diabetes-project

October 9, 2023

#MeriSKILL Project2:Diabetes Project

#About Dataset: This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether a patient has diabetes based on certain diagnostic measurements included in the dataset. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.2 From the data set in the (.csv) File We can find several variables, some of them are independent (several medical predictor variables) and only one target dependent variable (Outcome).

Features name: (diabetes.csv)

Pregnancies

Glucose

BloodPressure

SkinThickness

Insulin

BMI

DiabetesPedigreeFunction

Age

Outcome

#Importing All Necessary Libraries:

[250]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
```

```python
from sklearn.metrics import accuracy_score, confusion_matrix,␣
↪classification_report
```

#Importing diabetes dataset:

[251]:
```python
diabetes = pd.read_csv('diabetes.csv')
```

[252]:
```python
diabetes
```

[252]:
```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0              6      148             72             35        0  33.6
1              1       85             66             29        0  26.6
2              8      183             64              0        0  23.3
3              1       89             66             23       94  28.1
4              0      137             40             35      168  43.1
..           …        …            …            …      …     …
763           10      101             76             48      180  32.9
764            2      122             70             27        0  36.8
765            5      121             72             23      112  26.2
766            1      126             60              0        0  30.1
767            1       93             70             31        0  30.4

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

[768 rows x 9 columns]
```

#Find out shape of the dataset.

It will give you Number of columns and rows present in the dataset

[253]:
```python
diabetes.shape
```

[253]:
```
(768, 9)
```

#Finding to see the how many columns present in the dataset.

[254]:
```python
diabetes.columns
```

[254]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

#Checking Non-Null Count and Datatype of each column present in the Diabetes dataset:

[255]:
```python
diabetes.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
```

#Checking null values present in the diabetes dataset columnwise:

[256]:
```python
diabetes.isnull().sum()
```

[256]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

#Checking number of unique values columnwise:

[257]:
```python
diabetes.nunique()
```

[257]:
```
Pregnancies                  17
Glucose                     136
BloodPressure                47
SkinThickness                51
Insulin                     186
```

```
BMI                          248
DiabetesPedigreeFunction     517
Age                           52
Outcome                        2
dtype: int64
```

#Describe the dataset:

[258]:
```
diabetes.describe()
```

[258]:
```
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   768.000000  768.000000     768.000000     768.000000  768.000000
mean      3.845052  120.894531      69.105469      20.536458   79.799479
std       3.369578   31.972618      19.355807      15.952218  115.244002
min       0.000000    0.000000       0.000000       0.000000    0.000000
25%       1.000000   99.000000      62.000000       0.000000    0.000000
50%       3.000000  117.000000      72.000000      23.000000   30.500000
75%       6.000000  140.250000      80.000000      32.000000  127.250000
max      17.000000  199.000000     122.000000      99.000000  846.000000

