v-milesstone-heart-test-2

September 23, 2024

1 MACHINE LEARNING

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import␣
↪mean_squared_error,r2_score,confusion_matrix,classification_report,accuracy_score
from sklearn.preprocessing import StandardScaler
```

1.1 Exploring the CAR Dataset with different (REGRESSION ALGORITHMS)

We have a data which is car or not according to features in it. We will try to use this data to create a model which tries predict types of car and price. We will use regression algorithms.

[2]:
```
df = pd.read_csv("C:/kgisl class/MILESTONE - 2/heart.csv")
df
```

[2]:
```
      age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
0      52    1   0       125   212    0        1      168      0      1.0
1      53    1   0       140   203    1        0      155      1      3.1
2      70    1   0       145   174    0        1      125      1      2.6
3      61    1   0       148   203    0        1      161      0      0.0
4      62    0   0       138   294    1        1      106      0      1.9
…     …    …  ..       …     …    …      …        …      …        …
1020   59    1   1       140   221    0        1      164      1      0.0
1021   60    1   0       125   258    0        0      141      1      2.8
1022   47    1   0       110   275    0        0      118      1      1.0
1023   50    0   0       110   254    0        0      159      0      0.0
1024   54    1   0       120   188    0        1      113      0      1.4

      slope  ca  thal  target
0         2   2     3       0
1         0   0     3       0
2         0   0     3       0
```

```
3          2   1     3        0
4          1   3     2        0
…      …  ..   …     …
1020       2   0     2        1
1021       1   1     3        0
1022       1   1     2        0
1023       2   0     2        1
1024       1   1     3        0

[1025 rows x 14 columns]
```

Exploring the dataset

[3]:
```python
df.shape
```

[3]:
```
(1025, 14)
```

[4]:
```python
df.dtypes
```

[4]:
```
age           int64
sex           int64
cp            int64
trestbps      int64
chol          int64
fbs           int64
restecg       int64
thalach       int64
exang         int64
oldpeak     float64
slope         int64
ca            int64
thal          int64
target        int64
dtype: object
```

[5]:
```python
df.head()
```

[5]:
```
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   52    1   0       125   212    0        1      168      0      1.0      2
1   53    1   0       140   203    1        0      155      1      3.1      0
2   70    1   0       145   174    0        1      125      1      2.6      0
3   61    1   0       148   203    0        1      161      0      0.0      2
4   62    0   0       138   294    1        1      106      0      1.9      1

   ca  thal  target
0   2     3       0
1   0     3       0
```

```
2    0      3        0
3    1      3        0
4    3      2        0
```

[6]:
```
df.tail()
```

[6]:
```
      age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
1020   59    1   1       140   221    0        1      164      1      0.0
1021   60    1   0       125   258    0        0      141      1      2.8
1022   47    1   0       110   275    0        0      118      1      1.0
1023   50    0   0       110   254    0        0      159      0      0.0
1024   54    1   0       120   188    0        1      113      0      1.4

      slope  ca  thal  target
1020      2   0     2       1
1021      1   1     3       0
1022      1   1     2       0
1023      2   0     2       1
1024      1   1     3       0
```

[7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1025 non-null   int64
 1   sex       1025 non-null   int64
 2   cp        1025 non-null   int64
 3   trestbps  1025 non-null   int64
 4   chol      1025 non-null   int64
 5   fbs       1025 non-null   int64
 6   restecg   1025 non-null   int64
 7   thalach   1025 non-null   int64
 8   exang     1025 non-null   int64
 9   oldpeak   1025 non-null   float64
 10  slope     1025 non-null   int64
 11  ca        1025 non-null   int64
 12  thal      1025 non-null   int64
 13  target    1025 non-null   int64
dtypes: float64(1), int64(13)
memory usage: 112.2 KB
```

[8]:
```
df.describe()
```

[8]:
```
                 age          sex           cp     trestbps         chol  \
count  1025.000000  1025.000000  1025.000000  1025.000000  1025.00000
mean     54.434146     0.695610     0.942439   131.611707   246.00000
std       9.072290     0.460373     1.029641    17.516718    51.59251
min      29.000000     0.000000     0.000000    94.000000   126.00000
25%      48.000000     0.000000     0.000000   120.000000   211.00000
50%      56.000000     1.000000     1.000000   130.000000   240.00000
75%      61.000000     1.000000     2.000000   140.000000   275.00000
max      77.000000     1.000000     3.000000   200.000000   564.00000

                 fbs      restecg      thalach        exang      oldpeak  \
count  1025.000000  1025.000000  1025.000000  1025.000000  1025.000000
mean      0.149268     0.529756   149.114146     0.336585     1.071512
std       0.356527     0.527878    23.005724     0.472772     1.175053
min       0.000000     0.000000    71.000000     0.000000     0.000000
25%       0.000000     0.000000   132.000000     0.000000     0.000000
50%       0.000000     1.000000   152.000000     0.000000     0.800000
75%       0.000000     1.000000   166.000000     1.000000     1.800000
max       1.000000     2.000000   202.000000     1.000000     6.200000

               slope           ca         thal       target
count  1025.000000  1025.000000  1025.000000  1025.000000
mean      1.385366     0.754146     2.323902     0.513171
std       0.617755     1.030798     0.620660     0.500070
min       0.000000     0.000000     0.000000     0.000000
25%       1.000000     0.000000     2.000000     0.000000
50%       1.000000     0.000000     2.000000     1.000000
75%       2.000000     1.000000     3.000000     1.000000
max       2.000000     4.000000     3.000000     1.000000
```

[9]:
```python
df.columns
```

[9]:
```
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')
```

[10]:
```python
df.isnull().sum()
```

[10]:
```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
```

```
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

1.1.1 Exploratory data analysis - (EDA)

[11]:
```python
df.corr()
```

[11]:

	age	sex	cp	trestbps	chol	fbs
age	1.000000	-0.103240	-0.071966	0.271121	0.219823	0.121243
sex	-0.103240	1.000000	-0.041119	-0.078974	-0.198258	0.027200
cp	-0.071966	-0.041119	1.000000	0.038177	-0.081641	0.079294
trestbps	0.271121	-0.078974	0.038177	1.000000	0.127977	0.181767
chol	0.219823	-0.198258	-0.081641	0.127977	1.000000	0.026917
fbs	0.121243	0.027200	0.079294	0.181767	0.026917	1.000000
restecg	-0.132696	-0.055117	0.043581	-0.123794	-0.147410	-0.104051
thalach	-0.390227	-0.049365	0.306839	-0.039264	-0.021772	-0.008866
exang	0.088163	0.139157	-0.401513	0.061197	0.067382	0.049261
oldpeak	0.208137	0.084687	-0.174733	0.187434	0.064880	0.010859
slope	-0.169105	-0.026666	0.131633	-0.120445	-0.014248	-0.061902
ca	0.271551	0.111729	-0.176206	0.104554	0.074259	0.137156
thal	0.072297	0.198424	-0.163341	0.059276	0.100244	-0.042177
target	-0.229324	-0.279501	0.434854	-0.138772	-0.099966	-0.041164

	restecg	thalach	exang	oldpeak	slope	ca
age	-0.132696	-0.390227	0.088163	0.208137	-0.169105	0.271551
sex	-0.055117	-0.049365	0.139157	0.084687	-0.026666	0.111729
cp	0.043581	0.306839	-0.401513	-0.174733	0.131633	-0.176206
trestbps	-0.123794	-0.039264	0.061197	0.187434	-0.120445	0.104554
chol	-0.147410	-0.021772	0.067382	0.064880	-0.014248	0.074259
fbs	-0.104051	-0.008866	0.049261	0.010859	-0.061902	0.137156
restecg	1.000000	0.048411	-0.065606	-0.050114	0.086086	-0.078072
thalach	0.048411	1.000000	-0.380281	-0.349796	0.395308	-0.207888
exang	-0.065606	-0.380281	1.000000	0.310844	-0.267335	0.107849
oldpeak	-0.050114	-0.349796	0.310844	1.000000	-0.575189	0.221816
slope	0.086086	0.395308	-0.267335	-0.575189	1.000000	-0.073440
ca	-0.078072	-0.207888	0.107849	0.221816	-0.073440	1.000000
thal	-0.020504	-0.098068	0.197201	0.202672	-0.094090	0.149014
target	0.134468	0.422895	-0.438029	-0.438441	0.345512	-0.382085

	thal	target
age	0.072297	-0.229324
sex	0.198424	-0.279501
cp	-0.163341	0.434854

