1. Create flexible data aggregations using pivot tables and Represent data visually using pivot charts

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

```
[15] df=pd.read_csv('titanic.csv')
```

```
df.head(5)
```

	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

```
[17] table = pd.pivot_table(df,index=['Sex','Pclass'],aggfunc={'Survived':np.sum})
     table
```

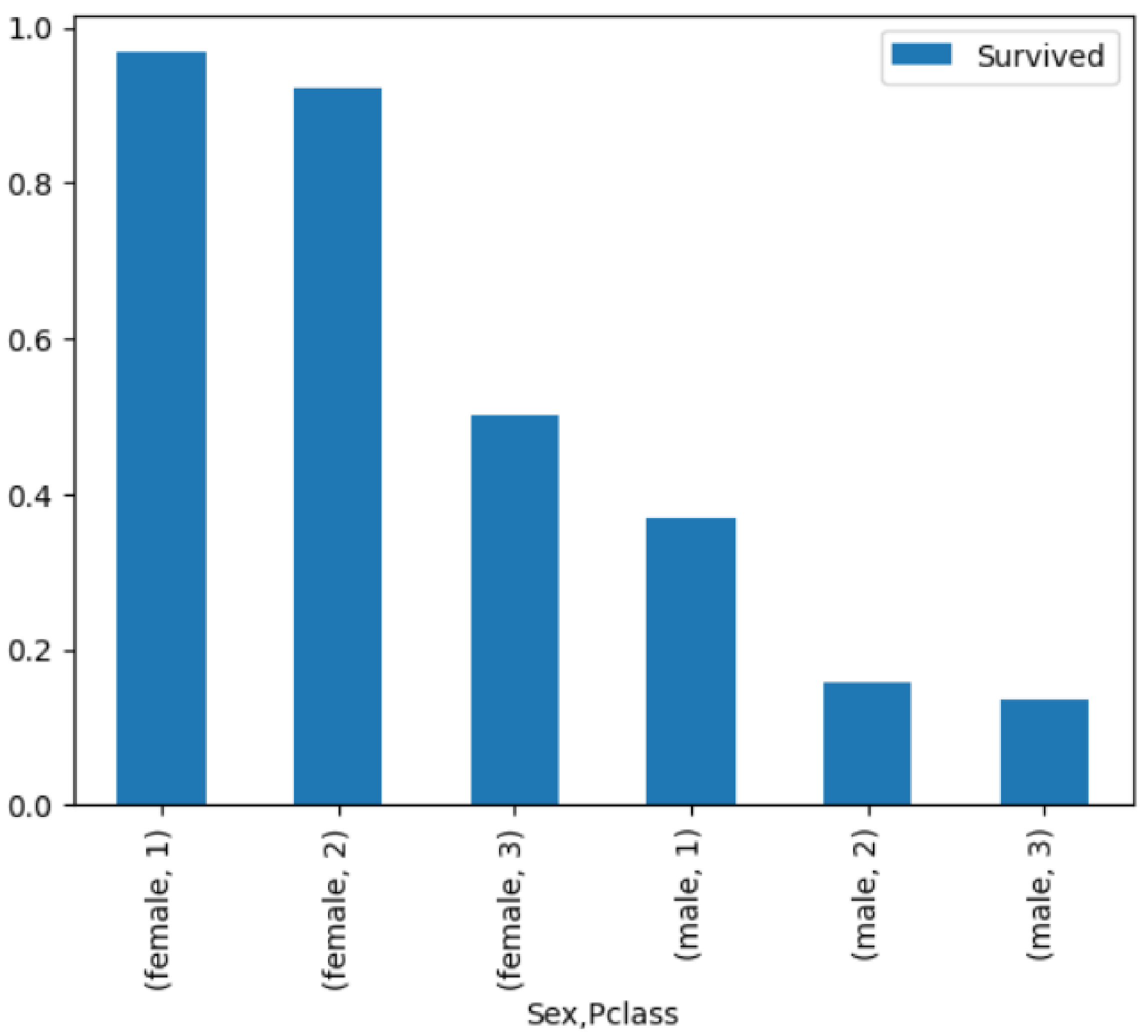
Sex	Pclass	Survived
female	1	91
	2	70
	3	72
male	1	45
	2	17
	3	47

[13]

```
table = pd.pivot_table(df,index=['Sex','Pclass'],values=['Survived'], aggfunc=np.mean)
table
```

Sex	Pclass	Survived
female	1	0.968085
	2	0.921053
	3	0.500000
male	1	0.368852
	2	0.157407
	3	0.135447

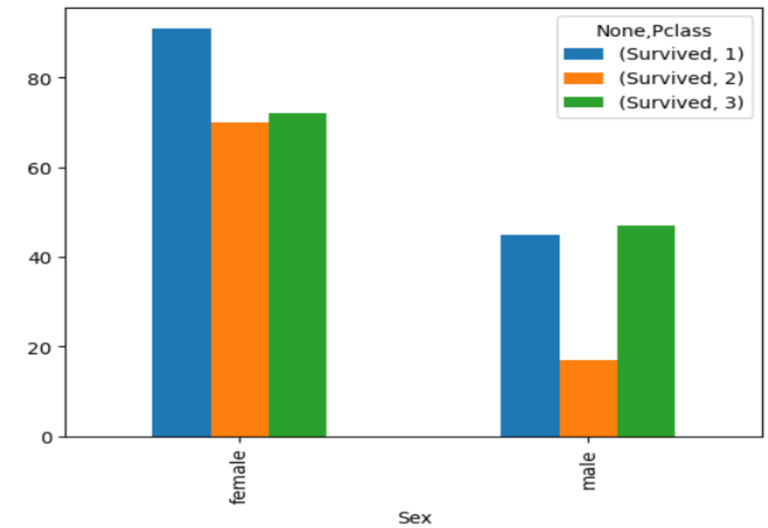
```
table.plot(kind='bar');
```



```python
table = pd.pivot_table(df,index=['Sex'],columns=['Pclass'],values=['Survived'],aggfunc=np.sum)
table
```

	Survived		
Pclass	1	2	3
Sex			
female	91	70	72
male	45	17	47

```python
[21] table.plot(kind='bar');
```



[22]
```python
#display null values
table = pd.pivot_table(df,index=['Sex','Survived','Pclass'],columns=['Embarked'],values=['Age'],aggfunc=np.mean)
table
```

			Age		
		Embarked	C	Q	S
Sex	Survived	Pclass			
female	0	1	50.000000	NaN	13.500000
		2	NaN	NaN	36.000000
		3	20.700000	28.100000	23.688889
	1	1	35.675676	33.000000	33.619048
		2	19.142857	30.000000	29.091667
		3	11.045455	17.600000	22.548387
male	0	1	43.050000	44.000000	45.362500
		2	29.500000	57.000000	33.414474
		3	27.555556	28.076923	27.168478
	1	1	36.437500	NaN	36.121667
		2	1.000000	NaN	17.095000
		3	18.488571	29.000000	22.933333

```python
#handling null values
table = pd.pivot_table(df,index=['Sex','Survived','Pclass'],columns=['Embarked'],values=['Age'],aggfunc=np.mean,fill_value=np.mean(df['Age']))
table
```

			Age		
		Embarked	C	Q	S
Sex	Survived	Pclass			
female	0	1	50.000000	29.699118	13.500000
		2	29.699118	29.699118	36.000000
		3	20.700000	28.100000	23.688889
	1	1	35.675676	33.000000	33.619048
		2	19.142857	30.000000	29.091667
		3	11.045455	17.600000	22.548387
male	0	1	43.050000	44.000000	45.362500
		2	29.500000	57.000000	33.414474
		3	27.555556	28.076923	27.168478
	1	1	36.437500	29.699118	36.121667
		2	1.000000	29.699118	17.095000
		3	18.488571	29.000000	22.933333

```
import pandas as pd
import numpy as np
df = pd.read_csv("tem.csv")
df
```

	city	temperature
0	Mumbai	34
1	Chennai	38
2	Hyderabad	43
3	Banagalore	30
4	Pune	-4
5	Kochi	33
6	Goa	50

```
df.shape
```

```
(7, 2)
```

```
df.dtypes
```

```
city           object
temperature     int64
dtype: object
```

```
df.head()
```

	city	temperature
0	Mumbai	34
1	Chennai	38
2	Hyderabad	43
3	Banagalore	30
4	Pune	-4

```
df.tail(3)
```

	city	temperature
4	Pune	-4
5	Kochi	33
6	Goa	50

```
df.isnull()
```

	city	temperature
0	False	False
1	False	False
2	False	False
3	False	False
4	False	False
5	False	False
6	False	False

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

```
city           0
temperature    0
dtype: int64
```

```
df.count()
```

```
city           7
temperature    7
dtype: int64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7 entries, 0 to 6
Data columns (total 2 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   city         7 non-null      object
 1   temperature  7 non-null      int64 
dtypes: int64(1), object(1)
memory usage: 240.0+ bytes
```

```
gk = df.groupby('city')
gk=gk.get_group('Mumbai')
gk
```

	city	temperature
0	Mumbai	34

Downlod Following CSV files and do all operations iris.csv

1. titanic.csv
2. car.csv
3. Iris.csv Solve the following 1)Download csv from google 2)upload in juypter notebook 3)load/read csv file 4)display count of rows and Columns 5) Display data type of each column 6)Displat first 3 record 7)Display last 3 record 8) Display count of null values 9)Display info of file 10 Diplay the data of one category

---

```
import pandas as pd
df = pd.read_csv("titanic.csv")
df
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fa
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.25
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.28
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.92
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.10

```
df.shape
```

```
(891, 12)
```

```
df.dtypes
```

```
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object
```

```
df.head(3)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500
				Cumings						

```
df.tail(3)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	C
888	889	0	3	Johnston, Miss. Catherine Helen	female	NaN	1	2	W./C. 6607	23.45	

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

```
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
```

```
df.count()
```

```
PassengerId    891
Survived       891
Pclass         891
Name           891
Sex            891
Age            714
SibSp          891
Parch          891
Ticket         891
Fare           891
Cabin          204
Embarked       889
dtype: int64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
```

```
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

```
gk = df.groupby('Pclass')
gk=gk.get_group(3)
gk
```

	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
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
882	883	0	3	Dahlberg, Miss. Gerda Ulrika	female	22.0	0	0	7552	10.5167	NaN	S
884	885	0	3	Sutehall, Mr. Henry Jr	male	25.0	0	0	SOTON/OQ 392076	7.0500	NaN	S
885	886	0	3	Rice, Mrs. William (Margaret Norton)	female	39.0	0	5	382652	29.1250	NaN	Q
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S

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

	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
...	...	...	...	...	...	...
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

150 rows × 6 columns

```
df.shape
```

```
(150, 6)
```

```
df.dtypes
```

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

```
Species           object
dtype: object
```

```
df.head(3)
```

	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

```
df.tail(3)
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
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

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

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

```
df.count()
```

```
Id               150
SepalLengthCm    150
SepalWidthCm     150
PetalLengthCm    150
PetalWidthCm     150
Species          150
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
```

```
gk = df.groupby('Species')
gk=gk.get_group('Iris-setosa')
gk
```

	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
10	11	5.4	3.7	1.5	0.2	Iris-setosa
11	12	4.8	3.4	1.6	0.2	Iris-setosa
12	13	4.8	3.0	1.4	0.1	Iris-setosa
13	14	4.3	3.0	1.1	0.1	Iris-setosa
14	15	5.8	4.0	1.2	0.2	Iris-setosa
15	16	5.7	4.4	1.5	0.4	Iris-setosa
16	17	5.4	3.9	1.3	0.4	Iris-setosa
17	18	5.1	3.5	1.4	0.3	Iris-setosa
18	19	5.7	3.8	1.7	0.3	Iris-setosa
19	20	5.1	3.8	1.5	0.3	Iris-setosa
20	21	5.4	3.4	1.7	0.2	Iris-setosa
21	22	5.1	3.7	1.5	0.4	Iris-setosa
22	23	4.6	3.6	1.0	0.2	Iris-setosa
23	24	5.1	3.3	1.7	0.5	Iris-setosa
24	25	4.8	3.4	1.9	0.2	Iris-setosa
25	26	5.0	3.0	1.6	0.2	Iris-setosa
26	27	5.0	3.4	1.6	0.4	Iris-setosa
27	28	5.2	3.5	1.5	0.2	Iris-setosa
28	29	5.2	3.4	1.4	0.2	Iris-setosa
29	30	4.7	3.2	1.6	0.2	Iris-setosa
30	31	4.8	3.1	1.6	0.2	Iris-setosa
31	32	5.4	3.4	1.5	0.4	Iris-setosa
32	33	5.2	4.1	1.5	0.1	Iris-setosa
33	34	5.5	4.2	1.4	0.2	Iris-setosa
34	35	4.9	3.1	1.5	0.1	Iris-setosa
35	36	5.0	3.2	1.2	0.2	Iris-setosa
36	37	5.5	3.5	1.3	0.2	Iris-setosa
37	38	4.9	3.1	1.5	0.1	Iris-setosa
38	39	4.4	3.0	1.3	0.2	Iris-setosa
39	40	5.1	3.4	1.5	0.2	Iris-setosa
40	41	5.0	3.5	1.3	0.3	Iris-setosa
41	42	4.5	2.3	1.3	0.3	Iris-setosa
42	43	4.4	3.2	1.3	0.2	Iris-setosa
43	44	5.0	3.5	1.6	0.6	Iris-setosa
44	45	5.1	3.8	1.9	0.4	Iris-setosa
45	46	4.8	3.0	1.4	0.3	Iris-setosa

```
import pandas as pd
import numpy as np
df = pd.read_csv("Book1.csv")
df
```

	city	temperature	humidity
0	new york	65	56
1	new york	65	66
2	new york	66	60
3	mumbai	75	80
4	mumbai	68	80

```
import statistics
```

```
statistics.stdev(df['humidity'])
```

```
11.171392035015153
```

Finding Frequency

```
count = df['city'].value_counts()
print(count)
```

```
new york    3
mumbai      2
Name: city, dtype: int64
```

```
count = df.groupby(['city']).count()
print(count)
```

```
          temperature  humidity
city
mumbai              2         2
new york            3         3
```

Double-click (or enter) to edit

```
df.mean()
```

```
<ipython-input-32-c61f0c8f89b5>:1: FutureWarning: The default value of numeric_only in DataFrame.mean is deprecated. In a future version
  df.mean()
temperature    67.8
humidity       68.4
dtype: float64
```

```
df.median()
```

```
<ipython-input-33-6d467abf240d>:1: FutureWarning: The default value of numeric_only in DataFrame.median is deprecated. In a future versi
  df.median()
temperature    66.0
humidity       66.0
dtype: float64
```

```
df.mode(numeric_only=True)
```

	temperature	humidity
0	65	80

```
df.describe()
```

	temperature	humidity
count	5.000000	5.000000
mean	67.800000	68.400000
std	4.207137	11.171392
min	65.000000	56.000000
25%	65.000000	60.000000
50%	66.000000	66.000000
75%	68.000000	80.000000
max	75.000000	80.000000

```
temperature_variance = df['temperature'].var()

print(temperature_variance)
```

```
17.7
```

Double-click (or enter) to edit

```
humidity_variance = df['humidity'].var()

print(humidity_variance)
```

```
124.79999999999998
```

```
temperature_stddev = df['temperature'].std()

print(temperature_stddev)
```

```
4.207136793592526
```

```
humidity_stddev = df['humidity'].std()

print(humidity_stddev)
```

```
11.171392035015153
```

b. Apply the basis of Data cleanup operation on the given *dataset*

Data Cleaning Data cleaning means fixing bad data in your data set.

Bad data could be:

Wrong data– Duplicates– Empty cells– Data in wrong format

Double-click (or enter) to edit

```
import pandas as pd
import numpy as np
df = pd.read_csv("calori.csv")
df
```

	Time	Date	Pulse	Calories
0	60	2020/12/01'	110.0	409
1	600	2020/12/02'	120.0	479
2	60	2020/12/03'	NaN	340
3	45	2020/12/05'	102.0	287
4	45	2020/12/05'	102.0	287
5	60	20201206'	2.0	300
6	60	NaN	104.0	374
7	450	2020/12/08'	102.0	253

Double-click (or enter) to edit

```
df.dtypes
```

```
Time           int64
Date          object
Pulse        float64
Calories       int64
dtype: object
```

Double-click (or enter) to edit

The data set contains some empty cells (row 2, and row 6).

The data set contains wrong format ("Date" in row 5).

The data set contains wrong data ("Duration" in row 1).

The data set contains duplicates (row 3 and 4).

```
df.shape
```

```
(8, 4)
```

Discovering Duplicates

Duplicate rows are rows that have been registered more than one time.

```
df.duplicated()
```

```
0    False
1    False
2    False
3    False
4     True
5    False
```

```
6     False
7     False
dtype: bool
```

To remove duplicates, use the drop_duplicates() m

```
df.drop_duplicates(inplace = True)
```

```
df.shape
```

```
(7, 4)
```

```
df
```

	Time	Date	Pulse	Calories
0	60	'2020/12/01'	110.0	409
1	600	'2020/12/02'	120.0	479
2	60	'2020/12/03'	NaN	340
3	45	'2020/12/05'	102.0	287
5	60	'20201206'	2.0	300
6	60	NaN	104.0	374
7	450	'2020/12/08'	102.0	253

Wrong Data

Replacing Values

```
for x in df.index:
  if df.loc[x, "Time"] >= 120:
    df.loc[x, "Time"] = 60
```

```
for x in df.index:
  if df.loc[x, "Pulse"] <100:
    df.loc[x, "Pulse"] = 110
```

```
df
```

	Time	Date	Pulse	Calories
0	60	'2020/12/01'	110.0	409
1	60	'2020/12/02'	120.0	479
2	60	'2020/12/03'	NaN	340
3	45	'2020/12/05'	102.0	287
5	60	'20201206'	110.0	300
6	60	NaN	104.0	374
7	60	'2020/12/08'	102.0	253

```
df['Date'] = pd.to_datetime(df['Date'])
```

```
df
```

	Time	Date	Pulse	Calories
0	60	2020-12-01	110.0	409
1	60	2020-12-02	120.0	479

```
df.dtypes
```

```
Time                 int64
Date        datetime64[ns]
Pulse              float64
Calories             int64
dtype: object
```

```
df
```

	Time	Date	Pulse	Calories
0	60	2020-12-01	110.0	409
1	60	2020-12-02	120.0	479
2	60	2020-12-03	NaN	340
3	45	2020-12-05	102.0	287
5	60	2020-12-06	110.0	300
6	60	NaT	104.0	374
7	60	2020-12-08	102.0	253

```
df['Pulse'].median()
```

```
107.0
```

```
d2=df.copy(deep=True)
d2
```

	Time	Date	Pulse	Calories
0	60	2020-12-01	110.0	409
1	60	2020-12-02	120.0	479
2	60	2020-12-03	NaN	340
3	45	2020-12-05	102.0	287
5	60	2020-12-06	110.0	300
6	60	NaT	104.0	374
7	60	2020-12-08	102.0	253

```
d2['Pulse'].median()
```

```
107.0
```

Pulse column has wrong value We replace it by mean /median/mode

Default title text

```
# @title Default title text
d2['Pulse'] = (d2['Pulse'].fillna(d2['Pulse'].median()))
d2
```

	Time	Date	Pulse	Calories
0	60	2020-12-01	110.0	409
1	60	2020-12-02	120.0	479

```
d3=df.copy(deep=True)
d3
```

	Time	Date	Pulse	Calories
0	60	2020-12-01	110.0	409
1	60	2020-12-02	120.0	479
2	60	2020-12-03	NaN	340
3	45	2020-12-05	102.0	287
5	60	2020-12-06	110.0	300
6	60	NaT	104.0	374
7	60	2020-12-08	102.0	253

```
d3['Pulse'].mode()
```

```
0    102.0
1    110.0
Name: Pulse, dtype: float64
```

```
d3.fillna(d3.mode().iloc[0])
#OR
d3.fillna(d3.mode().iloc[1])
```

	Time	Date	Pulse	Calories
0	60	2020-12-01	110.0	409
1	60	2020-12-02	120.0	479
2	60	2020-12-03	110.0	340
3	45	2020-12-05	102.0	287
5	60	2020-12-06	110.0	300
6	60	2020-12-02	104.0	374
7	60	2020-12-08	102.0	253

4. Use any standard data set and perform the following

a. Find the data distributions using box and scatter plot. b. Find the outliers using plot. c. Plot the histogram, bar chart and pie chart on sample data.

Double-click (or enter) to edit

```
#a. Find the data distributions using box
```

```
import pandas as pd
import numpy as np
df = pd.read_csv("tem.csv")
df
```

	city	temperature
0	Mumbai	34
1	Chennai	38
2	Hyderabad	43
3	Banagalore	30
4	Pune	1
5	Kochi	33
6	Goa	50

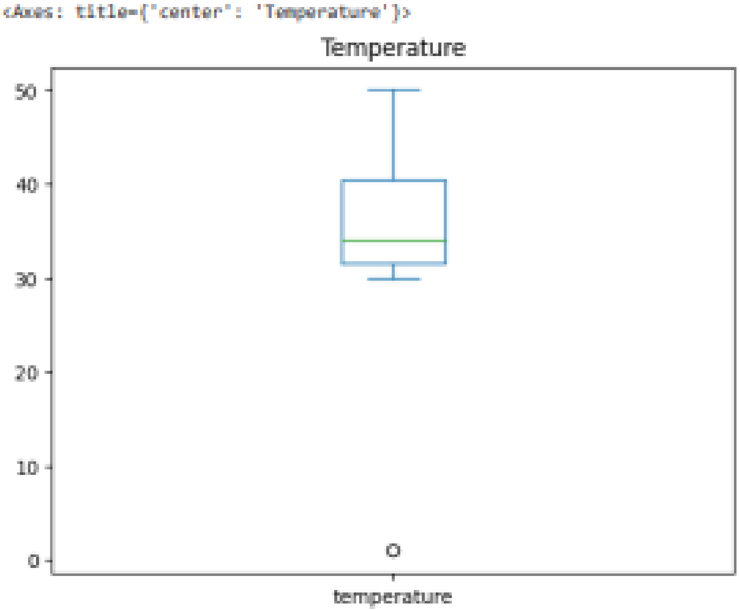
```
df.median()
```

```
<ipython-input-16-6d467abf240d>:1: FutureWarning: The default value of numeric_only in DataFrame.median is deprecated. In a future versi
  df.median()
temperature    34.0
dtype: float64
```

```
from scipy import stats
import numpy as np
```

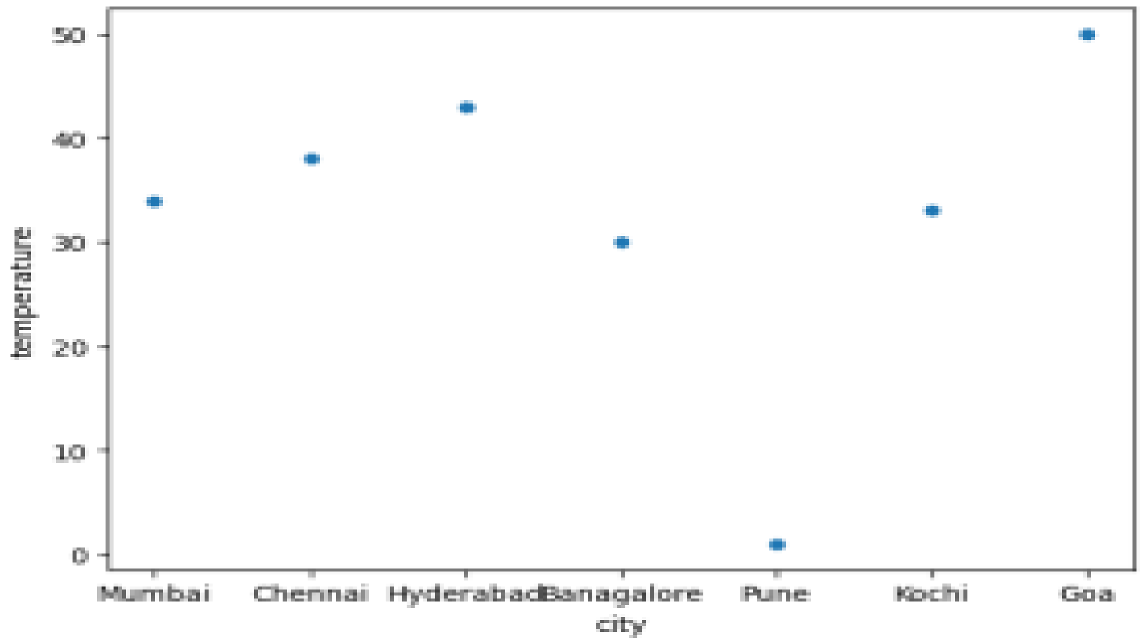
```
df['temperature'].plot(kind='box', title='Temperature')
```

```
<Axes: title={'center': 'Temperature'}>
```



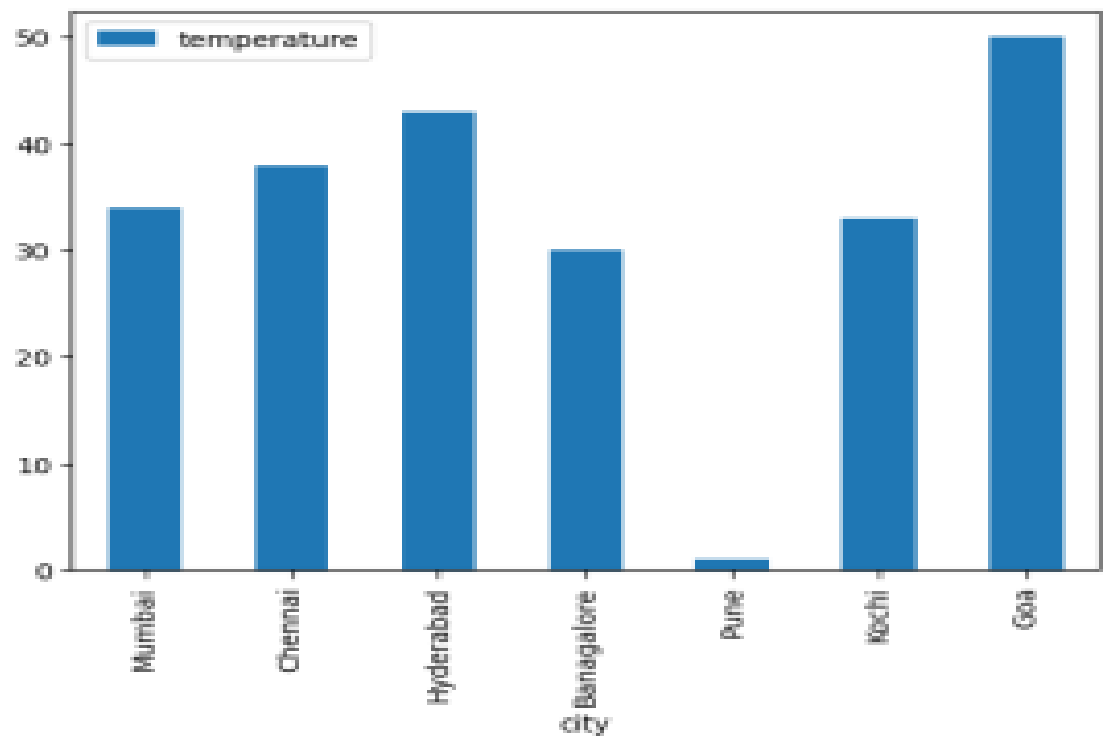
b.Find the data distributions using scatter plot

```
df.plot.scatter(x = 'city', y = 'temperature');
```




```
import matplotlib.pyplot as plt
df.plot(x="city", y="temperature", kind="bar")
```

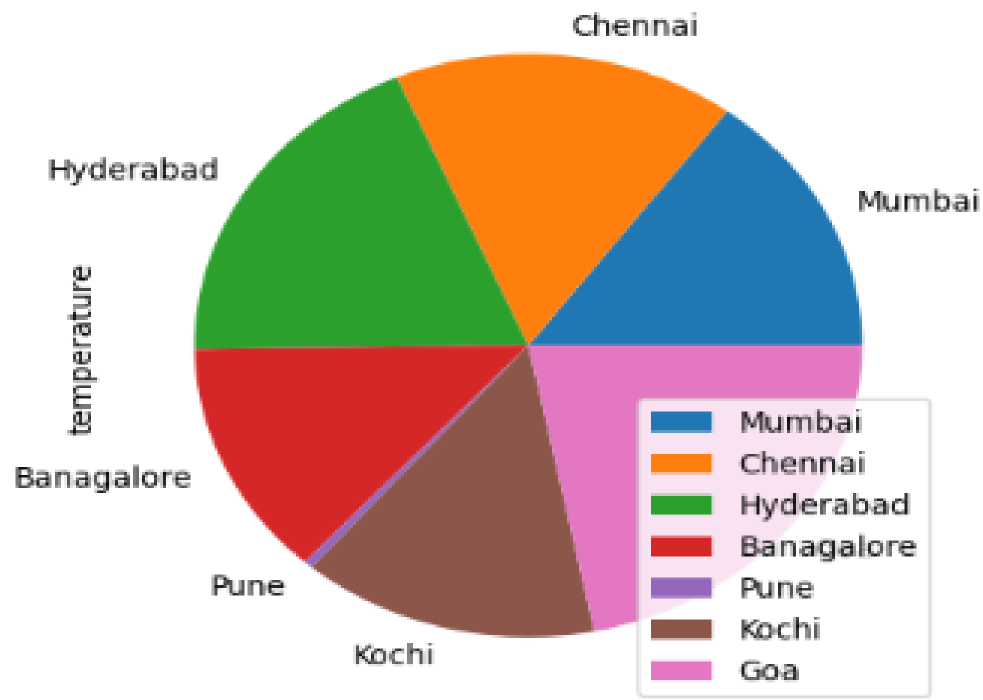
```
<Axes: xlabel='city'>
```



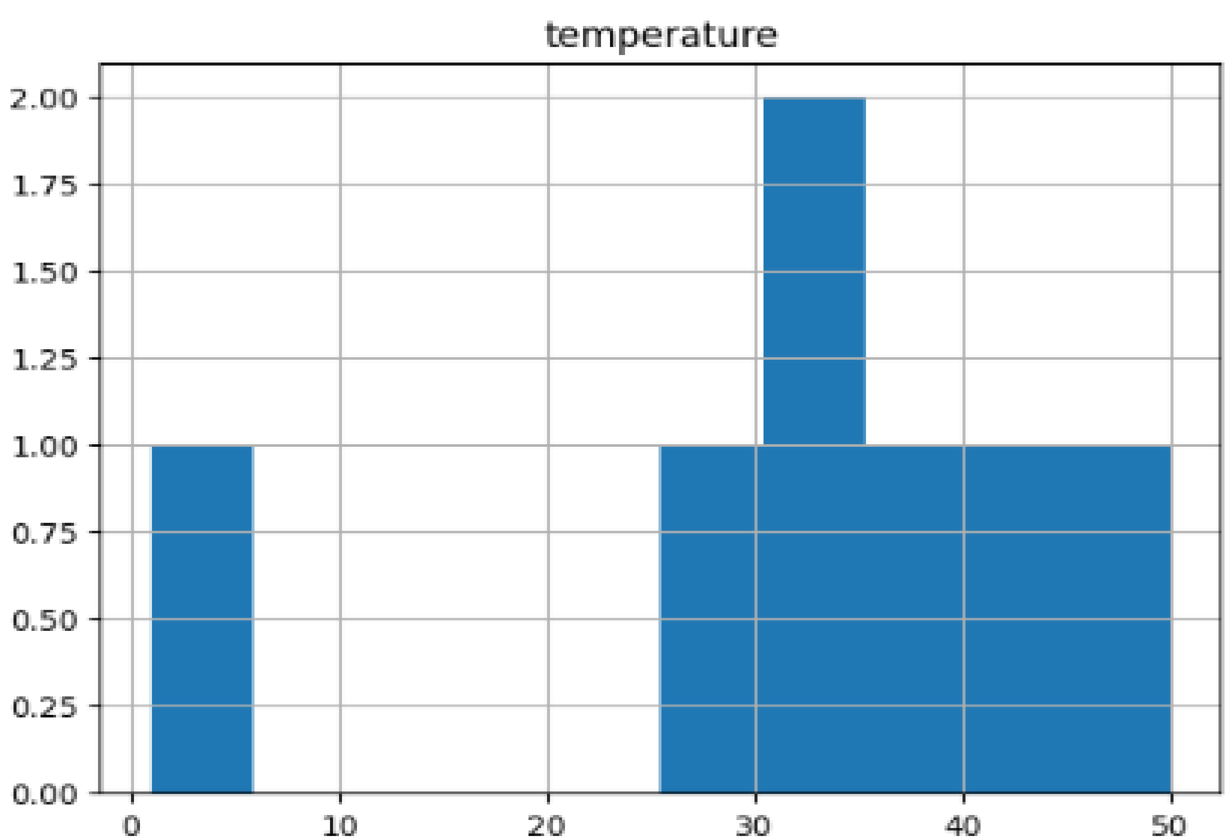

```
df.plot(kind='pie',x='city',labels=df['city'], y='temperature')
```

```python
df.plot(kind='pie',x='city',labels=df['city'], y='temperature')
```

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

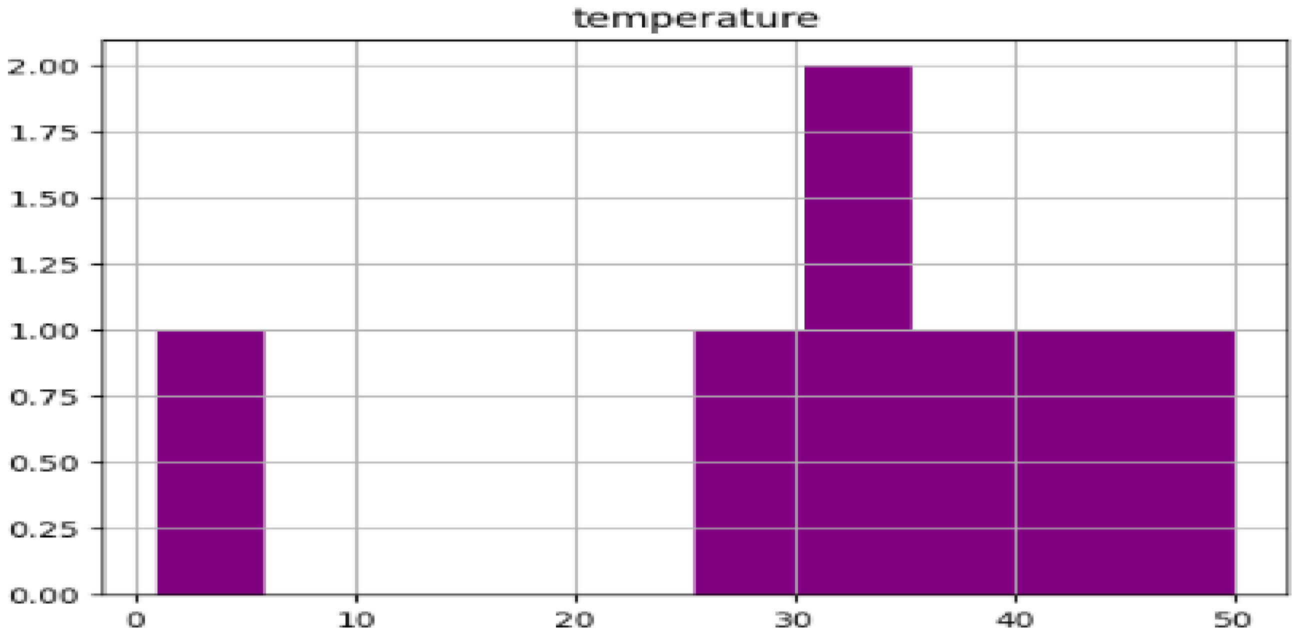



```python
df.hist()
plt.show()
```



```
df.hist(column='temperature', color='purple')
```

```
array([[<Axes: title={'center': 'temperature'}>]], dtype=object)
```



5. Import any CSV file to Pandas DataFrame and perform the following: a. Visualize the first and last 10 records b. Do required statistical operations on the given columns. c. Find the count and uniqueness of the given categorical values.

```
[ ] import pandas as pd
    import numpy as np
    import seaborn as sns
```

```
[ ] df = pd.read_csv("titanic_dataset.csv")
    df
```

	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
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

a. Visualize the first and last 10 records

```
df.head(10)
```

	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
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	NaN	S
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742	11.1333	NaN	S
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.0708	NaN	C

```
[ ] df.tail(10)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
881	882	0	3	Markun, Mr. Johann	male	33.0	0	0	349257	7.8958	NaN	S
882	883	0	3	Dahlberg, Miss. Gerda Ulrika	female	22.0	0	0	7552	10.5167	NaN	S
883	884	0	2	Banfield, Mr. Frederick James	male	28.0	0	0	C.A./SOTON 34068	10.5000	NaN	S
884	885	0	3	Sutehall, Mr. Henry Jr	male	25.0	0	0	SOTON/OQ 392076	7.0500	NaN	S

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

	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

```
[ ]
```

Statistical operations on the given columns

```
[ ] df.isnull().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
```

```
[ ] df['Fare'].mean()
```

```
32.204207968574636
```

```
[ ] df['Fare'].median()
```

```
14.4542
```

```
[ ] df['Fare'].mode()
```

```
0    8.05
Name: Fare, dtype: float64
```

```
[ ] df['Fare'].std()
```

```
49.6934285971809
```

```
[ ] df['Fare'].var()
```

```
2469.436845743116
```

c. Find the count and uniqueness of the given categorical values.

```
[ ] df['Sex'].value_counts()
```

```
male      577
female    314
Name: Sex, dtype: int64
```

```
[ ] df['Sex'].value_counts(ascending=True)
```

```
female    314
male      577
Name: Sex, dtype: int64
```

```
[ ]
```

```
[ ] df['Fare'].value_counts().max
```

```
<bound method NDFrame._add_numeric_operations.<locals>.max of 8.0500     43
13.0000    42
7.8958     38
7.7500     34
26.0000    31
           ..
35.0000     1
28.5000     1
6.2375      1
14.0000     1
10.5167     1
Name: Fare, Length: 248, dtype: int64>
```

We can see most people paid under 73.19 for their ticket.

```
[ ] df['Cabin'].value_counts()
```

```
B96 B98        4
G6             4
C23 C25 C27    4
C22 C26        3
F33            3
              ..
E34            1
C7             1
C54            1
E36            1
C148           1
Name: Cabin, Length: 147, dtype: int64
```

```
[ ] df['Embarked'].value_counts()
```

```
S    644
C    168
Q     77
Name: Embarked, dtype: int64
```

6. Import any CSV file to Pandas DataFrame and perform the following:

a. Handle missing data by detecting and dropping/ filling missing values.

b. Transform data using map() method.

c. Detect and filter outliers.

d. Visualize data using Line chart, Bar chart, Histograms, Density chart and Scatter chart.

```
import pandas as pd
df = pd.read_csv("map.csv")
df
```

	name	age	Score	Income
0	James	30.0	2.0	129999999.0
1	Jane	40.0	67.0	34000.0
2	Melissa	32.0	80.0	56000.0
3	Ed	67.0	NaN	34587.0
4	Neil	43.0	89.0	40000.0
5	Jaya	34.0	34.0	58000.0
6	Rita	23.0	NaN	34500.0
7	tiya	NaN	NaN	NaN
8	sevk	NaN	NaN	NaN

a. Handle missing data by detecting and dropping/ filling missing values.**

```
[ ] df.isnull().sum()
```

```
name      0
age       2
Score     4
Income    2
dtype: int64
```

```python
df.isnull()
```

	name	age	Score	Income
0	False	False	False	False
1	False	False	False	False
2	False	False	False	False
3	False	False	True	False
4	False	False	False	False
5	False	False	False	False
6	False	False	True	False
7	False	True	True	True
8	False	True	True	True

```python
df.dropna(inplace=True)
```

```python
df
```

	name	age	Score	Income
0	James	30.0	2.0	129999999.0
1	Jane	40.0	67.0	34000.0
2	Melissa	32.0	80.0	56000.0
4	Neil	43.0	89.0	40000.0
5	Jaya	34.0	34.0	58000.0

Transform data using map() method.

```python
genders = {'James': 'Male', 'Jane': 'Female', 'Melissa': 'Female', 'Ed': 'Male', 'Neil': 'Male', 'Jaya': 'Female', 'Rita': 'Female'}
```

```python
df['gender'] = df['name'].map(genders)
print(df)
```

```
      name   age  Score       Income  gender
0    James  30.0    2.0  129999999.0    Male
1     Jane  40.0   67.0      34000.0  Female
2  Melissa  32.0   80.0      56000.0  Female
4     Neil  43.0   89.0      40000.0    Male
5     Jaya  34.0   34.0      58000.0  Female
```

```python
last_names = pd.Series(['Doe', 'Miller', 'Edwards', 'Nelson', 'Raul'], index=df['name'])
df['Last Name'] = df['name'].map(last_names)
```

```python
df
```

	name	age	Score	Income	Last Name
0	James	30.0	2.0	129999999.0	Doe
1	Jane	40.0	67.0	34000.0	Miller
2	Melissa	32.0	80.0	56000.0	Edwards
4	Neil	43.0	89.0	40000.0	Nelson
5	Jaya	34.0	34.0	58000.0	Raul

6 c. Detect and filter outliers.

```
[10] import pandas as pd
     import numpy as np
```

```
[11] data={'score':[1,1,1,1,1,2,2,2,2,2,2,2,3,3,15]}
```

```
data=pd.DataFrame(data)
data
```

	score
0	1
1	1
2	1
3	1
4	1
5	2
6	2
7	2
8	2
9	2
10	2
11	2
12	3
13	3
14	15

Double-click (or enter) to edit

Outlier detection using ZScore

Double-click (or enter) to edit

```
[15] from scipy import stats
     data['z']=stats.zscore(data.score,ddof=1)
     data
```

	score	z
0	1	-0.479227
1	1	-0.479227
2	1	-0.479227
3	1	-0.479227
4	1	-0.479227
5	2	-0.191691
6	2	-0.191691
7	2	-0.191691
8	2	-0.191691
9	2	-0.191691
10	2	-0.191691
11	2	-0.191691
12	3	0.095845
13	3	0.095845
14	15	3.546282

Filtering outliers.

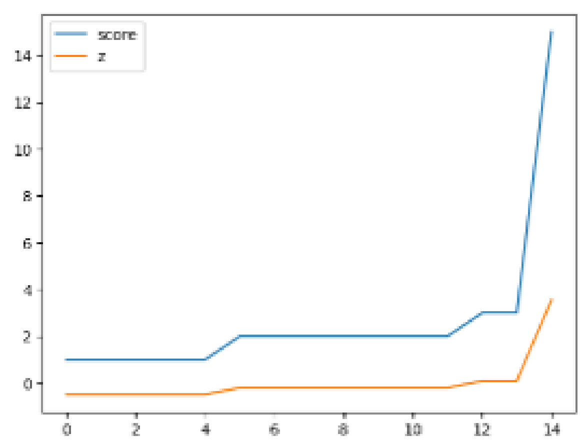
```
[16] nooutliersdata = data[(data.z>-3) & (data.z<3)]
     nooutliersdata
```

	score	z
0	1	-0.479227
1	1	-0.479227
2	1	-0.479227
3	1	-0.479227
4	1	-0.479227
5	2	-0.191691
6	2	-0.191691
7	2	-0.191691
8	2	-0.191691
9	2	-0.191691
10	2	-0.191691
11	2	-0.191691
12	3	0.095845
13	3	0.095845

d. Visualize data using Line chart, Bar chart, Histograms, Density chart and Scatter chart

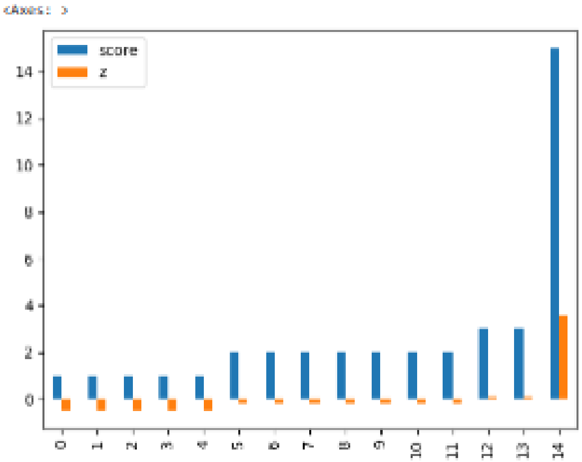
```
[17] import matplotlib.pyplot as plt
     data.plot( kind="line")
```

<Axes: >



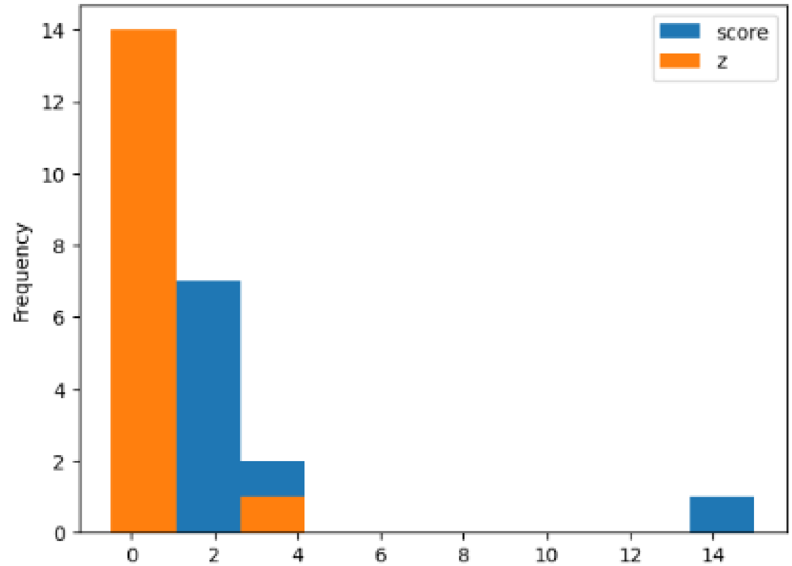

```
data.plot( kind="bar")
```

<Axes: >



```
data.plot( kind="hist")
```

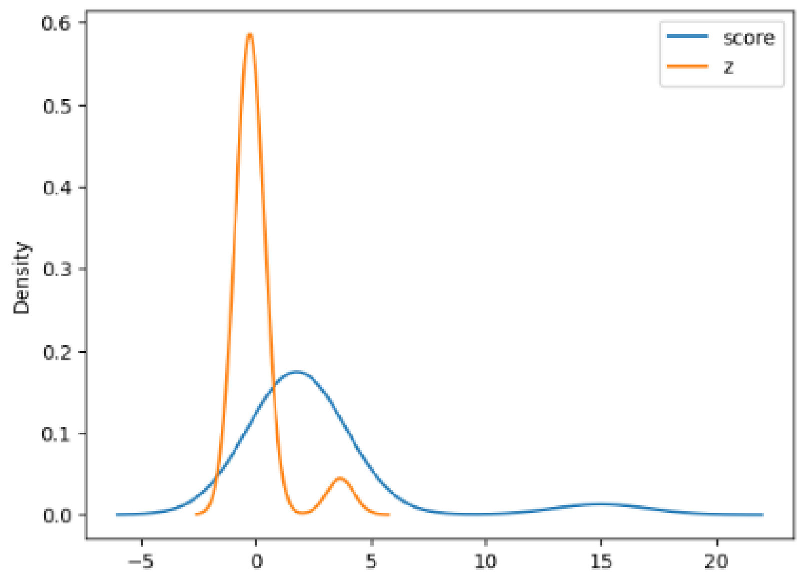
<Axes: ylabel='Frequency'>



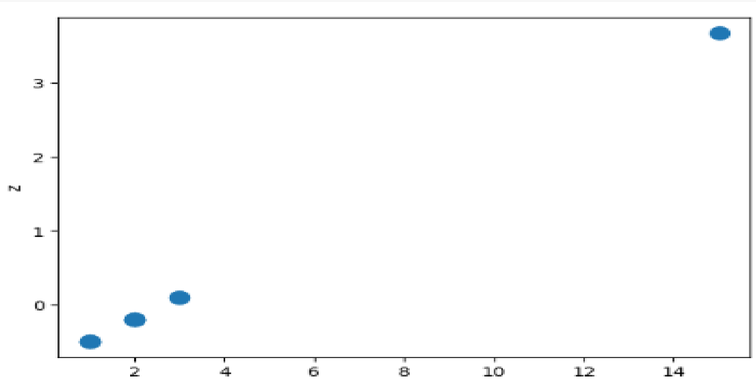
Double-click (or enter) to edit

```
[ ] data.plot( kind="density")
```

<Axes: ylabel='Density'>




```
x=data['score']
y=data['z']

data.plot.scatter(x = 'score', y = 'z', s = 100);
```



7. Develop the python script to import excel data into a Pandas data frame and pro excel data b. Fill in the missing values. c. Perform univariate analysis

```
import pandas as pd
```

```
[2] df1 = pd.read_excel('lab7.xlsx')
```

```
[3] print(df1)
```

```
      Name  Age    Stream  Percentage
0    Ankit   18      Math          95
1    Rahul   19   Science          85
2  Shaurya   20  Commerce          85
3    Raghu   18      Math          80
4    Priya   19   Science          75
```

a. Get the data types of the given excel data

```
[4] df1.dtypes
```

```
Name          object
Age            int64
Stream        object
Percentage     int64
dtype: object
```

c. Perform univariate analysis

```
[5] df1['Percentage'].mean()
```

```
84.0
```

```
[6] df1['Percentage'].median()
```

```
85.0
```

```
df1['Percentage'].mode()
```

```
0    85
Name: Percentage, dtype: int64
```

```
[8] df1['Percentage'].var()
```

```
55.0
```

```
[9] df1['Percentage'].std()
```

```
7.416198487095663
```

```python
df1.describe()
```

	Age	Percentage
count	5.00000	5.000000
mean	18.80000	84.000000
std	0.83666	7.416198
min	18.00000	75.000000
25%	18.00000	80.000000
50%	19.00000	85.000000
75%	19.00000	85.000000
max	20.00000	95.000000

b. Fill in the missing values.

[11]
```python
df2 = pd.read_excel('lab7.xlsx', sheet_name = 1)
```

[12]
```python
print(df2)
```

```
    Name  marks
0  akhil   80.0
1   banu   76.0
2   ravi    NaN
3  pooja   45.0
```

[13]
```python
df2.dtypes
```

```
Name      object
marks    float64
dtype: object
```

[14]
```python
require_cols = [0, 2]
```

[15]
```python
required_df = pd.read_excel('lab7.xlsx', usecols = require_cols)
```

[23]
```python
print(required_df)
```

```
      Name    Stream
0    Ankit      Math
1    Rahul   Science
2  Shaurya  Commerce
3    Raghu      Math
4    Priya   Science
```

```
[28] df = pd.read_excel('lab7.xlsx',sheet_name = 1)
```

```
[29] print(df)
```

```
    Name  marks
0  akhil   80.0
1   banu   76.0
2   ravi    NaN
3  pooja   45.0
```

```
[31] df['marks'].fillna(method='ffill')
```

```
0    80.0
1    76.0
2    76.0
3    45.0
Name: marks, dtype: float64
```

8. Develop the python script to import excel data into a Pandas data frame and process the following: a. Check duplicates and missing data b. Eliminate Mismatches c. Cleans line breaks, spaces, and special characters

a) Check duplicates and missing data

[1]
```
import pandas as pd

# read by default 1st sheet of an excel file
df = pd.read_excel('lab8.xlsx')
df
```

	A	B	C
0	L	ALpHA	'Hello, World!'
1	M	beta	'This\nis\na\ntest'
2	M	beta	'This\nis\na\ntest'
3	NaN	GaMMa	dfg
4	N	delta	*Some Extra '
5	P	PeNTa	'Special #$%@'

a. Check duplicates and missing data

```
df.drop_duplicates(inplace=True)
df
```

	A	B	C
0	L	ALpHA	'Hello, World!'
1	M	beta	'This\nis\na\ntest'
3	NaN	GaMMa	dfg
4	N	delta	*Some Extra '
5	P	PeNTa	'Special #$%@'

[3]
```
df.dropna(inplace=True)
df
```

	A	B	C
0	L	ALpHA	'Hello, World!'
1	M	beta	'This\nis\na\ntest'
4	N	delta	*Some Extra '
5	P	PeNTa	'Special #$%@'

b. Eliminate Mismatches

```
df['B'] = df['B'].str.lower()
df
```

	A	B	C
0	L	alpha	'Hello, World!'
1	M	beta	'This\nis\na\ntest'
4	N	delta	*Some Extra '
5	P	penta	'Special #$%@'

c)Clean link text line breaks, spaces, and special characters

```
df= df.apply(lambda x: x.str.replace(r'[\\n]', ' ', regex=True).str.strip())
df
```

```
df = df.apply(lambda x: x.str.replace(r'[^a-zA-Z0-9\s]', '', regex=True))
df
```

```python
# import the libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

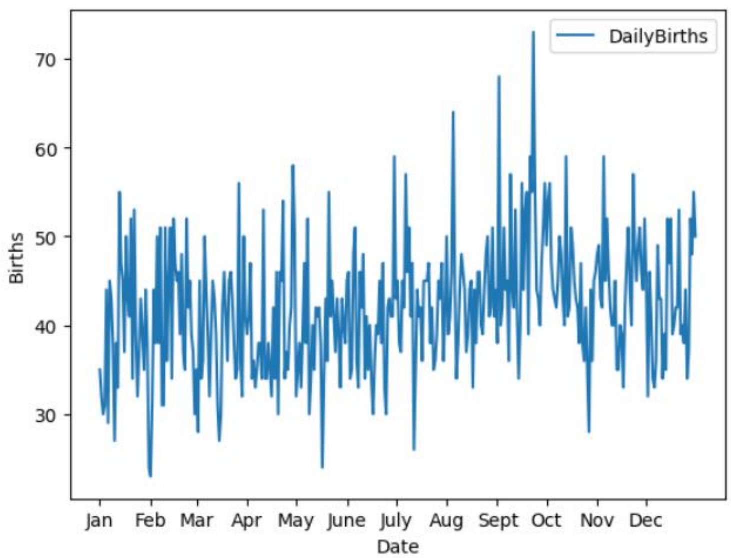
```python
data = pd.read_csv( "daily-total-female-births.csv")
data
```

	Date	Births
0	1959-01-01	35
1	1959-01-02	32
2	1959-01-03	30
3	1959-01-04	31
4	1959-01-05	44
...	...	...
360	1959-12-27	37
361	1959-12-28	52
362	1959-12-29	48
363	1959-12-30	55
364	1959-12-31	50

365 rows × 2 columns

```python
sns.lineplot( x = 'Date',
              y = 'Births',
              data = data,
              label = 'DailyBirths')
pos = [ '1959-01-01', '1959-02-01', '1959-03-01', '1959-04-01',
        '1959-05-01', '1959-06-01', '1959-07-01', '1959-08-01',
        '1959-09-01', '1959-10-01', '1959-11-01', '1959-12-01']

lab = [ 'Jan', 'Feb', 'Mar', 'Apr', 'May', 'June',
        'July', 'Aug', 'Sept', 'Oct', 'Nov', 'Dec']

plt.xticks( pos, lab)
```

```
([<matplotlib.axis.XTick at 0x7efee274a3b0>,
```



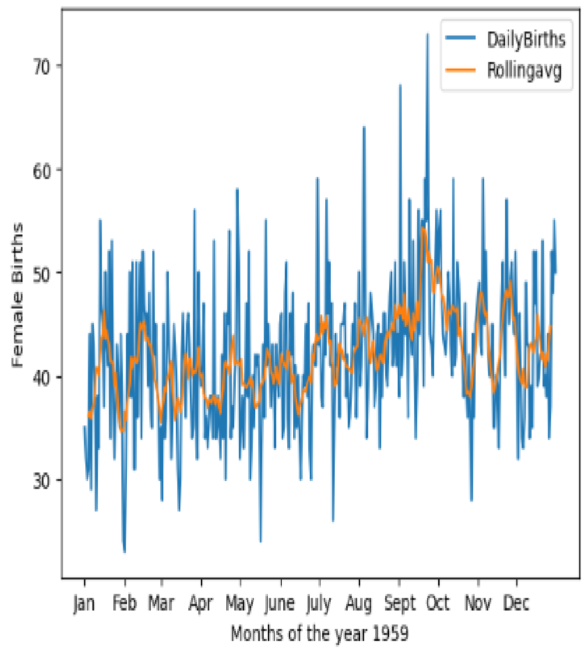
We can notice that it is very difficult to gain knowledge from the above plot as the data fluctuates a lot. So, let us plot it again but using the Rolling Average concept this time.

```python
# computing a 7 day rolling average
data[ '7day_rolling_avg' ] = data.Births.rolling(8).mean().shift(-4)

# viewing the dataset
data.head(10)
```

	Date	Births	7day_rolling_avg
0	1959-01-01	35	NaN
1	1959-01-02	32	NaN
2	1959-01-03	30	NaN
3	1959-01-04	31	36.125
4	1959-01-05	44	36.500
5	1959-01-06	29	35.875
6	1959-01-07	45	36.875
7	1959-01-08	43	37.125
8	1959-01-09	38	38.500
9	1959-01-10	27	40.750

```python
sns.lineplot( x = 'Date',
              y = 'Births',
              data = data,
              label = 'DailyBirths')

# plot using rolling average
sns.lineplot( x = 'Date',
              y = '7day_rolling_avg',
              data = data,
              label = 'Rollingavg')

plt.xlabel('Months of the year 1959')

# setting customized ticklabels for x axis
pos = [ '1959-01-01', '1959-02-01', '1959-03-01', '1959-04-01',
        '1959-05-01', '1959-06-01', '1959-07-01', '1959-08-01',
        '1959-09-01', '1959-10-01', '1959-11-01', '1959-12-01']

lab = [ 'Jan', 'Feb', 'Mar', 'Apr', 'May', 'June',
        'July', 'Aug', 'Sept', 'Oct', 'Nov', 'Dec']

plt.xticks( pos, lab)

plt.ylabel('Female Births')
```

```
Text(0, 0.5, 'Female Births')
```



We can clearly see through the above graph that the rolling average has smoothened the number of female births, and we can notice the peak more evidently.

10. Perform correlation analysis of all variables in python

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
```

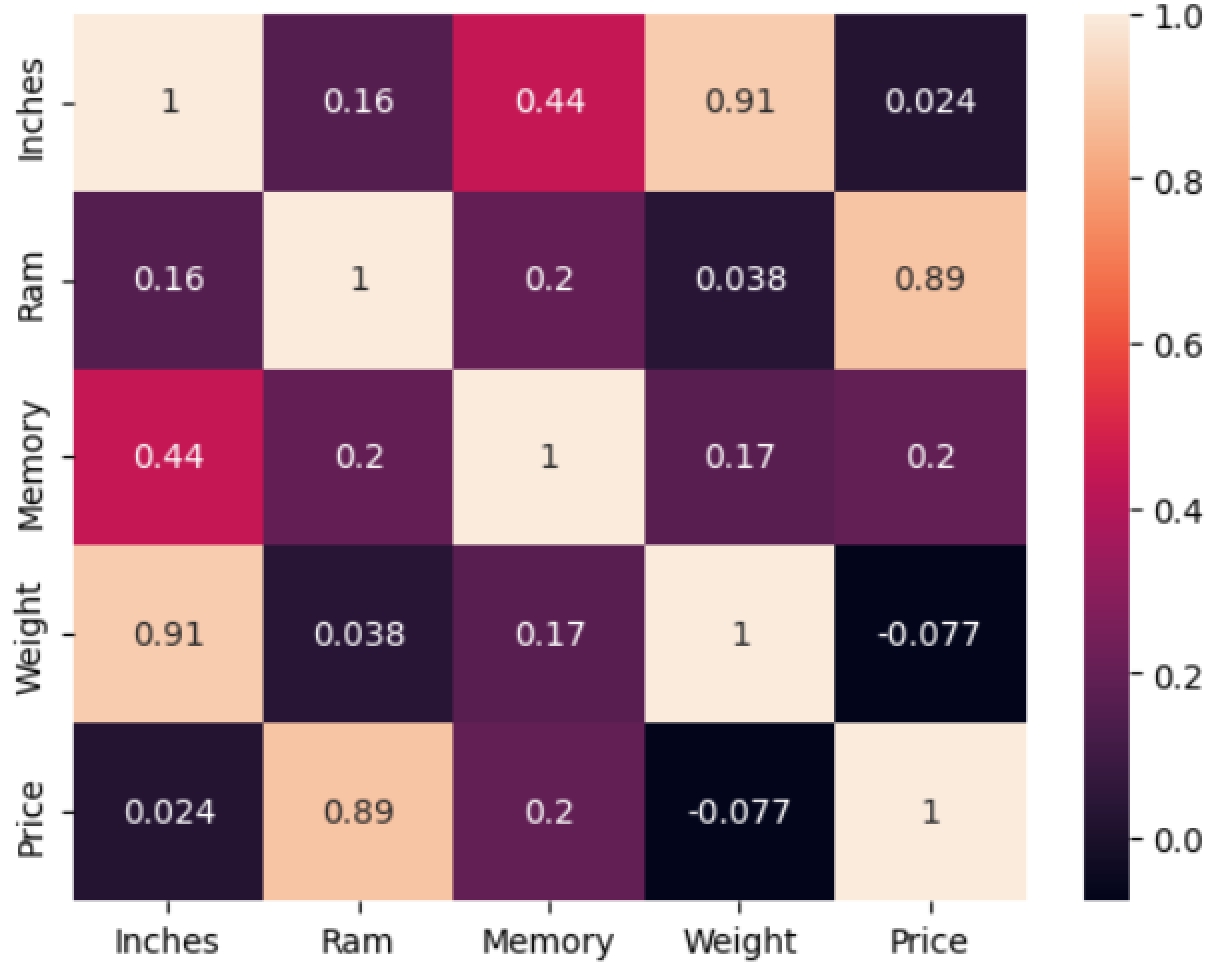
[3]
```
df = pd.read_csv('lab10.csv')
df.head(5)
```

	Inches	Ram	Memory	Weight	Price
0	13.3	8	128	1.37	71378.6832
1	13.3	8	128	1.34	47895.5232
2	15.6	8	256	1.86	30636.0000
3	15.4	16	512	1.83	135195.3360
4	13.3	8	256	1.37	96095.8080

```
corr = df.corr()
corr
```

	Inches	Ram	Memory	Weight	Price
Inches	1.000000	0.161631	0.442950	0.911092	0.024203
Ram	0.161631	1.000000	0.197623	0.037932	0.888951
Memory	0.442950	0.197623	1.000000	0.174460	0.195272
Weight	0.911092	0.037932	0.174460	1.000000	-0.077049
Price	0.024203	0.888951	0.195272	-0.077049	1.000000

```python
fig, ax = plt.subplots()
sb.heatmap(corr, annot=True)
plt.show()
```



Solve

Download heart-disease.csv

Perform all above operation

and Write About +Ve and -Ve correlation

11. Perform regression analysis in python.

```
[1] import pandas as pd
    import numpy as np
    from sklearn import linear_model
    import matplotlib.pyplot as plt
```

```
[2] df = pd.read_csv('lab11.csv')
    df
```

	area	price
0	2600	550000
1	3000	565000
2	3200	610000
3	3600	680000
4	4000	725000

```
[3] x = df.drop('price',axis='columns')
    x=x.values
    x
```

```
array([[2600],
       [3000],
       [3200],
       [3600],
       [4000]])
```

```
price = df.price
price
```

```
0    550000
1    565000
2    610000
3    680000
4    725000
Name: price, dtype: int64
```

```
[5] reg = linear_model.LinearRegression()
    reg.fit(x,price)
```

```
▾ LinearRegression
LinearRegression()
```

```
[6] reg.predict([[3000]])
```

```
array([587979.45205479])
```

```
[7] reg.coef_
```

```
array([135.78767123])
```

```
[8] reg.intercept_
```

```
180616.43835616432
```

	Area	Price		
	X Value	**Y Value**	**X*Y**	**X*X**
1	2600	550000	1430000000	6760000
2	3000	565000	1695000000	9000000
3	3200	610000	1952000000	10240000
4	3600	680000	2448000000	12960000
5	4000	725000	2900000000	16000000
Sum	16400	3130000	10425000000	54960000

coeff/Slope(m)= (NΣXY - (ΣX)(ΣY)) / (NΣX^2 - $(ΣX)^2$) 135.7876712

Intercept(b) = (ΣY - m(ΣX)) / N
180616.4384

y=mx+b 135.7876*(4000)+180616.4384= **587979.452**

Solve

crteate a CSV file and perform regression analysis and calculations with following data

X Value	Y Value
60	3.1
61	3.6
62	3.8
63	4
65	4.1

12. Perform data analysis on titanic dataset

```
import pandas as pd

#loading data
titanic = pd.read_csv('titanic.csv')
titanic.head(5)
```

	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

1. PassengerId: Unique Id of a passenger
2. Survived: If the passenger survived(0-No, 1-Yes)
3. Pclass: Passenger Class (1 = 1st, 2 = 2nd, 3 = 3rd)
4. Name: Name of the passenger
5. Sex: Male/Female
6. Age: Passenger age in years
7. SibSp: No of siblings/spouses aboard
8. Parch: No of parents/children aboard
9. Ticket: Ticket Number
10. Fare: Passenger Fare
11. Cabin: Cabin number
12. Embarked: Port of Embarkation (C = Cherbourg; Q = Queenstown; S = Southampton)

```
[ ] titanic.isnull().sum()
```

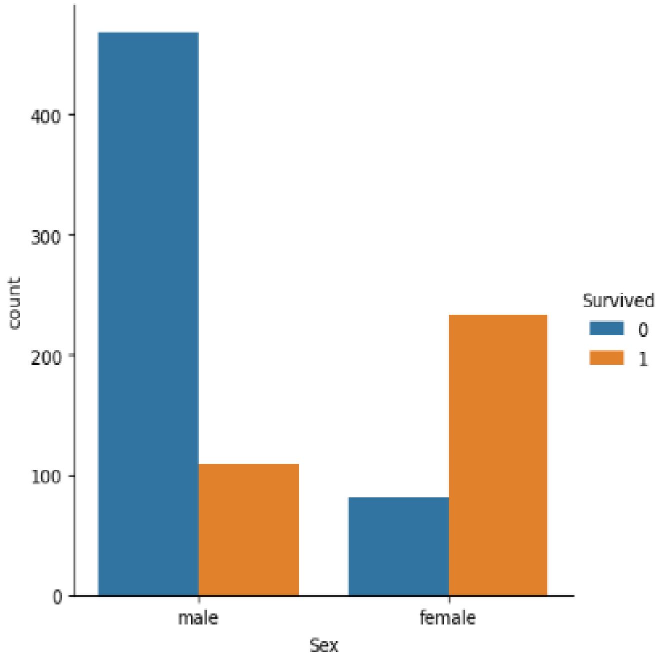
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

Features: The titanic dataset has roughly the following types of features:

- **Categorical/Nominal**: Variables that can be divided into multiple categories but having no order or priority.
  Eg. Embarked (C = Cherbourg; Q = Queenstown; S = Southampton)
- **Binary**: A subtype of categorical features, where the variable has only two categories.
  Eg: Sex (Male/Female)
- **Ordinal**: They are similar to categorical features but they have an order(i.e can be sorted).
  Eg. Pclass (1, 2, 3)
- **Continuous**: They can take up any value between the minimum and maximum values in a column.
  Eg. Age, Fare
- **Count**: They represent the count of a variable.
  Eg. SibSp, Parch
- **Useless**: They don't contribute Here, *PassengerId, Name, Cabin* and *Ticket* might fall into this category.

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

# Countplot
sns.catplot(x ="Sex", hue ="Survived",
kind ="count", data = titanic)
```

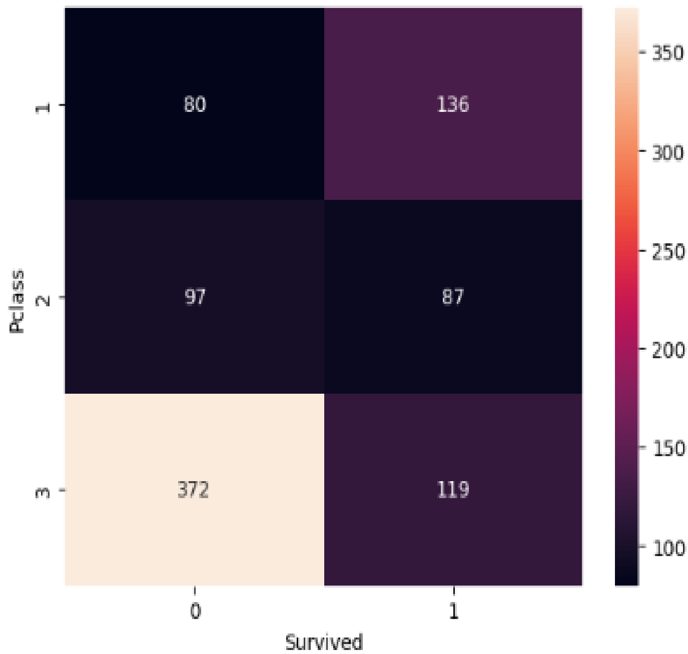
<seaborn.axisgrid.FacetGrid at 0x7ea0714254b0>



Just by observing the graph, it can be approximated that the survival rate of men is around 20% and that of women is around 75%. Therefore, whether a passenger is a male or a female plays an important role in determining if one is going to survive.

```python
# Group the dataset by Pclass and Survived and then unstack them
group = titanic.groupby(['Pclass', 'Survived'])
pclass_survived = group.size().unstack()

# Heatmap - Color encoded 2D representation of data.
sns.heatmap(pclass_survived, annot = True, fmt ="d")
```

```
<Axes: xlabel='Survived', ylabel='Pclass'>
```

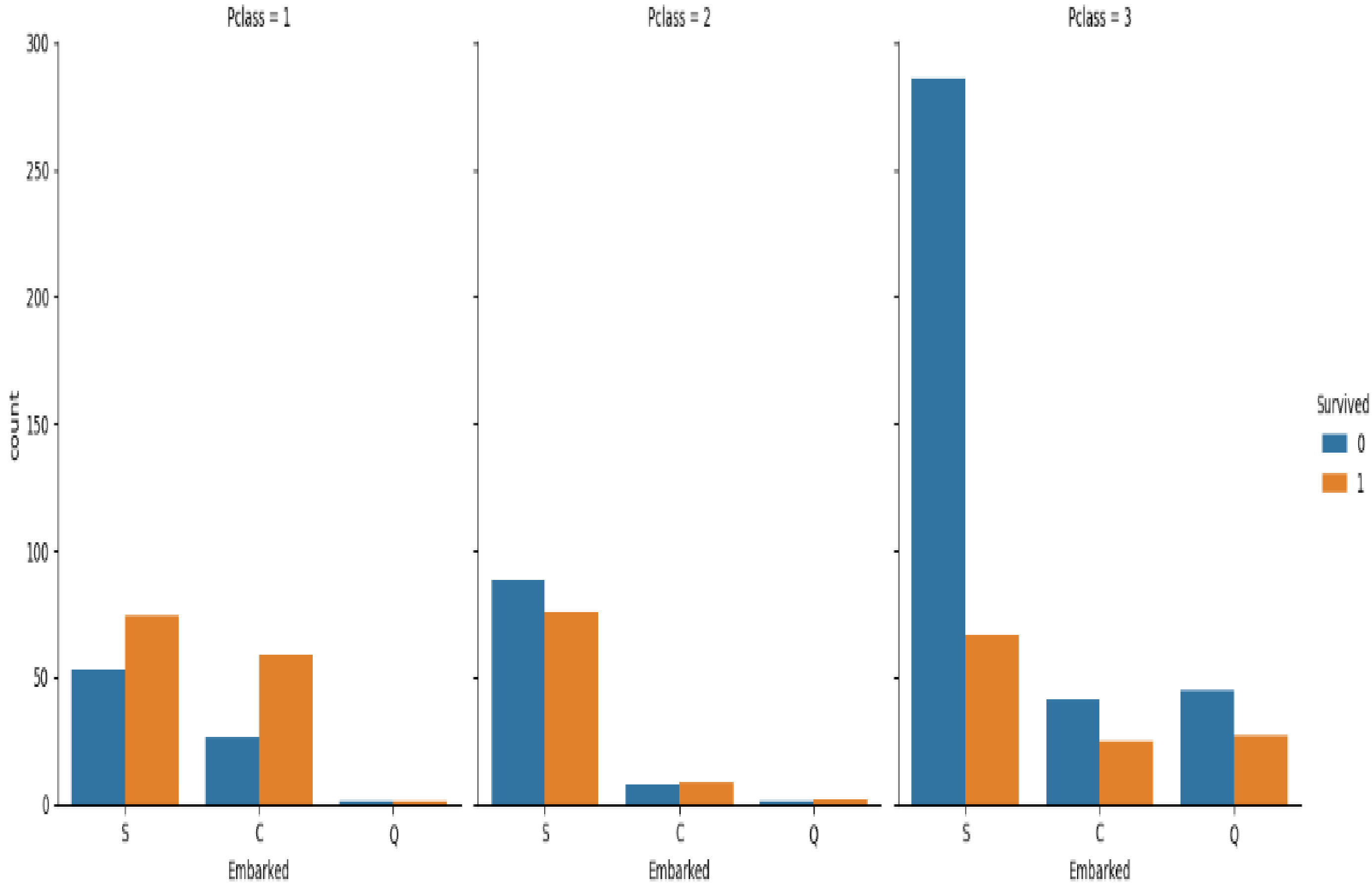


- It helps in determining if higher-class passengers had more survival rate than the lower class ones or vice versa.

Class 1 passengers have a higher survival chance compared to classes 2 and 3. It implies that Pclass contributes a lot to a passenger's survival rate.

```python
[ ] sns.catplot(x ='Embarked', hue ='Survived',
    kind ='count', col ='Pclass', data = titanic)
```

```
<seaborn.axisgrid.FacetGrid at 0x7ea067695ae0>
```



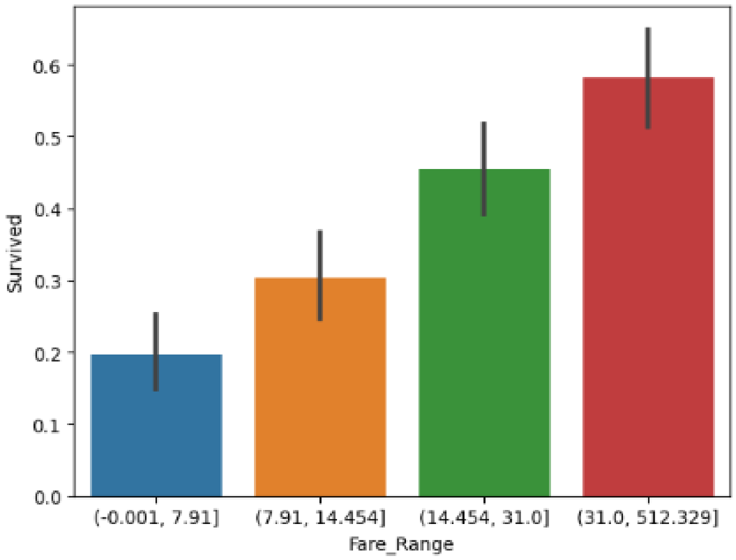
- Majority of the passengers boarded from S.

  S looks lucky for class 1 and 2 passengers compared to class 3.

[ ]

```
# Divide Fare into 4 bins
titanic['Fare_Range'] = pd.qcut(titanic['Fare'], 4)

# Barplot - Shows approximate values based
# on the height of bars.
sns.barplot(x ='Fare_Range', y ='Survived',
data = titanic)
```

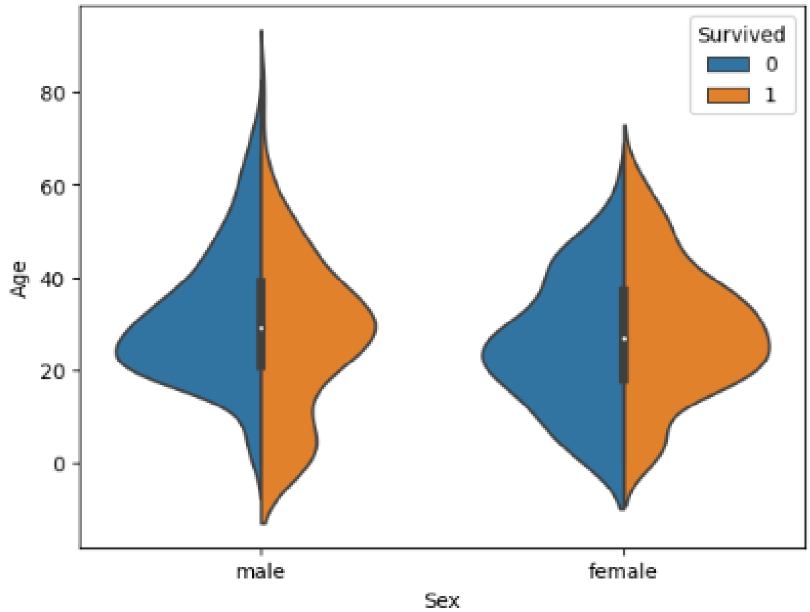
```
<Axes: xlabel='Fare_Range', ylabel='Survived'>
```



Fare denotes the fare paid by a passenger. As the values in this column are continuous, they need to be put in separate bins(as done for Age feature) to get a clear idea. It can be concluded that if a passenger paid a higher fare, the survival rate is more.

```python
# Violinplot Displays distribution of data
# across all levels of a category.
sns.violinplot(x ="Sex", y ="Age", hue ="Survived",
data = titanic, split = True)
```

```
<Axes: xlabel='Sex', ylabel='Age'>
```



This graph gives a summary of the age range of men, women and children who were saved. The survival rate is –

Good for children.

High for women in the age range 20-50.

Less for men as the age increases.

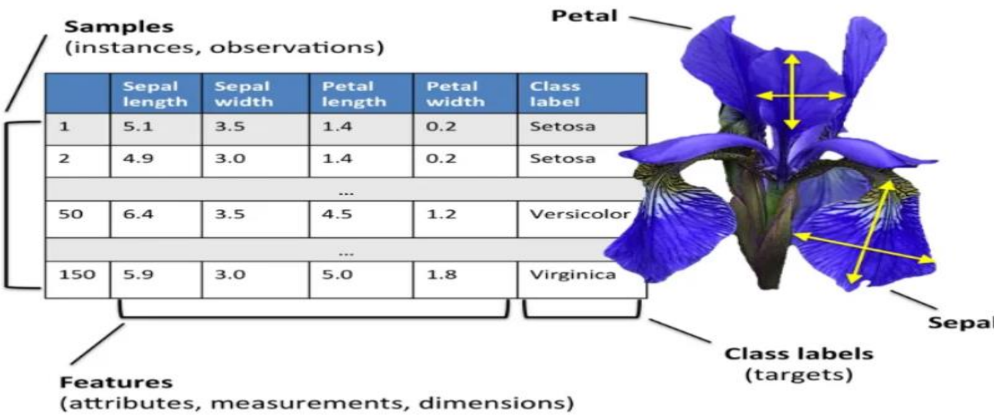
*Conclusion : *

The columns that can be dropped are:

PassengerId, Name, Ticket, Cabin: They are strings, cannot be categorized and don't contribute much to the outcome.

Age, Fare: Instead, the respective range columns are retained.

Lab 13-Perform data analysis on iris dataset.




```
import pandas as pd
 df = pd.read_csv("Iris.csv")
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

Getting Information about the Dataset

```
[10] 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
```

1 All columns are not having any Null Entries

2 Four columns are numerical type

3 Only Single column categorical type

Statistical Insight

```
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
max	150.000000	7.900000	4.400000	6.900000	2.500000

+ Code + Te

We can see the count of each column along with their mean value, standard deviation, minimum and maximum values.

Checking Missing Values

[12]
```
df.isnull().sum()
```

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

We can see that no column as any missing value.

Checking the balance

We can see that there are only three unique species. Let's see if the dataset is balanced or not i.e. all the species contain equal amounts of rows or not

[13]
```
df.value_counts("Species")
```

```
Species
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
dtype: int64
```

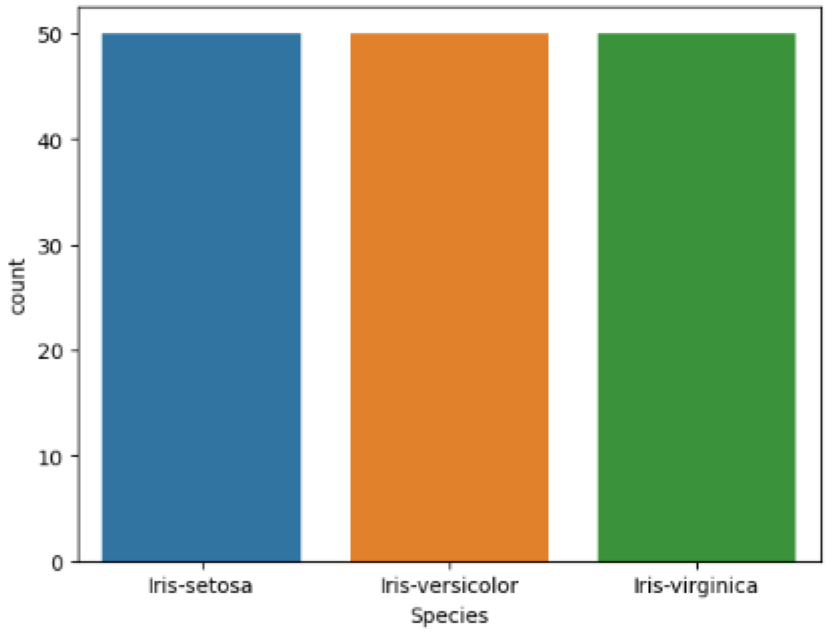
We can see that all the species contain an equal amount of rows

Data Visualization*

Visualizing the target column

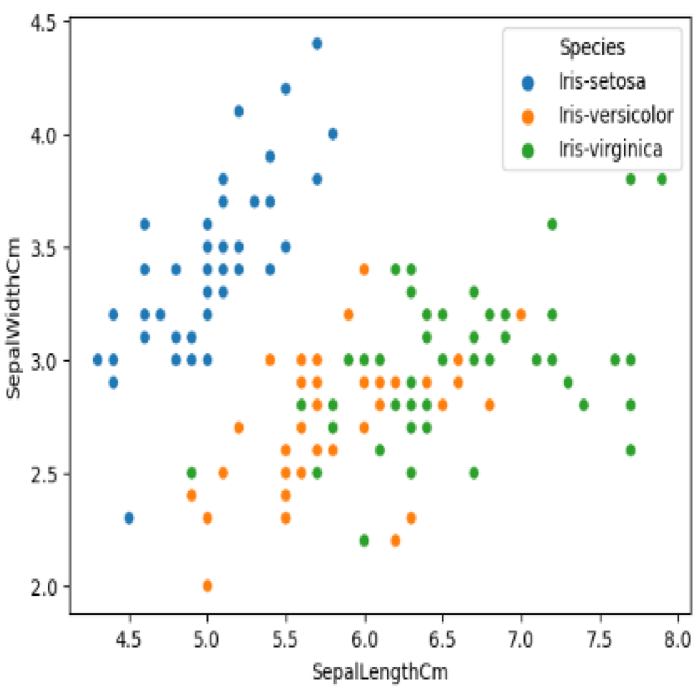
Our target column will be the Species column because at the end we will need the result according to the species only.

[14]
```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt


sns.countplot(x='Species', data=df, )
plt.show()
```



Comparing Sepal Length and Sepal Width

[16]
```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt

sns.scatterplot(x='SepalLengthCm', y='SepalWidthCm',
                hue='Species', data=df, )
plt.show()
```

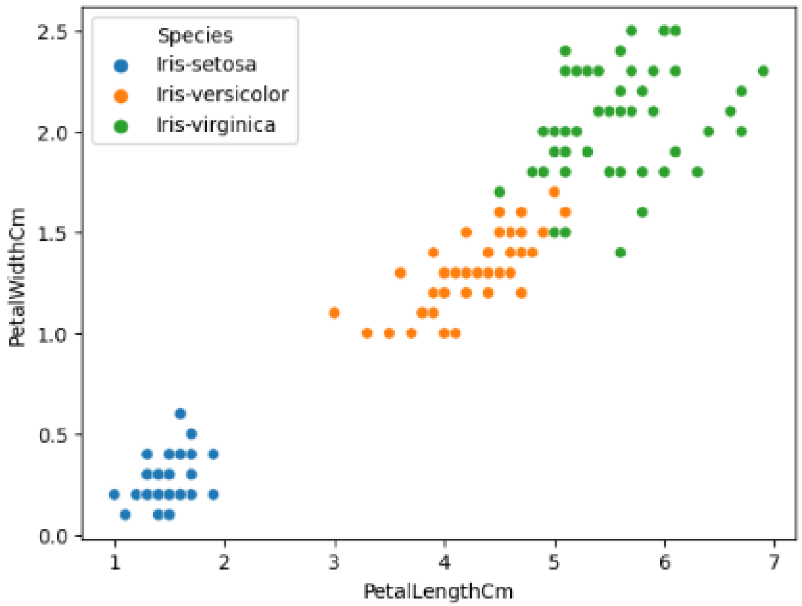


From the above plot, we can infer that –

Species Setosa has smaller sepal lengths but larger sepal widths. Versicolor Species lies in the middle of the other two species in terms of sepal length and width Species Virginica has larger sepal lengths but smaller sepal widths.

Comparing Petal Length and Petal Width

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.scatterplot(x='PetalLengthCm', y='PetalWidthCm',
                hue='Species', data=df, )
plt.show()
```



*From the above plot, we can infer that – *

Species Setosa has smaller petal lengths and widths.

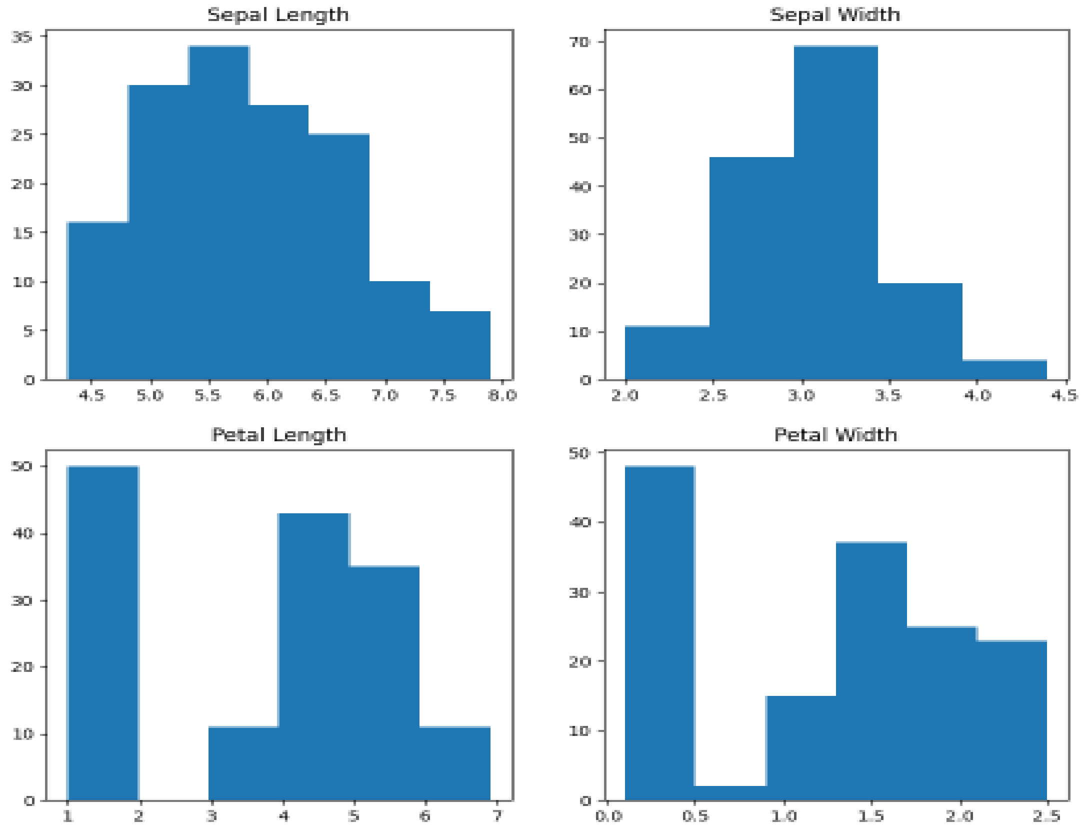
Versicolor Species lies in the middle of the other two species in terms of petal length and width

Species Virginica has the largest of petal lengths and widths.

Histograms

Histograms allow seeing the distribution of data for various columns. It can be used for uni as well as bi-variate analysis.

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt


fig, axes = plt.subplots(2, 2, figsize=(10,10))

axes[0,0].set_title("Sepal Length")
axes[0,0].hist(df['SepalLengthCm'], bins=7)

axes[0,1].set_title("Sepal Width")
axes[0,1].hist(df['SepalWidthCm'], bins=5);

axes[1,0].set_title("Petal Length")
axes[1,0].hist(df['PetalLengthCm'], bins=6);

axes[1,1].set_title("Petal Width")
axes[1,1].hist(df['PetalWidthCm'], bins=6);
```



From the above plot, we can see that

The highest frequency of the sepal length is between 30 and 35 which is between 5.5 and 6

The highest frequency of the sepal Width is around 70 which is between 3.0 and 3.5

The highest frequency of the petal length is around 50 which is between 1 and 2

The highest frequency of the petal width is between 40 and 50 which is between 0.0 and 0.5

Handling Correlation

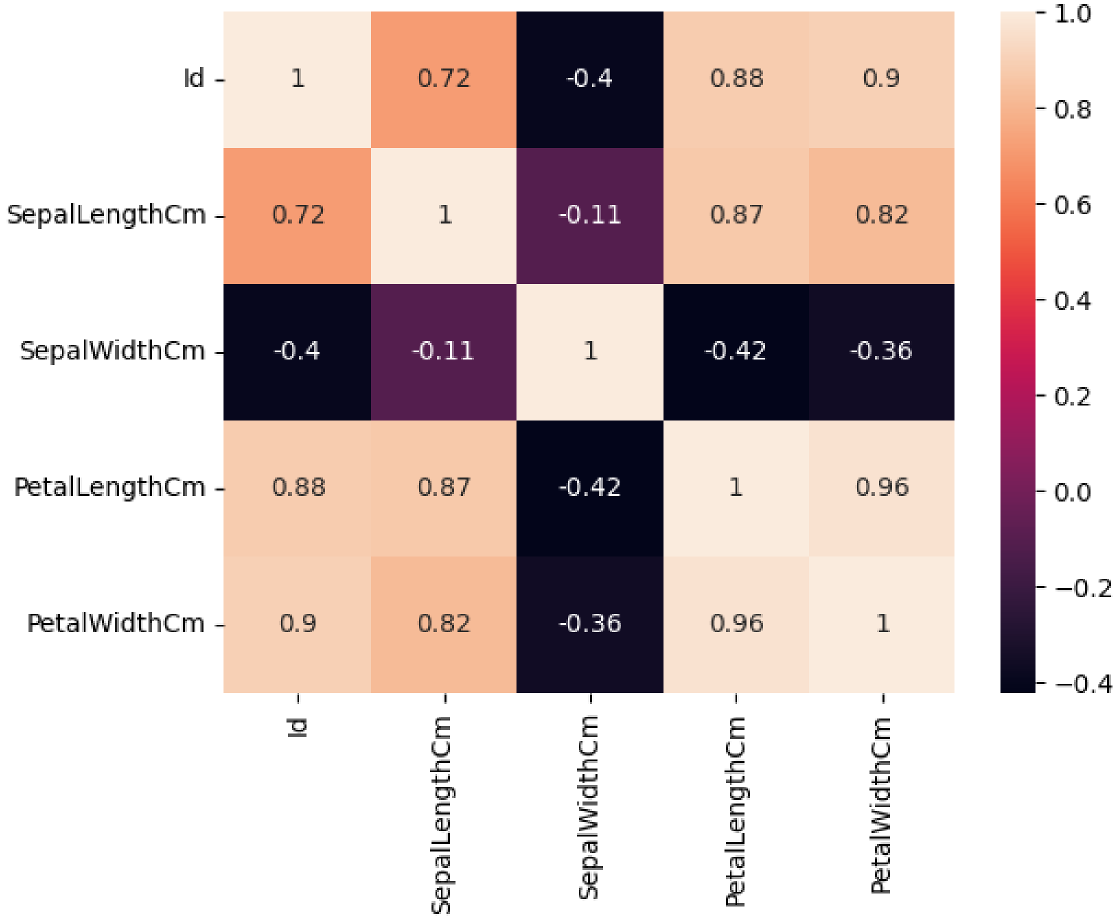
[21]
```
df.corr()
```

```
<ipython-input-21-2f6f6606aa2c>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is depre
  df.corr()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
Id	1.000000	0.716676	-0.397729	0.882747	0.899759
SepalLengthCm	0.716676	1.000000	-0.109369	0.871754	0.817954
SepalWidthCm	-0.397729	-0.109369	1.000000	-0.420516	-0.356544
PetalLengthCm	0.882747	0.871754	-0.420516	1.000000	0.962757
PetalWidthCm	0.899759	0.817954	-0.356544	0.962757	1.000000

```
sns.heatmap(df.corr(),annot = True);

plt.show()
```

```
<ipython-input-24-c0f1bc477367>:1: FutureWarning: The default value of numeric_only in DataFrame.co
  sns.heatmap(df.corr(),annot = True);
```



From the above graph, we can see that –

Petal width and petal length have high correlations.

Petal length and sepal width have good correlations.

Petal Width and Sepal length have good correlations.