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  768.000000                768.000000  768.000000  768.000000
mean    31.992578                  0.471876   33.240885    0.348958
std      7.884160                  0.331329   11.760232    0.476951
min      0.000000                  0.078000   21.000000    0.000000
25%     27.300000                  0.243750   24.000000    0.000000
50%     32.000000                  0.372500   29.000000    0.000000
75%     36.600000                  0.626250   41.000000    1.000000
max     67.100000                  2.420000   81.000000    1.000000
```

#Checking Type of data present in each column:

[259]:
```
diabetes.Pregnancies.unique()
```

[259]:
```
array([ 6,  1,  8,  0,  5,  3, 10,  2,  4,  7,  9, 11, 13, 15, 17, 12, 14])
```

[260]:
```
diabetes.Glucose.unique()
```

[260]:
```
array([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])
```

[261]:
```python
diabetes.BloodPressure.unique()
```

[261]:
```
array([ 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])
```

[262]:
```python
diabetes.SkinThickness.unique()
```

[262]:
```
array([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])
```

[263]:
```python
diabetes.Insulin.unique()
```

[263]:
```
array([  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])
```

[264]:
```python
diabetes.BMI.unique()
```

[264]:
```
array([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])
```

[265]:
```
diabetes.DiabetesPedigreeFunction.unique()
```

[265]:
```
array([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])
```

[266]:
```
diabetes.Age.unique()
```

[266]:
```
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])
```

[267]:
```
diabetes.Outcome.unique()
```

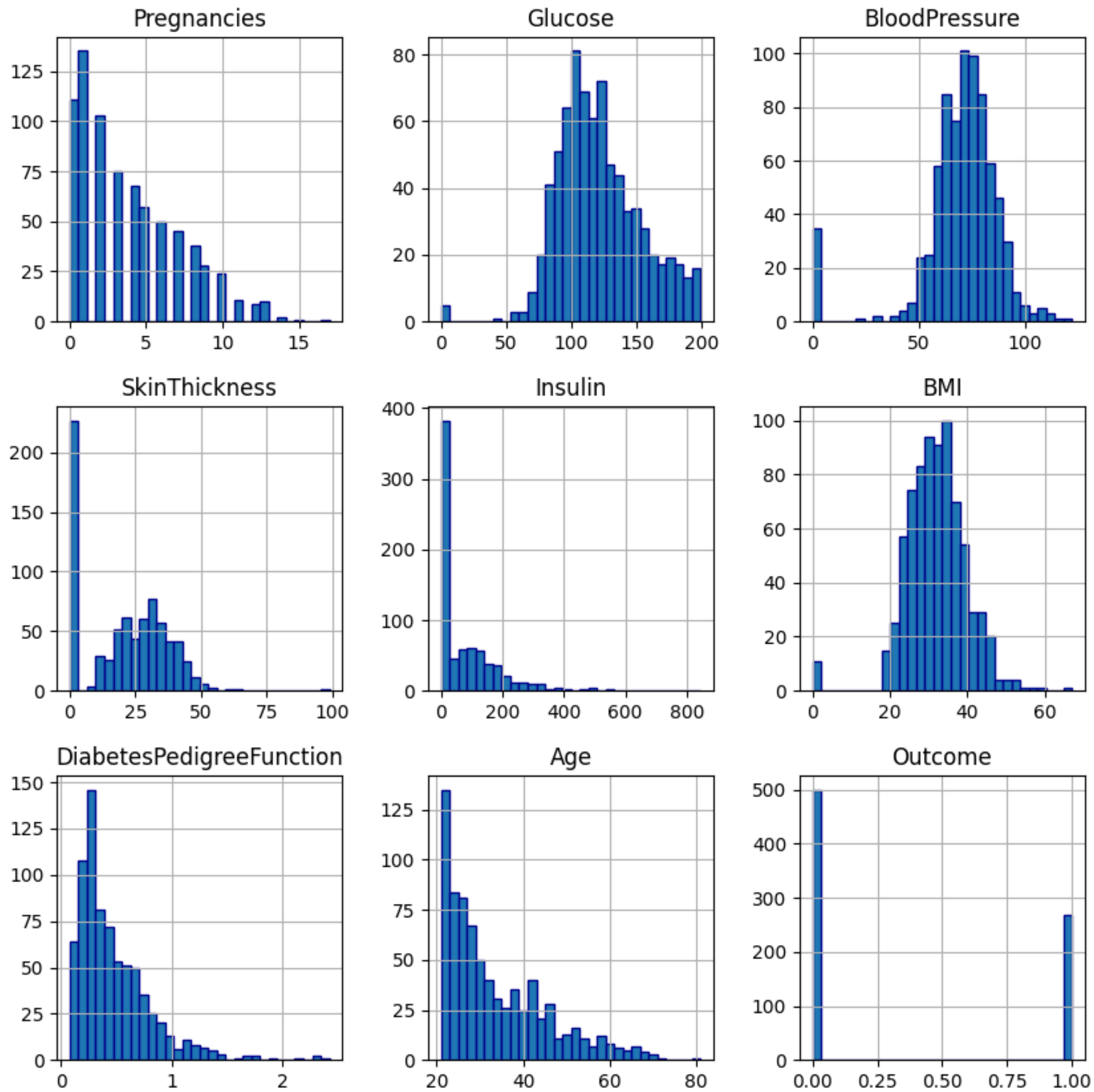
[267]:
```
array([1, 0])
```

#To see the Distribution of data present in each column using Histogram:

[268]:
```
diabetes.hist(bins = 30,figsize = (10,10),edgecolor = 'darkblue')
```

[268]:
```
array([[<Axes: title={'center': 'Pregnancies'}>,
        <Axes: title={'center': 'Glucose'}>,
        <Axes: title={'center': 'BloodPressure'}>],
       [<Axes: title={'center': 'SkinThickness'}>,
        <Axes: title={'center': 'Insulin'}>,
        <Axes: title={'center': 'BMI'}>],
```

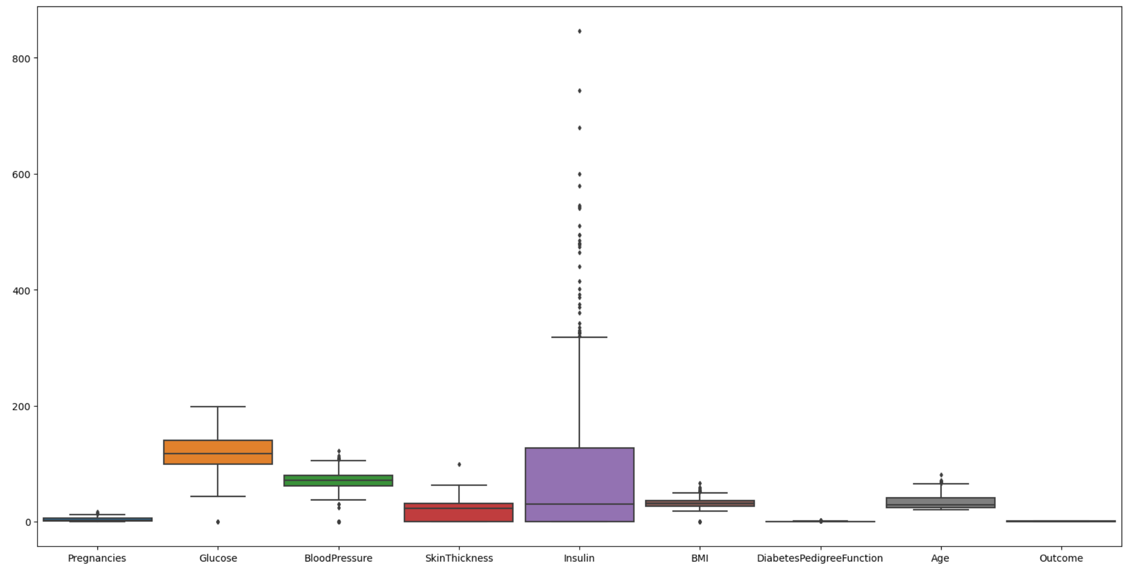
```
[<Axes: title={'center': 'DiabetesPedigreeFunction'}>,
 <Axes: title={'center': 'Age'}>,
 <Axes: title={'center': 'Outcome'}>]], dtype=object)
```



#To see the outliers present in each column using box plot:

[269]:
```
fig,ax = plt.subplots(figsize = (20,10))
sns.boxplot(data = diabetes,width = 0.9,fliersize = 3)
```

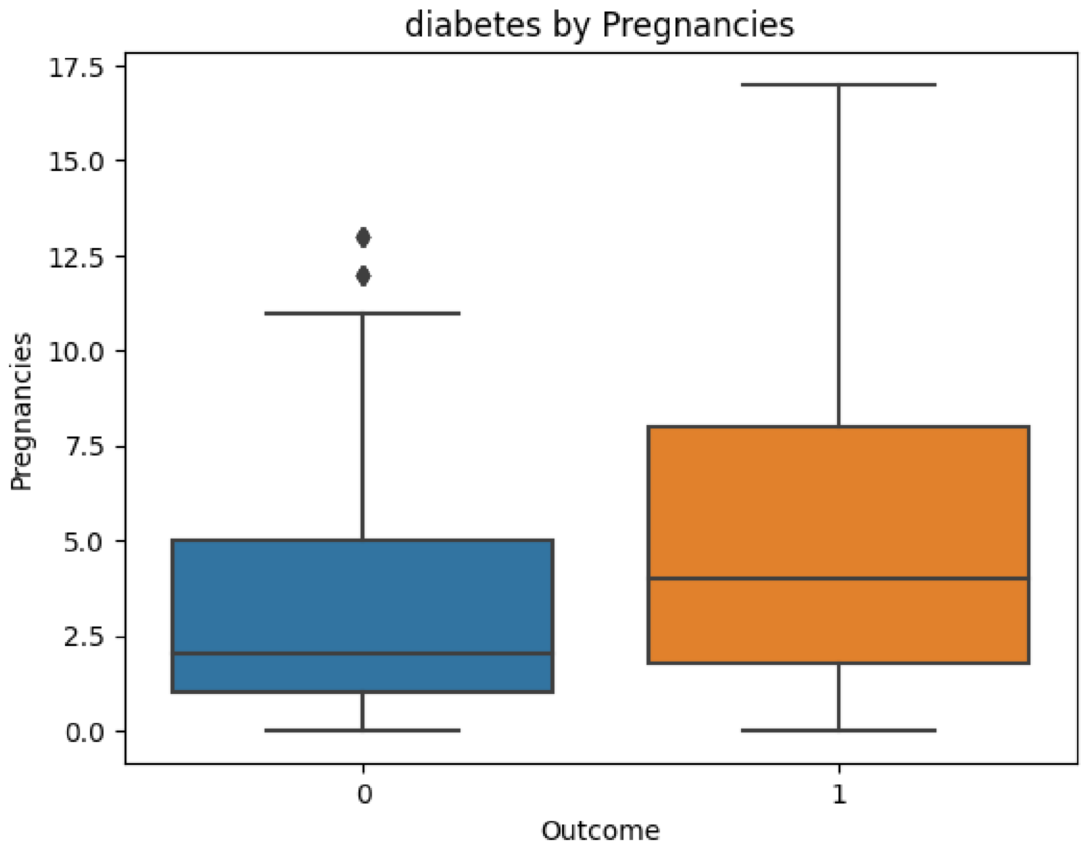
[269]: <Axes: >



#diabetes by Pregnancies:

[270]:
```