```
trestbps  0.059276 -0.138772
chol      0.100244 -0.099966
fbs      -0.042177 -0.041164
restecg  -0.020504  0.134468
thalach  -0.098068  0.422895
exang     0.197201 -0.438029
oldpeak   0.202672 -0.438441
slope    -0.094090  0.345512
ca        0.149014 -0.382085
thal      1.000000 -0.337838
target   -0.337838  1.000000
```

[12]:
```
df['target'].unique()                                    # this is Numerical␣
↪coloumn  bcoz its, countable
```

[12]:
```
array([0, 1], dtype=int64)
```

[13]:
```
df['oldpeak'].unique()
```

[13]:
```
array([1. , 3.1, 2.6, 0. , 1.9, 4.4, 0.8, 3.2, 1.6, 3. , 0.7, 4.2, 1.5,
       2.2, 1.1, 0.3, 0.4, 0.6, 3.4, 2.8, 1.2, 2.9, 3.6, 1.4, 0.2, 2. ,
       5.6, 0.9, 1.8, 6.2, 4. , 2.5, 0.5, 0.1, 2.1, 2.4, 3.8, 2.3, 1.3,
       3.5])
```

[14]:
```
df['age'].unique()
```

[14]:
```
array([52, 53, 70, 61, 62, 58, 55, 46, 54, 71, 43, 34, 51, 50, 60, 67, 45,
       63, 42, 44, 56, 57, 59, 64, 65, 41, 66, 38, 49, 48, 29, 37, 47, 68,
       76, 40, 39, 77, 69, 35, 74], dtype=int64)
```

[15]:
```
df['sex'].unique()
```

[15]:
```
array([1, 0], dtype=int64)
```

[16]:
```
df['cp'].unique()
```

[16]:
```
array([0, 1, 2, 3], dtype=int64)
```

[17]:
```
df['trestbps'].unique()
```

[17]:
```
array([125, 140, 145, 148, 138, 100, 114, 160, 120, 122, 112, 132, 118,
       128, 124, 106, 104, 135, 130, 136, 180, 129, 150, 178, 146, 117,
       152, 154, 170, 134, 174, 144, 108, 123, 110, 142, 126, 192, 115,
        94, 200, 165, 102, 105, 155, 172, 164, 156, 101], dtype=int64)
```

[18]:
```
df['chol'].unique()
```

```
[18]: array([212, 203, 174, 294, 248, 318, 289, 249, 286, 149, 341, 210, 298,
       204, 308, 266, 244, 211, 185, 223, 208, 252, 209, 307, 233, 319,
       256, 327, 169, 131, 269, 196, 231, 213, 271, 263, 229, 360, 258,
       330, 342, 226, 228, 278, 230, 283, 241, 175, 188, 217, 193, 245,
       232, 299, 288, 197, 315, 215, 164, 326, 207, 177, 257, 255, 187,
       201, 220, 268, 267, 236, 303, 282, 126, 309, 186, 275, 281, 206,
       335, 218, 254, 295, 417, 260, 240, 302, 192, 225, 325, 235, 274,
       234, 182, 167, 172, 321, 300, 199, 564, 157, 304, 222, 184, 354,
       160, 247, 239, 246, 409, 293, 180, 250, 221, 200, 227, 243, 311,
       261, 242, 205, 306, 219, 353, 198, 394, 183, 237, 224, 265, 313,
       340, 259, 270, 216, 264, 276, 322, 214, 273, 253, 176, 284, 305,
       168, 407, 290, 277, 262, 195, 166, 178, 141], dtype=int64)
```

```
[19]: df['fbs'].unique()
```

```
[19]: array([0, 1], dtype=int64)
```

```
[20]: df['restecg'].unique()
```

```
[20]: array([1, 0, 2], dtype=int64)
```

```
[21]: df['thalach'].unique()
```

```
[21]: array([168, 155, 125, 161, 106, 122, 140, 145, 144, 116, 136, 192, 156,
       142, 109, 162, 165, 148, 172, 173, 146, 179, 152, 117, 115, 112,
       163, 147, 182, 105, 150, 151, 169, 166, 178, 132, 160, 123, 139,
       111, 180, 164, 202, 157, 159, 170, 138, 175, 158, 126, 143, 141,
       167,  95, 190, 118, 103, 181, 108, 177, 134, 120, 171, 149, 154,
       153,  88, 174, 114, 195, 133,  96, 124, 131, 185, 194, 128, 127,
       186, 184, 188, 130,  71, 137,  99, 121, 187,  97,  90, 129, 113],
      dtype=int64)
```

```
[22]: df['exang'].unique()
```

```
[22]: array([0, 1], dtype=int64)
```

```
[23]: df['slope'].unique()
```

```
[23]: array([2, 0, 1], dtype=int64)
```

```
[24]: df['ca'].unique()
```

```
[24]: array([2, 0, 1, 3, 4], dtype=int64)
```

```
[25]: df['thal'].unique()
```

```
[25]: array([3, 2, 1, 0], dtype=int64)
```

[26]:
```python
numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
newdf = df.select_dtypes(include=numerics)
newdf
```

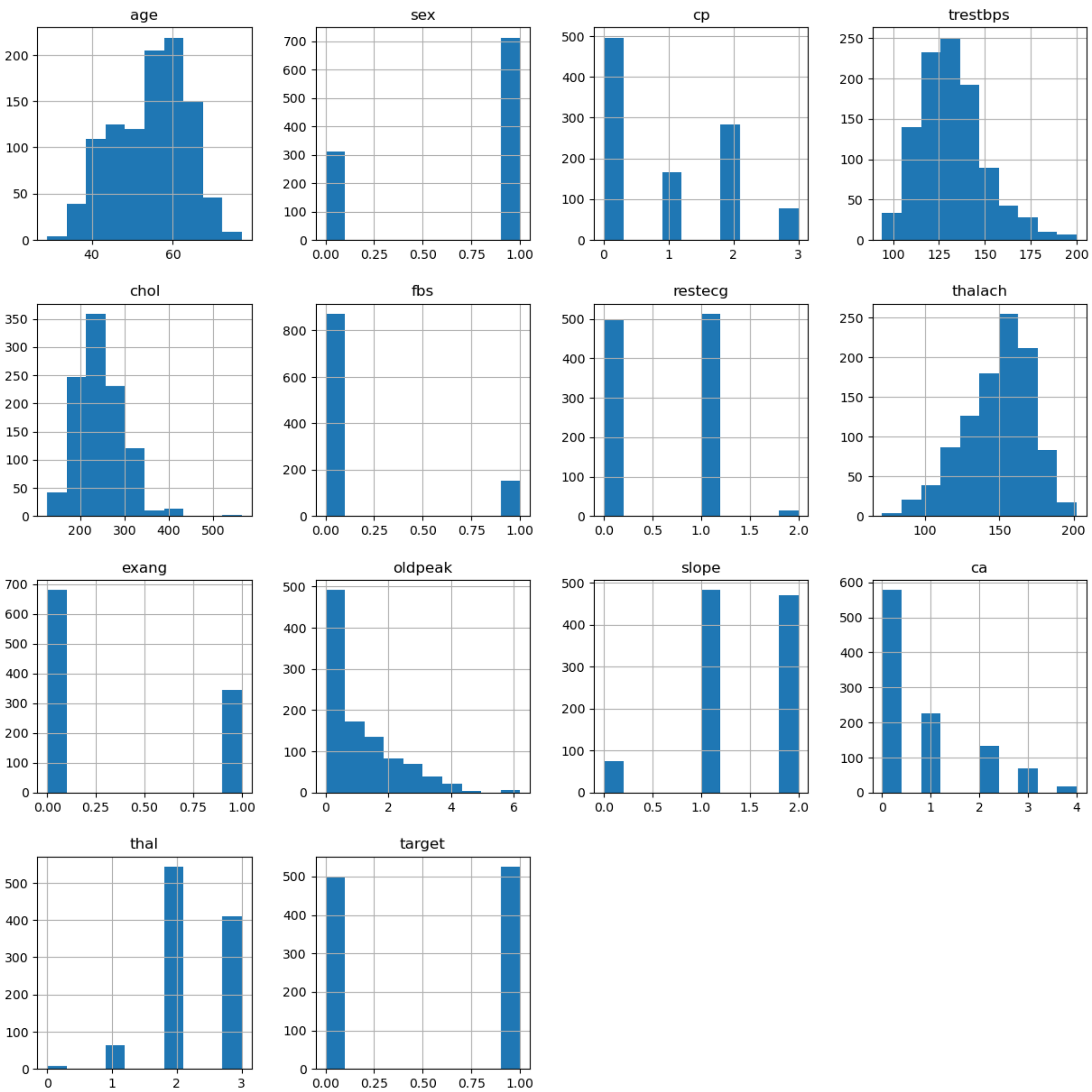
[26]:
```
      age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
0      52    1   0       125   212    0        1      168      0      1.0
1      53    1   0       140   203    1        0      155      1      3.1
2      70    1   0       145   174    0        1      125      1      2.6
3      61    1   0       148   203    0        1      161      0      0.0
4      62    0   0       138   294    1        1      106      0      1.9
…    …    …  ..     …   …  …      …      …    …      …
1020   59    1   1       140   221    0        1      164      1      0.0
1021   60    1   0       125   258    0        0      141      1      2.8
1022   47    1   0       110   275    0        0      118      1      1.0
1023   50    0   0       110   254    0        0      159      0      0.0
1024   54    1   0       120   188    0        1      113      0      1.4

      slope  ca  thal  target
0         2   2     3       0
1         0   0     3       0
2         0   0     3       0
3         2   1     3       0
4         1   3     2       0
…      …  ..   …     …
1020      2   0     2       1
1021      1   1     3       0
1022      1   1     2       0
1023      2   0     2       1
1024      1   1     3       0

[1025 rows x 14 columns]
```

1.1.2 Data Visualization

[27]:
```python
# Importing essential libraries
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

[28]:
```python
fig = plt.figure(figsize = (15,15))
ax = fig.gca()
g = df.hist(ax=ax)
```

```
C:\Users\DELL\AppData\Local\Temp\ipykernel_10700\3980286831.py:3: UserWarning:
To output multiple subplots, the figure containing the passed axes is being
cleared.
  g = df.hist(ax=ax)
```



find=iqr #using quantile for removing and detectioning those outlayers

[29]:
```
r=df.select_dtypes(exclude=['object'])
r
```

[29]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	\
0	52	1	0	125	212	0	1	168	0	1.0	
1	53	1	0	140	203	1	0	155	1	3.1	
2	70	1	0	145	174	0	1	125	1	2.6	
3	61	1	0	148	203	0	1	161	0	0.0	
4	62	0	0	138	294	1	1	106	0	1.9	
…	…	…	..	…	…	…	…	…	…	…	
1020	59	1	1	140	221	0	1	164	1	0.0	

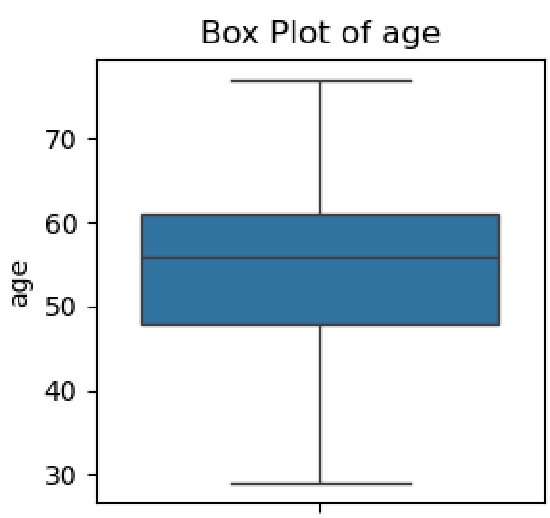
1021	60	1	0	125	258	0	0	141	1	2.8
1022	47	1	0	110	275	0	0	118	1	1.0
1023	50	0	0	110	254	0	0	159	0	0.0
1024	54	1	0	120	188	0	1	113	0	1.4

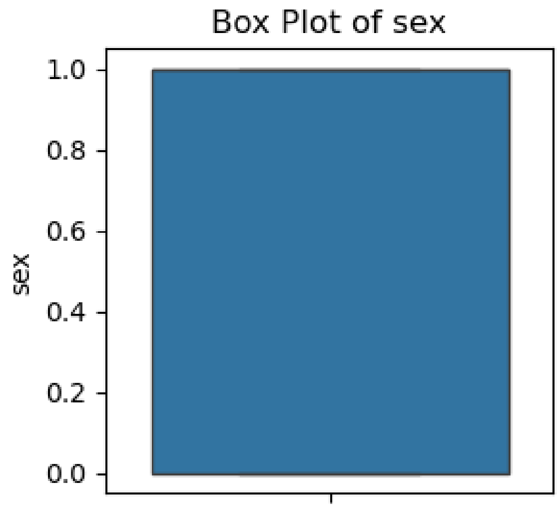
	slope	ca	thal	target
0	2	2	3	0
1	0	0	3	0
2	0	0	3	0
3	2	1	3	0
4	1	3	2	0
…	… ..	…	…	
1020	2	0	2	1
1021	1	1	3	0
1022	1	1	2	0
1023	2	0	2	1
1024	1	1	3	0

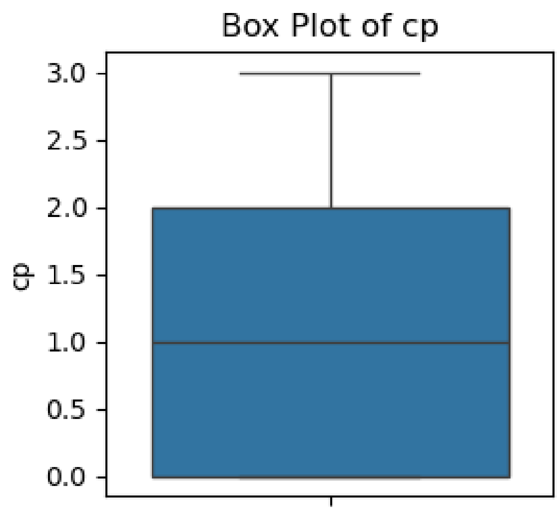
[1025 rows x 14 columns]

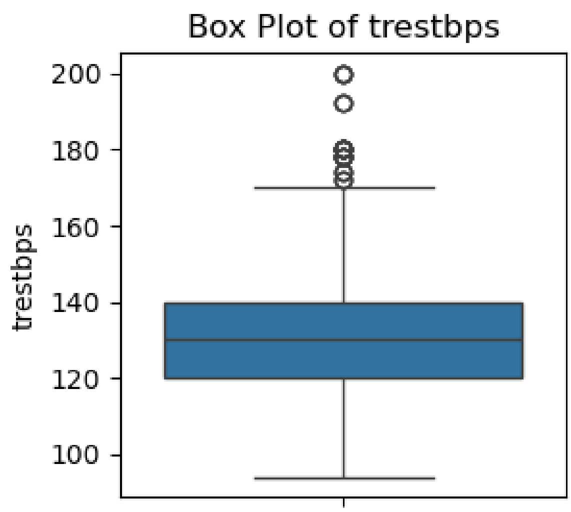
outlayer detection

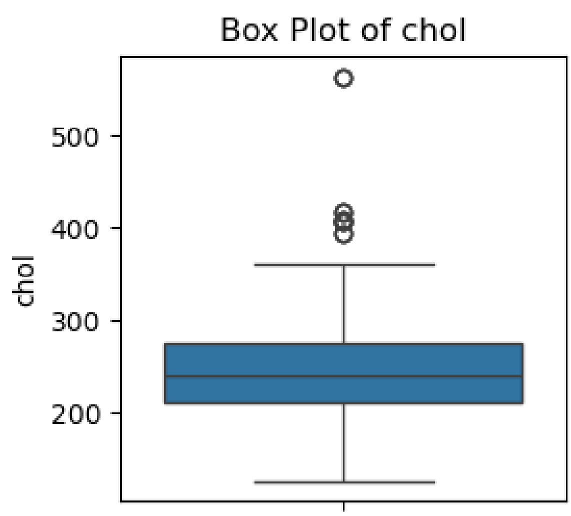
[30]:
```python
import seaborn as sns
#outlier detection
for col in df:
    plt.figure(figsize=(3, 3))
    sns.boxplot(y=df[col])
    plt.title(f'Box Plot of {col}')
    plt.show()
```

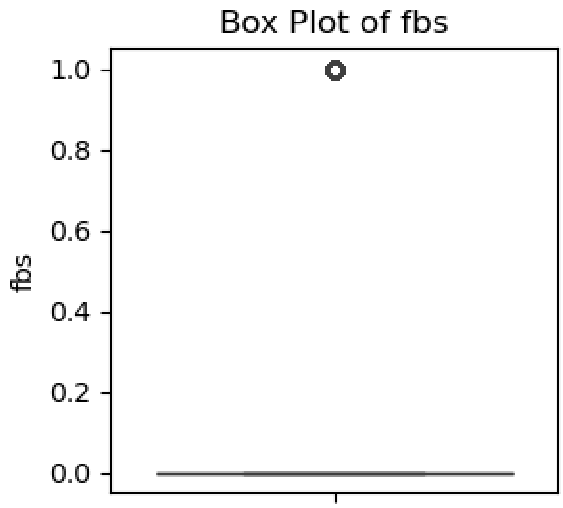


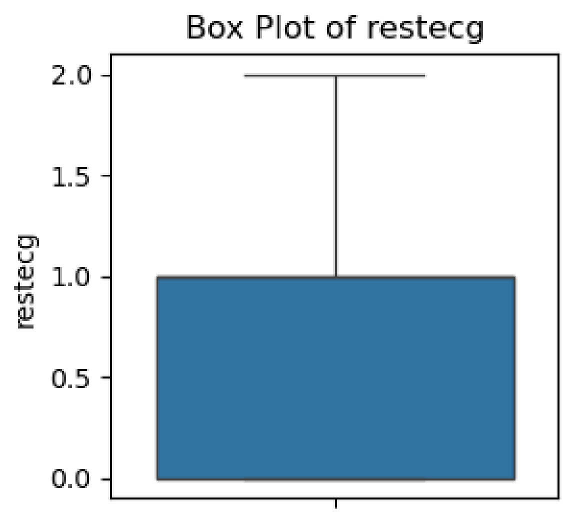

Box Plot of sex
1.0
0.8
0.6
0.4
0.2
0.0
sex

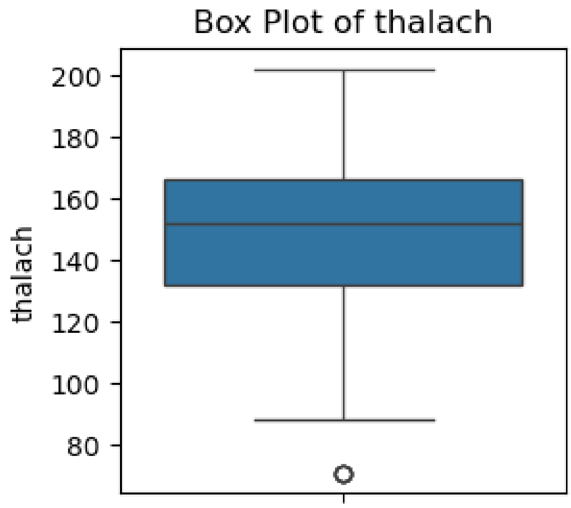

Box Plot of cp
3.0
2.5
2.0
1.5
1.0
0.5
0.0
cp

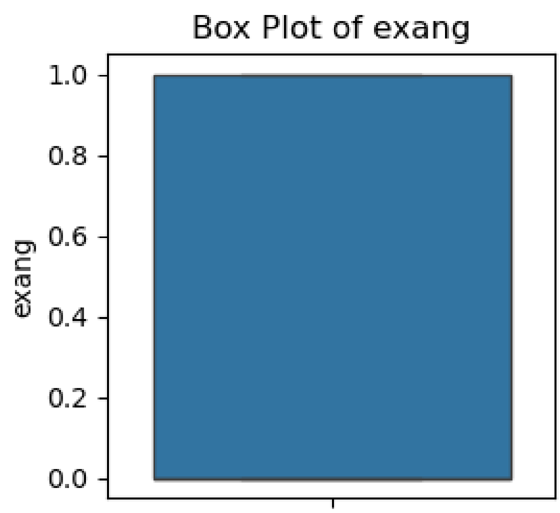

Box Plot of trestbps
trestbps
200
180
160
140
120
100

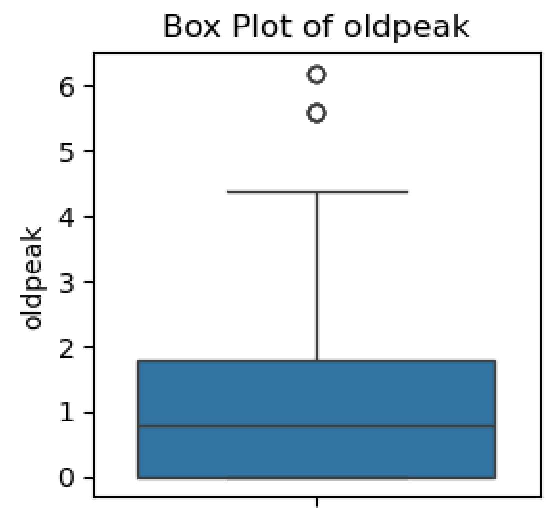

Box Plot of chol
chol
500
400
300
200

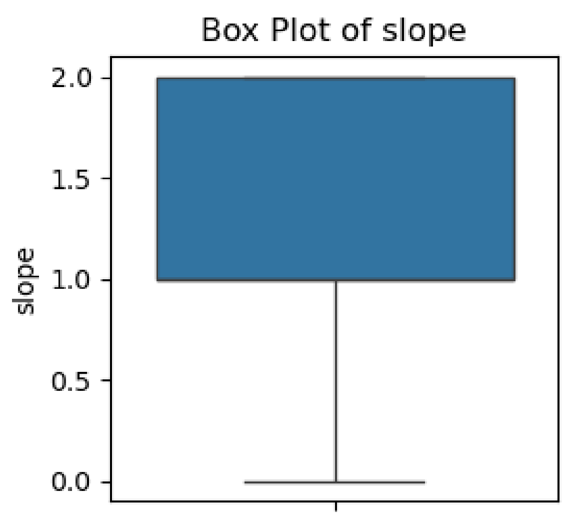

Box Plot of fbs
1.0
0.8
0.6
0.4
0.2
0.0
fbs

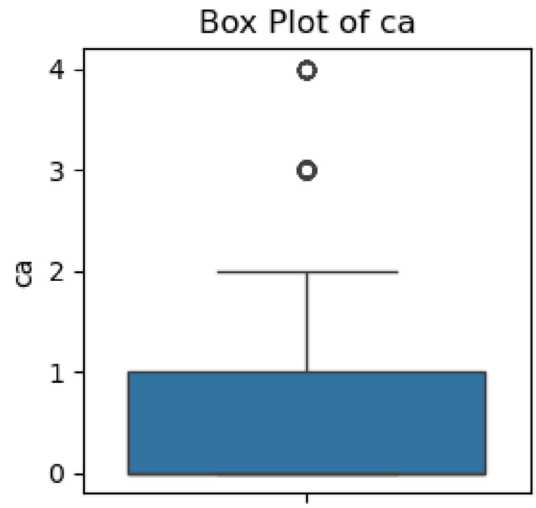

Box Plot of restecg
2.0
1.5
1.0
0.5
0.0
restecg

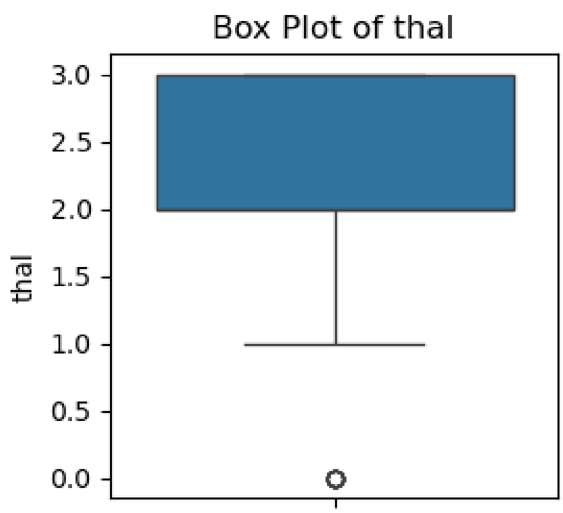

Box Plot of thalach
thalach
200
180
160
140
120
100
80

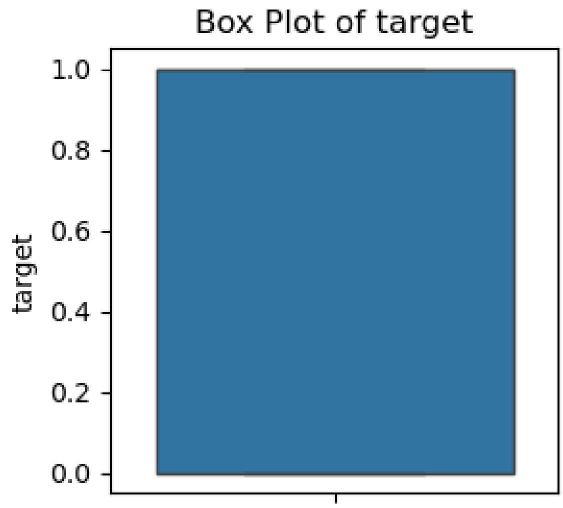

Box Plot of exang
exang
1.0
0.8
0.6
0.4
0.2
0.0


Box Plot of oldpeak
oldpeak
0
1
2
3
4
5
6


Box Plot of slope
slope
0.0
0.5
1.0
1.5
2.0


Box Plot of ca
ca
4
3
2
1
0


Box Plot of thal
thal
3.0
2.5
2.0
1.5
1.0
0.5
0.0



[31]:
```
q1=r.quantile(0.25)                                                    # outlayer detection
q1
```

[31]:
```
age          48.0
sex           0.0
cp            0.0
trestbps    120.0
chol        211.0
fbs           0.0
restecg       0.0
thalach     132.0
exang         0.0
oldpeak       0.0
slope         1.0
ca            0.0
thal          2.0
target        0.0
Name: 0.25, dtype: float64
```

[32]:
```
q3=r.quantile(0.75)
q3
```

[32]:
```
age          61.0
sex           1.0
cp            2.0
trestbps    140.0
chol        275.0
fbs           0.0
restecg       1.0
```

```
thalach     166.0
exang         1.0
oldpeak       1.8
slope         2.0
ca            1.0
thal          3.0
target        1.0
Name: 0.75, dtype: float64
```

[33]:
```
IQR=q3-q1
IQR
```

[33]:
```
age         13.0
sex          1.0
cp           2.0
trestbps    20.0
chol        64.0
fbs          0.0
restecg      1.0
thalach     34.0
exang        1.0
oldpeak      1.8
slope        1.0
ca           1.0
thal         1.0
target       1.0
dtype: float64
```

[34]:
```
a=((r<q1-1.5*IQR)|(r>q1+1.5*IQR))
a
```

[34]:
```
        age    sex     cp  trestbps   chol    fbs  restecg  thalach  exang  \
0     False  False  False     False  False  False    False    False  False
1     False  False  False     False  False   True    False    False  False
2      True  False  False     False  False  False    False    False  False
3     False  False  False     False  False  False    False    False  False
4     False  False  False     False  False   True    False    False  False
…       …    …    …       …    …    …      …      …    …
1020  False  False  False     False  False  False    False    False  False
1021  False  False  False     False  False  False    False    False  False
1022  False  False  False     False  False  False    False    False  False
1023  False  False  False     False  False  False    False    False  False
1024  False  False  False     False  False  False    False    False  False

      oldpeak  slope     ca   thal  target
0       False  False   True  False   False
1        True  False  False  False   False
```

```
2       False  False  False  False   False
3       False  False  False  False   False
4       False  False   True  False   False
…       …    …    …    …    …
1020    False  False  False  False   False
1021     True  False  False  False   False
1022    False  False  False  False   False
1023    False  False  False  False   False
1024    False  False  False  False   False

[1025 rows x 14 columns]
```

[35]:
```
df1=df[~((r<q1-1.5*IQR)|(r>q1+1.5*IQR)).any(axis=1)]
df1
```

[35]:
```
      age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
3      61    1   0       148   203    0        1      161      0      0.0
5      58    0   0       100   248    0        0      122      0      1.0
8      46    1   0       120   249    0        0      144      0      0.8
17     54    1   0       124   266    0        0      109      1      2.2
18     50    0   1       120   244    0        1      162      0      1.1
…     …   …  ..       …   …   …      …      …       …
1019   47    1   0       112   204    0        1      143      0      0.1
1020   59    1   1       140   221    0        1      164      1      0.0
1022   47    1   0       110   275    0        0      118      1      1.0
1023   50    0   0       110   254    0        0      159      0      0.0
1024   54    1   0       120   188    0        1      113      0      1.4

      slope  ca  thal  target
3         2   1     3       0
5         1   0     2       1
8         2   0     3       0
17        1   1     3       0
18        2   0     2       1
…     …  ..   …     …
1019      2   0     2       1
1020      2   0     2       1
1022      1   1     2       0
1023      2   0     2       1
1024      1   1     3       0

[474 rows x 14 columns]
```

[36]:
```
print(f"Original data shape: {df.shape}")
print(f"Cleaned data shape: {df1.shape}")
```

```
Original data shape: (1025, 14)
```

```
Cleaned data shape: (474, 14)
```

2 univarient analysis

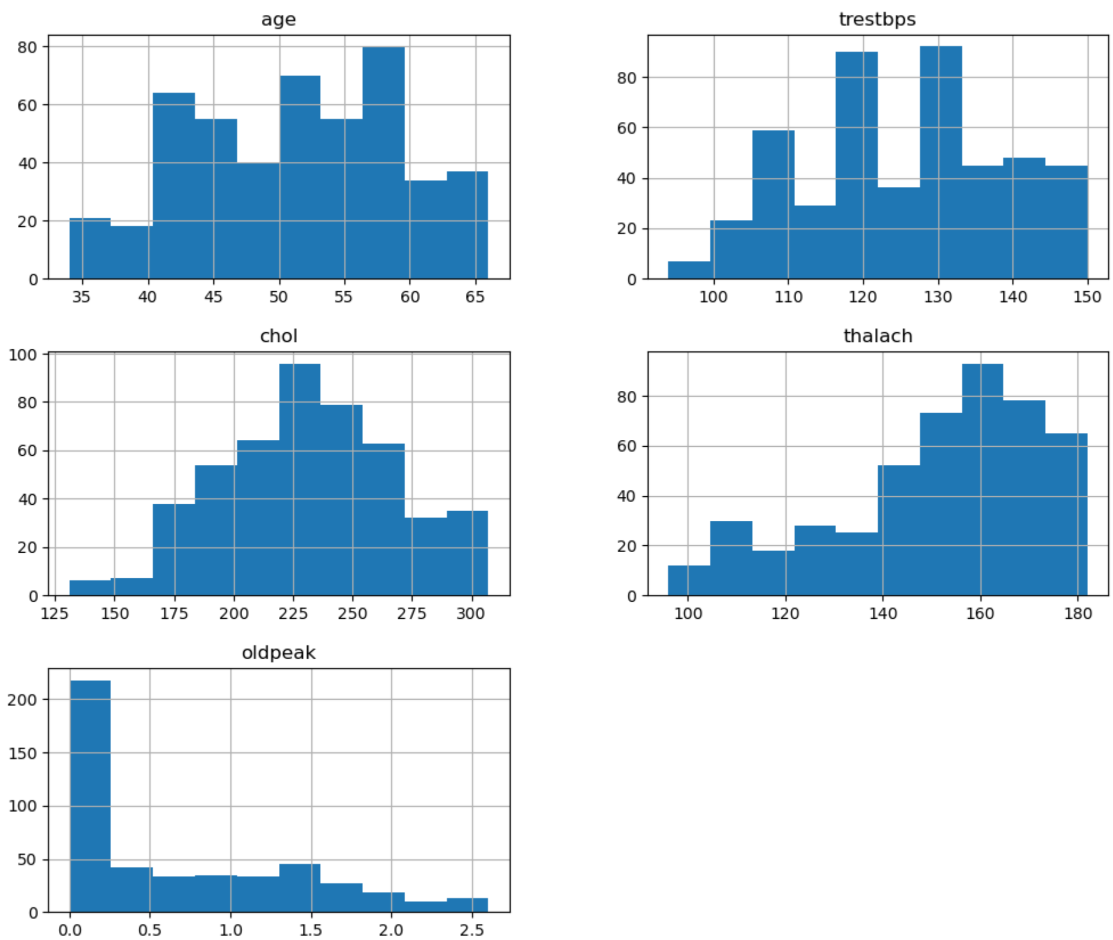
[37]:
```
df1.groupby(['target']).count()
```

[37]:
```
        age  sex   cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
target
0       173  173  173       173   173  173      173      173    173      173
1       301  301  301       301   301  301      301      301    301      301

        slope   ca  thal
target
0         173  173   173
1         301  301   301
```

[38]:
```
a=df1.groupby(['target']).size().reset_index(name='count')
a
```

[38]:
```
   target  count
0       0    173
1       1    301
```

[39]:
```
df1[['age', 'trestbps', 'chol', 'thalach', 'oldpeak']].hist(figsize=(12, 10))
plt.show()
```



[40]:
```
x=filter['target']
plt.hist(x,bins=2,color="skyblue")
plt.title("Histogram --- clarity  of target")
plt.xlabel("target")                                          #  Histogram
plt.ylabel("count")
plt.show()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[40], line 1
----> 1 x=filter['target']
      2 plt.hist(x,bins=2,color="skyblue")
      3 plt.title("Histogram --- clarity  of target")

TypeError: type 'filter' is not subscriptable
```

3 Bivariate Analysis

[ ]:
```
p=sns.pairplot(df1)
```

[ ]:
```
plt.figure(figsize=(10,10))
sns.heatmap(df1.corr(),annot=True,cmap='coolwarm')
plt.title('Correlation Matrix Heatmap')
plt.show()
```

4 M.L Model building

[41]:
```
x=df.iloc[:,:-1]
x
```

[41]:
```
      age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
0      52    1   0       125   212    0        1      168      0      1.0
1      53    1   0       140   203    1        0      155      1      3.1
2      70    1   0       145   174    0        1      125      1      2.6
3      61    1   0       148   203    0        1      161      0      0.0
4      62    0   0       138   294    1        1      106      0      1.9
…     …    …   ..        …    …   …        …       …      …
1020   59    1   1       140   221    0        1      164      1      0.0
1021   60    1   0       125   258    0        0      141      1      2.8
1022   47    1   0       110   275    0        0      118      1      1.0
1023   50    0   0       110   254    0        0      159      0      0.0
1024   54    1   0       120   188    0        1      113      0      1.4

      slope  ca  thal
0         2   2     3
1         0   0     3
2         0   0     3
3         2   1     3
4         1   3     2
…       …  ..    …
1020      2   0     2
1021      1   1     3
1022      1   1     2
1023      2   0     2
1024      1   1     3

[1025 rows x 13 columns]
```

[42]:
```
y=df.iloc[:,-1]
y
```

[42]:
```
0       0
1       0
2       0
3       0
4       0
       ..
1020    1
1021    0
1022    0
1023    1
1024    0
Name: target, Length: 1025, dtype: int64
```

[43]:
```python
from sklearn.model_selection import train_test_split
```

[44]:
```python
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.
 ↪2,random_state=42)
```

[45]:
```python
x_train.shape
```

[45]:
```
(820, 13)
```

[46]:
```python
x_train
```

[46]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	\
835	49	1	2	118	149	0	0	126	0	0.8	
137	64	0	0	180	325	0	1	154	1	0.0	
534	54	0	2	108	267	0	0	167	0	0.0	
495	59	1	0	135	234	0	1	161	0	0.5	
244	51	1	2	125	245	1	0	166	0	2.4	
..	…	…	..	…	…	…	…	…	…	…	
700	41	1	2	130	214	0	0	168	0	2.0	
71	61	1	0	140	207	0	0	138	1	1.9	
106	51	1	0	140	299	0	1	173	1	1.6	
270	43	1	0	110	211	0	1	161	0	0.0	
860	52	1	0	112	230	0	1	160	0	0.0	

	slope	ca	thal
835	2	3	2
137	2	0	2
534	2	0	2
495	1	0	3
244	1	0	2
..	…	..	…
700	1	0	2
71	2	1	3
106	2	0	3

```
270      2   0     3
860      2   1     2

[820 rows x 13 columns]
```

[47]:
```
x_test.shape
```

[47]:
```
(205, 13)
```

[48]:
```
x_test
```

[48]:
```
     age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
527   62    0   0       124   209    0        1      163      0      0.0
359   53    0   2       128   216    0        0      115      0      0.0
447   55    1   0       160   289    0        0      145      1      0.8
31    50    0   1       120   244    0        1      162      0      1.1
621   48    1   0       130   256    1        0      150      1      0.0
..   …   …  ..       …   …   …       …       …      …        …
832   68    1   2       118   277    0        1      151      0      1.0
796   41    1   1       135   203    0        1      132      0      0.0
644   44    1   2       120   226    0        1      169      0      0.0
404   61    1   0       140   207    0        0      138      1      1.9
842   58    1   2       112   230    0        0      165      0      2.5

     slope  ca  thal
527      2   0     2
359      2   0     0
447      1   1     3
31       2   0     2
621      2   2     3
..     …  ..    …
832      2   1     3
796      1   0     1
644      2   0     2
404      2   1     3
842      1   1     3

[205 rows x 13 columns]
```

[49]:
```
y_train.shape
```

[49]:
```
(820,)
```

[50]:
```
y_train
```

[50]:
```
835    0
137    1
```

```
534    1
495    1
244    1
      ..
700    1
71     0
106    0
270    1
860    0
Name: target, Length: 820, dtype: int64
```

[51]:
```
y_test.shape
```

[51]:
```
(205,)
```

[52]:
```
y_test
```

[52]:
```
527    1
359    1
447    0
31     1
621    0
      ..
832    1
796    1
644    1
404    0
842    0
Name: target, Length: 205, dtype: int64
```

[ ]:
```

```

5 LogisticRegression

[53]:
```
from sklearn.model_selection import train_test_split

# Define your feature columns and target column
x = df.drop('target', axis=1)
y = df['target']

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,␣
↪random_state=42)
```

[54]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
```

```
# Initialize and train the model
logistic_model = LogisticRegression()
logistic_model.fit(x_train, y_train)

# Make predictions
y_pred_logistic = logistic_model.predict(x_test)

# Evaluate the model
accuracy_logistic = accuracy_score(y_test, y_pred_logistic)
report_logistic = classification_report(y_test, y_pred_logistic)

print("Logistic Regression Accuracy:", accuracy_logistic)
print("Logistic Regression Classification Report:\n", report_logistic)
```

```
Logistic Regression Accuracy: 0.7853658536585366
Logistic Regression Classification Report:
               precision    recall  f1-score   support

           0       0.85      0.70      0.76       102
           1       0.74      0.87      0.80       103

    accuracy                           0.79       205
   macro avg       0.79      0.78      0.78       205
weighted avg       0.79      0.79      0.78       205


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

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

5.1 WITH HP

[55]:
```
from sklearn.model_selection import GridSearchCV

# Define parameter grid
param_grid_logistic = {
    'C': [0.01, 0.1, 1, 10, 100],
    'penalty': ['l1', 'l2']
}
```

```python
# Initialize and train the model with GridSearch
grid_search_logistic = GridSearchCV(LogisticRegression(max_iter=1000),␣
↪param_grid_logistic, cv=5, n_jobs=-1)
grid_search_logistic.fit(x_train, y_train)

# Best parameters and best score
print("Best Parameters for Logistic Regression:", grid_search_logistic.
↪best_params_)
print("Best Score for Logistic Regression:", grid_search_logistic.best_score_)

# Make predictions
y_pred_logistic_tuned = grid_search_logistic.predict(x_test)

# Evaluate the model
accuracy_logistic_tuned = accuracy_score(y_test, y_pred_logistic_tuned)
report_logistic_tuned = classification_report(y_test, y_pred_logistic_tuned)

print("Logistic Regression Accuracy (With Tuning):", accuracy_logistic_tuned)
print("Logistic Regression Classification Report (With Tuning):\n",␣
↪report_logistic_tuned)
```

```
C:\Users\DELL\anaconda3\Lib\site-
packages\sklearn\model_selection\_validation.py:378: FitFailedWarning:
25 fits failed out of a total of 50.
The score on these train-test partitions for these parameters will be set to
nan.
If these failures are not expected, you can try to debug them by setting
error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
25 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\DELL\anaconda3\Lib\site-
packages\sklearn\model_selection\_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\DELL\anaconda3\Lib\site-
packages\sklearn\linear_model\_logistic.py", line 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
             ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\DELL\anaconda3\Lib\site-
packages\sklearn\linear_model\_logistic.py", line 54, in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 penalty.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
C:\Users\DELL\anaconda3\Lib\site-
```

```
packages\sklearn\model_selection\_search.py:952: UserWarning: One or more of the
test scores are non-finite: [       nan 0.81585366        nan 0.84512195
nan 0.85
        nan 0.84878049        nan 0.84878049]
  warnings.warn(
```

```
Best Parameters for Logistic Regression: {'C': 1, 'penalty': 'l2'}
Best Score for Logistic Regression: 0.85
Logistic Regression Accuracy (With Tuning): 0.7951219512195122
Logistic Regression Classification Report (With Tuning):
               precision    recall  f1-score   support

           0       0.85      0.72      0.78       102
           1       0.76      0.87      0.81       103

    accuracy                           0.80       205
   macro avg       0.80      0.79      0.79       205
weighted avg       0.80      0.80      0.79       205
```

[ ]:

6 RandomForestClassifier

[56]:
```python
from sklearn.ensemble import RandomForestClassifier

# Initialize and train the model
rf_model = RandomForestClassifier()
rf_model.fit(x_train, y_train)

# Make predictions
y_pred_rf = rf_model.predict(x_test)

# Evaluate the model
accuracy_rf = accuracy_score(y_test, y_pred_rf)
report_rf = classification_report(y_test, y_pred_rf)

print("Random Forest Accuracy:", accuracy_rf)
print("Random Forest Classification Report:\n", report_rf)
```

```
Random Forest Accuracy: 0.9853658536585366
Random Forest Classification Report:
               precision    recall  f1-score   support

           0       0.97      1.00      0.99       102
           1       1.00      0.97      0.99       103
```

```
    accuracy                           0.99       205
   macro avg       0.99      0.99      0.99       205
weighted avg       0.99      0.99      0.99       205
```

6.1 WITH HP

[57]:
```
from sklearn.model_selection import GridSearchCV

# Define parameter grid
param_grid_rf = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}

# Initialize and train the model with GridSearch
grid_search_rf = GridSearchCV(RandomForestClassifier(), param_grid_rf, cv=5,␣
 ↪n_jobs=-1)
grid_search_rf.fit(x_train, y_train)

# Best parameters and best score
print("Best Parameters for Random Forest:", grid_search_rf.best_params_)
print("Best Score for Random Forest:", grid_search_rf.best_score_)

# Make predictions
y_pred_rf_tuned = grid_search_rf.predict(x_test)

# Evaluate the model
accuracy_rf_tuned = accuracy_score(y_test, y_pred_rf_tuned)
report_rf_tuned = classification_report(y_test, y_pred_rf_tuned)

print("Random Forest Accuracy (With Tuning):", accuracy_rf_tuned)
print("Random Forest Classification Report (With Tuning):\n", report_rf_tuned)
```

```
Best Parameters for Random Forest: {'max_depth': 10, 'min_samples_leaf': 1,
'min_samples_split': 2, 'n_estimators': 100}
Best Score for Random Forest: 0.9829268292682928
Random Forest Accuracy (With Tuning): 0.9853658536585366
Random Forest Classification Report (With Tuning):
               precision    recall  f1-score   support

           0       0.97      1.00      0.99       102
           1       1.00      0.97      0.99       103

    accuracy                           0.99       205
   macro avg       0.99      0.99      0.99       205
```

```
weighted avg       0.99      0.99      0.99       205
```

[ ]:

7 SVC Classifier

[58]:
```python
from sklearn.svm import SVC

# Initialize and train the model
svm_model = SVC()
svm_model.fit(x_train, y_train)

# Make predictions
y_pred_svm = svm_model.predict(x_test)

# Evaluate the model
accuracy_svm = accuracy_score(y_test, y_pred_svm)
report_svm = classification_report(y_test, y_pred_svm)

print("SVM Accuracy:", accuracy_svm)
print("SVM Classification Report:\n", report_svm)
```

```
SVM Accuracy: 0.6829268292682927
SVM Classification Report:
               precision    recall  f1-score   support

           0       0.71      0.61      0.66       102
           1       0.66      0.76      0.71       103

    accuracy                           0.68       205
   macro avg       0.69      0.68      0.68       205
weighted avg       0.69      0.68      0.68       205
```

7.1 WITH HP

[59]:
```python
from sklearn.model_selection import GridSearchCV

# Define parameter grid
param_grid_svm = {
    'C': [0.1, 1, 10],
    'kernel': ['linear'],

}

# Initialize and train the model with GridSearch
```

```
grid_search_svm = GridSearchCV(SVC(), param_grid_svm, cv=5, n_jobs=-1)
grid_search_svm.fit(x_train, y_train)

# Best parameters and best score
print("Best Parameters for SVM:", grid_search_svm.best_params_)
print("Best Score for SVM:", grid_search_svm.best_score_)

# Make predictions
y_pred_svm_tuned = grid_search_svm.predict(x_test)

# Evaluate the model
accuracy_svm_tuned = accuracy_score(y_test, y_pred_svm_tuned)
report_svm_tuned = classification_report(y_test, y_pred_svm_tuned)

print("SVM Accuracy (With Tuning):", accuracy_svm_tuned)
print("SVM Classification Report (With Tuning):\n", report_svm_tuned)
```

```
Best Parameters for SVM: {'C': 0.1, 'kernel': 'linear'}
Best Score for SVM: 0.8512195121951219
SVM Accuracy (With Tuning): 0.7951219512195122
SVM Classification Report (With Tuning):
               precision    recall  f1-score   support

           0       0.88      0.68      0.77       102
           1       0.74      0.91      0.82       103

    accuracy                           0.80       205
   macro avg       0.81      0.79      0.79       205
weighted avg       0.81      0.80      0.79       205
```

[ ]:

8 KNN Classifier

[60]:

```
from sklearn.neighbors import KNeighborsClassifier

# Initialize and train the model
knn_model = KNeighborsClassifier()
knn_model.fit(x_train, y_train)

# Make predictions
y_pred_knn = knn_model.predict(x_test)

# Evaluate the model
accuracy_knn = accuracy_score(y_test, y_pred_knn)
```

```
report_knn = classification_report(y_test, y_pred_knn)

print("KNN Accuracy:", accuracy_knn)
print("KNN Classification Report:\n", report_knn)
```

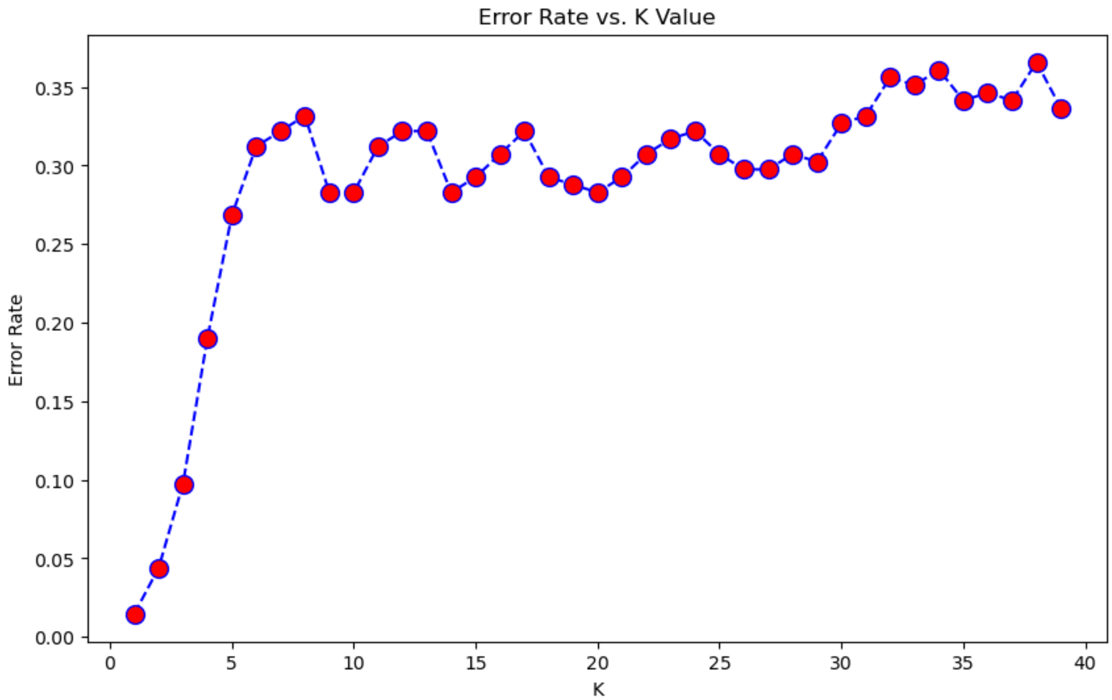
```
KNN Accuracy: 0.7317073170731707
KNN Classification Report:
               precision    recall  f1-score   support

           0       0.73      0.73      0.73       102
           1       0.73      0.74      0.73       103

    accuracy                           0.73       205
   macro avg       0.73      0.73      0.73       205
weighted avg       0.73      0.73      0.73       205
```

8.1 With HyperParametric Tuning

[61]:
```
error_rate = []
for  i in range(1,40):
    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))
```

[62]:
```
plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='blue',linestyle='dashed',marker='o',
         markerfacecolor='red',markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

[62]:
```
Text(0, 0.5, 'Error Rate')
```



[63]:
```
from sklearn.metrics import confusion_matrix
knn= KNeighborsClassifier(n_neighbors=3)

knn.fit(x_train,y_train)
pred = knn.predict(x_test)

print('With K=3')
print('\n')
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
With K=3


[[91 11]
 [ 9 94]]
              precision    recall  f1-score   support

           0       0.91      0.89      0.90       102
           1       0.90      0.91      0.90       103

    accuracy                           0.90       205
   macro avg       0.90      0.90      0.90       205
weighted avg       0.90      0.90      0.90       205
```

```
[ ]:
```

```
[ ]:
```