Diabetes

About Dataset

'Diabetes'

- Pregnancies: Number of times pregnant
- Glucose: Plasma glucose concentration a 2 hours in an oral glucose tolerance test

> - The 2-hour plasma glucose level <140 mg/dL is considered normal.
> - The 2-hour plasma glucose level of 140-199 mg/dL indicates impaired glucose tolerance.
> - The 2-hour plasma glucose level ≥200 mg/dL indicates diabetes.

- BloodPressure: Diastolic blood pressure (mm Hg)

> - The normal range for diastolic blood pressure is usually considered to be between 60 mm Hg and 80 mm Hg for adults.
> - However, blood pressure can vary depending on various factors such as age, health condition, and individual differences.

- SkinThickness: Triceps skin fold thickness (mm)

> Skin Thickness is a measurement of subcutaneous fat stored in the triceps area of the arm. It is commonly used as a clinical measure of body fatness and is part of various health and fitness assessments. As of my last update in September 2021, the American College of Sports Medicine (ACSM) provides reference ranges for skinfold measurements in adults. For example, in males aged 18-29 years, a normal range for triceps skinfold thickness may be around 7.0 to 15.5 millimeters. In females of the same age group, the normal range may be around 12.0 to 20.5 millimeters.

> For females:

> > - Younger adults (20-39 years): 14-28 mm
> > - Middle-aged adults (40-59 years): 16-30 mm
> > - Older adults (60+ years): 16-32 mm

> For males:

- Younger adults (20-39 years): 10-18 mm
- Middle-aged adults (40-59 years): 11-19 mm
- Older adults (60+ years): 12-21 mm

- Insulin: 2-Hour serum insulin (mu U/ml)
  - In general, normal fasting insulin levels are usually in the range of 5 to 20 micro International Units per milliliter ($\mu U/ml$) for adults.
- BMI: Body mass index (weight in kg/(height in m)^2)

The World Health Organization (WHO) and many health organizations define the following BMI categories for adults:

- Underweight: BMI less than 18.5
- Normal weight: BMI from 18.5 to 24.9
- Overweight: BMI from 25 to 29.9
- Obesity (Class 1): BMI from 30 to 34.9
- Obesity (Class 2): BMI from 35 to 39.9
- Extreme Obesity (Class 3): BMI 40 or higher

📝 📝 Purpose of the Project 📝 📝

- Exploratory data analysis (EDA) is one of the most significant methods to examine the data we have. In this project, we are going to explore the hidden patterns in the dataset and extract information from them.

Model implemented:

In the dataset, the target variable has 2 possible outcomes/classes. It is a classification problem.

- 🎯 Binary class Logistic Regression

- 🎯 Decision Tree Classifier

- 🎯 Bagging Classifier

- 🎯 Random Forest Classifier

- 🎯 KNearestNeighbour Classifier

- 🎯 Support Vector Machine

.

Importing Libraries

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

sns.set()
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
```

Necessary Sklean Libraries

In [2]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, ConfusionMatrixDisplay,
from sklearn.datasets import make_classification

from sklearn.tree import DecisionTreeClassifier, plot_tree, export_text
from sklearn.ensemble import RandomForestClassifier, BaggingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
```

Loading and Reading Dataset

In [3]:
```
dataset = 'diabetes.csv'
df = pd.read_csv(dataset)
df.head()
```

Out[3]:

	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

In [4]:
```
for col in df.columns:
    print('='*75)
    print("  "*15,col)
    print('='*75)
    print(df[col].unique())
```

```
================================================================================
                               Pregnancies
================================================================================
[ 6  1  8  0  5  3 10  2  4  7  9 11 13 15 17 12 14]
================================================================================
                                 Glucose
================================================================================
[148  85 183  89 137 116  78 115 197 125 110 168 139 189 166 100 118 107
 103 126  99 196 119 143 147  97 145 117 109 158  88  92 122 138 102  90
 111 180 133 106 171 159 146  71 105 101 176 150  73 187  84  44 141 114
  95 129  79   0  62 131 112 113  74  83 136  80 123  81 134 142 144  93
 163 151  96 155  76 160 124 162 132 120 173 170 128 108 154  57 156 153
 188 152 104  87  75 179 130 194 181 135 184 140 177 164  91 165  86 193
 191 161 167  77 182 157 178  61  98 127  82  72 172  94 175 195  68 186
 198 121  67 174 199  56 169 149  65 190]
================================================================================
                              BloodPressure
================================================================================
[ 72  66  64  40  74  50   0  70  96  92  80  60  84  30  88  90  94  76
  82  75  58  78  68 110  56  62  85  86  48  44  65 108  55 122  54  52
  98 104  95  46 102 100  61  24  38 106 114]
================================================================================
                              SkinThickness
================================================================================
[35 29  0 23 32 45 19 47 38 30 41 33 26 15 36 11 31 37 42 25 18 24 39 27
 21 34 10 60 13 20 22 28 54 40 51 56 14 17 50 44 12 46 16  7 52 43 48  8
 49 63 99]
================================================================================
                                  Insulin
================================================================================
[  0  94 168  88 543 846 175 230  83  96 235 146 115 140 110 245  54 192
 207  70 240  82  36  23 300 342 304 142 128  38 100  90 270  71 125 176
  48  64 228  76 220  40 152  18 135 495  37  51  99 145 225  49  50  92
 325  63 284 119 204 155 485  53 114 105 285 156  78 130  55  58 160 210
 318  44 190 280  87 271 129 120 478  56  32 744 370  45 194 680 402 258
 375 150  67  57 116 278 122 545  75  74 182 360 215 184  42 132 148 180
 205  85 231  29  68  52 255 171  73 108  43 167 249 293  66 465  89 158
  84  72  59  81 196 415 275 165 579 310  61 474 170 277  60  14  95 237
 191 328 250 480 265 193  79  86 326 188 106  65 166 274  77 126 330 600
 185  25  41 272 321 144  15 183  91  46 440 159 540 200 335 387  22 291
 392 178 127 510  16 112]
================================================================================
                                    BMI
================================================================================
[33.6 26.6 23.3 28.1 43.1 25.6 31.  35.3 30.5  0.  37.6 38.  27.1 30.1
 25.8 30.  45.8 29.6 43.3 34.6 39.3 35.4 39.8 29.  36.6 31.1 39.4 23.2
 22.2 34.1 36.  31.6 24.8 19.9 27.6 24.  33.2 32.9 38.2 37.1 34.  40.2
 22.7 45.4 27.4 42.  29.7 28.  39.1 19.4 24.2 24.4 33.7 34.7 23.  37.7
 46.8 40.5 41.5 25.  25.4 32.8 32.5 42.7 19.6 28.9 28.6 43.4 35.1 32.
 24.7 32.6 43.2 22.4 29.3 24.6 48.8 32.4 38.5 26.5 19.1 46.7 23.8 33.9
 20.4 28.7 49.7 39.  26.1 22.5 39.6 29.5 34.3 37.4 33.3 31.2 28.2 53.2
 34.2 26.8 55.  42.9 34.5 27.9 38.3 21.1 33.8 30.8 36.9 39.5 27.3 21.9
 40.6 47.9 50.  25.2 40.9 37.2 44.2 29.9 31.9 28.4 43.5 32.7 67.1 45.
 34.9 27.7 35.9 22.6 33.1 30.4 52.3 24.3 22.9 34.8 30.9 40.1 23.9 37.5
 35.5 42.8 42.6 41.8 35.8 37.8 28.8 23.6 35.7 36.7 45.2 44.  46.2 35.
 43.6 44.1 18.4 29.2 25.9 32.1 36.3 40.  25.1 27.5 45.6 27.8 24.9 25.3
 37.9 27.  26.  38.7 20.8 36.1 30.7 32.3 52.9 21.  39.7 25.5 26.2 19.3
 38.1 23.5 45.5 23.1 39.9 36.8 21.8 41.  42.2 34.4 27.2 36.5 29.8 39.2
 38.4 36.2 48.3 20.  22.3 45.7 23.7 22.1 42.1 42.4 18.2 26.4 45.3 37.
 24.5 32.2 59.4 21.2 26.7 30.2 46.1 41.3 38.8 35.2 42.3 40.7 46.5 33.5
 37.3 30.3 26.3 21.7 36.4 28.5 26.9 38.6 31.3 19.5 20.1 40.8 23.4 28.3
 38.9 57.3 35.6 49.6 44.6 24.1 44.5 41.2 49.3 46.3]
================================================================================
                         DiabetesPedigreeFunction
================================================================================
[0.627 0.351 0.672 0.167 2.288 0.201 0.248 0.134 0.158 0.232 0.191 0.537
 1.441 0.398 0.587 0.484 0.551 0.254 0.183 0.529 0.704 0.388 0.451 0.263
 0.205 0.257 0.487 0.245 0.337 0.546 0.851 0.267 0.188 0.512 0.966 0.42
 0.665 0.503 1.39  0.271 0.696 0.235 0.721 0.294 1.893 0.564 0.586 0.344
 0.305 0.491 0.526 0.342 0.467 0.718 0.962 1.781 0.173 0.304 0.27  0.699
 0.258 0.203 0.855 0.845 0.334 0.189 0.867 0.411 0.583 0.231 0.396 0.14
 0.391 0.37  0.307 0.102 0.767 0.237 0.227 0.698 0.178 0.324 0.153 0.165
 0.443 0.261 0.277 0.761 0.255 0.13  0.323 0.356 0.325 1.222 0.179 0.262
 0.283 0.93  0.801 0.207 0.287 0.336 0.247 0.199 0.543 0.192 0.588 0.539
 0.22  0.654 0.223 0.759 0.26  0.404 0.186 0.278 0.496 0.452 0.403 0.741
 0.361 1.114 0.457 0.647 0.088 0.597 0.532 0.703 0.159 0.268 0.286 0.318
 0.272 0.572 0.096 1.4   0.218 0.085 0.399 0.432 1.189 0.687 0.137 0.637
 0.833 0.229 0.817 0.204 0.368 0.743 0.722 0.256 0.709 0.471 0.495 0.18
 0.542 0.773 0.678 0.719 0.382 0.319 0.19  0.956 0.084 0.725 0.299 0.244
 0.745 0.615 1.321 0.64  0.142 0.374 0.383 0.578 0.136 0.395 0.187 0.905
 0.15  0.874 0.236 0.787 0.407 0.605 0.151 0.289 0.355 0.29  0.375 0.164
 0.431 0.742 0.514 0.464 1.224 1.072 0.805 0.209 0.666 0.101 0.198 0.652
 2.329 0.089 0.645 0.238 0.394 0.293 0.479 0.686 0.831 0.582 0.446 0.402
 1.318 0.329 1.213 0.427 0.282 0.143 0.38  0.284 0.249 0.926 0.557 0.092
 0.655 1.353 0.612 0.2   0.226 0.997 0.933 1.101 0.078 0.24  1.136 0.128
 0.422 0.251 0.677 0.296 0.454 0.744 0.881 0.28  0.259 0.619 0.808 0.34
 0.434 0.757 0.613 0.692 0.52  0.412 0.84  0.839 0.156 0.215 0.326 1.391
 0.875 0.313 0.433 0.626 1.127 0.315 0.345 0.129 0.527 0.197 0.731 0.148
 0.123 0.127 0.122 1.476 0.166 0.932 0.343 0.893 0.331 0.472 0.673 0.389
 0.485 0.349 0.279 0.346 0.252 0.243 0.58  0.559 0.302 0.569 0.378 0.385
 0.499 0.306 0.234 2.137 1.731 0.545 0.225 0.816 0.528 0.509 1.021 0.821
```

```
 0.947 1.268 0.221 0.66  0.239 0.949 0.444 0.463 0.803 1.6   0.944 0.196
 0.241 0.161 0.135 0.376 1.191 0.702 0.674 1.076 0.534 1.095 0.554 0.624
 0.219 0.507 0.561 0.421 0.516 0.264 0.328 0.233 0.108 1.138 0.147 0.727
 0.435 0.497 0.23  0.955 2.42  0.658 0.33  0.51  0.285 0.415 0.381 0.832
 0.498 0.212 0.364 1.001 0.46  0.733 0.416 0.705 1.022 0.269 0.6   0.571
 0.607 0.17  0.21  0.126 0.711 0.466 0.162 0.419 0.63  0.365 0.536 1.159
 0.629 0.292 0.145 1.144 0.174 0.547 0.163 0.738 0.314 0.968 0.409 0.297
 0.525 0.154 0.771 0.107 0.493 0.717 0.917 0.501 1.251 0.735 0.804 0.661
 0.549 0.825 0.423 1.034 0.16  0.341 0.68  0.591 0.3   0.121 0.502 0.401
 0.601 0.748 0.338 0.43  0.892 0.813 0.693 0.575 0.371 0.206 0.417 1.154
 0.925 0.175 1.699 0.682 0.194 0.4   0.1   1.258 0.482 0.138 0.593 0.878
 0.157 1.282 0.141 0.246 1.698 1.461 0.347 0.362 0.393 0.144 0.732 0.115
 0.465 0.649 0.871 0.149 0.695 0.303 0.61  0.73  0.447 0.455 0.133 0.155
 1.162 1.292 0.182 1.394 0.217 0.631 0.88  0.614 0.332 0.366 0.181 0.828
 0.335 0.856 0.886 0.439 0.253 0.598 0.904 0.483 0.565 0.118 0.177 0.176
 0.295 0.441 0.352 0.826 0.97  0.595 0.317 0.265 0.646 0.426 0.56  0.515
 0.453 0.785 0.734 1.174 0.488 0.358 1.096 0.408 1.182 0.222 1.057 0.766
 0.171]
================================================================================
                                 Age
================================================================================
[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]
================================================================================
                                Outcome
================================================================================
[1 0]
```

In [5]:
```
desc = df.describe().round(2).T
desc[['min','max']]
```

Out[5]:

	min	max
Pregnancies	0.00	17.00
Glucose	0.00	199.00
BloodPressure	0.00	122.00
SkinThickness	0.00	99.00
Insulin	0.00	846.00
BMI	0.00	67.10
DiabetesPedigreeFunction	0.08	2.42
Age	21.00	81.00
Outcome	0.00	1.00

📌In the dataset min =0 for `Glucose`, `Blood Pressure` , `Skin Thickness`, `Insulin`, `BMI` are impossible values. So they should be treated as null values.

In [6]:
```
for col in df.iloc[:,:-1].columns:
    print(col, df[df[col]==0][col].value_counts())
```

```
Pregnancies 0    111
Name: Pregnancies, dtype: int64
Glucose 0    5
Name: Glucose, dtype: int64
BloodPressure 0    35
Name: BloodPressure, dtype: int64
SkinThickness 0    227
Name: SkinThickness, dtype: int64
Insulin 0    374
Name: Insulin, dtype: int64
BMI 0.0    11
Name: BMI, dtype: int64
DiabetesPedigreeFunction Series([], Name: DiabetesPedigreeFunction, dtype: int64)
Age Series([], Name: Age, dtype: int64)
```

- The 0 values in Glucose, BloodPressure, SkinThickness, Insulin and BMI are unnatural inputs. It should be considered as null values.

In [7]:
```
null_col = ['Glucose', 'BloodPressure', 'SkinThickness','Insulin', 'BMI' ]
for col in null_col:
    df[col].replace(0, np.nan, inplace=True)
```

