Experiment No. 1

Aim:

Data Wrangling, I

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

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

Requirement:

- Anaconda Installer
- Windows 10 OS
- Jupyter Notebook

Theory:

Data wrangling is the process of cleaning and unifying messy and complex data sets for easy access and analysis.

With the amount of data and data sources rapidly growing and expanding, it is getting increasingly essential for large amounts of available data to be organized for analysis. This process typically includes manually converting and mapping data from one raw form into another format to allow for more convenient consumption and organization of the data.

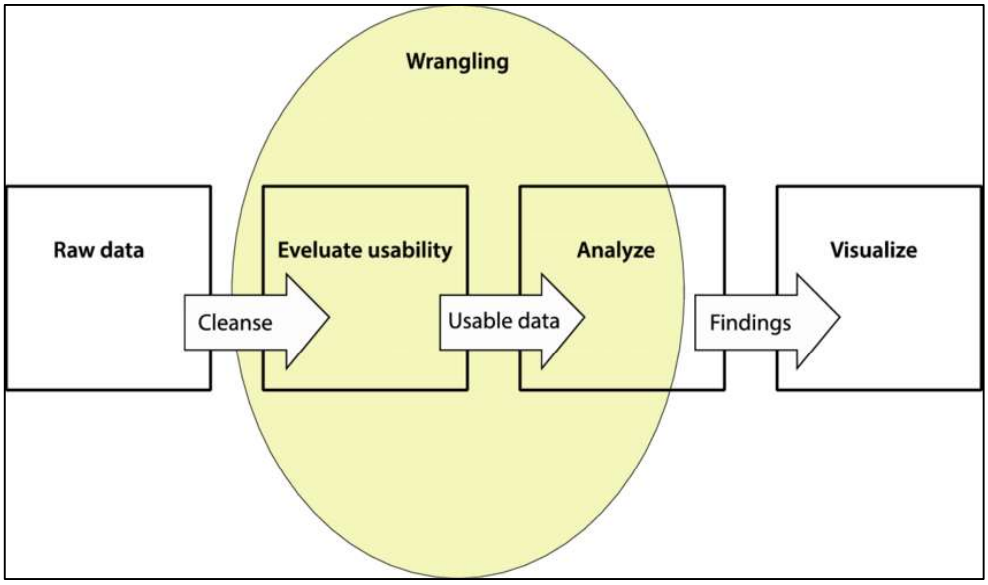
The Goals of Data Wrangling:

- Reveal a "deeper intelligence" by gathering data from multiple sources
- Provide accurate, actionable data in the hands of business analysts in a timely matter

- Reduce the time spent collecting and organizing unruly data before it can be utilized
- Enable data scientists and analysts to focus on the analysis of data, rather than the wrangling
- Drive better decision-making skills by senior leaders in an organization

Key steps to Data Wrangling:

- Data Acquisition: Identify and obtain access to the data within your sources.
- Joining Data: Combine the edited data for further use and analysis.
- Data Cleansing: Redesign the data into a usable and functional format and correct/remove any bad data.



Libraries Used:

Pandas: Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring and manipulating data.

Numpy: NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices.

Conclusion:

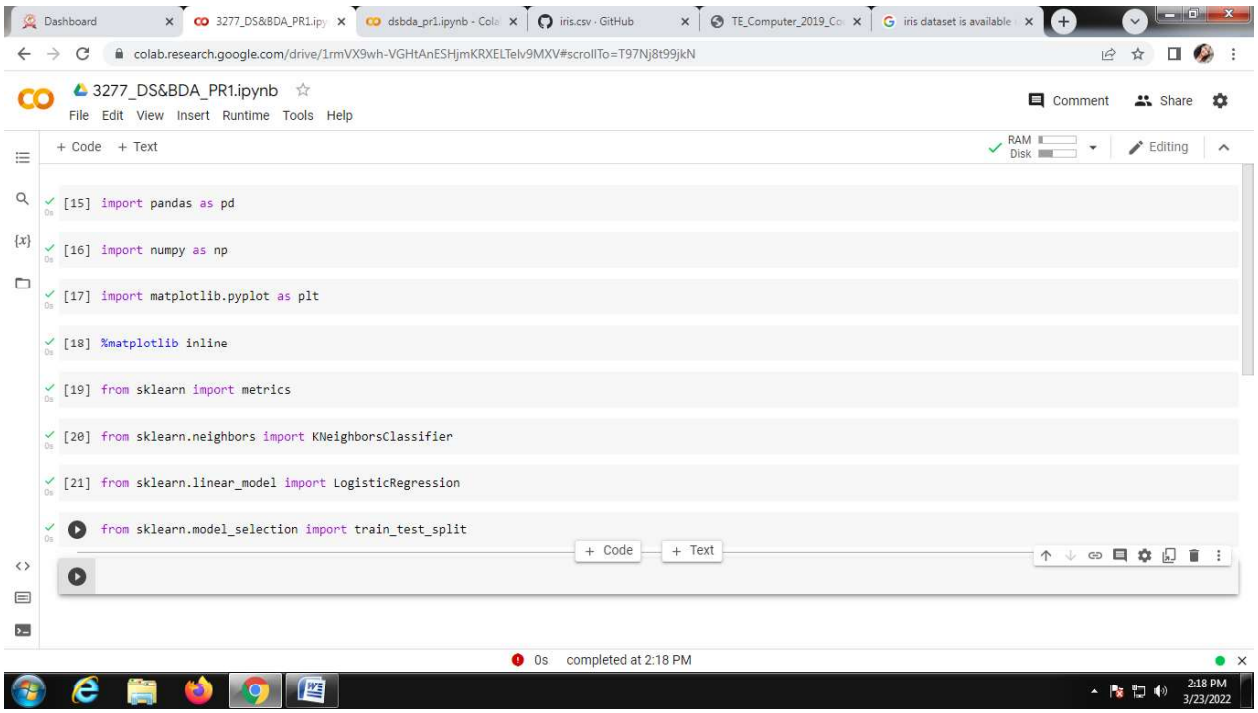
Hence, we have implemented data wrangling practical.

Practical 1

Data Wrangling I

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

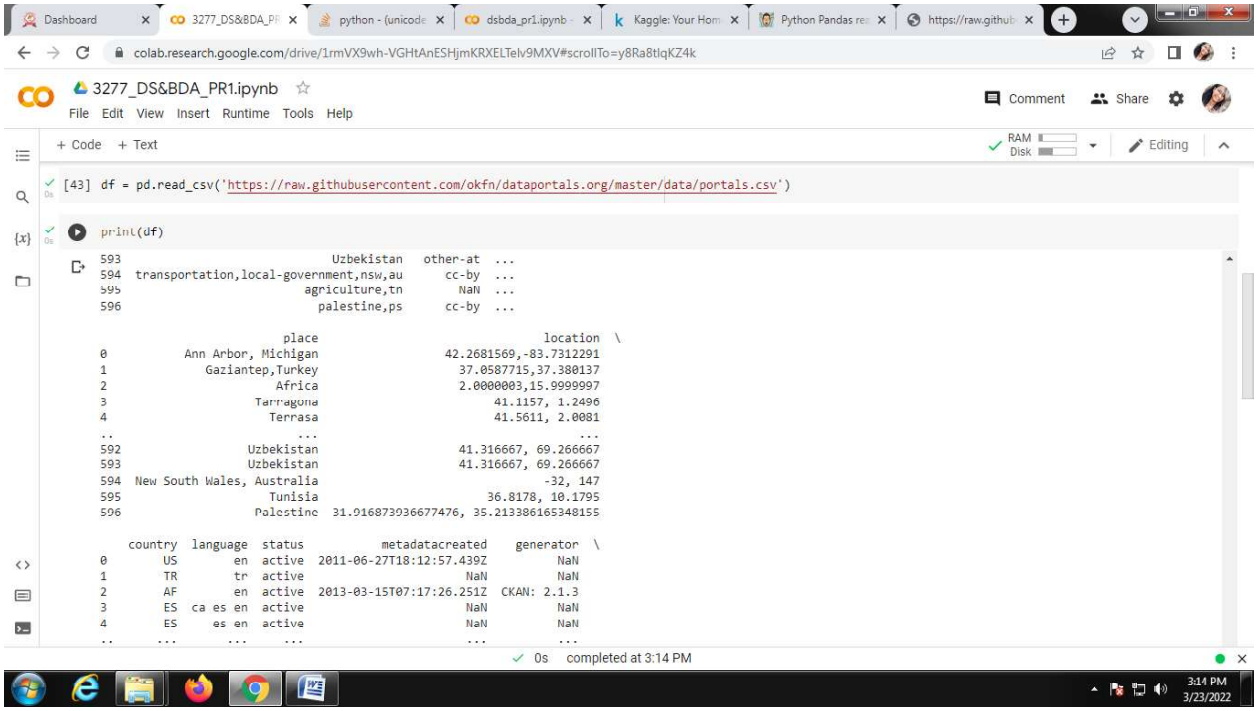
1. Import all the required Python Libraries



2. Locate an open source data from the web (e.g. https://www.kaggle.com). Provide a clear description of the data and its source (i.e. URL of the web site).

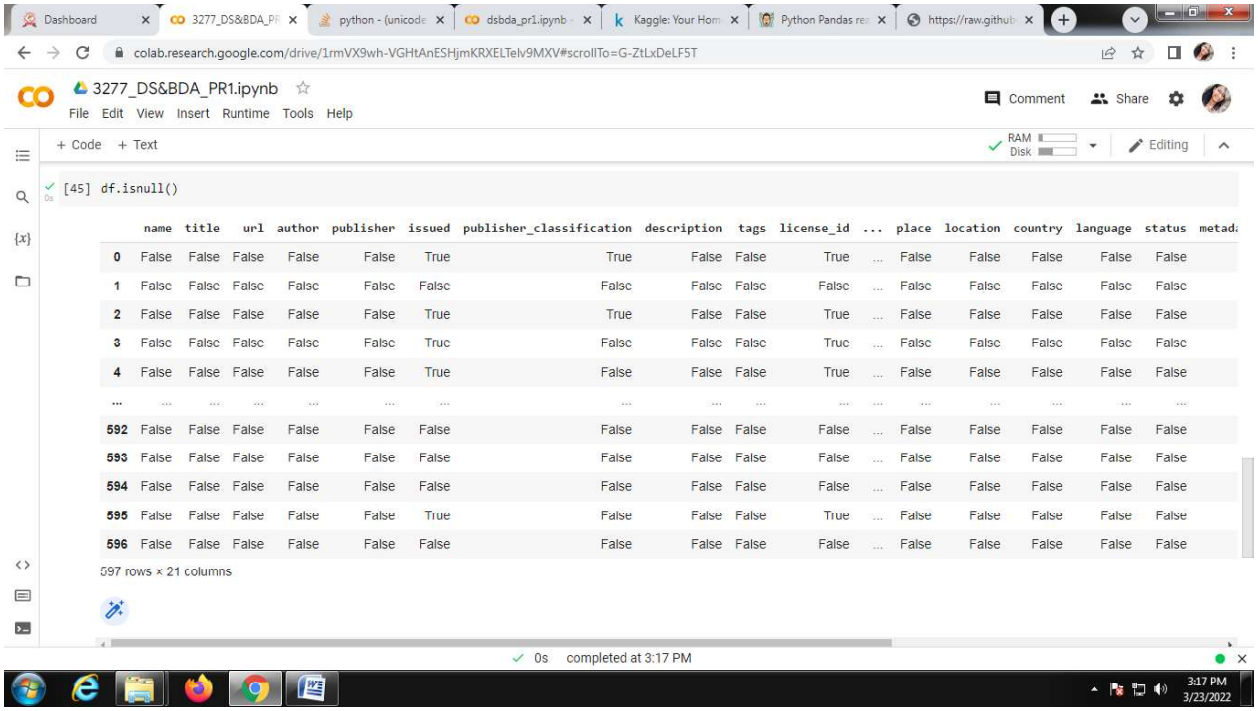
   Description: I've used portals dataset for this practical which is freely available on internet on data portals site.

3. Load the Dataset into Pandas DataFrame.



4. Data Preprocessing: check for missing values in the data using pandas isnull(), describe() function to get some initial statistics. Provide variable descriptions. Types of variables etc. Check the dimensions of the data frame.

isnull() : to check the missing values



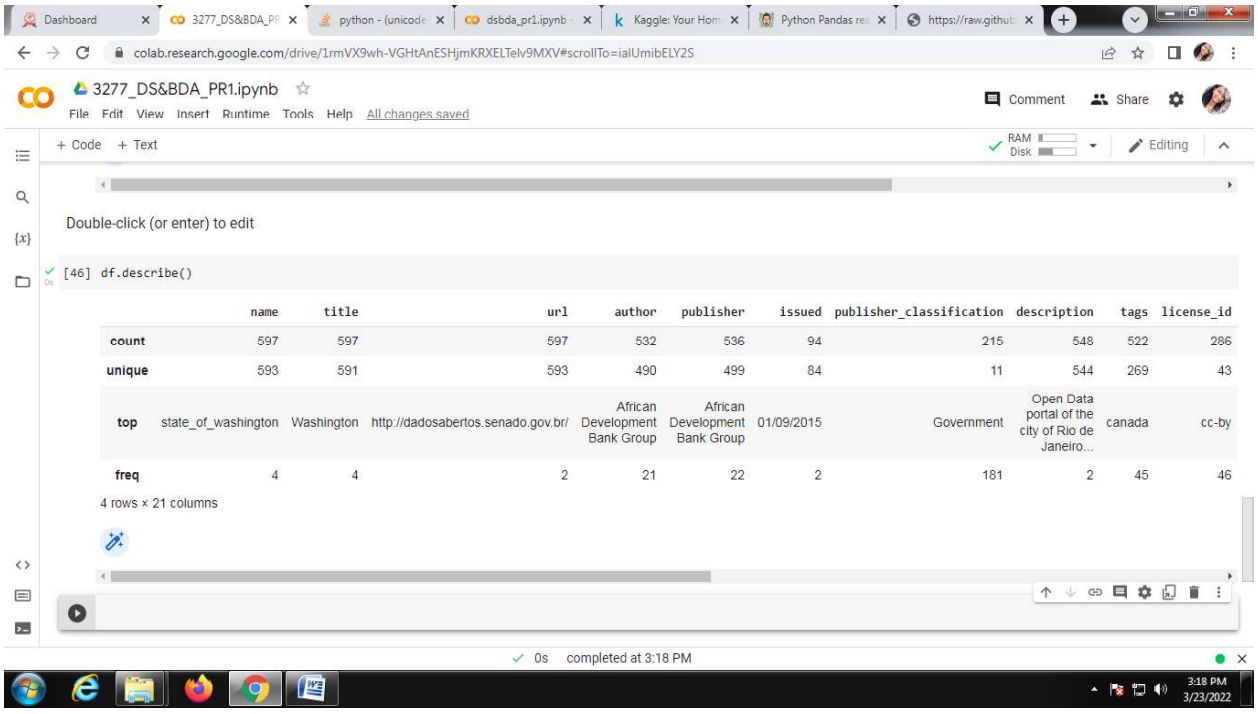
```
[45] df.isnull()
```

	name	title	url	author	publisher	issued	publisher_classification	description	tags	license_id	...	place	location	country	language	status	metad
0	False	False	False	False	False	True	True	False	False	True	...	False	False	False	False	False	
1	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	
2	False	False	False	False	False	True	True	False	False	True	...	False	False	False	False	False	
3	False	False	False	False	False	True	False	False	False	True	...	False	False	False	False	False	
4	False	False	False	False	False	True	False	False	False	True	...	False	False	False	False	False	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
592	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	
593	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	
594	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	
595	False	False	False	False	False	True	False	False	False	True	...	False	False	False	False	False	
596	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	

597 rows × 21 columns

0s completed at 3:17 PM

describe() : returns the statistical summary of dataframe or series.



Double-click (or enter) to edit

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

	name	title	url	author	publisher	issued	publisher_classification	description	tags	license_id
count	597	597	597	532	536	94	215	548	522	286
unique	593	591	593	490	499	84	11	544	269	43
top	state_of_washington	Washington	http://dadosabertos.senado.gov.br/	African Development Bank Group	African Development Bank Group	01/09/2015	Government	Open Data portal of the city of Rio de Janeiro...	canada	cc-by
freq	4	4	2	21	22	2	181	2	45	46

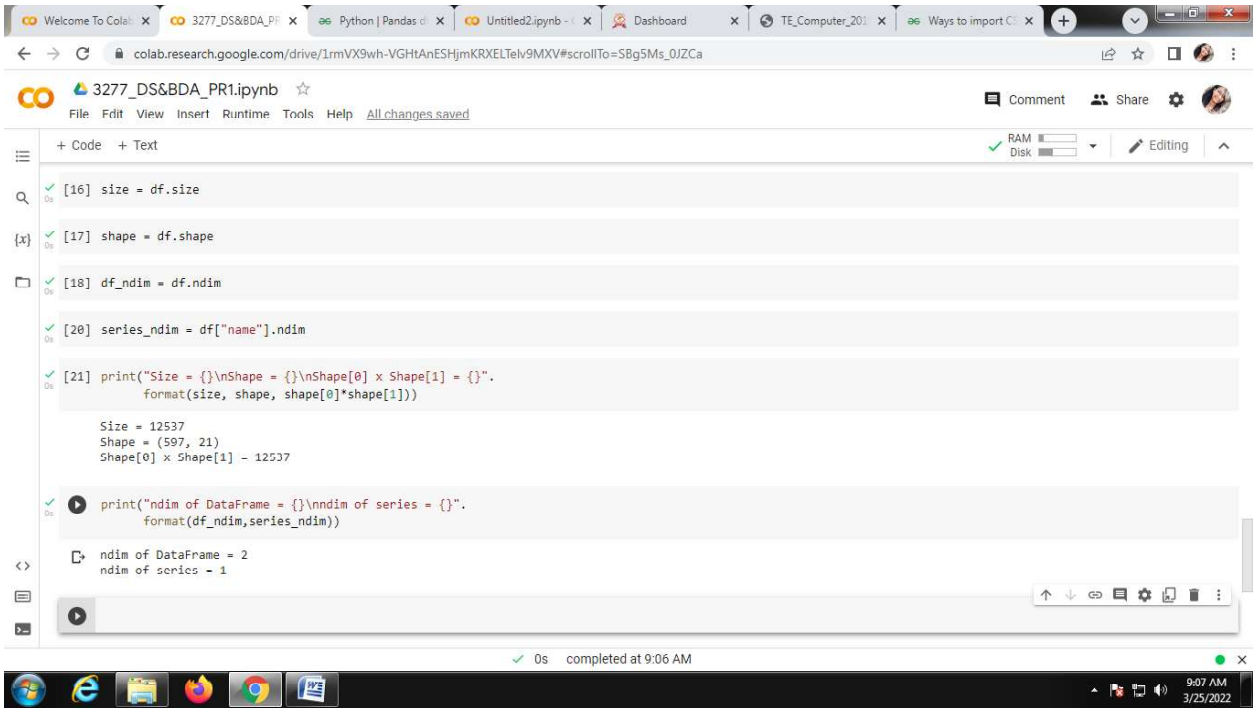
4 rows × 21 columns

0s completed at 3:18 PM

size() : count the number of element along given axis.

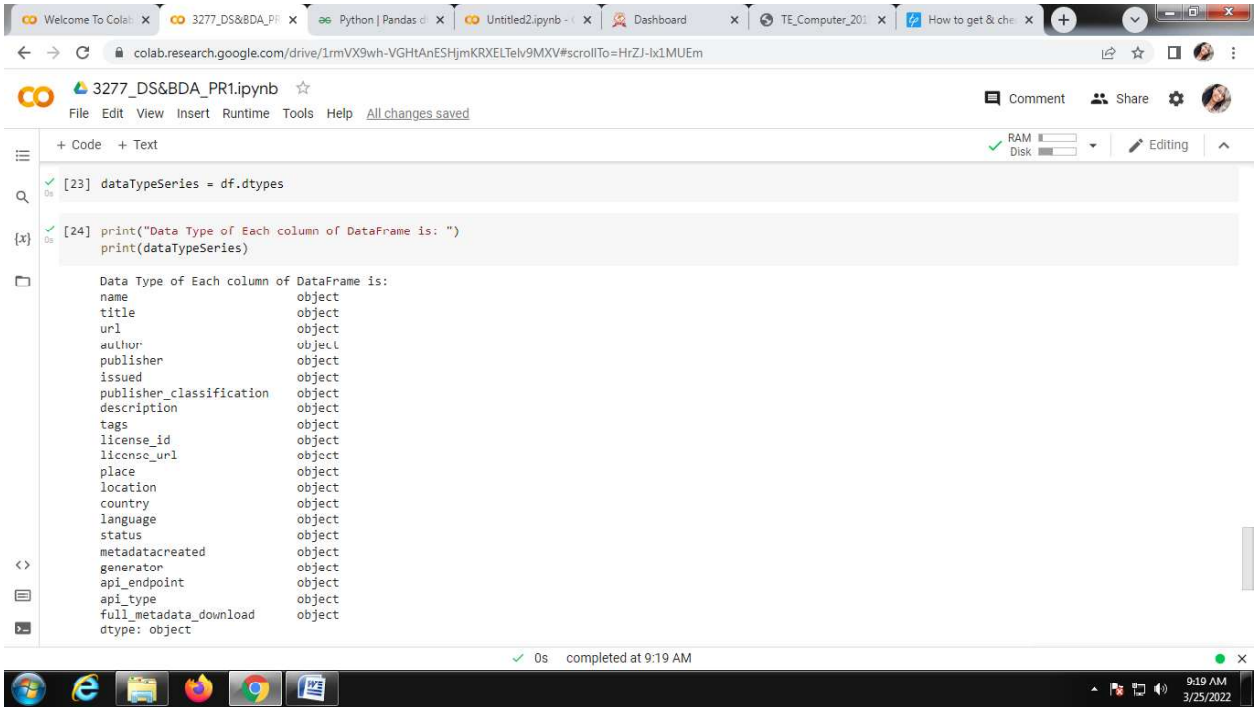
shape() : gives the number of elements in each dimension of an array.

ndim() : return the number of dimensions of an array.



5. Data Formatting and Data Normalization: Summarize the types of variables by checking the data types (i.e., character, numeric, integer, factor, and logical) of the variables in the data set. If variables are not in the correct data type, apply proper type conversions.

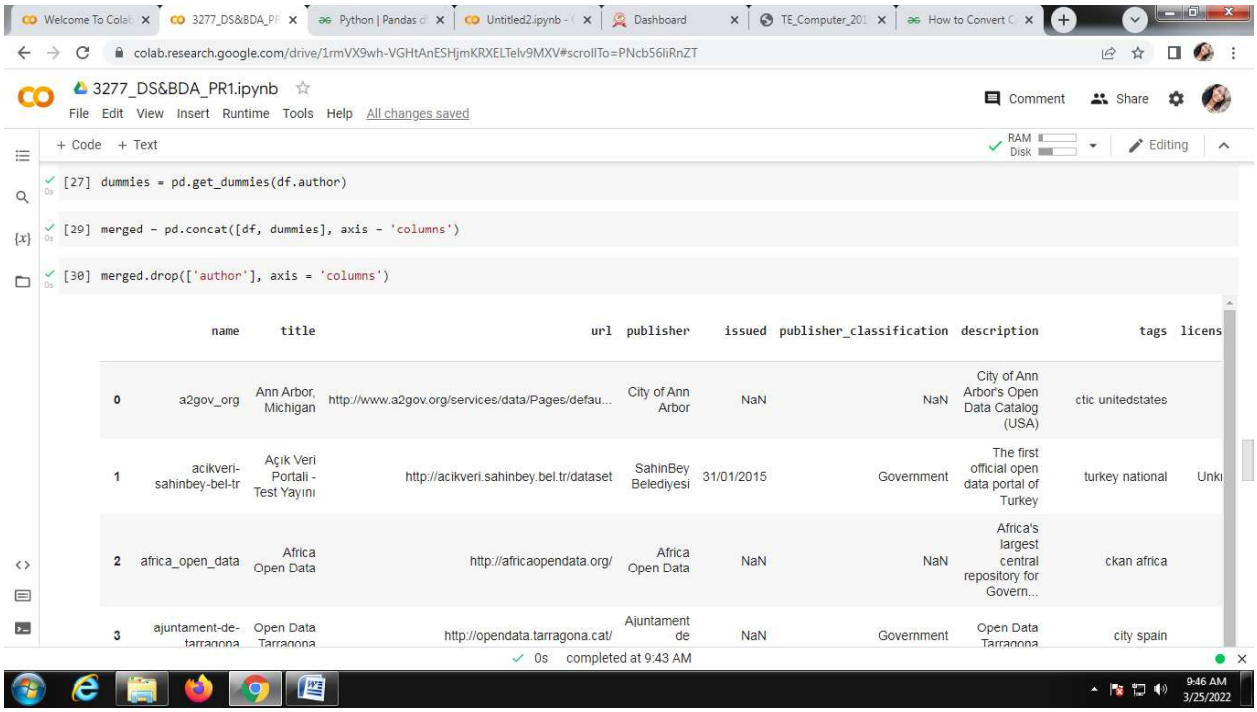
dtypes: to check the data types of colums in a DataFrame.



6. Turn categorical variables into quantitative variables in Python.

get_dummies() : this method will return dummy variable colums.

concat() : to concatenate dummy columns into DataFrames



3277_DS&BDA_PR1.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

RAM Disk Editing

```
[27] dummies = pd.get_dummies(df.author)
```

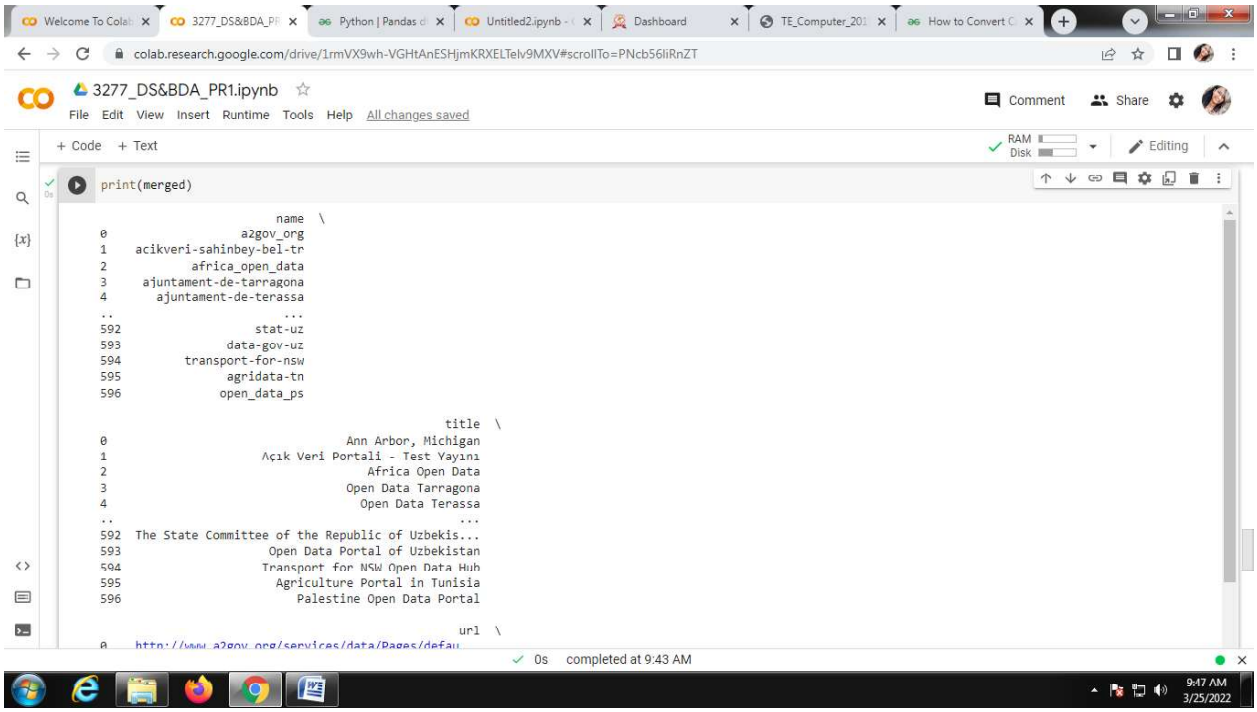
```
[29] merged = pd.concat([df, dummies], axis = 'columns')
```

```
[30] merged.drop(['author'], axis = 'columns')
```

	name	title	url	publisher	issued	publisher_classification	description	tags	licens
0	a2gov_org	Ann Arbor, Michigan	http://www.a2gov.org/services/data/Pages/defau...	City of Ann Arbor	NaN	NaN	City of Ann Arbor's Open Data Catalog (USA)	ctic unitedstates	
1	acikveri-sahinbey-bel-tr	Açık Veri Portali - Test Yayını	http://acikveri.sahinbey.bel.tr/dataset	SahinBey Belediyesi	31/01/2015	Government	The first official open data portal of Turkey	turkey national	Unk
2	africa_open_data	Africa Open Data	http://africaopendata.org/	Africa Open Data	NaN	NaN	Africa's largest central repository for Govern...	ckan africa	
3	ajuntament-de-tarragona	Open Data Tarragona	http://opendata.tarragona.cat/	Ajuntament de	NaN	Government	Open Data Tarragona	city spain	

0s completed at 9:43 AM

9:46 AM 3/25/2022



3277_DS&BDA_PR1.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

RAM Disk Editing

```
print(merged)
```

```
                        name  \
0                   a2gov_org
1      acikveri-sahinbey-bel-tr
2            africa_open_data
3       ajuntament-de-tarragona
4         ajuntament-de-terassa
..                          ...
592                   stat-uz
593               data-gov-uz
594          transport-for-nsw
595                agridata-tn
596               open_data_ps

                                        title  \
0                                Ann Arbor, Michigan
1                     Açık Veri Portali - Test Yayını
2                                 Africa Open Data
3                               Open Data Tarragona
4                                 Open Data Terassa
..                                              ...
592  The State Committee of the Republic of Uzbekis...
593                    Open Data Portal of Uzbekistan
594                  Transport for NSW Open Data Hub
595                    Agriculture Portal in Tunisia
596                       Palestine Open Data Portal

                                        url  \
0  http://www.a2gov.org/services/data/Pages/defau
```

0s completed at 9:43 AM

9:47 AM 3/25/2022

Experiment No. 2

Aim:

Data Wrangling II

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

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

Requirement:

- Anaconda Installer
- Windows 10 OS
- Jupyter Notebook

Theory:

Data Wrangling:

Data wrangling can be defined as the process of cleaning, organizing, and transforming raw data into the desired format for analysts to use for prompt decision-making. Also known as data cleaning or data munging, data wrangling enables businesses to tackle more complex data in less time, produce more accurate results, and make better decisions. The exact methods vary from project to project depending upon your data and the goal you are trying to achieve. More and more organizations are increasingly relying on data wrangling tools to make data ready for downstream analytics.

Benefits of Data Wrangling:

- Data wrangling helps to improve data usability as it converts data into a compatible format for the end system.
- It helps to quickly build data flows within an intuitive user interface and easily schedule and automate the data-flow process.

- Integrates various types of information and their sources (like databases, web services, files, etc.)
- Help users to process very large volumes of data easily and easily share data-flow techniques.

Data Wrangling Vs. ETL:

ETL stands for Extract, Transform and Load. ETL is a middleware process that involves mining or extracting data from various sources, joining the data, transforming data as per business rules, and subsequently loading data to the target systems. ETL is generally used for loading processed data to flat files or relational database tables.

Though Data Wrangling and ETL look similar, there are key differences between data wrangling and ETL processes that set them apart.

- Users – Analysts, statisticians, business users, executives, and managers use data wrangling. In comparison, DW/ETL developers use ETL as an intermediate process linking source systems and reporting layers.
- Data Structure – Data wrangling involves varied and complex data sets, while ETL involves structured or semi-structured relational data sets.
- Use Case – Data wrangling is normally used for exploratory data analysis, but ETL is used for gathering, transforming, and loading data for reporting.

Data Wrangling Tools:

- Spreadsheets / Excel Power Query - It is the most basic manual data wrangling tool
- Tabula – It is a tool suited for all data types
- Google DataPrep – It is a data service that explores, cleans, and prepares data
- Data wrangler – It is a data cleaning and transforming tool

Conclusion:

Hence, we have implemented Data Wrangling Practical II.

In [2]:
```
import os
os.getcwd()
```

Out[2]:
```
'C:\\Users\\rohit\\Downloads'
```

In [3]:
```
import pandas as pd
```

In [4]:
```
df = pd.read_csv('student2.csv')
```

In [5]:
```
df
```

Out[5]:

	roll	name	class	marks	age
0	1	anil	TE	56.77	22.0
1	2	amit	TE	59.77	21.0
2	3	aniket	BE	76.88	19.0
3	4	ajinkya	TE	69.66	NaN
4	5	asha	NaN	63.28	20.0
5	6	ayesha	BE	49.55	20.0
6	7	amar	BE	NaN	19.0
7	8	amita	BE	NaN	23.0
8	9	amol	TE	56.75	20.0
9	10	anmol	BE	78.66	21.0

In [6]:
```
df.shape
```

Out[6]:
```
(10, 5)
```

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

Out[7]:

	roll	name	class	marks	age
0	1	anil	TE	56.77	22.0
1	2	amit	TE	59.77	21.0
2	3	aniket	BE	76.88	19.0
3	4	ajinkya	TE	69.66	NaN
4	5	asha	NaN	63.28	20.0

In [8]:
```
df.tail()
```

Out[8]:

	roll	name	class	marks	age

	roll	name	class	marks	age
5	6	ayesha	BE	49.55	20.0
6	7	amar	BE	NaN	19.0
7	8	amita	BE	NaN	23.0
8	9	amol	TE	56.75	20.0
9	10	anmol	BE	78.66	21.0

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

Out[9]:
```
roll     10
name     10
class     9
marks     8
age       9
dtype: int64
```

In [10]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   roll    10 non-null     int64
 1   name    10 non-null     object
 2   class   9 non-null      object
 3   marks   8 non-null      float64
 4   age     9 non-null      float64
dtypes: float64(2), int64(1), object(2)
memory usage: 528.0+ bytes
```

In [11]:
```
df.isnull()
```

Out[11]:

	roll	name	class	marks	age
0	False	False	False	False	False
1	False	False	False	False	False
2	False	False	False	False	False
3	False	False	False	False	True
4	False	False	True	False	False
5	False	False	False	False	False
6	False	False	False	True	False
7	False	False	False	True	False
8	False	False	False	False	False
9	False	False	False	False	False

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

Out[12]:
```
roll     0
```

```
name     0
class    1
marks    2
age      1
dtype: int64
```

In [13]:
```
print(True+True)
```

```
2
```

In [14]:
```
df.dropna()
#drop all rows that having no value
```

Out[14]:

	roll	name	class	marks	age
0	1	anil	TE	56.77	22.0
1	2	amit	TE	59.77	21.0
2	3	aniket	BE	76.88	19.0
5	6	ayesha	BE	49.55	20.0
8	9	amol	TE	56.75	20.0
9	10	anmol	BE	78.66	21.0

In [15]:
```
df.fillna(0)
#identify misssing value using domain knowldege....fill absed n domain knowlwdge
```

Out[15]:

	roll	name	class	marks	age
0	1	anil	TE	56.77	22.0
1	2	amit	TE	59.77	21.0
2	3	aniket	BE	76.88	19.0
3	4	ajinkya	TE	69.66	0.0
4	5	asha	0	63.28	20.0
5	6	ayesha	BE	49.55	20.0
6	7	amar	BE	0.00	19.0
7	8	amita	BE	0.00	23.0
8	9	amol	TE	56.75	20.0
9	10	anmol	BE	78.66	21.0

In [16]:
```
#only using class column
df['class'].fillna('TE')
```

Out[16]:
```
0    TE
1    TE
2    BE
3    TE
4    TE
5    BE
6    BE
7    BE
8    TE
```

```
9    BE
Name: class, dtype: object
```

In [24]:
```
df['marks'].fillna(df['marks'].mean())
```

Out[24]:
```
0    56.770
1    59.770
2    76.880
3    69.660
4    63.280
5    49.550
6    63.915
7    63.915
8    56.750
9    78.660
Name: marks, dtype: float64
```

In [17]:
```
df['age'].fillna(df['age'].median())
```

Out[17]:
```
0    22.0
1    21.0
2    19.0
3    20.0
4    20.0
5    20.0
6    19.0
7    23.0
8    20.0
9    21.0
Name: age, dtype: float64
```

In [20]:
```
df['class'].value_counts()
```

Out[20]:
```
BE    5
TE    4
Name: class, dtype: int64
```

In [21]:
```
df['class'].fillna(df['class'].mode()[0])
```

Out[21]:
```
0    TE
1    TE
2    BE
3    TE
4    BE
5    BE
6    BE
7    BE
8    TE
9    BE
Name: class, dtype: object
```

In [22]:
```
df.fillna(method='backfill')
```

Out[22]:

	roll	name	class	marks	age
0	1	anil	TE	56.77	22.0
1	2	amit	TE	59.77	21.0
2	3	aniket	BE	76.88	19.0
3	4	ajinkya	TE	69.66	20.0

	roll	name	class	marks	age
4	5	asha	BE	63.28	20.0
5	6	ayesha	BE	49.55	20.0
6	7	amar	BE	56.75	19.0
7	8	amita	BE	56.75	23.0
8	9	amol	TE	56.75	20.0
9	10	anmol	BE	78.66	21.0

In [23]:
```
df.fillna(method='pad')
```

Out[23]:

	roll	name	class	marks	age
0	1	anil	TE	56.77	22.0
1	2	amit	TE	59.77	21.0
2	3	aniket	BE	76.88	19.0
3	4	ajinkya	TE	69.66	19.0
4	5	asha	TE	63.28	20.0
5	6	ayesha	BE	49.55	20.0
6	7	amar	BE	49.55	19.0
7	8	amita	BE	49.55	23.0
8	9	amol	TE	56.75	20.0
9	10	anmol	BE	78.66	21.0

In [25]:
```
df.describe()
```

Out[25]:

	roll	marks	age
count	10.00000	8.000000	9.000000
mean	5.50000	63.915000	20.555556
std	3.02765	10.315294	1.333333
min	1.00000	49.550000	19.000000
25%	3.25000	56.765000	20.000000
50%	5.50000	61.525000	20.000000
75%	7.75000	71.465000	21.000000
max	10.00000	78.660000	23.000000

In [82]:
```
import numpy as np
x = np.array([5,4,3,2,7,8,98,28])
```

In [83]:
```
np.mean(x)
```

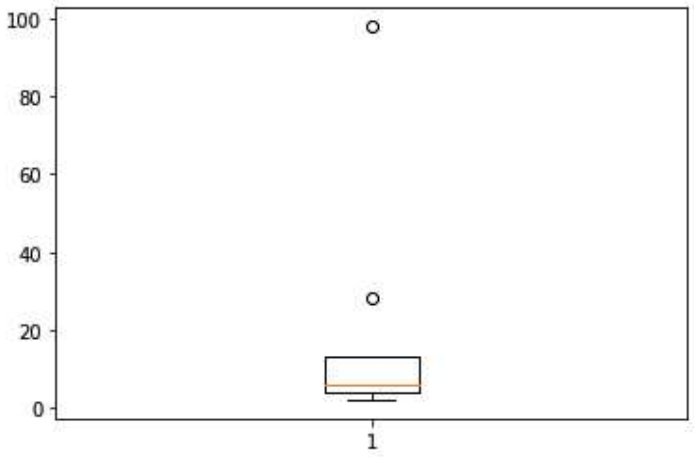
Out[83]: 19.375

In [84]:
```
np.median(x)
```

Out[84]: 6.0

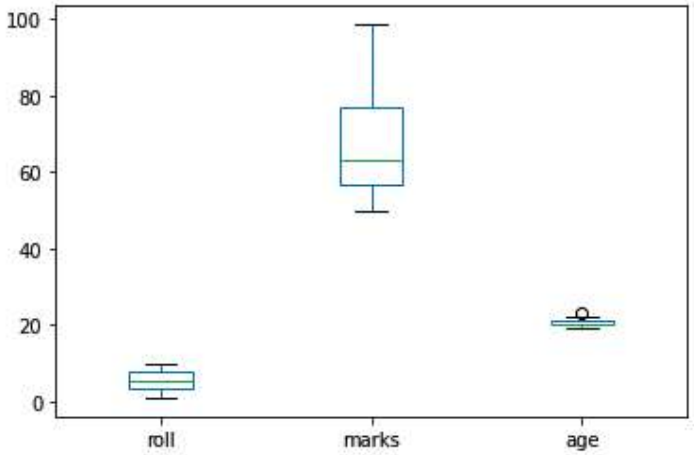
In [85]:
```
import matplotlib.pyplot as plt
```

In [86]:
```
plt.boxplot(x);
```



In [87]:
```
df.plot.box()
```

Out[87]: <AxesSubplot:>



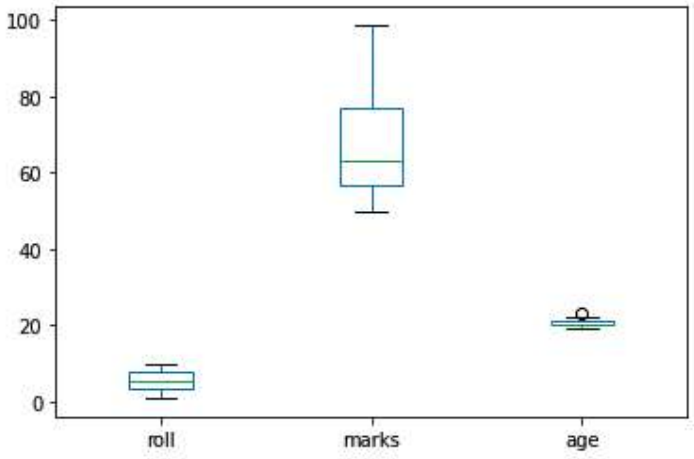
In [88]:
```
df.loc[6,'marks']
```

Out[88]: 98.45

In [89]:
```
df.loc[6,'marks']=98.45
```

In [90]:
```
df.plot.box()
```

Out[90]: <AxesSubplot:>

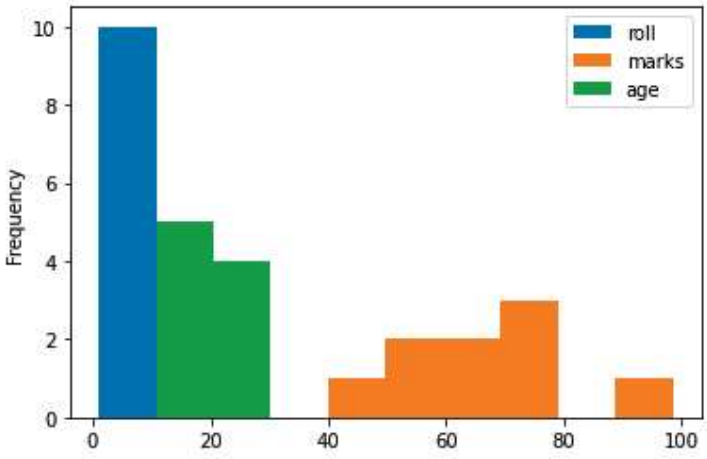


In [91]:
```
df.loc[6,'marks']
```

Out[91]: 98.45

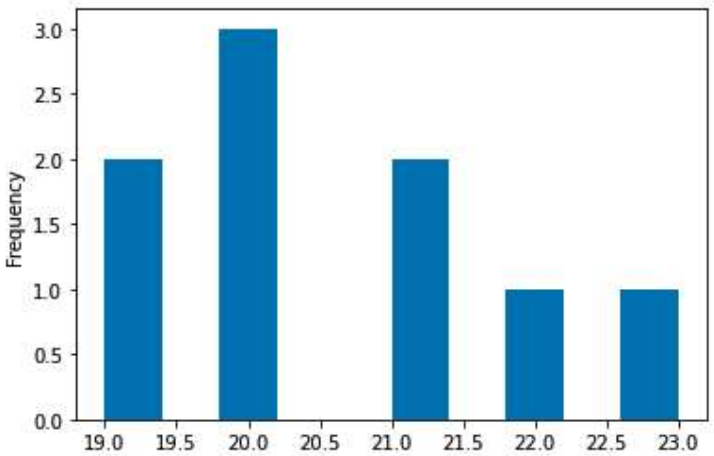
In [92]:
```
df.plot.hist()
```

Out[92]: <AxesSubplot:ylabel='Frequency'>



In [93]:
```
df['age'].plot.hist()
```

Out[93]: <AxesSubplot:ylabel='Frequency'>



In [97]:
```
x= df[['age','marks']]
```

In [98]:
```
x.describe()
```

Out[98]:

	age	marks
count	9.000000	9.000000
mean	20.555556	67.752222
std	1.333333	15.020755
min	19.000000	49.550000
25%	20.000000	56.770000
50%	20.000000	63.280000
75%	21.000000	76.880000
max	23.000000	98.450000

In [103…
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
x_scaled = scaler.fit_transform(x)
```

In [106…
```
pd.DataFrame(x_scaled).describe()
```

Out[106…

	0	1
count	9.000000	9.000000
mean	0.388889	0.372234
std	0.333333	0.307173
min	0.000000	0.000000
25%	0.250000	0.147648
50%	0.250000	0.280777
75%	0.500000	0.558896
max	1.000000	1.000000

In [107…
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)
```

In [110…
```
pd.DataFrame(x_scaled).describe()
```

Out[110…

	0	1
count	9.000000e+00	9.000000e+00
mean	-1.289092e-15	4.317534e-16
std	1.060660e+00	1.060660e+00

	0	1
min	-1.237437e+00	-1.285313e+00
25%	-4.419417e-01	-7.754873e-01
50%	-4.419417e-01	-3.157969e-01
75%	3.535534e-01	6.445395e-01
max	1.944544e+00	2.167661e+00

In [ ]:

In [ ]:

Experiment No. 3

Aim:

1) Provide summery statistics for a dataset with numeric varibles grouped by one of the qualitative variables.

2) write a python program to display some basic statistical details like percentile, mean, standard devivation , etc.

Requirement:

- Anaconda Installer
- Windows 10 OS
- Jupyter Notebook

Theory:

Step 1: Provide summery statistics such as mean, median, mode, standard deviation.

(1) Mean:

“Average” value is termed as mean of the dataset

means= sum of all data values / Total number of Data Values

(2) Median:

The middle values of sorted dataset is known as median.

(3) Mode:

mode refers to most frequently occuring values in the dataset.

e.g. Consider the weight (in kg) of 5children as 36,40,32,42,30.lets compute mean, median,mode

(1) Mean=(36+40+32+42+30) / 5

= 36 kg

(2) median: arrange the data in ascending order : 30,22,36,40,42

the middle value is 36. so median is 36 kg.

(3) mode: 36 kg occurs most number of times so mode=36 kg

Calculate mean using python:

```
df = pd.Dataframe(dict)
mean=df.['score'].mean()
print(mean)
```

Calculate median using python:

```
df = pd.Dataframe(dict)
mode=df.mode()
print(mode)
```

Calculating maximum and minimum python:

```
df=pd.Dataframe[[10,20,30,40],[7,14,21,28],[55,15,8,12],[15,14,1,8],[7,1,1,8],[5,4,9,2]]
coloumns=[‘Apple’, ‘Orange’, ‘Banana’ , ‘Peer’]
index=[‘Basket1’ , ‘Basket2’ , ‘Basket3’ , ‘Basket4’ , ‘ Basket5’ , ‘ Basket6’]

minimum=df.[[‘Apple’ , ‘Orange’ , ‘ Banana’, ‘Peer’]].min();
maximum=df.[[‘Apple’ , ‘Orange’ , ‘ Banana’, ‘Peer’]].max();

print(minimum);
print(maximum);
```

Standard Deviation:

The standard deviation is a statistic that measures the dispersion of a dataset relative to its mean and is calculated as the square root of the variance. The standard deviation is calculated as the square root of variance by determining each data point's deviation relative to the mean.

The standard deviation is a statistic that measures the dispersion of a dataset relative to its mean and is calculated as the square root of the variance. The standard deviation is calculated as the square root of variance by determining each data point's deviation relative to the mean.

Key Takeways:

- It is calculated as the square root of the variance.
- Standard deviation, in finance, is often used as a measure of a relative riskiness of an asset.
- A volatile stock has a high standard deviation, while the deviation of a stable blue-chip stock is usually rather low.
- As a downside, the standard deviation calculates all uncertainty as risk, even when it’s in the investor's favor—such as above-average returns.

$$\text{Standard Deviation} = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \overline{x})^2}{n-1}}$$

where:

x_i = Value of the i^{th} point in the data set

$\overline{x}$ = The mean value of the data set

n = The number of data points in the data

Central Tendency Measure	Pros	Cons
Mean	Sensitive as it takes all data values into account(reliable)	Biased output if outliers/extreme values exist in the data set
Median	Not affected by extreme values	-Less sensitive than Mean as it only focusses on giving out the middle data point irrespective of how far the other values are from the middle -Needs the data to be arranged in the ascending order before computing
Mode	Not affected by extreme values and can be used with non-numerical data	There may be more than one mode or no mode at all and it may not reflect data summary accurately

Libraries Used:

1. Pandas: Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring and manipulating data.
2. Numpy: is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed.
3. Seaborn: Seaborn is a data visualization library built on top of matplotlib and closely integrated with pandas data structures in Python.

Conclusion:

From this experiment we learnt how to calculate mean, median and mode.

In [2]:
```
import pandas as pd
```

In [5]:
```
df= pd.read_csv('student.csv')
```

In [6]:
```
df
```

Out[6]:

	roll	name	class	marks	age
0	1	anil	TE	56.77	22
1	2	amit	TE	59.77	21
2	3	aniket	BE	76.88	19
3	4	ajinkya	TE	69.66	20
4	5	asha	TE	63.28	20
5	6	ayesha	BE	49.55	20
6	7	amar	BE	65.34	19
7	8	amita	BE	68.33	23
8	9	amol	TE	56.75	20
9	10	anmol	BE	78.66	21

In [7]:
```
df.mean()
#mean
```

Out[7]:
```
roll      5.500
marks    64.499
age      20.500
dtype: float64
```

In [8]:
```
df.median()
#median
```

Out[8]:
```
roll      5.50
marks    64.31
age      20.00
dtype: float64
```

In [9]:
```
#standarad deviation
df.std()
```

Out[9]:
```
roll     3.027650
marks    9.207179
age      1.269296
dtype: float64
```

In [10]:
```
df.min()
#min
```

Out[10]:
```
roll           1
name     ajinkya
class         BE
```

```
marks      49.55
age           19
dtype: object
```

In [17]:
```
df.max()
#max
```

Out[17]:
```
roll         10
name     ayesha
class        TE
marks     78.66
age          23
dtype: object
```

In [12]:
```
import numpy as np
```

In [15]:
```
np.std(df['marks'])
```

Out[15]:
```
8.734696846485285
```

In [18]:
```
gr1 = df.groupby('class')
```

In [21]:
```
te = gr1.get_group('TE')
```

In [22]:
```
te.min()
```

Out[22]:
```
roll           1
name     ajinkya
class         TE
marks      56.75
age           20
dtype: object
```

In [23]:
```
te.max()
```

Out[23]:
```
roll         9
name      asha
class       TE
marks    69.66
age         22
dtype: object
```

In [24]:
```
gr2 = df.groupby('age')
```

In [25]:
```
gr2.groups
```

Out[25]:
```
{19: [2, 6], 20: [3, 4, 5, 8], 21: [1, 9], 22: [0], 23: [7]}
```

In [26]:
```
tw = gr2.get_group(20)
```

In [27]:
```
tw
```

Out[27]:

	roll	name	class	marks	age
3	4	ajinkya	TE	69.66	20
4	5	asha	TE	63.28	20
5	6	ayesha	BE	49.55	20
8	9	amol	TE	56.75	20

In [28]:
```
import seaborn as sns
```

In [29]:
```
df = sns.load_dataset('iris')
```

In [30]:
```
df
```

Out[30]:

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

150 rows × 5 columns

In [33]:
```
df = px.data.iris()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-33-dcabb90b9f03> in <module>
----> 1 df = px.data.iris()

NameError: name 'px' is not defined
```

In [34]:
```
#https://mitu.co.in


df=pd.read_csv('iris.csv')
df
df.describe()
```

Out[34]:

	sepal_length	sepal_width	petal_length	petal_width

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

In [35]:
```
gr = df.groupby('species')
```

In [37]:
```
se = gr.get_group('setosa')
ve = gr.get_group('versicolor')
vi = gr.get_group('virginica')
```

In [38]:
```
se.shape
```

Out[38]: (50, 5)

In [39]:
```
ve.shape
```

Out[39]: (50, 5)

In [40]:
```
vi.shape
```

Out[40]: (50, 5)

In [42]:
```
se.describe()
```

Out[42]:

	sepal_length	sepal_width	petal_length	petal_width
count	50.00000	50.000000	50.000000	50.000000
mean	5.00600	3.428000	1.462000	0.246000
std	0.35249	0.379064	0.173664	0.105386
min	4.30000	2.300000	1.000000	0.100000
25%	4.80000	3.200000	1.400000	0.200000
50%	5.00000	3.400000	1.500000	0.200000
75%	5.20000	3.675000	1.575000	0.300000
max	5.80000	4.400000	1.900000	0.600000

In [43]:
```
ve.describe()
```

Out[43]:

	sepal_length	sepal_width	petal_length	petal_width
count	50.000000	50.000000	50.000000	50.000000
mean	5.936000	2.770000	4.260000	1.326000
std	0.516171	0.313798	0.469911	0.197753
min	4.900000	2.000000	3.000000	1.000000
25%	5.600000	2.525000	4.000000	1.200000
50%	5.900000	2.800000	4.350000	1.300000
75%	6.300000	3.000000	4.600000	1.500000
max	7.000000	3.400000	5.100000	1.800000

In [44]:

```
vi.describe()
```

Out[44]:

	sepal_length	sepal_width	petal_length	petal_width
count	50.00000	50.000000	50.000000	50.00000
mean	6.58800	2.974000	5.552000	2.02600
std	0.63588	0.322497	0.551895	0.27465
min	4.90000	2.200000	4.500000	1.40000
25%	6.22500	2.800000	5.100000	1.80000
50%	6.50000	3.000000	5.550000	2.00000
75%	6.90000	3.175000	5.875000	2.30000
max	7.90000	3.800000	6.900000	2.50000

In [ ]:

```

```

Experiment No. 4

Aim: Create a Linear Regression model using python to predict home prise using Boston Housing dataset.

Requirement:

- Anaconda Installer
- Windows 10 OS
- Jupyter Notebook

Theory:

Linear Regression in data science:

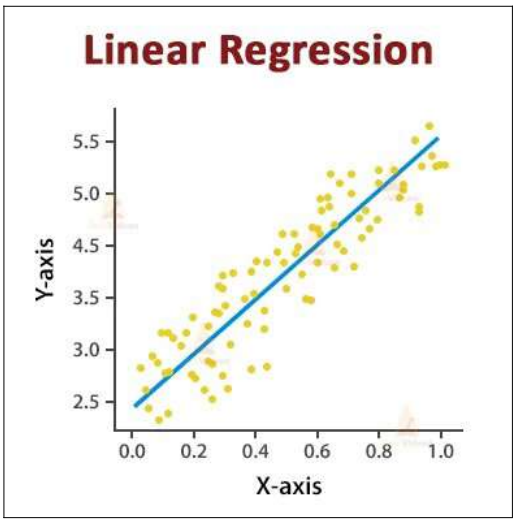


fig. Linear Regression

The term regression is used when you try to find the relationship between variables. In Machine Learning and in statistical modeling, that relationship is used to predict the outcome of events.

Simple Linear Regression:

Simple linear regression is an approach for predicting a response using a single feature. It is assumed that the two variables are linearly related. Hence, we try to find a linear function that predicts the response value(y) as accurately as possible as a function of the feature or independent variable(x).

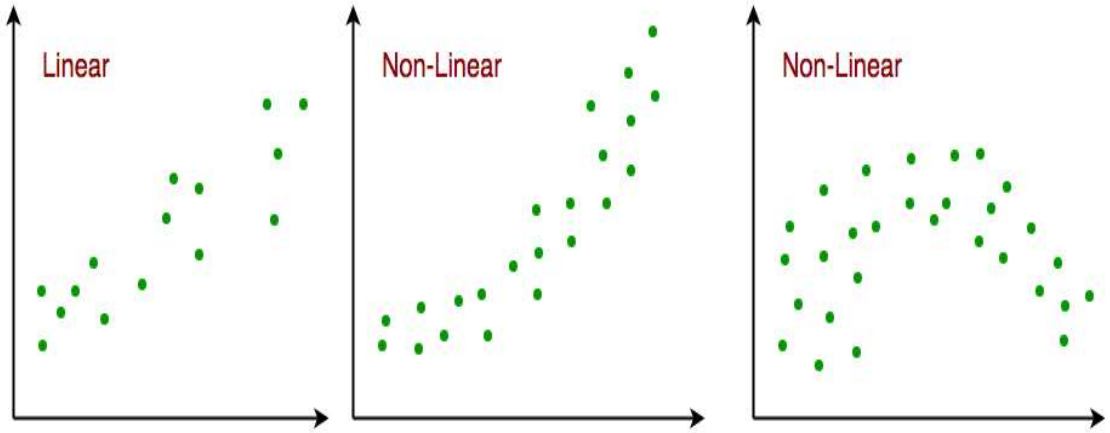
Multiple linear regression:

Multiple linear regression attempts to model the relationship between two or more features and a response by fitting a linear equation to the observed data. Clearly, it is nothing but an extension of simple linear regression.

Assumptions:

Given below are the basic assumptions that a linear regression model makes regarding a dataset on which it is applied:

• Linear relationship: Relationship between response and feature variables should be linear. The linearity assumption can be tested using scatter plots. As shown below, 1st figure represents linearly related variables whereas variables in the 2nd and 3rd figures are most likely non-linear. So, 1st figure will give better predictions using linear regression.



• Little or no multi-collinearity: It is assumed that there is little or no multicollinearity in the data. Multicollinearity occurs when the features (or independent variables) are not independent of each other.

• Little or no auto-correlation: Another assumption is that there is little or no autocorrelation in the data. Autocorrelation occurs when the residual errors are not independent of each other. You can refer here for more insight into this topic.

•Homoscedasticity: Homoscedasticity describes a situation in which the error term (that is, the “noise” or random disturbance in the relationship between the independent variables and the dependent variable) is the same across all values of the independent variables. As shown below, figure 1 has homoscedasticity while figure 2 has heteroscedasticity.

Applications:

1. Trend lines: A trend line represents the variation in quantitative data with the passage of time (like GDP, oil prices, etc.). These trends usually follow a linear relationship. Hence, linear regression can be applied to predict future values. However, this method suffers from a lack of scientific validity in cases where other potential changes can affect the data.
2. Economics: Linear regression is the predominant empirical tool in economics. For example, it is used to predict consumer spending, fixed investment spending, inventory investment, purchases of a country's exports, spending on imports, the demand to hold liquid assets, labor demand, and labor supply.
3. Finance: The capital price asset model uses linear regression to analyze and quantify the systematic risks of an investment.
   Biology: Linear regression is used to model causal relationships between parameters in biological systems.

Dataset used:

- In this experminet we are going to use the boston housing dataset which contain information about various houses in boston through different parameters.
- There are total 506 samples ad 14 features (columns) in this dataset.
- Our objective is to predict the value of prices of the house using features with the help of linear regression.

Libraries Uesd:

1. Pandas: Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring and manipulating data.
2. Sklearn: It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python.

Conclusion:

In this experiment we have studied abbout linear regression and done house prise prediction using boston housing dataset.

In [1]:
```
import pandas as pd
```

In [2]:
```
df=pd.read_csv('BostonHousing.csv')
```

In [3]:
```
df
```

Out[3]:

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	b	lstat	medv
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	273	21.0	391.99	9.67	22.4
502	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	273	21.0	396.90	9.08	20.6
503	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273	21.0	396.90	5.64	23.9
504	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273	21.0	393.45	6.48	22.0
505	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	273	21.0	396.90	7.88	11.9

506 rows × 14 columns

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

Out[4]:

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	b	lstat	medv
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

In [6]:
```
#import data
x=df.drop('medv',axis = 1)

#output data
y=df['medv']
```

In [7]:
```
x.shape
```

Out[7]:
```
(506, 13)
```

In [9]:
```

```

```
from sklearn.model_selection import train_test_split
```

In [10]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=0,test_size=0.25)
```

In [11]:
```
x_train
```

Out[11]:

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	b	lstat
245	0.19133	22.0	5.86	0	0.431	5.605	70.2	7.9549	7	330	19.1	389.13	18.46
59	0.10328	25.0	5.13	0	0.453	5.927	47.2	6.9320	8	284	19.7	396.90	9.22
276	0.10469	40.0	6.41	1	0.447	7.267	49.0	4.7872	4	254	17.6	389.25	6.05
395	8.71675	0.0	18.10	0	0.693	6.471	98.8	1.7257	24	666	20.2	391.98	17.12
416	10.83420	0.0	18.10	0	0.679	6.782	90.8	1.8195	24	666	20.2	21.57	25.79
...	...	...	...	...	...	...	...	...	...	...	...	...	...
323	0.28392	0.0	7.38	0	0.493	5.708	74.3	4.7211	5	287	19.6	391.13	11.74
192	0.08664	45.0	3.44	0	0.437	7.178	26.3	6.4798	5	398	15.2	390.49	2.87
117	0.15098	0.0	10.01	0	0.547	6.021	82.6	2.7474	6	432	17.8	394.51	10.30
47	0.22927	0.0	6.91	0	0.448	6.030	85.5	5.6894	3	233	17.9	392.74	18.80
172	0.13914	0.0	4.05	0	0.510	5.572	88.5	2.5961	5	296	16.6	396.90	14.69

379 rows × 13 columns

In [12]:
```
x_train.head()
```

Out[12]:

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	b	lstat
245	0.19133	22.0	5.86	0	0.431	5.605	70.2	7.9549	7	330	19.1	389.13	18.46
59	0.10328	25.0	5.13	0	0.453	5.927	47.2	6.9320	8	284	19.7	396.90	9.22
276	0.10469	40.0	6.41	1	0.447	7.267	49.0	4.7872	4	254	17.6	389.25	6.05
395	8.71675	0.0	18.10	0	0.693	6.471	98.8	1.7257	24	666	20.2	391.98	17.12
416	10.83420	0.0	18.10	0	0.679	6.782	90.8	1.8195	24	666	20.2	21.57	25.79

In [14]:
```
x_train.shape
```

Out[14]: (379, 13)

In [15]:
```
x_test.shape
```

Out[15]: (127, 13)

In [16]:
```
from sklearn.linear_model import LinearRegression
#import the class
```

In [17]:
```
# Create the object
regressor=LinearRegression()
```

In [18]:
```
# Train the Algorithm
regressor.fit(x_train,y_train)
```

Out[18]:
```
LinearRegression()
```

In [20]:
```
regressor.coef_
```

Out[20]:
```
array([-1.17735289e-01,  4.40174969e-02, -5.76814314e-03,  2.39341594e+00,
       -1.55894211e+01,  3.76896770e+00, -7.03517828e-03, -1.43495641e+00,
        2.40081086e-01, -1.12972810e-02, -9.85546732e-01,  8.44443453e-03,
       -4.99116797e-01])
```

In [21]:
```
regressor.intercept_
```

Out[21]:
```
36.93325545711978
```

In [22]:
```
#predictions
y_pred=regressor.predict(x_test)
```

In [23]:
```
y_pred.shape
```

Out[23]:
```
(127,)
```

In [24]:
```
result=pd.DataFrame({'Actual':y_test,'Producted':y_pred})
```

In [25]:
```
result
```

Out[25]:

	Actual	Producted
329	22.6	24.952333
371	50.0	23.616997
219	23.0	29.205886
403	8.3	11.960705
78	21.2	21.333620
...	...	...
49	19.4	17.538048
498	21.2	21.502223
309	20.3	23.632813
124	18.8	20.282598
306	33.4	35.179734

127 rows × 2 columns

In [26]:
```
residual_error=abs(y_test-y_pred)
```

In [27]:
```
residual_error
```

Out[27]:
```
329     2.352333
371    26.383003
219     6.205886
403     3.660705
78      0.133620
          ...
49      1.861952
498     0.302223
309     3.332813
124     1.482598
306     1.779734
Name: medv, Length: 127, dtype: float64
```

In [28]:
```
residual_error;
```

In [29]:
```
# mean absolute error
sum(residual_error)/len(residual_error)
```

Out[29]:
```
3.668330148135727
```

In [30]:
```
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y_pred,y_test)
```

Out[30]:
```
3.668330148135727
```

In [32]:
```
from sklearn.metrics import mean_absolute_percentage_error
```

In [34]:
```
mean_absolute_percentage_error(y_test,y_pred)
```

Out[34]:
```
0.1754993780061577
```

In [36]:
```
regressor.score(x_test,y_test)
```

Out[36]:
```
0.6354638433202124
```

In [37]:
```
# !pip install scikit-learn-U
#for the installation of Scikit
```

In [41]:
```
from sklearn.metrics import r2_score
r2_score(y_test,y_pred)
```

Out[41]:
```
0.6354638433202124
```

In [42]:
```
new=[[0.7258,0,8.64,0,0.538,5.727,69.6,3.7965,4,307,22,391.95,11.28]]
```

In [43]:
```
new
```

Out[43]:
```
[[0.7258, 0, 8.64, 0, 0.538, 5.727, 69.6, 3.7965, 4, 307, 22, 391.95, 11.28]]
```

In [44]:
```
regressor.predict(new)
```

Out[44]:
```
array([17.54806506])
```

In [ ]:

In [ ]:

Experiment No. 5

Aim: Implement logistic regression using python to perform classifiaction on social network_ads , cv dataset.

Requirement:

- Anaconda Installer
- Windows 10 OS
- Jupyter Notebook

Theory:

Logistic Regression?

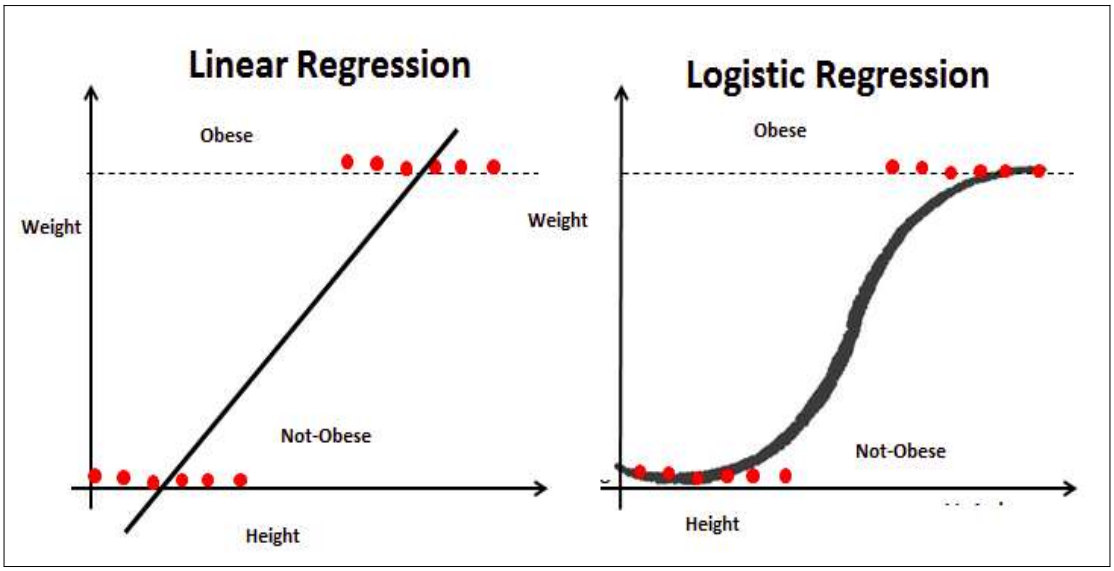


Fig. Linear Regression Vs Logistic Regression

- Logistic regression is a supervised learning algorithm used to predict a dependent categorical target variable. In essence, if you have a large set of data that you want to categorize, logistic regression may be able to help.
- For example, if you were given a dog and an orange and you wanted to find out whether each of these items was an animal or not, the desired result would be for the dog to end up classified as an animal, and for the orange to be categorized as not an animal.
- Animal is your target; it is dependent on your data in order to be able to classify the item correctly. In this example, there are only two possible answers (binary logistic regression), animal or not an animal. However, it is also possible to set up your logistic regression with more than two possible categories (multinomial logistic regression).
- To dive a little deeper into how your model might attempt to classify these two items directly, let's consider what else the model would need to know about the items in order to decide where they belong. Other similar aspects of these items

would need to be looked at when considering how to classify each item or data point. Aspects, or features, may include color, size, weight, shape, height, volume or amount of limbs.

- In this way, knowing that an orange's shape was a circle may help the algorithm to conclude that the orange was not an animal. Similarly, knowing that the orange had zero limbs would help as well.
- Logistic regression requires that the dependent variable, in this case whether the item was an animal or not, be categorical. The outcome is either animal or not an animal—there is no range in between.
- A problem that has a continuous outcome, such as predicting the grade of a student or the fuel tank range of a car, is not a good candidate to use logistic regression. Other options like linear regression may be more appropriate.

Types of Logistic Regression:

There are three main types of logistic regression:

1) binary
2) multinomial
3) ordinal.

They differ in execution and theory. Binary regression deals with two possible values, essentially: yes or no. Multinomial logistic regression deals with three or more values. And ordinal logistic regression deals with three or more classes in a predetermined order.

1. Binary logistic regression:

Binary logistic regression was mentioned earlier in the case of classifying an object as an animal or not an animal—it's an either/or solution. There are just two possible outcome answers. This concept is typically represented as a 0 or a 1 in coding.

Examples include:

• Whether or not to lend to a bank customer (outcomes are yes or no).
• Assessing cancer risk (outcomes are high or low).
• Will a team win tomorrow's game (outcomes are yes or no).

2. Multinomial logistic regression:

Multinomial logistic regression is a model where there are multiple classes that an item can be classified as. There is a set of three or more predefined classes set up prior to running the model.

Examples include:
• Classifying texts into what language they come from.
• Predicting whether a student will go to college, trade school or into the workforce.
• Does your cat prefer wet food, dry food or human food?

3. Ordinal logistic regression:

Ordinal logistic regression is also a model where there are multiple classes that an item can be classified as; however, in this case an ordering of classes is required. Classes do not need to be proportionate. The distance between each class can vary. Examples include:
• Ranking restaurants on a scale of 0 to 5 stars.
• Predicting the podium results of an Olympic event.
• Assessing a choice of candidates, specifically in places that institute ranked-choice voting.

4. Logistic regression assumptions:

• Remove highly correlated inputs.
• Consider removing outliers in your training set because logistic regression will not give significant weight to them during its calculations.
• Does not favor sparse (consisting of a lot of zero values) data.
• Logistic regression is a classification model, unlike linear regression.

Libraries Used:

1. Pandas: Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring and manipulating data.

2. Sklearn: It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python.

3. Seaborn: Seaborn is a data visualization library built on top of matplotlib and closely integrated with pandas data structures in Python.

4. Matplotlib: Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible.

Conclusion:

In this experiment we have studied about the logistic regression model. We ahve performed the classification on the social network. Ads dataset using various liatories of python.

In [1]:
```
import pandas as pd
```

[3]:
```
df = pd.read_csv('Social_Network_Ads.csv')
```

In [4]:
```
df
```

Out[4]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19.0	19000.0	0
1	15810944	Male	35.0	20000.0	0
2	15668575	Female	26.0	43000.0	0
3	15603246	Female	27.0	57000.0	0
4	15804002	Male	19.0	76000.0	0
...	...	...	...	...	...
395	15691863	Female	46.0	41000.0	1
396	15706071	Male	51.0	23000.0	1
397	15654296	Female	50.0	20000.0	1
398	15755018	Male	36.0	33000.0	0
399	15594041	Female	49.0	36000.0	1

400 rows × 5 columns

In [44]:
```
#input data
x=df[['Age','EstimatedSalary']]

#output data
y=df['Purchased']
```

In [82]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
x_scaled = scaler.fit_transform(x)
```

In [62]:
```
#cross. validation

from sklearn.model_selection import train_test_split
```

In [83]:
```
x_train, x_test, y_train, y_test = train_test_split(x_scaled,y,random_state=0,test_s
```

In [84]:
```
x_train
```

Out[84]:
```
array([[0.61904762, 0.17777778],
       [0.33333333, 0.77777778],
       [0.47619048, 0.25925926],
       [0.33333333, 0.88888889],
       [0.80952381, 0.04444444],
```

```
[0.83333333, 0.65925926],
[0.5       , 0.2       ],
[0.47619048, 0.34074074],
[0.42857143, 0.25925926],
[0.42857143, 0.35555556],
[0.4047619 , 0.07407407],
[0.4047619 , 0.25925926],
[0.57142857, 0.42962963],
[0.69047619, 0.25185185],
[0.97619048, 0.1037037 ],
[0.73809524, 0.37037037],
[0.64285714, 0.85925926],
[0.30952381, 0.54814815],
[0.66666667, 0.4962963 ],
[0.69047619, 0.26666667],
[0.19047619, 0.        ],
[1.        , 0.64444444],
[0.47619048, 0.71851852],
[0.52380952, 0.68148148],
[0.57142857, 0.28148148],
[0.4047619 , 0.32592593],
[0.71428571, 0.19259259],
[0.71428571, 0.88148148],
[0.47619048, 0.72592593],
[0.26190476, 0.98518519],
[0.19047619, 0.        ],
[1.        , 0.2       ],
[0.14285714, 0.02962963],
[0.57142857, 0.99259259],
[0.66666667, 0.6       ],
[0.23809524, 0.32592593],
[0.5       , 0.6       ],
[0.23809524, 0.54814815],
[0.54761905, 0.42222222],
[0.64285714, 0.08148148],
[0.35714286, 0.4       ],
[0.04761905, 0.4962963 ],
[0.30952381, 0.43703704],
[0.57142857, 0.48148148],
[0.4047619 , 0.42222222],
[0.35714286, 0.99259259],
[0.52380952, 0.41481481],
[0.78571429, 0.97037037],
[0.66666667, 0.47407407],
[0.4047619 , 0.44444444],
[0.47619048, 0.26666667],
[0.42857143, 0.44444444],
[0.45238095, 0.46666667],
[0.47619048, 0.34074074],
[1.        , 0.68888889],
[0.04761905, 0.4962963 ],
[0.92857143, 0.43703704],
[0.57142857, 0.37037037],
[0.19047619, 0.48148148],
[0.66666667, 0.75555556],
[0.4047619 , 0.34074074],
[0.07142857, 0.39259259],
[0.23809524, 0.21481481],
[0.54761905, 0.53333333],
[0.45238095, 0.13333333],
[0.21428571, 0.55555556],
[0.5       , 0.2       ],
[0.23809524, 0.8       ],
[0.30952381, 0.76296296],
[0.16666667, 0.53333333],
[0.4047619 , 0.41481481],
[0.45238095, 0.40740741],
[0.4047619 , 0.17777778],
[0.69047619, 0.05925926],
```

```
[0.4047619 , 0.97777778],
[0.71428571, 0.91111111],
[0.19047619, 0.52592593],
[0.16666667, 0.47407407],
[0.80952381, 0.91111111],
[0.78571429, 0.05925926],
[0.4047619 , 0.33333333],
[0.35714286, 0.72592593],
[0.28571429, 0.68148148],
[0.71428571, 0.13333333],
[0.54761905, 0.48148148],
[0.71428571, 0.6       ],
[0.30952381, 0.02222222],
[0.30952381, 0.41481481],
[0.5952381 , 0.84444444],
[0.97619048, 0.45185185],
[0.        , 0.21481481],
[0.42857143, 0.76296296],
[0.57142857, 0.55555556],
[0.69047619, 0.11111111],
[0.19047619, 0.20740741],
[0.52380952, 0.46666667],
[0.66666667, 0.32592593],
[0.97619048, 0.2       ],
[0.66666667, 0.43703704],
[0.4047619 , 0.56296296],
[0.23809524, 0.32592593],
[0.52380952, 0.31111111],
[0.97619048, 0.94814815],
[0.92857143, 0.08148148],
[0.80952381, 0.17037037],
[0.69047619, 0.72592593],
[0.83333333, 0.94814815],
[0.4047619 , 0.08888889],
[0.95238095, 0.63703704],
[0.64285714, 0.22222222],
[0.11904762, 0.4962963 ],
[0.66666667, 0.05925926],
[0.57142857, 0.37037037],
[0.23809524, 0.51111111],
[0.47619048, 0.32592593],
[0.19047619, 0.51111111],
[0.26190476, 0.0962963 ],
[0.45238095, 0.41481481],
[0.0952381 , 0.2962963 ],
[0.71428571, 0.14814815],
[0.73809524, 0.0962963 ],
[0.47619048, 0.37037037],
[0.21428571, 0.01481481],
[0.66666667, 0.0962963 ],
[0.71428571, 0.93333333],
[0.19047619, 0.01481481],
[0.4047619 , 0.60740741],
[0.5       , 0.32592593],
[0.14285714, 0.08888889],
[0.33333333, 0.02222222],
[0.66666667, 0.54074074],
[0.4047619 , 0.31851852],
[0.9047619 , 0.33333333],
[0.69047619, 0.14074074],
[0.52380952, 0.42222222],
[0.33333333, 0.62962963],
[0.02380952, 0.04444444],
[0.16666667, 0.55555556],
[0.4047619 , 0.54074074],
[0.23809524, 0.12592593],
[0.76190476, 0.03703704],
[0.52380952, 0.32592593],
[0.76190476, 0.21481481],
```

```
[0.4047619 , 0.42222222],
[0.52380952, 0.94074074],
[0.66666667, 0.12592593],
[0.5       , 0.41481481],
[0.04761905, 0.43703704],
[0.26190476, 0.44444444],
[0.30952381, 0.45185185],
[0.69047619, 0.07407407],
[0.52380952, 0.34074074],
[0.38095238, 0.71851852],
[0.47619048, 0.48148148],
[0.57142857, 0.44444444],
[0.69047619, 0.23703704],
[0.5       , 0.44444444],
[0.02380952, 0.07407407],
[0.45238095, 0.48148148],
[0.42857143, 0.33333333],
[0.54761905, 0.27407407],
[0.42857143, 0.81481481],
[0.71428571, 0.1037037 ],
[0.42857143, 0.82222222],
[0.78571429, 0.88148148],
[0.21428571, 0.31111111],
[0.47619048, 0.41481481],
[0.5       , 0.34074074],
[0.0952381 , 0.08888889],
[0.35714286, 0.33333333],
[0.71428571, 0.43703704],
[0.95238095, 0.05925926],
[0.83333333, 0.42222222],
[0.33333333, 0.75555556],
[0.85714286, 0.40740741],
[0.28571429, 0.48148148],
[0.95238095, 0.59259259],
[0.19047619, 0.27407407],
[0.64285714, 0.47407407],
[0.14285714, 0.2962963 ],
[0.52380952, 0.44444444],
[0.35714286, 0.0962963 ],
[0.61904762, 0.91851852],
[0.0952381 , 0.02222222],
[0.35714286, 0.26666667],
[0.5952381 , 0.87407407],
[0.14285714, 0.12592593],
[0.66666667, 0.05185185],
[0.4047619 , 0.2962963 ],
[0.85714286, 0.65925926],
[0.71428571, 0.77037037],
[0.4047619 , 0.28148148],
[0.45238095, 0.95555556],
[0.11904762, 0.37777778],
[0.45238095, 0.9037037 ],
[0.30952381, 0.31851852],
[0.35714286, 0.19259259],
[0.64285714, 0.05185185],
[0.28571429, 0.        ],
[0.02380952, 0.02962963],
[0.73809524, 0.43703704],
[0.5       , 0.79259259],
[0.4047619 , 0.42962963],
[0.5       , 0.41481481],
[0.14285714, 0.05925926],
[0.54761905, 0.42222222],
[0.26190476, 0.5037037 ],
[0.85714286, 0.08148148],
[0.4047619 , 0.21481481],
[0.45238095, 0.44444444],
[0.26190476, 0.23703704],
[0.30952381, 0.39259259],
```

```
[0.57142857, 0.28888889],
[0.28571429, 0.88888889],
[0.80952381, 0.73333333],
[0.76190476, 0.15555556],
[0.9047619 , 0.87407407],
[0.26190476, 0.34074074],
[0.28571429, 0.54814815],
[0.19047619, 0.00740741],
[0.35714286, 0.11851852],
[0.54761905, 0.42222222],
[0.42857143, 0.13333333],
[0.88095238, 0.81481481],
[0.71428571, 0.85925926],
[0.54761905, 0.41481481],
[0.28571429, 0.34814815],
[0.45238095, 0.42222222],
[0.54761905, 0.35555556],
[0.95238095, 0.23703704],
[0.28571429, 0.74814815],
[0.04761905, 0.25185185],
[0.45238095, 0.43703704],
[0.54761905, 0.32592593],
[0.73809524, 0.54814815],
[0.23809524, 0.47407407],
[0.83333333, 0.4962963 ],
[0.52380952, 0.31111111],
[1.        , 0.14074074],
[0.4047619 , 0.68888889],
[0.07142857, 0.42222222],
[0.47619048, 0.41481481],
[0.5       , 0.67407407],
[0.45238095, 0.31111111],
[0.19047619, 0.42222222],
[0.4047619 , 0.05925926],
[0.85714286, 0.68888889],
[0.28571429, 0.01481481],
[0.5       , 0.88148148],
[0.26190476, 0.20740741],
[0.35714286, 0.20740741],
[0.4047619 , 0.17037037],
[0.54761905, 0.22222222],
[0.54761905, 0.42222222],
[0.5       , 0.88148148],
[0.21428571, 0.9037037 ],
[0.07142857, 0.00740741],
[0.19047619, 0.12592593],
[0.30952381, 0.37777778],
[0.5       , 0.42962963],
[0.54761905, 0.47407407],
[0.69047619, 0.25925926],
[0.54761905, 0.11111111],
[0.45238095, 0.57777778],
[1.        , 0.22962963],
[0.16666667, 0.05185185],
[0.23809524, 0.16296296],
[0.47619048, 0.2962963 ],
[0.42857143, 0.28888889],
[0.04761905, 0.15555556],
[0.9047619 , 0.65925926],
[0.52380952, 0.31111111],
[0.57142857, 0.68888889],
[0.04761905, 0.05925926],
[0.52380952, 0.37037037],
[0.69047619, 0.03703704],
[0.        , 0.52592593],
[0.4047619 , 0.47407407],
[0.92857143, 0.13333333],
[0.38095238, 0.42222222],
[0.73809524, 0.17777778],
```

```
       [0.21428571, 0.11851852],
       [0.02380952, 0.40740741],
       [0.5       , 0.47407407],
       [0.19047619, 0.48888889],
       [0.16666667, 0.48148148],
       [0.23809524, 0.51851852],
       [0.88095238, 0.17777778],
       [0.76190476, 0.54074074],
       [0.73809524, 0.54074074],
       [0.80952381, 1.        ],
       [0.4047619 , 0.37037037],
       [0.57142857, 0.28888889],
       [0.38095238, 0.20740741],
       [0.45238095, 0.27407407],
       [0.71428571, 0.11111111],
       [0.26190476, 0.20740741],
       [0.42857143, 0.27407407],
       [0.21428571, 0.28888889],
       [0.19047619, 0.76296296]])
```

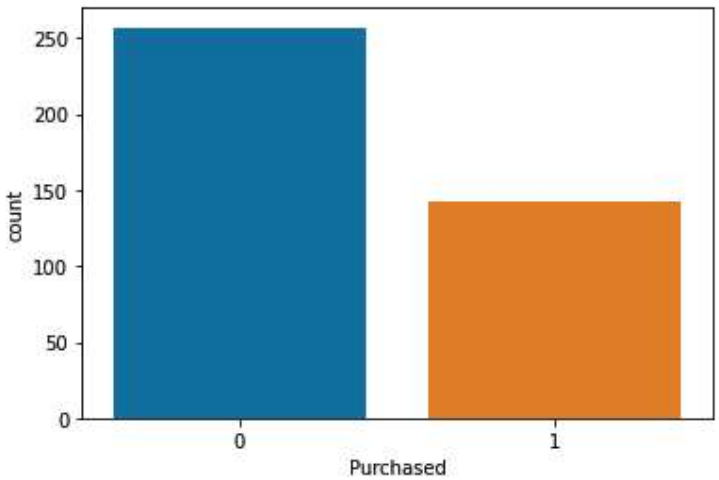
In [85]:
```
y_train
```

Out[85]:
```
250    0
63     1
312    0
159    1
283    1
      ..
323    1
192    0
117    0
47     0
172    0
Name: Purchased, Length: 300, dtype: int64
```

In [86]:
```
from sklearn.linear_model import LogisticRegression
```

In [87]:
```
import seaborn as sns
sns.countplot(x=y)
```

Out[87]:
```
<AxesSubplot:xlabel='Purchased', ylabel='count'>
```



In [88]:
```
y.value_counts()
```

Out[88]:
```
0    257
```

```
1    143
Name: Purchased, dtype: int64
```

In [89]:
```
#creat the object
classifier = LogisticRegression()
```

In [90]:
```
classifier.fit(x_train,y_train)
```

Out[90]:
```
LogisticRegression()
```

In [91]:
```
#predication
y_pred = classifier.predict(x_test)
```

In [92]:
```
y_train.shape
```

Out[92]:
```
(300,)
```

In [93]:
```
x_train.shape
```

Out[93]:
```
(300, 2)
```

In [94]:
```
y_pred
```

Out[94]:
```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1], dtype=int64)
```

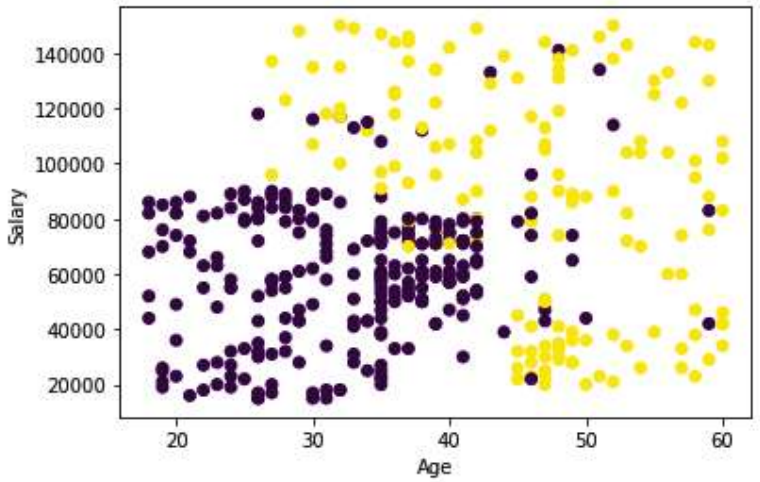
In [75]:
```
y_test
```

Out[75]:
```
132    0
309    0
341    0
196    0
246    0
      ..
146    1
135    0
390    1
264    1
364    1
Name: Purchased, Length: 100, dtype: int64
```

In [76]:
```
import matplotlib.pyplot as plt
```

In [77]:
```
plt.xlabel('Age')
plt.ylabel('Salary')
plt.scatter(x['Age'],x['EstimatedSalary'],c=y)
```

Out[77]:
```
<matplotlib.collections.PathCollection at 0x1814f5a3940>
```



In [78]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
x_scaled = scaler.fit_transform(x)
```

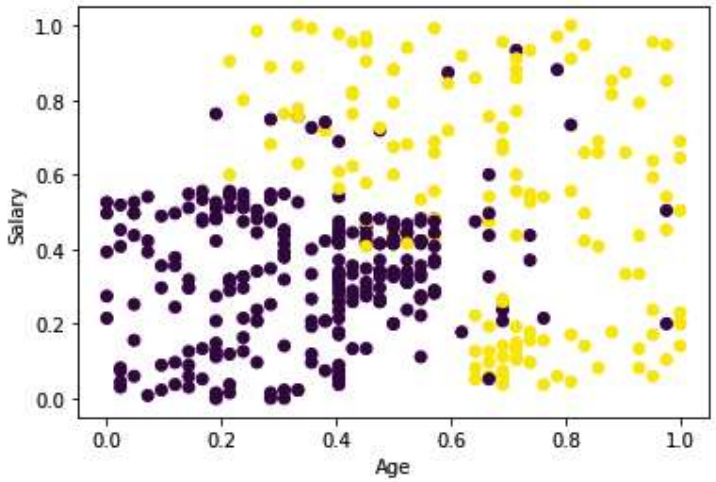
In [79]:
```
pd.DataFrame(x_scaled).describe()
```

Out[79]:

	0	1
count	400.000000	400.000000
mean	0.467976	0.405500
std	0.249592	0.252570
min	0.000000	0.000000
25%	0.279762	0.207407
50%	0.452381	0.407407
75%	0.666667	0.540741
max	1.000000	1.000000

In [80]:
```
plt.xlabel('Age')
plt.ylabel('Salary')
plt.scatter(x_scaled[:,0],x_scaled[:,1],c=y)
```

Out[80]:
```
<matplotlib.collections.PathCollection at 0x1814f8fda90>
```



In [95]:
```
from sklearn.metrics import confusion_matrix
```

In [96]:
```
confusion_matrix(y_test,y_pred)
```

Out[96]:
```
array([[67,  1],
       [10, 22]], dtype=int64)
```

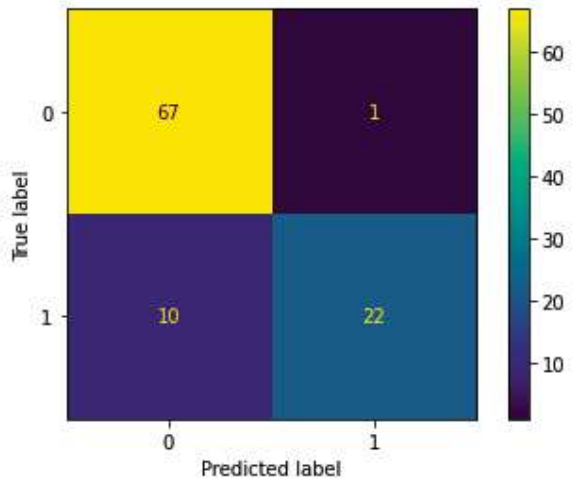
In [97]:
```
y_test.value_counts()
```

Out[97]:
```
0    68
1    32
Name: Purchased, dtype: int64
```

In [98]:
```
from sklearn.metrics import plot_confusion_matrix
```

In [103…
```
plot_confusion_matrix(classifier,x_test,y_test)
```

Out[103…
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1814f97daf0>
```



In [106…
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)
```

Out[106…
```
0.89
```

In [107…
```
from sklearn.metrics import classification_report
```

In [110…
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.87      0.99      0.92        68
           1       0.96      0.69      0.80        32

    accuracy                           0.89       100
   macro avg       0.91      0.84      0.86       100
weighted avg       0.90      0.89      0.88       100
```

In [111…
```
new1=[[26,34000]]
new2=[[57,138000]]
```

In [113…
```
classifier.predict(scaler.transform(new1))
```

Out[113…
```
array([0], dtype=int64)
```

In [112…
```
classifier.predict(scaler.transform(new2))
```

Out[112…
```
array([1], dtype=int64)
```

Experiment No. 6

Aim: Implement simple Navie Bayes Classifications algorithm. Using python on iris.csv dataset.

Requirement:

- Anaconda Installer
- Windows 10 OS
- Jupyter Notebook

Theory:

Naïve Bayes Classifier Algorithm:

- Naïve Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems.
- It is mainly used in text classification that includes a high-dimensional training dataset.
- Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.
- It is a probabilistic classifier, which means it predicts on the basis of the probability of an object.
- Some popular examples of Naïve Bayes Algorithm are spam filtration, Sentimental analysis, and classifying articles.

Why is it called Naïve Bayes?

The Naïve Bayes algorithm is comprised of two words Naïve and Bayes, Which can be described as:

- **Naïve**: It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the bases of color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other.
- **Bayes**: It is called Bayes because it depends on the principle of Bayes' Theorem.

Bayes' Theorem:

- Bayes' theorem is also known as **Bayes' Rule** or **Bayes' law**, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.

- The formula for Bayes' theorem is given as:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Where,

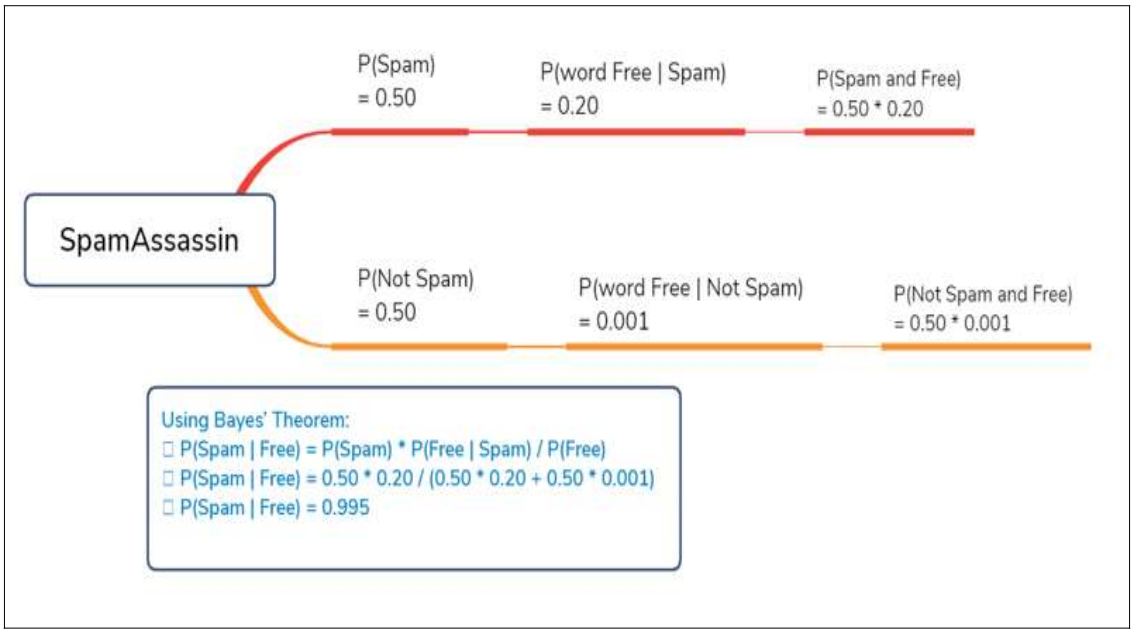
P(A|B) is Posterior probability: Probability of hypothesis A on the observed event B.

P(B|A) is Likelihood probability: Probability of the evidence given that the probability of a hypothesis is true.

P(A) is Prior Probability: Probability of hypothesis before observing the evidence.

P(B) is Marginal Probability: Probability of Evidence.

Example:



Advantages of Naive Bayes:

- This algorithm works very fast and can easily predict the class of a test dataset.

- You can use it to solve multi-class prediction problems as it's quite useful with them.
- Naive Bayes classifier performs better than other models with less training data if the assumption of independence of features holds.
- If you have categorical input variables, the Naive Bayes algorithm performs exceptionally well in comparison to numerical variables.

Disadvantages of Naive Bayes:

- If your test data set has a categorical variable of a category that wasn't present in the training data set, the Naive Bayes model will assign it zero probability and won't be able to make any predictions in this regard. This phenomenon is called 'Zero Frequency,' and you'll have to use a smoothing technique to solve this problem.
- This algorithm is also notorious as a lousy estimator. So, you shouldn't take the probability outputs of 'predict_proba' too seriously.
- It assumes that all the features are independent. While it might sound great in theory, in real life, you'll hardly find a set of independent features.

Applications of Naive Bayes Algorithm:

As you must've noticed, this algorithm offers plenty of advantages to its users. That's why it has a lot of applications in various sectors too. Here are some applications of Naive Bayes algorithm:

- As this algorithm is fast and efficient, you can use it to make real-time predictions.
- This algorithm is popular for multi-class predictions. You can find the probability of multiple target classes easily by using this algorithm.
- Email services (like Gmail) use this algorithm to figure out whether an email is a spam or not. This algorithm is excellent for spam filtering.

- Its assumption of feature independence, and its effectiveness in solving multi-class problems, makes it perfect for performing Sentiment Analysis. Sentiment Analysis refers to the identification of positive or negative sentiments of a target group (customers, audience, etc.)
- Collaborative Filtering and the Naive Bayes algorithm work together to build recommendation systems. These systems use data mining and machine learning to predict if the user would like a particular resource or not.

Types of Naive Bayes Classifier:
This algorithm has multiple kinds. Here are the main ones:

1. Bernoulli Naive Bayes:

Here, the predictors are boolean variables. So, the only values you have are 'True' and 'False' (you could also have 'Yes' or 'No'). We use it when the data is according to multivariate Bernoulli distribution.

2. Multinomial Naive Bayes:

People use this algorithm to solve document classification problems. For example, if you want to determine whether a document belongs to the 'Legal' category or 'Human Resources' category, you'd use this algorithm to sort it out. It uses the frequency of the present words as features.

3. Gaussian Naive Bayes

If the predictors aren't discrete but have a continuous value, we assume that they are a sample from a gaussian distribution.

Libraries Used:

1. Pandas: Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring and manipulating data.
2. Sklearn: It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python.

Conclusion:

In this experiment we have studied about Navie bayes algorithm and implemented it an the iris dataset for spam filteration

In [1]:
```
import pandas as pd
```

In [2]:
```
df = pd.read_csv('iris.csv')
```

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

Out[3]:
```
(150, 5)
```

In [4]:
```
#input data
x=df.drop('species',axis=1)

#output data
y=df['species']
```

In [5]:
```
y.value_counts()
```

Out[5]:
```
versicolor    50
setosa        50
virginica     50
Name: species, dtype: int64
```

#cross validation from sklearn.model_selection import train_test_split

In [8]:
```
#cross validation
from sklearn.model_selection import train_test_split
```

In [15]:
```
x_train ,x_test,y_train,y_test=train_test_split(x,y,random_state=0,test_size=0.25)
```

In [17]:
```
x_train.shape
```

Out[17]:
```
(112, 4)
```

In [18]:
```
x_test.shape
```

Out[18]:
```
(38, 4)
```

In [19]:
```
#import the class
from sklearn.naive_bayes import GaussianNB
```

In [20]:
```
#create the object
clf= GaussianNB()
```

In [22]:
```
#train the algorithm
clf.fit(x_train,y_train)
```

Out[22]:
```
GaussianNB()
```

In [23]:

```
y_pred=clf.predict(x_test)
```

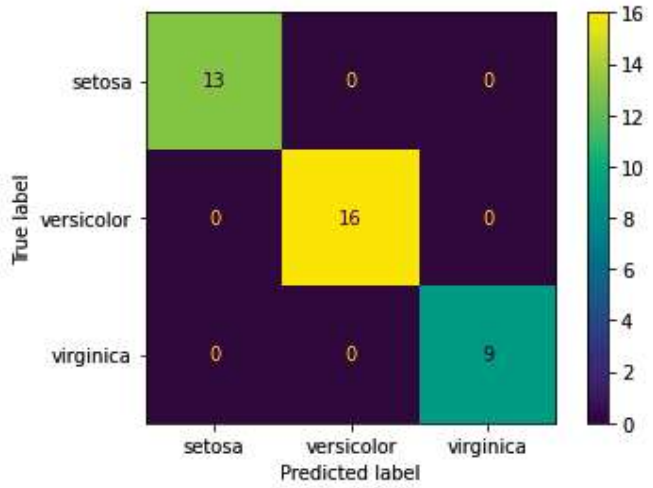
In [24]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import plot_confusion_matrix
```

In [25]:
```
confusion_matrix(y_test,y_pred)
```

Out[25]:
```
array([[13,  0,  0],
       [ 0, 16,  0],
       [ 0,  0,  9]], dtype=int64)
```

In [30]:
```
plot_confusion_matrix(clf,x_test,y_test)
```

Out[30]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x19e50f69dc0>
```



In [31]:
```
accuracy_score(y_test,y_pred)
```

Out[31]:
```
1.0
```

In [34]:
```
clf.predict_proba(x_test)
```

Out[34]:
```
array([[2.05841140e-233, 1.23816844e-006, 9.99998762e-001],
       [1.76139943e-084, 9.99998414e-001, 1.58647449e-006],
       [1.00000000e+000, 1.48308613e-018, 1.73234612e-027],
       [6.96767669e-312, 5.33743814e-007, 9.99999466e-001],
       [1.00000000e+000, 9.33944060e-017, 1.22124682e-026],
       [0.00000000e+000, 6.57075840e-011, 1.00000000e+000],
       [1.00000000e+000, 1.05531886e-016, 1.55777574e-026],
       [2.45560284e-149, 7.80950359e-001, 2.19049641e-001],
       [4.01160627e-153, 9.10103555e-001, 8.98964447e-002],
       [1.46667004e-094, 9.99887821e-001, 1.12179234e-004],
       [5.29999917e-215, 4.59787449e-001, 5.40212551e-001],
       [4.93479766e-134, 9.46482991e-001, 5.35170089e-002],
       [5.23735688e-135, 9.98906155e-001, 1.09384481e-003],
       [4.97057521e-142, 9.50340361e-001, 4.96596389e-002],
       [9.11315109e-143, 9.87982897e-001, 1.20171030e-002],
       [1.00000000e+000, 7.81797826e-019, 1.29694954e-028],
       [3.86310964e-133, 9.87665084e-001, 1.23349155e-002],
       [2.27343573e-113, 9.99940331e-001, 5.96690955e-005],
       [1.00000000e+000, 1.80007196e-015, 9.14666201e-026],
```

```
       [1.00000000e+000, 1.30351394e-015, 8.42776899e-025],
       [4.66537803e-188, 1.18626155e-002, 9.88137385e-001],
       [1.02677291e-131, 9.92205279e-001, 7.79472050e-003],
       [1.00000000e+000, 6.61341173e-013, 1.42044069e-022],
       [1.00000000e+000, 9.98321355e-017, 3.50690661e-027],
       [2.27898063e-170, 1.61227371e-001, 8.38772629e-001],
       [1.00000000e+000, 2.29415652e-018, 2.54202512e-028],
       [1.00000000e+000, 5.99780345e-011, 5.24260178e-020],
       [1.62676386e-112, 9.99340062e-001, 6.59938068e-004],
       [2.23238199e-047, 9.99999965e-001, 3.47984452e-008],
       [1.00000000e+000, 1.95773682e-013, 4.10256723e-023],
       [3.52965800e-228, 1.15450262e-003, 9.98845497e-001],
       [3.20480410e-131, 9.93956330e-001, 6.04366979e-003],
       [1.00000000e+000, 1.14714843e-016, 2.17310302e-026],
       [3.34423817e-177, 8.43422262e-002, 9.15657774e-001],
       [5.60348582e-264, 1.03689515e-006, 9.99998963e-001],
       [7.48035097e-091, 9.99950155e-001, 4.98452400e-005],
       [1.00000000e+000, 1.80571225e-013, 1.83435499e-022],
       [8.97496247e-182, 5.65567226e-001, 4.34432774e-001]])
```

In [35]:
```
newl=[[4.5,2.9,3.1,0.4]]
clf.predict(newl)[0]
```

Out[35]: 'versicolor'

In [36]:
```
newl=[[5.5,3.1,1.0,0.8]]
clf.predict(newl)[0]
```

Out[36]: 'setosa'

In [37]:
```
newl=[[6.5,3.3,4.9,1.8]]
clf.predict(newl)[0]
```

Out[37]: 'virginica'

In [39]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        13
  versicolor       1.00      1.00      1.00        16
   virginica       1.00      1.00      1.00         9

    accuracy                           1.00        38
   macro avg       1.00      1.00      1.00        38
weighted avg       1.00      1.00      1.00        38
```

Experiment No. 7

Aim: Extract Sample document and apply following document preprocessing methods: Tokenization, POS Tagging, stop words removal, Stemming and Lemmatization.

Requirement:

- Anaconda Installer
- Windows 10 OS
- Linux
- Jupyter Notebook

Theory:

Data Preprocessing in Python:

Pre-processing refers to the transformations applied to our data before feeding it to the algorithm. Data Preprocessing is a technique that is used to convert the raw data into a clean data set. In other words, whenever the data is gathered from different sources it is collected in raw format which is not feasible for the analysis.

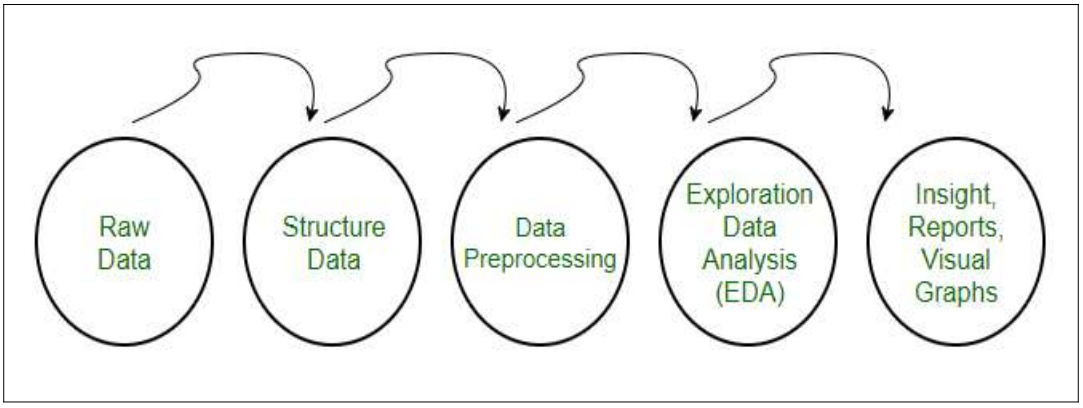
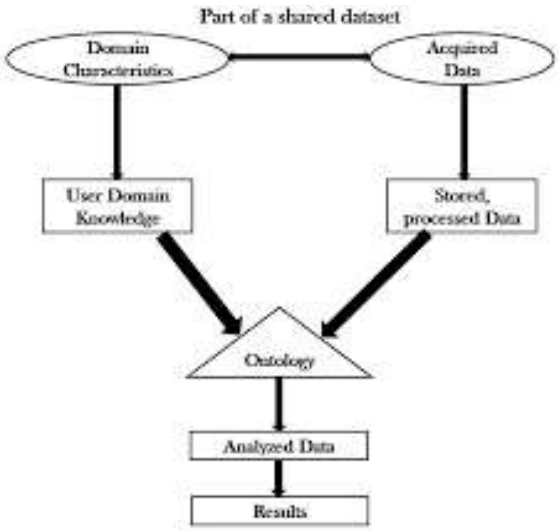


Fig. Data Preprocessing in Python

Need of Data Preprocessing:

- For achieving better results from the applied model in Machine Learning projects the format of the data has to be in a proper manner. Some specified Machine Learning model needs information in a specified format, for example, Random Forest algorithm does not support null values, therefore to execute random forest algorithm null values have to be managed from the original raw data set.

- Another aspect is that the data set should be formatted in such a way that more than one Machine Learning and Deep Learning algorithm are executed in one data set, and best out of them is chosen.



NLP Techniques in data science:

1. Tokenize text using NLTK in python:

To run the below python program, (NLTK) natural language toolkit has to be installed in your system.The NLTK module is a massive tool kit, aimed at helping you with the entire Natural Language Processing (NLP) methodology. In order to install NLTK run the following commands in your terminal.

- sudo pip install nltk
- Then, enter the python shell in your terminal by simply typing python
- Type import nltk
- nltk.download('all')

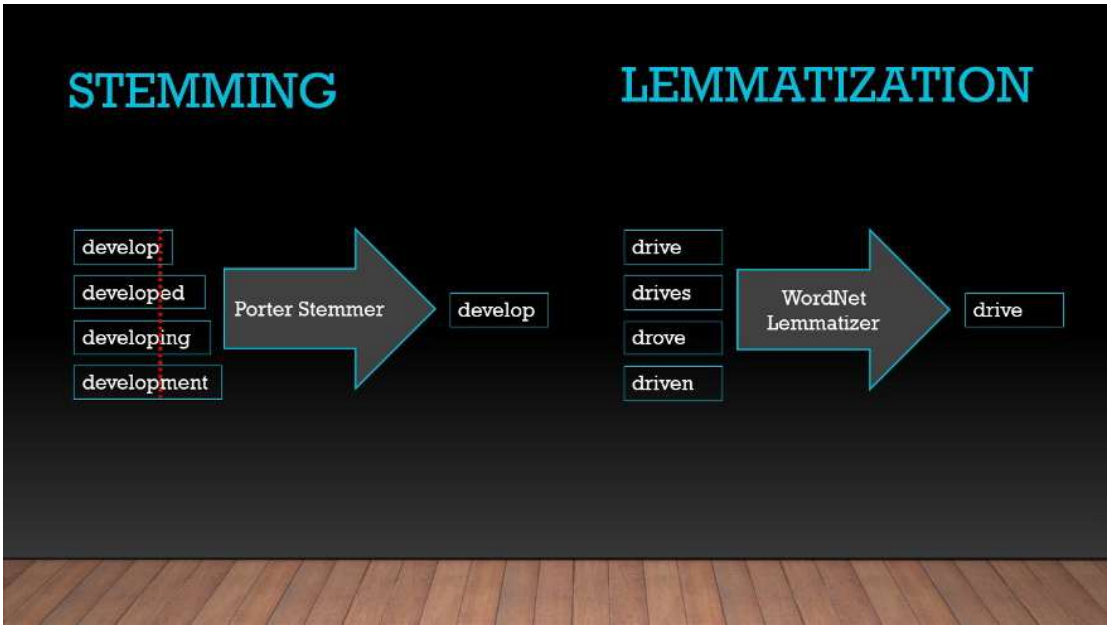
2. Removing stop words with NLTK in Python:

The process of converting data to something a computer can understand is referred to as pre-processing. One of the major forms of pre-processing is to filter out useless data. In natural language processing, useless words (data), are referred to as stop words.

What are Stop words?

Stop Words: A stop word is a commonly used word (such as “the”, “a”, “an”, “in”) that a search engine has been programmed to ignore, both when indexing entries for searching and when retrieving them as the result of a search query. We would not want these words to take up space in our database, or taking up valuable processing time. For this, we can remove them easily, by storing a list of words that you consider to stop words. NLTK(Natural Language Toolkit) in python has a list of stopwords stored in 16 different languages. You can find them in the nltk_data directory.

3. Lemmatization with NLTK:



Lemmatization is the process of grouping together the different inflected forms of a word so they can be analyzed as a single item. Lemmatization is similar to stemming but it brings context to the words. So it links words with similar meanings to one word.Text preprocessing includes both Stemming as well as Lemmatization. Many times people find these two terms confusing. Some treat

these two as the same. Actually, lemmatization is preferred over Stemming because lemmatization does morphological analysis of the words.

Applications of lemmatization are:

- Used in comprehensive retrieval systems like search engines.
- Used in compact indexing

Advantages and Disadvantages of Lemmatization:

As you could probably tell by now, the obvious advantage of lemmatization is that it is more accurate. So if you're dealing with an NLP application such as a chat bot or a virtual assistant where understanding the meaning of the dialogue is crucial, lemmatization would be useful. But this accuracy comes at a cost.

Because lemmatization involves deriving the meaning of a word from something like a dictionary, it's very time consuming. So most lemmatization algorithms are slower compared to their stemming counterparts. There is also a computation overhead for lemmatization, however, in an ML problem, computational resources are rarely a cause of concern.

4. Stemming words with NLTK:

Stemming is the process of producing morphological variants of a root/base word. Stemming programs are commonly referred to as stemming algorithms or stemmers.

A stemming algorithm reduces the words "chocolates", "chocolatey", "choco" to the root word, "chocolate" and "retrieval", "retrieved", "retrieves" reduce to the stem "retrieve".

Some more example of stemming for root word "like" include:

-> "likes"

-> "liked"

-> "likely"

-> "liking"

Errors in Stemming:

There are mainly two errors in stemming – *Overstemming* and *Understemming*. Overstemming occurs when two words are stemmed to same root that are of different stems. Under-stemming occurs when two words are stemmed to same root that are not of different stems.

Libraries Used:

1. NLTK: NLTK is a standard python library that provides a set of diverse algorithms for NLP. It is one of the most used libraries for NLP and Computational Linguistics.
2. Sklearn: It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python.
3. Pandas: Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring and manipulating data.
4. String: Python String module contains some constants, utility function, and classes for string manipulation.

Conclusion:

In this experiment we have studied about data preprocessingusing natural language processing technique.

In [1]:
```
!pip install nltk -U
!pip install bs4 -U
```

```
Requirement already satisfied: nltk in c:\users\ganes\anaconda3\lib\site-packages
(3.6.1)
Collecting nltk
  Downloading nltk-3.7-py3-none-any.whl (1.5 MB)
Requirement already satisfied: tqdm in c:\users\ganes\anaconda3\lib\site-packages (f
rom nltk) (4.59.0)
Requirement already satisfied: joblib in c:\users\ganes\anaconda3\lib\site-packages
(from nltk) (1.0.1)
Collecting regex>=2021.8.3
  Downloading regex-2022.3.15-cp38-cp38-win_amd64.whl (274 kB)
Requirement already satisfied: click in c:\users\ganes\anaconda3\lib\site-packages
(from nltk) (7.1.2)
Installing collected packages: regex, nltk
  Attempting uninstall: regex
    Found existing installation: regex 2021.4.4
    Uninstalling regex-2021.4.4:
      Successfully uninstalled regex-2021.4.4
  Attempting uninstall: nltk
    Found existing installation: nltk 3.6.1
    Uninstalling nltk-3.6.1:
      Successfully uninstalled nltk-3.6.1
Successfully installed nltk-3.7 regex-2022.3.15
Collecting bs4
  Downloading bs4-0.0.1.tar.gz (1.1 kB)
Requirement already satisfied: beautifulsoup4 in c:\users\ganes\anaconda3\lib\site-p
ackages (from bs4) (4.9.3)
Requirement already satisfied: soupsieve>1.2 in c:\users\ganes\anaconda3\lib\site-pa
ckages (from beautifulsoup4->bs4) (2.2.1)
Building wheels for collected packages: bs4
  Building wheel for bs4 (setup.py): started
  Building wheel for bs4 (setup.py): finished with status 'done'
  Created wheel for bs4: filename=bs4-0.0.1-py3-none-any.whl size=1273 sha256=721f8e
06d273b0ecb1434819b8f36360384fc191f4103b160e7ea3cf97e0f1db
  Stored in directory: c:\users\ganes\appdata\local\pip\cache\wheels\75\78\21\68b124
549c9bdc94f822c02fb9aa3578a669843f9767776bca
Successfully built bs4
Installing collected packages: bs4
Successfully installed bs4-0.0.1
```

In [4]:
```
import nltk
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\ganes\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\ganes\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\ganes\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\ganes\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping taggers\averaged_perceptron_tagger.zip.
```

Out[4]:
```
True
```

In [5]:
```
import nltk
```

In [6]:
```
para='Rajgad (literal meaning Ruling Fort) is a hill fort situated in the Pune distr
```

In [7]:
```
print(para)
```

```
Rajgad (literal meaning Ruling Fort) is a hill fort situated in the Pune district of
Maharashtra, India. Formerly known as Murumdev, the fort was the capital of the Mara
tha Empire under the rule of Chatrapati Shivaji Maharaj for almost 26 years, after w
hich the capital was moved to the Raigad Fort.[1] Treasures discovered from an adjac
ent fort called Torna were used to completely build and fortify the Rajgad Fort.
```

In [8]:
```
para.split()
```

Out[8]:
```
['Rajgad',
 '(literal',
 'meaning',
 'Ruling',
 'Fort)',
 'is',
 'a',
 'hill',
 'fort',
 'situated',
 'in',
 'the',
 'Pune',
 'district',
 'of',
 'Maharashtra,',
 'India.',
 'Formerly',
 'known',
 'as',
 'Murumdev,',
 'the',
 'fort',
 'was',
 'the',
 'capital',
 'of',
 'the',
 'Maratha',
 'Empire',
 'under',
 'the',
 'rule',
 'of',
 'Chatrapati',
 'Shivaji',
 'Maharaj',
 'for',
 'almost',
 '26',
 'years,',
 'after',
 'which',
 'the',
 'capital',
 'was',
 'moved',
 'to',
 'the',
 'Raigad',
 'Fort.[1]',
 'Treasures',
 'discovered',
```

```
 'from',
 'an',
 'adjacent',
 'fort',
 'called',
 'Torna',
 'were',
 'used',
 'to',
 'completely',
 'build',
 'and',
 'fortify',
 'the',
 'Rajgad',
 'Fort.']
```

In [9]:
```
from nltk.tokenize import sent_tokenize
from nltk.tokenize import word_tokenize
```

In [10]:
```
sent=sent_tokenize(para)
```

In [11]:
```
sent[2]
```

Out[11]:
```
'[1] Treasures discovered from an adjacent fort called Torna were used to completely
build and fortify the Rajgad Fort.'
```

In [12]:
```
words=word_tokenize(para)
```

In [13]:
```
words
```

Out[13]:
```
['Rajgad',
 '(',
 'literal',
 'meaning',
 'Ruling',
 'Fort',
 ')',
 'is',
 'a',
 'hill',
 'fort',
 'situated',
 'in',
 'the',
 'Pune',
 'district',
 'of',
 'Maharashtra',
 ',',
 'India',
 '.',
 'Formerly',
 'known',
 'as',
 'Murumdev',
 ',',
 'the',
 'fort',
 'was',
 'the',
 'capital',
```

```
 'of',
 'the',
 'Maratha',
 'Empire',
 'under',
 'the',
 'rule',
 'of',
 'Chatrapati',
 'Shivaji',
 'Maharaj',
 'for',
 'almost',
 '26',
 'years',
 ',',
 'after',
 'which',
 'the',
 'capital',
 'was',
 'moved',
 'to',
 'the',
 'Raigad',
 'Fort',
 '.',
 '[',
 '1',
 ']',
 'Treasures',
 'discovered',
 'from',
 'an',
 'adjacent',
 'fort',
 'called',
 'Torna',
 'were',
 'used',
 'to',
 'completely',
 'build',
 'and',
 'fortify',
 'the',
 'Rajgad',
 'Fort',
 '.']
```

In [14]:
```
from nltk.corpus import stopwords
```

In [15]:
```
swords=stopwords.words('english')
```

In [16]:
```
swords
```

Out[16]:
```
['i',
 'me',
 'my',
 'myself',
 'we',
 'our',
 'ours',
 'ourselves',
 'you'
```

```
"you're",
"you've",
"you'll",
"you'd",
'your',
'yours',
'yourself',
'yourselves',
'he',
'him',
'his',
'himself',
'she',
"she's",
'her',
'hers',
'herself',
'it',
"it's",
'its',
'itself',
'they',
'them',
'their',
'theirs',
'themselves',
'what',
'which',
'who',
'whom',
'this',
'that',
"that'll",
'these',
'those',
'am',
'is',
'are',
'was',
'were',
'be',
'been',
'being',
'have',
'has',
'had',
'having',
'do',
'does',
'did',
'doing',
'a',
'an',
'the',
'and',
'but',
'if',
'or',
'because',
'as',
'until',
'while',
'of',
'at',
'by',
'for',
'with',
'about',
'against',
```

```
'between',
'into',
'through',
'during',
'before',
'after',
'above',
'below',
'to',
'from',
'up',
'down',
'in',
'out',
'on',
'off',
'over',
'under',
'again',
'further',
'then',
'once',
'here',
'there',
'when',
'where',
'why',
'how',
'all',
'any',
'both',
'each',
'few',
'more',
'most',
'other',
'some',
'such',
'no',
'nor',
'not',
'only',
'own',
'same',
'so',
'than',
'too',
'very',
's',
't',
'can',
'will',
'just',
'don',
"don't",
'should',
"should've",
'now',
'd',
'll',
'm',
'o',
're',
've',
'y',
'ain',
'aren',
"aren't",
'couldn',
```

```
"couldn't",
'didn',
"didn't",
'doesn',
"doesn't",
'hadn',
"hadn't",
'hasn',
"hasn't",
'haven',
"haven't",
'isn',
"isn't",
'ma',
'mightn',
"mightn't",
'mustn',
"mustn't",
'needn',
"needn't",
'shan',
"shan't",
'shouldn',
"shouldn't",
'wasn',
"wasn't",
'weren',
"weren't",
'won',
"won't",
'wouldn',
"wouldn't"]
```

In [17]:
```
x=[word for word in words if word not in swords]
```

In [18]:
```
x
```

Out[18]:
```
['Rajgad',
 '(',
 'literal',
 'meaning',
 'Ruling',
 'Fort',
 ')',
 'hill',
 'fort',
 'situated',
 'Pune',
 'district',
 'Maharashtra',
 ',',
 'India',
 '.',
 'Formerly',
 'known',
 'Murumdev',
 ',',
 'fort',
 'capital',
 'Maratha',
 'Empire',
 'rule',
 'Chatrapati',
 'Shivaji',
 'Maharaj',
 'almost',
```

```
 '26',
 'years',
 ',',
 'capital',
 'moved',
 'Raigad',
 'Fort',
 '.',
 '[',
 '1',
 ']',
 'Treasures',
 'discovered',
 'adjacent',
 'fort',
 'called',
 'Torna',
 'used',
 'completely',
 'build',
 'fortify',
 'Rajgad',
 'Fort',
 '.']
```

In [19]:
```
x=[word for word in words if word.lower() not in swords]
```

In [20]:
```
x
```

Out[20]:
```
['Rajgad',
 '(',
 'literal',
 'meaning',
 'Ruling',
 'Fort',
 ')',
 'hill',
 'fort',
 'situated',
 'Pune',
 'district',
 'Maharashtra',
 ',',
 'India',
 '.',
 'Formerly',
 'known',
 'Murumdev',
 ',',
 'fort',
 'capital',
 'Maratha',
 'Empire',
 'rule',
 'Chatrapati',
 'Shivaji',
 'Maharaj',
 'almost',
 '26',
 'years',
 ',',
 'capital',
 'moved',
 'Raigad',
 'Fort',
 '.',
```

```
 '[',
 '1',
 ']',
 'Treasures',
 'discovered',
 'adjacent',
 'fort',
 'called',
 'Torna',
 'used',
 'completely',
 'build',
 'fortify',
 'Rajgad',
 'Fort',
 '.']
```

In [21]:
```
from nltk.stem import PorterStemmer
```

In [22]:
```
ps=PorterStemmer()
```

In [24]:
```
ps.stem('working')
```

Out[24]:
```
'work'
```

In [25]:
```
y=[ps.stem(word) for word in x]
```

In [26]:
```
y
```

Out[26]:
```
['rajgad',
 '(',
 'liter',
 'mean',
 'rule',
 'fort',
 ')',
 'hill',
 'fort',
 'situat',
 'pune',
 'district',
 'maharashtra',
 ',',
 'india',
 '.',
 'formerli',
 'known',
 'murumdev',
 ',',
 'fort',
 'capit',
 'maratha',
 'empir',
 'rule',
 'chatrapati',
 'shivaji',
 'maharaj',
 'almost',
 '26',
 'year',
 '.']
```

```
'capit',
'move',
'raigad',
'fort',
'.',
'[',
'1',
']',
'treasur',
'discov',
'adjac',
'fort',
'call',
'torna',
'use',
'complet',
'build',
'fortifi',
'rajgad',
'fort',
'.']
```

In [30]:
```
from nltk.stem import WordNetLemmatizer
```

In [31]:
```
wnl=WordNetLemmatizer()
```

In [32]:
```
wnl.lemmatize('working',pos='v')
#a-adjective
#n-noun
#r-adverb
```

```
---------------------------------------------------------------------------
LookupError                               Traceback (most recent call last)
~\anaconda3\lib\site-packages\nltk\corpus\util.py in __load(self)
     83                 try:
---> 84                     root = nltk.data.find(f"{self.subdir}/{zip_name}")
     85                 except LookupError:

~\anaconda3\lib\site-packages\nltk\data.py in find(resource_name, paths)
    582     resource_not_found = f"\n{sep}\n{msg}\n{sep}\n"
--> 583     raise LookupError(resource_not_found)
    584

LookupError:
**********************************************************************
  Resource omw-1.4 not found.
  Please use the NLTK Downloader to obtain the resource:

  >>> import nltk
  >>> nltk.download('omw-1.4')

  For more information see: https://www.nltk.org/data.html

  Attempted to load corpora/omw-1.4.zip/omw-1.4/

  Searched in:
    - 'C:\\Users\\ganes/nltk_data'
    - 'C:\\Users\\ganes\\anaconda3\\nltk_data'
    - 'C:\\Users\\ganes\\anaconda3\\share\\nltk_data'
    - 'C:\\Users\\ganes\\anaconda3\\lib\\nltk_data'
    - 'C:\\Users\\ganes\\AppData\\Roaming\\nltk_data'
    - 'C:\\nltk_data'
    - 'D:\\nltk_data'
    - 'E:\\nltk_data'
```

```
     83                 try:

~\anaconda3\lib\site-packages\nltk\data.py in find(resource_name, paths)
    581     sep = "*" * 70
    582     resource_not_found = f"\n{sep}\n{msg}\n{sep}\n"
--> 583     raise LookupError(resource_not_found)
    584
    585

LookupError:
**********************************************************************
  Resource omw-1.4 not found.
  Please use the NLTK Downloader to obtain the resource:

  >>> import nltk
  >>> nltk.download('omw-1.4')

  For more information see: https://www.nltk.org/data.html

  Attempted to load corpora/omw-1.4

  Searched in:
    - 'C:\\Users\\ganes/nltk_data'
    - 'C:\\Users\\ganes\\anaconda3\\nltk_data'
    - 'C:\\Users\\ganes\\anaconda3\\share\\nltk_data'
    - 'C:\\Users\\ganes\\anaconda3\\lib\\nltk_data'
    - 'C:\\Users\\ganes\\AppData\\Roaming\\nltk_data'
    - 'C:\\nltk_data'
    - 'D:\\nltk_data'
    - 'E:\\nltk_data'
**********************************************************************
```

In [33]:
```
nltk.download('omw-1.4')
```

```
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     C:\Users\ganes\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping corpora\omw-1.4.zip.
```

Out[33]: True

In [34]:
```
wnl.lemmatize('working',pos='v')
#a-adjective
#n-noun
#r-adverb
```

Out[34]: 'work'

In [35]:
```
print(ps.stem('went'))
print(wnl.lemmatize('went',pos='v'))
```

```
went
go
```

In [36]:
```
z=[wnl.lemmatize(word,pos='v') for word in x]
```

In [37]:
```
z
```

Out[37]:
```
['Rajgad',
 '(',
 'literal',
 'mean',
 'Ruling',
```

```
 'Fort',
 ')',
 'hill',
 'fort',
 'situate',
 'Pune',
 'district',
 'Maharashtra',
 ',',
 'India',
 '.',
 'Formerly',
 'know',
 'Murumdev',
 ',',
 'fort',
 'capital',
 'Maratha',
 'Empire',
 'rule',
 'Chatrapati',
 'Shivaji',
 'Maharaj',
 'almost',
 '26',
 'years',
 ',',
 'capital',
 'move',
 'Raigad',
 'Fort',
 '.',
 '[',
 '1',
 ']',
 'Treasures',
 'discover',
 'adjacent',
 'fort',
 'call',
 'Torna',
 'use',
 'completely',
 'build',
 'fortify',
 'Rajgad',
 'Fort',
 '.']
```

In [38]:
```
import string
```

In [39]:
```
string.punctuation
```

Out[39]:
```
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

In [40]:
```
t=[word for word in words if word not in string.punctuation]
```

In [41]:
```
t
```

Out[41]:
```
['Rajgad',
 'literal',
 'meaning',
 'Ruling',
```

```
'Fort',
'is',
'a',
'hill',
'fort',
'situated',
'in',
'the',
'Pune',
'district',
'of',
'Maharashtra',
'India',
'Formerly',
'known',
'as',
'Murumdev',
'the',
'fort',
'was',
'the',
'capital',
'of',
'the',
'Maratha',
'Empire',
'under',
'the',
'rule',
'of',
'Chatrapati',
'Shivaji',
'Maharaj',
'for',
'almost',
'26',
'years',
'after',
'which',
'the',
'capital',
'was',
'moved',
'to',
'the',
'Raigad',
'Fort',
'1',
'Treasures',
'discovered',
'from',
'an',
'adjacent',
'fort',
'called',
'Torna',
'were',
'used',
'to',
'completely',
'build',
'and',
'fortify',
'the',
'Rajgad',
'Fort']
```

In [42]:

```
from nltk import pos_tag
```

In [43]:
```
pos_tag(t)
```

Out[43]:
```
[('Rajgad', 'NNP'),
 ('literal', 'JJ'),
 ('meaning', 'NN'),
 ('Ruling', 'NNP'),
 ('Fort', 'NNP'),
 ('is', 'VBZ'),
 ('a', 'DT'),
 ('hill', 'NN'),
 ('fort', 'NN'),
 ('situated', 'VBN'),
 ('in', 'IN'),
 ('the', 'DT'),
 ('Pune', 'NNP'),
 ('district', 'NN'),
 ('of', 'IN'),
 ('Maharashtra', 'NNP'),
 ('India', 'NNP'),
 ('Formerly', 'RB'),
 ('known', 'VBN'),
 ('as', 'IN'),
 ('Murumdev', 'NNP'),
 ('the', 'DT'),
 ('fort', 'NN'),
 ('was', 'VBD'),
 ('the', 'DT'),
 ('capital', 'NN'),
 ('of', 'IN'),
 ('the', 'DT'),
 ('Maratha', 'NNP'),
 ('Empire', 'NNP'),
 ('under', 'IN'),
 ('the', 'DT'),
 ('rule', 'NN'),
 ('of', 'IN'),
 ('Chatrapati', 'NNP'),
 ('Shivaji', 'NNP'),
 ('Maharaj', 'NNP'),
 ('for', 'IN'),
 ('almost', 'RB'),
 ('26', 'CD'),
 ('years', 'NNS'),
 ('after', 'IN'),
 ('which', 'WDT'),
 ('the', 'DT'),
 ('capital', 'NN'),
 ('was', 'VBD'),
 ('moved', 'VBN'),
 ('to', 'TO'),
 ('the', 'DT'),
 ('Raigad', 'NNP'),
 ('Fort', 'NNP'),
 ('1', 'CD'),
 ('Treasures', 'NNS'),
 ('discovered', 'VBN'),
 ('from', 'IN'),
 ('an', 'DT'),
 ('adjacent', 'JJ'),
 ('fort', 'NN'),
 ('called', 'VBN'),
 ('Torna', 'NNP'),
 ('were', 'VBD'),
 ('used', 'VBN'),
 ('to', 'TO'),
 ('completely', 'RB'),
 ('build', 'VB'),
```

```
('and', 'CC'),
('fortify', 'VB'),
('the', 'DT'),
('Rajgad', 'NNP'),
('Fort', 'NNP')]
```

In [44]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [45]:
```
tfidf = TfidfVectorizer()
```

In [46]:
```
v=tfidf.fit_transform(t)
```

In [47]:
```
v.shape
```

Out[47]: (70, 50)

In [48]:
```
import pandas as pd
pd.DataFrame(v)
```

Out[48]:

	0
0	(0, 35)\t1.0
1	(0, 25)\t1.0
2	(0, 29)\t1.0
3	(0, 37)\t1.0
4	(0, 17)\t1.0
...	...
65	(0, 5)\t1.0
66	(0, 18)\t1.0
67	(0, 40)\t1.0
68	(0, 35)\t1.0
69	(0, 17)\t1.0

70 rows × 1 columns

In [ ]:
```
```

Experiment No. 8

Aim: Use the inbuilt dataset “titanic” the dataset contains 891 rows and contains informationabout the passengers who boarded the unfortunate titanic ship use the seaborn liabrary to see if we can find any patterns in the data.

Requirement:

- Anaconda Installer
- Windows 10 OS
- Linux
- Jupyter Notebook

Theory:

What is Data Visualization?

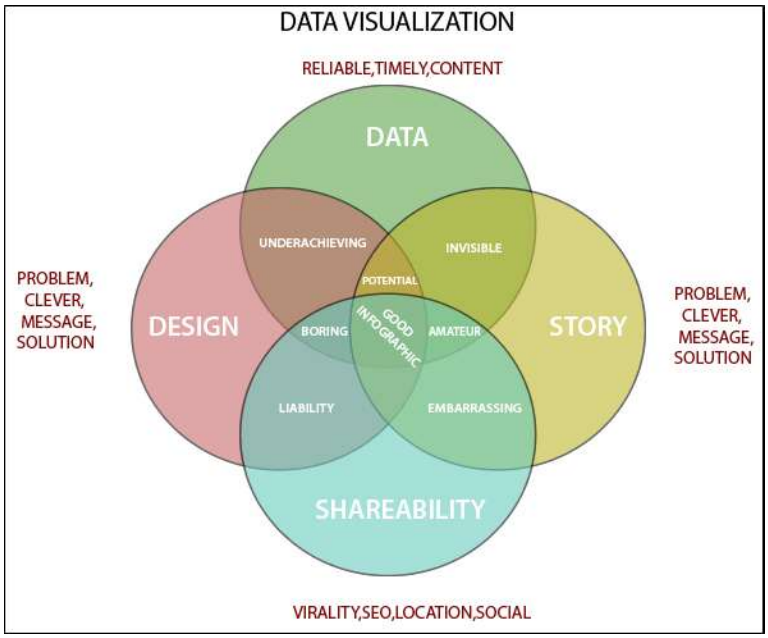


Fig. Data Visualization

Data visualization is the graphical representation of information and data. By using visual elements like charts, graphs, and maps, data visualization tools provide an accessible way to see and understand trends, outliers, and patterns in data. In the world of Big Data, data visualization tools and technologies are essential to analyze massive amounts of information and make data-driven decisions.

The benefits of data visualization:

When considering business strategies and goals, data visualization benefits decision makers in several ways to improve data insights. Let's explore seven major benefits in detail:

- Better analysis
- Quick action
- Identifying patterns
- Finding errors
- Understanding the story
- Exploring business insights
- Grasping the Latest Trends

Better analysis:

Data visualization helps business stakeholders analyze reports regarding sales, marketing strategies, and product interest. Based on the analysis, they can focus on the areas that require attention to increase profits, which in turn makes the business more productive.

Quick action:

As mentioned previously, the human brain grasps visuals more easily than table reports. Data visualizations allow decision makers to be notified quickly of new data insights and take necessary actions for business growth.

Identifying patterns:

Large amounts of complicated data can provide many opportunities for insights when we visualize them. Visualization allows business users to recognize relationships between the data, providing greater meaning to it. Exploring these patterns helps users focus on specific areas that require attention in the data, so that they can identify the significance of those areas to drive their business forward.

Finding errors:

Visualizing your data helps quickly identify any errors in the data. If the data tends to suggest the wrong actions, visualizations help identify erroneous data sooner so that it can be removed from analysis.

Understanding the story:

Storytelling is the purpose of your dashboard. By designing your visuals in a meaningful way, you help the target audience grasp the story in a single glance. Always be sure to convey the story in the simplest way, without excessive complicated visuals.

Exploring business insights:

In the current competitive business environment, finding data correlations using visual representations is key to identifying business insights. Exploring these insights is important for business users or executives to set the right path to achieving the business' goals.

Grasping the latest trends:

Using data visualization, you can discover the latest trends in your business to provide quality products and identify problems before they arise. Staying on top of trends, you can put more effort into increasing profits for your business.

How data visualization works:

Data visualization involves handling tons of data that will be converted into meaningful visuals using widgets. To achieve this, we require the best software tools to operate various types of data sources such as files, web API data, database-maintained sources, and others. Organizations should choose the best data visualization tool to meet all their requirements.

At a minimum, the tool should support interactive visual creation, flexible connectivity to data sources, combining data sources, automatic refresh of data, sharing visuals with others, secured access to data sources, and exporting widgets. These features allow you to make the best visuals of your data and also save your business time.

Trusted, real-time data visualization:

You need a way to quickly pivot your company's efforts in response to world and customer-evolving expectations. You also need a way to make these quick business decisions using big data. But big data has been increasing in volume-becoming even bigger data. As a result, the massive amount of data is slow to sort through, comprehend and especially explain. And, if you can pull outcomes from disparate sources, it isn't easy to interpret their numerical outputs.

Libraries Used:

1. Seaborn: Seaborn is a data visualization library built on top of matplotlib and closely integrated with pandas data structures in Python.

Conclusion:

In this experiment we have studied about what is data visualization. How data visualization helps us in different ways. We have also used the inbuilt data set i.r., Titanic from the seaborn liabrary to perform data visualization on it.

In [2]:
```
import seaborn as sns
df= sns.load_dataset('titanic')
```

In [3]:
```
df
```

Out[3]:

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

891 rows × 15 columns

In [4]:
```
df=df[['survived','class','sex','age','fare']]
```

In [5]:
```
df
```

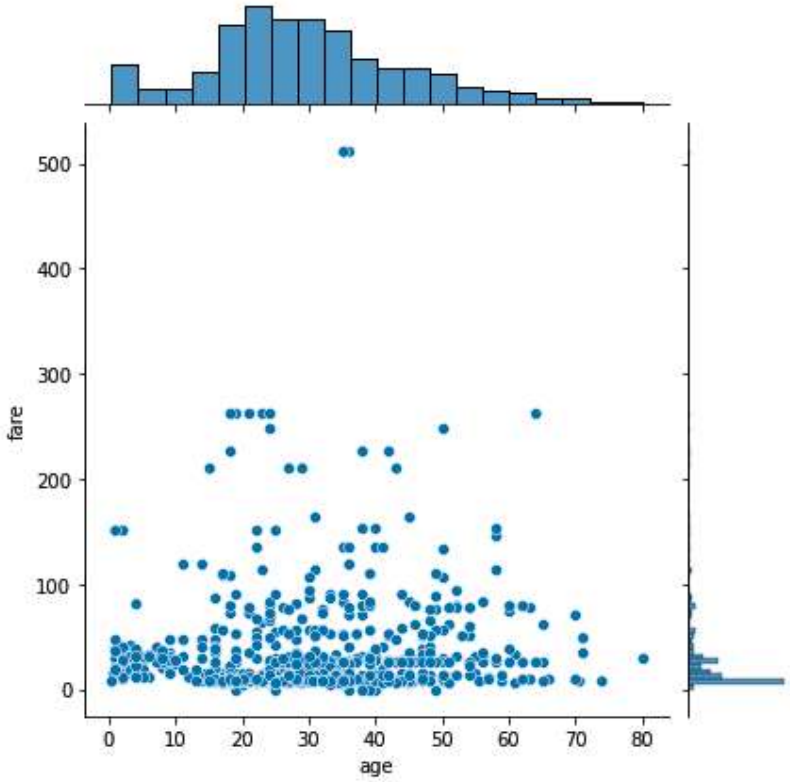
Out[5]:

	survived	class	sex	age	fare
0	0	Third	male	22.0	7.2500
1	1	First	female	38.0	71.2833
2	1	Third	female	26.0	7.9250
3	1	First	female	35.0	53.1000
4	0	Third	male	35.0	8.0500
...	...	...	...	...	...
886	0	Second	male	27.0	13.0000
887	1	First	female	19.0	30.0000
888	0	Third	female	NaN	23.4500
889	1	First	male	26.0	30.0000
890	0	Third	male	32.0	7.7500

891 rows × 5 columns

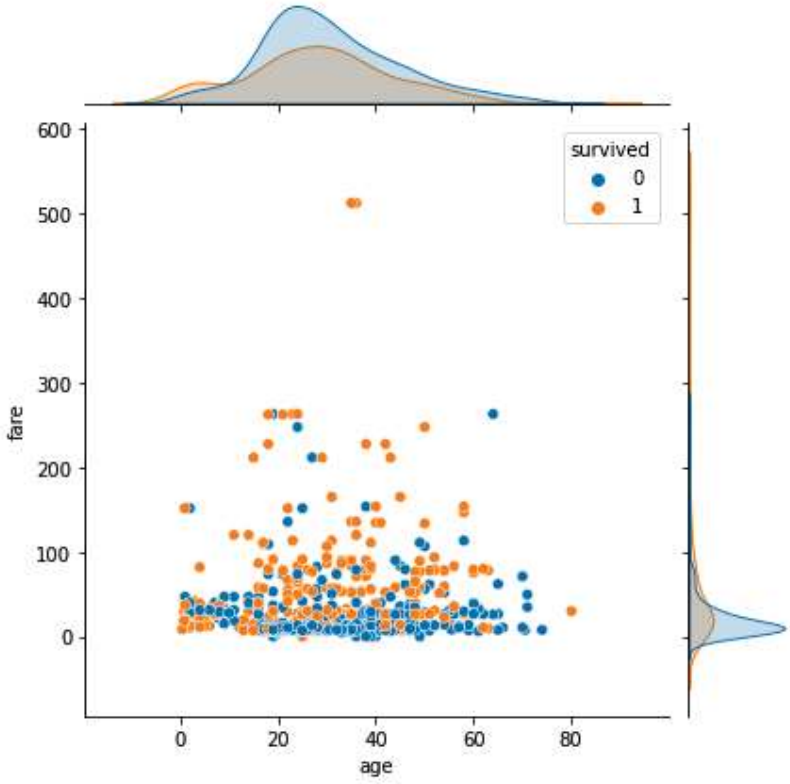
In [6]:
```
sns.jointplot(x='age',y='fare',data=df)
```

Out[6]: <seaborn.axisgrid.JointGrid at 0x1e32202c700>



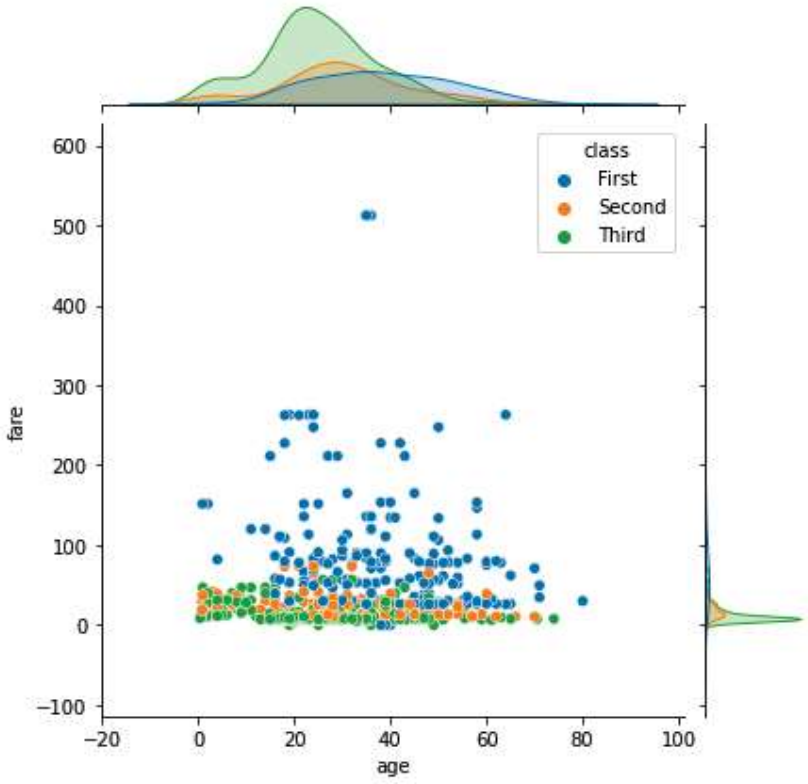
In [7]:
```
sns.jointplot(x='age',y='fare',data=df,hue='survived')
```

Out[7]: <seaborn.axisgrid.JointGrid at 0x1e322a5dfd0>



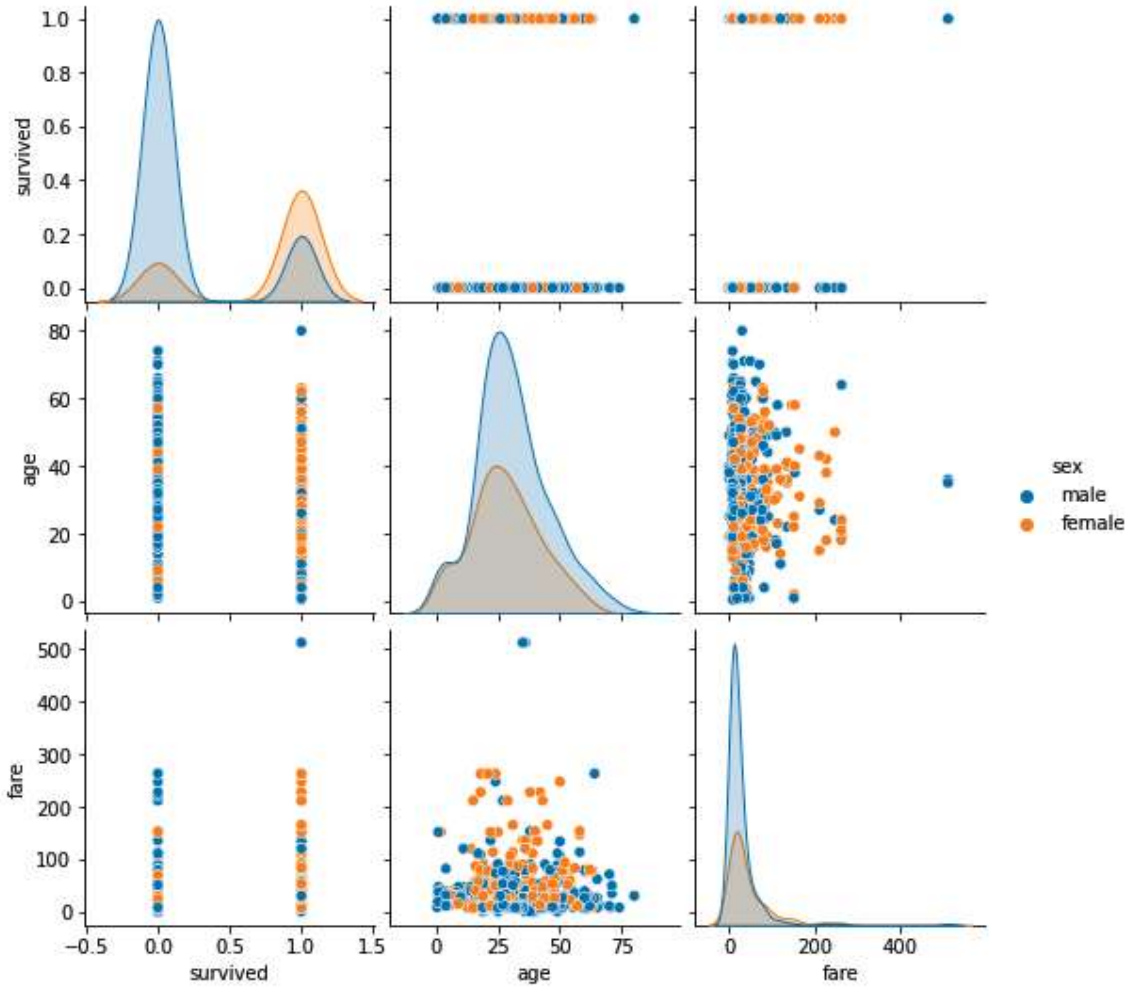
In [8]:
```
sns.jointplot(x='age',y='fare',data=df,hue='class')
```

Out[8]:
```
<seaborn.axisgrid.JointGrid at 0x1e322b88550>
```



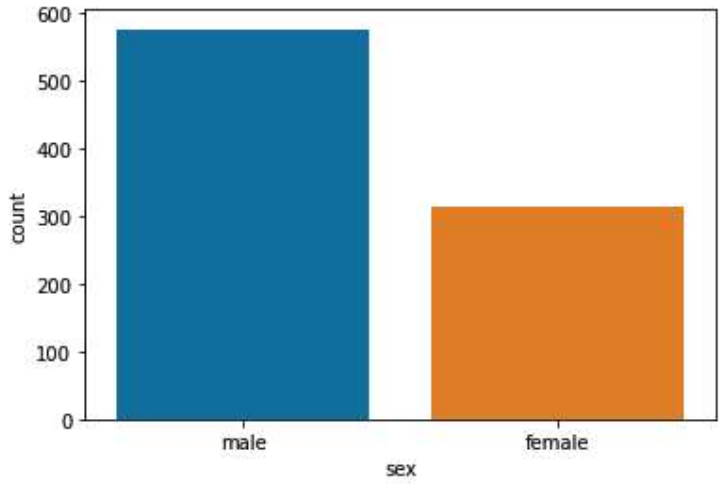
In [9]:
```
sns.pairplot(df,hue='sex')
```

Out[9]:
```
<seaborn.axisgrid.PairGrid at 0x1e322c6b7c0>
```



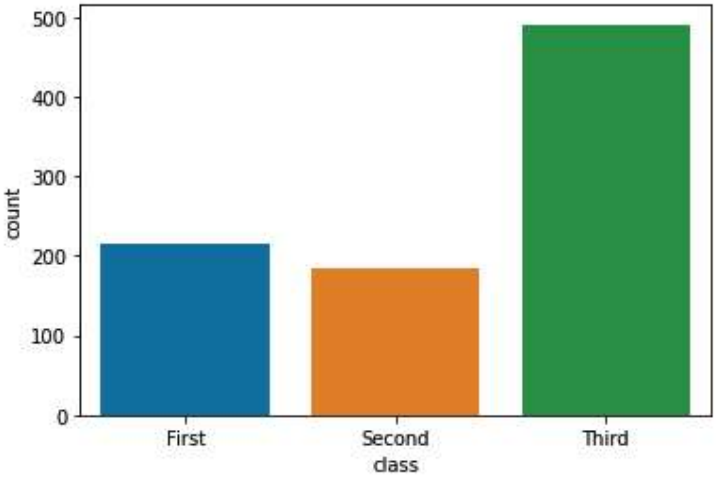
In [10]:
```
sns.countplot(x=df['sex'])
```

Out[10]: <AxesSubplot:xlabel='sex', ylabel='count'>



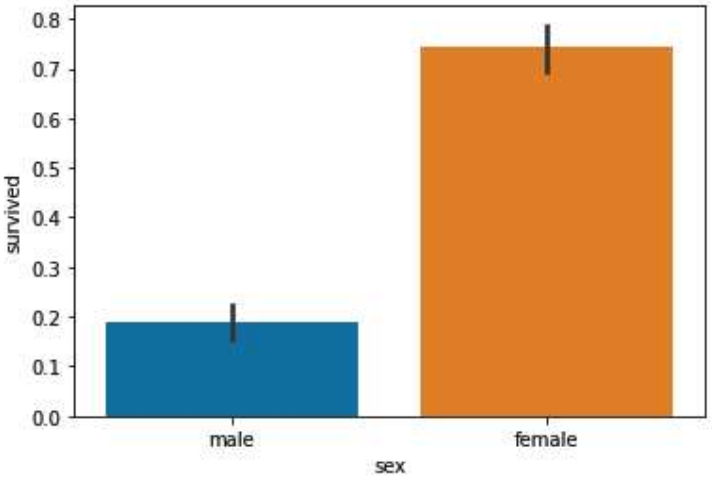
In [11]:
```
sns.countplot(x=df['class'])
```

Out[11]: <AxesSubplot:xlabel='class', ylabel='count'>



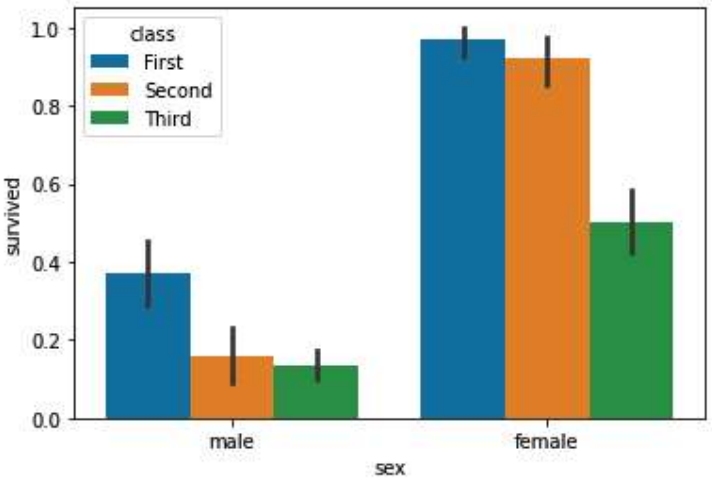
In [12]:
```
sns.barplot(x='sex',y='survived',data=df)
```

Out[12]: <AxesSubplot:xlabel='sex', ylabel='survived'>



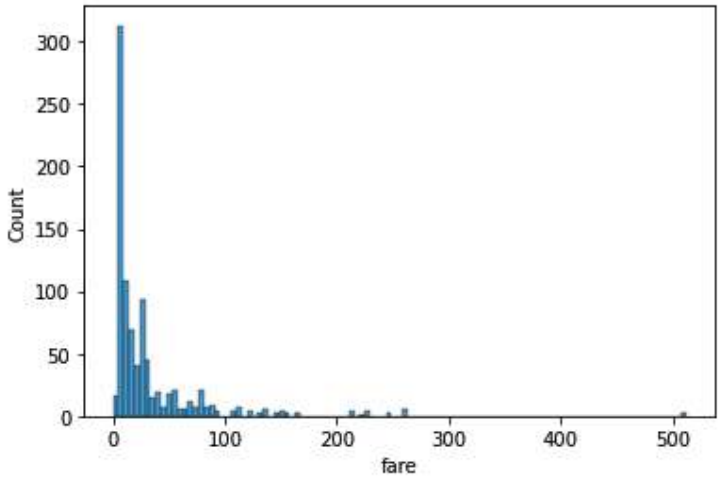
In [13]:
```
sns.barplot(x='sex',y='survived',hue='class',data=df)
```

Out[13]: <AxesSubplot:xlabel='sex', ylabel='survived'>



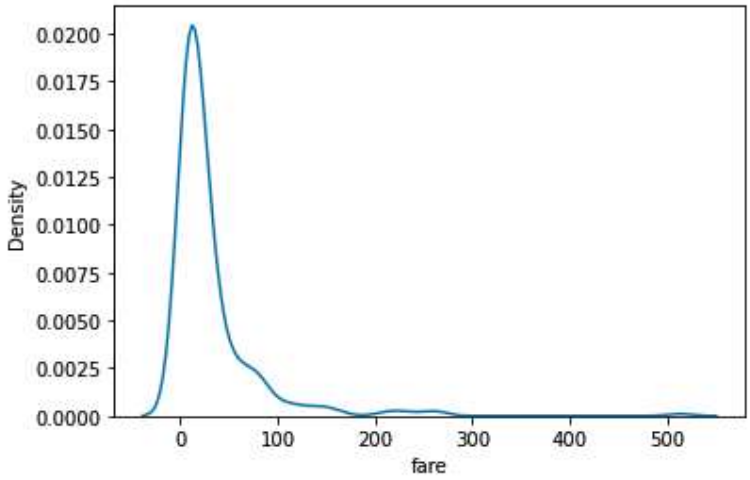
In [14]:
```
sns.histplot(df['fare'])
```

Out[14]: <AxesSubplot:xlabel='fare', ylabel='Count'>



In [15]:
```
sns.kdeplot(df['fare'])
```

Out[15]: <AxesSubplot:xlabel='fare', ylabel='Density'>



In [ ]:

Experiment No. 9

Aim: Use the inbuilt dataset 'Titanic' as used in previous experiment . Plot a box plot for distribution of age with respect to each gender along with the information anout wheather they survived or not.(Cloumn name 'sex' and 'age')

Requirement:

- Anaconda Installer
- Windows 10 OS
- Linux
- Jupyter Notebook

Theory:

What is Data Visualization:

Data visualization is defined as a graphical representation that contains the information and the data. By using visual elements like charts, graphs, and maps, data visualization techniques provide an accessible way to see and understand trends, outliers, and patterns in data.

In modern days we have a lot of data in our hands i.e, in the world of Big Data, data visualization tools, and technologies are crucial to analyze massive amounts of information and make data-driven decisions.

It is used in many areas such as:

• To model complex events.
• Visualize phenomenons that cannot be observed directly, such as weather patterns, medical conditions, or mathematical relationships.

Data Visualization Technique:

1. Box and Whisker Plot

• This plot can be used to obtain more statistical details about the data.
• The straight lines at the maximum and minimum are also called whiskers.
• Points that lie outside the whiskers will be considered as an outlier.
• The box plot also gives us a description of the 25th, 50th,75th quartiles.
• With the help of a box plot, we can also determine the Interquartile range(IQR) where maximum details of the data will be present. Therefore, it can also give us a clear idea about the outliers in the dataset.

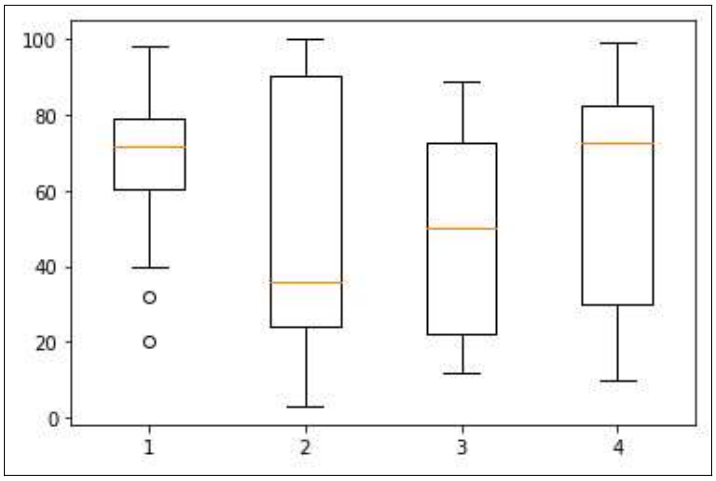


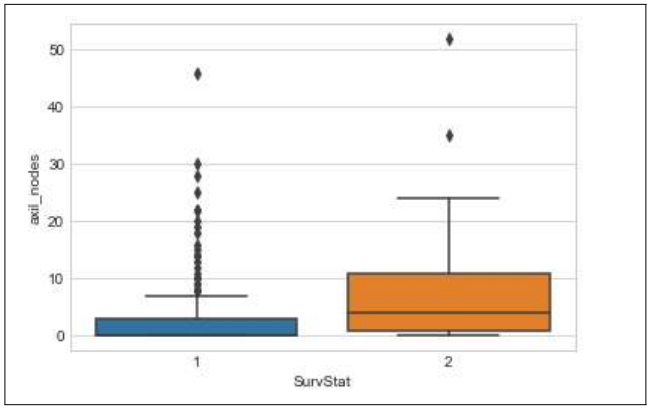
Fig. General Diagram for a Box-plot

Implementation:

- Boxplot is available in the Seaborn library.
- Here x is considered as the dependent variable and y is considered as the independent variable. These box plots come under univariate analysis, which means that we are exploring data only with one variable.
- Here we are trying to check the impact of a feature named “axil_nodes” on the class named “Survival status” and not between any two independent features.

The code snippet is as follows:

```
sns.boxplot(x='SurvStat',y='axil_nodes',data=hb)
```



Libraries Used:

Seaborn: Seaborn is a data visualization library built on top of matplotlib and closely integrated with pandas data structures in Python.

Conclusion: In this experiment we have studied about the data visualization technique and implemented data visualization on the built in data set in seaborn laibrary i.e., titanic dataset

In [2]:
```
import seaborn as sns
```

In [3]:
```
df = sns. load_dataset('titanic')
```

In [6]:
```
df= df[['sex','age','survived']]
```

In [7]:
```
df
```

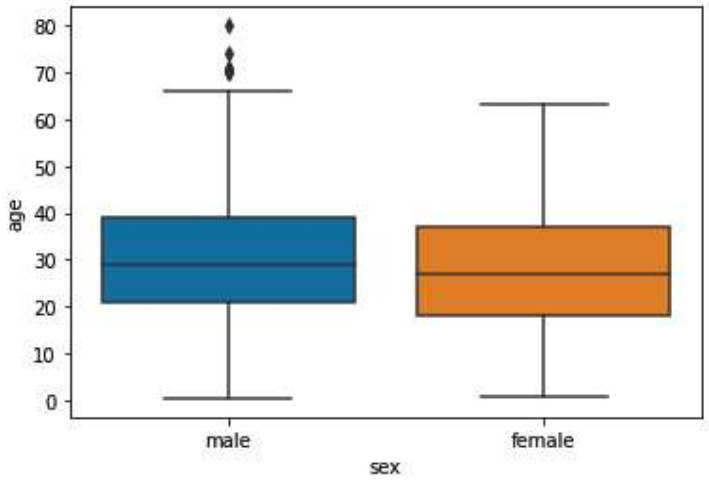
Out[7]:

	sex	age	survived
0	male	22.0	0
1	female	38.0	1
2	female	26.0	1
3	female	35.0	1
4	male	35.0	0
...	...	...	...
886	male	27.0	0
887	female	19.0	1
888	female	NaN	0
889	male	26.0	1
890	male	32.0	0

891 rows × 3 columns

In [8]:
```
sns.boxplot(x='sex',y='age',data=df)
```

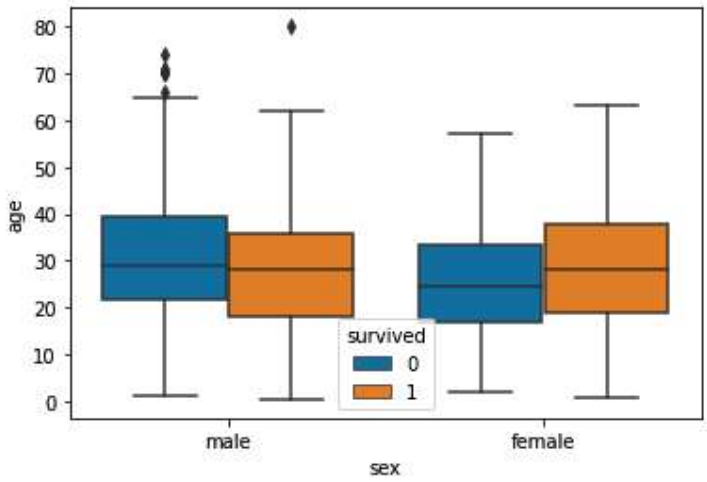
Out[8]:
```
<AxesSubplot:xlabel='sex', ylabel='age'>
```



In [9]:
```
sns.boxplot(x='sex',y='age',hue='survived', data=df)
```

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

Out[9]:



In [ ]:

Experiment No. 10

Aim:

Download the iris flower dataset or any other dataset intoa dataframe. Scan the dataset and given the interference as

1) List down the feature and types
2) Create histogram for each feature in the dataset
3) Create a boxplot for each seature in the dataset
4) Compare distributions and identify outliers.

Requirement:

- Anaconda Installer
- Windows 10 OS
- Linux
- Jupyter Notebook

Theory:

What is Data Visualization?

Data visualization simply refers to techniques that are used to communicate with insights from data through a visual representation. To simplify the thing, you can say that data visualization visually puts data. So we can easily understand it. One has to master different data visualization tools to become effective.

The main goal of data visualization is to put large datasets into visual graphics. And it is one of the important steps when it comes to data science. Also, it is a simple way to track different data points. No matter if you wish to track website metrics, sales team performance, marketing campaign, product adoption rate, or any other thing.

Data visualization is what will help you out. To learn more about data visualization and other visual representation in data science, check out our data science certifications from recognized universities.

Categorical Data:

Categorical data can be:

- nominal, qualitative
- ordinal

For visualization, the main difference is that ordinal data suggests a particular display order.Purely categorical data can come in a range of formats. The most common are:

- raw data: individual observations;
- aggregated data: counts for each unique combination of levels
- cross-tabulated data

Working With Categorical Variables:

Categorical variables are usually represented as:

- character vectors
- factors.

Some advantages of factors:

- more control over ordering of levels
- levels are preserved when forming subsets

Most plotting and modeling functions will convert character vectors to factors with levels ordered alphabetically. Some standard R functions for working with factors include:

- factor creates a factor from another type of variable
- levels returns the levels of a factor
- reorder changes level order to match another variable
- relevel moves a particular level to the first position as a base line
- droplevels removes levels not in the variable.

The tidyverse package forcats adds some more tools, including,

- fct_inorder creates a factor with levels ordered by first appearance
- fct_infreq orders levels by decreasing frequency
- fct_rev reverses the levels
- fct_recode changes factor levels
- fct_relevel moves one or more levels
- fct_c merges two or more factors

Seaborn:

Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics. To see the code or report a bug, please visit the GitHub repository. General support questions are most at home on stackoverflow or discourse, which have dedicated channels for seaborn.

Why data visualization is important for any career:

It's hard to think of a professional industry that doesn't benefit from making data more understandable. Every STEM field benefits from understanding data—and so do fields in government, finance, marketing, history, consumer goods, service industries, education, sports, and so on. While we'll always wax poetically about data visualization (you're on the Tableau website, after all) there are practical, real-life applications that are undeniable. And, since visualization is so prolific, it's also one of the most useful professional skills to develop.

The better you can convey your points visually, whether in a dashboard or a slide deck, the better you can leverage that information. The concept of the citizen data scientist is on the rise. Skill sets are changing to accommodate a data-driven world. It is increasingly valuable for professionals to be able to use data to make decisions and use visuals to tell stories of when data informs the who, what, when, where, and how. While traditional education typically draws a distinct line between creative storytelling and technical analysis, the modern professional world also values those who can cross between the two: data visualization sits right in the middle of analysis and visual storytelling.

Libraries Used:

1. Seaborn: Seaborn is a data visualization library built on top of matplotlib and closely integrated with pandas data structures in Python.
2. Pandas: Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring and manipulating data.
3. Numpy: is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed.
4. Sklearn: It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python.

Conclusion:

In this experiment we have studied about the data visualization and performed data visualization on the iris data set in different ways.

In [2]:
```
import seaborn as sns
```

In [3]:
```
df =sns.load_dataset('iris')
```

In [4]:
```
df
```

Out[4]:

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

150 rows × 5 columns

In [6]:
```
#list down there features and tere types available in dataset
df.columns
```

Out[6]:
```
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
       'species'],
      dtype='object')
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

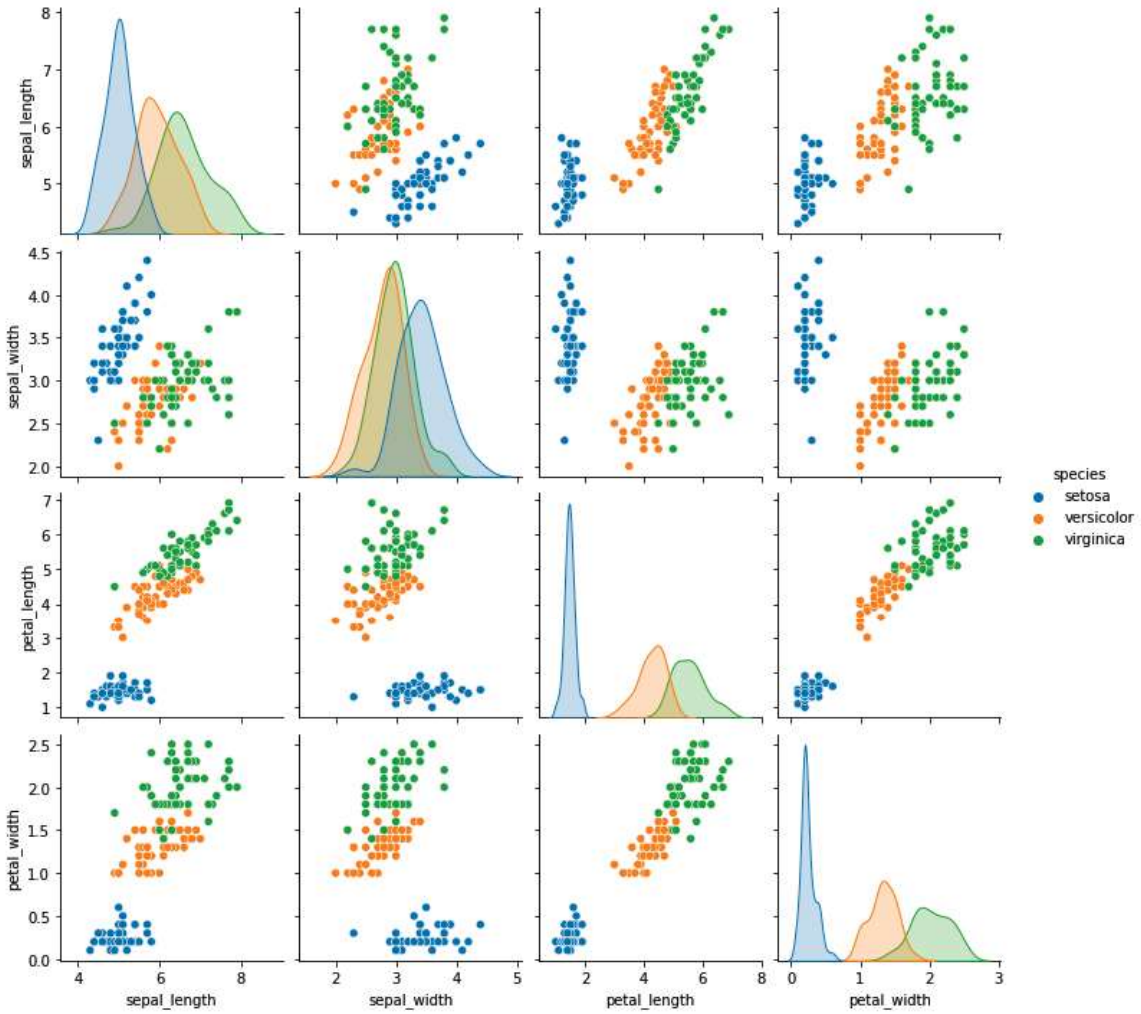
In [8]:
```
df.dtypes
```

Out[8]:
```
sepal_length    float64
sepal_width     float64
petal_length    float64
```

```
petal_width     float64
species          object
dtype: object
```

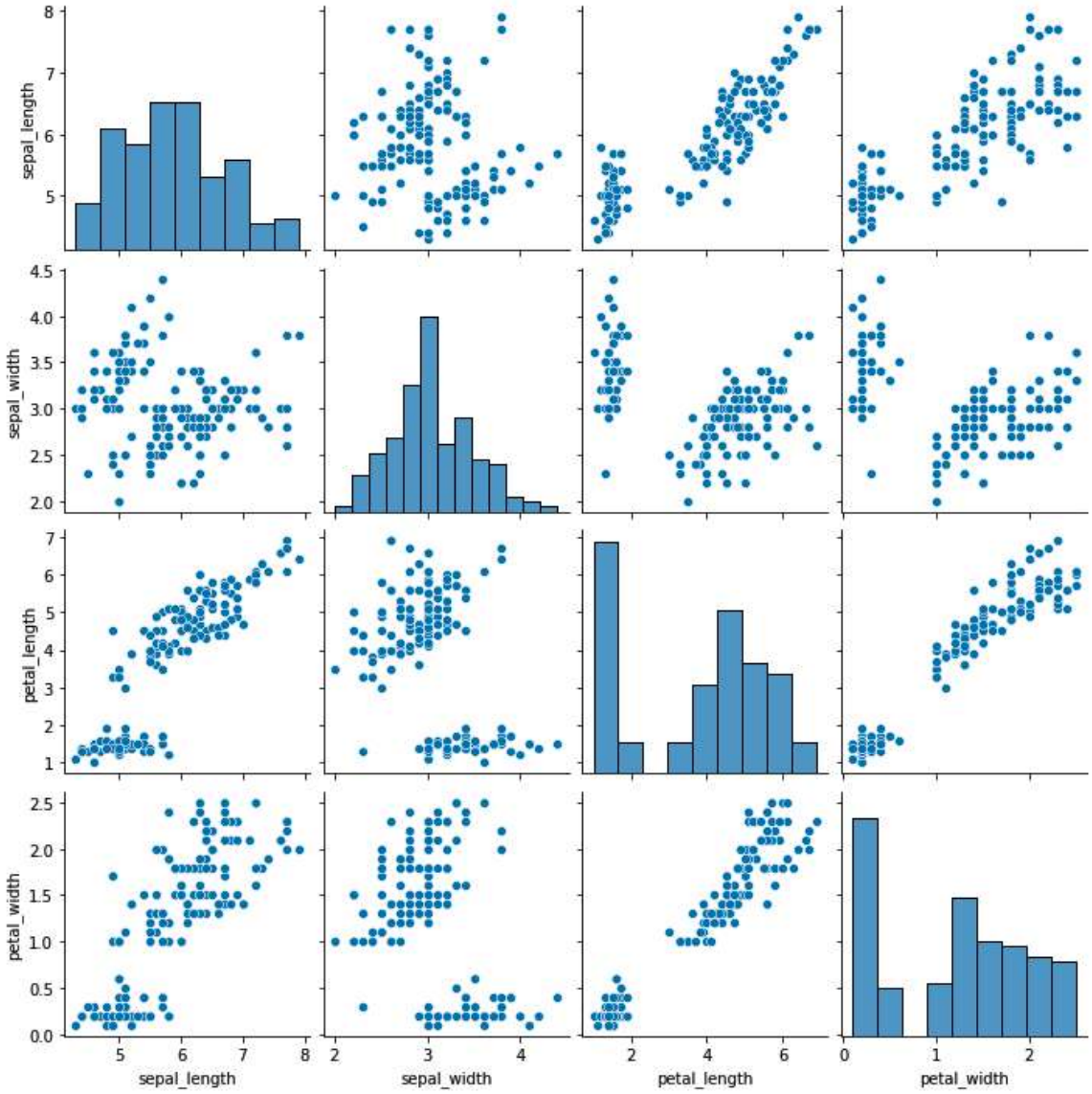
In [10]:
```
sns.pairplot(df,hue='species')
```

Out[10]: <seaborn.axisgrid.PairGrid at 0x1f2f00c93d0>



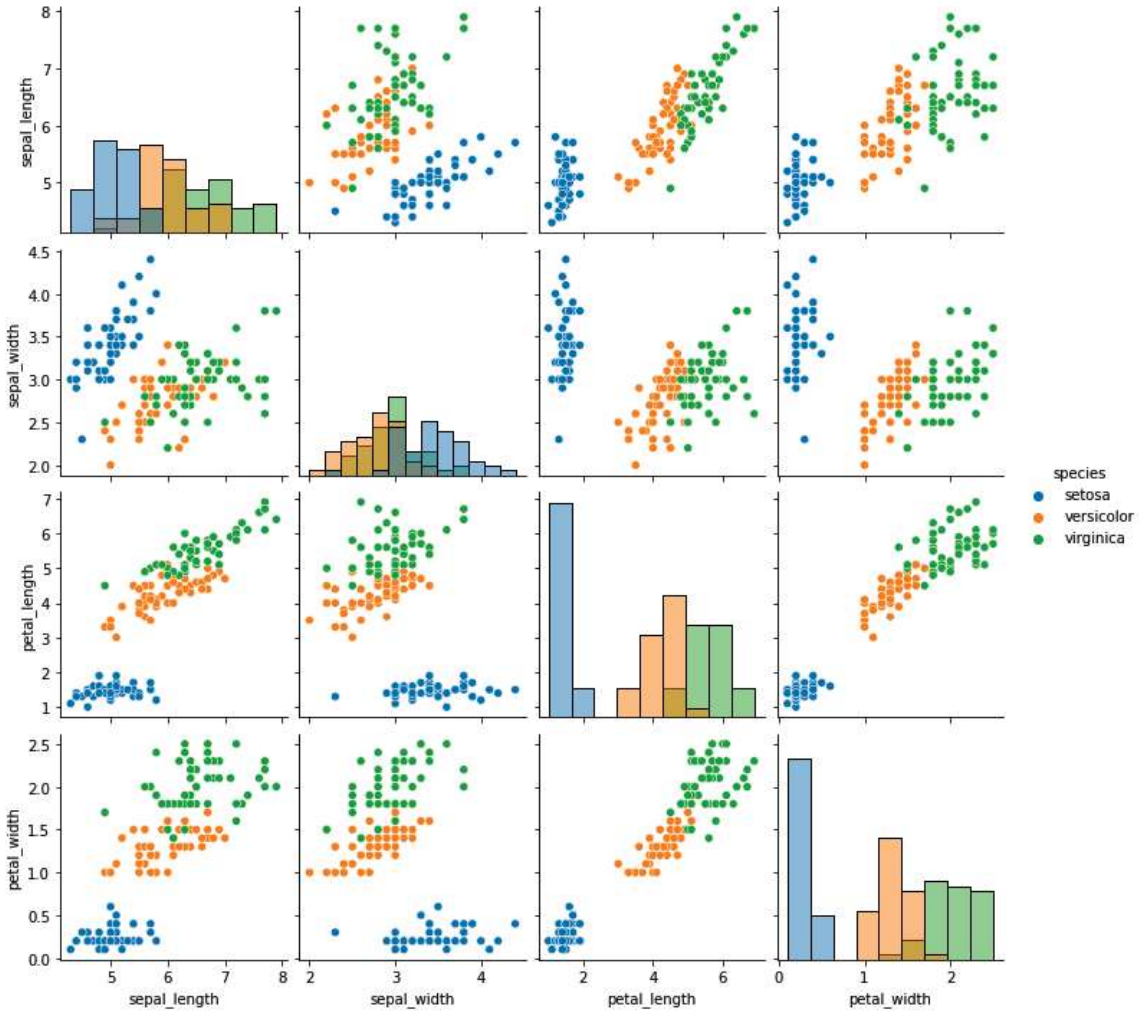
In [11]:
```
sns.pairplot(df)
```

Out[11]: <seaborn.axisgrid.PairGrid at 0x1f2f0371490>



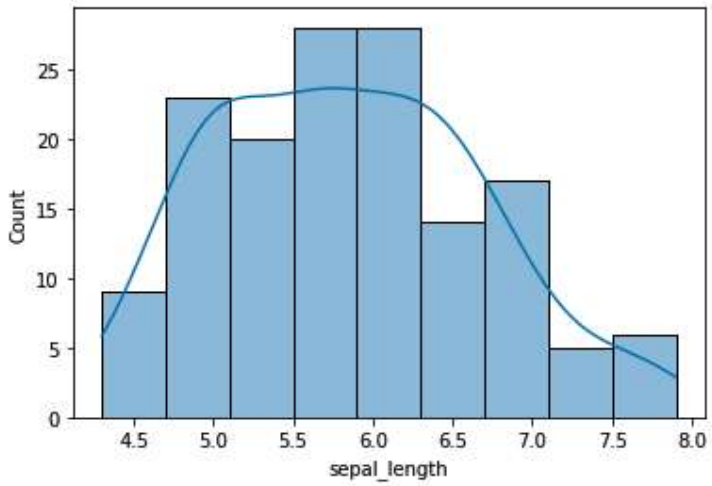
In [12]:
```
sns.pairplot(df,hue='species',diag_kind='hist')
```

Out[12]:
```
<seaborn.axisgrid.PairGrid at 0x1f2f03caaf0>
```



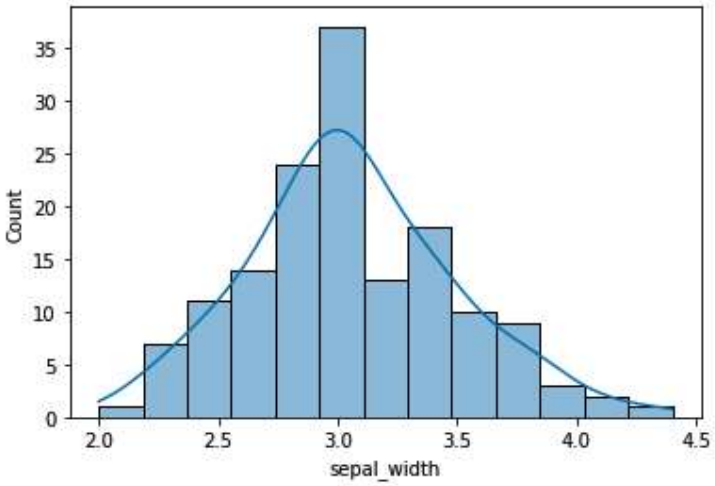
In [13]:
```
sns.histplot(df['sepal_length'],kde=True)
```

Out[13]: <AxesSubplot:xlabel='sepal_length', ylabel='Count'>



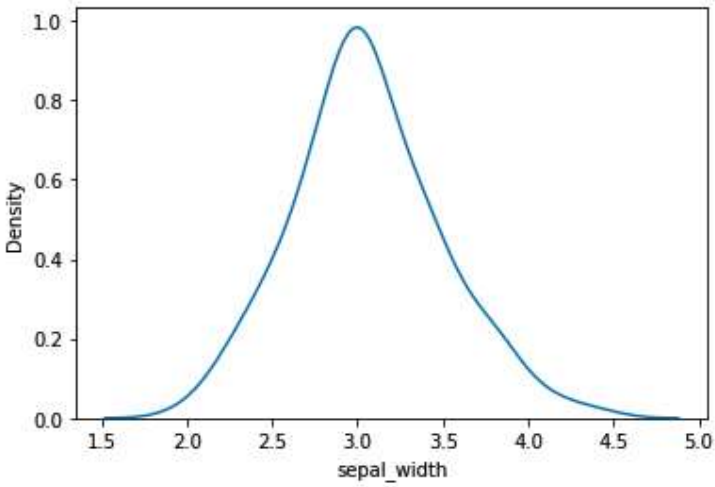
In [14]:
```
sns.histplot(df['sepal_width'],kde=True)
```

Out[14]: <AxesSubplot:xlabel='sepal_width', ylabel='Count'>



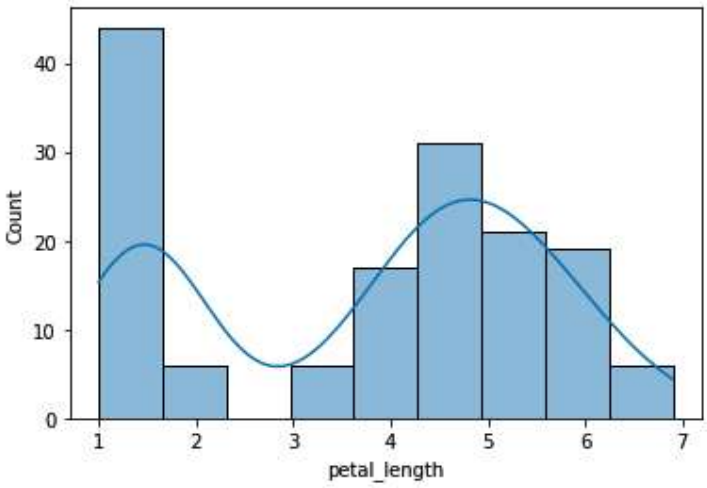
In [16]:
```
sns.kdeplot(df['sepal_width'])
```

Out[16]: <AxesSubplot:xlabel='sepal_width', ylabel='Density'>



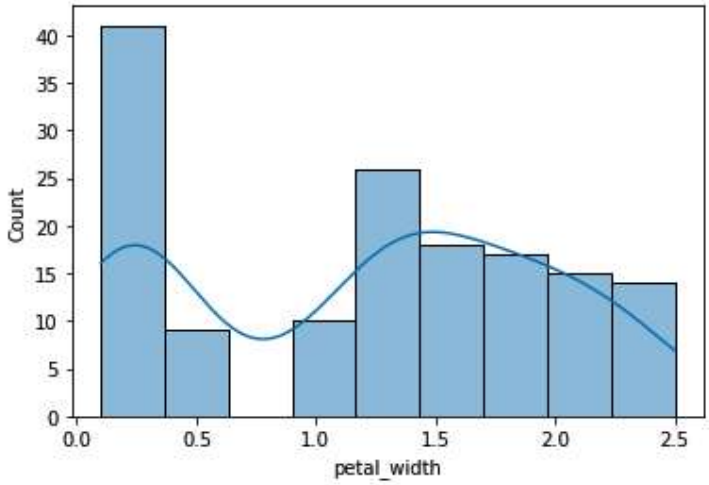
In [17]:
```
sns.histplot(df['petal_length'],kde=True)
```

Out[17]: <AxesSubplot:xlabel='petal_length', ylabel='Count'>



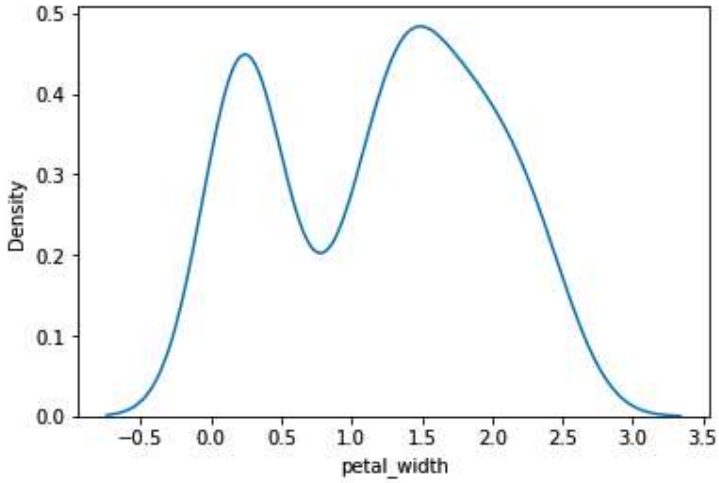
In [18]:
```
sns.histplot(df['petal_width'],kde=True)
```

Out[18]: <AxesSubplot:xlabel='petal_width', ylabel='Count'>

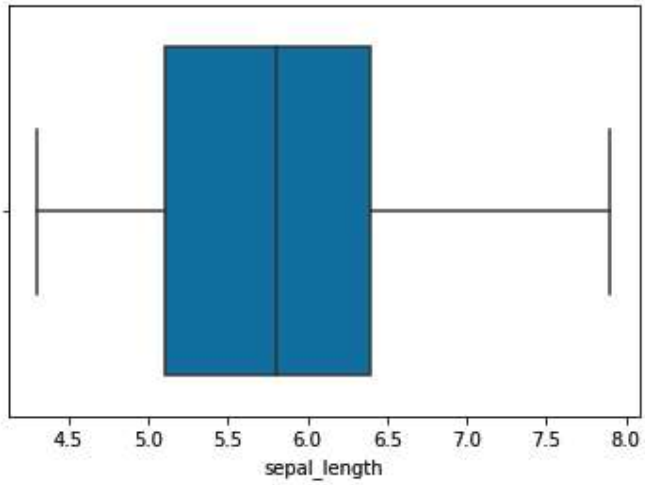


In [19]:
```
sns.kdeplot(df['petal_width'])
```

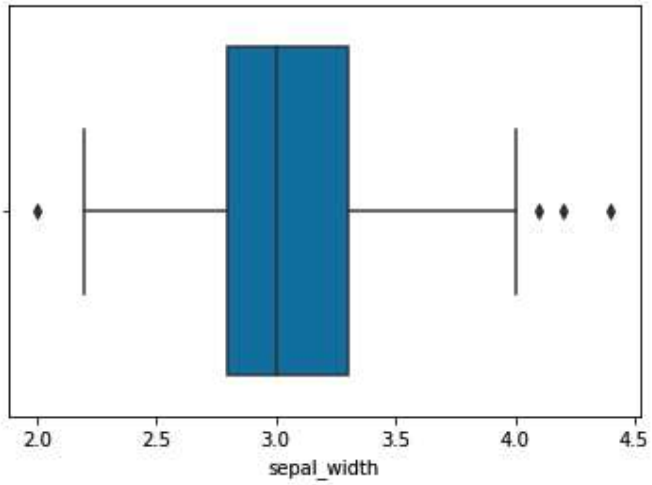
Out[19]: <AxesSubplot:xlabel='petal_width', ylabel='Density'>



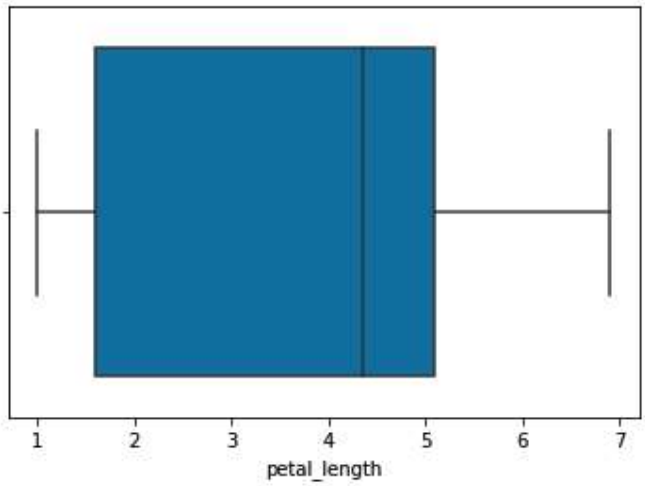
In [20]:
```
sns.boxplot(x=df['sepal_length']);
```



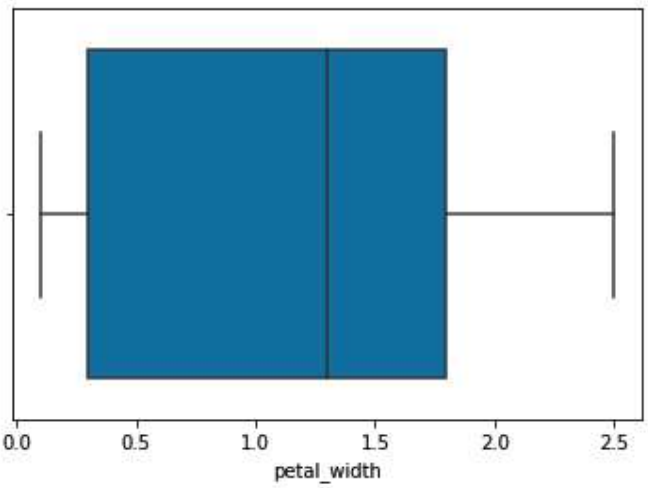
In [21]:
```
sns.boxplot(x=df['sepal_width']);
```



In [22]:
```
sns.boxplot(x=df['petal_length']);
```



In [23]:
```
sns.boxplot(x=df['petal_width']);
```



In [ ]:
```

```

Experiment No. 11 (Mini Project)

Aim:

Use the following dataset and classify tweets into positive and negative tweets. https://www.kaggle.com/ruchi798/data-science-tweet

---

Requirement:

- Anaconda Installer
- Windows 10 OS
- Jupyter Notebook

Theory:

What is Sentiment Analysis?

Sentiment Analysis is the process of 'computationally' determining whether a piece of writing is positive, negative or neutral. It's also known as **opinion mining**, deriving the opinion or attitude of a speaker.

Why Sentiment Analysis?

- Business: In marketing field companies use it to develop their strategies, to understand customers' feelings towards products or brand, how people respond to their campaigns or product launches and why consumers don't buy some products.
- Politics: In political field, it is used to keep track of political view, to detect consistency and inconsistency between statements and actions at the government level. It can be used to predict election results as well!
- Public Actions: Sentiment analysis also is used to monitor and analyse social phenomena, for the spotting of potentially dangerous situations and determining the general mood of the blogosphere.

Libraries Used:

1. Pandas: Pandas is an open source Python package that is most widely used for data science/data analysis and machine learning tasks. It is built on top of another package named Numpy, which provides support for multi-dimensional arrays.

2. String: The string module contains a number of functions to process standard Python strings.

3. Sklearn: It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python.

Conslusion:

Hence, we successfully implemented sentiment analysis using python.

In [1]:
```
import pandas as pd
```

In [4]:
```
df=pd.read_csv('tweets.csv')
```

In [5]:
```
df.shape
```

Out[5]: (31962, 3)

In [6]:
```
df
```

Out[6]:

	id	label	tweet
0	1	0	@user when a father is dysfunctional and is s...
1	2	0	@user @user thanks for #lyft credit i can't us...
2	3	0	bihday your majesty
3	4	0	#model i love u take with u all the time in ...
4	5	0	factsguide: society now #motivation
...	...	...	...
31957	31958	0	ate @user isz that youuu?ð ð ð ð ð...
31958	31959	0	to see nina turner on the airwaves trying to...
31959	31960	0	listening to sad songs on a monday morning otw...
31960	31961	1	@user #sikh #temple vandalised in in #calgary,...
31961	31962	0	thank you @user for you follow

31962 rows × 3 columns

In [7]:
```
df=pd.read_csv('tweets.csv',nrows=10000)
```

In [8]:
```
df
```

Out[8]:

	id	label	tweet
0	1	0	@user when a father is dysfunctional and is s...
1	2	0	@user @user thanks for #lyft credit i can't us...
2	3	0	bihday your majesty
3	4	0	#model i love u take with u all the time in ...
4	5	0	factsguide: society now #motivation
...	...	...	...
9995	9996	0	@user my routine is out of whack! evening wal...
9996	9997	0	i'm dead but still happy #poledance #madrid ##...
9997	9998	0	â #united kingdom claimant count rate up to...

	id	label	tweet
9998	9999	0	rip my friend ð ¢ð ¢ #shocked #dismay #hea...
9999	10000	0	how to open... your , loving hea #thursdayth...

10000 rows × 3 columns

In [9]:
```
df.shape
```

Out[9]:
```
(10000, 3)
```

In [10]:
```
df['tweets_len']=df['tweet'].apply(lambda x : len(x))
```

In [11]:
```
df
```

Out[11]:

	id	label	tweet	tweets_len
0	1	0	@user when a father is dysfunctional and is s...	102
1	2	0	@user @user thanks for #lyft credit i can't us...	122
2	3	0	bihday your majesty	21
3	4	0	#model i love u take with u all the time in ...	86
4	5	0	factsguide: society now #motivation	39
...	...	...	...	...
9995	9996	0	@user my routine is out of whack! evening wal...	120
9996	9997	0	i'm dead but still happy #poledance #madrid ##...	90
9997	9998	0	â #united kingdom claimant count rate up to...	106
9998	9999	0	rip my friend ð ¢ð ¢ #shocked #dismay #hea...	102
9999	10000	0	how to open... your , loving hea #thursdayth...	78

10000 rows × 4 columns

In [65]:
```
sent='Hii , Where are you ?'
```

In [66]:
```
import string
```

In [67]:
```
string.punctuation
```

Out[67]:
```
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

In [68]:
```
count=sum([1 for x in sent if x in string.punctuation])
```

In [69]:
```
per=count/(len(sent)-sent.count(' '))
```

In [70]:
```
per
```

Out[70]: 0.125

In [71]:
```
import string
```

In [72]:
```
string.punctuation
```

Out[72]: '!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'

In [73]:
```
def count_punct(sent):
    count =sum([1 for x in sent if x in string.punctuation])
    p=round(count/(len(sent)-sent.count(' '))*100,2)
    return p
```

In [74]:
```
count_punct(sent)
```

Out[74]: 12.5

In [75]:
```
df['punct%']=df['tweet'].apply(lambda x:count_punct(x))
```

In [84]:
```
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
ps=PorterStemmer()
```

In [85]:
```
s_words=stopwords.words('english')
```

In [86]:
```
#analyser funtion
def clean_text(text):
    data=[x for x in text if x not in string.punctuation]
    data=''.join(data)
    data=[ps.stem(x) for x in data.split() if x not in s_words]
    return data
```

In [87]:
```
clean_text(sent)
```

Out[87]: ['hii', 'where']

In [88]:
```
# inputdata
X=df.drop(['label','id'],axis=1)
# output data
y=df['label']
```

In [89]:
```
X
```

Out[89]:

	tweet	tweets_len	punct%
0	@user when a father is dysfunctional and is s...	102	3.66

	tweet	tweets_len	punct%
1	@user @user thanks for #lyft credit i can't us...	122	7.92
2	bihday your majesty	21	0.00
3	#model i love u take with u all the time in ...	86	5.71
4	factsguide: society now #motivation	39	6.25
...	...	...	...
9995	@user my routine is out of whack! evening wal...	120	11.22
9996	i'm dead but still happy #poledance #madrid ##...	90	11.84
9997	â #united kingdom claimant count rate up to...	106	10.47
9998	rip my friend ð ¢ð ¢ #shocked #dismay #hea...	102	7.95
9999	how to open... your , loving hea #thursdayth...	78	10.61

10000 rows × 3 columns

In [90]:
```
y
```

Out[90]:
```
0       0
1       0
2       0
3       0
4       0
       ..
9995    0
9996    0
9997    0
9998    0
9999    0
Name: label, Length: 10000, dtype: int64
```

In [92]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf=TfidfVectorizer(analyzer=clean_text)
X_trans=tfidf.fit_transform(X['tweet'])
```

In [93]:
```
X_trans.shape
```

Out[93]:
```
(10000, 18712)
```

In [97]:
```
X_vect=pd.concat([X[['tweets_len','punct%']].reset_index(drop=True),pd.DataFrame(X_t
```

In [98]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X_vect,y,stratify=y,random_state=0)
```

In [99]:
```
from sklearn.linear_model import LogisticRegression
clf=LogisticRegression()
clf.fit(X_train,y_train)
```

```
C:\Users\ganes\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Co
nvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

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

Out[99]:
```
LogisticRegression()
```

In [101…
```
y_pred=clf.predict(X_test)
```

In [102…
```
from sklearn.metrics import accuracy_score
```

In [103…
```
accuracy_score(y_test,y_pred)
```

Out[103…
```
0.9336
```

In [104…
```
accuracy_score(y_test,y_pred)*100
```

Out[104…
```
93.36
```

In [ ]:
```

```

Experiment No. 12 (Mini Project)

Aim:

Use the following covid_vaccine_statewise.csv dataset and perform following analytics on given dataset.
https://www.kaggle.com/sudalairajkumar/covid19-in-india?select=covid_vaccine_statewise.csv

a. Describe the dataset
b. Number of persons statewise vaccinated for first dose in India
c. Number of persons statewise vaccinated for second dose in India
d. Number of Males vaccinated
e. Number of Females vaccinated

Requirement:

- Anaconda Installer
- Windows 10 OS
- Jupyter Notebook

Theory:

About this dataset:

Coronaviruses are a large family of viruses which may cause illness in animals or humans. In humans, several coronaviruses are known to cause respiratory infections ranging from the common cold to more severe diseases such as Middle East Respiratory Syndrome (MERS) and Severe Acute Respiratory Syndrome (SARS). The most recently discovered coronavirus causes coronavirus disease COVID-19 - World Health Organization The number of new cases are increasing day by day around the world. This dataset has information from the states and union territories of India at daily level.

State level data comes from Ministry of Health & Family Welfare Testing data and vaccination data comes from covid19india. Huge thanks to them for their efforts! Update on April 20, 2021: Thanks to the Team at ISIBang, I was able to get the historical data for the periods that I missed to collect and updated the csv file.

Libraries Used:

1. Pandas: Pandas is an open source Python package that is most widely used for data science/data analysis and machine learning tasks. It is built on top of another package named Numpy, which provides support for multi-dimensional arrays.

2. NumPy: NumPy can be used to perform a wide variety of mathematical operations on arrays. It adds powerful data structure to python that guarantee efficient calculations with arrays and matrices and it supplies on enormous library of high-level mathematical functions that operate on these arrays and matrices.

Conclusion:

Hence, we successfully implemented mini project using covid_vaccine_statewise.csv dataset.

In [2]:
```
import pandas as pd
```

In [4]:
```
df=pd.read_csv('covid_vaccine_statewise.csv')
```

In [5]:
```
df
```

Out[5]:

	Updated On	State	Total Doses Administered	Sessions	Sites	First Dose Administered	Second Dose Administered	Male (Do Administer
0	16/01/2021	India	48276.0	3455.0	2957.0	48276.0	0.0	N
1	17/01/2021	India	58604.0	8532.0	4954.0	58604.0	0.0	N
2	18/01/2021	India	99449.0	13611.0	6583.0	99449.0	0.0	N
3	19/01/2021	India	195525.0	17855.0	7951.0	195525.0	0.0	N
4	20/01/2021	India	251280.0	25472.0	10504.0	251280.0	0.0	N
...	...	...	...	...	...	...	...	
7840	11/08/2021	West Bengal	NaN	NaN	NaN	NaN	NaN	N
7841	12/08/2021	West Bengal	NaN	NaN	NaN	NaN	NaN	N
7842	13/08/2021	West Bengal	NaN	NaN	NaN	NaN	NaN	N
7843	14/08/2021	West Bengal	NaN	NaN	NaN	NaN	NaN	N
7844	15/08/2021	West Bengal	NaN	NaN	NaN	NaN	NaN	N

7845 rows × 24 columns

In [6]:
```
df.columns
```

Out[6]:
```
Index(['Updated On', 'State', 'Total Doses Administered', 'Sessions',
       ' Sites ', 'First Dose Administered', 'Second Dose Administered',
       'Male (Doses Administered)', 'Female (Doses Administered)',
       'Transgender (Doses Administered)', ' Covaxin (Doses Administered)',
       'CoviShield (Doses Administered)', 'Sputnik V (Doses Administered)',
       'AEFI', '18-44 Years (Doses Administered)',
       '45-60 Years (Doses Administered)', '60+ Years (Doses Administered)',
       '18-44 Years(Individuals Vaccinated)',
       '45-60 Years(Individuals Vaccinated)',
       '60+ Years(Individuals Vaccinated)', 'Male(Individuals Vaccinated)',
       'Female(Individuals Vaccinated)', 'Transgender(Individuals Vaccinated)',
       'Total Individuals Vaccinated'],
      dtype='object')
```

In [9]:
```
# Describe the dataset
df.describe()
```

Out[9]:

	Total Doses Administered	Sessions	Sites	First Dose Administered	Second Dose Administered	Male (Doses Administered)	Adml
count	7.621000e+03	7.621000e+03	7621.000000	7.621000e+03	7.621000e+03	7.461000e+03	7.46
mean	9.188171e+06	4.792358e+05	2282.872064	7.414415e+06	1.773755e+06	3.620156e+06	3.16
std	3.746180e+07	1.911511e+06	7275.973730	2.995209e+07	7.570382e+06	1.737938e+07	1.51
min	7.000000e+00	0.000000e+00	0.000000	7.000000e+00	0.000000e+00	0.000000e+00	2.00
25%	1.356570e+05	6.004000e+03	69.000000	1.166320e+05	1.283100e+04	5.655500e+04	5.21
50%	8.182020e+05	4.547000e+04	597.000000	6.614590e+05	1.388180e+05	3.897850e+05	3.34
75%	6.625243e+06	3.428690e+05	1708.000000	5.387805e+06	1.166434e+06	2.735777e+06	2.56
max	5.132284e+08	3.501031e+07	73933.000000	4.001504e+08	1.130780e+08	2.701636e+08	2.39

8 rows × 22 columns

In [11]:
```
#Number of persons vaccinated first time for india
df[df['State']=='India']
```

Out[11]:

	Updated On	State	Total Doses Administered	Sessions	Sites	First Dose Administered	Second Dose Administered	Male (Doses Administered)
0	16/01/2021	India	48276.0	3455.0	2957.0	48276.0	0.0	NaN
1	17/01/2021	India	58604.0	8532.0	4954.0	58604.0	0.0	NaN
2	18/01/2021	India	99449.0	13611.0	6583.0	99449.0	0.0	NaN
3	19/01/2021	India	195525.0	17855.0	7951.0	195525.0	0.0	NaN
4	20/01/2021	India	251280.0	25472.0	10504.0	251280.0	0.0	NaN
...	...	...	...	...	...	...	...	...
207	11/08/2021	India	NaN	NaN	NaN	NaN	NaN	NaN
208	12/08/2021	India	NaN	NaN	NaN	NaN	NaN	NaN
209	13/08/2021	India	NaN	NaN	NaN	NaN	NaN	NaN
210	14/08/2021	India	NaN	NaN	NaN	NaN	NaN	NaN
211	15/08/2021	India	NaN	NaN	NaN	NaN	NaN	NaN

212 rows × 24 columns

In [12]:
```
import numpy as np
```

In [15]:
```
states=np.unique(df['State'])
```

In [16]:
```
mh=df[df['State']=='Maharashtra']
```

In [18]:
```
mh['First Dose Administered'].sum()
```

Out[18]:
```
2784364331.0
```

In [22]:
```
for state in states:
    temp=df[df['State']==state]
    print(state,temp['First Dose Administered'].sum())
```

```
Andaman and Nicobar Islands 16425854.0
Andhra Pradesh 1232860845.0
Arunachal Pradesh 49004980.0
Assam 585600226.0
Bihar 1470502878.0
Chandigarh 44703105.0
Chhattisgarh 796002902.0
Dadra and Nagar Haveli and Daman and Diu 33595063.0
Delhi 624339473.0
Goa 75991368.0
Gujarat 2131646009.0
Haryana 755798352.0
Himachal Pradesh 316294004.0
India 28262144791.0
Jammu and Kashmir 410101777.0
Jharkhand 603673726.0
Karnataka 1873329968.0
Kerala 1193845072.0
Ladakh 17809249.0
Lakshadweep 4363655.0
Madhya Pradesh 1796604591.0
Maharashtra 2784364331.0
Manipur 67409568.0
Meghalaya 62615974.0
Mizoram 47873077.0
Nagaland 42410766.0
Odisha 1032633168.0
Puducherry 41346858.0
Punjab 584346582.0
Rajasthan 2201044187.0
Sikkim 36980929.0
Tamil Nadu 1288532512.0
Telangana 880320645.0
Tripura 192689726.0
Uttar Pradesh 2788411358.0
Uttarakhand 363191446.0
West Bengal 1796449989.0
```

In [24]:
```
first_dose=df[df['Updated On']=='09/08/2021']
```

In [26]:
```
first_dose[['State','First Dose Administered']]
```

Out[26]:

	State	First Dose Administered
205	India	400150406.0
417	Andaman and Nicobar Islands	216046.0
629	Andhra Pradesh	17628583.0
841	Arunachal Pradesh	692475.0
1053	Assam	10495293.0
1265	Bihar	23350171.0

	State	First Dose Administered
1477	Chandigarh	700285.0
1689	Chhattisgarh	9181482.0
1901	Dadra and Nagar Haveli and Daman and Diu	584370.0
2113	Delhi	7835546.0
2326	Goa	1094392.0
2538	Gujarat	28101222.0
2750	Haryana	10086831.0
2962	Himachal Pradesh	4249849.0
3174	Jammu and Kashmir	5318516.0
3386	Jharkhand	8382280.0
3598	Karnataka	25847691.0
3810	Kerala	15670747.0
4022	Ladakh	188699.0
4234	Lakshadweep	51156.0
4446	Madhya Pradesh	29723036.0
4658	Maharashtra	35040812.0
4870	Manipur	1159424.0
5082	Meghalaya	938572.0
5294	Mizoram	654946.0
5506	Nagaland	632120.0
5718	Odisha	13954592.0
5930	Puducherry	601591.0
6142	Punjab	8005636.0
6354	Rajasthan	27008606.0
6566	Sikkim	497851.0
6778	Tamil Nadu	20836674.0
6990	Telangana	11649268.0
7202	Tripura	2411195.0
7414	Uttar Pradesh	45932488.0
7626	Uttarakhand	5070544.0
7838	West Bengal	23257417.0

In [27]:
```
first_dose=df[(df['Updated On']=='09/08/2021') &(df['State'] != 'India')]
```

In [29]:
```
x=first_dose[['State','First Dose Administered']]
```

In [30]:
```
x.to_csv('FirstDoseIndia.csv',index=False)
```

In [32]:
```
Second_dose=df[(df['Updated On']=='09/08/2021') &(df['State'] != 'India')]
x=Second_dose[['State','Second Dose Administered']]
x.to_csv('SecondDoseIndia.csv',index=False)
```

In [33]:
```
# 4. Number of males vaccinated
males=df['Male(Individuals Vaccinated)']
```

In [35]:
```
males.max()
```

Out[35]: 134941971.0

In [36]:
```
females=df['Female(Individuals Vaccinated)']
```

In [37]:
```
females.max()
```

Out[37]: 115668447.0

In [ ]:
```

```