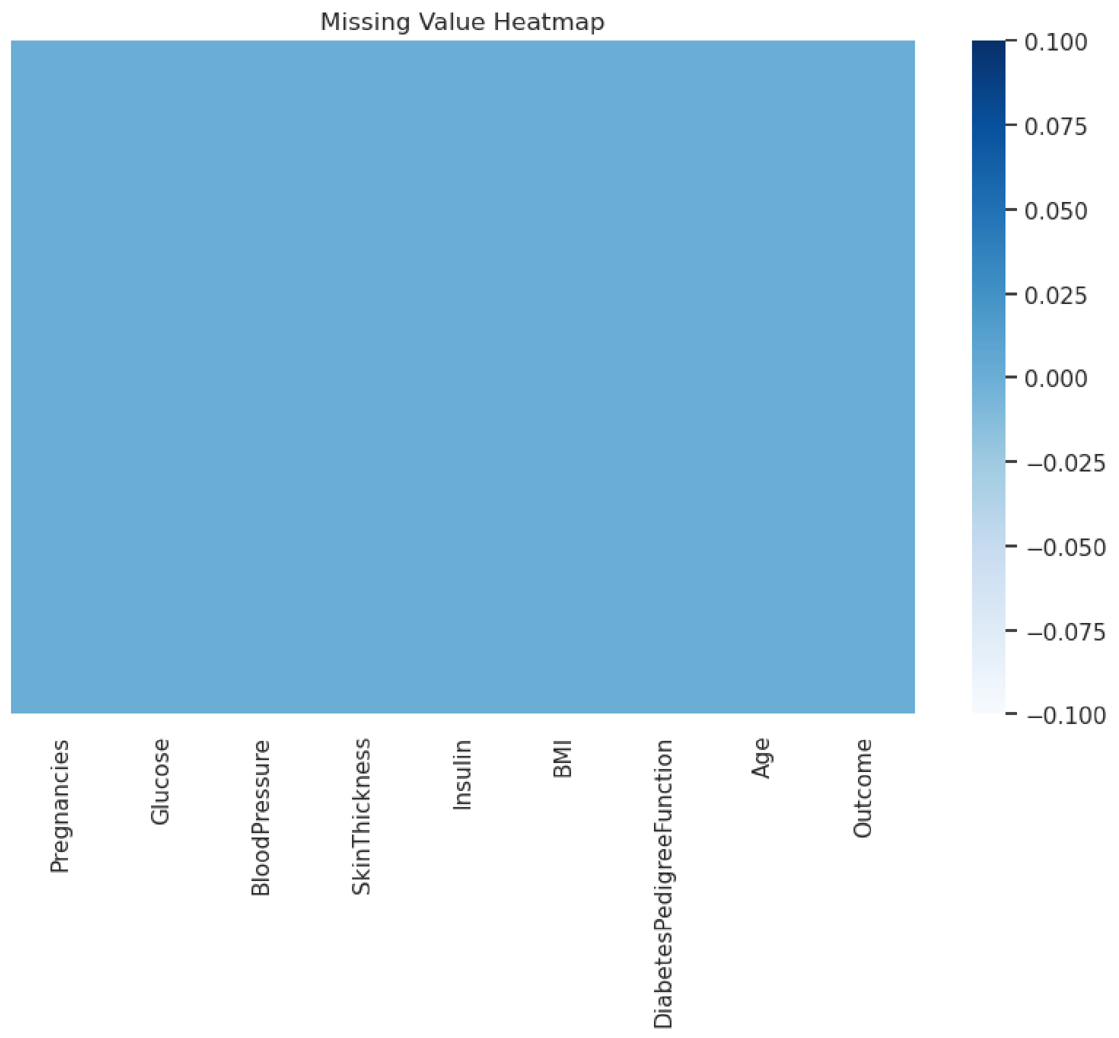
[107]:
```
#check for missing values
missing_values = df.isnull().sum()
#count of missing values
print("Missing Values", missing_values)
```

```
Missing Values Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

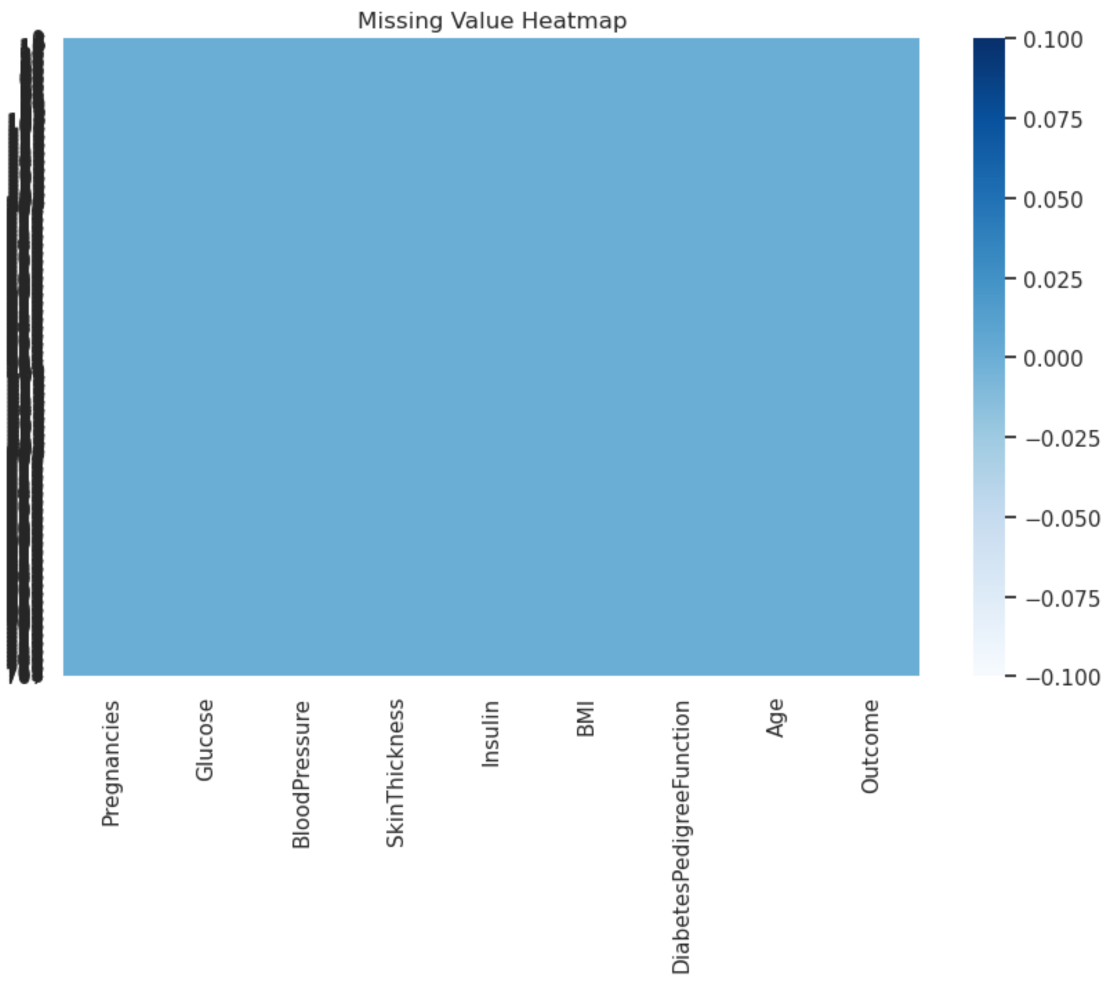
```
[108]: #heatmap of missing values with different color palette
plt.figure(figsize=(10,6))
sns.heatmap(df.isnull(), cmap="Blues", cbar=False, yticklabels=False)
plt.title("Missing Value Heatmap")
plt.show()
```



```
[109]: plt.figure(figsize=(10,6))
sns.heatmap(df.isnull(), cmap="Blues", cbar=True, yticklabels=False)
plt.title("Missing Value Heatmap")
plt.show()
```

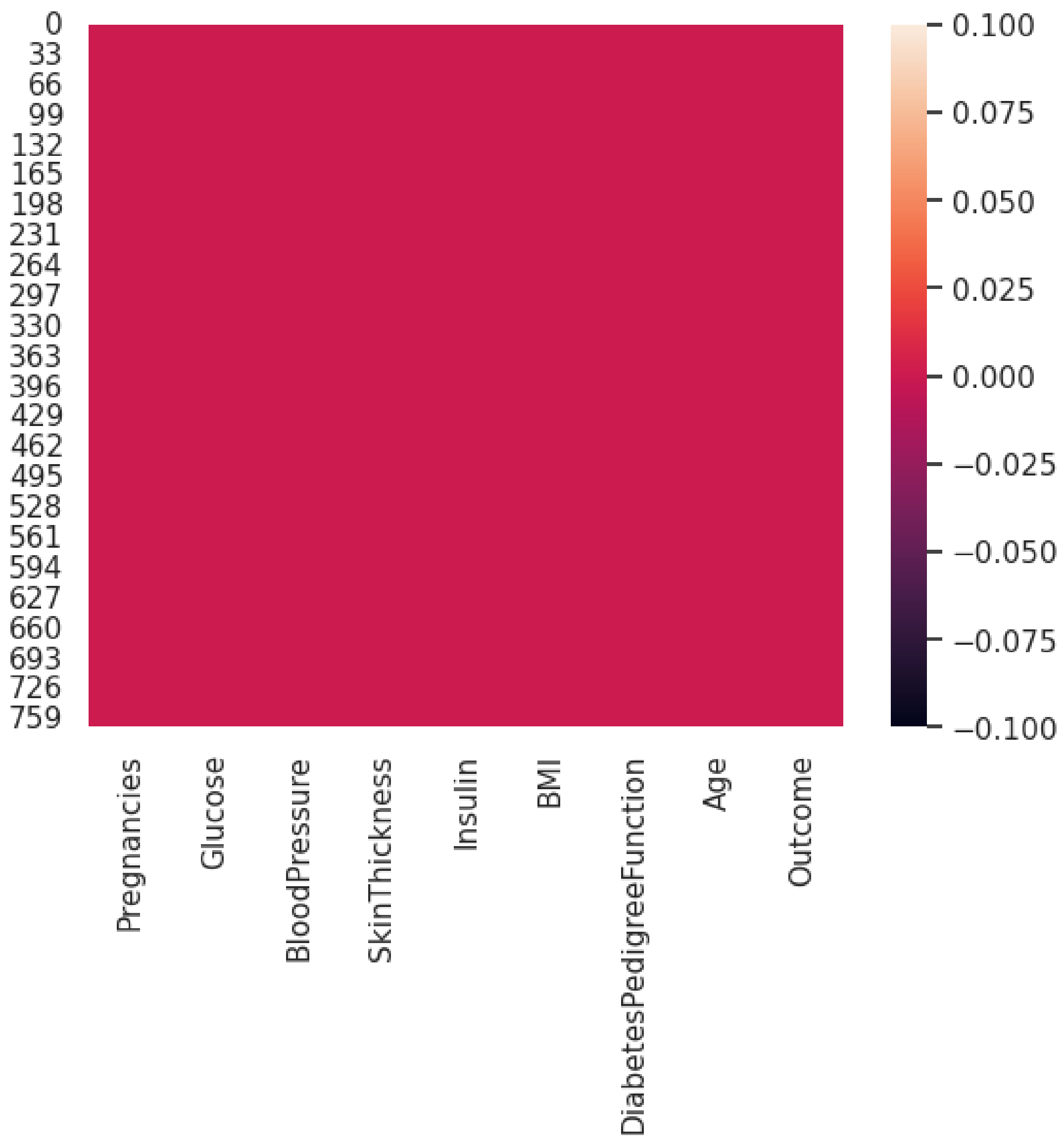


[110]:
```
plt.figure(figsize=(10,6))
sns.heatmap(df.isnull(), cmap="Blues", cbar=True, yticklabels=True)
plt.title("Missing Value Heatmap")
plt.show()
```