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   763 non-null    float64
 2   BloodPressure             733 non-null    float64
 3   SkinThickness             541 non-null    float64
 4   Insulin                   394 non-null    float64
 5   BMI                       757 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(6), int64(3)
memory usage: 54.1 KB
```

In [9]:
```
from summarytools import dfSummary
dfSummary(df)
```

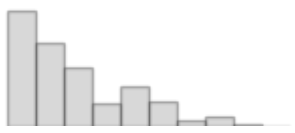
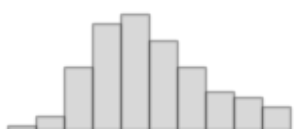
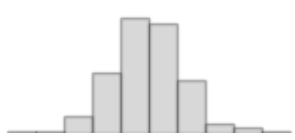
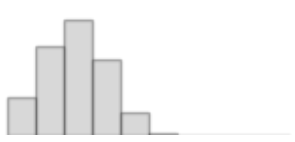
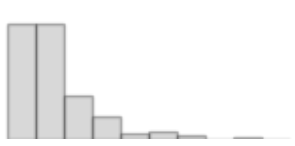
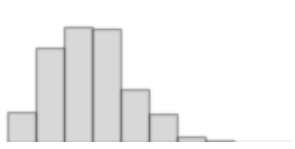
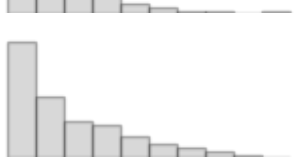
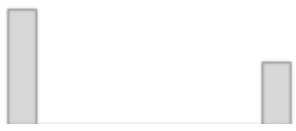
Out[9]:

Data Frame Summary

df

Dimensions: 768 x 9

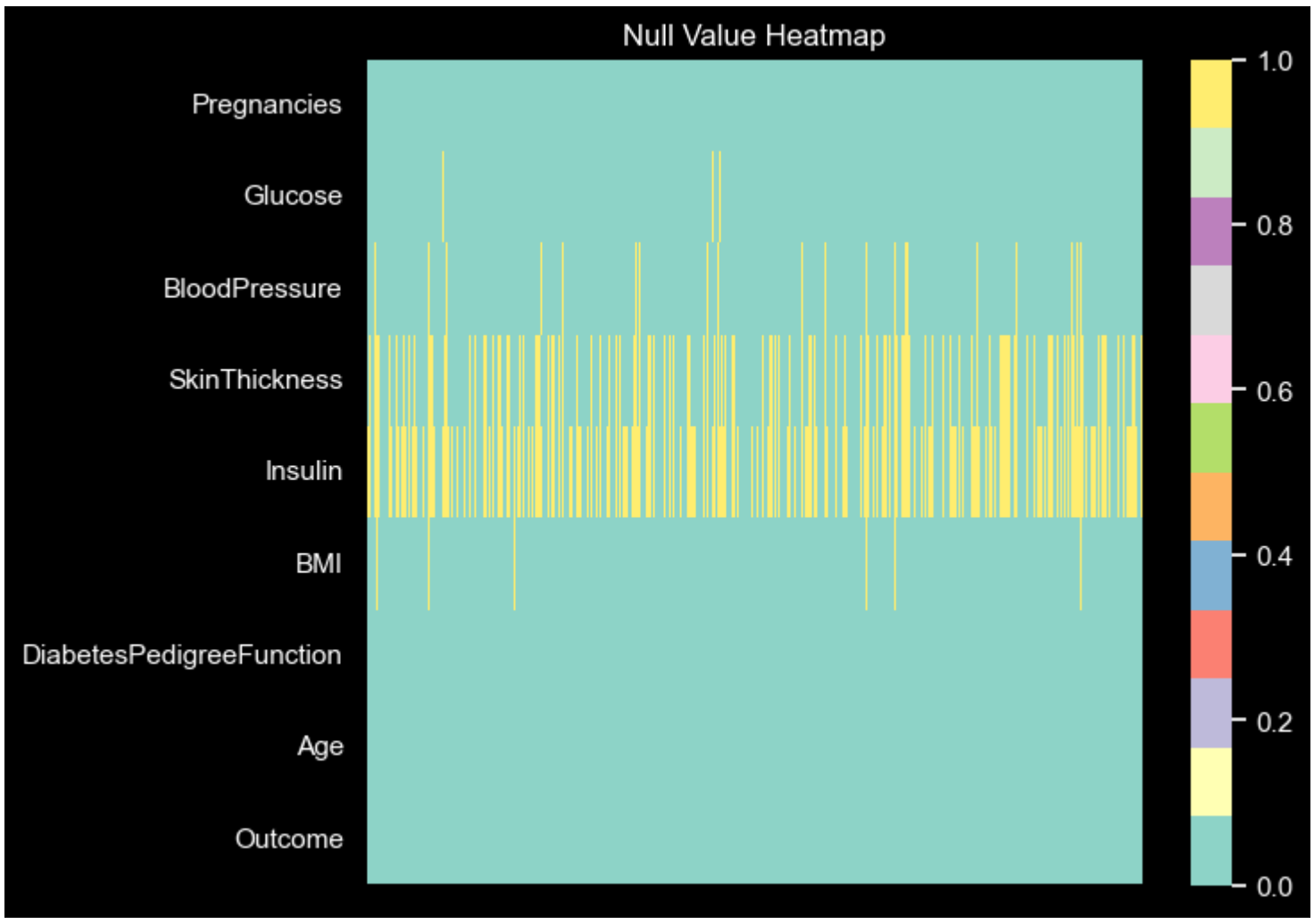
Duplicates: 0

No	Variable	Stats / Values	Freqs / (% of Valid)	Graph	Missing
1	**Pregnancies** [int64]	Mean (sd) : 3.8 (3.4) min < med < max: 0.0 < 3.0 < 17.0 IQR (CV) : 5.0 (1.1)	17 distinct values		0 (0.0%)
2	**Glucose** [float64]	Mean (sd) : 121.7 (30.5) min < med < max: 44.0 < 117.0 < 199.0 IQR (CV) : 42.0 (4.0)	135 distinct values		5 (0.7%)
3	**BloodPressure** [float64]	Mean (sd) : 72.4 (12.4) min < med < max: 24.0 < 72.0 < 122.0 IQR (CV) : 16.0 (5.8)	46 distinct values		35 (4.6%)
4	**SkinThickness** [float64]	Mean (sd) : 29.2 (10.5) min < med < max: 7.0 < 29.0 < 99.0 IQR (CV) : 14.0 (2.8)	50 distinct values		227 (29.6%)
5	**Insulin** [float64]	Mean (sd) : 155.5 (118.8) min < med < max: 14.0 < 125.0 < 846.0 IQR (CV) : 113.8 (1.3)	185 distinct values		374 (48.7%)
6	**BMI** [float64]	Mean (sd) : 32.5 (6.9) min < med < max: 18.2 < 32.3 < 67.1 IQR (CV) : 9.1 (4.7)	247 distinct values		11 (1.4%)
7	**DiabetesPedigreeFunction** [float64]	Mean (sd) : 0.5 (0.3) min < med < max: 0.1 < 0.4 < 2.4 IQR (CV) : 0.4 (1.4)	517 distinct values		0 (0.0%)
8	**Age** [int64]	Mean (sd) : 33.2 (11.8) min < med < max: 21.0 < 29.0 < 81.0 IQR (CV) : 17.0 (2.8)	52 distinct values		0 (0.0%)
9	**Outcome** [int64]	Mean (sd) : 0.3 (0.5) min < med < max: 0.0 < 0.0 < 1.0 IQR (CV) : 1.0 (0.7)	2 distinct values		0 (0.0%)

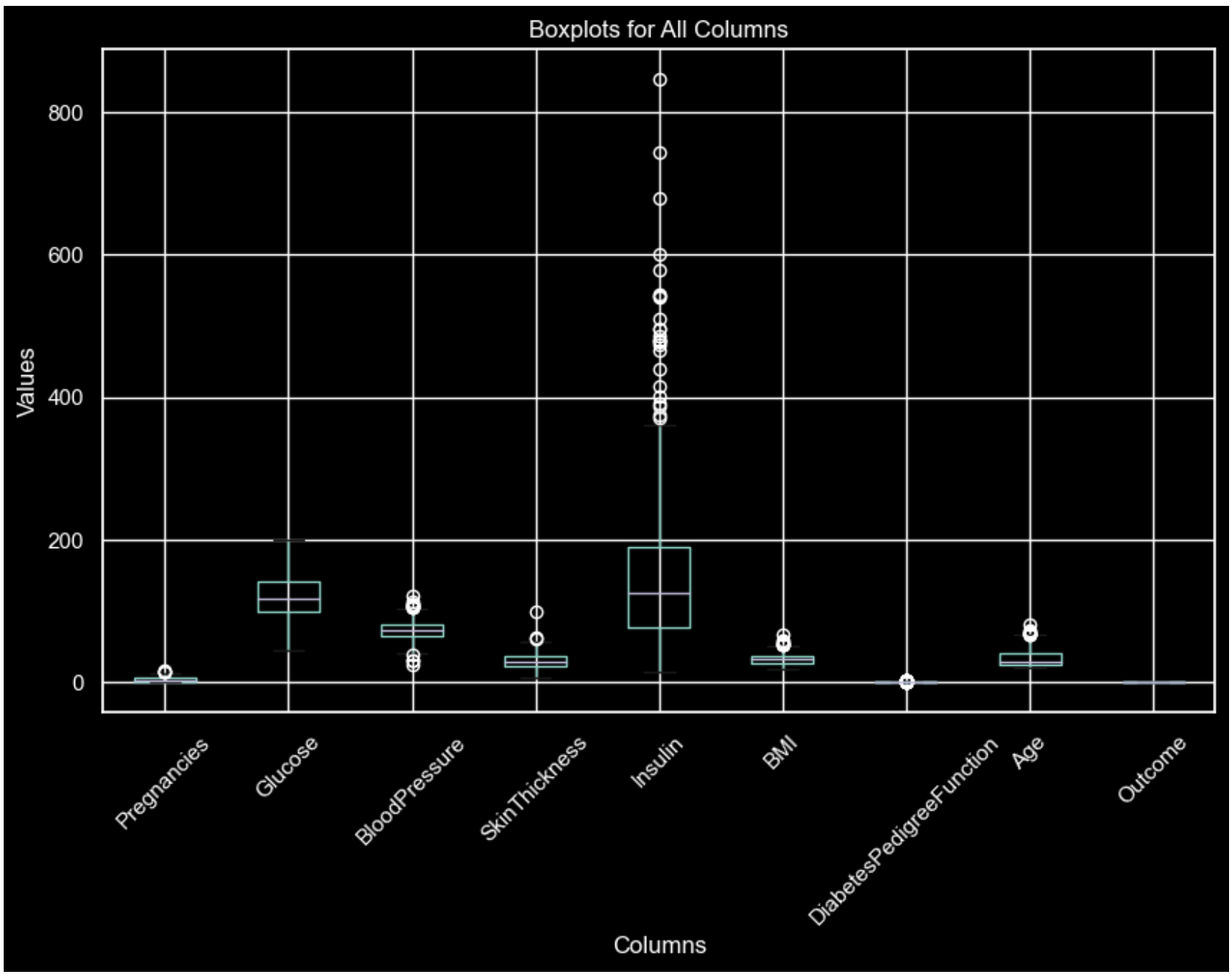
Treating Null values

Checking null values

In [10]:
```
# Create a heatmap of null values
plt.style.use('dark_background')
plt.figure(figsize=(7, 6))
sns.heatmap(df.isnull().T, cmap='Set3', cbar=True, xticklabels= False)
plt.title('Null Value Heatmap')

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



In [11]:
```
num_col = df.select_dtypes(exclude='object').columns.tolist()

plt.figure(figsize=(10, 6))  # Adjust the figure size as needed

df.boxplot(column=num_col)
plt.title('Boxplots for All Columns')
plt.xticks(rotation=45)  # Rotate x-axis labels for better visibility
plt.xlabel('Columns')
plt.ylabel('Values')
plt.grid(True)
plt.show()
```



Checking Null Percentage

In [12]:
```
null_details = {'col_name' : df.columns.tolist(),
                'col_type' : [df[col].dtype for col in df.columns],
                'null_col(%)' :  [df[col].isnull().sum()*100/len(df) for col in df.columns]}

df_null_details = pd.DataFrame(null_details)
df_null_details
```

Out[12]:

	col_name	col_type	null_col(%)
0	Pregnancies	int64	0.000000
1	Glucose	float64	0.651042
2	BloodPressure	float64	4.557292
3	SkinThickness	float64	29.557292
4	Insulin	float64	48.697917
5	BMI	float64	1.432292
6	DiabetesPedigreeFunction	float64	0.000000
7	Age	int64	0.000000
8	Outcome	int64	0.000000

Although for 'SkinThickness' and 'Insulin' the null percentage is 29 and 48 respectively, but they are significant for diabetes analysis. So better to go with imputation method. The distribution is skewed, so median imputaion approach is advisable.

In [13]:
```
#
desc.iloc[:-1,:][['mean','50%']]
```

Out[13]:

	mean	50%
Pregnancies	3.85	3.00
Glucose	120.89	117.00
BloodPressure	69.11	72.00
SkinThickness	20.54	23.00
Insulin	79.80	30.50
BMI	31.99	32.00
DiabetesPedigreeFunction	0.47	0.37
Age	33.24	29.00

In [14]:
```
# filling up null value with median

for col in null_col:
    df[col].fillna(df[col].median(), inplace = True)

#df.drop('SkinThickness', axis=1, inplace =True)

df.isnull().sum()
```

Out[14]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

Descriptive Statistics

In [15]:
```
desc =df.describe().T
def descriptive_stats(df):

    plt.style.use('dark_background')
    plt.figure(figsize=(14,6))
    sns.heatmap(df, annot=True, cmap='rainbow', fmt=".2f")
    plt.xticks(size = 12)
    plt.yticks(size = 12, rotation = 0)
    plt.title('Statistical Description')
    plt.show()

descriptive_stats(desc)
```

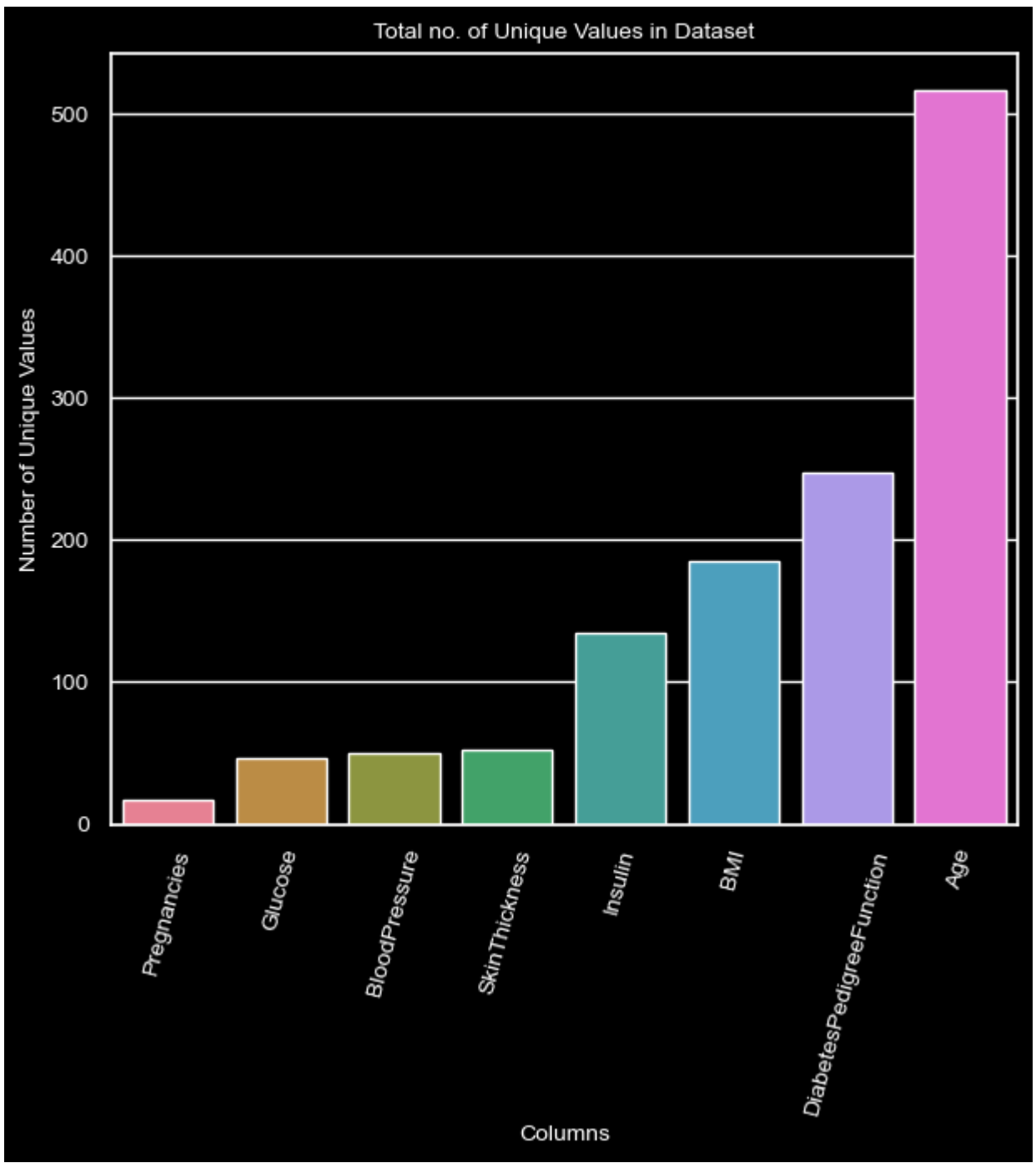
Checking and Removing Duplicates

In [16]:
```
# Remove duplicates

def drop_dup(df):
    if df.duplicated().any() == True:
        print('The total duplicate row before removing duplicate:', df.duplicated().sum())
        df.drop_duplicates(inplace=True , keep = 'last')  # Remove duplicates
        df = df.reset_index(drop=True) #Reset the index
        print('The total duplicate row after removing duplicate:', df.duplicated().sum(), ' \nshape of datase
    else:
        return 'No duplicate entries'
drop_dup(df)
```

Out[16]: 'No duplicate entries'

In [17]:
```
# Count the number of unique values in each column
def check_unquie_count(df):
    unique_counts = df.nunique()
    print('=='*30)
    print('  '*10, 'Total no. of Unique Values')
    print('=='*30)
    print(unique_counts.sort_values())
    print('=='*30)
# Create a bar plot or count plot of unique values
    plt.style.use('dark_background')
    plt.figure(figsize=(7, 6))
    sns.barplot(x=unique_counts.index, y=unique_counts.sort_values(),palette='husl' )

    plt.xticks(rotation=75, fontsize= 10)
    plt.yticks( fontsize= 10 )
    plt.xlabel('Columns',fontsize=10)
    plt.ylabel('Number of Unique Values', fontsize=10)
    plt.title('Total no. of Unique Values in Dataset', fontsize=10)

# Display the plot
    plt.show()

check_unquie_count(df.iloc[:,0:-1])
```

```
============================================================
                    Total no. of Unique Values
============================================================
Pregnancies                  17
BloodPressure                46
SkinThickness                50
Age                          52
Glucose                     135
Insulin                     185
BMI                         247
DiabetesPedigreeFunction    517
dtype: int64
============================================================
```



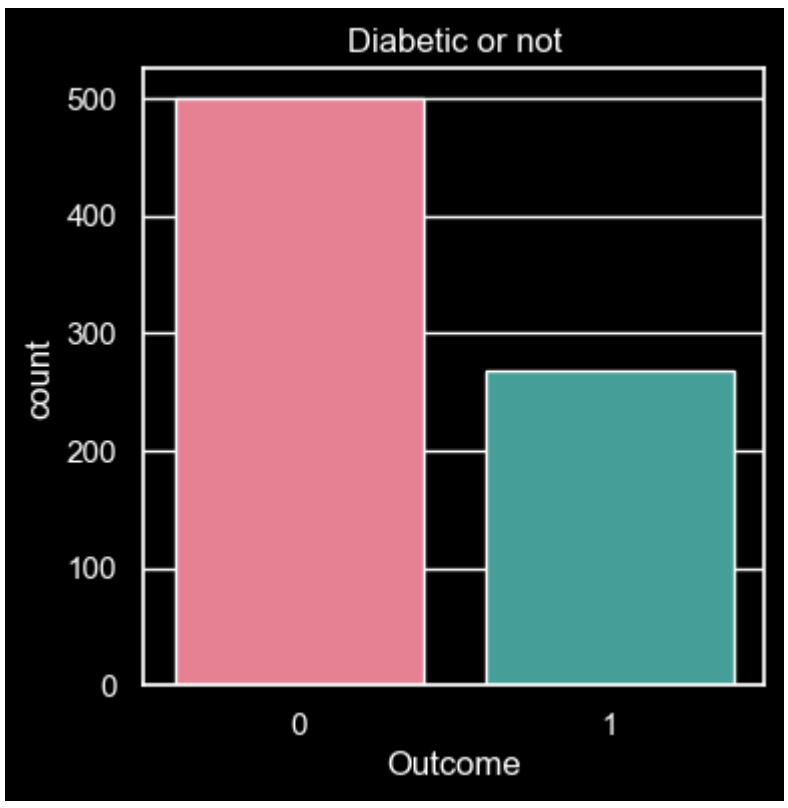
Checking Imbalance between two classes

In [18]:
```
vc = df['Outcome'].value_counts()
print('Values and no of values in target feature\n' , vc)
if vc.values[0] > vc.values[1] :
    if vc.values[0] > 2*vc.values[1]:
        print('The imbalance between classes found')
    else:
        print('No or littel imbalancy found')
elif vc.values[0] < vc.values[1]:
    if vc.values[0] < 2*vc.values[1]:
        print('The imbalance between classes found')
    else:
          print('No or littel imbalancy found')
else:
      print('No or littel imbalancy found')
```

```
Values and no of values in target feature
 0    500
1    268
Name: Outcome, dtype: int64
No or littel imbalancy found
```

Here 0 and 1 corresponds to non-diabetic and diabetic respectively.

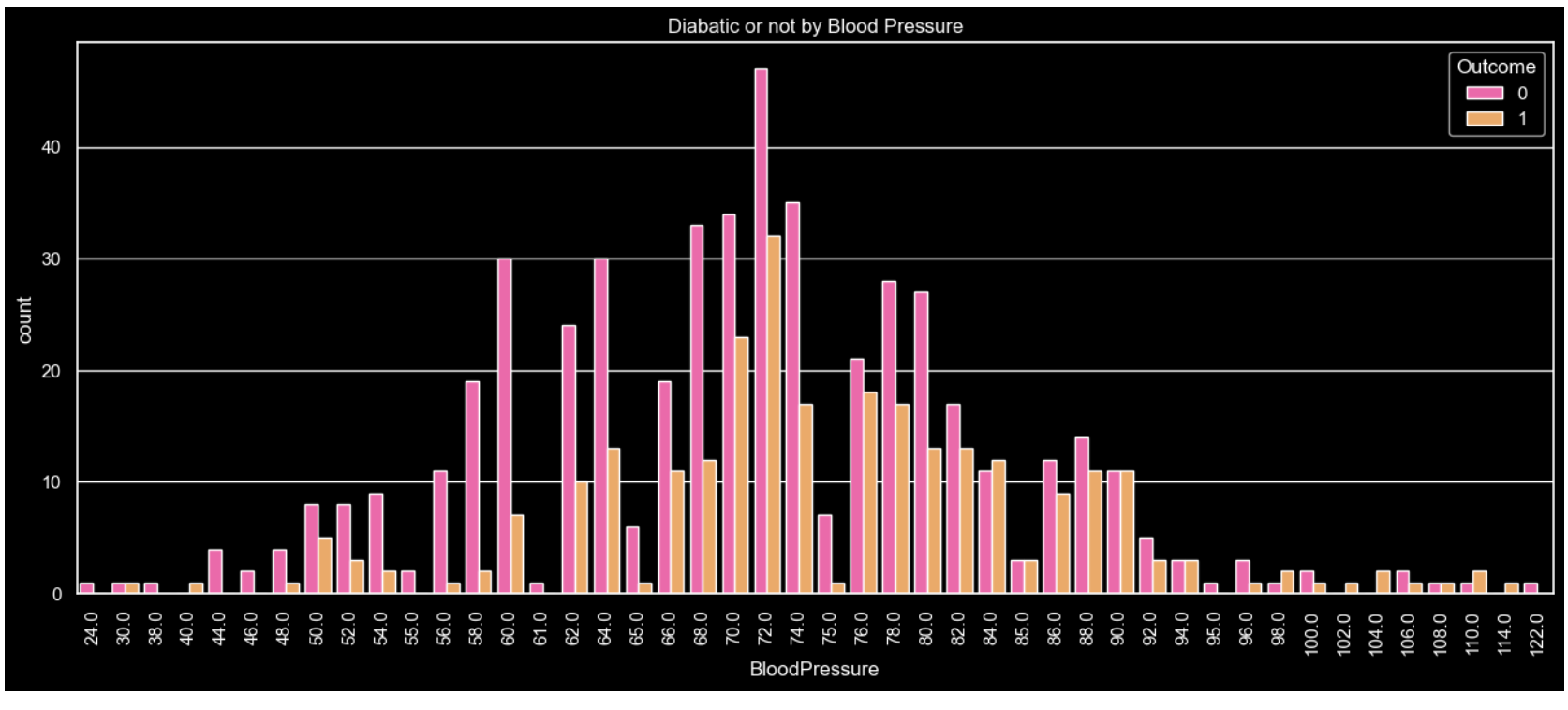
In [19]:
```
plt.style.use('dark_background')
plt.figure(figsize = (4,4))
ax = sns.countplot(x= df.Outcome , data= df, palette ='husl')
ax.set_title('Diabetic or not')
plt.show()
```



