Data Wrangling II

Create an “Academic performance” dataset of students and perform the following operations using Python.

1. Scan all variables for missing values and inconsistencies. If there are missing values and/or inconsistencies, use any of the suitable techniques to deal with them.
2. Scan all numeric variables for outliers. If there are outliers, use any of the suitable techniques to deal with them.
3. Apply data transformations on at least one of the variables. The purpose of this transformation should be one of the following reasons: to change the scale for better understanding of the variable, to convert a non-linear relation into a linear one, or to decrease the skewness and convert the distribution into a normal distribution.

IMPORTING ALL REQUIRED LIBRARIES

```
import pandas as pd
```

```
from google.colab import files
files.upload()
```

Choose files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving om2.csv to om2.csv
{'om2.csv': b'Roll
No.Name.DSBDA.AI.WT.CC.TOTAL.PERCENTAGE.RESULT\r\n1.Om.44.47.65.27.183.45.75.PASS\r\r
```

Load the Dataset into pandas dataframe.

```
om=pd.read_csv("/content/om2.csv")
```

```
om
```

	Roll No	Name	DSBDA	AI	WT	CC	TOTAL	PERCENTAGE	RESULT
0	1	Om	44	47.0	65.0	27.0	183	45.75	PASS
1	2	NaN	47	45.0	53.0	41.0	186	46.50	PASS
2	3	NaN	70	70.0	75.0	50.0	265	66.25	PASS

Data Preprocessing

```
om.isnull()
```

	Roll No	Name	DSBDA	AI	WT	CC	TOTAL	PERCENTAGE	RESULT
0	False	False	False	False	False	False	False	False	False
1	False	True	False	False	False	False	False	False	False
2	False	True	False	False	False	False	False	False	False
3	False	True	False	True	False	False	False	False	False
4	False	True	False	False	False	False	False	False	False
5	False	True	False	False	False	True	False	False	False
6	False	True	False	False	False	False	False	False	False
7	False	True	False	False	True	False	False	False	False
8	False	True	False	False	False	False	False	False	False
9	False	True	False	False	False	False	False	False	False
10	False	True	False	False	False	False	False	False	False
11	False	True	False	True	False	False	False	False	False
12	False	True	False	False	False	False	False	False	False
13	False	True	False	False	False	False	False	False	False
14	False	True	False	False	False	False	False	False	False
15	False	True	False	False	False	False	False	False	False
16	False	True	False	False	True	False	False	False	False
17	False	True	False	False	False	False	False	False	False
18	False	True	False	False	False	False	False	False	False
19	False	True	False	False	False	False	False	False	False
20	False	True	False	False	False	False	False	False	False
21	False	True	False	False	False	False	False	False	False
22	False	True	False	False	False	False	False	False	False
23	False	True	False	False	False	False	False	False	False
24	False	True	False	False	False	False	False	False	False
25	False	True	False	False	False	False	False	False	False
26	False	True	False	False	False	False	False	False	False
27	False	True	False	False	False	False	False	False	False
28	False	True	False	False	False	False	False	False	False
29	False	True	False	False	False	False	False	False	False

CHECKING FOR NULL VALUES

```
pd.isnull(om['WT'])
```

```
0     False
1     False
2     False
3     False
4     False
5     False
6     False
7      True
8     False
9     False
10    False
11    False
```

```
12    False
13    False
14    False
15    False
16     True
17    False
18    False
19    False
20    False
21    False
22    False
23    False
24    False
25    False
26    False
27    False
28    False
29    False
Name: WT, dtype: bool
```

```
om.notnull()
```

	Roll No	Name	DSBDA	AI	WT	CC	TOTAL	PERCENTAGE	RESULT
0	True	True	True	True	True	True	True	True	True
1	True	False	True	True	True	True	True	True	True
2	True	False	True	True	True	True	True	True	True
3	True	False	True	False	True	True	True	True	True
4	True	False	True	True	True	True	True	True	True
5	True	False	True	True	True	False	True	True	True
6	True	False	True	True	True	True	True	True	True
7	True	False	True	True	False	True	True	True	True
8	True	False	True	True	True	True	True	True	True
9	True	False	True	True	True	True	True	True	True
10	True	False	True	True	True	True	True	True	True
11	True	False	True	False	True	True	True	True	True
12	True	False	True	True	True	True	True	True	True
13	True	False	True	True	True	True	True	True	True
14	True	False	True	True	True	True	True	True	True
15	True	False	True	True	True	True	True	True	True
16	True	False	True	True	False	True	True	True	True
17	True	False	True	True	True	True	True	True	True
18	True	False	True	True	True	True	True	True	True
19	True	False	True	True	True	True	True	True	True
20	True	False	True	True	True	True	True	True	True
21	True	False	True	True	True	True	True	True	True
22	True	False	True	True	True	True	True	True	True
23	True	False	True	True	True	True	True	True	True
24	True	False	True	True	True	True	True	True	True
25	True	False	True	True	True	True	True	True	True
26	True	False	True	True	True	True	True	True	True
27	True	False	True	True	True	True	True	True	True
28	True	False	True	True	True	True	True	True	True
29	True	False	True	True	True	True	True	True	True

FILLING NULL VALUES

```
om.fillna(0)
```

	Roll No	Name	DSBDA	AI	WT	CC	TOTAL	PERCENTAGE	RESULT
0	1	Om	44	47.0	65.0	27.0	183	45.75	PASS
1	2	0	47	45.0	53.0	41.0	186	46.50	PASS
2	3	0	70	70.0	75.0	50.0	265	66.25	PASS
3	4	0	58	0.0	42.0	34.0	134	33.50	FAIL
4	5	0	56	40.0	41.0	78.0	215	53.75	PASS
5	6	0	29	32.0	69.0	0.0	130	32.50	FAIL
6	7	0	68	33.0	51.0	42.0	194	48.50	PASS
7	8	0	29	67.0	0.0	58.0	154	38.50	FAIL
8	9	0	43	63.0	59.0	55.0	220	55.00	PASS
9	10	0	45	78.0	55.0	50.0	228	57.00	PASS
10	11	0	68	67.0	72.0	38.0	245	61.25	PASS
11	12	0	24	0.0	58.0	58.0	140	35.00	FAIL
12	13	0	38	67.0	43.0	77.0	225	56.25	PASS
13	14	0	75	54.0	44.0	42.0	215	53.75	PASS
14	15	0	41	70.0	39.0	63.0	213	53.25	PASS
15	16	0	27	55.0	48.0	74.0	204	51.00	PASS
16	17	0	33	63.0	0.0	68.0	164	41.00	PASS
17	18	0	34	61.0	63.0	29.0	187	46.75	PASS
18	19	0	28	29.0	41.0	26.0	124	31.00	FAIL
19	20	0	37	67.0	76.0	23.0	203	50.75	PASS
20	21	0	41	31.0	34.0	55.0	161	40.25	PASS
21	22	0	69	45.0	77.0	59.0	250	62.50	PASS
22	23	0	73	47.0	39.0	55.0	214	53.50	PASS
23	24	0	44	50.0	51.0	65.0	210	52.50	PASS
24	25	0	38	43.0	61.0	52.0	194	48.50	PASS
25	26	0	62	48.0	42.0	68.0	220	55.00	PASS
26	27	0	41	74.0	37.0	40.0	192	48.00	PASS
27	28	0	64	33.0	44.0	60.0	201	50.25	PASS
28	29	0	67	45.0	38.0	61.0	211	52.75	PASS

```
cols_with_na=[]
for col in om.columns:
  if om[col].isna().any():
    cols_with_na.append(col)
cols_with_na
```

```
['Name', 'AI', 'WT', 'CC']
```

```
om.fillna(1)
```

	Roll No	Name	DSBDA	AI	WT	CC	TOTAL	PERCENTAGE	RESULT
0	1	Om	44	47.000000	65.000000	27.000000	183	45.75	PASS
1	2	1	47	45.000000	53.000000	41.000000	186	46.50	PASS
2	3	1	70	70.000000	75.000000	50.000000	265	66.25	PASS
3	4	1	58	53.535714	42.000000	34.000000	134	33.50	FAIL
4	5	1	56	40.000000	41.000000	78.000000	215	53.75	PASS
5	6	1	29	32.000000	69.000000	51.551724	130	32.50	FAIL
6	7	1	68	33.000000	51.000000	42.000000	194	48.50	PASS
7	8	1	29	67.000000	52.428571	58.000000	154	38.50	FAIL
8	9	1	43	63.000000	59.000000	55.000000	220	55.00	PASS
9	10	1	45	78.000000	55.000000	50.000000	228	57.00	PASS
10	11	1	68	67.000000	72.000000	38.000000	245	61.25	PASS
11	12	1	24	53.535714	58.000000	58.000000	140	35.00	FAIL
12	13	1	38	67.000000	43.000000	77.000000	225	56.25	PASS
13	14	1	75	54.000000	44.000000	42.000000	215	53.75	PASS
14	15	1	41	70.000000	39.000000	63.000000	213	53.25	PASS
15	16	1	27	55.000000	48.000000	74.000000	204	51.00	PASS
16	17	1	33	63.000000	52.428571	68.000000	164	41.00	PASS
18	19	1	28	29.000000	41.000000	26.000000	124	31.00	FAIL
20	21	1	41	31.000000	34.000000	55.000000	161	40.25	PASS
23	24	1	44	50.000000	51.000000	65.000000	210	52.50	PASS
25	26	1	62	48.000000	42.000000	68.000000	220	55.00	PASS
28	29	1	67	45.000000	38.000000	61.000000	211	52.75	PASS

FILLING NULL VALUES WITH MEAN

```
om['DSBDA']=om['DSBDA'].fillna(om['DSBDA'].mean())
```

```
om['WT']=om['WT'].fillna(om['WT'].mean())
```

```
 om['CC']=om['CC'].fillna(om['CC'].mean())
```

```
om['AI']=om['AI'].fillna(om['AI'].mean())
```

```
om['PERCENTAGE']=om['PERCENTAGE'].fillna(om['PERCENTAGE'].mean())
```

```
om['TOTAL']=om['TOTAL'].fillna(om['TOTAL'].mean())
```

```
om
```

	Roll No	Name	DSBDA	AI	WT	CC	TOTAL	PERCENTAGE	RESULT
0	1	Om	44	47.000000	65.000000	27.000000	183	45.75	PASS
1	2	NaN	47	45.000000	53.000000	41.000000	186	46.50	PASS
2	3	NaN	70	70.000000	75.000000	50.000000	265	66.25	PASS
3	4	NaN	58	53.535714	42.000000	34.000000	134	33.50	FAIL
4	5	NaN	56	40.000000	41.000000	78.000000	215	53.75	PASS
5	6	NaN	29	32.000000	69.000000	51.551724	130	32.50	FAIL
6	7	NaN	68	33.000000	51.000000	42.000000	194	48.50	PASS
7	8	NaN	29	67.000000	52.428571	58.000000	154	38.50	FAIL
8	9	NaN	43	63.000000	59.000000	55.000000	220	55.00	PASS
9	10	NaN	45	78.000000	55.000000	50.000000	228	57.00	PASS
10	11	NaN	68	67.000000	72.000000	38.000000	245	61.25	PASS
11	12	NaN	24	53.535714	58.000000	58.000000	140	35.00	FAIL
12	13	NaN	38	67.000000	43.000000	77.000000	225	56.25	PASS
13	14	NaN	75	54.000000	44.000000	42.000000	215	53.75	PASS

DATA VISUALIZATION

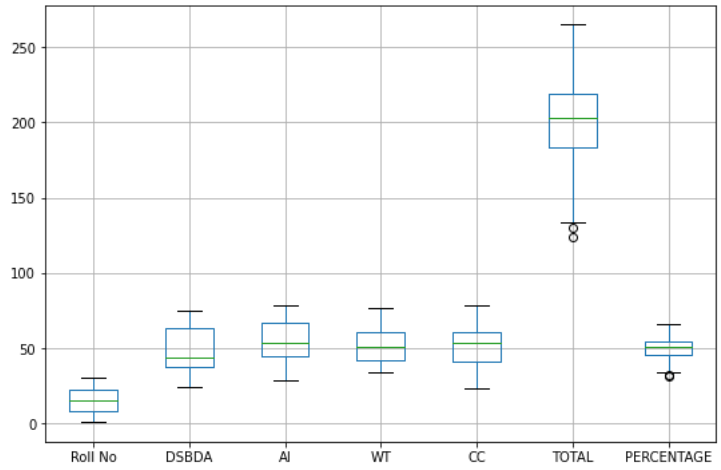
```
import seaborn as sns
import matplotlib.pyplot as plt
```

17	18	NaN	34	61.000000	63.000000	29.000000	187	46.75	PASS

CHECKING FOR OUTLIERS

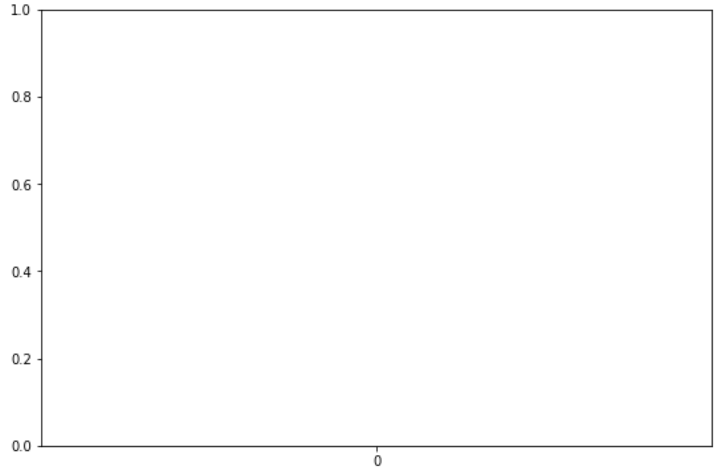
```
om.boxplot()
```

```
<AxesSubplot:>
```



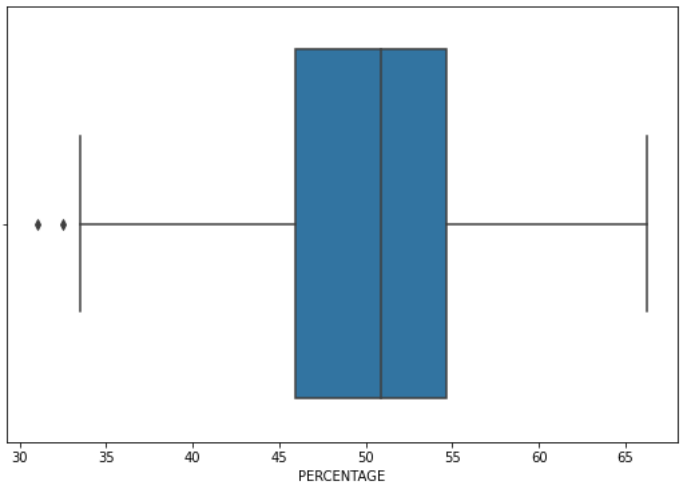

```
sns.boxplot()
```

```
<AxesSubplot:>
```



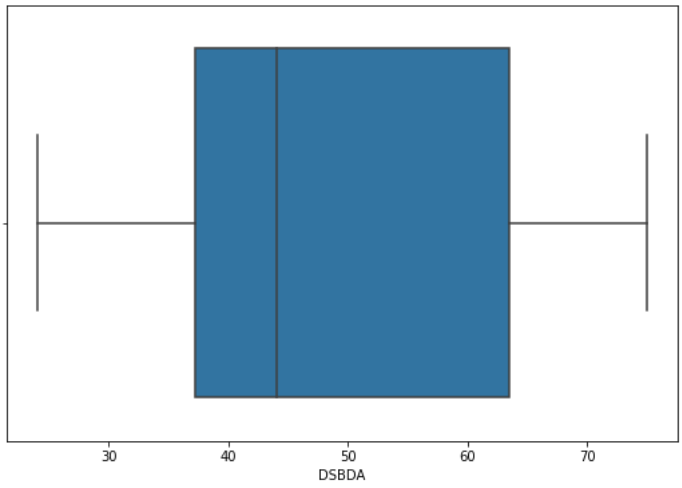
```
sns.boxplot(x=om.PERCENTAGE)
```

```
<AxesSubplot:xlabel='PERCENTAGE'>
```



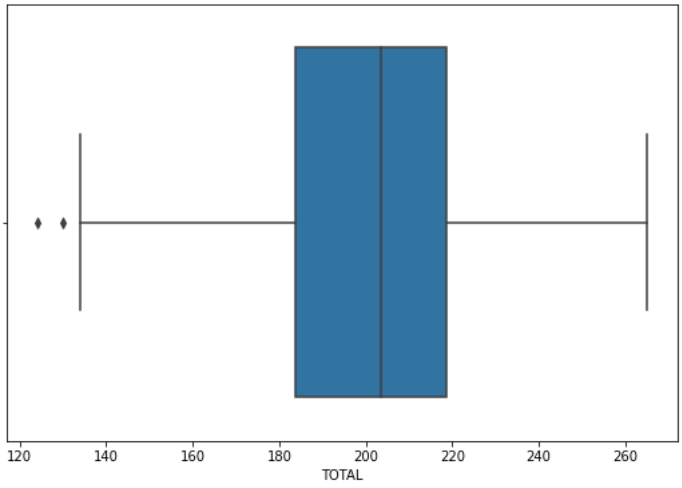

```
sns.boxplot(x=om.DSBDA)
```

```
<AxesSubplot:xlabel='DSBDA'>
```

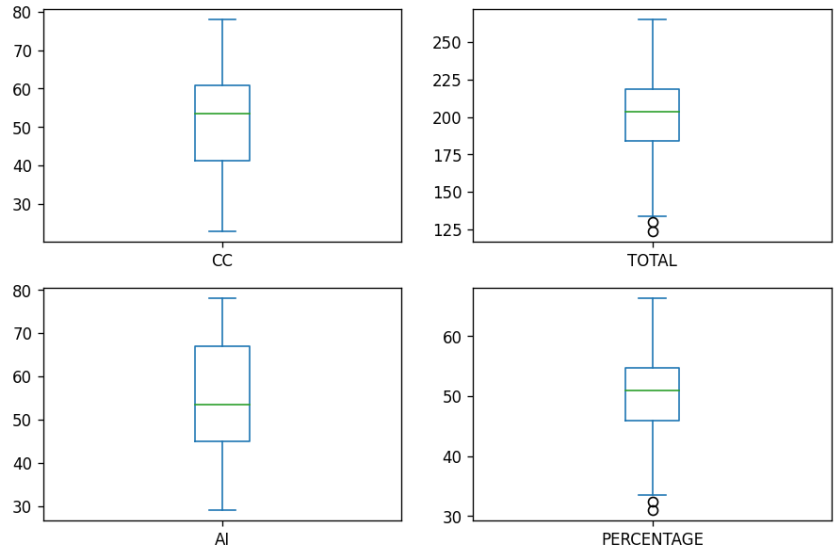



```
sns.boxplot(x=om.TOTAL)
```

```
<AxesSubplot:xlabel='TOTAL'>
```




```
plt.rcParams["figure.figsize"]=(9,6)
om_list=['CC','TOTAL','AI','PERCENTAGE']
fig,axes=plt.subplots(2,2)
fig.set_dpi(120)
count=0
for r in range(2):
  for c in range(2):
    _=om[om_list[count]].plot(kind='box',ax=axes[r,c])
    count+=1
```



REMOVING OUTLIERS

```
Q1=om['PERCENTAGE'].quantile(0.25)
Q3=om['PERCENTAGE'].quantile(0.75)
IQR=Q3-Q1
Lower_limit=Q1 - 1.5*IQR
Upper_limit=Q1 + 1.5*IQR
print(f'Q1={Q1},Q3={Q3},IQR={IQR},LOWER LIMIT={Lower_limit}, UPPER LIMIT={Upper_limit}')
```

