Project : Diabetes Prediction

Parameter which are invole

1. Pregnancies: Number of times pregnant
2. Glucose: Plasma glucose concentration a 2 hours in an oral glucose tolerance test
3. BloodPressure: Diastolic blood pressure (mm Hg)
4. SkinThickness: Triceps skin fold thickness (mm)
5. Insulin: 2-Hour serum insulin (mu U/ml)
6. BMI: Body mass index (weight in kg/(height in m)^2)
7. DiabetesPedigreeFunction: Diabetes pedigree function
8. Age: Age (years)
9. Outcome: Class variable (0 or 1)

Importing Requried Python Libraries

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

Uploading Dataset

We have our data saved in a CSV file called Diabetes_1.csv. We first read our dataset into a pandas dataframe called data, and then use the head() function to show the first five records from our dataset.

In [2]:
```
# loading the data using pandas

Data = pd.read_csv('Diabetes_1.csv')

# Displaying top 5 rows

Data.head()
```

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Exploratory Data Analysis

In [3]:
```
# Checking for Null Values in Dataset

Data.isnull().sum()
```

Out[3]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [4]:
```
# Summery of Dataframe

Data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

In [5]:
```
# Displaying total rows and columns

Data.shape
```

Out[5]: (768, 9)

In [6]:
```
# Display Distinct values

Data['Outcome'].unique()
```

Out[6]: array([1, 0], dtype=int64)

In [7]:
```
# Display Distinct values

Data['Age'].unique()
```

Out[7]:
```
array([50, 31, 32, 21, 33, 30, 26, 29, 53, 54, 34, 57, 59, 51, 27, 41, 43,
       22, 38, 60, 28, 45, 35, 46, 56, 37, 48, 40, 25, 24, 58, 42, 44, 39,
       36, 23, 61, 69, 62, 55, 65, 47, 52, 66, 49, 63, 67, 72, 81, 64, 70,
       68], dtype=int64)
```

In [8]:
```
# Classified Age into Young , Middle , Siniors

def Age_Class(Age):

    if Age <= 35:
        return 'Young Age'

    elif Age <= 60:
        return 'Middle Age'

    else:
        return 'Siniors'

Data['Age Classification'] = Data['Age'].apply(Age_Class)
Data.head()
```

Out[8]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	Age Classification
0	6	148	72	35	0	33.6	0.627	50	1	Middle Age
1	1	85	66	29	0	26.6	0.351	31	0	Young Age
2	8	183	64	0	0	23.3	0.672	32	1	Young Age
3	1	89	66	23	94	28.1	0.167	21	0	Young Age
4	0	137	40	35	168	43.1	2.288	33	1	Young Age

In [9]:
```
# Classified BMI into Under Weight , Normal Weight , Over Weight

def BMI_Class(BMI):

    if BMI <= 45:
        return 'Under Weight'

    elif BMI <= 60:
        return 'Normal Weight'

    else:
        return 'Over Weight'

Data['BMI Classification'] = Data['BMI'].apply(BMI_Class)
Data.head()
```

Out[9]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	Age Classification	Classifi
0	6	148	72	35	0	33.6	0.627	50	1	Middle Age	Under \
1	1	85	66	29	0	26.6	0.351	31	0	Young Age	Under \
2	8	183	64	0	0	23.3	0.672	32	1	Young Age	Under \
3	1	89	66	23	94	28.1	0.167	21	0	Young Age	Under \
4	0	137	40	35	168	43.1	2.288	33	1	Young Age	Under \

In [10]:
```
# Display Descriptive Statistics of Data

Data.describe().T
```

Out[10]:

	count	mean	std	min	25%	50%	75%	max
Pregnancies	768.0	3.845052	3.369578	0.000	1.00000	3.0000	6.00000	17.00
Glucose	768.0	120.894531	31.972618	0.000	99.00000	117.0000	140.25000	199.00
BloodPressure	768.0	69.105469	19.355807	0.000	62.00000	72.0000	80.00000	122.00
SkinThickness	768.0	20.536458	15.952218	0.000	0.00000	23.0000	32.00000	99.00
Insulin	768.0	79.799479	115.244002	0.000	0.00000	30.5000	127.25000	846.00
BMI	768.0	31.992578	7.884160	0.000	27.30000	32.0000	36.60000	67.10
DiabetesPedigreeFunction	768.0	0.471876	0.331329	0.078	0.24375	0.3725	0.62625	2.42
Age	768.0	33.240885	11.760232	21.000	24.00000	29.0000	41.00000	81.00
Outcome	768.0	0.348958	0.476951	0.000	0.00000	0.0000	1.00000	1.00

In [11]:
```
# Display Average value of each Parameter from Table