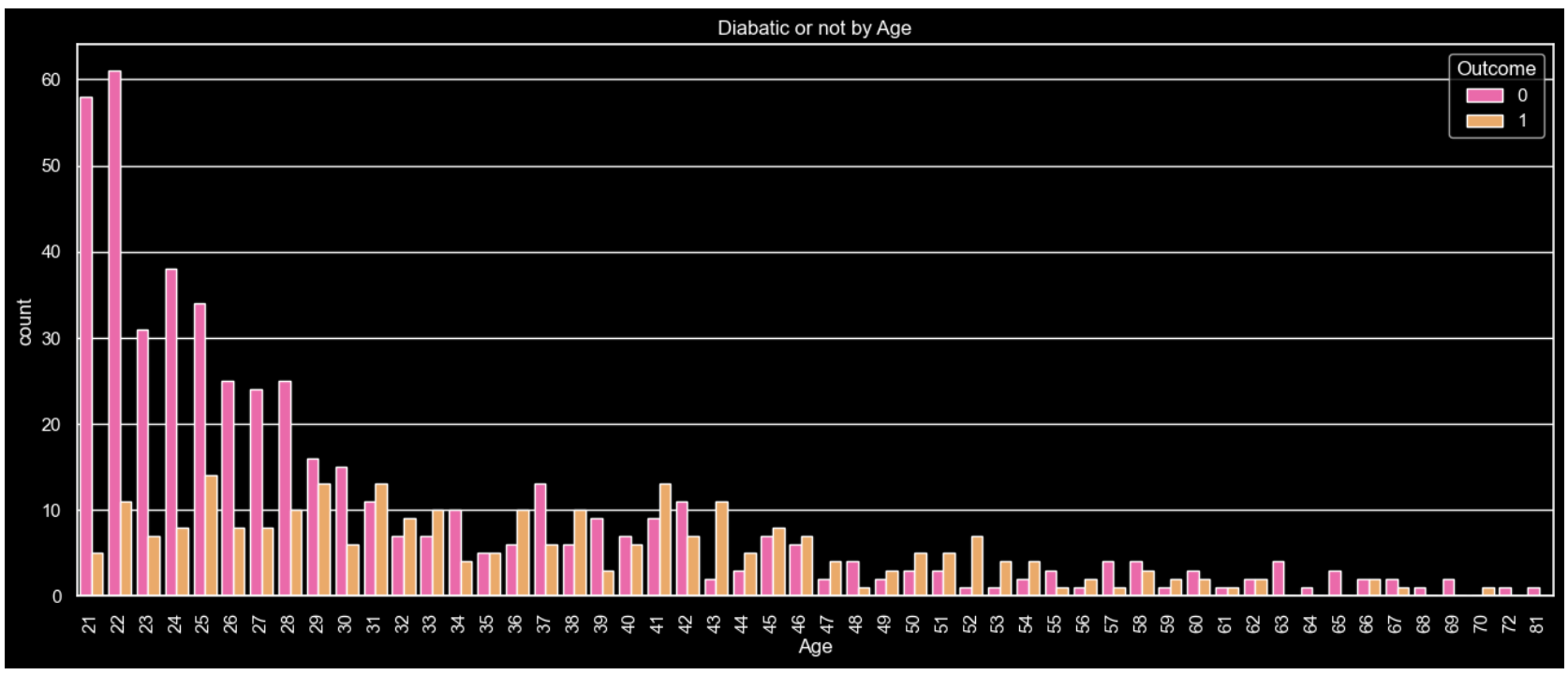
 # majority class $\leq$ 2 $\times$ # minority class

>>** A little Imbalancy in dataset is visualized**<<

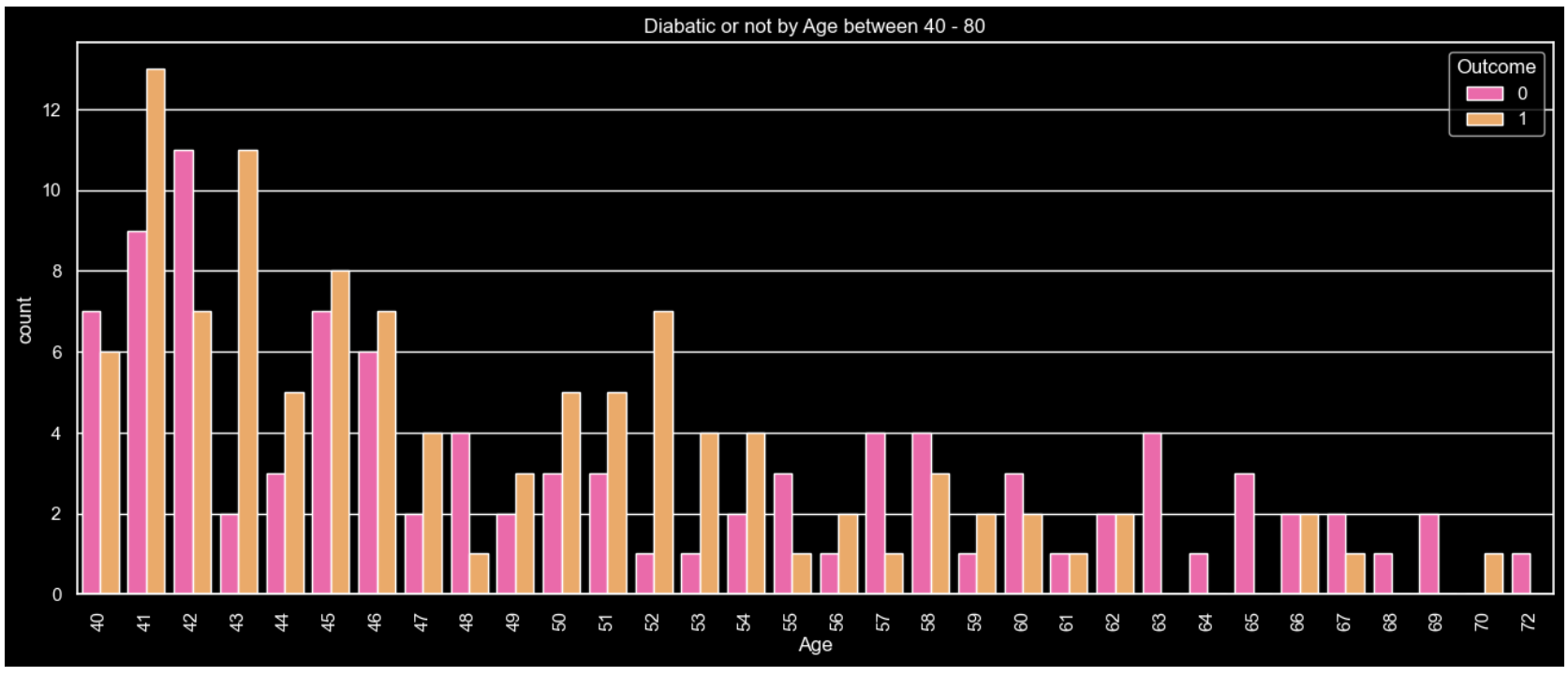
In [20]:
```
plt.style.use('dark_background')
plt.figure(figsize=(16,6))
plt.xticks(rotation = 90)
ax = sns.countplot(x= df['BloodPressure'] , hue= df['Outcome'], palette ='spring')
ax.set_title('Diabatic or not by Blood Pressure')
plt.show()
```



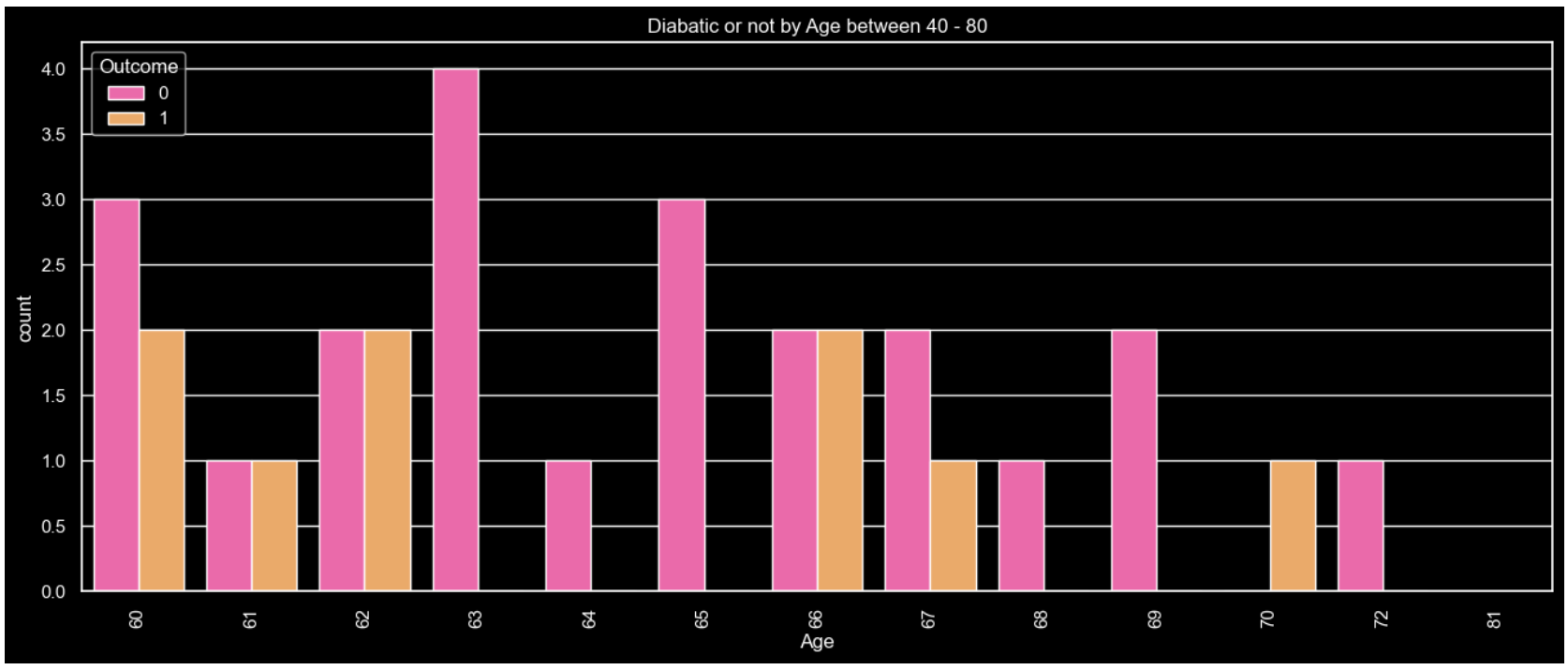
In [21]:
```
plt.style.use('dark_background')
plt.figure(figsize=(16,6))
plt.xticks(rotation = 90)
ax = sns.countplot(x= df['Age'] , hue= df['Outcome'], palette ='spring')
ax.set_title('Diabatic or not by Age')
plt.show()
```



In [22]:
```
age_diabetic = df[(df['Age']<=80) & (df['Age']>=40)][['Age','Outcome']]
plt.figure(figsize=(16,6))
plt.xticks(rotation = 90)
ax = sns.countplot(x= age_diabetic['Age'] , hue= age_diabetic['Outcome'], palette ='spring')
ax.set_title('Diabatic or not by Age between 40 - 80')
plt.show()
```



In [23]:
```
age_diabetic_sr = df[(df['Age']>=60)][['Age','Outcome']]
plt.figure(figsize=(16,6))
plt.xticks(rotation = 90)
ax = sns.countplot(x= age_diabetic_sr['Age'] , hue= age_diabetic['Outcome'], palette ='spring')
ax.set_title('Diabatic or not by Age between 40 - 80')
plt.show()
```



In [24]:
```
# Diabetic having insulin count :
Diabetic_Insulin =df[df.Outcome ==1][['Insulin','Outcome']]
Diabetic_Insulin.dropna(how='any', axis =0, inplace =True)
Diabetic_Insulin.describe()
```

Out[24]:

	Insulin	Outcome
count	268.000000	268.0
mean	164.701493	1.0
std	100.932249	0.0
min	14.000000	1.0
25%	125.000000	1.0
50%	125.000000	1.0
75%	167.250000	1.0
max	846.000000	1.0

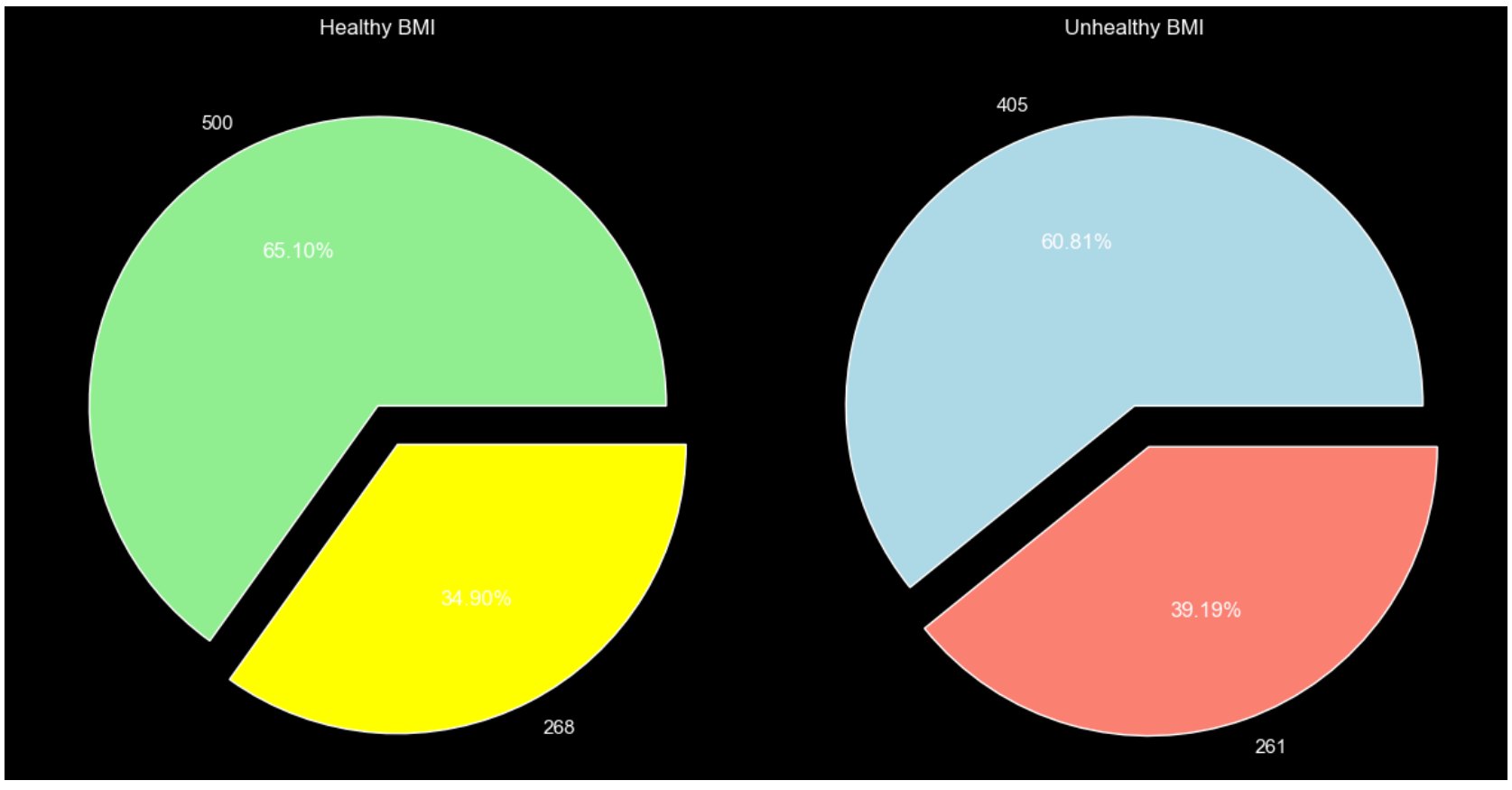
- min and max of different features of a diabetic person

In [25]:
```
diabetic = df[df.Outcome == 1][list(df.iloc[:,:-1])].describe().T[['min','max']]
sns.heatmap(diabetic, annot=True, cmap='Set2', fmt=".2f")
```

Out[25]: <Axes: >

In [29]:
```
# Healthy BMI 18.5 to <25
healthy_BMI = df[(df['BMI']<=25) | (df['BMI']>=18.5)]['Outcome'].value_counts()
unhealthy_BMI = df[(df['BMI']>=25) | (df['BMI']<=18.5)]['Outcome'].value_counts()
print('Healthy_BMI\n',healthy_BMI)
print('=='*30)
print('Unhealthy_BMI\n',unhealthy_BMI)
#================================
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.pie(healthy_BMI, labels=healthy_BMI,colors = ['lightgreen', 'yellow'],autopct='%.2f%%', explode =[0,0.15]
plt.title('Healthy BMI')

plt.subplot(1, 2, 2)
plt.pie(unhealthy_BMI, labels=unhealthy_BMI,colors =['lightblue', 'salmon'],  autopct='%.2f%%', explode =[0,0
plt.title('Unhealthy BMI')

plt.tight_layout()
plt.show()
```

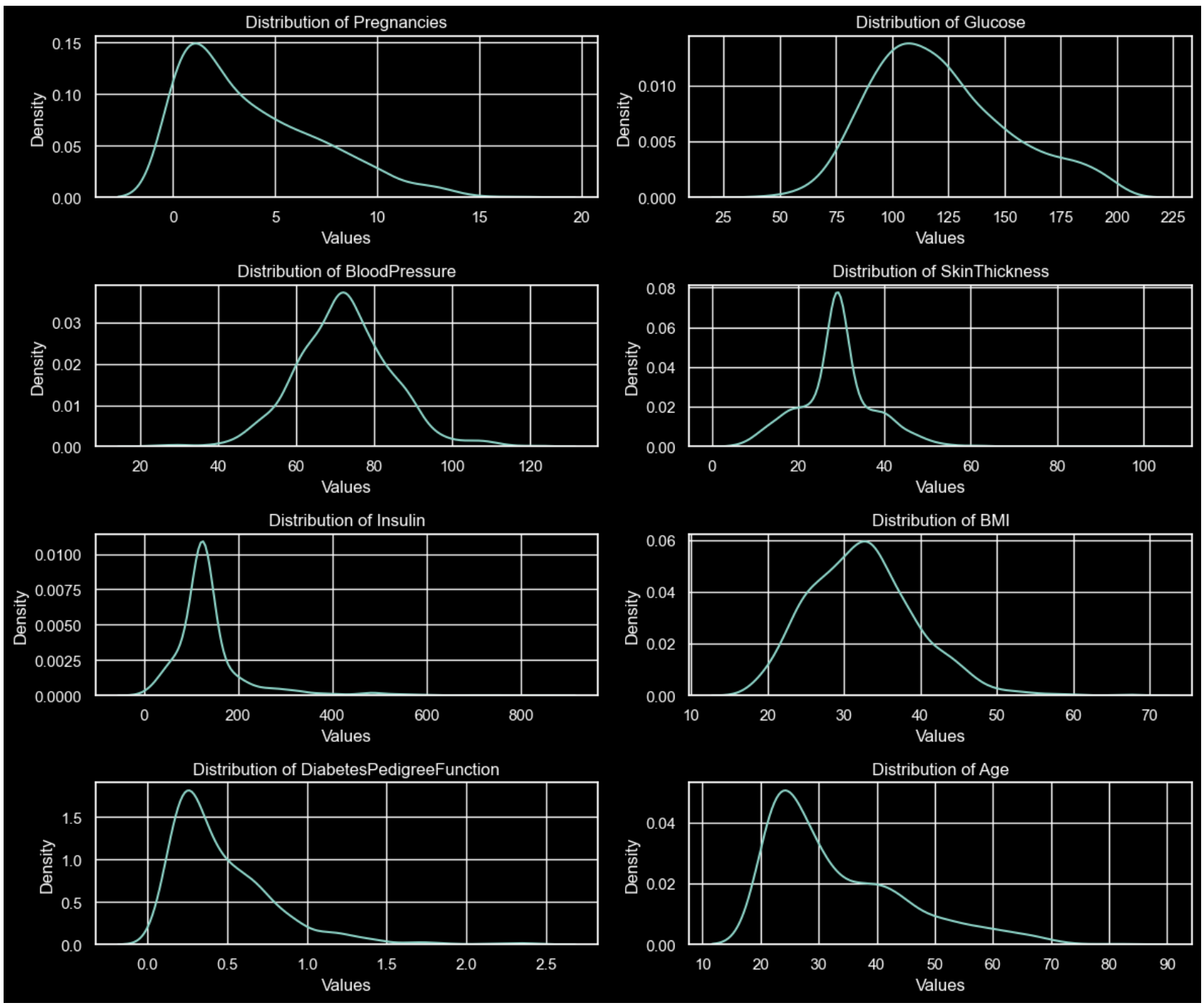
```
Healthy_BMI
 0    500
1    268
Name: Outcome, dtype: int64
============================================================
Unhealthy_BMI
 0    405
1    261
Name: Outcome, dtype: int64
```