```
Q1=45.9375,Q3=54.6875,IQR=8.75,LOWER LIMIT=32.8125, UPPER LIMIT=59.0625
```

```
om[(om['PERCENTAGE']<Lower_limit)| (om['PERCENTAGE']>Upper_limit)]
```

	Roll No	Name	DSBDA	AI	WT	CC	TOTAL	PERCENTAGE	RESULT
2	3	NaN	70	70.0	75.0	50.000000	265	66.25	PASS
5	6	NaN	29	32.0	69.0	51.551724	130	32.50	FAIL
10	11	NaN	68	67.0	72.0	38.000000	245	61.25	PASS
18	19	NaN	28	29.0	41.0	26.000000	124	31.00	FAIL
21	22	NaN	69	45.0	77.0	59.000000	250	62.50	PASS

```
Q1=om['TOTAL'].quantile(0.25)
Q3=om['TOTAL'].quantile(0.75)
IQR=Q3-Q1
Lower_limit=Q1 - 1.5*IQR
Upper_limit=Q1 + 1.5*IQR
print(f'Q1={Q1},Q3={Q3},IQR={IQR},LOWER LIMIT={Lower_limit}, UPPER LIMIT={Upper_limit}')
```

```
Q1=183.75,Q3=218.75,IQR=35.0,LOWER LIMIT=131.25, UPPER LIMIT=236.25
```

```
om[(om['TOTAL']<Lower_limit)| (om['TOTAL']>Upper_limit)]
```

	Roll No	Name	DSBDA	AI	WT	CC	TOTAL	PERCENTAGE	RESULT
2	3	NaN	70	70.0	75.0	50.000000	265	66.25	PASS
5	6	NaN	29	32.0	69.0	51.551724	130	32.50	FAIL
10	11	NaN	68	67.0	72.0	38.000000	245	61.25	PASS
18	19	NaN	28	29.0	41.0	26.000000	124	31.00	FAIL
21	22	NaN	69	45.0	77.0	59.000000	250	62.50	PASS

```
outliers=[]
for i in om.PERCENTAGE:
  if i<Lower_limit or i>Upper_limit:
    outliers.append(i)
print("outliers are",outliers)
```

```
outliers are [45.75, 46.5, 66.25, 33.5, 53.75, 32.5, 48.5, 38.5, 55.0, 57.0, 61.25, 35.0, 56.25, 53.75, 53.25, 51.0, 41.0, 46.75, 3
```

```
outliers=[]
for i in om.TOTAL:
  if i<Lower_limit or i>Upper_limit:
    outliers.append(i)
print("outliers are",outliers)
```

```
outliers are [265, 130, 245, 124, 250]
```

```
Upper_limit
```

```
236.25
```

```
Lower_limit
```

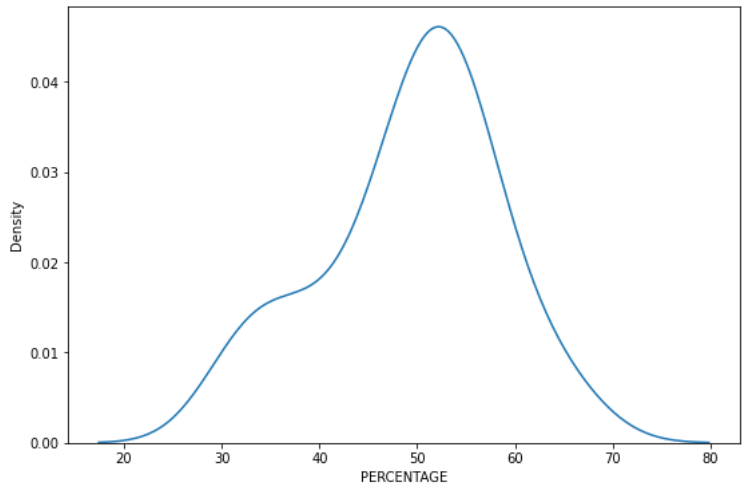
```
131.25
```

```
om2=om[om.PERCENTAGE<Lower_limit]
om2
```

	Roll No	Name	DSBDA	AI	WT	CC	TOTAL	PERCENTAGE	RESULT
0	1	Om	44	47.000000	65.000000	27.000000	183	45.75	PASS
1	2	NaN	47	45.000000	53.000000	41.000000	186	46.50	PASS
2	3	NaN	70	70.000000	75.000000	50.000000	265	66.25	PASS
3	4	NaN	58	53.535714	42.000000	34.000000	134	33.50	FAIL
4	5	NaN	56	40.000000	41.000000	78.000000	215	53.75	PASS
5	6	NaN	29	32.000000	69.000000	51.551724	130	32.50	FAIL
6	7	NaN	68	33.000000	51.000000	42.000000	194	48.50	PASS
7	8	NaN	29	67.000000	52.428571	58.000000	154	38.50	FAIL
8	9	NaN	43	63.000000	59.000000	55.000000	220	55.00	PASS
9	10	NaN	45	78.000000	55.000000	50.000000	228	57.00	PASS
10	11	NaN	68	67.000000	72.000000	38.000000	245	61.25	PASS
11	12	NaN	24	53.535714	58.000000	58.000000	140	35.00	FAIL
12	13	NaN	38	67.000000	43.000000	77.000000	225	56.25	PASS
13	14	NaN	75	54.000000	44.000000	42.000000	215	53.75	PASS
14	15	NaN	41	70.000000	39.000000	63.000000	213	53.25	PASS
15	16	NaN	27	55.000000	48.000000	74.000000	204	51.00	PASS
16	17	NaN	33	63.000000	52.428571	68.000000	164	41.00	PASS
17	18	NaN	34	61.000000	63.000000	29.000000	187	46.75	PASS
18	19	NaN	28	29.000000	41.000000	26.000000	124	31.00	FAIL
19	20	NaN	37	67.000000	76.000000	23.000000	203	50.75	PASS
20	21	NaN	41	31.000000	34.000000	55.000000	161	40.25	PASS
21	22	NaN	69	45.000000	77.000000	59.000000	250	62.50	PASS
22	23	NaN	73	47.000000	39.000000	55.000000	214	53.50	PASS
23	24	NaN	44	50.000000	51.000000	65.000000	210	52.50	PASS
24	25	NaN	38	43.000000	61.000000	52.000000	194	48.50	PASS
25	26	NaN	62	48.000000	42.000000	68.000000	220	55.00	PASS
26	27	NaN	41	74.000000	37.000000	40.000000	192	48.00	PASS
27	28	NaN	64	33.000000	44.000000	60.000000	201	50.25	PASS
28	29	NaN	67	45.000000	38.000000	61.000000	211	52.75	PASS
29	30	NaN	58	75.000000	51.000000	47.000000	231	57.75	PASS

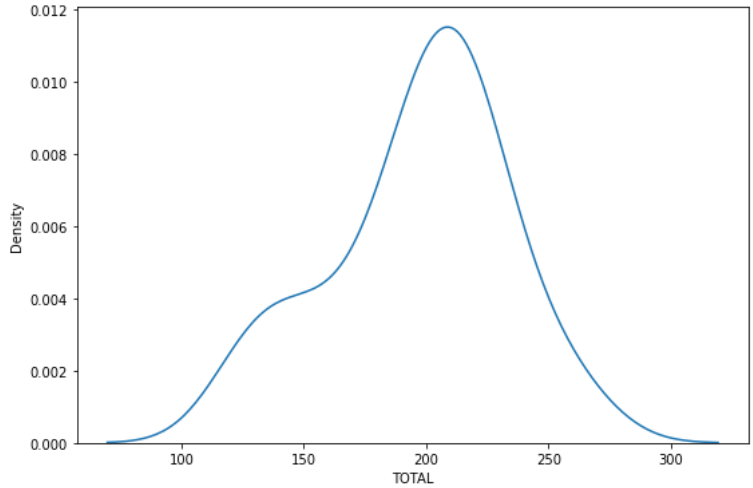
```
sns.kdeplot(om.PERCENTAGE)
```

```
<AxesSubplot:xlabel='PERCENTAGE', ylabel='Density'>
```



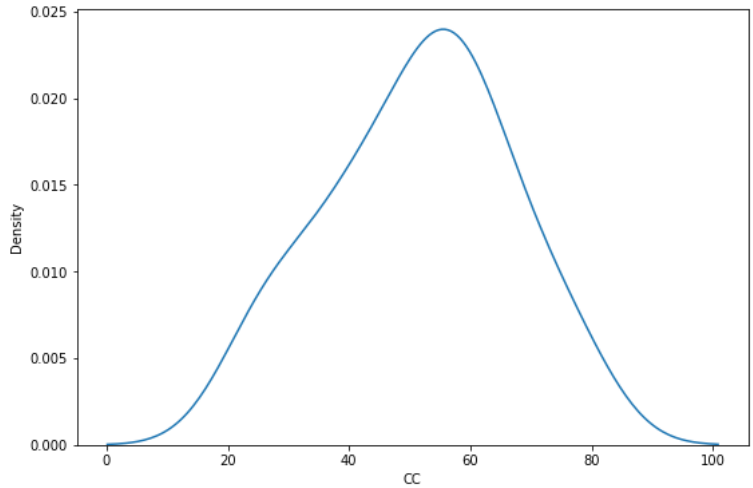

```
sns.kdeplot(om.TOTAL)
```

```
<AxesSubplot:xlabel='TOTAL', ylabel='Density'>
```



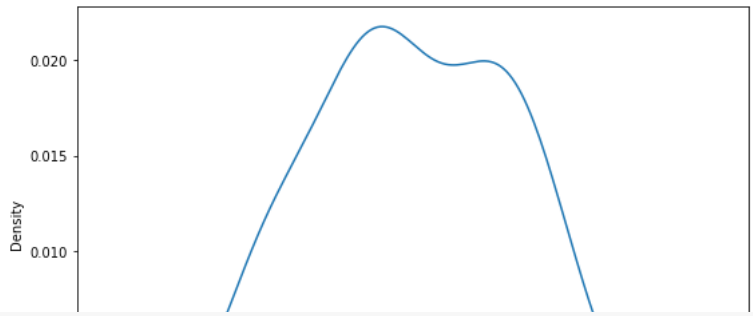

```
sns.kdeplot(om.CC)
```

```
<AxesSubplot:xlabel='CC', ylabel='Density'>
```



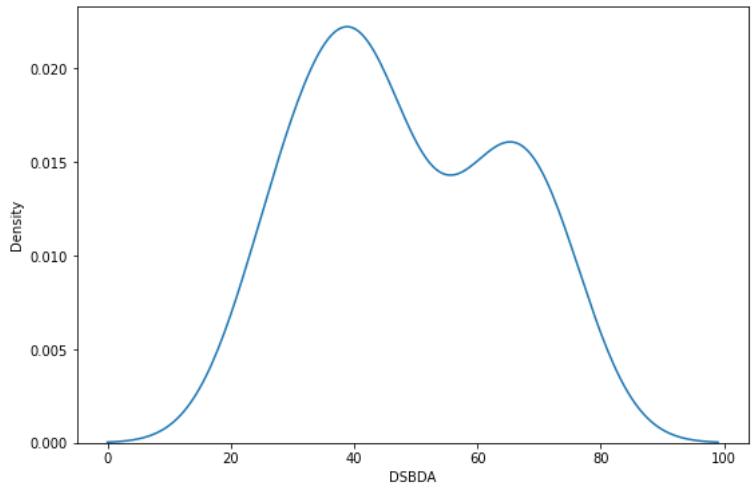

```
sns.kdeplot(om.AI)
```

```
<AxesSubplot:xlabel='AI', ylabel='Density'>
```




```
sns.kdeplot(om.DSBDA)
```

```
<AxesSubplot:xlabel='DSBDA', ylabel='Density'>
```




```
om.PERCENTAGE
```

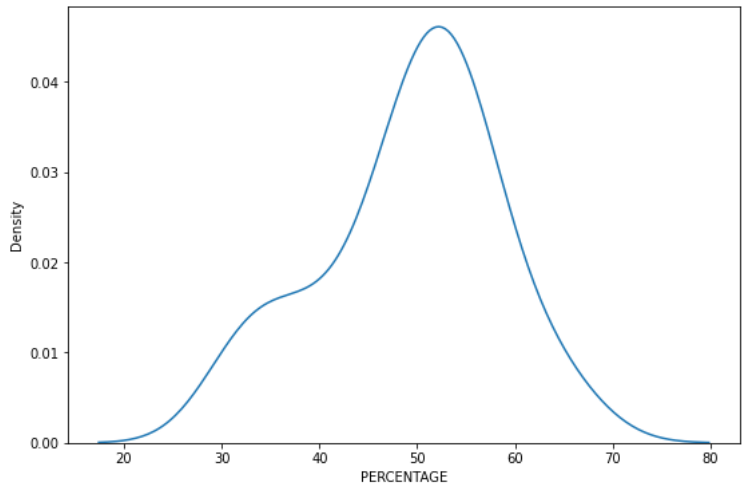
```
0     45.75
1     46.50
2     66.25
3     33.50
4     53.75
5     32.50
6     48.50
7     38.50
8     55.00
9     57.00
10    61.25
11    35.00
12    56.25
13    53.75
14    53.25
15    51.00
16    41.00
17    46.75
18    31.00
19    50.75
20    40.25
21    62.50
22    53.50
23    52.50
24    48.50
25    55.00
26    48.00
27    50.25
28    52.75
29    57.75
Name: PERCENTAGE, dtype: float64
```

```
om.skew()
```

```
<ipython-input-154-54adde609add>:1: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None')
  om.skew()
Roll No       0.000000
DSBDA         0.223449
AI           -0.072577
WT            0.559667
CC           -0.184889
TOTAL        -0.475043
PERCENTAGE   -0.475043
dtype: float64
```

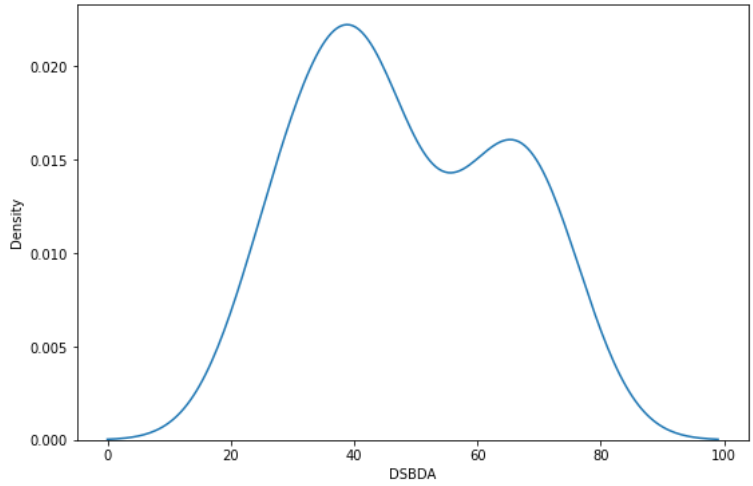
```
sns.kdeplot(om.PERCENTAGE)
```

```
<AxesSubplot:xlabel='PERCENTAGE', ylabel='Density'>
```



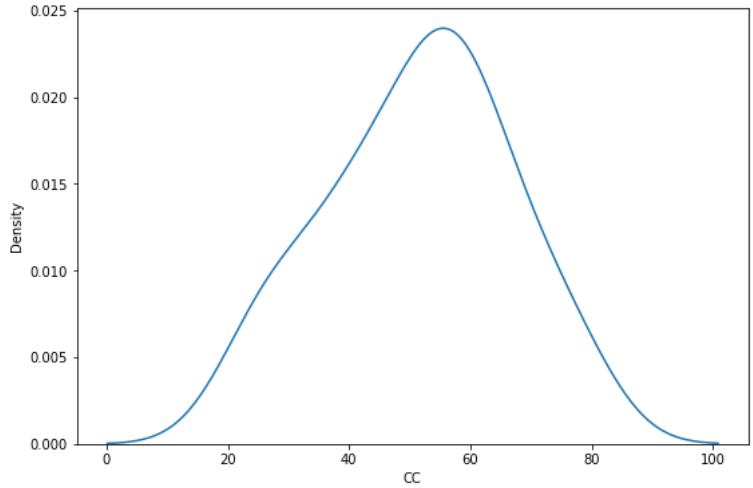

```
sns.kdeplot(om.DSBDA)
```

```
<AxesSubplot:xlabel='DSBDA', ylabel='Density'>
```




```
sns.kdeplot(om.CC)
```

```
<AxesSubplot:xlabel='CC', ylabel='Density'>
```



Descriptive Statistics -

Measures of Central Tendency and variability Perform the following operations on any open source dataset (e.g., data.csv)

1. Provide summary statistics (mean, median, minimum, maximum, standard deviation) for a dataset (age, income etc.) with numeric variables grouped by one of the qualitative (categorical) variable. For example, if your categorical variable is age groups and quantitative variable is income, then provide summary statistics of income grouped by the age groups. Create a list that contains a numeric value for each response to the categorical variable.
2. Write a Python program to display some basic statistical details like percentile, mean, standard deviation etc. of the species of 'Iris-setosa', 'Iris-versicolor' and 'Iris-versicolor' of iris.csv dataset

IMPORTING ALL REQUIRED LIBRARIES

```
import pandas as pd
```

```
from google.colab import files
files.upload()
```

Choose files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving Iris.csv to Iris.csv
{'Iris.csv':
b'Id,SepalLengthCm,SepalWidthCm,PetalLengthCm,PetalWidthCm,Species\n1,5.1,3.5,1.4,0.2
setosa\n2,4.9,3.0,1.4,0.2,Iris-setosa\n3,4.7,3.2,1.3,0.2,Iris-
setosa\n4,4.6,3.1,1.5,0.2,Iris-setosa\n5,5.0,3.6,1.4,0.2,Iris-
setosa\n6,5.4,3.9,1.7,0.4,Iris-setosa\n7,4.6,3.4,1.4,0.3,Iris-
setosa\n8,5.0,3.4,1.5,0.2,Iris-setosa\n9,4.4,2.9,1.4,0.2,Iris-
setosa\n10,4.9,3.1,1.5,0.1,Iris-setosa\n11,5.4,3.7,1.5,0.2,Iris-
setosa\n12,4.8,3.4,1.6,0.2,Iris-setosa\n13,4.8,3.0,1.4,0.1,Iris-
setosa\n14,4.3,3.0,1.1,0.1,Iris-setosa\n15,5.8,4.0,1.2,0.2,Iris-
setosa\n16,5.7,4.4,1.5,0.4,Iris-setosa\n17,5.4,3.9,1.3,0.4,Iris-
setosa\n18,5.1,3.5,1.4,0.3,Iris-setosa\n19,5.7,3.8,1.7,0.3,Iris-
setosa\n20,5.1,3.8,1.5,0.3,Iris-setosa\n21,5.4,3.4,1.7,0.2,Iris-
setosa\n22,5.1,3.7,1.5,0.4,Iris-setosa\n23,4.6,3.6,1.0,0.2,Iris-
setosa\n24,5.1,3.3,1.7,0.5,Iris-setosa\n25,4.8,3.4,1.9,0.2,Iris-
setosa\n26,5.0,3.0,1.6,0.2,Iris-setosa\n27,5.0,3.4,1.6,0.4,Iris-
setosa\n28,5.2,3.5,1.5,0.2,Iris-setosa\n29,5.2,3.4,1.4,0.2,Iris-
setosa\n30,4.7,3.2,1.6,0.2,Iris-setosa\n31,4.8,3.1,1.6,0.2,Iris-
setosa\n32,5.4,3.4,1.5,0.4,Iris-setosa\n33,5.2,4.1,1.5,0.1,Iris-
setosa\n34,5.5,4.2,1.4,0.2,Iris-setosa\n35,4.9,3.1,1.5,0.1,Iris-
setosa\n36,5.0,3.2,1.2,0.2,Iris-setosa\n37,5.5,3.5,1.3,0.2,Iris-
setosa\n38,4.9,3.1,1.5,0.1,Iris-setosa\n39,4.4,3.0,1.3,0.2,Iris-
setosa\n40,5.1,3.4,1.5,0.2,Iris-setosa\n41,5.0,3.5,1.3,0.3,Iris-
setosa\n42,4.5,2.3,1.3,0.3,Iris-setosa\n43,4.4,3.2,1.3,0.2,Iris-
setosa\n44,5.0,3.5,1.6,0.6,Iris-setosa\n45,5.1,3.8,1.9,0.4,Iris-
setosa\n46,4.8,3.0,1.4,0.3,Iris-setosa\n47,5.1,3.8,1.6,0.2,Iris-
setosa\n48,4.6,3.2,1.4,0.2,Iris-setosa\n49,5.3,3.7,1.5,0.2,Iris-
setosa\n50,5.0,3.3,1.4,0.2,Iris-setosa\n51,7.0,3.2,4.7,1.4,Iris-
versicolor\n52,6.4,3.2,4.5,1.5,Iris-versicolor\n53,6.9,3.1,4.9,1.5,Iris-
versicolor\n54,5.5,2.3,4.0,1.3,Iris-versicolor\n55,6.5,2.8,4.6,1.5,Iris-
versicolor\n56,5.7,2.8,4.5,1.3,Iris-versicolor\n57,6.3,3.3,4.7,1.6,Iris-
versicolor\n58,4.9,2.4,3.3,1.0,Iris-versicolor\n59,6.6,2.9,4.6,1.3,Iris-
versicolor\n60,5.2,2.7,3.9,1.4,Iris-versicolor\n61,5.0,2.0,3.5,1.0,Iris-
versicolor\n62,5.9,3.0,4.2,1.5,Iris-versicolor\n63,6.0,2.2,4.0,1.0,Iris-
versicolor\n64,6.1,2.9,4.7,1.4,Iris-versicolor\n65,5.6,2.9,3.6,1.3,Iris-
versicolor\n66,6.7,3.1,4.4,1.4,Iris-versicolor\n67,5.6,3.0,4.5,1.5,Iris-
```

