DSBDA PRACTICALS

Name: Devshree Kulkarni
Roll No: -64

Practical No:-1

[36]:
```
# to create this type of file press ctrl+shift+p and search Create New jupyter
↪notebook and click on it.
# You can search dataset on Kaggle.com
import pandas as pd
import numpy as np
```

[37]:
```
df = pd.read_csv("Iris.csv") # we stored iris.csv file in df i.e dataframes or
↪variable name
```

[38]:
```
df
```

[38]:
```
      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   ris-virginica
149   Iris-virgini
```

[39]:

	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 x 6 columns]

[39]:
```python
df.isnull()
```

[150 rows x 6 columns]

[40]:
```python
df.isnull().any()
```

[40]:
```
Id               False
SepalLengthCm    False
SepalWidthCm     False
PetalLengthCm    False
PetalWidthCm     False
Species          False
dtype: bool
```

[41]:
```python
df.dtypes # dtypes stands for data types
```

[41]:
```
Id                 int64
SepalLengthCm    float64
SepalWidthCm     float64
PetalLengthCm    float64
PetalWidthCm     float64
Species           object
dtype: object
```

[42]:
```python
df["Species"].unique()
```

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

[43]:
```

```

[44]:
```
df.dtypes
```

Practical No:-2

Step-I

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

In [17]:

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

In [4]:

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

In [6]:

```
df
```

Out[6]:

	Sno	gender	NationalITy	PlaceofBirth	StageID	GradeID	SectionID	Topic	Sem
0	1	M	KW	KuwaIT	lowerlevel	G-04	A	IT	
1	2	M	KW	KuwaIT	lowerlevel	G-04	A	IT	
2	3	M	KW	KuwaIT	lowerlevel	G-04	A	IT	
3	4	M	KW	KuwaIT	lowerlevel	G-04	A	IT	
4	5	M	KW	KuwaIT	lowerlevel	G-04	A	IT	
...	...	...	...	...	...	...	...	...	
475	476	F	Jordan	Jordan	MiddleSchool	G-08	A	Chemistry	
476	477	F	Jordan	Jordan	MiddleSchool	G-08	A	Geology	
477	478	F	Jordan	Jordan	MiddleSchool	G-08	A	Geology	
478	479	F	Jordan	Jordan	MiddleSchool	G-08	A	History	
479	480	F	Jordan	Jordan	MiddleSchool	G-08	A	History	

480 rows × 18 columns

In [7]:

```
df.head()
```

Out[7]:

	Sno	gender	NationalITy	PlaceofBirth	StageID	GradeID	SectionID	Topic	Semester	R
0	1	M	KW	KuwaIT	lowerlevel	G-04	A	IT	F	
1	2	M	KW	KuwaIT	lowerlevel	G-04	A	IT	F	
2	3	M	KW	KuwaIT	lowerlevel	G-04	A	IT	F	
3	4	M	KW	KuwaIT	lowerlevel	G-04	A	IT	F	
4	5	M	KW	KuwaIT	lowerlevel	G-04	A	IT	F	

In [8]:

```
df.tail()
```

Out[8]:

	Sno	gender	NationalITy	PlaceofBirth	StageID	GradeID	SectionID	Topic	Sem
475	476	F	Jordan	Jordan	MiddleSchool	G-08	A	Chemistry	
476	477	F	Jordan	Jordan	MiddleSchool	G-08	A	Geology	
477	478	F	Jordan	Jordan	MiddleSchool	G-08	A	Geology	
478	479	F	Jordan	Jordan	MiddleSchool	G-08	A	History	
479	480	F	Jordan	Jordan	MiddleSchool	G-08	A	History	

In [9]:

```
df.describe()
```

Out[9]:

	Sno	raisedhands	VisITedResources	AnnouncementsView	Discussion
count	480.000000	480.000000	480.000000	480.000000	478.000000
mean	240.500000	46.775000	54.797917	38.462500	43.278243
std	138.708327	30.779223	33.080007	30.095579	27.646238
min	1.000000	0.000000	0.000000	0.000000	1.000000
25%	120.750000	15.750000	20.000000	14.000000	20.000000
50%	240.500000	50.000000	65.000000	33.000000	39.000000
75%	360.250000	75.000000	84.000000	58.000000	70.000000
max	480.000000	100.000000	99.000000	350.000000	99.000000

In [10]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 480 entries, 0 to 479
Data columns (total 18 columns):
 #   Column                    Non-Null Count  Dtype
---                            --------------  -----
 0   Sno                       480 non-null    int64
 1   gender                    480 non-null    object
 2   NationalITy               480 non-null    object
 3   PlaceofBirth              480 non-null    object
 4   StageID                   480 non-null    object
 5   GradeID                   480 non-null    object
 6   SectionID                 480 non-null    object
 7   Topic                     480 non-null    object
 8   Semester                  480 non-null    object
 9   Relation                  480 non-null    object
 10  raisedhands               480 non-null    int64
 11  VisITedResources          480 non-null    int64
 12  AnnouncementsView         480 non-null    int64
 13  Discussion                478 non-null    float64
 14  ParentAnsweringSurvey     480 non-null    object
 15  ParentschoolSatisfaction  480 non-null    object
 16  StudentAbsenceDays        480 non-null    object
 17  Class                     480 non-null    object
dtypes: float64(1), int64(4), object(13)
memory usage: 67.6+ KB
```

In [11]:

```
df.shape
```

Out[11]:

```
(480, 18)
```

In [12]:

```
df.isnull().any().any()
```

Out[12]:

```
True
```

In [13]:

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

Out[13]:

```
Sno                        0
gender                     0
NationalITy                0
PlaceofBirth               0
StageID                    0
GradeID                    0
SectionID                  0
Topic                      0
Semester                   0
Relation                   0
raisedhands                0
VisITedResources           0
AnnouncementsView          0
Discussion                 2
ParentAnsweringSurvey      0
ParentschoolSatisfaction   0
StudentAbsenceDays         0
Class                      0
dtype: int64
```

In [14]:

```
avg_val = df["Discussion"].astype("float").mean()
avg_val
```

Out[14]:

```
43.27824267782427
```

In [15]:

```
df["Discussion"].replace(np.NaN, avg_val, inplace=True)
```

In [16]:

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

Out[16]:

```
Sno                         0
gender                      0
NationalITy                 0
PlaceofBirth                0
StageID                     0
GradeID                     0
SectionID                   0
Topic                       0
Semester                    0
Relation                    0
raisedhands                 0
VisITedResources            0
AnnouncementsView           0
Discussion                  0
ParentAnsweringSurvey       0
ParentschoolSatisfaction    0
StudentAbsenceDays          0
Class                       0
dtype: int64
```

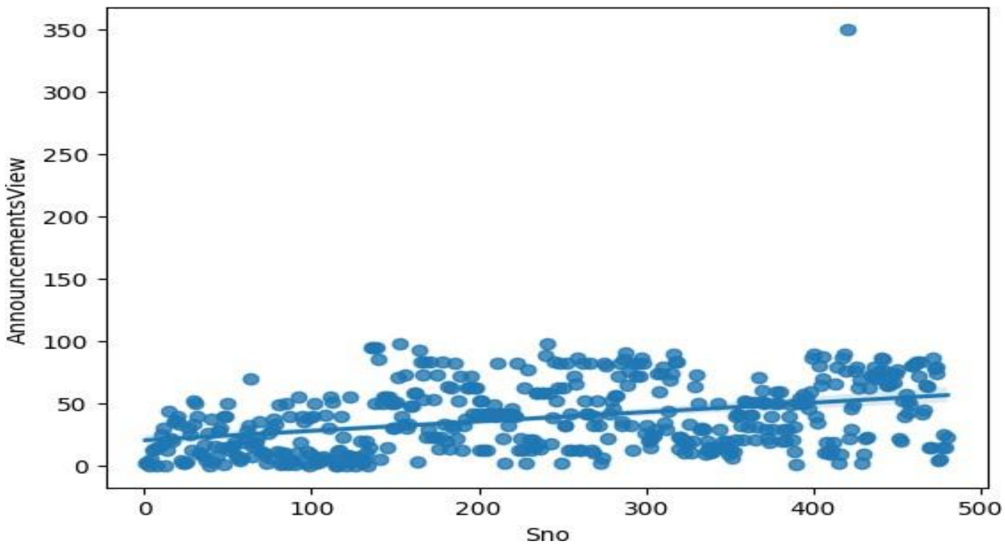
Step-II

Scan all numeric variables for outliers. If there are outliers, use any of the suitable techniques to deal with them.

In [20]:

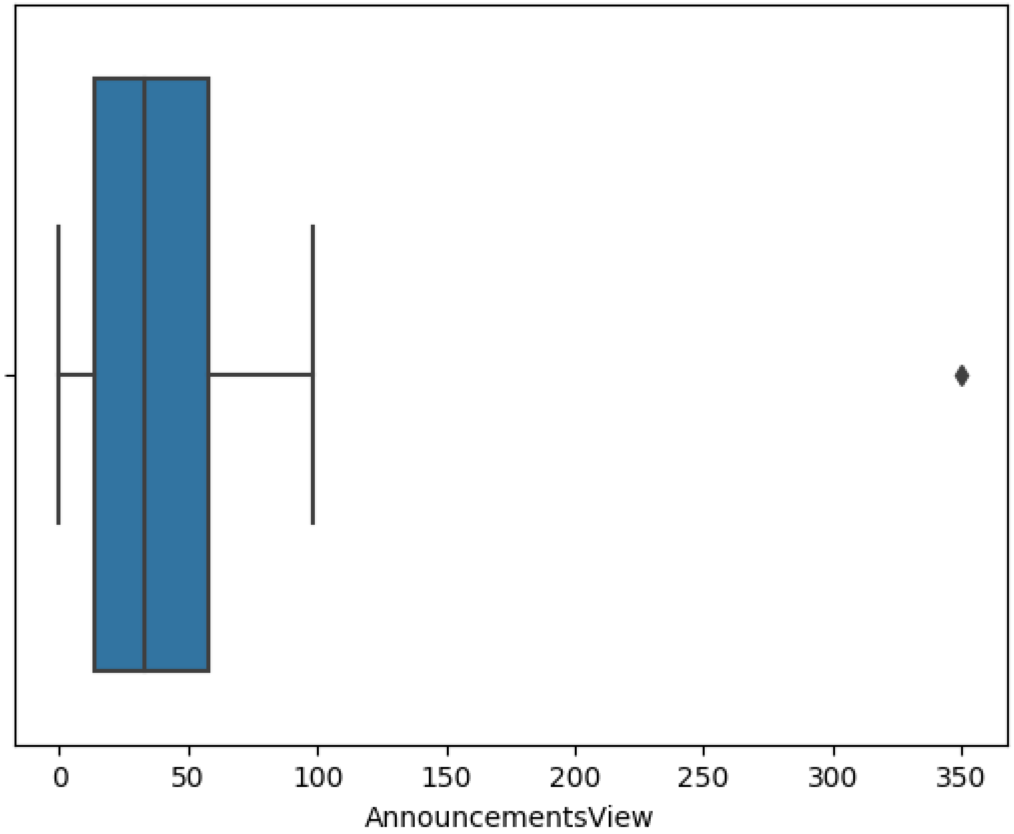
```
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats
```

```
sns.regplot(x='Sno', y='AnnouncementsView', data=df)
plt.show()
```



In [47]:

```
sns.boxplot(x=df['AnnouncementsView'])
plt.show()
```



In [57]:

```
z = np.abs(stats.zscore(df['AnnouncementsView']))
print(z)
```

```
0      1.212821
1      1.179559
2      1.279345
3      1.113034
4      0.880199
         ...
475    1.113034
476    0.813675
477    0.447792
478    0.813675
479    0.514316
Name: AnnouncementsView, Length: 480, dtype: float64
```

In [51]:

```
threshold = 3
print(np.where(z > 3))
```

```
(array([419], dtype=int64),)
```

In [60]:

```
z[419]
```

Out[60]:

```
10.3624031636167
```

The standard z score is calculated by dividing the difference from the mean by the standard deviation. The modified z score is calculated from the mean absolute deviation (MeanAD) or median absolute deviation (MAD). These values must be multiplied by a constant to approximate the standard deviation.

Step-III

Apply data transformations on at least one of the variables

In [62]:

```
df1 = pd.DataFrame({ 'Income': [15000, 1800, 120000, 10000],
                    'Age': [25, 18, 42, 51],
                    'Department': ['HR','Legal','Marketing','Management']})
```

In [63]:

```
df1
```

Out[63]:

	Income	Age	Department
0	15000	25	HR
1	1800	18	Legal
2	120000	42	Marketing
3	10000	51	Management

In [65]:

```
df1_scaled = df1.copy()
col_names = ['Income', 'Age']
features = df1_scaled[col_names]
```

In [67]:

```
features
```

Out[67]:

	Income	Age
0	15000	25
1	1800	18
2	120000	42
3	10000	51

In [70]:

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df1_scaled[col_names] = scaler.fit_transform(features.values)
```

scikit-learn

- scikit-learn is an open-source Python library that implements a range of machine learning, pre-processing, cross-validation, and visualization algorithms using a unified interface.
- Important features of scikit-learn:
  - Simple and efficient tools for data mining and data analysis. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means, etc.
  - Accessible to everybody and reusable in various contexts.
  - Built on the top of NumPy, SciPy, and matplotlib.
  - Open source, commercially usable – BSD license.

In [71]:

```
print(df1_scaled[col_names])
```

```
     Income       Age
0  0.111675  0.212121
1  0.000000  0.000000
2  1.000000  0.727273
3  0.069374  1.000000
```

```
There is another way of data scaling, where the minimum of feature is made equal to zero
and the maximum of feature equal to one. MinMax Scaler shrinks the data within the given
range, usually of 0 to 1. It transforms data by scaling features to a given range. It
scales the values to a specific value range without changing the shape of the original
distribution.

The MinMax scaling is done using:

x_std = (x – x.min(axis=0)) / (x.max(axis=0) – x.min(axis=0))

x_scaled = x_std * (max – min) + min

Where,

min, max = feature_range
x.min(axis=0) : Minimum feature value
x.max(axis=0):Maximum feature value
Sklearn preprocessing defines MinMaxScaler() method to achieve this.
```

In [ ]:

```
```

Practical No:-3

In [1]:

```
import numpy as np
import pandas as pd
import statistics as st
```

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.
- Provide the codes with outputs and explain everything that you do in this step.

1. Summary statistics
2. Types of Variables
3. Summary statistics of income grouped by the age groups
4. Display basic statistical details on the iris dataset.