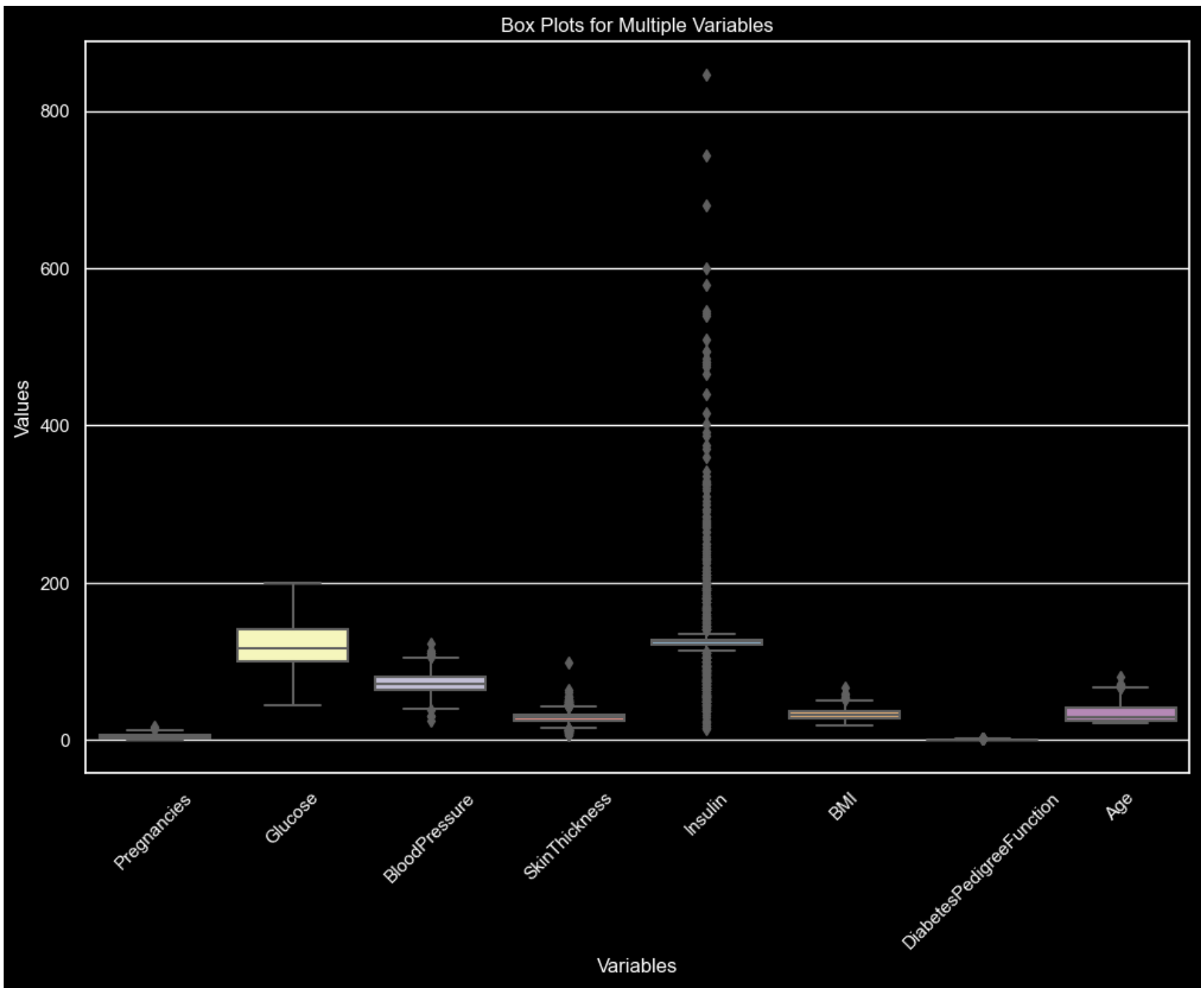
From BMI difficult to predict diabetes.

Distributions

In [30]:
```
num_cols = len(df.columns)

# Determine the number of rows and columns for the subplot grid
num_rows = (num_cols + 1) // 2
num_cols_subplot = 2

# Set the figure size for the plot
plt.figure(figsize=(12, 12))

# Create subplots for each column's distribution
for i, column in enumerate(df.iloc[:,:-1].columns):
    plt.subplot(num_rows, num_cols_subplot, i + 1)
    sns.distplot(df[column], bins = 50, kde = True,hist=False)
    plt.xlabel('Values')
    plt.ylabel('Density')
    plt.title(f'Distribution of {column}')

# Adjust spacing between subplots
plt.tight_layout()

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



In [31]:
```
# Set the figure size for the plot
plt.figure(figsize=(12, 8))

# Create box plots for each column in the DataFrame

sns.boxplot(data=df.iloc[:,:-1])

# Customize the plot with labels and title
plt.xlabel('Variables')
plt.ylabel('Values')
plt.title('Box Plots for Multiple Variables')

# Rotate x-axis labels for better visibility
plt.xticks(rotation=45)

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



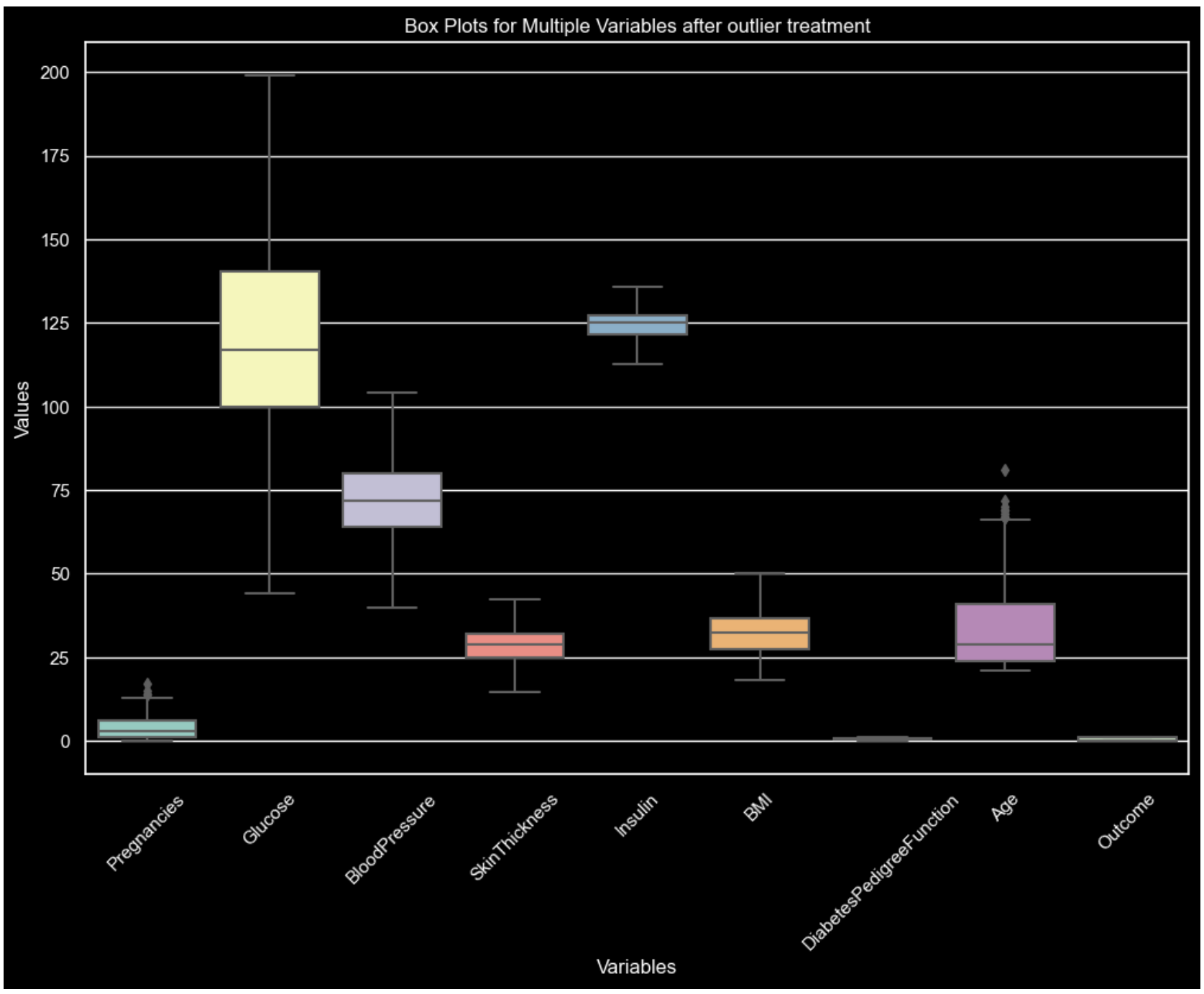
Outlier treatment with IQR method using caping technique

In [32]:
```
outlier_list = list(df.iloc[:,:-1])
outlier_list.remove('Pregnancies')
outlier_list.remove('Age')
outlier_list
```

Out[32]:
```
['Glucose',
 'BloodPressure',
 'SkinThickness',
 'Insulin',
 'BMI',
 'DiabetesPedigreeFunction']
```

In [33]:
```
def outlier(data):
    Q1 = data.quantile(0.25)
    Q3 = data.quantile(0.75)
    IQR = Q3-Q1
    upper_bound = Q3 + 1.5*IQR
    lower_bound = Q1 - 1.5*IQR
    return data.clip(upper_bound, lower_bound)

for col in outlier_list:
    df[col] = outlier(df[col])
```

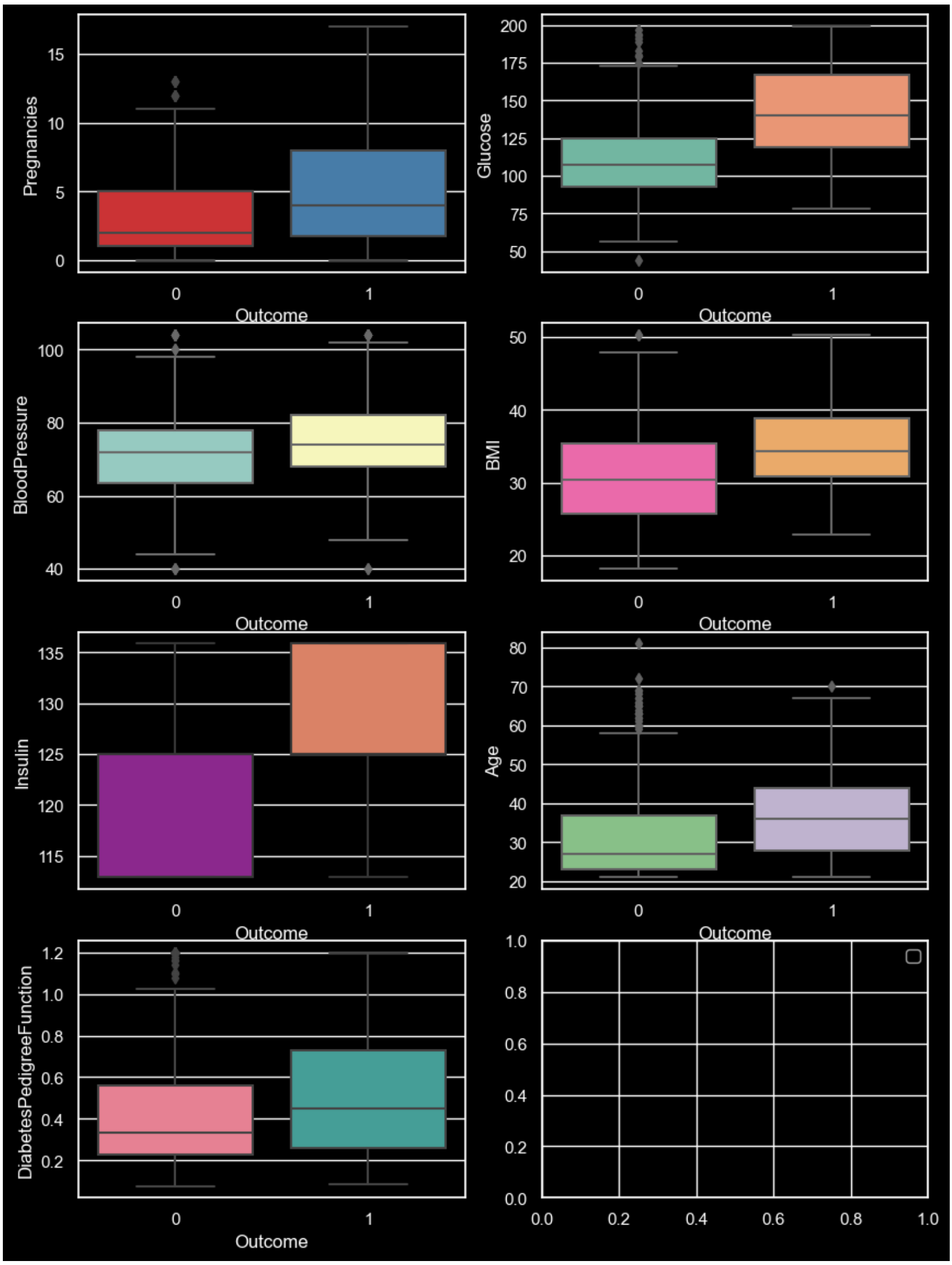
In [32]:
```
plt.figure(figsize=(12, 8))
sns.boxplot(data=df)


plt.xlabel('Variables')
plt.ylabel('Values')
plt.title('Box Plots for Multiple Variables after outlier treatment')

plt.xticks(rotation=45)
plt.show()
```



In [34]:
```
fig, axes = plt.subplots(4, 2, figsize=(10, 14))

sns.boxplot(x="Outcome", y='Pregnancies', data=df, palette='Set1',ax=axes[0, 0],)
sns.boxplot(x="Outcome", y='Glucose', data=df, palette='Set2',ax=axes[0, 1],)
sns.boxplot(x="Outcome", y='BloodPressure', data=df, palette='Set3',ax=axes[1, 0],)
sns.boxplot(x="Outcome", y='BMI', data=df, palette='spring',ax=axes[1, 1],)
sns.boxplot(x="Outcome", y='Insulin', data=df, palette='plasma',ax=axes[2, 0],)
sns.boxplot(x="Outcome", y='Age', data=df, palette='Accent',ax=axes[2, 1],)
sns.boxplot(x="Outcome", y='DiabetesPedigreeFunction', data=df, palette='husl',ax=axes[3, 0],)

plt.legend()
```

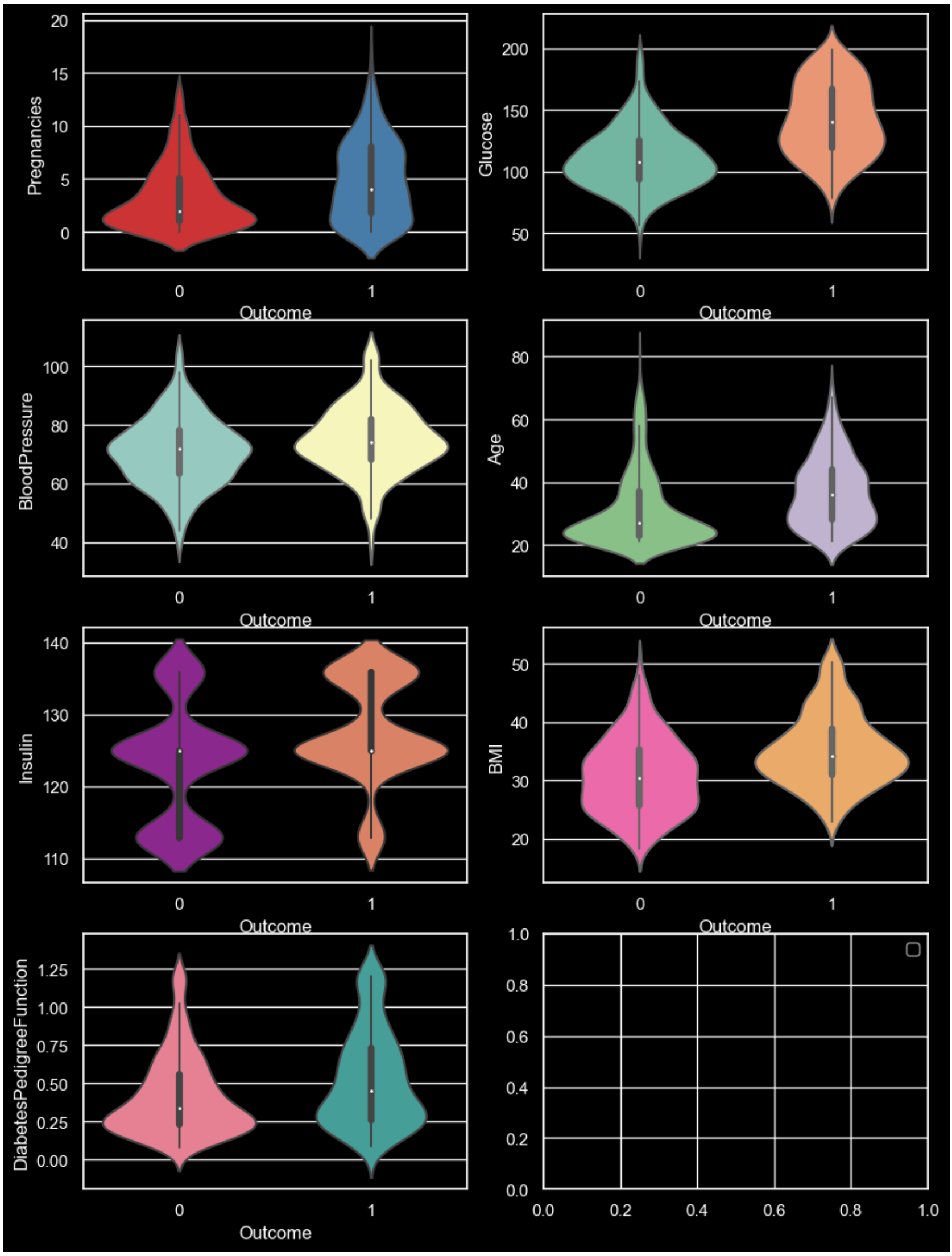
```
No artists with labels found to put in legend.  Note that artists whose label start with an underscore are i
gnored when legend() is called with no argument.
```

Out[34]: <matplotlib.legend.Legend at 0x1e7d46083d0>

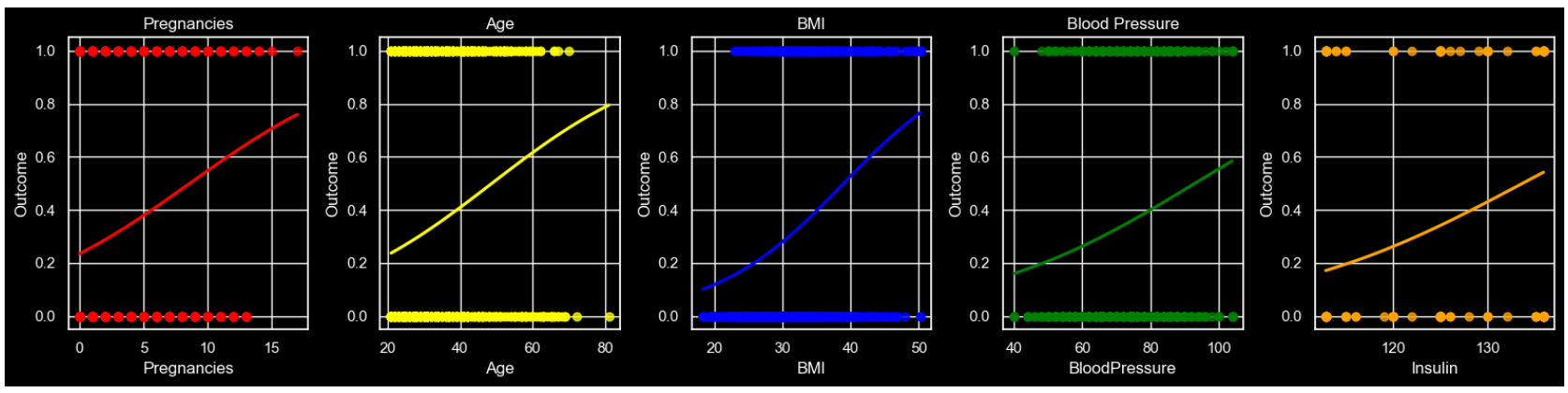


In [35]:
```
fig, axes = plt.subplots(4, 2, figsize=(10, 14))

sns.violinplot(x="Outcome", y='Pregnancies', data=df, palette='Set1',ax=axes[0, 0],)
sns.violinplot(x="Outcome", y='Glucose', data=df, palette='Set2',ax=axes[0, 1],)
sns.violinplot(x="Outcome", y='BloodPressure', data=df, palette='Set3',ax=axes[1, 0],)
sns.violinplot(x="Outcome", y='Age', data=df, palette='Accent',ax=axes[1,1],)
sns.violinplot(x="Outcome", y='BMI', data=df, palette='spring',ax=axes[2, 1],)
sns.violinplot(x="Outcome", y='Insulin', data=df, palette='plasma',ax=axes[2, 0],)
sns.violinplot(x="Outcome", y='BMI', data=df, palette='spring',ax=axes[2, 1],)
sns.violinplot(x="Outcome", y='DiabetesPedigreeFunction', data=df, palette='husl',ax=axes[3, 0],)

plt.legend()
plt.show()
```