round(Data.mean() , 2)
```

```
C:\Users\admin\AppData\Local\Temp\ipykernel_12336\3818815197.py:3: FutureWarning: The default value of numer
ic_only in DataFrame.mean is deprecated. In a future version, it will default to False. In addition, specify
ing 'numeric_only=None' is deprecated. Select only valid columns or specify the value of numeric_only to sil
ence this warning.
  round(Data.mean() , 2)
```

Out[11]:
```
Pregnancies                 3.85
Glucose                   120.89
BloodPressure              69.11
SkinThickness              20.54
Insulin                    79.80
BMI                        31.99
DiabetesPedigreeFunction    0.47
Age                        33.24
Outcome                     0.35
dtype: float64
```

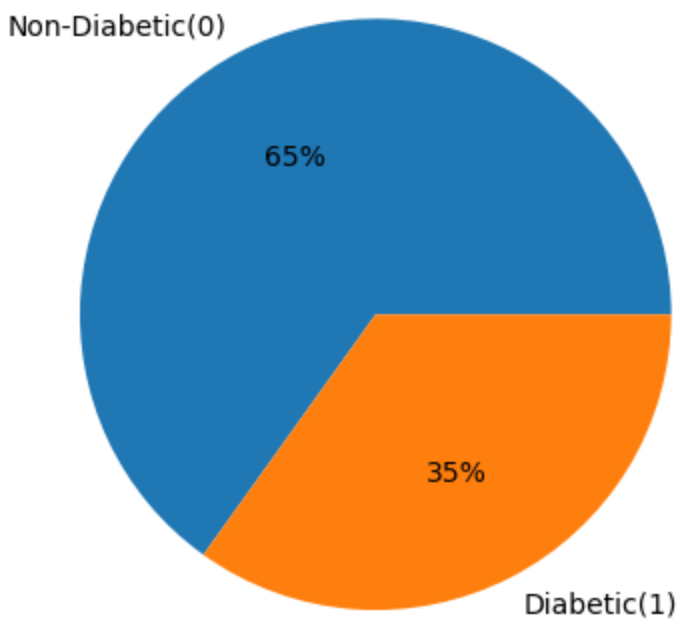
Data Visualization

In [12]:
```
# Display Count Diabetic Patient and Not Diabetic Patients

Data['Outcome'].value_counts(normalize = True) * 100
```

Out[12]:
```
0    65.104167
1    34.895833
Name: Outcome, dtype: float64
```

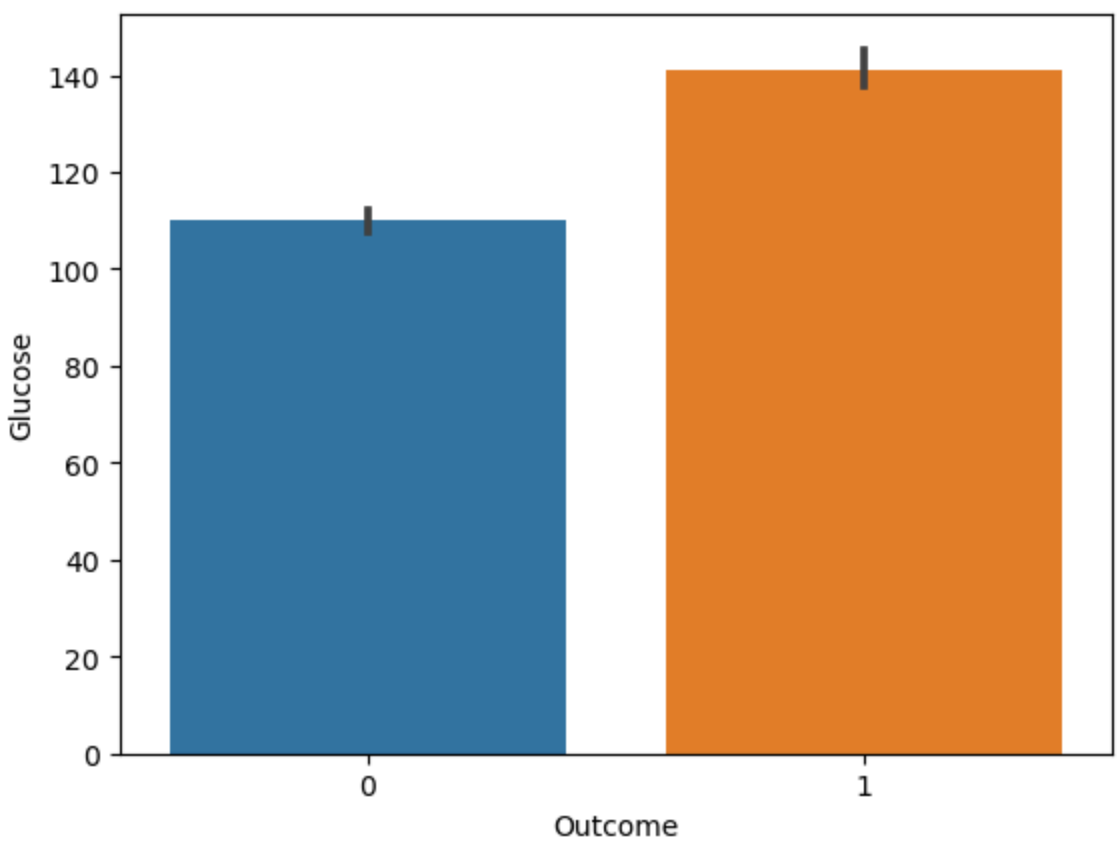
In [13]:
```
plt.pie(Data['Outcome'].value_counts(normalize = True) * 100 , labels = ['Non-Diabetic(0)' , 'Diabetic(1)
```

Out[13]:
```
([<matplotlib.patches.Wedge at 0x13e781066d0>,
  <matplotlib.patches.Wedge at 0x13e78118490>],
 [Text(-0.5025941410457841, 0.9784677457057288, 'Non-Diabetic(0)'),
  Text(0.502593957824251, -0.9784678398182308, 'Diabetic(1)')],
 [Text(-0.27414225875224585, 0.533709679475852, '65%'),
  Text(0.2741421588132278, -0.533709730809944, '35%')])
```



In [14]:
```
# Plot for Glucose and Insulin

sns.barplot(y = 'Glucose' , x = 'Outcome' , data = Data)
```

Out[14]:
```
<Axes: xlabel='Outcome', ylabel='Glucose'>
```



In [15]:
```
Age_C = Data.groupby(['Age Classification', 'Outcome']).size().unstack(fill_value = 0)
Age_C
```

Out[15]:

Outcome	0	1
Age Classification		
Middle Age	113	130
Siniors	20	7
Young Age	367	131

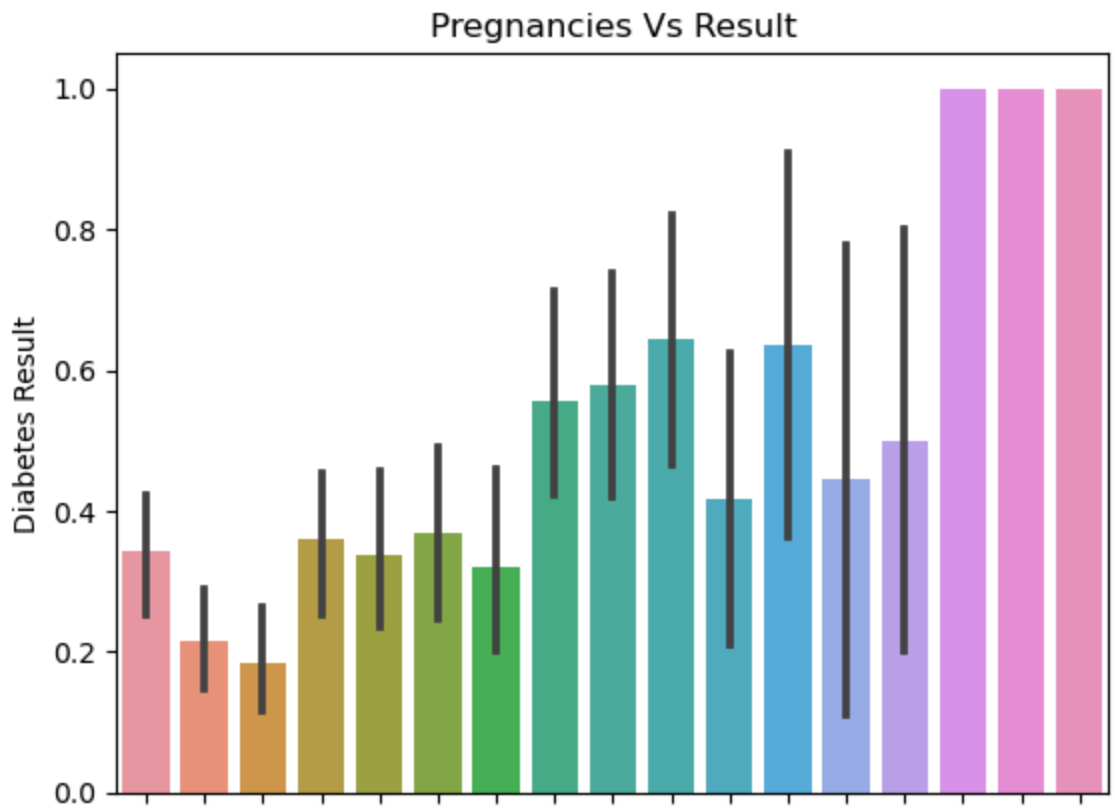
In [16]:
```
BMI_C = Data.groupby(['BMI Classification', 'Outcome']).size().unstack(fill_value = 0)
BMI_C
```

Out[16]:

Outcome	0	1
BMI Classification		
Normal Weight	13	21
Over Weight	0	1
Under Weight	487	246

In [17]:
```
sns.barplot(x = 'Pregnancies' , y = 'Outcome' , data = Data)
plt.xlabel("Total Number of Pregnancies")
plt.ylabel("Diabetes Result")
plt.title("Pregnancies Vs Result")
```

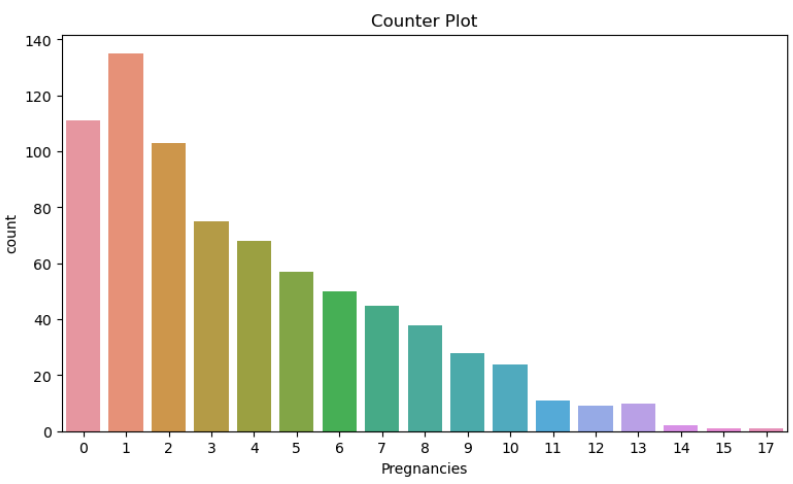
Out[17]: Text(0.5, 1.0, 'Pregnancies Vs Result')

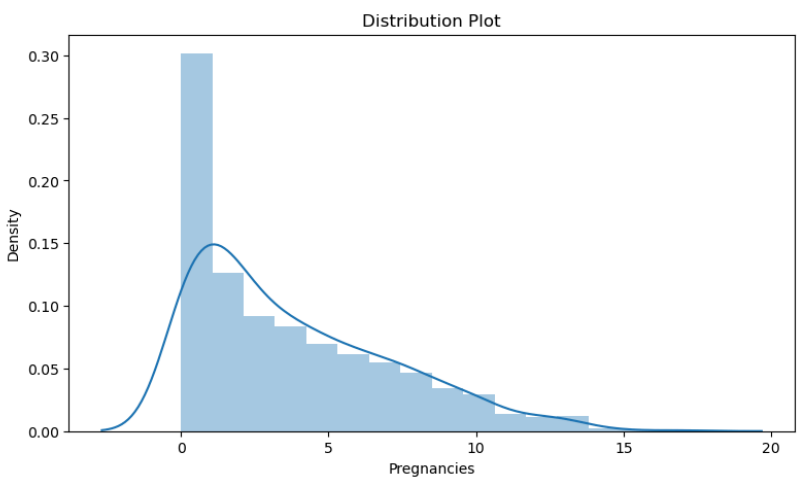


In [18]:
```
plt.figure(figsize = (20,5))

plt.subplot(1,2,1)
plt.title('Counter Plot')
sns.countplot(x = 'Pregnancies',data = Data)

plt.subplot(1,2,2)
plt.title('Distribution Plot')
sns.distplot(Data["Pregnancies"])
```

```
C:\Users\admin\AppData\Local\Temp\ipykernel_12336\913136878.py:9: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751 (https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751)

  sns.distplot(Data["Pregnancies"])
```

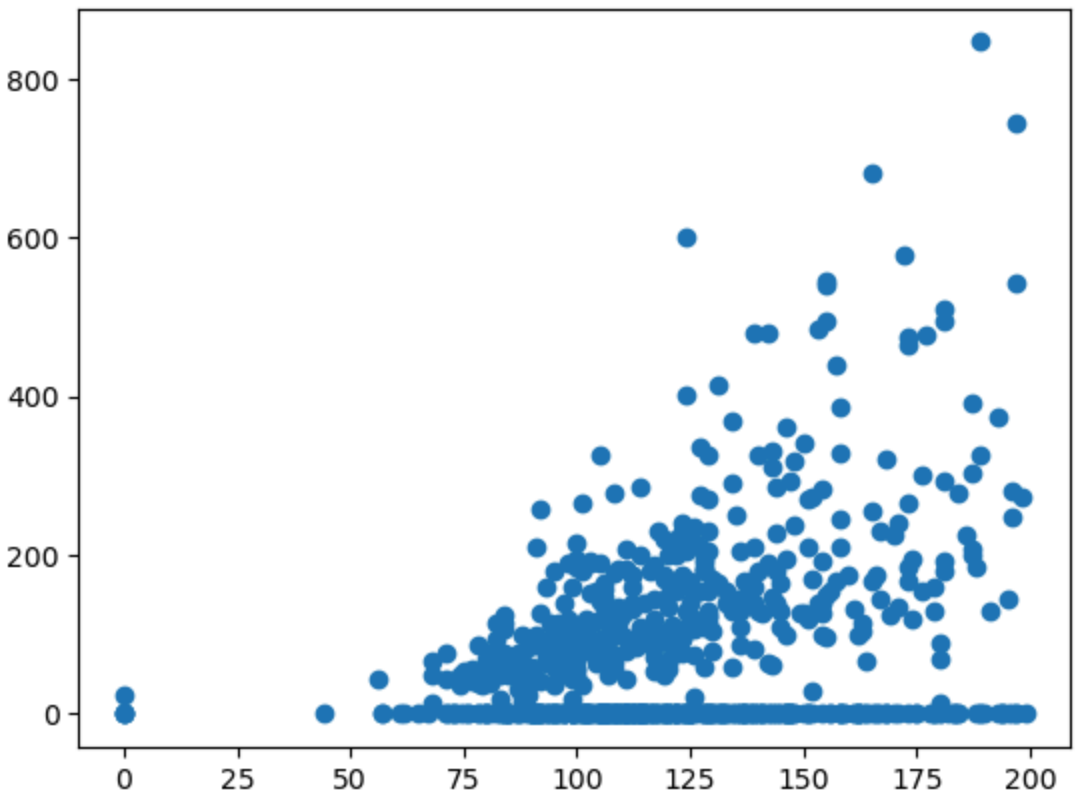
Out[18]: <Axes: title={'center': 'Distribution Plot'}, xlabel='Pregnancies', ylabel='Density'>





In [19]:
```
1 plt.scatter(Data['Glucose'] , Data['Insulin'])
```

Out[19]: <matplotlib.collections.PathCollection at 0x13e78b2ee50>



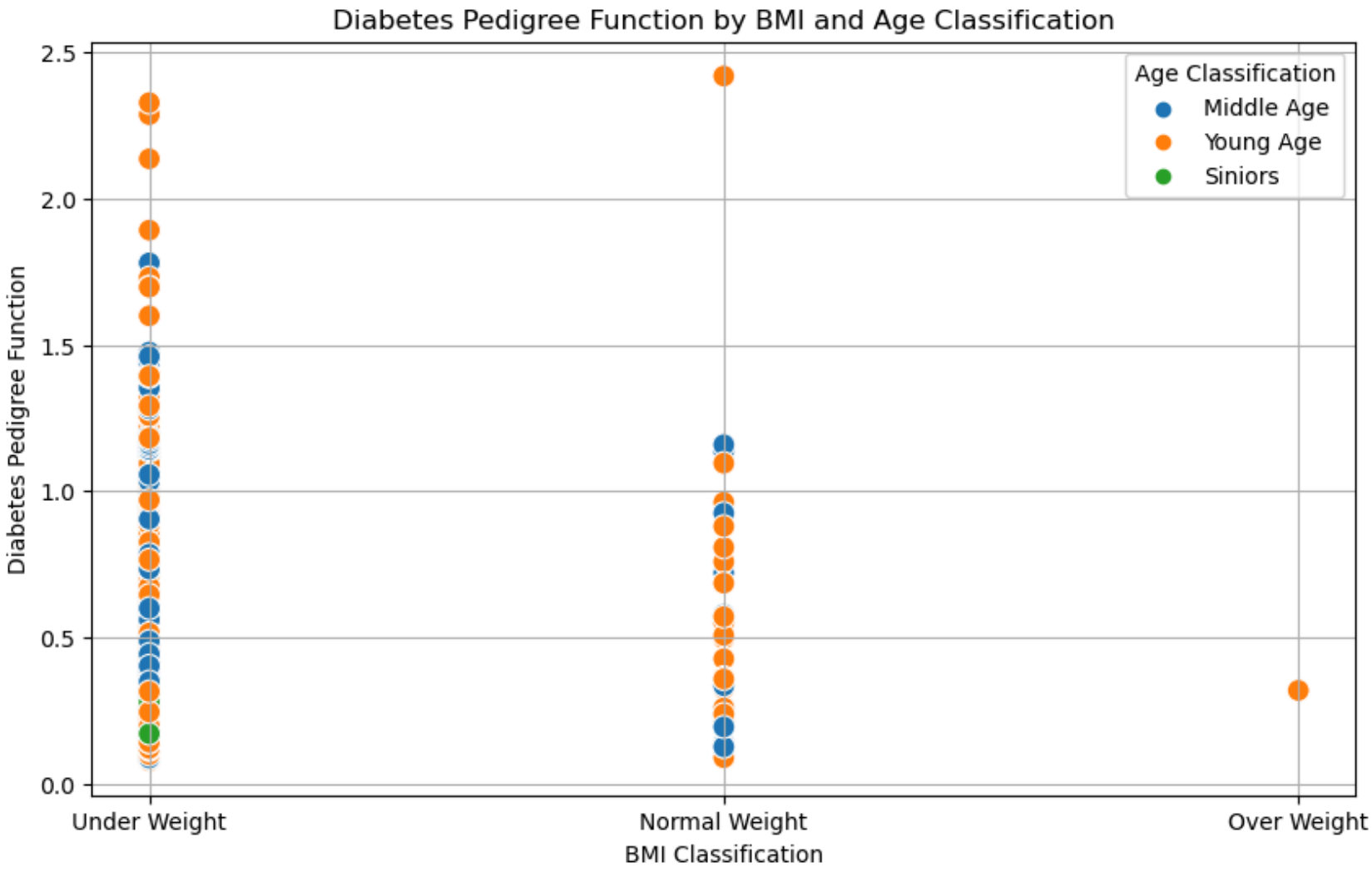
In [20]:
```
1 B = Data.groupby(['BMI Classification', 'Age Classification' , 'DiabetesPedigreeFunction']).size().unstac
2 B
```

Out[20]:

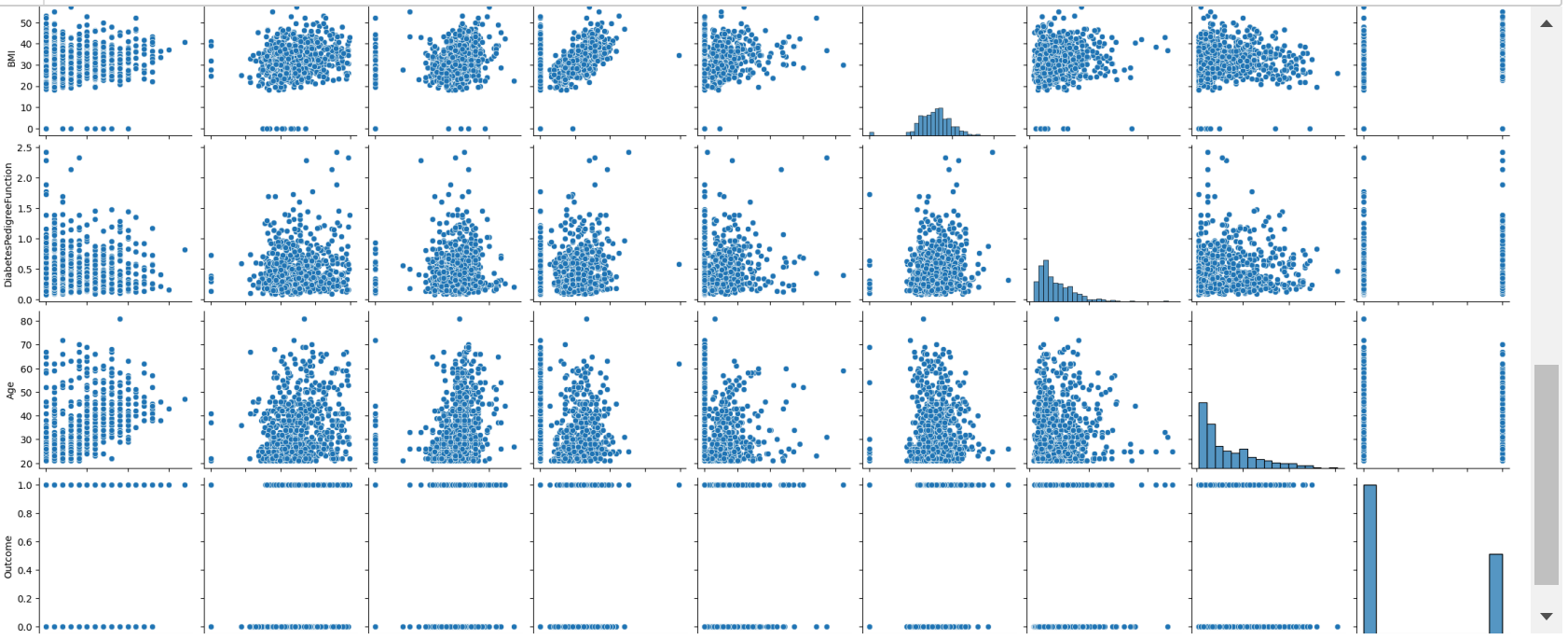
BMI Classification	Normal Weight		Over Weight	Under Weight		
Age Classification	Middle Age	Young Age	Young Age	Middle Age	Siniors	Young Age
DiabetesPedigreeFunction						
0.078	0	0	0	0	0	1
0.084	0	0	0	0	0	1
0.085	0	0	0	1	0	1
0.088	0	0	0	1	0	1
0.089	0	1	0	0	0	0
...	...	...	...	...	...	...
1.893	0	0	0	0	0	1
2.137	0	0	0	0	0	1
2.288	0	0	0	0	0	1
2.329	0	0	0	0	0	1
2.420	0	1	0	0	0	0

517 rows × 6 columns

In [21]:
```
# Plotting
plt.figure(figsize=(10, 6))
sns.scatterplot(x='BMI Classification', y='DiabetesPedigreeFunction', data=Data , hue='Age Classification