sns.boxplot(x = "Outcome", y = "Pregnancies", data = diabetes)
plt.title("diabetes by Pregnancies")
```

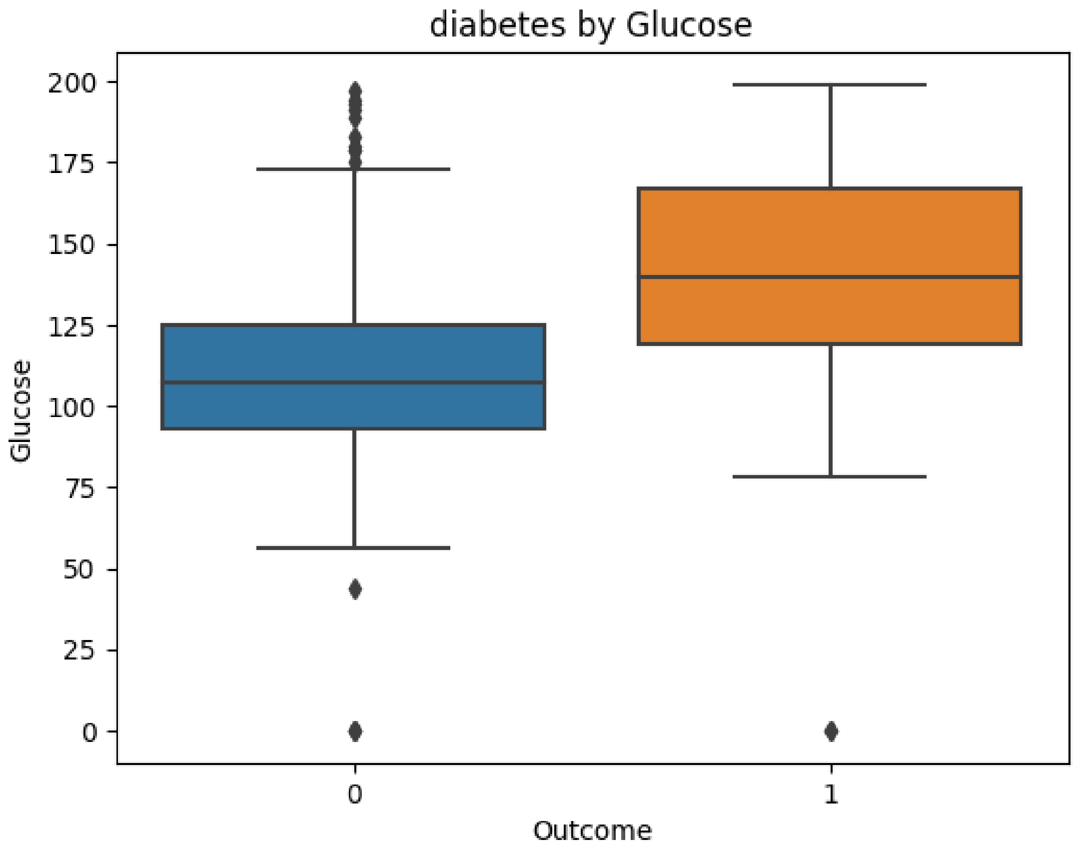
[270]: Text(0.5, 1.0, 'diabetes by Pregnancies')



#diabetes by Glucose:

[271]:
```
sns.boxplot(x = "Outcome", y = "Glucose", data = diabetes)
plt.title("diabetes by Glucose")
```

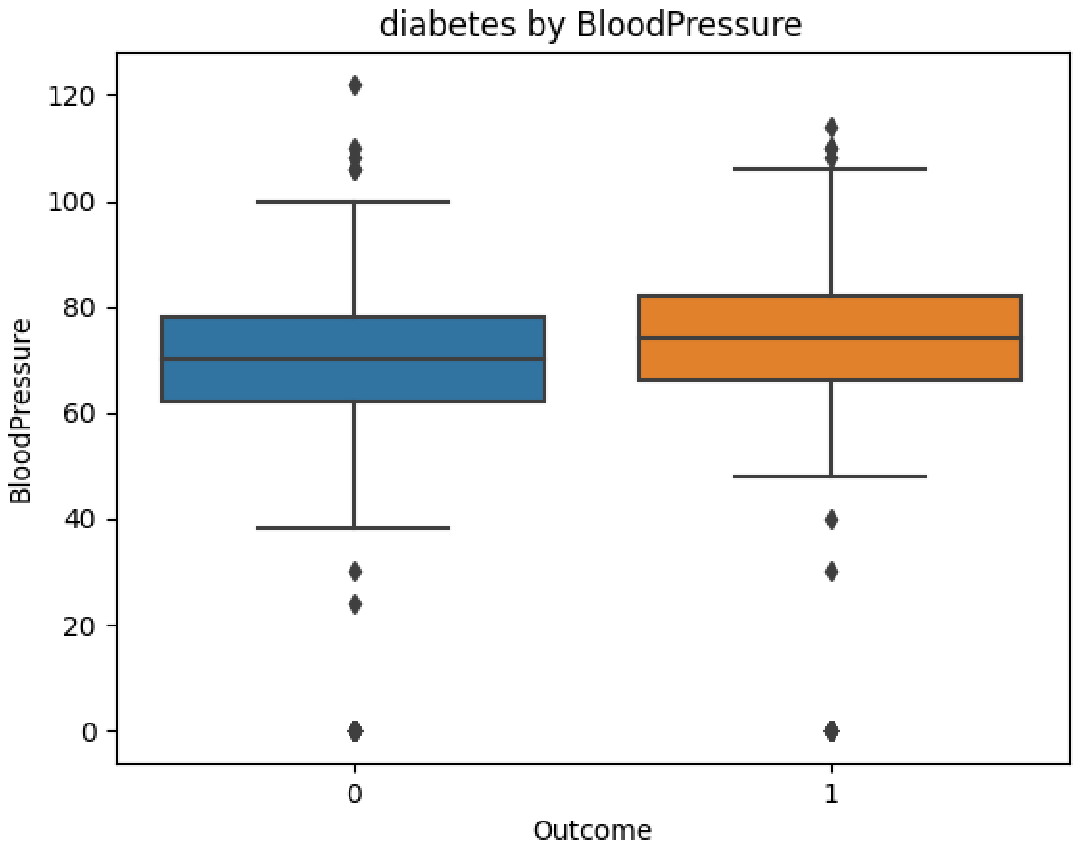
[271]: `Text(0.5, 1.0, 'diabetes by Glucose')`



#diabetes by BloodPressure:

[272]:
```
sns.boxplot(x = "Outcome", y = "BloodPressure", data = diabetes)
plt.title("diabetes by BloodPressure")
```

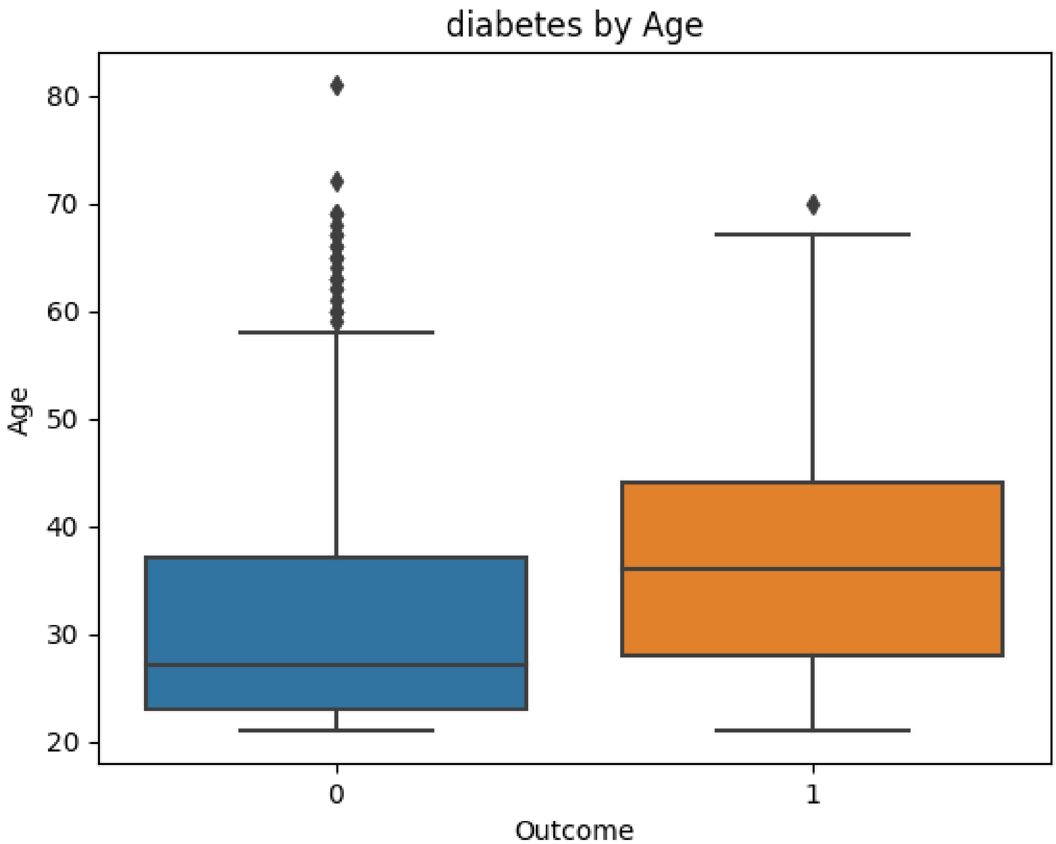
[272]: Text(0.5, 1.0, 'diabetes by BloodPressure')



#diabetes by Age:

[273]:
```
sns.boxplot(x = "Outcome", y = "Age", data = diabetes)
plt.title("diabetes by Age")
```

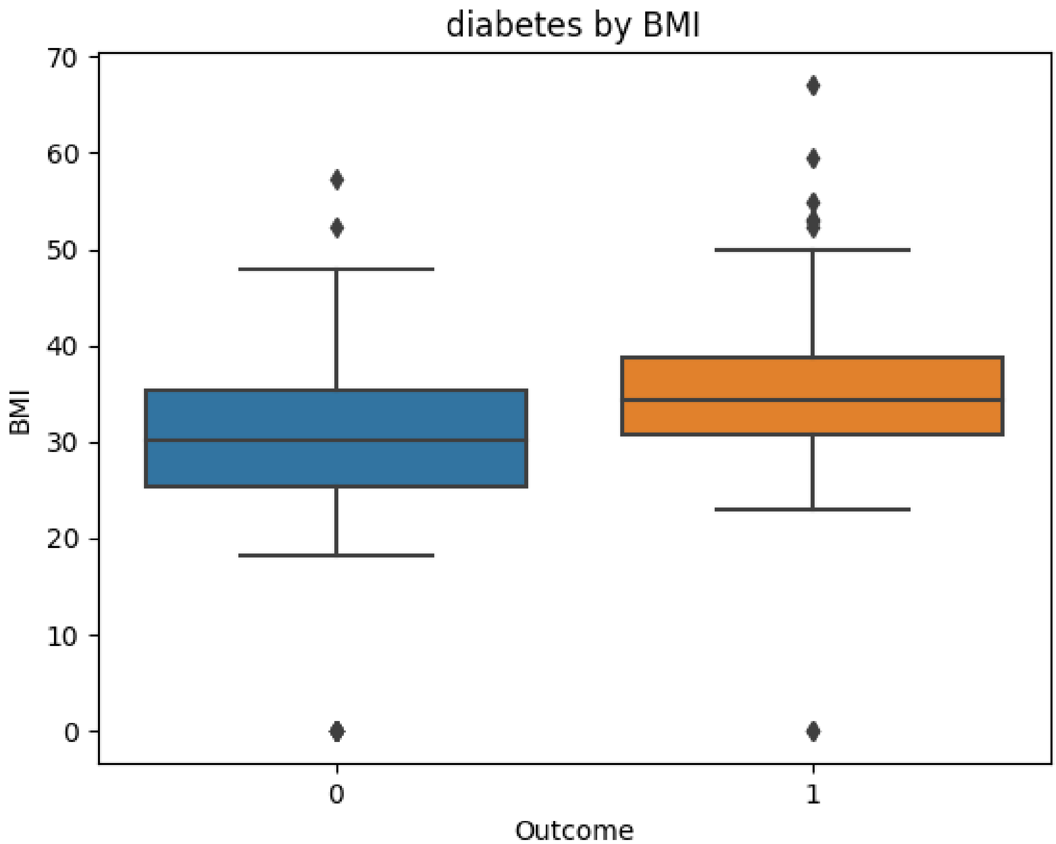
[273]: Text(0.5, 1.0, 'diabetes by Age')



#diabetes by BMI:

[274]:
```
sns.boxplot(x = "Outcome", y = "BMI", data = diabetes)
plt.title("diabetes by BMI")
```

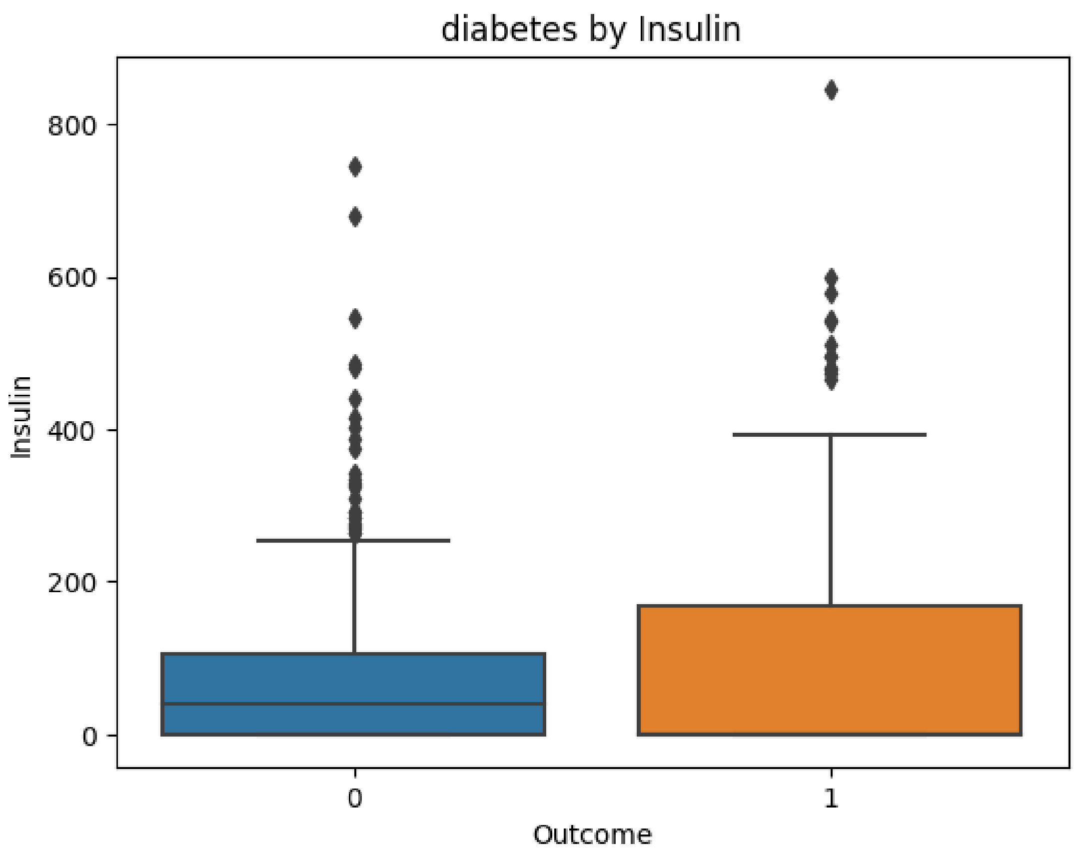
[274]: Text(0.5, 1.0, 'diabetes by BMI')



#diabetes by Insulin:

[275]:
```
sns.boxplot(x = "Outcome", y = "Insulin", data = diabetes)
plt.title("diabetes by Insulin")
```

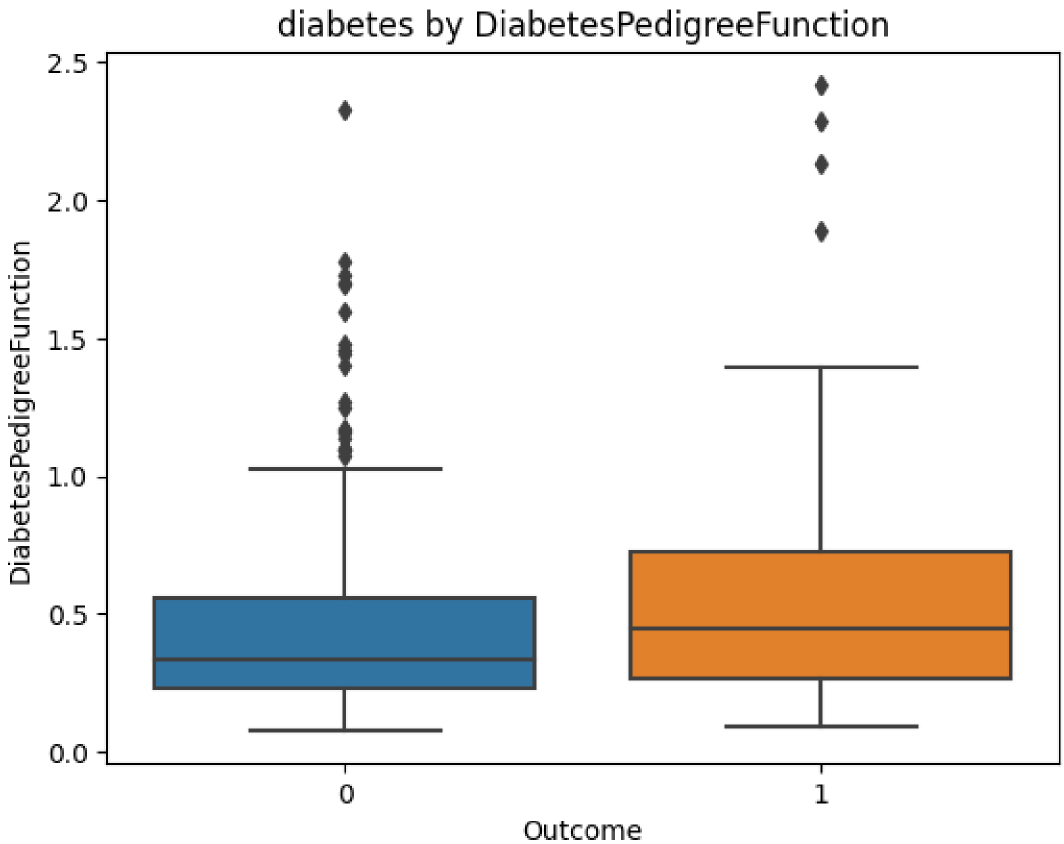
[275]: Text(0.5, 1.0, 'diabetes by Insulin')



#diabetes by DiabetesPedigreeFunction:

[276]:
```