In [2]:

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

In [3]:

```
df
```

Out[3]:

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40
...	...	...	...	...	...
195	196	Female	35	120	79
196	197	Female	45	126	28
197	198	Male	32	126	74
198	199	Male	32	137	18
199	200	Male	30	137	83

200 rows × 5 columns

1. Summary statistics

1. Mean()

In [5]:

```
df.mean()  # mean of all columns
```

```
C:\Users\COMPHOD\AppData\Local\Temp\ipykernel_12048\734542734.py:1: Future
Warning: The default value of numeric_only in DataFrame.mean is deprecate
d. In a future version, it will default to False. In addition, specifying
'numeric_only=None' is deprecated. Select only valid columns or specify th
e value of numeric_only to silence this warning.
  df.mean()  # mean of all columns
```

Out[5]:

```
CustomerID                100.50
Age                        38.85
Annual Income (k$)         60.56
Spending Score (1-100)     50.20
dtype: float64
```

In [6]:

```
df.loc[:,'Age'].mean()   # mean of specific column
```

Out[6]:

```
38.85
```

In [10]:

```
df.mean(axis=1)[0:4] # mean row wise
```

```
C:\Users\COMPHOD\AppData\Local\Temp\ipykernel_12048\1227886012.py:1: Futur
eWarning: Dropping of nuisance columns in DataFrame reductions (with 'nume
ric_only=None') is deprecated; in a future version this will raise TypeErr
or. Select only valid columns before calling the reduction.
  df.mean(axis=1)[0:4] # mean row wise
```

Out[10]:

```
0    18.50
1    29.75
2    11.25
3    30.00
dtype: float64
```

2. Median

In [11]:

```
df.median() # median of all columns
```

```
C:\Users\COMPHOD\AppData\Local\Temp\ipykernel_12048\3838006088.py:1: Futur
eWarning: The default value of numeric_only in DataFrame.median is depreca
ted. In a future version, it will default to False. In addition, specifyin
g 'numeric_only=None' is deprecated. Select only valid columns or specify
the value of numeric_only to silence this warning.
  df.median() # median of all columns
```

Out[11]:

```
CustomerID                100.5
Age                        36.0
Annual Income (k$)         61.5
Spending Score (1-100)     50.0
dtype: float64
```

In [12]:

```
df.loc[:,'Age'].median() # median of specific column
```

Out[12]:

```
36.0
```

In [13]:

```
df.median(axis=1)[0:4] #median row wise
```

```
C:\Users\COMPHOD\AppData\Local\Temp\ipykernel_12048\2735118674.py:1: Futur
eWarning: Dropping of nuisance columns in DataFrame reductions (with 'nume
ric_only=None') is deprecated; in a future version this will raise TypeErr
or. Select only valid columns before calling the reduction.
  df.median(axis=1)[0:4] #median row wise
```

Out[13]:

```
0    17.0
1    18.0
2    11.0
3    19.5
dtype: float64
```

3. Mode

In [14]:

```
df.mode() # mode of all columns
```

Out[14]:

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Female	32.0	54.0	42.0
1	2	NaN	NaN	78.0	NaN
2	3	NaN	NaN	NaN	NaN
3	4	NaN	NaN	NaN	NaN
4	5	NaN	NaN	NaN	NaN
...	...	...	...	...	...
195	196	NaN	NaN	NaN	NaN
196	197	NaN	NaN	NaN	NaN
197	198	NaN	NaN	NaN	NaN
198	199	NaN	NaN	NaN	NaN
199	200	NaN	NaN	NaN	NaN

200 rows × 5 columns

In [15]:

```
df.loc[:,'Age'].mode() # mode of a specific column.
```

Out[15]:

```
0    32
Name: Age, dtype: int64
```

4. Minimum

In [16]:

```
df.min() # minimum of all columns
```

Out[16]:

```
CustomerID                     1
Genre                     Female
Age                           18
Annual Income (k$)            15
Spending Score (1-100)         1
dtype: object
```

In [17]:

```
df.loc[:,'Age'].min(skipna = False)  # minimum of Specific column
```

Out[17]:

```
18
```

5.Maximum

In [18]:

```
df.max() # Maximum of all columns
```

Out[18]:

```
CustomerID                 200
Genre                     Male
Age                         70
Annual Income (k$)         137
Spending Score (1-100)      99
dtype: object
```

In [19]:

```
df.loc[:,'Age'].max(skipna = False) # Maximum of Specific column
```

Out[19]:

```
70
```

6. Standard Deviation

In [20]:

```
df.std() # Standard Deviation of all columns
```

```
C:\Users\COMPHOD\AppData\Local\Temp\ipykernel_12048\301237031.py:1: Future
Warning: The default value of numeric_only in DataFrame.std is deprecated.
In a future version, it will default to False. In addition, specifying 'nu
meric_only=None' is deprecated. Select only valid columns or specify the v
alue of numeric_only to silence this warning.
  df.std() # Standard Deviation of all columns
```

Out[20]:

```
CustomerID                57.879185
Age                       13.969007
Annual Income (k$)        26.264721
Spending Score (1-100)    25.823522
dtype: float64
```

In [21]:

```
df.loc[:,'Age'].std() # Standard Deviation of specific column
```

Out[21]:

```
13.96900733155888
```

In [22]:

```
df.std(axis=1)[0:4] # Standard Deviation row wise
```

```
C:\Users\COMPHOD\AppData\Local\Temp\ipykernel_12048\1639279273.py:1: Futur
eWarning: Dropping of nuisance columns in DataFrame reductions (with 'nume
ric_only=None') is deprecated; in a future version this will raise TypeErr
or. Select only valid columns before calling the reduction.
  df.std(axis=1)[0:4] # Standard Deviation row wise
```

Out[22]:

```
0    15.695010
1    35.074920
2     8.057088
3    32.300671
dtype: float64
```

2. Types of Variables:

A variable is a characteristic that can be measured and that can assume different values. Height, age, income, province or country of birth, grades obtained at school and type of housing are all examples of variables. Variables may be classified into two main categories:

1. Categorical and
2. Numeric.

Each category is then classified in two subcategories: nominal or ordinal for categorical variables, discrete or continuous for numeric variables.

3. Summary statistics of income grouped by the age groups

Problem Statement:

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.

1. Categorical Variable: Genre
2. Quantitative Variable : Age

In [24]:

```
df.groupby(['Genre'])['Age'].mean()
```

Out[24]:

```
Genre
Female    38.098214
Male      39.806818
Name: Age, dtype: float64
```

1. Categorical Variable: Genre
2. Quantitative Variable : Income

In [43]:

```
df_u=df.rename(columns= {'Annual Income (k$)':'Income'}, inplace= False)
```

In [46]:

```
df_u
```

Out[46]:

	CustomerID	Genre	Age	Income	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40
...	...	...	...	...	...
195	196	Female	35	120	79
196	197	Female	45	126	28
197	198	Male	32	126	74
198	199	Male	32	137	18
199	200	Male	30	137	83

200 rows × 5 columns

In [45]:

```
df_u.groupby(['Genre']).Income.mean()
```

Out[45]:

```
Genre
Female    59.250000
Male      62.227273
Name: Income, dtype: float64
```

To create a list that contains a numeric value for each response to the categorical variable

- One hot encoding is a technique used to represent categorical variables as numerical values in a machine learning model.
- In this technique, the categorical parameters will prepare separate columns for both Male and Female labels. So, wherever there is Male, the value will be 1 in Male column and 0 in Female column, and vice-versa.

In [47]:

```
from sklearn import preprocessing
enc = preprocessing.OneHotEncoder()
enc_df = pd.DataFrame(enc.fit_transform(df[['Genre']]).toarray())
enc_df
```

Out[47]:

	0	1
0	0.0	1.0
1	0.0	1.0
2	1.0	0.0
3	1.0	0.0
4	1.0	0.0
...	...	...
195	1.0	0.0
196	1.0	0.0
197	0.0	1.0
198	0.0	1.0
199	0.0	1.0

200 rows × 2 columns

To concat numerical list to dataframe

In [48]:

```
df_encode =df_u.join(enc_df)
df_encode
```

Out[48]:

	CustomerID	Genre	Age	Income	Spending Score (1-100)	0	1
0	1	Male	19	15	39	0.0	1.0
1	2	Male	21	15	81	0.0	1.0
2	3	Female	20	16	6	1.0	0.0
3	4	Female	23	16	77	1.0	0.0
4	5	Female	31	17	40	1.0	0.0
...	...	...	...	...	...	...	...
195	196	Female	35	120	79	1.0	0.0
196	197	Female	45	126	28	1.0	0.0
197	198	Male	32	126	74	0.0	1.0
198	199	Male	32	137	18	0.0	1.0
199	200	Male	30	137	83	0.0	1.0

200 rows × 7 columns

4. Display basic statistical details on the iris dataset.

In [49]:

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

In [64]:

```
df_iris = pd.read_csv("Iris.csv")
df_iris.head()
```

Out[64]:

	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	NaN	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

Basic statistical details of Iris dataset

In [69]:

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

```
Iris-setosa
             Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
count  50.00000       50.00000     50.000000      49.000000      50.00000
mean   25.50000        5.00600      3.418000       1.467347       0.24400
std    14.57738        0.35249      0.381024       0.173671       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       1.600000       0.30000
max    50.00000        5.80000      4.400000       1.900000       0.60000

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

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
```

In [70]:

```
df_iris.dtypes.value_counts()
```

Out[70]:

```
float64    4
int64      1
object     1
dtype: int64
```

In [ ]:

In [ ]:

Practical No:-4

Create a Linear Regression Model using Python/R to predict home prices using Boston Housing Dataset (https://www.kaggle.com/c/boston-housing (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.

The objective is to predict the value of prices of the house using the given features.

```
The Boston Housing Dataset

The Boston Housing Dataset is a derived from information collected by the U.S. Census
Service concerning housing in the area of Boston MA. The following describes the dataset
columns:

CRIM - per capita crime rate by town
ZN - proportion of residential land zoned for lots over 25,000 sq.ft.
INDUS - proportion of non-retail business acres per town.
CHAS - Charles River dummy variable (1 if tract bounds river; 0 otherwise)
NOX - nitric oxides concentration (parts per 10 million)
RM - average number of rooms per dwelling
AGE - proportion of owner-occupied units built prior to 1940
DIS - weighted distances to five Boston employment centres
RAD - index of accessibility to radial highways
TAX - full-value property-tax rate per $10,000
PTRATIO - pupil-teacher ratio by town
B - 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
LSTAT - % lower status of the population
MEDV - Median value of owner-occupied homes in $1000's
```

In [1]:

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

In [2]:

```
# Importing DataSet and take a look at Data
Boston = pd.read_csv("Boston.csv")
Boston.head()
```

Out[2]:

	Unnamed: 0	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO
0	1	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3
1	2	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8
2	3	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8
3	4	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7
4	5	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7

In [12]:

```
Boston.info()
Boston.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 15 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Unnamed: 0  506 non-null    int64  
 1   CRIM        506 non-null    float64
 2   ZN          506 non-null    float64
 3   INDUS       506 non-null    float64
 4   CHAS        506 non-null    int64  
 5   NOX         506 non-null    float64
 6   RM          506 non-null    float64
 7   AGE         506 non-null    float64
 8   DIS         506 non-null    float64
 9   RAD         506 non-null    int64  
 10  TAX         506 non-null    int64  
 11  PTRATIO     506 non-null    float64
 12  BLACK       506 non-null    float64
 13  LSTAT       506 non-null    float64
 14  MEDV        506 non-null    float64
dtypes: float64(11), int64(4)
memory usage: 59.4 KB
```

Out[12]:

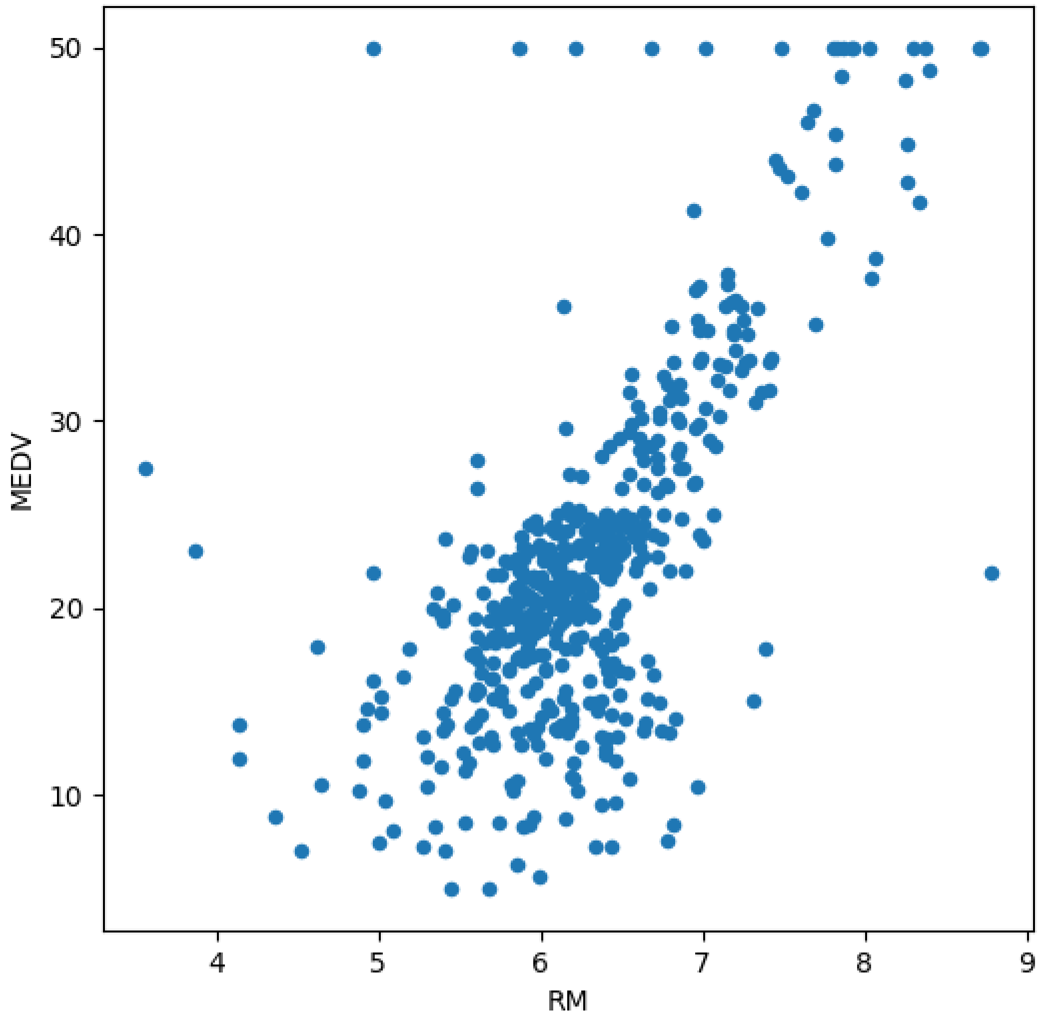
	Unnamed: 0	CRIM	ZN	INDUS	CHAS	NOX	RM
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	253.500000	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634
std	146.213884	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617
min	1.000000	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000
25%	127.250000	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500
50%	253.500000	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500
75%	379.750000	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500
max	506.000000	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000

In

```
Boston.plot.scatter('RM', 'MEDV', figsize=(6, 6))
```

Out[4]:

```
<Axes: xlabel='RM', ylabel='MEDV'>
```



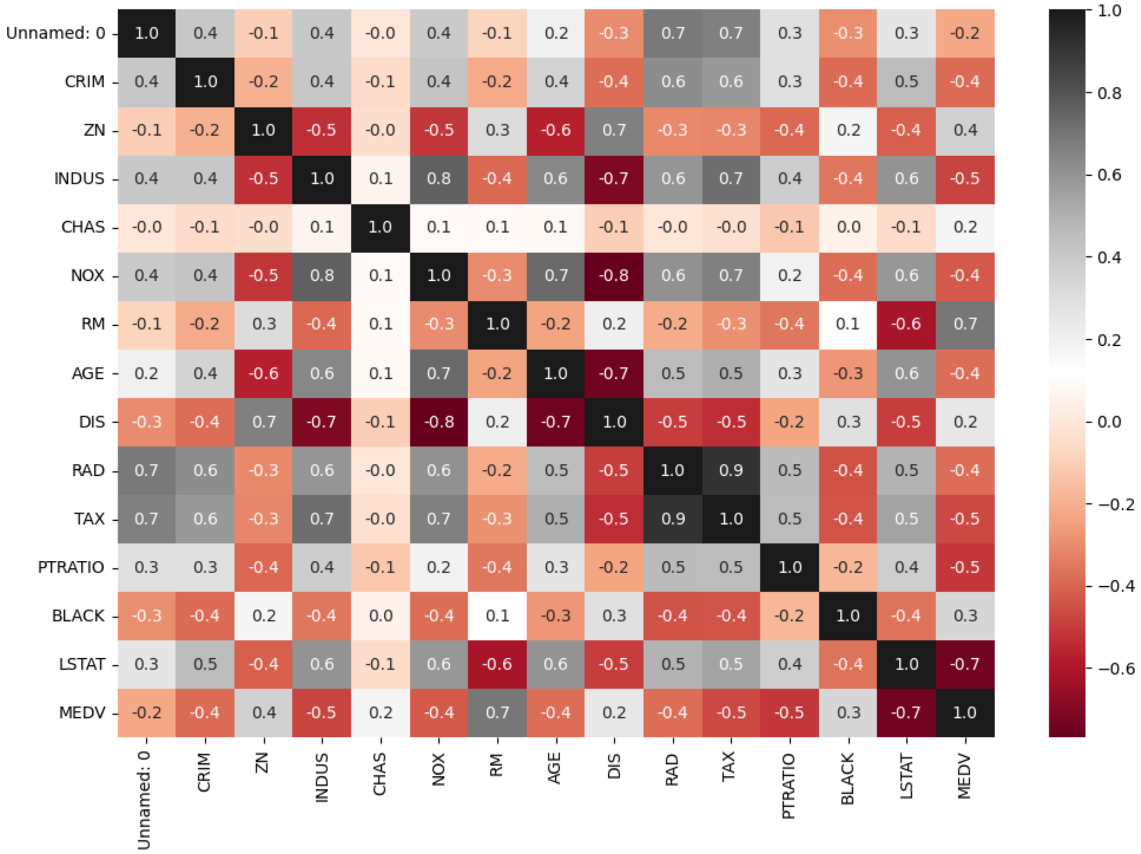
In this plot its clearly to see a linear pattern. Wheter more average number of rooms per dwelling, more expensive the median value is.

In

```
plt.subplots(figsize=(12,8))
sns.heatmap(Boston.corr(), cmap = 'RdGy', annot = True, fmt = '.1f')
```

Out[5]:

<Axes: >



```
At this heatmap plot, we can do our analysis better than the pairplot.
Lets focus at the last line, where y = MEDV:
When shades of Red/Orange: the more red the color is on X axis, smaller the MEDV.
Negative correlation
When light colors: those variables at axis x and y, they dont have any relation. Zero
correlation
When shades of Gray/BLACK : the more BLACK the color is on X axis, more higher the value
medv is. Positive correlation
```

Trainning Linear Regression Model

Define X and Y

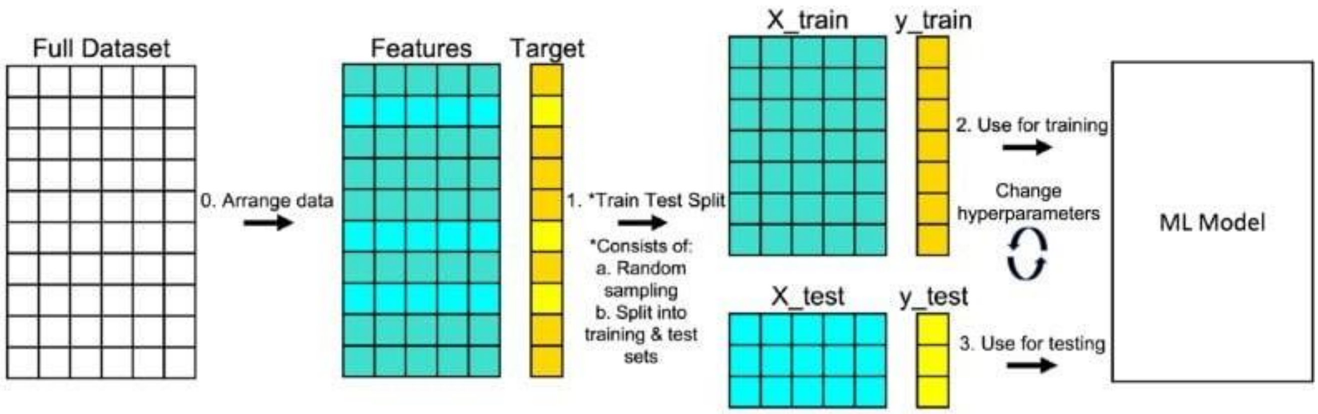
- X: Varibles named as predictors, independent variables, features.
- Y: Variable named as response or dependent variable

In [6]:

```
X = Boston[['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX','PTRAT
Y = Boston['MEDV']
```

Import sklearn librarys:

```
train_test_split, to split our data in two DF, one for build a model and other to
validate.
LinearRegression, to apply the linear regression.
```



In [7]:

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

In [8]:

```
# Split DataSet
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3,random_state=0)
```

In [14]:

```
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: (354, 13), Y: (354,)
Test Dataset Size - X: (152, 13), Y: (152,)
```

In [16]:

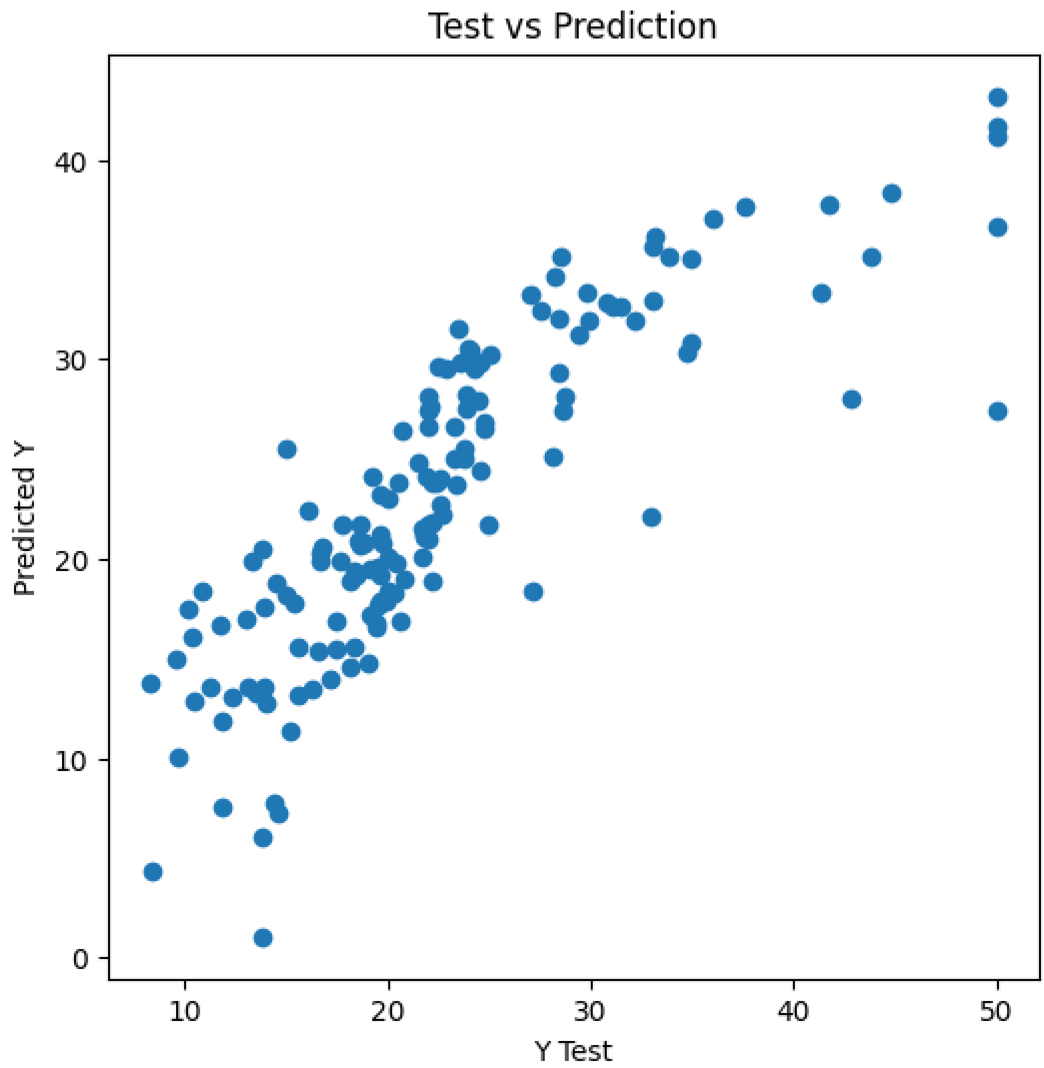
```
# Model Building
lm = LinearRegression()
lm.fit(X_train,Y_train)
predictions = lm.predict(X_test)
```

In [22]:

```
# Model Visualization
plt.figure(figsize=(6, 6))
plt.scatter(Y_test, predictions)
plt.xlabel('Y Test')
plt.ylabel('Predicted Y')
plt.title('Test vs Prediction')
```

Out[22]:

```
Text(0.5, 1.0, 'Test vs Prediction')
```

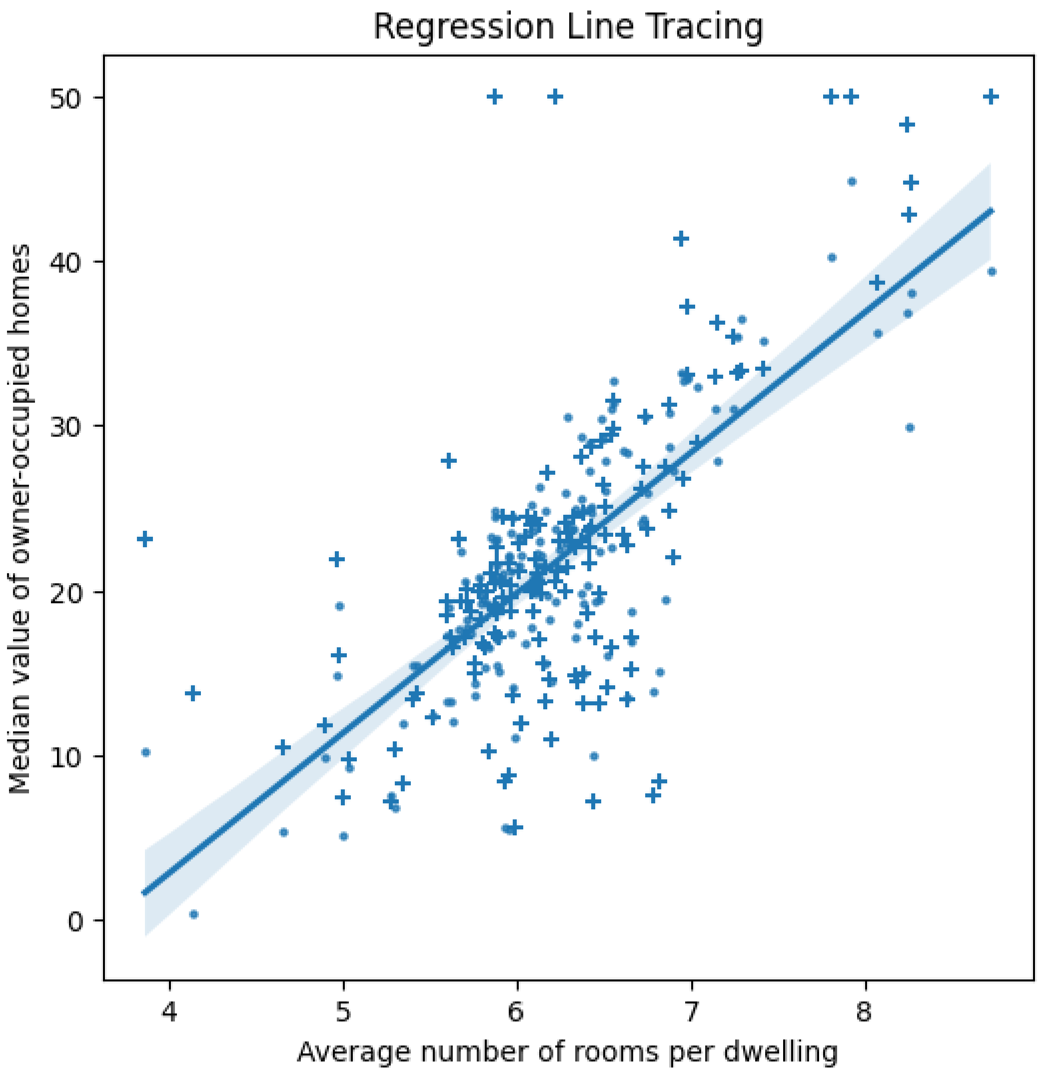


In [21]:

```
plt.figure(figsize=(6, 6))
sns.regplot(x = X_test['RM'], y = predictions, scatter_kws={'s':5})
plt.scatter(X_test['RM'], Y_test, marker = '+')
plt.xlabel('Average number of rooms per dwelling')
plt.ylabel('Median value of owner-occupied homes')
plt.title('Regression Line Tracing')
```

Out[21]:

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



In [23]:

```
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(Y_test, predictions))
print('Mean Square Error:', metrics.mean_squared_error(Y_test, predictions))
print('Root Mean Square Error:', np.sqrt(metrics.mean_squared_error(Y_test, predictions)))
```

```
Mean Absolute Error: 3.609904060381827
Mean Square Error: 27.195965766883510
Root Mean Square Error: 5.2149751453754325
```

In [13]:

```
# Model Coefficients
coefficients = pd.DataFrame(lm.coef_.round(2), X.columns)
coefficients.columns = ['coefficients']
coefficients
```

Out[13]:

	coefficients
CRIM	-0.12
ZN	0.04
INDUS	0.01
CHAS	2.51
NOX	-16.23
RM	3.86
AGE	-0.01
DIS	-1.50
RAD	0.24
TAX	-0.01
PTRATIO	-1.02
BLACK	0.01
LSTAT	-0.49

In [ ]:

Practical No:-5

Data Analyitcs 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..

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
df = pd.read_csv('Social_Network_Ads.csv')
df.head()
```

Out[2]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

In [3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   User ID          400 non-null    int64
 1   Gender           400 non-null    object
 2   Age              400 non-null    int64
 3   EstimatedSalary  400 non-null    int64
 4   Purchased        400 non-null    int64
dtypes: int64(4), object(1)
memory usage: 15.8+ KB
```

In [4]:
```
df.describe()
```

Out[4]:

	User ID	Age	EstimatedSalary	Purchased
count	4.000000e+02	400.000000	400.000000	400.000000
mean	1.569154e+07	37.655000	69742.500000	0.357500
std	7.165832e+04	10.482877	34096.960282	0.479864
min	1.556669e+07	18.000000	15000.000000	0.000000
25%	1.562676e+07	29.750000	43000.000000	0.000000
50%	1.569434e+07	37.000000	70000.000000	0.000000
75%	1.575036e+07	46.000000	88000.000000	1.000000
max	1.581524e+07	60.000000	150000.000000	1.000000

In [5]:
```
X = df[['Age', 'EstimatedSalary']]
Y = df['Purchased']
```

StandardScaler

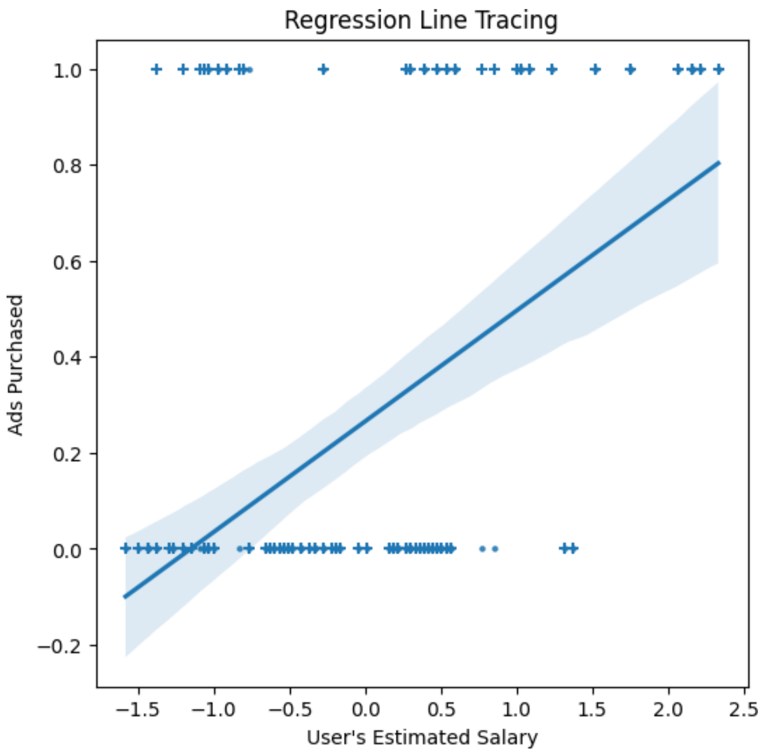
Standardization doesn't have any fixed minimum or maximum value. Here, the values of all the columns are scaled in such a way that they all have a mean equal to 0 and standard deviation equal to 1. This scaling technique works well with outliers. Thus, this technique is preferred if outliers are present in the dataset.

In [6]:
```
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,)
```

In [8]:
```
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')
```

Out[8]:
```
Text(0.5, 1.0, 'Regression Line Tracing')
```



What is a classification report?

As the name suggests, it is the report which explains everything about the classification. This is the summary of the quality of classification made by the constructed ML model. It comprises mainly 5 columns and (N+3) rows. The first column is the class label's name and followed by Precision, Recall, F1-score, and Support. N rows are for N class labels and other three rows are for accuracy, macro average, and weighted average.

Precision:

It is calculated with respect to the predicted values. For class-A, out of total predictions how many were really belong to class-A in actual dataset, is defined as the precision. It is the ratio of [i][i] cell of confusion matrix and sum of the [i] column.

Recall:

It is calculated with respect to the actual values in dataset. For class-A, out of total entries in dataset, how many were actually classified in class-A by the ML model, is defined as the recall. It is the ratio of [i][i] cell of confusion matrix and sum of the [i] row.

F1-score:

It is the harmonic mean of precision and recall.

Support:

It is the total entries of each class in the actual dataset. It is simply the sum of rows for every class-i.

Confusion matrix

In [9]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
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''')
cr = classification_report(Y_test, predictions)
print('Classification report : \n', cr)
```

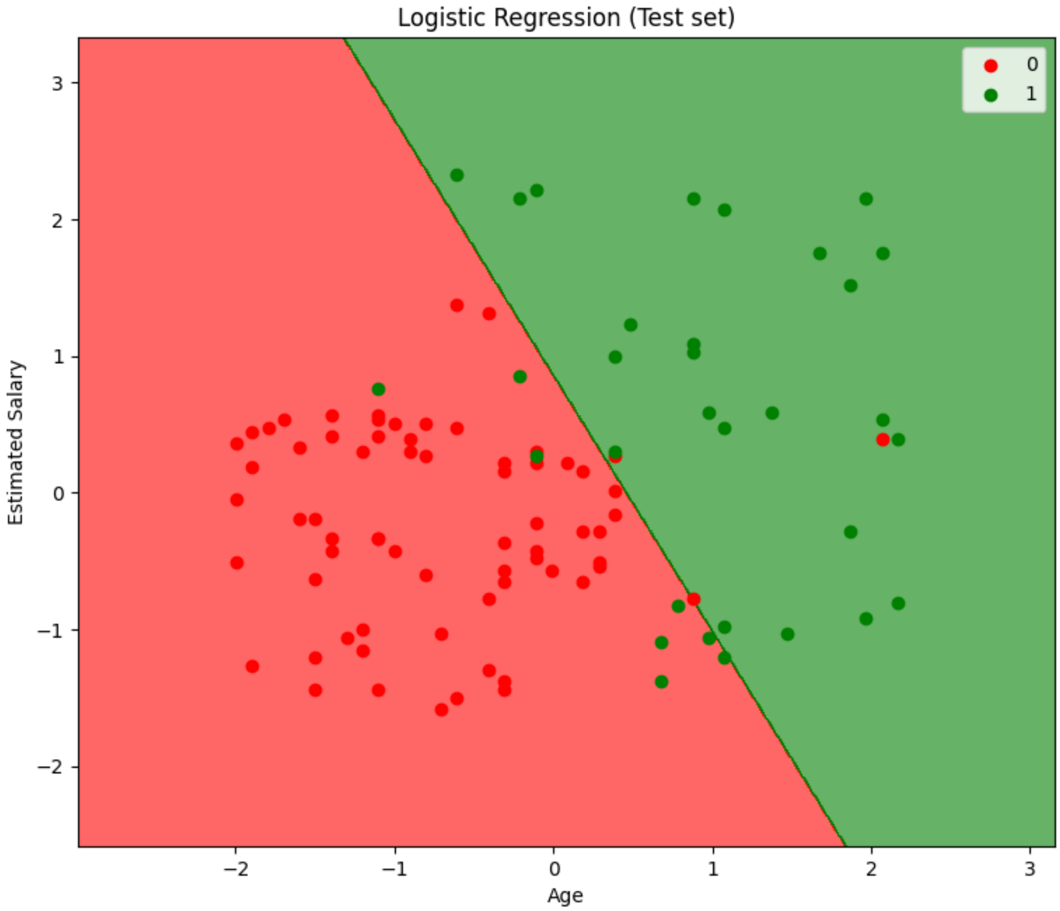
```
Confusion matrix :

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

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
```

In [10]:

```
# Visualizing the Training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, Y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.figure(figsize=(9, 7.5))
plt.contourf(X1, X2, lm.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
alpha = 0.6, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                color = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Logistic Regression (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

In [36]:
```
# Visualizing the Test set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, Y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.figure(figsize=(9, 7.5))
plt.contourf(X1, X2, lm.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
            alpha = 0.6, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(),X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Logistic Regression (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
C:\Users\COMPHOD\AppData\Local\Temp\ipykernel_11192\1618129411.py:12: UserWarning: *c* argument looks like a singl
e numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length m
atches with *x* & *y*. Please use the *color* keyword-argument or provide a 2D array with a single row if you int
end to specify the same RGB or RGBA value for all points.
  plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
```



Practical No:-6

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.

In [8]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
df = pd.read_csv('iris.csv')
df.head()
```

Out[8]:

	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	1.4	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

In [9]:
```
df.info()
```

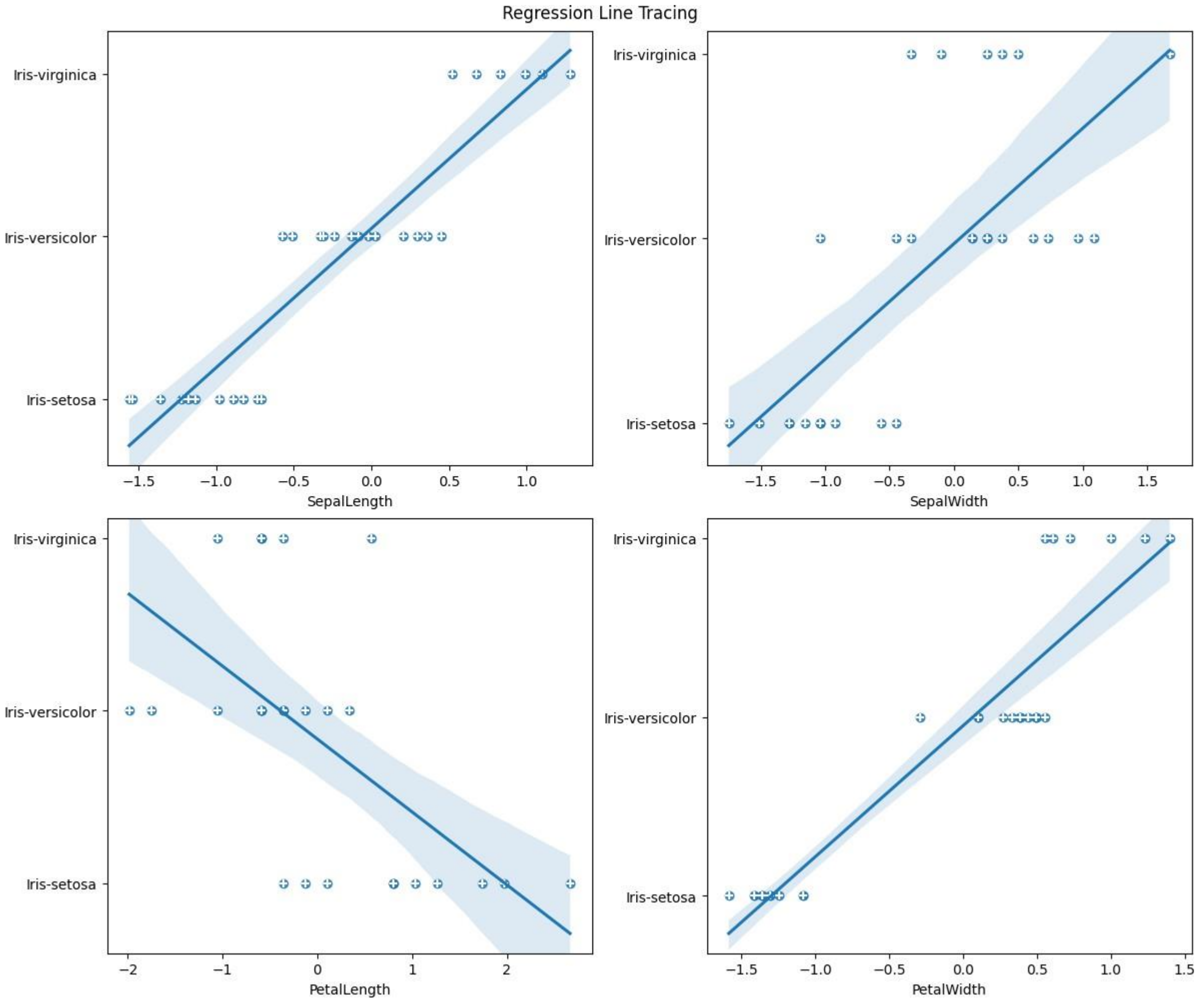
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             150 non-null    int64  
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   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
```

In [10]:
```
X = df.iloc[:, :4].values
Y = df['Species'].values
```

In [11]:
```
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 = 0.2, 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: (120, 4), Y: (120,)
Test Dataset Size - X: (30, 4), Y: (30,)
```

In [14]:
```
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, Y_train)
predictions = classifier.predict(X_test)
mapper = {'Iris-setosa': 0, 'Iris-versicolor': 1, 'Iris-virginica': 2}
predictions_ = [mapper[i] for i in predictions]
fig, axs = plt.subplots(2, 2, figsize = (12, 10), constrained_layout = True)
fig.suptitle('Regression Line Tracing')
for i in range(4):
    x, y = i // 2, i % 2
    sns.regplot(x = X_test[:, i], y = predictions_, ax=axs[x, y])
    axs[x, y].scatter(X_test[:, i][::-1], Y_test[::-1], marker = '+', color="white")
    axs[x, y].set_xlabel(df.columns[i + 1][:-2])
```



Confusion matrix

In [16]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
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''')
cm = classification_report(Y_test, predictions)
print('Classification report : \n', cm)
```

```
Confusion matrix :

| Positive Prediction    | Negative Prediction
---------------+------------------------+----------------------
Positive Class | True Positive (TP) 11  | False Negative (FN) 0
---------------+------------------------+----------------------
Negative Class | False Positive (FP) 0 | True Negative (TN) 13

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
```

Practical No:-7

In [1]:
```
#Download the required packages
import nltk
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package punkt to /home/student/nltk_dat
a...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     /home/student/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /home/student/nltk_dat
a...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /home/student/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

Out[1]: True

In [2]:
```
#Initialize the text
#Sentence Tokenization
text= "Tokenization is the first step in text analytics.
The process of breaking down a text paragraph into smaller
chunks such as words or sentences is called Tokenization."
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  ch
unks such as words or sentences is called Tokenization.']
```

In [3]:
```
#Word Tokenization
from nltk.tokenize import word_tokenize
tokenized_word=word_tokenize(text)
print(tokenized_word)
```

```
['Tokenization', 'is', 'the', 'first', 'step', 'in', 'text', 'anal
ytics', '.', 'The', 'process', 'of', 'breaking', 'down', 'a', 'tex
t', 'paragraph', 'into', 'smaller', 'chunks', 'such', 'as', 'words
', 'or', 'sentences', 'is', 'called', 'Tokenization', '.']
```

In [4]:
```
# print stop words of English
from nltk.corpus import stopwords
stop_words=set(stopwords.words("english"))
print(stop_words)
```

```
{'who', 'has', 'which', 'over', 'himself', 'at', "she's", 'because
', 'won', "haven't", 'most', "don't", 'hasn', 'can', 'wouldn', "di
dn't", 'than', 'we', 'me', 'she', "doesn't", 'he', 'some', 'just',
"you'll", 'few', 'yourselves', 'from', 'where', 'about', 'both', '
being', 'very', 'been', 'but', "wasn't", 'no', 'such', "won't", 'w
asn', 'didn', 'll', 'our', 'as', "you're", 'ain', 'against', 'in',
'an', 'up', 'ma', 'was', "hadn't", 'through', 'any', 'weren', 'you
', "couldn't", 'his', 'when', "you've", 'they', 's', 'below', 'y',
'ours', 'couldn', 'isn', 'own', 'hers', "weren't", 'now', 'aren',
'theirs', 'once', "shan't", 'themselves', 'more', "isn't", 'what',
'there', 'don', 'this', 'off', 'd', 'so', "shouldn't", 'how', 'and
', 'after', "hasn't", 'yours', "mightn't", 'having', 'have', 'her
', 'your', 'while', 'herself', 'too', 'hadn', 'needn', 'i', "needn
't", 'be', 'am', 'between', 'to', 'into', 'on', 'does', 'had', "it
's", 'shouldn', 'under', 'further', 'mightn', 'a', 'then', 'shan',
'until', 'those', 'their', 'by', 'whom', 'each', 'if', 'above', 'o
urselves', 'o', 'should', "should've", 'these', 'that', 'during',
'myself', 're', 'do', 'out', 'yourself', 'only', 'same', 'not', 'n
or', 'haven', 'doing', 'here', 'all', 'the', 'him', 'of', 'my', 'd
own', 'will', 'them', 'other', 'or', 'is', 'for', "you'd", 'its',
'doesn', 'before', 'm', 've', 'mustn', "wouldn't", 'with', "mustn'
t", "aren't", 'why', "that'll", 'again', 'were', 'did', 'itself',
'are', 't', 'it'}
```

In [6]:
```
#Removing Punctuations and Stop Word
text= "How to remove stop words with NLTK library in Python?"
word_tokens= word_tokenize(text.lower())
filtered_sentence = []

for w in word_tokens:
    if w not in stop_words:
        filtered_sentence.append(w)

print("Tokenized Sentence:",word_tokens)
print("Filterd Sentence:",filtered_sentence)
```

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

In [7]:
```
#Perform Stemming
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
wait
wait
wait
```

In [8]:

```
#Perform 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.lemmat
ize(w)))
```

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

In [9]:

```
#Apply POS Tagging to text
from nltk.tokenize import word_tokenize
data="The pink sweater fit her perfectly"
words=word_tokenize(data)
for word in words:
    print(nltk.pos_tag([word]))
```

```
[('The', 'DT')]
[('pink', 'NN')]
[('sweater', 'NN')]
[('fit', 'NN')]
[('her', 'PRP$')]
[('perfectly', 'RB')]
```

In [ ]:

Practical No:-8

Data Visualization I

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

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.

In [5]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [19]:
```
data = pd.read_csv('https://raw.githubusercontent.com/dphi-official/Datasets/master/titanic_data.csv')
data.head()
```

Out[19]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

In [20]:
```
data.shape
```

Out[20]: (891, 12)

In [21]:
```
data.describe()
```

Out[21]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

In [22]:
```
data.describe(include = 'object')
```

Out[22]:

	Name	Sex	Ticket	Cabin	Embarked
count	891	891	891	204	889
unique	891	2	681	147	3
top	Braund, Mr. Owen Harris	male	347082	B96 B98	S
freq	1	577	7	4	644

In [23]:
```
data.isnull().sum()
```

Out[23]:
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

In [24]:
```python
data['Age'] = data['Age'].fillna(np.mean(data['Age']))
```

In [25]:
```python
data['Cabin'] = data['Cabin'].fillna(data['Cabin'].mode()[0])
```

In [31]:
```python
data['Embarked'] = data['Embarked'].fillna(data['Embarked'].mode()[0])
```

In [32]:
```python
data.isnull().sum()
```

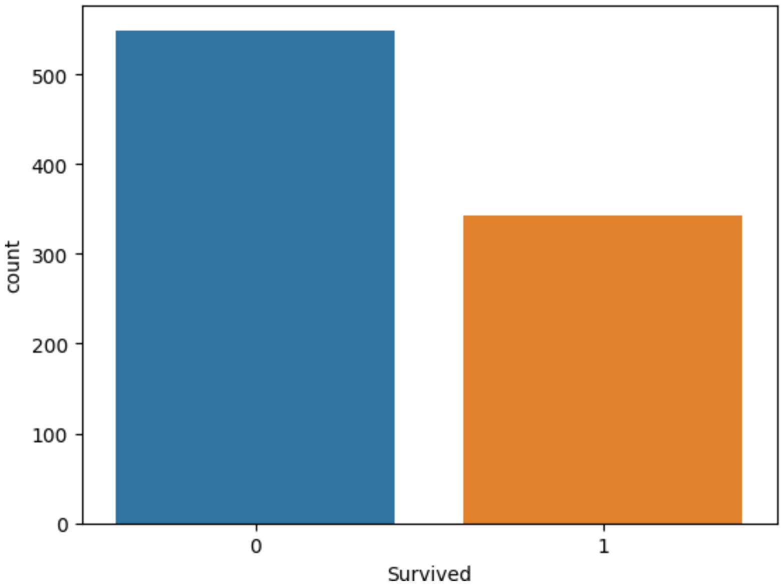
Out[32]:
```
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Cabin          0
Embarked       0
dtype: int64
```

Countplot

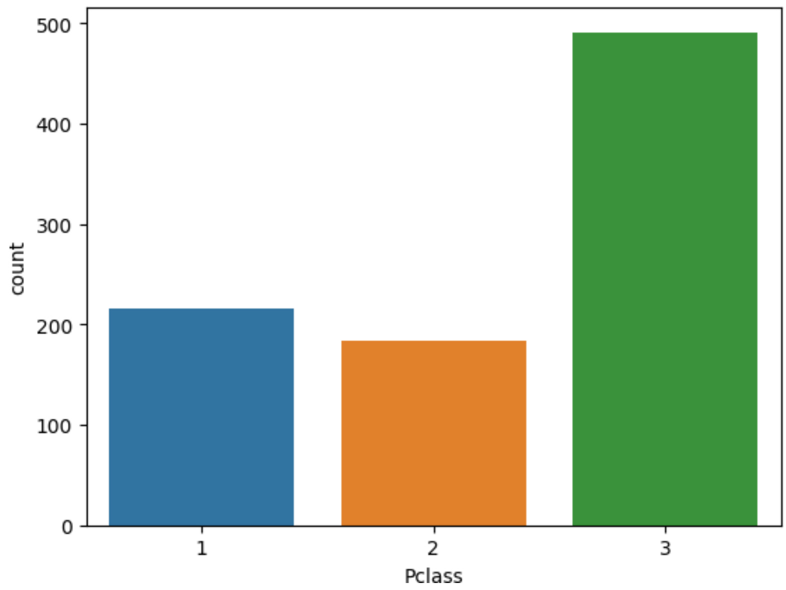
- The countplot is used to represent the occurrence(counts) of the observation present in the categorical variable.

In [35]:
```python
sns.countplot(x='Survived',data=data)
```

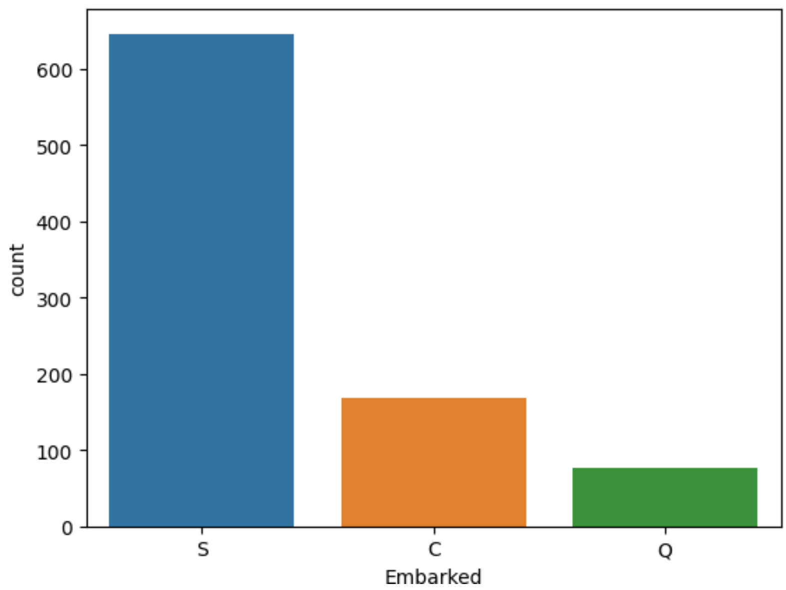
Out[35]:
```
<Axes: xlabel='Survived', ylabel='count'>
```



In [36]:
```python
sns.countplot(x='Pclass',data=data)
```
Out[36]: <Axes: xlabel='Pclass', ylabel='count'>

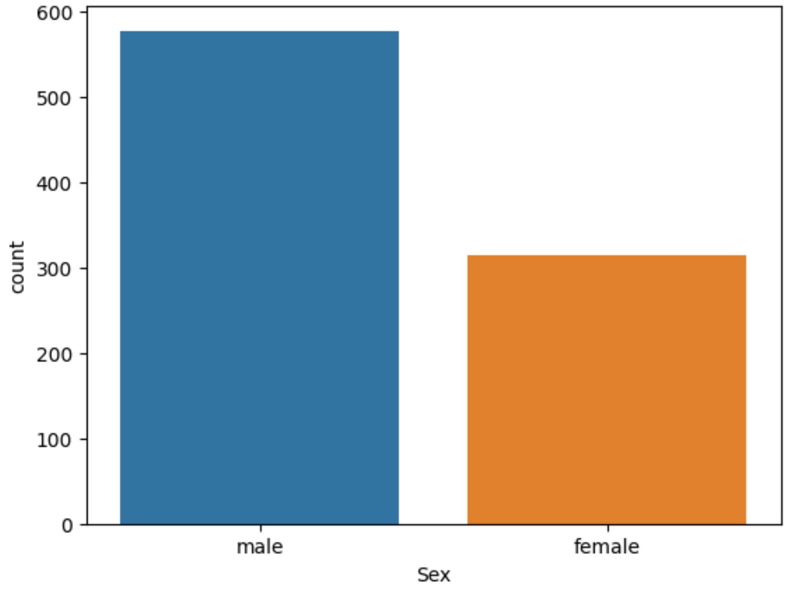


In [37]:
```python
sns.countplot(x='Embarked',data=data)
```
Out[37]: <Axes: xlabel='Embarked', ylabel='count'>



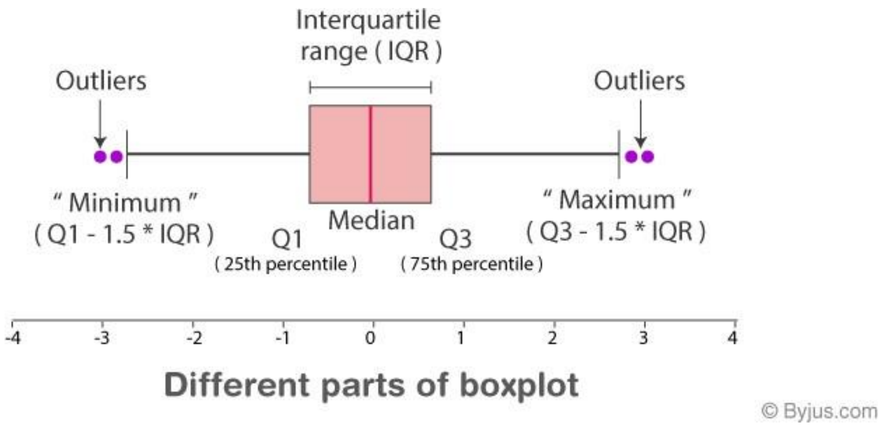
```
In [38]: sns.countplot(x='Sex',data=data)
```

```
Out[38]: <Axes: xlabel='Sex', ylabel='count'>
```

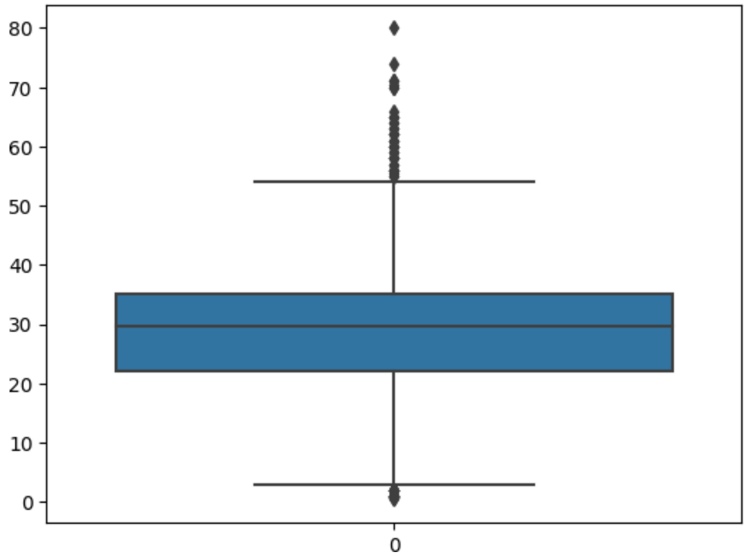


Boxplot

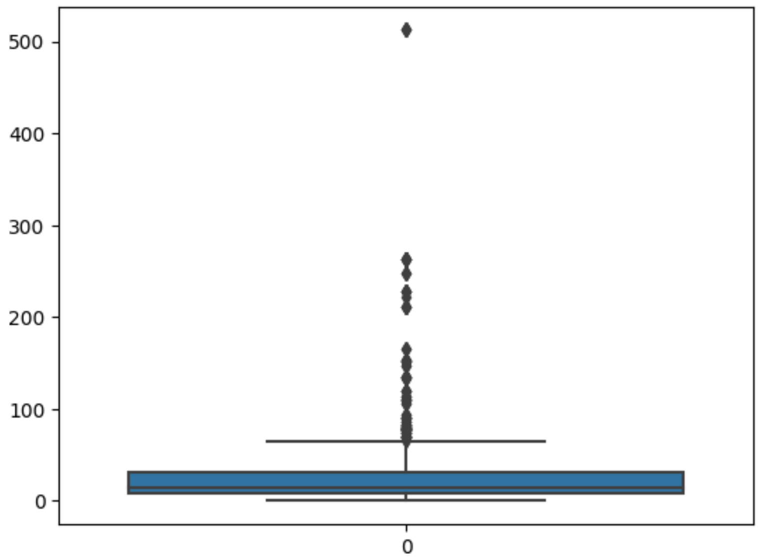
- A boxplot is a standardized way of displaying the distribution of data based on a five number summary (“minimum”, first quartile [Q1], median, third quartile [Q3] and “maximum”). It can tell you about your outliers and what their values are.



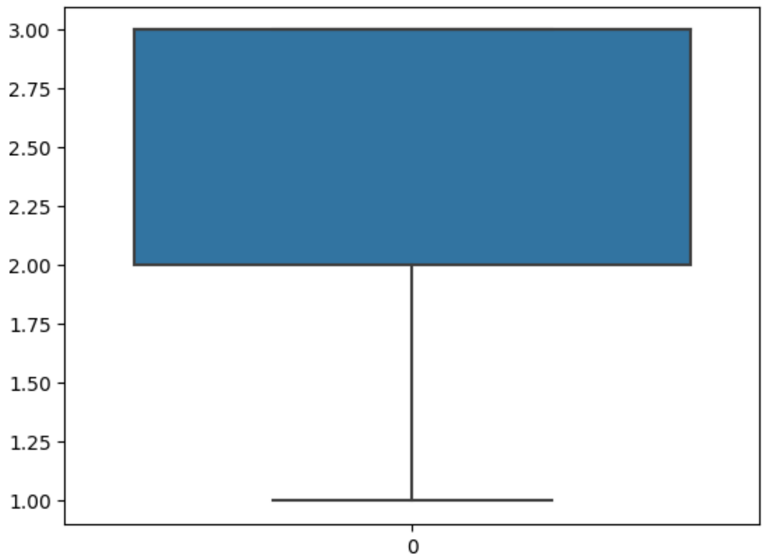
In [39]:
```
sns.boxplot(data['Age'])
```
Out[39]: <Axes: >



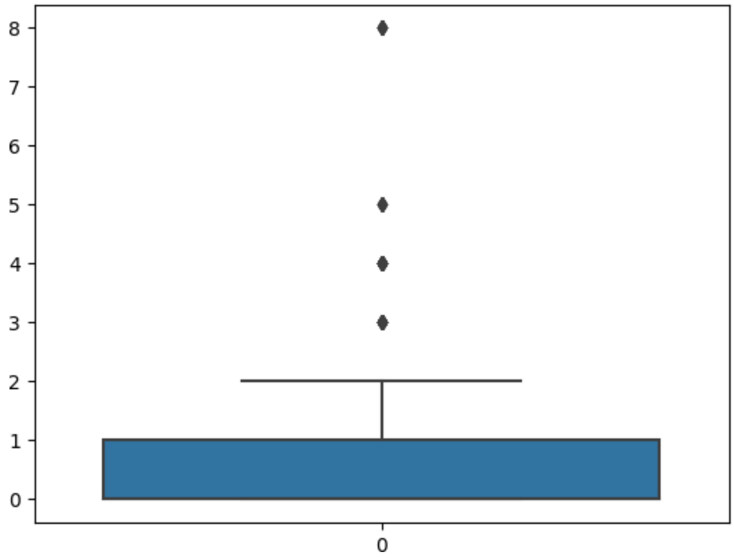
In [40]:
```
sns.boxplot(data['Fare'])
```
Out[40]: <Axes: >



In [41]:
```
sns.boxplot(data['Pclass'])
```
Out[41]: <Axes: >



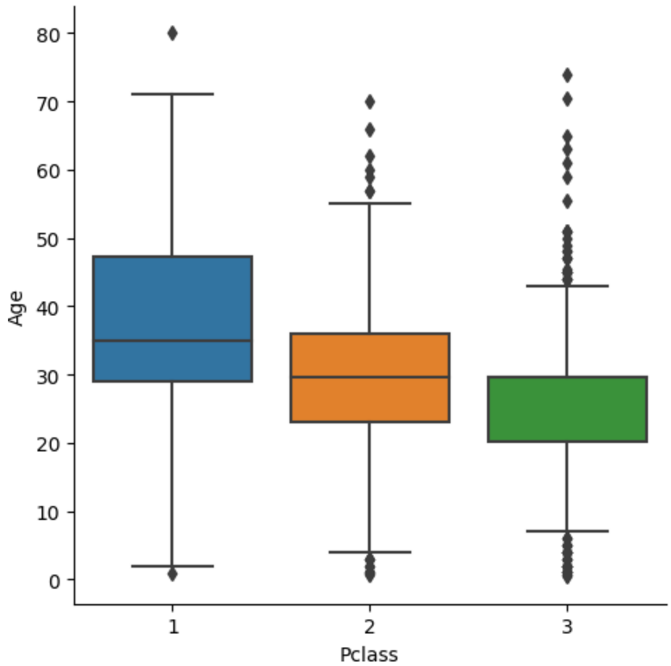
In [42]:
```
sns.boxplot(data['SibSp'])
```
Out[42]: <Axes: >



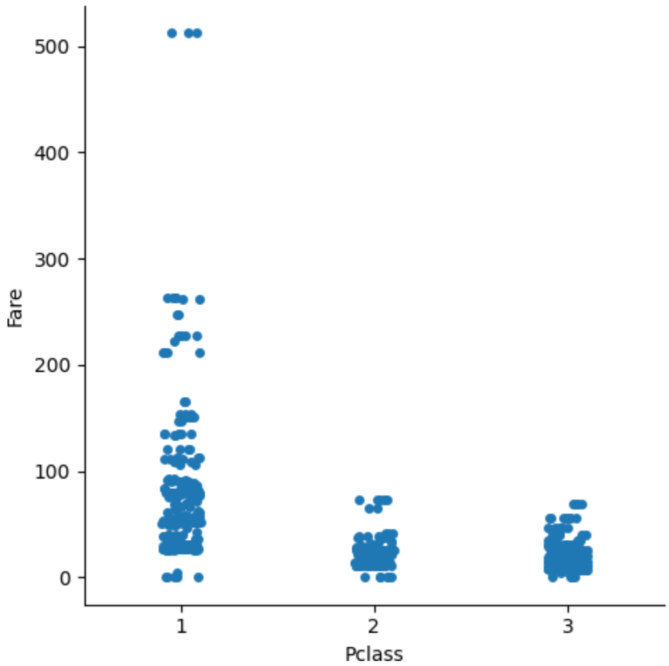
catplot

- The Seaborn catplot() function provides a figure-level interface for creating categorical plots. This means that the function allows you to map to a figure, rather than an axes object.

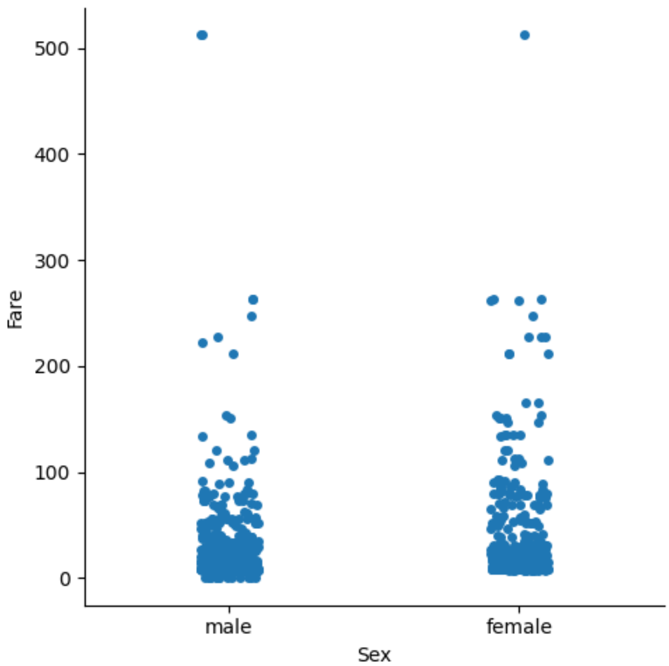
In [43]:
```
sns.catplot(x= 'Pclass', y = 'Age', data=data, kind = 'box')
```
Out[43]: <seaborn.axisgrid.FacetGrid at 0x1913988db50>



In [44]:
```
sns.catplot(x= 'Pclass', y = 'Fare', data=data, kind = 'strip')
```
Out[44]: <seaborn.axisgrid.FacetGrid at 0x19139676c10>

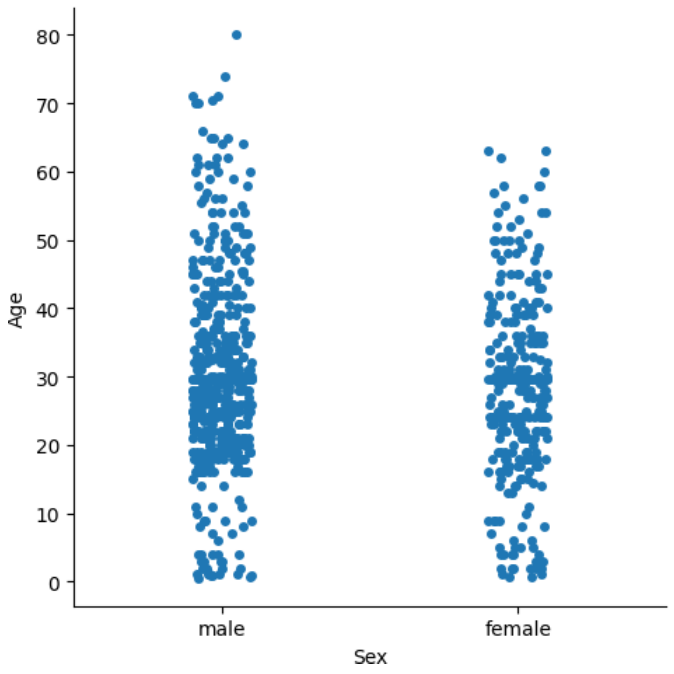


In [45]:
```
sns.catplot(x= 'Sex', y = 'Fare', data=data, kind = 'strip')
```
Out[45]: <seaborn.axisgrid.FacetGrid at 0x19139967210>



```
In [46]: sns.catplot(x= 'Sex', y = 'Age', data=data, kind = 'strip')
```

```
Out[46]: <seaborn.axisgrid.FacetGrid at 0x191395fbc90>
```

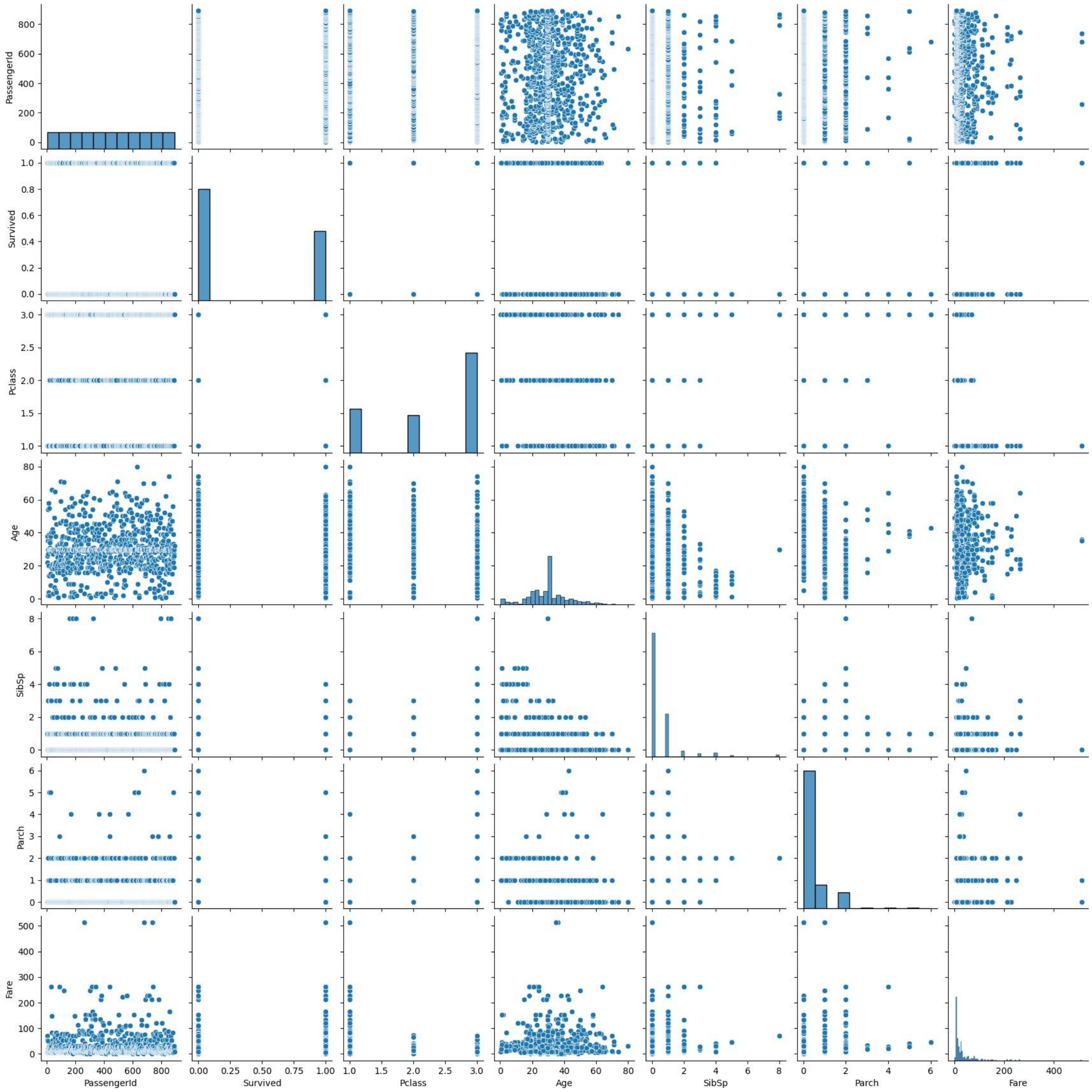


pairplot

- To plot multiple pairwise bivariate distributions in a dataset, you can use the .pairplot() function.
- The diagonal plots are the univariate plots, and this displays the relationship for the (n, 2) combination of variables in a DataFrame as a matrix of plots.

In [47]:
```
sns.pairplot(data)
```

Out[47]:
```
<seaborn.axisgrid.PairGrid at 0x1913aa5e010>
```

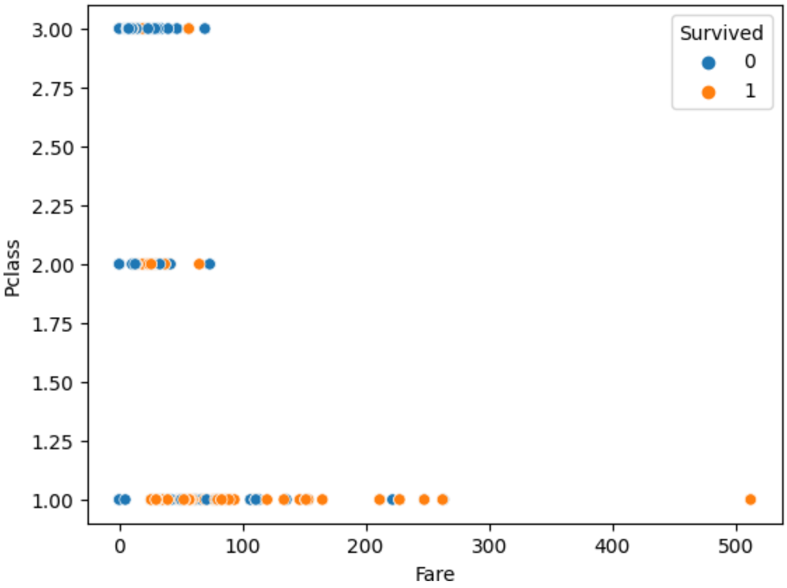


scatterplot

- Scatter plots are the graphs that present the relationship between two variables in a data-set. It represents data points on a two-dimensional plane or on a Cartesian system. The independent variable or attribute is plotted on the X-axis, while the dependent variable is plotted on the Y-axis. These plots are often called scatter graphs or scatter diagrams.

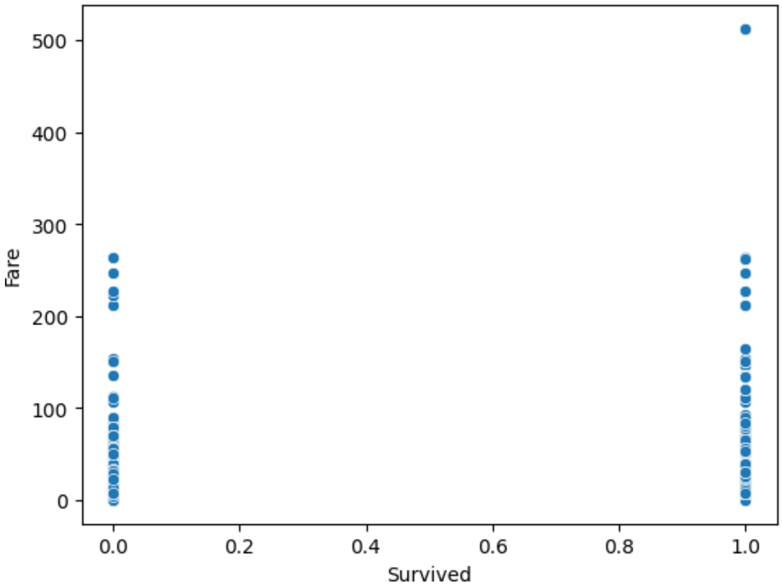
In [48]:
```
sns.scatterplot(x = 'Fare', y = 'Pclass', hue = 'Survived', data = data)
```

Out[48]:
```
<Axes: xlabel='Fare', ylabel='Pclass'>
```



In [49]:
```
sns.scatterplot(x = 'Survived', y = 'Fare', data = data)
```

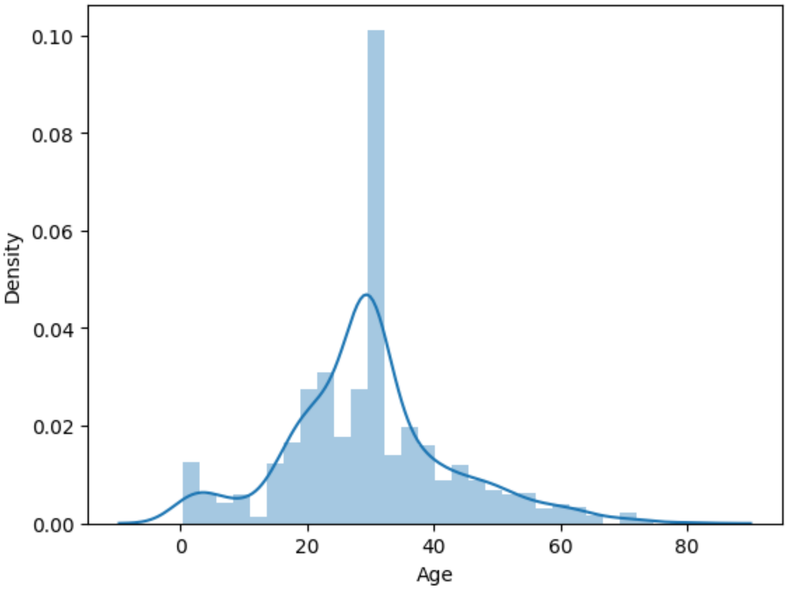
Out[49]:
```
<Axes: xlabel='Survived', ylabel='Fare'>
```



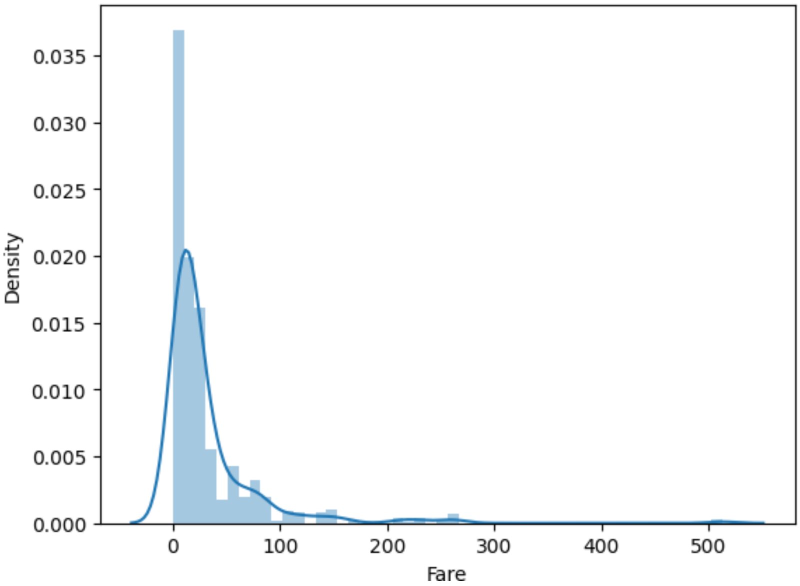
distplot

- These plots help us to visualise the distribution of data. We can use these plots to understand the mean, median, range, variance, deviation, etc of the data.

In [50]:
```
sns.distplot(data['Age'])
```
Out[50]: <Axes: xlabel='Age', ylabel='Density'>



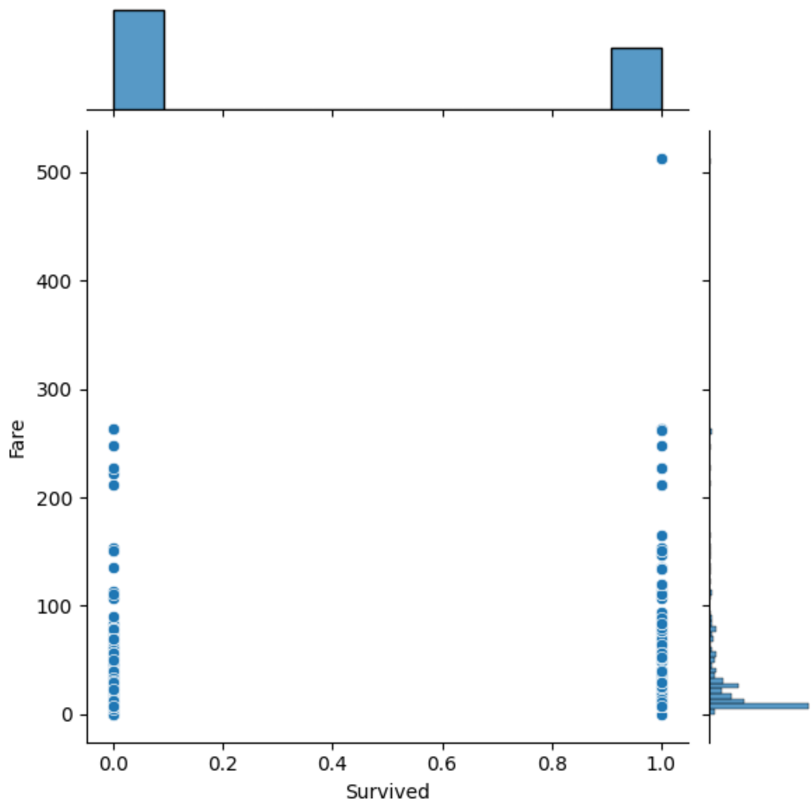
In [51]:
```
sns.distplot(data['Fare'])
```
Out[51]: <Axes: xlabel='Fare', ylabel='Density'>



jointplot

- The joint plot is a way of understanding the relationship between two variables and the distribution of individuals of each variable.

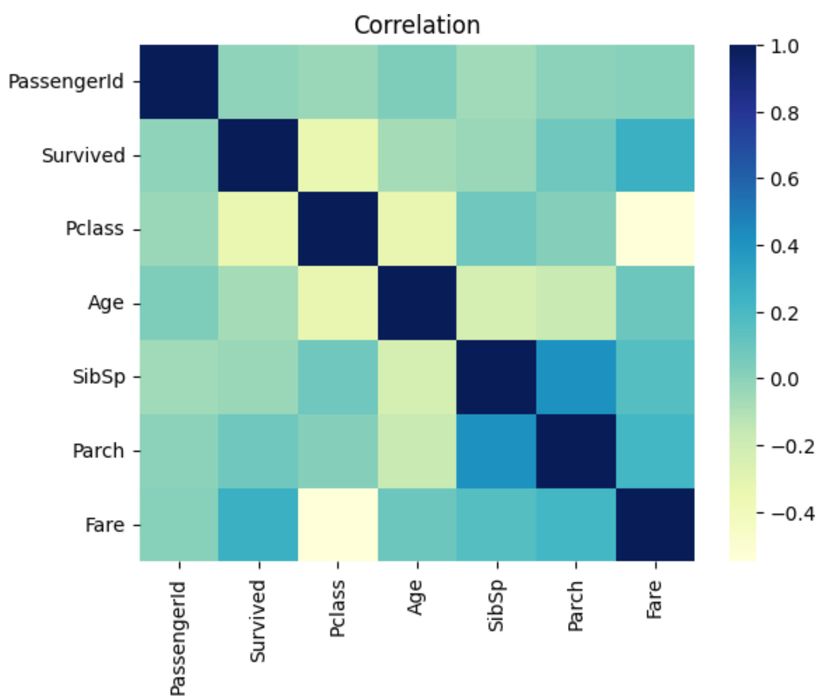
In [52]:
```
sns.jointplot(x = "Survived", y = "Fare", kind = "scatter", data = data)
```
Out[52]: <seaborn.axisgrid.JointGrid at 0x1913e94cad0>



corr()

- Pandas dataframe.corr() is used to find the pairwise correlation of all columns in the Pandas Dataframe in Python.

In [53]:
```
tc = data.corr()
sns.heatmap(tc, cmap="YlGnBu")
plt.title('Correlation')
```
Out[53]: Text(0.5, 1.0, 'Correlation')

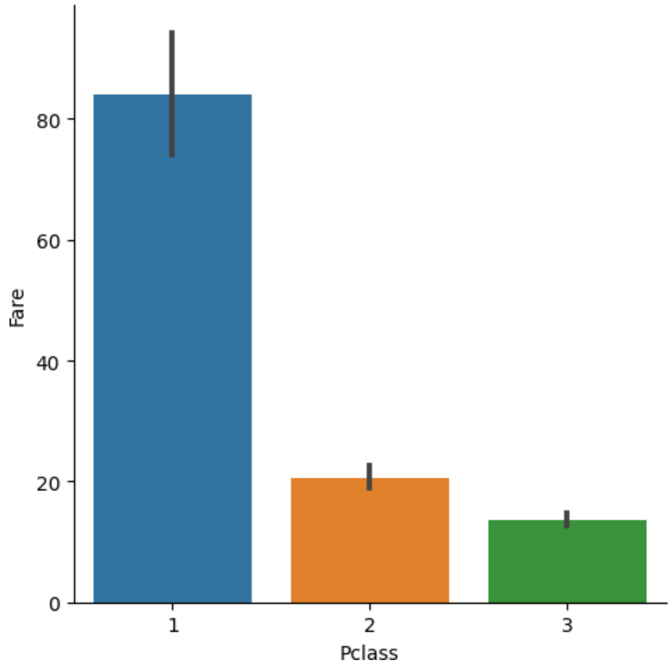


Price of Ticket for each passenger is distributed

2. Write a code to check how the price of the ticket (column name: 'fare') for each passenger is distributed by plotting a histogram

In [54]:
```
sns.catplot(x='Pclass', y='Fare', data=data, kind='bar')
```

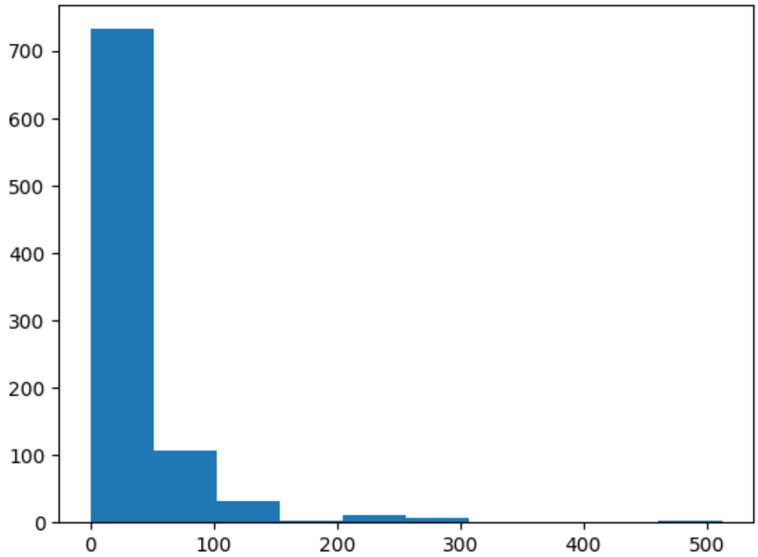
Out[54]:
```
<seaborn.axisgrid.FacetGrid at 0x1913ed76e10>
```



In [56]:
```
import matplotlib.pyplot as plt
```

In [57]:
```
plt.hist(data['Fare'])
```

Out[57]:
```
(array([732., 106.,  31.,   2.,  11.,   6.,   0.,   0.,   0.,   3.]),
 array([  0.      ,  51.23292, 102.46584, 153.69876, 204.93168, 256.1646 ,
        307.39752, 358.63044, 409.86336, 461.09628, 512.3292 ]),
 <BarContainer object of 10 artists>)
```



In [ ]:

Practical No:-9

Title of the Assignment: 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.

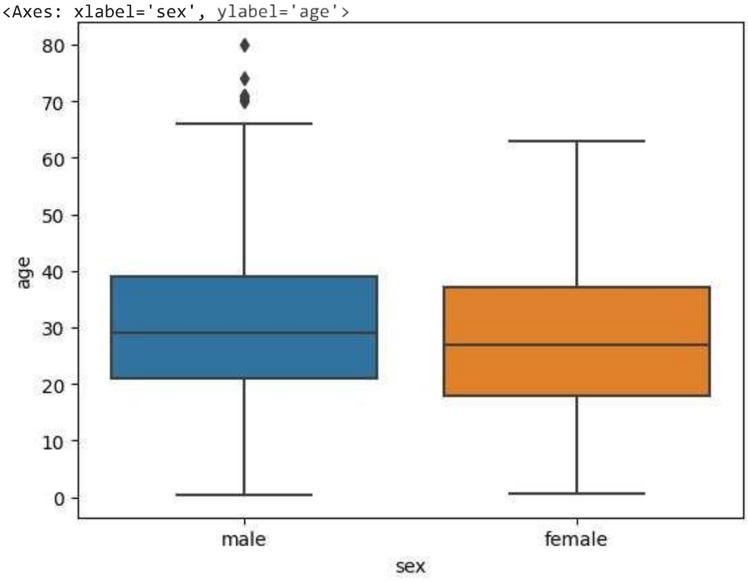
```
import seaborn as sns
dataset = sns.load_dataset('titanic')
dataset.head()
```

	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	

Double-click (or enter) to edit

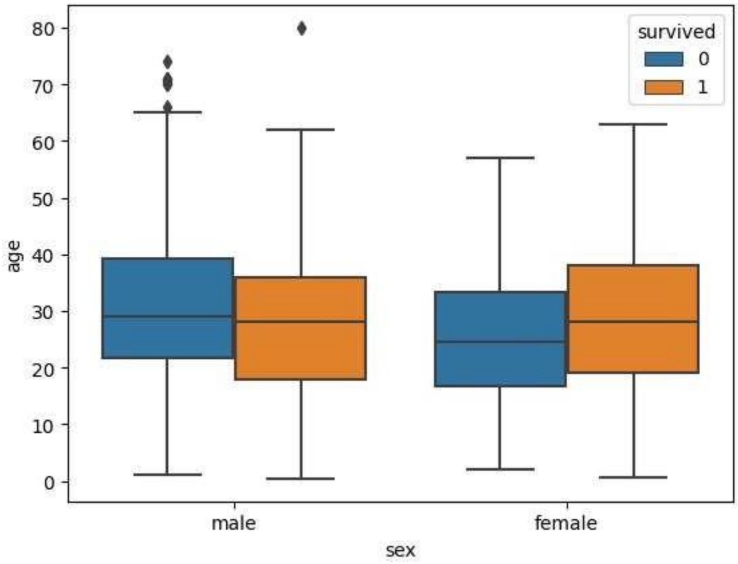
```
sns.boxplot(x='sex',y='age',data=dataset)
```

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




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

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



Colab paid products - Cancel contracts here

✓ 0s completed at 7:05 AM

Practical No:-10

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 (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.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
df = pd.read_csv('iris.csv')
df.head()
```

Out[1]:

	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	NaN	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

In [14]:
```
df.isnull().sum()
```

Out[14]:
```
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    1
PetalWidthCm     0
Species          0
dtype: int64
```

In [15]:
```
df['PetalLengthCm']=df['PetalLengthCm'].fillna(np.mean(df['PetalLengthCm']))
```

In [16]:
```
df.isnull().sum()
```

Out[16]:
```
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

1. List down the features and their types (e.g., numeric, nominal) available in the dataset.

In [17]:
```
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
```

Hence the dataset contains 4 numerical columns and 1 object column

In [18]:
```
np.unique(df["Species"])
```

Out[18]:
```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

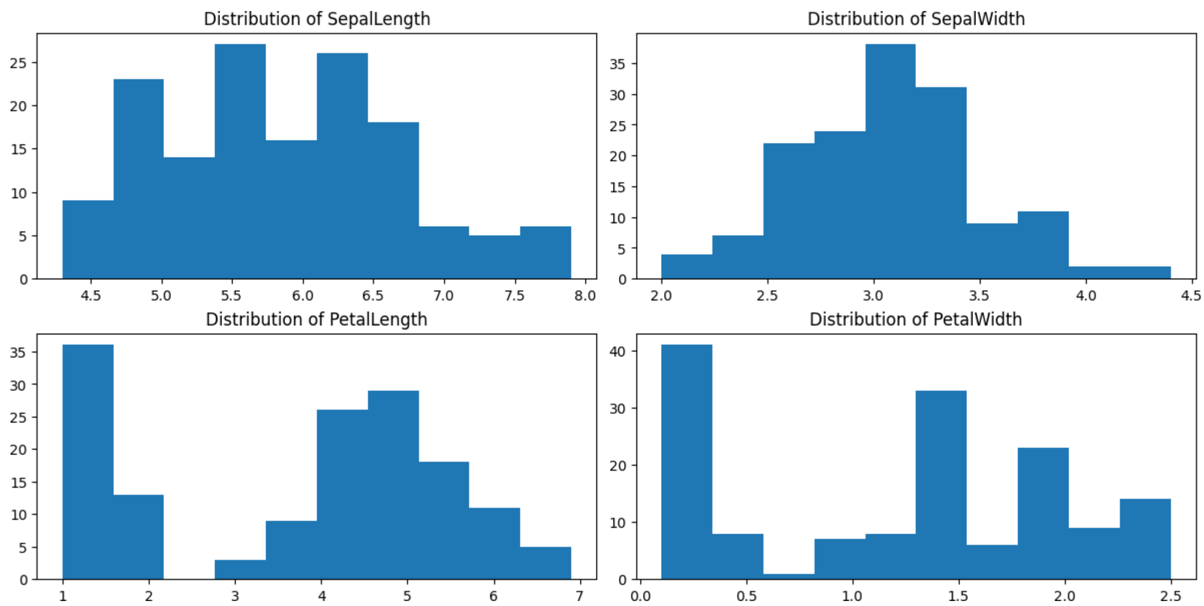
In [19]:
```
df.describe()
```

Out[19]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.775168	1.198667
std	43.445368	0.828066	0.433594	1.752808	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
max	150.000000	7.900000	4.400000	6.900000	2.500000

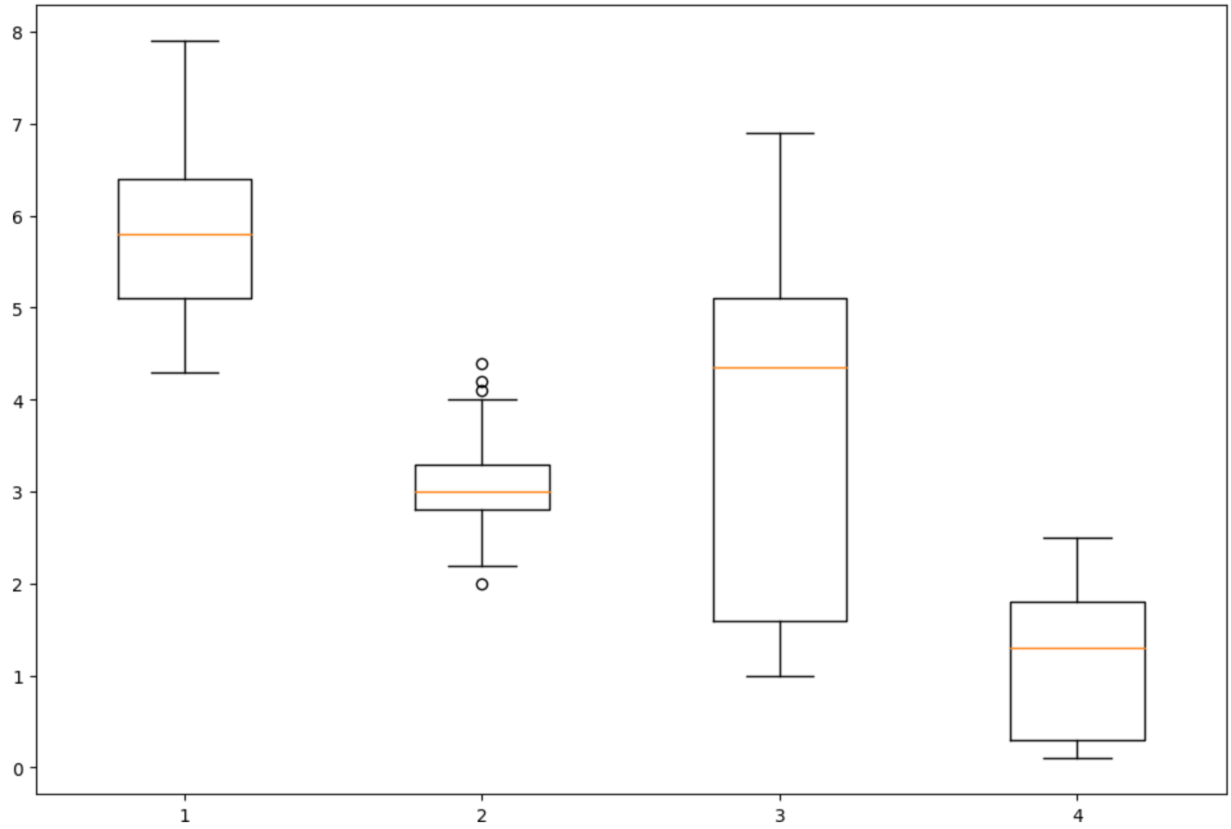
2. Create a histogram for each feature in the dataset to illustrate the feature distributions.

In [20]:
```
fig, axes = plt.subplots(2, 2, figsize=(12, 6), constrained_layout = True)
for i in range(4):
    x, y = i // 2, i % 2
    axes[x, y].hist(df[df.columns[i + 1]])
    axes[x, y].set_title(f"Distribution of {df.columns[i + 1][:-2]}")
```



3. Create a boxplot for each feature in the dataset.

In [21]:
```
data_to_plot = [df[x] for x in df.columns[1:-1]]
fig, axes = plt.subplots(1, figsize=(12,8))
bp = axes.boxplot(data_to_plot)
```



4. Compare distributions and identify outliers.

If we observe closely for the box 2, interquartile distance is roughly around 0.75 hence the values lying beyond this range of (third quartile + interquartile distance) i.e. roughly around 4.05 will be considered as outliers. Similarly outliers with other boxplots can be found.

Practical No:-11

//BY Devshree kulkarni

Java Code for word count:

```
import java.io.IOException;
import java.util.*;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.fs.*;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.*;
import org.apache.hadoop.mapreduce.lib.output.*;
import org.apache.hadoop.util.*;
public class WordCount extends Configured implements Tool
{
 public static void main(String args[]) throws Exception
 {
 int res = ToolRunner.run(new WordCount(), args);
 System.exit(res);
 }
 public int run(String[] args) throws Exception
 {
 Path inputPath = new Path(args[0]);
 Path outputPath = new Path(args[1]);
 Configuration conf = getConf();
```



```
 Job job = new Job(conf, this.getClass().toString());
 job.setJarByClass(WordCount.class);
 FileInputFormat.setInputPaths(job, inputPath);
 FileOutputFormat.setOutputPath(job, outputPath);
 job.setJobName("WordCount");
```

```
job.setMapperClass(Map.class);
job.setCombinerClass(Reduce.class);
job.setReducerClass(Reduce.class);
job.setMapOutputKeyClass(Text.class);
job.setMapOutputValueClass(IntWritable.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);

return job.waitForCompletion(true) ? 0 : 1;
}
public static class Map extends Mapper<LongWritable, Text, Text,
IntWritable>
{
private final static IntWritable one = new IntWritable(1);
private Text word = new Text();

public void map(LongWritable key, Text value, Mapper.Context
context) throws IOException, InterruptedException
{
String line = value.toString();
StringTokenizer tokenizer = new StringTokenizer(line);
while (tokenizer.hasMoreTokens())
{
word.set(tokenizer.nextToken());
context.write(word, one);
}
```

```
}
}
public static class Reduce extends Reducer<Text, IntWritable, Text,
IntWritable>
{

public void reduce(Text key, Iterable<IntWritable> values, Context
context) throws IOException, InterruptedException
{
int sum = 0;
for(IntWritable value : values)
{
sum += value.get();
}
context.write(key, new IntWritable(sum));
}
}
}
```

Input File

Pune

Mumbai

Nashik

Pune

Nashik

Kolapur

Practical No:-12

// Devshree Kulkarni

Java Code to process logfile

Mapper Class:

```
package SalesCountry;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.*;
public class SalesMapper extends MapReduceBase implements Mapper<LongWritable,
Text, Text, IntWritable> {
private final static IntWritable one = new IntWritable(1);
public void map(LongWritable key, Text value, OutputCollector<Text,
IntWritable> output, Reporter reporter) throws IOException {
String valueString = value.toString();
String[] SingleCountryData = valueString.split("-");
output.collect(new Text(SingleCountryData[0]), one);
}
}
```

Reducer Class:

```
package SalesCountry;


import java.io.IOException;
import java.util.*;
import org.apache.hadoop.io.IntWritable;
import  org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.*;
public class SalesCountryReducer extends MapReduceBase implements Reducer<Text,
IntWritable, Text, IntWritable> {
public void reduce(Text t_key, Iterator<IntWritable> values,
```

```
OutputCollector<Text,IntWritable> output, Reporter reporter) throws IOException
{
Text key = t_key;
int frequencyForCountry = 0;
while (values.hasNext()) {
// replace type of value with the actual type of our value
IntWritable value = (IntWritable) values.next();
frequencyForCountry += value.get();
}
output.collect(key, new IntWritable(frequencyForCountry));
}
}
```

Driver Class:

```
package SalesCountry;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
public class SalesCountryDriver {
public static void main(String[] args) {
JobClient my_client = new JobClient();
// Create a configuration object for the job
JobConf job_conf = new JobConf(SalesCountryDriver.class);


// Set a name of the Job
job_conf.setJobName("SalePerCountry");
// Specify data type of output key and value
job_conf.setOutputKeyClass(Text.class);
job_conf.setOutputValueClass(IntWritable.class);
// Specify names of Mapper and Reducer Class
job_conf.setMapperClass(SalesCountry.SalesMapper.class);
```

```
job_conf.setReducerClass(SalesCountry.SalesCountryReducer.class);
// Specify formats of the data type of Input and output
job_conf.setInputFormat(TextInputFormat.class);
job_conf.setOutputFormat(TextOutputFormat.class);
// Set input and output directories using command line arguments,
//arg[0] = name of input directory on HDFS, and arg[1] = name of
output directory to be created to store the output file.
FileInputFormat.setInputPaths(job_conf, new Path(args[0]));
FileOutputFormat.setOutputPath(job_conf, new Path(args[1]));
my_client.setConf(job_conf);
try {
// Run the job
JobClient.runJob(job_conf);
} catch (Exception e) {
e.printStackTrace();
}
}
}
```

Input File

Pune

Mumbai

Nashik

Pune

Nashik

Kolapu

Practical No:-13

```
//Devshree Kulkarni

/* Sample Code to print Statement */

object ExampleString {

def main(args: Array[String]) {

//declare and assign string variable "text"

val text : String = "You are reading SCALA programming language.";

//print the value of string variable "text"

println("Value of text is: " + text);

}

}

/**Scala program to find a number is positive, negative or positive.*/

object ExCheckNumber {

def main(args: Array[String]) {

/**declare a variable*/

var number= (-100);

if(number==0){

println("number is zero");

}

else if(number>0){
```



```
println("number is positive");

}

else{

println("number is negative");

}

}

}

/*Scala program to print your name*/

object ExPrintName {

def main(args: Array[String]) {
```

```
println("My name is Mike!")
}
}
/**Scala Program to find largest number among two numbers.*/
object ExFindLargest {
def main(args: Array[String]) {
var number1=20;
var number2=30;
var x = 10;
if( number1>number2){
println("Largest number is:" + number1);
}
else{
}
}
}
```