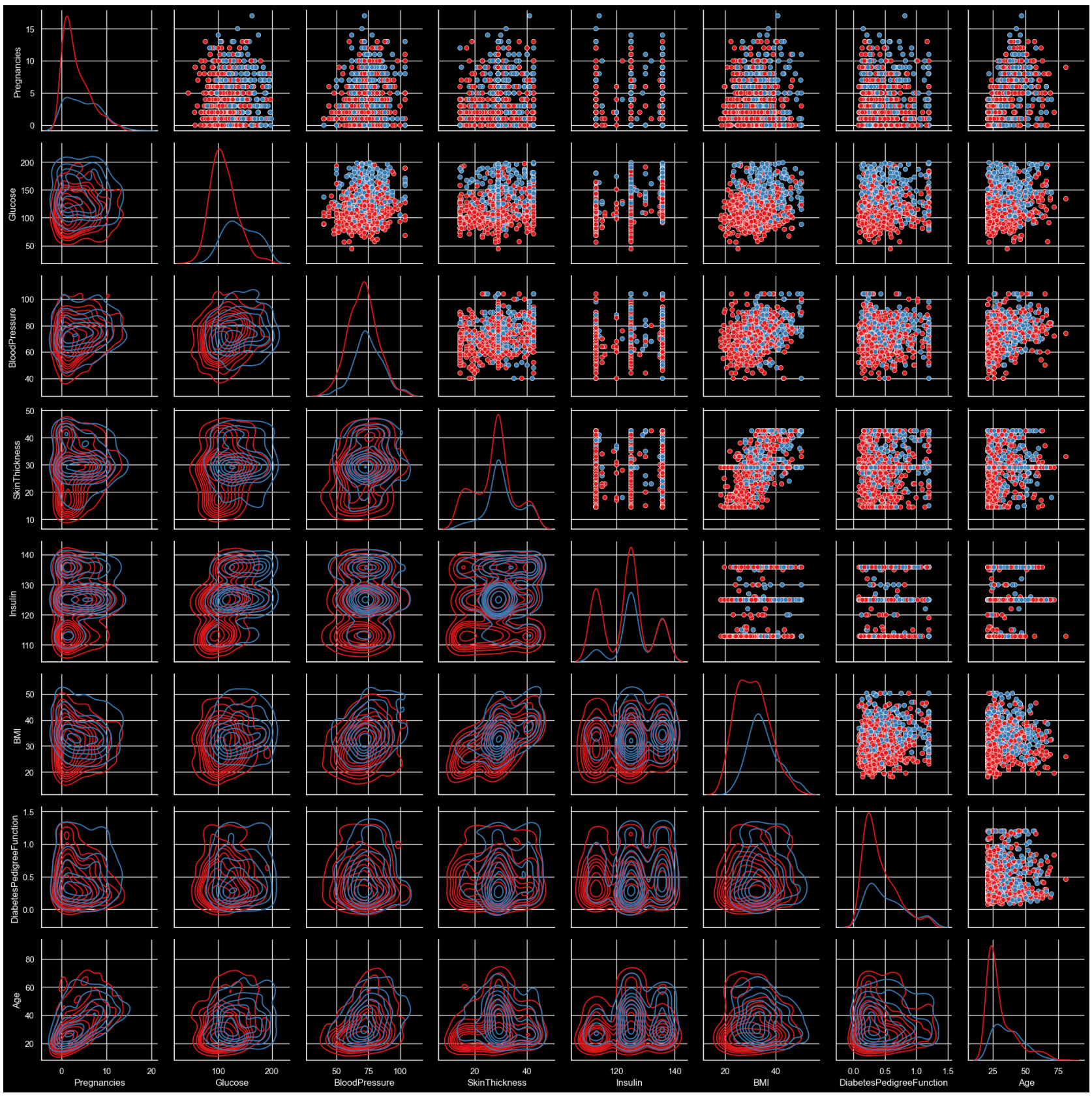
```
No artists with labels found to put in legend.  Note that artists whose label start with an underscore are i
gnored when legend() is called with no argument.
```



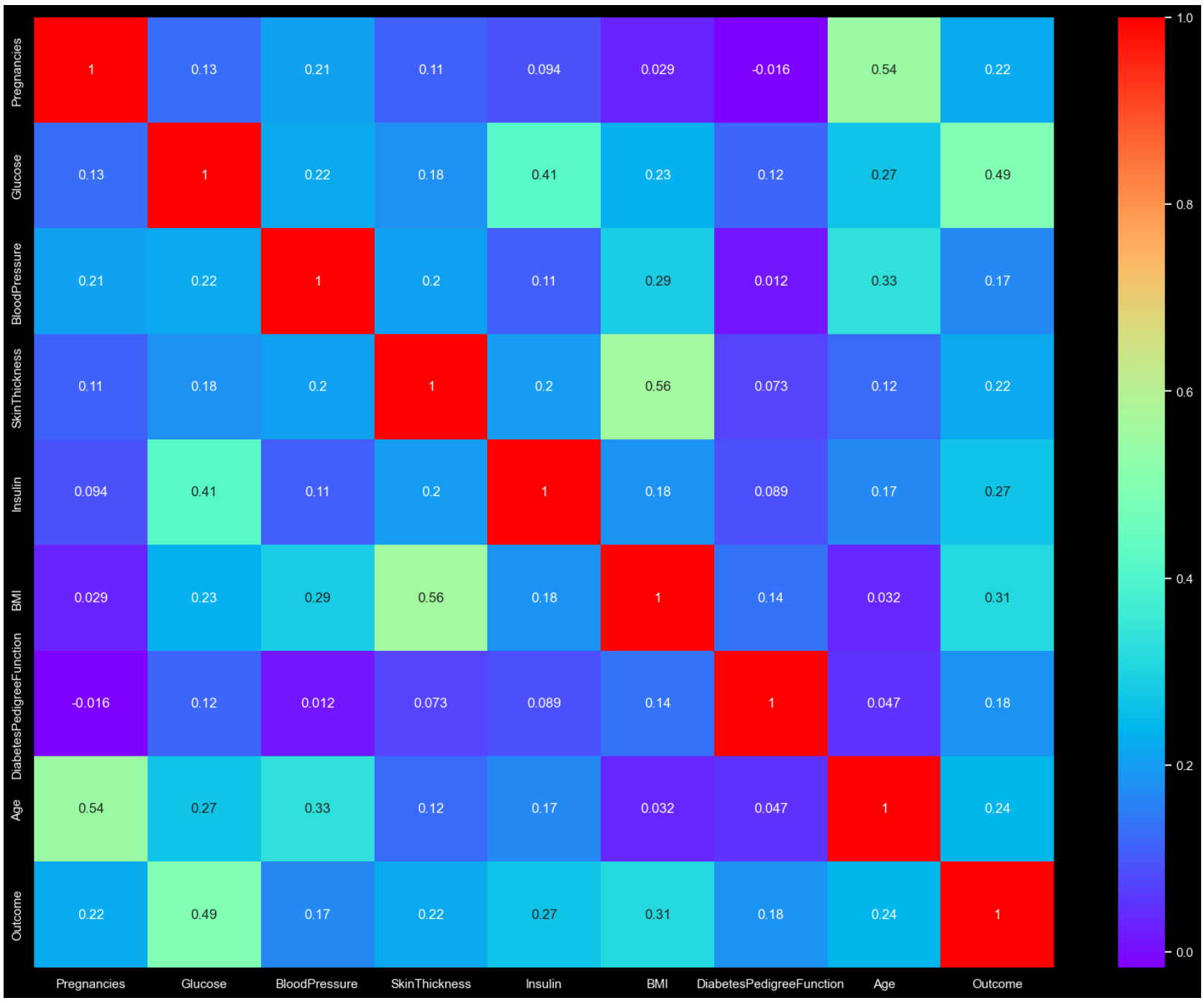
In [36]:
```
fig, axes = plt.subplots(1, 5, figsize=(16, 4))
plt.style.use('dark_background')

sns.regplot(x='Pregnancies', y='Outcome', data=df, logistic=True, ci=None, ax=axes[0],color ='red')
axes[0].set_title('Pregnancies')

sns.regplot(x='Age', y='Outcome', data=df, logistic=True, ci=None, ax=axes[1], color = 'yellow')
axes[1].set_title('Age')

sns.regplot(x='BMI', y='Outcome', data=df, logistic=True, ci=None, ax=axes[2], color = 'blue')
axes[2].set_title('BMI')

sns.regplot(x='BloodPressure', y='Outcome', data=df, logistic=True, ci=None, ax=axes[3], color = 'green')
axes[3].set_title('Blood Pressure')

sns.regplot(x='Insulin', y='Outcome', data=df, logistic=True, ci=None, ax=axes[4], color = 'orange')
axes[3].set_title('Blood Pressure')

plt.tight_layout()

plt.show()
```



In [37]:
```
g = sns.PairGrid(df, hue='Outcome',palette ='Set1' , diag_sharey=False)
g.map_upper(sns.scatterplot)
g.map_lower(sns.kdeplot)
g.map_diag(sns.kdeplot)
plt.show()
```



In [38]:
```
# Finding correlation
plt.style.use('dark_background')
plt.figure(figsize=(20,15))
corr = df.corr()
sns.heatmap(corr, annot=True, cmap='rainbow')
plt.show()
```



In [39]:
```
# handling multicollinearity
df = df.drop(columns = 'Pregnancies', axis=1)
```

Splitting into target and features before model building

In [40]:
```
target = "Outcome"
x = df.drop(columns=target)
y = df[target]
```

In [41]:
```
x.head()
```

Out[41]:

	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	148.0	72.0	35.0	125.000	33.6	0.627	50
1	85.0	66.0	29.0	125.000	26.6	0.351	31
2	183.0	64.0	29.0	125.000	23.3	0.672	32
3	89.0	66.0	23.0	112.875	28.1	0.167	21
4	137.0	40.0	35.0	135.875	43.1	1.200	33

Feature selection

In [42]:
```
#StandardScaler in dataframe
sc = StandardScaler()
sc_x = pd.DataFrame(sc.fit_transform(x) , columns=x.columns)
sc_x.head()
```

Out[42]:

	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	0.866045	-0.030632	0.824667	0.039062	0.181092	0.588927	1.425995
1	-1.205066	-0.543914	0.017945	0.039062	-0.869465	-0.378101	-0.190672
2	2.016662	-0.715008	0.017945	0.039062	-1.364728	0.746595	-0.105584
3	-1.073567	-0.543914	-0.788777	-1.494110	-0.644346	-1.022787	-1.041549
4	0.504422	-2.768136	0.824667	1.414175	1.606849	2.596563	-0.020496

- 🎯 Binary class Logistic Regression

In [43]:
```
x_train,x_test,y_train,y_test = train_test_split(sc_x,y,test_size=0.25,random_state=123,  stratify = y)
logit = LogisticRegression()
logit.fit(x_train,y_train)
y_pred_train_sc = logit.predict(x_train)
y_pred_test_sc = logit.predict(x_test)
```

Evaluation:

In [44]:
```
Train_acc = accuracy_score(y_train, y_pred_train_sc)
Test_acc = accuracy_score(y_test, y_pred_test_sc)
print('=='*35)
print("Trainging Accuracy Score :", Train_acc,)

print("Test Accuracy Score      :", Test_acc)
print()
print('=='*35)
print( ' '*15,'Classification_report_Train',)
print('=='*35)
print( classification_report(y_train, y_pred_train_sc))
print('=='*35)
print( ' '*15,'Classification_report_Test',)
print('=='*35)
print(classification_report(y_test, y_pred_test_sc))
print('=='*35)
```

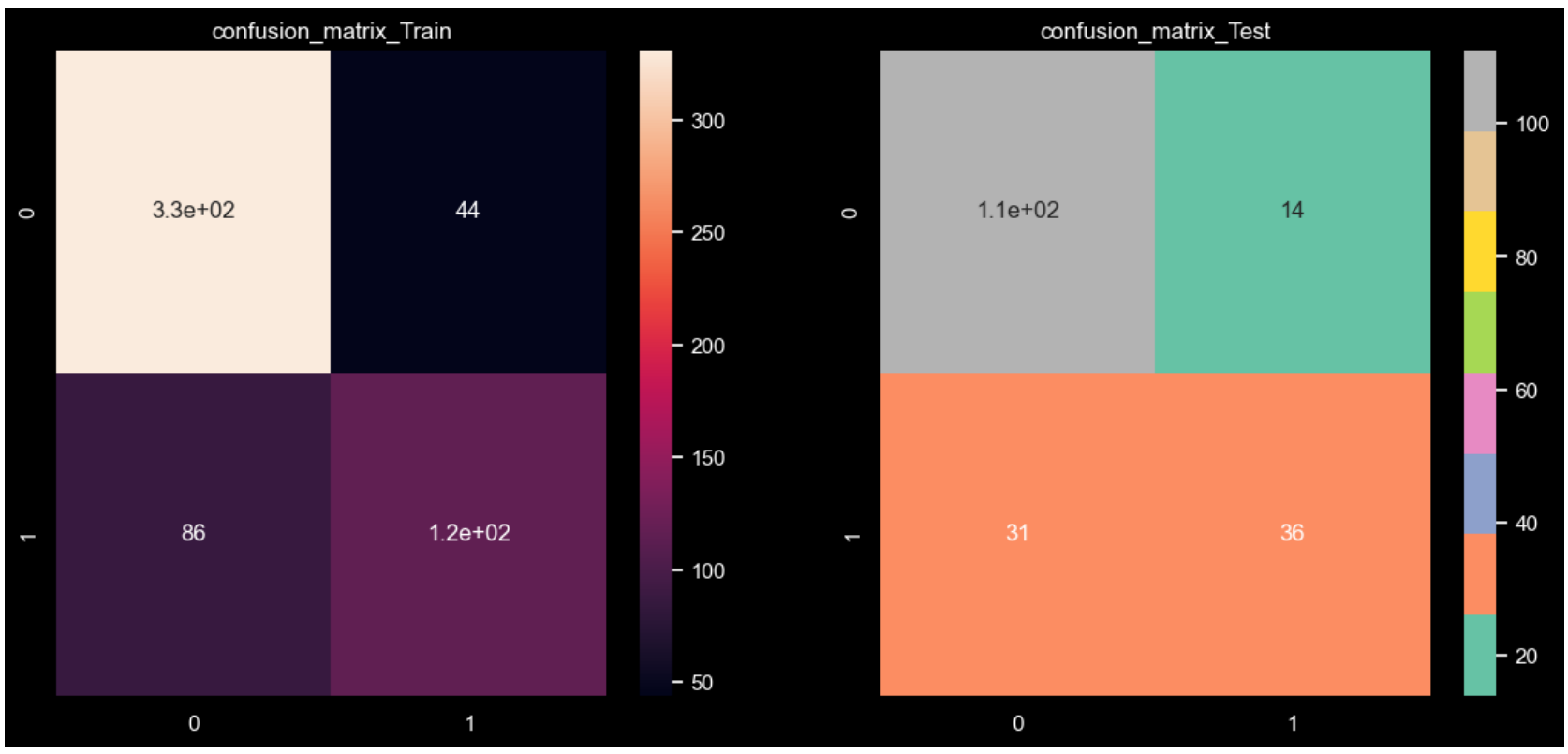
```
======================================================================
Trainging Accuracy Score : 0.7743055555555556
Test Accuracy Score      : 0.765625

======================================================================
               Classification_report_Train
======================================================================
              precision    recall  f1-score   support

           0       0.79      0.88      0.84       375
           1       0.72      0.57      0.64       201

    accuracy                           0.77       576
   macro avg       0.76      0.73      0.74       576
weighted avg       0.77      0.77      0.77       576

======================================================================
               Classification_report_Test
======================================================================
              precision    recall  f1-score   support

           0       0.78      0.89      0.83       125
           1       0.72      0.54      0.62        67

    accuracy                           0.77       192
   macro avg       0.75      0.71      0.72       192
weighted avg       0.76      0.77      0.76       192

======================================================================
```

Confusion matrix

In [45]:
```
cm_train_sc= confusion_matrix(y_train, y_pred_train_sc)
cm_test_sc = confusion_matrix(y_test, y_pred_test_sc)

print('===='*10)
print( 'confusion_matrix_Test :','\n','\n',cm_test_sc)
print('===='*10)
print( 'confusion_matrix_Train :','\n','\n',cm_train_sc)
print('===='*10)
plt.figure(figsize = (14,6))
plt.subplot(1,2,1)
sns.heatmap(cm_train_sc, annot = True, )
plt.title('confusion_matrix_Train')
plt.subplot(1,2,2)
sns.heatmap(cm_test_sc, annot = True, cmap= 'Set2')
plt.title('confusion_matrix_Test')
```

```
========================================
confusion_matrix_Test :

 [[111  14]
 [ 31  36]]
========================================
confusion_matrix_Train :

 [[331  44]
 [ 86 115]]
========================================
```

Out[45]: Text(0.5, 1.0, 'confusion_matrix_Test')



Cross Validation

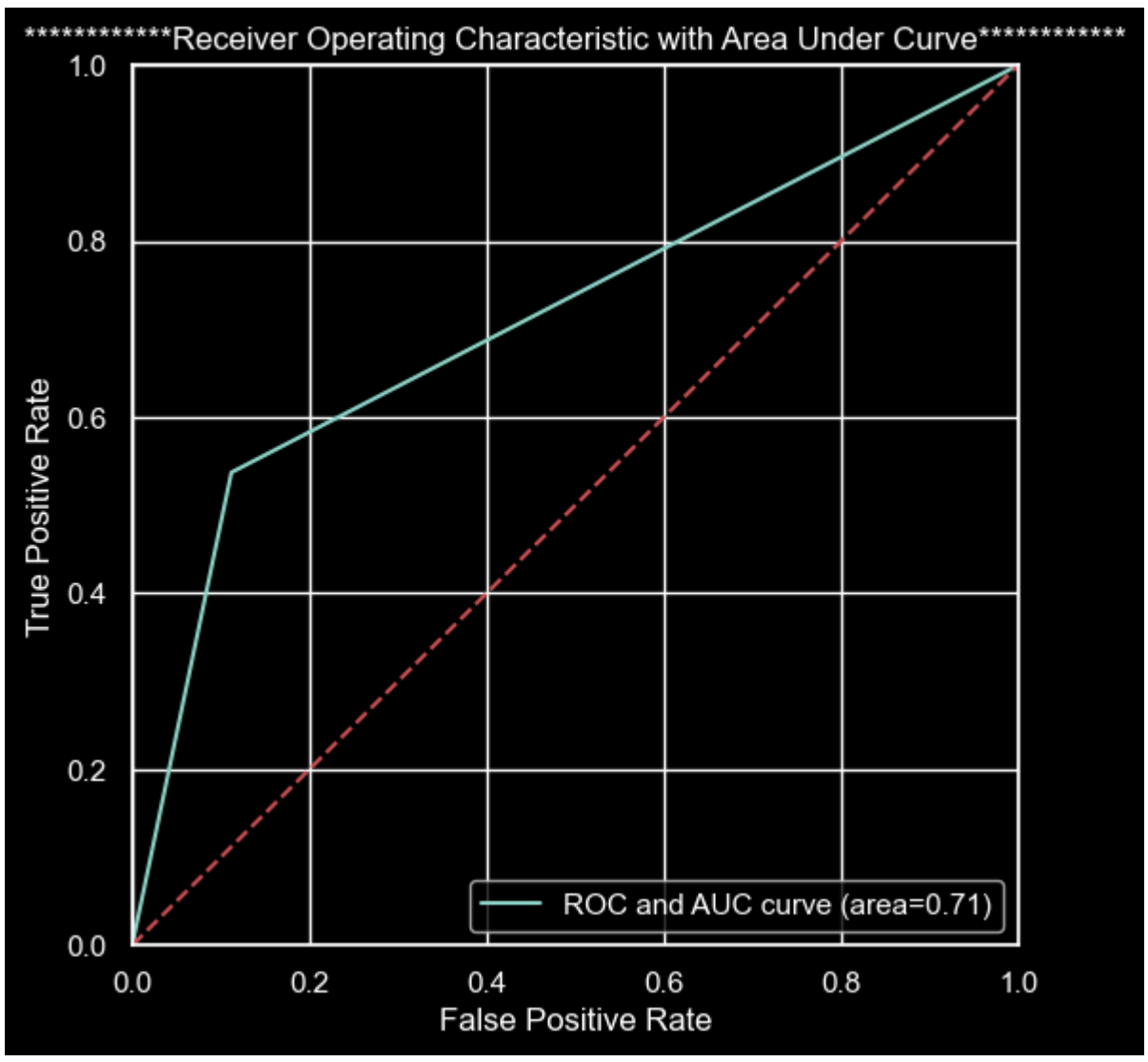
In [46]:
```
training_accuracy =cross_val_score(logit, x_train, y_train, cv =20)
test_accuracy =cross_val_score(logit, x_test, y_test, cv =20)
print(training_accuracy.mean())
print(test_accuracy.mean())
```

```
0.7655172413793103
0.7922222222222223
```

In [47]:
```
logit_roc_auc = roc_auc_score(y_test, y_pred_test_sc)
print('Area under curve :',logit_roc_auc)
fpr, tpr, thresholds = roc_curve(y_test, y_pred_test_sc)
display(fpr[:10])
display(tpr[:10])
display(thresholds[:10])
```

```
Area under curve : 0.7126567164179104

array([0.   , 0.112, 1.   ])

array([0.        , 0.53731343, 1.        ])

array([inf,  1.,  0.])
```

In [48]:
```
plt.figure(figsize=(6,6))
plt.plot(fpr, tpr, label="ROC and AUC curve (area=%0.2f)" % logit_roc_auc)
plt.plot([0,1],[0,1], 'r--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title("************Receiver Operating Characteristic with Area Under Curve************")
plt.legend(loc='lower right')
plt.show()
```



Summary of my Logistic Model

Based on the provided training and testing results, along with the AUC-ROC curve analysis, let's interpret the performance of the logistic regression model:

Training and Testing Accuracy (after cross validation):

- Training Accuracy Score: Approximately 76%
- Test Accuracy Score: Approximately 79%

Interpretation: The model achieves an accuracy of approximately 76% on the training data and 79% on the testing data. This indicates that the model performs reasonably well in predicting the correct class labels for both datasets. There is a scope for improvement.

Overall Interpretation:

- The logistic regression model shows a reasonably good performance, with accuracy scores of around 76% on the training data and 79% on the testing data even after cross validation

- The classification report indicates that the model performs better in identifying negative cases (0) compared to positive cases (1) for both the training and testing datasets. There is imbalancy between Recall, precision and F1 score

- The AUC-ROC curve analysis suggests moderate discrimination power of the model in distinguishing between positive and negative cases.

Type *Markdown* and LaTeX: α^2

==========

🚀 Model Improvement 🚀

- ⚙️ 1. Logistic Regression with Imbalance treatment
- ⚙️ 2. Decision Tree Classifier
- ⚙️ 3. Bagging Classifier
- ⚙️ 3. Random Forest Classifier
- ⚙️ 5. KNearest Neighbour
- ⚙️ 6. Support Vector Machine

In [49]:
```
# Random Search
from sklearn.model_selection import RandomizedSearchCV
```

⚖️ Imbalance treatment

In [50]:
```
from imblearn.over_sampling import RandomOverSampler
over = RandomOverSampler()
x_over, y_over = over.fit_resample(sc_x,y)
```

✂️ Splitting into train and test

In [51]:
```
x_train,x_test,y_train,y_test = train_test_split(x_over,y_over,test_size=0.25,random_state=123,  stratify = y
```

1. - 🎯 Binary class Logistic Regression with imbalance treatment (Oversampling Method)

In [57]:
```
logit.fit(x_train,y_train)
y_pred_train_logit = logit.predict(x_train)
Train_acc = accuracy_score(y_train, y_pred_train_logit)

y_pred_test_logit  = logit.predict(x_test)
Test_acc = accuracy_score(y_test, y_pred_test_logit)
print("---"*20)
print('  '*5,'Logistic Regression Accuracy: (Before tunning)')
print("---"*20)
print(f' \n   => Train_Accuracy :{Train_acc.round(2)*100} %, \n   => Test_Accuracy :{Test_acc.round(2)*100} %
print("---"*20)
```

```
------------------------------------------------------------
           Logistic Regression Accuracy: (Before tunning)
------------------------------------------------------------

   => Train_Accuracy :74.0 %,
   => Test_Accuracy :80.0 %
------------------------------------------------------------
```

Type *Markdown* and LaTeX: α^2

In [ ]:

In [58]:
```
parameter_grid = {'penalty': ['l1','l2','elasticnet',None],
                  'fit_intercept' : [True, False],
                  'multi_class' : ['auto', 'ovr', 'multinomial'],
                  'dual' : [True,False],
                  'class_weight' : ['balanced', None],
                  'solver' : ['lbfgs', 'liblinear', 'newton-cg', 'newton-cholesky', 'sag', 'saga']
                 }
#print(parameter_grid)
model = LogisticRegression()
logit_hyper =RandomizedSearchCV(estimator = model,
                                      param_distributions = parameter_grid,
                                      cv =10,
                                      n_jobs =-1,
                                      verbose = 2,
                                      random_state =122 )
logit_hyper.fit(x_train, y_train)
print('--'*40)
print('Best Parameter', logit_hyper.best_params_)
print('Best score', logit_hyper.best_score_)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
--------------------------------------------------------------------------------
Best Parameter {'solver': 'saga', 'penalty': None, 'multi_class': 'multinomial', 'fit_intercept': True, 'dua
l': False, 'class_weight': None}
Best score 0.7333333333333333
```

In [59]:
```
logit_new = LogisticRegression(solver = 'saga',
                               penalty =  None,
                               multi_class = 'multinomial',
                               fit_intercept = True,
                               dual = False,
                               class_weight = None,
                              )

logit_new.fit(x_train, y_train)

#Predicted y value for train and test
y_pred_train_logit = logit_new.predict(x_train)
y_pred_test_logit  = logit_new.predict(x_test)

Train_acc = accuracy_score(y_train, y_pred_train_logit)
Test_acc = accuracy_score(y_test, y_pred_test_logit)
print("---"*20)
print('  '*5,'Logistic Regresion Accuracy: (After tunning)')
print("---"*20)
print(f' \n   => Train_Accuracy :{Train_acc.round(2)*100} %, \n   => Test_Accuracy :{Test_acc.round(2)*100} %
print("---"*20)
```

```
------------------------------------------------------------
           Logistic Regresion Accuracy: (After tunning)
------------------------------------------------------------

   => Train_Accuracy :74.0 %,
   => Test_Accuracy :80.0 %
------------------------------------------------------------
```

2 - 🎯 Decision Tree Classification

In [60]:
```
dtc = DecisionTreeClassifier()
dtc.fit(x_train, y_train)

y_pred_train_dtc = dtc.predict(x_train)
Train_acc = accuracy_score(y_train, y_pred_train_dtc)

y_pred_test_dtc  = dtc.predict(x_test)
Test_acc = accuracy_score(y_test, y_pred_test_dtc)

print("---"*20)
print('  '*5,'Decision Tree Accuracy: (Before tunning)')
print("---"*20)
print(f' \n   => Train_Accuracy :{Train_acc.round(2)*100} %, \n   => Test_Accuracy :{Test_acc.round(2)*100} %
print("---"*20)
```

```
------------------------------------------------------------
           Decision Tree Accuracy: (Before tunning)
------------------------------------------------------------

   => Train_Accuracy :100.0 %,
   => Test_Accuracy :78.0 %
------------------------------------------------------------
```

Hyper-parameter tunning

In [98]:
```
parameter_grid = {
                  'criterion' : ["gini", "entropy", "log_loss"],
                  'max_depth': [2,3,4,5,6,7],
                  'min_samples_split' : [2,4,6,8,10],
                  'min_samples_leaf' : [1,2,3,4,5,6],
                  'max_features' : ["auto", "sqrt", "log2"],
                  #'max_leaf_nodes': [2,3,4],
                  'class_weight' : ['balanced', None]
                 }

model = DecisionTreeClassifier()
dct_hyper =RandomizedSearchCV(estimator = model,
                                      param_distributions = parameter_grid,
                                      cv =20,
                                      n_jobs =-1,
                                      #verbose = 2,
                                      random_state =122 )
dct_hyper.fit(x_train, y_train)
print('Best Parameter', dct_hyper.best_params_)
print('Best score', dct_hyper.best_score_)
```

```
Best Parameter {'min_samples_split': 6, 'min_samples_leaf': 5, 'max_features': 'log2', 'max_depth': 4, 'crit
erion': 'log_loss', 'class_weight': 'balanced'}
Best score 0.7413940256045519
```

In [100]:
```
dtc = DecisionTreeClassifier(min_samples_split= 6,
                             min_samples_leaf = 5,
                             max_depth =6,
                             criterion='log_loss',
                             max_features='log2',
                             class_weight= None,
                             random_state =123
                            )
dtc.fit(x_train, y_train)

y_pred_train_dtc = dtc.predict(x_train)
Train_acc = accuracy_score(y_train, y_pred_train_dtc)

y_pred_test_dtc  = dtc.predict(x_test)
Test_acc = accuracy_score(y_test, y_pred_test_dtc)

print("---"*20)
print('  '*5,'Decision Tree Accuracy: (after tunning)')
print("---"*20)
print(f' \n   => Train_Accuracy :{Train_acc.round(2)*100} %, \n   => Test_Accuracy :{Test_acc.round(2)*100} %
print("---"*20)
```

```
------------------------------------------------------------
           Decision Tree Accuracy: (after tunning)
------------------------------------------------------------

   => Train_Accuracy :79.0 %,
   => Test_Accuracy :78.0 %
------------------------------------------------------------
```

In [ ]:
```
```

3. 🎯 Bagging

In [211]:
```
bag = BaggingClassifier()

bag.fit(x_train, y_train)

y_pred_train_bag = bag.predict(x_train)
Train_acc = accuracy_score(y_train, y_pred_train_bag)

y_pred_test_bag  = bag.predict(x_test)
Test_acc = accuracy_score(y_test, y_pred_test_bag)
print("---"*20)
print('  '*5,'Bagging Accuracy:')
print("---"*20)
print(f' \n   => Train_Accuracy :{Train_acc.round(2)*100} %, \n   => Test_Accuracy :{Test_acc.round(2)*100} %
print("---"*20)
```

```
------------------------------------------------------------
           Bagging Accuracy:
------------------------------------------------------------

   => Train_Accuracy :100.0 %,
   => Test_Accuracy :86.0 %
------------------------------------------------------------
```

In [103]:
```
parameter_grid = {'estimator' : [DecisionTreeClassifier(), RandomForestClassifier(), KNeighborsClassifier(),
                  'n_estimators' : [10, 20, 50, 100, 150, 200],
                  'max_samples' : [1.0, 2.0,4.0, 6.0,8.0],
                  'max_features' : [1.0,2,3,4],
                  'bootstrap' : [True,False],
                  'bootstrap_features' : [True, False],
                  'oob_score' : [True, False],
                  }

model = BaggingClassifier()
bag_hyper =RandomizedSearchCV(estimator = model,
                                      param_distributions = parameter_grid,
                                      cv =20,
                                      n_jobs =-1,
                                      verbose = 2,
                                      random_state =122 )
bag_hyper.fit(x_train, y_train)
print('--'*40)
print('Best Parameter', bag_hyper.best_params_)
print('Best score', bag_hyper.best_score_)
```

```
Fitting 20 folds for each of 10 candidates, totalling 200 fits
--------------------------------------------------------------------------------
Best Parameter {'oob_score': False, 'n_estimators': 100, 'max_samples': 1.0, 'max_features': 3, 'estimator':
None, 'bootstrap_features': False, 'bootstrap': True}
Best score 0.8213371266002845
```

In [107]:
```
bag = BaggingClassifier( oob_score = False,
                         n_estimators = 100,
                         max_samples = 1.0,
                         max_features = 4,
                         estimator = KNeighborsClassifier(),
                         bootstrap_features = False,
                         bootstrap = True,
                         random_state =123)

bag.fit(x_train, y_train)

y_pred_train_bag = bag.predict(x_train)
Train_acc = accuracy_score(y_train, y_pred_train_bag)

y_pred_test_bag  = bag.predict(x_test)
Test_acc = accuracy_score(y_test, y_pred_test_bag)
print("---"*20)
print('  '*5,'Bagging Accuracy (after tunning):')
print("---"*20)
print(f' \n   => Train_Accuracy :{Train_acc.round(2)*100} %, \n   => Test_Accuracy :{Test_acc.round(2)*100} %
print("---"*20)
```

```
------------------------------------------------------------
           Bagging Accuracy (after tunning):
------------------------------------------------------------

   => Train_Accuracy :90.0 %,
   => Test_Accuracy :83.0 %
------------------------------------------------------------
```

4 - 🎯 Random Forest Classification

In [108]:
```
rfc = RandomForestClassifier()
rfc.fit(x_train, y_train)

y_pred_train_rfc = rfc.predict(x_train)
Train_acc = accuracy_score(y_train, y_pred_train_rfc)

y_pred_test_rfc  = rfc.predict(x_test)
Test_acc = accuracy_score(y_test, y_pred_test_rfc)
print(f'Train_Accuracy :{Train_acc.round(2)*100} %, \nTest_Accuracy :{Test_acc.round(2)*100} %')
```

```
Train_Accuracy :100.0 %,
Test_Accuracy :88.0 %
```

Hyperparameter tunning

In [109]:
```
param_grid = {'n_estimators': [30, 60,100,150,200,300,500],
              'criterion': ['gini','entropy','log_loss'],
               'max_depth' :[None, 2,4,6],
               'min_samples_split' :[2,4,6,8,10],
               'min_samples_leaf' :[1,2,4],
               'max_features':['sqrt','log2',None],
               'bootstrap' :[True,False],
               'oob_score' :[True,False],
              #'warm_start':[True,False]
               'class_weight' : [None, "balanced", "balanced_subsample"]
             }

model = RandomForestClassifier()
rfc_hyper =RandomizedSearchCV(estimator = model,
                              param_distributions = param_grid,
                              cv =10,
                              n_jobs =-1,
                              verbose = 2,
                              random_state =123
                             )
rfc_hyper.fit(x_train, y_train)
print()
print('Best Parameter', rfc_hyper.best_params_)
print('Best score', rfc_hyper.best_score_)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits

Best Parameter {'oob_score': False, 'n_estimators': 100, 'min_samples_split': 8, 'min_samples_leaf': 1, 'max
_features': 'log2', 'max_depth': None, 'criterion': 'gini', 'class_weight': 'balanced_subsample', 'bootstra
p': True}
Best score 0.8146666666666667
```

In [110]:
```
rfc  = RandomForestClassifier(n_estimators =150,
                              min_samples_split = 4,
                              min_samples_leaf = 2,
                              max_features = 'sqrt',
                              max_depth = 7,
                              criterion = 'log_loss',
                              class_weight = 'balanced_subsample',
                              bootstrap = True,
                              oob_score = True,
                              random_state =123
                             )
rfc.fit(x_train, y_train)
y_pred_train_rfc = rfc.predict(x_train)
Train_acc = accuracy_score(y_train, y_pred_train_rfc)

y_pred_test_rfc  = rfc.predict(x_test)
Test_acc = accuracy_score(y_test, y_pred_test_rfc)
print(f'Train_Accuracy :{Train_acc.round(2)*100} %, \nTest_Accuracy :{Test_acc.round(2)*100} %')
```

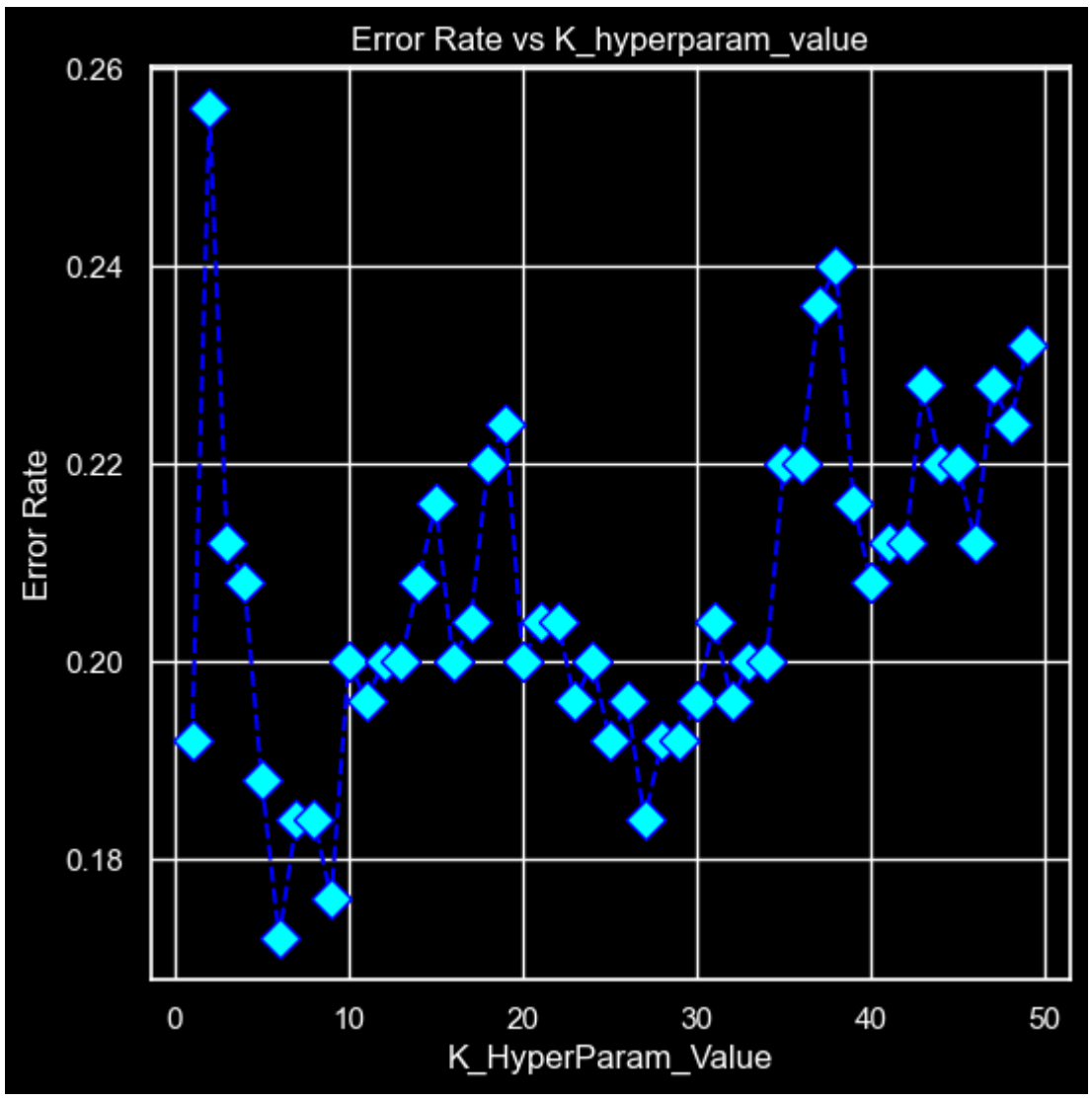
```
Train_Accuracy :90.0 %,
Test_Accuracy :85.0 %
```

5 - 🎯 KNearest Neighbour Classification (KNN Classification)

Hyperparameter tunning

In [111]:
```
error_rate  = []

for i in range(1,50):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(x_train, y_train)
    pred_i = knn.predict(x_test)
    error_rate.append(np.mean(pred_i != y_test))
```

In [112]:
```
plt.figure(figsize=(6,6))
plt.plot(range(1,50), error_rate, color='blue', linestyle='dashed', marker='D',
         markerfacecolor='cyan', markersize=10)
plt.title("Error Rate vs K_hyperparam_value")
plt.xlabel("K_HyperParam_Value")
plt.ylabel("Error Rate")
plt.show()
```



In [115]:
```
knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(x_train, y_train)
y_pred_train_knn = knn.predict(x_train)
Train_acc = accuracy_score(y_train, y_pred_train_knn)

y_pred_test_knn  = knn.predict(x_test)
Test_acc = accuracy_score(y_test, y_pred_test_knn)
print(f'Train_Accuracy :{Train_acc.round(2)*100} %, \nTest_Accuracy :{Test_acc.round(2)*100} %')
```

```
Train_Accuracy :81.0 %,
Test_Accuracy :82.0 %
```