sns.boxplot(x = "Outcome", y = "DiabetesPedigreeFunction", data = diabetes)
plt.title("diabetes by DiabetesPedigreeFunction")
```

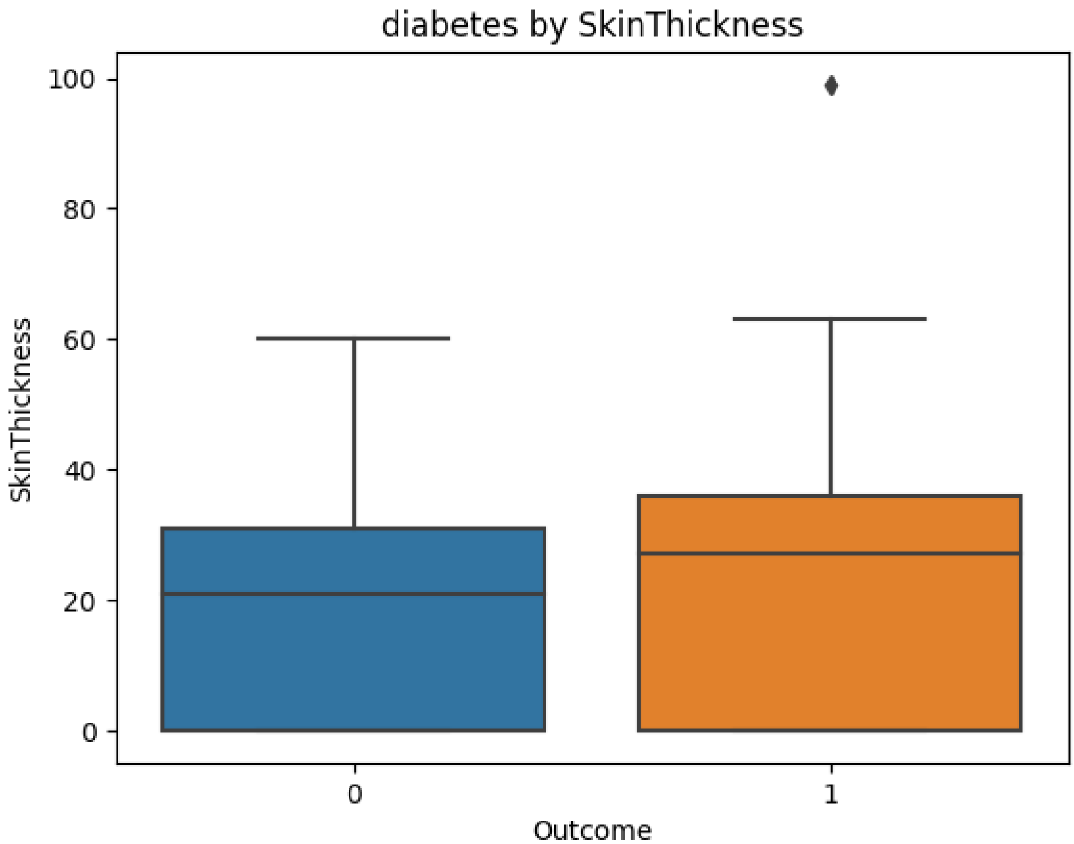
[276]: Text(0.5, 1.0, 'diabetes by DiabetesPedigreeFunction')



#diabetes by SkinThickness:

[277]:
```
sns.boxplot(x = "Outcome", y = "SkinThickness", data = diabetes)
plt.title("diabetes by SkinThickness")
```

[277]: Text(0.5, 1.0, 'diabetes by SkinThickness')



#Diabetes outcome:

Percentage of people who are having diabetes and not having Diabetes

0-Person not having diabetes

1-Person having diabetes

[278]:
```
px.pie(diabetes,names='Outcome', title='Diabetes outcome')
```

#diabetes by BMI and Glucose:

[279]:
```
px.scatter(diabetes,x = "BMI",y = "Glucose",color = "Outcome",title = "diabetes␣
↪by BMI and Glucose")
```

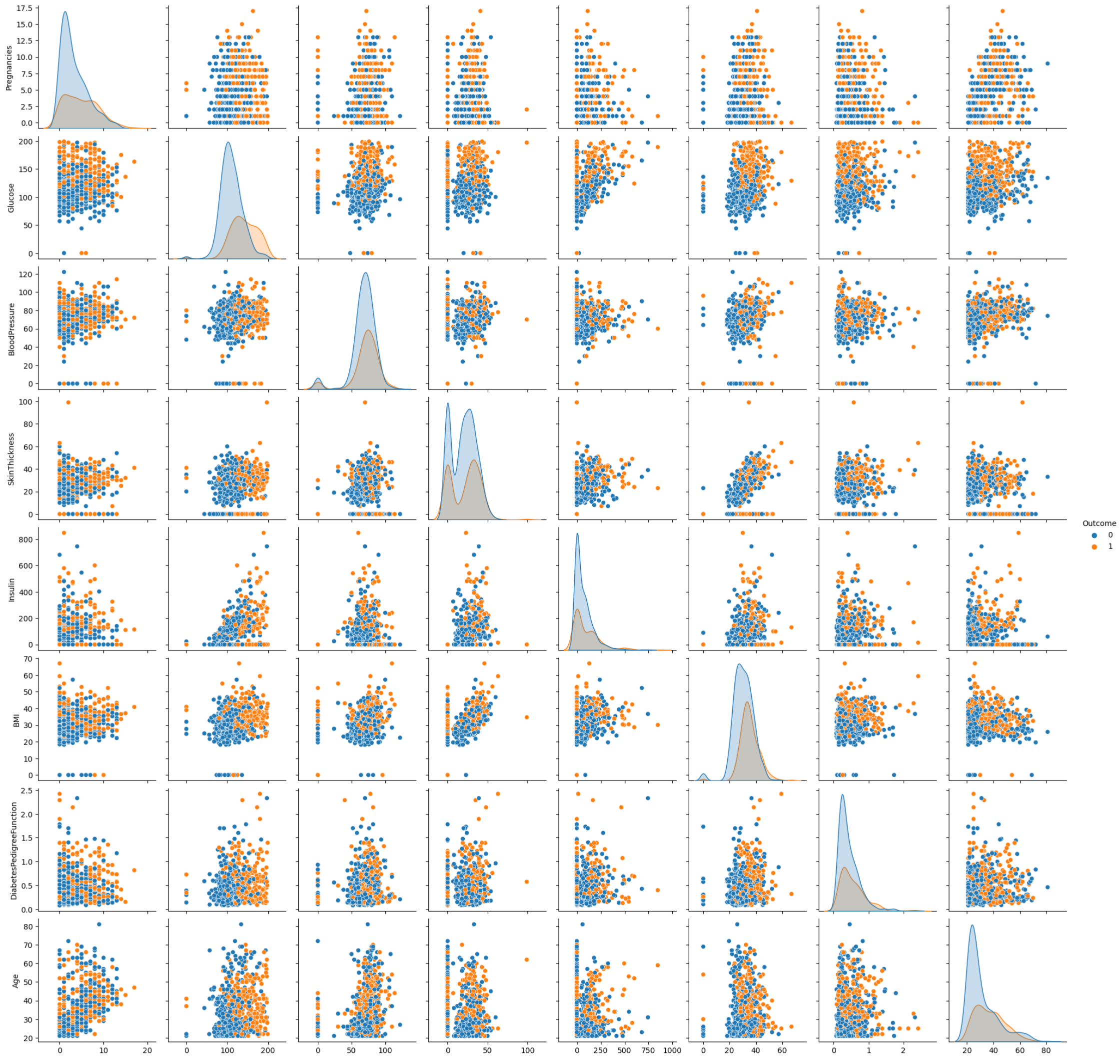
#diabetes by Age and BloodPressure:

[280]:
```
px.scatter(diabetes,x = "Age",y = "BloodPressure",color = "Outcome",title =␣
↪"diabetes by Age and BloodPressure")
```

#Pair Plot to see the relationship between Variables:

[281]:
```
sns.pairplot(data = diabetes,hue = 'Outcome')
```

[281]: <seaborn.axisgrid.PairGrid at 0x79d684fa1ba0>



#Finding duplicate rows in Table:

[282]:
```
diabetes.duplicated().sum()
```

[282]: 0

#Finding Zeros in Table:

[283]:
```
# Sum of counts of zeros for each column
zeros_sum = (diabetes == 0).sum()
```

```
print(zeros_sum)
```

```
Pregnancies                 111
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
Outcome                     500
dtype: int64
```

#Replacing Unwantedzeros present in the table with mean value of respected column:

[284]:
```
unvalid_zeros_in_columns = ['Glucose','BloodPressure','SkinThickness','BMI']
for column in unvalid_zeros_in_columns:
  mean = diabetes[column].mean()
  diabetes[column]=diabetes[column].replace(0,mean)
```

#Again Checking is zero's present in the columns:

[285]:
```
for column in unvalid_zeros_in_columns:
  count = (diabetes[column]==0).sum()
  print('count of zeros in column',column,'is:',count)
```

```
count of zeros in column Glucose is: 0
count of zeros in column BloodPressure is: 0
count of zeros in column SkinThickness is: 0
count of zeros in column BMI is: 0
```

#Finding outliers using IQR:

[286]:
```
# Function to find outliers using IQR
def find_outliers(column):
    Q1 = column.quantile(0.25)
    Q3 = column.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = (column < lower_bound) | (column > upper_bound)
    return outliers

# Finding outliers in each column
outliers_in_columns = diabetes.apply(find_outliers)

# Displaying the count of outliers in each column
outliers_count = outliers_in_columns.sum()
print("Number of outliers in each column:")
```

```
print(outliers_count)
```

```
Number of outliers in each column:
Pregnancies                  4
Glucose                      0
BloodPressure               14
SkinThickness               12
Insulin                     34
BMI                          8
DiabetesPedigreeFunction    29
Age                          9
Outcome                      0
dtype: int64
```

#Replacing outliers with median value of their respected columns:

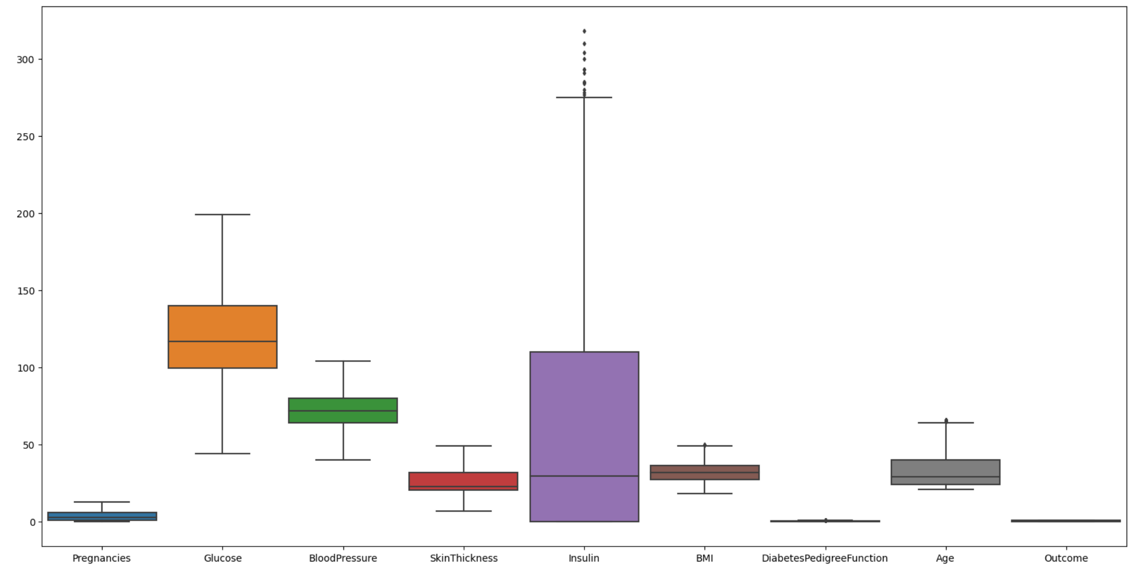
[287]:
```
def outlier_removal(diabetes, columns):
    for column in columns:
# Calculate the IQR and bounds
        quartiles = np.quantile(diabetes[column], [0.25, 0.75])
        iqr = quartiles[1] - quartiles[0]
        upper_bound = quartiles[1] + 1.5 * iqr
        lower_bound = quartiles[0] - 1.5 * iqr

# Replace outliers with median
        median_value = diabetes[column].median()
        diabetes[column] = np.where((diabetes[column] > upper_bound) |
 ↪(diabetes[column] < lower_bound), median_value, diabetes[column])

#Replace median value in the respective columns which contain ouliers.
columns_to_remove_outliers =
 ↪['Pregnancies','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFunction','
outlier_removal(diabetes, columns_to_remove_outliers)
```

#again checking outliers in the dataset after replacing it with median values:

in the below box plot it is showing outliers but those are not actual outliers,those are the part of data.

[288]:
```
fig,ax = plt.subplots(figsize = (20,10))
sns.boxplot(data = diabetes,width = 0.9,fliersize = 3)
```

[288]: <Axes: >



#Selecting Indepenent variables:

[291]:
```
X =␣
↪diabetes[['Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFuncti
X
```

[291]:
```
     Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0      148.0           72.0      35.000000      0.0  33.6
1       85.0           66.0      29.000000      0.0  26.6
2      183.0           64.0      20.536458      0.0  23.3
3       89.0           66.0      23.000000     94.0  28.1
4      137.0           40.0      35.000000    168.0  43.1
..       …              …            …        …    …
763    101.0           76.0      48.000000    180.0  32.9
764    122.0           70.0      27.000000      0.0  36.8
765    121.0           72.0      23.000000    112.0  26.2
766    126.0           60.0      20.536458      0.0  30.1
767     93.0           70.0      31.000000      0.0  30.4

     DiabetesPedigreeFunction   Age
0                      0.6270  50.0
1                      0.3510  31.0
2                      0.6720  32.0
3                      0.1670  21.0
4                      0.3725  33.0
..                        …    …
763                    0.1710  63.0
764                    0.3400  27.0
```

```
765                      0.2450  30.0
766                      0.3490  47.0
767                      0.3150  23.0

[768 rows x 7 columns]
```

#Selecting Target Variable::

[292]:
```
y = diabetes[['Outcome']]
y
```

[292]:
```
     Outcome
0          1
1          0
2          1
3          0
4          1
..       ...
763        0
764        0
765        0
766        1
767        0

[768 rows x 1 columns]
```

#Split the dataset into X_train, X_test, y_train, y_test:

[293]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
↪random_state=0)
```

[294]:
```
X_train
```

[294]:
```
     Glucose  BloodPressure  SkinThickness  Insulin        BMI  \
762     89.0           62.0      20.536458      0.0  22.500000
127    118.0           58.0      36.000000     94.0  33.300000
564     91.0           80.0      20.536458      0.0  32.400000
375    140.0           82.0      43.000000     30.5  39.200000
663    145.0           80.0      46.000000    130.0  37.900000
..       ...            ...            ...      ...        ...
763    101.0           76.0      48.000000    180.0  32.900000
192    159.0           66.0      20.536458      0.0  30.400000
629     94.0           65.0      22.000000      0.0  24.700000
559     85.0           74.0      20.536458      0.0  30.100000
684    136.0           82.0      20.536458      0.0  31.992578

     DiabetesPedigreeFunction   Age
762                     0.142  33.0
```

```
127                          0.261  23.0
564                          0.601  27.0
375                          0.528  58.0
663                          0.637  40.0
..                             …    …
763                          0.171  63.0
192                          0.383  36.0
629                          0.148  21.0
559                          0.300  35.0
684                          0.640  29.0

[576 rows x 7 columns]
```

[295]:
```
X_test
```

[295]:
```
     Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
661    199.0           76.0      43.000000      0.0  42.9
122    107.0           74.0      30.000000    100.0  33.6
113     76.0           62.0      20.536458      0.0  34.0
14     166.0           72.0      19.000000    175.0  25.8
529    111.0           65.0      20.536458      0.0  24.6
..       …             …             …        …    …
366    124.0           72.0      20.536458      0.0  27.6
301    144.0           58.0      33.000000    135.0  31.6
382    109.0           60.0       8.000000    182.0  25.4
140    128.0           78.0      20.536458      0.0  21.1
463     88.0           78.0      30.000000      0.0  27.6

     DiabetesPedigreeFunction   Age
661                    0.3725  22.0
122                    0.4040  23.0
113                    0.3910  25.0
14                     0.5870  51.0
529                    0.6600  31.0
..                        …    …
366                    0.3680  29.0
301                    0.4220  25.0
382                    0.9470  21.0
140                    0.2680  55.0
463                    0.2580  37.0

[192 rows x 7 columns]
```