# Adding labels and title
plt.xlabel('BMI Classification')
plt.ylabel('Diabetes Pedigree Function')
plt.title('Diabetes Pedigree Function by BMI and Age Classification')

# Show the plot
plt.legend(title='Age Classification')
plt.grid(True)
plt.show()
```



In [22]:
```
sns.pairplot(Data)
```



In [23]:
```
# Checking Data Type of all Parameters

Data.dtypes
```

Out[23]:
```
Pregnancies                 int64
Glucose                     int64
BloodPressure               int64
SkinThickness               int64
Insulin                     int64
BMI                       float64
DiabetesPedigreeFunction  float64
Age                         int64
Outcome                     int64
Age Classification         object
BMI Classification         object
dtype: object
```

Converting Object Data Type into Integer

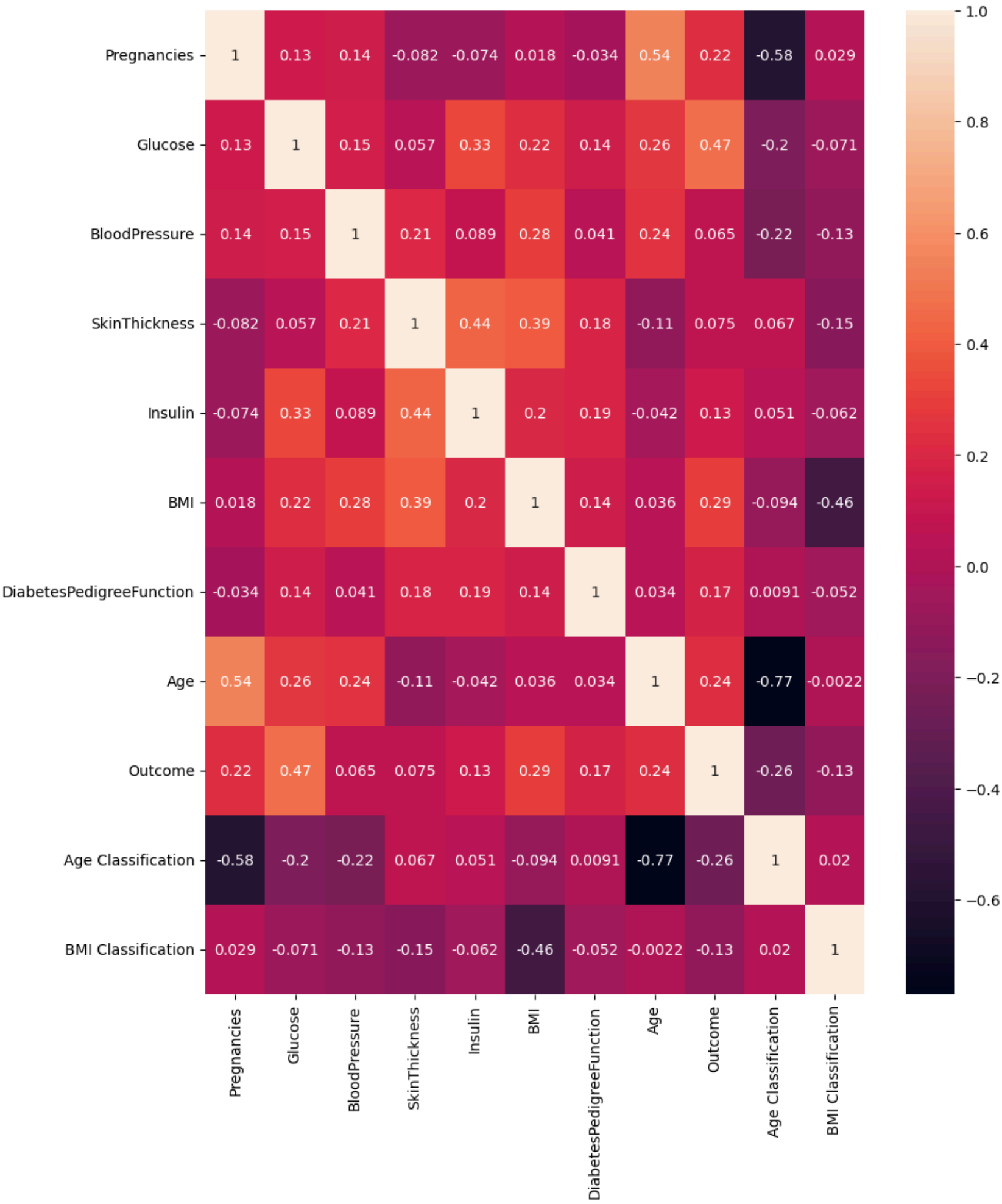
In [24]:
```
from sklearn.preprocessing import LabelEncoder

L = LabelEncoder()

Data['Age Classification']= L.fit_transform(Data['Age Classification'])
Data['BMI Classification'] = L.fit_transform(Data['BMI Classification'])

Data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 11 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  
 9   Age Classification        768 non-null    int32  
 10  BMI Classification        768 non-null    int32  
dtypes: float64(2), int32(2), int64(7)
memory usage: 60.1 KB
```

In [25]:
```
# Correlation between Columns

Corr = Data.corr()
```

In [26]:
```
plt.figure(figsize = (10 , 12))
sns.heatmap(Corr , annot = True)
```

Out[26]: <Axes: >



Machine Learning Models

In [27]:
```
# Splitting Columns into X(Independent Columns) and Y(Dependent Columns)

X = Data.drop(columns = ['Outcome'] , axis = 1)
Y = Data['Outcome']
```

In [28]:
```
# Splitting Splitting Training and Testing Data

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=42 , stratif
```

In [29]:
```
Y.value_counts()
```

Out[29]:
```
0    500
1    268
Name: Outcome, dtype: int64
```

In [30]:
```
# # Train Test Split

print(" X\tY ".center(66))
print ('Train set\t:\t', X_train.shape,  Y_train.shape)
print ('Test set\t:\t', X_test.shape,  Y_test.shape)
```

```
                              X        Y
Train set       :        (614, 10) (614,)
Test set        :        (154, 10) (154,)
```

Importing Machine Learning Algorithms

The following algorithms will be tested on the given dataset and the one with the best performance would be declared suitable:

1. Logistic Regression
2. Random Forest Classifier
3. Decision Tree Classifier
4. K-Nearest Neighbors Classifier

In [31]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score , f1_score , classification_report , confusion_matrix , roc_cu
```

Logistic Regression

In [32]:
```
LR = LogisticRegression()

LR.fit(X_train , Y_train)

Y_true , Y_pred = Y_test , LR.predict(X_test)
```

```
C:\Users\admin\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.py:458: ConvergenceWarning: lbfgs
failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/prep
rocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/
stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

In [33]:
```
print("Accuracy Score\t:\t" , accuracy_score(Y_true , Y_pred) * 100)
print('\n')
print("Classification Report\t:\n" , classification_report(Y_true , Y_pred))
```

```
Accuracy Score  :        71.42857142857143


Classification Report   :
               precision    recall  f1-score   support

           0       0.75      0.83      0.79       100
           1       0.61      0.50      0.55        54

    accuracy                           0.71       154
   macro avg       0.68      0.67      0.67       154
weighted avg       0.71      0.71      0.71       154
```

Random Forest Classifier

In [34]:
```
RF = RandomForestClassifier(n_estimators=100, random_state=42)

RF.fit(X_train, Y_train)

Y_pred = RF.predict(X_test)
```

In [35]:
```
print("Accuracy Score\t:\t" , accuracy_score(Y_true , Y_pred) * 100)
print('\n')
print("Classification Report\t:\n" , classification_report(Y_true , Y_pred))
```

```
Accuracy Score  :        72.72727272727273


Classification Report    :
               precision    recall  f1-score   support

           0       0.77      0.82      0.80       100
           1       0.62      0.56      0.59        54

    accuracy                           0.73       154
   macro avg       0.70      0.69      0.69       154
weighted avg       0.72      0.73      0.72       154
```

Decision Tree Classifier

In [36]:
```
DF = DecisionTreeClassifier()

DF.fit(X_train, Y_train)

Y_pred = DF.predict(X_test)
```

In [37]:
```
print("Accuracy Score\t:\t" , accuracy_score(Y_true , Y_pred) * 100)
print('\n')
print("Classification Report\t:\n" , classification_report(Y_true , Y_pred))
```

```
Accuracy Score  :        74.67532467532467


Classification Report    :
               precision    recall  f1-score   support

           0       0.77      0.87      0.82       100
           1       0.68      0.52      0.59        54

    accuracy                           0.75       154
   macro avg       0.73      0.69      0.70       154
weighted avg       0.74      0.75      0.74       154
```

K-Nearest Neighbors Classifier

In [38]:
```
KNN = KNeighborsClassifier(n_neighbors=5)

KNN.fit(X_train , Y_train)

Y_pred = KNN.predict(X_test)
```

In [39]:
```
print("Accuracy Score\t:\t" , accuracy_score(Y_true , Y_pred) * 100)
print('\n')
print("Classification Report\t:\n" , classification_report(Y_true , Y_pred))
```

```
Accuracy Score  :        66.23376623376623


Classification Report    :
               precision    recall  f1-score   support

           0       0.73      0.76      0.75       100
           1       0.52      0.48      0.50        54

    accuracy                           0.66       154
   macro avg       0.63      0.62      0.62       154
weighted avg       0.66      0.66      0.66       154
```

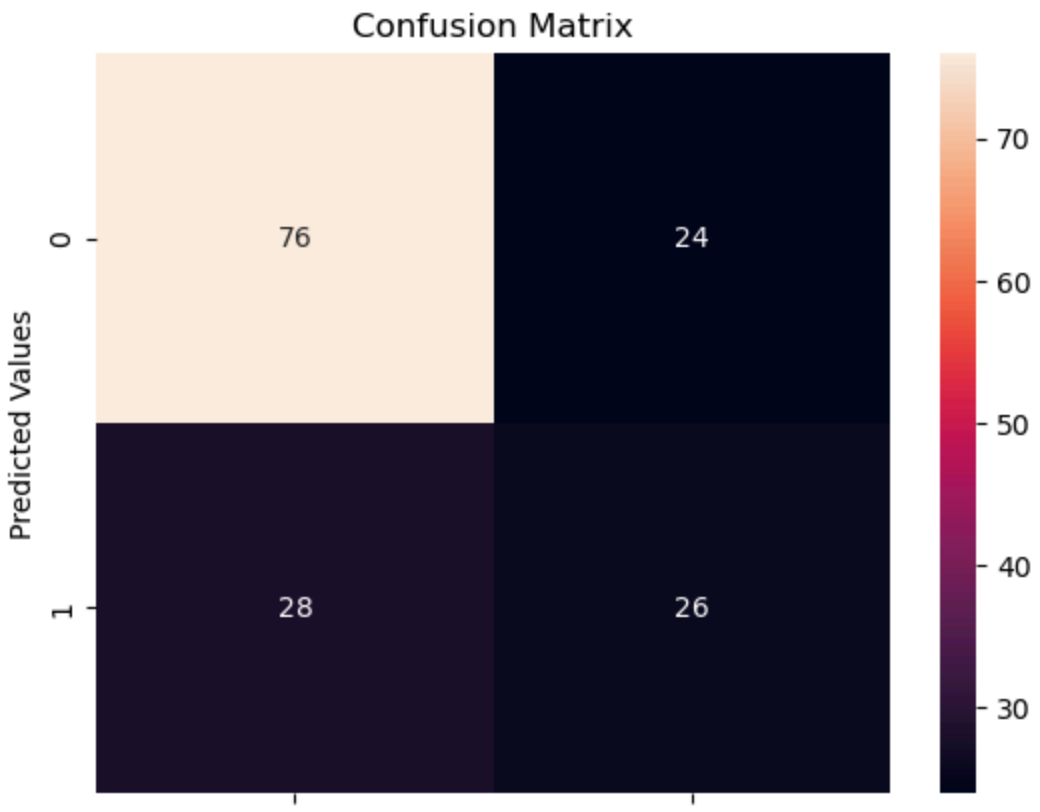
Confusion Matrix

In [40]:
```
confusion_matrix(Y_true , Y_pred) * 100
```

Out[40]:
```
array([[7600, 2400],
       [2800, 2600]], dtype=int64)
```

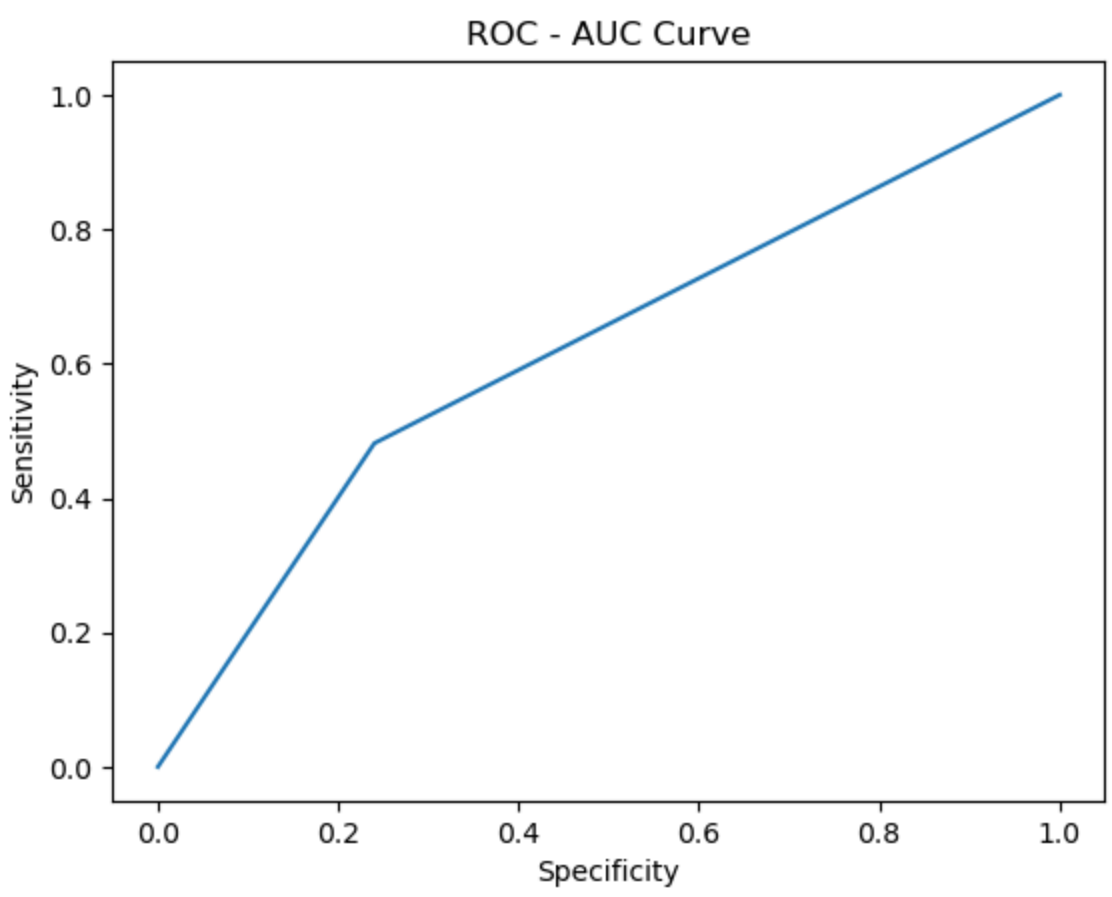
In [41]:
```
sns.heatmap(confusion_matrix(Y_true , Y_pred) , annot = True)
plt.title("Confusion Matrix")
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
```

Out[41]: Text(50.722222222222214, 0.5, 'Predicted Values')



In [42]:
```
tnr , tpr ,_ = roc_curve(Y_test , Y_pred)
plt.plot(tnr , tpr)
plt.title('ROC - AUC Curve')
plt.xlabel('Specificity')
plt.ylabel('Sensitivity')
```

Out[42]: Text(0, 0.5, 'Sensitivity')



In [ ]:
```
```