In [ ]:
```python
# Diabetes data science analysis project
```

Load Dataset

In [164…
```python
# Reading the dataset into a pandas DataFrame

df = pd.read_csv("C:/Users/zizhe/Desktop/Leo Zhao/Diabetes Prediction Project/diabetes_prediction_dataset.csv")

df.head() # view the first few rows of the DataFrame
```

Out[164]:

	Gender	Age	Hypertension	Heart_Disease	Smoking_History	BMI	HbA1c_Level	Blood_Glucose_Level	Diabetes
0	Female	80.0	0	1	never	25.19	6.6	140	0
1	Female	54.0	0	0	No Info	27.32	6.6	80	0
2	Male	28.0	0	0	never	27.32	5.7	158	0
3	Female	36.0	0	0	current	23.45	5.0	155	0
4	Male	76.0	1	1	current	20.14	4.8	155	0

Dimensions of the dataframe

In [165…
```python
# Checking the dimensions of the DataFrame 'df'
df.shape
```

Out[165]:
```
(100000, 9)
```

In [166…
```python
print('Number of observations :',df.shape[0])
print('Number of variables    :',df.shape[1])
```

```
Number of observations : 100000
Number of variables    : 9
```

Information about the schema

In [167…
```python
# Displaying the first few rows of the DataFrame 'df'
df.head(10)
```

Out[167]:

	Gender	Age	Hypertension	Heart_Disease	Smoking_History	BMI	HbA1c_Level	Blood_Glucose_Level	Diabetes
0	Female	80.0	0	1	never	25.19	6.6	140	0
1	Female	54.0	0	0	No Info	27.32	6.6	80	0
2	Male	28.0	0	0	never	27.32	5.7	158	0
3	Female	36.0	0	0	current	23.45	5.0	155	0
4	Male	76.0	1	1	current	20.14	4.8	155	0
5	Female	20.0	0	0	never	27.32	6.6	85	0
6	Female	44.0	0	0	never	19.31	6.5	200	1
7	Female	79.0	0	0	No Info	23.86	5.7	85	0
8	Male	42.0	0	0	never	33.64	4.8	145	0
9	Female	32.0	0	0	never	27.32	5.0	100	0

In [170…

```python
# Displaying the last few rows of the DataFrame 'df'
df.tail(10)
```

Out[170]:

	Gender	Age	Hypertension	Heart_Disease	Smoking_History	BMI	HbA1c_Level	Blood_Glucose_Level	Diabetes
99990	Male	39.0	0	0	No Info	27.32	6.1	100	0
99991	Male	22.0	0	0	current	29.65	6.0	80	0
99992	Female	26.0	0	0	never	34.34	6.5	160	0
99993	Female	40.0	0	0	never	40.69	3.5	155	0
99994	Female	36.0	0	0	No Info	24.60	4.8	145	0
99995	Female	80.0	0	0	No Info	27.32	6.2	90	0
99996	Female	2.0	0	0	No Info	17.37	6.5	100	0
99997	Male	66.0	0	0	former	27.83	5.7	155	0
99998	Female	24.0	0	0	never	35.42	4.0	100	0
99999	Female	57.0	0	0	current	22.43	6.6	90	0

In [171...
```
# Retrieving the column names of the DataFrame 'df'
df.columns
```

Out[171]:
```
Index(['Gender', 'Age', 'Hypertension', 'Heart_Disease', 'Smoking_History',
       'BMI', 'HbA1c_Level', 'Blood_Glucose_Level', 'Diabetes'],
      dtype='object')
```

In [172...
```
list(df.columns)
```

Out[172]:
```
['Gender',
 'Age',
 'Hypertension',
 'Heart_Disease',
 'Smoking_History',
 'BMI',
 'HbA1c_Level',
 'Blood_Glucose_Level',
 'Diabetes']
```

In [174...
```
# Unique Values in Given Columns
columns = ['Gender', 'Age', 'Hypertension', 'Heart_Disease', 'Smoking_History',
       'BMI', 'HbA1c_Level', 'Blood_Glucose_Level', 'Diabetes']

for col in columns:
    print(col, df[col].unique())
```

```
Gender ['Female' 'Male' 'Other']
Age [80.   54.   28.   36.   76.   20.   44.   79.   42.   32.   53.   78.
 67.   15.   37.   40.    5.   69.   72.    4.   30.   45.   43.   50.
 41.   26.   34.   73.   77.   66.   29.   60.   38.    3.   57.   74.
 19.   46.   21.   59.   27.   13.   56.    2.    7.   11.    6.   55.
  9.   62.   47.   12.   68.   75.   22.   58.   18.   24.   17.   25.
  0.08 33.   16.   61.   31.    8.   49.   39.   65.   14.   70.    0.56
 48.   51.   71.    0.88 64.   63.   52.    0.16 10.   35.   23.    0.64
  1.16  1.64  0.72  1.88  1.32  0.8   1.24  1.    1.8   0.48  1.56  1.08
  0.24  1.4   0.4   0.32  1.72  1.48]
Hypertension [0 1]
Heart_Disease [1 0]
Smoking_History ['never' 'No Info' 'current' 'former' 'ever' 'not current']
BMI [25.19 27.32 23.45 ... 59.42 44.39 60.52]
HbA1c_Level [6.6 5.7 5.  4.8 6.5 6.1 6.  5.8 3.5 6.2 4.  4.5 9.  7.  8.8 8.2 7.5 6.8]
Blood_Glucose_Level [140  80 158 155  85 200 145 100 130 160 126 159  90 260 220 300 280 240]
Diabetes [0 1]
```

In [175...
```
df[df['Gender'] == "Other"].shape
```

Out[175]: (18, 9)

In [176…
```
df[df['Gender'] == "Female"].shape
```
Out[176]: (58552, 9)

In [177…
```
df[df['Gender'] == "Male"].shape
```
Out[177]: (41430, 9)

In [178…
```
df[df['Smoking_History'] == "not current"].shape
```
Out[178]: (6447, 9)

In [179…
```
df[df['Smoking_History'] == "No Info"].shape
```
Out[179]: (35816, 9)

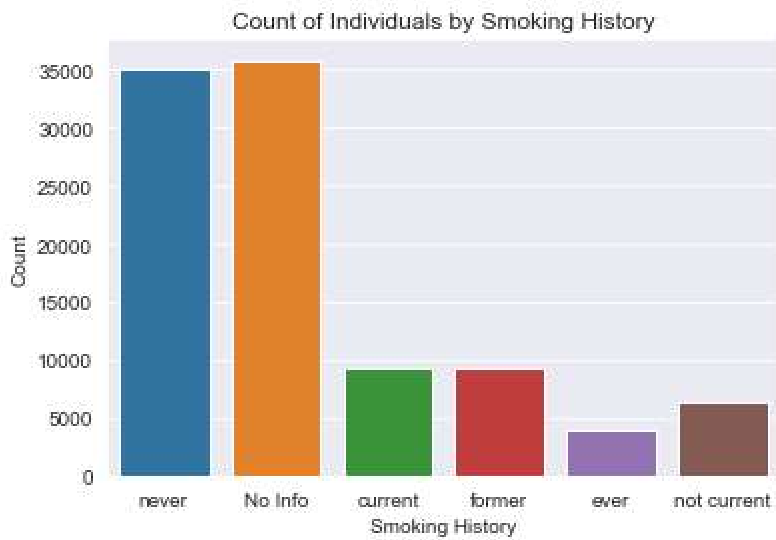
In [180…
```
df[df['Smoking_History'] == "never"].shape
```
Out[180]: (35095, 9)

In [181…
```
# Generating descriptive statistics for the DataFrame 'df'
df.describe()
```
Out[181]:

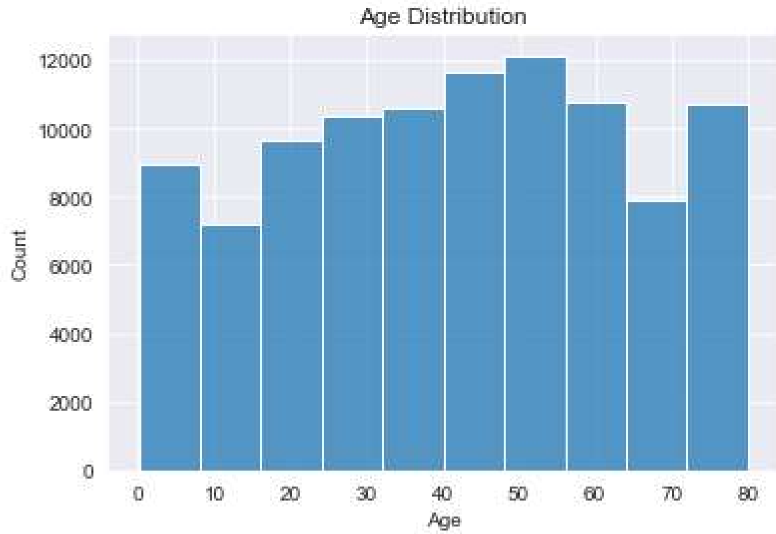
	Age	Hypertension	Heart_Disease	BMI	HbA1c_Level	Blood_Glucose_Level	Diabetes
count	100000.000000	100000.00000	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000
mean	41.885856	0.07485	0.039420	27.320767	5.527507	138.058060	0.085000
std	22.516840	0.26315	0.194593	6.636783	1.070672	40.708136	0.278883
min	0.080000	0.00000	0.000000	10.010000	3.500000	80.000000	0.000000
25%	24.000000	0.00000	0.000000	23.630000	4.800000	100.000000	0.000000
50%	43.000000	0.00000	0.000000	27.320000	5.800000	140.000000	0.000000
75%	60.000000	0.00000	0.000000	29.580000	6.200000	159.000000	0.000000
max	80.000000	1.00000	1.000000	95.690000	9.000000	300.000000	1.000000