```
[111]: sns.heatmap(df.isnull())
```

```
[111]: <Axes: >
```



[112]:
```
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3-Q1
outliers = ((df < (Q1-1.5*IQR)) | (df> (Q3+1.5*IQR))).any(axis = 1)
outliers #a boolean array where each value will return a True or False
```

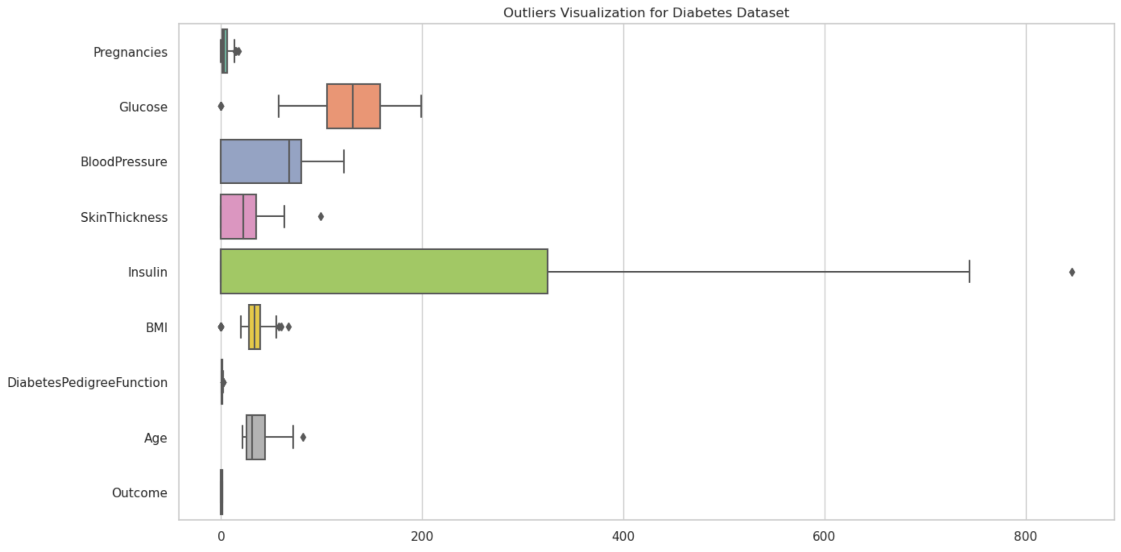
[112]:
```
0      False
1      False
2      False
3      False
4       True
```

```
        …
763    False
764    False
765    False
766    False
767    False
Length: 768, dtype: bool
```

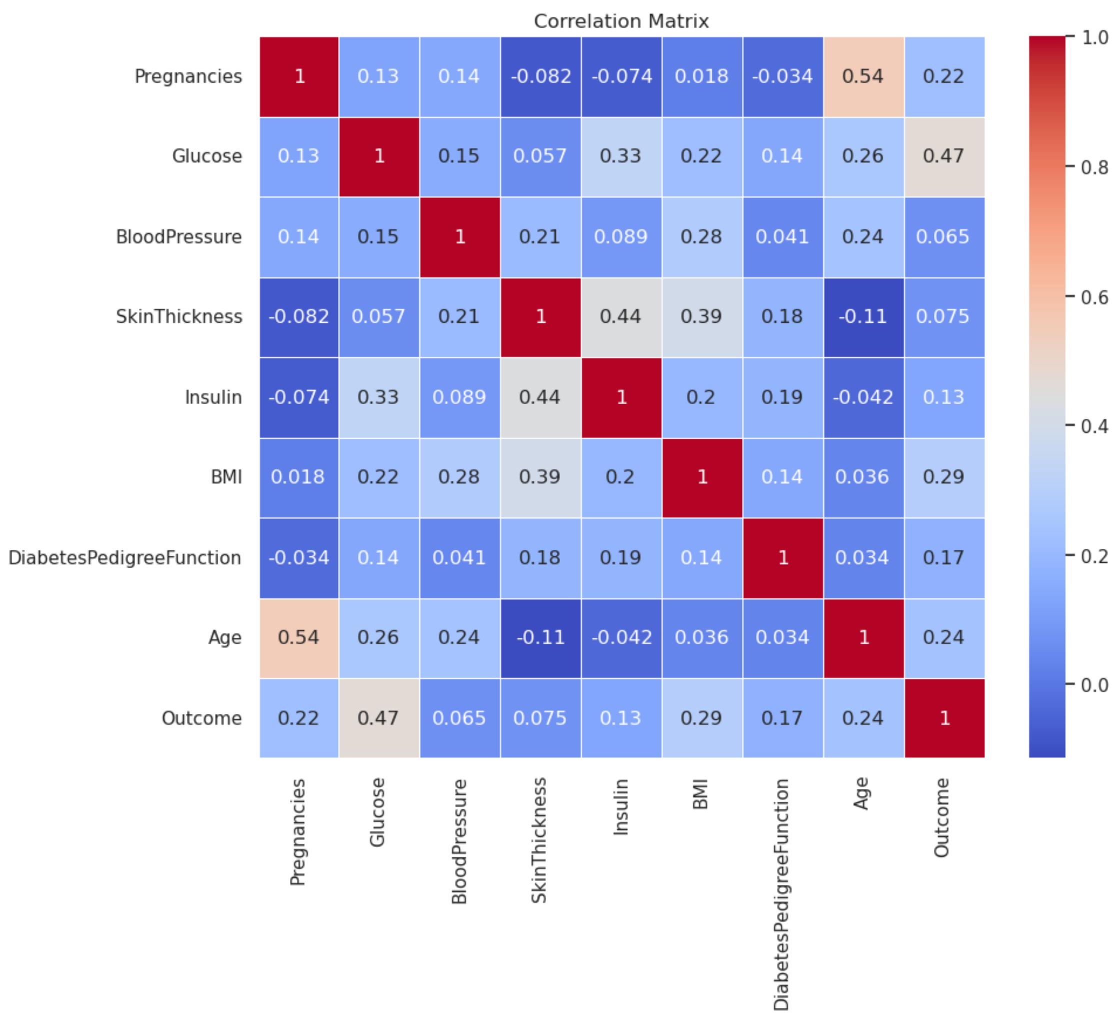
[113]:
```python
#Filter the data frame so that only outliers are present
outliers_data = df[outliers]
#rows with outliers
print(outliers_data)
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
4              0      137             40             35      168  43.1
7             10      115              0              0        0  35.3
8              2      197             70             45      543  30.5
9              8      125             96              0        0   0.0
12            10      139             80              0        0  27.1
..           …        …            …              …        …    …
706           10      115              0              0        0   0.0
707            2      127             46             21      335  34.4
710            3      158             64             13      387  31.2
715            7      187             50             33      392  33.9
753            0      181             88             44      510  43.3

     DiabetesPedigreeFunction  Age  Outcome
4                       2.288   33        1
7                       0.134   29        0
8                       0.158   53        1
9                       0.232   54        1
12                      1.441   57        0
..                        …    …      …
706                     0.261   30        1
707                     0.176   22        0
710                     0.295   24        0
715                     0.826   34        1
753                     0.222   26        1

[129 rows x 9 columns]
```

[114]:
```python
#Visualize Outliers
plt.figure(figsize=(15,8))
sns.set(style = "whitegrid")
sns.boxplot(data = df[outliers], orient = "h", palette = "Set2")
plt.title("Outliers Visualization for Diabetes Dataset")
plt.show()
```



[115]:
```
#correlation matrix
correlation_matrix = df.corr()
#set up matplotlib
plt.figure(figsize=(10,8))
#create a heatmap of the correlation matrix
sns.heatmap(correlation_matrix,annot = True,cmap = 'coolwarm',linewidth=.5)
plt.title('Correlation Matrix')
plt.show()
```

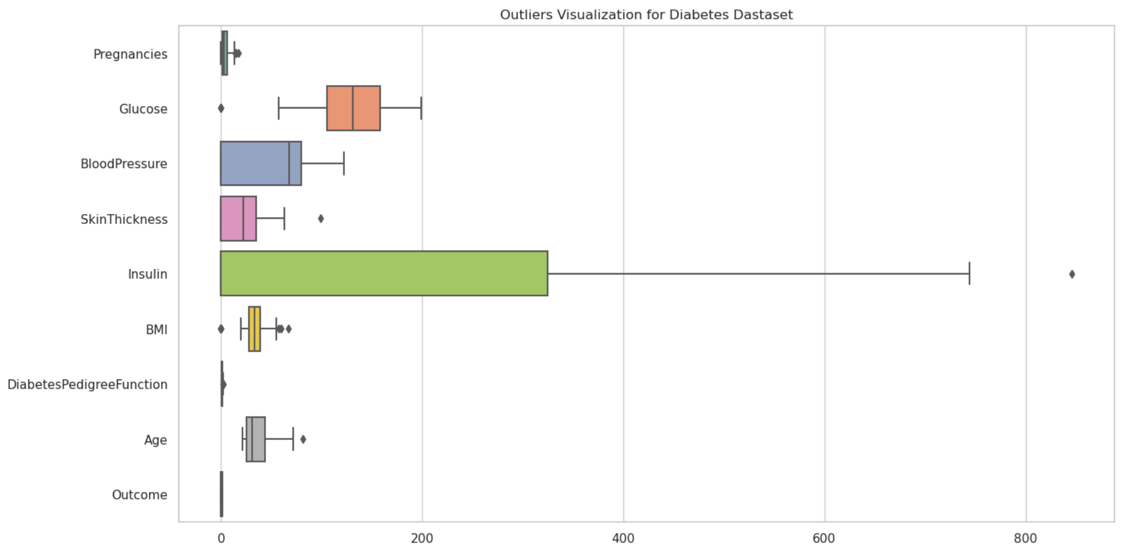


