# DATA SCIENCE WITH PYTHON

**Student Name:N.HARI KISHORE REDDY**

**ID Number: N191092**

**Class:CSE-3**

**DEPARTMENT OF COMPUTER SCIENCE AND**

**ENGINEERING RGUKT-NUZ-AP**

**E2-SEMESTER-II, AY-2022-23**

**TABLE OF CONTENTS:**

# LAB-01

**AIM :** a) Python Basics: Your first program, Types Expressions and Variables String Operations

**Code:**

```
print("hello world")

color="green"
print(type(color))

a=3
print(a,type(a))

b=-3.5
print(b,type(b))

c=2+3j
print(type(c))

d,e,f=2,3,-4
print(f)
print(e)


print(d)

h=j=k="RAM"
print(h,j,k)

id1='How are you?'
print(id1[1:7])

x=0b11
print(type(x))

val=None
print(val)

  #python string
  id1="HARI"
  print(id1[1])

#negative indexing
print(id1[-3])
```

```
#id1[3]=q

#multiline strings
string="""THIS IS THE FIRST LAB"""
print(string)

#python string operation
id2=" is the roommate of    Jana"
print(id1+id2)

id3="Hari"
id4="Hari"
print(id3==id4)

id3="Hari"
id4="Hari1"
print(id3==id4)


#iterton
gr='welcome'
for letter in gr:
      print(letter)

gr='welcome' for
   letter in gr:
     print(gr)
 print(len(gr))

#membership
print("a" in gr)
print("a" not in gr)

print(gr.upper())
print(gr.lower())
print(gr.startswith("h"))

id='name' name='HARI'
print(f'my {id} is {name}')
```

**output:**

```
hello world
<class 'str'>
3 <class 'int'>
-3.5 <class 'float'>
<class 'complex'>
-4
3
2
RAM    RAM    RAM
ow are
<class 'int'>
None

 A
 A
 THIS IS THE FIRST LAB
 HARIis the roommate of Jana
  True
 False
W
E
L
C
O
M
E
 welcome
 welcome
 welcome
 welcome
 welcome
 welcome
 welcome
 7
 False
  True
 WELCOME
 Welcome
  False
my name is HARI
```

# LAB-02

**AIM :** Python Data Structures: Lists and Tuples Sets,and Dictionaries

**CODE:**

```
a=[2,'a','aba','aaa']
print(a)

num=(1,5,3)
print(num)
b={'a':3,'ba':456,'a':4}
print(b)
c={1,4,3,2,5,}
print(c)
d={2,'a','aba','aaa'}
print(d)
lan=["telugu","tamil","kannada"]
print(lan[2])
print(type(lan))
e={2,2,2,3}
print(e)
a=True
print(a)
b=False
print(b)

#list
a=[4,6,7]
print(a)

print(a[0])
print(a[-3])

print(a[0:2])

#append


a.append(2)
print(a)

#extend

b=[8,9,7]
a.extend(b)
```

```
print(a)

a[0]=0
print(a)

#del
del b[1]
print(b)
a.remove(0)

a. sort()
print(a)
a.reverse()
print(a)

a. pop(2)
print(a)
#checking
print(1 in a)
print(len(a))

#list comprehensionc=[]
  for x in range(1,6):
      c.append(x*x)
print(c)


  #tuple
  print("tuples")
  a=(3,4,5)
print(a)


b="hello",
print(type(b))

c=("hello")
print(type(c))

#tuple accessing
print(a[-1])
print(a[1])
print(a[0:2])

#tuple methods
d=(6,5,7,7,7,8,4,9,0)

print(d.count(7))
```

```
print(d.index(6))

#iteration

 for x in d:
     print(x)

print(7 in d)
```

**#sets**

```
a={3,5,6,7,8,9,4,5,6}
b={10,20,30,40}

print("set")
print(a)
print(type(a))


a.add(10)  print(a)
#min print(min(a))

#max
print(max(a))

#len
print(len(a))

#all print(all(a))

#any
print(any(a))

#enumerate
print(enumerate(a))

#sum
print(sum(a))

#sorted
print(sorted(a))


#union
print(a|b)
```

```
print(a.union(b))
#intersection


print(a&b)
print(a.intersection(b))


#symmetric difference
print(a^b)

#equal
print(a==b)
```

**#dictonary**

```
dic={1:"a",2:"b",3:"c",4:"d",5:"e"}
print(dic)
print(type(dic))

#adding
dic[6]="f"
print(dic)

#changing
dic[3]="C"
print(dic)

#accessing
print(dic[3])

#remove

del dic[6]
print(dic)


# sorted
sorted(c)
print(dic)

#membership

print(1 in dic) print(4
not in dic)
```

