Subject Code &Name : AD3301 DATA EXPLORATION AND VISUALIZATION LABORATORY

Branch : AI&DS

Year/Semester : II/03

LIST OF EXPERIMENTS

S.No	Description of Experiment
1	Install the data Analysis and Visualization tool: R/ Python /Tableau Public/ Power BI
2	Working with Numpy arrays, Pandas data frames , Basic plots using Matplotlib
3	Perform exploratory data analysis (EDA) on with datasets like email data set. Export all your emails as a dataset, import them inside a pandas data frame, visualize them and get different insights from the data.
4	Perform Time Series Analysis and apply the various visualization techniques
5	Build cartographic visualization for multiple datasets involving various countries of the world; states and districts in India etc
6	Perform Data Analysis and representation on a Map using various Map data sets with Mouse Rollover effect, user interaction, etc..
7	Explore various variable and rowfilters in R for cleaning data.Apply various plot features in R on sample data sets and visualize
8	Perform EDA on Wine Quality Data Set
9	Use a case study on a data set and apply the various EDA and visualization techniques and present an analysis report.

INDEX

Sl. No.	Date	Program Name	Signature

Ex.No.1	**INSTALL THE DATA ANALYSIS AND VISUALIZATION TOOL: R/ PYTHON /TABLEAU PUBLIC/ POWER BI**

1a. Aim:

To download, install and explore the features of NumPy package.

Problem Description

Python is an open-source object-oriented language. It has many features of which one is the wide range of external packages. There are a lot of packages for installation and use for expanding functionalities. These packages are a repository of functions in python script. NumPy is one such package to ease array computations. To install all these python packages we use the pip- package installer. Pip is automatically installed along with Python. We can then use pip in the command line to install packages from PyPI.

NumPy

NumPy (Numerical Python) is an open-source library for the Python programming language. It is used for scientific computing and working with arrays.

Apart from its multidimensional array object, it also provides high-level functioning tools for working with arrays.

Prerequisites

- Access to a terminal window/command line
- A user account with sudo privileges
- Python installed on your system

Downloading and installing Numpy:

Python NumPy is a general-purpose array processing package that provides tools for handling n-dimensional arrays. It provides various computing tools such as comprehensive mathematical functions, linear algebra routines.Use the below command to install NumPy:

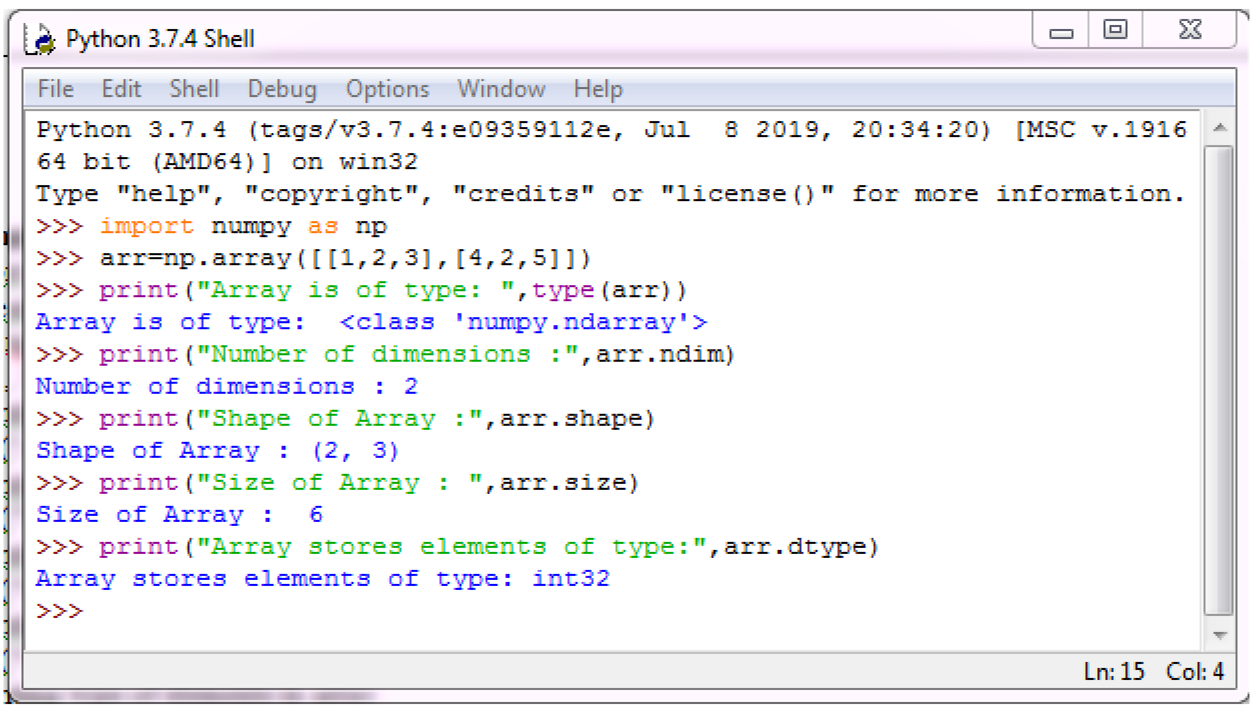
pip install numpy

output:

- **Sample python program using numpy**:

```
import numpy as np
# Creating array object
arr = np.array( [[ 1, 2, 3],
[ 4, 2, 5]] )
# Printing type of arr object
print("Array is of type: ", type(arr))
# Printing array dimensions (axes)
print("No. of dimensions: ", arr.ndim)
# Printing shape of array
print("Shape of array: ", arr.shape)
# Printing size (total number of elements) of array
print("Size of array: ", arr.size)
# Printing type of elements in array
print("Array stores elements of type: ", arr.dtype)
```

OUTPUT



```
Python 3.7.4 Shell
File Edit Shell Debug Options Window Help
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 20:34:20) [MSC v.1916
64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> import numpy as np
>>> arr=np.array([[1,2,3],[4,2,5]])
>>> print("Array is of type: ",type(arr))
Array is of type:  <class 'numpy.ndarray'>
>>> print("Number of dimensions :",arr.ndim)
Number of dimensions : 2
>>> print("Shape of Array :",arr.shape)
Shape of Array : (2, 3)
>>> print("Size of Array : ",arr.size)
Size of Array :  6
>>> print("Array stores elements of type:",arr.dtype)
Array stores elements of type: int32
>>>
Ln: 15  Col: 4
```

RESULT:

Thus the NumPy package is downloaded, installed and the features are explored.

1 B) Aim:

To download, install and explore the features of Scipy package.

Problem Description

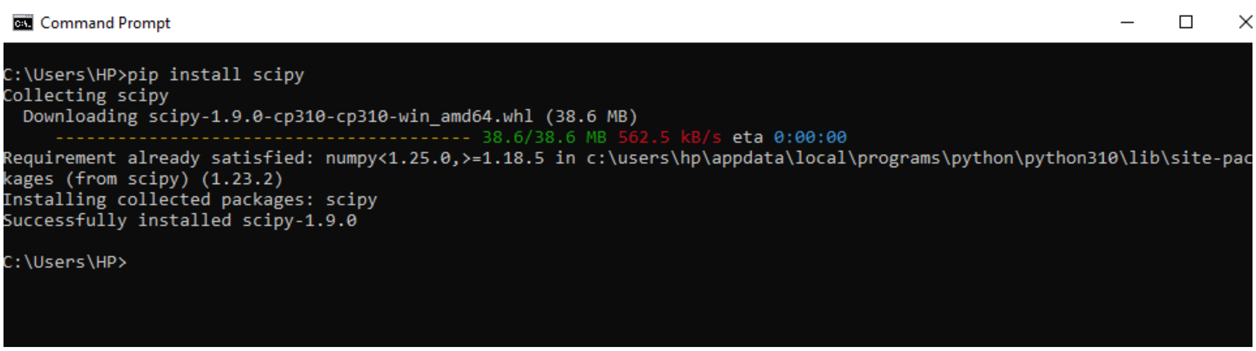
Scipy is a python library that is useful in solving many mathematical equations and algorithms. It is designed on the top of Numpy library that gives more extension of finding scientific mathematical formulae like Matrix Rank, Inverse, polynomial equations, LU Decomposition, etc. Using its high-level functions will significantly reduce the complexity of the code and helps in better analyzing the data.

Downloading and Installing Scipy:

pip use the below command to install Scipy package on Windows:

pip install scipy

output



Sample python code using Scipy:

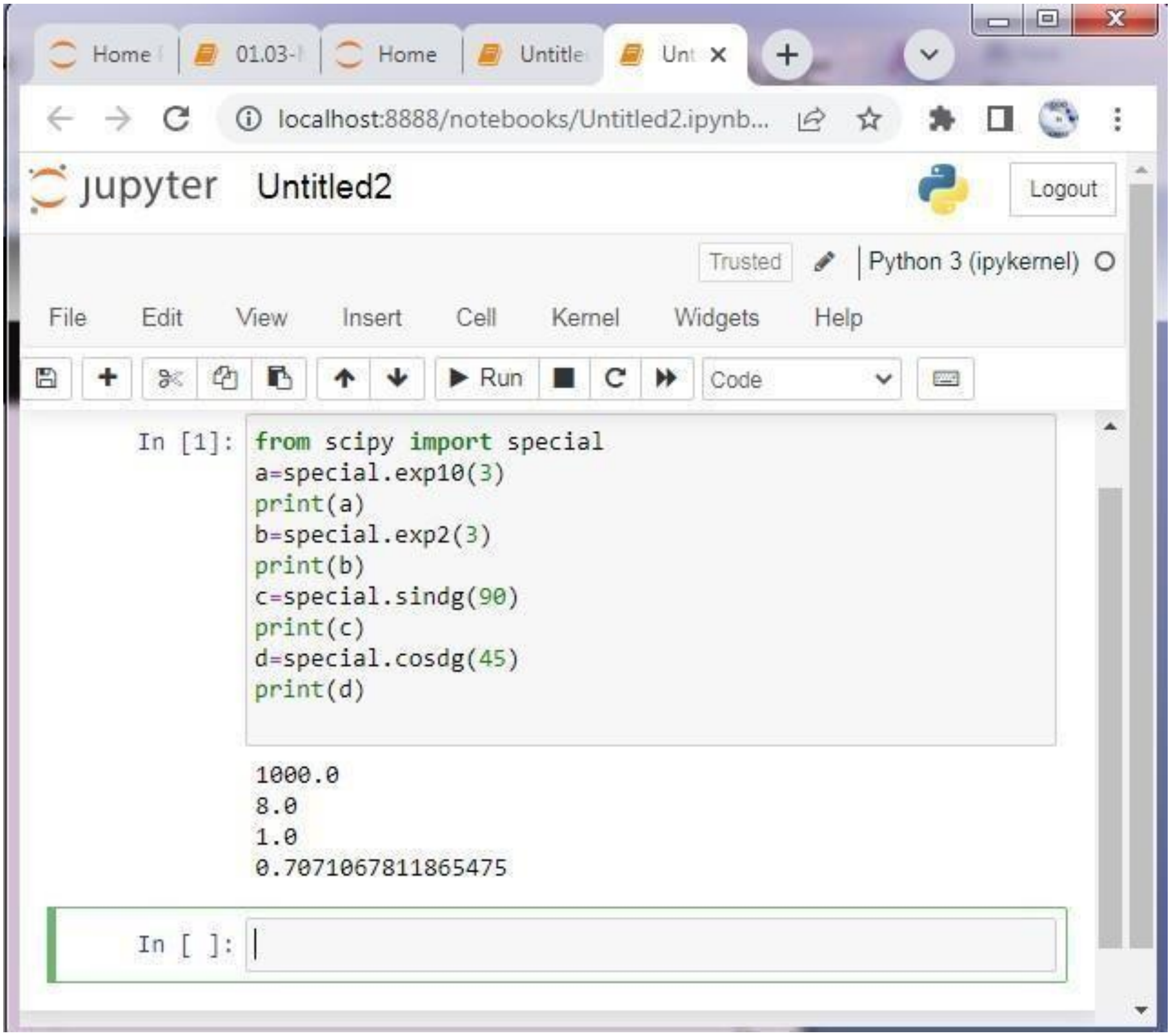
<u>Type the program in Jupyter notebook</u>

```
from scipy import special
a = special.exp10(3)
print(a)
b = special.exp2(3)
print(b)
c = special.sindg(90)
print(c)

d = special.cosdg(45)

print(d)
```

Output:



RESULT

Thus the SciPy package is downloaded, installed and the features are explored.

1 C). **Aim** :

To download, install and explore the features of Panda packages.

Problem Description

Pandas is one of the most popular open-source frameworks available for Python. It is among the fastest and most easy-to-use libraries for data analysis and manipulation. Pandas dataframes are some of the most useful data structures available in any library. It has uses in every data-intensive field, including but not limited to scientific computing, data science, and machine learning.

The library does not come included with a regular install of Python. To use it, you must install the Pandas framework separately.

Installing Pandas on Windows

There are two ways of installing Pandas on Windows.

Method #1: Installing with pip

It is a package installation manager that makes installing Python libraries and frameworks straightforward.

As long as you have a newer version of Python installed (> Python 3.4), pip will be installed on your computer along with Python by default.

However, if you're using an older version of Python, you will need to install pip on your computer before installing Pandas.

Step #1: Launch Command Prompt

Press the Windows key on your keyboard or click on the Start button to open the start menu. Type **cmd**, and the Command Prompt app should appear as a listing in the start menu.

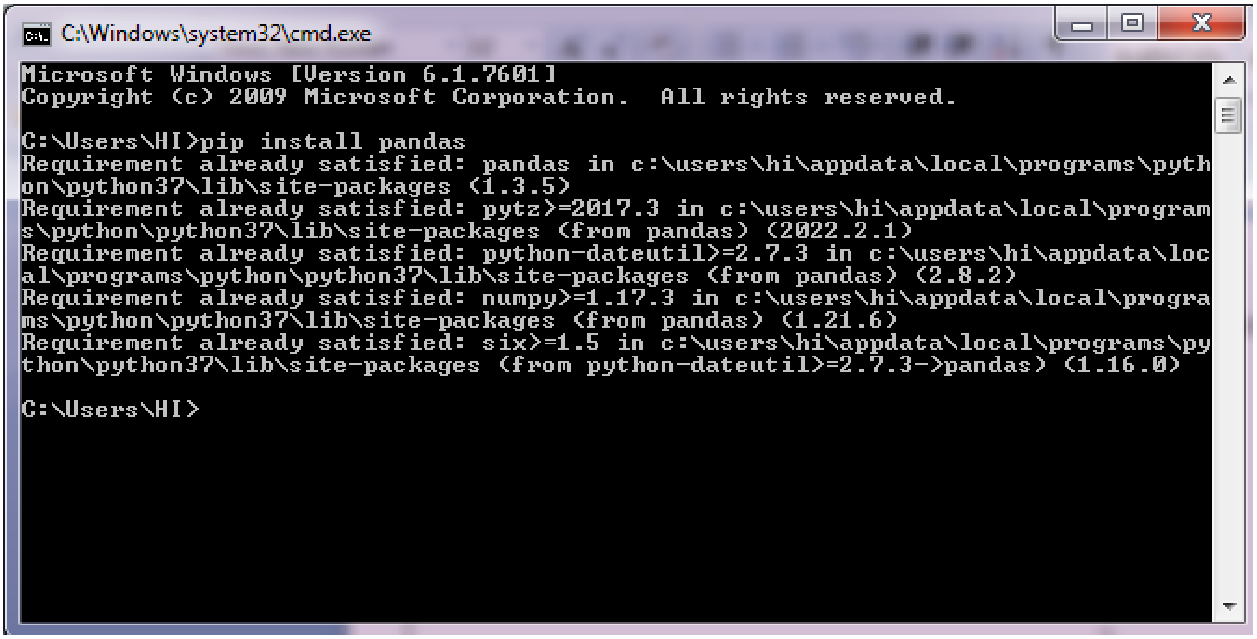
Step #2: Enter the Required Command

After you launch the command prompt, the next step in the process is to type in the required command to initialize pip installation.

Enter the command

pip install pandas

on the terminal. This should launch the pip installer. The required files will be downloaded, and Pandas will be ready to run on your computer.



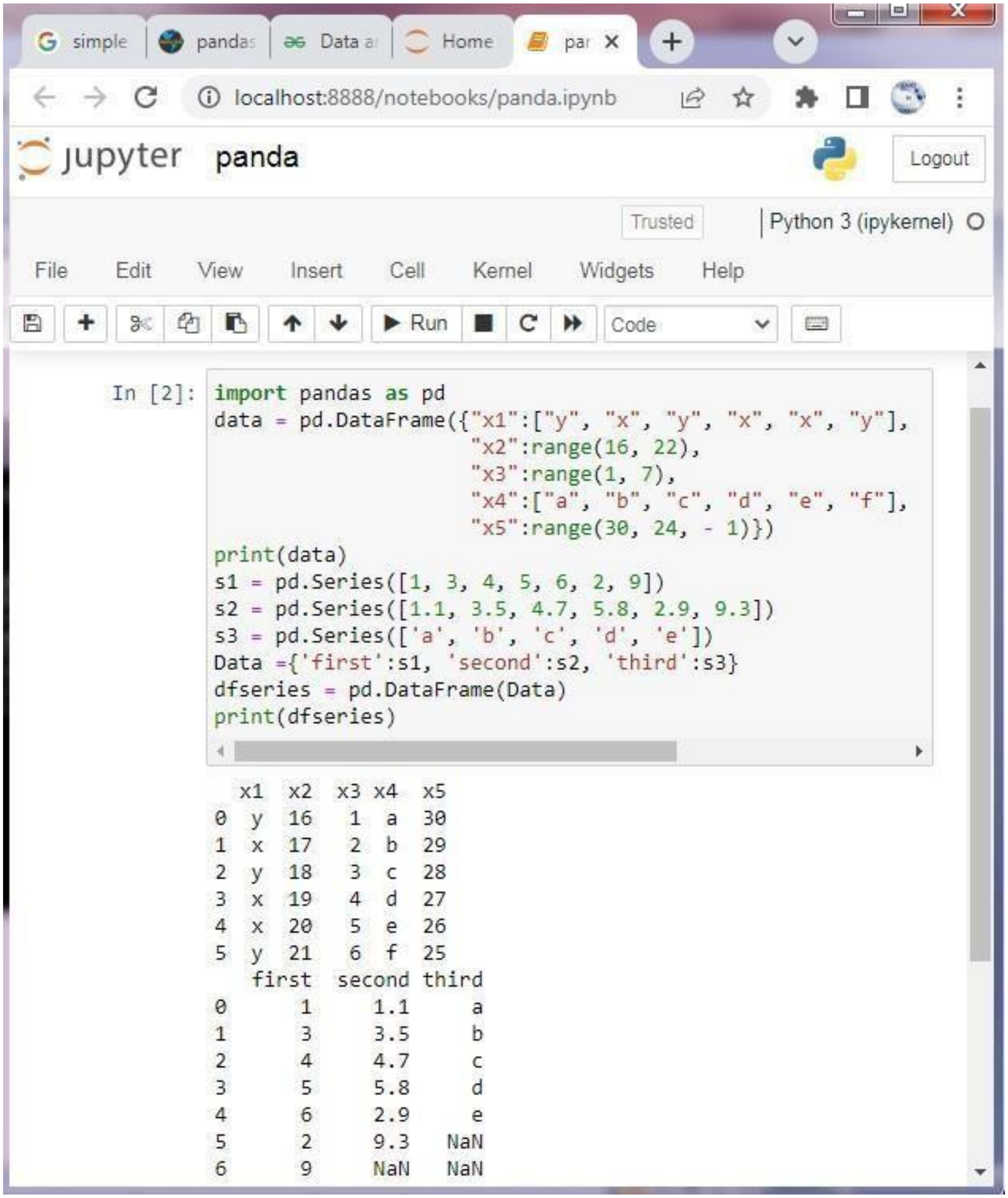
Panda package is successfully installed.