```
[116]: correlation = df.corr()
       print(correlation)
```

```
                          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
```

[117]:
```
#DF is Dataframe
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3-Q1
outliers = ((df < (Q1-1.5*IQR)) | (df > (Q3+1.5*IQR))).any(axis = 1)
#visualize outliers for each feature using box plots
plt.figure(figsize = (15,8))
sns.set(style = "whitegrid")
sns.boxplot(data = df[outliers], orient = "h", palette = "Set2")
plt.title("Outliers Visualization for Diabetes Dastaset")
plt.show()
```



[118]:
```
df.info
```

[118]:
```
<bound method DataFrame.info of      Pregnancies  Glucose  BloodPressure
SkinThickness  Insulin   BMI  \
0              6      148             72             35        0  33.6
1              1       85             66             29        0  26.6
2              8      183             64              0        0  23.3
3              1       89             66             23       94  28.1
4              0      137             40             35      168  43.1
..           …        …              …              …      …    …
763           10      101             76             48      180  32.9
764            2      122             70             27        0  36.8
765            5      121             72             23      112  26.2
766            1      126             60              0        0  30.1
767            1       93             70             31        0  30.4

     DiabetesPedigreeFunction  Age  Outcome
0                       0.627   50        1
1                       0.351   31        0
2                       0.672   32        1
3                       0.167   21        0
4                       2.288   33        1
..                        …   …      …
763                     0.171   63        0
764                     0.340   27        0
765                     0.245   30        0
766                     0.349   47        1
767                     0.315   23        0

[768 rows x 9 columns]>
```

[119]:
```
df_copy = df.copy(deep = True)
df_copy[["Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI"]]  =␣
 ↪df_copy[["Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI"]]
```

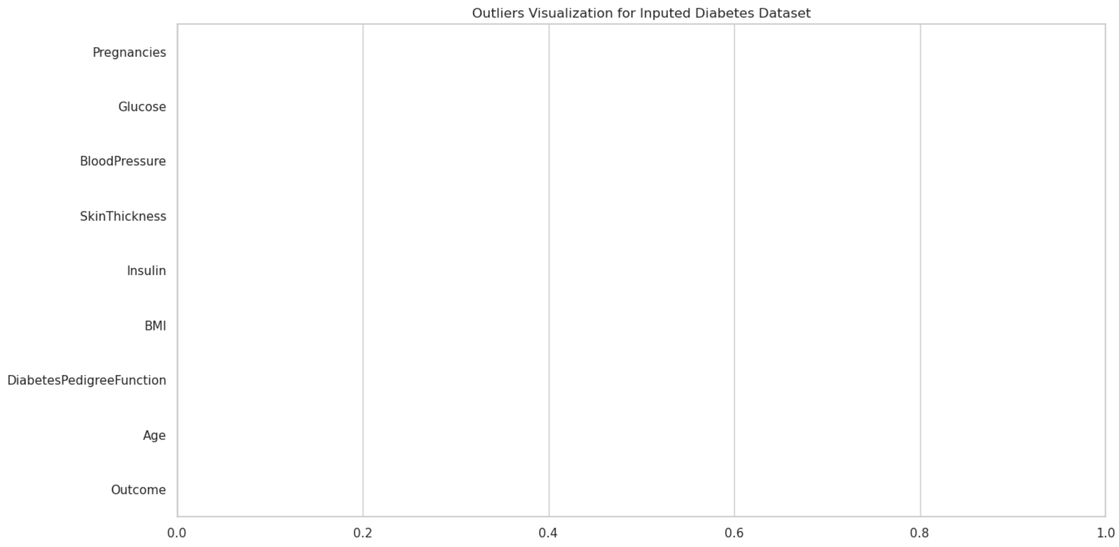
[120]:
```
#Describe diabetes
print(df.describe())
#know more of dataset with transpose
print(df.describe().T)
#Checks for null values
df.isnull()
#finding total null values
df.isnull().sum()
```

```
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   768.000000  768.000000     768.000000     768.000000  768.000000
mean      3.845052  120.894531      69.105469      20.536458   79.799479
std       3.369578   31.972618      19.355807      15.952218  115.244002
min       0.000000    0.000000       0.000000       0.000000    0.000000
25%       1.000000   99.000000      62.000000       0.000000    0.000000
50%       3.000000  117.000000      72.000000      23.000000   30.500000
75%       6.000000  140.250000      80.000000      32.000000  127.250000
max      17.000000  199.000000     122.000000      99.000000  846.000000

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  768.000000                768.000000  768.000000  768.000000
mean    31.992578                  0.471876   33.240885    0.348958
std      7.884160                  0.331329   11.760232    0.476951
min      0.000000                  0.078000   21.000000    0.000000
25%     27.300000                  0.243750   24.000000    0.000000
50%     32.000000                  0.372500   29.000000    0.000000
75%     36.600000                  0.626250   41.000000    1.000000
max     67.100000                  2.420000   81.000000    1.000000
                          count        mean         std     min       25%  \
Pregnancies               768.0    3.845052    3.369578   0.000   1.00000
Glucose                   768.0  120.894531   31.972618   0.000  99.00000
BloodPressure             768.0   69.105469   19.355807   0.000  62.00000
SkinThickness             768.0   20.536458   15.952218   0.000   0.00000
Insulin                   768.0   79.799479  115.244002   0.000   0.00000
BMI                       768.0   31.992578    7.884160   0.000  27.30000
DiabetesPedigreeFunction  768.0    0.471876    0.331329   0.078   0.24375
Age                       768.0   33.240885   11.760232  21.000  24.00000
Outcome                   768.0    0.348958    0.476951   0.000   0.00000

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

[120]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
```

```
Age                         0
Outcome                     0
dtype: int64
```

[121]:
```
Q1_inputed = data_inputed.quantile(0.25)
Q3_inputed = data_inputed.quantile(0.75)
IQR_inputed = Q3_inputed-Q1_inputed
outliers_inputed = ((data_inputed < (Q1_inputed-1.5*IQR))|␣
 ↪data_inputed>(Q3_inputed+1.5*IQR))
plt.figure(figsize=(15,8))
sns.set(style = "whitegrid")
sns.boxplot(data=data_inputed[outliers_inputed], orient ="h", Pazlette = "Set2")
plt.title("Outliers Visualization for Inputed Diabetes Dataset")
plt.show()
print(data_inputed[outliers_inputed])
```



```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin  BMI  \
0            NaN      NaN            NaN            NaN      NaN  NaN
1            NaN      NaN            NaN            NaN      NaN  NaN
2            NaN      NaN            NaN            NaN      NaN  NaN
3            NaN      NaN            NaN            NaN      NaN  NaN
4            NaN      NaN            NaN            NaN      NaN  NaN
..             …        …              …              …        …    …
763          NaN      NaN            NaN            NaN      NaN  NaN
764          NaN      NaN            NaN            NaN      NaN  NaN
765          NaN      NaN            NaN            NaN      NaN  NaN
766          NaN      NaN            NaN            NaN      NaN  NaN
767          NaN      NaN            NaN            NaN      NaN  NaN
```

```
     DiabetesPedigreeFunction  Age  Outcome
0                         NaN  NaN      NaN
1                         NaN  NaN      NaN
2                         NaN  NaN      NaN
3                         NaN  NaN      NaN
4                         NaN  NaN      NaN
..                        …   …       …
763                       NaN  NaN      NaN
764                       NaN  NaN      NaN
765                       NaN  NaN      NaN
766                       NaN  NaN      NaN
767                       NaN  NaN      NaN

[768 rows x 9 columns]
```

[122]:
```
#assuming df would be dataframe
class_distribution = df['Outcome'].value_counts()
print(class_distribution)
imbalance_ratio = 268/500
print(f"Imbalance Ratio: {imbalance_ratio:.2f}")
```

```
Outcome
0    500
1    268
Name: count, dtype: int64
Imbalance Ratio: 0.54
```

[123]:
```
#Split - Train(80%) and Test(20%)
#In X all the independent variables are stored
#In Y the predictor variable("OUTCOME") is stored.
#Train-test split is a technique used in machine learning to assess model␣
 ↪performance. It divides the dataset into a training set and a testing set,
#with a 0.2 test size indicating that
#20% of the data is used for testing and 80% for training.

X = df.drop ("Outcome", axis = 1)
Y = df['Outcome']
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.2)


# Apply SMOTE to the training set.
#SMOTE method from the imbalanced-learn library is used to perform synthetic␣
 ↪oversampling on the training set.
#This ensures that the model is trained on a more balanced dataset, reducing␣
 ↪the risk of bias and improving its ability to generalize to both classes.
smote = SMOTE(random_state=42)
X_resampled, Y_resampled = smote.fit_resample(X_train, Y_train)
```

```
#logistic Regression
model = LogisticRegression()
#model.fit(X_train,Y_train)
model.fit(X_resampled, Y_resampled)
```

[123]:
```
LogisticRegression()
```

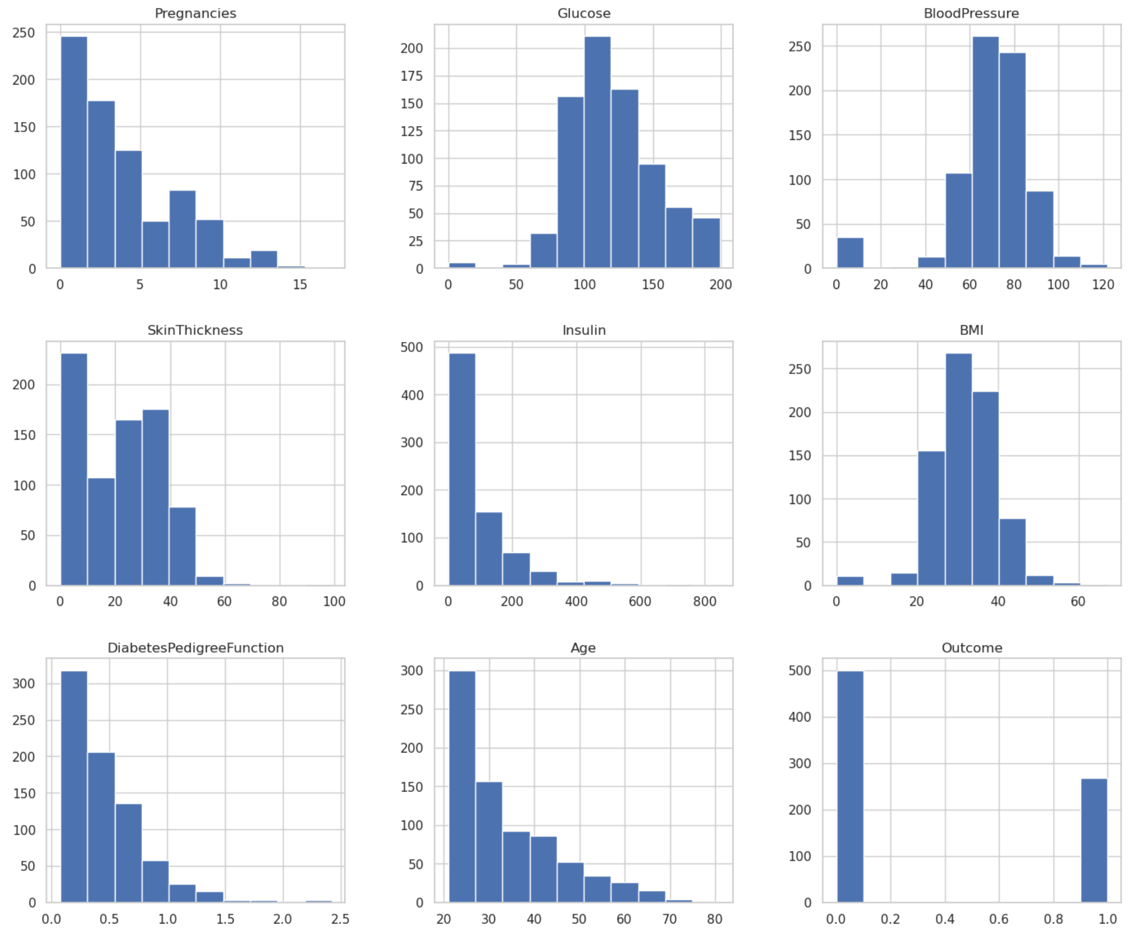
[124]:
```
#Total No. of Column
print(df.columns)
#info of the dataset
df.describe().T
```

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

[124]:
```
                          count        mean         std     min       25%  \
Pregnancies               768.0    3.845052    3.369578   0.000   1.00000
Glucose                   768.0  120.894531   31.972618   0.000  99.00000
BloodPressure             768.0   69.105469   19.355807   0.000  62.00000
SkinThickness             768.0   20.536458   15.952218   0.000   0.00000
Insulin                   768.0   79.799479  115.244002   0.000   0.00000
BMI                       768.0   31.992578    7.884160   0.000  27.30000
DiabetesPedigreeFunction  768.0    0.471876    0.331329   0.078   0.24375
Age                       768.0   33.240885   11.760232  21.000  24.00000
Outcome                   768.0    0.348958    0.476951   0.000   0.00000