LOADING DATASET

```
iris=pd.read_csv("/content/Iris.csv")
```

```
iris
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

DATA PREPROCESSING

```
iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             150 non-null    int64
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

```
iris.describe(include='all')
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Speci
count	150.000000	150.000000	150.000000	150.000000	150.000000	
unique	NaN	NaN	NaN	NaN	NaN	
top	NaN	NaN	NaN	NaN	NaN	I set
freq	NaN	NaN	NaN	NaN	NaN	
mean	75.500000	5.843333	3.054000	3.758667	1.198667	N
std	43.445368	0.828066	0.433594	1.764420	0.763161	N
min	1.000000	4.300000	2.000000	1.000000	0.100000	N
25%	38.250000	5.100000	2.800000	1.600000	0.300000	N
50%	75.500000	5.800000	3.000000	4.350000	1.300000	N
75%	112.750000	6.400000	3.300000	5.100000	1.800000	N

```
iris.head(10)
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
5	6	5.4	3.9	1.7	0.4	Iris-setosa
6	7	4.6	3.4	1.4	0.3	Iris-setosa
7	8	5.0	3.4	1.5	0.2	Iris-setosa
8	9	4.4	2.9	1.4	0.2	Iris-setosa
9	10	4.9	3.1	1.5	0.1	Iris-setosa

MEAN

```
iris.mean()
```

```
<ipython-input-12-7eed97565d6e>:1: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') i
  iris.mean()
Id               75.500000
SepalLengthCm     5.843333
SepalWidthCm      3.054000
PetalLengthCm     3.758667
PetalWidthCm      1.198667
dtype: float64
```

```
iris.loc[:,'PetalLengthCm'].mean()
```

```
3.758666666666666
```

```
iris.mean(axis=1)[0:3]
```

```
<ipython-input-15-ac32167c8676>:1: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') i
  iris.mean(axis=1)[0:3]
0    2.24
1    2.30
2    2.48
dtype: float64
```

MEDIAN

```
iris.median()
```

```
<ipython-input-16-7f9b4600b8ea>:1: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') i
  iris.median()
Id               75.50
SepalLengthCm     5.80
SepalWidthCm      3.00
PetalLengthCm     4.35
PetalWidthCm      1.30
dtype: float64
```

```
iris.loc[:,'SepalWidthCm'].median()
```

```
3.0
```

MODE

```
iris.mode()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.0	3.0	1.5	0.2	Iris-setosa
1	2	NaN	NaN	NaN	NaN	Iris-versicolor
2	3	NaN	NaN	NaN	NaN	Iris-virginica
3	4	NaN	NaN	NaN	NaN	NaN
4	5	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...
145	146	NaN	NaN	NaN	NaN	NaN
146	147	NaN	NaN	NaN	NaN	NaN
147	148	NaN	NaN	NaN	NaN	NaN
148	149	NaN	NaN	NaN	NaN	NaN
149	150	NaN	NaN	NaN	NaN	NaN

150 rows × 6 columns

```
iris.loc[:,'SepalWidthCm'].mode()
```

```
0    3.0
dtype: float64
```

MIN AND MAX

```
iris.min()
```

```
Id                      1
SepalLengthCm         4.3
SepalWidthCm          2.0
PetalLengthCm         1.0
PetalWidthCm          0.1
Species       Iris-setosa
dtype: object
```

```
iris.max()
```

```
Id                         150
SepalLengthCm              7.9
SepalWidthCm               4.4
PetalLengthCm              6.9
PetalWidthCm               2.5
Species          Iris-virginica
dtype: object
```

STANDARD DEVIATION

```
iris.std()
```

```
<ipython-input-22-c5ab3f85284a>:1: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') i
  iris.std()
Id               43.445368
SepalLengthCm     0.828066
SepalWidthCm      0.433594
PetalLengthCm     1.764420
PetalWidthCm      0.763161
dtype: float64
```

```
iris.std(axis=1)[0:3]
```

```
<ipython-input-23-3fedfc1ee20e>:1: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') i
  iris.std(axis=1)[0:3]
0    2.010721
1    1.772005
2    1.754138
dtype: float64
```

PRINTING STATISTICAL DATA

```
grouped_data = iris.groupby('Species')
grouped_data
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7fe6d68bb730>
```

```
setosa_data = grouped_data.get_group('Iris-setosa')
versicolor_data = grouped_data.get_group('Iris-versicolor')
virginica_data = grouped_data.get_group('Iris-virginica')
```

```
print("Statistical details for Iris-setosa:")
print(setosa_data.describe())
print("\nStatistical details for Iris-versicolor:")
print(versicolor_data.describe())
print("\nStatistical details for Iris-virginica:")
print(virginica_data.describe())
```

```
Statistical details for Iris-setosa:
             Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
count  50.00000       50.00000     50.000000      50.000000      50.00000
mean   25.50000        5.00600      3.418000       1.464000       0.24400
std    14.57738        0.35249      0.381024       0.173511       0.10721
min     1.00000        4.30000      2.300000       1.000000       0.10000
25%    13.25000        4.80000      3.125000       1.400000       0.20000
50%    25.50000        5.00000      3.400000       1.500000       0.20000
75%    37.75000        5.20000      3.675000       1.575000       0.30000
max    50.00000        5.80000      4.400000       1.900000       0.60000

Statistical details for Iris-versicolor:
              Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
count   50.00000      50.000000     50.000000      50.000000     50.000000
mean    75.50000       5.936000      2.770000       4.260000      1.326000
std     14.57738       0.516171      0.313798       0.469911      0.197753
min     51.00000       4.900000      2.000000       3.000000      1.000000
25%     63.25000       5.600000      2.525000       4.000000      1.200000
50%     75.50000       5.900000      2.800000       4.350000      1.300000
```

```
75%     87.75000       6.300000      3.000000       4.600000      1.500000
max    100.00000       7.000000      3.400000       5.100000      1.800000

Statistical details for Iris-virginica:
                Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
count   50.00000       50.00000     50.000000      50.000000      50.00000
mean   125.50000        6.58800      2.974000       5.552000       2.02600
std     14.57738        0.63588      0.322497       0.551895       0.27465
min    101.00000        4.90000      2.200000       4.500000       1.40000
25%    113.25000        6.22500      2.800000       5.100000       1.80000
50%    125.50000        6.50000      3.000000       5.550000       2.00000
75%    137.75000        6.90000      3.175000       5.875000       2.30000
max    150.00000        7.90000      3.800000       6.900000       2.50000
```

```
print('Iris-setosa')
setosa = iris['Species'] == 'Iris-setosa'
print(iris[setosa].describe())
```

```
Iris-setosa
             Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
count  50.00000       50.00000     50.000000      50.000000      50.00000
mean   25.50000        5.00600      3.418000       1.464000       0.24400
std    14.57738        0.35249      0.381024       0.173511       0.10721
min     1.00000        4.30000      2.300000       1.000000       0.10000
25%    13.25000        4.80000      3.125000       1.400000       0.20000
50%    25.50000        5.00000      3.400000       1.500000       0.20000
75%    37.75000        5.20000      3.675000       1.575000       0.30000
max    50.00000        5.80000      4.400000       1.900000       0.60000
```

```
print('\nIris-versicolor')
versicolor = iris['Species'] == 'Iris-versicolor'
print(iris[versicolor].describe())
```

```
Iris-versicolor
                Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
count   50.00000      50.000000     50.000000      50.000000     50.000000
mean    75.50000       5.936000      2.770000       4.260000      1.326000
std     14.57738       0.516171      0.313798       0.469911      0.197753
min     51.00000       4.900000      2.000000       3.000000      1.000000
25%     63.25000       5.600000      2.525000       4.000000      1.200000
50%     75.50000       5.900000      2.800000       4.350000      1.300000
75%     87.75000       6.300000      3.000000       4.600000      1.500000
max    100.00000       7.000000      3.400000       5.100000      1.800000
```

```
print('\nIris-virginica')
virginica = iris['Species'] == 'Iris-virginica'
print(iris[virginica].describe())
```

```
Iris-virginica
                Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
count   50.00000       50.00000     50.000000      50.000000      50.00000
mean   125.50000        6.58800      2.974000       5.552000       2.02600
std     14.57738        0.63588      0.322497       0.551895       0.27465
min    101.00000        4.90000      2.200000       4.500000       1.40000
25%    113.25000        6.22500      2.800000       5.100000       1.80000
50%    125.50000        6.50000      3.000000       5.550000       2.00000
75%    137.75000        6.90000      3.175000       5.875000       2.30000
max    150.00000        7.90000      3.800000       6.900000       2.50000
```

```
import pandas as pd
```

```
from google.colab import files
files.upload()
```

Choose files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving dsbda A3.xlsx to dsbda A3.xlsx
{'dsbda A3.xlsx':
b'PK\x03\x04\x14\x00\x06\x00\x08\x00\x00\x00!\x00\x9e,lok\x01\x00\x00\x10\x05\x0
\xa2\x04\x02(\xa0\x00\x02\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
$\xee\x1a-
M\xa28\x1b\xdb\xdb\xe3fcB\xa8\xacB\xeb\xa5Q\x1b\xfb\xff\xbf\xb8v&\xb3]c\xb2-
\x04\xd4\xce\x96l\\\x8cX\x06V:\xa5\xed\xaad\xef\xcb\xe7\xfc\x9ee\x18\x85U\xc28\x
\x96\xac\x8e\xd1?
p\x8e\xb2\x86F`\xe1<X\xda\xa9\\hD\xa4\xd7\xb0\xe2^\xc8\xb5X\x01\xbf\x1d\x8d\xee\
\x9c[\x17\xe7E:
(]Ui\t\xca\xc9MC\x15(\xd0\x07\x10\nk\x80\xd8\x98"\xadE#\xb4\xfd\xe6>\xe3\x9f\x82
2p\x11N\xba\xe7\xfc\xa9o\x17\xc1y\xa4\t\x0e\xf0\x7f\x80\xef\x11m\xb3sOB\x10\xa2\
\xa2\x04\x02(\xa0\x00\x02\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
\x889L\xec\xd2\x99\x15\xc8sbg\xd9\xae|\xc8l!\xf5\xf9\x1aUSh9i\xb0b\x9er:"y_dl\xc
N\x9c\xc8R"4\x12\xf82\xcfG\xc7%\xa0\xf5\x7fZ\xb44\xf1\xcb\x9dy\xc47\t\xc3\xab\xc
\x03\x00\x00\x1a\x00\x08\x01xl/_rels/workbook.xml.rels
\xa2\x04\x01(\xa0\x00\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
?\xd3\x14\x87R\x81?\x94\xb7
\x8a\xa1\x8f\xce\xffk\xbb\xaaj4=;\xfd\xd1\x91\xe5\x0b\x162\xf0\xd0\xc6\x05D\x81\
\x06C\xc4\x13\xc7\xa9\x15\xe48Y\x84\xb9_\x13Fc\xab\x9f\x0c6v\x829\xb5\x96.r\xb7j
\x00\x00\x00\xff\xff\x03\x00PK\x03\x04\x14\x00\x06\x00\x08\x00\x00\x00!\x007\x93
\x10\xbc;Z"%\xb1
)Hl\x175P\x14A\x9a\xe5\xe2\x0bM\x8d,\xc2\\T\x92\xaa\x1d\x14\xfd\xf7\x8e\xa4\xaaM
{\xdd\t\x89\xabQ\x92\xc4!\r\xca\x9fV\xdcXRA\xcd:\xe9\xef\xd0\x84\t\x1e7\xa6\xa7c
\xd4\xffW\xbd\x06\xecEc\xd0\x1dr\x0b\x9f;a\x01\x8b\xa2\x97\xad\xcc\xb1e<c[w\xc3|
%6n\xefM\xbby&1{\xe9\xdf_\x88\xccx\xcf6@\xbacJc\xfc\'\xf52\xef\x0b\xf8A\xc0\xc1\
\x8a\xca7\x05\x8d\xc3d\x8e\xeb\xe3\xdc{\x10\xbb\xc6\x17t\x1e&\xe9p\xf73\xe8\xa1\
[}\xbf\xee\xdd\xa4\xc4f\x02\x03\xbb\xae\xa2\x01a:\xc6\x99\xe4hm\xdf\r\x1b\xd38\x
d47\x9b\xfe\x82>\xcb\x86Y\x7fg\x19\xdf\xe3\x0fr\x0b\xf55sX\xd8#!\xcc\x13\x8d\x98
fB\x0b\xbf\xc8\xd0\xba\xe5\xc20\x8b\xaa\xb7\xc4\x16\x06\x91\xe9Kk\xee\x06L\x9e\x
\x0csw\xfckfl/\xf1\x06{lg\xdb\xec&!\xeb\xa4\xe4\xa8\xb5e\x8f\xb2\x9a\x91\x19\xc9
-
Y\xc32\xa3\xf9\xf4\xf4\xe9\xbd7\xdf\x934\x17/\xdd\x8d\xa9s\x84SNX\xd2v\xab\x17*\
\x11\xe1\x18;\xd0?
\xe1[\xa8\xedFB\xcc\xb6\xcae>\x82f\xc4/\xb0\x19N\xe0\xd9\x84\xa51\x12p\x9bN\xcb\
/\x9f}\xff\xf2\xd9\x13\xe7\xe5\xb3\xc7\'\x0f\x9e\x9e<\xf8\xe9\xe4\xe1\xc3\x93\x0
f\xb6\x8c\x8e;
(\x99\xea\x1d_|\xfb\xd9\x9f_\x7f\xec\xfc\xf1\xe4\x9b\x17\x8f\xbe\xb0\xe3\xb9\x8e
[x\xe1\xf9\x97\x8f\x7f{\xfa\xf8\xf9W\x9f\xfe\xfe\xdd#\x0b\xbc\x93\xa2\x03\x1d>$1
j\xc3u\xb1\xe9\xbc[)\x08\x8c\rxy~\xc7\xe0\xba\x1f\xa5sA,#_\x89b\x03\xb8\xc7\x18\
```

LOADING DATASET

```
\xfb\x86\x9a\xc7$9S\xda0\x89\xba\xffN\xd4\xb3\xaat7\xd4:)\xb1\xbe7:\xa7\xa4|\x1
```

```
om=pd.read_excel("/content/dsbda A3.xlsx")
```

```
\x90\x95\xb7\x8d\x1c\x04\xdd\x0c2\x9eL\xf0W\xe8a\xd77\xa4\xa7\xb3[P\xf8L+\xacOU\
```

```
om
```

DATA PREPROCESSING

```
om.info
```

```
<bound method DataFrame.info of     Id  Age  Gender  Income
0    1   40    male   32879
1    2   32  Female   29976
2    3   47    male   39440
3    4   49  Female   39453
4    5   32  Female   24008
5    6   30  Female   22621
6    7   50    male   27621
7    8   40  Female   35625
8    9   45    male   29312
9   10   43  Female   36535
10  11   40  Female   25391
11  12   41    male   28793
12  13   39  Female   23070
13  14   32  Female   38285
14  15   35    male   34431
15  16   40  Female   23633
16  17   42    male   21205
17  18   46    male   21464
18  19   45  Female   23773
19  20   31    male   38252>
```

```
om.Id
```

```
0      1
1      2
2      3
3      4
4      5
5      6
6      7
7      8
8      9
9     10
10    11
11    12
12    13
13    14
14    15
15    16
16    17
17    18
18    19
19    20
Name: Id, dtype: int64
```

```
om.describe(include="all")
```

	Id	Age	Gender	Income
count	20.00000	20.000000	20	20.000000
unique	NaN	NaN	2	NaN
top	NaN	NaN	Female	NaN
freq	NaN	NaN	11	NaN
mean	10.50000	39.950000	NaN	29788.350000
std	5.91608	6.194012	NaN	6542.670912
min	1.00000	30.000000	NaN	21205.000000
25%	5.75000	34.250000	NaN	23738.000000
50%	10.50000	40.000000	NaN	29052.500000
75%	15.25000	45.000000	NaN	35852.500000
max	20.00000	50.000000	NaN	39453.000000

```
summary = om.groupby("Gender")["Income"].describe()
summary
```

	count	mean	std	min	25%	50%	75%	max

```
counts = om["Gender"].value_counts().tolist()
counts
```

```
[11, 9]
```

```
om.groupby("Gender")["Income"].mean()
```

```
Gender
Female    29306.363636
male      30377.444444
Name: Income, dtype: float64
```

```
om.groupby("Gender")["Income"].median()
```

```
Gender
Female    25391.0
male      29312.0
Name: Income, dtype: float64
```

```
om.groupby("Gender")["Income"].std()
```

```
Gender
Female    6826.354214
male      6535.044780
Name: Income, dtype: float64
```

```
om.groupby("Gender")["Income"].quantile(0.25)
```

```
Gender
Female    23703.0
male      27621.0
Name: Income, dtype: float64
```

```
om.groupby("Gender")["Income"].quantile(0.50)
```

```
Gender
Female    25391.0
male      29312.0
Name: Income, dtype: float64
```

```
om.groupby("Gender")["Income"].quantile(0.75)
```

```
Gender
Female    36080.0
male      34431.0
Name: Income, dtype: float64
```

```
om.groupby("Gender")["Income"].min()
```

```
Gender
Female    22621
male      21205
Name: Income, dtype: int64
```

```
om.groupby("Gender")["Income"].max()
```

```
Gender
Female    39453
male      39440
Name: Income, dtype: int64
```

Data Analytics I

Create a Linear Regression Model using Python/R to predict home prices using Boston Housing Dataset (https://www.kaggle.com/c/boston-housing). The Boston Housing dataset contains information about various houses in Boston through different parameters. There are 506 samples and 14 feature variables in this dataset.