In [121]:
```
param_grid = {'algorithm' : ['auto', 'ball_tree', 'kd_tree', 'brute'],
              'p' : [1,2], # 1: manhattan_distance, 2 : euclidean_distance
              'weights' :['uniform', 'distance'],
              'n_neighbors': [3,5,7,9,]
             }

model = KNeighborsClassifier()
knn_hyper =RandomizedSearchCV(estimator = model,
                              param_distributions = param_grid,
                              cv =20,
                              n_jobs =-1,
                              random_state = 123
                             )
knn_hyper.fit(x_train, y_train)
print('Best Parameter', knn_hyper.best_params_)
print('Best score', knn_hyper.best_score_)
```

```
Best Parameter {'weights': 'distance', 'p': 2, 'n_neighbors': 9, 'algorithm': 'ball_tree'}
Best score 0.7999644381223329
```

In [129]:
```
knn  = KNeighborsClassifier(n_neighbors= 9,  #5,
                            weights ='uniform',
                            p= 2,

                            )
knn.fit(x_train, y_train)
y_pred_train_knn = knn.predict(x_train)
Train_acc = accuracy_score(y_train, y_pred_train_knn)

y_pred_test_knn  = knn.predict(x_test)
Test_acc = accuracy_score(y_test, y_pred_test_knn)
print(f'Train_Accuracy :{Train_acc.round(2)*100} %, \nTest_Accuracy :{Test_acc.round(2)*100} %')
```

```
Train_Accuracy :82.0 %,
Test_Accuracy :82.0 %
```

6 - 🎯 Support Vector Machine (SVM

In [133]:
```
svc = SVC()
svc.fit(x_train, y_train)
y_pred_train_svc = svc.predict(x_train)
Train_acc = accuracy_score(y_train, y_pred_train_svc)

y_pred_test_svc  = svc.predict(x_test)
Test_acc = accuracy_score(y_test, y_pred_test_svc)
print(f'Train_Accuracy :{Train_acc.round(2)*100} %, \nTest_Accuracy :{Test_acc.round(2)*100} %')
```

```
Train_Accuracy :83.0 %,
Test_Accuracy :84.0 %
```

In [139]:
```
SVC??
```

In [141]:
```
svc_model = SVC()
param_grid = {'C': [0.1, 1, 10],
              'kernel': ['linear', 'rbf', 'sigmoid', 'poly'],
             'decision_function_shape' : ['ovo', 'ovr']}
model = svc_model
svc_hyper =RandomizedSearchCV(estimator = model,
                              param_distributions = param_grid,
                              cv =20,
                              n_jobs =-1,
                              random_state = 123
                             )
svc_hyper.fit(x_train, y_train)
print('Best Parameter', svc_hyper.best_params_)
print('Best score', svc_hyper.best_score_)
```

```
Best Parameter {'kernel': 'rbf', 'decision_function_shape': 'ovr', 'C': 10}
Best score 0.785099573257468
```

In [142]:
```
svc = SVC( kernel ='rbf', C =1, decision_function_shape = 'ovr')

svc.fit(x_train, y_train)
y_pred_train_svc = svc.predict(x_train)
Train_acc = accuracy_score(y_train, y_pred_train_svc)

y_pred_test_svc  = svc.predict(x_test)
Test_acc = accuracy_score(y_test, y_pred_test_svc)
print(f'Train_Accuracy :{Train_acc.round(2)*100} %, \nTest_Accuracy :{Test_acc.round(2)*100} %')
```

```
Train_Accuracy :83.0 %,
Test_Accuracy :84.0 %
```

Summary of all the models

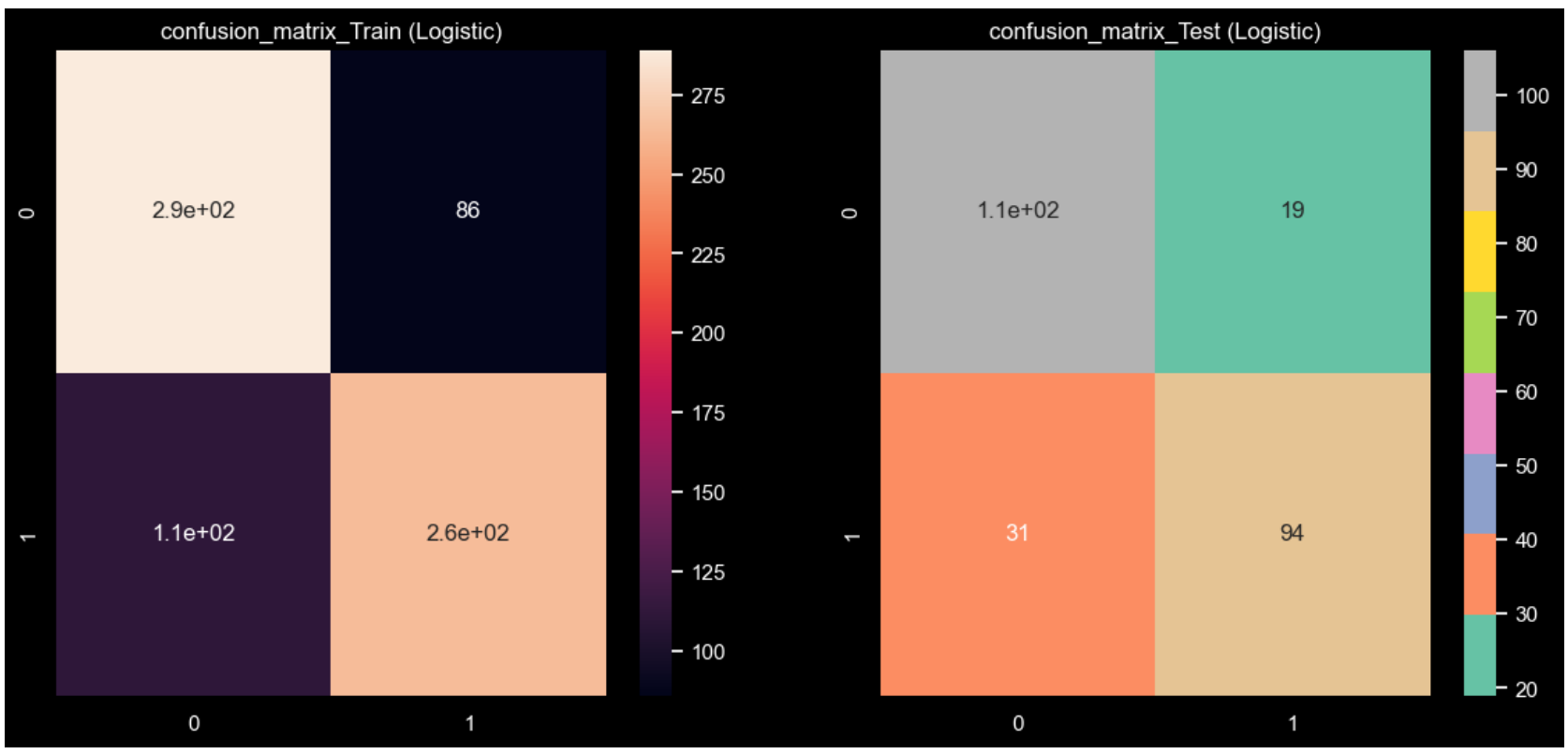
In [144]:
```
models = [('Logistic',logit,y_pred_train_logit,y_pred_test_logit),
          ('DecisionTree',dtc,y_pred_train_dtc,y_pred_test_dtc),
          ('Bagging',bag,y_pred_train_bag,y_pred_test_bag),
          ('RandomForest',rfc,y_pred_train_rfc,y_pred_test_rfc),
          ('KNearestNeighbour',knn,y_pred_train_knn,y_pred_test_knn),
          ('SupportVector',svc,y_pred_train_svc,y_pred_test_svc)
         ]

for model_name,fit_model, y_pred_train, y_pred_test in models:
    Train_acc = accuracy_score(y_train, y_pred_train)
    Test_acc = accuracy_score(y_test, y_pred_test)
    print( '**'*30)
    print( ' '*25, model_name)
    print( '**'*30)
    print('Accuracy  :\n')
    print(f"Train Score :", Train_acc.round(2))

    print(f"Test  Score :", Test_acc.round(2))
    varience = abs(Train_acc.round(2)*100 - Test_acc.round(2)*100)
    print(f'Varience : {varience}')
    print()

    print( 'Classification_report:\n')
    print( 'Train:')
    print( '--'*30)
    print( classification_report(y_train, y_pred_train))

    print( 'Test:')
    print( '--'*30)
    print(classification_report(y_test, y_pred_test))
    print( '--'*30)

    roc_auc = roc_auc_score(y_test, y_pred_test)
    print('Area under ROC-AUC curve =',roc_auc.round(2))
    print( '--'*30)
    print( 'Croos validation :')
    print( )
    training_accuracy =cross_val_score(fit_model, x_train, y_train, cv =20)
    test_accuracy =cross_val_score(fit_model, x_test, y_test, cv =20)
    print(training_accuracy.mean())
    print(test_accuracy.mean())
    print( '--'*30)


    if varience >= 10:
        print()
        print(f'ALERT : {model_name} model is overfitting')
        print()
    cm_train= confusion_matrix(y_train, y_pred_train)
    cm_test = confusion_matrix(y_test, y_pred_test)

    print('===='*10)
    print( 'confusion_matrix_Test :','\n','\n',cm_test)
    print('===='*10)
    print( 'confusion_matrix_Train :','\n','\n',cm_train)
    print('===='*10)
    plt.figure(figsize = (14,6))
    plt.subplot(1,2,1)
    sns.heatmap(cm_train, annot = True, )
    plt.title(f'confusion_matrix_Train ({model_name})')
    plt.subplot(1,2,2)
    sns.heatmap(cm_test, annot = True, cmap= 'Set2')
    plt.title(f'confusion_matrix_Test ({model_name})')

    print()
    #print(f'confusion_matrix ({model_name})')
```

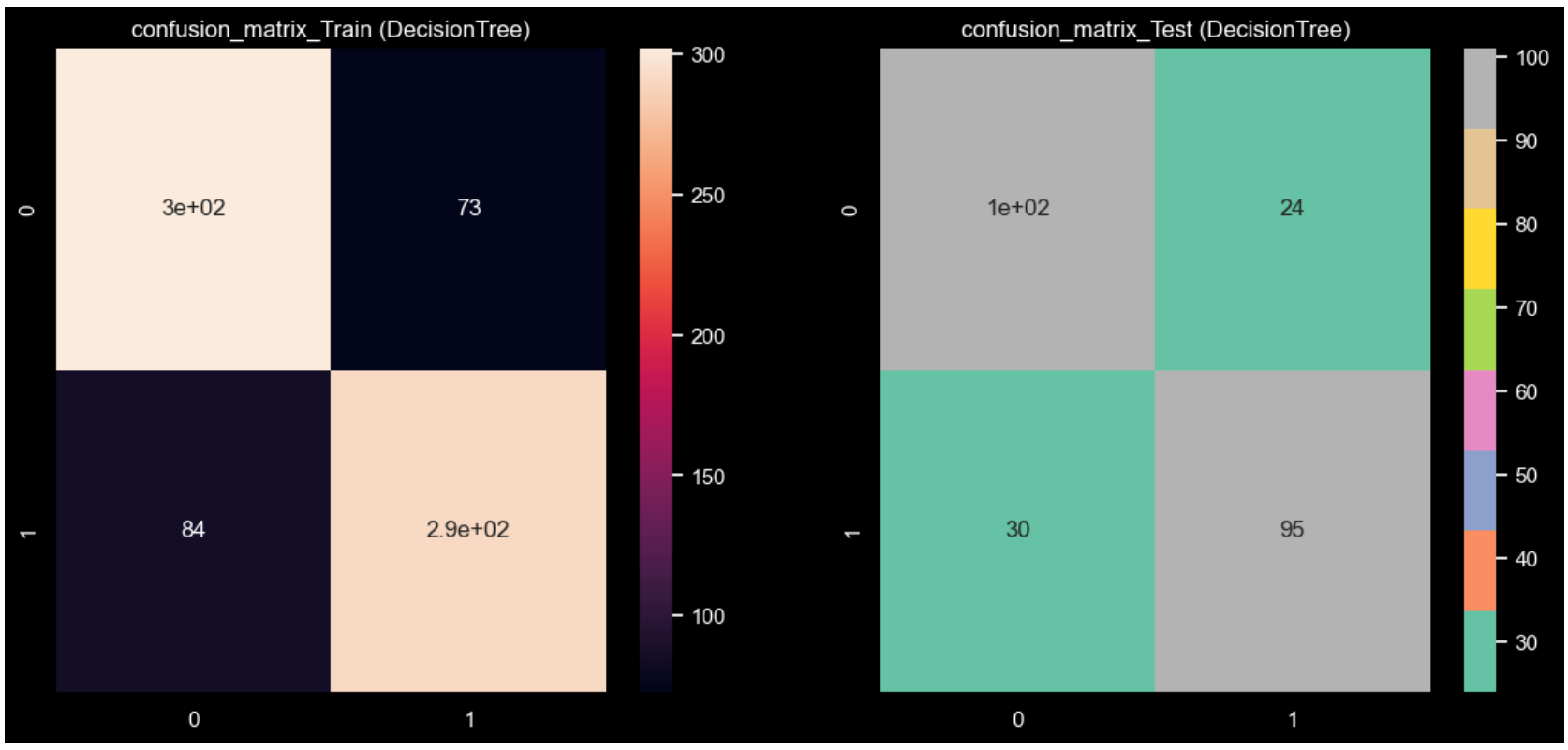
```
************************************************************
                          Logistic
************************************************************
Accuracy  :

Train Score : 0.74
Test  Score : 0.8
Varience : 6.0

Classification_report:

Train:
------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.72      0.77      0.75       375
           1       0.75      0.70      0.73       375

    accuracy                           0.74       750
   macro avg       0.74      0.74      0.74       750
weighted avg       0.74      0.74      0.74       750

Test:
------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.77      0.85      0.81       125
           1       0.83      0.75      0.79       125

    accuracy                           0.80       250
   macro avg       0.80      0.80      0.80       250
weighted avg       0.80      0.80      0.80       250

------------------------------------------------------------
Area under ROC-AUC curve = 0.8
------------------------------------------------------------
Croos validation :

0.732076813655761
0.8035256410256411
------------------------------------------------------------
========================================
confusion_matrix_Test :

 [[106  19]
 [ 31  94]]
========================================
confusion_matrix_Train :

 [[289  86]
 [111 264]]
========================================

************************************************************
                        DecisionTree
************************************************************
Accuracy  :

Train Score : 0.79
Test  Score : 0.78
Varience : 1.0

Classification_report:

Train:
------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.78      0.81      0.79       375
           1       0.80      0.78      0.79       375

    accuracy                           0.79       750
   macro avg       0.79      0.79      0.79       750
weighted avg       0.79      0.79      0.79       750

Test:
------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.77      0.81      0.79       125
           1       0.80      0.76      0.78       125

    accuracy                           0.78       250
   macro avg       0.78      0.78      0.78       250
weighted avg       0.78      0.78      0.78       250

------------------------------------------------------------
Area under ROC-AUC curve = 0.78
------------------------------------------------------------
Croos validation :
```

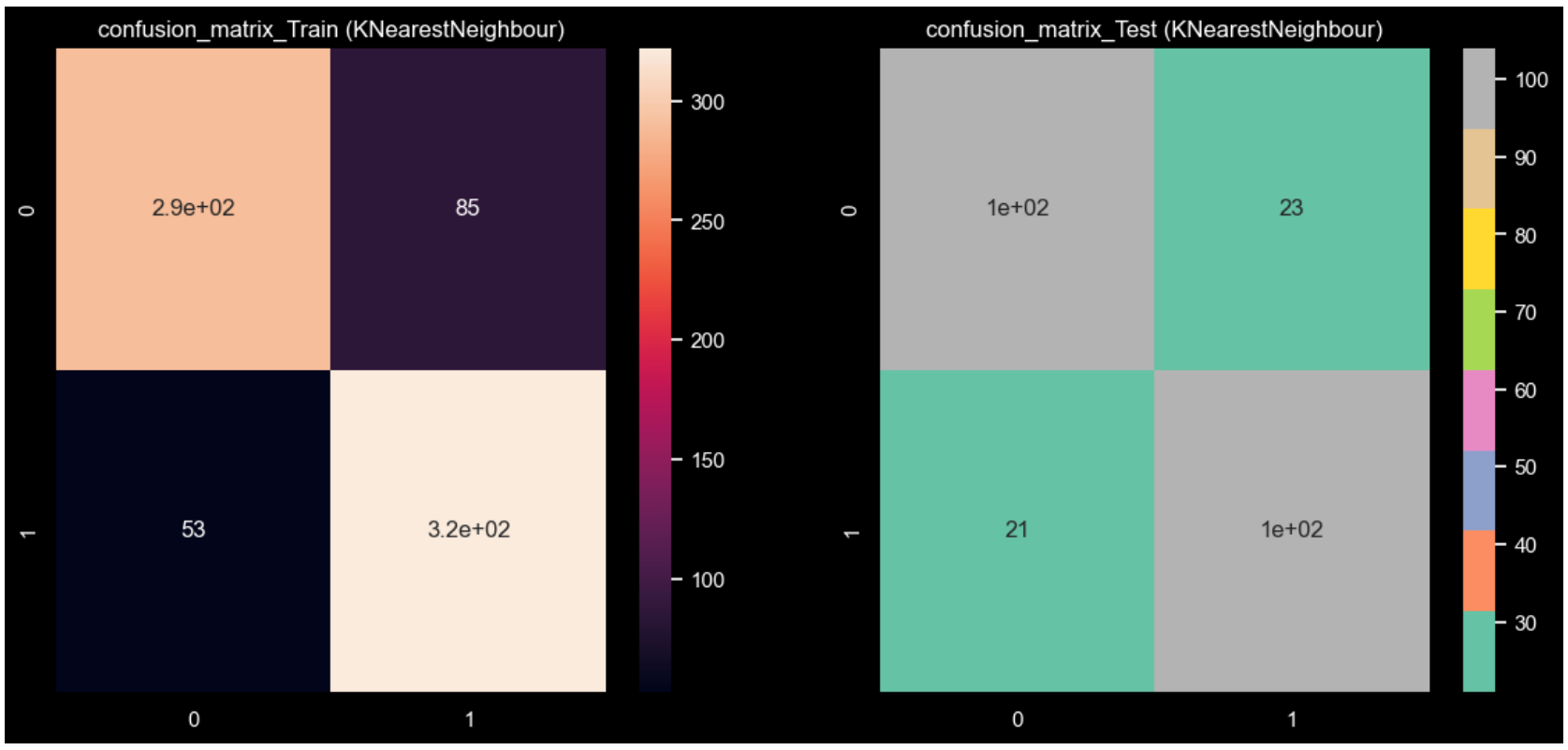
```
0.7240398293029872
0.6907051282051283
------------------------------------------------------------
========================================
confusion_matrix_Test :

 [[101  24]
 [ 30  95]]
========================================
confusion_matrix_Train :

 [[302  73]
 [ 84 291]]
========================================

************************************************************
                          Bagging
************************************************************
Accuracy  :

Train Score : 0.9
Test  Score : 0.83
Varience : 7.0

Classification_report:

Train:
------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.94      0.87      0.90       375
           1       0.88      0.94      0.91       375

    accuracy                           0.90       750
   macro avg       0.91      0.90      0.90       750
weighted avg       0.91      0.90      0.90       750

Test:
------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.88      0.76      0.82       125
           1       0.79      0.90      0.84       125

    accuracy                           0.83       250
   macro avg       0.83      0.83      0.83       250
weighted avg       0.83      0.83      0.83       250

------------------------------------------------------------
Area under ROC-AUC curve = 0.83
------------------------------------------------------------
Croos validation :

0.7758534850640113
0.7910256410256411
------------------------------------------------------------
========================================
confusion_matrix_Test :

 [[ 95  30]
 [ 13 112]]
========================================
confusion_matrix_Train :

 [[325  50]
 [ 22 353]]
========================================

************************************************************
                        RandomForest
************************************************************
Accuracy  :

Train Score : 0.9
Test  Score : 0.85
Varience : 5.0

Classification_report:

Train:
------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.95      0.85      0.90       375
           1       0.86      0.96      0.91       375

    accuracy                           0.90       750
   macro avg       0.91      0.90      0.90       750
weighted avg       0.91      0.90      0.90       750

Test:
```

```
------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.91      0.77      0.83       125
           1       0.80      0.93      0.86       125

    accuracy                           0.85       250
   macro avg       0.86      0.85      0.85       250
weighted avg       0.86      0.85      0.85       250

------------------------------------------------------------
Area under ROC-AUC curve = 0.85
------------------------------------------------------------
Croos validation :

0.7919630156472263
0.7993589743589744
------------------------------------------------------------
========================================
confusion_matrix_Test :

 [[ 96  29]
 [  9 116]]
========================================
confusion_matrix_Train :

 [[318  57]
 [ 16 359]]
========================================
************************************************************
                        KNearestNeighbour
************************************************************
Accuracy  :

Train Score : 0.82
Test  Score : 0.82
Varience : 0.0

Classification_report:

Train:
------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.85      0.77      0.81       375
           1       0.79      0.86      0.82       375

    accuracy                           0.82       750
   macro avg       0.82      0.82      0.82       750
weighted avg       0.82      0.82      0.82       750

Test:
------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.83      0.82      0.82       125
           1       0.82      0.83      0.83       125

    accuracy                           0.82       250
   macro avg       0.82      0.82      0.82       250
weighted avg       0.82      0.82      0.82       250

------------------------------------------------------------
Area under ROC-AUC curve = 0.82
------------------------------------------------------------
Croos validation :

0.7613086770981508
0.7711538461538461
------------------------------------------------------------
========================================
confusion_matrix_Test :

 [[102  23]
 [ 21 104]]
========================================
confusion_matrix_Train :

 [[290  85]
 [ 53 322]]
========================================

************************************************************
                          SupportVector
************************************************************
Accuracy  :

Train Score : 0.83
Test  Score : 0.84
Varience : 1.0
```