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

[125]:
```
#inputing outliers with median
data_inputed = df.copy()
for col in df.columns:
    lower_bound = Q1[col] - 1.5*IQR[col]
    upper_bound = Q3[col] +1.5 *IQR[col]
    median_value = data_inputed[col].median()
    data_inputed[col] = np.where((data_inputed[col] < lower_bound) |␣
↪(data_inputed[col] > upper_bound), median_value, data_inputed[col])
```

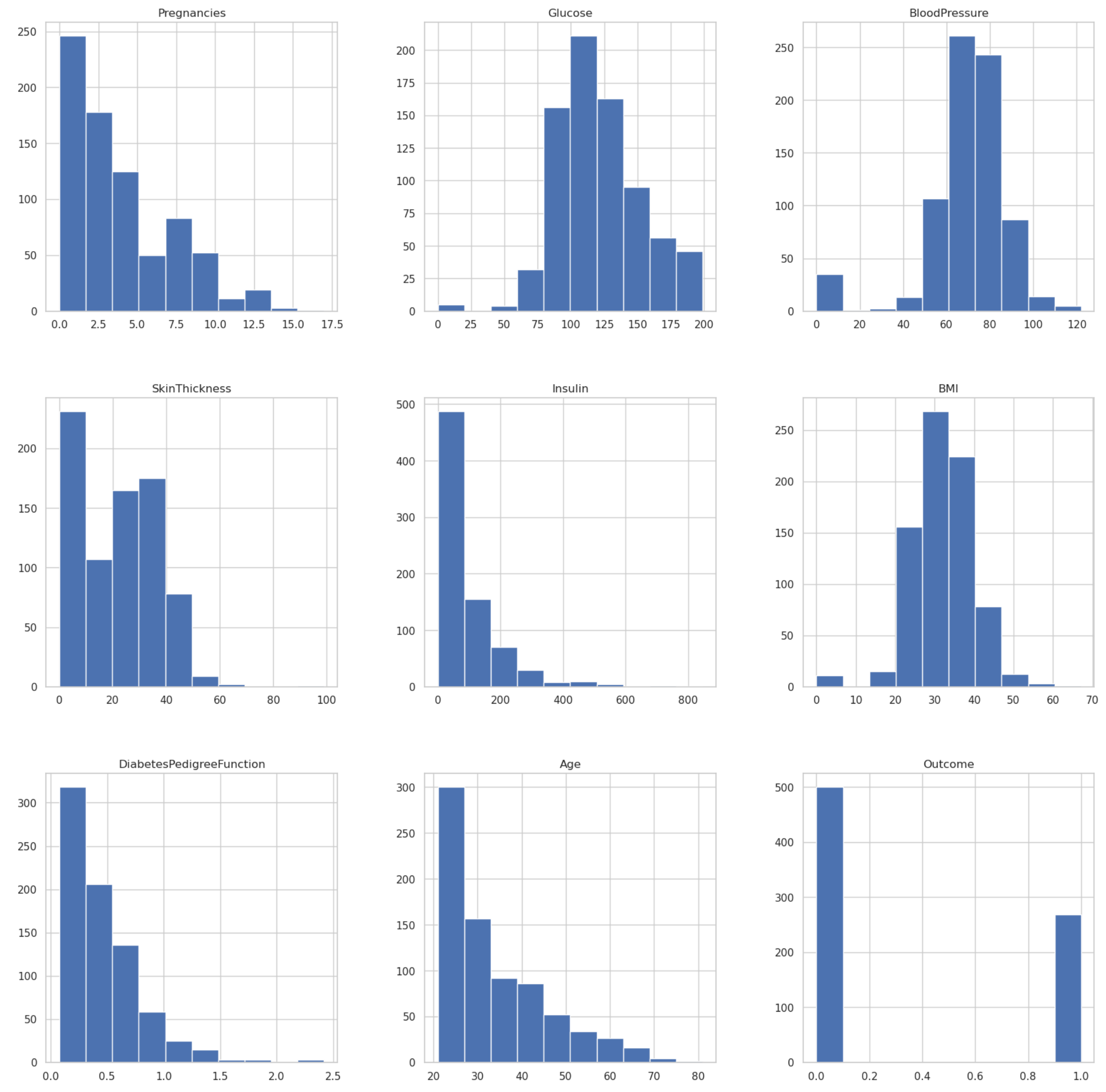
[126]:
```
df.isnull().sum()
df.hist(figsize=(17,14))
plt.show()
#plot data distribution points
```



[127]:
```
#aiming to input nan values for the column in accordance
df_copy['Glucose'].fillna(df_copy['Glucose'].mean(),inplace = True)
df_copy['BloodPressure'].fillna(df_copy['BloodPressure'].mean(),inplace = True)
df_copy['SkinThickness'].fillna(df_copy['SkinThickness'].mean(),inplace = True)
df_copy['Insulin'].fillna(df_copy['Insulin'].mean(),inplace = True)
df_copy['BMI'].fillna(df_copy['BMI'].mean(),inplace = True)
```

[128]:
```
#Plotting distribution
df_copy.isnull().sum()
df_copy.hist(figsize=(20,20))
```

```
plt.show()
```



[129]:
```
prediction = model.predict(X_test)
print(prediction)
```

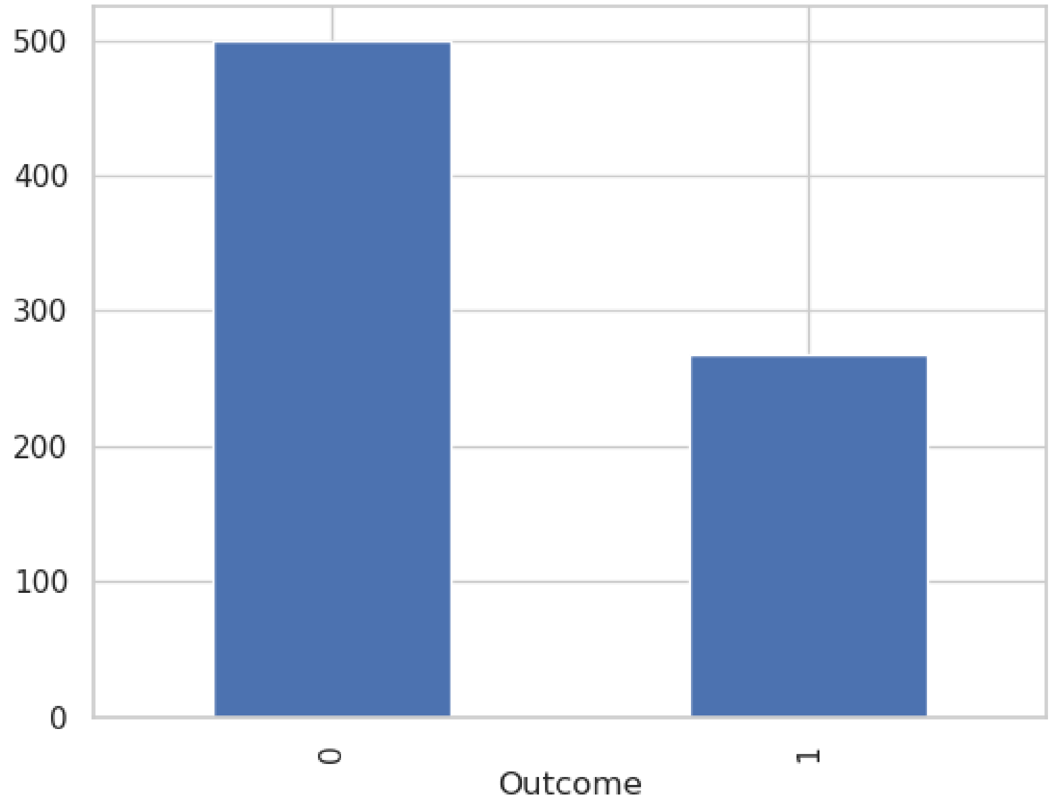
```
[0 1 0 1 1 1 1 1 0 1 1 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0
 1 0 0 0 0 0 1 1 1 0 0 0 1 0 0 1 1 0 1 0 1 0 0 0 1 1 1 0 1 1 0 0 0 0 0 1 1
 1 1 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 1 0 1 0 1 0 0 0 1 1 0 0 1 0 0 0 1 0 1 1
 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 1 1 0 0 0 0 0 0 0 0 1 1 1
 0 0 1 1 1 0]
```

[130]:
```
accuracy = accuracy_score(prediction,Y_test)
print(accuracy)
```

```
0.7857142857142857
```

[131]:
```
#plot counts of outcome checking balance
colorwheels = {1: "#0392cf",2: "#7bc043"}
color = df["Outcome"].map(lambda x:colorwheels.get(x+1))
print(df['Outcome'].value_counts())
p = df['Outcome'].value_counts().plot(kind = "bar")
```