IMPORTING REQUIRED LIBRARIES

```
import pandas as pd
```

```
from google.colab import files
files.upload()
```

LOADING DATASET

```
om=pd.read_csv("/content/HousingData.csv")
```

```
om
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	L
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	
...	...	...	...	...	...	...	...	...	...	...	...	...	
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1	273	21.0	391.99	
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1	273	21.0	396.90	
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1	273	21.0	396.90	
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1	273	21.0	393.45	
505	0.04741	0.0	11.93	0.0	0.573	6.030	NaN	2.5050	1	273	21.0	396.90	

506 rows × 14 columns

DATA PREPROCESSING

```
om.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   CRIM     486 non-null    float64
 1   ZN       486 non-null    float64
 2   INDUS    486 non-null    float64
 3   CHAS     486 non-null    float64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      486 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    int64  
 9   TAX      506 non-null    int64  
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    486 non-null    float64
 13  MEDV     506 non-null    float64
dtypes: float64(12), int64(2)
memory usage: 55.5 KB
```

```
om.describe()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	
count	486.000000	486.000000	486.000000	486.000000	506.000000	506.000000	486.00
mean	3.611874	11.211934	11.083992	0.069959	0.554695	6.284634	68.51
std	8.720192	23.388876	6.835896	0.255340	0.115878	0.702617	27.99
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.90
25%	0.081900	0.000000	5.190000	0.000000	0.449000	5.885500	45.17
50%	0.253715	0.000000	9.690000	0.000000	0.538000	6.208500	76.80
75%	3.560263	12.500000	18.100000	0.000000	0.624000	6.623500	93.97

```
om.isnull()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	E
0	False	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	..
501	False	False	False	False	False	False	False	False	False	False	False	False
502	False	False	False	False	False	False	False	False	False	False	False	False
503	False	False	False	False	False	False	False	False	False	False	False	False
504	False	False	False	False	False	False	False	False	False	False	False	False
505	False	False	False	False	False	False	True	False	False	False	False	False

506 rows × 14 columns

```
om.fillna(0)
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90
...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1	273	21.0	391.99
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1	273	21.0	396.90
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1	273	21.0	396.90
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1	273	21.0	393.45
505	0.04741	0.0	11.93	0.0	0.573	6.030	0.0	2.5050	1	273	21.0	396.90

506 rows × 14 columns

CHECKING FOR NULL VALUES

```
om.isnull().sum()
```

```
CRIM       20
ZN         20
INDUS      20
CHAS       20
NOX         0
RM          0
AGE        20
DIS         0
RAD         0
TAX         0
PTRATIO     0
B           0
LSTAT      20
```

```
MEDV        0
dtype: int64
```

FILLING NULL VALUES

```
om['CRIM']=om["CRIM"].fillna(om['CRIM'].mean())
```

```
om.isnull().sum()
```

```
CRIM        0
ZN         20
INDUS      20
CHAS       20
NOX         0
RM          0
AGE        20
DIS         0
RAD         0
TAX         0
PTRATIO     0
B           0
LSTAT      20
MEDV        0
dtype: int64
```

```
om['ZN']=om["ZN"].fillna(om['ZN'].mean())
```

```
om['INDUS']=om["INDUS"].fillna(om['INDUS'].mean())
```

```
om['CHAS']=om["CHAS"].fillna(om['CHAS'].mean())
```

```
om['AGE']=om["AGE"].fillna(om['AGE'].mean())
```

```
om['LSTAT']=om["LSTAT"].fillna(om['LSTAT'].mean())
```

```
om.isnull().sum()
```

```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
MEDV       0
dtype: int64
```

DATA VISUALIZATION

```
import seaborn as sns
import matplotlib.pyplot as plt
```

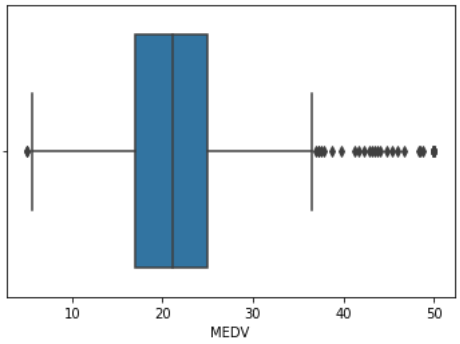
CHECKING FOR OUTLIERS

```
om.boxplot()
```

```
<Axes: >
```

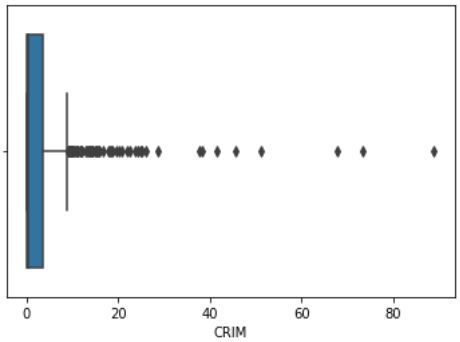
```
sns.boxplot(x=om.MEDV)
```

```
<Axes: xlabel='MEDV'>
```



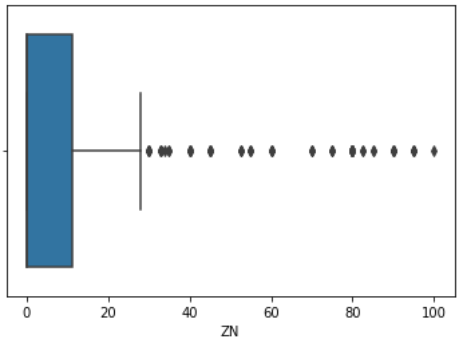

```
sns.boxplot(x=om.CRIM)
```

```
<Axes: xlabel='CRIM'>
```



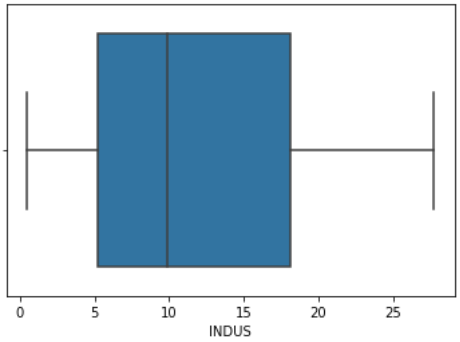

```
sns.boxplot(x=om.ZN)
```

```
<Axes: xlabel='ZN'>
```



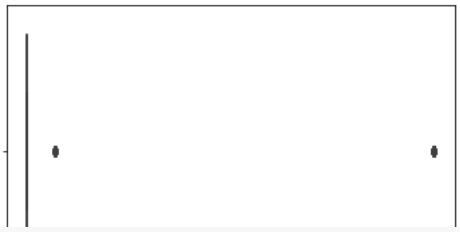

```
sns.boxplot(x=om.INDUS)
```

```
<Axes: xlabel='INDUS'>
```



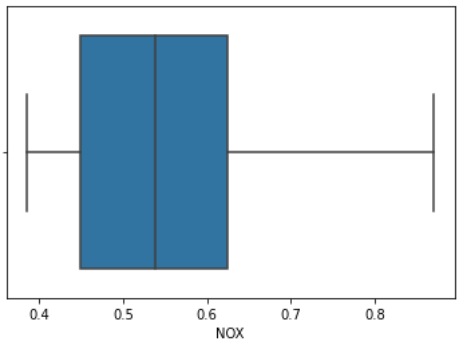

```
sns.boxplot(x=om.CHAS)
```

```
<Axes: xlabel='CHAS'>
```



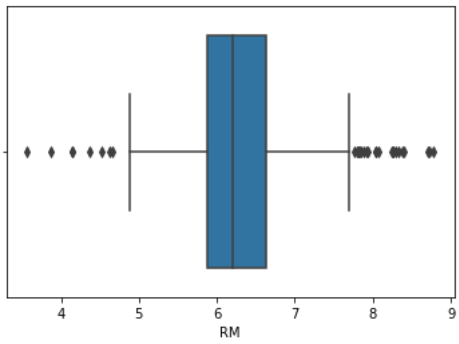
```
sns.boxplot(x=om.NOX)
```

```
<Axes: xlabel='NOX'>
```



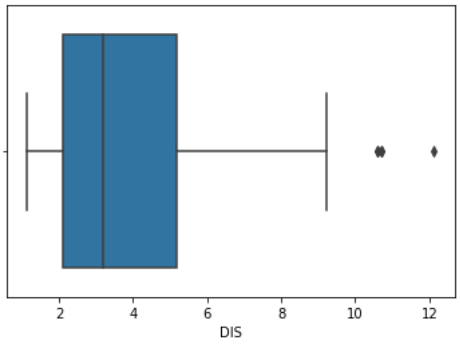

```
sns.boxplot(x=om.RM)
```

```
<Axes: xlabel='RM'>
```




```
sns.boxplot(x=om.DIS)
```

```
<Axes: xlabel='DIS'>
```

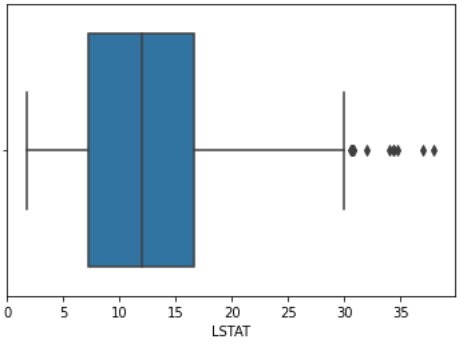



```
sns.boxplot(x=om.B)
```

```
<Axes: xlabel='B'>
```

```
sns.boxplot(x=om.LSTAT)
```

```
<Axes: xlabel='LSTAT'>
```



REMOVING OUTLIERS

```
Q1=om['MEDV'].quantile(0.25)
Q3=om['MEDV'].quantile(0.75)
IQR=Q3-Q1
Lower_limit=Q1 - 1.5*IQR
Upper_limit=Q1 + 1.5*IQR
print(f'Q1={Q1},Q3={Q3},IQR={IQR},LOWER LIMIT={Lower_limit}, UPPER LIMIT={Upper_limit}')
```

```
Q1=17.025,Q3=25.0,IQR=7.975000000000001,LOWER LIMIT=5.0624999999999964, UPPER LIMIT=28.9875
```

```
om[(om['MEDV']<Lower_limit)| (om["MEDV"]>Upper_limit)]
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2	242	17.8	392.
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3	222	18.7	394.
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3	222	18.7	396.
39	0.02763	75.0	2.95	0.0	0.428	6.595	21.8	5.4011	3	252	18.3	395.
40	0.03359	75.0	2.95	0.0	0.428	7.024	15.8	5.4011	3	252	18.3	395.
...	...	...	...	...	...	...	...	...	...	...	...	
371	9.23230	0.0	18.10	0.0	0.631	6.216	100.0	1.1691	24	666	20.2	366.
372	8.26725	0.0	18.10	1.0	0.668	5.875	89.6	1.1296	24	666	20.2	347.
398	38.35180	0.0	18.10	0.0	0.693	5.453	100.0	1.4896	24	666	20.2	396.
405	67.92080	0.0	18.10	0.0	0.693	5.683	100.0	1.4254	24	666	20.2	384.
473	4.64689	0.0	18.10	0.0	0.614	6.980	67.6	2.5329	24	666	20.2	374.

96 rows × 14 columns

```
outliers=[]
for i in om.MEDV:
  if i<Lower_limit or i>Upper_limit:
    outliers.append(i)
print("outliers are",outliers)
```

```
outliers are [34.7, 33.4, 36.2, 30.8, 34.9, 35.4, 31.6, 33.0, 38.7, 43.8, 33.2, 41.3, 50.0, 50.0, 50.0, 50.0, 29.4, 29.9, 37.2, 39.
```

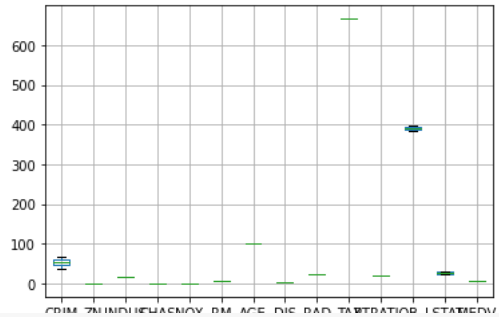
```
om1=om.drop(om[om.MEDV<Lower_limit].index)
```

```
om2=om[om.MEDV<Lower_limit]
om2
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
398	38.3518	0.0	18.1	0.0	0.693	5.453	100.0	1.4896	24	666	20.2	396.90
405	67.9208	0.0	18.1	0.0	0.693	5.683	100.0	1.4254	24	666	20.2	384.97

```
om2.boxplot()
```

```
<Axes: >
```




```
Upper_limit
```

```
28.9875
```

```
Lower_limit
```

```
5.0624999999999964
```

DISPLOT

```
sns.set(rc={'figure.figsize':(11.7,8.27)})
sns.distplot(om['MEDV'], bins=30)
plt.show()
```

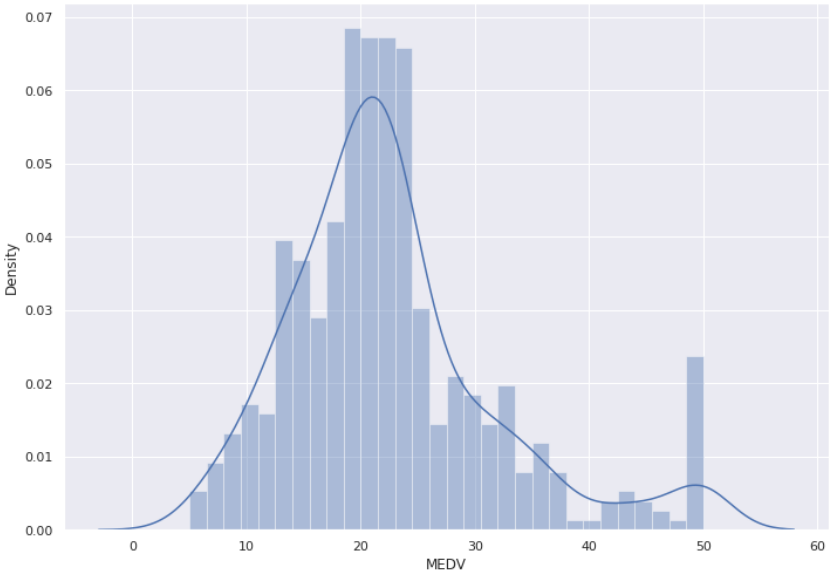
```
<ipython-input-39-d4bd883d1b20>:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(om['MEDV'], bins=30)
```



CORELATION MATRIX

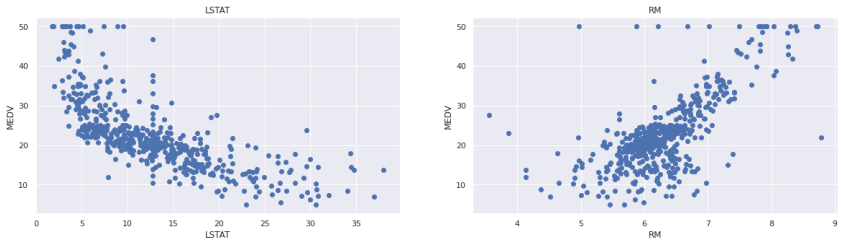
```
correlation_matrix = om.corr().round(2)
sns.heatmap(data=correlation_matrix, annot=True)
```

```
<Axes: >
```


```
plt.figure(figsize=(20, 5))

features = ['LSTAT', 'RM']
target = om['MEDV']

for i, col in enumerate(features):
    plt.subplot(1, len(features) , i+1)
    x = om[col]
    y = target
    plt.scatter(x, y, marker='o')
    plt.title(col)
    plt.xlabel(col)
    plt.ylabel('MEDV')
```




```
X=om.drop('MEDV',axis=1)
Y=om['MEDV']
```

```
X
Y
```

```
0      24.0
1      21.6
2      34.7
3      33.4
4      36.2
       ...
501    22.4
502    20.6
503    23.9
504    22.0
505    11.9
Name: MEDV, Length: 506, dtype: float64
```

LINEAR REGRESSION

```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state=5)
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
(404, 13)
(102, 13)
(404,)
(102,)
```

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

lin_model = LinearRegression()
lin_model.fit(X_train, Y_train)
```

```
LinearRegression
LinearRegression()
```

```
lin_model
```

```
LinearRegression
LinearRegression()
```

```
import numpy as np
```

TRAINING AND TESTING

```
y_train_predict = lin_model.predict(X_train)
rmse = (np.sqrt(mean_squared_error(Y_train, y_train_predict)))
r2 = r2_score(Y_train, y_train_predict)

print("The model performance for training set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
print("\n")


y_test_predict = lin_model.predict(X_test)
rmse = (np.sqrt(mean_squared_error(Y_test, y_test_predict)))
r2 = r2_score(Y_test, y_test_predict)

print("The model performance for testing set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
```

```
The model performance for training set
--------------------------------------
RMSE is 4.840940238126834
R2 score is 0.7271916265566591


The model performance for testing set
--------------------------------------
RMSE is 4.586413823734064
R2 score is 0.7313302802551167
```

```
mse=mean_squared_error(Y_test,y_test_predict)
print(mse)
mse=mean_squared_error(y_train_predict,Y_train)
print(mse)
```

```
21.035191762538922
23.434702389115483
```

Data Analytics II

1. Implement logistic regression using Python/R to perform classification on Social_Network_Ads.csv dataset.
2. Compute Confusion matrix to find TP, FP, TN, FN, Accuracy, Error rate, Precision, Recall on the given dataset

IMPORTING REQUIRED LIBRARIES

Double-click (or enter) to edit

```
import pandas as pd
```

```
from google.colab import files
files.upload()
```

```
Choose files No file chosen  Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving social network ads.csv to social network ads.csv
{'social network ads.csv':
b'Age,EstimatedSalary,Purchased\r\n19,19000.0\r\n35,20000.0\r\n26,43000.0\r\n27,57000
```

LOADING DATASET

```
om=pd.read_csv("/content/social network ads.csv")
```

```
om
```

	Age	EstimatedSalary	Purchased
0	19	19000	0
1	35	20000	0
2	26	43000	0
3	27	57000	0
4	19	76000	0
...	...	...	...
395	46	41000	1
396	51	23000	1
397	50	20000	1
398	36	33000	0
399	49	36000	1

400 rows × 3 columns

DATA PREPROCESSING

```
om.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 3 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Age              400 non-null    int64
 1   EstimatedSalary  400 non-null    int64
 2   Purchased        400 non-null    int64
dtypes: int64(3)
memory usage: 9.5 KB
```

```
om.describe()
```

	Age	EstimatedSalary	Purchased
count	400.000000	400.000000	400.000000
mean	37.655000	69742.500000	0.357500
std	10.482877	34096.960282	0.479864
min	18.000000	15000.000000	0.000000

CHECKING FOR NULL VALUES

```
om.isnull()
```

	Age	EstimatedSalary	Purchased
0	False	False	False
1	False	False	False
2	False	False	False
3	False	False	False
4	False	False	False
...	...	...	...
395	False	False	False
396	False	False	False
397	False	False	False
398	False	False	False
399	False	False	False

400 rows × 3 columns

```
om.isnull().sum
```

```
<bound method NDFrame._add_numeric_operations.<locals>.sum of        Age  EstimatedSalary  Purchased
0    False            False      False
1    False            False      False
2    False            False      False
3    False            False      False
4    False            False      False
..     ...              ...        ...
395  False            False      False
396  False            False      False
397  False            False      False
398  False            False      False
399  False            False      False