In [ ]:
```
EDA
```

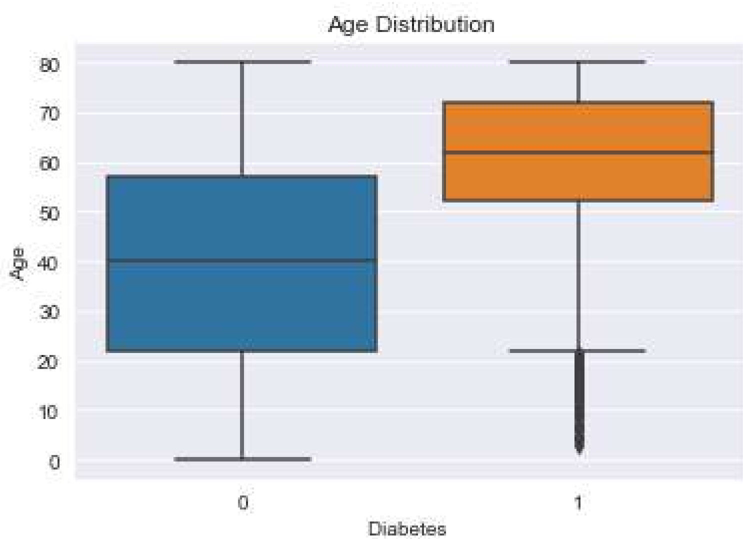
In [182…
```python
# Count of Individuals by Smoking History

sns.countplot(x='Smoking_History', data=df)
plt.title('Count of Individuals by Smoking History')
plt.xlabel('Smoking History')
plt.ylabel('Count')
plt.show()
```



In [183…
```python
# Age Distribution
sns.histplot(df['Age'], bins=10)
plt.title('Age Distribution')
plt.xlabel('Age')
plt.ylabel('Count')
plt.show()
```

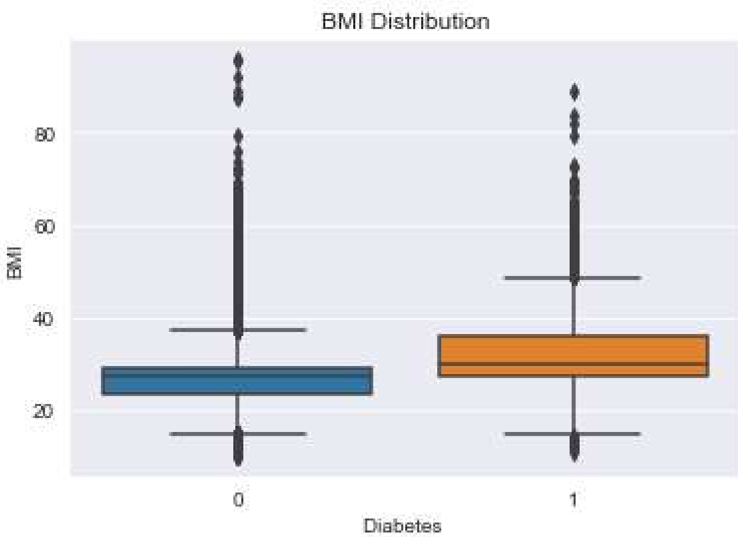


In [184…
```
sns.boxplot(x=df['Diabetes'], y=df['Age'])
plt.title('Age Distribution')
plt.xlabel('Diabetes ')
plt.ylabel('Age')
plt.show()
```



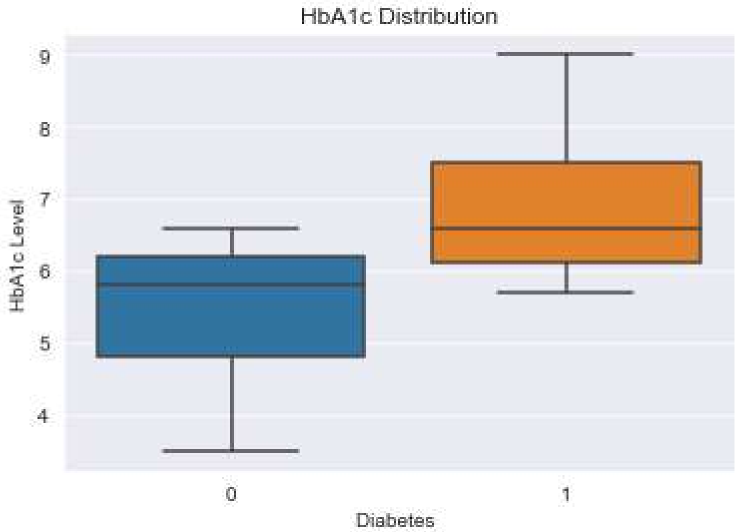
In [185…
```
# BMI Distribution
sns.boxplot(x=df['Diabetes'], y=df['BMI'])
plt.title('BMI Distribution')
```

```python
plt.xlabel('Diabetes ')
plt.ylabel('BMI')
plt.show()
```



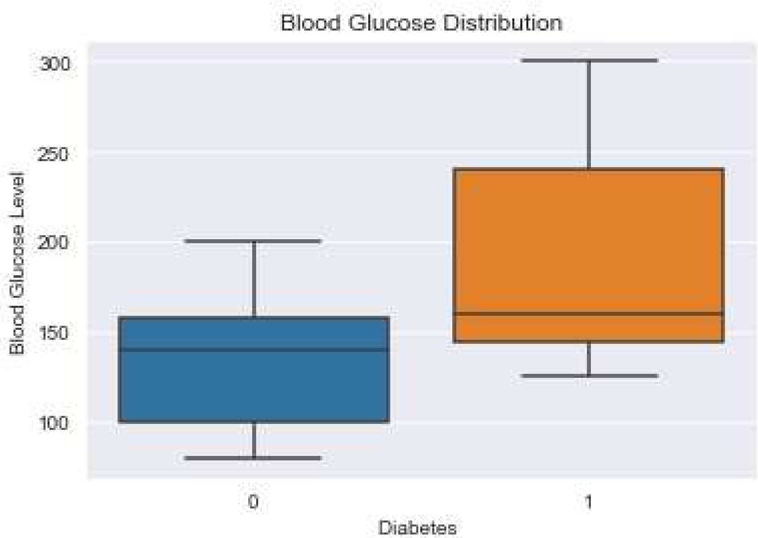
In [186…

```python
# HbA1c Level Distribution
sns.boxplot(x=df['Diabetes'], y=df['HbA1c_Level'])
plt.title('HbA1c Distribution')
plt.xlabel('Diabetes ')
plt.ylabel('HbA1c Level')
plt.show()
```



In [187…

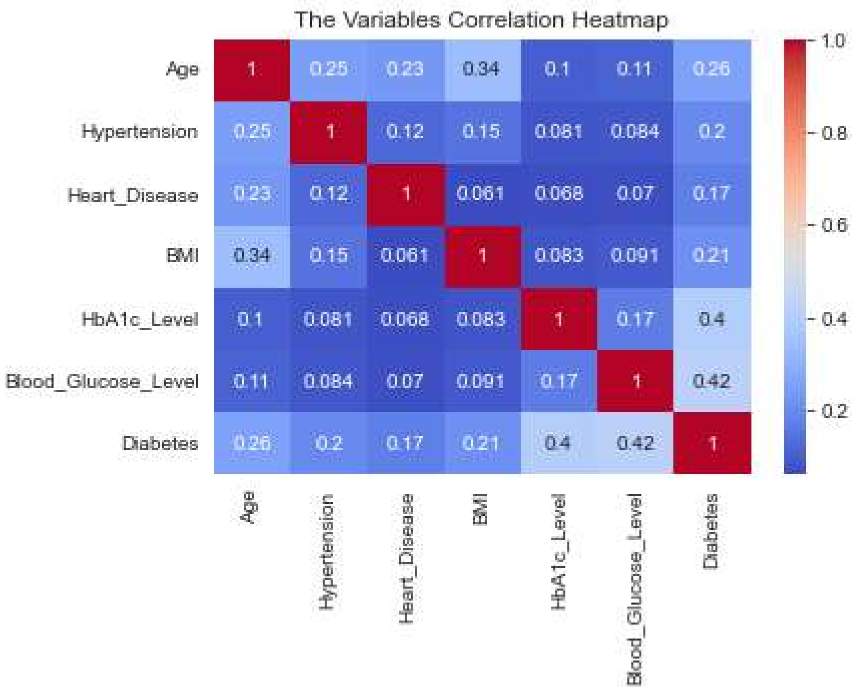
```
# Blood Glucose Level Distribution
sns.boxplot(x=df['Diabetes'], y=df['Blood_Glucose_Level'])
plt.title('Blood Glucose Distribution')
plt.xlabel('Diabetes ')
plt.ylabel('Blood Glucose Level')
plt.show()
```



In [188…

```
# The Variables Correlation Heatmap
corr = df[['Gender', 'Age', 'Hypertension', 'Heart_Disease', 'Smoking_History',
```

```
        'BMI', 'HbA1c_Level', 'Blood_Glucose_Level', 'Diabetes']].corr()
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.title('The Variables Correlation Heatmap')
plt.show()
```



In [189...

```
# Doing One-Hot Encoding of 'Gender' and 'Smoking_History' columns

dummy_df = pd.get_dummies(df[["Gender","Smoking_History"]], drop_first=True)
dummy_df.head()
```

Out[189]:

	Gender_Male	Gender_Other	Smoking_History_current	Smoking_History_ever	Smoking_History_former	Smoking_History_never	Smoking_Hi
0	0	0	0	0	0	1	
1	0	0	0	0	0	0	
2	1	0	0	0	0	1	
3	0	0	1	0	0	0	
4	1	0	1	0	0	0	

In [190…

```
# Concatinating One-Hot Encoding columns with other columns
X = pd.concat([df[['Age','Hypertension','Heart_Disease','BMI','HbA1c_Level','Blood_Glucose_Level']],dummy_df], axis=1)
X.head(10)
```

Out[190]:

	Age	Hypertension	Heart_Disease	BMI	HbA1c_Level	Blood_Glucose_Level	Gender_Male	Gender_Other	Smoking_History_current	Smoki
0	80.0	0	1	25.19	6.6	140	0	0	0	
1	54.0	0	0	27.32	6.6	80	0	0	0	
2	28.0	0	0	27.32	5.7	158	1	0	0	
3	36.0	0	0	23.45	5.0	155	0	0	1	
4	76.0	1	1	20.14	4.8	155	1	0	1	
5	20.0	0	0	27.32	6.6	85	0	0	0	
6	44.0	0	0	19.31	6.5	200	0	0	0	
7	79.0	0	0	23.86	5.7	85	0	0	0	
8	42.0	0	0	33.64	4.8	145	1	0	0	
9	32.0	0	0	27.32	5.0	100	0	0	0	

In [191…

```
# Scaling X
scaler = MinMaxScaler()

X = scaler.fit_transform(X)
X[0]
```

Out[191]:

```
array([1.        , 0.        , 1.        , 0.17717087, 0.56363636,
       0.27272727, 0.        , 0.        , 0.        , 0.        ,
       0.        , 1.        , 0.        ])
```

In [192…

```
# Defining y
y = df['Diabetes']
```

In [193…

```
# Value Counts of y; Check if  one individual with diabetes or not
# 0 means without diabetes, 1 means with diabetes.
y.value_counts()
```

Out[193]:
```
0    91500
1     8500
Name: Diabetes, dtype: int64
```

In [ ]:
```
# We can see the y value is imbalanced. For solving this problem, we can oversample our dataset with creating imaginary
```

In [194…
```
# SMOTE
smote = SMOTE(sampling_strategy='minority')
X, y = smote.fit_resample(X, y)

y.value_counts()
```

Out[194]:
```
0    91500
1    91500
Name: Diabetes, dtype: int64
```

In [ ]:
```
```

In [195…
```
# Generating descriptive statistics for object (string) columns in the DataFrame 'df'
df_object_summary = df.describe(include='object')
```

In [196…
```
# Calculating the number of missing values in each column of the DataFrame 'df'
#Checking Null Counts
df.isnull().sum()
```

Out[196]:
```
Gender                 0
Age                    0
Hypertension           0
Heart_Disease          0
Smoking_History        0
BMI                    0
HbA1c_Level            0
Blood_Glucose_Level    0
Diabetes               0
dtype: int64
```

In [197…
```
# Duplicate Values Count
df.duplicated().sum()
```

Out[197]:
```
3854
```

In [198…
```
# Remove Duplicate Values
df.drop_duplicates(inplace=True)
```

In [199…
```
df.shape
```

Out[199]:
```
(96146, 9)
```

In [202…
```
# Converting data types of specific columns in the DataFrame 'df'
df.Age = df.Age.astype(int)
df.Diabetes = df.Diabetes.astype('category')
df.Hypertension = df.Hypertension.astype('category')
df.Heart_Disease = df.Heart_Disease.astype('category')
```

In [203…
```
# Displaying a concise summary of the DataFrame 'df'
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 96146 entries, 0 to 99999
Data columns (total 9 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Gender               96146 non-null  object
 1   Age                  96146 non-null  int32
 2   Hypertension         96146 non-null  category
 3   Heart_Disease        96146 non-null  category
 4   Smoking_History      96146 non-null  object
 5   BMI                  96146 non-null  float64
 6   HbA1c_Level          96146 non-null  float64
 7   Blood_Glucose_Level  96146 non-null  int64
 8   Diabetes             96146 non-null  category
dtypes: category(3), float64(2), int32(1), int64(1), object(2)
memory usage: 5.0+ MB
```

In [204…
```
# Counting the number of unique values in each column of the DataFrame 'df'
df.nunique()
```

Out[204]:
```
Gender                    3
Age                      81
Hypertension              2
Heart_Disease             2
Smoking_History           6
BMI                    4247
HbA1c_Level              18
Blood_Glucose_Level      18
Diabetes                  2
dtype: int64
```

In [205…

```
# Removing rows with Gender as 'Other' from the DataFrame 'df'
df = df[~df.Gender.isin(['Other'])]
```

In [206…

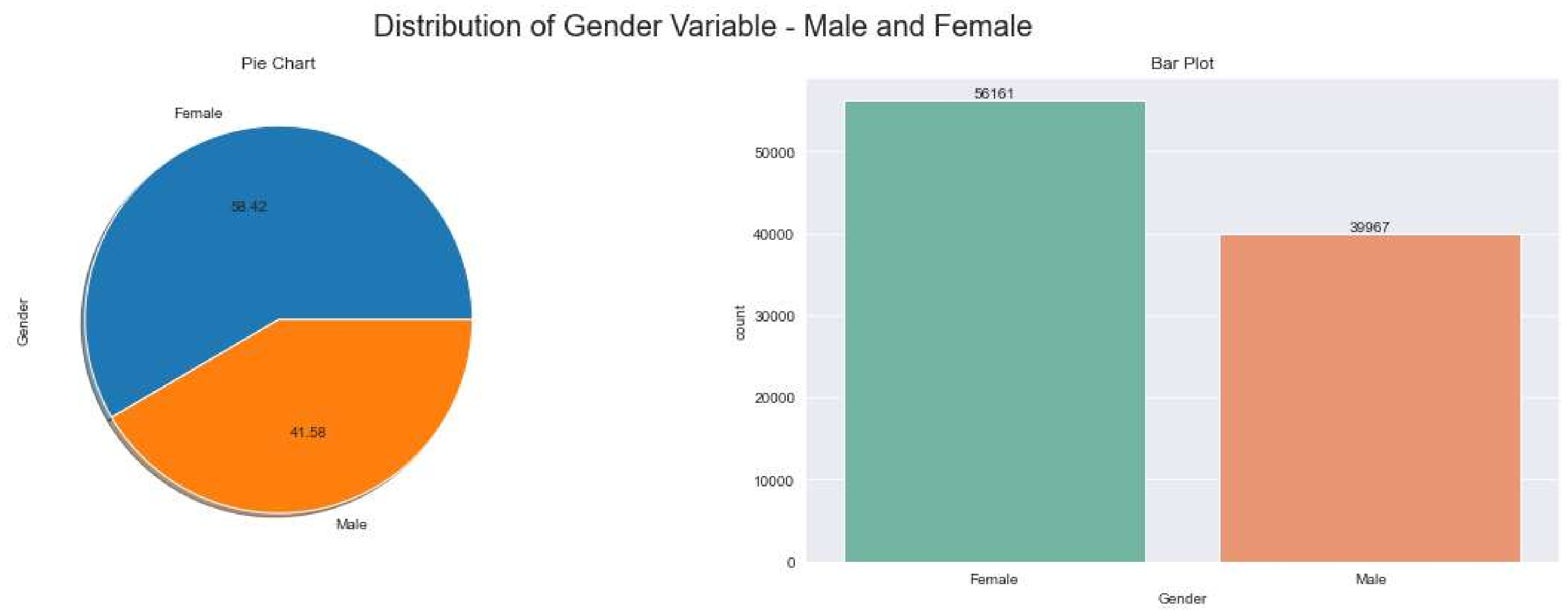
```
# Retrieving the unique values in the 'Gender' column of the DataFrame 'df'
df.Gender.unique()
```