**output:**

[2, 'a', 'aba', 'aaa']
(1, 5, 3)
{'a': 4, 'ba': 456}
{1, 2, 3, 4, 5}
{2, 'aba', 'a', 'aaa'}
kannada
<class 'list'>
{2, 3}
True
False
[4, 6, 7]
4
4
[4, 6]
[4, 6, 7, 2]
[4, 6, 7, 2, 8, 9, 7]
[0, 6, 7, 2, 8, 9, 7]
[8, 7]
[2, 6, 7, 7, 8, 9]
[9, 8, 7, 7, 6, 2]
[9, 8, 7, 6, 2]
False
5
[1, 4, 9, 16, 25]
Tuples
(3, 4, 5)
<class 'tuple'>
<class 'str'>
5
4
(3, 4)
3
0
6
5
7
7
7
8
4
9
0
True
set
{3, 4, 5, 6, 7, 8, 9}

```
<class 'set'>
{3, 4, 5, 6, 7, 8, 9, 10}
3
10
8
True
True
<enumerate object at 0x000001803DC96E80>
52
[3, 4, 5, 6, 7, 8, 9, 10]
{3, 4, 5, 6, 7, 8, 9, 10, 40, 20, 30}
{3, 4, 5, 6, 7, 8, 9, 10, 40, 20, 30}
{10}
{10}
{3, 4, 5, 6, 7, 40, 8, 9, 20, 30}
False
{1: 'a', 2: 'b', 3: 'c', 4: 'd', 5: 'e'}
<class 'dict'>
{1: 'a', 2: 'b', 3: 'c', 4: 'd', 5: 'e', 6: 'f'}
{1: 'a', 2: 'b', 3: 'C', 4: 'd', 5: 'e', 6: 'f'}
C
{1: 'a', 2: 'b', 3: 'C', 4: 'd', 5: 'e'}
{1: 'a', 2: 'b', 3: 'C', 4: 'd', 5: 'e'}
True
False
```

# LAB- 03

**Python Programming Fundamentals:** Conditions and BranchingLoops, Functions,Objects and Classes

## Code:

**if-else:**

```
number=int(input("Enter a Number:"))
if number>10:
    print('Number is greater than 10')
else:
    print('Number is less than 10')
```

**Output:**

Enter a number:11
Number is greater than 10

**If-elif-else:**

```
num=int(input('Enter a Number:'))
if num>0:
    print('Positive Number')
elif num<0:
    print('Negative Number')
else:
    print('Positive Number')
print('This statement is always executed')
```

**Output:**

Enter a number:10
Positive Number

## nested-if:

```
num=int(input('Enter a Number:'))
if(num>=0):
    if num==0:
        print('Number is 0')
    else:
```

```
        print('Number is positive')
else:
    print('Number is Negative')
```

**output:**

Enter a Number:15

Number is positive

**short-hand-if:**

```
a=10;
b=20;
if a<b: print('This is if')
```

**Output:**

This is if

**shorthand-if-else:**

```
a=30;
b=20;
print('This is if') if a<b else print('this is else')'''
```

**Output:**

This is else

**for-loop:**

```
lang=['swift','c','python','c++']
for x in lang:
    print(x)
```

**range function:**

```
a=range(6)
 for x in a:
 print(x)

a=range(1,6)
for x in a:
    print(x)

a=range(2,22,2)
for x in a:
    print(x)
 for i in range(1,1001):
     for j in range (1,11):
         print(i*j,end=" ")
```

```
print()
```

**for loops with else:**

```
digits=[0,1,2]
for i in digits:
    print(i)
else:
    print("No items left.")
```

**while loop:**

```
i=1
n=5
while i<=n:
    print(i)
    i=i+1
```

**Python oops Concept**

**python inheritence:**

```
class Animal:
  def speak(self):
    print("Animal Speaking")
class Dog(Animal):
  def bark(self):
    print("dog barking")
class DogChild(Dog):
  def eat(self):
     print("Eating bread...")
d=DogChild()
d.speak()
d.bark()
d.eat()
```

**Output:**

Dog Barking

Animal Speaking

Eating Bread

**Method overriding:**

```
'class Animal:
  def speak(self):
    print("Speaking")
```

```
class Dog(Animal):
  def speak(self):
    print("Not Speaking")
class Cat(Dog):
  def speak(self):
   print("Is this a cat")
d=Cat()
d.speak()
```

**Output:**

Speaking Not
SpeakingIs this
a cat

**Data Abstraction:**

```
class Employee:
 ____count=0;
  def  ____init___(self):
    Employee.___count=Employee.____count+1

  def display(self):
    print("The number of Employees",Employee._count)
emp=Employee()
try:
 print(emp._count)
finally:
  emp.display()
```

**Output:**

Number of Employees:3

**Abstract Method:**

```
from abc import ABC, abstractmethod
class Car(ABC):
    def mileage(self):
        pass

class Tesla(Car):
    def mileage(self):
        print("The mileage is 30kmph")
class Suzuki(Car):
    def mileage(self):
```

```
        print("The mileage is 25kmph ")class
Duster(Car):
     def mileage(self):
          print("The mileage is 24kmph ")
class Renault(Car): def
    mileage(self):
            print("The mileage is 27kmph ")

# Driver codet=
Tesla ()
t.mileage()

r = Renault()
r.mileage()

s = Suzuki()
s.mileage()
d = Duster()
d.mileage()
```

**Output**

The mileage is 30kmph

The mileage is 25kmph

The mileage is 24kmph

The mileage is 27kmph

# LAB-04

**AIM:** Working with Data in Python: Reading files with open, Writing files with open,Loading data with Pandas, Working with and Saving data with Pandas

**CODE:**

```
import pandas as pd import
numpy as np
print(pd._version_)
b=[1,2,3,4]
c=pd.Series(b)

print©
b=['s','d']
c=pd.Series(b[-1])
print(c)
d=np.array(['a','b','c','d'])
s=pd.Series(d)
r=pd.DataFrame(d)
print(s)
print(r)
print(len(s))
s=pd.Series(d,index=[101,103,103,104])
j=pd.Series(d,index=["x","y","z","w"])
print(s)
print(j)

  dataset={'icecreams':['vanila','strawberry','badam','pista'], 'rating':[4.5,3.8,4.2,4.6]
}

ds=pd.DataFrame(dataset)print(ds)

ds=pd.Series(dataset)print(ds)
```