[400 rows x 3 columns]>
```

```
om.isnull().sum()
```

```
Age               0
EstimatedSalary   0
Purchased         0
dtype: int64
```

```
import seaborn as sns
import matplotlib.pyplot as plt
```

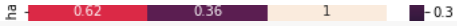
CORRELATION MATRIX

```
correlation_matrix = om.corr().round(2)
sns.heatmap(data=correlation_matrix, annot=True)
```

```
<Axes: >
```


```
X=om.drop('Purchased',axis=1)
Y=om['Purchased']
```


LOGISTIC REGRESSION




```
X
```

	Age	EstimatedSalary
0	19	19000
1	35	20000
2	26	43000
3	27	57000
4	19	76000
...	...	...
395	46	41000
396	51	23000
397	50	20000
398	36	33000
399	49	36000

400 rows × 2 columns

```
Y
```

```
0      0
1      0
2      0
3      0
4      0
      ..
395    1
396    1
397    1
398    0
399    1
Name: Purchased, Length: 400, dtype: int64
```

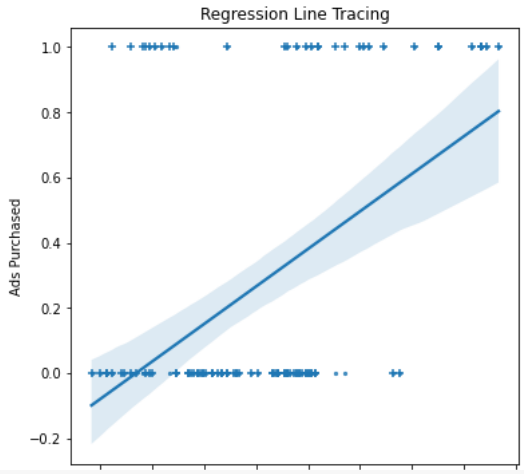
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.25, random_state=0)
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
print(f'Train Dataset Size - X: {X_train.shape}, Y: {Y_train.shape}')
print(f'Test Dataset Size - X: {X_test.shape}, Y: {Y_test.shape}')
```

```
Train Dataset Size - X: (300, 2), Y: (300,)
Test Dataset Size - X: (100, 2), Y: (100,)
```

```
from sklearn.linear_model import LogisticRegression
lm = LogisticRegression(random_state = 0, solver='lbfgs' )
lm.fit(X_train, Y_train)
predictions = lm.predict(X_test)
plt.figure(figsize=(6, 6))
sns.regplot(x = X_test[:, 1], y = predictions, scatter_kws={'s':5})
plt.scatter(X_test[:, 1], Y_test, marker = '+')
plt.xlabel("User's Estimated Salary")
plt.ylabel('Ads Purchased')
plt.title('Regression Line Tracing')
```

```
Text(0.5, 1.0, 'Regression Line Tracing')
```




```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report,precision_score,accuracy_score,recall_score
cm = confusion_matrix(Y_test, predictions)
print(f'''Confusion matrix :\n
| Positive Prediction\t| Negative Prediction
---------------+-----------------------+----------------------
Positive Class | True Positive (TP) {cm[0, 0]}\t| False Negative (FN) {cm[0, 1]}
---------------+-----------------------+----------------------
Negative Class | False Positive (FP) {cm[1, 0]}\t| True Negative (TN) {cm[1, 1]}\n\n''')
```

```
Confusion matrix :

| Positive Prediction   | Negative Prediction
---------------+-----------------------+----------------------
Positive Class | True Positive (TP) 65  | False Negative (FN) 3
---------------+-----------------------+----------------------
Negative Class | False Positive (FP) 8  | True Negative (TN) 24
```

```
cr = classification_report(Y_test, predictions)
print('Classification report : \n', cr)
```

```
Classification report :
                precision    recall  f1-score   support

           0       0.89      0.96      0.92        68
           1       0.89      0.75      0.81        32

    accuracy                           0.89       100
   macro avg       0.89      0.85      0.87       100
weighted avg       0.89      0.89      0.89       100
```

```
import numpy as np
```

```
precision_score=precision_score(Y_test,predictions)
print(precision_score)
```

```
0.8888888888888888
```

```
ac=accuracy_score(Y_test,predictions)
print(ac)
```

```
0.89
```

```
rc=recall_score(Y_test,predictions)
print(rc)
```

```
0.75
```

```
error_rate=1-ac
print(error_rate)
```

```
0.10999999999999999
```

Data Analytics III

1. Implement Simple Naïve Bayes classification algorithm using Python/R on iris.csv dataset.
2. Compute Confusion matrix to find TP, FP, TN, FN, Accuracy, Error rate, Precision, Recall on the given dataset

IMPORTING REQUIRED LIBRARIES

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

```
from google.colab import files
files.upload()
```

Choose files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving Iris.csv to Iris.csv
{'Iris.csv':
b'Id,SepalLengthCm,SepalWidthCm,PetalLengthCm,PetalWidthCm,Species\n1,5.1,3.5,1.4,0.2
setosa\n2,4.9,3.0,1.4,0.2,Iris-setosa\n3,4.7,3.2,1.3,0.2,Iris-
setosa\n4,4.6,3.1,1.5,0.2,Iris-setosa\n5,5.0,3.6,1.4,0.2,Iris-
setosa\n6,5.4,3.9,1.7,0.4,Iris-setosa\n7,4.6,3.4,1.4,0.3,Iris-
setosa\n8,5.0,3.4,1.5,0.2,Iris-setosa\n9,4.4,2.9,1.4,0.2,Iris-
setosa\n10,4.9,3.1,1.5,0.1,Iris-setosa\n11,5.4,3.7,1.5,0.2,Iris-
setosa\n12,4.8,3.4,1.6,0.2,Iris-setosa\n13,4.8,3.0,1.4,0.1,Iris-
setosa\n14,4.3,3.0,1.1,0.1,Iris-setosa\n15,5.8,4.0,1.2,0.2,Iris-
setosa\n16,5.7,4.4,1.5,0.4,Iris-setosa\n17,5.4,3.9,1.3,0.4,Iris-
setosa\n18,5.1,3.5,1.4,0.3,Iris-setosa\n19,5.7,3.8,1.7,0.3,Iris-
setosa\n20,5.1,3.8,1.5,0.3,Iris-setosa\n21,5.4,3.4,1.7,0.2,Iris-
setosa\n22,5.1,3.7,1.5,0.4,Iris-setosa\n23,4.6,3.6,1.0,0.2,Iris-
setosa\n24,5.1,3.3,1.7,0.5,Iris-setosa\n25,4.8,3.4,1.9,0.2,Iris-
setosa\n26,5.0,3.0,1.6,0.2,Iris-setosa\n27,5.0,3.4,1.6,0.4,Iris-
setosa\n28,5.2,3.5,1.5,0.2,Iris-setosa\n29,5.2,3.4,1.4,0.2,Iris-
setosa\n30,4.7,3.2,1.6,0.2,Iris-setosa\n31,4.8,3.1,1.6,0.2,Iris-
setosa\n32,5.4,3.4,1.5,0.4,Iris-setosa\n33,5.2,4.1,1.5,0.1,Iris-
setosa\n34,5.5,4.2,1.4,0.2,Iris-setosa\n35,4.9,3.1,1.5,0.1,Iris-
setosa\n36,5.0,3.2,1.2,0.2,Iris-setosa\n37,5.5,3.5,1.3,0.2,Iris-
setosa\n38,4.9,3.1,1.5,0.1,Iris-setosa\n39,4.4,3.0,1.3,0.2,Iris-
setosa\n40,5.1,3.4,1.5,0.2,Iris-setosa\n41,5.0,3.5,1.3,0.3,Iris-
setosa\n42,4.5,2.3,1.3,0.3,Iris-setosa\n43,4.4,3.2,1.3,0.2,Iris-
setosa\n44,5.0,3.5,1.6,0.6,Iris-setosa\n45,5.1,3.8,1.9,0.4,Iris-
setosa\n46,4.8,3.0,1.4,0.3,Iris-setosa\n47,5.1,3.8,1.6,0.2,Iris-
setosa\n48,4.6,3.2,1.4,0.2,Iris-setosa\n49,5.3,3.7,1.5,0.2,Iris-
setosa\n50,5.0,3.3,1.4,0.2,Iris-setosa\n51,7.0,3.2,4.7,1.4,Iris-
versicolor\n52,6.4,3.2,4.5,1.5,Iris-versicolor\n53,6.9,3.1,4.9,1.5,Iris-
versicolor\n54,5.5,2.3,4.0,1.3,Iris-versicolor\n55,6.5,2.8,4.6,1.5,Iris-
versicolor\n56,5.7,2.8,4.5,1.3,Iris-versicolor\n57,6.3,3.3,4.7,1.6,Iris-
versicolor\n58,4.9,2.4,3.3,1.0,Iris-versicolor\n59,6.6,2.9,4.6,1.3,Iris-
versicolor\n60,5.2,2.7,3.9,1.4,Iris-versicolor\n61,5.0,2.0,3.5,1.0,Iris-
versicolor\n62,5.9,3.0,4.2,1.5,Iris-versicolor\n63,6.0,2.2,4.0,1.0,Iris-
versicolor\n64,6.1,2.9,4.7,1.4,Iris-versicolor\n65,5.6,2.9,3.6,1.3,Iris-
versicolor\n66,6.7,3.1,4.4,1.4,Iris-versicolor\n67,5.6,3.0,4.5,1.5,Iris-
```

LOADING DATASET

```
df=pd.read_csv("Iris.csv")
```

DATA PREPROCESSING

```
df.head()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

```
df.describe()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000

CHECKING FOR NULL VALUES

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

```
Id              0
SepalLengthCm   0
SepalWidthCm    0
PetalLengthCm   0
PetalWidthCm    0
Species         0
dtype: int64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             150 non-null    int64  
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object 
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

```
X = df.drop(['Species'], axis = 1)
Y = df['Species']
```

NAIVE BAYES CLASSIFICATION

```
from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(X, Y, test_size =0.2,random_state = 0)
```

```
from sklearn.naive_bayes import GaussianNB
gaussian = GaussianNB()
```

```
gaussian.fit(xtrain, ytrain)
```

```
▾ GaussianNB
GaussianNB()
```

```
y_pred = gaussian.predict(xtest)
```

```
print(xtrain)
print("------------\n")
print(xtest)
print("------------\n")
print(ytrain)
print("------------\n")
print(ytest)
print("------------\n")
print(y_pred)
```

```
       Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
137  138            6.4           3.1            5.5           1.8
84    85            5.4           3.0            4.5           1.5
27    28            5.2           3.5            1.5           0.2
127  128            6.1           3.0            4.9           1.8
132  133            6.4           2.8            5.6           2.2
..   ...            ...           ...            ...           ...
```

```
9     10            4.9           3.1            1.5           0.1
103  104            6.3           2.9            5.6           1.8
67    68            5.8           2.7            4.1           1.0
117  118            7.7           3.8            6.7           2.2
47    48            4.6           3.2            1.4           0.2

[120 rows x 5 columns]
------------

      Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
114  115            5.8           2.8            5.1           2.4
62    63            6.0           2.2            4.0           1.0
33    34            5.5           4.2            1.4           0.2
107  108            7.3           2.9            6.3           1.8
7      8            5.0           3.4            1.5           0.2
100  101            6.3           3.3            6.0           2.5
40    41            5.0           3.5            1.3           0.3
86    87            6.7           3.1            4.7           1.5
76    77            6.8           2.8            4.8           1.4
71    72            6.1           2.8            4.0           1.3
134  135            6.1           2.6            5.6           1.4
51    52            6.4           3.2            4.5           1.5
73    74            6.1           2.8            4.7           1.2
54    55            6.5           2.8            4.6           1.5
63    64            6.1           2.9            4.7           1.4
37    38            4.9           3.1            1.5           0.1
78    79            6.0           2.9            4.5           1.5
90    91            5.5           2.6            4.4           1.2
45    46            4.8           3.0            1.4           0.3
16    17            5.4           3.9            1.3           0.4
121  122            5.6           2.8            4.9           2.0
66    67            5.6           3.0            4.5           1.5
24    25            4.8           3.4            1.9           0.2
8      9            4.4           2.9            1.4           0.2
126  127            6.2           2.8            4.8           1.8
22    23            4.6           3.6            1.0           0.2
44    45            5.1           3.8            1.9           0.4
97    98            6.2           2.9            4.3           1.3
93    94            5.0           2.3            3.3           1.0
26    27            5.0           3.4            1.6           0.4
------------

137     Iris-virginica
84     Iris-versicolor
27         Iris-setosa
127     Iris-virginica
132     Iris-virginica
            ...
9          Iris-setosa
103     Iris-virginica
67     Iris-versicolor
```

CONFUSION MATRIX

```
from sklearn.metrics import precision_score,confusion_matrix,accuracy_score,recall_score, classification_report
cm= confusion_matrix(ytest, y_pred)
```

```
cm
```

```
array([[11,  0,  0],
       [ 0, 13,  0],
       [ 0,  0,  6]])
```

```
from sklearn.metrics import accuracy_score

print ("Accuracy : ", accuracy_score(ytest, y_pred))
```

```
Accuracy :  1.0
```

```
error_rate = 1- accuracy_score(ytest, y_pred)
```

```
error_rate
```

```
0.0
```

```
print("classification report: ",classification_report(ytest, y_pred))
```

```
classification report:                   precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       1.00      1.00      1.00        13
 Iris-virginica       1.00      1.00      1.00         6
```

```
              accuracy                           1.00        30
             macro avg       1.00      1.00      1.00        30
          weighted avg       1.00      1.00      1.00        30
```

ACCURACY SCORE, PRECISION SCORE, RECALL SCORE

```
accuracy = accuracy_score(ytest,y_pred)
precision =precision_score(ytest, y_pred,average='macro')
recall = recall_score(ytest, y_pred,average='macro')
```

```
accuracy
```

```
1.0
```

```
precision
```

```
1.0
```

```
recall
```

```
1.0
```

Text Analytics

1. Extract Sample document and apply following document preprocessing methods: Tokenization, POS Tagging, stop words removal, Stemming and Lemmatization.
2. Create representation of document by calculating Term Frequency and Inverse Document Frequency.

IMPORTING REQUIRED LIBRARIES

```
pip install nltk
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: nltk in /usr/local/lib/python3.9/dist-packages (3.8.1)
Requirement already satisfied: click in /usr/local/lib/python3.9/dist-packages (from nltk) (8.1.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.9/dist-packages (from nltk) (4.65.0)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.9/dist-packages (from nltk) (2022.10.31)
Requirement already satisfied: joblib in /usr/local/lib/python3.9/dist-packages (from nltk) (1.2.0)
```

```
import nltk
```

```
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
True
```

TOKENIZATION

```
text= "Tokenization is the first step in text analytics. The process of breaking down a text paragraph into smaller chunks such as words
```

```
from nltk.tokenize import sent_tokenize
tokenized_text= sent_tokenize(text)
print(tokenized_text)
```

```
['Tokenization is the first step in text analytics.', 'The process of breaking down a text paragraph into smaller chunks such as wo
```

```
from nltk.tokenize import word_tokenize
tokenized_word=word_tokenize(text)
print(tokenized_word)
```

```
['Tokenization', 'is', 'the', 'first', 'step', 'in', 'text', 'analytics', '.', 'The', 'process', 'of', 'breaking', 'down', 'a', 'te
```

STOP WORDS REMOVAL

```
from nltk.corpus import stopwords
stop_words=set(stopwords.words("english"))
print(stop_words)
```

```
{'all', 'those', 'weren', 'myself', 'by', 'before', 'further', "didn't", 'while', 'against', 'hers', 'now', "won't", "you'd", 'his'
```

```
import re
re.compile('<title>(.*)</title>')
```

```
re.compile(r'<title>(.*)</title>', re.UNICODE)
```

STEMMING

```
text= "How to remove stop words with NLTK library in Python?"
text= re.sub('[^a-zA-Z]', ' ',text)
tokens = word_tokenize(text.lower())
```

```
filtered_text=[]
for w in tokens:
    if w not in stop_words:
        filtered_text.append(w)
print("Tokenized Sentence:",tokens)
print("Filterd Sentence:",filtered_text)
```

```
Tokenized Sentence: ['how', 'to', 'remove', 'stop', 'words', 'with', 'nltk', 'library', 'in', 'python']
Filterd Sentence: ['remove', 'stop', 'words', 'nltk', 'library', 'python']
```

```
from nltk.stem import PorterStemmer
e_words= ["wait", "waiting", "waited", "waits"]
ps =PorterStemmer()
for w in e_words:
  rootWord=ps.stem(w)
print(rootWord)
```

```
wait
```

LEMMATIZATION

```
from nltk.stem import WordNetLemmatizer
wordnet_lemmatizer = WordNetLemmatizer()
text = "studies studying cries cry"
tokenization = nltk.word_tokenize(text)
for w in tokenization:
  print("Lemma for {} is {}".format(w,wordnet_lemmatizer.lemmatize(w)))
```

```
Lemma for studies is study
Lemma for studying is studying
Lemma for cries is cry
Lemma for cry is cry
```

```
import pandas as pd
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
documentA = 'Jupiter is the largest Planet'
documentB = 'Mars is the fourth planet from the Sun'
```

```
bagOfWordsA = documentA.split(' ')
bagOfWordsB = documentB.split(' ')
```

```
uniqueWords = set(bagOfWordsA).union(set(bagOfWordsB))
```

```
numOfWordsA = dict.fromkeys(uniqueWords, 0)
for word in bagOfWordsA:
  numOfWordsA[word] += 1
  numOfWordsB = dict.fromkeys(uniqueWords, 0)
for word in bagOfWordsB:
  numOfWordsB[word] += 1
```

```
def computeTF(wordDict, bagOfWords):
  tfDict = {}
  bagOfWordsCount = len(bagOfWords)
  for word, count in wordDict.items():
    tfDict[word] = count / float(bagOfWordsCount)
  return tfDict
tfA = computeTF(numOfWordsA, bagOfWordsA)
tfB = computeTF(numOfWordsB, bagOfWordsB)
```

```
def computeIDF(documents):
  import math
  N = len(documents)
  idfDict = dict.fromkeys(documents[0].keys(), 0)
  for document in documents:
    for word, val in document.items():
      if val > 0:
        idfDict[word] += 1
  for word, val in idfDict.items():
    idfDict[word] = math.log(N / float(val))
  return idfDict
idfs = computeIDF([numOfWordsA, numOfWordsB])
idfs
```

```
{'Mars': 0.6931471805599453,
 'Sun': 0.6931471805599453,
 'from': 0.6931471805599453,
 'fourth': 0.6931471805599453,
 'is': 0.0,
 'largest': 0.6931471805599453,
 'Jupiter': 0.6931471805599453,
 'the': 0.0,
 'planet': 0.6931471805599453,
 'Planet': 0.6931471805599453}
```

```
def computeTFIDF(tfBagOfWords, idfs):
  tfidf = {}
  for word, val in tfBagOfWords.items():
    tfidf[word] = val * idfs[word]
  return tfidf
tfidfA = computeTFIDF(tfA, idfs)
tfidfB = computeTFIDF(tfB, idfs)
df = pd.DataFrame([tfidfA, tfidfB])
df
```

	Mars	Sun	from	fourth	is	largest	Jupiter	the	planet	Pl
0	0.000000	0.000000	0.000000	0.000000	0.0	0.138629	0.138629	0.0	0.000000	0.13
1	0.086643	0.086643	0.086643	0.086643	0.0	0.000000	0.000000	0.0	0.086643	0.00

Data Visualization I

1. Use the inbuilt dataset 'titanic'. The dataset contains 891 rows and contains information about the passengers who boarded the unfortunate Titanic ship. Use the Seaborn library to see if we can find any patterns in the data.
2. Write a code to check how the price of the ticket (column name: 'fare') for each passenger is distributed by plotting a histogram.

