In [1]:

```python
import numpy as np
import pandas as pd
from collections import Counter

import seaborn as sns
sns.set(rc={'figure.figsize':(6,4)})
import matplotlib.pyplot as plt
%matplotlib inline

from tqdm import tqdm
import random
import pickle
import time

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MaxAbsScaler
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import QuantileTransformer
from sklearn.preprocessing import PowerTransformer
from sklearn.preprocessing import Normalizer

from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import log_loss
from sklearn.metrics import cohen_kappa_score
from sklearn.metrics import confusion_matrix
from sklearn import metrics

# Root Mean Square Error
from sklearn.metrics import mean_squared_error
from math import sqrt

# for ignore warnings
import warnings
warnings.filterwarnings("ignore")

plot_data_list = []
```

```
plot_RSME_list = []
```

In [2]:

```
df = pd.read_csv('Dataset\Pima diabetes_csv.csv')
df.head()
```

Out[2]:

	preg	plas	pres	skin	insu	mass	pedi	age	class
0	6	148	72	35	0	33.6	0.627	50	tested_positive
1	1	85	66	29	0	26.6	0.351	31	tested_negative
2	8	183	64	0	0	23.3	0.672	32	tested_positive
3	1	89	66	23	94	28.1	0.167	21	tested_negative
4	0	137	40	35	168	43.1	2.288	33	tested_positive

In [3]:

```
df.shape
```

Out[3]:

```
(768, 9)
```

In [4]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   preg    768 non-null    int64
 1   plas    768 non-null    int64
 2   pres    768 non-null    int64
 3   skin    768 non-null    int64
 4   insu    768 non-null    int64
 5   mass    768 non-null    float64
 6   pedi    768 non-null    float64
 7   age     768 non-null    int64
 8   class   768 non-null    object
dtypes: float64(2), int64(6), object(1)
memory usage: 54.1+ KB
```

In [5]:
```
round(df.describe(),2)
```

Out[5]:

	preg	plas	pres	skin	insu	mass	pedi	age
count	768.00	768.00	768.00	768.00	768.00	768.00	768.00	768.00
mean	3.85	120.89	69.11	20.54	79.80	31.99	0.47	33.24
std	3.37	31.97	19.36	15.95	115.24	7.88	0.33	11.76
min	0.00	0.00	0.00	0.00	0.00	0.00	0.08	21.00
25%	1.00	99.00	62.00	0.00	0.00	27.30	0.24	24.00
50%	3.00	117.00	72.00	23.00	30.50	32.00	0.37	29.00
75%	6.00	140.25	80.00	32.00	127.25	36.60	0.63	41.00
max	17.00	199.00	122.00	99.00	846.00	67.10	2.42	81.00

In [6]:
```
# check null
df.isnull().sum()
```

Out[6]:
```
preg     0
plas     0
pres     0
skin     0
insu     0
mass     0
pedi     0
age      0
class    0
dtype: int64
```

In [7]:

```python
# list of numerical variables
numerical_features = [feature for feature in df.columns if df[feature].dtypes != 'O']

print('Number of numerical variables: ', len(numerical_features))

numerical_variables_df = df[numerical_features]

# visualise the numerical variables
df[numerical_features].head()
```

```
Number of numerical variables:  8
```

Out[7]:

	preg	plas	pres	skin	insu	mass	pedi	age
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33

In [8]:

```
# Histogram:distribution of a single numerical variable
numerical_variables_df.hist(bins=50, figsize=(20, 15))
plt.show()
```

preg
plas
pres
skin
insu
mass
pedi
age

In [10]:

```
# Density plots for all attributes to visualize the distribution of each attribute
numerical_variables_df.plot(kind='density', subplots=True, layout=(3,3), figsize=(20, 15), sharex=False)
plt.show()
```

preg
Density
plas
Density
pres
Density
skin
Density
insu
Density
mass
Density
pedi
Density
age
Density

In [11]:

```
# Box Plot (Box-and-Whisker Plot): Summarizes the distribution of a numerical variable's
numerical_variables_df.plot(kind='box', subplots=True, layout=(3,3), sharex=False, sharey=False, figsize=(20,15))
plt.show()
```

17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
preg
200
175
150
125
100
75
50
25
0
plas
120
100
80
60
40
20
0
pres
100
80
60
40
20
0
skin
800
600
400
200
0
insu
70
60
50
40
30
20
10
0
mass
2.5
2.0
1.5
1.0
0.5
0.0
pedi
80
70
60
50
40
30
20
age

observation : outlier is detected in all features

In [12]:

```
# Correlation between the different characteristics. Closer to 1 better is the correlation.
sns.heatmap(round(numerical_variables_df.corr(method='pearson'),2), annot = True)
plt.show()
```

In [15]:

```
numerical_variables_df_class = pd.concat([numerical_variables_df, df['class']], axis=1)
# Pairplot
sns.pairplot(numerical_variables_df_class, hue='class')
plt.show()
```

preg
plas
pres
skin
insu
mass
pedi
class
tested_positive
tested_negative

In [18]:

```
df['classification'] = df['class'].apply(lambda x: 1 if x=='tested_negative' else 0)
# Remove two columns name is 'class'
df.drop(['class'], axis=1, inplace=True)
df.head(10)
```

Out[18]:

	preg	plas	pres	skin	insu	mass	pedi	age	classification
0	6	148	72	35	0	33.6	0.627	50	0
1	1	85	66	29	0	26.6	0.351	31	1
2	8	183	64	0	0	23.3	0.672	32	0
3	1	89	66	23	94	28.1	0.167	21	1
4	0	137	40	35	168	43.1	2.288	33	0
5	5	116	74	0	0	25.6	0.201	30	1
6	3	78	50	32	88	31.0	0.248	26	0
7	10	115	0	0	0	35.3	0.134	29	1
8	2	197	70	45	543	30.5	0.158	53	0
9	8	125	96	0	0	0.0	0.232	54	0

Remove Outlier using IQR

In [19]:

```python
def remove_outliers_iqr_all_columns(df):
    # Create an empty DataFrame to store filtered data
    filtered_df = pd.DataFrame()

    # Iterate through each column in the input DataFrame
    for col in df.columns:
        # Calculate the first quartile (Q1) and third quartile (Q3) for the column
        q1 = df[col].quantile(0.25)
        q3 = df[col].quantile(0.75)

        # Calculate the IQR (Interquartile Range) for the column
        iqr = q3 - q1

        # Define the lower and upper bounds to identify outliers for the column
        lower_bound = q1 - 1.5 * iqr
        upper_bound = q3 + 1.5 * iqr

        # Remove outliers from the column and add it to the filtered DataFrame
        filtered_df[col] = df[(df[col] >= lower_bound) & (df[col] <= upper_bound)][col]

        # Drop nan value from the column and add it to the filtered DataFrame
        filtered_df = filtered_df.dropna(axis=0).reset_index(drop=True)

    return filtered_df

df = remove_outliers_iqr_all_columns(df)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 629 entries, 0 to 628
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   preg            629 non-null    int64
 1   plas            629 non-null    float64
 2   pres            629 non-null    float64
 3   skin            629 non-null    float64
 4   insu            629 non-null    float64
 5   mass            629 non-null    float64
 6   pedi            629 non-null    float64
 7   age             629 non-null    float64
 8   classification  629 non-null    int64
dtypes: float64(7), int64(2)
memory usage: 44.4 KB
```

Check SMOTE for Imbalanced Classification

In [22]:

```python
# Check If A Dataset Is Imbalanced
sns.countplot(x=df["classification"])
plt.title('Countplot for Classification')
plt.xlabel('Classification')
plt.ylabel('Count')
plt.show()
```

In [23]:

```python
y = df['classification']

# summarize class distribution
counter = Counter(y)
print(f"Classification class distribution : {counter}")

# summarize class proportion
unique_labels, counts = np.unique(y, return_counts=True)
proportions = (counts / len(y))*100
print(f"Proportion of samples in each class (0,1) : {proportions}")
```

```
Classification class distribution : Counter({1: 413, 0: 216})
Proportion of samples in each class (0,1) : [34.34022258 65.65977742]
```

In [24]:

```python
# get unique values from columns
caltegorical_feature, numarical_feature =[], []
for col in df.columns:
    print(f"Unique values in {col} is : {sorted(df[col].unique())}, Count : {len(sorted(df[col].unique()))}")
    if len(sorted(df[col].unique())) > 150:
        numarical_feature.append(col)
    else:
        caltegorical_feature.append(col)

print(f"Numarical features : {numarical_feature}")
print(f"Caltegorical features : {caltegorical_feature}")
```