Out[206]:

```
array(['Female', 'Male'], dtype=object)
```

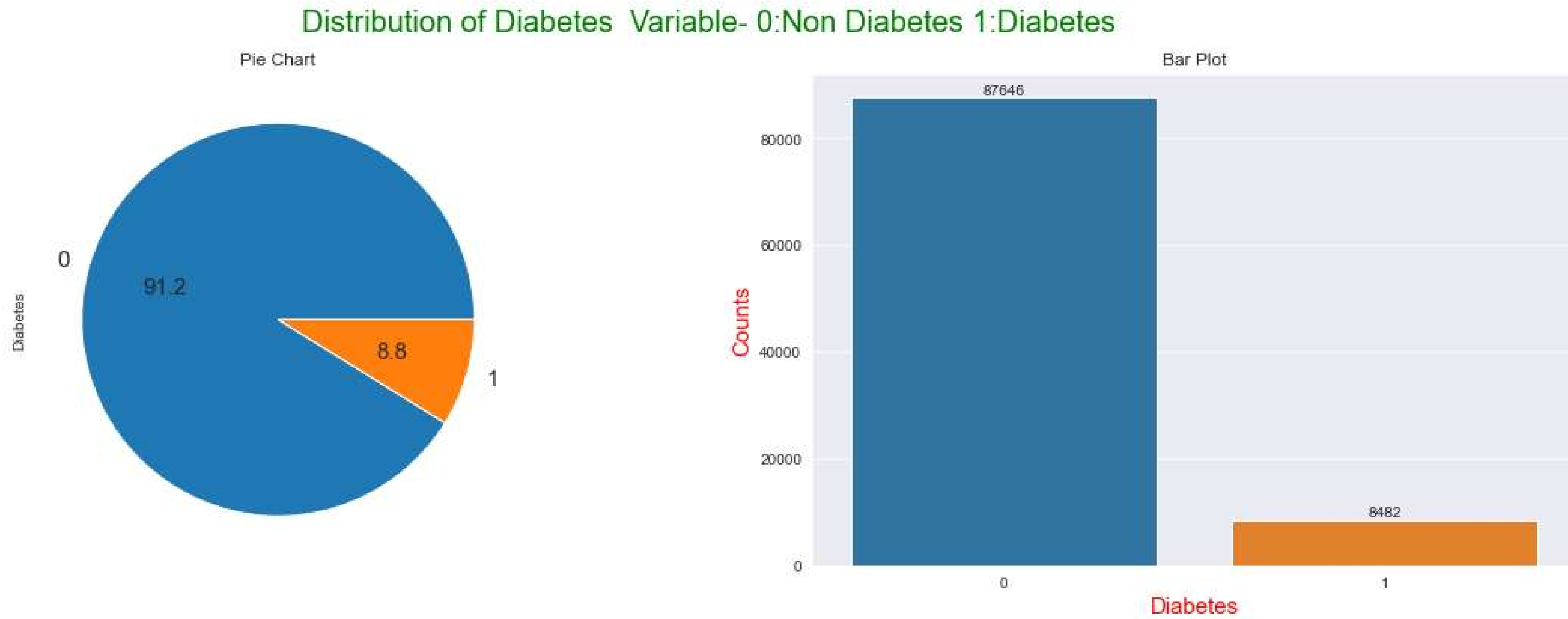
In [207…

```
# Creating a figure with two subplots to visualize the distribution of the 'Gender' variable
plt.figure(figsize=(20, 6))
sns.set_style('darkgrid')
plt.suptitle('Distribution of Gender Variable - Male and Female', fontsize=20)

# Subplot 1 - Pie Chart
plt.subplot(1, 2, 1)
df['Gender'].value_counts().plot(kind='pie', autopct='%.2f', shadow=True)
plt.title('Pie Chart')

# Subplot 2 - Bar Plot
plt.subplot(1, 2, 2)
ax = sns.countplot(x='Gender', data=df, palette='Set2')
ax.bar_label(ax.containers[0])
plt.title('Bar Plot')

plt.show()
```



In [208…

```
# Creating a figure with two subplots to visualize the distribution of the 'Diabetes Type' variable
plt.figure(figsize=(20, 6))
sns.set_style('darkgrid')
plt.suptitle('Distribution of Diabetes  Variable- 0:Non Diabetes 1:Diabetes', fontsize=20, color='green')

# Subplot 1 - Pie Chart
plt.subplot(1, 2, 1)
df['Diabetes'].value_counts().plot(kind='pie', autopct='%.1f', fontsize=15)
plt.title('Pie Chart')

# Subplot 2 - Bar Plot
plt.subplot(1, 2, 2)
ax = sns.countplot(x=df.Diabetes)
ax.bar_label(ax.containers[0])
plt.xlabel('Diabetes', fontsize=15, color='red')
plt.ylabel('Counts', fontsize=15, color='red')
plt.title('Bar Plot')

plt.show()
```



In [209…

```
def distribution_chart(df, feature):
    """
    Creates distribution charts for specified columns in the DataFrame.

    Parameters:
        df (DataFrame): The DataFrame containing the data.
        feature (list): List of column names to visualize their distributions.

    Returns:
        None (Displays distribution charts for each specified feature.)
    """
    for column in feature:
        # Creating a figure with two subplots to visualize the distribution of the current 'column'
        plt.figure(figsize=(18, 6))
        sns.set_style('darkgrid')
        plt.suptitle(f"Distribution of {column} Variable", fontsize=20)

        # Subplot 1 - Histogram with KDE (Kernel Density Estimate)
        plt.subplot(1, 2, 1)
        sns.histplot(df[column], kde=True)
        plt.axvline(x=df[column].mean(), color='red', alpha=0.5, label='Mean')
        plt.axvline(x=df[column].median(), ls='--', color='blue', alpha=0.5, label='Median')
        plt.legend()
```

```
        plt.xlabel(column)

        # Subplot 2 - Box Plot
        plt.subplot(1, 2, 2)
        sns.boxplot(x=df[column])
        plt.xlabel(column)

        plt.show()
```

In [210…
```
# Selecting columns with numeric data types from the DataFrame 'df'
num_var = df.select_dtypes('number').columns
num_var
```

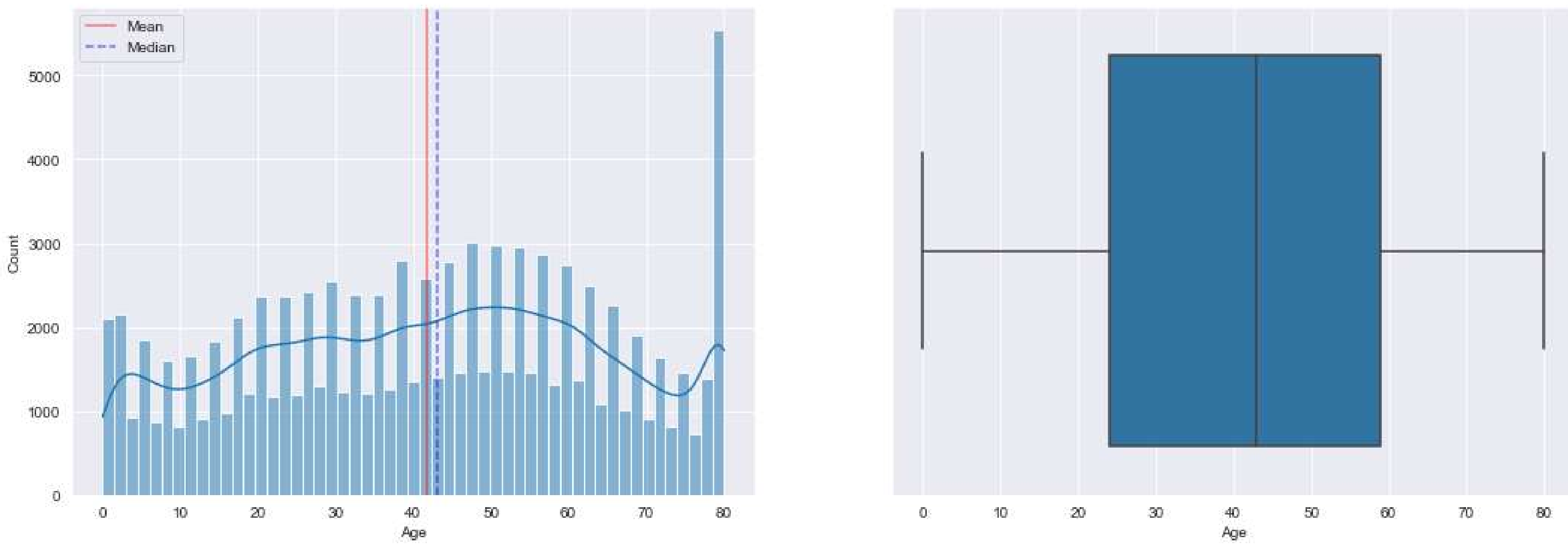
Out[210]:
```
Index(['Age', 'BMI', 'HbA1c_Level', 'Blood_Glucose_Level'], dtype='object')
```

Explanation:

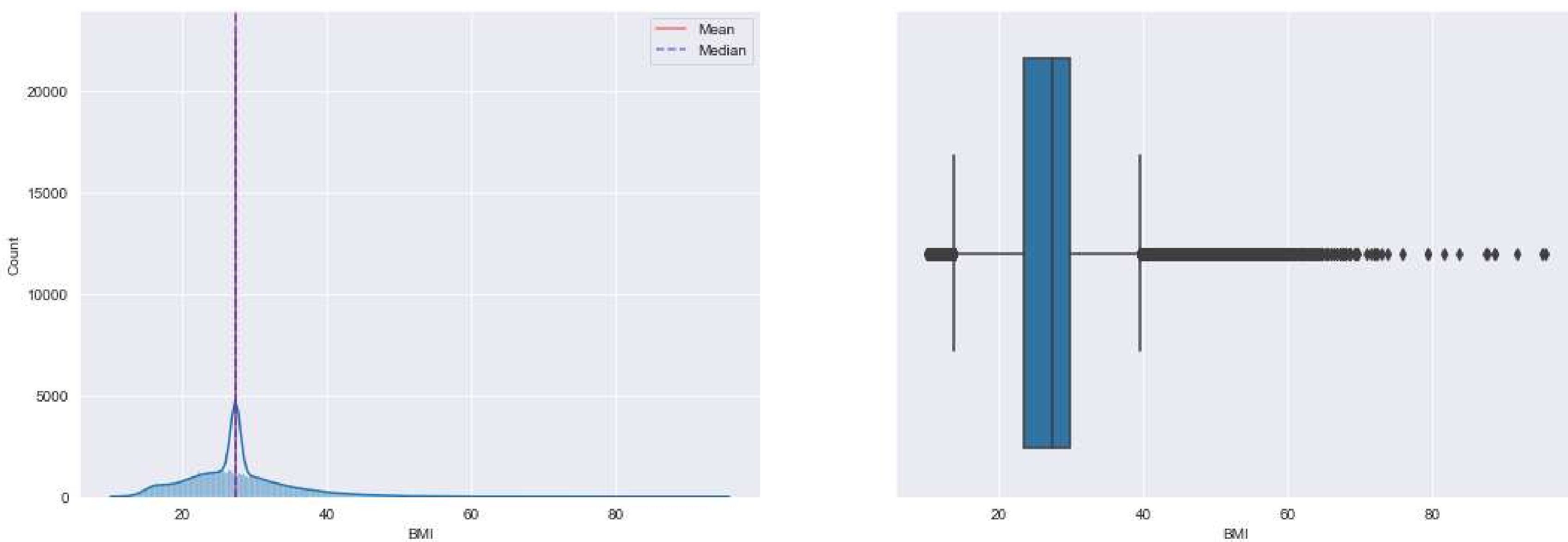
- The `select_dtypes()` function of a DataFrame in pandas is used to select columns based on their data types.
- When `select_dtypes('number')` is used, it selects only those columns that have numeric data types, such as integers and floating-point numbers.
- The `columns` attribute is then used to retrieve the names of the selected columns as a list.
- The resulting list `num_var` contains the names of columns that contain numerical data, which can be used for further analysis or visualization tasks.
- This is useful when you have a mix of numeric and non-numeric columns in the DataFrame and want to focus only on the numerical ones for specific operations.

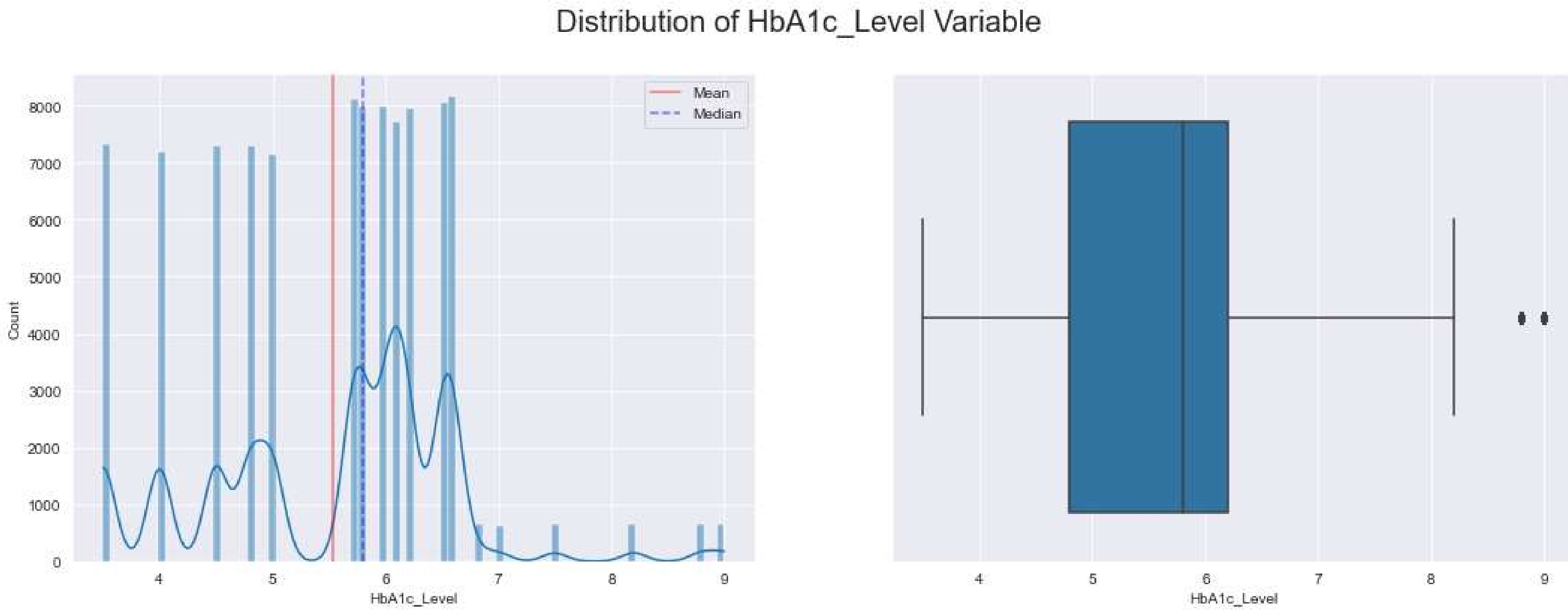
In [211…
```
# Using the function to visualize the distribution of numerical variables in the DataFrame 'df'
num_var = df.select_dtypes('number').columns
distribution_chart(df, num_var)
```

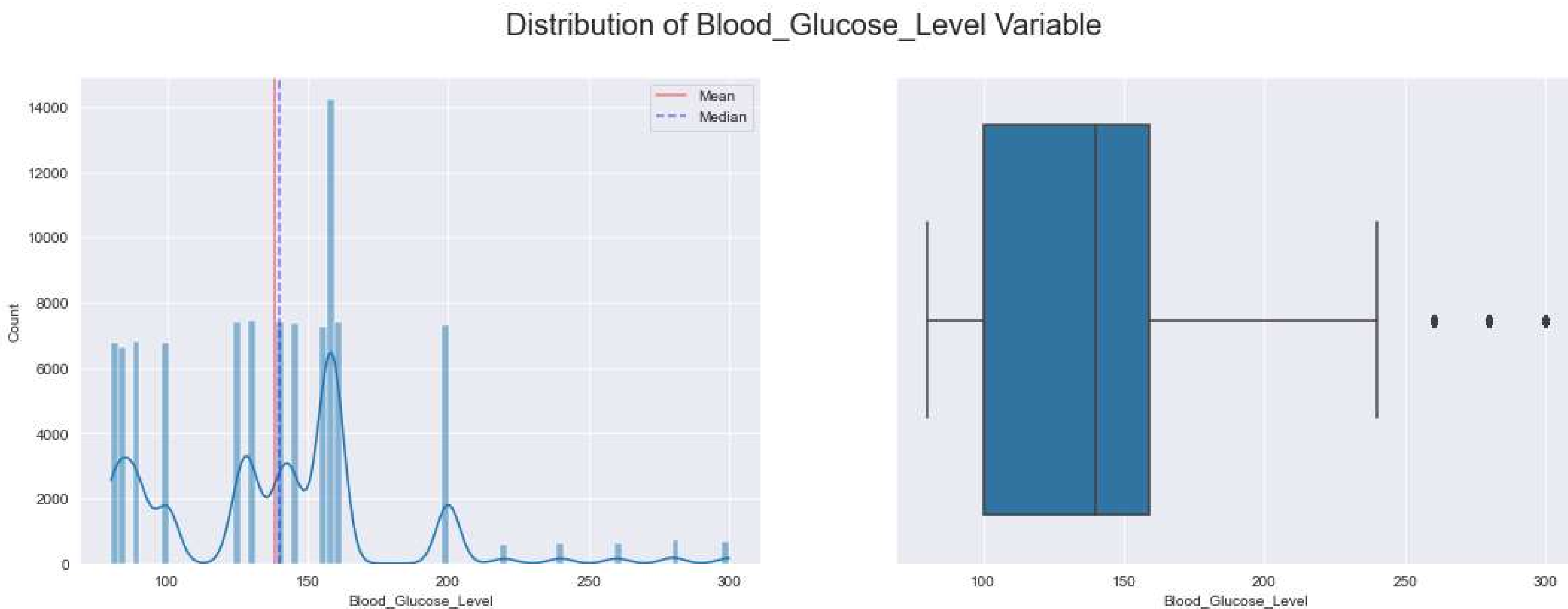
Distribution of Age Variable



Distribution of BMI Variable







In [212…

```
def detect_outliers(df, column):
    # Step 1: Extract the column values and sort them in ascending order
    column_values = df[column].values
```

```python
    column_values_sorted = np.sort(column_values)

    # Step 2: Calculate the first quartile (Q1) and third quartile (Q3)
    q1, q3 = np.percentile(column_values_sorted, [25, 75])

    # Step 3: Calculate the Interquartile Range (IQR)
    iqr = q3 - q1

    # Step 4: Calculate the lower and upper fences to identify outliers
    lower_fence = q1 - 1.5 * iqr
    upper_fence = q3 + 1.5 * iqr

    # Step 5: Identify outliers in the column based on the lower and upper fences
    outliers = [value for value in column_values if value < lower_fence or value > upper_fence]

    # Step 6: Return the list of identified outliers
    return outliers
```

In [214…

```python
# Assuming we have already defined the 'detect_outliers()' function

# Detect outliers in each column
Age_outliers = detect_outliers(df, 'Age')
BMI_outliers = detect_outliers(df, 'BMI')
HbA1c_outliers = detect_outliers(df, 'HbA1c_Level')
Blood_Glucose_outliers = detect_outliers(df, 'Blood_Glucose_Level')

# Remove outliers from the DataFrame
df = df[~df['Age'].isin(Age_outliers)]
df = df[~df['BMI'].isin(BMI_outliers)]
df = df[~df['HbA1c_Level'].isin(HbA1c_outliers)]
df = df[~df['Blood_Glucose_Level'].isin(Blood_Glucose_outliers)]

# The DataFrame 'df' now contains the data with outliers removed from each column.
```

In [215…

```python
# Resetting the index of the DataFrame 'df' and dropping the existing index
df = df.reset_index(drop=True)
```

In [216…

```python
# Checking the updated shape of the DataFrame 'df'
df.shape
```

Out[216]:

```
(88177, 9)
```

In [217…

```
# Counting the number of unique values in each column of the DataFrame 'df' after resetting the index
df.nunique()
```