**Output:**

```
2.0.1
0    1
1    2
2    3
3    4
dtype: int64
0    d
```

```
dtype: object
  0    a
  1    b
  2    c
  3    d
  dtype: object0
  0  a
  1  b
  2  c
  3  d
  4
 101     a
 103     b
  103    c
  104    d
  dtype: object
  x    a
  y    b
  z    c
 w    d
  dtype: object
    icecreams  rating
  0      vanila     4.5
  1  strawberry     3.8
  2       badam     4.2
  3       pista     4.6
icecreams       [vanila, strawberry, badam, pista]
rating                      [4.5, 3.8, 4.2, 4.6]
dtype: object
```

**Attribute of series**

```
import pandas as pdimport
numpy as np

ds=np.array(['a','b','c','d'])
d=pd.Series(ds)
print(d)
d=pd.Series(ds ,index=[101,102,103,"e"])
print(d)
print(d[103])
ds1={'d1':100,'d2':200,'d3':300}
d=pd.Series(ds1)
print(d)
j=pd.Series(ds1,index=['d1','d2'])
print(j)
print(j.name)
print(j.values)
print(j.size)
print(d.shape)
print(d.ndim)
print(d.nbytes)
```

```
print(d.memory_usage)
print(j.empty)
j.name='raj'
print(j.name)
```

**output:**

```
  0    a
  1    b
  2    c
  3    d
dtype: object
  101    a
  102    b
  103    c
   e     d
dtype: objectc
d1       100
d2       200
d3       300
dtype: int64
d1       100
d2       200
dtype: int64
None
[100 200]
2
(3,)
1
24
<bound method Series.memory_usage of
d1       100
d2       200
d3       300
dtype: int64>
False
raj
```

**Multiplication of series :**

```
import pandas as pd
import numpy as np

ds1=np.array([1,1,2,3,4])
d1=pd.Series(ds1)
ds2=np.array([2,2,3,4,5])
d2=pd.Series(ds2)
a=d1.add(d2)
print(a)
b=d1.sub(d2)
print(b)
c=d1.mul(d2)
print(c)
```

```
d=d1.multiply(4)
print(d)
e=d1.div(d2)
print(e)
f=d2.mod(d1)
print(f)
g=d2.pow(3)
print(g)
h=d2.le(d1)
print(h)
i=d2.gt(d1)
print(i)
j=d2.equals(d1)
print(j)
```

**output:**

```
0      3
1      3
2      5
3      7
4      9
dtype: int32
0  -1
1   -1
2   -1
3   -1
4   -1
dtype: int32
0       2
1       2
2       6
3      12
4      20
dtype: int32
0       4
1       4
2       8
3      12
4      16
dtype: int32
0      0.500000
1      0.500000
2      0.666667
3      0.750000
4      0.800000
dtype: float64
0      0
1      0
2      1
3      1
4      1
dtype: int32
```

```
0      8
1      8
2     27
3     64
4    125
dtype: int32
 0    False
 1    False
 2    False
 3    False
 4    False
 dtype: bool
 0     True
 1     True
 2     True
 3     True
 4     True
 dtype: bool
 False
```

# LAB-05

**Aim:** Working with Numpy Arrays: Numpy 1d Arrays, Numpy 2d Arrays

**Code:**

```
import numpy as np
from numpy import random

a=np.array([1,2,3,4])
print(a)

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

c=np.array([[[1,2,3],[4,5,6],[7,0,9]]])
print(c)

d=np.array(32)
print(d)

print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)

e=np.array([1,2,3,4] ,ndmin=5)
print(e)

f=np.array([5,6],ndmin=3)
print(f)
print(f.ndim)

print(b[1,2])
#slicing
print(a[0:2])
print(a[2:])
print(a[:3])
print(a[-4:-2])
print(a[1:4:2])
print(a[1:4:3])
print(a[::1])
print(b[1,0:3:2])

g=np.array([1,2,3,4],dtype='S')
print(g)

print(b[1,0::3])
```

```
print(type(g))
print(g.dtype)

i=np.array([1.1,2.2,3.3,4.4])
print(i) j=i.astype('i')
print(j) print(i)


a=([1,3,4],[5,6,7])
b=np.asarray(a,order='f')
print(b)
  for i in np.nditer(b):
  print(i)

a=np.zeros((5,2 ),dtype=int)
print(a)
b=np.full([2,3],56 ,dtype=float)
print(b)

c=np.ones(([4,2]),dtype=int)
print(c)
x=random.randint(10000)
print(x)
  for i in range(1,5):
      x=random.randint(10)
  print(x)

d=np.eye(5,3 ,dtype=int, k=-1)
print(d)
a=np.eye(3,3, dtype=int)
print(a)
b=np.asarray(a,order='f')
for i in np.nditer(b):
      print(i)

#captcha
x=random.randint(10000)
print(x)
c=int(input('enter the capctha'))
while(c!=x):
print("invalid captcha")
c=int(input('enter'))



print("valid")

c=random.rand(3,2)
print(c)
d=random.ranf([3,2])
print(d)
```