```
Unique values in preg is : [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13], Count : 14
Unique values in plas is : [44.0, 56.0, 57.0, 61.0, 62.0, 65.0, 67.0, 68.0, 71.0, 73.0, 74.0, 75.0, 76.0, 77.0, 78.0, 79.0, 80.0, 81.0, 82.0, 83.
0, 84.0, 85.0, 86.0, 87.0, 88.0, 89.0, 90.0, 91.0, 92.0, 93.0, 94.0, 95.0, 96.0, 97.0, 98.0, 99.0, 100.0, 101.0, 102.0, 103.0, 104.0, 105.0, 106.
0, 107.0, 108.0, 109.0, 110.0, 111.0, 112.0, 113.0, 114.0, 115.0, 116.0, 117.0, 118.0, 119.0, 120.0, 121.0, 122.0, 123.0, 124.0, 125.0, 126.0, 12
7.0, 128.0, 129.0, 130.0, 131.0, 132.0, 133.0, 134.0, 135.0, 136.0, 137.0, 138.0, 139.0, 140.0, 141.0, 142.0, 143.0, 144.0, 145.0, 146.0, 147.0,
148.0, 149.0, 150.0, 151.0, 152.0, 153.0, 154.0, 155.0, 156.0, 157.0, 158.0, 159.0, 160.0, 161.0, 162.0, 163.0, 164.0, 165.0, 166.0, 167.0, 168.
0, 170.0, 171.0, 172.0, 173.0, 174.0, 175.0, 176.0, 177.0, 178.0, 179.0, 180.0, 181.0, 183.0, 184.0, 186.0, 187.0, 188.0, 189.0, 190.0, 191.0, 19
3.0, 194.0, 195.0, 196.0, 197.0, 198.0, 199.0], Count : 132
Unique values in pres is : [38.0, 44.0, 46.0, 48.0, 50.0, 52.0, 54.0, 55.0, 56.0, 58.0, 60.0, 61.0, 62.0, 64.0, 65.0, 66.0, 68.0, 70.0, 72.0, 74.
0, 75.0, 76.0, 78.0, 80.0, 82.0, 84.0, 85.0, 86.0, 88.0, 90.0, 92.0, 94.0, 95.0, 96.0, 98.0, 100.0, 102.0, 104.0, 106.0], Count : 39
Unique values in skin is : [0.0, 7.0, 8.0, 10.0, 11.0, 12.0, 13.0, 14.0, 15.0, 16.0, 17.0, 18.0, 19.0, 20.0, 21.0, 22.0, 23.0, 24.0, 25.0, 26.0,
27.0, 28.0, 29.0, 30.0, 31.0, 32.0, 33.0, 34.0, 35.0, 36.0, 37.0, 38.0, 39.0, 40.0, 41.0, 42.0, 43.0, 44.0, 45.0, 46.0, 47.0, 48.0, 49.0, 50.0, 5
1.0, 52.0, 54.0, 56.0, 60.0, 63.0], Count : 50
Unique values in insu is : [0.0, 14.0, 15.0, 18.0, 22.0, 23.0, 25.0, 29.0, 32.0, 36.0, 37.0, 38.0, 40.0, 41.0, 42.0, 43.0, 44.0, 45.0, 46.0, 48.
0, 49.0, 50.0, 51.0, 52.0, 53.0, 54.0, 55.0, 56.0, 57.0, 58.0, 59.0, 60.0, 63.0, 64.0, 65.0, 66.0, 67.0, 68.0, 70.0, 71.0, 72.0, 73.0, 74.0, 75.
0, 76.0, 77.0, 78.0, 79.0, 82.0, 83.0, 84.0, 85.0, 86.0, 87.0, 88.0, 89.0, 90.0, 91.0, 92.0, 94.0, 95.0, 96.0, 99.0, 100.0, 105.0, 106.0, 108.0,
110.0, 114.0, 115.0, 116.0, 119.0, 120.0, 122.0, 125.0, 126.0, 129.0, 130.0, 132.0, 135.0, 140.0, 142.0, 144.0, 145.0, 146.0, 148.0, 150.0, 152.
0, 155.0, 156.0, 158.0, 159.0, 160.0, 165.0, 166.0, 167.0, 168.0, 170.0, 171.0, 175.0, 176.0, 180.0, 182.0, 183.0, 184.0, 185.0, 188.0, 190.0, 19
1.0, 192.0, 193.0, 194.0, 196.0, 200.0, 205.0, 207.0, 210.0, 215.0, 220.0, 225.0, 228.0, 230.0, 231.0, 235.0, 237.0, 240.0, 245.0, 249.0, 250.0,
255.0, 258.0, 265.0, 270.0, 271.0, 272.0, 274.0, 275.0, 277.0, 278.0, 280.0, 284.0, 285.0, 293.0, 300.0, 304.0, 310.0, 318.0], Count : 147
Unique values in mass is : [18.2, 18.4, 19.1, 19.3, 19.4, 19.5, 19.6, 19.9, 20.0, 20.1, 20.4, 20.8, 21.0, 21.1, 21.2, 21.7, 21.8, 21.9, 22.1, 22.
2, 22.3, 22.4, 22.5, 22.6, 22.7, 22.9, 23.0, 23.1, 23.2, 23.3, 23.4, 23.5, 23.6, 23.7, 23.8, 23.9, 24.0, 24.2, 24.3, 24.4, 24.5, 24.6, 24.7, 24.
8, 24.9, 25.0, 25.1, 25.2, 25.3, 25.4, 25.5, 25.6, 25.8, 25.9, 26.0, 26.1, 26.2, 26.3, 26.4, 26.5, 26.6, 26.7, 26.8, 26.9, 27.0, 27.1, 27.2, 27.
3, 27.4, 27.5, 27.6, 27.7, 27.8, 27.9, 28.0, 28.1, 28.2, 28.3, 28.4, 28.5, 28.6, 28.7, 28.8, 28.9, 29.0, 29.2, 29.3, 29.5, 29.6, 29.7, 29.8, 29.
9, 30.0, 30.1, 30.2, 30.3, 30.4, 30.5, 30.7, 30.8, 30.9, 31.0, 31.1, 31.2, 31.3, 31.6, 31.9, 32.0, 32.1, 32.2, 32.3, 32.4, 32.5, 32.6, 32.7, 32.
8, 32.9, 33.1, 33.2, 33.3, 33.5, 33.6, 33.7, 33.8, 33.9, 34.0, 34.1, 34.2, 34.3, 34.4, 34.5, 34.6, 34.7, 34.8, 34.9, 35.0, 35.1, 35.2, 35.3, 35.
4, 35.5, 35.6, 35.7, 35.8, 35.9, 36.0, 36.1, 36.2, 36.3, 36.4, 36.5, 36.6, 36.7, 36.8, 36.9, 37.0, 37.1, 37.2, 37.3, 37.4, 37.5, 37.6, 37.7, 37.
8, 37.9, 38.0, 38.1, 38.2, 38.3, 38.4, 38.5, 38.6, 38.7, 38.8, 38.9, 39.0, 39.1, 39.2, 39.3, 39.4, 39.5, 39.7, 39.8, 39.9, 40.0, 40.1, 40.2, 40.
5, 40.6, 40.7, 40.9, 41.0, 41.3, 41.5, 41.8, 42.0, 42.1, 42.2, 42.3, 42.4, 42.6, 42.7, 42.8, 42.9, 43.2, 43.3, 43.4, 43.5, 43.6, 44.0, 44.2, 45.
0, 45.2, 45.3, 45.4, 45.5, 45.6, 45.7, 45.8, 46.1, 46.2, 46.5, 46.7, 46.8, 47.9, 48.3, 48.8, 49.7, 50.0], Count : 229
Unique values in pedi is : [0.078, 0.084, 0.085, 0.088, 0.089, 0.092, 0.096, 0.1, 0.101, 0.102, 0.107, 0.108, 0.115, 0.121, 0.122, 0.123, 0.126,
0.127, 0.128, 0.129, 0.13, 0.133, 0.134, 0.135, 0.136, 0.137, 0.138, 0.14, 0.141, 0.142, 0.143, 0.144, 0.145, 0.147, 0.148, 0.149, 0.15, 0.151,
0.153, 0.154, 0.155, 0.156, 0.157, 0.158, 0.159, 0.16, 0.161, 0.162, 0.163, 0.164, 0.165, 0.166, 0.167, 0.17, 0.173, 0.174, 0.175, 0.178, 0.179,
0.18, 0.182, 0.183, 0.186, 0.187, 0.188, 0.189, 0.19, 0.191, 0.192, 0.194, 0.196, 0.197, 0.198, 0.199, 0.2, 0.201, 0.203, 0.204, 0.205, 0.206, 0.
207, 0.209, 0.21, 0.212, 0.215, 0.218, 0.219, 0.22, 0.221, 0.223, 0.225, 0.226, 0.227, 0.229, 0.23, 0.231, 0.232, 0.233, 0.234, 0.235, 0.236, 0.2
37, 0.238, 0.239, 0.24, 0.241, 0.243, 0.244, 0.245, 0.246, 0.247, 0.248, 0.249, 0.251, 0.252, 0.254, 0.255, 0.256, 0.257, 0.258, 0.259, 0.26, 0.2
61, 0.262, 0.263, 0.264, 0.267, 0.268, 0.269, 0.27, 0.271, 0.272, 0.277, 0.278, 0.279, 0.28, 0.282, 0.283, 0.284, 0.285, 0.286, 0.287, 0.289, 0.2
9, 0.292, 0.293, 0.294, 0.296, 0.299, 0.302, 0.303, 0.304, 0.305, 0.306, 0.307, 0.313, 0.314, 0.315, 0.318, 0.319, 0.323, 0.324, 0.325, 0.326, 0.
328, 0.329, 0.33, 0.331, 0.334, 0.336, 0.337, 0.338, 0.34, 0.341, 0.342, 0.343, 0.344, 0.345, 0.346, 0.347, 0.349, 0.351, 0.355, 0.356, 0.361, 0.
362, 0.364, 0.365, 0.368, 0.37, 0.371, 0.374, 0.375, 0.376, 0.378, 0.38, 0.381, 0.382, 0.383, 0.385, 0.388, 0.389, 0.391, 0.393, 0.394, 0.395, 0.
396, 0.398, 0.399, 0.4, 0.401, 0.402, 0.403, 0.404, 0.407, 0.409, 0.411, 0.412, 0.415, 0.416, 0.417, 0.419, 0.42, 0.421, 0.422, 0.423, 0.427, 0.4
3, 0.431, 0.432, 0.433, 0.434, 0.435, 0.443, 0.444, 0.446, 0.447, 0.451, 0.452, 0.454, 0.455, 0.457, 0.46, 0.463, 0.464, 0.465, 0.466, 0.467, 0.4
71, 0.472, 0.479, 0.482, 0.484, 0.485, 0.487, 0.491, 0.493, 0.495, 0.496, 0.497, 0.498, 0.499, 0.501, 0.502, 0.503, 0.507, 0.509, 0.51, 0.512, 0.
514, 0.516, 0.52, 0.525, 0.526, 0.527, 0.528, 0.529, 0.532, 0.534, 0.536, 0.537, 0.539, 0.542, 0.543, 0.545, 0.546, 0.547, 0.549, 0.551, 0.554,
0.557, 0.559, 0.561, 0.564, 0.569, 0.571, 0.572, 0.575, 0.578, 0.58, 0.582, 0.583, 0.586, 0.587, 0.588, 0.591, 0.593, 0.597, 0.6, 0.601, 0.605,
0.607, 0.61, 0.612, 0.613, 0.615, 0.619, 0.624, 0.626, 0.627, 0.629, 0.63, 0.637, 0.64, 0.645, 0.647, 0.649, 0.652, 0.654, 0.655, 0.658, 0.66, 0.
661, 0.665, 0.666, 0.672, 0.673, 0.674, 0.677, 0.678, 0.68, 0.682, 0.686, 0.687, 0.692, 0.693, 0.695, 0.696, 0.698, 0.699, 0.702, 0.703, 0.704,
0.705, 0.709, 0.711, 0.717, 0.718, 0.719, 0.721, 0.722, 0.725, 0.727, 0.73, 0.731, 0.732, 0.733, 0.735, 0.738, 0.742, 0.743, 0.744, 0.745, 0.748,
0.757, 0.759, 0.761, 0.767, 0.771, 0.773, 0.787, 0.801, 0.803, 0.804, 0.805, 0.808, 0.813, 0.816, 0.817, 0.821, 0.825, 0.831, 0.832, 0.833, 0.83
```

```
9, 0.84, 0.845, 0.851, 0.855, 0.867, 0.871, 0.874, 0.875, 0.878, 0.881, 0.892, 0.893, 0.905, 0.917, 0.925, 0.926, 0.93, 0.932, 0.933, 0.944, 0.94
7, 0.949, 0.955, 0.956, 0.962, 0.966, 0.968, 0.997, 1.001, 1.021, 1.022, 1.034, 1.072, 1.076, 1.095, 1.101, 1.114, 1.127, 1.136, 1.138, 1.144, 1.
154, 1.159, 1.162, 1.189, 1.191], Count : 441
Unique values in age is : [21.0, 22.0, 23.0, 24.0, 25.0, 26.0, 27.0, 28.0, 29.0, 30.0, 31.0, 32.0, 33.0, 34.0, 35.0, 36.0, 37.0, 38.0, 39.0, 40.
0, 41.0, 42.0, 43.0, 44.0, 45.0, 46.0, 47.0, 48.0, 49.0, 50.0, 51.0, 52.0, 53.0, 54.0, 55.0, 56.0, 57.0, 58.0, 59.0, 60.0, 61.0, 62.0, 63.0, 64.
0, 65.0, 66.0], Count : 46
Unique values in classification is : [0, 1], Count : 2
Numarical features : ['mass', 'pedi']
Caltegorical features : ['preg', 'plas', 'pres', 'skin', 'insu', 'age', 'classification']
```

In [25]:

```python
from imblearn.over_sampling import SMOTE

# split data into X, y
X, y = df.drop(['classification'], axis = 1),  df['classification']

# fit data in oversample
oversample = SMOTE()
X, y = oversample.fit_resample(X, y)

# concat data in df
df = pd.concat([X, y], axis=1)

# change data type
for col in caltegorical_feature:
    df[col] = df[col].astype(int)

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 826 entries, 0 to 825
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   preg            826 non-null    int32
 1   plas            826 non-null    int32
 2   pres            826 non-null    int32
 3   skin            826 non-null    int32
 4   insu            826 non-null    int32
 5   mass            826 non-null    float64
 6   pedi            826 non-null    float64
 7   age             826 non-null    int32
 8   classification  826 non-null    int32
dtypes: float64(2), int32(7)
memory usage: 35.6 KB
```

In [26]:

```
# Check If A Dataset Is Imbalanced
sns.countplot(x=df["classification"])
plt.title('Countplot for Classification')
plt.xlabel('Classification')
plt.ylabel('Count')
plt.show()
```

In [27]:

```
y = df['classification']

# summarize class distribution
counter = Counter(y)
print(f"Classification class distribution : {counter}")

# summarize class proportion
unique_labels, counts = np.unique(y, return_counts=True)
proportions = (counts / len(y))*100
print(f"Proportion of samples in each class (0,1) : {proportions}")
```

```
Classification class distribution : Counter({0: 413, 1: 413})
Proportion of samples in each class (0,1) : [50. 50.]
```

In [33]:

```
dataset = df
dataset.head()
```

Out[33]:

	preg	plas	pres	skin	insu	mass	pedi	age	classification
0	6	148	72	35	0	33.6	0.627	50	0
1	1	85	66	29	0	26.6	0.351	31	1
2	8	183	64	0	0	23.3	0.672	32	0
3	1	89	66	23	94	28.1	0.167	21	1
4	5	116	74	0	0	25.6	0.201	33	0

Classification

In [38]:

```
X, Y = dataset.drop(['classification'], axis = 1),  dataset['classification']
# train_test_split 80/20
X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size = 0.2, random_state = 42, stratify = Y)
```

In [39]:

```
# Model initialization
lr_Classifier = LogisticRegression()
knn_Classifier = KNeighborsClassifier()
gnb_Classifier = GaussianNB()
dt_Classifier = DecisionTreeClassifier()
rf_Classifier = RandomForestClassifier()
model_list = [lr_Classifier, knn_Classifier, gnb_Classifier, dt_Classifier, rf_Classifier]

# # Scaler initialization
# MinMax_scaler = MinMaxScaler()
# Standard_scaler = StandardScaler()
# MaxAbs_scaler = MaxAbsScaler()
# Robust_scaler = RobustScaler()
# Quantile_scaler = QuantileTransformer()
# Power_scaler = PowerTransformer()
# Normalizer_scaler = Normalizer()
# scaler_list = [MinMax_scaler, Standard_scaler, MaxAbs_scaler, Robust_scaler,
#                Quantile_scaler, Power_scaler, Normalizer_scaler]
```

In [40]:

```python
# def run_pipeline(X_train, X_test, y_train, y_test, scaler, classifier):
def run_pipeline(X_train, X_test, y_train, y_test, classifier):
    # Model Information
    print(f"Modele name : {type(classifier).__name__}")
    # print(f"Scaler name : {type(scaler).__name__}")

    # process 1 : fit and transform X_train data
    # scaled_X_train = scaler.fit_transform(X_train)

    # process 2 : train model
    # classifier.fit(scaled_X_train, y_train)
    classifier.fit(X_train, y_train)

    # process 3 : transform X_test data
    # scaled_X_test = scaler.transform(X_test)

    # process 4 : test model
    # y_pred = classifier.predict(scaled_X_test)
    y_pred = classifier.predict(X_test)
    # print(y_pred, le.inverse_transform(y_pred))

    # process 5 : Perform k-fold cross-validation using cross_val_score
    # scores = cross_val_score(classifier, scaled_X_train, y_train, cv=10, scoring='accuracy')
    scores = cross_val_score(classifier, X_train, y_train, cv=10, scoring='accuracy')
    print(f"10 K-Fold Accuracy_score : {np.round_(scores,4)}")
    print(f"10 K-Fold Average Accuracy_score : {round(np.average(scores)*100,2)} %")

    # process 6 : model evalution
    print("Accuracy_score:", round((accuracy_score(y_test, y_pred))*100,2),'%')
    print("Loss:", round((1-accuracy_score(y_test, y_pred))*100,2),'%')
    print("Cohen_kappa_score:", round((cohen_kappa_score(y_test, y_pred))*100,2),'%')
    print("Classification_report:\n",metrics.classification_report(y_test, y_pred))
    print("confusion_matrix:\n", confusion_matrix(y_test, y_pred))
    # plot confusion_matrix
    fig, ax = plt.subplots()
    fig.set_size_inches(6,4) # WH
    sns.heatmap(confusion_matrix(y_test, y_pred),
                annot=True,
                fmt=".1f",
                linewidths = 2,
                linecolor = "blue",
                center=0)
    plt.show()
    print("Root Mean Square Error (RMSE):", round(sqrt(mean_squared_error(y_test, y_pred)), 4))

    # process 7 : save model in pkl file
    # filename = f'Moduls\\{str(type(classifier).__name__)}_{str(type(scaler).__name__)}_01_LS_Disease_Prediction.pkl'
    filename = f'Moduls\\{str(type(classifier).__name__)}_01_LS_Disease_Prediction.pkl'
    pickle.dump(classifier, open(filename, 'wb'))
```

```python

    # collect data for bar plot
    global plot_data_list
    plot_data_list.append([str(type(classifier).__name__),
                           round((accuracy_score(y_test, y_pred))*100,2)])

    # collect data for bar plot (RSME)
    global plot_RSME_list
    plot_RSME_list.append([str(type(classifier).__name__),
                           round(sqrt(mean_squared_error(y_test, y_pred)), 4)])

    # end
    print("==="*30)
    print("\n\n")
    time.sleep(0.5)
```

In [41]:

```python
for model in model_list:
    # for scaler in scaler_list:
    run_pipeline(X_train, X_test, y_train, y_test, model)

# plot data
# plot_df = pd.DataFrame(plot_data_list, columns=['classifier', 'scaler', 'accuracy_score'])
plot_df = pd.DataFrame(plot_data_list, columns=['classifier', 'accuracy_score'])
plot_df.to_csv(f"Dataset\\accuracy_score_plot_data_01_LS_Disease_Prediction.csv", index=False)

sns.set(rc={'figure.figsize':(10,6)})
# ax = sns.barplot(data=plot_df, x="classifier", y="accuracy_score", hue="scaler")
ax = sns.barplot(data=plot_df, x="classifier", y="accuracy_score") # , hue="classifier"
plt.title('Accuracy Score Plot')
plt.xlabel('Classifier')
plt.ylabel('Accuracy Score')
ax.tick_params(axis='x', rotation=5)
for i in ax.containers:
    ax.bar_label(i,)
plt.show()


plot_RSME = pd.DataFrame(plot_RSME_list, columns=['classifier', 'RSME_score'])
plot_RSME.to_csv(f"Dataset\\RSME_score_plot_data_01_LS_Disease_Prediction.csv", index=False)

sns.set(rc={'figure.figsize':(10,6)})
# ax = sns.barplot(data=plot_df, x="classifier", y="accuracy_score", hue="scaler")
ax = sns.barplot(data=plot_RSME, x="classifier", y="RSME_score") # , hue="classifier"
plt.title('RSME Score Plot')
plt.xlabel('Classifier')
plt.ylabel('RSME Score')
ax.tick_params(axis='x', rotation=5)
for i in ax.containers:
    ax.bar_label(i,)
plt.show()


# empty list
plot_data_list = []
plot_RSME_list = []
print("\n\n")

print("Done...")
```

```
Modele name : LogisticRegression
10 K-Fold Accuracy_score : [0.5758 0.6364 0.5455 0.5455 0.5455 0.4848 0.5455 0.5303 0.5606 0.6212]
10 K-Fold Average Accuracy_score : 55.91 %
Accuracy_score: 46.99 %
Loss: 53.01 %
Cohen_kappa_score: -6.02 %
Classification_report:
              precision    recall  f1-score   support

           0       0.47      0.51      0.49        83
           1       0.47      0.43      0.45        83

    accuracy                           0.47       166
   macro avg       0.47      0.47      0.47       166
weighted avg       0.47      0.47      0.47       166

confusion_matrix:
 [[42 41]
 [47 36]]
```

```
Root Mean Square Error (RMSE): 0.7281
==========================================================================================



Modele name : KNeighborsClassifier
10 K-Fold Accuracy_score : [0.6364 0.5909 0.6515 0.5758 0.6515 0.6061 0.5455 0.5909 0.697  0.5606]
10 K-Fold Average Accuracy_score : 61.06 %
Accuracy_score: 69.88 %
Loss: 30.12 %
Cohen_kappa_score: 39.76 %
Classification_report:
               precision    recall  f1-score   support

           0       0.67      0.77      0.72        83
           1       0.73      0.63      0.68        83

    accuracy                           0.70       166
   macro avg       0.70      0.70      0.70       166
weighted avg       0.70      0.70      0.70       166

confusion_matrix:
 [[64 19]
 [31 52]]
```

```
Root Mean Square Error (RMSE): 0.5488
==========================================================================================


Modele name : GaussianNB
10 K-Fold Accuracy_score : [0.5606 0.5909 0.5455 0.5    0.5909 0.4697 0.5303 0.5758 0.5909 0.6212]
10 K-Fold Average Accuracy_score : 55.76 %
Accuracy_score: 49.4 %
Loss: 50.6 %
Cohen_kappa_score: -1.2 %
Classification_report:
               precision    recall  f1-score   support

           0       0.49      0.51      0.50        83
           1       0.49      0.48      0.49        83

    accuracy                           0.49       166
   macro avg       0.49      0.49      0.49       166
weighted avg       0.49      0.49      0.49       166

confusion_matrix:
 [[42 41]
 [43 40]]
```

```
Root Mean Square Error (RMSE): 0.7114
==========================================================================================


Modele name : DecisionTreeClassifier
10 K-Fold Accuracy_score : [0.6364 0.5758 0.5455 0.5152 0.5455 0.6818 0.5606 0.5606 0.6515 0.7121]
10 K-Fold Average Accuracy_score : 59.85 %
Accuracy_score: 57.23 %
Loss: 42.77 %
Cohen_kappa_score: 14.46 %
Classification_report:
               precision    recall  f1-score   support

           0       0.57      0.61      0.59        83
           1       0.58      0.53      0.55        83

    accuracy                           0.57       166
   macro avg       0.57      0.57      0.57       166
weighted avg       0.57      0.57      0.57       166

confusion_matrix:
 [[51 32]
 [39 44]]
```

```
Root Mean Square Error (RMSE): 0.654
==========================================================================================



Modele name : RandomForestClassifier
10 K-Fold Accuracy_score : [0.7121 0.6364 0.7424 0.6212 0.6515 0.6364 0.697  0.5455 0.6515 0.5909]
10 K-Fold Average Accuracy_score : 64.85 %
Accuracy_score: 66.27 %
Loss: 33.73 %
Cohen_kappa_score: 32.53 %
Classification_report:
               precision    recall  f1-score   support

           0       0.66      0.69      0.67        83
           1       0.67      0.64      0.65        83

    accuracy                           0.66       166
   macro avg       0.66      0.66      0.66       166
weighted avg       0.66      0.66      0.66       166

confusion_matrix:
 [[57 26]
 [30 53]]
```

```
Root Mean Square Error (RMSE): 0.5808
==========================================================================================
```

RSME Score Plot
0.7281
0.5488
0.7114
0.654
0.5808
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
RSME Score
LogisticRegression
KNeighborsClassifier
GaussianNB
DecisionTreeClassifier
RandomForestClassifier
Classifier

```
Done...
```

In [ ]:

```
1
```