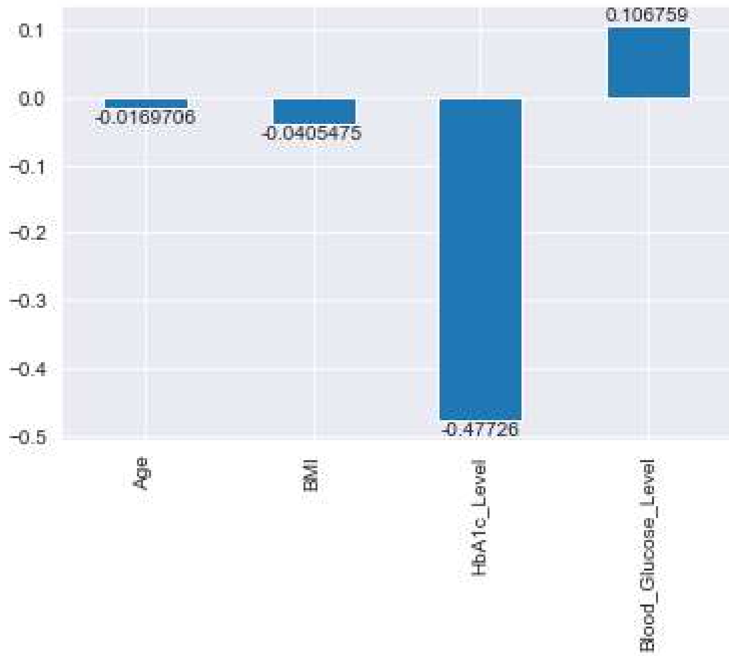
Out[217]:

```
Gender                    2
Age                      81
Hypertension              2
Heart_Disease             2
Smoking_History           6
BMI                    2582
HbA1c_Level              16
Blood_Glucose_Level      15
Diabetes                  2
dtype: int64
```

In [218…

```
# Calculating and printing the mean, median, and skewness for each numerical variable in the DataFrame 'df'
print('----------- Mean -----------')
print(df[num_var].mean())
print()

print('---------- Median -----------')
print(df[num_var].median())
print()

print('---------- Skewness -----------')
# Calculating skewness for each numerical variable and plotting it as a bar plot
ax = df[num_var].skew().plot(kind='bar')
ax.bar_label(ax.containers[0])
print()
```

```
----------- Mean -----------
Age                     41.061683
BMI                     26.314402
HbA1c_Level              5.459199
Blood_Glucose_Level    134.629144
dtype: float64

---------- Median -----------
Age                     42.00
BMI                     27.32
HbA1c_Level              5.80
Blood_Glucose_Level    140.00
dtype: float64

---------- Skewness -----------
```



Outcome with other features

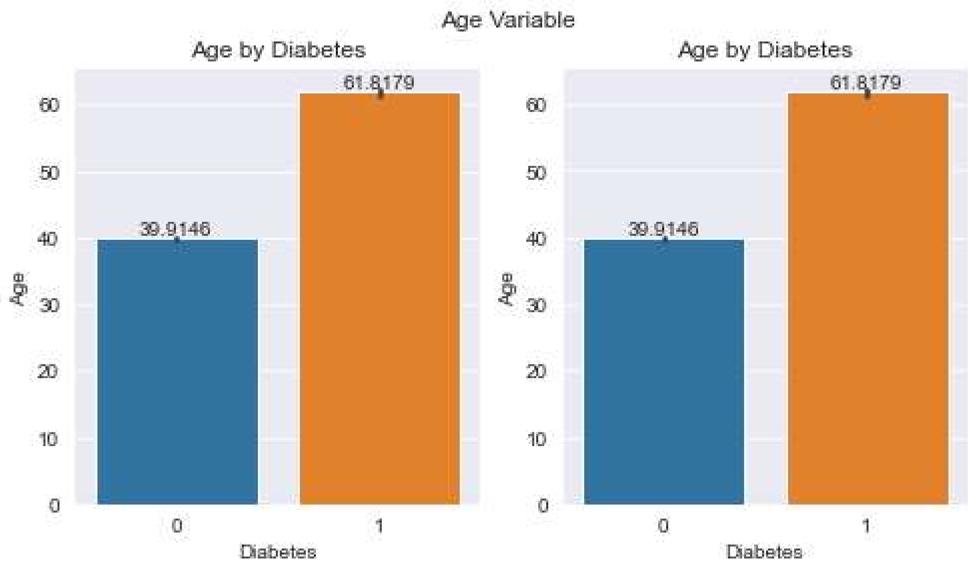
In [219…

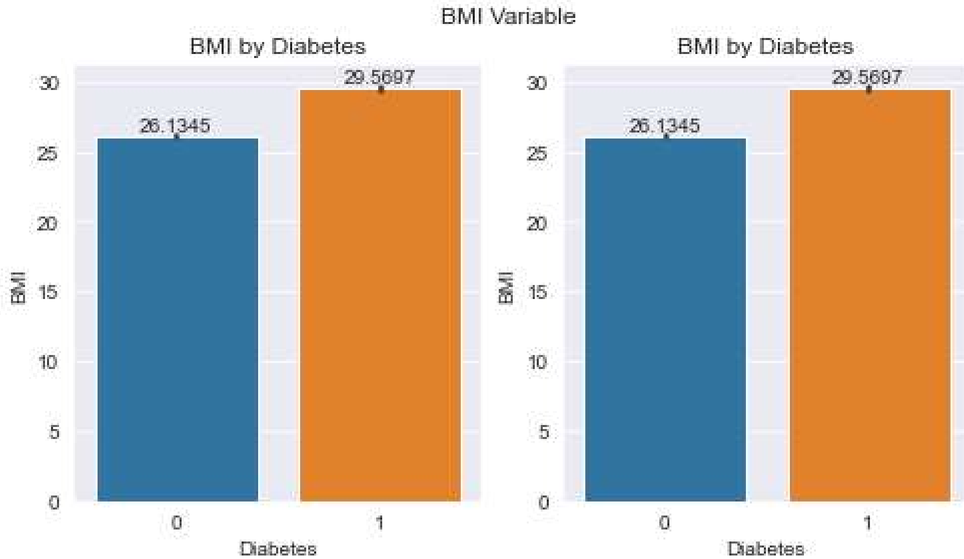
```python
def outcome_features(df, features, x_var='Diabetes'):
    """
    Creates a pair of bar plots for specified features in the DataFrame 'df' based on the variable 'x_var'.

    Parameters:
        df (DataFrame): The DataFrame containing the data.
        features (list): List of column names to be visualized.
        x_var (str): The variable to compare the features with (default: 'Diabetes').

    Returns:
        None (Displays the pair of bar plots for each specified feature).
    """
    for column in features:
        # Creating a figure with two subplots to visualize the features based on 'x_var'
        plt.figure(figsize=(8, 4))
        sns.set_style('darkgrid')
        plt.suptitle(f"{column} Variable")

        # Subplot 1 - Bar Plot with bar labels
        plt.subplot(1, 2, 1)
        ax = sns.barplot(x=x_var, y=column, data=df)
```

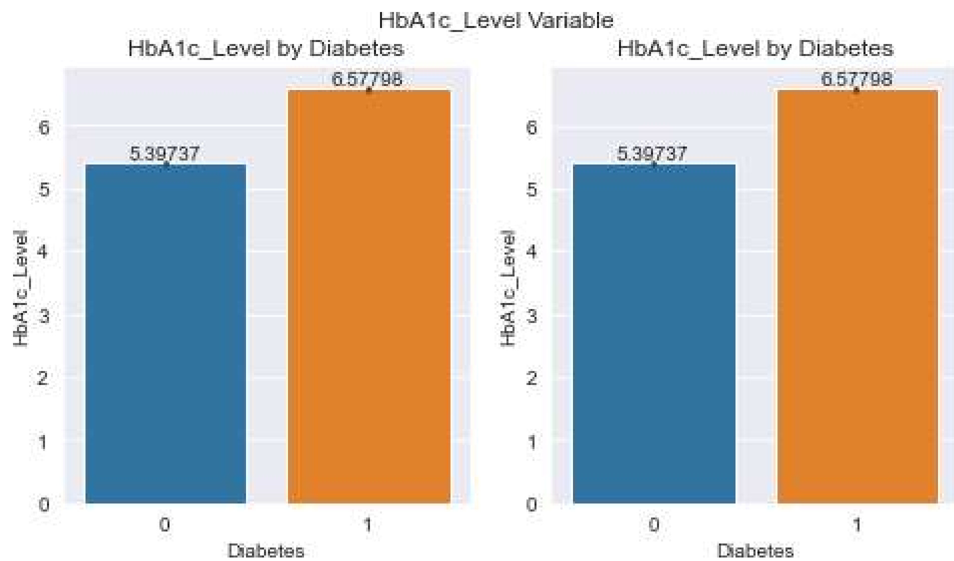
```
        ax.bar_label(ax.containers[0])
        plt.title(f"{column} by {x_var}")

        # Subplot 2 - Another Bar Plot (similar to Subplot 1) with bar labels
        plt.subplot(1, 2, 2)
        ax = sns.barplot(x=x_var, y=column, data=df)
        ax.bar_label(ax.containers[0])
        plt.title(f"{column} by {x_var}")

        plt.show()
```

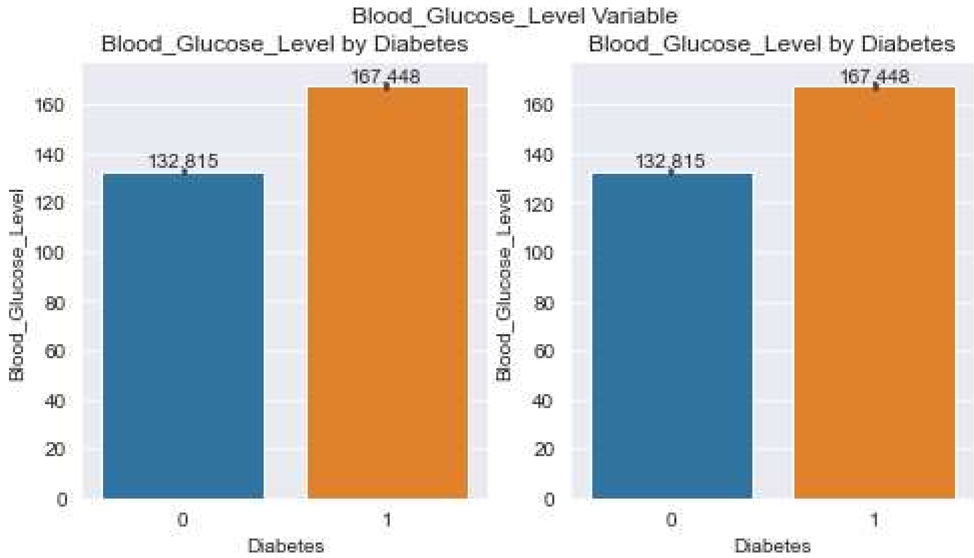
In [220…

```
# Using the function to visualize the numerical variables based on the 'Diabetes' variable
outcome_features(df, num_var)
```