[296]:
```
y_train
```

[296]:
```
     Outcome
762        0
```

```
127         0
564         0
375         1
663         1
..        ...
763         0
192         1
629         0
559         0
684         0

[576 rows x 1 columns]
```

[297]:
```
y_test
```

[297]:
```
     Outcome
661        1
122        0
113        0
14         1
529        0
..       ...
366        1
301        1
382        0
140        0
463        0

[192 rows x 1 columns]
```

#Performing StandardScaling:

[298]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
columns_to_scale =␣
↪['Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFunction','Age']
# Use the fit_transform method to scale the selected columns
X_train[columns_to_scale] = scaler.fit_transform(X_train[columns_to_scale])
X_test[columns_to_scale] = scaler.transform(X_test[columns_to_scale])
```

[299]:
```
X_train
```

[299]:
```
      Glucose  BloodPressure  SkinThickness   Insulin       BMI  \
762 -1.099479      -0.949426      -0.665373 -0.808896 -1.519640
127 -0.133147      -1.314271       1.130832  0.391396  0.134135
564 -1.032836       0.692376      -0.665373 -0.808896 -0.003679
375  0.599933       0.874798       1.943934 -0.419439  1.037586
```

```
663  0.766542       0.692376       2.292406  0.851083  0.838521
..        …             …             …         …         …
763 -0.699618       0.327531       2.524721  1.489536  0.072884
192  1.233047      -0.584581      -0.665373 -0.808896 -0.309934
629 -0.932870      -0.675792      -0.495372 -0.808896 -1.182760
559 -1.232767       0.145109      -0.665373 -0.808896 -0.355872
684  0.466645       0.874798      -0.665373 -0.808896 -0.066067

     DiabetesPedigreeFunction       Age
762                 -1.164414  0.005171
127                 -0.672609 -0.897409
564                  0.732549 -0.536377
375                  0.430853  2.261622
663                  0.881330  0.636977
..                        …         …
763                 -1.044562  2.712912
192                 -0.168405  0.275945
629                 -1.139617 -1.077925
559                 -0.511429  0.185687
684                  0.893729 -0.355861

[576 rows x 7 columns]
```

[300]:

```
X_test
```

[300]:

```
      Glucose  BloodPressure  SkinThickness   Insulin       BMI  \
661  2.565919       0.327531       1.943934 -0.808896  1.604157
122 -0.499687       0.145109       0.433888  0.468011  0.180073
113 -1.532663      -0.949426      -0.665373 -0.808896  0.241324
14   1.466299      -0.037314      -0.843844  1.425691 -1.014320
529 -0.366400      -0.675792      -0.665373 -0.808896 -1.198072
..        …             …             …         …         …
366  0.066784      -0.037314      -0.665373 -0.808896 -0.738691
301  0.733220      -1.314271       0.782360  0.914928 -0.126181
382 -0.433043      -1.131848      -2.121575  1.515074 -1.075571
140  0.200071       0.509954      -0.665373 -0.808896 -1.734018
463 -1.132801       0.509954       0.433888 -0.808896 -0.738691

     DiabetesPedigreeFunction       Age
661                 -0.211800 -0.987667
122                 -0.081616 -0.897409
113                 -0.135343 -0.716893
14                   0.674689  1.629816
529                  0.976385 -0.175345
..                        …         …
366                 -0.230397 -0.355861
301                 -0.007225 -0.716893
```

```
382                     2.162503 -1.077925
140                    -0.643679  1.990848
463                    -0.685007  0.366203

[192 rows x 7 columns]
```

#1)Model1:LogisticRegression:

[301]:
```python
# Create a logistic regression model instance
model = LogisticRegression()
#Train the model using the training sets
model.fit(X_train, y_train)
#Predict the response for test dataset
y_pred = model.predict(X_test)

#Accuracy,Confusion Matrix,Classification Report
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", class_report)
```

```
Accuracy: 0.7604166666666666
Confusion Matrix:
 [[113  17]
 [ 29  33]]
Classification Report:
               precision    recall  f1-score   support

           0       0.80      0.87      0.83       130
           1       0.66      0.53      0.59        62

    accuracy                           0.76       192
   macro avg       0.73      0.70      0.71       192
weighted avg       0.75      0.76      0.75       192


/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143:
DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the
shape of y to (n_samples, ), for example using ravel().
```

#2)Model2:Support Vector Machine:

[302]:

```
#Creating a svm Classifier
clf = svm.SVC(kernel='linear') # Linear Kernel

#Train the model using the training sets
clf.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)

#Accuracy,Confusion Matrix,Classification Report
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", class_report)
```

```
Accuracy: 0.7708333333333334
Confusion Matrix:
 [[113  17]
 [ 27  35]]
Classification Report:
               precision    recall  f1-score   support

           0       0.81      0.87      0.84       130
           1       0.67      0.56      0.61        62

    accuracy                           0.77       192
   macro avg       0.74      0.72      0.73       192
weighted avg       0.76      0.77      0.77       192


/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143:
DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the
shape of y to (n_samples, ), for example using ravel().
```

#3)Model3:RandomForestClassifier:

[303]:

```
#Fitting Decision Tree classifier to the training set
classifier= RandomForestClassifier(n_estimators= 9, criterion="entropy")

#Train the model using the training sets
classifier.fit(X_train, y_train)

#Predicting the test set result
```

```
y_pred= classifier.predict(X_test)

#Accuracy,Confusion Matrix,Classification Report
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", class_report)
```

```
Accuracy: 0.7604166666666666
Confusion Matrix:
 [[109  21]
 [ 25  37]]
Classification Report:
               precision    recall  f1-score   support

           0       0.81      0.84      0.83       130
           1       0.64      0.60      0.62        62

    accuracy                           0.76       192
   macro avg       0.73      0.72      0.72       192
weighted avg       0.76      0.76      0.76       192


<ipython-input-303-8e6f7911cad3>:5: DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the
shape of y to (n_samples,), for example using ravel().
```

#4)Model4:GradientBoostingClassifier:

[304]:
```
# Create a Gradient Boosting Classifier instance with 100 trees and a fixed␣
↪random seed
model = GradientBoostingClassifier(n_estimators=100, random_state=42)

#Train the model using the training sets
model.fit(X_train, y_train)

#Predicting the test set result
y_pred = model.predict(X_test)

#Accuracy,Confusion Matrix,Classification Report
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
print("Accuracy:", accuracy)
```

```python
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", class_report)
```

```
Accuracy: 0.7604166666666666
Confusion Matrix:
 [[113  17]
 [ 29  33]]
Classification Report:
               precision    recall  f1-score   support

           0       0.80      0.87      0.83       130
           1       0.66      0.53      0.59        62

    accuracy                           0.76       192
   macro avg       0.73      0.70      0.71       192
weighted avg       0.75      0.76      0.75       192


/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_gb.py:437:
DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the
shape of y to (n_samples, ), for example using ravel().
```

[305]:

```python
models_accuracy = np.array({'LogisticRegression':76,'SVM model':
  ↪77,'RandomForestClassifier':76,'GradientBoostingClassifier':76})
print(models_accuracy)
```

```
{'LogisticRegression': 76, 'SVM model': 77, 'RandomForestClassifier': 76,
'GradientBoostingClassifier': 76}
```