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

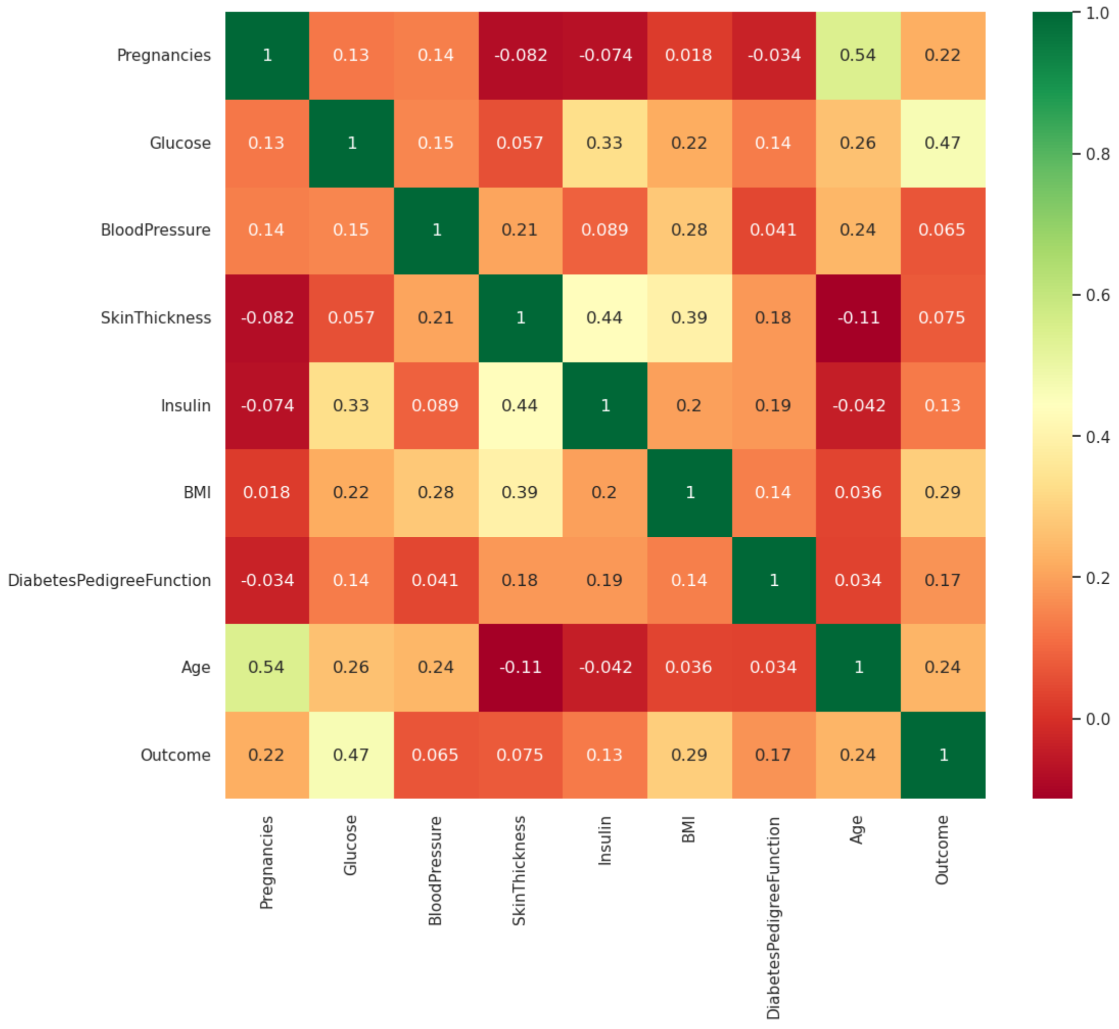


[132]:
```
# Which feature has the highest importance according to our logistic regression␣
↪mod
# Assuming 'model' is your trained logistic regression model
feature_important=model.coef_[0]

# Create a DataFrame to associate feature names with their importance values
importance_df=pd.DataFrame({'Feature': df.columns[:-1], 'Importance': np.
↪abs(feature_important)})
```

```
# Sort the DataFrame by absolute importance values in descending order
importance_df=importance_df.sort_values(by='Importance', ascending=False)
print(importance_df) #displays the sorted dataframe
```

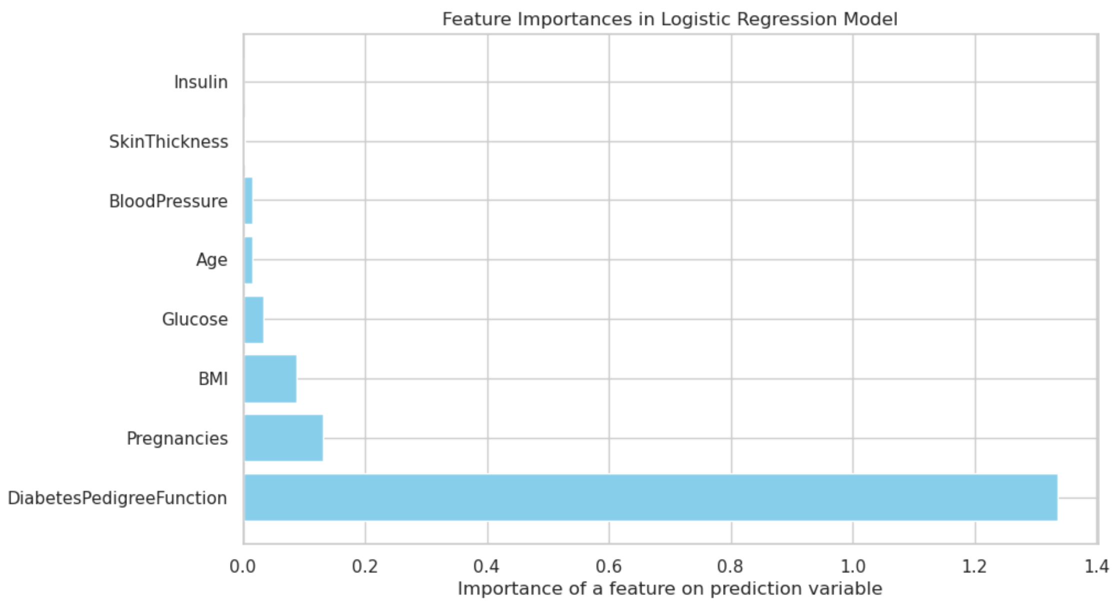
```
                    Feature  Importance
6  DiabetesPedigreeFunction    1.335583
0               Pregnancies    0.131790
5                       BMI    0.087122
1                   Glucose    0.033596
7                       Age    0.014949
2             BloodPressure    0.014481
3             SkinThickness    0.002316
4                   Insulin    0.001243
```

[133]:
```
plt.figure(figsize=(12,10)) #heatmap
p = sns.heatmap(df.corr(),annot = True,cmap = "RdYlGn")
```

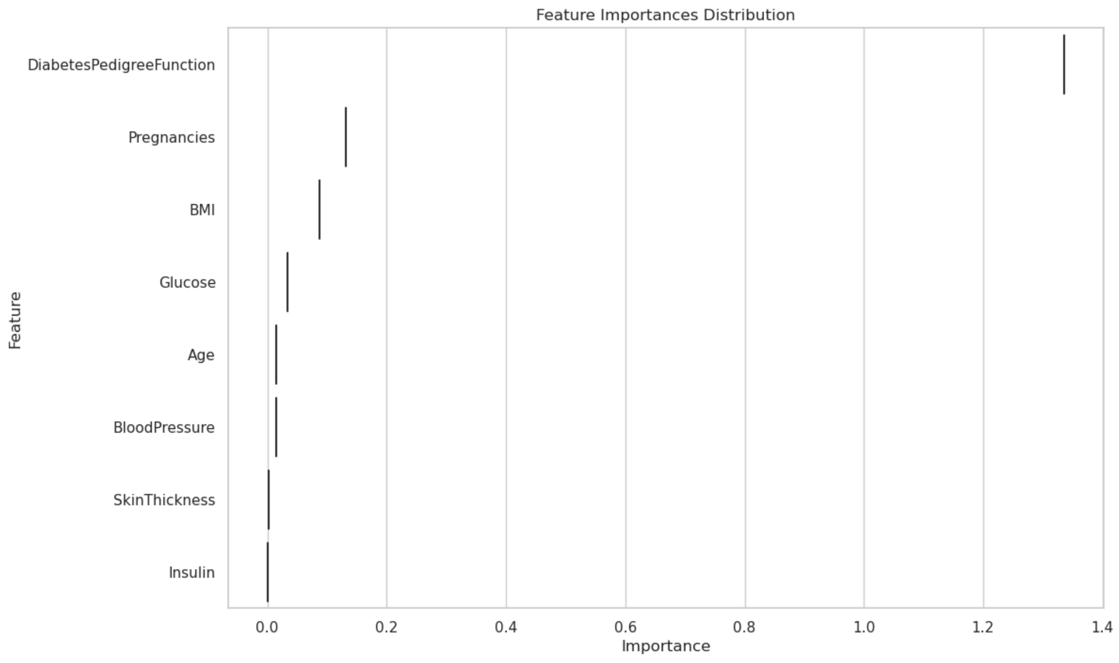


	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	1	0.13	0.14	-0.082	-0.074	0.018	-0.034	0.54	0.22
Glucose	0.13	1	0.15	0.057	0.33	0.22	0.14	0.26	0.47
BloodPressure	0.14	0.15	1	0.21	0.089	0.28	0.041	0.24	0.065
SkinThickness	-0.082	0.057	0.21	1	0.44	0.39	0.18	-0.11	0.075
Insulin	-0.074	0.33	0.089	0.44	1	0.2	0.19	-0.042	0.13
BMI	0.018	0.22	0.28	0.39	0.2	1	0.14	0.036	0.29
DiabetesPedigreeFunction	-0.034	0.14	0.041	0.18	0.19	0.14	1	0.034	0.17
Age	0.54	0.26	0.24	-0.11	-0.042	0.036	0.034	1	0.24
Outcome	0.22	0.47	0.065	0.075	0.13	0.29	0.17	0.24	1

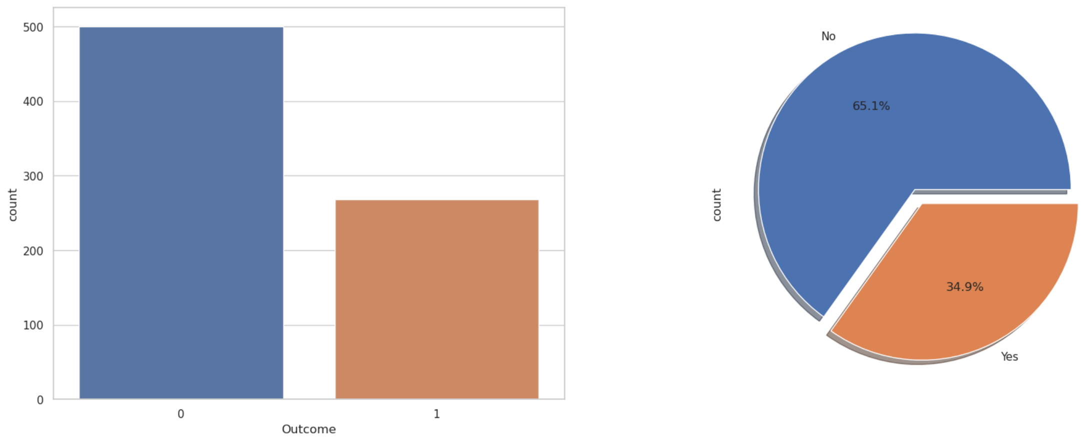
[134]:
```
import matplotlib.pyplot as plt
# Sort the DataFrame by absolute importance values in descending order
importance_df = importance_df.sort_values(by='Importance', ascending=False)
# Plotting the bar chart
plt.figure(figsize=(10, 6))
plt.barh(importance_df['Feature'], importance_df['Importance'], color='skyblue')
plt.xlabel('Importance of a feature on prediction variable')
plt.title('Feature Importances in Logistic Regression Model')
plt.show()
```



[135]:
```
import seaborn as sns
# Create a violin plot
plt.figure(figsize=(12, 8))
sns.violinplot(x='Importance', y='Feature', data=importance_df,␣
 ↪palette='viridis')
plt.title('Feature Importances Distribution')
plt.show()
```

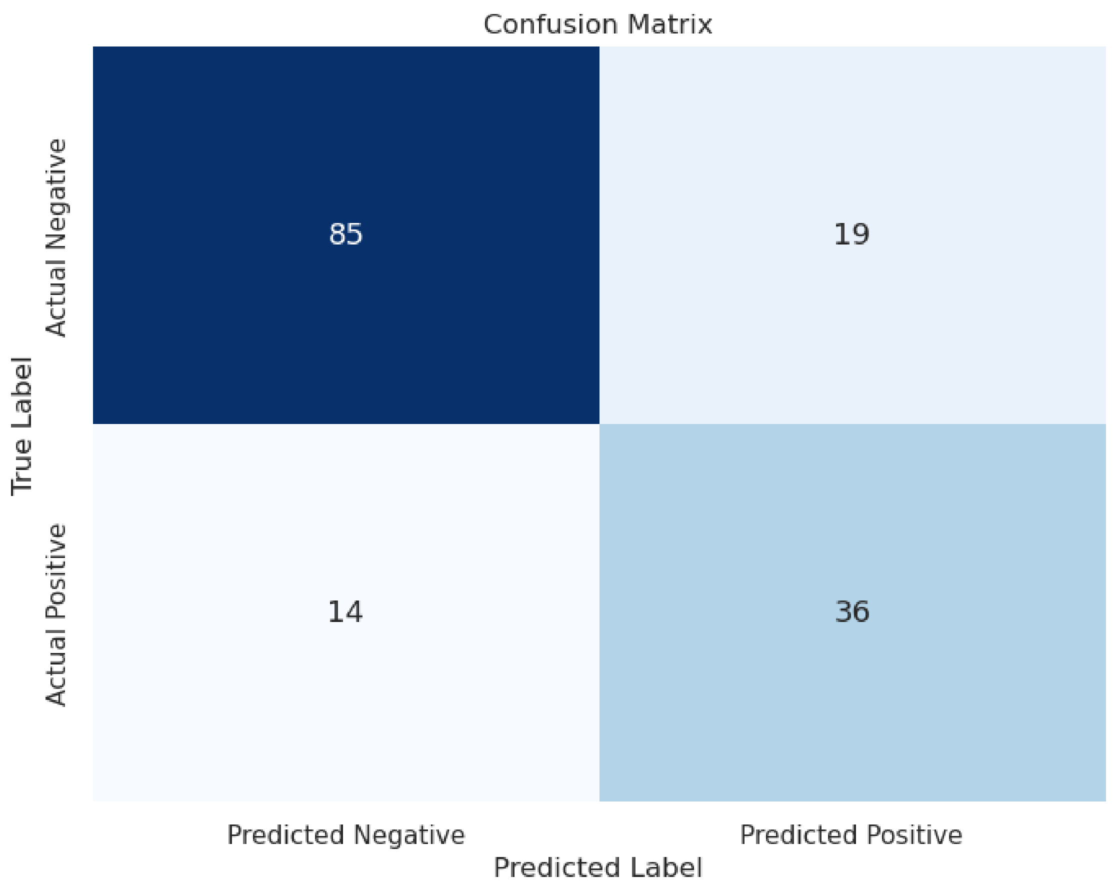


[136]:
```
fig, ax = plt.subplots(1,2, figsize=(20,7)) #barplot, and pie chart for the␣
↪outcome
sns.countplot(data=df, x = "Outcome",ax = ax[0])
df["Outcome"].value_counts().plot.pie(explode = [0.1,0], autopct="%1.1f%%",␣
↪labels = ["No","Yes"],shadow = True, ax = ax[1])
plt.show()
```

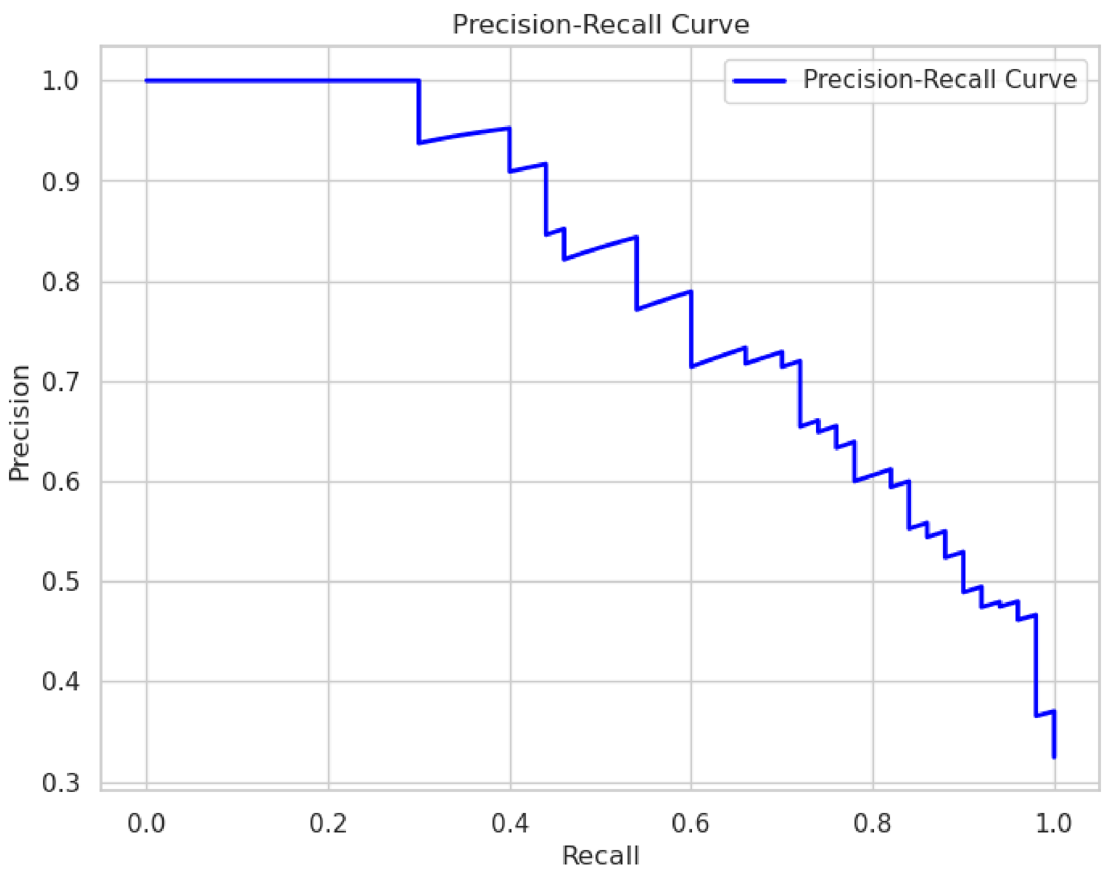


[137]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
```

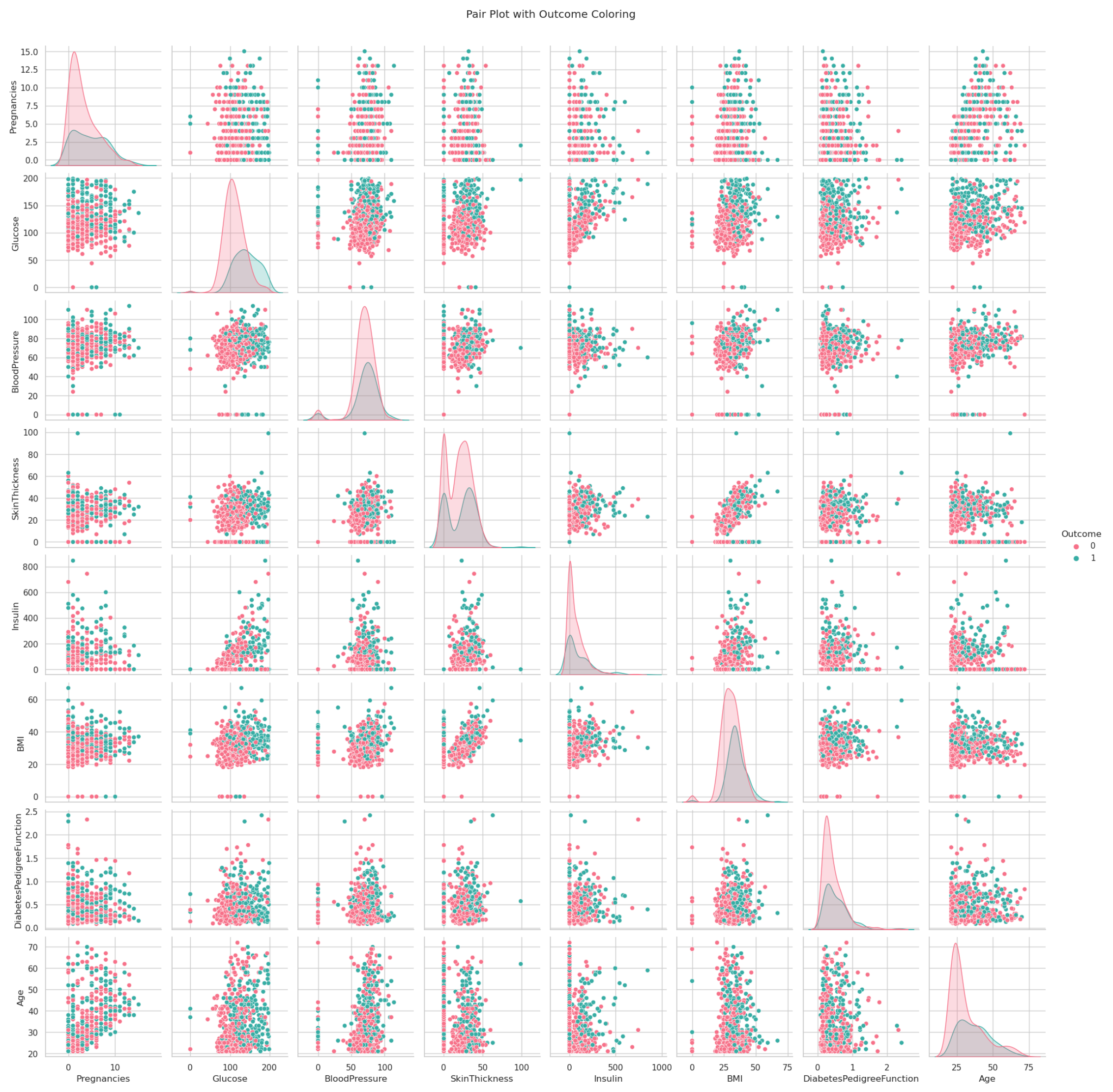
```
# Assuming 'model' is your trained logistic regression model
predictions = model.predict(X_test)

# Create a confusion matrix
conf_matrix = confusion_matrix(Y_test, predictions)

# Define labels for the confusion matrix

labels = [['True Negative', 'False Positive'], ['False Negative', 'True␣
 ↪Positive']]

# Plot the confusion matrix with actual numbers and annotations


plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            annot_kws={"size": 13}, # Setting annotation size
            xticklabels=['Predicted Negative', 'Predicted Positive'],
            yticklabels=['Actual Negative', 'Actual Positive'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```



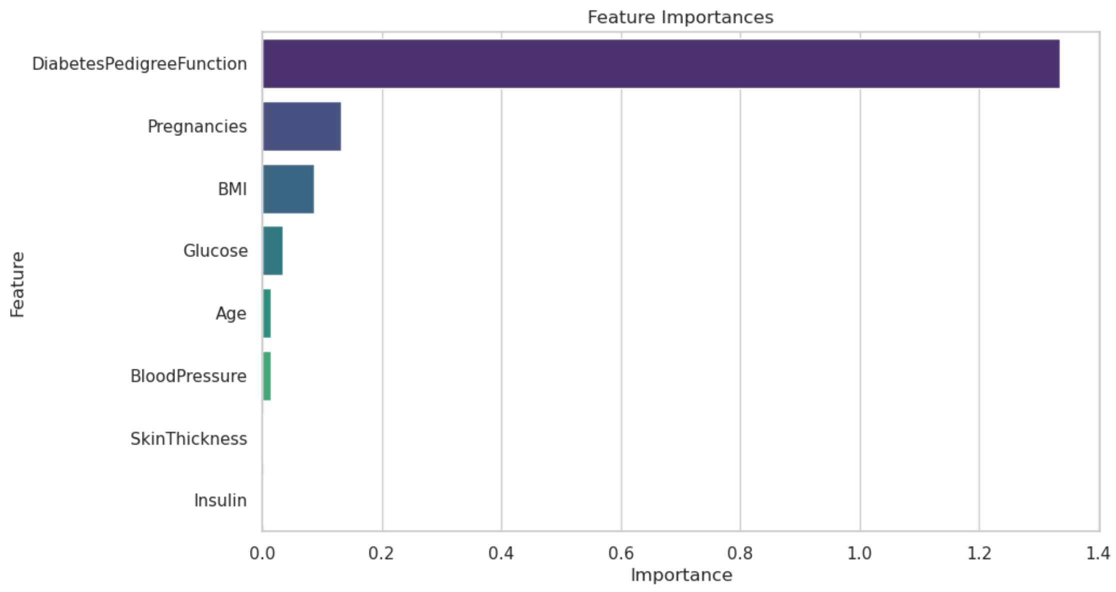
[138]:
```python
#Precision recall curve
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import auc

# Assuming 'model' is your trained logistic regression model
probas = model.predict_proba(X_test)
precision, recall, thresholds = precision_recall_curve(Y_test, probas[:, 1])

# Plot Precision-Recall curve
plt.figure(figsize=(8, 6))
plt.plot(recall, precision, color='blue', lw=2, label='Precision-Recall Curve')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend()
plt.show()
```



[139]:
```
# Combine the features and outcome into one DataFrame
pair_plot_data = pd.concat([X_train, Y_train], axis=1)
# Plot pair plot with outcome coloring
sns.pairplot(pair_plot_data, hue='Outcome', diag_kind='kde', palette='husl')
plt.suptitle('Pair Plot with Outcome Coloring', y=1.02)
plt.show()
```



```
[140]: #Features importance
       #This assumes model is trained logistic regression model
       feature_importance = model.coef_[0]
       # Create a DataFrame for feature importance
       importance_df = pd.DataFrame({'Feature': X_train.columns, 'Importance': np.
        ↪abs(feature_importance)})
       # Sort the DataFrame by importance values
       importance_df = importance_df.sort_values(by='Importance', ascending=False)
       # Plot the bar chart
       plt.figure(figsize=(10, 6))
       sns.barplot(x='Importance', y='Feature', data=importance_df, palette='viridis')
       plt.xlabel('Importance')
       plt.ylabel('Feature')
       plt.title('Feature Importances')
       plt.show()
```



[141]:
```
from sklearn.metrics import f1_score
# Assuming 'Y_test' is your true labels and 'predictions' is your predicted␣
↪labels
# Make sure 'predictions' are obtained from your model
# Calculate F1 score
f1 = f1_score(Y_test, prediction)
print(f"F1 Score: {f1:.4f}")
```

```
F1 Score: 0.6857
```

[142]:
```
#KNN
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report

# Split the data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,␣
↪random_state=42)

# Standardize features (important for KNN)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
# K-Nearest Neighbors
knn_model = KNeighborsClassifier(n_neighbors=5) # You can adjust the number of␣
↪nei
knn_model.fit(X_train_scaled, Y_train)

# Make predictions on the test set
knn_predictions = knn_model.predict(X_test_scaled)

# Evaluate the model
print("K-Nearest Neighbors Model:")
print("Accuracy:", accuracy_score(Y_test, knn_predictions))
print("Classification Report:\n", classification_report(Y_test,␣
↪knn_predictions))
```

```
K-Nearest Neighbors Model:
Accuracy: 0.6948051948051948
Classification Report:
               precision    recall  f1-score   support

           0       0.75      0.80      0.77        99
           1       0.58      0.51      0.54        55

    accuracy                           0.69       154
   macro avg       0.66      0.65      0.66       154
weighted avg       0.69      0.69      0.69       154
```

[143]:
```
#KNN using Hyper parameter
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
# Define the parameter grid
param_grid = {'n_neighbors': [3, 5, 7, 9, 11, 13, 15, 17, 19, 21]}
# Initialize the KNN model
knn_model = KNeighborsClassifier()
# Perform GridSearchCV
grid_search = GridSearchCV(knn_model, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train_scaled, Y_train)
# Get the best hyperparameters
best_n_neighbors = grid_search.best_params_['n_neighbors']
# Train the model with the best hyperparameters
best_knn_model = KNeighborsClassifier(n_neighbors=best_n_neighbors)
best_knn_model.fit(X_train_scaled, Y_train)
```

[143]:
```
KNeighborsClassifier(n_neighbors=11)
```

[144]:
```python
#Fitting the KNN model where n =11 so that our model is more stable for the new␣
  ↪dat
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
# Assuming X and Y are your features and labels
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,␣
  ↪random_state=42)
# Standardize the features (you can use your own preprocessing steps)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
# Create and fit the KNN model
knn_model = KNeighborsClassifier(n_neighbors=11)
knn_model.fit(X_train_scaled, Y_train)
# Make predictions on the test set
predictions = knn_model.predict(X_test_scaled)
# Now you can evaluate the accuracy and other metrics
from sklearn.metrics import accuracy_score, classification_report,␣
  ↪confusion_matrix
# Calculate accuracy
accuracy = accuracy_score(Y_test, predictions)
# Display classification report
print("Classification Report:")
print(classification_report(Y_test, predictions))
# Display confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(Y_test, predictions))
# Display accuracy
print(f"Accuracy: {accuracy:.4f}")
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.79      0.78        99
           1       0.60      0.56      0.58        55

    accuracy                           0.71       154
   macro avg       0.68      0.68      0.68       154
weighted avg       0.70      0.71      0.71       154

Confusion Matrix:
[[78 21]
 [24 31]]
Accuracy: 0.7078
```

[145]:
```
#We made a prediction after fitting
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix
# Assuming you have already trained your KNN model (replace knn_model with your␣
 ↪act
# Make predictions on the test set
predictions = knn_model.predict(X_test_scaled)
# Calculate accuracy
accuracy = accuracy_score(Y_test, predictions)
# Display classification report
print("Classification Report:")
print(classification_report(Y_test, predictions))
# Display confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(Y_test, predictions))
# Display accuracy
print(f"Accuracy: {accuracy:.4f}")
```

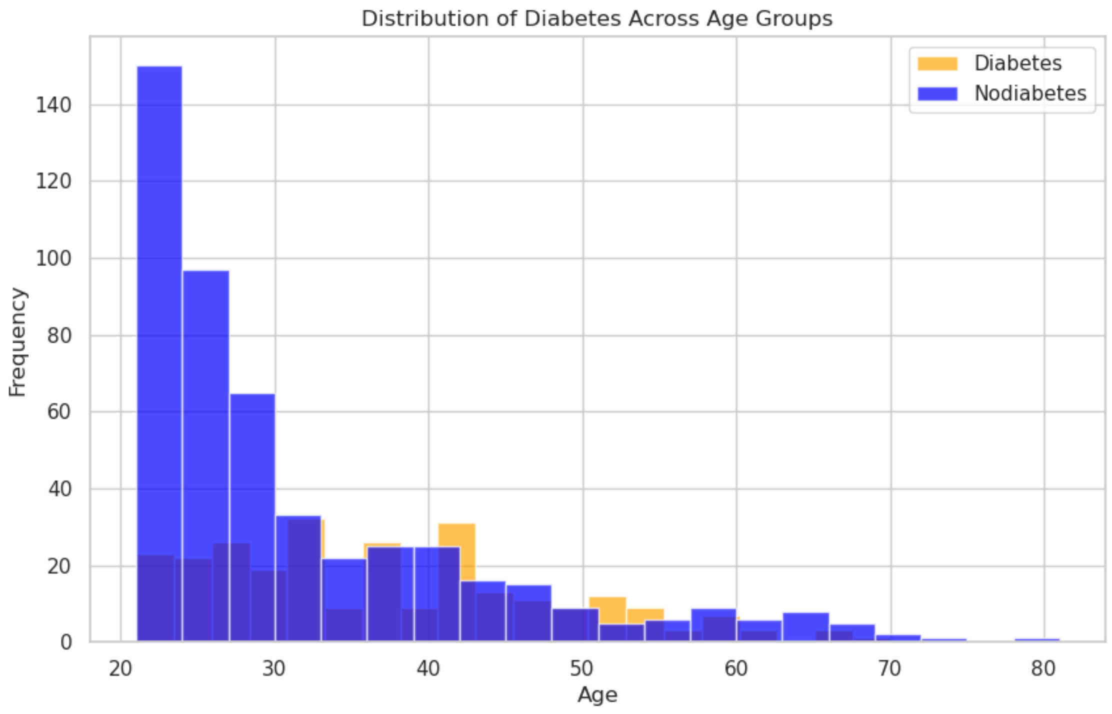
```
Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.79      0.78        99
           1       0.60      0.56      0.58        55

    accuracy                           0.71       154
   macro avg       0.68      0.68      0.68       154
weighted avg       0.70      0.71      0.71       154

Confusion Matrix:
[[78 21]
 [24 31]]
Accuracy: 0.7078
```

[146]:
```
# Get the best hyperparameters
best_params = grid_search.best_params_
# Access the best k value
best_k_value = best_params['n_neighbors']
print(f"The best k value selected by GridSearchCV is: {best_k_value}")
```

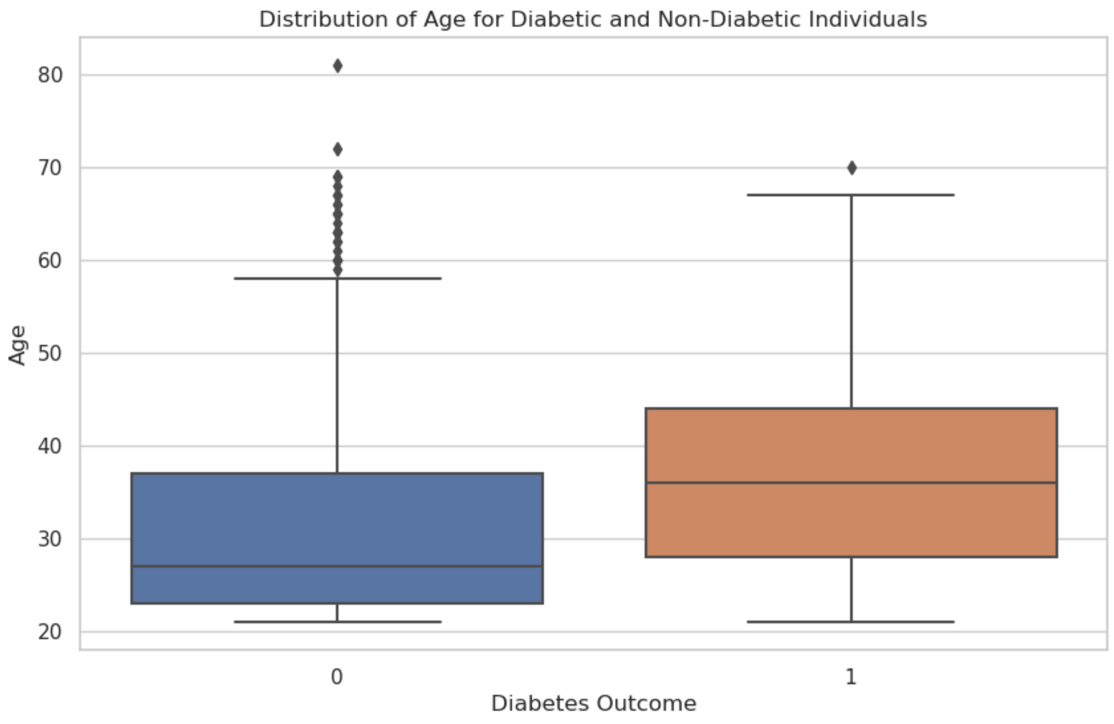
```
The best k value selected by GridSearchCV is: 11
```

[147]:
```
import matplotlib.pyplot as plt
# Assuming 'df' is your DataFrame and 'Age' is the column representing age
plt.figure(figsize=(10, 6))
plt.hist(df[df['Outcome'] == 1]['Age'], bins=20, color='orange', alpha=0.7,␣
 ↪label='Diabetes')
```

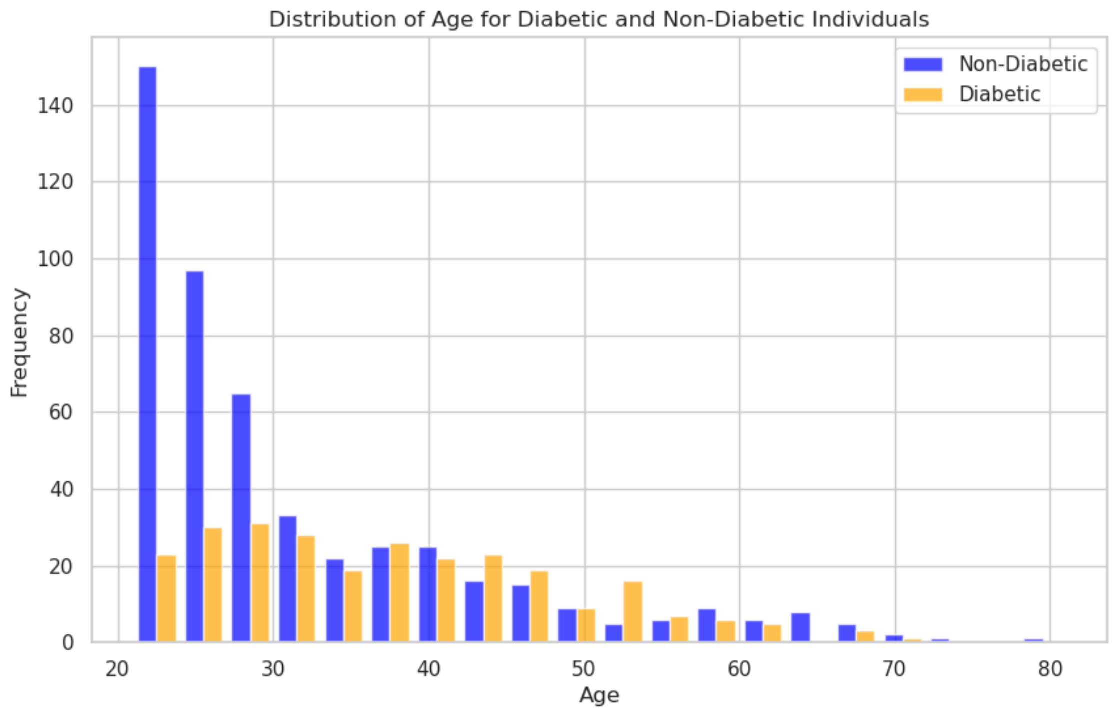
```
plt.hist(df[df['Outcome'] == 0]['Age'], bins=20, color='blue', alpha=0.7,␣
 ↪label='Nodiabetes')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.title('Distribution of Diabetes Across Age Groups')
plt.legend()
plt.show()
```



[148]:
```
import seaborn as sns
import matplotlib.pyplot as plt
# Assuming 'df' is your DataFrame and 'Age' is the column representing age
plt.figure(figsize=(10, 6))
sns.boxplot(x='Outcome', y='Age', data=df)
plt.xlabel('Diabetes Outcome')
plt.ylabel('Age')
plt.title('Distribution of Age for Diabetic and Non-Diabetic Individuals')
plt.show()
```

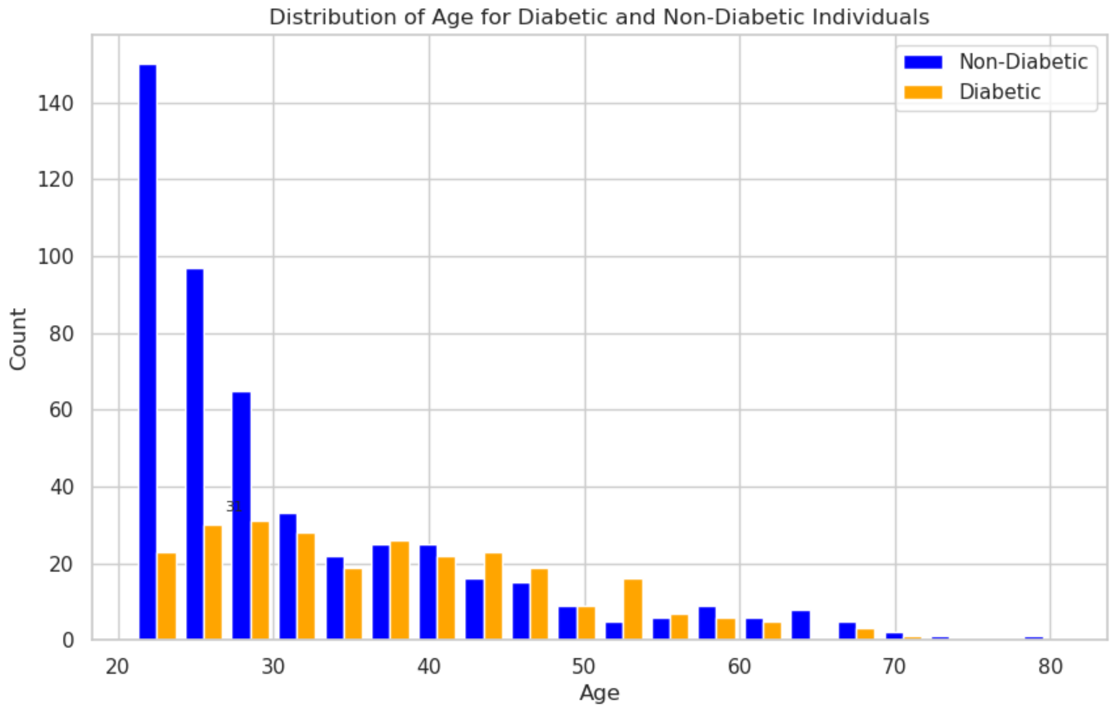


[153]:
```
import matplotlib.pyplot as plt
# Assuming 'df' is your DataFrame and 'Age' is the column representing age
plt.figure(figsize=(10, 6))
plt.hist([df[df['Outcome'] == 0]['Age'], df[df['Outcome'] == 1]['Age']],
 bins=20, color=['blue', 'orange'], label=['Non-Diabetic', 'Diabetic'], alpha=0.
 ↪7)
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.title('Distribution of Age for Diabetic and Non-Diabetic Individuals')
plt.legend()
plt.show()
```



[154]:
```
import matplotlib.pyplot as plt
# Assuming 'data' is your DataFrame
plt.figure(figsize=(10, 6))

# Create a histogram for the 'Age' column, grouped by 'Outcome' (Diabetic or␣
 ↪Not)
n, bins, patches = plt.hist([df[df['Outcome'] == 0]['Age'], df[df['Outcome'] ==␣
 ↪1]['Age']],
                            bins=20, color=['blue', 'orange'],␣
 ↪label=['Non-Diabetic', 'Diabetic'])

#Set labels and title
plt.xlabel('Age')
plt.ylabel('Count')
plt.title('Distribution of Age for Diabetic and Non-Diabetic Individuals')
plt.legend()

#Find the index of the bin with the highest diabetic cases
max_diabetic_bin = n[1].argmax()

#Display the number on the bar with the highest diabetic cases
plt.text(bins[max_diabetic_bin] + 0.5, n[1][max_diabetic_bin] + 2,
         int(n[1][max_diabetic_bin]), ha='center', va='bottom', fontsize=8)
```

```
#Show the plot
plt.show()
```



```
[ ]:
```