MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

(AUTONOMOUS INSTITUTION – UGC, GOVT. OF INDIA)

MRCET CAMPUS

Department of CSE

(Emerging Technologies)

(DATA SCIENCE,IOT,CYBER SECURITY)

B.TECH(R-22 Regulation)
(III YEAR – II SEM)
(2024-2025)

MACHINE LEARNING LAB

Compiled by

V.Suneetha, Associate Professor

V.Bala, Associate Professor

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY
(Autonomous Institution – UGC, Govt. of India)
Recognized under 2(f) and 12(B) of UGC ACT 1956
(Affiliated to JNTUH, Hyderabad, Approved by AICTE-Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2015 Certified)
Maisammaguda, Dhulapally (Post Via. Hakimpet), Secunderabad–500100, Telangana State, India

EMERGING TECHNOLOGIES

Vision

- "To be at the forefront of Emerging Technologies and to evolve as a Centre of Excellence in Research, Learning and Consultancy to foster the students into globally competent professionals useful to the Society."

Mission

The department of CSE (Emerging Technologies) is committed to:

- To offer highest Professional and Academic Standards in terms of Personal growth and satisfaction.
- Make the society as the hub of emerging technologies and thereby capture opportunities in new age technologies.
- To create a benchmark in the areas of Research, Education and Public Outreach.
- To provide students a platform where independent learning and scientific study are encouraged with emphasis on latest engineering techniques.

QUALITY POLICY

- To pursue continual improvement of teaching learning process of Undergraduate and PostGraduateprogramsin Engineering &Managementvigorously.
- To provide state of art infrastructure and expertise to impart the quality education and research environment to students for a complete learning experiences.
- Developing students with a disciplined and integrated personality.
- To offer quality relevant and cost effective programmes to produce engineers as per requirements of the industry need.

For more information:www.mrcet.ac.in

List of Experiments

S. No	Name of the Program
1.	Implementation of Python Basic Libraries such as Statistics, Math, Numpy and Scipy a) Usage of methods such as floor(), ceil(), sqrt(), isqrt(), gcd() etc. b) Usage of attributes of array such as ndim, shape, size, methods such as sum(), mean(), sort(), sin() etc. c) Usage of methods such as det(), eig() etc. d) Consider a list datatype(1D) then reshape it into2D, 3D matrix using numpy e) Generater and ommatrices using numpy f) Find the determinant of a matrix using scipy g) Find eigen value and eigen vector of a matrix using scipy
2.	Implementation of Python Libraries for ML application such as Pandas and Matplotlib. a) Create a Series using pandas and display b) Access the index and the values of our Series c) Compare an array using Numpy with a series using pandas d) Define Series objects with individual indices e) Access single value of a series f) Load datasets in a Data frame variable using pandas g) Usage of different methods in Matplotlib.
3.	a) Creation and Loading different types of datasets in Python using the required libraries. i. Creation using pandas ii. Loading CSV dataset files using Pandas iii. Loading datasets using sklearn b) Write a python program to compute Mean, Median, Mode, Variance, Standard Deviation using Datasets c) Demonstrate various data pre-processing techniques for a given dataset. Write a python program to compute i. Reshaping the data, ii. Filtering the data, iii. Merging the data iv. Handling the missing values in datasets v. Feature Normalization: Min-max normalization
4	Implement Dimensionality reduction using Principle component Analysis method on a dataset iris
5	Write a program to demonstrate the working of the decision tree based ID3 algorithm by considering a dataset.
6.	Consider a dataset, use Random Forest to predict the output class. Vary the number of trees as follows and compare the results: i.20 ii.50 iii.100 iv.200 v.500

7.	Write a Python program to implement Simple Linear Regression and plot the graph.
8	Write a Python program to implement Logistic Regression for iris using sklearn and plot the confusion matrix.
9	Build KNN Classification model for a given dataset. Vary the number of k values as follows and compare the results: i. 1 ii. 3 iii. 5 iv. 7 v. 11
10	Implement Support Vector Machine for a dataset and compare the accuracy by applying the following kernel functions: i. Linear ii. Polynomial iii. RBF
11	Write a python program to implement K-Means clustering Algorithm. Vary the number of k values as follows and compare the results: i. 1 ii. 3 iii. 5

Week 1:

a)Implementation of Python Basic Libraries such as Math, Numpy and Scipy

Theory/Description:

- **Python Libraries**

  There are a lot of reasons why Python is popular among developers and one of them is that it has an amazingly large collection of libraries that users can work with. In this Python Library, we will discuss Python Standard library and different libraries offered by Python Programming Language: scipy, numpy, etc.

  We know that a module is a file with some Python code, and a package is a directory for sub packages and modules. A Python library is a reusable chunk of code that you may want to includein your programs/ projects. Here, a _library‘ loosely describes a collection of core modules. Essentially, then, a library is a collection of modules. A package is a library that can be installed using a package manager like npm.

- **Python Standard Library**

  The Python Standard Library is a collection of script modules accessible to a Python program to simplify the programming process and removing the need to rewrite commonly used commands. They can be used by 'calling/importing' them at the beginning of a script. A list of the Standard Library modules that are most important

   time
   sys
   csv
   math
   random
   pip
   os
   statistics
   tkinter
   socket

  To display a list of all available modules, use the following command in the Python console:
  >>> help('modules')

  - List of important Python Libraries
    - Python Libraries for Data Collection
      - Beautiful Soup
      - Scrapy
      - Selenium
    - Python Libraries for Data Cleaning and Manipulation
      - Pandas
      - PyOD

        - NumPy
        - Scipy
        - Spacy
    - Python Libraries for Data Visualization
        - Matplotlib
        - Seaborn
        - Bokeh
    - Python Libraries for Modeling
        - Scikit-learn
        - TensorFlow
        - PyTorch

Implementation of Python Basic Libraries such as Math, Numpy and Scipy

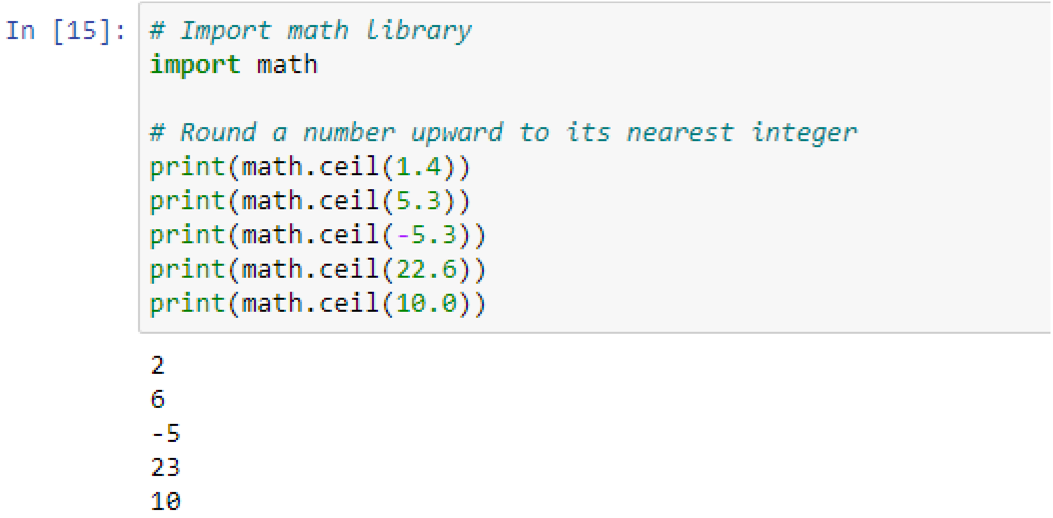
- **Python Math Library**

The math module is a standard module in Python and is always available. To use mathematical functions under this module, you have to import the module using import math. It gives access tothe underlying C library functions. This module does not support complex datatypes. The cmath module is the complex counterpart.

List of Functions in Python Math Module

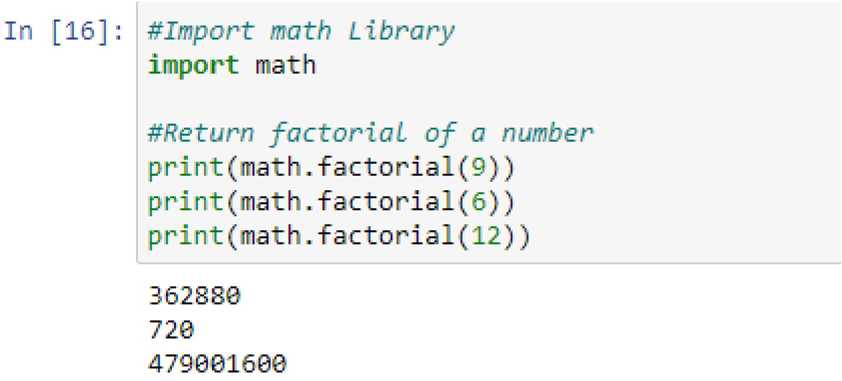
Function	Description
ceil(x)	Returns the smallest integer greater than or equal to x.
copysign(x, y)	Returns x with the sign of y
fabs(x)	Returns the absolute value of x
factorial(x)	Returns the factorial of x
floor(x)	Returns the largest integer less than or equal to x
fmod(x, y)	Returns the remainder when x is divided by y
frexp(x)	Returns the mantissa and exponent of x as the pair (m, e)
fsum(iterable)	Returns an accurate floating point sum of values in the iterable
isfinite(x)	Returns True if x is neither an infinity nor a NaN (Not a Number)
isinf(x)	Returns True if x is a positive or negative infinity
isnan(x)	Returns True if x is a NaN
ldexp(x, i)	Returns x * (2**i)
modf(x)	Returns the fractional and integer parts of x
trunc(x)	Returns the truncated integer value of x
exp(x)	Returns e**x
expm1(x)	Returns e**x - 1

Program-1



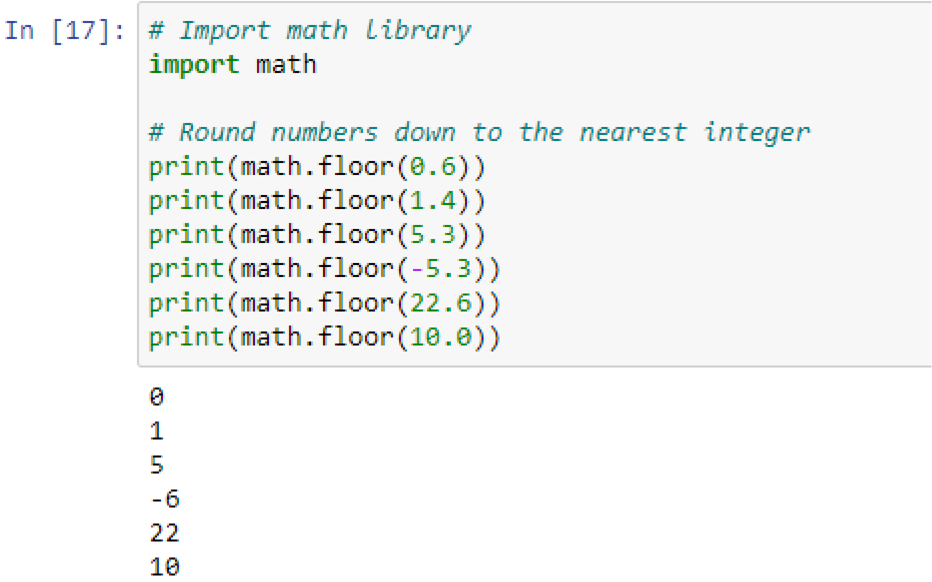
```
In [15]: # Import math library
         import math

         # Round a number upward to its nearest integer
         print(math.ceil(1.4))
         print(math.ceil(5.3))
         print(math.ceil(-5.3))
         print(math.ceil(22.6))
         print(math.ceil(10.0))

         2
         6
         -5
         23
         10
```

Program-2



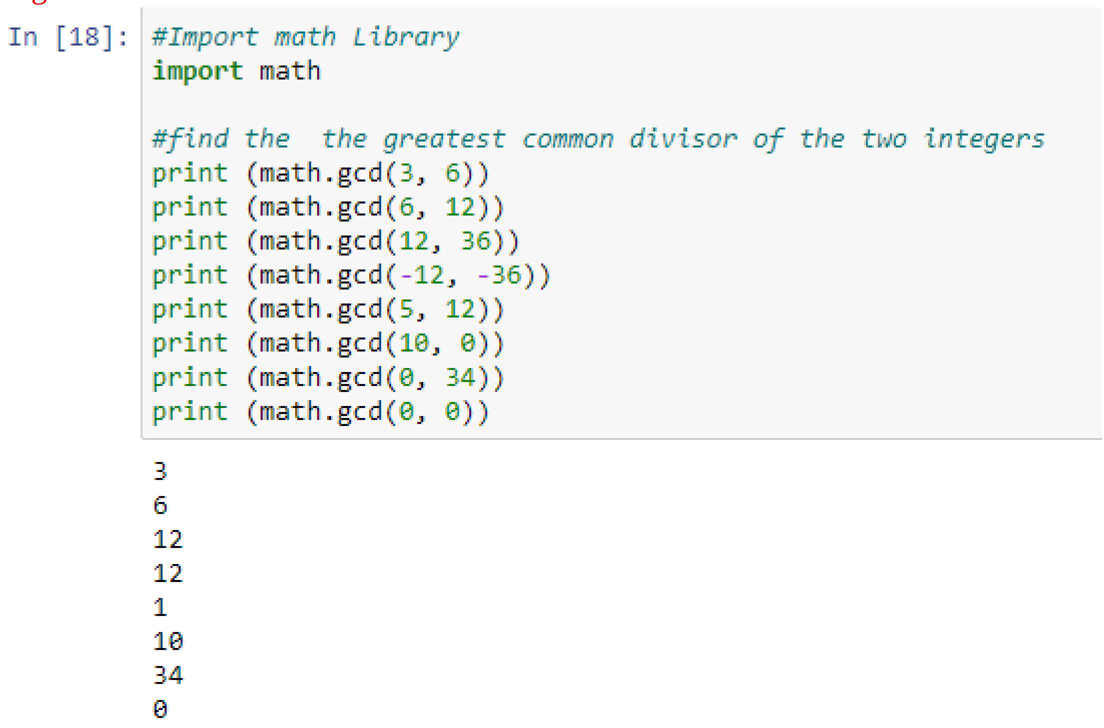
```
In [16]: #Import math Library
         import math

         #Return factorial of a number
         print(math.factorial(9))
         print(math.factorial(6))
         print(math.factorial(12))

         362880
         720
         479001600
```

Program-3



```
In [17]: # Import math library
         import math

         # Round numbers down to the nearest integer
         print(math.floor(0.6))
         print(math.floor(1.4))
         print(math.floor(5.3))
         print(math.floor(-5.3))
         print(math.floor(22.6))
         print(math.floor(10.0))

         0
         1
         5
         -6
         22
         10
```

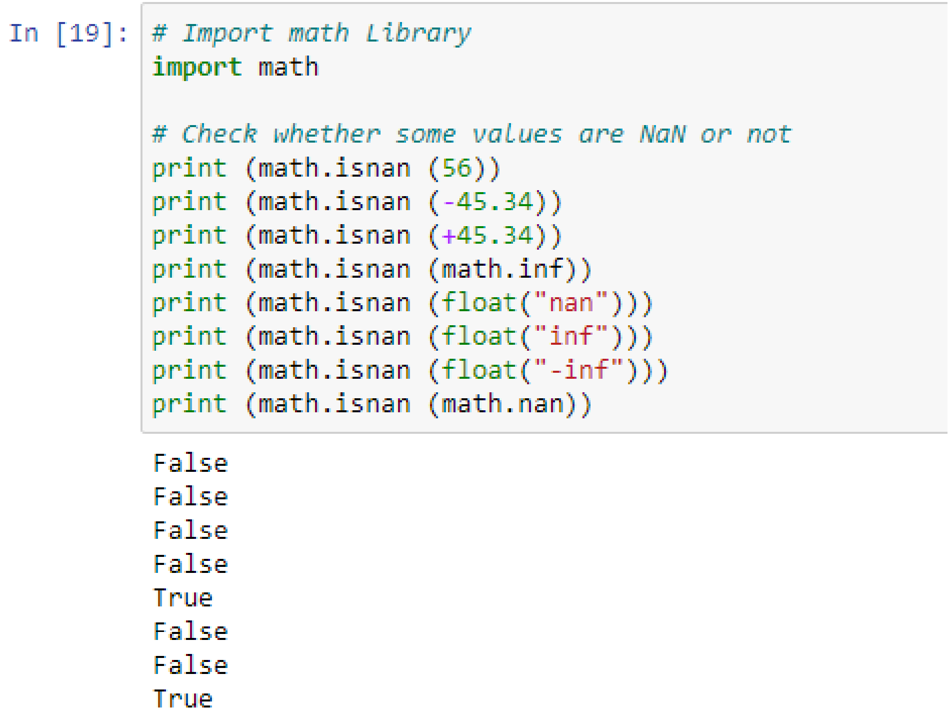
Program-4



```
In [18]: #Import math Library
         import math

         #find the  the greatest common divisor of the two integers
         print (math.gcd(3, 6))
         print (math.gcd(6, 12))
         print (math.gcd(12, 36))
         print (math.gcd(-12, -36))
         print (math.gcd(5, 12))
         print (math.gcd(10, 0))
         print (math.gcd(0, 34))
         print (math.gcd(0, 0))
```

```
3
6
12
12
1
10
34
0
```

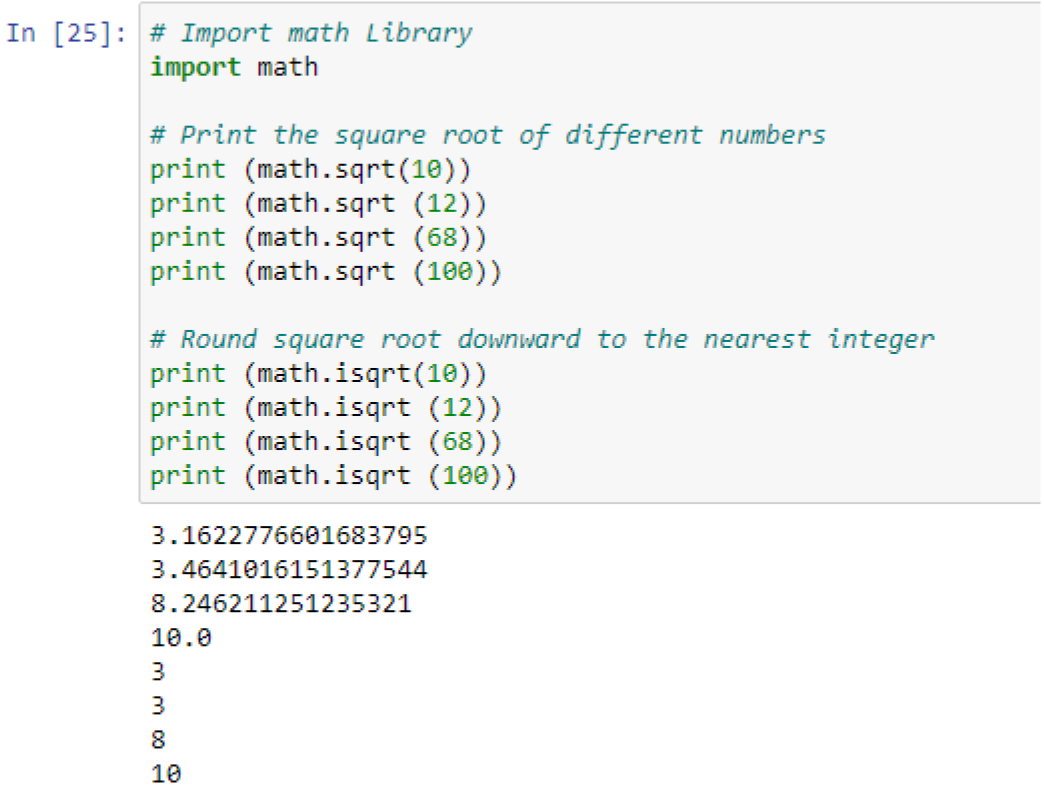
Program-5



```
In [19]: # Import math Library
         import math

         # Check whether some values are NaN or not
         print (math.isnan (56))
         print (math.isnan (-45.34))
         print (math.isnan (+45.34))
         print (math.isnan (math.inf))
         print (math.isnan (float("nan")))
         print (math.isnan (float("inf")))
         print (math.isnan (float("-inf")))
         print (math.isnan (math.nan))
```

```
False
False
False
False
True
False
False
True
```

Program-6



```
In [25]: # Import math Library
         import math

         # Print the square root of different numbers
         print (math.sqrt(10))
         print (math.sqrt (12))
         print (math.sqrt (68))
         print (math.sqrt (100))

         # Round square root downward to the nearest integer
         print (math.isqrt(10))
         print (math.isqrt (12))
         print (math.isqrt (68))
         print (math.isqrt (100))
```

```
3.1622776601683795
3.4641016151377544
8.246211251235321
10.0
3
3
8
10
```

- **Python Numpy Library**

NumPy is an open source library available in Python that aids in mathematical, scientific, engineering, and data science programming. NumPy is an incredible library to perform mathematical and statisticaloperations. It works perfectly well for multi-dimensional arrays and matrices multiplication

For any scientific project, NumPy is the tool to know. It has been built to work with the N- dimensional array, linear algebra, random number, Fourier transform, etc. It can be integrated toC/C++ and Fortran.

NumPy is a programming language that deals with multi-dimensional arrays and matrices. On top ofthe arrays and matrices, NumPy supports a large number of mathematical operations.

NumPy is memory efficiency, meaning it can handle the vast amount of data more accessible than anyother library. Besides, NumPy is very convenient to work with, especially for matrix multiplication and reshaping. On top of that, NumPy is fast. In fact, TensorFlow and Scikit learn to use NumPy arrayto compute the matrix multiplication in the back end.

- **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 positiveintegers.
  - 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**.

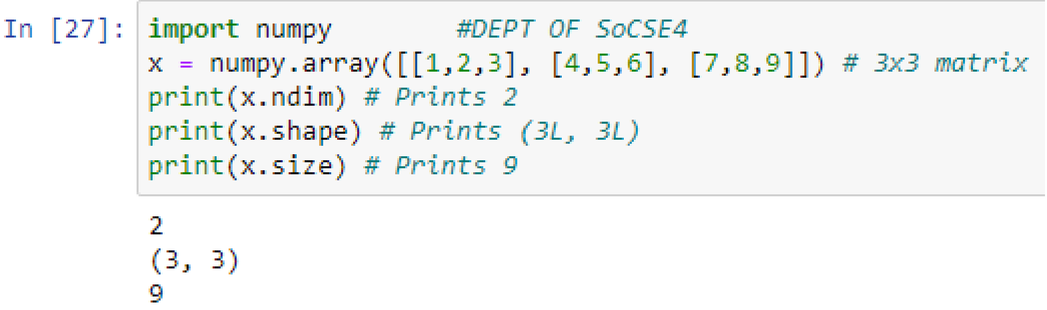
  We use python numpy array instead of a list because of the below three reasons:

  1. Less Memory
  2. Fast
  3. Convenient

- **Numpy Functions**

  Numpy arrays carry attributes around with them. The most important
  ones are:ndim: The number of axes or rank of the array
  shape: A tuple containing the length in each
  dimensionsize: The total number of elements

Program-1

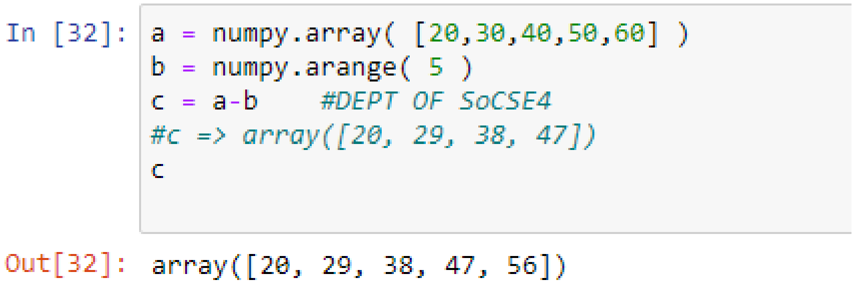


```
In [27]: import numpy        #DEPT OF SoCSE4
         x = numpy.array([[1,2,3], [4,5,6], [7,8,9]]) # 3x3 matrix
         print(x.ndim) # Prints 2
         print(x.shape) # Prints (3L, 3L)
         print(x.size) # Prints 9

         2
         (3, 3)
         9
```

Can be used just like Python lists
x[1] will access the second element
x[-1] will access the last element

Program-2

Arithmetic operations apply element wise

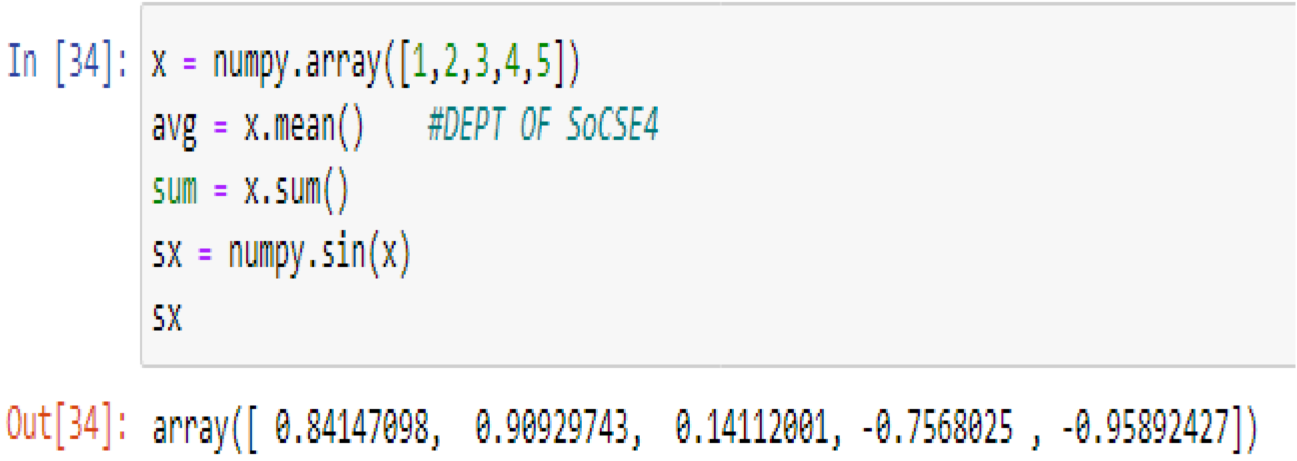


```
In [32]: a = numpy.array( [20,30,40,50,60] )
         b = numpy.arange( 5 )
         c = a-b     #DEPT OF SoCSE4
         #c => array([20, 29, 38, 47])
         c

Out[32]: array([20, 29, 38, 47, 56])
```

- **Built-in Methods**

Many standard numerical functions are available as methods out of the box:

Program-3



```
In [34]: x = numpy.array([1,2,3,4,5])
         avg = x.mean()    #DEPT OF SoCSE4
         sum = x.sum()
         sx = numpy.sin(x)
         sx

Out[34]: array([ 0.84147098,  0.90929743,  0.14112001, -0.7568025 , -0.95892427])
```

- **Python Scipy Library**

SciPy is an Open Source Python-based library, which is used in mathematics, scientific computing, Engineering, and technical computing. SciPy also pronounced as "Sigh Pi."

- SciPy contains varieties of sub packages which help to solve the most common issue related to Scientific Computation.
- SciPy is the most used Scientific library only second to GNU Scientific Library for C/C++ or Matlab's.
- Easy to use and understand as well as fast computational power.
- It can operate on an array of NumPy library.

Numpy VS SciPyNumpy:

1. Numpy is written in C and use for mathematical or numeric calculation.
2. It is faster than other Python Libraries
3. Numpy is the most useful library for Data Science to perform basic calculations.
4. Numpy contains nothing but array data type which performs the most basic operation like sorting,shaping, indexing, etc.

SciPy:

1. SciPy is built in top of the NumPy
2. SciPy is a fully-featured version of Linear Algebra while Numpy contains only a few features.
3. Most new Data Science features are available in Scipy rather than Numpy.

Linear Algebra with SciPy

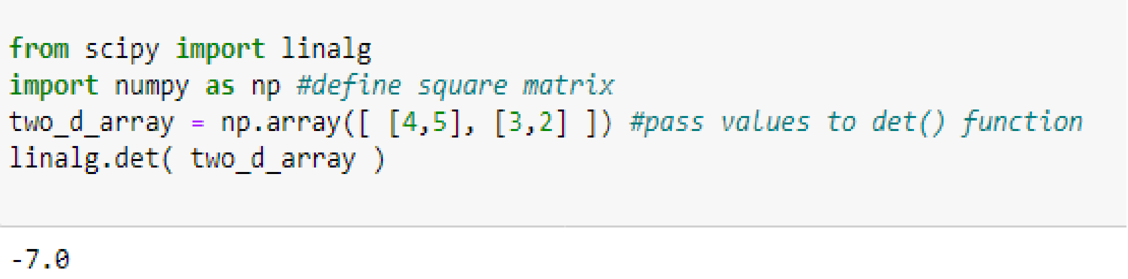
1. Linear Algebra of SciPy is an implementation of BLAS and ATLAS LAPACK libraries.
2. Performance of Linear Algebra is very fast compared to BLAS and LAPACK.

Linear algebra routine accepts two-dimensional array object and output is also a two-dimensional array.

Now let's do some test with **scipy.linalg,**

Calculating **determinant** of a two-dimensional matrix,

Program-1



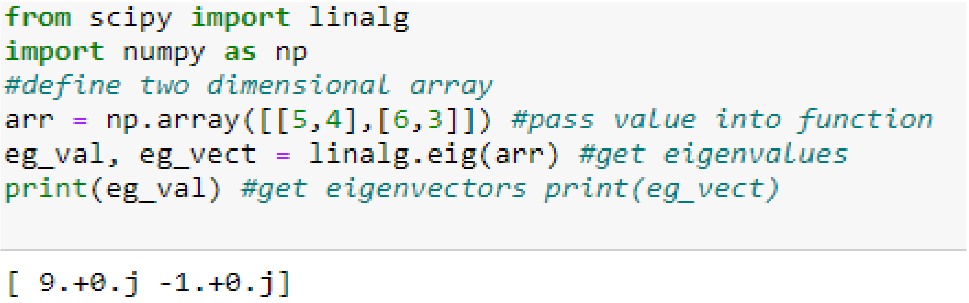
```
from scipy import linalg
import numpy as np #define square matrix
two_d_array = np.array([ [4,5], [3,2] ]) #pass values to det() function
linalg.det( two_d_array )
```

```
-7.0
```

Eigenvalues and Eigenvector – scipy.linalg.eig()

- The most common problem in linear algebra is eigenvalues and eigenvector which can beeasily solved using **eig()** function.
- Now lets we find the Eigenvalue of (**X**) and correspond eigenvector of a two-dimensionalsquare matrix.

Program-2



```
from scipy import linalg
import numpy as np
#define two dimensional array
arr = np.array([[5,4],[6,3]]) #pass value into function
eg_val, eg_vect = linalg.eig(arr) #get eigenvalues
print(eg_val) #get eigenvectors print(eg_vect)
```

```
[ 9.+0.j -1.+0.j]
```

<u>Exercise programs:</u>

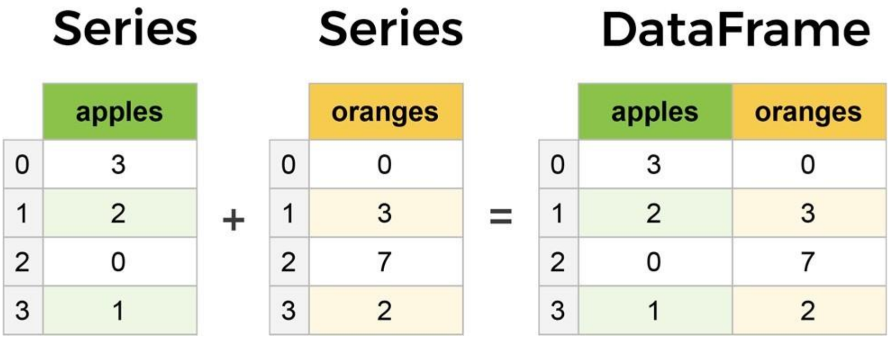
1. consider a list datatype then reshape it into 2d,3d matrix using numpy
2. Genrate random matrices using numpy
3. Find the determinant of a matrix using scipy
4. Find eigenvalue and eigenvector of a matrix using scipy

Week 2:

Implementation of Python Libraries for ML application such as Pandas and Matplotlib.

- **Pandas Library**

The primary two components of pandas are the Series and DataFrame.
A Series is essentially a column, and a DataFrame is a multi-dimensional table made up of acollection of Series.
DataFrames and Series are quite similar in that many operations that you can do with oneyou can do with the other, such as filling in null values and calculating the mean.



Series

	apples
0	3
1	2
2	0
3	1

+

Series

	oranges
0	0
1	3
2	7
3	2

=

DataFrame

	apples	oranges
0	3	0
1	2	3
2	0	7
3	1	2

- **Reading data from CSVs**

With CSV files all you need is a single line to load in the data:

```
df =
pd.read_csv('purchases.csv')df
```

Let's load in the IMDB movies dataset to begin:

```
movies_df = pd.read_csv("IMDB-Movie-Data.csv", index_col="Title")
```

We're loading this dataset from a CSV and designating the movie titles to be our index.

- **Viewing your data**

The first thing to do when opening a new dataset is print out a few rows to keep as a visualreference. We accomplish this with .head():

```
movies_df.head()
```

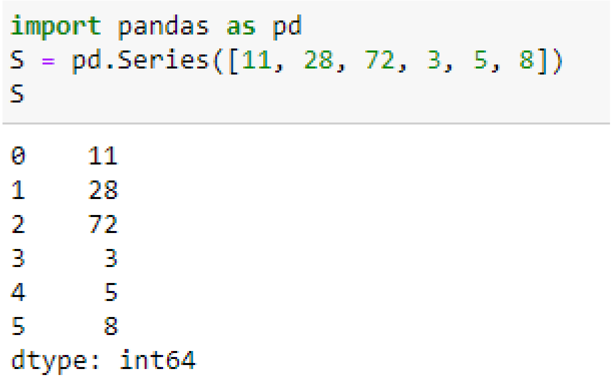
Another fast and useful attribute is .shape, which outputs just a tuple of (rows, columns):

```
movies_df.shape
```

Note that .shape has no parentheses and is a simple tuple of format (rows, columns). So we have1000 rows and 11 columns in our movies DataFrame.

You'll be going to .shape a lot when cleaning and transforming data. For example, you might filtersome rows based on some criteria and then want to know quickly how many rows were removed.

Program-1



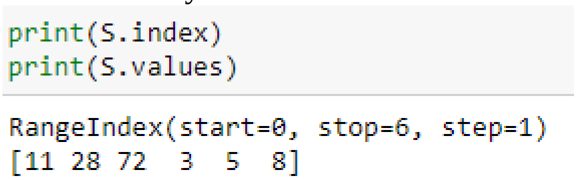
```
import pandas as pd
S = pd.Series([11, 28, 72, 3, 5, 8])
S
```

```
0    11
1    28
2    72
3     3
4     5
5     8
dtype: int64
```

We haven't defined an index in our example, but we see two columns in our output: The right column contains our data, whereas the left column contains the index. Pandas created a default index starting with 0 going to 5, which is the length of the data minus 1.

Program-2

We can directly access the index and the values of our Series S:

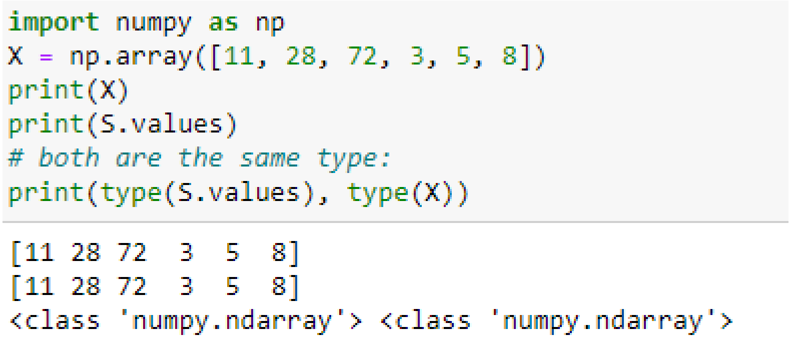


```
print(S.index)
print(S.values)
```

```
RangeIndex(start=0, stop=6, step=1)
[11 28 72  3  5  8]
```

Program-3

If we compare this to creating an array in numpy, we will find lots of similarities:

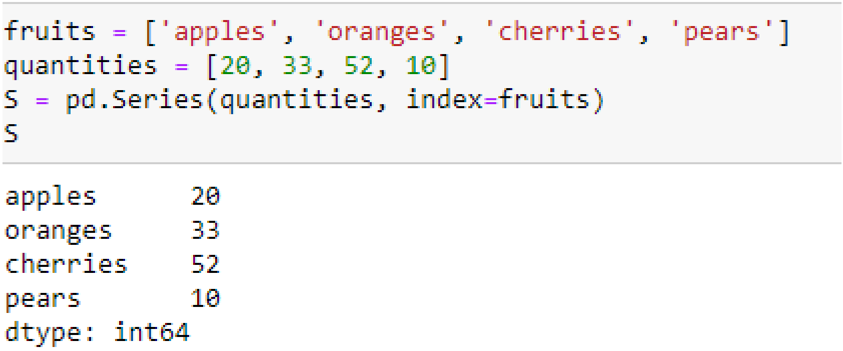


```
import numpy as np
X = np.array([11, 28, 72, 3, 5, 8])
print(X)
print(S.values)
# both are the same type:
print(type(S.values), type(X))
```

```
[11 28 72  3  5  8]
[11 28 72  3  5  8]
<class 'numpy.ndarray'> <class 'numpy.ndarray'>
```

So far our Series have not been very different to ndarrays of Numpy. This changes, as soon as we start defining Series objects with individual indices:

Program-4



```
fruits = ['apples', 'oranges', 'cherries', 'pears']
quantities = [20, 33, 52, 10]
S = pd.Series(quantities, index=fruits)
S
```

```
apples      20
oranges     33
cherries    52
pears       10
dtype: int64
```

Program-5

A big advantage to NumPy arrays is obvious from the previous example: We can use arbitrary indices. If we add two series with the same indices, we get a new series with the same index and the correponding values will be added:

```
fruits = ['apples', 'oranges', 'cherries', 'pears']
S = pd.Series([20, 33, 52, 10], index=fruits)
S2 = pd.Series([17, 13, 31, 32], index=fruits)
print(S + S2)
print("sum of S: ", sum(S))
```

OUTPUT:

```
apples      37
oranges     46
cherries    83
pears       42
dtype: int64
sum of S:  115
```

Program-6

The indices do not have to be the same for the Series addition. The index will be the "union" of both indices. If an index doesn't occur in both Series, the value for this Series will be NaN:

```
fruits = ['peaches', 'oranges', 'cherries', 'pears']
fruits2 = ['raspberries', 'oranges', 'cherries', 'pears']

S = pd.Series([20, 33, 52, 10], index=fruits)
S2 = pd.Series([17, 13, 31, 32], index=fruits2)
print(S + S2)
```

OUTPUT:

```
cherries      83.0
oranges       46.0
peaches        NaN
pears         42.0
raspberries    NaN
dtype: float64
```

Program-7

In principle, the indices can be completely different, as in the following example. We have two indices. One is the Turkish translation of the English fruit names:

```
fruits = ['apples', 'oranges', 'cherries', 'pears']

fruits_tr = ['elma', 'portakal', 'kiraz', 'armut']

S = pd.Series([20, 33, 52, 10], index=fruits)
S2 = pd.Series([17, 13, 31, 32], index=fruits_tr)
print(S + S2)
```

OUTPUT:

```
apples      NaN
armut       NaN
cherries    NaN
elma        NaN
kiraz       NaN
oranges     NaN
pears       NaN
portakal    NaN
dtype: float64
```

Program-8

Indexing

It's possible to access single values of a Series.

```
print(S['apples'])
```

OUTPUT:

```
20
```

- **Matplotlib Library**

  Pyplot is a module of Matplotlib which provides simple functions to add plot elementslike lines, images, text, etc. to the current axes in the current figure.

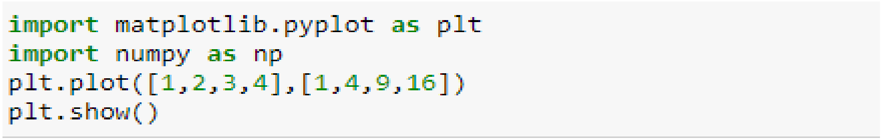
  - **Make a simple plot**
    import matplotlib.pyplot as pltimport
    numpy as np

  List of all the methods as they appeared.

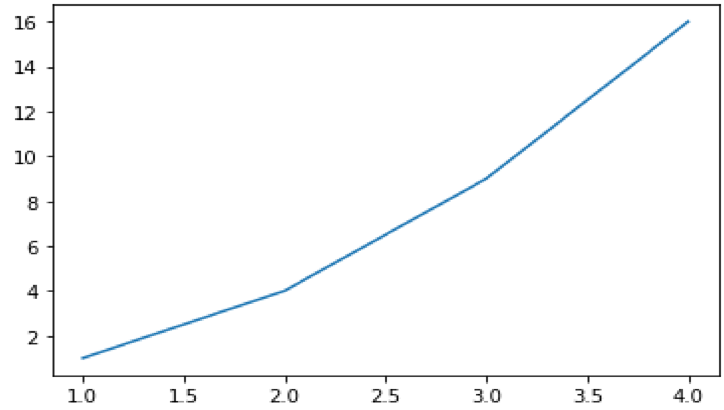
  - plot(x-axis values, y-axis values) — plots a simple line graph with x-axis values against y-axis values
  - show() — displays the graph
  - title(—string‖) — set the title of the plot as specified by the string
  - xlabel(—string‖) — set the label for x-axis as specified by the string
  - ylabel(—string‖) — set the label for y-axis as specified by the string
  - figure() — used to control a figure level attributes
  - subplot(nrows, ncols, index) — Add a subplot to the current figure
  - suptitle(—string‖) — It adds a common title to the figure specified by the string
  - subplots(nrows, ncols, figsize) — a convenient way to create subplots, in a single call. It returns a tuple of a figure and number of axes.
  - set_title(—string‖) — an axes level method used to set the title of subplots in a figure
  - bar(categorical variables, values, color) — used to create vertical bar graphs
  - barh(categorical variables, values, color) — used to create horizontal bar graphs
  - legend(loc) — used to make legend of the graph
  - xticks(index, categorical variables) — Get or set the current tick locations and labels of the x-axis
  - pie(value, categorical variables) — used to create a pie chart
  - hist(values, number of bins) — used to create a histogram
  - xlim(start value, end value) — used to set the limit of values of the x-axis
  - ylim(start value, end value) — used to set the limit of values of the y-axis
  - scatter(x-axis values, y-axis values) — plots a scatter plot with x-axis values against y-axis values
  - axes() — adds an axes to the current figure
  - set_xlabel(—string‖) — axes level method used to set the x-label of the plot specified as a string
  - set_ylabel(—string‖) — axes level method used to set the y-label of the plot specified as a string
  - scatter3D(x-axis values, y-axis values) — plots a three-dimensional scatter plot with x-axis values against y-axis values
  - plot3D(x-axis values, y-axis values) — plots a three-dimensional line graph with x-axis values against y-axis values

Here we import Matplotlib‘s Pyplot module and Numpy library as most of the data thatwe will be working with will be in the form of arrays only.

Program-1

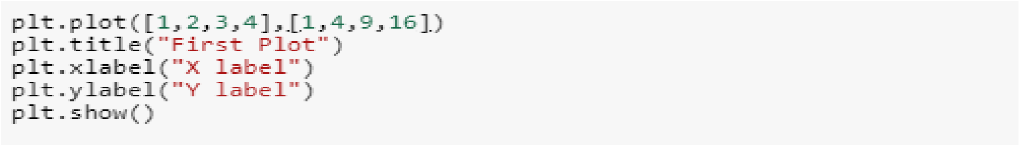


```
import matplotlib.pyplot as plt
import numpy as np
plt.plot([1,2,3,4],[1,4,9,16])
plt.show()
```

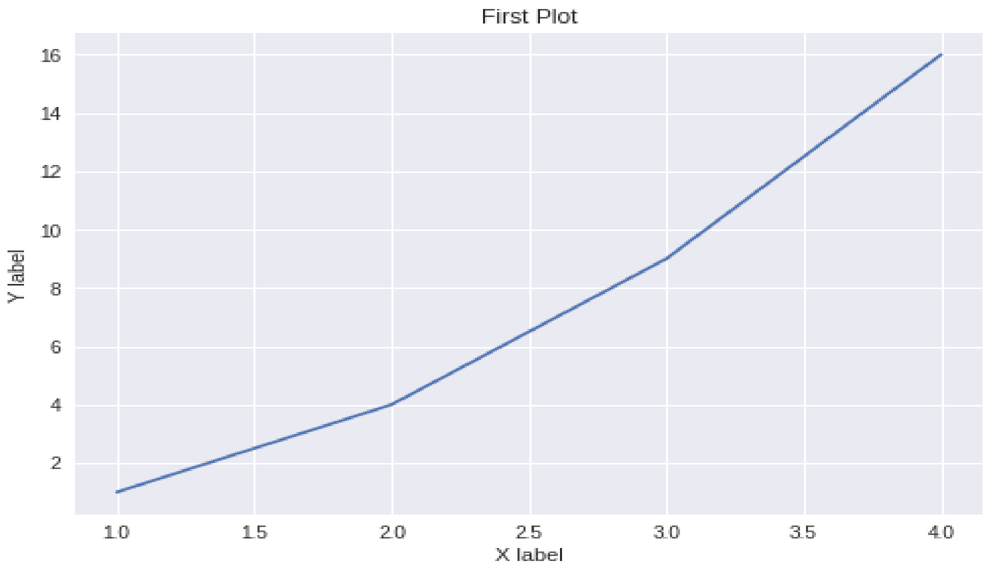


Program-2

We pass two arrays as our input arguments to Pyplot‘s plot() method and use show() method to invoke the required plot. Here note that the first array appears on the x-axis andsecond array appears on the y-axis of the plot. Now that our first plot is ready, let us add the title, and name x-axis and y axis using methods title(), xlabel() and ylabel() respectively.

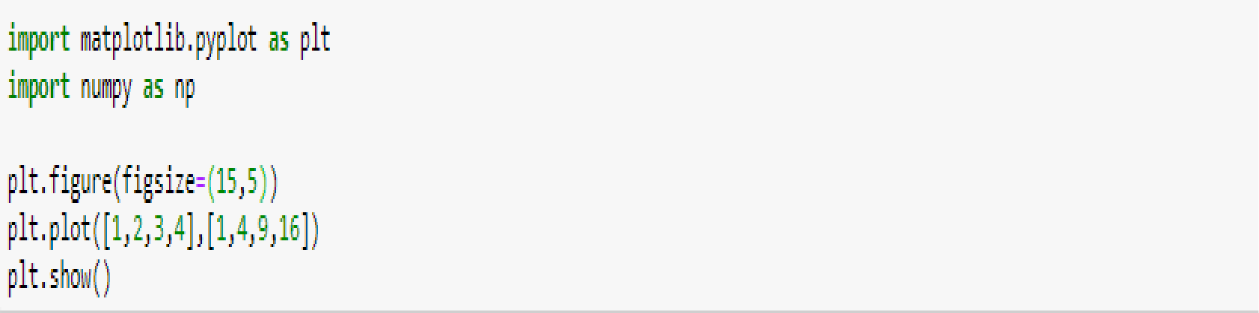


```
plt.plot([1,2,3,4],[1,4,9,16])
plt.title("First Plot")
plt.xlabel("X label")
plt.ylabel("Y label")
plt.show()
```

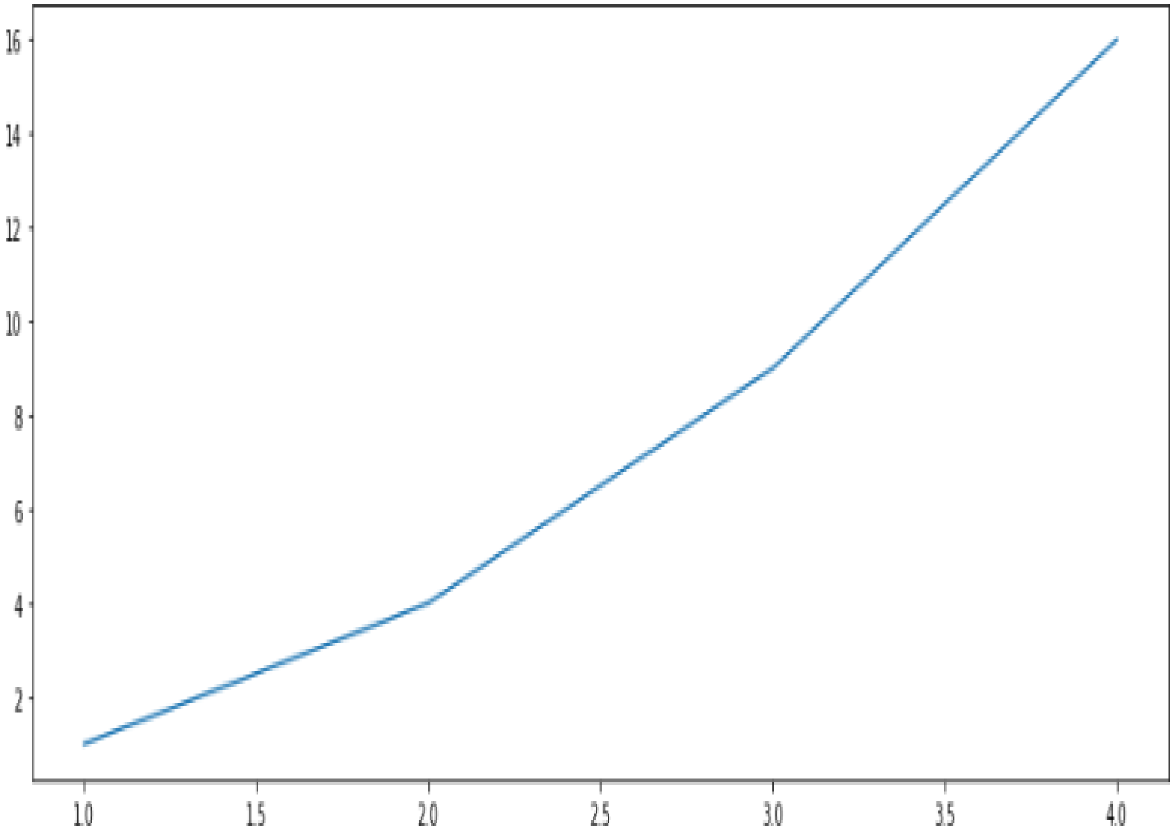


Program-3

We can also specify the size of the figure using method figure()and passing the valuesas a tuple of the length of rows and columns to the argument figsize

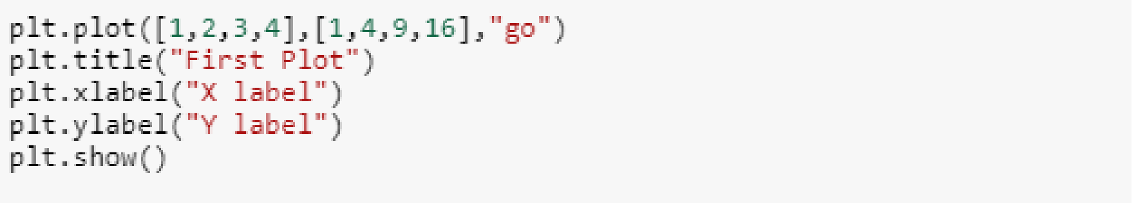


```
import matplotlib.pyplot as plt
import numpy as np

plt.figure(figsize=(15,5))
plt.plot([1,2,3,4],[1,4,9,16])
plt.show()
```

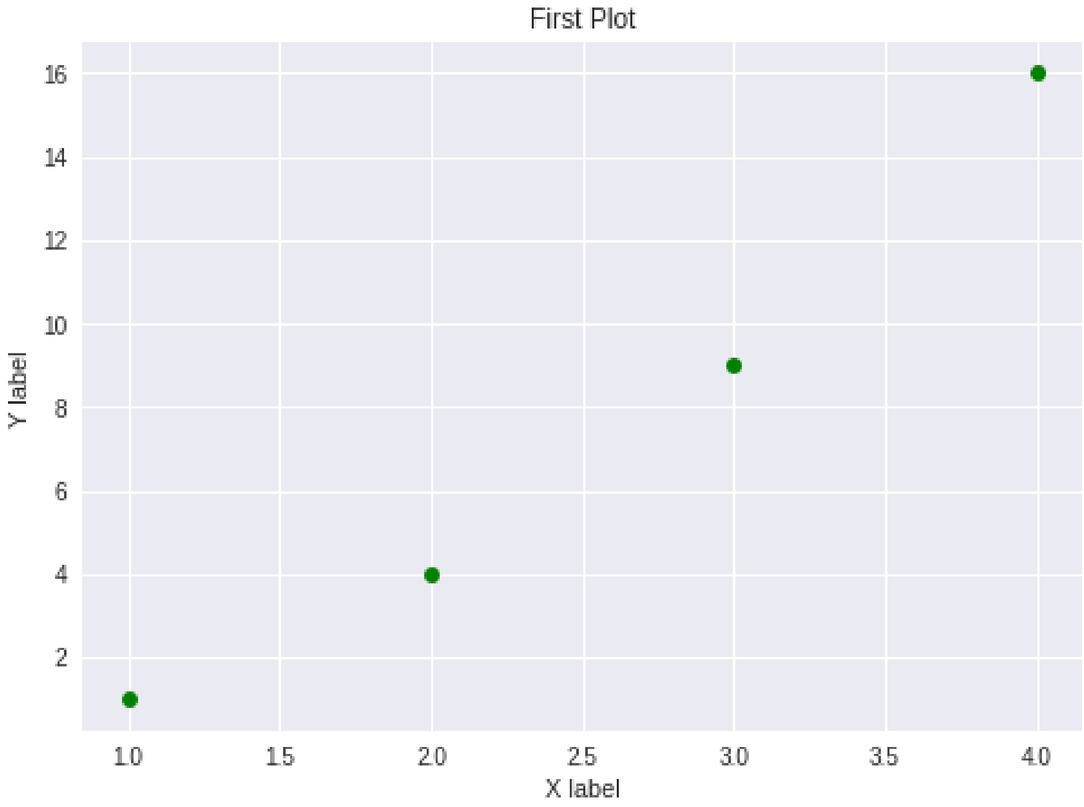


Program-4

With every X and Y argument, you can also pass an optional third argument in the formof a string which indicates the colour and line type of the plot. The default format is **b-** which means a solid blue line. In the figure below we use **go** which means green circles.Likewise, we can make many such combinations to format our plot.



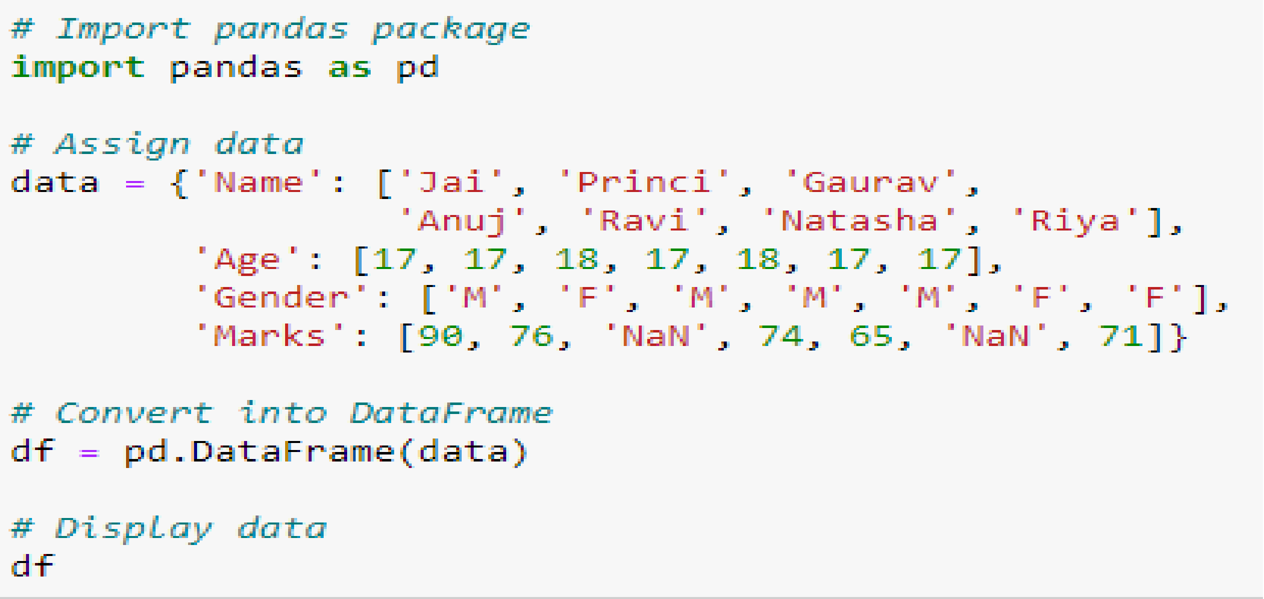
```
plt.plot([1,2,3,4],[1,4,9,16],"go")
plt.title("First Plot")
plt.xlabel("X label")
plt.ylabel("Y label")
plt.show()
```



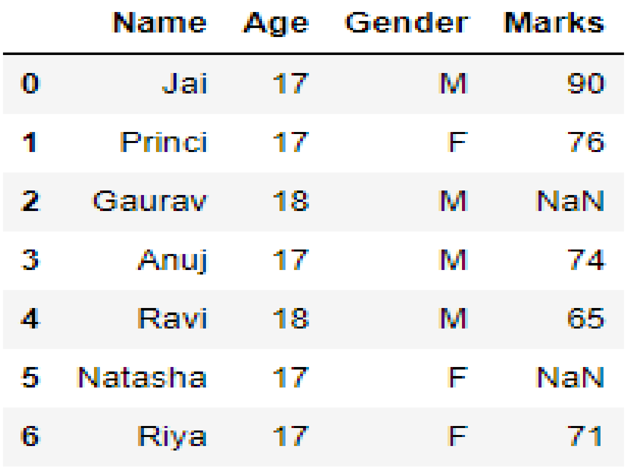
Week 3: Creation and Loading different datasets in Python

Program-1

Method-I



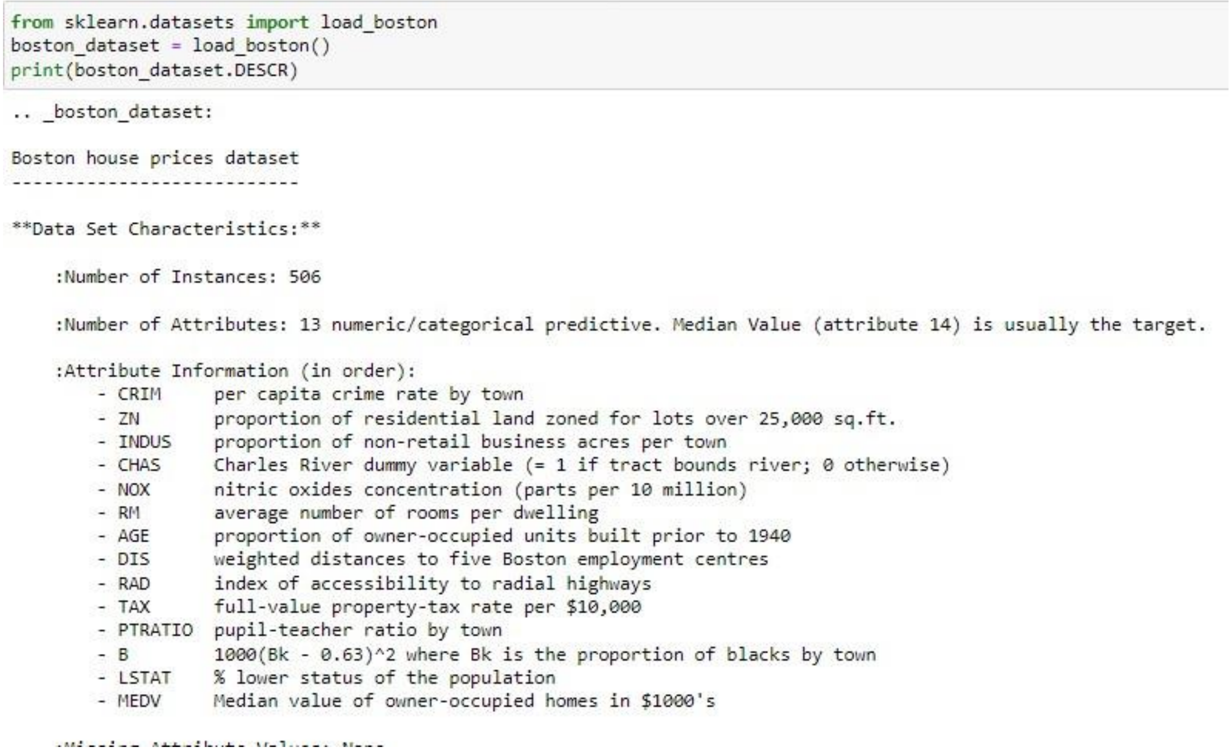
```
# Import pandas package
import pandas as pd

# Assign data
data = {'Name': ['Jai', 'Princi', 'Gaurav',
                 'Anuj', 'Ravi', 'Natasha', 'Riya'],
        'Age': [17, 17, 18, 17, 18, 17, 17],
        'Gender': ['M', 'F', 'M', 'M', 'M', 'F', 'F'],
        'Marks': [90, 76, 'NaN', 74, 65, 'NaN', 71]}

# Convert into DataFrame
df = pd.DataFrame(data)

# Display data
df
```



```
       Name  Age  Gender  Marks
0       Jai   17       M     90
1    Princi   17       F     76
2    Gaurav   18       M    NaN
3      Anuj   17       M     74
4      Ravi   18       M     65
5   Natasha   17       F    NaN
6      Riya   17       F     71
```

Program-2

Method-II:

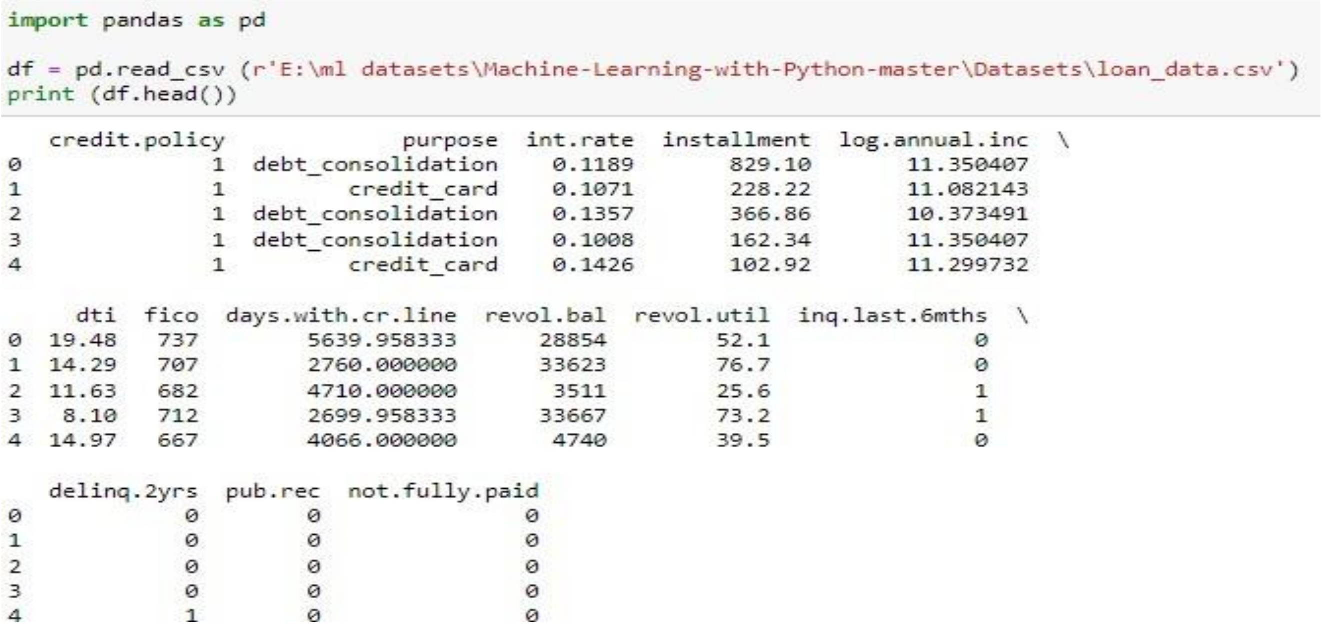


```
from sklearn.datasets import load_boston
boston_dataset = load_boston()
print(boston_dataset.DESCR)
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 14) is usually the target.

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

    :Missing Attribute Values: None
```

Program-3 **Uploading csv file:**

Method-III:



```
import pandas as pd

df = pd.read_csv (r'E:\ml datasets\Machine-Learning-with-Python-master\Datasets\loan_data.csv')
print (df.head())
```

```
   credit.policy             purpose  int.rate  installment  log.annual.inc  \
0              1  debt_consolidation    0.1189       829.10       11.350407
1              1         credit_card    0.1071       228.22       11.082143
2              1  debt_consolidation    0.1357       366.86       10.373491
3              1  debt_consolidation    0.1008       162.34       11.350407
4              1         credit_card    0.1426       102.92       11.299732

     dti  fico  days.with.cr.line  revol.bal  revol.util  inq.last.6mths  \
0  19.48   737        5639.958333      28854        52.1               0
1  14.29   707        2760.000000      33623        76.7               0
2  11.63   682        4710.000000       3511        25.6               1
3   8.10   712        2699.958333      33667        73.2               1
4  14.97   667        4066.000000       4740        39.5               0

   delinq.2yrs  pub.rec  not.fully.paid
0            0        0               0
1            0        0               0
2            0        0               0
3            0        0               0
4            1        0               0
```

b) Write a python program to compute Mean, Median, Mode, Variance, Standard Deviation using Datasets

- **Python Statistics library**

  This module provides functions for calculating mathematical statistics of numeric (Real-valued) data. The statistics module comes with very useful functions like: Mean, median, mode, standard deviation,and variance.

  The four functions we'll use in this post are common in statistics:

  1. mean - average value
  2. median - middle value
  3. mode - most often value
  4. standard deviation - spread of values

- **Averages and measures of central location**

  These functions calculate an average or typical value from a population or

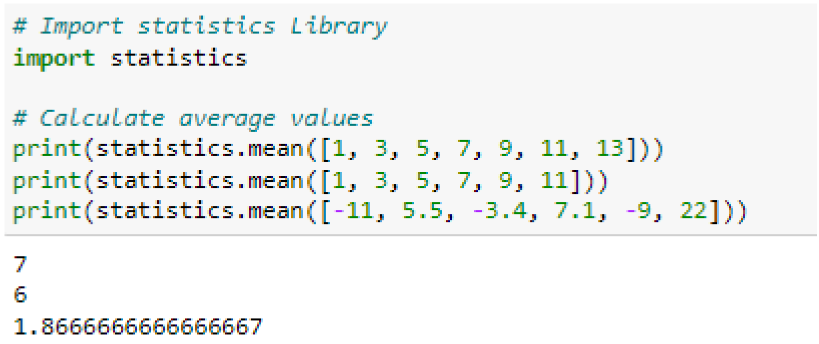
sample.mean()	Arithmetic mean (―average‖) of data.
harmonic_mean()	Harmonic mean of data.
median()	Median (middle value) of
data.median_low()	Low median of data.
median_high()	High median of data.
median_grouped()	Median, or 50th percentile, of grouped
data.mode()	Mode (most common value) of discrete data.

- **Measures of spread**

  These functions calculate a measure of how much the population or sample tends to deviate fromthe typical or average values.

pstdev()	Population standard deviation of data.
pvariance()	Population variance of data.
stdev()	Sample standard deviation of data.
variance()	Sample variance of data.

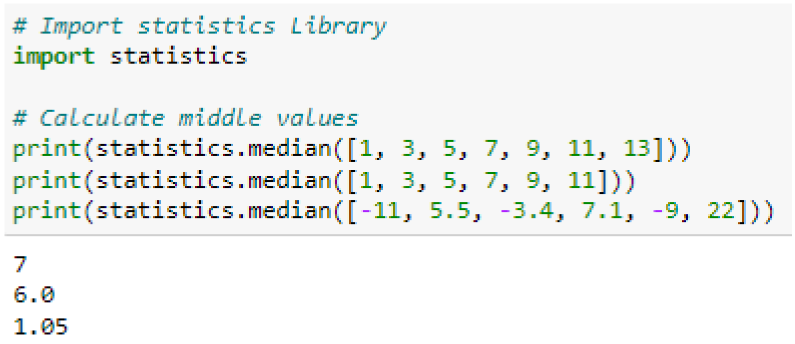
Program-1



```
# Import statistics Library
import statistics

# Calculate average values
print(statistics.mean([1, 3, 5, 7, 9, 11, 13]))
print(statistics.mean([1, 3, 5, 7, 9, 11]))
print(statistics.mean([-11, 5.5, -3.4, 7.1, -9, 22]))
```

```
7
6
1.8666666666666667
```

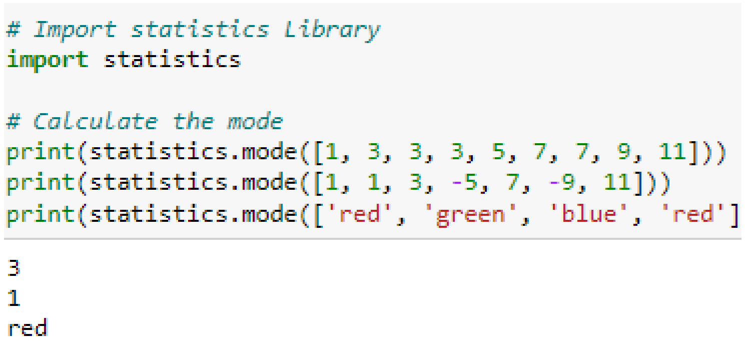
Program-2



```
# Import statistics Library
import statistics

# Calculate middle values
print(statistics.median([1, 3, 5, 7, 9, 11, 13]))
print(statistics.median([1, 3, 5, 7, 9, 11]))
print(statistics.median([-11, 5.5, -3.4, 7.1, -9, 22]))
```

```
7
6.0
1.05
```

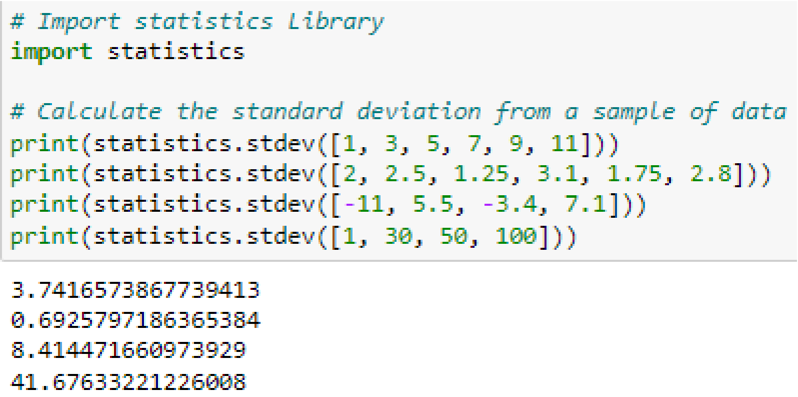
Program-3



```
# Import statistics Library
import statistics

# Calculate the mode
print(statistics.mode([1, 3, 3, 3, 5, 7, 7, 9, 11]))
print(statistics.mode([1, 1, 3, -5, 7, -9, 11]))
print(statistics.mode(['red', 'green', 'blue', 'red']
```

```
3
1
red
```

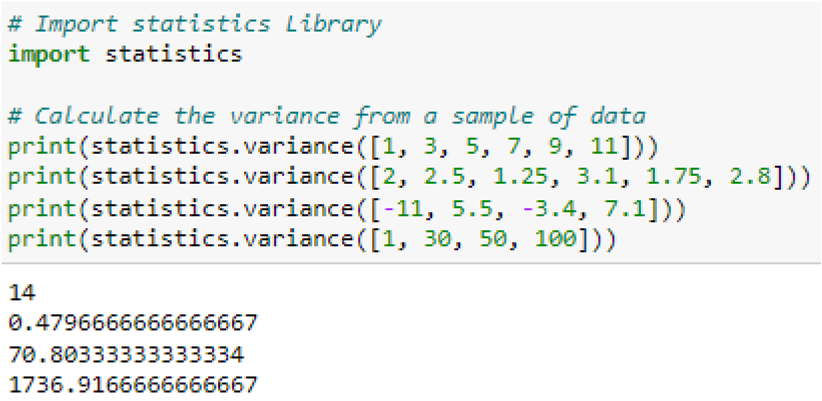
Program-4



```python
# Import statistics Library
import statistics

# Calculate the standard deviation from a sample of data
print(statistics.stdev([1, 3, 5, 7, 9, 11]))
print(statistics.stdev([2, 2.5, 1.25, 3.1, 1.75, 2.8]))
print(statistics.stdev([-11, 5.5, -3.4, 7.1]))
print(statistics.stdev([1, 30, 50, 100]))
```

```
3.7416573867739413
0.6925797186365384
8.414471660973929
41.67633221226008
```

Program-5



```python
# Import statistics Library
import statistics

# Calculate the variance from a sample of data
print(statistics.variance([1, 3, 5, 7, 9, 11]))
print(statistics.variance([2, 2.5, 1.25, 3.1, 1.75, 2.8]))
print(statistics.variance([-11, 5.5, -3.4, 7.1]))
print(statistics.variance([1, 30, 50, 100]))
```

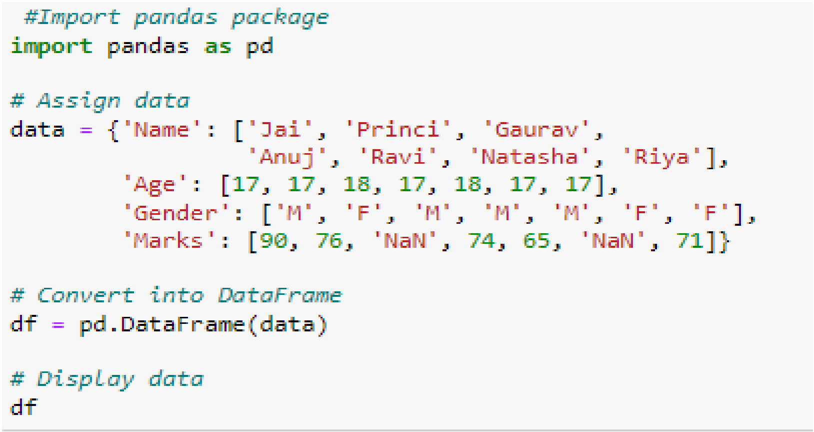
```
14
0.4796666666666667
70.80333333333334
1736.9166666666667
```

c) Write a python program to compute reshaping the data, Filtering the data , merging the data and handling the missing values in datasets.

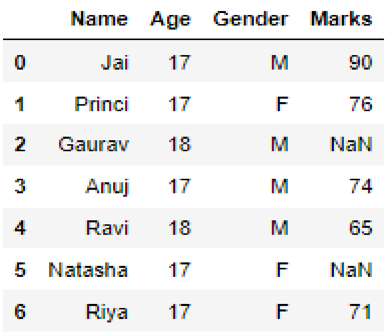
Program-2

Method:II

Assigning the data:

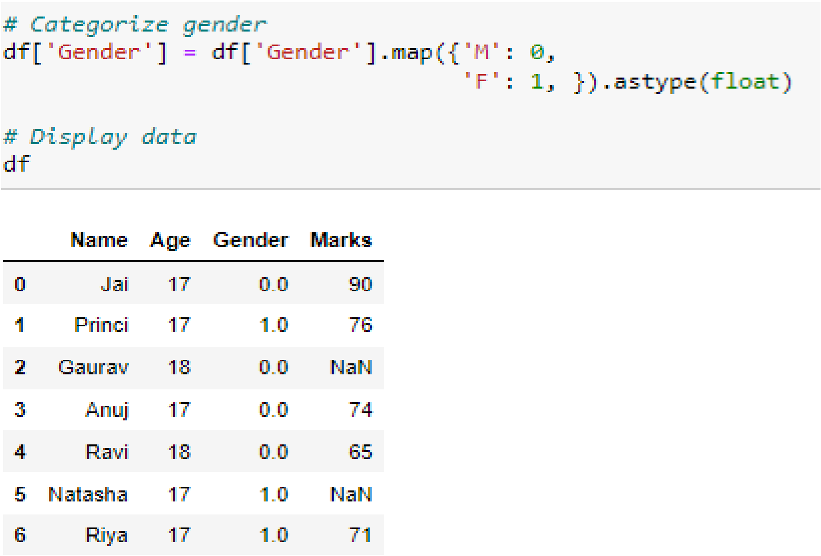


```
 #Import pandas package
import pandas as pd

# Assign data
data = {'Name': ['Jai', 'Princi', 'Gaurav',
                 'Anuj', 'Ravi', 'Natasha', 'Riya'],
        'Age': [17, 17, 18, 17, 18, 17, 17],
        'Gender': ['M', 'F', 'M', 'M', 'M', 'F', 'F'],
        'Marks': [90, 76, 'NaN', 74, 65, 'NaN', 71]}

# Convert into DataFrame
df = pd.DataFrame(data)

# Display data
df
```



	Name	Age	Gender	Marks
0	Jai	17	M	90
1	Princi	17	F	76
2	Gaurav	18	M	NaN
3	Anuj	17	M	74
4	Ravi	18	M	65
5	Natasha	17	F	NaN
6	Riya	17	F	71

Program-3



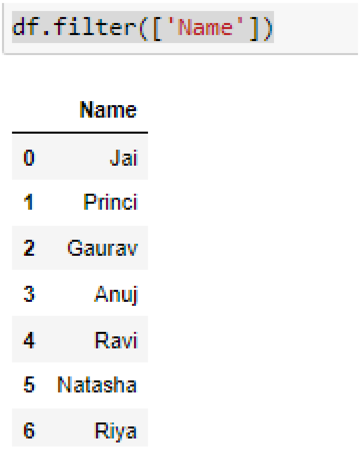
```
# Categorize gender
df['Gender'] = df['Gender'].map({'M': 0,
                                 'F': 1, }).astype(float)

# Display data
df
```

	Name	Age	Gender	Marks
0	Jai	17	0.0	90
1	Princi	17	1.0	76
2	Gaurav	18	0.0	NaN
3	Anuj	17	0.0	74
4	Ravi	18	0.0	65
5	Natasha	17	1.0	NaN
6	Riya	17	1.0	71

Filtering the data

suppose there is a requirement for the details regarding name, gender, marks of the top-scoring students. Here we need to remove some unwanted data.

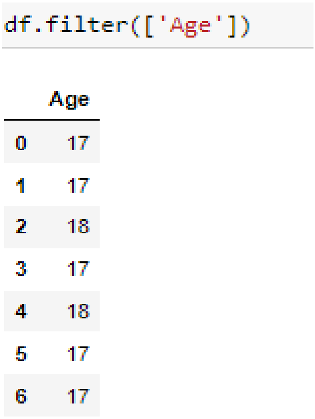
Program-1



```
df.filter(['Name'])
```

```
      Name
0      Jai
1   Princi
2   Gaurav
3     Anuj
4     Ravi
5  Natasha
6     Riya
```

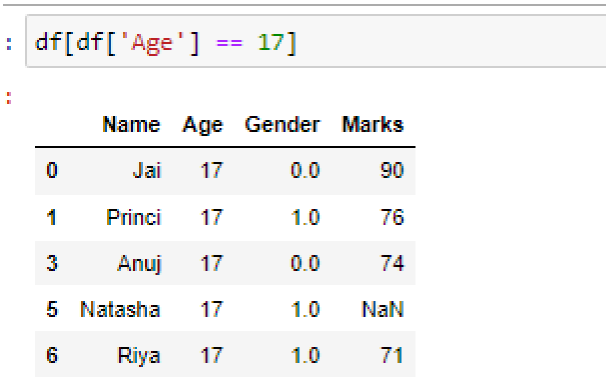
Program-2



```
df.filter(['Age'])
```

```
   Age
0   17
1   17
2   18
3   17
4   18
5   17
6   17
```

Program-3



```
: df[df['Age'] == 17]
```

```
:
      Name  Age  Gender  Marks
0      Jai   17     0.0     90
1   Princi   17     1.0     76
3     Anuj   17     0.0     74
5  Natasha   17     1.0    NaN
6     Riya   17     1.0     71
```

Merge data:

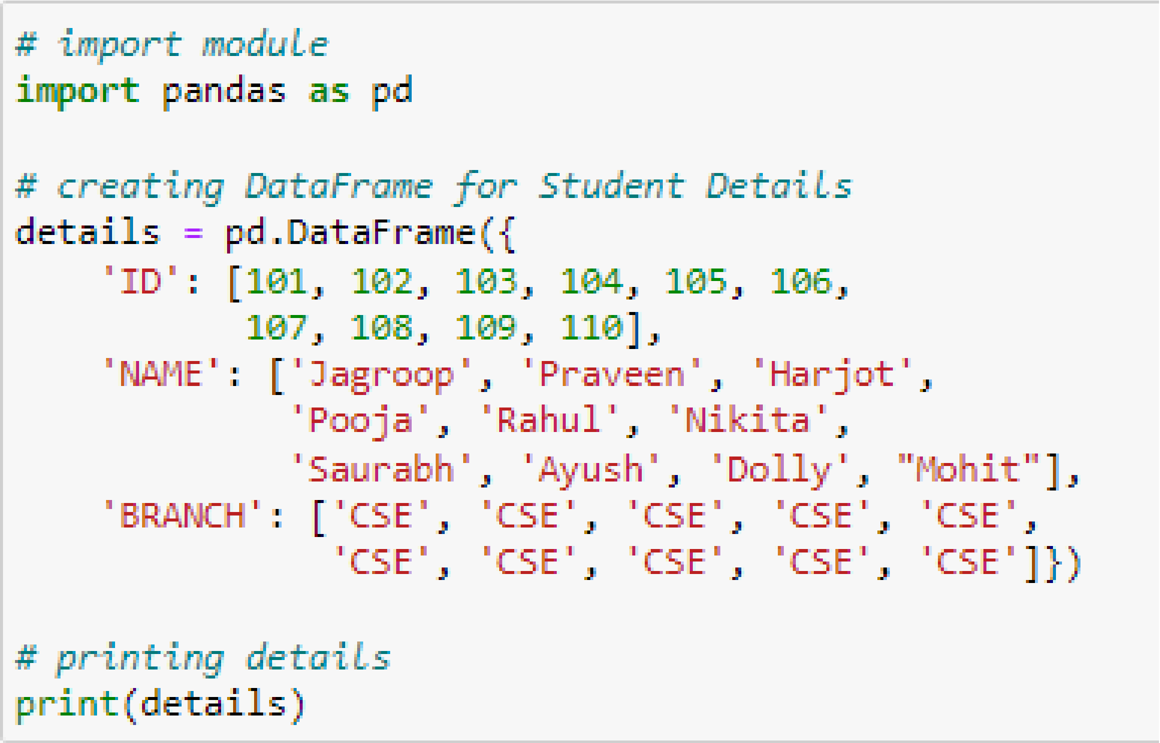
Merge operation is used to merge raw data and into the desired format.

Syntax:

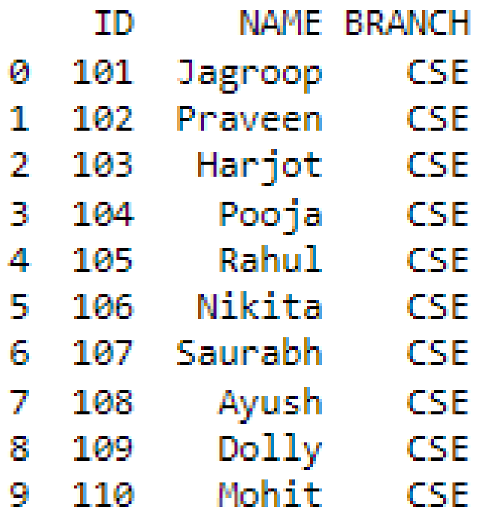
pd.merge(data_frame1,data_frame2, on="field ")

Program-4

First type of data:



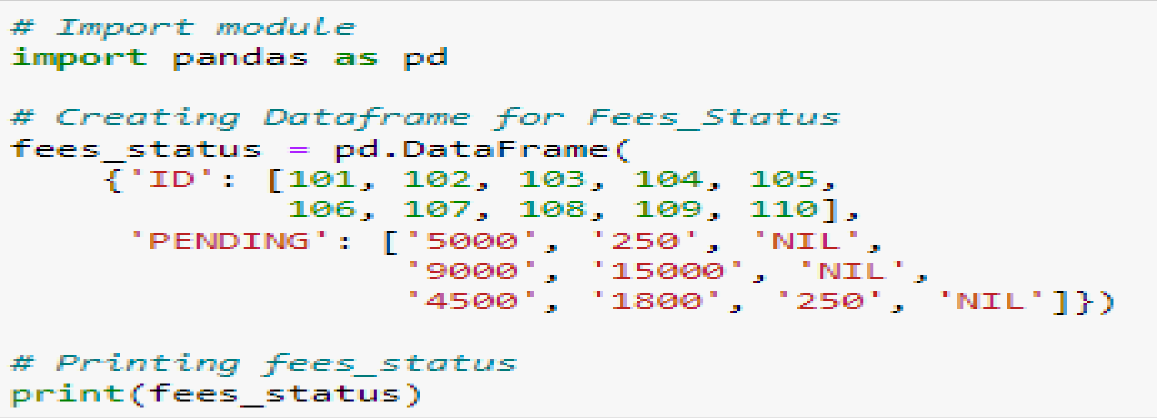
```python
# import module
import pandas as pd

# creating DataFrame for Student Details
details = pd.DataFrame({
    'ID': [101, 102, 103, 104, 105, 106,
           107, 108, 109, 110],
    'NAME': ['Jagroop', 'Praveen', 'Harjot',
             'Pooja', 'Rahul', 'Nikita',
             'Saurabh', 'Ayush', 'Dolly', "Mohit"],
    'BRANCH': ['CSE', 'CSE', 'CSE', 'CSE', 'CSE',
               'CSE', 'CSE', 'CSE', 'CSE', 'CSE']})

# printing details
print(details)
```



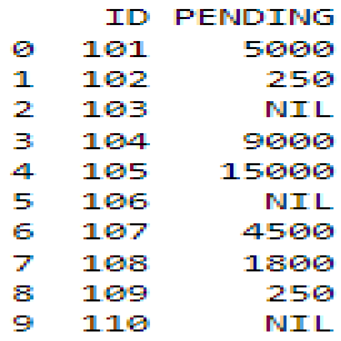
```
    ID     NAME BRANCH
0  101  Jagroop    CSE
1  102  Praveen    CSE
2  103   Harjot    CSE
3  104    Pooja    CSE
4  105    Rahul    CSE
5  106   Nikita    CSE
6  107  Saurabh    CSE
7  108    Ayush    CSE
8  109    Dolly    CSE
9  110    Mohit    CSE
```

Program-5

Second type of data:

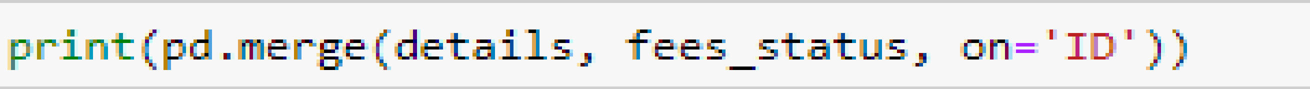


```
# Import module
import pandas as pd

# Creating Dataframe for Fees_Status
fees_status = pd.DataFrame(
    {'ID': [101, 102, 103, 104, 105,
            106, 107, 108, 109, 110],
     'PENDING': ['5000', '250', 'NIL',
                 '9000', '15000', 'NIL',
                 '4500', '1800', '250', 'NIL']})

# Printing fees_status
print(fees_status)
```

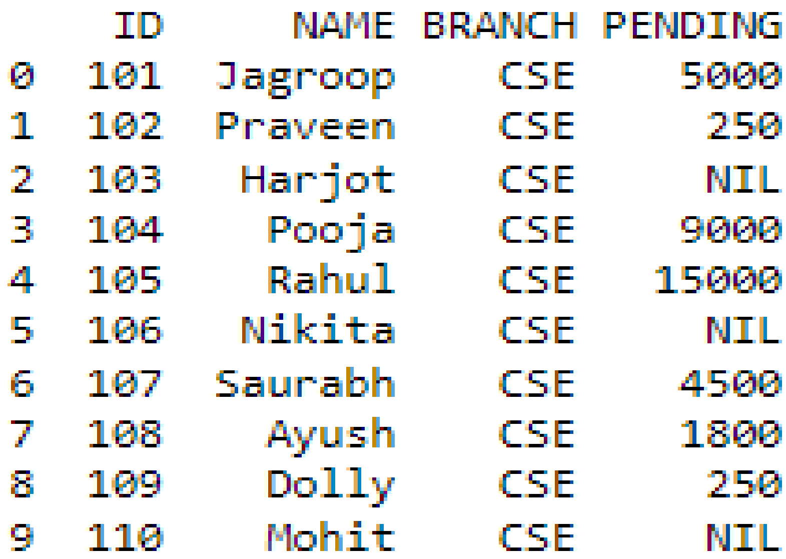


```
    ID PENDING
0  101    5000
1  102     250
2  103     NIL
3  104    9000
4  105   15000
5  106     NIL
6  107    4500
7  108    1800
8  109     250
9  110     NIL
```

Program-6



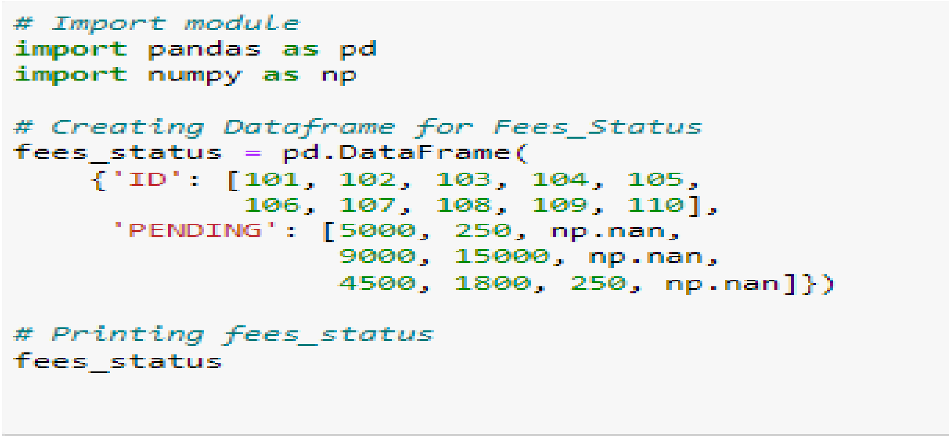
```
print(pd.merge(details, fees_status, on='ID'))
```



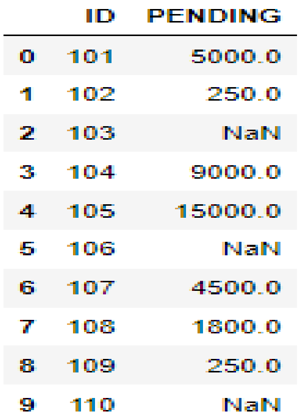
```
    ID     NAME BRANCH PENDING
0  101  Jagroop    CSE    5000
1  102  Praveen    CSE     250
2  103   Harjot    CSE     NIL
3  104    Pooja    CSE    9000
4  105    Rahul    CSE   15000
5  106   Nikita    CSE     NIL
6  107  Saurabh    CSE    4500
7  108    Ayush    CSE    1800
8  109    Dolly    CSE     250
9  110    Mohit    CSE     NIL
```

Handling the missing values:

Program-1



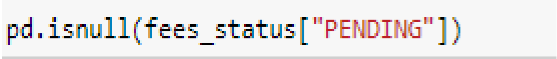
```
# Import module
import pandas as pd
import numpy as np

# Creating Dataframe for Fees_Status
fees_status = pd.DataFrame(
    {'ID': [101, 102, 103, 104, 105,
            106, 107, 108, 109, 110],
     'PENDING': [5000, 250, np.nan,
                 9000, 15000, np.nan,
                 4500, 1800, 250, np.nan]})

# Printing fees_status
fees_status
```



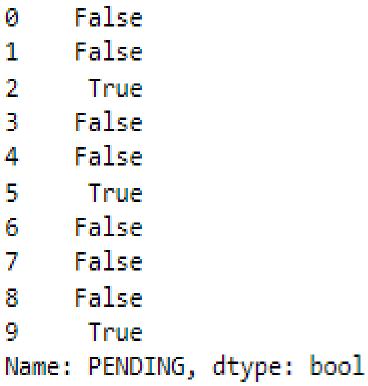
```
    ID   PENDING
0  101    5000.0
1  102     250.0
2  103       NaN
3  104    9000.0
4  105   15000.0
5  106       NaN
6  107    4500.0
7  108    1800.0
8  109     250.0
9  110       NaN
```

Program-2

In order to check null values in Pandas DataFrame, we use isnull() function this function return dataframe of Boolean values which are True for NaN values.



```
pd.isnull(fees_status["PENDING"])
```

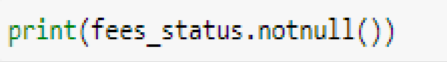


```
0    False
1    False
2     True
3    False
4    False
5     True
6    False
7    False
8    False
9     True
Name: PENDING, dtype: bool
```

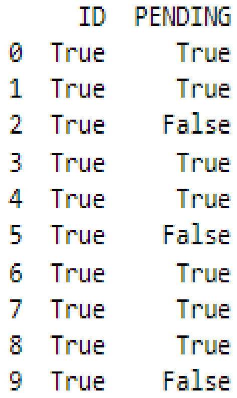
Program-7

Program-3

In order to check null values in Pandas Dataframe, we use notnull() function this function return dataframe of Boolean values which are False for NaN values.

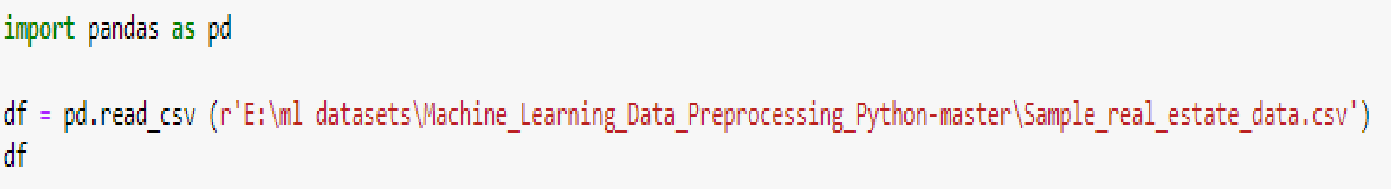


```
print(fees_status.notnull())
```

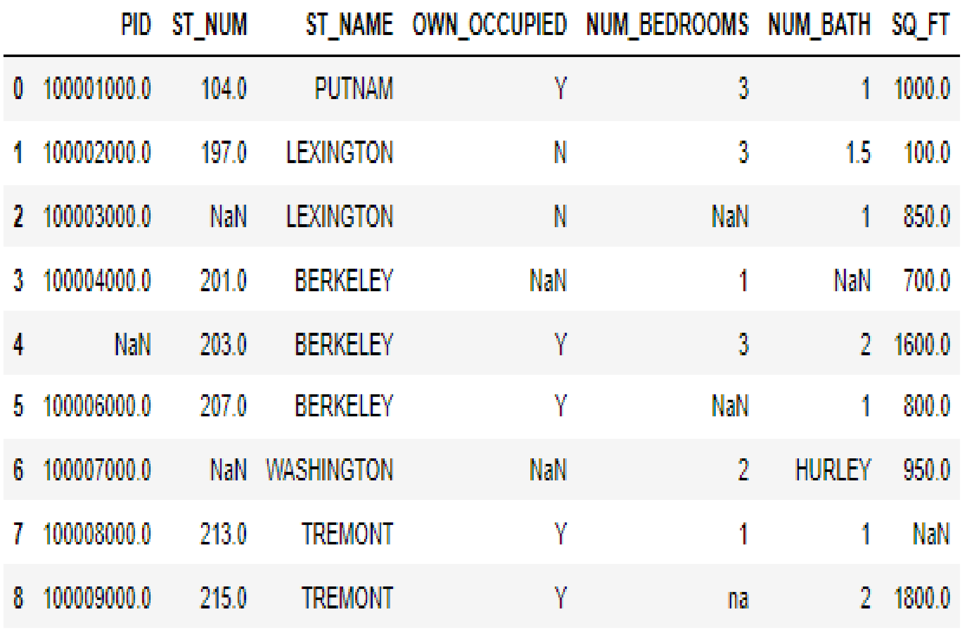


```
     ID  PENDING
0  True     True
1  True     True
2  True    False
3  True     True
4  True     True
5  True    False
6  True     True
7  True     True
8  True     True
9  True    False
```

Program-4

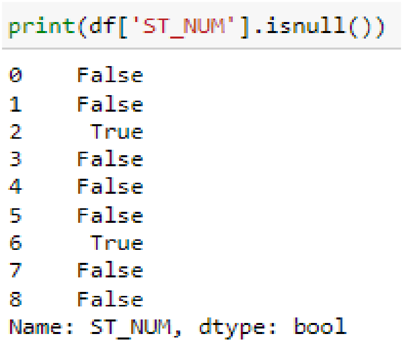


```
import pandas as pd

df = pd.read_csv (r'E:\ml datasets\Machine_Learning_Data_Preprocessing_Python-master\Sample_real_estate_data.csv')
df
```



	PID	ST_NUM	ST_NAME	OWN_OCCUPIED	NUM_BEDROOMS	NUM_BATH	SQ_FT
0	100001000.0	104.0	PUTNAM	Y	3	1	1000.0
1	100002000.0	197.0	LEXINGTON	N	3	1.5	100.0
2	100003000.0	NaN	LEXINGTON	N	NaN	1	850.0
3	100004000.0	201.0	BERKELEY	NaN	1	NaN	700.0
4	NaN	203.0	BERKELEY	Y	3	2	1600.0
5	100006000.0	207.0	BERKELEY	Y	NaN	1	800.0
6	100007000.0	NaN	WASHINGTON	NaN	2	HURLEY	950.0
7	100008000.0	213.0	TREMONT	Y	1	1	NaN
8	100009000.0	215.0	TREMONT	Y	na	2	1800.0

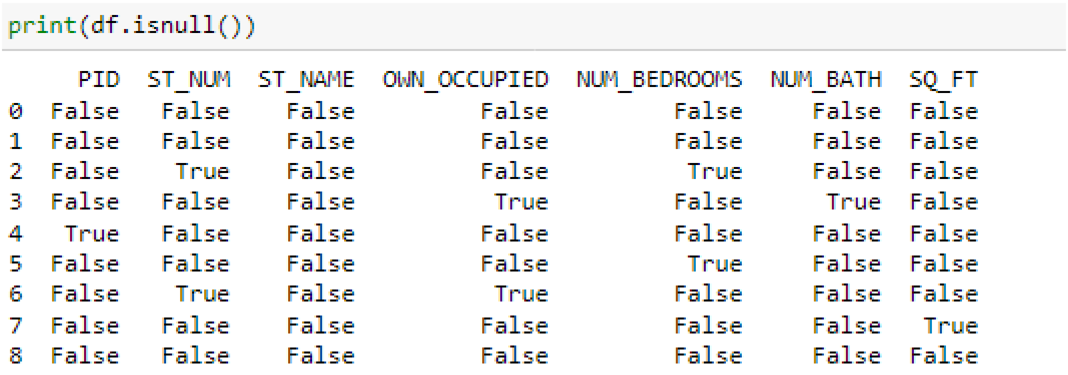
Program-5



```
print(df['ST_NUM'].isnull())
```

```
0    False
1    False
2     True
3    False
4    False
5    False
6     True
7    False
8    False
Name: ST_NUM, dtype: bool
```

Program-6



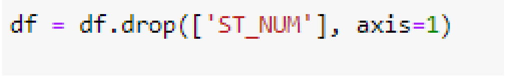
```
print(df.isnull())
```

```
     PID  ST_NUM  ST_NAME  OWN_OCCUPIED  NUM_BEDROOMS  NUM_BATH  SQ_FT
0  False   False    False         False         False     False  False
1  False   False    False         False         False     False  False
2  False    True    False         False          True     False  False
3  False   False    False          True         False      True  False
4   True   False    False         False         False     False  False
5  False   False    False         False          True     False  False
6  False    True    False          True         False     False  False
7  False   False    False         False         False     False   True
8  False   False    False         False         False     False  False
```

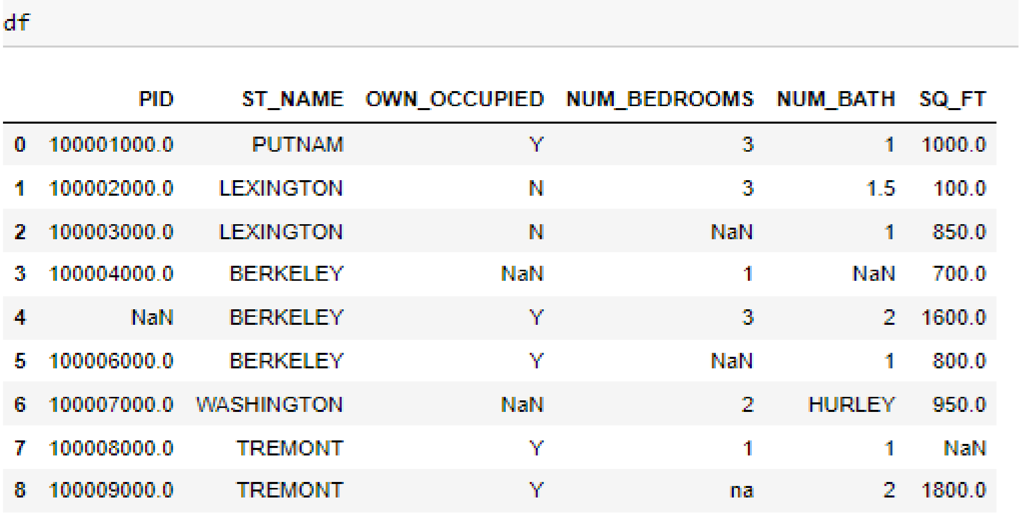
Program-7

Method-I

Drop Columns with Missing Values



```
df = df.drop(['ST_NUM'], axis=1)
```



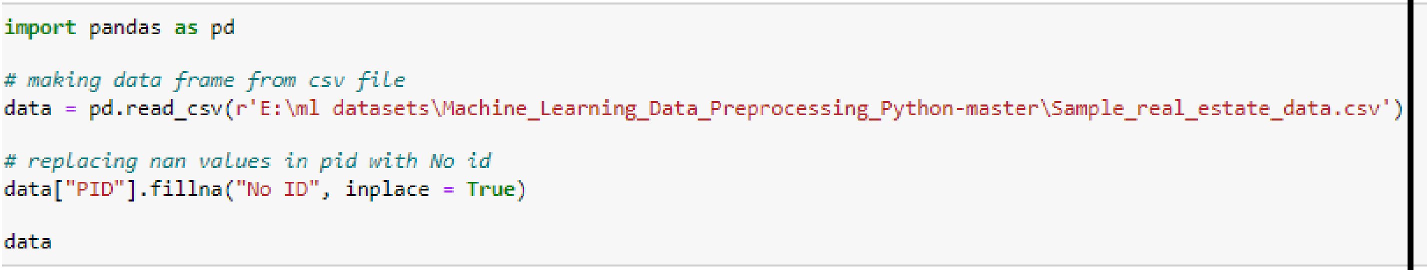
```
df
```

	PID	ST_NAME	OWN_OCCUPIED	NUM_BEDROOMS	NUM_BATH	SQ_FT
0	100001000.0	PUTNAM	Y	3	1	1000.0
1	100002000.0	LEXINGTON	N	3	1.5	100.0
2	100003000.0	LEXINGTON	N	NaN	1	850.0
3	100004000.0	BERKELEY	NaN	1	NaN	700.0
4	NaN	BERKELEY	Y	3	2	1600.0
5	100006000.0	BERKELEY	Y	NaN	1	800.0
6	100007000.0	WASHINGTON	NaN	2	HURLEY	950.0
7	100008000.0	TREMONT	Y	1	1	NaN
8	100009000.0	TREMONT	Y	na	2	1800.0

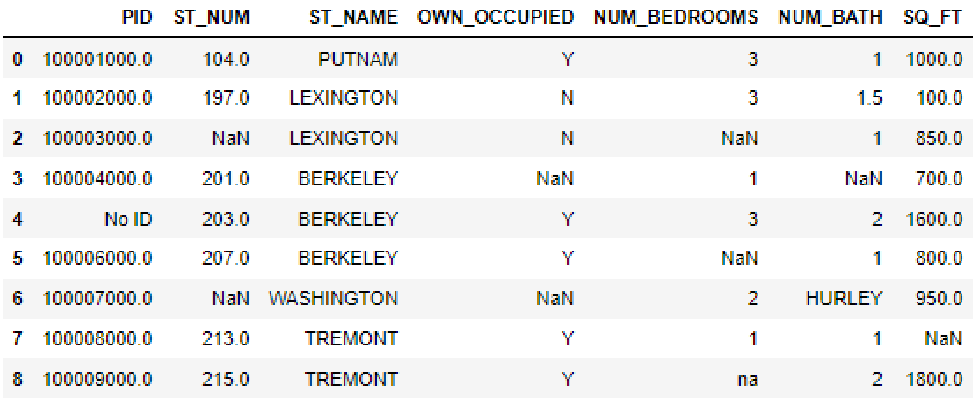
Program-8

Method-II

fillna() manages and let the user replace NaN values with some value of their own

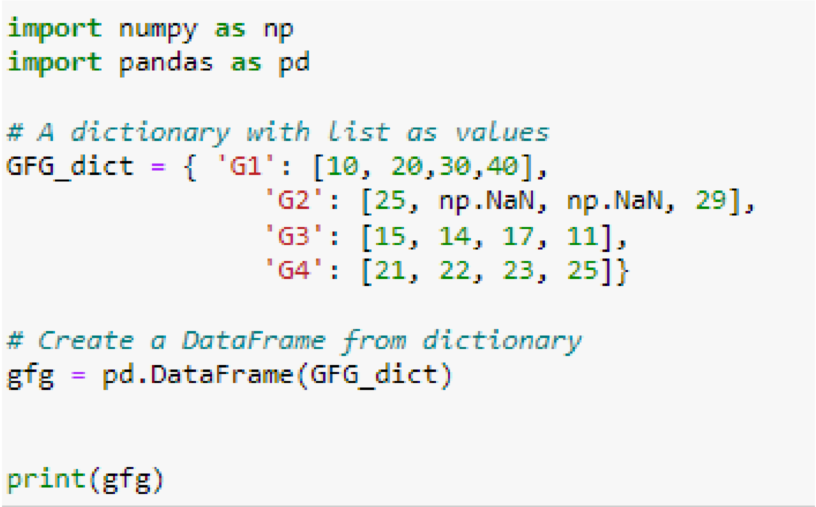


```python
import pandas as pd

# making data frame from csv file
data = pd.read_csv(r'E:\ml datasets\Machine_Learning_Data_Preprocessing_Python-master\Sample_real_estate_data.csv')

# replacing nan values in pid with No id
data["PID"].fillna("No ID", inplace = True)

data
```

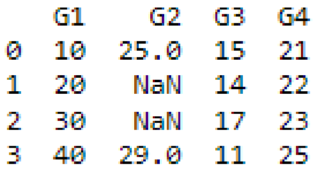


	PID	ST_NUM	ST_NAME	OWN_OCCUPIED	NUM_BEDROOMS	NUM_BATH	SQ_FT
0	100001000.0	104.0	PUTNAM	Y	3	1	1000.0
1	100002000.0	197.0	LEXINGTON	N	3	1.5	100.0
2	100003000.0	NaN	LEXINGTON	N	NaN	1	850.0
3	100004000.0	201.0	BERKELEY	NaN	1	NaN	700.0
4	No ID	203.0	BERKELEY	Y	3	2	1600.0
5	100006000.0	207.0	BERKELEY	Y	NaN	1	800.0
6	100007000.0	NaN	WASHINGTON	NaN	2	HURLEY	950.0
7	100008000.0	213.0	TREMONT	Y	1	1	NaN
8	100009000.0	215.0	TREMONT	Y	na	2	1800.0

Program-9



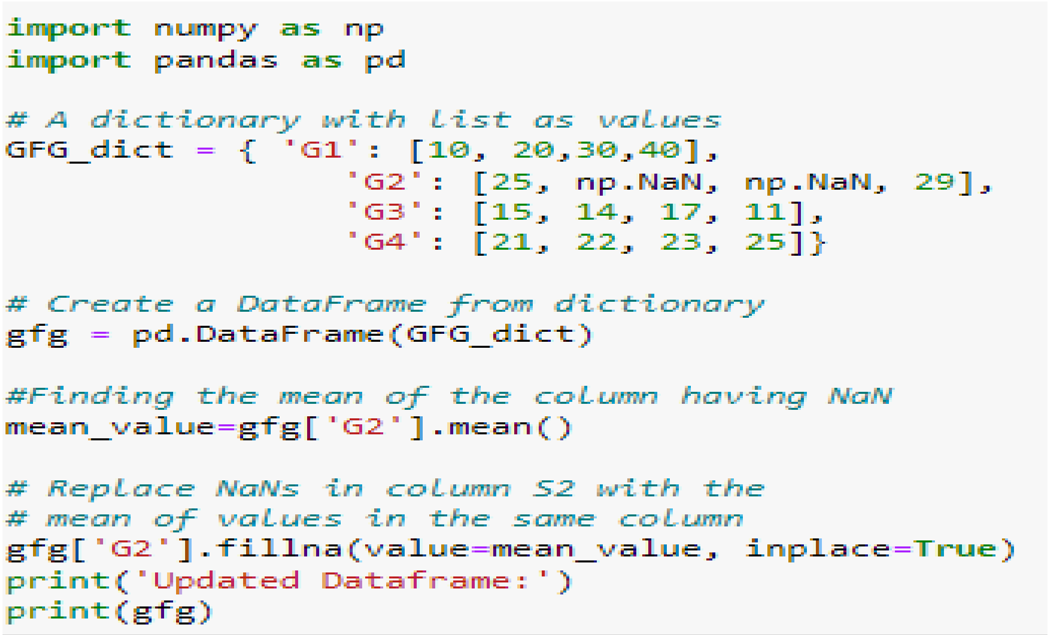
```python
import numpy as np
import pandas as pd

# A dictionary with list as values
GFG_dict = { 'G1': [10, 20,30,40],
                'G2': [25, np.NaN, np.NaN, 29],
                'G3': [15, 14, 17, 11],
                'G4': [21, 22, 23, 25]}

# Create a DataFrame from dictionary
gfg = pd.DataFrame(GFG_dict)


print(gfg)
```



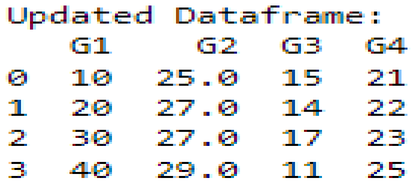
```
   G1    G2  G3  G4
0  10  25.0  15  21
1  20   NaN  14  22
2  30   NaN  17  23
3  40  29.0  11  25
```

Program-10

Filling missing values with mean



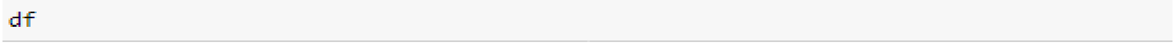
```
import numpy as np
import pandas as pd

# A dictionary with list as values
GFG_dict = { 'G1': [10, 20,30,40],
                'G2': [25, np.NaN, np.NaN, 29],
                'G3': [15, 14, 17, 11],
                'G4': [21, 22, 23, 25]}

# Create a DataFrame from dictionary
gfg = pd.DataFrame(GFG_dict)

#Finding the mean of the column having NaN
mean_value=gfg['G2'].mean()

# Replace NaNs in column S2 with the
# mean of values in the same column
gfg['G2'].fillna(value=mean_value, inplace=True)
print('Updated Dataframe:')
print(gfg)
```



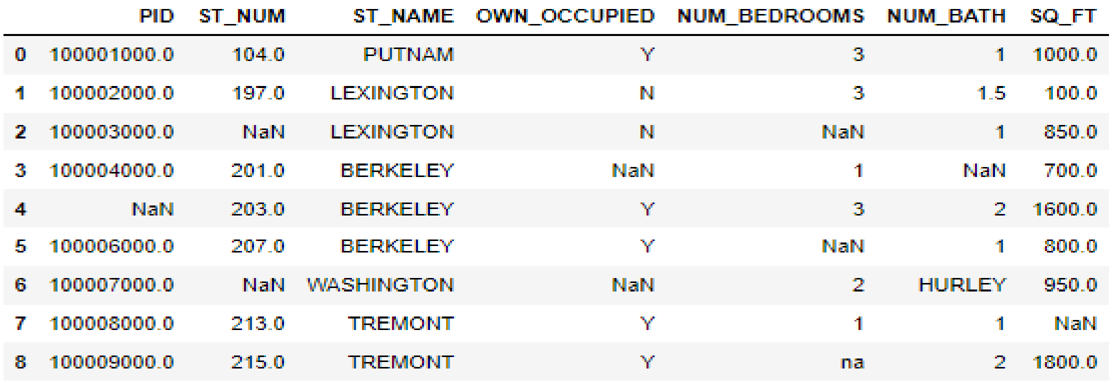
```
Updated Dataframe:
   G1    G2  G3  G4
0  10  25.0  15  21
1  20  27.0  14  22
2  30  27.0  17  23
3  40  29.0  11  25
```

Program-11

Filling missing values in csv files:
df=pd.read_csv(r'E:\mldatasets\Machine_Learning_Data_Preprocessing_Python-master\Sample_real_estate_data.csv', na_values='NAN')

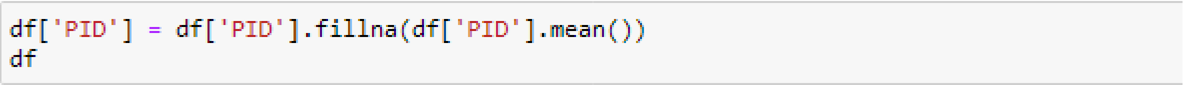


```
df
```

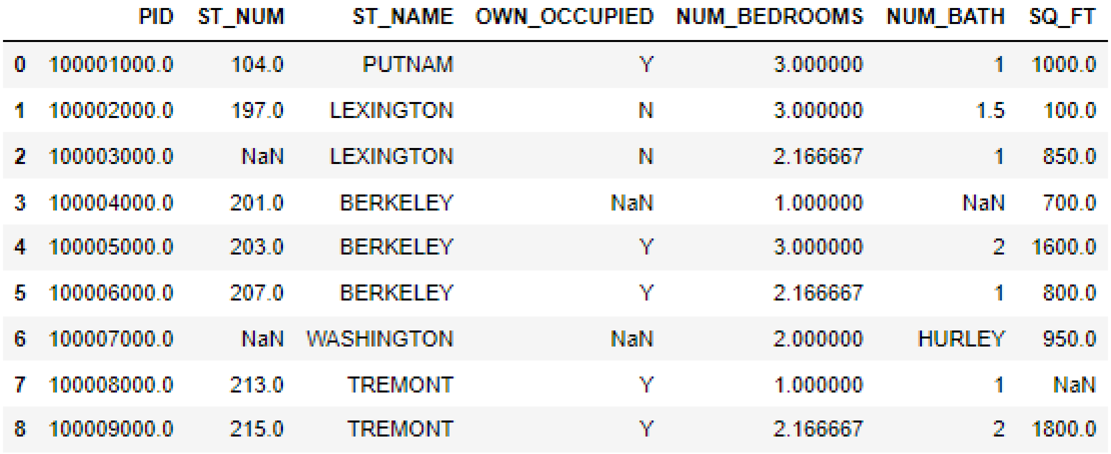


	PID	ST_NUM	ST_NAME	OWN_OCCUPIED	NUM_BEDROOMS	NUM_BATH	SQ_FT
0	100001000.0	104.0	PUTNAM	Y	3	1	1000.0
1	100002000.0	197.0	LEXINGTON	N	3	1.5	100.0
2	100003000.0	NaN	LEXINGTON	N	NaN	1	850.0
3	100004000.0	201.0	BERKELEY	NaN	1	NaN	700.0
4	NaN	203.0	BERKELEY	Y	3	2	1600.0
5	100006000.0	207.0	BERKELEY	Y	NaN	1	800.0
6	100007000.0	NaN	WASHINGTON	NaN	2	HURLEY	950.0
7	100008000.0	213.0	TREMONT	Y	1	1	NaN
8	100009000.0	215.0	TREMONT	Y	na	2	1800.0

Program-12



```
df['PID'] = df['PID'].fillna(df['PID'].mean())
df
```

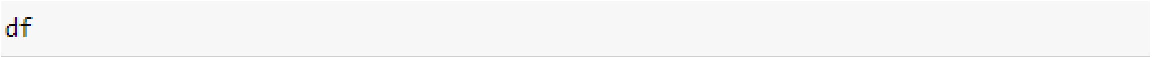


	PID	ST_NUM	ST_NAME	OWN_OCCUPIED	NUM_BEDROOMS	NUM_BATH	SQ_FT
0	100001000.0	104.0	PUTNAM	Y	3.000000	1	1000.0
1	100002000.0	197.0	LEXINGTON	N	3.000000	1.5	100.0
2	100003000.0	NaN	LEXINGTON	N	2.166667	1	850.0
3	100004000.0	201.0	BERKELEY	NaN	1.000000	NaN	700.0
4	100005000.0	203.0	BERKELEY	Y	3.000000	2	1600.0
5	100006000.0	207.0	BERKELEY	Y	2.166667	1	800.0
6	100007000.0	NaN	WASHINGTON	NaN	2.000000	HURLEY	950.0
7	100008000.0	213.0	TREMONT	Y	1.000000	1	NaN
8	100009000.0	215.0	TREMONT	Y	2.166667	2	1800.0

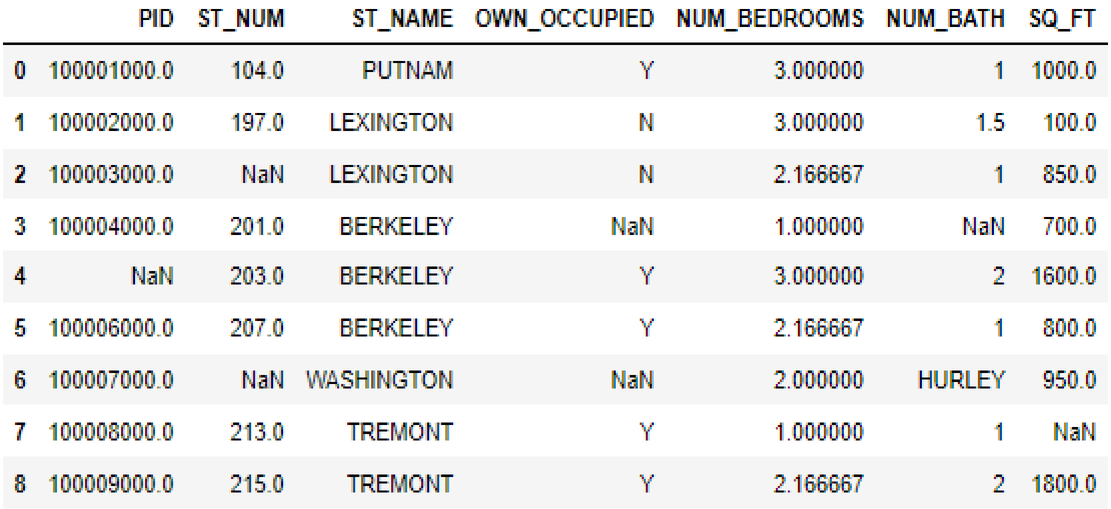
Program-13

Code:

```
missing_value = ["n/a","na","--"]
data1=pd.read_csv(r'E:\mldatasets\Machine_Learning_Data_Preprocessing_Python-master\Sample_real_estate_data.csv', na_values = missing_value)
df = data1
```



```
df
```

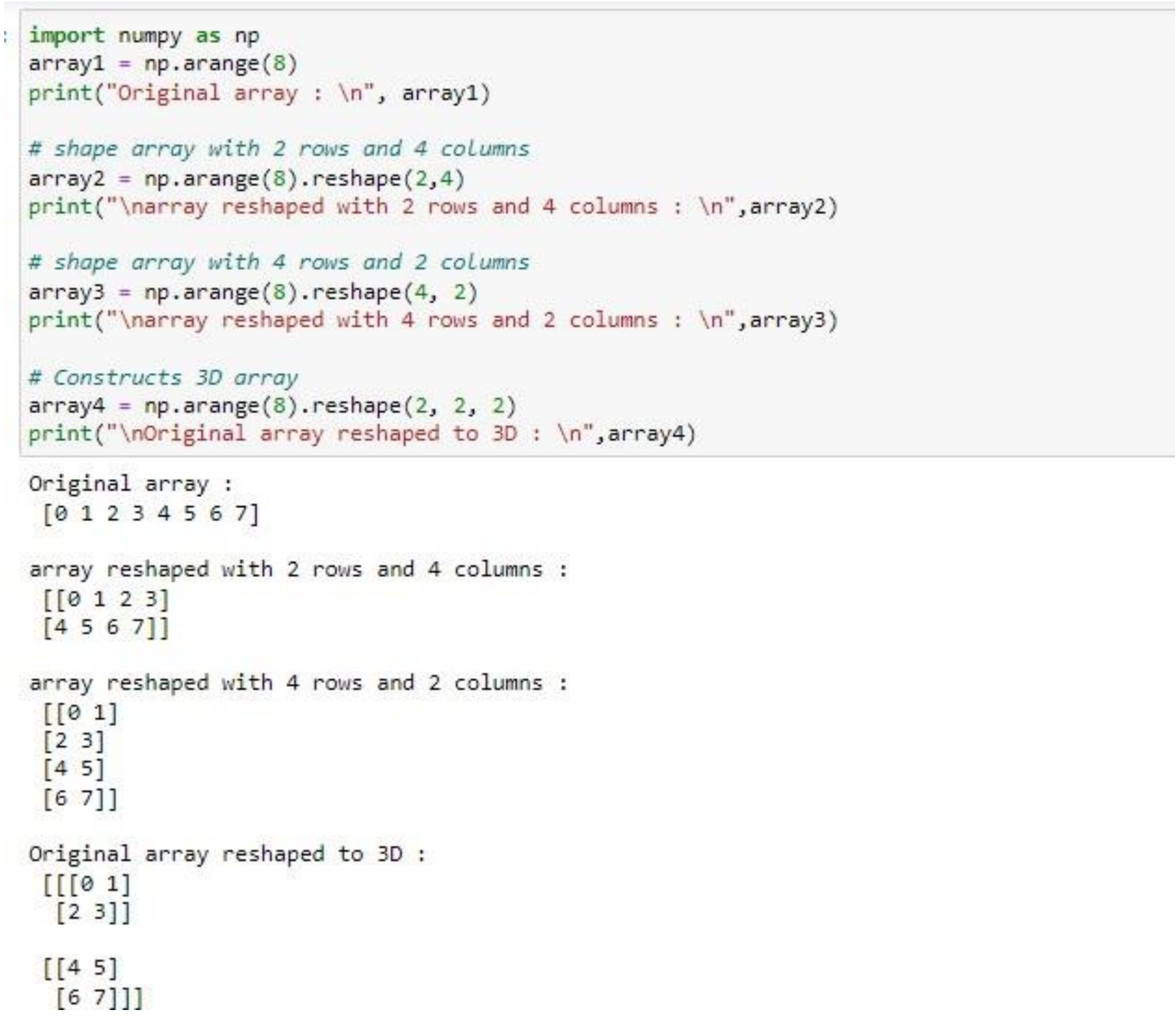


	PID	ST_NUM	ST_NAME	OWN_OCCUPIED	NUM_BEDROOMS	NUM_BATH	SQ_FT
0	100001000.0	104.0	PUTNAM	Y	3.000000	1	1000.0
1	100002000.0	197.0	LEXINGTON	N	3.000000	1.5	100.0
2	100003000.0	NaN	LEXINGTON	N	2.166667	1	850.0
3	100004000.0	201.0	BERKELEY	NaN	1.000000	NaN	700.0
4	NaN	203.0	BERKELEY	Y	3.000000	2	1600.0
5	100006000.0	207.0	BERKELEY	Y	2.166667	1	800.0
6	100007000.0	NaN	WASHINGTON	NaN	2.000000	HURLEY	950.0
7	100008000.0	213.0	TREMONT	Y	1.000000	1	NaN
8	100009000.0	215.0	TREMONT	Y	2.166667	2	1800.0

Program-1

Reshaping the data:

Method-I



```
import numpy as np
array1 = np.arange(8)
print("Original array : \n", array1)

# shape array with 2 rows and 4 columns
array2 = np.arange(8).reshape(2,4)
print("\narray reshaped with 2 rows and 4 columns : \n",array2)

# shape array with 4 rows and 2 columns
array3 = np.arange(8).reshape(4, 2)
print("\narray reshaped with 4 rows and 2 columns : \n",array3)

# Constructs 3D array
array4 = np.arange(8).reshape(2, 2, 2)
print("\nOriginal array reshaped to 3D : \n",array4)
```

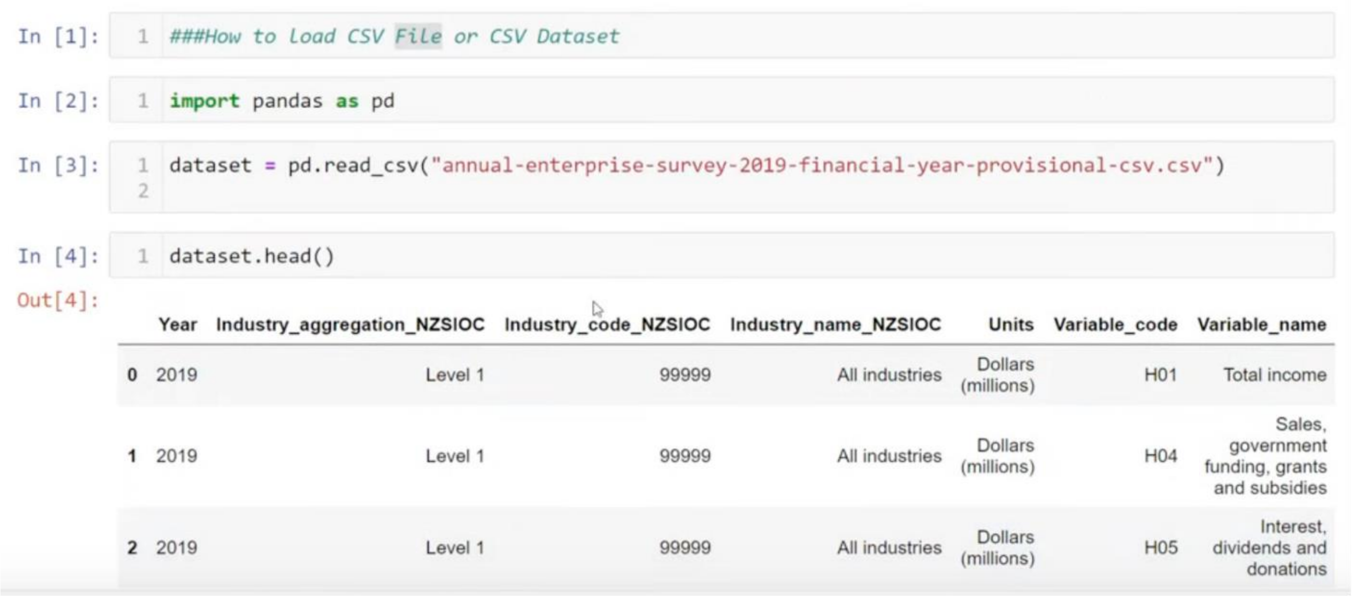
```
Original array :
 [0 1 2 3 4 5 6 7]

array reshaped with 2 rows and 4 columns :
 [[0 1 2 3]
 [4 5 6 7]]

array reshaped with 4 rows and 2 columns :
 [[0 1]
 [2 3]
 [4 5]
 [6 7]]

Original array reshaped to 3D :
 [[[0 1]
  [2 3]]

 [[4 5]
  [6 7]]]
```

Program:

Write a python program to loading csv dataset files using Pandas library functions.

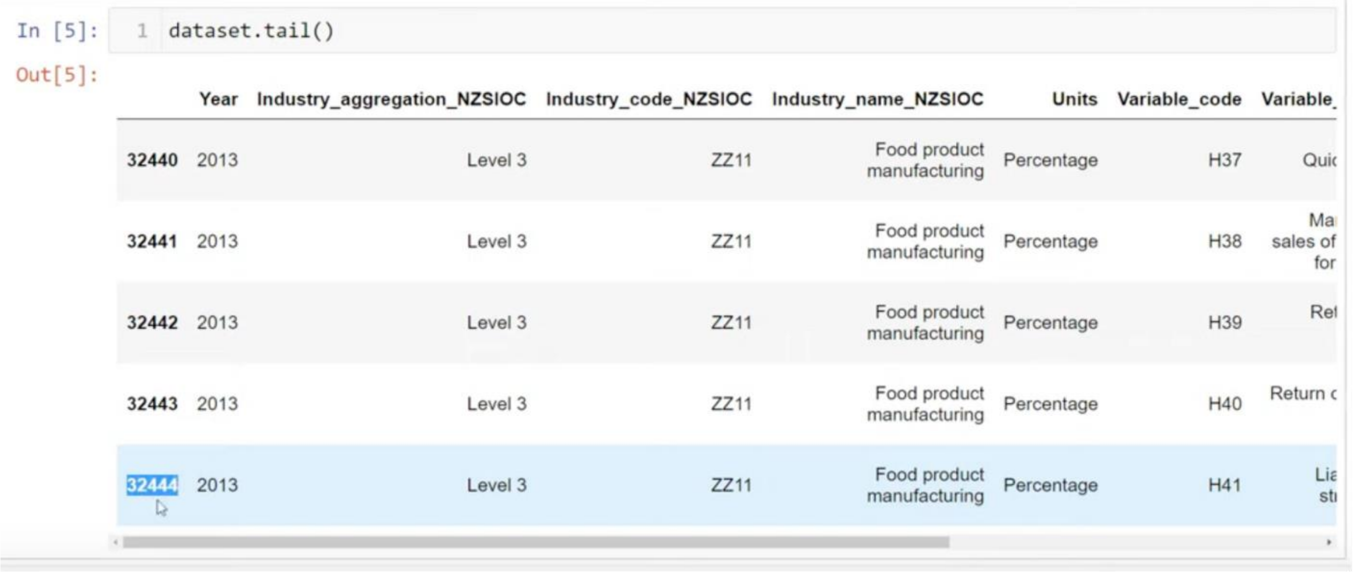
Program:

a. **Importing data(CSV)**



```
In [1]: ###How to Load CSV File or CSV Dataset

In [2]: import pandas as pd

In [3]: dataset = pd.read_csv("annual-enterprise-survey-2019-financial-year-provisional-csv.csv")

In [4]: dataset.head()
Out[4]:
```

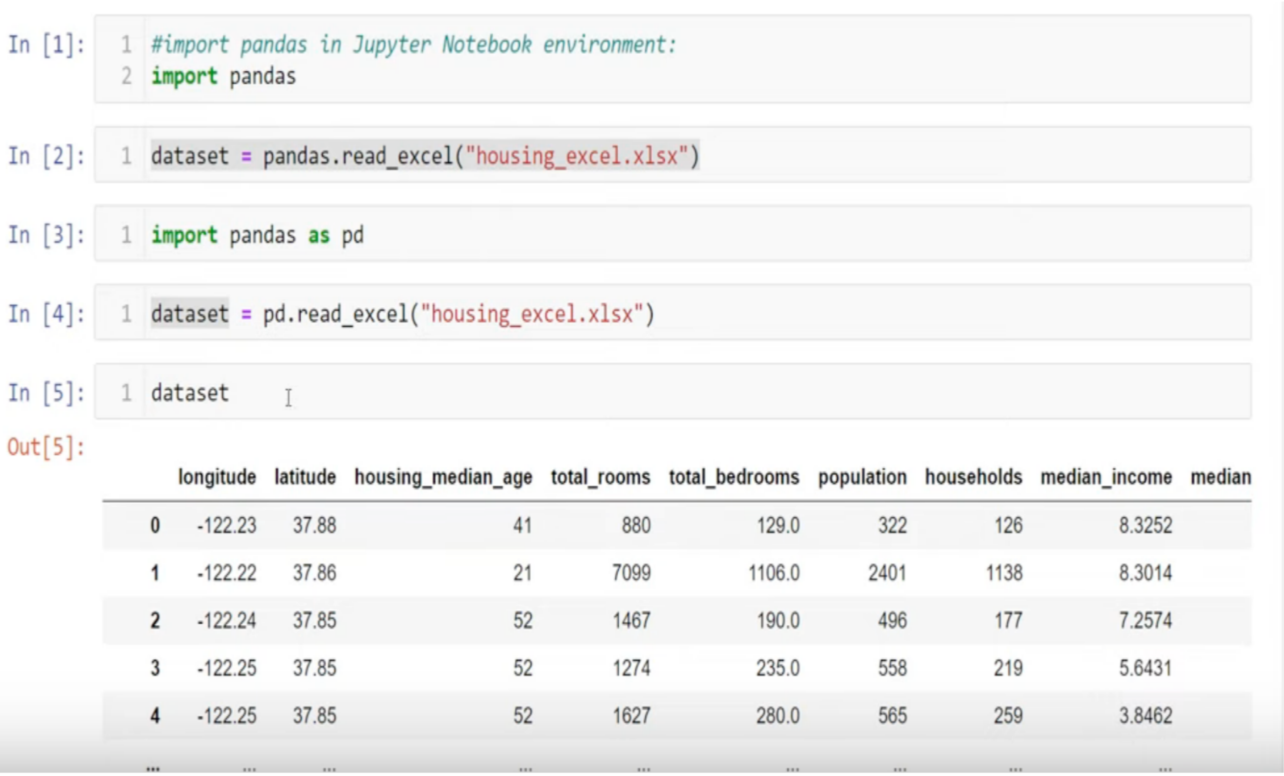
	Year	Industry_aggregation_NZSIOC	Industry_code_NZSIOC	Industry_name_NZSIOC	Units	Variable_code	Variable_name
0	2019	Level 1	99999	All industries	Dollars (millions)	H01	Total income
1	2019	Level 1	99999	All industries	Dollars (millions)	H04	Sales, government funding, grants and subsidies
2	2019	Level 1	99999	All industries	Dollars (millions)	H05	Interest, dividends and donations



```
In [5]: dataset.tail()
Out[5]:
```

	Year	Industry_aggregation_NZSIOC	Industry_code_NZSIOC	Industry_name_NZSIOC	Units	Variable_code	Variable_
32440	2013	Level 3	ZZ11	Food product manufacturing	Percentage	H37	Quic
32441	2013	Level 3	ZZ11	Food product manufacturing	Percentage	H38	Ma sales of for
32442	2013	Level 3	ZZ11	Food product manufacturing	Percentage	H39	Ret
32443	2013	Level 3	ZZ11	Food product manufacturing	Percentage	H40	Return c
32444	2013	Level 3	ZZ11	Food product manufacturing	Percentage	H41	Lia sti

b. Importing data(EXCEL)



In [1]:
```
#import pandas in Jupyter Notebook environment:
import pandas
```

In [2]:
```
dataset = pandas.read_excel("housing_excel.xlsx")
```

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

In [4]:
```
dataset = pd.read_excel("housing_excel.xlsx")
```

In [5]:
```
dataset
```

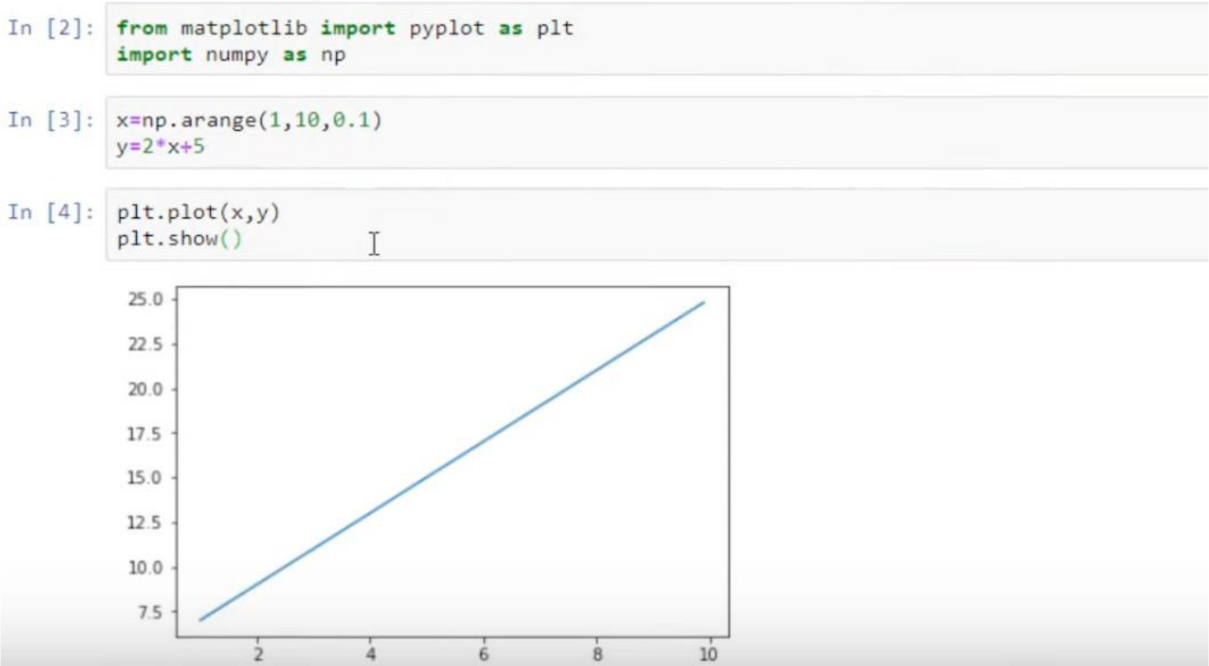
Out[5]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median
0	-122.23	37.88	41	880	129.0	322	126	8.3252	
1	-122.22	37.86	21	7099	1106.0	2401	1138	8.3014	
2	-122.24	37.85	52	1467	190.0	496	177	7.2574	
3	-122.25	37.85	52	1274	235.0	558	219	5.6431	
4	-122.25	37.85	52	1627	280.0	565	259	3.8462	
...	...	...	...	...	...	...	...	...	

Excersice:

Demonstrate various data pre-processing techniques for a given dataset.

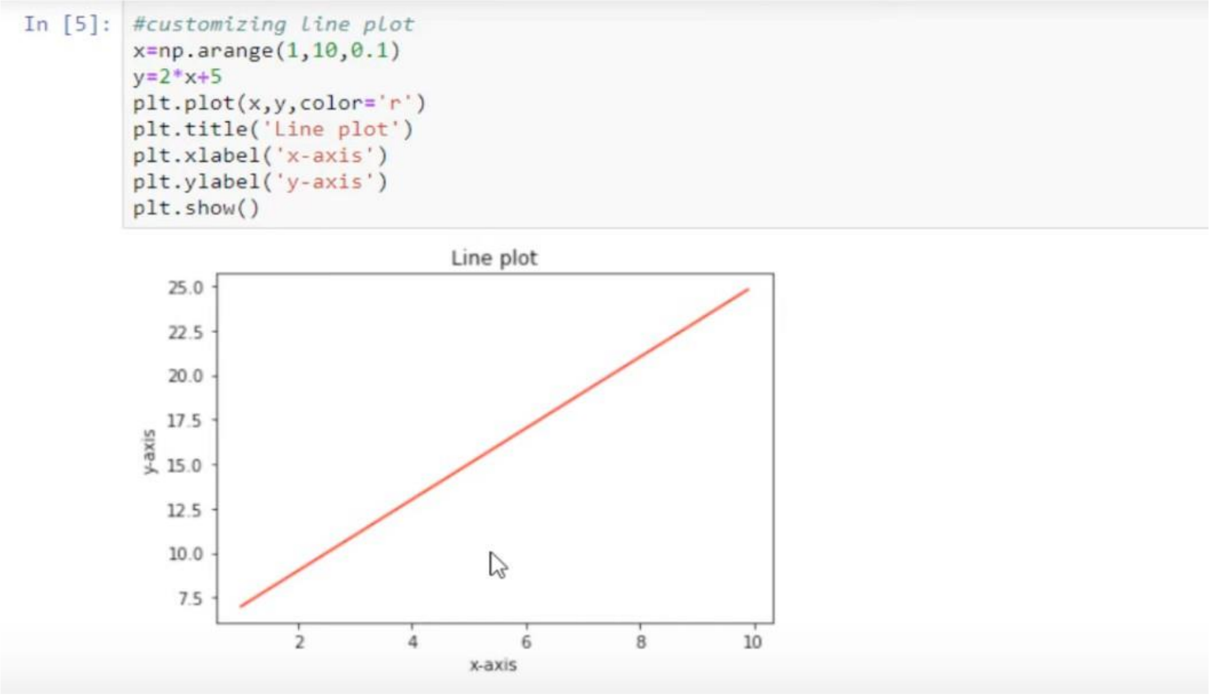
Program:



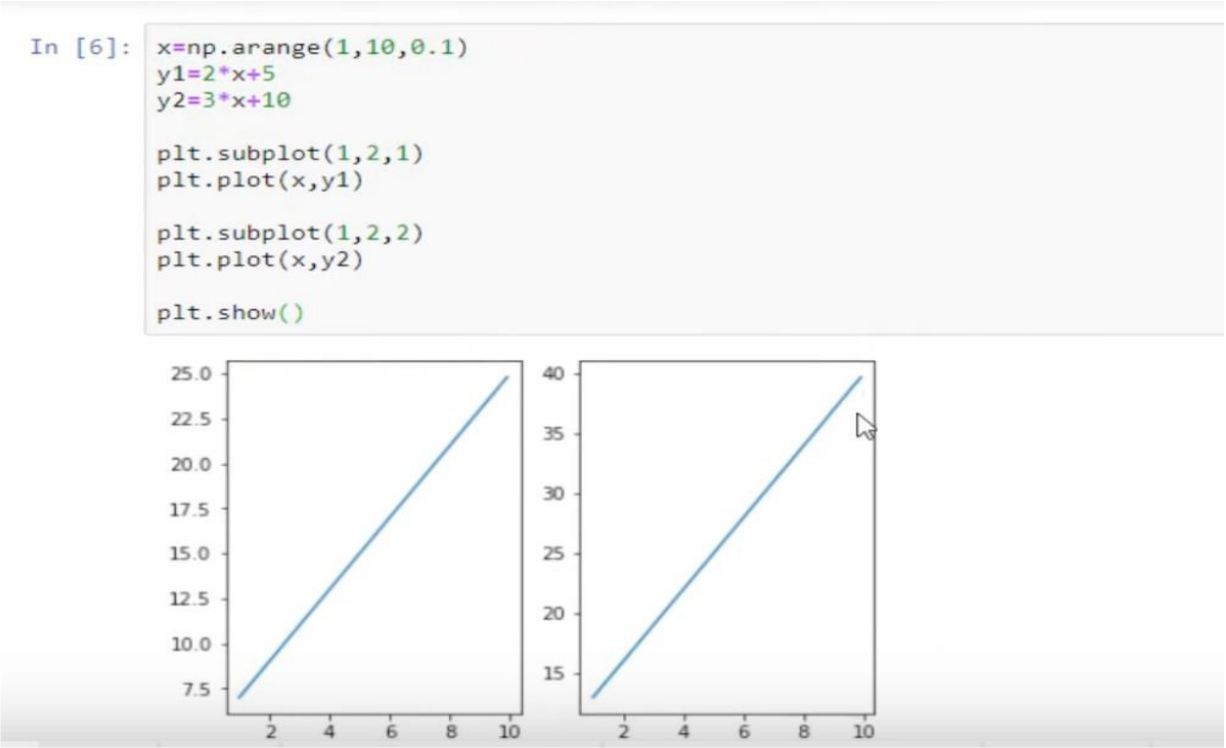
```
In [2]: from matplotlib import pyplot as plt
        import numpy as np
```

```
In [3]: x=np.arange(1,10,0.1)
        y=2*x+5
```

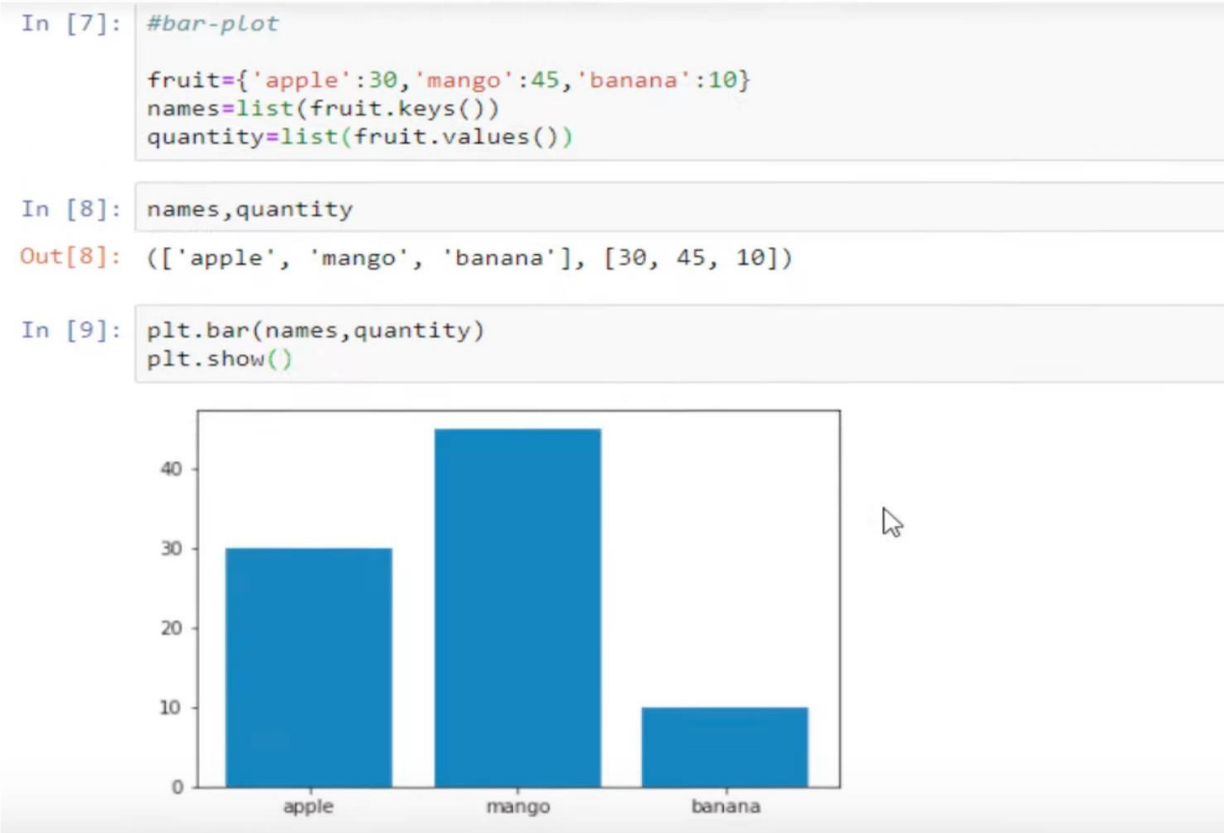
```
In [4]: plt.plot(x,y)
        plt.show()
```



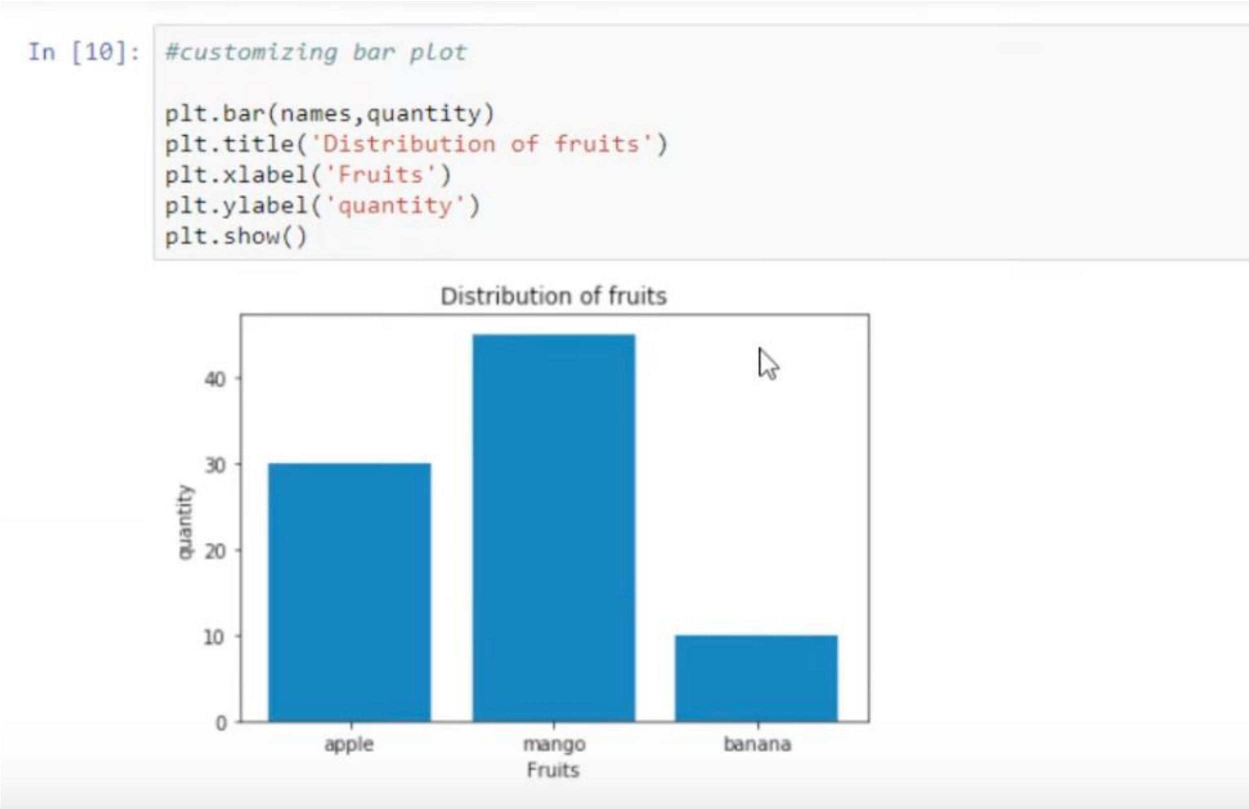
```
In [5]: #customizing line plot
        x=np.arange(1,10,0.1)
        y=2*x+5
        plt.plot(x,y,color='r')
        plt.title('Line plot')
        plt.xlabel('x-axis')
        plt.ylabel('y-axis')
        plt.show()
```



```
In [6]: x=np.arange(1,10,0.1)
        y1=2*x+5
        y2=3*x+10

        plt.subplot(1,2,1)
        plt.plot(x,y1)

        plt.subplot(1,2,2)
        plt.plot(x,y2)

        plt.show()
```

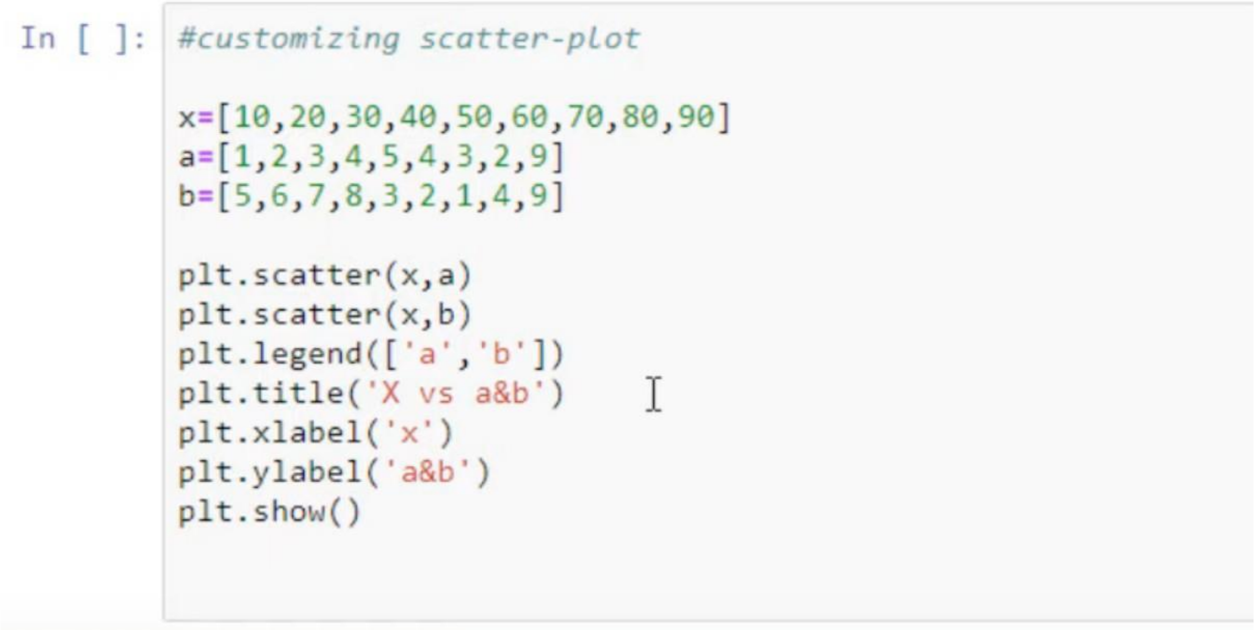


```
In [7]: #bar-plot

        fruit={'apple':30,'mango':45,'banana':10}
        names=list(fruit.keys())
        quantity=list(fruit.values())

In [8]: names,quantity

Out[8]: (['apple', 'mango', 'banana'], [30, 45, 10])

In [9]: plt.bar(names,quantity)
        plt.show()
```



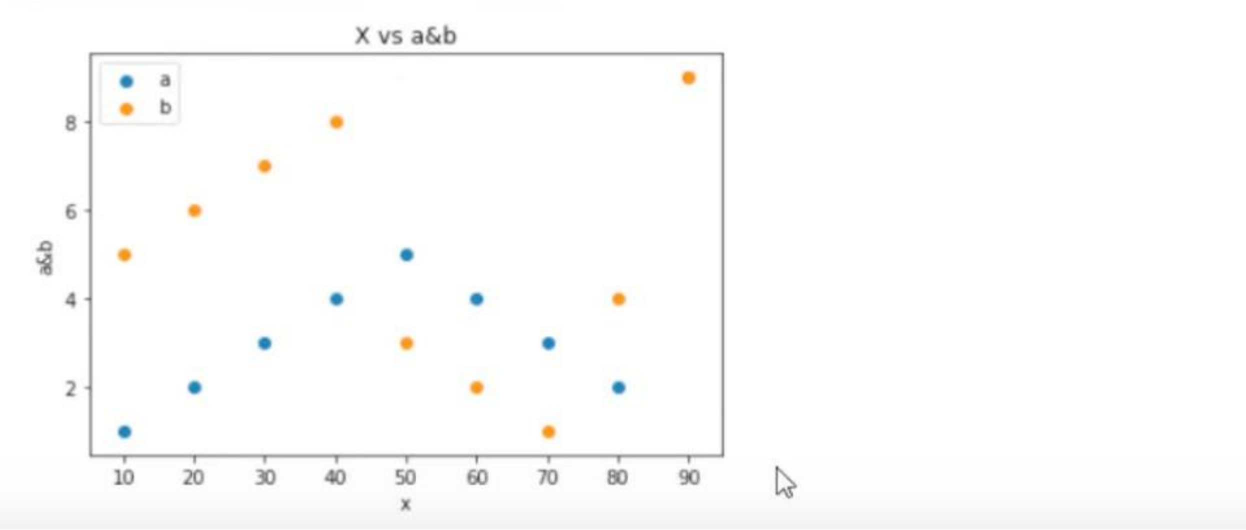
In [10]:

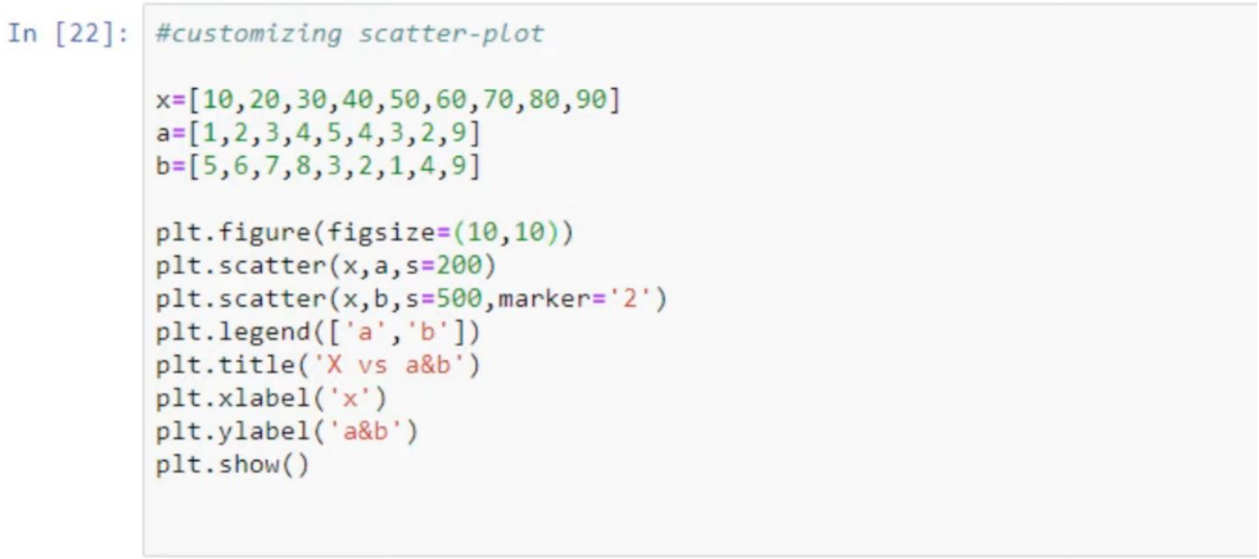
```
#customizing bar plot

plt.bar(names,quantity)
plt.title('Distribution of fruits')
plt.xlabel('Fruits')
plt.ylabel('quantity')
plt.show()
```



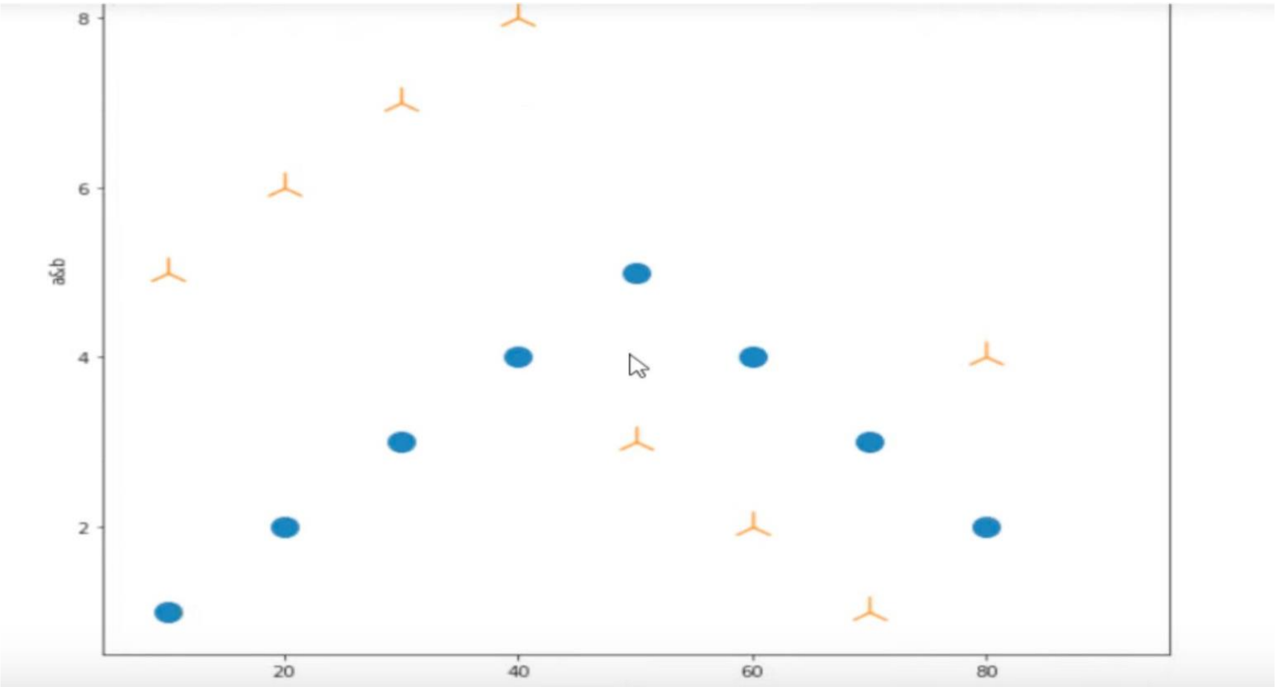
In [ ]:

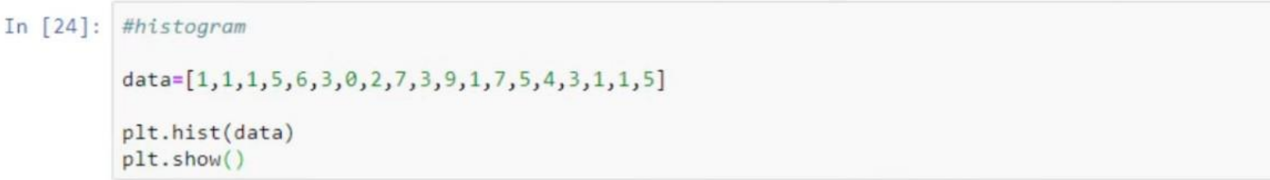
```
#customizing scatter-plot

x=[10,20,30,40,50,60,70,80,90]
a=[1,2,3,4,5,4,3,2,9]
b=[5,6,7,8,3,2,1,4,9]

plt.scatter(x,a)
plt.scatter(x,b)
plt.legend(['a','b'])
plt.title('X vs a&b')
plt.xlabel('x')
plt.ylabel('a&b')
plt.show()
```



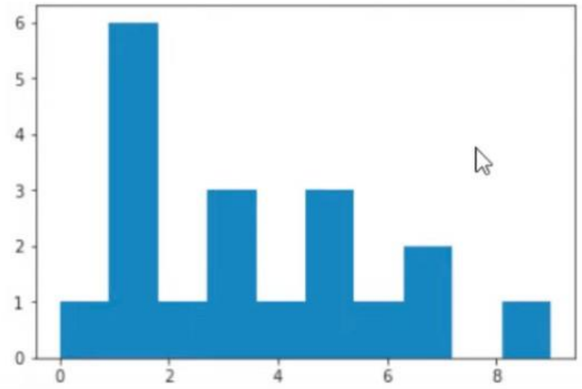


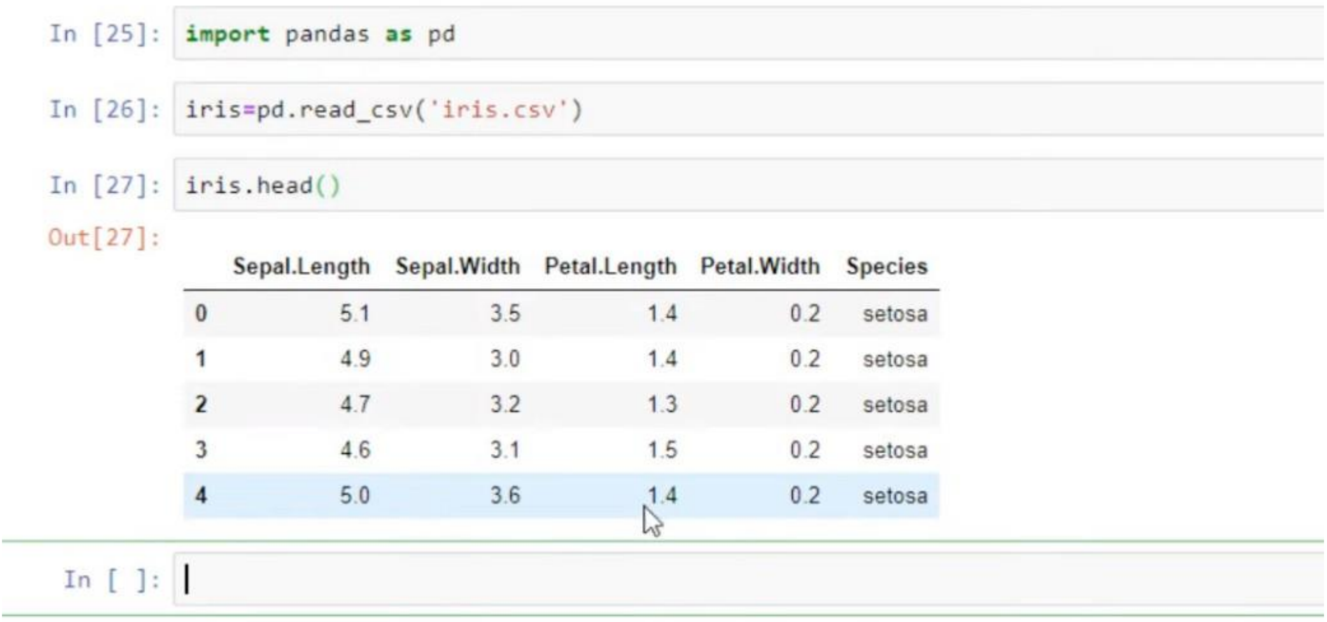
```
In [22]: #customizing scatter-plot

         x=[10,20,30,40,50,60,70,80,90]
         a=[1,2,3,4,5,4,3,2,9]
         b=[5,6,7,8,3,2,1,4,9]

         plt.figure(figsize=(10,10))
         plt.scatter(x,a,s=200)
         plt.scatter(x,b,s=500,marker='2')
         plt.legend(['a','b'])
         plt.title('X vs a&b')
         plt.xlabel('x')
         plt.ylabel('a&b')
         plt.show()
```





```
In [24]: #histogram

         data=[1,1,1,5,6,3,0,2,7,3,9,1,7,5,4,3,1,1,5]

         plt.hist(data)
         plt.show()
```



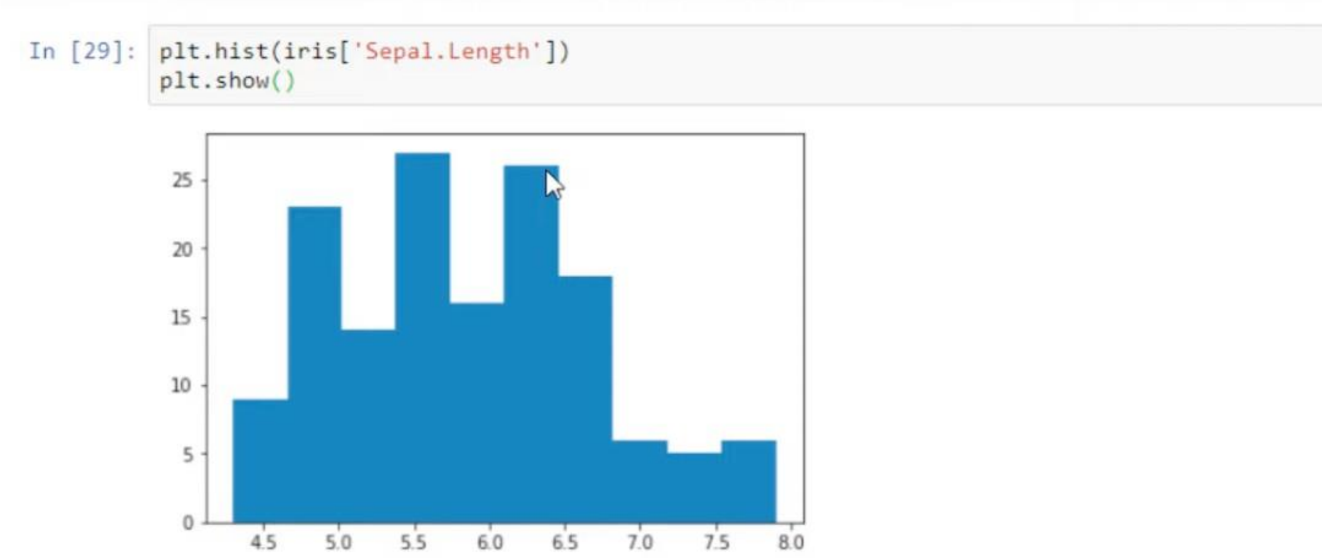


```
In [25]: import pandas as pd

In [26]: iris=pd.read_csv('iris.csv')

In [27]: iris.head()
```

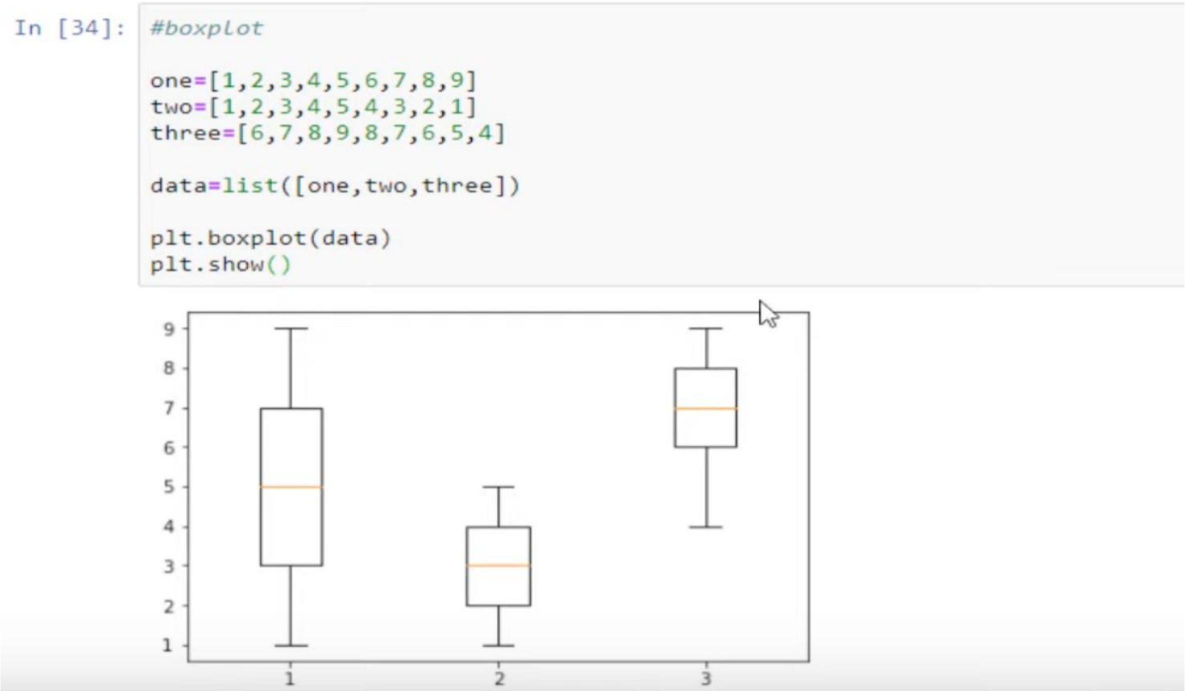
Out[27]:

	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

```
In [ ]:
```

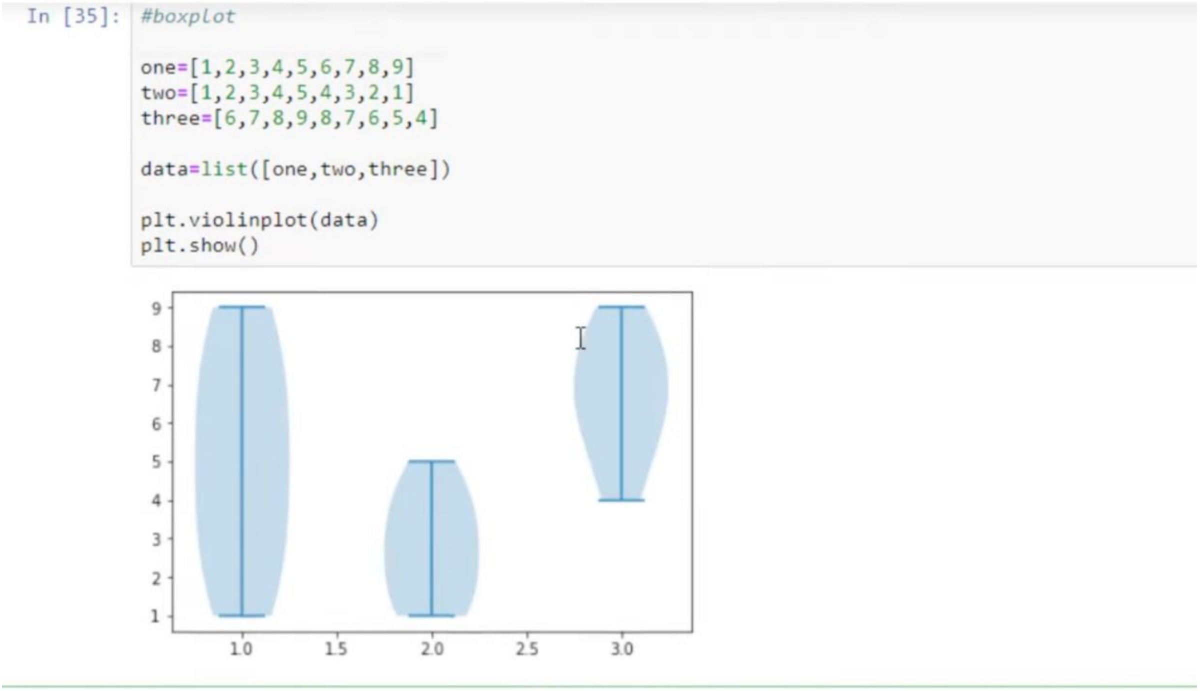


```
In [29]: plt.hist(iris['Sepal.Length'])
         plt.show()
```



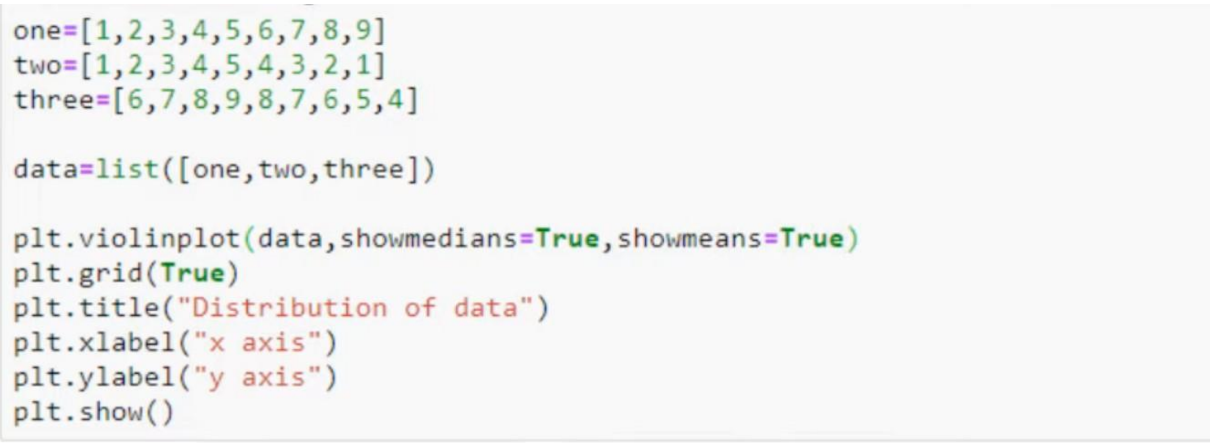
In [34]:

```
#boxplot

one=[1,2,3,4,5,6,7,8,9]
two=[1,2,3,4,5,4,3,2,1]
three=[6,7,8,9,8,7,6,5,4]

data=list([one,two,three])

plt.boxplot(data)
plt.show()
```

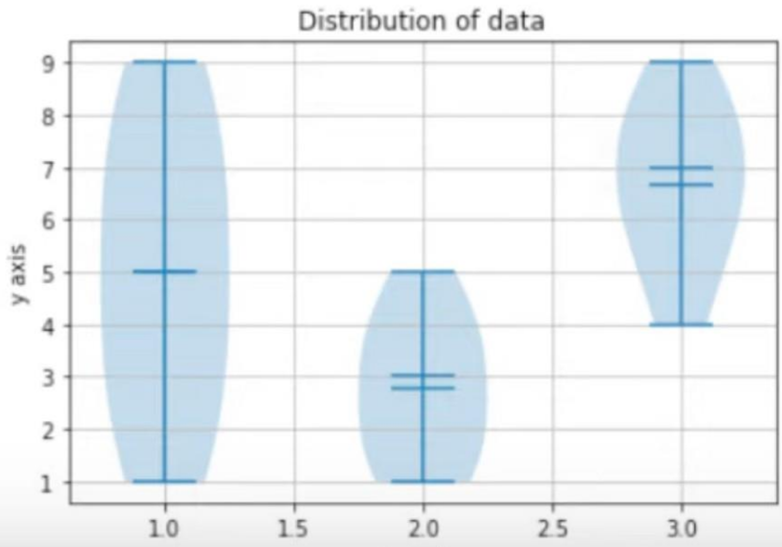


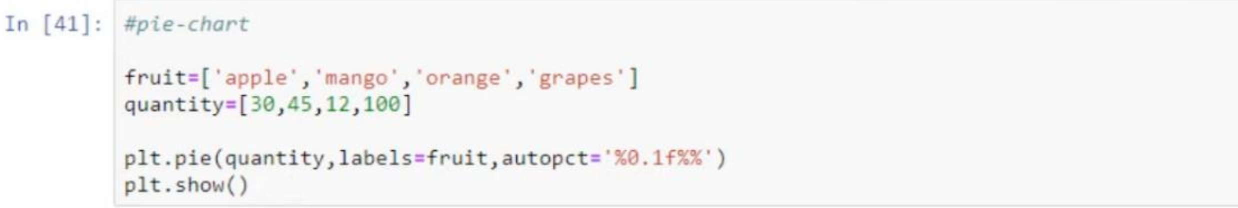
In [35]:

```
#boxplot

one=[1,2,3,4,5,6,7,8,9]
two=[1,2,3,4,5,4,3,2,1]
three=[6,7,8,9,8,7,6,5,4]

data=list([one,two,three])

plt.violinplot(data)
plt.show()
```

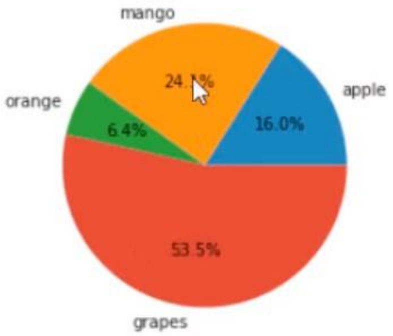


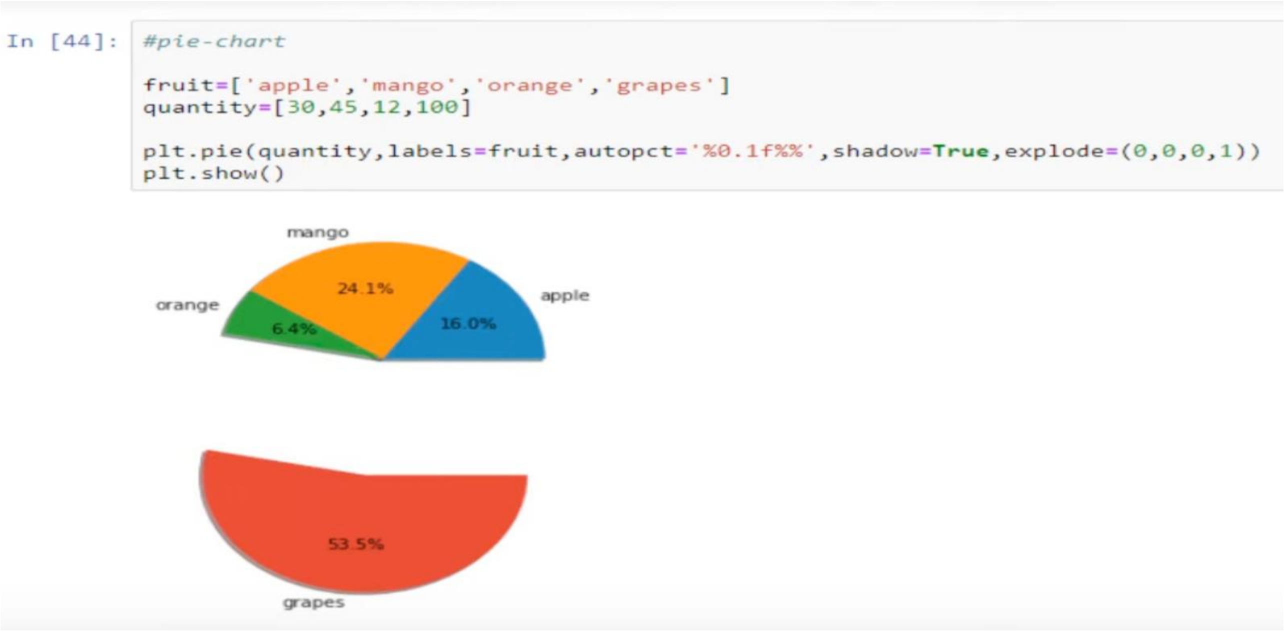
```
one=[1,2,3,4,5,6,7,8,9]
two=[1,2,3,4,5,4,3,2,1]
three=[6,7,8,9,8,7,6,5,4]

data=list([one,two,three])

plt.violinplot(data,showmedians=True,showmeans=True)
plt.grid(True)
plt.title("Distribution of data")
plt.xlabel("x axis")
plt.ylabel("y axis")
plt.show()
```



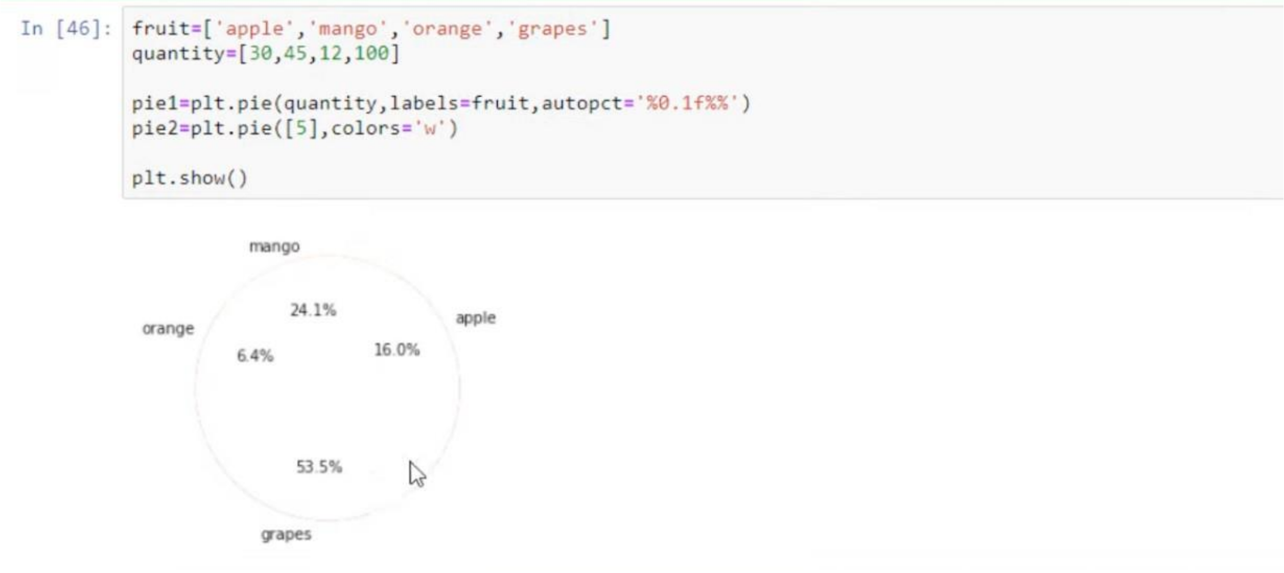


```
In [41]: #pie-chart

         fruit=['apple','mango','orange','grapes']
         quantity=[30,45,12,100]

         plt.pie(quantity,labels=fruit,autopct='%0.1f%%')
         plt.show()
```

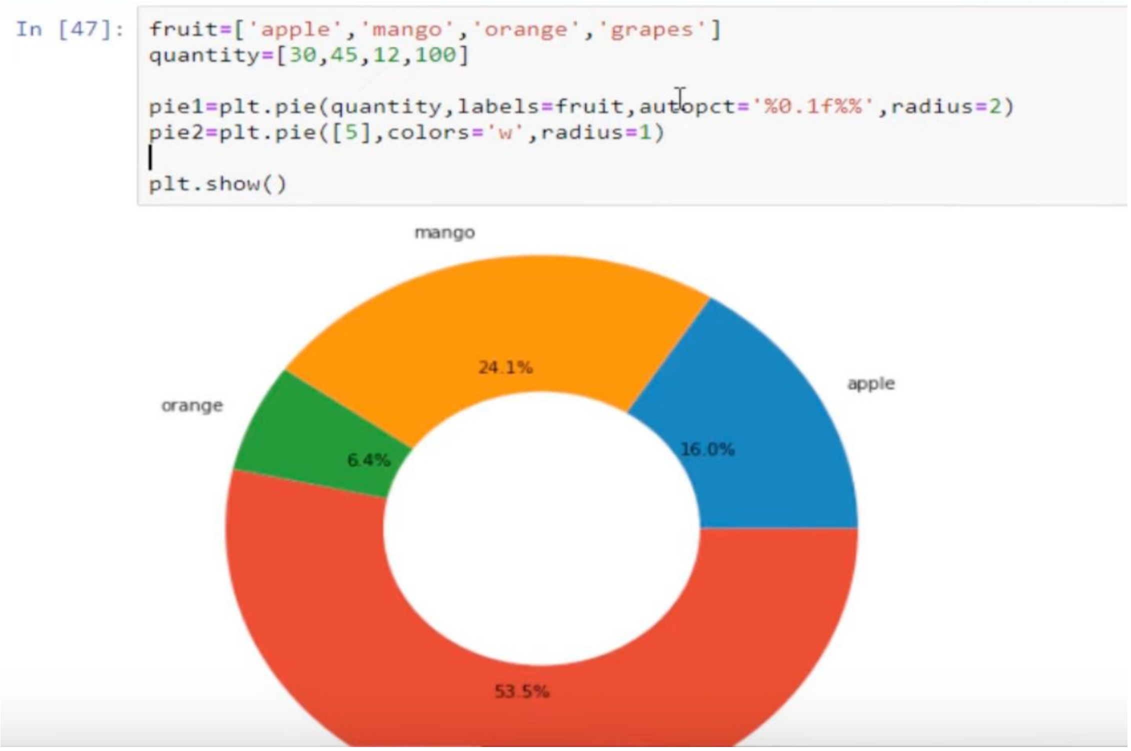




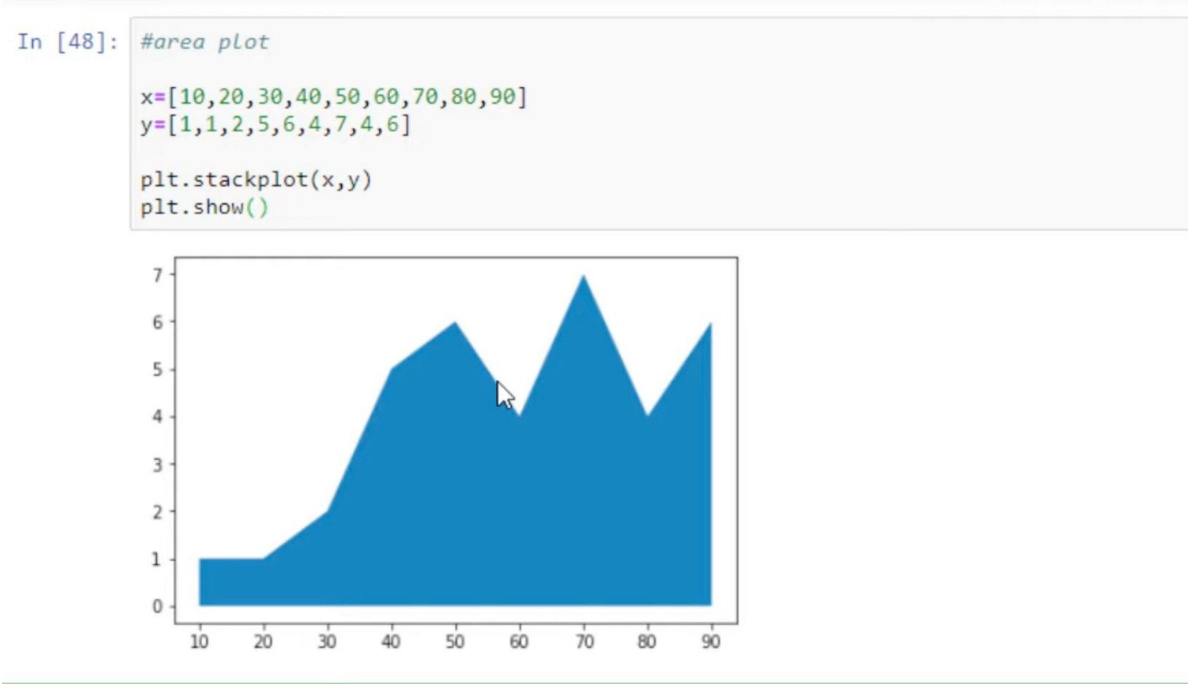
```
In [44]: #pie-chart

         fruit=['apple','mango','orange','grapes']
         quantity=[30,45,12,100]

         plt.pie(quantity,labels=fruit,autopct='%0.1f%%',shadow=True,explode=(0,0,0,1))
         plt.show()
```



```
In [46]: fruit=['apple','mango','orange','grapes']
         quantity=[30,45,12,100]

         pie1=plt.pie(quantity,labels=fruit,autopct='%0.1f%%')
         pie2=plt.pie([5],colors='w')

         plt.show()
```



```
In [47]: fruit=['apple','mango','orange','grapes']
         quantity=[30,45,12,100]

         pie1=plt.pie(quantity,labels=fruit,autopct='%0.1f%%',radius=2)
         pie2=plt.pie([5],colors='w',radius=1)

         plt.show()
```

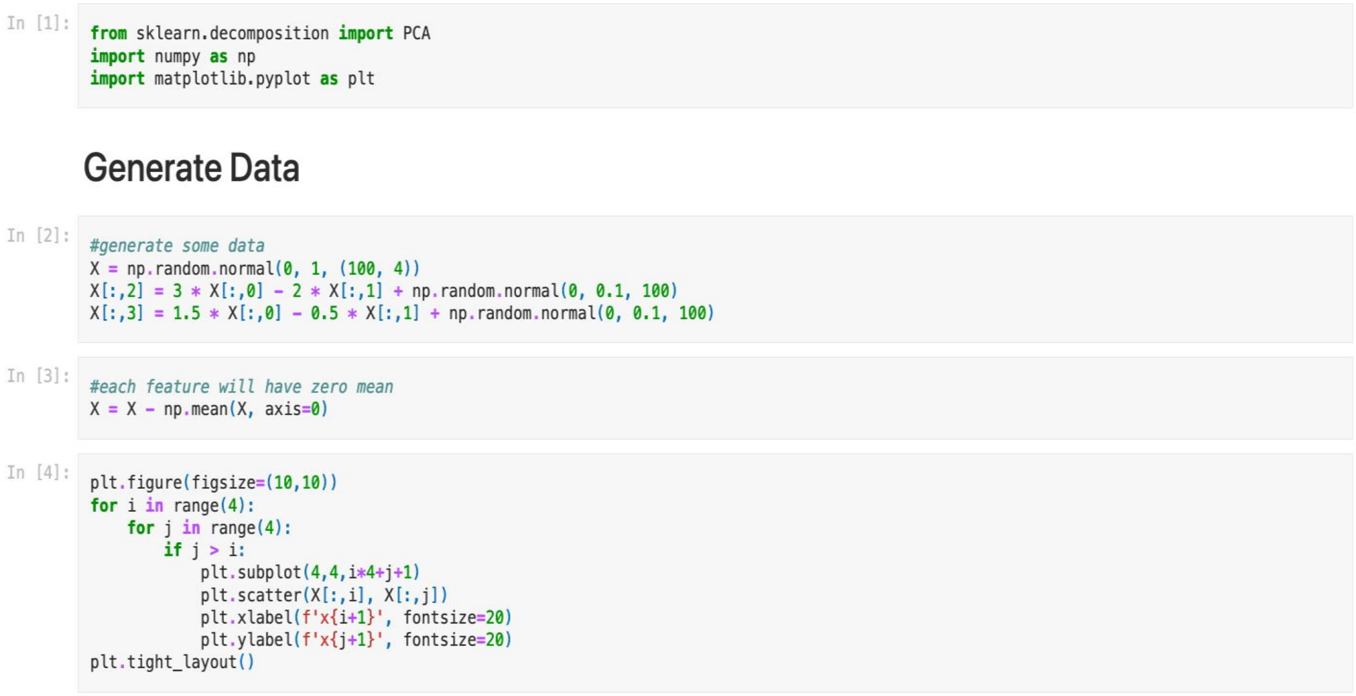


```
In [48]: #area plot

         x=[10,20,30,40,50,60,70,80,90]
         y=[1,1,2,5,6,4,7,4,6]

         plt.stackplot(x,y)
         plt.show()
```

Week 4:

Implement Dimensionality reduction using Principle Component Analysis (PCA) method.

Program:



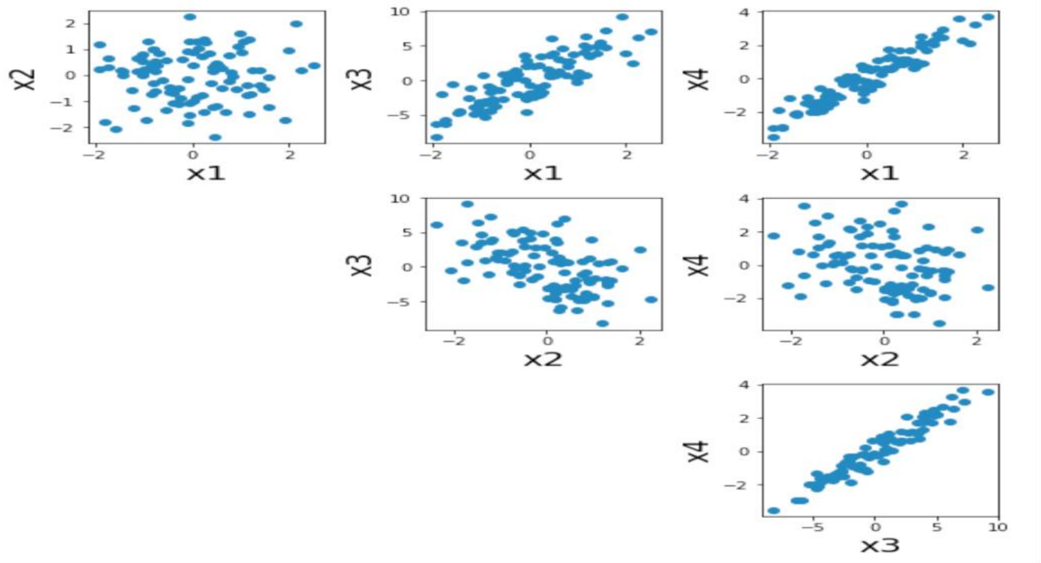
In [1]:
```
from sklearn.decomposition import PCA
import numpy as np
import matplotlib.pyplot as plt
```

Generate Data

In [2]:
```
#generate some data
X = np.random.normal(0, 1, (100, 4))
X[:,2] = 3 * X[:,0] - 2 * X[:,1] + np.random.normal(0, 0.1, 100)
X[:,3] = 1.5 * X[:,0] - 0.5 * X[:,1] + np.random.normal(0, 0.1, 100)
```

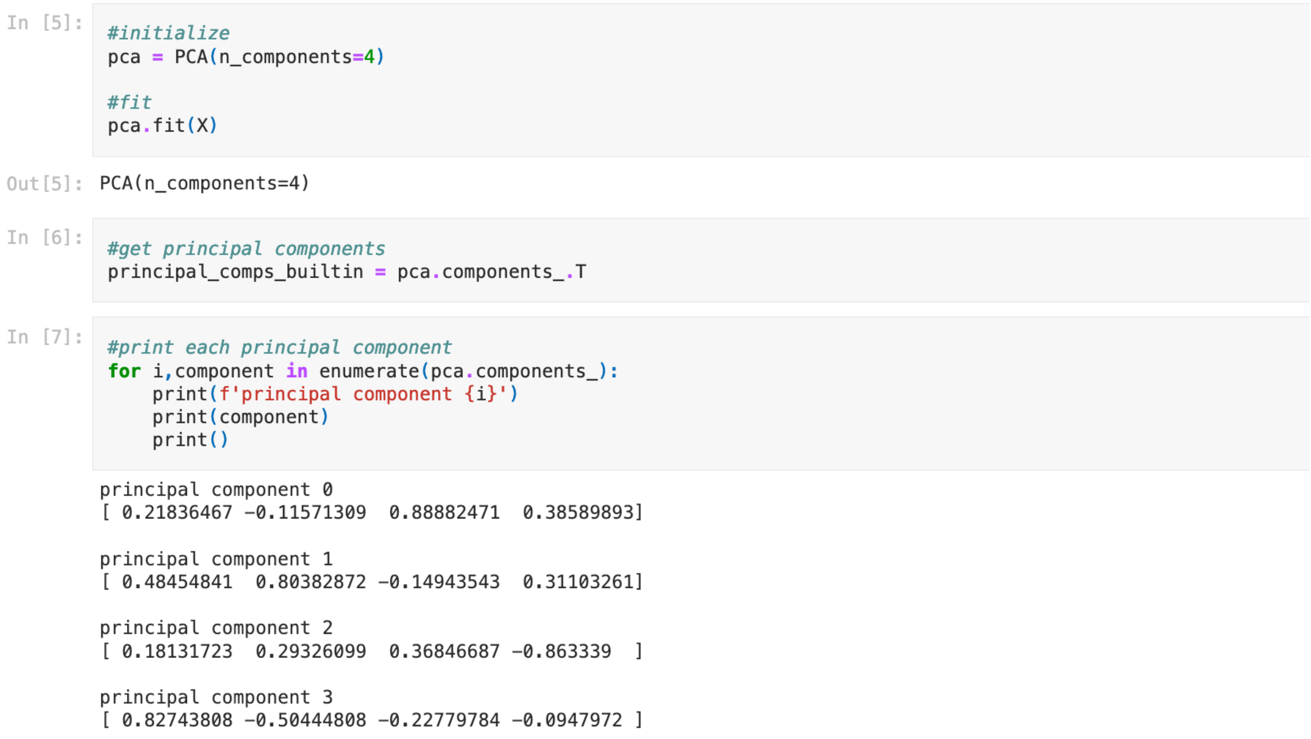
In [3]:
```
#each feature will have zero mean
X = X - np.mean(X, axis=0)
```

In [4]:
```
plt.figure(figsize=(10,10))
for i in range(4):
    for j in range(4):
        if j > i:
            plt.subplot(4,4,i*4+j+1)
            plt.scatter(X[:,i], X[:,j])
            plt.xlabel(f'x{i+1}', fontsize=20)
            plt.ylabel(f'x{j+1}', fontsize=20)
plt.tight_layout()
```



Observations:

- x1 and x2 do not seem correlated
- x1 seems very correlated with both x3 and x4
- x2 seems somewhat correlated with both x3 and x4
- x3 and x4 seem very correlated

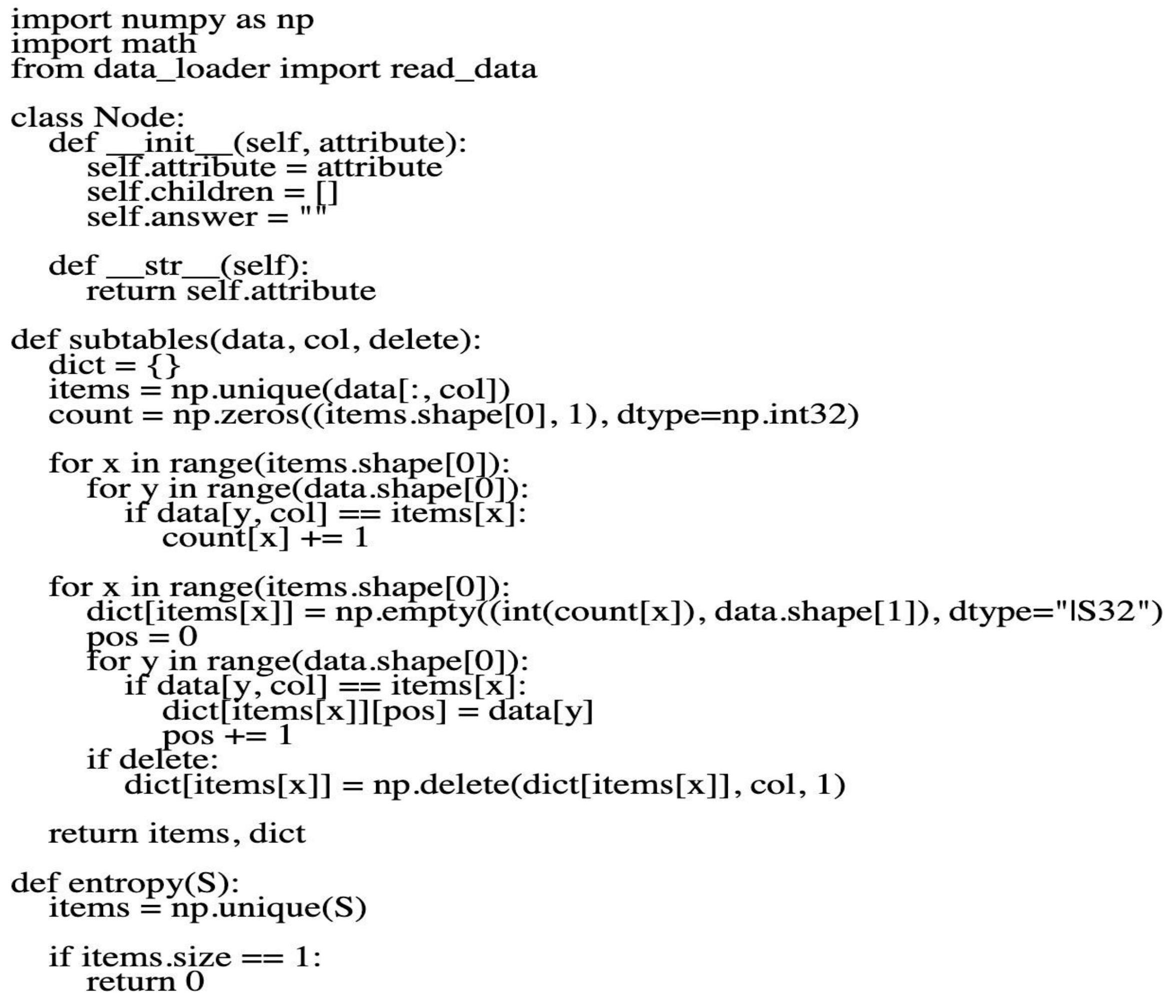


```
In [5]: #initialize
        pca = PCA(n_components=4)

        #fit
        pca.fit(X)

Out[5]: PCA(n_components=4)

In [6]: #get principal components
        principal_comps_builtin = pca.components_.T

In [7]: #print each principal component
        for i,component in enumerate(pca.components_):
            print(f'principal component {i}')
            print(component)
            print()

        principal component 0
        [ 0.21836467 -0.11571309  0.88882471  0.38589893]

        principal component 1
        [ 0.48454841  0.80382872 -0.14943543  0.31103261]

        principal component 2
        [ 0.18131723  0.29326099  0.36846687 -0.863339  ]

        principal component 3
        [ 0.82743808 -0.50444808 -0.22779784 -0.0947972 ]
```

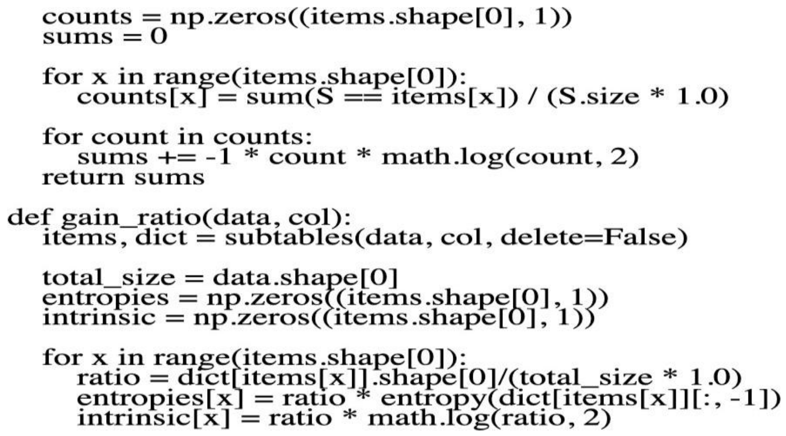
Week 5:

Develop Decision Tree Classification model for a given dataset and use it to classify a new sample.

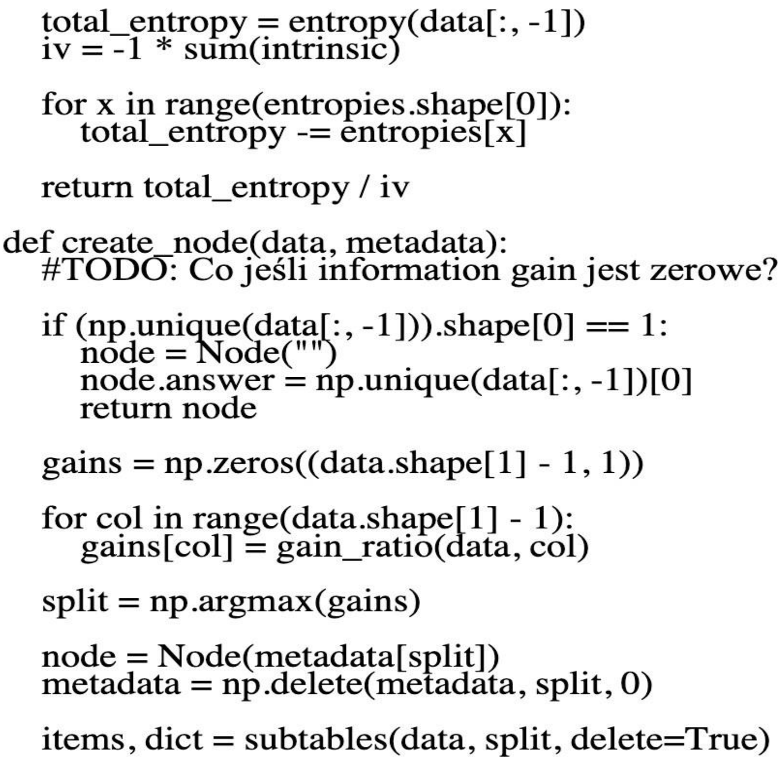
Program:



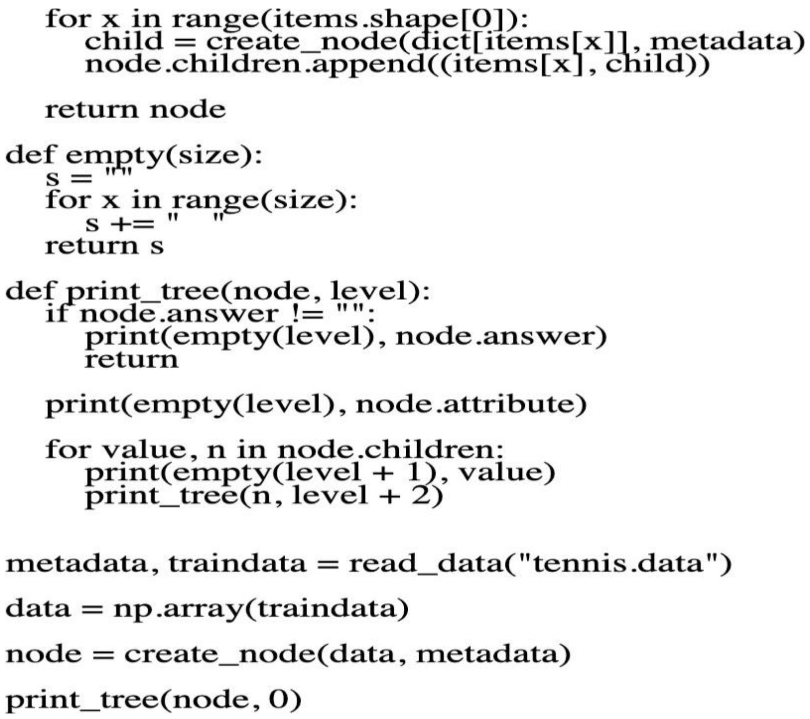
```
import numpy as np
import math
from data_loader import read_data

class Node:
    def __init__(self, attribute):
        self.attribute = attribute
        self.children = []
        self.answer = ""

    def __str__(self):
        return self.attribute

def subtables(data, col, delete):
    dict = {}
    items = np.unique(data[:, col])
    count = np.zeros((items.shape[0], 1), dtype=np.int32)

    for x in range(items.shape[0]):
        for y in range(data.shape[0]):
            if data[y, col] == items[x]:
                count[x] += 1

    for x in range(items.shape[0]):
        dict[items[x]] = np.empty((int(count[x]), data.shape[1]), dtype="|S32")
        pos = 0
        for y in range(data.shape[0]):
            if data[y, col] == items[x]:
                dict[items[x]][pos] = data[y]
                pos += 1
        if delete:
            dict[items[x]] = np.delete(dict[items[x]], col, 1)

    return items, dict

def entropy(S):
    items = np.unique(S)

    if items.size == 1:
        return 0
```



```
    counts = np.zeros((items.shape[0], 1))
    sums = 0

    for x in range(items.shape[0]):
        counts[x] = sum(S == items[x]) / (S.size * 1.0)

    for count in counts:
        sums += -1 * count * math.log(count, 2)
    return sums

def gain_ratio(data, col):
    items, dict = subtables(data, col, delete=False)

    total_size = data.shape[0]
    entropies = np.zeros((items.shape[0], 1))
    intrinsic = np.zeros((items.shape[0], 1))

    for x in range(items.shape[0]):
        ratio = dict[items[x]].shape[0]/(total_size * 1.0)
        entropies[x] = ratio * entropy(dict[items[x]][:, -1])
        intrinsic[x] = ratio * math.log(ratio, 2)
```



```
    total_entropy = entropy(data[:, -1])
    iv = -1 * sum(intrinsic)

    for x in range(entropies.shape[0]):
        total_entropy -= entropies[x]

    return total_entropy / iv

def create_node(data, metadata):
    #TODO: Co jeśli information gain jest zerowe?

    if (np.unique(data[:, -1])).shape[0] == 1:
        node = Node("")
        node.answer = np.unique(data[:, -1])[0]
        return node

    gains = np.zeros((data.shape[1] - 1, 1))

    for col in range(data.shape[1] - 1):
        gains[col] = gain_ratio(data, col)

    split = np.argmax(gains)

    node = Node(metadata[split])
    metadata = np.delete(metadata, split, 0)

    items, dict = subtables(data, split, delete=True)
```



```
    for x in range(items.shape[0]):
        child = create_node(dict[items[x]], metadata)
        node.children.append((items[x], child))

    return node

def empty(size):
    s = ""
    for x in range(size):
        s += "   "
    return s

def print_tree(node, level):
    if node.answer != "":
        print(empty(level), node.answer)
        return

    print(empty(level), node.attribute)

    for value, n in node.children:
        print(empty(level + 1), value)
        print_tree(n, level + 2)


metadata, traindata = read_data("tennis.data")

data = np.array(traindata)

node = create_node(data, metadata)

print_tree(node, 0)
```

```
outlook
overcast
b'yes'
rain
wind
b'strong'
b'no'
b'weak'
b'yes'
sunny
humidity
b'high'
b'no'
b'normal'
b'yes'
```

Week 6:

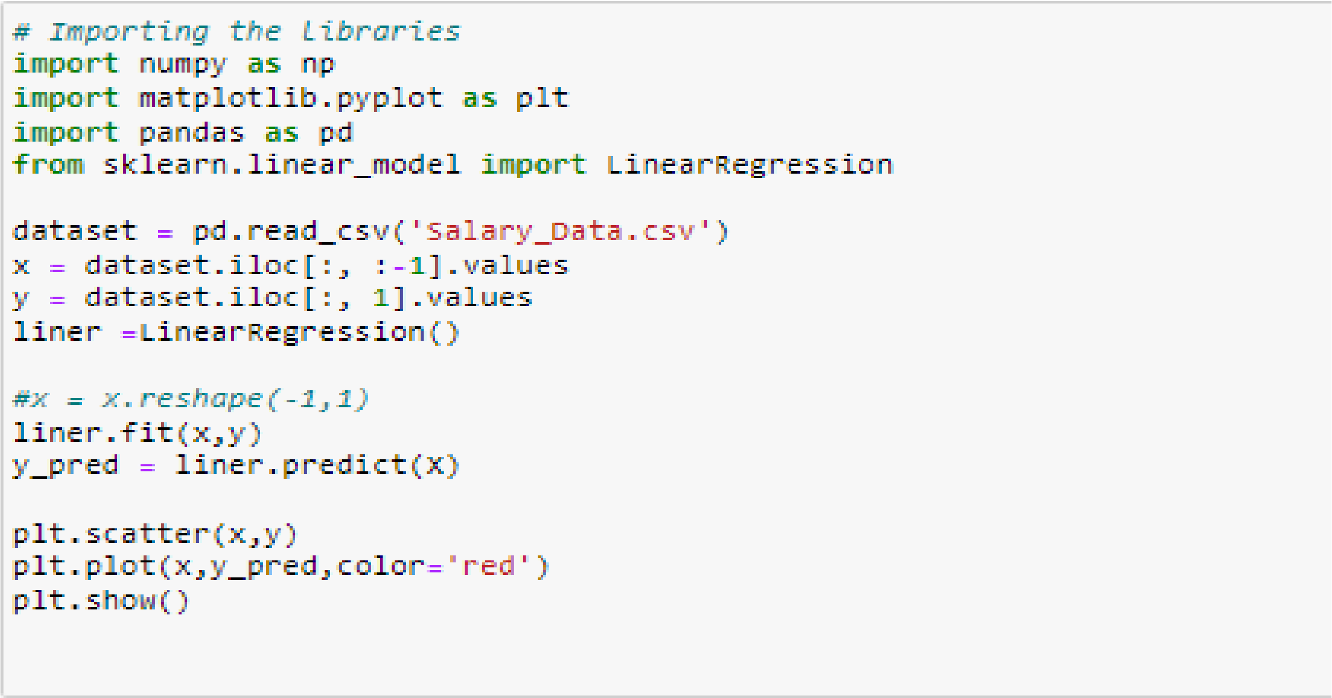
Consider a dataset use Random Forest to predict the output class vary the number of trees as follows and compare the results. i) 20 ii)50 iii)100 iv)200 v)500

Week 7:

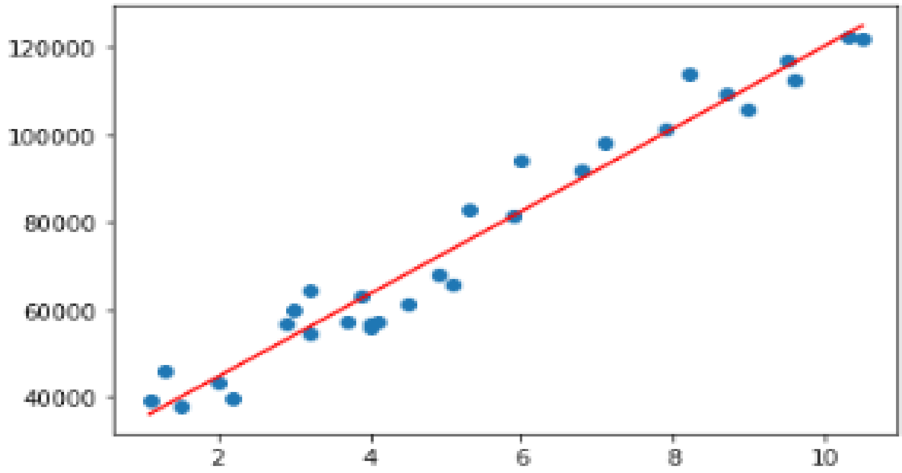
Write a python program to implement Simple Linear Regression Models and plot the graph.

Program:

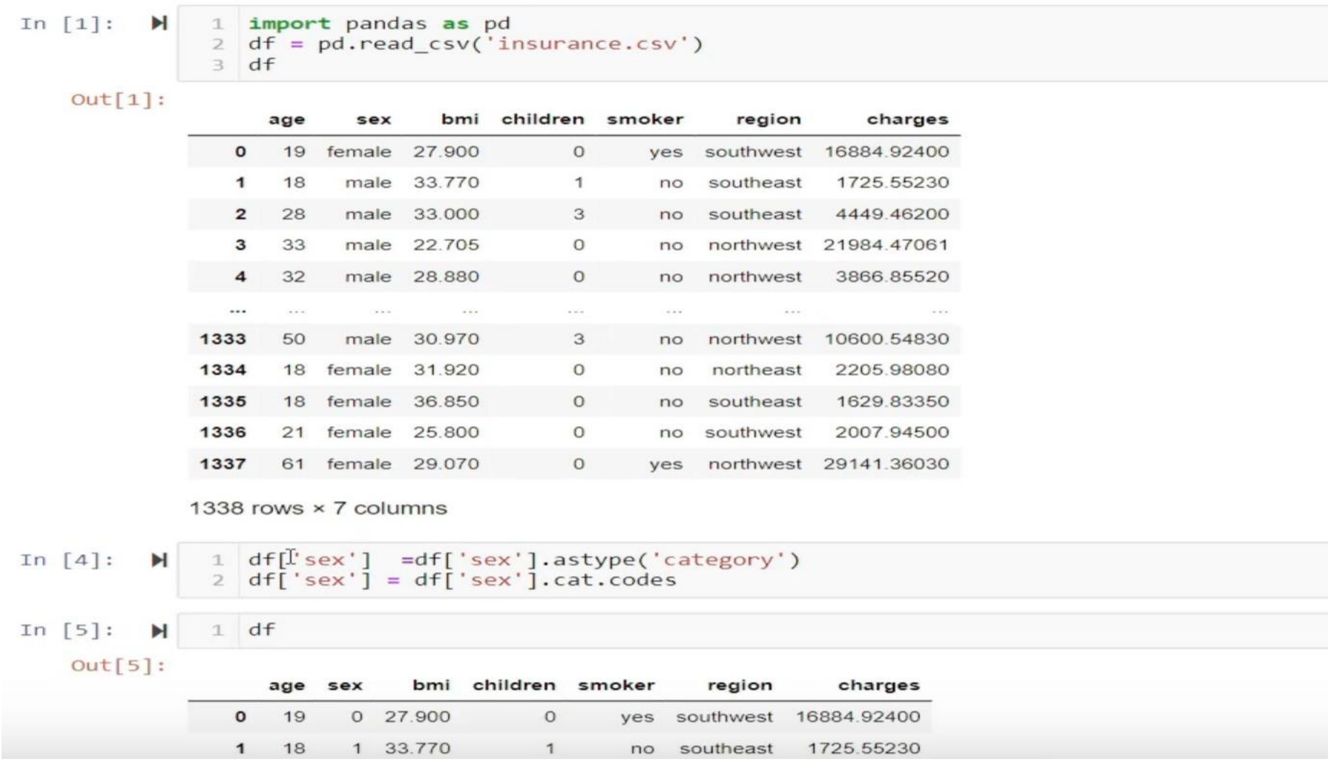
a) To implement Simple Linear Regression.



```
# Importing the Libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.linear_model import LinearRegression

dataset = pd.read_csv('Salary_Data.csv')
x = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values
liner =LinearRegression()

#x = x.reshape(-1,1)
liner.fit(x,y)
y_pred = liner.predict(x)

plt.scatter(x,y)
plt.plot(x,y_pred,color='red')
plt.show()
```



b) To implement Multiple Linear Regression.



In [1]:
```
import pandas as pd
df = pd.read_csv('insurance.csv')
df
```

Out[1]:

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520
...	...	...	...	...	...	...	...
1333	50	male	30.970	3	no	northwest	10600.54830
1334	18	female	31.920	0	no	northeast	2205.98080
1335	18	female	36.850	0	no	southeast	1629.83350
1336	21	female	25.800	0	no	southwest	2007.94500
1337	61	female	29.070	0	yes	northwest	29141.36030

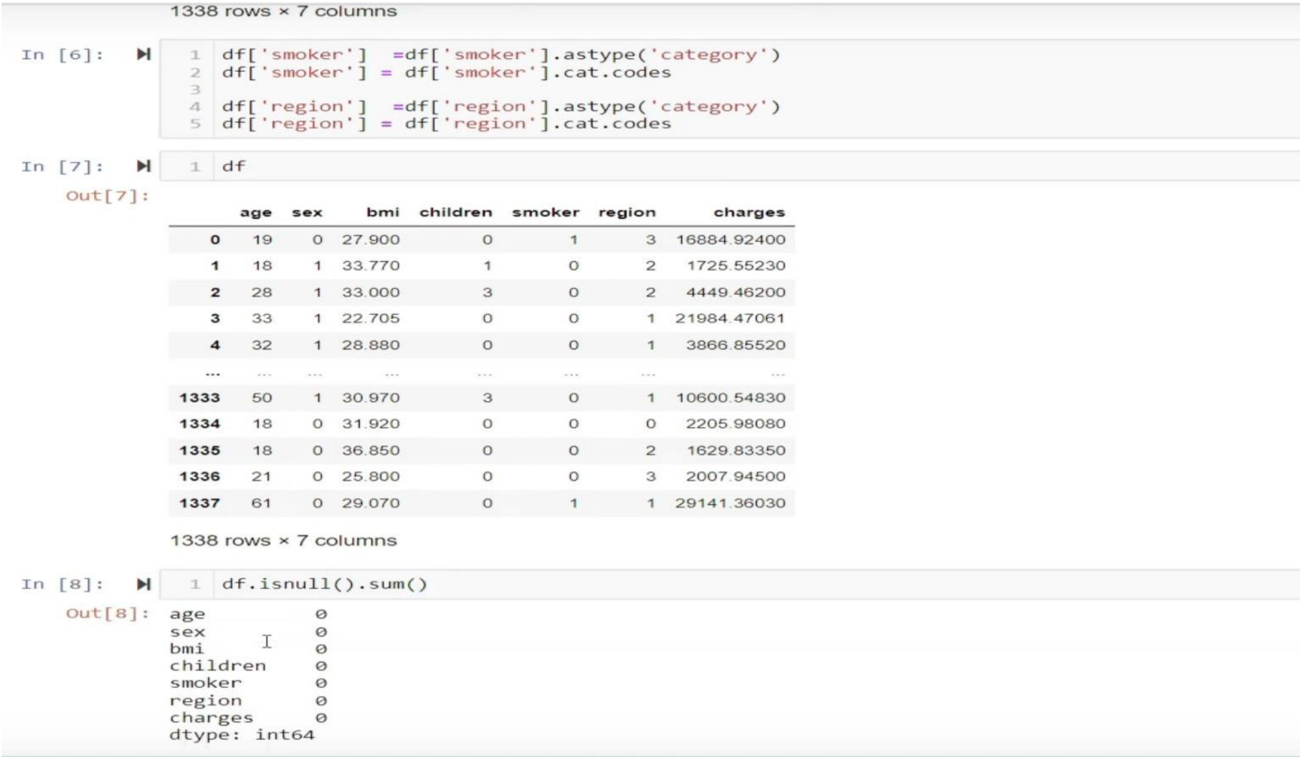
1338 rows × 7 columns

In [4]:
```
df['sex']  =df['sex'].astype('category')
df['sex'] = df['sex'].cat.codes
```

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

Out[5]:

	age	sex	bmi	children	smoker	region	charges
0	19	0	27.900	0	yes	southwest	16884.92400
1	18	1	33.770	1	no	southeast	1725.55230



1338 rows × 7 columns

In [6]:
```
df['smoker']  =df['smoker'].astype('category')
df['smoker'] = df['smoker'].cat.codes

df['region']  =df['region'].astype('category')
df['region'] = df['region'].cat.codes
```

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

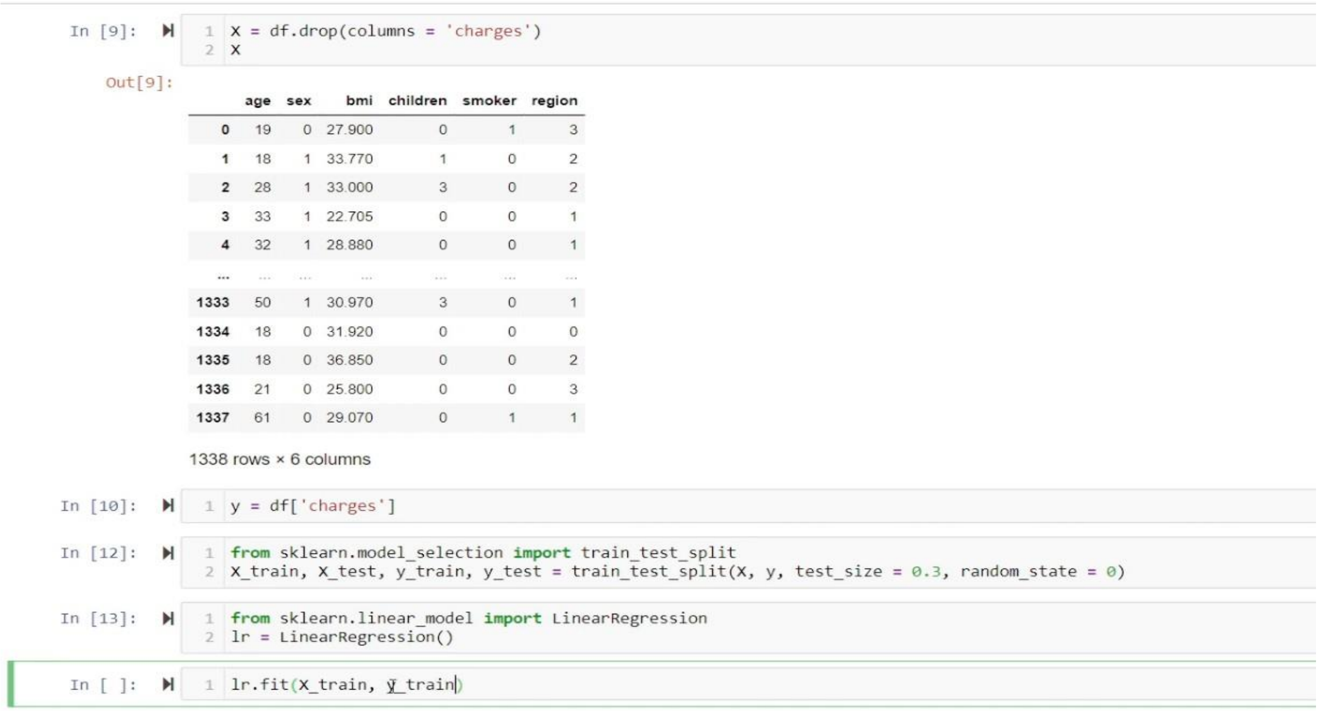
Out[7]:

	age	sex	bmi	children	smoker	region	charges
0	19	0	27.900	0	1	3	16884.92400
1	18	1	33.770	1	0	2	1725.55230
2	28	1	33.000	3	0	2	4449.46200
3	33	1	22.705	0	0	1	21984.47061
4	32	1	28.880	0	0	1	3866.85520
...	...	...	...	...	...	...	...
1333	50	1	30.970	3	0	1	10600.54830
1334	18	0	31.920	0	0	0	2205.98080
1335	18	0	36.850	0	0	2	1629.83350
1336	21	0	25.800	0	0	3	2007.94500
1337	61	0	29.070	0	1	1	29141.36030

1338 rows × 7 columns

In [8]:
```
df.isnull().sum()
```

Out[8]:
```
age         0
sex         0
bmi         0
children    0
smoker      0
region      0
charges     0
dtype: int64
```



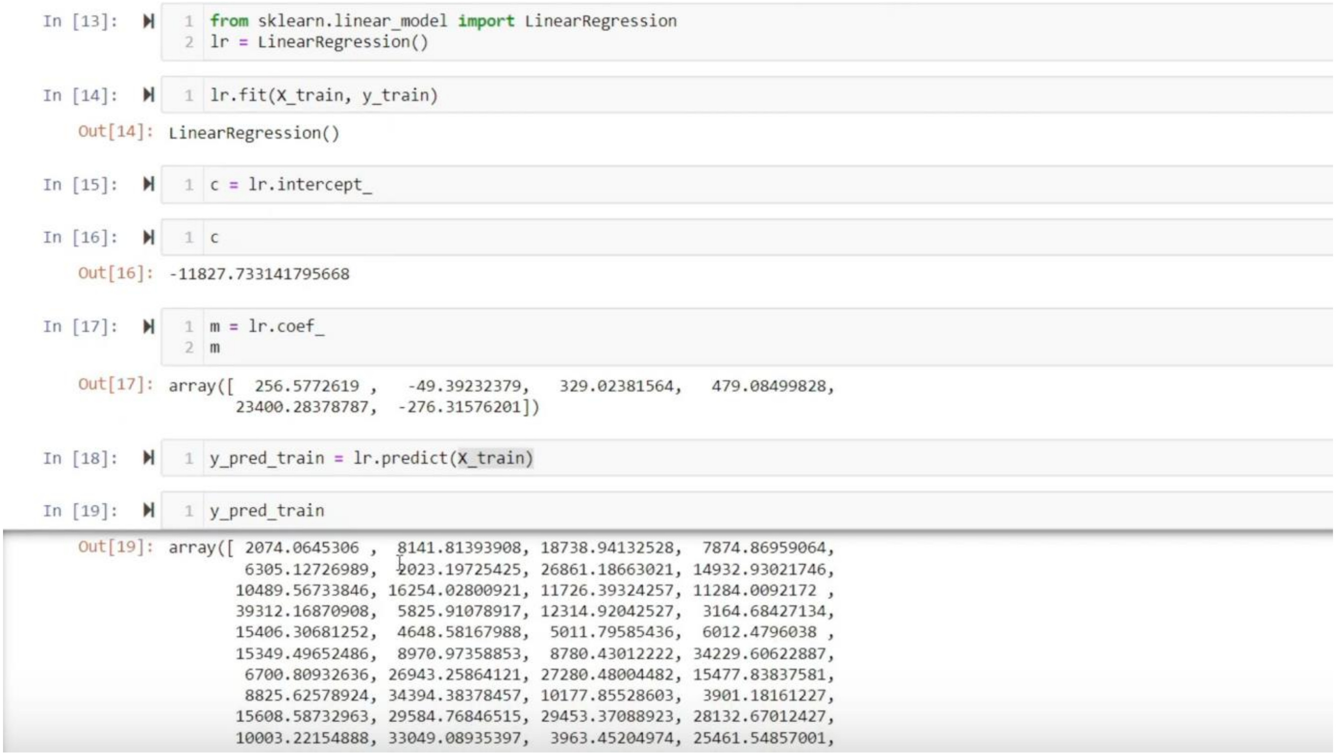
```
In [9]:   X = df.drop(columns = 'charges')
          X
```

Out[9]:

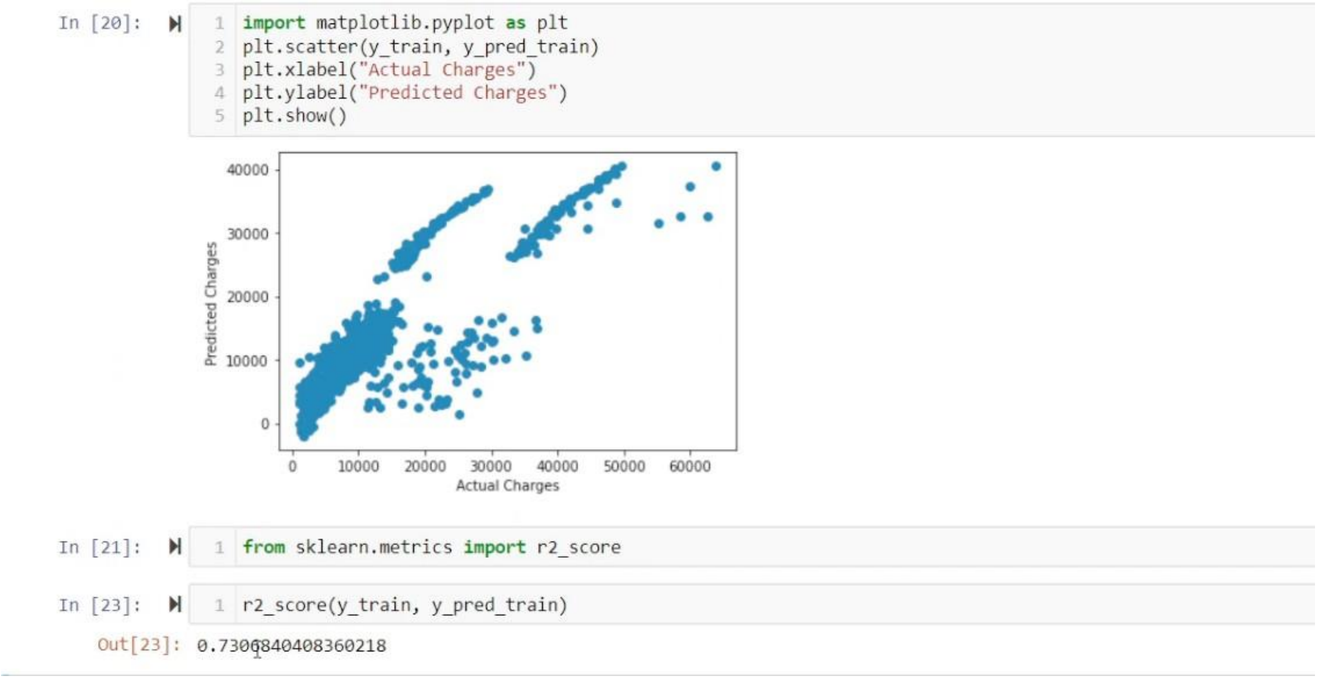
	age	sex	bmi	children	smoker	region
0	19	0	27.900	0	1	3
1	18	1	33.770	1	0	2
2	28	1	33.000	3	0	2
3	33	1	22.705	0	0	1
4	32	1	28.880	0	0	1
...	...	...	...	...	...	...
1333	50	1	30.970	3	0	1
1334	18	0	31.920	0	0	0
1335	18	0	36.850	0	0	2
1336	21	0	25.800	0	0	3
1337	61	0	29.070	0	1	1

1338 rows × 6 columns

```
In [10]:  y = df['charges']

In [12]:  from sklearn.model_selection import train_test_split
          X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 0)

In [13]:  from sklearn.linear_model import LinearRegression
          lr = LinearRegression()

In [ ]:   lr.fit(X_train, y_train)
```



```
In [13]:  from sklearn.linear_model import LinearRegression
          lr = LinearRegression()

In [14]:  lr.fit(X_train, y_train)

Out[14]: LinearRegression()

In [15]:  c = lr.intercept_

In [16]:  c

Out[16]: -11827.733141795668

In [17]:  m = lr.coef_
          m

Out[17]: array([  256.5772619 ,   -49.39232379,   329.02381564,   479.08499828,
                23400.28378787,  -276.31576201])

In [18]:  y_pred_train = lr.predict(X_train)

In [19]:  y_pred_train

Out[19]: array([ 2074.0645306 ,  8141.81393908, 18738.94132528,  7874.86959064,
                 6305.12726989,  2023.19725425, 26861.18663021, 14932.93021746,
                10489.56733846, 16254.02800921, 11726.39324257, 11284.0092172 ,
                39312.16870908,  5825.91078917, 12314.92042527,  3164.68427134,
                15406.30681252,  4648.58167988,  5011.79585436,  6012.4796038 ,
                15349.49652486,  8970.97358853,  8780.43012222, 34229.60622887,
                 6700.80932636, 26943.25864121, 27280.48004482, 15477.83837581,
                 8825.62578924, 34394.38378457, 10177.85528603,  3901.18161227,
                15608.58732963, 29584.76846515, 29453.37088923, 28132.67012427,
                10003.22154888, 33049.08935397,  3963.45204974, 25461.54857001,
```

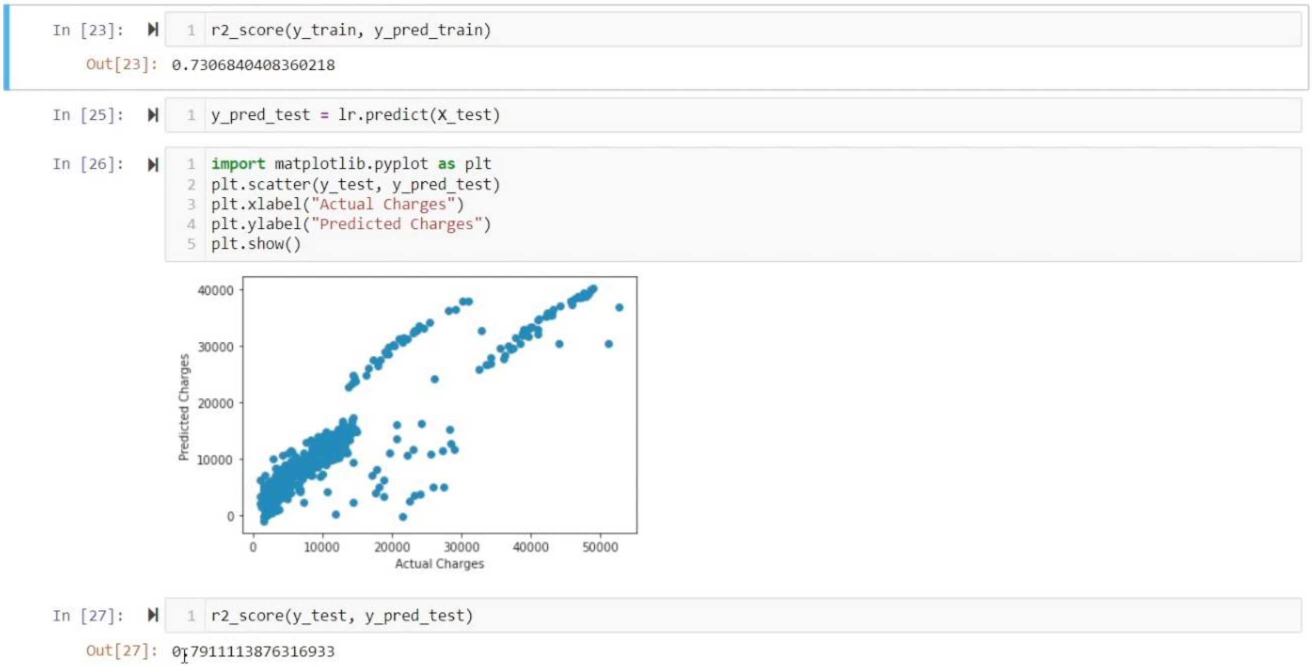


```
import matplotlib.pyplot as plt
plt.scatter(y_train, y_pred_train)
plt.xlabel("Actual Charges")
plt.ylabel("Predicted Charges")
plt.show()
```

```
from sklearn.metrics import r2_score
```

```
r2_score(y_train, y_pred_train)
```

```
0.7306840408360218
```



```
r2_score(y_train, y_pred_train)
```

```
0.7306840408360218
```

```
y_pred_test = lr.predict(X_test)
```

```
import matplotlib.pyplot as plt
plt.scatter(y_test, y_pred_test)
plt.xlabel("Actual Charges")
plt.ylabel("Predicted Charges")
plt.show()
```

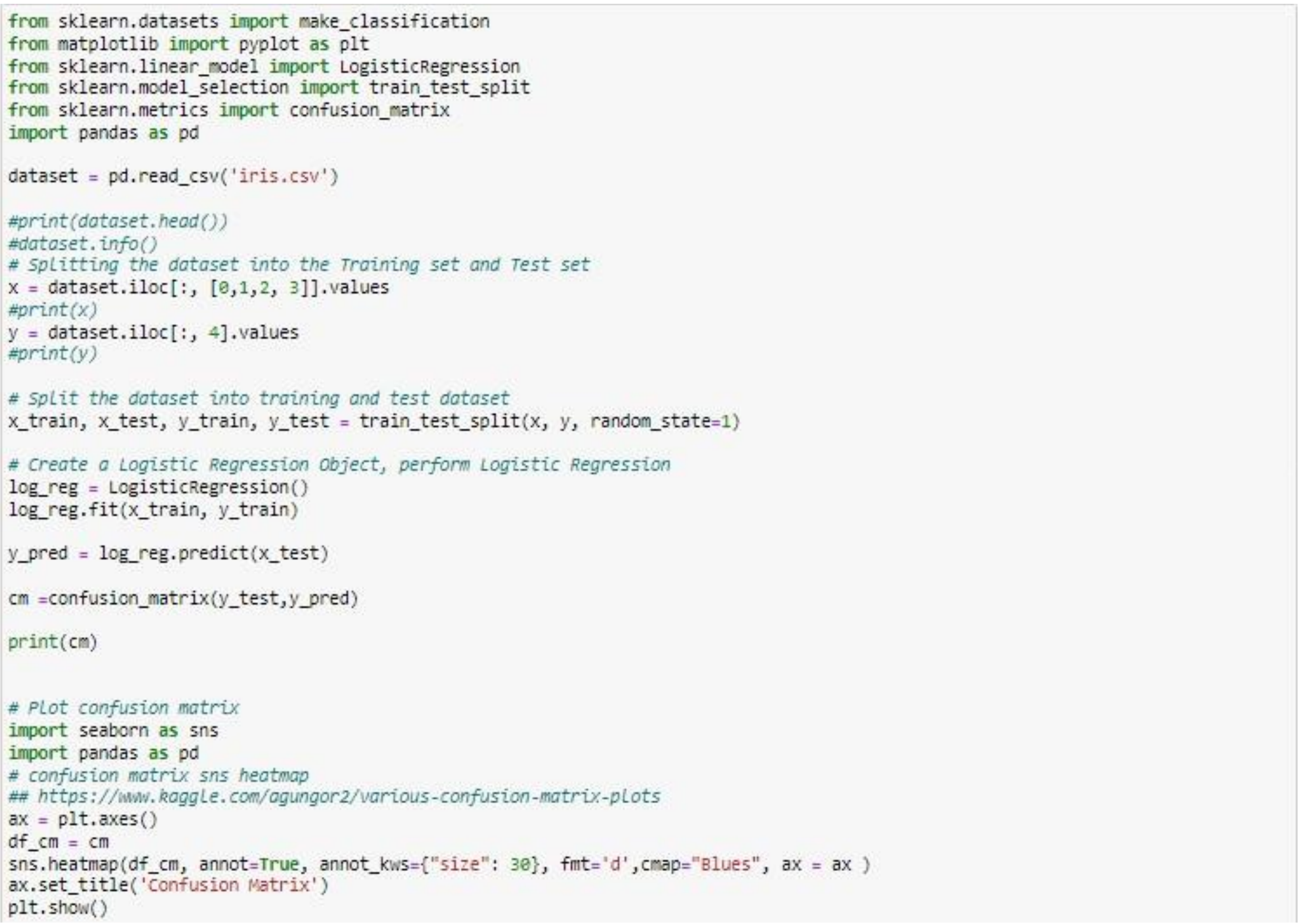
```
r2_score(y_test, y_pred_test)
```

```
0.7911113876316933
```

Week 8:

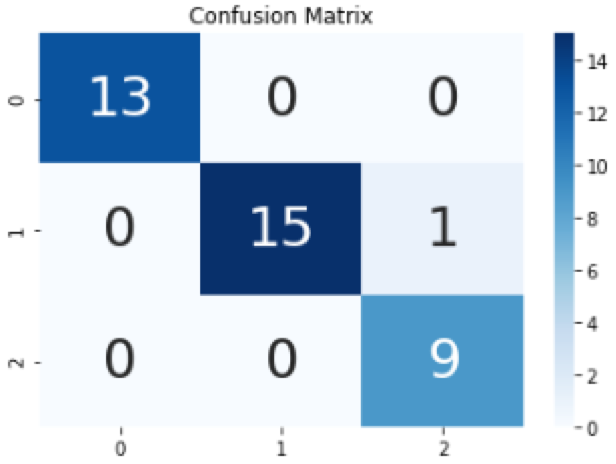
Write a python program to implement Logistic Regression Model for a given dataset.

Program:



```
from sklearn.datasets import make_classification
from matplotlib import pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import pandas as pd

dataset = pd.read_csv('iris.csv')

#print(dataset.head())
#dataset.info()
# Splitting the dataset into the Training set and Test set
x = dataset.iloc[:, [0,1,2, 3]].values
#print(x)
y = dataset.iloc[:, 4].values
#print(y)

# Split the dataset into training and test dataset
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=1)

# Create a Logistic Regression Object, perform Logistic Regression
log_reg = LogisticRegression()
log_reg.fit(x_train, y_train)

y_pred = log_reg.predict(x_test)

cm =confusion_matrix(y_test,y_pred)

print(cm)


# Plot confusion matrix
import seaborn as sns
import pandas as pd
# confusion matrix sns heatmap
## https://www.kaggle.com/agungor2/various-confusion-matrix-plots
ax = plt.axes()
df_cm = cm
sns.heatmap(df_cm, annot=True, annot_kws={"size": 30}, fmt='d',cmap="Blues", ax = ax )
ax.set_title('Confusion Matrix')
plt.show()
```

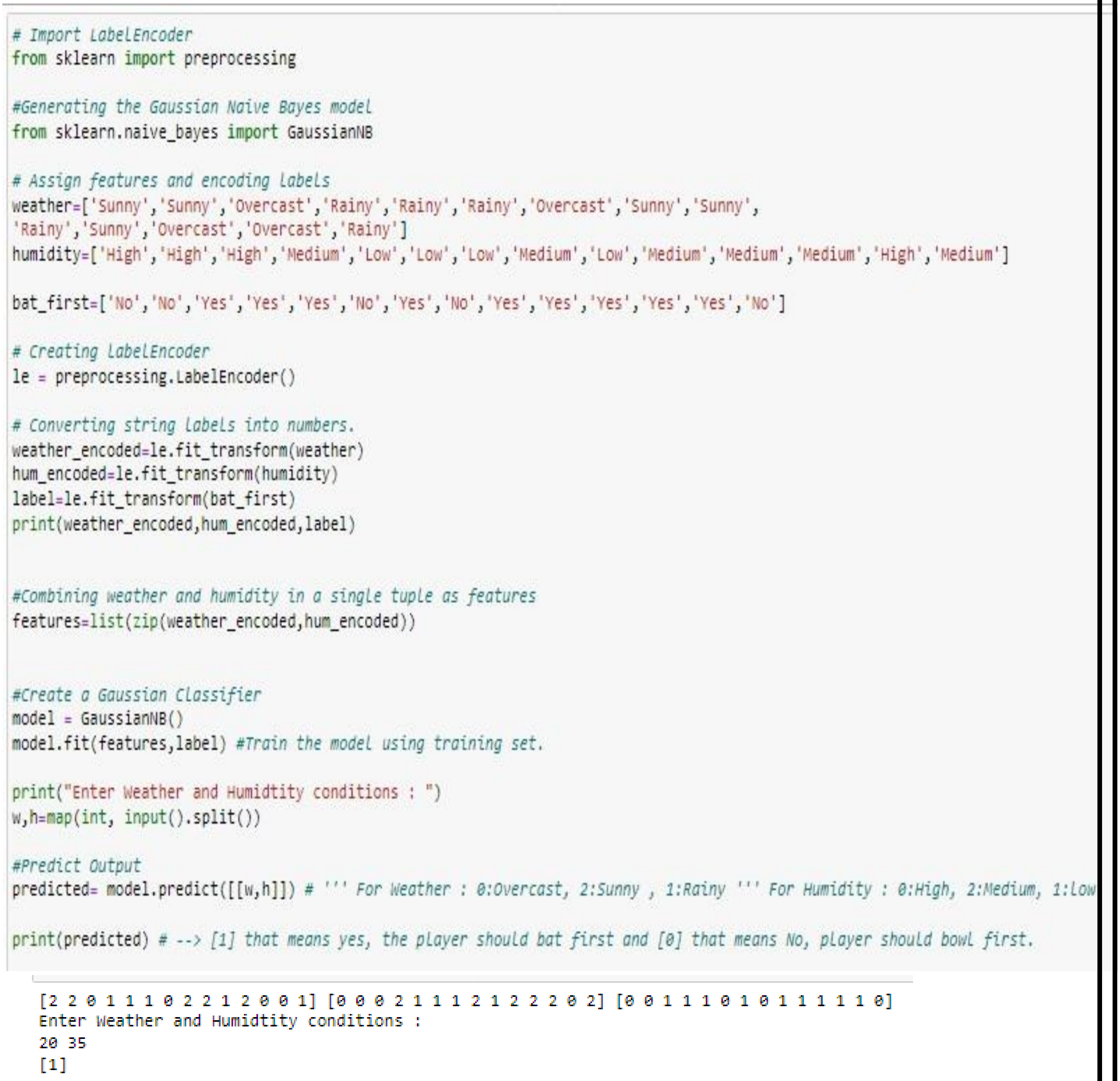
```
[[13  0  0]
 [ 0 15  1]
 [ 0  0  9]]
```



Excersice:

Implement Naive Bayes classification in python.

Program:



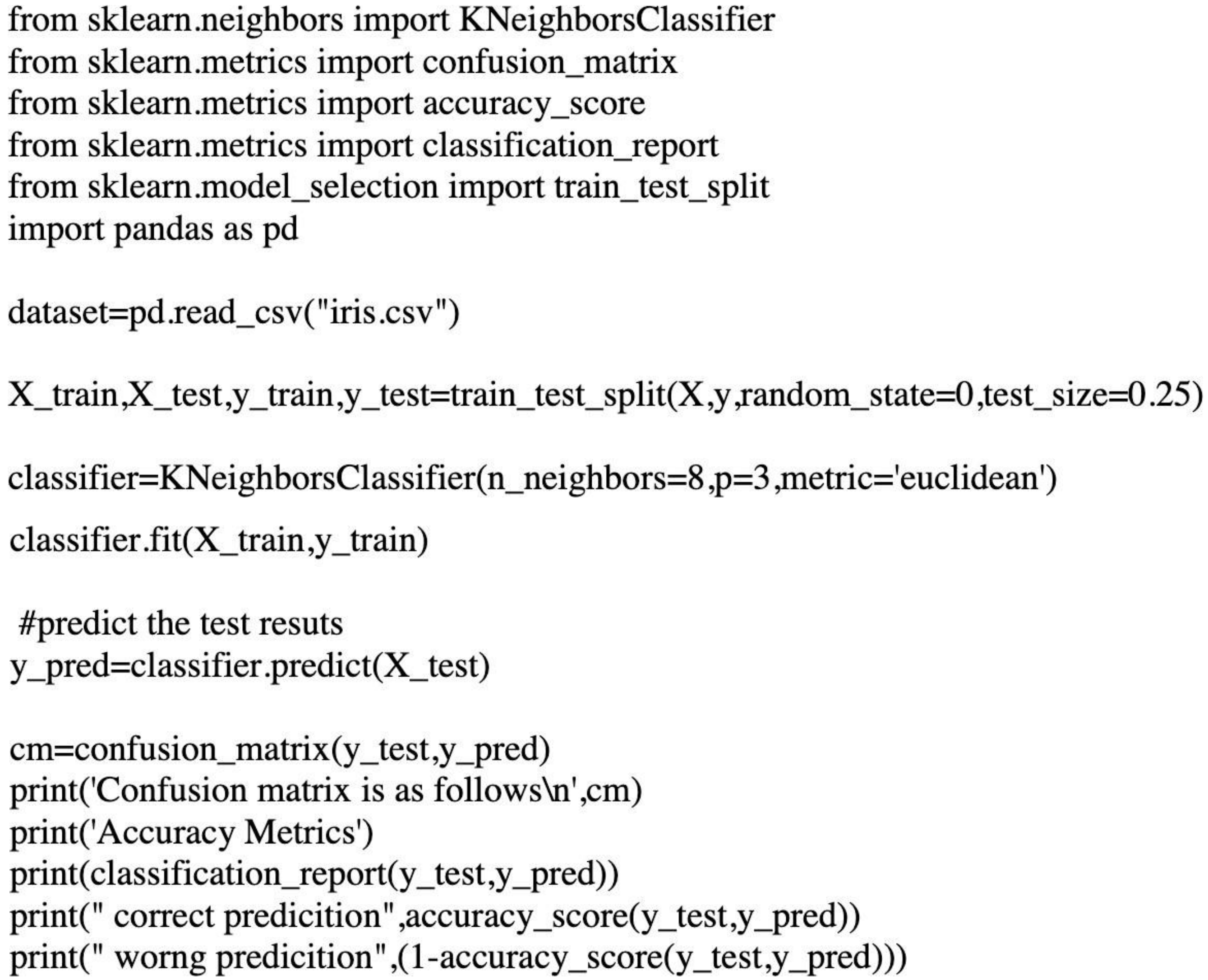
```
# Import LabelEncoder
from sklearn import preprocessing

#Generating the Gaussian Naive Bayes model
from sklearn.naive_bayes import GaussianNB

# Assign features and encoding labels
weather=['Sunny','Sunny','Overcast','Rainy','Rainy','Rainy','Overcast','Sunny','Sunny',
'Rainy','Sunny','Overcast','Overcast','Rainy']
humidity=['High','High','High','Medium','Low','Low','Low','Medium','Low','Medium','Medium','Medium','High','Medium']

bat_first=['No','No','Yes','Yes','Yes','No','Yes','No','Yes','Yes','Yes','Yes','Yes','No']

# Creating LabelEncoder
le = preprocessing.LabelEncoder()

# Converting string labels into numbers.
weather_encoded=le.fit_transform(weather)
hum_encoded=le.fit_transform(humidity)
label=le.fit_transform(bat_first)
print(weather_encoded,hum_encoded,label)


#Combining weather and humidity in a single tuple as features
features=list(zip(weather_encoded,hum_encoded))


#Create a Gaussian Classifier
model = GaussianNB()
model.fit(features,label) #Train the model using training set.

print("Enter Weather and Humidtity conditions : ")
w,h=map(int, input().split())

#Predict Output
predicted= model.predict([[w,h]]) # ''' For Weather : 0:Overcast, 2:Sunny , 1:Rainy ''' For Humidity : 0:High, 2:Medium, 1:Low

print(predicted) # --> [1] that means yes, the player should bat first and [0] that means No, player should bowl first.
```

```
[2 2 0 1 1 1 0 2 2 1 2 0 0 1] [0 0 0 2 1 1 1 2 1 2 2 2 0 2] [0 0 1 1 1 0 1 0 1 1 1 1 1 0]
Enter Weather and Humidtity conditions :
20 35
[1]
```

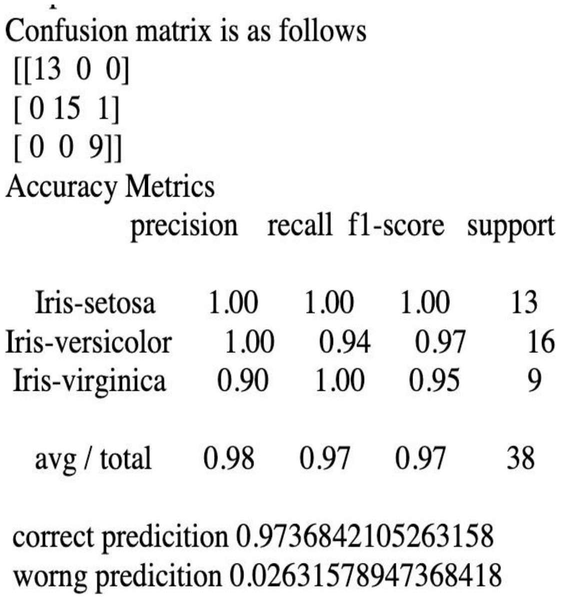
Week 9:

Build KNN Classification model for a given dataset.

Program:



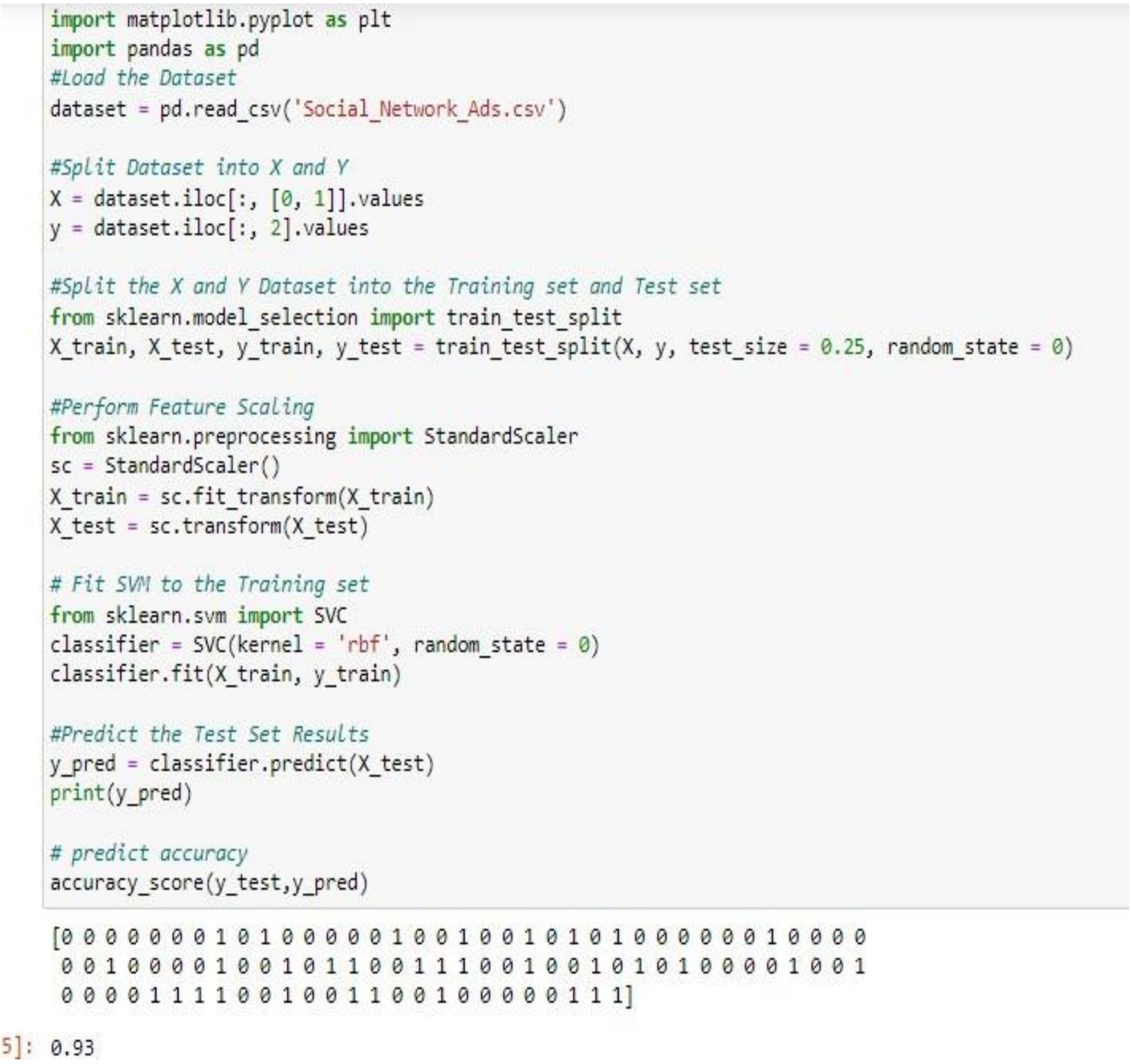
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
import pandas as pd

dataset=pd.read_csv("iris.csv")

X_train,X_test,y_train,y_test=train_test_split(X,y,random_state=0,test_size=0.25)

classifier=KNeighborsClassifier(n_neighbors=8,p=3,metric='euclidean')
classifier.fit(X_train,y_train)

 #predict the test resuts
y_pred=classifier.predict(X_test)

cm=confusion_matrix(y_test,y_pred)
print('Confusion matrix is as follows\n',cm)
print('Accuracy Metrics')
print(classification_report(y_test,y_pred))
print(" correct predicition",accuracy_score(y_test,y_pred))
print(" worng predicition",(1-accuracy_score(y_test,y_pred)))
```



```
Confusion matrix is as follows
 [[13  0  0]
 [ 0 15  1]
 [ 0  0  9]]
Accuracy Metrics
              precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        13
Iris-versicolor       1.00      0.94      0.97        16
 Iris-virginica       0.90      1.00      0.95         9

    avg / total       0.98      0.97      0.97        38

 correct predicition 0.9736842105263158
 worng predicition 0.02631578947368418
```

Week-10

Implement Support Vector Machine for a dataset.



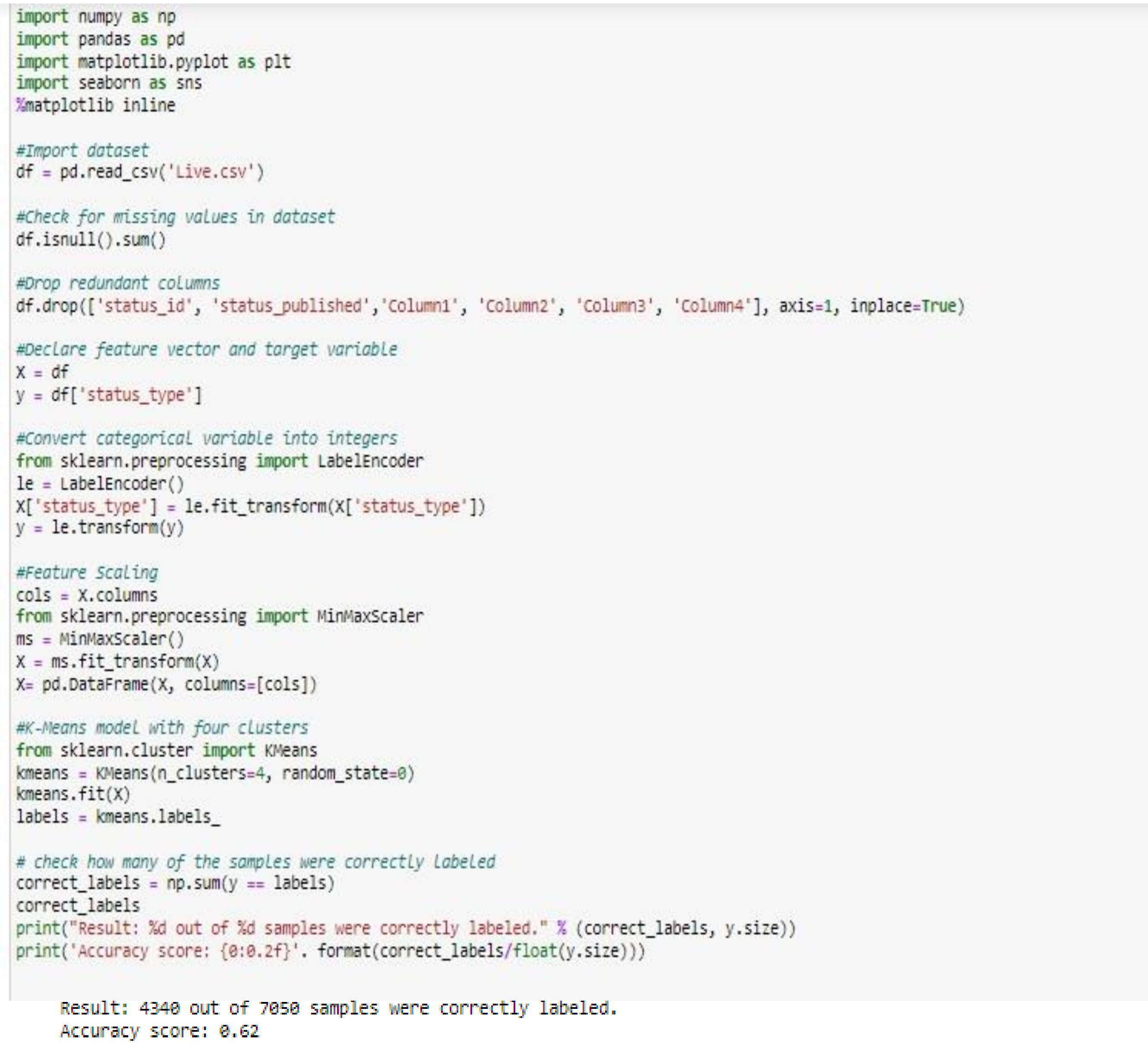
```
import matplotlib.pyplot as plt
import pandas as pd
#Load the Dataset
dataset = pd.read_csv('Social_Network_Ads.csv')

#Split Dataset into X and Y
X = dataset.iloc[:, [0, 1]].values
y = dataset.iloc[:, 2].values

#Split the X and Y Dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)

#Perform Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Fit SVM to the Training set
from sklearn.svm import SVC
classifier = SVC(kernel = 'rbf', random_state = 0)
classifier.fit(X_train, y_train)

#Predict the Test Set Results
y_pred = classifier.predict(X_test)
print(y_pred)

# predict accuracy
accuracy_score(y_test,y_pred)
```

```
[0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0
 0 0 1 0 0 0 0 1 0 0 1 0 1 1 0 0 1 1 1 0 0 1 0 0 1 0 1 0 1 0 0 0 0 1 0 0 1
 0 0 0 0 1 1 1 1 0 0 1 0 0 1 1 0 0 1 0 0 0 0 0 1 1 1]

5]: 0.93
```

Week-11

Write a python program to implement K-Means clustering Algorithm.

Program:



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

#Import dataset
df = pd.read_csv('Live.csv')

#Check for missing values in dataset
df.isnull().sum()

#Drop redundant columns
df.drop(['status_id', 'status_published','Column1', 'Column2', 'Column3', 'Column4'], axis=1, inplace=True)

#Declare feature vector and target variable
X = df
y = df['status_type']

#Convert categorical variable into integers
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
X['status_type'] = le.fit_transform(X['status_type'])
y = le.transform(y)

#Feature Scaling
cols = X.columns
from sklearn.preprocessing import MinMaxScaler
ms = MinMaxScaler()
X = ms.fit_transform(X)
X= pd.DataFrame(X, columns=[cols])

#K-Means model with four clusters
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=4, random_state=0)
kmeans.fit(X)
labels = kmeans.labels_

# check how many of the samples were correctly labeled
correct_labels = np.sum(y == labels)
correct_labels
print("Result: %d out of %d samples were correctly labeled." % (correct_labels, y.size))
print('Accuracy score: {0:0.2f}'. format(correct_labels/float(y.size)))
```

```
Result: 4340 out of 7050 samples were correctly labeled.
Accuracy score: 0.62
```