**output:**

```
[1 2 3 4]
[[1 2 3 4 5]
 [6 7 8 9 0]]
[[[1 2 3]
  [4 5 6]
  [7 0 9]]]
32
1
2
3
0
[[[[[1 2 3 4]]]]]
[[[5 6]]]
3
8
[1 2]
[3 4]
[1 2 3]
[1 2]
[2 4]
[2]
[1 2 3 4]
[6 8]
[b'1' b'2' b'3' b'4']
[6 9]
<class 'numpy.ndarray'>
|S1
[1.1 2.2 3.3 4.4]
[1 2 3 4]
[1.1 2.2 3.3 4.4]
[[1 3 4]
 [5 6 7]]
1
5
3
6
4
7
[[0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]]
[[56. 56. 56.]
 [56. 56. 56.]]
[[1 1]
 [1 1]
 [1 1]
 [1 1]]
5425
7
2
```

```
4
3
[[0 0 0]
 [1 0 0]
 [0 1 0]
 [0 0 1]
 [0 0 0]]
[[1 0 0]
 [0 1 0]
 [0 0 1]]
1
0
0
0
1
0
0
0
1
7479
enter the capctha7479
valid
[[0.14702871 0.94097438]
 [0.80805663 0.52615084]
 [0.45495018 0.4452953 ]]
[[0.99567496 0.61726301]
 [0.44050543 0.35901677]
 [0.69665999 0.3356309 ]]
```

# LAB-06

**Aim:** Importing Datasets: Learning Objectives, Understanding the Domain, Understandingthe Dataset, Python package for data science, Importing and Exporting Data in Python, BasicInsights from DatasetsCleaning and Preparing the Data: Identify and Handle Missing Values, DataFormatting, Data Normalization Sets, Binning, Indicator variables

## Code:

```
import pandas as pd
import numpy as np

technologies =
    { 'Courses':["Spark","PySpark","Hadoop","Python"],
    'Fee' :[20000,25000,26000,22000],
    'Duration':['30day','40days',np.nan, None],
    'Discount':[1000,2300,1500,1200]
              }

indexes=['r1','r2','r3','r4']
df = pd.DataFrame(technologies,index=indexes)print(df)


# Drop rows by Index Label
df = pd.DataFrame(technologies,index=indexes)df1 =
df.drop(['r1','r2'])
print('\n\n',df1)


# Delete Rows by Index numbers
df = pd.DataFrame(technologies,index=indexes)
df1=df.drop(df.index[[1,3]])
df.drop(df.index[-1],inplace=True)

for col in df.columns:if
    'Fee' in col:
        del df[col]
print('\n\n',df)


import   pandas   as   pd
import   numpy   as   np
technologies = {
```

```
    'Courses':["Spark","PySpark","Hadoop","Python","pandas",np.nan],
    'Fee' :[20000,25000,26000,23093,24000,np.nan],
    'Duration':['30day','40days','35days','45days',np.nan,np.nan],
    'Discount':[1000,np.nan,1200,2500,pd.NaT,np.nan],
    'one':[np.nan,np.nan,np.nan,np.nan,np.nan,np.nan]
            }
index_labels=['r1','r2','r3','r4','r5','re']
df = pd.DataFrame(technologies,index=index_labels)print(df)

df.columns.values[-1]='one'
print(df) print(df.dropna(),'\n\n')
print(df.dropna(axis=1),'\n\n')

print(df.dropna(how='all'),'\n\n') print(df.dropna(how='all',axis=1),'\n\n')

print(df.dropna(thresh=1))
print(df.dropna(thresh=2,axis=1))

# Drop rows that has NaN values on selected columns
df2=df.dropna(subset=['Courses','Duration']) print('\n\n',df2)
```

**output:**

```
    Courses     Fee Duration  Discount
r1      Spark  20000     30day       1000
r2    PySpark  25000    40days        2300
r3     Hadoop  26000       NaN        1500
r4     Python  22000      None       1200


    Courses     Fee Duration  Discountr3
Hadoop  26000            NaN       1500
r4  Python  22000        None       1200
```

```
     Courses Duration      Discount
r1     Spark     30day        1000
r2  PySpark     40days        2300
r3    Hadoop       NaN       1500
```

```
Empty DataFrame
Columns: [] Index:
[r1, r2, r3]
```

```
Empty DataFrame
Columns: [] Index:
[r1, r2, r3]
```

```
    Courses        Fee Duration  Discount  one r1
      Spark  20000.0        30day      1000  NaN r2
PySpark  25000.0            40days       NaN NaNr3
      Hadoop  26000.0      35days      1200  NaN
r4   Python  23093.0     45days     2500   NaN r5
pandas  24000.0       NaN       NaT   NaN re   NaN
NaN  NaN  NaN  NaN
    Courses        Fee Duration Discount oner1
       Spark  20000.0       30day      1000  NaN
r2      PySpark  25000.0     40days         NaN NaN
r3       Hadoop  26000.0     35days       1200  NaN
r4   Python  23093.0      45days      2500  NaN
r5   pandas  24000.0         NaN       NaT  NaN
```

```
     Courses        Fee Duration Discount  one r1
      Spark  20000.0        30day      1000  NaN r2
PySpark  25000.0            40days       NaN NaNr3
      Hadoop  26000.0      35days      1200  NaN
r4   Python  23093.0      45days      2500  NaN
```

# LAB-07

**AIM: Model Development (Simple and Multiple Linear Regression, Model evaluation esing Visualization, Polynomial Regression and Pipelines, R-squared and MSE for In-Sample Evaluation, Prediction and Decision Making)**

## Code: Linear Regression

```
import pandas as pd
df=pd.read_csv("data.csv")
df.head()
data_=df.loc[:,['Weight','CO2']]
print(data_.head(10))
#showing the data in matplotlib
#to use we need to first install matplotlib
import matplotlib.pyplot as plt
df.plot(x='Weight',y='CO2',style='o')
plt.xlabel('Weight')
plt.ylabel('CO2')
plt.show()
#dividing the variables into dependent and independent
X=pd.DataFrame(df['Weight'])
y=pd.DataFrame(df['CO2'])
#Split the data into train and test sets
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=1)
#knowing the shapes of the test and train
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)

#train the algorithm
from sklearn.linear_model import LinearRegression
regressor=LinearRegression()
regressor.fit(X_train,y_train)
```

```
#retriving the intercept
print(regressor.intercept_)
#retriving the slope
print(regressor.coef_)
#predecting the test results
y_pred = regressor.predict(X_test)
y_test
print(y_pred)
print(y_test)
#evaluting the algorithm
from sklearn import metrics
import numpy as np
print('Mean Absolute Error:',metrics.mean_absolute_error(y_test,y_pred))
print('Mean Squared Error:',metrics.mean_squared_error(y_test,y_pred))
print('Root Mean Squared Error:',np.sqrt(metrics.mean_squared_error(y_test,y_pred)))

#plot for the train set
plt.scatter(X_train, y_train, color='red') # plotting the observation line

plt.plot(X_train, regressor.predict(X_train), color='blue') # plotting the regression line

plt.title("Weight vs CO2 (Training set)") # stating the title of the graph

plt.xlabel("Weight") # adding the name of x-axis
plt.ylabel("CO2") # adding the name of y-axis
plt.show() # specifies end of graph


#plot for the test set

plt.scatter(X_test, y_test, color='red')
plt.plot(X_train, regressor.predict(X_train), color='blue') # plotting the regression line
plt.title("Weight vs CO2 (Testing set)")
```

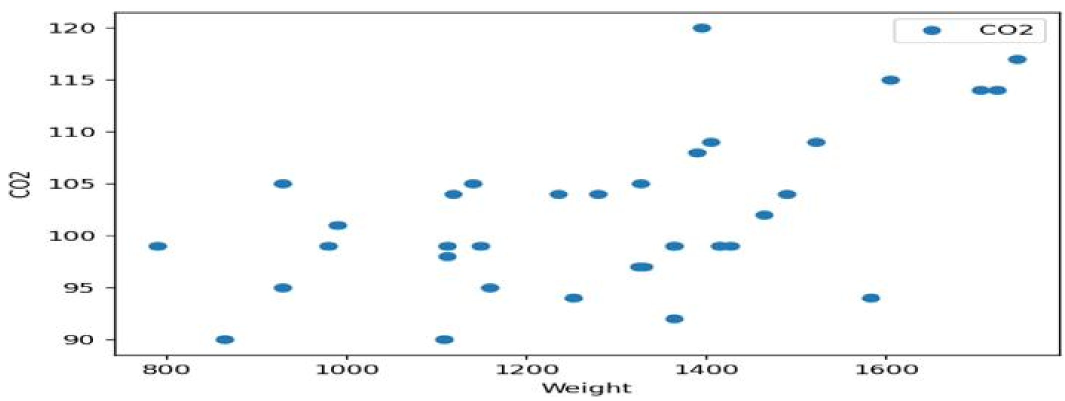
```
plt.xlabel("Weight")
plt.ylabel("CO2")
plt.show()

#importing pandas
import pandas as pd
#importing data set
df=pd.read_csv("data.csv")
#making list of independent variales as x and dependent variable as y
X = df[['Weight', 'Volume']]
y = df['CO2']
#to import this sklearn pip install -U scikit-learn
from sklearn import linear_model
regr = linear_model.LinearRegression()
regr.fit(X, y)
predictedCO2 = regr.predict([[2300, 1300]])
print(predictedCO2)
print(regr.coef_)
predictedCO2 = regr.predict([[3300, 1300]])
print(predictedCO2)
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

df=pd.read_csv('data.csv')


x=df['Weight']
y=df['CO2']

mymodel=np.poly1d(np.polyfit(x, y, 3))

myline=np.linspace(1,30,100)
```

```
plt.scatter(x,y)
plt.plot(myline,mymodel(myline))
plt.show()
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_diabetes
from sklearn.linear_model import Ridge
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Load the dataset
diabetes = load_diabetes()

# Separate the features and target variable
X = diabetes.data
y = diabetes.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and fit the Ridge regression model
ridge = Ridge(alpha=1.0)  # You can adjust the regularization strength with the 'alpha' parameter
ridge.fit(X_train, y_train)

# Predict on the test set
y_pred = ridge.predict(X_test)

# Calculate coefficients and intercept
coefficients = ridge.coef_
intercept = ridge.intercept_

# Print the coefficients and intercept
print("Coefficients:", coefficients)
print("Intercept:", intercept)
```

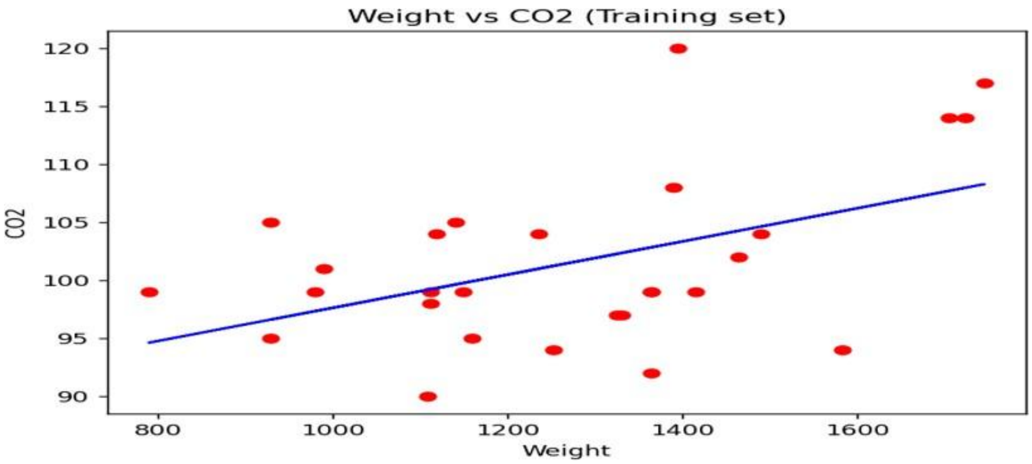
```
# Calculate mean squared error
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)

# Plot the predicted values against the true values
plt.scatter(y_test, y_pred)
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=2)
plt.xlabel('True Values')
plt.ylabel('Predicted Values')
plt.title('Ridge Regression - True vs Predicted')
plt.show()
```

## Output:

```
   Weight  CO2
0     790   99
1    1160   95
2     929   95
3     865   90
4    1140  105
5     929  105
6    1109   90
7    1365   92
8    1112   98
9    1150   99
```

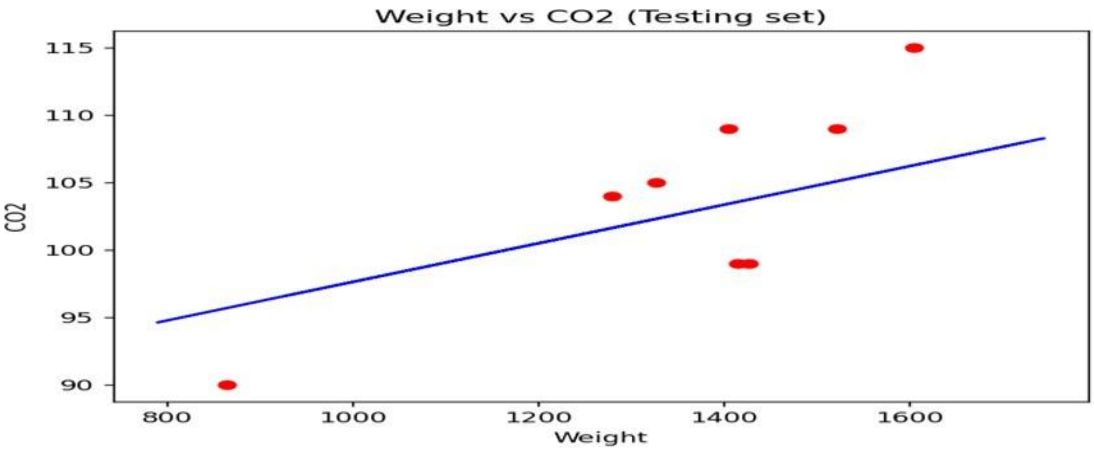


```
(28, 1)
(8, 1)
(28, 1)
(8, 1)
[83.33027919]
[[0.01428958]]
[[106.26505488]
 [103.40713891]
 [105.09330933]
 [ 95.69076578]
 [102.30684126]
 [101.62094142]
 [103.73579924]
 [103.5500347 ]]
   CO2
30  115
34  109
28  109
3    90
19  105
17  104
21   99
23   99
Mean Absolute Error: 4.785414241420883
```




Mean Squared Error: 26.40875532851579

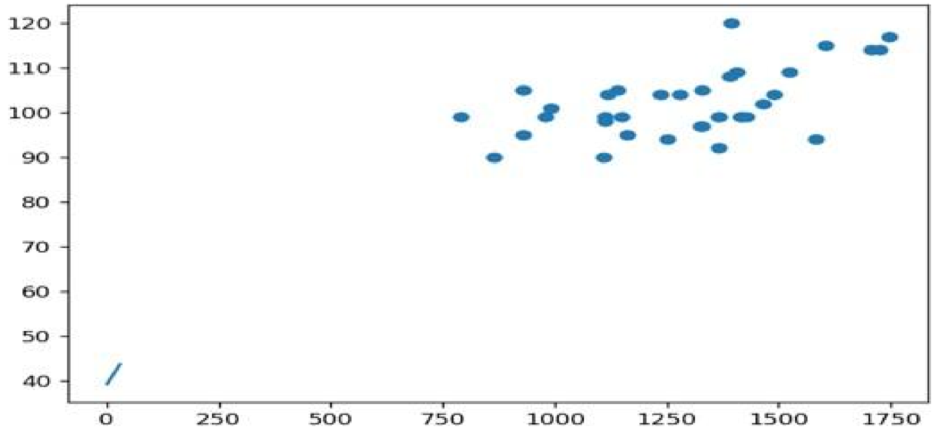
Root Mean Squared Error: 5.138944962588702




[107.2087328]

[0.00755095 0.00780526]

[114.75968007]

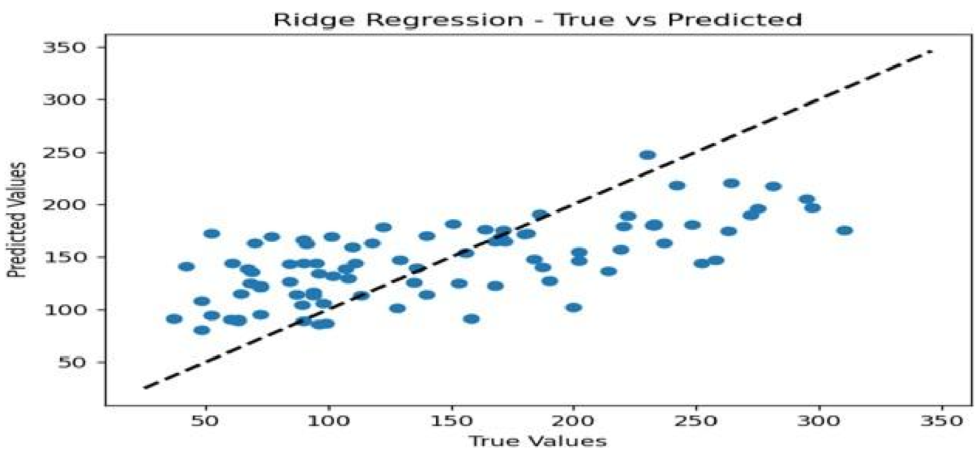



Coefficients: [ 45.36737726 -76.66608563 291.33883165 198.99581745 -0.53030959

-28.57704987 -144.51190505 119.26006559 230.22160832 112.14983004]

Intercept: 152.241675211113

Mean Squared Error: 3077.41593882723



# LAB-08

## AIM: Model Evaluation

**(Over-fitting, Under-fitting and Model Selection, Ridge Regression, Grid Search, Model Refinement)**

**Code:**

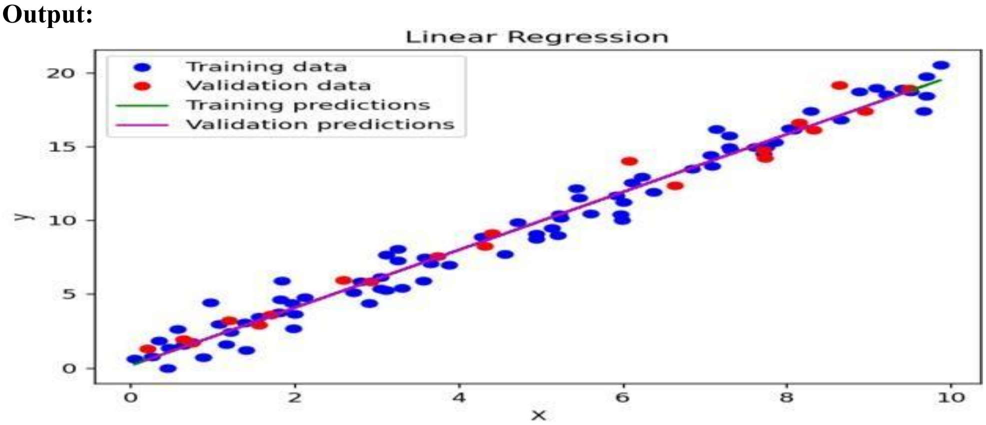
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
# Generate some sample data
np.random.seed(42)
X = np.random.rand(100, 1) * 10
y = 2 * X + np.random.randn(100, 1)
# Split the data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)
# Fit a linear regression model on the training data
model = LinearRegression()
model.fit(X_train, y_train)
# Make predictions on training and validation data
y_train_pred  =  model.predict(X_train)
y_val_pred = model.predict(X_val)
# Calculate mean squared errors
train_error = mean_squared_error(y_train, y_train_pred)
val_error = mean_squared_error(y_val, y_val_pred)
# Plot the learning curves
plt.plot(X_train, y_train, 'bo', label='Training data')
plt.plot(X_val, y_val, 'ro', label='Validation data')
plt.plot(X_train, y_train_pred, 'g-', label='Training predictions')
plt.plot(X_val, y_val_pred, 'm-', label='Validation predictions')
plt.legend()
plt.xlabel('X')
plt.ylabel('y')
plt.title('Linear Regression')
plt.show()
print('Training MSE:', train_error)
```

```
print('Validation MSE:', val_error)
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
# Generate some sample data
np.random.seed(42)
X = np.random.rand(100, 1) * 10
y = 2 * X + np.random.randn(100, 1)
# Split the data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)
# Fit a linear regression model on a subset of the training data
model = LinearRegression()
model.fit(X_train[:10], y_train[:10])
# Make predictions on training and validation data
y_train_pred = model.predict(X_train)
y_val_pred = model.predict(X_val)
# Calculate mean squared errors
train_error = mean_squared_error(y_train, y_train_pred)
val_error = mean_squared_error(y_val, y_val_pred)
# Plot the learning curves
plt.plot(X_train, y_train, 'bo', label='Training data')
plt.plot(X_val, y_val, 'ro', label='Validation data')
plt.plot(X_train, y_train_pred, 'g-', label='Training predictions')
plt.plot(X_val, y_val_pred, 'm-', label='Validation predictions')
plt.legend()
plt.xlabel('X')
plt.ylabel('y')
plt.title('Linear Regression')
plt.show()
print('Training MSE:', train_error)
print('Validation MSE:', val_error)
import numpy as np
```