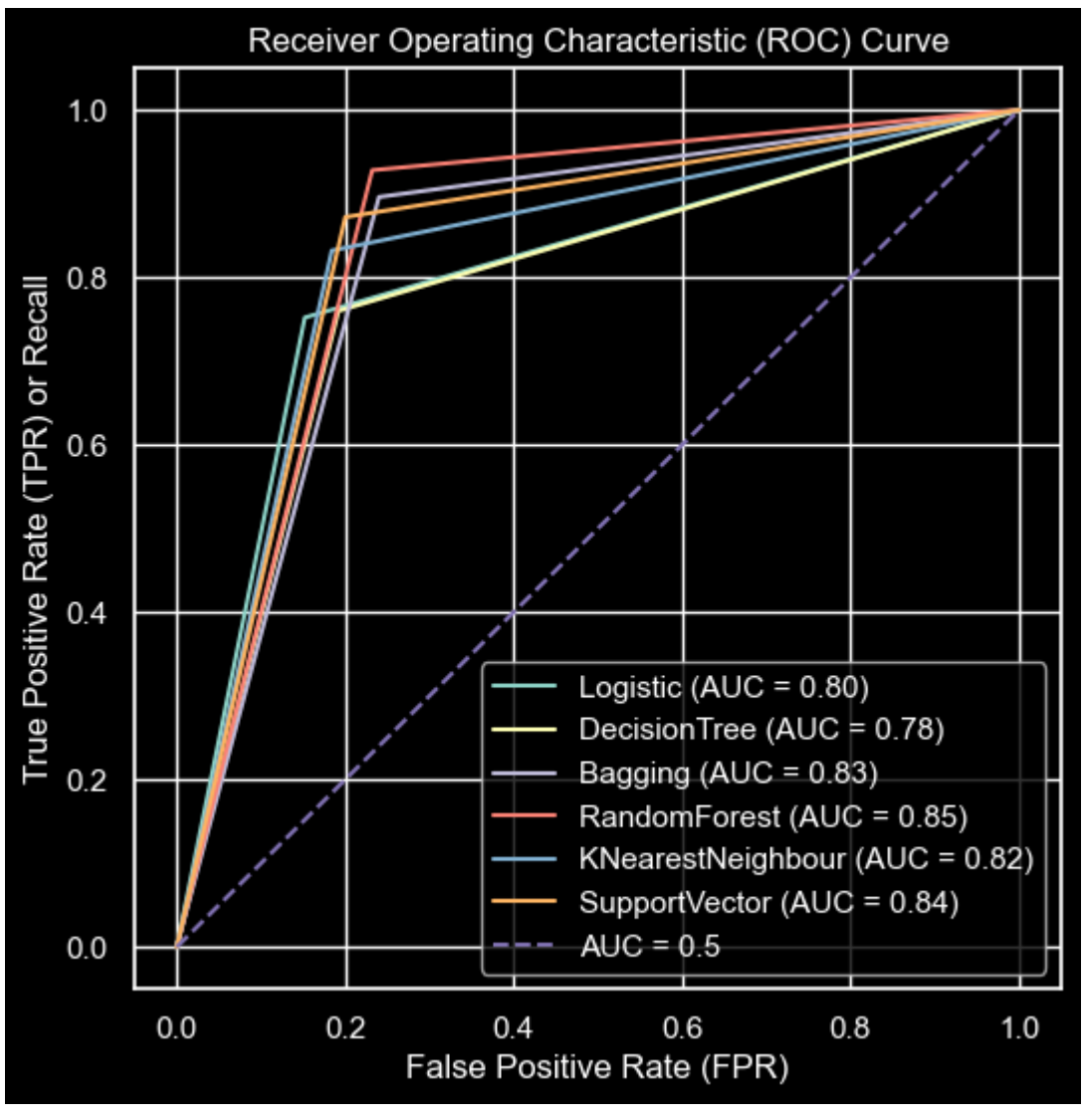
```
Classification_report:

Train:
------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.84      0.80      0.82       375
           1       0.81      0.85      0.83       375

    accuracy                           0.83       750
   macro avg       0.83      0.83      0.83       750
weighted avg       0.83      0.83      0.83       750

Test:
------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.86      0.80      0.83       125
           1       0.81      0.87      0.84       125

    accuracy                           0.84       250
   macro avg       0.84      0.84      0.84       250
weighted avg       0.84      0.84      0.84       250

------------------------------------------------------------
Area under ROC-AUC curve = 0.84
------------------------------------------------------------
Croos validation :

0.7615220483641536
0.7951923076923076
------------------------------------------------------------
========================================
confusion_matrix_Test :

 [[100  25]
 [ 16 109]]
========================================
confusion_matrix_Train :

 [[299  76]
 [ 55 320]]
========================================
```




confusion_matrix_Train (DecisionTree)
3e+02
73
84
2.9e+02
confusion_matrix_Test (DecisionTree)
1e+02
24
30
95
confusion_matrix_Train (Bagging)
3.2e+02
50
22
3.5e+02
confusion_matrix_Test (Bagging)
95
30
13
1.1e+02
confusion_matrix_Train (RandomForest)
3.2e+02
57
16
3.6e+02
confusion_matrix_Test (RandomForest)
96
29
9
1.2e+02


confusion_matrix_Train (KNearestNeighbour)
2.9e+02
85
53
3.2e+02
confusion_matrix_Test (KNearestNeighbour)
1e+02
23
21
1e+02
confusion_matrix_Train (SupportVector)
3e+02
76
55
3.2e+02
confusion_matrix_Test (SupportVector)
1e+02
25
16
1.1e+02

In [146]:
```
plt.figure(figsize=(6,6))

for model_name,fit_model, y_pred_train, y_pred in models:
    fpr, tpr, _ = roc_curve(y_test, y_pred)
    roc_auc = auc(fpr, tpr)

    plt.plot(fpr, tpr, label=f'{model_name} (AUC = {roc_auc:.2f})')

# Plot the diagonal line representing a random classifier (AUC = 0.5)

plt.plot([0, 1], [0, 1], 'm--', label='AUC = 0.5')

# Set labels and title

plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR) or Recall')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')

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



Summary of metrics of different Model

In [198]:
```
row_label=[ 'Logistic', 'DecisionTree','Bagging', 'RandomForest' , 'KNearestNeighbour', 'SVM']
len(row_label)
train_acc =[74, 79,90,90,82,83,]

test_acc =[ 80, 78, 83, 85, 82, 84]
auc =[0.8,0.78, 0.83, 0.85, 0.82, 0.84]
train_acc_CV =[73, 72,77,79,76, 76]
test_acc_CV =[80,69,79,79,79,79,]
sensitivity =[80,79,83,85,82,84]
specificity =[(100-i) for i in sensitivity]
precision =[80,79,83,86,82,84]
f1score=[80,79,83,85,82,84]

metric ={      'Train_Accuracy(%)' : train_acc,
               'Test_Accuracy(%)' : test_acc  ,
                'Sensitivity': sensitivity,
                'Specificity': specificity,
                'AUC':[i*100 for i in auc],
                'Train_Acc_CV':train_acc_CV,
                'Test_Acc_CV' :test_acc_CV,
                }
metric_df =pd.DataFrame(metric, index =row_label)
metric_df
```

Out[198]:

	Train_Accuracy(%)	Test_Accuracy(%)	Sensitivity	Specificity	AUC	Train_Acc_CV	Test_Acc_CV
Logistic	74	80	80	20	80.0	73	80
DecisionTree	79	78	79	21	78.0	72	69
Bagging	90	83	83	17	83.0	77	79
RandomForest	90	85	85	15	85.0	79	79
KNearestNeighbour	82	82	82	18	82.0	76	79
SVM	83	84	84	16	84.0	76	79

In [199]:
```
metric_df['varience']=abs(metric_df['Train_Accuracy(%)'] - metric_df['Test_Accuracy(%)'])
metric_df
```

Out[199]:

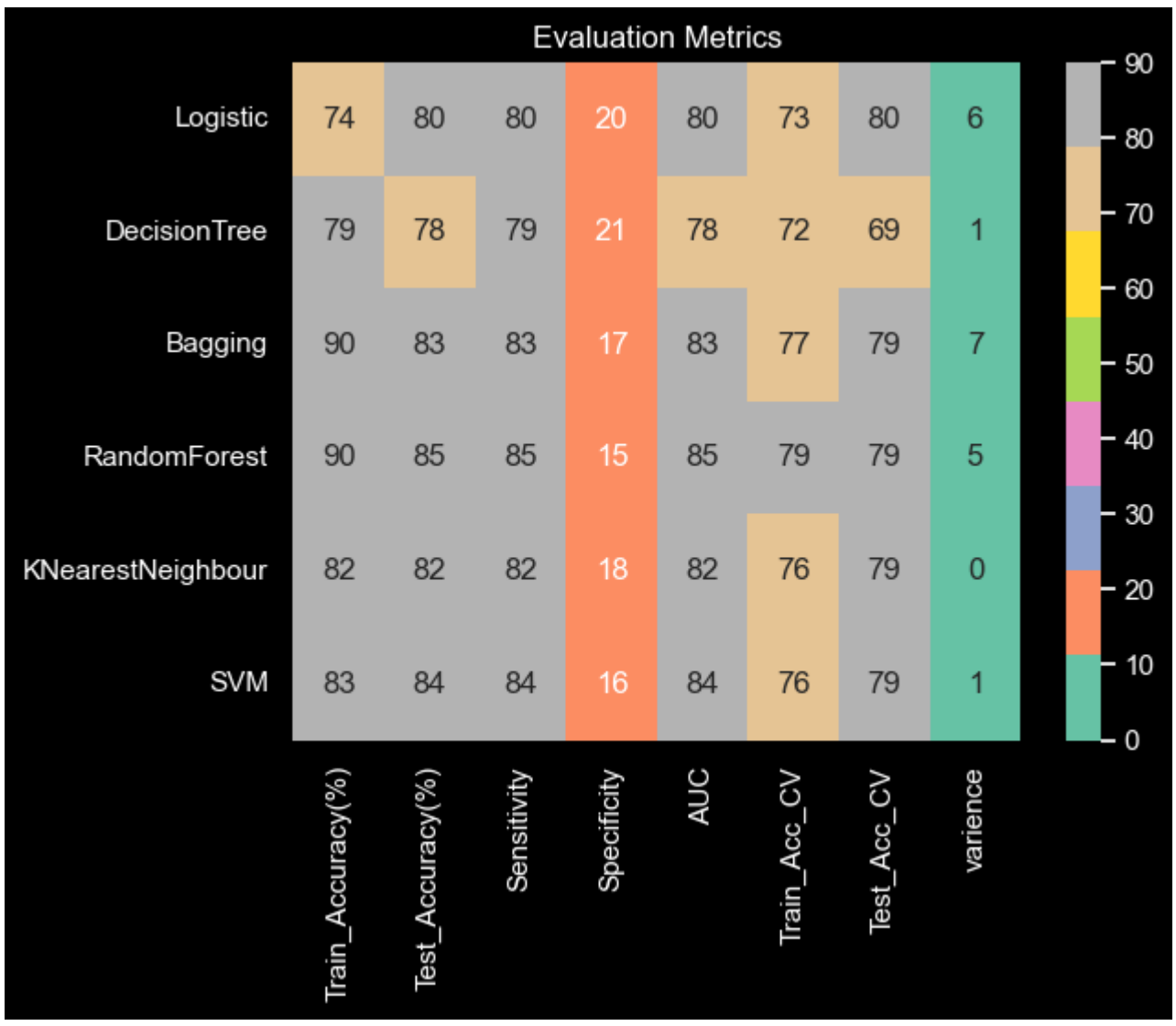
	Train_Accuracy(%)	Test_Accuracy(%)	Sensitivity	Specificity	AUC	Train_Acc_CV	Test_Acc_CV	varience
Logistic	74	80	80	20	80.0	73	80	6
DecisionTree	79	78	79	21	78.0	72	69	1
Bagging	90	83	83	17	83.0	77	79	7
RandomForest	90	85	85	15	85.0	79	79	5
KNearestNeighbour	82	82	82	18	82.0	76	79	0
SVM	83	84	84	16	84.0	76	79	1

In [210]:
```
condition = (metric_df['varience'] > 10) | (metric_df['Train_Accuracy(%)'] >= 99)
metric_df['overfitting'] = np.where(condition, True, False)
metric_df
```

Out[210]:

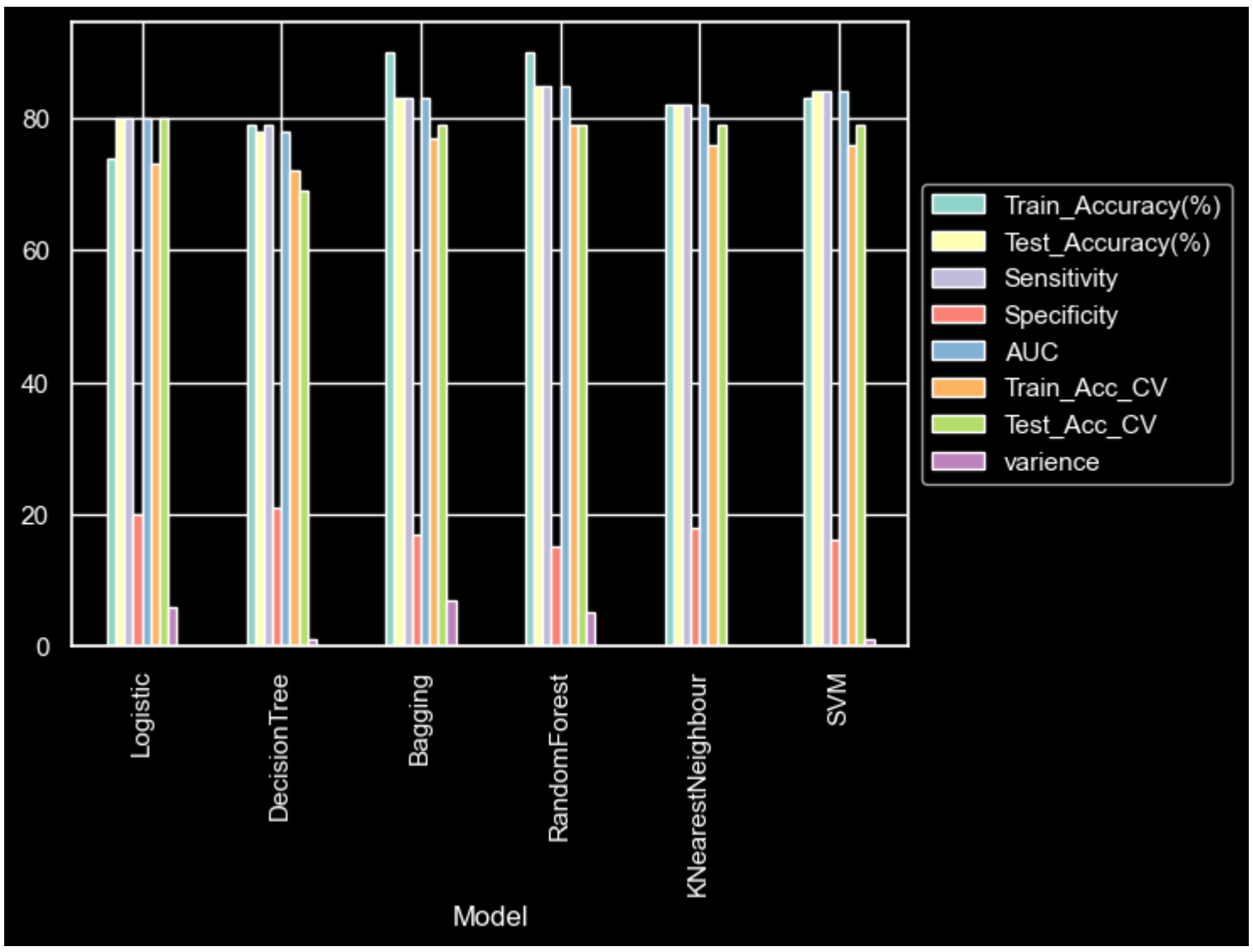
	Train_Accuracy(%)	Test_Accuracy(%)	Sensitivity	Specificity	AUC	Train_Acc_CV	Test_Acc_CV	varience	overfittin
Logistic	74	80	80	20	80.0	73	80	6	Fals
DecisionTree	79	78	79	21	78.0	72	69	1	Fals
Bagging	90	83	83	17	83.0	77	79	7	Fals
RandomForest	90	85	85	15	85.0	79	79	5	Fals
KNearestNeighbour	82	82	82	18	82.0	76	79	0	Fals
SVM	83	84	84	16	84.0	76	79	1	Fals

In [209]:
```
sns.heatmap(metric_df.iloc[:,:-1], annot = True, cmap= 'Set2', )
plt.title('Evaluation Metrics')
plt.show()
```



In [197]:
```
plt.figure(figsize=(12,6))
metric_df.plot(kind='bar',)
plt.legend(loc='center left', bbox_to_anchor=(1.0, 0.5))
plt.show()
```

```
<Figure size 1200x600 with 0 Axes>
```



📝📢🔎 Findings:

📌 - All the models are performing well without overfitting with hyper-parameter tunning.

> **Based on the information provided, it seems that the Random Forest model and Bagging classifier (estimator = KNN Model) has the highest Train accuracy (90%), however Test Accuracy is better for Random Forest than Bagging Classifier(85% and 83% respectively) and ROC AUC ($AUC_{RFC} = 0.86$) among the models evaluated.**

📌 -The classifiaction report shows the average Precision slightly more than Recall which indicates model is predicting true case efficiently.

🔎 for class 0 (Non-Diabetic): Recall < Precision :

> **The model misses some of the actual negative cases. It is not as good at correctly identifying all the negative instances in the dataset which is acceptibale**

🔎 for class 1 (Diabetic): Recall > Precision :

> **This suggests that the model is more inclusive in predicting positive cases for class 1. It efficiently classifies more positive instances, even if some of them are incorrect (false positives). Consequently, the model has a higher recall for class 1, which means that it identifies a larger portion of the actual positive cases (Actual diabetic patitents).**

confusion_matrix for Random Forest

In [217]:
```
matrix_rfc= np.array([['TP =96','FP=29'],['FN = 9','TN =116']])
print(matrix_rfc)
```

```
[['TP =96' 'FP=29']
 ['FN = 9' 'TN =116']]
```

Based on confusion matrix, False Positive (FP) > False negative (FN) which means model is predicting true cases efficiently.

> **False negatives occur when the model predicts an individual as non-diabetic when they are actually diabetic. This situation can be risky in a medical setting because it means the model failed to identify a person who may need medical attention and treatment for diabetes. By having lower recall for class 0 (non-diabetic), the model is less likely to miss potential diabetic patients (i.e., fewer false negatives), reducing the chances of undetected cases.**

📚 Acknowledgement:

I acknowledge and appreciate Learnaby, Swapnil Desai, Kumar Sundram, and Ashish Som for their dedicated efforts in teaching and sharing their knowledge in the fields of Python, Machine Learning, and Exploratory Data Analysis (EDA). By imparting their expertise, they have undoubtedly made a positive impact on my learning journeys.

Payal Mohanty

Project complition date= 1st August 2023

In [ ]:

In [ ]: