Machine learning

Lab manual

Course code : AI-41

Submitted to :

Mr. Saeed

Submitted by :

Seemal Mustafa **Registration no : 2023-BS-AI-013**

Bachelor of science in Artificial intelligence

DEPARTMENT OF COMPUTER SCIENCE

Contents

Lab 1 : Choosing a dataset

Description of dataset:

The dataset contains specifications and sales data for 161 different cellphone models. It includes 14 columns with various numeric features. Here's a summary of each column:

1. **Product_id**: Unique identifier for each phone.

2. **Price**: Price of the phone (likely in a given currency unit).

3. **Sale**: Sales quantity or rank.

4. **weight**: Weight of the phone in grams.

5. **resoloution**: Likely the screen size in inches (note the misspelling).

6. **ppi**: Pixels per inch – screen sharpness metric.

7. **cpu core**: Number of CPU cores.

8. **cpu freq**: CPU frequency in GHz.

9. **internal mem**: Internal memory (e.g., storage) in GB.

10. **ram**: RAM in GB.

11. **RearCam**: Rear camera resolution in megapixels.

12. **Front_Cam**: Front camera resolution in megapixels.

13. **battery**: Battery capacity in mAh.

14. **thickness**: Thickness of the phone in millimeters.

Lab 2 :Applying pre-processing steps

Importing library :

-import pandas as pd

Providing dataset path :

data = pd.read_csv('C:/Users/HP/Downloads/archive (2)/Cellphone.csv')

```
[10]: import pandas as pd
```

```
[12]: data = pd.read_csv('C:/Users/HP/Downloads/archive (2)/Cellphone.csv')
```

```
[14]: data
```

[14]:

	Product_id	Price	Sale	weight	resoloution	ppi	cpu core	cpu freq	internal mem	ram	RearCam	Front_Cam	battery	thickness
0	203	2357	10	135.0	5.20	424	8	1.350	16.0	3.000	13.00	8.0	2610	7.4
1	880	1749	10	125.0	4.00	233	2	1.300	4.0	1.000	3.15	0.0	1700	9.9
2	40	1916	10	110.0	4.70	312	4	1.200	8.0	1.500	13.00	5.0	2000	7.6
3	99	1315	11	118.5	4.00	233	2	1.300	4.0	0.512	3.15	0.0	1400	11.0
4	880	1749	11	125.0	4.00	233	2	1.300	4.0	1.000	3.15	0.0	1700	9.9
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
156	1206	3551	4638	178.0	5.46	538	4	1.875	128.0	6.000	12.00	16.0	4080	8.4
157	1296	3211	8016	170.0	5.50	534	4	1.975	128.0	6.000	20.00	8.0	3400	7.9
158	856	3260	8809	150.0	5.50	401	8	2.200	64.0	4.000	20.00	20.0	3000	6.8
159	1296	3211	8946	170.0	5.50	534	4	1.975	128.0	6.000	20.00	8.0	3400	7.9

Importing required libraries :

```
[19]: import numpy as np
```

```
[21]: import matplotlib.pyplot as plt
```

```
[22]: import seaborn as sns
```

```
[ ]: 1: Reading Data
```

```
[28]: data = pd.read_csv(r'C:/Users/HP/Downloads/archive (2)/Cellphone.csv')

      data.head()
```

[28]:

	Product_id	Price	Sale	weight	resoloution	ppi	cpu core	cpu freq	internal mem	ram	RearCam	Front_Cam	battery	thickness
0	203	2357	10	135.0	5.2	424	8	1.35	16.0	3.000	13.00	8.0	2610	7.4
1	880	1749	10	125.0	4.0	233	2	1.30	4.0	1.000	3.15	0.0	1700	9.9
2	40	1916	10	110.0	4.7	312	4	1.20	8.0	1.500	13.00	5.0	2000	7.6
3	99	1315	11	118.5	4.0	233	2	1.30	4.0	0.512	3.15	0.0	1400	11.0
4	880	1749	11	125.0	4.0	233	2	1.30	4.0	1.000	3.15	0.0	1700	9.9

Displaying data head:

[30]:
```
data.head(2)
```

[30]:

	Product_id	Price	Sale	weight	resoloution	ppi	cpu core	cpu freq	internal mem	ram	RearCam	Front_Cam	battery	thickness
0	203	2357	10	135.0	5.2	424	8	1.35	16.0	3.0	13.00	8.0	2610	7.4
1	880	1749	10	125.0	4.0	233	2	1.30	4.0	1.0	3.15	0.0	1700	9.9

[32]:
```
data.head(30)
```

[32]:

	Product_id	Price	Sale	weight	resoloution	ppi	cpu core	cpu freq	internal mem	ram	RearCam	Front_Cam	battery	thickness
0	203	2357	10	135.0	5.2	424	8	1.350	16.0	3.000	13.00	8.0	2610	7.4
1	880	1749	10	125.0	4.0	233	2	1.300	4.0	1.000	3.15	0.0	1700	9.9
2	40	1916	10	110.0	4.7	312	4	1.200	8.0	1.500	13.00	5.0	2000	7.6
3	99	1315	11	118.5	4.0	233	2	1.300	4.0	0.512	3.15	0.0	1400	11.0
4	880	1749	11	125.0	4.0	233	2	1.300	4.0	1.000	3.15	0.0	1700	9.9
5	947	2137	12	150.0	5.5	401	4	2.300	16.0	2.000	16.00	8.0	2500	9.5
6	774	1238	13	134.1	4.0	233	2	1.200	8.0	1.000	2.00	0.0	1560	11.7
7	947	2137	13	150.0	5.5	401	4	2.300	16.0	2.000	16.00	8.0	2500	9.5
8	99	1315	14	118.5	4.0	233	2	1.300	4.0	0.512	3.15	0.0	1400	11.0
9	1103	2580	15	145.0	5.1	432	4	2.500	16.0	2.000	16.00	2.0	2800	8.1
10	289	2438	16	162.0	5.3	277	8	1.500	32.0	4.000	13.00	8.0	4000	7.7
11	605	2006	16	161.0	5.5	200	8	1.400	4.0	1.000	5.00	0.0	2500	8.9
12	622	2174	16	140.0	5.0	294	4	1.300	16.0	1.000	13.00	5.0	2000	8.2
13	1058	2744	16	174.0	5.6	524	4	2.700	32.0	3.000	16.00	3.7	3000	8.3
14	1103	2580	16	145.0	5.1	432	4	2.500	16.0	2.000	16.00	2.0	2800	8.1
15	1120	1612	17	141.0	5.0	294	4	1.200	8.0	1.500	8.00	1.2	2040	10.0
16	187	2258	17	150.0	5.0	441	4	2.300	16.0	2.000	13.00	2.0	2300	10.0
17	315	2938	19	168.0	5.5	534	4	1.875	32.0	4.000	12.30	8.0	3450	8.5
18	1120	1612	19	141.0	5.0	294	4	1.200	8.0	1.500	8.00	1.2	2040	10.0
19	774	1238	20	134.1	4.0	233	2	1.200	8.0	1.000	2.00	0.0	1560	11.7
20	289	2438	21	162.0	5.3	277	8	1.500	32.0	4.000	13.00	8.0	4000	7.7
21	860	2392	22	147.0	5.2	282	8	1.400	32.0	3.000	13.00	16.0	2900	7.7

22	990	2977	22	152.0	5.1	577	8	2.300	32.0	4.000	12.00	5.0	3000	7.9
23	1058	2744	23	174.0	5.6	524	4	2.700	32.0	3.000	16.00	3.7	3000	8.3
24	104	1942	24	139.2	4.7	469	4	2.150	16.0	2.000	16.00	4.0	2200	10.3
25	776	1390	24	146.0	5.0	220	4	1.200	8.0	1.000	5.00	5.0	1905	8.8
26	605	2006	24	161.0	5.5	200	8	1.400	4.0	1.000	5.00	0.0	2500	8.9
27	315	2938	25	168.0	5.5	534	4	1.875	32.0	4.000	12.30	8.0	3450	8.5
28	776	1390	25	146.0	5.0	220	4	1.200	8.0	1.000	5.00	5.0	1905	8.8
29	10	1950	26	118.0	5.0	187	4	1.300	8.0	1.000	8.00	2.0	2000	6.4

```
data.tail()
```

	Product_id	Price	Sale	weight	resoloution	ppi	cpu core	cpu freq	internal mem	ram	RearCam	Front_Cam	battery	thickness
156	1206	3551	4638	178.0	5.46	538	4	1.875	128.0	6.0	12.0	16.0	4080	8.4
157	1296	3211	8016	170.0	5.50	534	4	1.975	128.0	6.0	20.0	8.0	3400	7.9
158	856	3260	8809	150.0	5.50	401	8	2.200	64.0	4.0	20.0	20.0	3000	6.8
159	1296	3211	8946	170.0	5.50	534	4	1.975	128.0	6.0	20.0	8.0	3400	7.9

Displaying data shape and tail:

```
36]: data.shape
```

```
36]: (161, 14)
```

```
38]: data.tail(20)
```

38]:

	Product_id	Price	Sale	weight	resoloution	ppi	cpu core	cpu freq	internal mem	ram	RearCam	Front_Cam	battery	thickness
141	701	628	1274	102.9	2.20	128	0	0.000	0.256	0.128	1.3	0.0	950	18.5
142	1161	2508	1530	152.0	5.20	424	4	2.500	16.000	3.000	20.7	2.2	3100	7.3
143	1161	2508	1584	152.0	5.20	424	4	2.500	16.000	3.000	20.7	2.2	3100	7.3
144	32	1921	1781	179.0	6.00	184	4	1.300	8.000	1.000	13.0	8.0	2580	8.0
145	32	1921	1862	179.0	6.00	184	4	1.300	8.000	1.000	13.0	8.0	2580	8.0
146	1137	3102	2071	180.0	5.50	806	8	1.750	32.000	3.000	23.0	5.1	3430	7.8
147	1137	3102	2088	180.0	5.50	806	8	1.750	32.000	3.000	23.0	5.1	3430	7.8
148	851	3055	2106	158.0	5.50	401	4	1.875	64.000	6.000	16.0	8.0	3000	7.4
149	826	614	2159	69.8	1.40	129	0	0.000	0.000	0.004	0.0	0.0	800	14.1
150	826	614	2171	69.8	1.40	129	0	0.000	0.000	0.004	0.0	0.0	800	14.1
151	851	3055	2173	158.0	5.50	401	4	1.875	64.000	6.000	16.0	8.0	3000	7.4
152	290	4361	3248	238.0	5.70	515	8	1.950	128.000	6.000	12.0	8.0	7000	7.4
153	290	4361	3291	238.0	5.70	515	8	1.950	128.000	6.000	12.0	8.0	7000	7.4
154	1131	2536	3619	202.0	6.00	367	8	1.500	16.000	3.000	21.5	16.0	2700	8.4
155	1206	3551	4408	178.0	5.46	538	4	1.875	128.000	6.000	12.0	16.0	4080	8.4
156	1206	3551	4638	178.0	5.46	538	4	1.875	128.000	6.000	12.0	16.0	4080	8.4
157	1296	3211	8016	170.0	5.50	534	4	1.975	128.000	6.000	20.0	8.0	3400	7.9
158	856	3260	8809	150.0	5.50	401	8	2.200	64.000	4.000	20.0	20.0	3000	6.8
159	1296	3211	8946	170.0	5.50	534	4	1.975	128.000	6.000	20.0	8.0	3400	7.9
160	1131	2536	9807	202.0	6.00	367	8	1.500	16.000	3.000	21.5	16.0	2700	8.4

```
: data.sample()
```

:

	Product_id	Price	Sale	weight	resoloution	ppi	cpu core	cpu freq	internal mem	ram	RearCam	Front_Cam	battery	thickness
68	198	1734	87	128.0	4.5	245	4	1.2	4.0	1.0	8.0	0.0	1840	8.5

Displaying data sample:

```
[42]: data.sample(30)
```

[42]:

	Product_id	Price	Sale	weight	resoloution	ppi	cpu core	cpu freq	internal mem	ram	RearCam	Front_Cam	battery	thickness
129	696	2466	499	154.0	5.5	534	4	2.700	32.000	3.000	13.00	2.1	3000	9.1
132	696	2466	567	154.0	5.5	534	4	2.700	32.000	3.000	13.00	2.1	3000	9.1
111	30	2975	302	149.0	5.5	534	8	1.600	32.000	3.000	16.00	8.0	3000	7.0
43	907	2087	40	147.0	5.0	294	4	1.300	32.000	3.000	8.00	5.0	2450	7.6
19	774	1238	20	134.1	4.0	233	2	1.200	8.000	1.000	2.00	0.0	1560	11.7
72	56	2044	93	310.0	8.0	283	8	2.000	8.000	2.000	5.00	2.0	4060	7.3
38	575	1777	36	174.0	5.5	178	4	1.300	4.000	0.512	5.00	0.0	2250	9.2
98	183	1522	187	160.0	5.0	220	2	1.200	0.000	1.000	8.00	0.0	2500	10.8
16	187	2258	17	150.0	5.0	441	4	2.300	16.000	2.000	13.00	2.0	2300	10.0
41	907	2087	37	147.0	5.0	294	4	1.300	32.000	3.000	8.00	5.0	2450	7.6
78	1221	2714	101	156.0	5.5	401	8	1.350	16.000	2.000	13.00	5.0	2300	5.1
39	860	2392	36	147.0	5.2	282	8	1.400	32.000	3.000	13.00	16.0	2900	7.7
140	701	628	1224	102.9	2.2	128	0	0.000	0.256	0.128	1.30	0.0	950	18.5
8	99	1315	14	118.5	4.0	233	2	1.300	4.000	0.512	3.15	0.0	1400	11.0
159	1296	3211	8946	170.0	5.5	534	4	1.975	128.000	6.000	20.00	8.0	3400	7.9
95	1062	1810	166	393.0	8.0	189	4	1.200	16.000	1.500	3.15	1.2	4450	9.7
25	776	1390	24	146.0	5.0	220	4	1.200	8.000	1.000	5.00	5.0	1905	8.8
135	301	2445	616	183.0	5.0	294	4	1.300	32.000	3.000	8.00	5.0	4000	8.5
120	460	1734	382	118.0	4.0	245	4	1.200	4.000	0.512	5.00	2.0	1730	10.9
6	774	1238	13	134.1	4.0	233	2	1.200	8.000	1.000	2.00	0.0	1560	11.7
18	1120	1612	19	141.0	5.0	294	4	1.200	8.000	1.500	8.00	1.2	2040	10.0
74	937	2571	96	97.0	4.8	306	4	1.200	16.000	2.000	8.00	5.0	2000	5.1
126	1198	705	427	110.0	2.2	128	0	0.000	0.128	0.032	2.00	0.0	900	15.6
44	974	2859	40	169.0	5.7	515	4	1.875	64.000	4.000	12.00	5.0	3500	7.9
7	947	2137	13	150.0	5.5	401	4	2.300	16.000	2.000	16.00	8.0	2500	9.5
113	64	754	308	77.9	2.4	167	0	0.000	0.004	0.004	0.00	0.0	850	12.4
122	1327	2001	393	194.8	5.7	258	4	1.200	16.000	2.000	8.00	1.0	3400	10.2
70	198	1734	89	128.0	4.5	245	4	1.200	4.000	1.000	8.00	0.0	1840	8.5

Describing data:

116	143	3287	344	170.0	5.5	401	8	2.000	32.000	4.000	12.00	13.0	5000	8.0
92	131	1831	156	154.0	5.0	294	4	1.200	8.000	1.000	13.00	5.0	2100	8.4

```
data.describe()
```

	Product_id	Price	Sale	weight	resoloution	ppi	cpu core	cpu freq	internal mem	ram	RearCam	Front_Cam	battery
count	161.000000	161.000000	161.000000	161.000000	161.000000	161.000000	161.000000	161.000000	161.000000	161.000000	161.000000	161.000000	161.000000
mean	675.559006	2215.596273	621.465839	170.426087	5.209938	335.055901	4.857143	1.502832	24.501714	2.204994	10.378261	4.503106	2842.111801
std	410.851583	768.187171	1546.618517	92.888612	1.509953	134.826659	2.444016	0.599783	28.804773	1.609831	6.181585	4.342053	1366.990838
min	10.000000	614.000000	10.000000	66.000000	1.400000	121.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	800.000000
25%	237.000000	1734.000000	37.000000	134.100000	4.800000	233.000000	4.000000	1.200000	8.000000	1.000000	5.000000	0.000000	2040.000000
50%	774.000000	2258.000000	106.000000	153.000000	5.150000	294.000000	4.000000	1.400000	16.000000	2.000000	12.000000	5.000000	2800.000000
75%	1026.000000	2744.000000	382.000000	170.000000	5.500000	428.000000	8.000000	1.875000	32.000000	3.000000	16.000000	8.000000	3240.000000
max	1339.000000	4361.000000	9807.000000	753.000000	12.200000	806.000000	8.000000	2.700000	128.000000	6.000000	23.000000	20.000000	9500.000000

Data cleaning:

2: Data Cleaning

```
data.isnull().sum()
```

```
Product_id      0
Price           0
Sale            0
weight          0
resoloution     0
ppi             0
cpu core        0
cpu freq        0
internal mem    0
ram             0
RearCam         0
Front_Cam       0
battery         0
thickness       0
dtype: int64
```

```
import pandas as pd
import numpy as np
```

Displaying data shape and removing duplicates:

```
numeric_cols = data.select_dtypes(include=[np.number])
non_numeric_cols = data.select_dtypes(exclude=[np.number])


numeric_cols = numeric_cols.fillna(numeric_cols.mean())


for col in non_numeric_cols.columns:
    if non_numeric_cols[col].isnull().any():
        non_numeric_cols[col] = non_numeric_cols[col].fillna(non_numeric_cols[col].mode()[0])


data = pd.concat([numeric_cols, non_numeric_cols], axis=1)


missing_values = data.isnull().sum()
print(missing_values)
```

```
Product_id      0
Price           0
Sale            0
weight          0
resoloution     0
ppi             0
cpu core        0
```

```
cpu freq        0
internal mem    0
ram             0
RearCam         0
Front_Cam       0
battery         0
thickness       0
dtype: int64
```

```
data.shape
```

```
(161, 14)
```

```
Removal: Deleting rows with missing values.
```

```
data.shape
```

```
(161, 14)
```

```
data.dropna(inplace=True)


missing_values = data.isnull().sum()
print(missing_values)
```

```
Product_id      0
Price           0
Sale            0
weight          0
resoloution     0
ppi             0
cpu core        0
cpu freq        0
internal mem    0
ram             0
RearCam         0
Front_Cam       0
battery         0
thickness       0
dtype: int64
```

```
data.shape
```

```
(161, 14)
```

```
Removing Duplicates
```

```
data = pd.read_csv(r'C:/Users/HP/Downloads/archive (2)/Cellphone.csv')
data.shape
```

Detecting and removing outliers:

```
]:  (161, 14)
```

```
]:  data.drop_duplicates(inplace=True)
    data.shape
```

```
]:  (161, 14)
```

```
]:  3: Outlier Detection and Removal
```

```
]:  import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt

    data = pd.read_csv(r'C:/Users/HP/Downloads/archive (2)/Cellphone.csv')



    data.describe()
```

]:

	Product_id	Price	Sale	weight	resoloution	ppi	cpu core	cpu freq	internal mem	ram	RearCam	Front_Cam	battery
count	161.000000	161.000000	161.000000	161.000000	161.000000	161.000000	161.000000	161.000000	161.000000	161.000000	161.000000	161.000000	161.000000

mean	675.559006	2215.596273	621.465839	170.426087	5.209938	335.055901	4.857143	1.502832	24.501714	2.204994	10.378261	4.503106	2842.111801
std	410.851583	768.187171	1546.618517	92.888612	1.509953	134.826659	2.444016	0.599783	28.804773	1.609831	6.181585	4.342053	1366.990838
min	10.000000	614.000000	10.000000	66.000000	1.400000	121.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	800.000000
25%	237.000000	1734.000000	37.000000	134.100000	4.800000	233.000000	4.000000	1.200000	8.000000	1.000000	5.000000	0.000000	2040.000000
50%	774.000000	2258.000000	106.000000	153.000000	5.150000	294.000000	4.000000	1.400000	16.000000	2.000000	12.000000	5.000000	2800.000000
75%	1026.000000	2744.000000	382.000000	170.000000	5.500000	428.000000	8.000000	1.875000	32.000000	3.000000	16.000000	8.000000	3240.000000
max	1339.000000	4361.000000	9807.000000	753.000000	12.200000	806.000000	8.000000	2.700000	128.000000	6.000000	23.000000	20.000000	9500.000000

```
0.25-1.5*0.5
```

```
-0.5
```

```
0.75 + 1.5 * 0.5
```

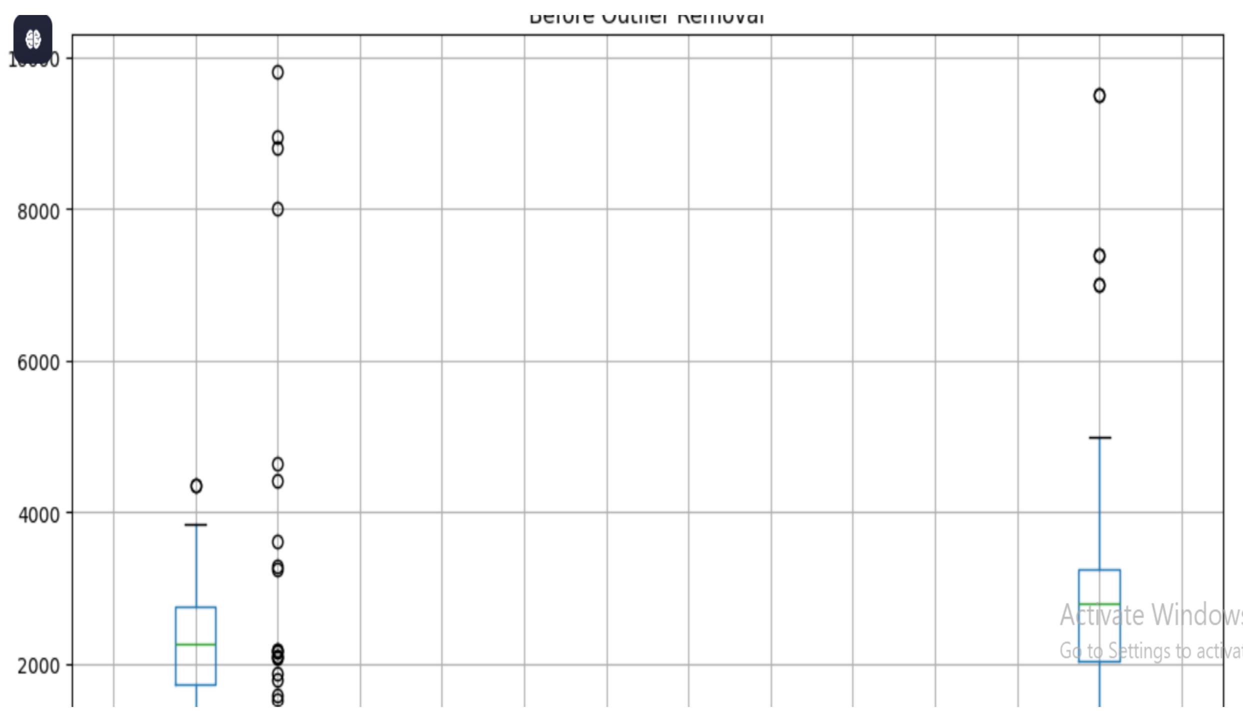
```
1.5
```

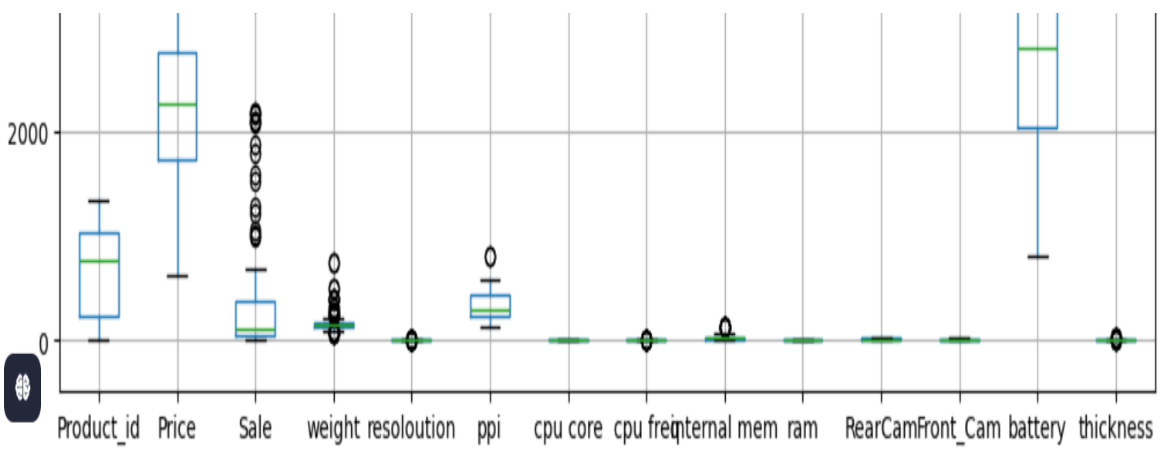


```
numeric_cols = data.select_dtypes(include=[np.number])
```

Verifying numeric cols and displaying graph before outlier removal:

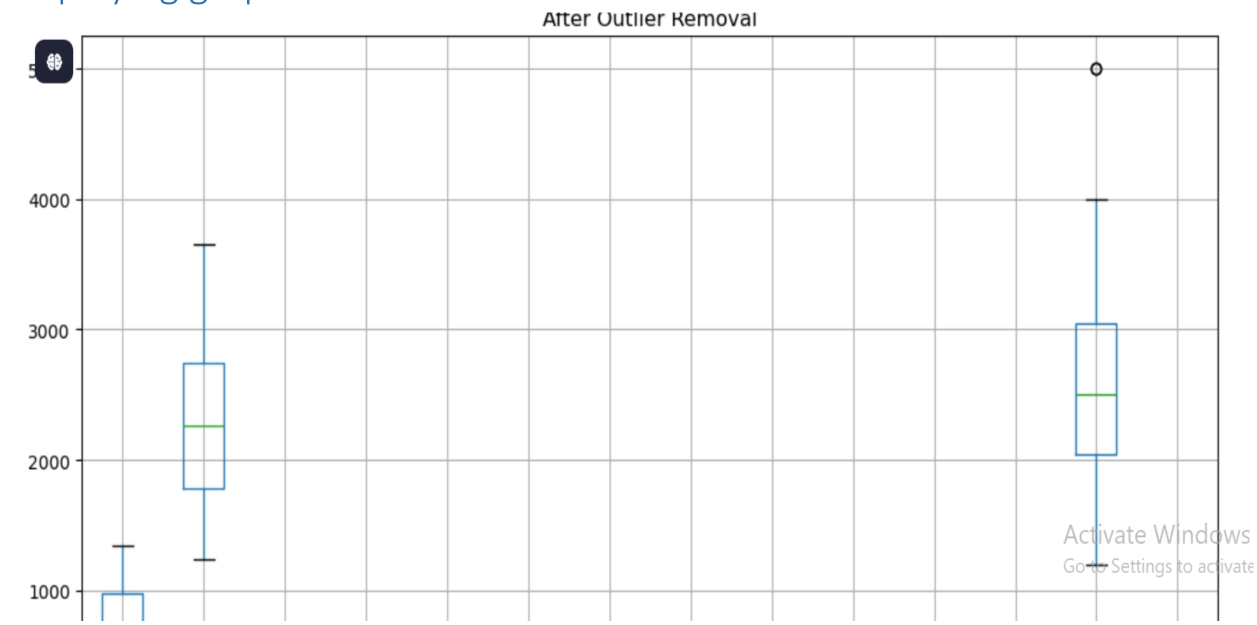
```
Q1 = numeric_cols.quantile(0.25)
Q3 = numeric_cols.quantile(0.75)
IQR = Q3 - Q1



data_cleaned = data[~((numeric_cols < (Q1 - 1.5 * IQR)) | (numeric_cols > (Q3 + 1.5 * IQR))).any(axis=1)]


plt.figure(figsize=(20, 6))


plt.subplot(1, 2, 1)
numeric_cols.boxplot()
plt.title("Before Outlier Removal")


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

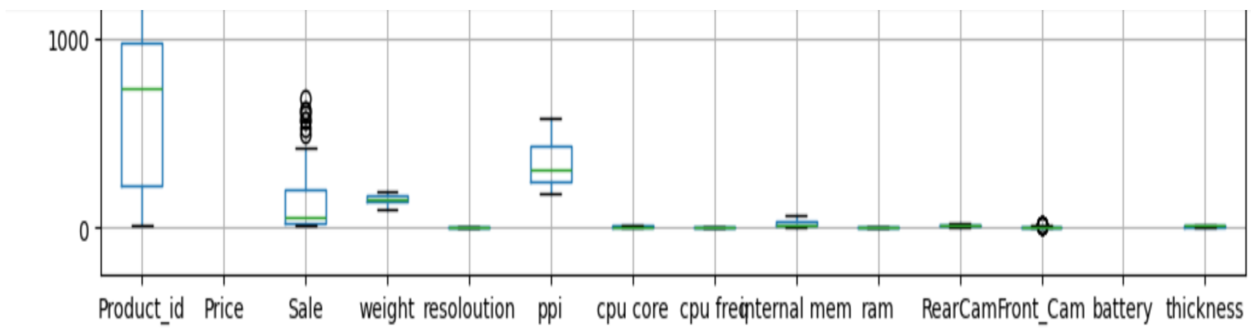





```
plt.figure(figsize=(20, 6))


plt.subplot(1, 2, 2)
data_cleaned.select_dtypes(include=[np.number]).boxplot()
plt.title("After Outlier Removal")

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

Displaying graph after outlier removal:






```
data_cleaned.shape
```

```
(113, 14)
```

```
data_cleaned.head()
```

	Product_id	Price	Sale	weight	resoloution	ppi	cpu core	cpu freq	internal mem	ram	RearCam	Front_Cam	battery	thickness
0	203	2357	10	135.0	5.2	424	8	1.35	16.0	3.000	13.00	8.0	2610	7.4
1	880	1749	10	125.0	4.0	233	2	1.30	4.0	1.000	3.15	0.0	1700	9.9
2	40	1916	10	110.0	4.7	312	4	1.20	8.0	1.500	13.00	5.0	2000	7.6
3	99	1315	11	118.5	4.0	233	2	1.30	4.0	0.512	3.15	0.0	1400	11.0

In upper cells we displayed the shape of cleaned data:

Data transformation :

[ ]:

```
4. Data Transformation
```

[78]:

```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler


data = pd.read_csv(r'C:/Users/HP/Downloads/archive (2)/Cellphone.csv')


numeric_cols = data.select_dtypes(include=[np.number])
non_numeric_cols = data.select_dtypes(exclude=[np.number])


scaler = MinMaxScaler()
scaled_numeric_data = scaler.fit_transform(numeric_cols)


scaled_numeric_df = pd.DataFrame(scaled_numeric_data, columns=numeric_cols.columns)


scaled_data = pd.concat([scaled_numeric_df, non_numeric_cols.reset_index(drop=True)], axis=1)
```

Standardizing data :

```python
print(scaled_data.shape)
print()
print('*' * 60)
scaled_data.head()
```

```
(161, 14)

************************************************************
```

	Product_id	Price	Sale	weight	resoloution	ppi	cpu core	cpu freq	internal mem	ram	RearCam	Front_Cam	battery	thickness
0	0.145222	0.465172	0.000000	0.100437	0.351852	0.442336	1.00	0.500000	0.12500	0.500000	0.565217	0.40	0.208046	0.171642
1	0.654628	0.302909	0.000000	0.085881	0.240741	0.163504	0.25	0.481481	0.03125	0.166667	0.136957	0.00	0.103448	0.358209
2	0.022573	0.347478	0.000000	0.064047	0.305556	0.278832	0.50	0.444444	0.06250	0.250000	0.565217	0.25	0.137931	0.186567
3	0.066968	0.187083	0.000102	0.076419	0.240741	0.163504	0.25	0.481481	0.03125	0.085333	0.136957	0.00	0.068966	0.440299
4	0.654628	0.302909	0.000102	0.085881	0.240741	0.163504	0.25	0.481481	0.03125	0.166667	0.136957	0.00	0.103448	0.358209

```
Standardization
```

```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
```

Displaying scaled data :

```
data = pd.read_csv(r'C:/Users/HP/Downloads/archive (2)/Cellphone.csv')


numeric_cols = data.select_dtypes(include=[np.number])
non_numeric_cols = data.select_dtypes(exclude=[np.number])


scaler = StandardScaler()
scaled_numeric_data = scaler.fit_transform(numeric_cols)


scaled_numeric_df = pd.DataFrame(scaled_numeric_data, columns=numeric_cols.columns)


scaled_data = pd.concat([scaled_numeric_df, non_numeric_cols.reset_index(drop=True)], axis=1)


print(scaled_data.shape)
print()
print('*' * 60)
scaled_data.head()
```

```
(161, 14)
```

Doing one hot encoding to show data in binary form:

```
************************************************************
```

]:

	Product_id	Price	Sale	weight	resoloution	ppi	cpu core	cpu freq	internal mem	ram	RearCam	Front_Cam	battery	thickness
0	-1.153783	0.184649	-0.396590	-0.382572	-0.006602	0.661751	1.289952	-0.255608	-0.296070	0.495385	0.425444	0.807868	-0.170327	-0.696211
1	0.499156	-0.609294	-0.396590	-0.490564	-0.803808	-0.759303	-1.172684	-0.339231	-0.713968	-0.750857	-1.172970	-1.040327	-0.838100	0.447564
2	-1.551758	-0.391221	-0.396590	-0.652552	-0.338771	-0.171538	-0.351805	-0.506479	-0.574669	-0.439297	0.425444	0.114795	-0.617955	-0.604709
3	-1.407705	-1.176024	-0.395942	-0.560759	-0.803808	-0.759303	-1.172684	-0.339231	-0.713968	-1.054940	-1.172970	-1.040327	-1.058245	0.950825
4	0.499156	-0.609294	-0.395942	-0.490564	-0.803808	-0.759303	-1.172684	-0.339231	-0.713968	-0.750857	-1.172970	-1.040327	-0.838100	0.447564

]:

```
5: One-Hot Encoding
```

]:

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler


data = pd.read_csv(r'C:/Users/HP/Downloads/archive (2)/Cellphone.csv')
data.head(2)
```


]:

Product_id	Price	Sale	weight	resoloution	ppi	cpu core	cpu freq	internal mem	ram	RearCam	Front_Cam	battery	thickness

[82]:

	Product_id	Price	Sale	weight	resoloution	ppi	cpu core	cpu freq	internal mem	ram	RearCam	Front_Cam	battery	thickness
0	203	2357	10	135.0	5.2	424	8	1.35	16.0	3.0	13.00	8.0	2610	7.4
1	880	1749	10	125.0	4.0	233	2	1.30	4.0	1.0	3.15	0.0	1700	9.9

[88]:

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler


data = pd.read_csv(r'C:/Users/HP/Downloads/archive (2)/Cellphone.csv')


cat_features = [feature for feature in data.columns if data[feature].dtype == 'O']


data1 = pd.get_dummies(data, columns=cat_features)


scaled_data = pd.concat([data, data1], axis=1)


print(scaled_data.shape)
```



```
print('*' * 70)

scaled_data.head()
```

(161, 28)

****

38]:

	Product_id	Price	Sale	weight	resoloution	ppi	cpu core	cpu freq	internal mem	ram	...	resoloution	ppi	cpu core	cpu freq	internal mem	ram	RearCam	Front_Cam	battery	thickn
0	203	2357	10	135.0	5.2	424	8	1.35	16.0	3.000	...	5.2	424	8	1.35	16.0	3.000	13.00	8.0	2610	
1	880	1749	10	125.0	4.0	233	2	1.30	4.0	1.000	...	4.0	233	2	1.30	4.0	1.000	3.15	0.0	1700	
2	40	1916	10	110.0	4.7	312	4	1.20	8.0	1.500	...	4.7	312	4	1.20	8.0	1.500	13.00	5.0	2000	
3	99	1315	11	118.5	4.0	233	2	1.30	4.0	0.512	...	4.0	233	2	1.30	4.0	0.512	3.15	0.0	1400	1
4	880	1749	11	125.0	4.0	233	2	1.30	4.0	1.000	...	4.0	233	2	1.30	4.0	1.000	3.15	0.0	1700	

5 rows × 28 columns

90]:

```
data.columns
```

Displaying data columns and scaled data columns:

```
[0]: data.columns
```

```
[0]: Index(['Product_id', 'Price', 'Sale', 'weight', 'resoloution', 'ppi',
       'cpu core', 'cpu freq', 'internal mem', 'ram', 'RearCam', 'Front_Cam',
       'battery', 'thickness'],
      dtype='object')
```

```
2]: scaled_data.columns
```

```
2]: Index(['Product_id', 'Price', 'Sale', 'weight', 'resoloution', 'ppi',
       'cpu core', 'cpu freq', 'internal mem', 'ram', 'RearCam', 'Front_Cam',
       'battery', 'thickness', 'Product_id', 'Price', 'Sale', 'weight',
       'resoloution', 'ppi', 'cpu core', 'cpu freq', 'internal mem', 'ram',
       'RearCam', 'Front_Cam', 'battery', 'thickness'],
      dtype='object')
```

```
4]: data1.head()
```

4]:

	Product_id	Price	Sale	weight	resoloution	ppi	cpu core	cpu freq	internal mem	ram	RearCam	Front_Cam	battery	thickness
0	203	2357	10	135.0	5.2	424	8	1.35	16.0	3.000	13.00	8.0	2610	7.4
1	880	1749	10	125.0	4.0	233	2	1.30	4.0	1.000	3.15	0.0	1700	9.9
2	40	1916	10	110.0	4.7	312	4	1.20	8.0	1.500	13.00	5.0	2000	7.6
3	99	1315	11	118.5	4.0	233	2	1.30	4.0	0.512	3.15	0.0	1400	11.0
4	880	1749	11	125.0	4.0	233	2	1.30	4.0	1.000	3.15	0.0	1700	9.9

Doing data reduction and handling imbalanced data:

```
: 6: Data Reduction¶
```

```
: scaled_data.shape
```

```
: (161, 28)
```

```
: 7: Handling Imbalanced Data
  Resampling Techniques
  Oversampling
```

```
: data.shape
```

```
: (161, 14)
```

```
: from sklearn.preprocessing import StandardScaler
  import pandas as pd


  numeric_features = data.select_dtypes(include=[np.number]).columns


  scaler = StandardScaler()
```

```
data[numeric_features] = scaler.fit_transform(data[numeric_features])


if 'price' in data.columns:
    if data['price'].dtype != 'int64' and data['price'].dtype != 'bool':
        data['price'] = (data['price'] > 0.5).astype(int)


if 'price' in data.columns:
    X = data.drop(columns=['price'])
else:
    print("'price' column does not exist. Cannot drop it.")
    X = data.copy()
```

```
'price' column does not exist. Cannot drop it.
```

```
import pandas as pd
from sklearn.utils import resample


data = pd.read_csv(r'C:/Users/HP/Downloads/archive (2)/Cellphone.csv')


print("Columns before resampling:", data.columns)
```

Resampling the data and displaying the resampled data shape:

```
data_resampled = resample(data, replace=True, n_samples=len(data), random_state=42)


print("Columns after resampling:", data_resampled.columns)


if 'price' in data_resampled.columns:
    print(data_resampled['price'].value_counts(normalize=True))
else:
    print("'price' column not found in data_resampled.")
```

```
Columns before resampling: Index(['Product_id', 'Price', 'Sale', 'weight', 'resoloution', 'ppi',
       'cpu core', 'cpu freq', 'internal mem', 'ram', 'RearCam', 'Front_Cam',
       'battery', 'thickness'],
      dtype='object')
Columns after resampling: Index(['Product_id', 'Price', 'Sale', 'weight', 'resoloution', 'ppi',
       'cpu core', 'cpu freq', 'internal mem', 'ram', 'RearCam', 'Front_Cam',
       'battery', 'thickness'],
      dtype='object')
'price' column not found in data_resampled.
```

```
data_resampled.shape
```

Splitting the data :

```
[122]:  (161, 14)
```

```
[ ]:  8: Splitting Data
```

```
[3]:  import pandas as pd
      from sklearn.model_selection import train_test_split


      data = pd.read_csv(r'C:/Users/HP/Downloads/archive (2)/Cellphone.csv')


      if 'price' in data.columns:

          X = data.drop('price', axis=1)
          y = data['price']


          X = pd.get_dummies(X, drop_first=True)


          X_train, X_test, y_train, y_test = train_test_split(
              X, y, test_size=0.3, random_state=42
          )
```

Performing regression :creating and training regression model and making prediction.

```
    print("Data split successful!")
    print("X_train shape:", X_train.shape)
    print("X_test shape:", X_test.shape)
else:
    print("Column 'price' not found in the dataset.")
```

```
Column 'price' not found in the dataset.
```

```
:  Regression
```

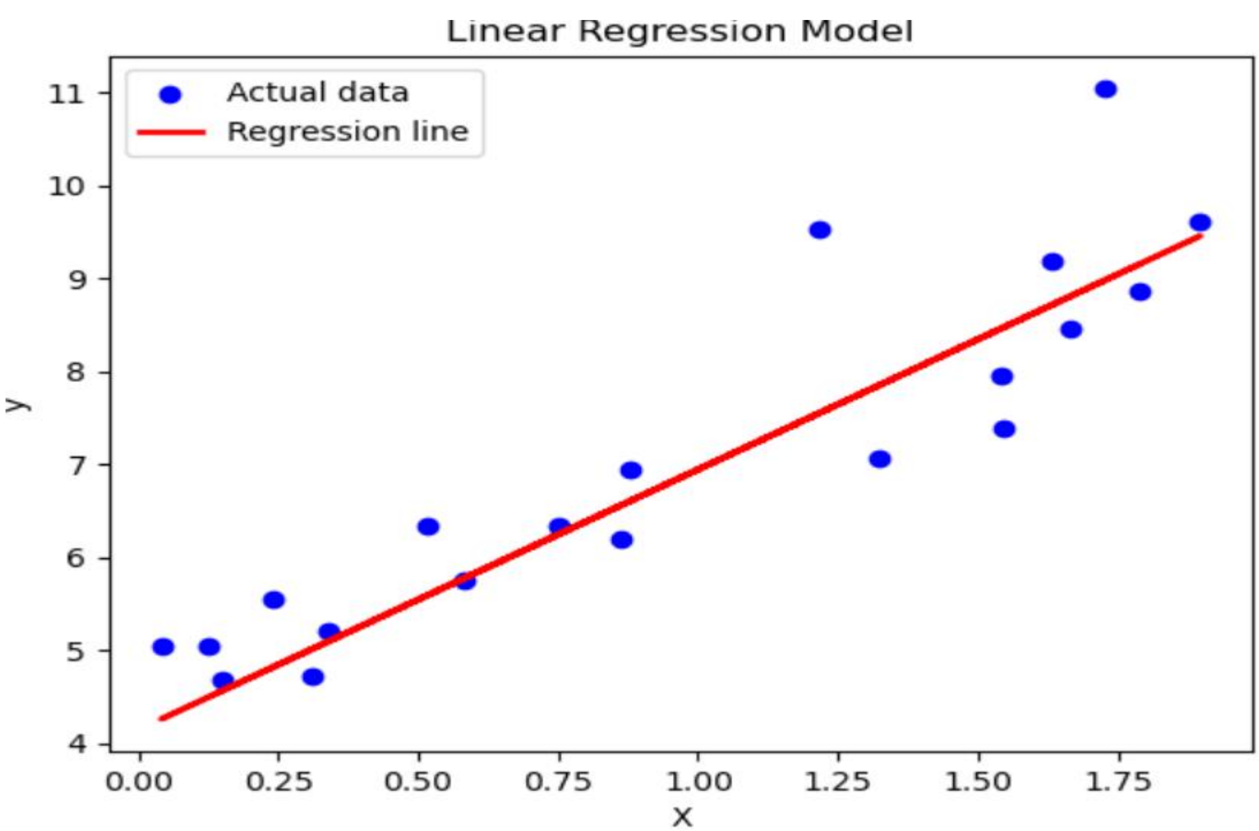
```
:  # Importing necessary libraries
   import numpy as np
   import pandas as pd
   import matplotlib.pyplot as plt
   from sklearn.model_selection import train_test_split
   from sklearn.linear_model import LinearRegression
   from sklearn.metrics import mean_squared_error

   # Generating synthetic data
   np.random.seed(42)
   X = 2 * np.random.rand(100, 1)
   y = 4 + 3 * X + np.random.randn(100, 1)

   # Splitting the dataset into training and testing sets
```

We evaluated the model and displayed the graph :

```
# Splitting the dataset 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)

# Creating and training the regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Making predictions
y_pred = model.predict(X_test)

# Evaluating the model
mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')

# Plotting the regression line
plt.scatter(X_test, y_test, color='blue', label='Actual data')
plt.plot(X_test, y_pred, color='red', linewidth=2, label='Regression line')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.title('Linear Regression Model')
plt.show()
```

```
Mean Squared Error: 0.6536995137170021
```

Linear Regression Model



Now we evaluated the whole data set :

```
[ ]: Evaluation
```

```python
[8]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

np.random.seed(42)
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)
```

```python
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error (MSE): {mse}')
print(f'Mean Absolute Error (MAE): {mae}')
print(f'Root Mean Squared Error (RMSE): {rmse}')
print(f'R² Score: {r2}')
```

```
Mean Squared Error (MSE): 0.6536995137170021
Mean Absolute Error (MAE): 0.5913425779189777
Root Mean Squared Error (RMSE): 0.8085168605026132
R² Score: 0.8072059636181392
```

Lab 3 : classification dataset.

"Diabetes Classification" dataset:

- **Total Records:**

  128 individuals

- **Purpose:**

  To classify whether a person has diabetes based on health and lifestyle factors

- **Number of Features:**

  11 columns including the target label

- **Feature Types:**

  Mix of numerical (e.g., Age, BMI, FBS, HbA1c) and categorical (e.g., Gender, Blood Pressure, Diet) variables

- **Health-Related Features:**
  - **Age:** Age of the individual
  - **BMI:** Body Mass Index
  - **Blood Pressure:** Categorical (Normal, High, Low)
  - **FBS:** Fasting Blood Sugar (mg/dL)
  - **HbA1c:** Average blood glucose level over 3 months (%)

- **Lifestyle and Background Factors:**
  - **Gender:** Male or Female
  - **Family History of Diabetes:** Yes or No
  - **Smoking:** Yes or No
  - **Diet:** Healthy or Poor
  - **Exercise:** Regular or No

- **Target Variable:**

  Daignosis— indicates if the person is diabetic (`Yes`) or not (`No`)

- **Use Case:**

  Suitable for classification tasks in machine learning, especially in medical risk prediction and health analytics

Importing required libraries and showing shape of the data :

```
[93]: import pandas as pd
      from sklearn.model_selection import train_test_split
      from sklearn.preprocessing import StandardScaler, LabelEncoder


      df = pd.read_csv("Diabetes Classification.csv")

      df.shape

[93]: (128, 11)

[94]: df.columns

[94]: Index(['Age', 'Gender', 'BMI', 'Blood Pressure', 'FBS', 'HbA1c',
             'Family History of Diabetes', 'Smoking', 'Diet', 'Exercise',
             'Diagnosis'],
            dtype='object')
```

Displaying information of dataset:

[95]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 128 entries, 0 to 127
Data columns (total 11 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   Age                          128 non-null    int64
 1   Gender                       128 non-null    object
 2   BMI                          128 non-null    int64
 3   Blood Pressure               128 non-null    object
 4   FBS                          128 non-null    int64
 5   HbA1c                        128 non-null    float64
 6   Family History of Diabetes   128 non-null    object
 7   Smoking                      128 non-null    object
 8   Diet                         128 non-null    object
 9   Exercise                     128 non-null    object
 10  Diagnosis                    128 non-null    object
dtypes: float64(1), int64(3), object(7)
memory usage: 11.1+ KB
```

Describing the dataset :

[96]:
```
df.describe()
```

[96]:

	Age	BMI	FBS	HbA1c
count	128.000000	128.000000	128.000000	128.000000
mean	42.031250	35.359375	162.500000	7.887500
std	16.783915	14.981739	61.323975	2.146339
min	12.000000	10.000000	80.000000	5.000000
25%	28.000000	24.000000	120.000000	6.400000
50%	40.000000	34.000000	160.000000	7.800000
75%	55.000000	45.500000	205.000000	9.375000
max	75.000000	67.000000	280.000000	12.000000

Displaying head of data :

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

	Age	Gender	BMI	Blood Pressure	FBS	HbA1c	Family History of Diabetes	Smoking	Diet	Exercise	Diagnosis
0	45	Male	25	Normal	100	5.7	No	No	Healthy	Regular	No
1	55	Female	30	High	120	6.4	Yes	Yes	Poor	No	Yes
2	65	Male	35	High	140	7.1	Yes	Yes	Poor	No	Yes
3	75	Female	40	High	160	7.8	Yes	Yes	Poor	No	Yes
4	40	Male	20	Normal	80	5.0	No	No	Healthy	Regular	No

Encoding and splitting features:

Encode categorical features

```
label_encoders = {}
categorical_cols = ['Gender', 'Blood Pressure', 'Family History of Diabetes',
                    'Smoking', 'Diet', 'Exercise', 'Diagnosis']

for col in categorical_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le
```

Split features and target

```
X = df.drop('Diagnosis', axis=1)
y = df['Diagnosis']
```

Training and standardizing numerical features:

Train-test split

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

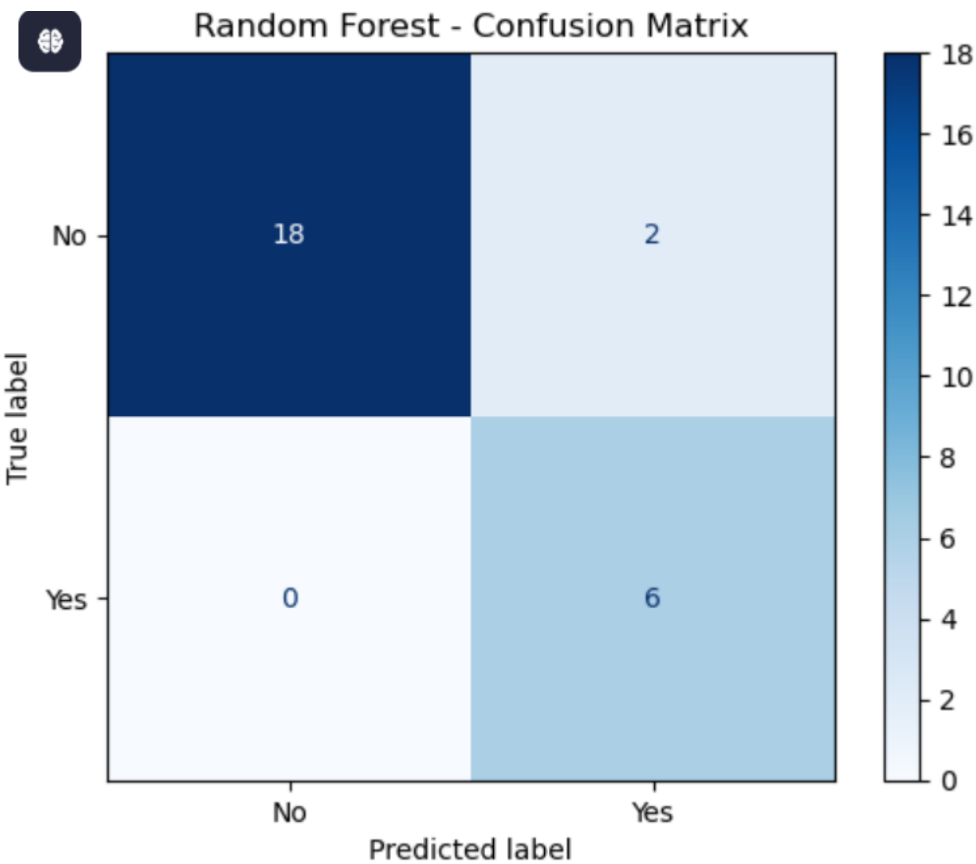
Standardize numerical features

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Applying random forest classifier and showing in graph form :

Random Forest Classifier

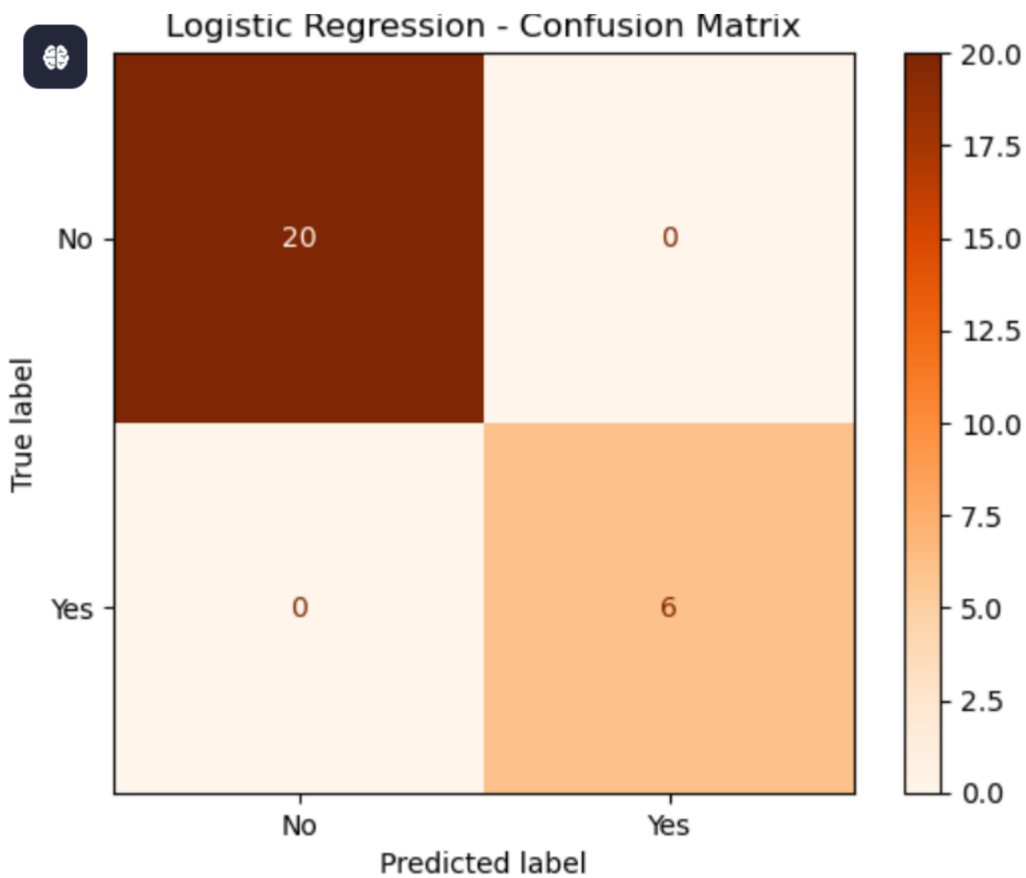
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt


rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train_scaled, y_train)
y_pred_rf = rf_model.predict(X_test_scaled)


cm_rf = confusion_matrix(y_test, y_pred_rf)
disp_rf = ConfusionMatrixDisplay(cm_rf, display_labels=label_encoders['Diagnosis'].classes_)
disp_rf.plot(cmap=plt.cm.Blues)
plt.title("Random Forest - Confusion Matrix")
plt.show()
```



Applying logistic regression and displaying it in graph form:

Logistic Regression

```
from sklearn.linear_model import LogisticRegression


lr_model = LogisticRegression()
lr_model.fit(X_train_scaled, y_train)
y_pred_lr = lr_model.predict(X_test_scaled)


cm_lr = confusion_matrix(y_test, y_pred_lr)
disp_lr = ConfusionMatrixDisplay(cm_lr, display_labels=label_encoders['Diagnosis'].classes_)
disp_lr.plot(cmap=plt.cm.Oranges)
plt.title("Logistic Regression - Confusion Matrix")
plt.show()
```



Applying ANN and displaying it in graph form :

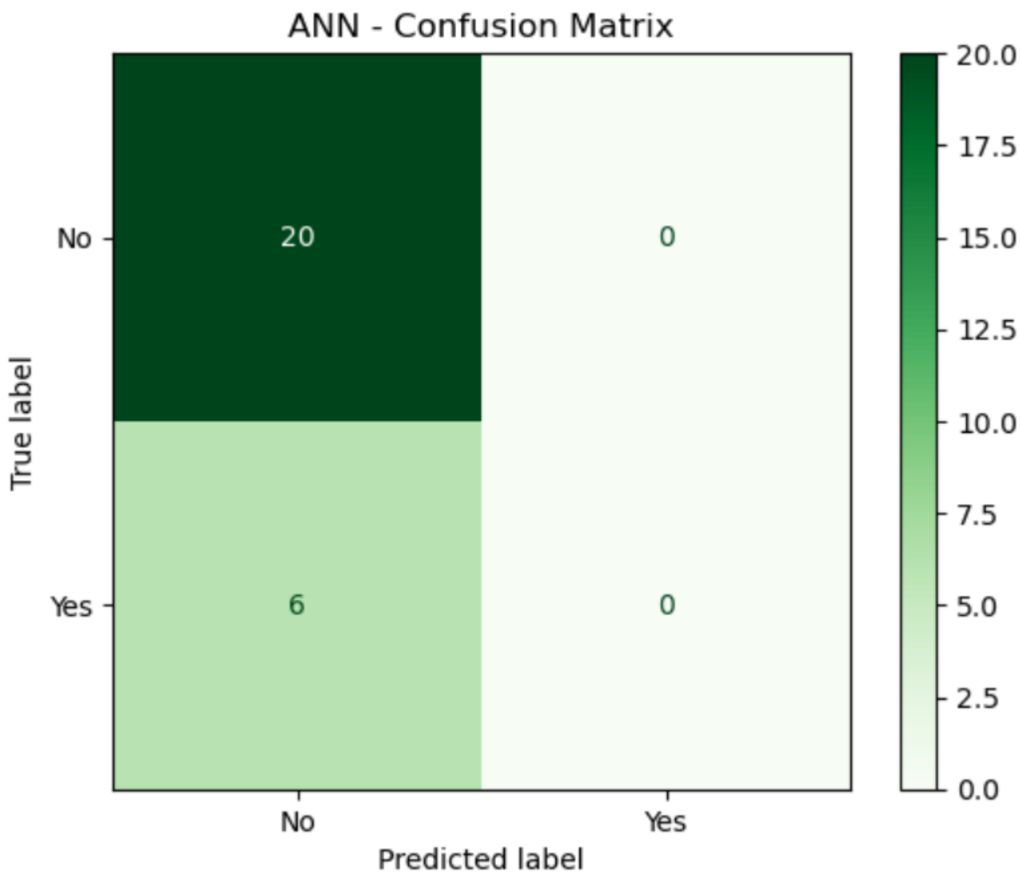
Artificial Neural Network (ANN)

```
]: import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

ann_model = Sequential([
    Dense(10, activation='relu', input_shape=(X_train_scaled.shape[1],)),
    Dense(10, activation='relu'),
    Dense(1, activation='sigmoid')
])

ann_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

ann_model.fit(X_train_scaled, y_train, epochs=20, batch_size=32, validation_split=0.2)

y_pred_ann_prob = ann_model.predict(X_test_scaled)
y_pred_ann = (y_pred_ann_prob > 0.5).astype(int).flatten()
```

```
cm_ann = confusion_matrix(y_test, y_pred_ann)
disp_ann = ConfusionMatrixDisplay(confusion_matrix=cm_ann, display_labels=label_encoders['Diagnosis'].classes_)
disp_ann.plot(cmap=plt.cm.Greens)
plt.title("ANN - Confusion Matrix")
plt.show()
```

```
Epoch 1/20
C:\Users\HP\anaconda3\Lib\site-packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an `input_shape`/
n using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 312ms/step - accuracy: 0.7554 - loss: 0.5591 - val_accuracy: 0.7143 - val_loss: 0.5237
Epoch 2/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 146ms/step - accuracy: 0.7912 - loss: 0.5333 - val_accuracy: 0.7143 - val_loss: 0.5150
Epoch 3/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 104ms/step - accuracy: 0.7678 - loss: 0.5333 - val_accuracy: 0.7143 - val_loss: 0.5066
Epoch 4/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 95ms/step - accuracy: 0.7694 - loss: 0.5136 - val_accuracy: 0.7143 - val_loss: 0.4991
Epoch 5/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 140ms/step - accuracy: 0.7538 - loss: 0.5135 - val_accuracy: 0.7143 - val_loss: 0.4920
Epoch 6/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 118ms/step - accuracy: 0.7772 - loss: 0.4785 - val_accuracy: 0.7143 - val_loss: 0.4852
Epoch 7/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 114ms/step - accuracy: 0.7694 - loss: 0.4756 - val_accuracy: 0.7143 - val_loss: 0.4787
Epoch 8/20
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 107ms/step - accuracy: 0.7812 - loss: 0.4586 - val_accuracy: 0.7143 - val_loss: 0.4724
Epoch 9/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 90ms/step - accuracy: 0.7499 - loss: 0.4694 - val_accuracy: 0.7143 - val_loss: 0.4663
Epoch 10/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 100ms/step - accuracy: 0.7382 - loss: 0.4727 - val_accuracy: 0.7143 - val_loss: 0.4604
Epoch 11/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 101ms/step - accuracy: 0.7694 - loss: 0.4400 - val_accuracy: 0.7143 - val_loss: 0.4546
Epoch 12/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 91ms/step - accuracy: 0.7460 - loss: 0.4593 - val_accuracy: 0.7143 - val_loss: 0.4488
Epoch 13/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 107ms/step - accuracy: 0.7772 - loss: 0.4216 - val_accuracy: 0.7143 - val_loss: 0.4432
Epoch 14/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 105ms/step - accuracy: 0.7851 - loss: 0.4154 - val_accuracy: 0.7143 - val_loss: 0.4378
Epoch 15/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 90ms/step - accuracy: 0.7460 - loss: 0.4301 - val_accuracy: 0.7143 - val_loss: 0.4328
Epoch 16/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 91ms/step - accuracy: 0.7460 - loss: 0.4247 - val_accuracy: 0.7143 - val_loss: 0.4280
Epoch 17/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 97ms/step - accuracy: 0.7577 - loss: 0.4047 - val_accuracy: 0.7143 - val_loss: 0.4235
Epoch 18/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 112ms/step - accuracy: 0.7538 - loss: 0.3894 - val_accuracy: 0.7143 - val_loss: 0.4191
Epoch 19/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 120ms/step - accuracy: 0.7733 - loss: 0.3936 - val_accuracy: 0.7143 - val_loss: 0.4151
Epoch 20/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 78ms/step - accuracy: 0.7616 - loss: 0.3858 - val_accuracy: 0.7143 - val_loss: 0.4113
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 130ms/step
```



Applying smote to balance the imbalance dataset :

```
import pandas as pd
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split


print("Columns:", df.columns)


X = df.drop('Diagnosis', axis=1)
y = df['Diagnosis']


X = pd.get_dummies(X, drop_first=True)


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)


smote = SMOTE(random_state=42)


X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)
```

```
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)


print("Before SMOTE:")
print(y_train.value_counts())
print("\nAfter SMOTE:")
print(y_train_smote.value_counts())
```

```
Columns: Index(['Age', 'Gender', 'BMI', 'Blood Pressure', 'FBS', 'HbA1c',
       'Family History of Diabetes', 'Smoking', 'Diet', 'Exercise',
       'Diagnosis'],
      dtype='object')
Before SMOTE:
Diagnosis
No     77
Yes    25
Name: count, dtype: int64

After SMOTE:
Diagnosis
No     77
```

```
Before SMOTE:
Diagnosis
No     77
Yes    25
Name: count, dtype: int64

After SMOTE:
Diagnosis
No     77
Yes    77
Name: count, dtype: int64
```

]: