BMS INSTITUTE OF TECHNOLOGY AND MANAGEMENT

(An Autonomous Institution, Affiliated to VTU Belagavi)

DEPARTMENT OF MCA

MACHINE LEARNING LAB MANUAL

THIRD SEMESTER MCA

22MCA301
(Batch: 2022-24)

Course Co-ordinator: Dr. Aparna K
Associate Professor

Vision and Mission of Department of MCA

VISION

To develop quality professionals in Computer Applications who can provide sustainable solutions to the societal and industrial needs.

MISSION

Facilitate effective learning environment through quality education, state-of-the-art facilities, and orientation towards research and entrepreneurial skills.

Programme Educational Objectives of Department of MCA

PEO 1

Develop innovative IT applications to meet industrial and societal needs

PEO 2

Adapt themselves to changing IT requirements through life-long learning

PEO 3

Exhibit leadership skills and advance in their chosen career

Programme Outcomes of MCA

MCA Graduates will be able to

PO1: Apply knowledge of computing fundamentals, computing specialization, mathematics and domain knowledge to provide IT solutions

PO2: Identify, analyse and solve IT problems using fundamental principles of mathematics and computing sciences

PO3: Design, Develop and evaluate software solutions to meet societal and environmental concerns

PO4: Conduct investigations of complex problems using research based knowledge and methods to provide valid conclusions.

PO5: Select and apply appropriate techniques and modern tools for complex computing activities

PO6: Practice and follow professional ethics and cyber regulations

PO7: Involve in life-long learning for continual development as an IT professional.

PO8: Apply and demonstrate computing and management principles to manage projects in multidisciplinary environments by involving in different roles

PO9: Comprehend& write effective reports and make quality presentations.

PO10: Understand and assess the impact of IT solutions on socio-environmental issues

PO11: Work collaboratively as a member or leader in multidisciplinary teams.

PO12: Identify potential business opportunities and innovate to create value to the society and seize that opportunity

Course Outcomes

At the end of the course, students will be able to

CO1: Explore the Machine Learning concepts.
CO2: Build suitable Decision tree for a given data set.
CO3: Apply machine learning algorithms for the given problems.
CO4: Perform statistical and probabilistic analysis of machine learning techniques.
CO5: Implement machine learning algorithms for a given use case.

Table of Contents

1. LAB INSTRUCTIONS

1. **Duration of each laboratory session is 3 hours /week.**
2. **Maximum marks for Each Internal Assessment is 50.**
3. **Two internal tests will be conducted for the laboratory.**
4. **Award of I.A marks is based on the two internal tests and Continuous Evaluation.**
5. **There will be no SEE examination.**

2. HARDWARE AND SOFTWARE REQUIREMENTS

Hardware:

RAM : 512 MB

HDD : 40 GB

Software:

OS : WINDOWS XP

Packages : Python

Anaconda

Google Colab

3. INTRODUCTION TO DATA ANALYTICS USING PYTHON

Google Colab – It's an online IDE for Python created by Google. It is free of cost. To work with Google Colab, we need to have a Google Account. Stay logged in into your Google account.

To open Google Colab: https://colab.research. google.com
Click on "New Notebook" – to open a new page

Basic knowledge of Python is required.
.ipynb extension – interactive python note book

Benefits of Google Colab

- Not to worry about installation
- Can write the code from anywhere, any system/mobile
- All files will be saved in google drive – to know where it is saved, click on File – locate in drive. It will take you to file in the folder where it is saved.

You can also save it in the system.

Numpy

- It's a package / library in Python
- Stands for Numerical Python
- You can apply any numerical operations on data
- We generally deal with arrays

Import numpy file

import numpy as np - np is alias, any name can be used in place of numpy
To run the code, press shift + enter

Let us create 3 arrays – 1D, 2D and 3D array

```
arr1=np.array([1,2,3,4,5,6,7,8,9,0])
```
arr1 is a 1-d array

Bracket indicates the dimension

```
arr2 = np.array([[1,2,3], [4,5,6], [7,8,9],[0,1,1]])
```
arr2 is a 2-d array

```
arr3=np.array([[[1,2,3],[4,5,6], [7,8,9], [0,1,1]]])
```

arr3 is a 3-d array

To print the array, just give the array name, say, arr1

arr3

To know the dimension of the array
print(arr1.ndim)
print(arr2.ndim)

len function – counts the number of elements

```
len("aparna")
6
```

If we apply len function for array

```
print (len(arr1))
```

10

Arr1 has 10 elements in the list

```
print(len(arr2))
```

4

It counts the number of 1-d array present in arr2

```
print (len(arr3))
```

1

Gives the count of number of 2d-array in arr3

Shape of array – we can know the dimension

```
print (arr1.shape)
10
Means it is 1-d

print (arr2.shape)
(4,3)
2 elements means 2d, 4 rows and 3 cols

print(arr4.shape)
```

(1,4,3)

1 2d array of (4,3)

Datatype of array

```
print (arr1.dtype)
```

int 64

```
print (arr2.dtype)
```

int 64

```
arr4=np.array([[['1',2,3], [4,6,7],[7,8,9]], [[1,2,3],[4,5,6],[7,8,9
]],[[1,2,3], [4,5,6], [7,8,9]]])
```

In the above array, 1 is changed to '1'. Now if we try to print the data type of the array,

```
print(arr4.dtype)
```

<U21

This means we have passed combination of 2 data types.

If we print this array, we will see all data to be converted to string. Due to a single data '1' , all are converted to string. Because int can be converted to string. But string cannot be converted to int. we cannot have mixture of datatypes.

```
array([[['1', '2', '3'],
        ['4', '6', '7'],
        ['7', '8', '9']],

       [['1', '2', '3'],
        ['4', '5', '6'],
        ['7', '8', '9']],

       [['1', '2', '3'],
        ['4', '5', '6'],
        ['7', '8', '9']]], dtype='<U21')
```

Functions on arrays

```
np.max(arr2)
```

9 – will print the maximum number of that array

```
np.min(arr2)
```

0 – will print the minimum number of that array

```
np.argmax(arr2)
```

8 – will give the position of the maximum number

```
np.argmin(arr2)
```

9 – will give the position of the minimum number

```
np.sum(arr2)
```

47 – sum of all the elements of array

```
np.sum(arr2,axis=0)
```

array([12,16,19]) – column wise addition

```
np.sum(arr2,axis=1)
```

array([6,15,24,2]) – row wise addition

```
np.mean(arr2)
```
3.9166666 – average of all elements of array

```
round (np.mean(arr2))
```
4 - to round off

```
round(np.mean(arr2),2)
```
3.92 – to round upto 2 decimal places

Array slicing

Picking elements from the array

```
arr2[:]
```
displays all elements. Every row and every column elements

```
arr2[2:4]
```
```
array([[7, 8, 9],
       [0, 1, 1]])
```

Row2 to row 3 will be displayed. 4-1=3 it will take, row 2 and row 3

```
arr2[1:]
```
```
array([[4, 5, 6],
       [7, 8, 9],
       [0, 1, 1]])
```
Row 1 to last

```
arr2[:3]
```
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```
Row 0 to row 2

```
arr2[:,:]
```
all data will be displayed

```
arr2[1:3, 1:3]
```
```
array([[5, 6],
       [8, 9]])
```

Linspace function – generates equidistant samples between a range

np.linspace(start, stop, no. of samples, endpoint, retstep)

```
np.linspace(10,100,10)
  array([ 10.,  20.,  30.,  40.,  50.,  60.,  70.,  80.,  90., 100.])
```

endpoint = True is default - stop value mentioned will be the last value considered

endpoint = False – stop value will not be considered. So range will be different.

```
np.linspace(10,100,10, endpoint=False)
 array([10., 19., 28., 37., 46., 55., 64., 73., 82., 91.])
```

Here 100 is not included
retstep – by default it is false. If true, difference value will be displayed.

```
np.linspace(10,100,10,endpoint=False, retstep=True)
  (array([10., 19., 28., 37., 46., 55., 64., 73., 82., 91.]), 9.0)
```

Arange function

np.arange(start, stop, diff-value)

```
np.arange(10,100,3)

 array([10, 13, 16, 19, 22, 25, 28, 31, 34, 37, 40, 43, 46, 49, 52, 55, 58,
        61, 64, 67, 70, 73, 76, 79, 82, 85, 88, 91, 94, 97])
```

Last number will not be displayed

Decimal values also allowed

```
np.arange(10,101,4.5)
  array([ 10. ,  14.5,  19. ,  23.5,  28. ,  32.5,  37. ,  41.5,  46. ,
         50.5,  55. ,  59.5,  64. ,  68.5,  73. ,  77.5,  82. ,  86.5,
         91. ,  95.5, 100. ])
```

Random function

```
np.random.randint(10)
4 – Will generate a single random number from 0 to 9.
```

Each time it is executed, the value will be different.

Suppose we don't want the value to be changed, then

```
np.random.seed(1)
```

```
np.random.randint(10)
```

When we execute the above, the same value will be generated. If we change the seed value, then the result will change. Range of seed value is 0 to 2^32 -1

```
np.random.seed(10)
np.random.randint(12,100,5)
```

5 numbers between 12 to 99 will be generated. Same numbers will be generated until seed value is changed.

```
np.random.randint(12,100,(5,3))
 array([[20, 85, 12],
        [52, 48, 28],
        [23, 66, 74],
        [45, 84, 90],
        [61, 63, 66]])
```

Will generate 5x3 2d array

How to create standard matrix

```
np.ones((5,7))
 array([[1., 1., 1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1., 1., 1.]])


np.zeros((5,7))
 array([[1., 0., 0., 0., 0.],
        [0., 1., 0., 0., 0.],
        [0., 0., 1., 0., 0.],
        [0., 0., 0., 1., 0.],
        [0., 0., 0., 0., 1.]])


np.identity(5)
 array([[1., 0., 0., 0., 0.],
        [0., 1., 0., 0., 0.],
        [0., 0., 1., 0., 0.],
        [0., 0., 0., 1., 0.],
        [0., 0., 0., 0., 1.]])
```

We can convert the data type

```
np.ones((5,7)).astype("str")
```

```
array([['1.0', '1.0', '1.0', '1.0', '1.0', '1.0', '1.0'],
       ['1.0', '1.0', '1.0', '1.0', '1.0', '1.0', '1.0'],
       ['1.0', '1.0', '1.0', '1.0', '1.0', '1.0', '1.0'],
       ['1.0', '1.0', '1.0', '1.0', '1.0', '1.0', '1.0'],
       ['1.0', '1.0', '1.0', '1.0', '1.0', '1.0', '1.0']], dtype='<U32')
```

“str” can be replace with “float” or “int”

4. INDEX OF LAB PROGRAMS

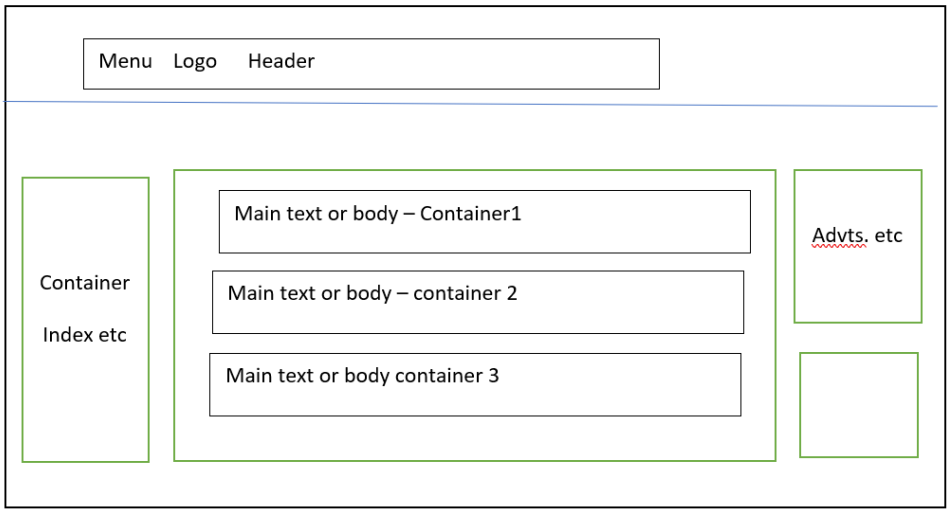
WEB SCRAPING

Scraping useful information from the web is called web scraping.

The data can be available in 2 forms – Private and Public
Private data are owned by the private companies. So it is illegal to scrap such data. Or we need to obtain permission to scrap such data.
Public data – to scrap such data, no permission is required. It is available in NGOs/Govt. reports etc. We can search in Google say, Public data for web scraping.

To do web scraping, we need libraries. 2 of them are very popular – **BeautifulSoup and Selenium.** Other libraries are fiddler and playwright.
BeautifulSoup is simple, elegant and powerful.

Lets first understand the webpage design. We divide the page into several containers when we design a webpage. The entire coding is done using HTML.



Web Scraping

```
# importing important libraries for web scraping
import requests
from bs4 import BeautifulSoup
```

```
import pandas as pd
url="https://books.toscrape.com/catalogue/page-
1.html"
```

```
response=requests.get(url)
```

```
response=response.content

soup = BeautifulSoup(response, 'html.parser')
soup
ol=soup.find('ol')
articles=ol.find_all('article',
class_='product_pod')
articles

books=[]
for article in articles:
  image=article.find('img')
  title=image.attrs['alt']
  star=article.find('p')
  star=star['class'][1]
  price=article.find('p',
class_='price_color').text
  #price=price[1:] ---  it gives string.. to
convert to number add float
  price=float(price[1:])
  #print(price)
  #print(star)
  #print(title)
  books.append([title, price, star])
print(books)

df=pd.DataFrame(books, columns=['Title', 'Price',
'Star Rating'])

df.to_csv('books.csv')
```

Multiple Pages

```
# importing important libraries for web scraping
import requests
from bs4 import BeautifulSoup
import pandas as pd


books=[]
```

```
for i in range(1,5):
  url=f"https://books.toscrape.com/catalogue/page-{i}.html"
  response=requests.get(url)
  response=response.content
  soup=BeautifulSoup(response, 'html.parser')
  ol=soup.find('ol')
  articles=ol.find_all('article', class_='product_pod')
  #books=[]
  for article in articles:
    image=article.find('img')
    title=image.attrs['alt']
    star=article.find('p')
    star=star['class'][1]
    price=article.find('p', class_='price_color').text

    #price=price[1:] ---  it gives string.. to convert to number add float

    price=float(price[1:])
    #print(price)
    #print(star)
    #print(title)
    books.append([title, price, star])


df=pd.DataFrame(books, columns=['Title', 'Price', 'Star Rating'])

df.to_csv('books.csv')
```

DATA PREPROCESSING

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

Upload the .csv file in Google Colab. After uploading, right click on the file and copy the path and paste it in the command below.

```
data = pd.read_csv("/content/movies.csv")
```

```
type(data)
 pandas.core.frame.DataFrame
```

The data type of the dataset is called DataFrame. Any data in a tabular form is a DataFrame. Each column of the DataFrame is known as series.

`data`
This command will display the entire dataset.

`data.head()`
This command will display the first 5 rows by default

`data.head(3)`
This command will display the first 3 rows

`data.tail()`
This command will display the last 5 rows by default

`data.sample()`
This command will display any one row randomly

`data.sample(3)`
This command will display any 3 rows randomly. The output will change everytime you run it. If the random rows should not change, we can use seed value. Seed here is called random_state.

`data.sample(random_state = 3)`
In the above case, every time same 3 rows will be displayed.

`data.info()`
This command will give complete information about the data set. It will also display the data type of every column, for ex, object means string.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9999 entries, 0 to 9998
Data columns (total 9 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   MOVIES    9999 non-null   object
 1   YEAR      9355 non-null   object
 2   GENRE     9919 non-null   object
 3   RATING    8179 non-null   float64
 4   ONE-LINE  9999 non-null   object
 5   STARS     9999 non-null   object
 6   VOTES     8179 non-null   object
 7   RunTime   7041 non-null   float64
 8   Gross     460 non-null    object
dtypes: float64(2), object(7)
memory usage: 703.2+ KB
```

Working with Missing values

```
data.isnull().sum()
```

```
MOVIES         0
YEAR         644
GENRE         80
RATING      1820
ONE-LINE       0
STARS          0
VOTES       1820
RunTime     2958
Gross       9539
dtype: int64
```

```
data_1=data.dropna(axis=0, how="all")
```

```
len(data)
```

9999

```
len(data_1)
```

9999

```
data_1=data.dropna(axis=0, how="any")
```

```
len(data_1)
```

460

```
data_1.isnull().sum()
```

```
MOVIES      0
YEAR        0
GENRE       0
RATING      0
ONE-LINE    0
STARS       0
VOTES       0
RunTime     0
Gross       0
dtype: int64
```

```
data_1=data.dropna(axis=0, how="all", subset=["GENRE"])
```

```
data_1.isnull().sum()
```

```
MOVIES        0
YEAR        603
GENRE         0
RATING     1751
ONE-LINE      0
STARS         0
VOTES      1751
RunTime    2887
Gross      9459
dtype: int64
```

```
data=data.drop(["Gross"],axis=1)
```

```
data.isnull().sum()
```

```
MOVIES        0
YEAR        644
GENRE        80
RATING     1820
ONE-LINE      0
STARS         0
VOTES      1820
RunTime    2958
dtype: int64
```

```
data['VOTES']
```

```
0         21,062
1         17,870
2       8,85,805
3       4,14,849
4            NaN
          ...
9994         NaN
9995         NaN
9996         NaN
9997         NaN
9998         NaN
Name: VOTES, Length: 9999, dtype: object
```

```
data['VOTES']=data['VOTES'].fillna("0")
```

```
data['VOTES']
```

```
0         21,062
1         17,870
2       8,85,805
3       4,14,849
4              0
          ...
9994           0
9995           0
9996           0
9997           0
9998           0
Name: VOTES, Length: 9999, dtype: object
```

```
data['RunTime']
```

```
 0       121.0
 1        25.0
 2        44.0
 3        23.0
 4         NaN
          ...
 9994      NaN
 9995      NaN
 9996      NaN
 9997      NaN
 9998      NaN
 Name: RunTime, Length: 9999, dtype: float64
```

```
meanRT=data['RunTime'].mean()
```

```
meanRT
```

```
68.68853855986366
```

```
meanRT=round(meanRT,1)
```

```
 68.7
```

```
data['RunTime']=data['RunTime'].fillna(meanRT)
```

```
data['RunTime']
```

```
0       121.0
1        25.0
2        44.0
3        23.0
4        68.7
        ...
9994     68.7
9995     68.7
9996     68.7
9997     68.7
9998     68.7
Name: RunTime, Length: 9999, dtype: float64
```

```
data.isnull().sum()
```

```
MOVIES         0
YEAR         644
GENRE         80
RATING      1820
ONE-LINE       0
STARS          0
VOTES          0
RunTime        0
dtype: int64
```

```
data['RATING']
```

```
0       6.1
1       5.0
2       8.2
3       9.2
4       NaN
       ...
9994    NaN
9995    NaN
9996    NaN
9997    NaN
9998    NaN
Name: RATING, Length: 9999, dtype: float64
```

```
meanRating=data['RATING'].mean()
```

```
meanRating
```

```
6.921176182907446
```

```
meanRating=round(meanRating,1)
```

```
meanRating
```

```
6.9
```

```
data['RATING']=data['RATING'].fillna(meanRating)
```

```
data['RATING']
```

```
0       6.1
1       5.0
2       8.2
3       9.2
4       6.9
       ...
9994    6.9
9995    6.9
9996    6.9
9997    6.9
9998    6.9
Name: RATING, Length: 9999, dtype: float64
```

```
data.isnull().sum()
```

```
MOVIES        0
YEAR        644
GENRE        80
RATING        0
ONE-LINE      0
STARS         0
VOTES         0
RunTime       0
dtype: int64
```

```
data['GENRE']
```

```
0           \nAction, Horror, Thriller
1       \nAnimation, Action, Adventure
2            \nDrama, Horror, Thriller
3       \nAnimation, Adventure, Comedy
4              \nAction, Crime, Horror
                          ...
9994       \nAdventure, Drama, Fantasy
9995    \nAnimation, Action, Adventure
9996              \nDocumentary, Sport
9997       \nAdventure, Drama, Fantasy
9998       \nAdventure, Drama, Fantasy
Name: GENRE, Length: 9999, dtype: object
```

```
data['GENRE']=data['GENRE'].fillna("Comedy")
```

```
data.isnull().sum()
```

```
MOVIES        0
YEAR        644
GENRE         0
RATING        0
ONE-LINE      0
STARS         0
VOTES         0
RunTime       0
dtype: int64
```

```
data['YEAR']
```

```
0             -2021
1          (2021- )
2       (2010–2022)
3          (2013- )
4             -2021
           ...
9994       (2021- )
9995       (2021- )
9996       (2022- )
9997       (2021- )
9998       (2021- )
Name: YEAR, Length: 9999, dtype: object
```

```
data['YEAR']=data['YEAR'].fillna(1999)
```

```
data.isnull().sum()
```

```
MOVIES      0
YEAR        0
GENRE       0
RATING      0
ONE-LINE    0
STARS       0
VOTES       0
RunTime     0
dtype: int64
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9999 entries, 0 to 9998
Data columns (total 8 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   MOVIES    9999 non-null   object
 1   YEAR      9999 non-null   object
 2   GENRE     9999 non-null   object
 3   RATING    9999 non-null   float64
 4   ONE-LINE  9999 non-null   object
 5   STARS     9999 non-null   object
 6   VOTES     9999 non-null   object
 7   RunTime   9999 non-null   float64
dtypes: float64(2), object(6)
memory usage: 625.1+ KB
```

LINEAR REGRESSION

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

Upload and read the .csv file: salary_data.csv

```
data = pd.read_csv("/content/salary_data.csv")
```

```
data.head()
```

	YearsExperience	Salary
0	1.1	39343
1	1.3	46205
2	1.5	37731
3	2.0	43525
4	2.2	39891

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 2 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   YearsExperience  30 non-null     float64
 1   Salary           30 non-null     int64  
dtypes: float64(1), int64(1)
memory usage: 608.0 bytes
```

```
x=data['YearsExperience']
```

```
x
```

```
0      1.1
1      1.3
2      1.5
3      2.0
4      2.2
5      2.9
6      3.0
7      3.2
8      3.2
9      3.7
10     3.9
11     4.0
12     4.0
13     4.1
14     4.5
15     4.9
16     5.1
17     5.3
18     5.9
19     6.0
20     6.8
21     7.1
22     7.9
23     8.2
24     8.7
25     9.0
26     9.5
27     9.6
28    10.3
29    10.5
Name: YearsExperience, dtype: float64
```

```
x=np.array(data.iloc[:,0])
```

```
x
```

```
array([ 1.1,  1.3,  1.5,  2. ,  2.2,  2.9,  3. ,  3.2,  3.2,  3.7,  3.9,
        4. ,  4. ,  4.1,  4.5,  4.9,  5.1,  5.3,  5.9,  6. ,  6.8,  7.1,
        7.9,  8.2,  8.7,  9. ,  9.5,  9.6, 10.3, 10.5])
```

```
x.shape
```

```
(30,)
```

```
x=np.array(data.iloc[:][["YearsExperience"]])
x
```

```
array([[ 1.1],
       [ 1.3],
       [ 1.5],
       [ 2. ],
       [ 2.2],
       [ 2.9],
       [ 3. ],
       [ 3.2],
       [ 3.2],
       [ 3.7],
       [ 3.9],
       [ 4. ],
       [ 4. ],
       [ 4.1],
       [ 4.5],
       [ 4.9],
       [ 5.1],
       [ 5.3],
       [ 5.9],
       [ 6. ],
       [ 6.8],
       [ 7.1],
       [ 7.9],
       [ 8.2],
       [ 8.7],
       [ 9. ],
       [ 9.5],
       [ 9.6],
       [10.3],
       [10.5]])
```

```
x.shape
```

```
(30, 1)
```

```
y
```

```
array([ 39343,  46205,  37731,  43525,  39891,  56642,  60150,  54445,
        64445,  57189,  63218,  55794,  56957,  57081,  61111,  67938,
        66029,  83088,  81363,  93940,  91738,  98273, 101302, 113812,
       109431, 105582, 116969, 112635, 122391, 121872])
```

```
y.shape
```

```
(30,)
```

```
from sklearn.model_selection import train_test_split

from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest=train_test_split(x,y,train_size=.80, ra
ndom_state=1)
```

```
xtrain
```

```
array([[ 9.5],
       [ 2. ],
       [ 8.7],
       [ 7.9],
       [ 8.2],
       [ 2.2],
       [ 1.5],
       [ 9. ],
       [ 3. ],
       [ 5.9],
       [ 4.1],
       [ 3.2],
       [ 9.6],
       [ 1.3],
       [ 5.1],
       [ 1.1],
       [ 4.9],
       [10.5],
       [10.3],
       [ 3.7],
       [ 3.2],
       [ 4. ],
       [ 4. ],
       [ 2.9]])
```

```
xtest
```

```
array([[5.3],
       [7.1],
       [3.9],
       [6. ],
       [4.5],
       [6.8]])
```

```
ytrain
```

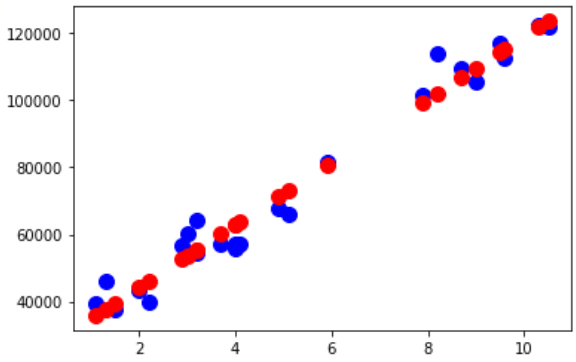
```
array([116969,  43525, 109431, 101302, 113812,  39891,  37731, 105582,
        60150,  81363,  57081,  54445, 112635,  46205,  66029,  39343,
        67938, 121872, 122391,  57189,  64445,  56957,  55794,  56642])
```

```
ytest
```

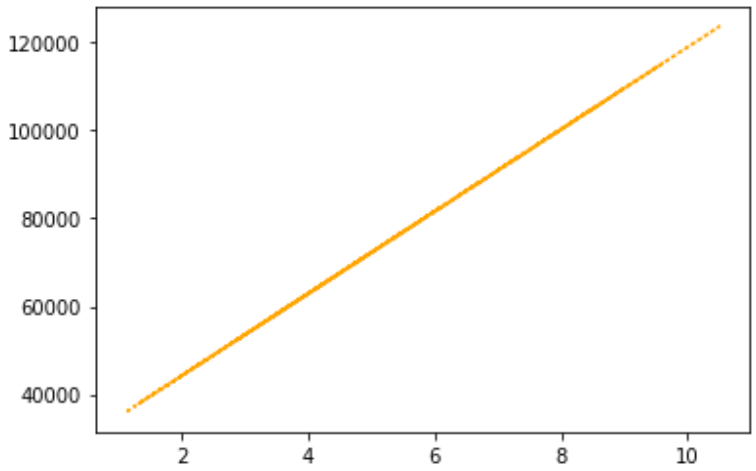
```
array([83088, 98273, 63218, 93940, 61111, 91738])
```

```
from sklearn.linear_model import LinearRegression

model=LinearRegression()
```

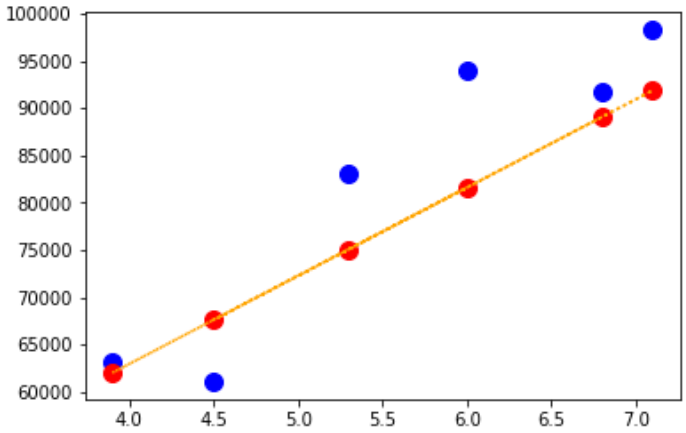
```
model.fit(xtrain, ytrain)
 LinearRegression()

ypred = model.predict(xtest)

ypred
 array([75074.50510972, 91873.8056381 , 62008.38247653, 81607.56642631,
        67608.14931932, 89073.92221671])

ytest
 array([83088, 98273, 63218, 93940, 61111, 91738])

from sklearn.metrics import r2_score
r2=r2_score(ytest, ypred)---- here r2 is a variable
r2
0.7616681465472094


plt.figure(figsize=(8,5))
plt.scatter(xtrain, ytrain, color='blue', s=100, label="Actual Point
")
plt.scatter(xtrain, model.predict(xtrain), color='red', s=100, label
="Predicted point")
plt.show()
```




```
plt.plot(xtrain, model.predict(xtrain), linestyle='dotted', color='o
range', label="line of regression")
```

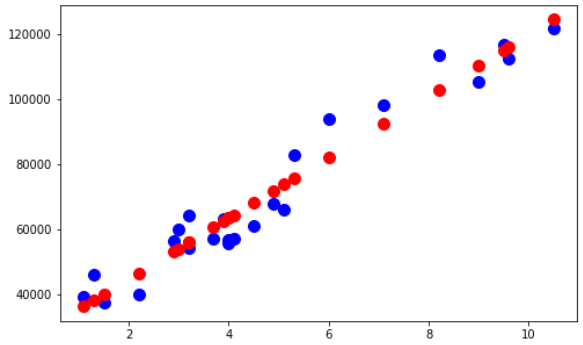



```
plt.figure(figsize=(8,5))
plt.scatter(xtest, ytest, color='blue', s=100, label="Actual Point")
plt.scatter(xtest, model.predict(xtest), color='red', s=100, label="
Predicted point")
plt.plot(xtest, model.predict(xtest), linestyle='dotted', color='ora
nge', label="Line of regression")
plt.show()
```

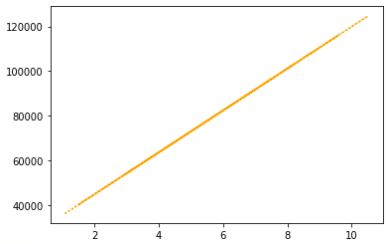



```
scores=[]
for i in range(5000):
  xtrain1, xtest1, ytrain1, ytest1 = train_test_split(x,y,train_size
=0.80, random_state=i)
  model1=LinearRegression()
  model1.fit(xtrain1, ytrain1)
  ypred1=model1.predict(xtest1)
  scores.append(r2_score(ytest1, ypred1))

scores
```

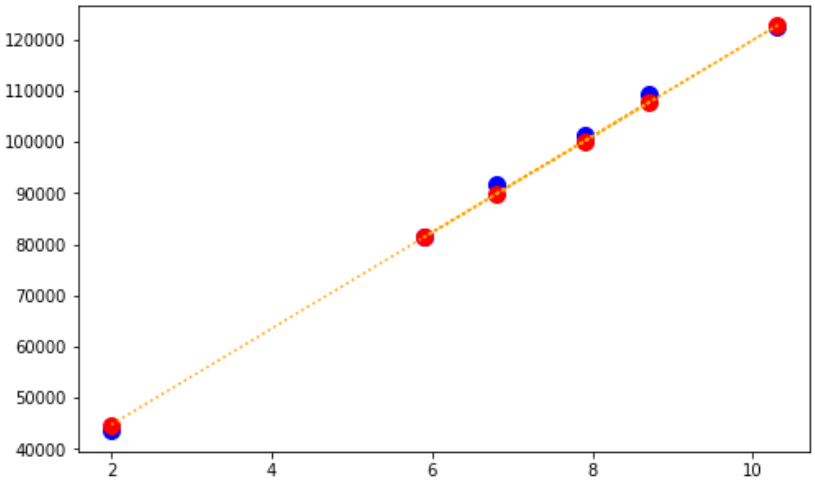
```
[0.988169515729126,
 0.7616681465472094,
 0.8886956733784563,
 0.9695039421049821,
 0.9504404484884267,
 0.9439628569611376,
 0.9368146227107087,
 0.8143022783109006,
 0.9631182154839476,
 0.9388416537799072,

np.max(scores)
0.9974925617006956
np.argmax(scores)
4697

xtrain, xtest, ytrain, ytest = train_test_split(x,y,train_size=.80,
random_state=4697)
from sklearn.linear_model import LinearRegression

model=LinearRegression()

model.fit(xtrain, ytrain)

LinearRegression()

ypred=model.predict(xtest)

from sklearn.metrics import r2_score

r2=r2_score(ytest, ypred)

r2

0.9974925617006956
plt.figure(figsize=(8,5))
plt.scatter(xtrain, ytrain, color='blue', s=100, label="Actual Point
")
plt.scatter(xtrain, model.predict(xtrain), color='red', s=100, label
="Predicted point")
plt.show()
```




```
plt.plot(xtrain, model.predict(xtrain), linestyle='dotted', color='o
range', label="Line of regression")
```




```
plt.figure(figsize=(8,5))
plt.scatter(xtest, ytest, color='blue', s=100, label="Actual Point")
plt.scatter(xtest, model.predict(xtest), color='red', s=100, label="
Predicted point")
plt.plot(xtest, model.predict(xtest), linestyle='dotted', color='ora
nge', label="Line of regression")
plt.show()
```



Multiple Linear Regression – More than one variable is an independent variable.
Try the above with headbrain.csv dataset. It has 4 columns – gender, age, head size and brain weight. We have to predict the weight of the brain. So brain weight is the target

variable. The features are Gender and Head Size. Age also can be a feature. But since the data is in the form of 1 and 2, we cannot consider. So we go with 2 independent variables.
To plot in the graph, we may have to plot for individual features. Plot separately. Single graph is not possible.

In the code, we have to give
x = np.array(data.iloc[:,[0,2]])
rest of the code is same.
Accuracy we get 0.8389 = 83.89%
Seed value = 4978

FIND-S ALGORITHM

```
import pandas as pd
import numpy as np

data=pd.read_csv("/content/enjoysport.csv")

data
```

	sky	airtemp	humidity	wind	water	forcast	enjoysport
0	sunny	warm	normal	strong	warm	same	yes
1	sunny	warm	high	strong	warm	same	yes
2	rainy	cold	high	strong	warm	change	no
3	sunny	warm	high	strong	cool	change	yes

```
concepts = np.array(data)[:,:-1]
concepts
```

```
array([['sunny', 'warm', 'normal', 'strong', 'warm', 'same'],
       ['sunny', 'warm', 'high', 'strong', 'warm', 'same'],
       ['rainy', 'cold', 'high', 'strong', 'warm', 'change'],
       ['sunny', 'warm', 'high', 'strong', 'cool', 'change']],
      dtype=object)
```

```
target = np.array(data)[:,-1]
target
```

```
array(['yes', 'yes', 'no', 'yes'], dtype=object)
```

```
def train(con, tar):
    for i, val in enumerate(tar):
        if val == 'yes':
            specific_h = con[i].copy()
            break

    for i, val in enumerate(con):
        if tar[i] == 'yes':
            for x in range(len(specific_h)):
                if val[x] != specific_h[x]:
                    specific_h[x] = '?'
                else:
                    pass
    return specific_h
print(train(concepts, target))
```

Output

```
['sunny' 'warm' '?' 'strong' '?' '?']
```