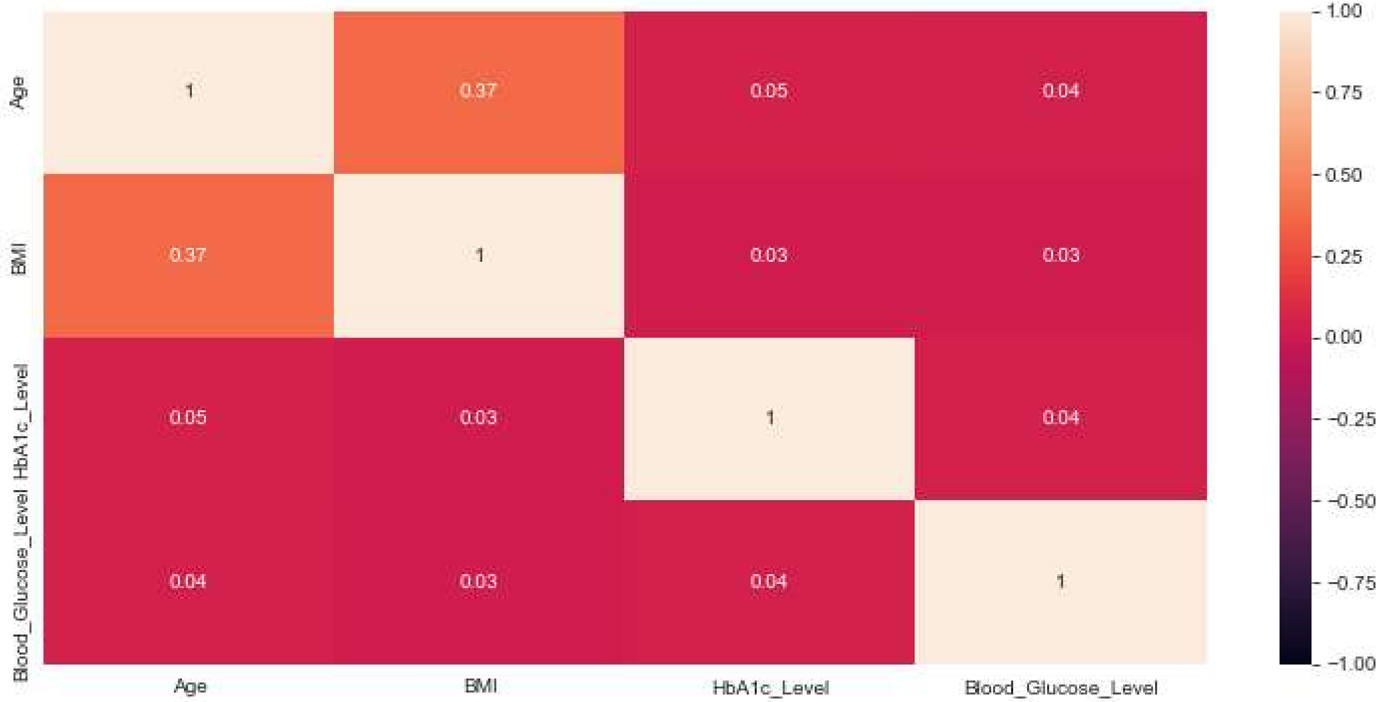

BMI Variable
BMI by Diabetes
BMI by Diabetes
26.1345
29.5697
26.1345
29.5697
BMI
BMI
Diabetes
Diabetes
0
1
0
5
10
15
20
25
30


HbA1c_Level Variable
HbA1c_Level by Diabetes
HbA1c_Level by Diabetes
5.39737
6.57798
5.39737
6.57798
HbA1c_Level
HbA1c_Level
Diabetes
Diabetes
0
1
0
1
2
3
4
5
6



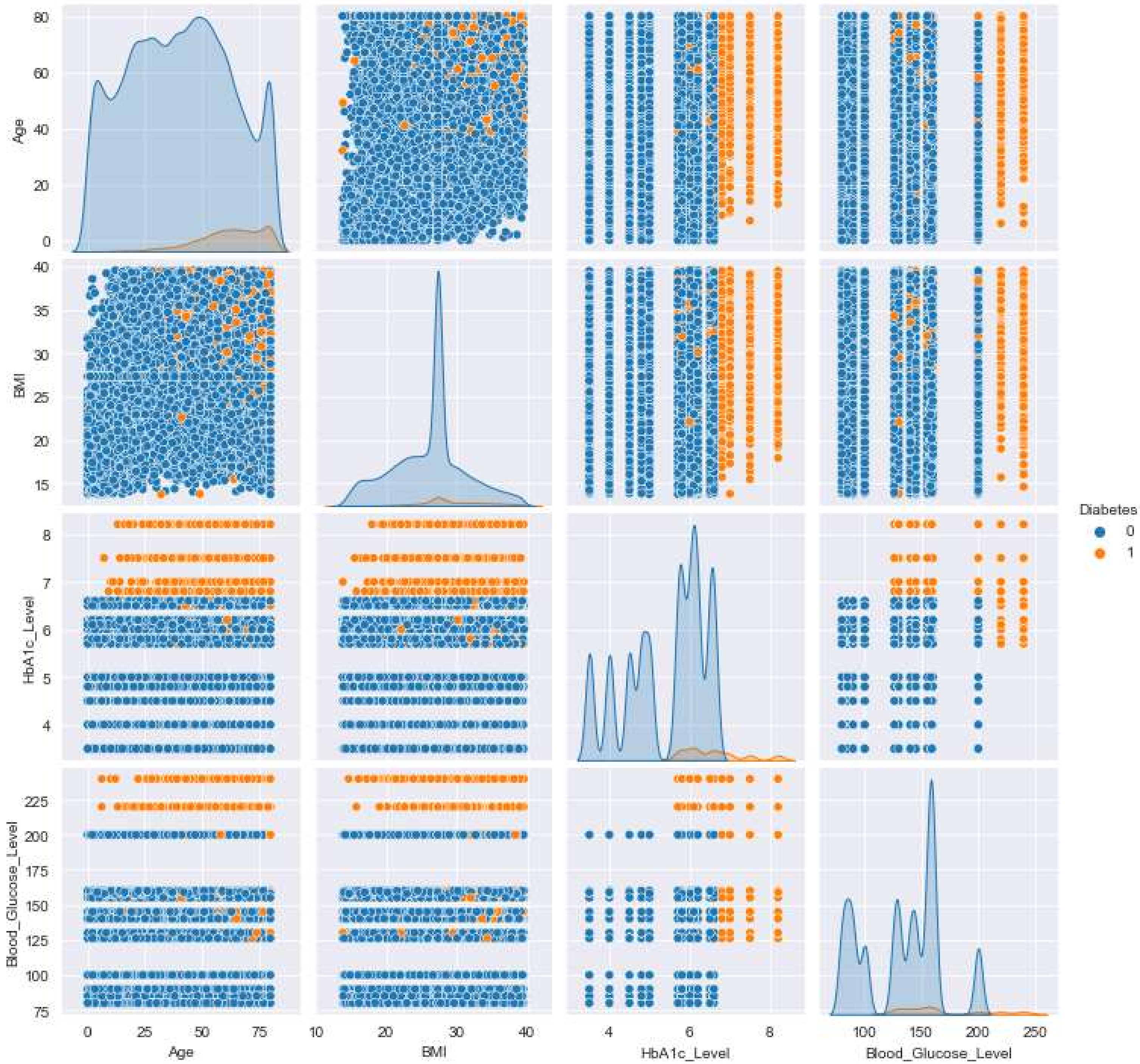
In [221…

```
# Selecting only the numerical columns from the DataFrame 'df' and creating DataFrame 'df2'
df2 = df[num_var]

# Creating a figure and a heatmap to visualize the Spearman correlation matrix
plt.figure(figsize=(13, 6))
sns.heatmap(df2.corr(method='spearman').round(2), annot=True, vmin=-1, vmax=1)
plt.show()
```



In [222...

```
# Creating a pair plot of scatter plots for numerical variables, colored by the 'Diabetes' variable
sns.pairplot(data=df, hue='Diabetes')
plt.show()
```



Age
BMI
HbA1c_Level
Blood_Glucose_Level
Diabetes
0
1

Export dataset

In [223…

```
# Saving the DataFrame 'df' as a CSV file named 'clean_dataset_diabetes.csv' without including the index
df.to_csv('clean_dataset_diabetes.csv', index=False)
```

Explanation:

- The `to_csv()` function is used to save the DataFrame 'df' as a CSV (Comma-Separated Values) file.
- The first argument 'clean_dataset_diabetes.csv' specifies the name of the CSV file to be created.
- The second argument 'index=False' ensures that the DataFrame's index is not included in the CSV file.
- If 'index=True' were used, the index would be saved as an additional column in the CSV file.
- By setting 'index=False', the CSV file will only contain the data from the DataFrame 'df'.
- The CSV file will be saved in the current working directory (the location where the Python script or Jupyter Notebook is running).
- The saved CSV file can be later loaded back into a DataFrame using the `pd.read_csv()` function in pandas.

Models with their results

In [224…

```
# Creating feature variables (X) and the target variable (y) for machine learning tasks

# Dropping columns 'gender', 'smoking_history', and 'diabetes' to form the feature variables (X)
X = df.drop(['Gender', 'Smoking_History', 'Diabetes'], axis=1)

# Assigning the target variable 'diabetes' to y
y = df['Diabetes']
```

In [225…

```
# Standardizing the feature variables in 'X' using the StandardScaler

# Importing the StandardScaler from scikit-learn
from sklearn.preprocessing import StandardScaler

# Creating a StandardScaler object
sc = StandardScaler()

# Using the StandardScaler object to standardize the feature variables in 'X'
x = sc.fit_transform(X)
```

In [226…

```
# Splitting the standardized feature variables 'x' and the target variable 'y' into training and testing sets

# Importing the train_test_split function from scikit-learn
```

```python
from sklearn.model_selection import train_test_split

# Using train_test_split to split 'x' and 'y' into training and testing sets
# test_size=0.1 specifies that 10% of the data will be used for testing, and 90% will be used for training
# random_state=0 sets the random seed for reproducibility, ensuring the same split is obtained on each run
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.1, random_state=0)
```

In [227…

```python
# Importing performance metrics from scikit-learn

# Accuracy Score: Evaluates the accuracy of classification models by comparing predicted labels to true labels.
from sklearn.metrics import accuracy_score

# Classification Report: Provides a comprehensive report including precision, recall, F1-score, and support for each cl
from sklearn.metrics import classification_report

# Confusion Matrix: Evaluates the performance of a classification model by showing the counts of true positive, false p
from sklearn.metrics import confusion_matrix

# R-squared Score (R2 Score): Evaluates the goodness-of-fit of regression models, indicating how well the model fits th
from sklearn.metrics import r2_score

# Mean Squared Error (MSE): Evaluates the performance of regression models by measuring the average squared difference
from sklearn.metrics import mean_squared_error
```

1. Logistic Regression Classifier

In [228…

```python
from sklearn.linear_model import LogisticRegression
reg = LogisticRegression()
reg.fit(x_train, y_train)
```

Out[228]:

```
LogisticRegression()
```

In [229…

```python
y_pred = reg.predict(x_test)
print('-----------------------------------------------------')
print('Classification Report is : ↴\n',classification_report(y_test, y_pred))
print('-----------------------------------------------------')
print('Confusion Metrix : ↴\n',confusion_matrix(y_test, y_pred))
print('-----------------------------------------------------')
print('\tTraining Score : ', str(round(reg.score(x_train, y_train)*100, 1))+'%')
print('-----------------------------------------------------')
print('\tMean Square Error : ',str(round(mean_squared_error(y_test, y_pred)*100, 2))+'%')
print('-----------------------------------------------------')
```

```
print('\tR2 score is : ',str(round(r2_score(y_test,y_pred)*100,2))+'%')
print('-----------------------------------------------------')
```

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

           0       0.97      1.00      0.98      8370
           1       0.82      0.37      0.51       448

    accuracy                           0.96      8818
   macro avg       0.89      0.68      0.75      8818
weighted avg       0.96      0.96      0.96      8818

-----------------------------------------------------
Confusion Metrix : ↴
 [[8334   36]
 [ 282  166]]
-----------------------------------------------------
        Training Score :  96.2%
-----------------------------------------------------
        Mean Square Error :  3.61%
-----------------------------------------------------
        R2 score is :  25.22%
-----------------------------------------------------
```

In [230…

```
reg_score = round(accuracy_score(y_test, y_pred)*100,2) # getting score
print(str(reg_score)+'%')
```

```
96.39%
```

2. Decision Tree Classifier

In [231…

```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier(max_depth=6, random_state=123, criterion='entropy')
dtree.fit(x_train,y_train)
```

Out[231]:

```
DecisionTreeClassifier(criterion='entropy', max_depth=6, random_state=123)
```

In [232…

```
y_pred = dtree.predict(x_test)
print('-----------------------------------------------------')
print('Classification Report is : ↴\n',classification_report(y_test, y_pred))
print('-----------------------------------------------------')
```

```
print('Confusion Metrix : ↴\n',confusion_matrix(y_test, y_pred))
print('-----------------------------------------------------')
print('\tTraining Score : ', str(round(dtree.score(x_train, y_train)*100, 1))+'%')
print('-----------------------------------------------------')
print('\tMean Square Error : ',str(round(mean_squared_error(y_test, y_pred)*100, 2))+'%')
print('-----------------------------------------------------')
print('\tR2 score is : ',str(round(r2_score(y_test,y_pred)*100,2))+'%')
print('-----------------------------------------------------')
```

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

           0       0.97      1.00      0.99      8370
           1       1.00      0.49      0.66       448

    accuracy                           0.97      8818
   macro avg       0.99      0.74      0.82      8818
weighted avg       0.97      0.97      0.97      8818

-----------------------------------------------------
Confusion Metrix : ↴
 [[8370    0]
 [ 229  219]]
-----------------------------------------------------
        Training Score :  97.3%
-----------------------------------------------------
        Mean Square Error :  2.6%
-----------------------------------------------------
        R2 score is :  46.15%
-----------------------------------------------------
```

In [233…

```
dt_score = round(accuracy_score(y_test, y_pred)*100,2) # getting score
print(str(dt_score)+'%')
```

```
97.4%
```

3. Random Forest Classifier

In [234…

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
rfc.fit(x_train, y_train)
```

Out[234]:

```
RandomForestClassifier()
```

In [235…

```
y_pred = rfc.predict(x_test)
print('-----------------------------------------------------')
print('Classification Report is : ↴\n',classification_report(y_test, y_pred))
print('-----------------------------------------------------')
print('Confusion Metrix : ↴\n',confusion_matrix(y_test, y_pred))
print('-----------------------------------------------------')
print('\tTraining Score : ', str(round(rfc.score(x_train, y_train)*100, 1))+'%')
print('-----------------------------------------------------')
print('\tMean Square Error : ',str(round(mean_squared_error(y_test, y_pred)*100, 2))+'%')
print('-----------------------------------------------------')
print('\tR2 score is : ',str(round(r2_score(y_test,y_pred)*100,2))+'%')
print('-----------------------------------------------------')
```

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

           0       0.98      0.99      0.98      8370
           1       0.79      0.53      0.64       448

    accuracy                           0.97      8818
   macro avg       0.88      0.76      0.81      8818
weighted avg       0.97      0.97      0.97      8818

-----------------------------------------------------
Confusion Metrix : ↴
 [[8307   63]
 [ 209  239]]
-----------------------------------------------------
        Training Score :  99.7%
-----------------------------------------------------
        Mean Square Error :  3.08%
-----------------------------------------------------
        R2 score is :  36.04%
-----------------------------------------------------
```

In [236…

```
rfc_score = round(accuracy_score(y_test, y_pred)*100,2) # getting score
print(str(rfc_score)+'%')
```

```
96.92%
```

4. KNN Classifier

In [237…

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=10)
knn.fit(x_train, y_train)
```

Out[237]:

```
KNeighborsClassifier(n_neighbors=10)
```

In [238…

```
y_pred = knn.predict(x_test)
print('-----------------------------------------------------')
print('Classification Report is : ↴\n',classification_report(y_test, y_pred))
print('-----------------------------------------------------')
print('Confusion Metrix : ↴\n',confusion_matrix(y_test, y_pred))
print('-----------------------------------------------------')
print('\tTraining Score : ', str(round(knn.score(x_train, y_train)*100, 1))+'%')
print('-----------------------------------------------------')
print('\tMean Square Error : ',str(round(mean_squared_error(y_test, y_pred)*100, 2))+'%')
print('-----------------------------------------------------')
print('\tR2 score is : ',str(round(r2_score(y_test,y_pred)*100,2))+'%')
print('-----------------------------------------------------')
```

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

           0       0.97      1.00      0.98      8370
           1       0.94      0.43      0.59       448

    accuracy                           0.97      8818
   macro avg       0.96      0.71      0.79      8818
weighted avg       0.97      0.97      0.96      8818

----------------------------------------------------
Confusion Metrix : ↴
 [[8358   12]
 [ 257  191]]
----------------------------------------------------
        Training Score :  96.9%
----------------------------------------------------
        Mean Square Error :  3.05%
----------------------------------------------------
        R2 score is :  36.74%
----------------------------------------------------
```

In [241…

```
knn_score = round(accuracy_score(y_test, y_pred)*100,2) # getting score
print(str(knn_score)+'%')
```

```
96.95%
```

In [245…

```
# Printing the accuracy scores of different machine learning models

# 'reg_score', 'dt_score', 'rfc_score', and 'knn_score' are variables that hold the accuracy scores of different models
# These variables are assumed to have been computed previously by evaluating the respective models on the test data.

# Printing the accuracy scores for each model in a formatted table.
print('-------- Accuracy Table --------')
print()
print('Logistic Regression : ', reg_score)
print()
print('Decision Tree       : ', dt_score)
print()
print('Random Forest       : ', rfc_score)
print()
print('KNN                 : ', knn_score)
print()
print('-------- -------------- --------')
```

```
-------- Accuracy Table --------

Logistic Regression :  96.39

Decision Tree       :  97.4

Random Forest       :  96.92

KNN                 :  96.95

-------- ------------- --------
```

- The classifier Decision Tree has highest percentage for the data classification prediction accuracy of 97.4 %.

End