IMPORTING REQUIRED LIBRARIES

```
import pandas as pd
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
import seaborn as sns
```

LOADING DATASET

```
dataset=sns.load_dataset('titanic')
```

DATA PREPROCESSING

```
dataset.head()
```

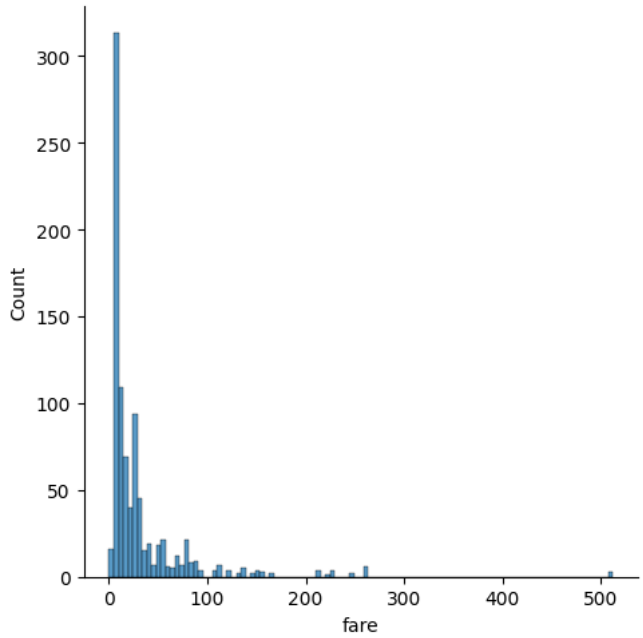
	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adul
0	0	3	male	22.0	1	0	7.2500	S	Third	man	
1	1	1	female	38.0	1	0	71.2833	C	First	woman	
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	
3	1	1	female	35.0	1	0	53.1000	S	First	woman	
4	0	3	male	35.0	0	0	8.0500	S	Third	man	

DATA VISUALIZATION

DISPLOT

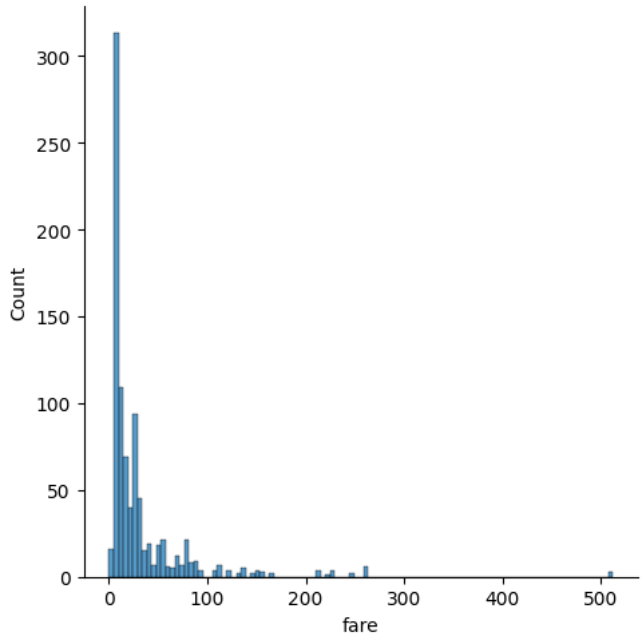
```
sns.displot(dataset['fare'])
```

```
<seaborn.axisgrid.FacetGrid at 0x7fb13030a880>
```



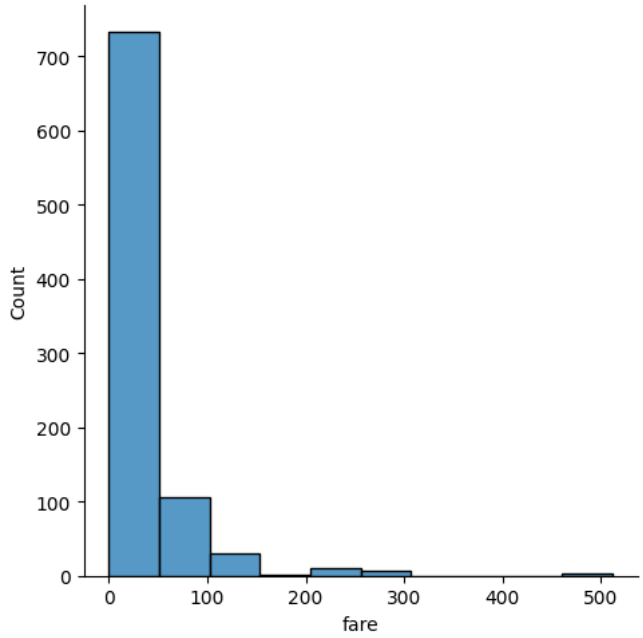
```
sns.displot(dataset['fare'],kde=False)
```

```
<seaborn.axisgrid.FacetGrid at 0x7fb1311c8970>
```




```
sns.displot(dataset['fare'],kde=False,bins=10)
```

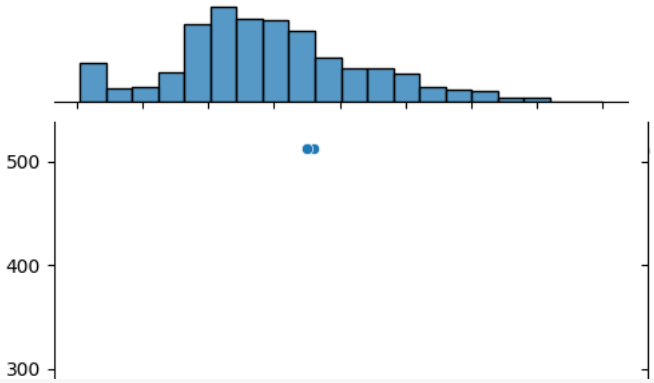
```
<seaborn.axisgrid.FacetGrid at 0x7fb12d75bcd0>
```



JOINTPLOT

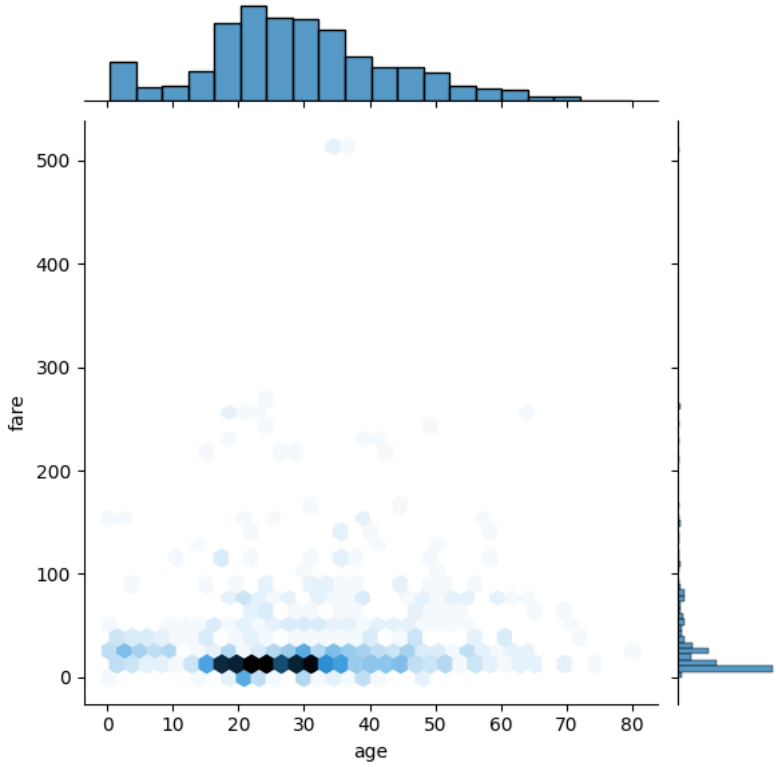
```
sns.jointplot(x='age',y='fare',data=dataset)
```

```
<seaborn.axisgrid.JointGrid at 0x7fb12d5f8e20>
```




```
sns.jointplot(x='age',y='fare',data=dataset,kind='hex')
```

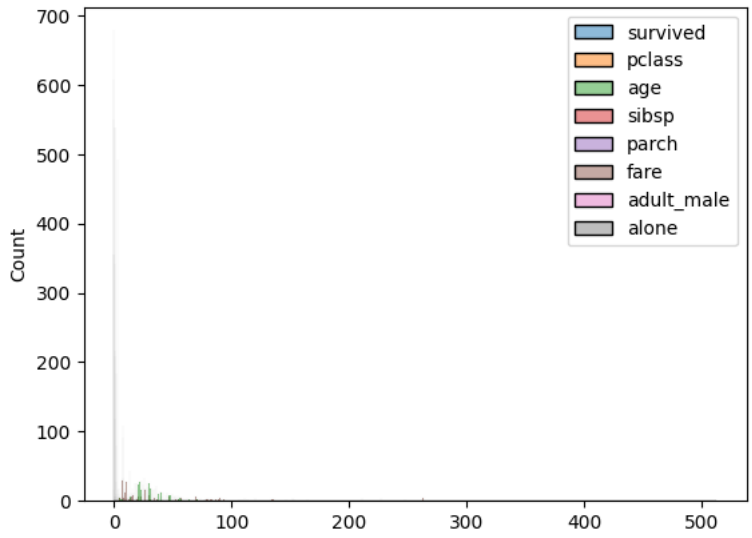
```
<seaborn.axisgrid.JointGrid at 0x7fb12d677ee0>
```



HISTPLOT

```
sns.histplot(dataset)
```

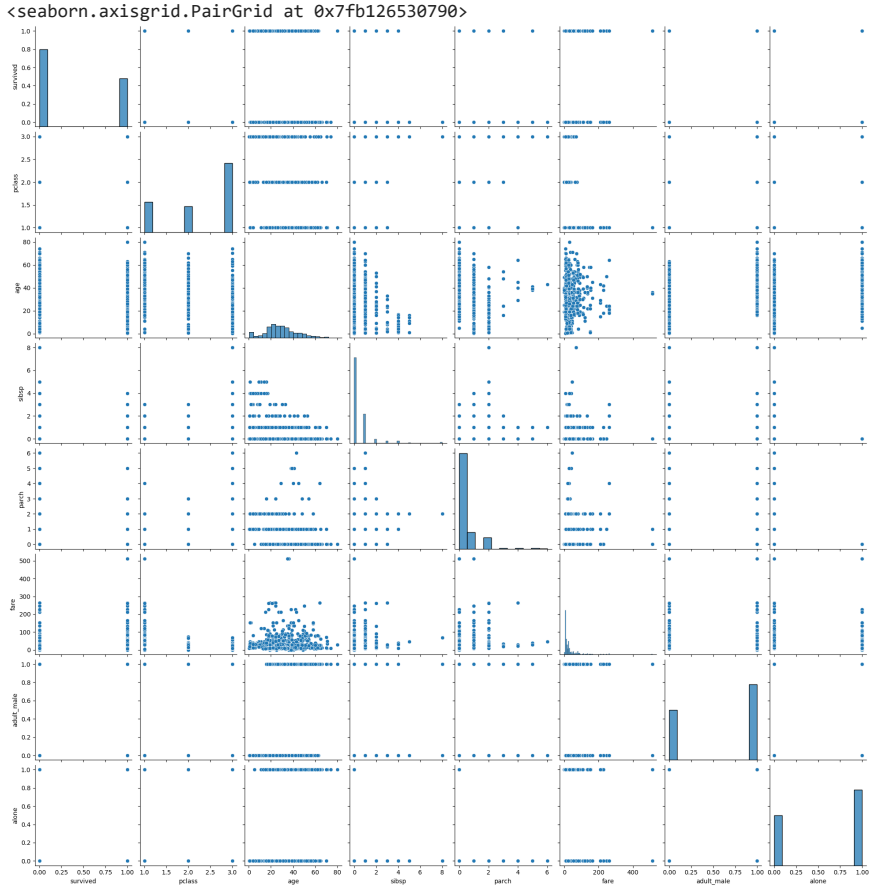
```
<Axes: ylabel='Count'>
```



PAIRPLOT

```
sns.pairplot(dataset)
```

```
<__array_function__ internals>:180: RuntimeWarning: Converting input from bool to <cl
<__array_function__ internals>:180: RuntimeWarning: Converting input from bool to <cl
<seaborn.axisgrid.PairGrid at 0x7fb126530790>
```



Data Visualization II

1. Use the inbuilt dataset 'titanic' as used in the above problem. Plot a box plot for distribution of age with respect to each gender along with the information about whether they survived or not. (Column names : 'sex' and 'age')
2. Write observations on the inference from the above statistics.

IMPORTING REQUIRED LIBRARIES

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
from seaborn import load_dataset
```

```
sns.get_dataset_names()
```

```
['anagrams',
 'anscombe',
 'attention',
 'brain_networks',
 'car_crashes',
 'diamonds',
 'dots',
 'dowjones',
 'exercise',
 'flights',
 'fmri',
 'geyser',
 'glue',
 'healthexp',
 'iris',
 'mpg',
 'penguins',
 'planets',
 'seaice',
 'taxis',
 'tips',
 'titanic']
```

LOADING DATASET

```
data=load_dataset('titanic')
```

```
data
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	a
0	0	3	male	22.0	1	0	7.2500	S	Third	man	
1	1	1	female	38.0	1	0	71.2833	C	First	woman	
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	
3	1	1	female	35.0	1	0	53.1000	S	First	woman	
4	0	3	male	35.0	0	0	8.0500	S	Third	man	
...	...	...	...	...	...	...	...	...	...	...	
886	0	2	male	27.0	0	0	13.0000	S	Second	man	
887	1	1	female	19.0	0	0	30.0000	S	First	woman	
888	0	3	female	NaN	1	2	23.4500	S	Third	woman	
889	1	1	male	26.0	0	0	30.0000	C	First	man	
890	0	3	male	32.0	0	0	7.7500	Q	Third	man	

891 rows × 15 columns

DATA PREPROCESSING

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
```

```
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype   
---  ------       --------------  -----   
 0   survived     891 non-null    int64   
 1   pclass       891 non-null    int64   
 2   sex          891 non-null    object  
 3   age          714 non-null    float64 
 4   sibsp        891 non-null    int64   
 5   parch        891 non-null    int64   
 6   fare         891 non-null    float64 
 7   embarked     889 non-null    object  
 8   class        891 non-null    category
 9   who          891 non-null    object  
 10  adult_male   891 non-null    bool    
 11  deck         203 non-null    category
 12  embark_town  889 non-null    object  
 13  alive        891 non-null    object  
 14  alone        891 non-null    bool    
dtypes: bool(2), category(2), float64(2), int64(4), object(5)
memory usage: 80.7+ KB
```

```
data.describe()
```

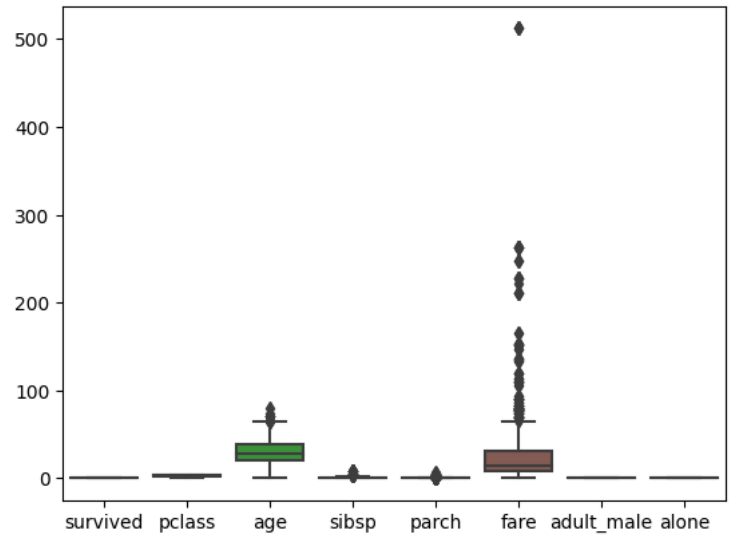
	survived	pclass	age	sibsp	parch	fare
count	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

DATA VISUALIZATION

BOXPLOT

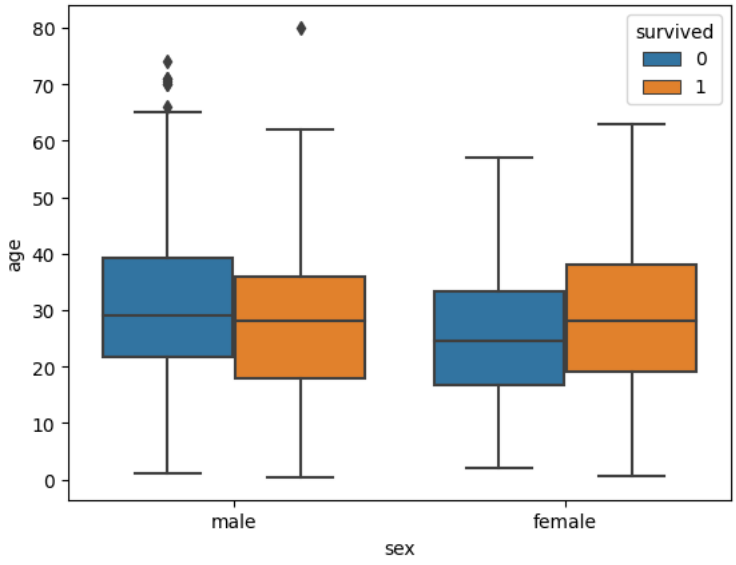
```
sns.boxplot(data)
```

```
<Axes: >
```




```
sns.boxplot(x=data['sex'],y=data['age'],hue=data['survived'])
```

```
<Axes: xlabel='sex', ylabel='age'>
```



Data Visualization III Download the Iris flower dataset or any other dataset into a DataFrame. (e.g., https://archive.ics.uci.edu/ml/datasets/Iris). Scan the dataset and give the inference as:

1. List down the features and their types (e.g., numeric, nominal) available in the dataset.
2. Create a histogram for each feature in the dataset to illustrate the feature distributions.
3. Create a boxplot for each feature in the dataset.
4. Compare distributions and identify outliers

IMPORTING REQUIRED LIBRARIES

```
import seaborn as sns
```

```
sns.get_dataset_names()
```

```
['anagrams',
 'anscombe',
 'attention',
 'brain_networks',
 'car_crashes',
 'diamonds',
 'dots',
 'dowjones',
 'exercise',
 'flights',
 'fmri',
 'geyser',
 'glue',
 'healthexp',
 'iris',
 'mpg',
 'penguins',
 'planets',
 'seaice',
 'taxis',
 'tips',
 'titanic']
```

LOADING DATASET

```
dataset=sns.load_dataset('iris')
```

DATA PREPROCESSING

```
dataset.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

```
dataset.describe
```

```
<bound method NDFrame.describe of      sepal_length  sepal_width  petal_length  petal_width    species
0             5.1          3.5           1.4          0.2     setosa
1             4.9          3.0           1.4          0.2     setosa
2             4.7          3.2           1.3          0.2     setosa
3             4.6          3.1           1.5          0.2     setosa
4             5.0          3.6           1.4          0.2     setosa
..            ...          ...           ...          ...        ...
145           6.7          3.0           5.2          2.3  virginica
146           6.3          2.5           5.0          1.9  virginica
147           6.5          3.0           5.2          2.0  virginica
148           6.2          3.4           5.4          2.3  virginica
149           5.9          3.0           5.1          1.8  virginica

[150 rows x 5 columns]>
```

```
dataset.info
```

```
<bound method DataFrame.info of      sepal_length  sepal_width  petal_length  petal_width    species
0             5.1          3.5           1.4          0.2     setosa
1             4.9          3.0           1.4          0.2     setosa
2             4.7          3.2           1.3          0.2     setosa
3             4.6          3.1           1.5          0.2     setosa
4             5.0          3.6           1.4          0.2     setosa
..            ...          ...           ...          ...        ...
145           6.7          3.0           5.2          2.3  virginica
146           6.3          2.5           5.0          1.9  virginica
147           6.5          3.0           5.2          2.0  virginica
148           6.2          3.4           5.4          2.3  virginica
149           5.9          3.0           5.1          1.8  virginica

[150 rows x 5 columns]>
```

CHECKING FOR NULL VALUES

```
dataset.isnull().any()
```

```
sepal_length    False
sepal_width     False
petal_length    False
petal_width     False
species         False
dtype: bool
```

```
dataset.isnull().sum()
```

```
sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
species         0
dtype: int64
```

MAX AND MIN

```
dataset.max()
```

```
sepal_length          7.9
sepal_width           4.4
petal_length          6.9
petal_width           2.5
species         virginica
dtype: object
```

```
dataset.min()
```

```
sepal_length       4.3
sepal_width        2.0
petal_length       1.0
petal_width        0.1
species         setosa
dtype: object
```

MEAN

```
dataset.mean()
```

```
<ipython-input-16-e55bc0ed4499>:1: FutureWarning: The default value of numeric_only in DataFrame.mean is deprecated. In a future ve
  dataset.mean()
sepal_length    5.843333
sepal_width     3.057333
petal_length    3.758000
petal_width     1.199333
dtype: float64
```

STANDARD DEVIATION

```
a=dataset.std()
a
```

```
<ipython-input-19-e1e8b4cdc687>:1: FutureWarning: The default value of numeric_only in DataFrame.std is deprecated. In a future ver
  a=dataset.std()
sepal_length    0.828066
sepal_width     0.435866
petal_length    1.765298
```

```
petal_width     0.762238
dtype: float64
```

VARIANCE

```
variance = a*a
variance
```

```
sepal_length    0.685694
sepal_width     0.189979
petal_length    3.116278
petal_width     0.581006
dtype: float64
```

```
import numpy as np
```

PERCENTILE

```
np.percentile(dataset['sepal_length'],90)
```

```
6.9
```

```
np.percentile(dataset['sepal_width'],90)
```

```
3.6099999999999994
```

```
np.percentile(dataset['petal_length'],90)
```

```
5.8
```

```
np.percentile(dataset['petal_width'],90)
```

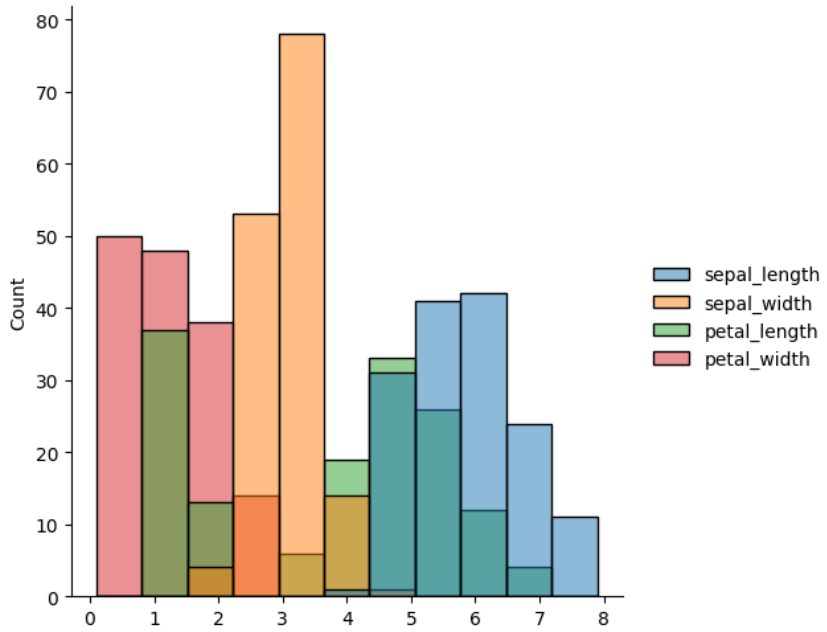
```
2.2
```

DATA VISUALIZTION

DISPLOT

```
sns.displot(dataset)
```

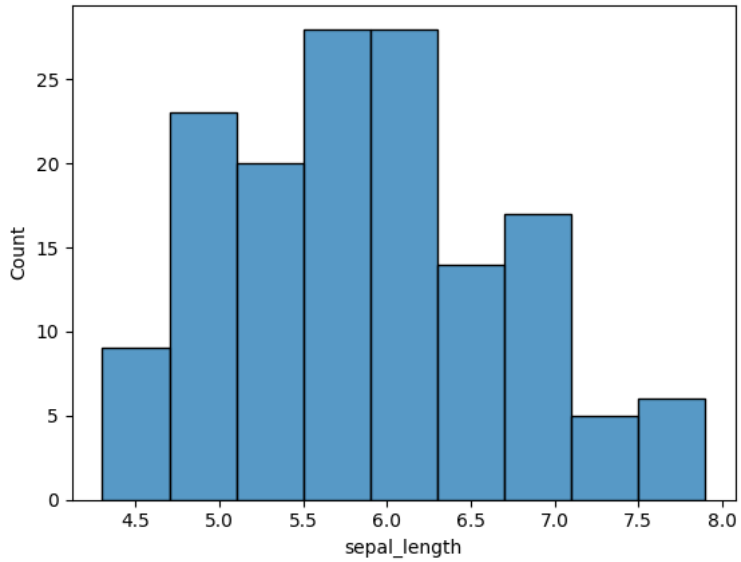
```
<seaborn.axisgrid.FacetGrid at 0x7f87a2fa0c40>
```



HISTPLOT

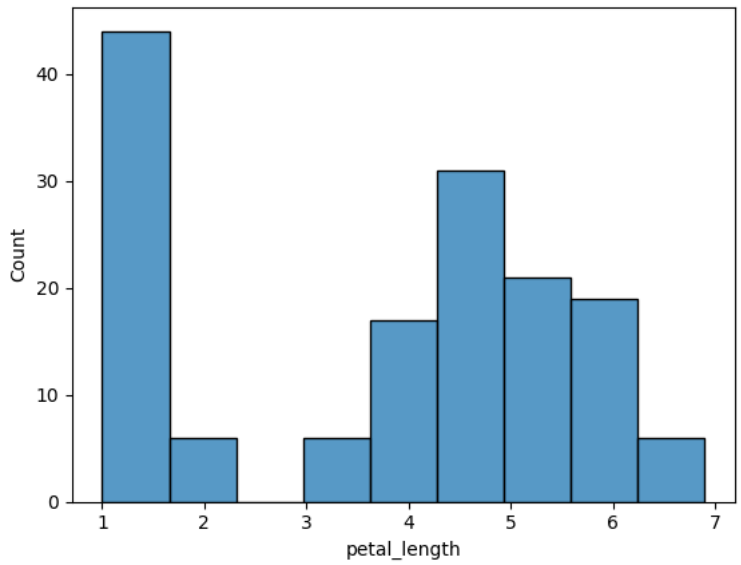
```
sns.histplot(dataset['sepal_length'])
```

```
<Axes: xlabel='sepal_length', ylabel='Count'>
```



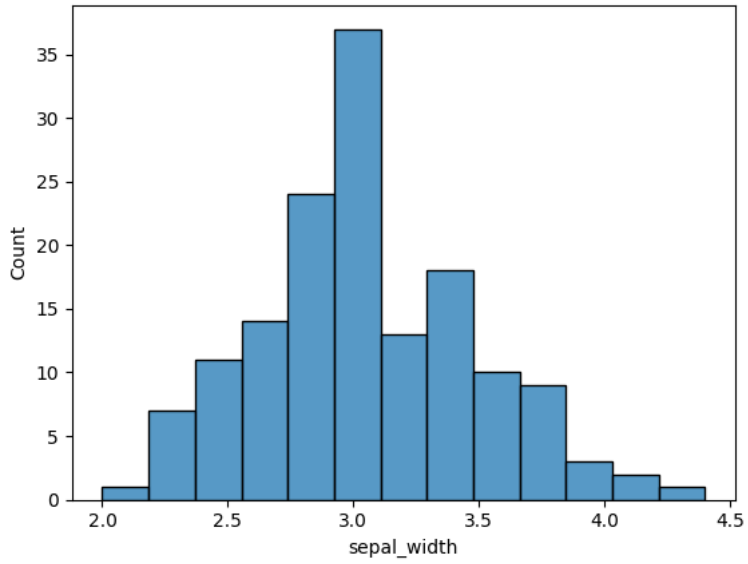

```
sns.histplot(dataset['petal_length'])
```

```
<Axes: xlabel='petal_length', ylabel='Count'>
```



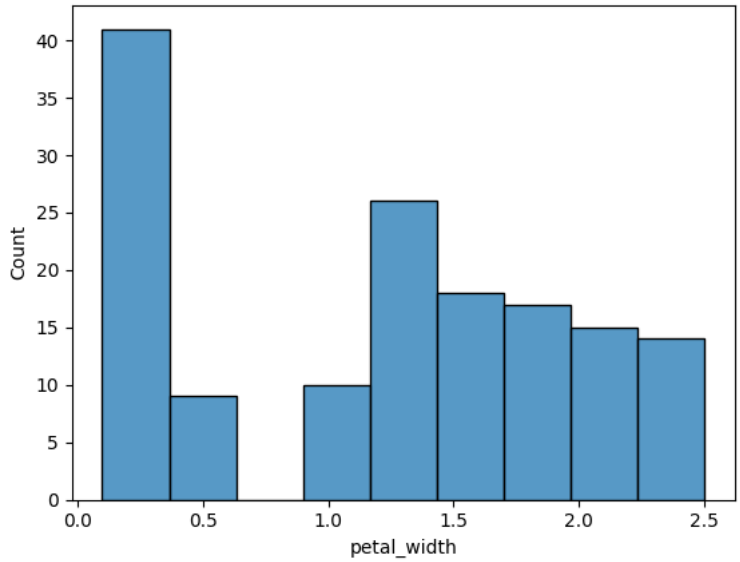

```
sns.histplot(dataset['sepal_width'])
```

```
<Axes: xlabel='sepal_width', ylabel='Count'>
```



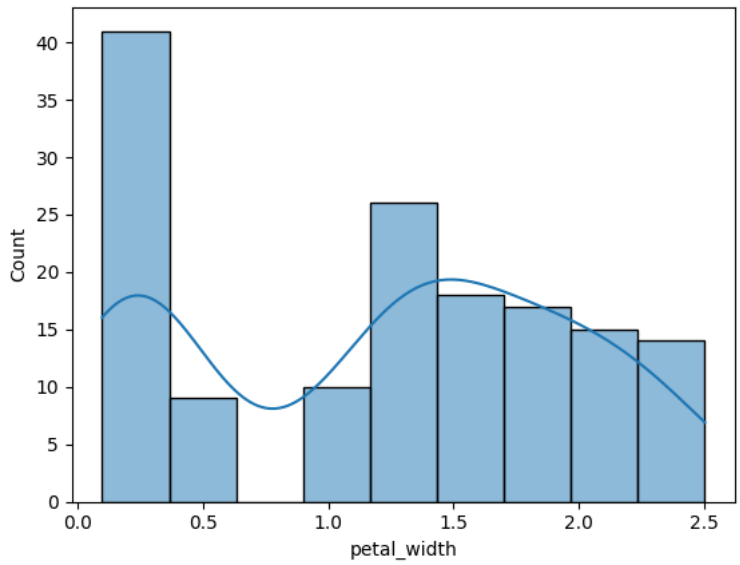
```
sns.histplot(dataset['petal_width'])
```

```
<Axes: xlabel='petal_width', ylabel='Count'>
```



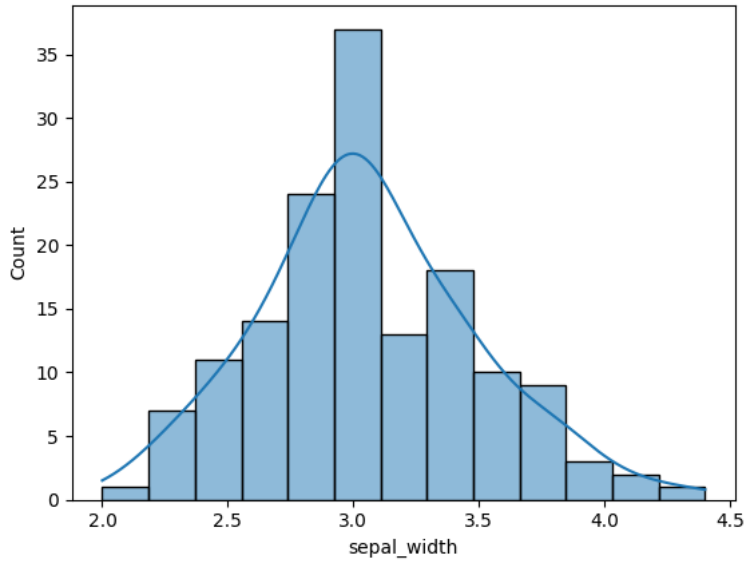

```
sns.histplot(dataset['petal_width'],kde=True)
```

```
<Axes: xlabel='petal_width', ylabel='Count'>
```



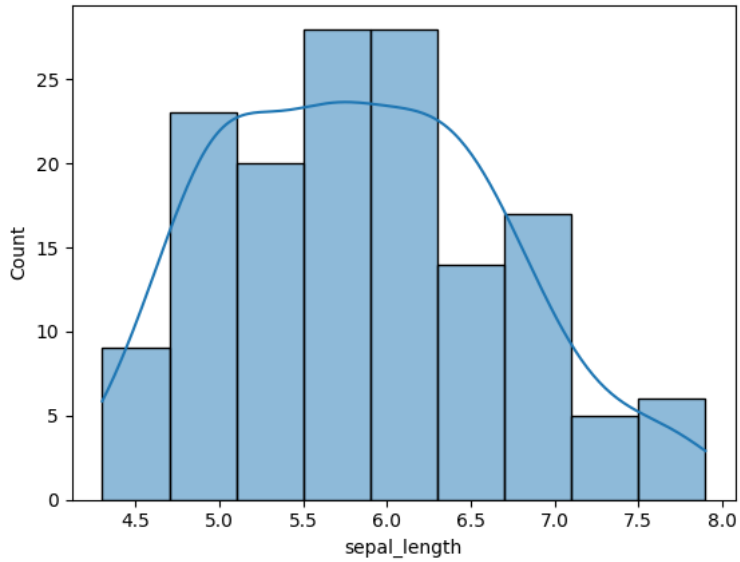

```
sns.histplot(dataset['sepal_width'],kde=True)
```

```
<Axes: xlabel='sepal_width', ylabel='Count'>
```



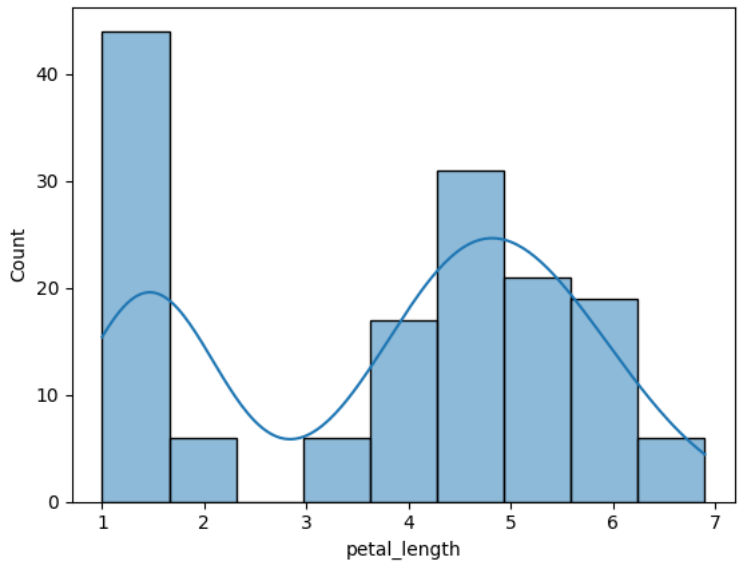
```
sns.histplot(dataset['sepal_length'],kde=True)
```

```
<Axes: xlabel='sepal_length', ylabel='Count'>
```




```
sns.histplot(dataset['petal_length'],kde=True)
```

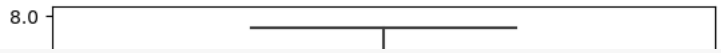
```
<Axes: xlabel='petal_length', ylabel='Count'>
```



BOXPLOT

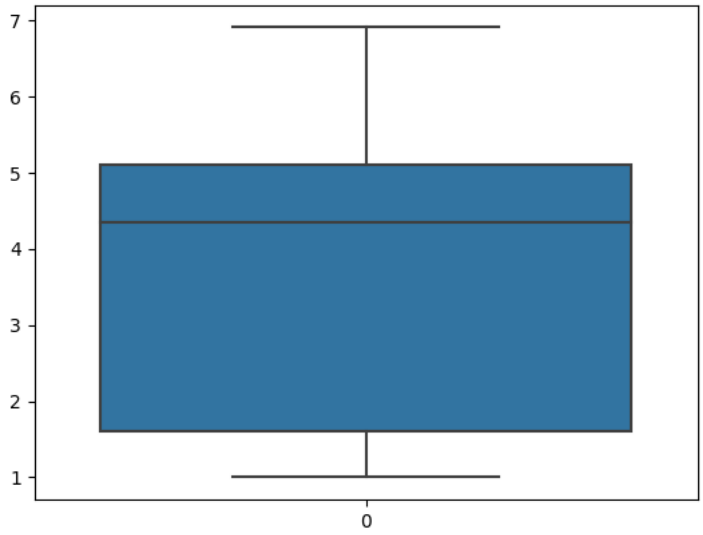
```
sns.boxplot(dataset['sepal_length'])
```

```
<Axes: >
```



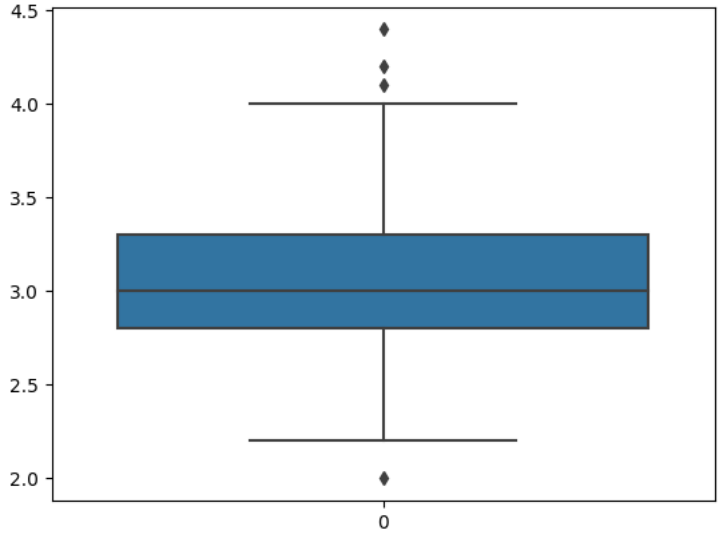

```
sns.boxplot(dataset['petal_length'])
```

```
<Axes: >
```



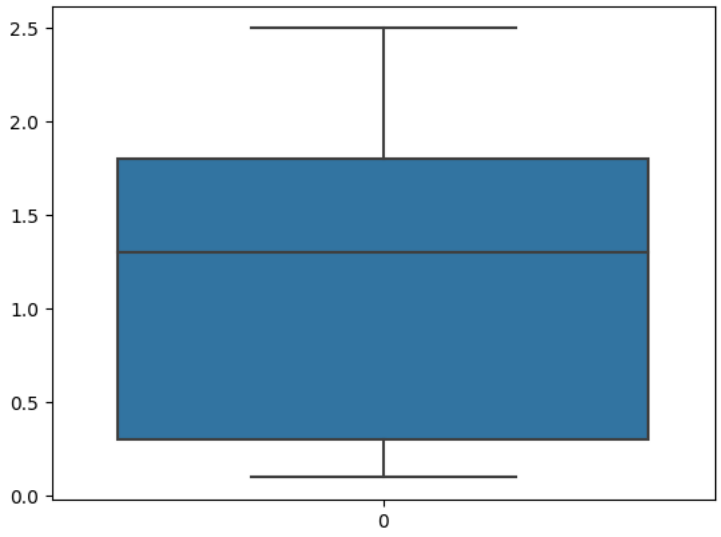

```
sns.boxplot(dataset['sepal_width'])
```

```
<Axes: >
```



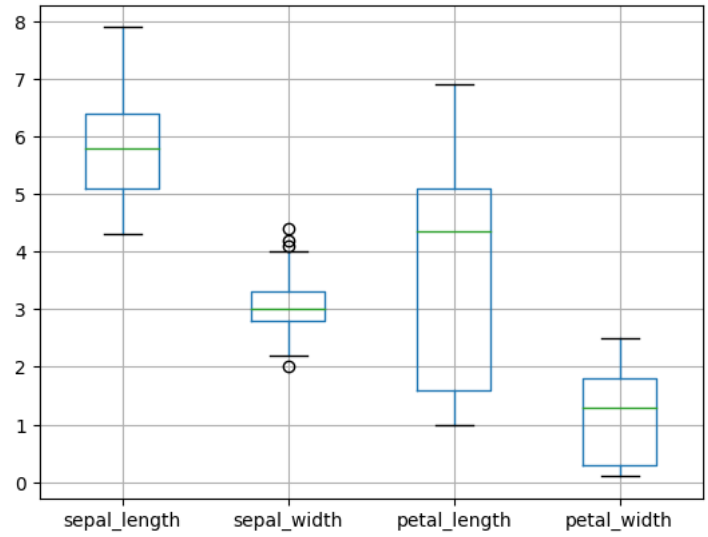

```
sns.boxplot(dataset['petal_width'])
```

```
<Axes: >
```



```
dataset.boxplot()
```

```
<Axes: >
```



**Data Wrangling, I

Perform the following operations using Python on any open source dataset (e.g., data.csv)

1. Import all the required Python Libraries.
2. Locate an open source data from the web (e.g., https://www.kaggle.com). Provide a clear description of the data and its source (i.e., URL of the web site).
3. Load the Dataset into pandas dataframe.
4. Data Preprocessing: check for missing values in the data using pandas isnull(), describe() function to get some initial statistics. Provide variable descriptions. Types of variables etc. Check the dimensions of the data frame.
5. Data Formatting and Data Normalization: Summarize the types of variables by checking the data types (i.e., character, numeric, integer, factor, and logical) of the variables in the data set. If variables are not in the correct data type, apply proper type conversions.
6. Turn categorical variables into quantitative variables in Python

Import all the required Python Libraries

```
import pandas as pd
```

```
!pip install -q kaggle
```

```
from google.colab import files
files.upload()
```

```
Choose files No file chosen  Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving Iris.csv to Iris.csv
{'Iris.csv':
b'Id,SepalLengthCm,SepalWidthCm,PetalLengthCm,PetalWidthCm,Species\n1,5.1,3.5,1.4,
setosa\n2,4.9,3.0,1.4,0.2,Iris-setosa\n3,4.7,3.2,1.3,0.2,Iris-
setosa\n4,4.6,3.1,1.5,0.2,Iris-setosa\n5,5.0,3.6,1.4,0.2,Iris-
setosa\n6,5.4,3.9,1.7,0.4,Iris-setosa\n7,4.6,3.4,1.4,0.3,Iris-
setosa\n8,5.0,3.4,1.5,0.2,Iris-setosa\n9,4.4,2.9,1.4,0.2,Iris-
setosa\n10,4.9,3.1,1.5,0.1,Iris-setosa\n11,5.4,3.7,1.5,0.2,Iris-
setosa\n12,4.8,3.4,1.6,0.2,Iris-setosa\n13,4.8,3.0,1.4,0.1,Iris-
setosa\n14,4.3,3.0,1.1,0.1,Iris-setosa\n15,5.8,4.0,1.2,0.2,Iris-
setosa\n16,5.7,4.4,1.5,0.4,Iris-setosa\n17,5.4,3.9,1.3,0.4,Iris-
setosa\n18,5.1,3.5,1.4,0.3,Iris-setosa\n19,5.7,3.8,1.7,0.3,Iris-
setosa\n20,5.1,3.8,1.5,0.3,Iris-setosa\n21,5.4,3.4,1.7,0.2,Iris-
setosa\n22,5.1,3.7,1.5,0.4,Iris-setosa\n23,4.6,3.6,1.0,0.2,Iris-
setosa\n24,5.1,3.3,1.7,0.5,Iris-setosa\n25,4.8,3.4,1.9,0.2,Iris-
setosa\n26,5.0,3.0,1.6,0.2,Iris-setosa\n27,5.0,3.4,1.6,0.4,Iris-
setosa\n28,5.2,3.5,1.5,0.2,Iris-setosa\n29,5.2,3.4,1.4,0.2,Iris-
setosa\n30,4.7,3.2,1.6,0.2,Iris-setosa\n31,4.8,3.1,1.6,0.2,Iris-
setosa\n32,5.4,3.4,1.5,0.4,Iris-setosa\n33,5.2,4.1,1.5,0.1,Iris-
setosa\n34,5.5,4.2,1.4,0.2,Iris-setosa\n35,4.9,3.1,1.5,0.1,Iris-
setosa\n36,5.0,3.2,1.2,0.2,Iris-setosa\n37,5.5,3.5,1.3,0.2,Iris-
setosa\n38,4.9,3.1,1.5,0.1,Iris-setosa\n39,4.4,3.0,1.3,0.2,Iris-
setosa\n40,5.1,3.4,1.5,0.2,Iris-setosa\n41,5.0,3.5,1.3,0.3,Iris-
setosa\n42,4.5,2.3,1.3,0.3,Iris-setosa\n43,4.4,3.2,1.3,0.2,Iris-
setosa\n44,5.0,3.5,1.6,0.6,Iris-setosa\n45,5.1,3.8,1.9,0.4,Iris-
setosa\n46,4.8,3.0,1.4,0.3,Iris-setosa\n47,5.1,3.8,1.6,0.2,Iris-
setosa\n48,4.6,3.2,1.4,0.2,Iris-setosa\n49,5.3,3.7,1.5,0.2,Iris-
setosa\n50,5.0,3.3,1.4,0.2,Iris-setosa\n51,7.0,3.2,4.7,1.4,Iris-
versicolor\n52,6.4,3.2,4.5,1.5,Iris-versicolor\n53,6.9,3.1,4.9,1.5,Iris-
versicolor\n54,5.5,2.3,4.0,1.3,Iris-versicolor\n55,6.5,2.8,4.6,1.5,Iris-
versicolor\n56,5.7,2.8,4.5,1.3,Iris-versicolor\n57,6.3,3.3,4.7,1.6,Iris-
versicolor\n58,4.9,2.4,3.3,1.0,Iris-versicolor\n59,6.6,2.9,4.6,1.3,Iris-
versicolor\n60,5.2,2.7,3.9,1.4,Iris-versicolor\n61,5.0,2.0,3.5,1.0,Iris-
versicolor\n62,5.9,3.0,4.2,1.5,Iris-versicolor\n63,6.0,2.2,4.0,1.0,Iris-
versicolor\n64,6.1,2.9,4.7,1.4,Iris-versicolor\n65,5.6,2.9,3.6,1.3,Iris-
versicolor\n66,6.7,3.1,4.4,1.4,Iris-versicolor\n67,5.6,3.0,4.5,1.5,Iris-
```

Load the Dataset into pandas dataframe.

> Indented block

```
g,mial
```

Data Preprocessing

```
iris.head()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

```
iris.tail()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

```
iris.describe(include="all")
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Speci
count	150.000000	150.000000	150.000000	150.000000	150.000000	
unique	NaN	NaN	NaN	NaN	NaN	
top	NaN	NaN	NaN	NaN	NaN	I set
freq	NaN	NaN	NaN	NaN	NaN	
mean	75.500000	5.843333	3.054000	3.758667	1.198667	N
std	43.445368	0.828066	0.433594	1.764420	0.763161	N
min	1.000000	4.300000	2.000000	1.000000	0.100000	N
25%	38.250000	5.100000	2.800000	1.600000	0.300000	N
50%	75.500000	5.800000	3.000000	4.350000	1.300000	N
75%	112.750000	6.400000	3.300000	5.100000	1.800000	N

```
iris.shape
```

```
(150, 6)
```

```
iris.columns
```

```
Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
       'Species'],
      dtype='object')
```

```
iris['Id']
```

```
0        1
1        2
2        3
3        4
4        5
       ...
145    146
146    147
147    148
148    149
149    150
Name: Id, Length: 150, dtype: int64
```

```
iris[0:3]
```

```
iris.loc[0:2]
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa

```
iris.loc[0:2,'Id':'PetalWidthCm']
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
0	1	5.1	3.5	1.4	0.2
1	2	4.9	3.0	1.4	0.2
2	3	4.7	3.2	1.3	0.2

```
iris.iloc[1:3]
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa

```
iris.iloc[1:5,1:5]
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Check for missing values in the data using pandas isnull()

```
iris.isnull()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	False	False	False	False	False	False
1	False	False	False	False	False	False
2	False	False	False	False	False	False
3	False	False	False	False	False	False
4	False	False	False	False	False	False
...	...	...	...	...	...	...
145	False	False	False	False	False	False
146	False	False	False	False	False	False
147	False	False	False	False	False	False
148	False	False	False	False	False	False
149	False	False	False	False	False	False

150 rows × 6 columns

```
iris.isna()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	False	False	False	False	False	False
1	False	False	False	False	False	False
2	False	False	False	False	False	False
3	False	False	False	False	False	False
4	False	False	False	False	False	False
...	...	...	...	...	...	...

```
iris.isnull().any()
```

```
Id              False
SepalLengthCm   False
SepalWidthCm    False
PetalLengthCm   False
PetalWidthCm    False
Species         False
dtype: bool
```

```
iris.head
```

```
<bound method NDFrame.head of       Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  \
0      1            5.1           3.5            1.4           0.2
1      2            4.9           3.0            1.4           0.2
2      3            4.7           3.2            1.3           0.2
3      4            4.6           3.1            1.5           0.2
4      5            5.0           3.6            1.4           0.2
..   ...            ...           ...            ...           ...
145  146            6.7           3.0            5.2           2.3
146  147            6.3           2.5            5.0           1.9
147  148            6.5           3.0            5.2           2.0
148  149            6.2           3.4            5.4           2.3
149  150            5.9           3.0            5.1           1.8

            Species
0       Iris-setosa
1       Iris-setosa
2       Iris-setosa
3       Iris-setosa
4       Iris-setosa
..              ...
145  Iris-virginica
146  Iris-virginica
147  Iris-virginica
148  Iris-virginica
149  Iris-virginica

[150 rows x 6 columns]>
```

```
iris.isnull().sum()
```

```
Id              0
SepalLengthCm   0
SepalWidthCm    0
PetalLengthCm   0
PetalWidthCm    0
Species         0
dtype: int64
```

```
iris.head
```

```
<bound method NDFrame.head of       Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  \
0      1            5.1           3.5            1.4           0.2
1      2            4.9           3.0            1.4           0.2
2      3            4.7           3.2            1.3           0.2
3      4            4.6           3.1            1.5           0.2
4      5            5.0           3.6            1.4           0.2
..   ...            ...           ...            ...           ...
145  146            6.7           3.0            5.2           2.3
146  147            6.3           2.5            5.0           1.9
147  148            6.5           3.0            5.2           2.0
148  149            6.2           3.4            5.4           2.3
149  150            5.9           3.0            5.1           1.8

            Species
0       Iris-setosa
1       Iris-setosa
2       Iris-setosa
3       Iris-setosa
4       Iris-setosa
..              ...
145  Iris-virginica
```

```
146  Iris-virginica
147  Iris-virginica
148  Iris-virginica
149  Iris-virginica

[150 rows x 6 columns]>
```

Count of missing values of a specific column

```
iris.SepalLengthCm.isnull().sum()
```

```
0
```

```
iris.tail
```

```
<bound method NDFrame.tail of       Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  \
0      1            5.1           3.5            1.4           0.2
1      2            4.9           3.0            1.4           0.2
2      3            4.7           3.2            1.3           0.2
3      4            4.6           3.1            1.5           0.2
4      5            5.0           3.6            1.4           0.2
..   ...            ...           ...            ...           ...
145  146            6.7           3.0            5.2           2.3
146  147            6.3           2.5            5.0           1.9
147  148            6.5           3.0            5.2           2.0
148  149            6.2           3.4            5.4           2.3
149  150            5.9           3.0            5.1           1.8

            Species
0       Iris-setosa
1       Iris-setosa
2       Iris-setosa
3       Iris-setosa
4       Iris-setosa
..              ...
145  Iris-virginica
146  Iris-virginica
147  Iris-virginica
148  Iris-virginica
149  Iris-virginica

[150 rows x 6 columns]>
```

```
iris.head
```

```
<bound method NDFrame.head of       Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  \
0      1            5.1           3.5            1.4           0.2
1      2            4.9           3.0            1.4           0.2
2      3            4.7           3.2            1.3           0.2
3      4            4.6           3.1            1.5           0.2
4      5            5.0           3.6            1.4           0.2
..   ...            ...           ...            ...           ...
145  146            6.7           3.0            5.2           2.3
146  147            6.3           2.5            5.0           1.9
147  148            6.5           3.0            5.2           2.0
148  149            6.2           3.4            5.4           2.3
149  150            5.9           3.0            5.1           1.8

            Species
0       Iris-setosa
1       Iris-setosa
2       Iris-setosa
3       Iris-setosa
4       Iris-setosa
..              ...
145  Iris-virginica
146  Iris-virginica
147  Iris-virginica
148  Iris-virginica
149  Iris-virginica

[150 rows x 6 columns]>
```

Data Formatting

```
iris.dtypes
```

```
Id                int64
SepalLengthCm    float64
SepalWidthCm     float64
PetalLengthCm    float64
```

```
PetalWidthCm     float64
Species           object
dtype: object
```

```
iris.SepalLengthCm=iris.SepalLengthCm.astype("int")
```

```
iris.dtypes
```

```
Id                int64
SepalLengthCm     int64
SepalWidthCm    float64
PetalLengthCm   float64
PetalWidthCm    float64
Species          object
dtype: object
```

Data Normalization

```
from sklearn import preprocessing
```

```
iris.head()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5	3.5	1.4	0.2	Iris-setosa
1	2	4	3.0	1.4	0.2	Iris-setosa
2	3	4	3.2	1.3	0.2	Iris-setosa
3	4	4	3.1	1.5	0.2	Iris-setosa
4	5	5	3.6	1.4	0.2	Iris-setosa

```
min_max_scaler = preprocessing.MinMaxScaler() #min-max scalar
```

```
x=iris.iloc[:,:4] #separte input from dataset
```

```
iris.head
```

```
<bound method NDFrame.head of       Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  \
0      1              5           3.5            1.4           0.2
1      2              4           3.0            1.4           0.2
2      3              4           3.2            1.3           0.2
3      4              4           3.1            1.5           0.2
4      5              5           3.6            1.4           0.2
..   ...            ...           ...            ...           ...
145  146              6           3.0            5.2           2.3
146  147              6           2.5            5.0           1.9
147  148              6           3.0            5.2           2.0
148  149              6           3.4            5.4           2.3
149  150              5           3.0            5.1           1.8

            Species
0       Iris-setosa
1       Iris-setosa
2       Iris-setosa
3       Iris-setosa
4       Iris-setosa
..              ...
145  Iris-virginica
146  Iris-virginica
147  Iris-virginica
148  Iris-virginica
149  Iris-virginica

[150 rows x 6 columns]>
```

```
x_scaled = min_max_scaler.fit_transform(x)
```

```
iris.head
```

```
<bound method NDFrame.head of       Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  \
0      1              5           3.5            1.4           0.2
1      2              4           3.0            1.4           0.2
2      3              4           3.2            1.3           0.2
3      4              4           3.1            1.5           0.2
4      5              5           3.6            1.4           0.2
..   ...            ...           ...            ...           ...
```

```
145  146            6          3.0           5.2          2.3
146  147            6          2.5           5.0          1.9
147  148            6          3.0           5.2          2.0
148  149            6          3.4           5.4          2.3
149  150            5          3.0           5.1          1.8

            Species
0       Iris-setosa
1       Iris-setosa
2       Iris-setosa
3       Iris-setosa
4       Iris-setosa
..              ...
145  Iris-virginica
146  Iris-virginica
147  Iris-virginica
148  Iris-virginica
149  Iris-virginica

[150 rows x 6 columns]>
```

```
df_normalized = pd.DataFrame(x_scaled)
```

```
iris.tail
```

```
<bound method NDFrame.tail of       Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  \
0      1            5          3.5           1.4          0.2
1      2            4          3.0           1.4          0.2
2      3            4          3.2           1.3          0.2
3      4            4          3.1           1.5          0.2
4      5            5          3.6           1.4          0.2
..   ...          ...          ...           ...          ...
145  146            6          3.0           5.2          2.3
146  147            6          2.5           5.0          1.9
147  148            6          3.0           5.2          2.0
148  149            6          3.4           5.4          2.3
149  150            5          3.0           5.1          1.8

            Species
0       Iris-setosa
1       Iris-setosa
2       Iris-setosa
3       Iris-setosa
4       Iris-setosa
..              ...
145  Iris-virginica
146  Iris-virginica
147  Iris-virginica
148  Iris-virginica
149  Iris-virginica

[150 rows x 6 columns]>
```

```
iris.Id
```

```
0        1
1        2
2        3
3        4
4        5
      ...
145    146
146    147
147    148
148    149
149    150
Name: Id, Length: 150, dtype: int64
```

```
df_normalized
```

	0	1	2	3
0	0.000000	0.333333	0.625000	0.067797
1	0.006711	0.000000	0.416667	0.067797
2	0.013423	0.000000	0.500000	0.050847
3	0.020134	0.000000	0.458333	0.084746
...	...	...	...	...

150 rows × 4 columns

Label Encoding

```
from sklearn import preprocessing
```

```
iris['Species'].unique()
```

```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

```
label_encoder = preprocessing.LabelEncoder()
```

```
iris['Species']= label_encoder.fit_transform(iris['Species'])
```

```
iris['Species'].unique()
```

```
array([0, 1, 2])
```

```
iris.Id
```

```
0        1
1        2
2        3
3        4
4        5
       ...
145    146
146    147
147    148
148    149
149    150
Name: Id, Length: 150, dtype: int64
```

```
iris.Species
```

```
0      0
1      0
2      0
3      0
4      0
      ..
145    2
146    2
147    2
148    2
149    2
Name: Species, Length: 150, dtype: int64
```

```
iris.PetalLengthCm
```

```
0      1.4
1      1.4
2      1.3
3      1.5
4      1.4
      ...
145    5.2
146    5.0
147    5.2
148    5.4
149    5.1
Name: PetalLengthCm, Length: 150, dtype: float64
```