Sample program

// to be typed in Jupyter notebook

```
import pandas as pd
data = pd.DataFrame({"x1":["y", "x", "y", "x", "x", "y"],
                    "x2":range(16, 22),
                    "x3":range(1, 7),
                    "x4":["a", "b", "c", "d", "e", "f"],
                    "x5":range(30, 24, - 1)})
print(data)
s1 = pd.Series([1, 3, 4, 5, 6, 2, 9])
s2 = pd.Series([1.1, 3.5, 4.7, 5.8, 2.9, 9.3])
s3 = pd.Series(['a', 'b', 'c', 'd', 'e'])
Data ={'first':s1, 'second':s2, 'third':s3}
```

dfseries = pd.DataFrame(Data)
print(dfseries)



Result:

Thus the Panda package is downloaded, installed and the features are explored.

Ex.No.2	**WORKING WITH NUMPY ARRAYS**

Aim :

Write a python program to show the woking of NumPy Arrays in Python.

2a) Use Numpy array to demonstrate basic array characteristics

b) Create Numpy array using list and tuple

c) Apply basic operations (+,_,*./) and find the transpose of the matrix

d) Perform sorting operation with Numpy arrays

Problem Description

Arrays in NumPy: NumPy's main object is the homogeneous multidimensional array.

- It is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers.
- In NumPy dimensions are called *axes*. The number of axes is *rank*.
- NumPy's array class is called **ndarray**. It is also known by the alias **array**.

Example 1:

Write a python program to demonstrate the basic NumPy array characteristics

```
import numpy as np

# Creating array object
arr = np.array( [[ 1, 2, 3],
                [ 4, 2, 5]] )

# Printing type of arr object
print("Array is of type: ", type(arr))

# Printing array dimensions (axes)
print("No. of dimensions: ", arr.ndim)

# Printing shape of array
print("Shape of array: ", arr.shape)

# Printing size (total number of elements) of array
print("Size of array: ", arr.size)
```

# Printing type of elements in array
print("Array stores elements of type: ", arr.dtype)

Output :

Array is of type: <class 'numpy.ndarray'>

No. of dimensions: 2

Shape of array: (2, 3)

Size of array: 6

Array stores elements of type: int64

2. Array creation:

There are various ways to create arrays in NumPy.

- For example, you can create an array from a regular Python **list** or **tuple** using the **array** function. The type of the resulting array is deduced from the type of the elements in the sequences.
- Often, the elements of an array are originally unknown, but its size is known. Hence, NumPy offers several functions to create arrays with **initial placeholder content**. These minimize the necessity of growing arrays, an expensive operation. **For example:** np.zeros, np.ones, np.full, np.empty, etc.
- To create sequences of numbers, NumPy provides a function analogous to range that returns arrays instead of lists.
- **arange:** returns evenly spaced values within a given interval. **step** size is specified.
- **linspace:** returns evenly spaced values within a given interval. **num** no. of elements are returned.
- **Reshaping array:** We can use **reshape** method to reshape an array. Consider an array with shape (a1, a2, a3, …, aN). We can reshape and convert it into another array with shape (b1, b2, b3, …, bM). The only required condition is: a1 x a2 x a3 … x aN = b1 x b2 x b3 … x bM . (i.e original size of array remains unchanged.)
- **Flatten array:** We can use **flatten** method to get a copy of array collapsed into **one dimension**. It accepts *order* argument. Default value is 'C' (for row-major order). Use 'F' for column major order.

Example 2:

import numpy as np

# Creating array from list with type float
a = np.array([[1, 2, 4], [5, 8, 7]], dtype = 'float')
print ("Array created using passed list:\n", a)

```
# Creating array from tuple
b = np.array((1 , 3, 2))
print ("\nArray created using passed tuple:\n", b)

# Creating a 3X4 array with all zeros
c = np.zeros((3, 4))
print ("\nAn array initialized with all zeros:\n", c)

# Create a constant value array of complex type
d = np.full((3, 3), 6, dtype = 'complex')
print ("\nAn array initialized with all 6s.")
 print( "Array type is complex:\n", d)

# Create an array with random values
e = np.random.random((2, 2))
print ("\nA random array:\n", e)

# Create a sequence of integers
# from 0 to 30 with steps of 5
f = np.arange(0, 30, 5)
print ("\nA sequential array with steps of 5:\n", f)

# Create a sequence of 10 values in range 0 to 5
g = np.linspace(0, 5, 10)
print ("\nA sequential array with 10 values between"
                                        "0 and 5:\n", g)

# Reshaping 3X4 array to 2X2X3 array
arr = np.array([[1, 2, 3, 4],
                [5, 2, 4, 2],
                [1, 2, 0, 1]])

newarr = arr.reshape(2, 2, 3)

print ("\nOriginal array:\n", arr)
print ("Reshaped array:\n", newarr)

# Flatten array
arr = np.array([[1, 2, 3], [4, 5, 6]])
flarr = arr.flatten()

print ("\nOriginal array:\n", arr)
print ("Fattened array:\n", flarr)
```

OUTPUT

```
Array created using passed list:
[[ 1. 2. 4.]
[ 5. 8.  7.]]

Array created using passed tuple:
[1 3 2]

An array initialized with all zeros:
[[ 0. 0. 0. 0.]
[ 0.  0.  0.  0.]

[ 0.  0.  0.  0.]]

An array initialized with all 6s. Array type is complex:
[[ 6.+0.j 6.+0.j 6.+0.j]
[ 6.+0.j 6.+0.j  6.+0.j]
[ 6.+0.j 6.+0.j 6.+0.j]]

A random array:
[[ 0.46829566  0.67079389]
[ 0.09079849  0.95410464]]

A sequential array with steps of 5:
[ 0 5 10 15 20 25]

A sequential array with 10 values between 0 and 5:
[ 0.         0.55555556 1.11111111 1.66666667 2.22222222  2.77777778
3.33333333  3.88888889  4.44444444  5.        ]
```

Original array:

```
[[1 2 3 4]
[5 2 4 2]
[1 2 0 1]]
```

Reshaped array:

```
[[[1 2 3]
 [4 5 2]]
[[4 2 1]
 [2 0 1]]]
```

Original array:

```
[[1 2 3]
[4 5 6]]
```

Fattened array:

```
[1 2 3 4 5 6]
```

3. Basic operations:

Arithmetic oprations on NumPy Array:

Program 3:

```
import numpy as np

array1 = np.array([[1, 2, 3], [4, 5, 6]])
array2 = np.array([[7, 8, 9], [10, 11, 12]])

print("Addition")
print(array1 + array2)
print("-" * 20)
print("Subtraction")
print(array1 - array2)
print("-" * 20)
print("Multiplication")
print(array1 * array2)
print("-" * 20)
print("Division")
```

```
print(array2 / array1)
print("-" * 40)
print(array1 ** array2)
print("-" * 40)
a = np.array([1, 2, 5, 3])
print ("Adding 1 to every element:", a+1)
print ("Subtracting 3 from each element:", a-3)
print ("Multiplying each element by 10:", a*10)
print ("Squaring each element:", a**2)
a *= 2
print ("Doubled each element of original array:", a)
a = np.array([[1, 2, 3], [3, 4, 5], [9, 6, 0]])
print ("\nOriginal array:\n",  a)
print ("Transpose of array:\n", a.T)
```

Output

```
Addition
[[ 8 10 12]
 [14 16 18]]
----------------------------------------
Subtraction
[[-6 -6 -6]
 [-6 -6 -6]]
----------------------------------------
Multiplication
[[ 7 16 27]
 [40 55 72]]
----------------------------------------
Division
[[7.  4.  3. ]
 [2.5 2.2 2. ]]
----------------------------------------
[[          1         256       19683]
 [    1048576    48828125 -2118184960]]
----------------------------------------
```

Adding 1 to every element: [2 3 6 4]

Subtracting 3 from each element: [-2 -1 2 0]

Multiplying each element by 10: [10 20 50 30]

Squaring each element: [1 4 25 9]

Doubled each element of original array: [2 4 10 6]

Original array:

[[1 2 3]

[3 4 5]

[9 6 0]]

Transpose of array:

[[1 3 9]

[2 4 6]

[3 5 0]]

4. Sorting array: There is a simple **np.sort** method for sorting NumPy arrays. Let's explore it a bit.

Program 4:

```
import numpy as np


a = np.array([[1, 4, 2],
              [3, 4, 6],
            [0, -1, 5]])

# sorted array
print ("Array elements in sorted order:\n",
                np.sort(a, axis = None))

# sort array row-wise
print ("Row-wise sorted array:\n",
             np.sort(a, axis = 1))

# specify sort algorithm
print ("Column wise sort by applying merge-sort:\n",
          np.sort(a, axis = 0, kind = 'mergesort'))

# Example to show sorting of structured array
# set alias names for dtypes
dtypes = [('name', 'S10'), ('grad_year', int), ('cgpa', float)]

# Values to be put in array
values = [('Hrithik', 2009, 8.5), ('Ajay', 2008, 8.7),
         ('Pankaj', 2008, 7.9), ('Aakash', 2009, 9.0)]

# Creating array
arr = np.array(values, dtype = dtypes)
print ("\nArray sorted by names:\n",
```

```
    np.sort(arr, order = 'name'))

print ("Array sorted by graduation year and then cgpa:\n",
       np.sort(arr, order = ['grad_year', 'cgpa']))
```

OUTPUT

```
Array elements in sorted order:
        [-1 0 1 2  3  4  4  5  6]
Row-wise sorted array:
        [[ 1  2  4]
         [ 3  4  6]
         [-1  0  5]]
Column wise sort by applying merge-sort:
        [[ 0 -1 2]
         [ 1  4  5]
         [ 3 4  6]]

Array sorted by names:
        [('Aakash', 2009, 9.0) ('Ajay', 2008, 8.7) ('Hrithik', 2009, 8.5)
         ('Pankaj', 2008, 7.9)]
Array sorted by graduation year and then cgpa:
        [('Pankaj', 2008, 7.9) ('Ajay', 2008, 8.7) ('Hrithik', 2009, 8.5)
         ('Aakash', 2009, 9.0)]
```

Result

Thus the python programs are written and executed to explain the features of NumPy array.

Ex.No.3	PERFORM EXPLORATORY DATA ANALYSIS (EDA) ON WITH DATASETS LIKE EMAIL DATA SET. EXPORT ALL YOUR EMAILS AS A DATASET, IMPORT THEM INSIDE A PANDAS DATA FRAME, VISUALIZE THEM AND GET DIFFERENT INSIGHTS FROM THE DATA.

Aim:

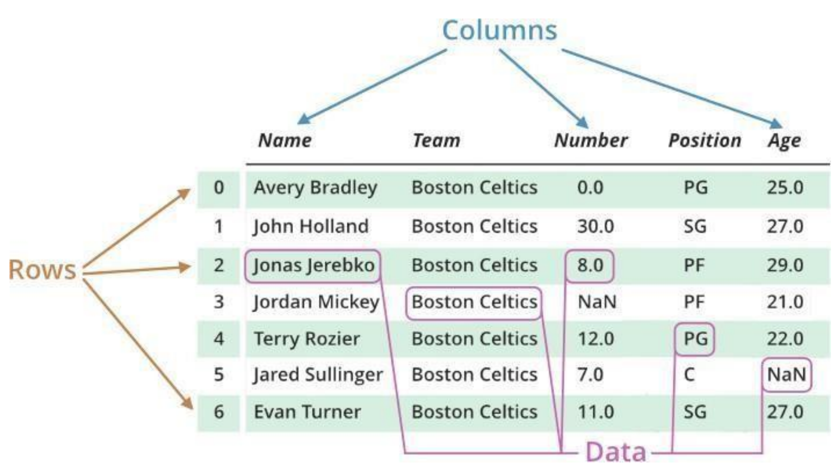
Write a python program to work with Panda data frames.

Pandas is an open-source library that is built on top of NumPy library. It is a Python package that offers various data structures and operations for manipulating numerical data andtime series. It is mainly popular for importing and analyzing data much easier. Pandas is fast and it has high-performance & productivity for users.

Pandas DataFrame

In the real world, a Pandas DataFrame will be created by loading the datasets from existing storage, storage can be SQL Database, CSV file, and Excel file. Pandas DataFrame can be created from the lists, dictionary, and from a list of dictionary etc.

Pandas DataFrame is two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns. Pandas DataFrame consists of three principal components, the **data**, **rows**, and **columns**.



Creating a Panda Data Frames

A pandas DataFrame can be created using the following constructor −

pandas.DataFrame(data, index, columns, dtype, copy)

Creating an empty dataframe :
A basic DataFrame, which can be created is an Empty Dataframe. An Empty Dataframe is created just by calling a dataframe constructor.

Creating a dataframe using List:

DataFrame can be created using a single list or a list of lists.

Creating dataframe from dict of ndarray/lists:

To create dataframe from dict of narray/list, all the narray must be of same length. If index is passed then the length index should be equal to the length of arrays. If no index is passed, then by default, index will be range(n) where n is the array length.

Iterating over rows :

In order to iterate over rows, we can use three function iteritems(), iterrows(), itertuples() . These three function will help in iteration over rows.

Program

```
import pandas as pd


# Calling DataFrame constructor
print("Empty dataframe")
df = pd.DataFrame()
print(df)
print("Dataframe creation using List")
# list of strings
lst = ['Geeks', 'For', 'Geeks', 'is', 'portal', 'for', 'Geeks']
# Calling DataFrame constructor on list
df = pd.DataFrame(lst)
print(df)
# initialise data of lists.
Data = {'Name':['Tom', 'nick', 'krish', 'jack'], 'Age':[20, 21, 19, 18]}
# Create dataframe

df = pd.DataFrame(Data)
```

```
# Print the output.
print(df)
print("Create dataframe from dictionoary of lists")
# dictionary of lists
dict = {'name':["aparna", "pankaj", "sudhir", "Geeku"],
        'Degree': ["MBA", "BCA", "M.Tech", "MBA"],
        'Score':[90, 40, 80, 98]}

# creating a dataframe from a dictionary
df = pd.DataFrame(dict)
print(df)
# iterating over rows using iterrows() function
for i, j in df.iterrows():
    print(i, j)
    print()
```

OUTPUT

```
Empty dataframe
Empty DataFrame
Columns: []
Index: []

Dataframe creation using List
       0
1   Geeks
2     For
3   Geeks
4      is
5  portal
6     for
6  Geeks
    Name  Age
1    Tom  20
2   nick 21
2  krish  19
```

3 jack 18

Create dataframe from dictionoary of lists

```
     name Degree  Score
0  aparna     MBA     90
1  pankaj     BCA     40
2  sudhir  M.Tech     80
3   Geeku     MBA     98

0 name      aparna
Degree       MBA
Score        90
Name: 0, dtype: object

1 name  pankaj
Degree      BCA
Score       40
Name: 1, dtype: object

2 name      sudhir
Degree    M.Tech
Score 80
Name: 2, dtype: object

3 name    Geeku
Degree MBA
Score       98
Name: 3, dtype: object
```

Pandas Dataframe visualization
Retrieving data from the web

In[1]:

```
import pandas as pd
url =
'https://github.com/chris1610/pbpython/blob/master/data/2018_Sales_Total_v2.xlsx?raw=True'
df = pd.read_excel(url)
df
```

OUTPUT

	account number	name	sku	quantity	unit price	ext price	date
0	740150	Barton LLC	B1-20000	39	86.69	3380.91	2018-01-01 07:21:51
1	714466	Trantow-Barrows	S2-77896	-1	63.16	-63.16	2018-01-01 10:00:47
2	218895	Kulas Inc	B1-69924	23	90.70	2086.10	2018-01-01 13:24:58
3	307599	Kassulke, Ondricka and Metz	S1-65481	41	21.05	863.05	2018-01-01 15:05:22
4	412290	Jerde-Hilpert	S2-34077	6	83.21	499.26	2018-01-01 23:26:55
...	...	...	...	...	...	...	...
1502	424914	White-Trantow	B1-69924	37	42.77	1582.49	2018-11-27 14:29:02
1503	424914	White-Trantow	S1-47412	16	65.58	1049.28	2018-12-19 15:15:41
1504	424914	White-Trantow	B1-86481	75	28.89	2166.75	2018-12-29 13:03:54
1505	424914	White-Trantow	S1-82801	20	95.75	1915.00	2018-12-22 03:31:36
1506	424914	White-Trantow	S2-83881	100	88.19	8819.00	2018-12-16 00:46:26

1507 rows × 7 columns

Pandas for retrieving data from the csv file

In[2]:

```
import pandas as pd

data = pd.read_csv(r'C:\Users\HI\Downloads\PythonDataScienceHandbook-master\notebooks\data\iris.csv')
df = pd.DataFrame(data)

print (df)
```

Out[2]:

```
     sepallength  sepalwidth  petallength  petalwidth           class
0            5.1         3.5          1.4         0.2     Iris-setosa
1            4.9         3.0          1.4         0.2     Iris-setosa
2            4.7         3.2          1.3         0.2     Iris-setosa
3            4.6         3.1          1.5         0.2     Iris-setosa
4            5.0         3.6          1.4         0.2     Iris-setosa
..           ...         ...          ...         ...             ...
145          6.7         3.0          5.2         2.3  Iris-virginica
146          6.3         2.5          5.0         1.9  Iris-virginica
147          6.5         3.0          5.2         2.0  Iris-virginica
148          6.2         3.4          5.4         2.3  Iris-virginica
149          5.9         3.0          5.1         1.8  Iris-virginica

[150 rows x 5 columns]
```

Result:

Thus the python program is written to show the working of pandas dataframes

Ex.No.4	PERFORM TIME SERIES ANALYSIS AND APPLY THE VARIOUS VISUALIZATION TECHNIQUES

Aim:

To Perform time series analysis and apply the various visualization techniques

What is Exploratory Data Analysis?

Exploratory Data Analysis (EDA) is a technique to analyze data using some visual Techniques. With this technique, we can get detailed information about the statistical summary of the data. We will also be able to deal with the duplicates values, outliers, and also see some trends or patterns present in the dataset.

Now let's see a brief about the Iris dataset.

Iris Dataset

If you are from a data science background you all must be familiar with the Iris Dataset. If you are not then don't worry we will discuss this here.

Iris Dataset is considered as the Hello World for data science. It contains five columns namely – Petal Length, Petal Width, Sepal Length, Sepal Width, and Species Type. Iris is a flowering plant, the researchers have measured various features of the different iris flowers and recorded them digitally.

4A). Aim:

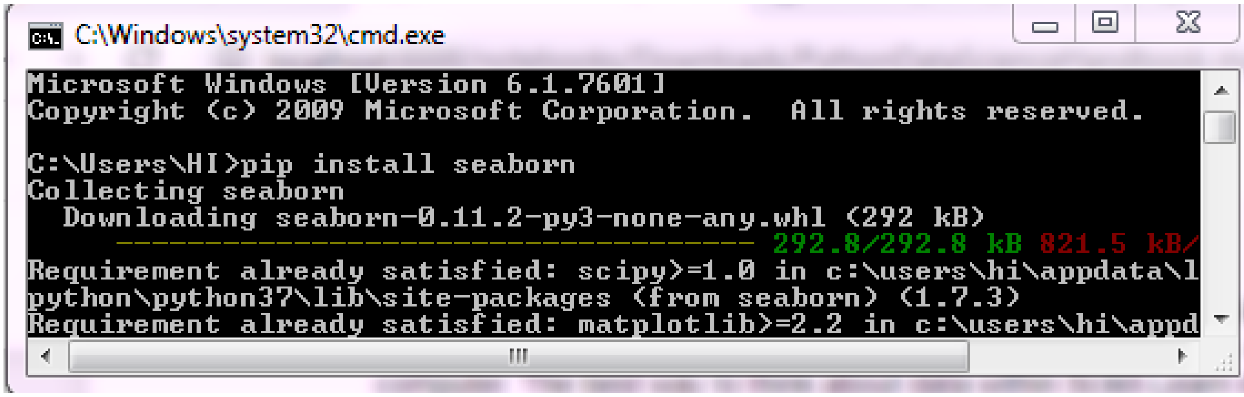
Reading data from Text file and exploring various commands for doing descriptive analytics on the Iris data set.

Seaborn Package:

Seaborn has many of its own high-level plotting routines, but it can also overwrite Matplotlib's default parameters and in turn get even simple Matplotlib scripts to produce vastly superior output. We can set the style by calling Seaborn's set() method. By convention, Seaborn is imported as sns:

Seaborn package is installed by typing the following command in the command prompt

pip install seaborn



Step 2:

After the successful insertion of seaborn package launch into jupyter using jupyter notebook command. Type the following program. Give the correct path name for iris dataset. The Iris dataset, which lists measurements of petals and sepals of three iris species.

Step 1:

Import Packages

In[1]:

```
import numpy as np
import pandas as pd # package for working with data frames in python
import seaborn as sns # package for visualization (more on seaborn later)
import matplotlib.pyplot as plt
%matplotlib inline
```

Step 2:

Import iris dataset

In[2]:

```
iris = sns.load_dataset('iris')
my_data_frame = pd.DataFrame(iris)
my_data_frame.head()
```

OUTPUT

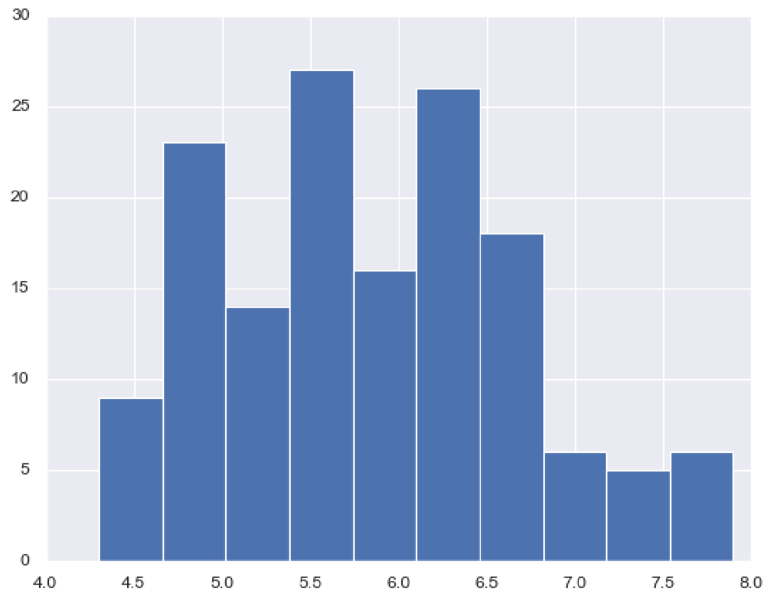
	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Step 3: Simple plot

In[3]:

```
p=plt.hist(my data frame.sepal length)
```

OUTPUT

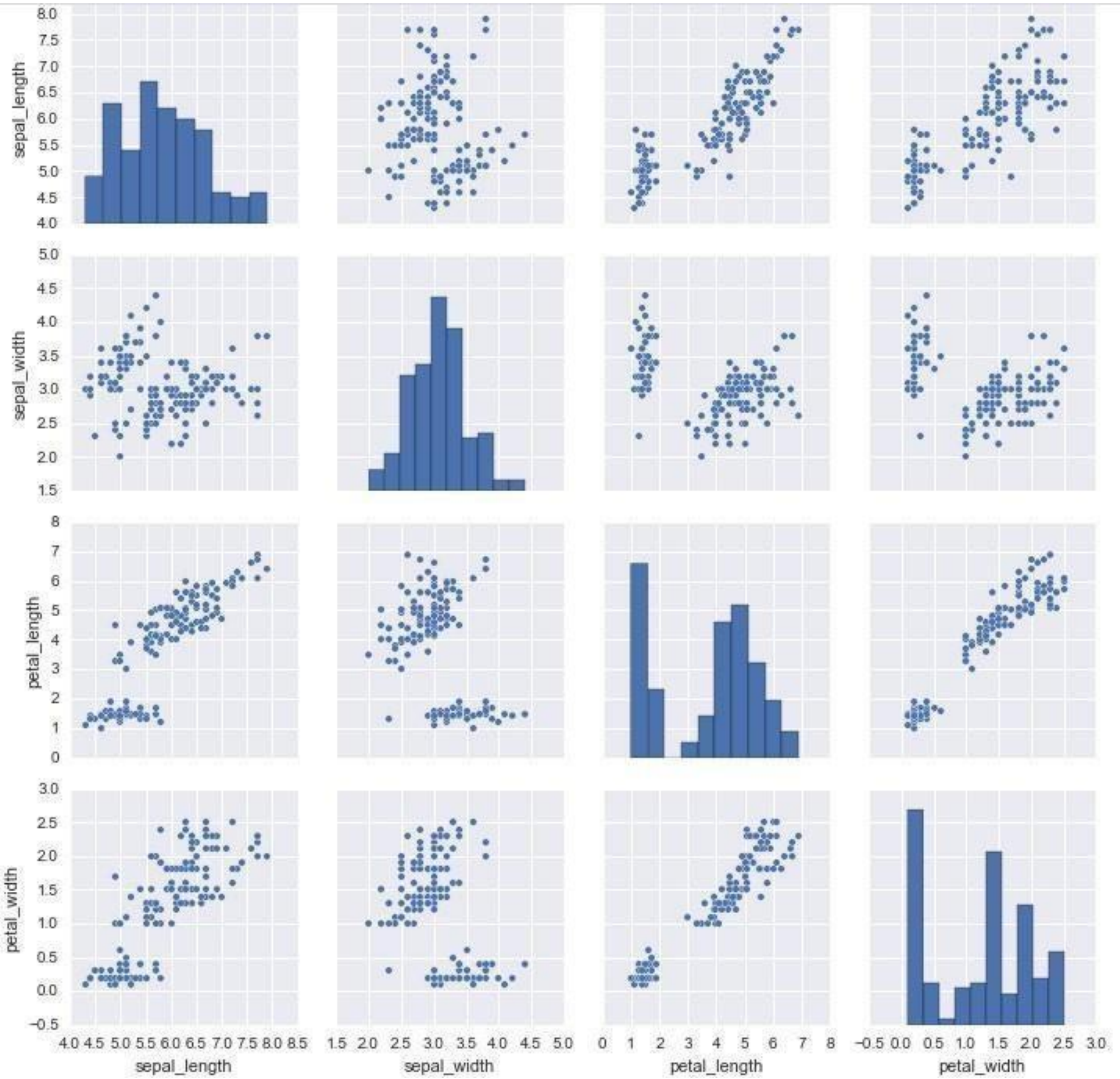


Step: 4 Plot using Seaborn

In[4]:

```
g = sns.pairplot(my data frame)
```

OUTPUT

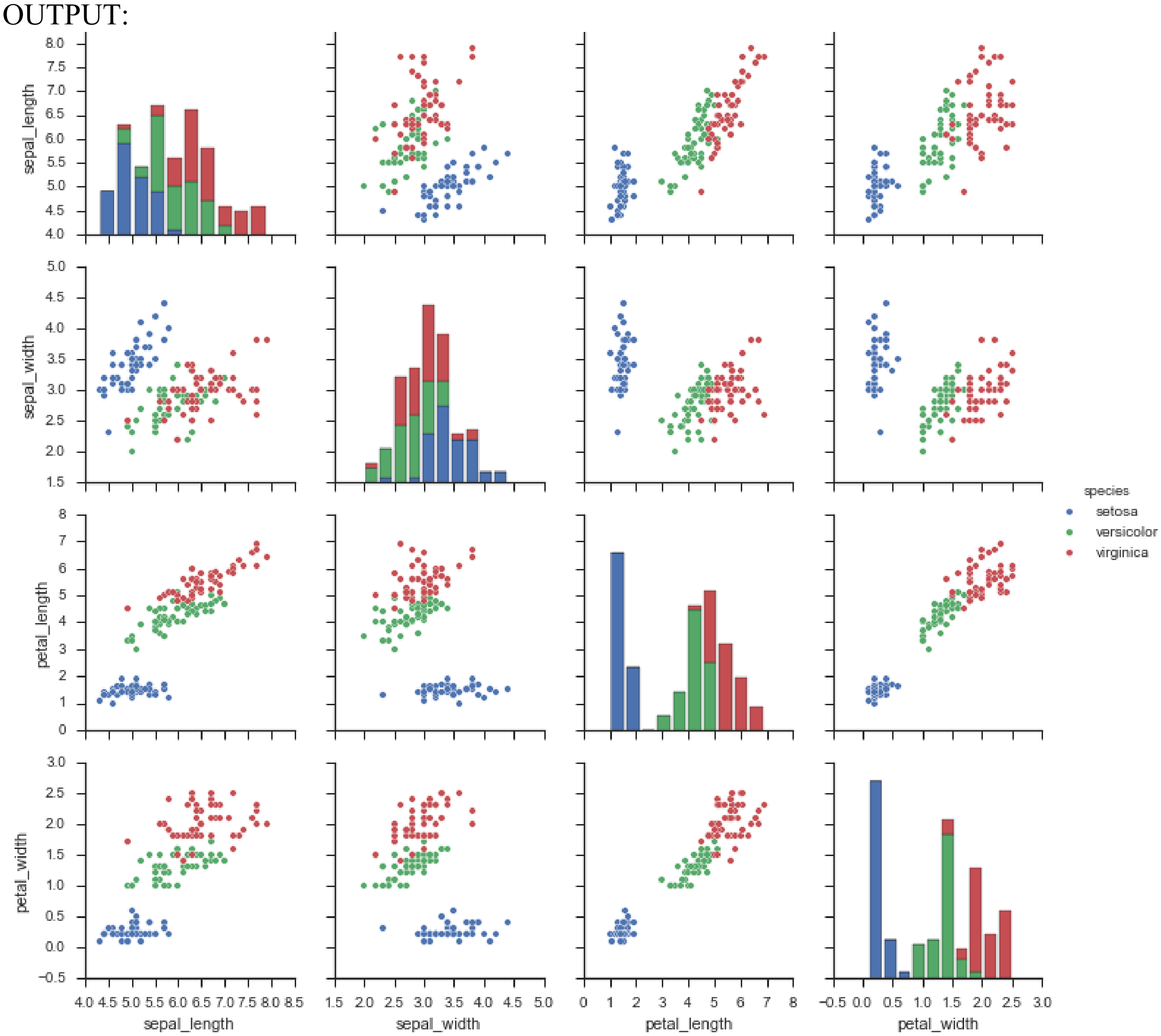


Step 5: Colour plot

In[5]:

```
sns.set(style="ticks", color_codes=True) # change style g =
sns.pairplot(iris, hue="species")
```

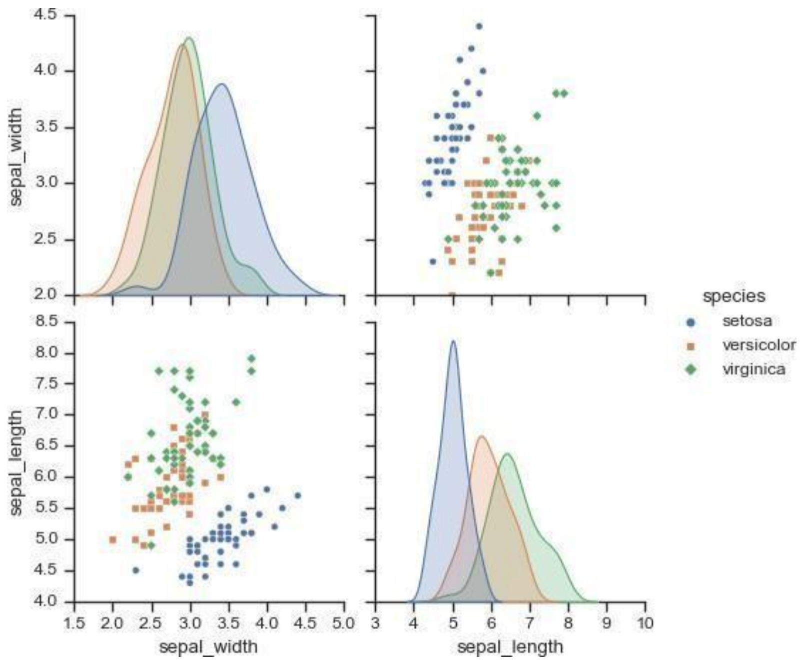
OUTPUT:



In [6]:

```
g = sns.pairplot(iris, height=3, vars=["sepal_width", "sepal_length"], \
                 markers=["o", "s", "D"], hue="species")
```

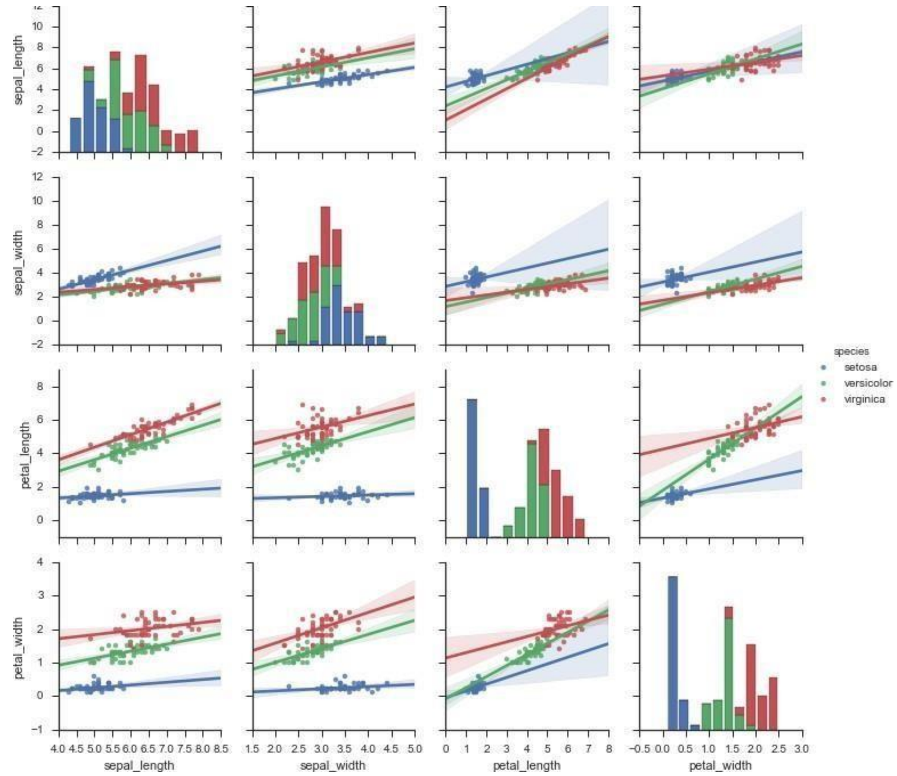
OUTPUT



In [7]:

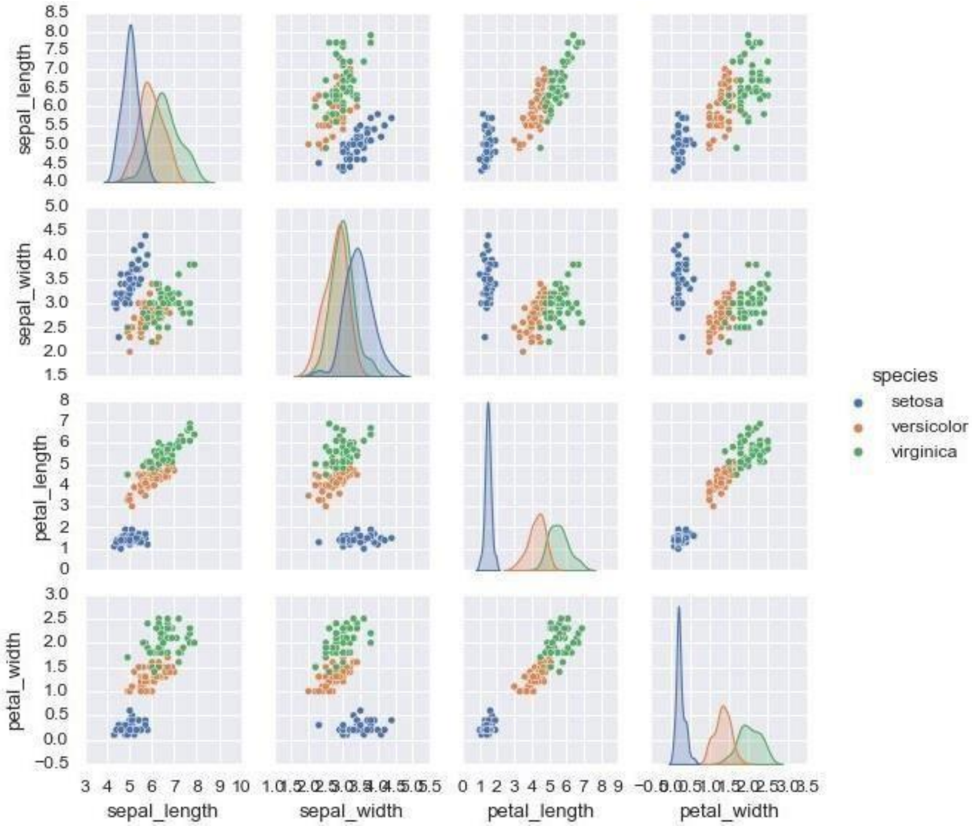
```
g = sns.pairplot(iris, kind="reg", hue="species")
```

OUTPUT



In[8]:

```
sns.set(style="ticks", color_codes=True) # change style
g = sns.pairplot(iris, hue="species")
```



Result:

Thus time series analysis and the various visualization techniques are applid and ececuted**.**

Ex.No.5	VISUALIZING GEOGRAPHIC DATA WITH BASEMAP

Aim:

To visualizing geographic data with basemap

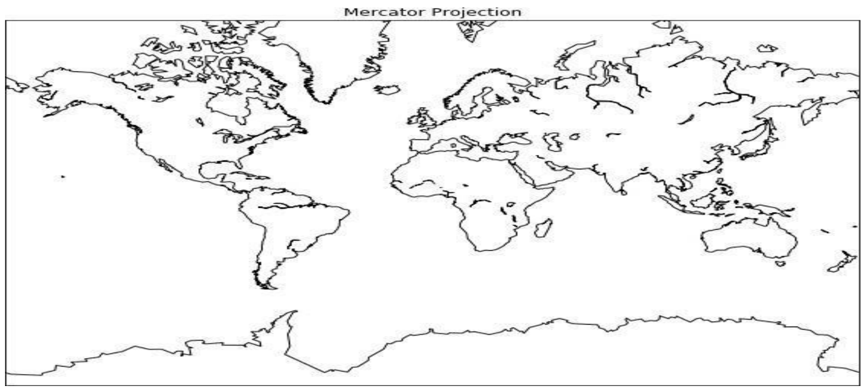
PROGRAM

1. **Simple Maps and color it**

To start, import Basemap as well as matplotlib and numpy:

```
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
import warnings
import matplotlib.cbook
warnings.filterwarnings("ignore",category=matplotlib.cbook.mplDeprecation)
Basemap?
fig = plt.figure(num=None, figsize=(12, 8) )
m = Basemap(projection='merc',llcrnrlat=-80,urcrnrlat=80,llcrnrlon=-
180,urcrnrlon=180,resolution='c')
m.drawcoastlines()
plt.title("Mercator Projection")
plt.show()
```

OUTPUT:

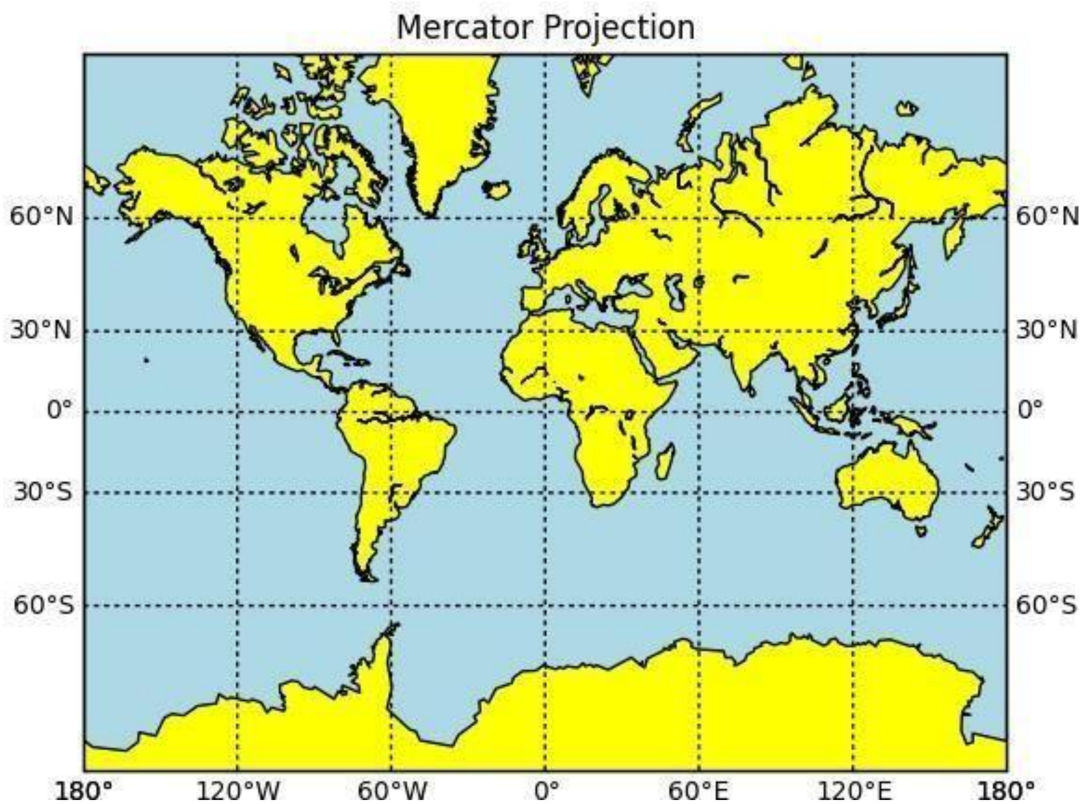


1a. Coding for Coloring

```
fig = plt.figure(num=None, figsize=(12, 8) )
m = Basemap(projection='merc',llcrnrlat=-80,urcrnrlat=80,llcrnrlon=-
180,urcrnrlon=180,resolution='c')
m.drawcoastlines()
m.fillcontinents(color='tan',lake_color='lightblue')
# draw parallels and meridians.
m.drawparallels(np.arange(-90.,91.,30.),labels=[True,True,False,False],dashes=[2,2])
m.drawmeridians(np.arange(-180.,181.,60.),labels=[False,False,False,True],dashes=[2,2])
m.drawmapboundary(fill_color='lightblue')
plt.title("Mercator Projection")
```

Output

```
Out[5]: Text(0.5, 1.0, 'Mercator Projection')
```



Result:

Thus visualizing geographic data with basemap is implemented.

Ex.No.6	BUILD CARTOGRAPHIC VISUALIZATION FOR MULTIPLE DATASETS INVOLVING VARIOUS COUNTRIES OF THE WORLD

Aim:

To Build cartographic visualization for multiple datasets involving various countries of the world; states and districts in India etc

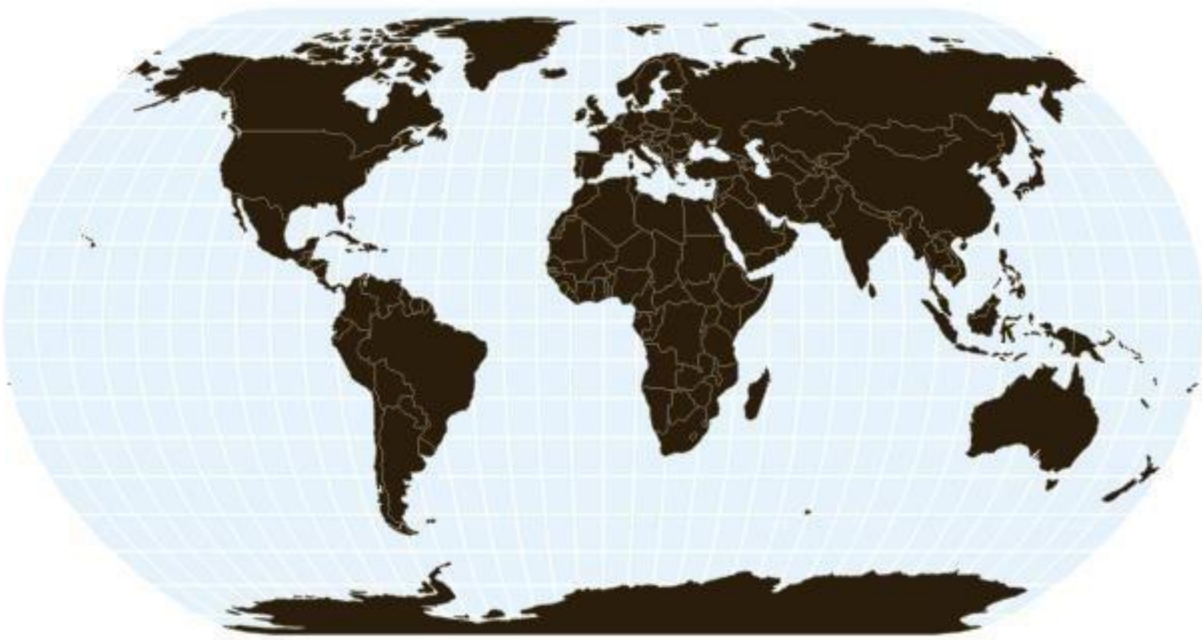
Program:

```
map = alt.layer(
    # use the sphere of the Earth as the base layer
    alt.Chart({'sphere': True}).mark_geoshape(
        fill='#e6f3ff'
    ),
    # add a graticule for geographic reference lines
    alt.Chart({'graticule': True}).mark_geoshape(
        stroke='#ffffff', strokeWidth=1
    ),
    # and then the countries of the world
    alt.Chart(alt.topo_feature(world, 'countries')).mark_geoshape(
        fill='#2a1d0c', stroke='#706545', strokeWidth=0.5
    )
).properties(
    width=600,
    height=400
)
```

We can extend the map with a desired projection and draw the result. Here we apply a Natural Earth projection. The *sphere* layer provides the light blue background; the *graticule* layer provides the white geographic reference lines.

```
map.project(
    type='naturalEarth1', scale=110, translate=[300, 200]
).configure_view(stroke=None)
```

OUTPUT:



RESULT:

Thus the cartographic visualization for multiple datasets was executed successfully.

Ex.No.7	EXPLORE VARIOUS VARIABLE AND ROWFILTERS IN R FOR CLEANING DATA.APPLY VARIOUS PLOT FEATURES IN R ON SAMPLE DATA SETS AND VISUALIZE

AIM:

To explore various variable and rowfilters in r for cleaning data.

PROGRAM:

we will use inbuilt datasets(air quality datasets) which are available in R.

1. `head(airquality)`

OUTPUT:

Ozone	Solar.R	Wind	Temp	Month	Day
41	190	7.4	67	5	1
36	118	8.0	72	5	2
12	149	12.6	74	5	3
18	313	11.5	62	5	4
NA	NA	14.3	56	5	5
28	NA	14.9	66	5	6

2. Handling missing value in R

```
mean(airquality$Solar.R)
```

OUTPUT:

```
<NA>
```

3.Checking another column

```
mean(airquality$Wind)
```

OUTPUT:

9.95751633986928

4. Handling NA values

```
mean(airquality$Solar.R, na.rm = TRUE)
```

OUTPUT:

185.931506849315

Data Cleaning Operation

1. After checking the summary of the dataset and we found the number on NA in two columns(Ozone and Solar.R)

```
summary(airquality)
```

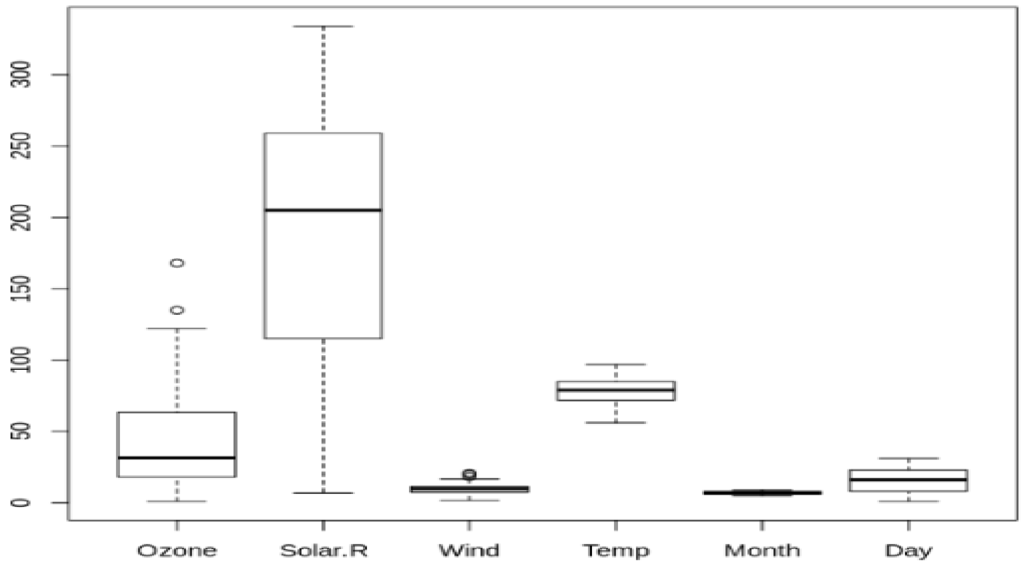
OUTPUT:

```
     Ozone           Solar.R           Wind             Temp      
 Min.   :  1.00   Min.   :  7.0   Min.   : 1.700   Min.   :56.00  
 1st Qu.: 18.00   1st Qu.:115.8   1st Qu.: 7.400   1st Qu.:72.00  
 Median : 31.50   Median :205.0   Median : 9.700   Median :79.00  
 Mean   : 42.13   Mean   :185.9   Mean   : 9.958   Mean   :77.88  
 3rd Qu.: 63.25   3rd Qu.:258.8   3rd Qu.:11.500   3rd Qu.:85.00  
 Max.   :168.00   Max.   :334.0   Max.   :20.700   Max.   :97.00  
 NA's   :37       NA's   :7                                       
     Month            Day      
 Min.   :5.000   Min.   : 1.0  
 1st Qu.:6.000   1st Qu.: 8.0  
 Median :7.000   Median :16.0  
 Mean   :6.993   Mean   :15.8  
 3rd Qu.:8.000   3rd Qu.:23.0  
 Max.   :9.000   Max.   :31.0  
```

2. We can get a clear visual of the irregular data using a boxplot.

```
boxplot(airquality)
```

OUTPUT:



RESULT:

Thus to explore various variable and rowfilters in r for cleaning data was executed successfully.

Ex.No.8	PERFORM EDA ON WINE QUALITY DATA SET

AIM:
To Perform EDA on Wine Quality Data Set.

PROGRAM:

EDA on the wine data set-

Firstly importing some essential libraries in Python.

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

Then load the data using the pandas' library.

```
In [2]: #load the data
        df=pd.read_csv("winequality_white.csv")

In [3]: #shape of data
        df.shape
Out[3]: (4898, 12)
```

The shape of the data is (4898,12), which shows there are 4898 rows and 12 columns in the data.
To know the columns of the data, we can do df.columns, it will give all the features name present in the data.

```
In [4]: #features in data
        df.columns

Out[4]: Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual su
        gar',
               'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'den
        sity',
               'pH', 'sulphates', 'alcohol', 'quality'],
              dtype='object')
```

Let’s see some data points present in the data.

```
In [5]: #few datapoints
df.head()
```

Out[5]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6

The describe () function in Python summarizes statistics. This function returns the count, mean, standard deviation, minimum and maximum values, and the quantiles of the data.

```
In [6]: df.describe()
```

Out[6]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
count	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000
mean	6.854788	0.278241	0.334192	6.391415	0.045772	35.308085	138.360657	0.994027	3.188267	0.489847	10.514267	5.877909
std	0.843868	0.100795	0.121020	5.072058	0.021848	17.007137	42.498065	0.002991	0.151001	0.114126	1.230621	0.885639
min	3.800000	0.080000	0.000000	0.600000	0.009000	2.000000	9.000000	0.987110	2.720000	0.220000	8.000000	3.000000
25%	6.300000	0.210000	0.270000	1.700000	0.036000	23.000000	108.000000	0.991723	3.090000	0.410000	9.500000	5.000000
50%	6.800000	0.260000	0.320000	5.200000	0.043000	34.000000	134.000000	0.993740	3.180000	0.470000	10.400000	6.000000
75%	7.300000	0.320000	0.390000	9.900000	0.050000	46.000000	167.000000	0.996100	3.280000	0.550000	11.400000	6.000000
max	14.200000	1.100000	1.660000	65.800000	0.346000	289.000000	440.000000	1.038980	3.820000	1.080000	14.200000	9.000000

As we can see here, mean value is less than the median value of each column.
There is a large difference between the 75th% tile and max values of residual sugar, free sulfur dioxide & total sulfur dioxide.
Let's check if there is any missing value in the data.

```
In [7]: #checking the missing data
df.isnull().any().any()
```

```
Out[7]: False
```

there is no missing data.

here is no missing data.
df.info return information about the data frame including the data types of each column and memory usage of the entire data.

```
In [8]: #data information
        df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4898 entries, 0 to 4897
Data columns (total 12 columns):
fixed acidity           4898 non-null float64
volatile acidity        4898 non-null float64
citric acid             4898 non-null float64
residual sugar          4898 non-null float64
chlorides               4898 non-null float64
free sulfur dioxide     4898 non-null float64
total sulfur dioxide    4898 non-null float64
density                 4898 non-null float64
pH                      4898 non-null float64
sulphates               4898 non-null float64
alcohol                 4898 non-null float64
quality                 4898 non-null int64
dtypes: float64(11), int64(1)
memory usage: 459.3 KB
```

Data has only float and integer values.
The below-shown function will print the number of unique values in each of the features.

```
In [9]: #number of unique value in each features
        for col in df.columns.values:
            print("Number of unique values of {} : {}".format(col, df[col]
```

```
Number of unique values of fixed acidity : 68
Number of unique values of volatile acidity : 125
Number of unique values of citric acid : 87
Number of unique values of residual sugar : 310
Number of unique values of chlorides : 160
Number of unique values of free sulfur dioxide : 132
Number of unique values of total sulfur dioxide : 251
Number of unique values of density : 890
Number of unique values of pH : 103
Number of unique values of sulphates : 79
Number of unique values of alcohol : 103
Number of unique values of quality : 7
```

The feature that has a maximum unique value is density.
The feature that has a minimum unique value is quality.

RESULT:

Thus to Perform EDA on Wine Quality Data Set was executed successfull

Ex.No.9	USE A CASE STUDY ON A DATA SET AND APPLY THE VARIOUS EDA AND VISUALIZATION TECHNIQUES AND PRESENT AN ANALYSIS REPORT

DATA VISUALIZATION

Data Visualization represents the text or numerical data in a visual format, which makes it easy to grasp the information the data express. We, humans, remember the pictures more easily than readable text, so Python provides us various libraries for data visualization like matplotlib, seaborn, plotly, etc. In this tutorial, we will use Matplotlib and seaborn for performing various techniques to explore data using various plots.

EXPLORATORY DATA ANALYSIS

Creating Hypotheses, testing various business assumptions while dealing with any Machine learning problem statement is very important and this is what EDA helps to accomplish. There are various tootle and techniques to understand your data, And the basic need is you should have the knowledge of Numpy for mathematical operations and Pandas for data manipulation.

```
import numpy as np
import pandas pd
import matplotlib.pyplot as plt
import seaborn as sns
from seaborn import load_dataset
#titanic dataset
data = pd.read_csv("titanic_train.csv")
#tips dataset
tips = load_dataset("tips")
```

UNIVARIATE ANALYSIS

Univariate analysis is the simplest form of analysis where we explore a single variable. Univariate analysis is performed to describe the data in a better way. we perform Univariate analysis of Numerical and categorical variables differently because plotting uses different plots.

CATEGORICAL DATA

A variable that has text-based information is referred to as categorical variables. let’s look at various plots which we can use for visualizing Categorical data.

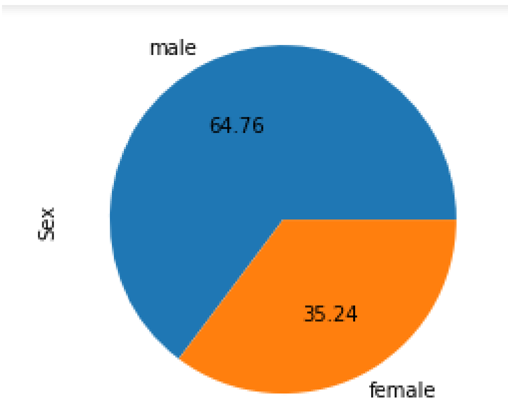
1) CountPlot

Countplot is basically a count of frequency plot in form of a bar graph. It plots the count of each category in a separate bar. When we use the pandas’ value counts function on any column, It is the same visual form of the value counts function. In our data-target variable is survived and it is categorical so let us plot a countplot of this.

2) Pie Chart

The pie chart is also the same as the countplot, only gives you additional information about the percentage presence of each category in data means which category is getting how much weightage in data.

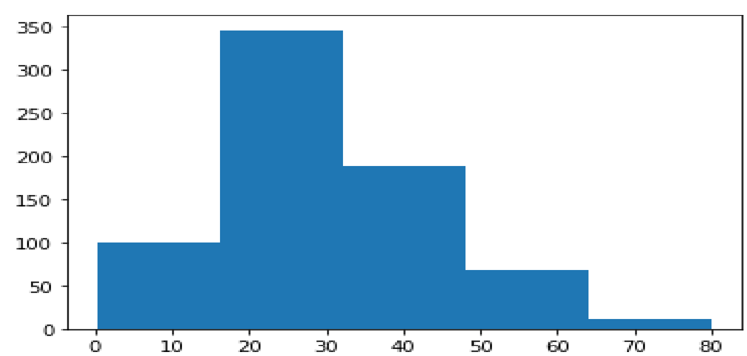
```
data['Sex'].value_counts().plot(kind="pie", autopct="%.2f")
plt.show()
```



Numerical Data
Analyzing Numerical data is important because understanding the distribution of variables helps to further process the data. Most of the time you will find much inconsistency with numerical data so do explore numerical variables.

1) Histogram
A histogram is a value distribution plot of numerical columns. It basically creates bins in various ranges in values and plots it where we can visualize how values are distributed. We can have a look where more values lie like in positive, negative, or at the center(mean). Let's have a look at the Age column.

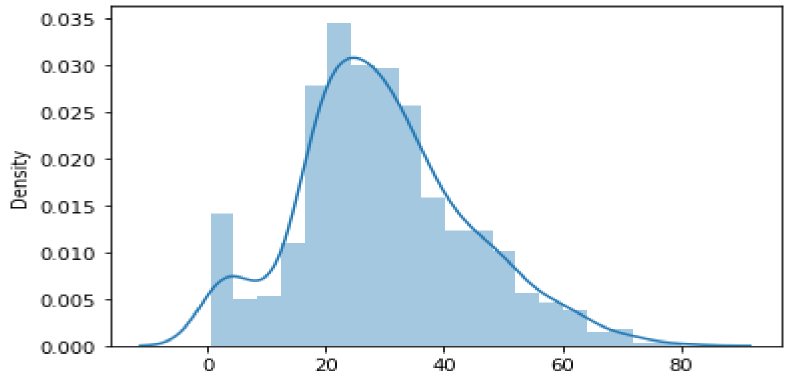
```
plt.hist(data['Age'], bins=5)
plt.show()
```



2) Distplot
Distplot is also known as the second Histogram because it is a slight improvement version of the Histogram. Distplot gives us a KDE(Kernel Density Estimation) over histogram which explains PDF(Probability

Density Function) which means what is the probability of each value occurring in this column. If you have study statistics before then definitely you should know about PDF function.

```
sns.distplot(data['Age'])
plt.show()
```



3) Boxplot

Boxplot is a very interesting plot that basically plots a 5 number summary. to get 5 number summary some terms we need to describe.

Median – Middle value in series after sorting
Percentile – Gives any number which is number of values present before this percentile like for example 50 under 25th percentile so it explains total of 50 values are there below 25th percentile
Minimum and Maximum – These are not minimum and maximum values, rather they describe the lower and upper boundary of standard deviation which is calculated using Interquartile range(IQR).

```
IQR = Q3 - Q1
Lower_boundary = Q1 - 1.5 * IQR
Upper_bounday = Q3 +  1.5 * IQR
```