```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
# Generate some sample data
np.random.seed(42)
X = np.random.rand(100, 1) * 10
y = 2 * X + np.random.randn(100, 1)
# Define the models
models = [
    ('Linear Regression', LinearRegression()),
    ('Decision Tree', DecisionTreeRegressor()),
    ('Random Forest', RandomForestRegressor())
]
# Evaluate each model using cross-validation
for model_name, model in models:
    scores = cross_val_score(model, X, y, scoring='neg_mean_squared_error', cv=5)
    rmse_scores = np.sqrt(-scores)
    avg_rmse = np.mean(rmse_scores)
    print(model_name)
    print('RMSE scores:', rmse_scores)
    print('Average RMSE:', avg_rmse)
    print('---')
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVC
iris = load_iris()    # Load the iris dataset
X, y = iris.data, iris.target
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Define the parameter grid for grid search
param_grid = {
```

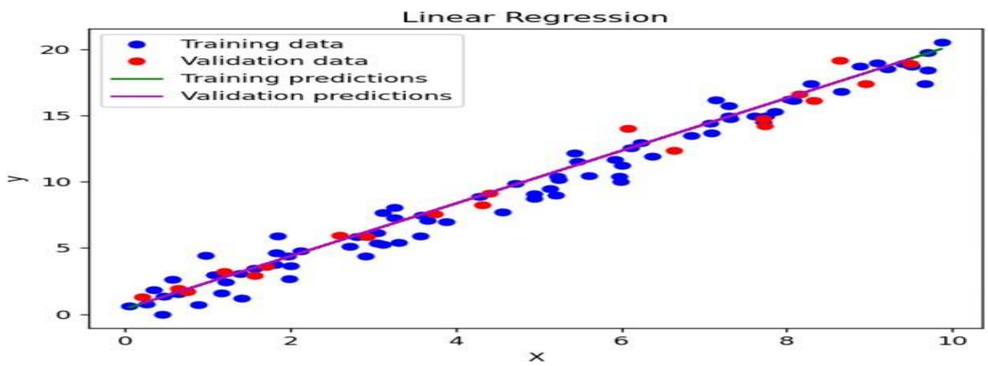
```
    'C': [0.1, 1, 10, 100],           # Regularization parameter
    'kernel': ['linear', 'rbf', 'poly'] # Kernel function
}
svm = SVC()# Create a SVM classifier
# Create the GridSearchCV object
grid_search = GridSearchCV(estimator=svm, param_grid=param_grid, cv=5, scoring='accuracy')
# Perform grid search on the training data
grid_search.fit(X_train, y_train)
# Print the best hyperparameters and the corresponding accuracy
print("Best hyperparameters:", grid_search.best_params_)
print("Best accuracy:", grid_search.best_score_)
# Evaluate the model on the test data using the best hyperparameters
best_model = grid_search.best_estimator_
test_accuracy = best_model.score(X_test, y_test)
print("Test accuracy with best hyperparameters:", test_accuracy)
```

**Output:**




Training MSE: 0.8476788564209707

Validation MSE: 0.653699513717003




Training MSE: 1.009101708602413

Validation MSE: 0.6575637436381261

Linear Regression

RMSE scores: [0.77953381 0.89086877 1.0326569  0.90782046 0.98769618]

Average RMSE: 0.919715223669806

---

Decision Tree

RMSE scores: [1.08188241 1.45774641 1.36210368 0.91080956 1.36022072]

Average RMSE: 1.2345525556529042

---

Random Forest

RMSE scores: [0.95060216 1.19257089 1.16866416 0.9016125  1.19856087]

Average RMSE: 1.0824021156248491

---

Best hyperparameters: {'C': 1, 'kernel': 'linear'}

Best accuracy: 0.9583333333333334

Test accuracy with best hyperparameters: 1.0

# LAB-09

**AIM: Introduction to Visualization Tools (Introduction to Data Visualization, Introductionto Matplotlib, Basic Plotting with Matplotlib, Dataset on Immigration to Canada, Line Plots)**

## Code:

#scatterplot d1=df.head(50) x_scatter=d1['yearsExperience']

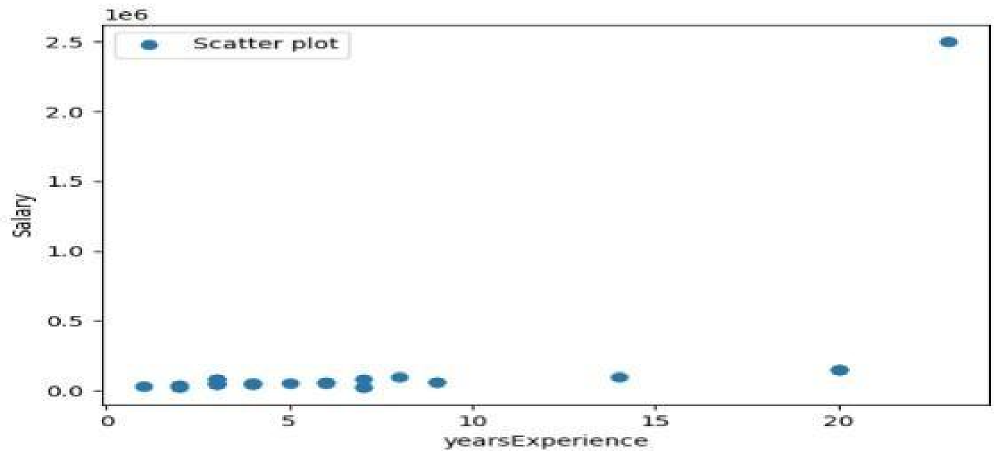
y_scatter=d1['salary'] plt.xlabel('yearsExperience')

plt.ylabel('Salary')

plt.scatter(x_scatter,y_scatter,label="Scatter plot")

plt.legend() plt.show()

**Output:**



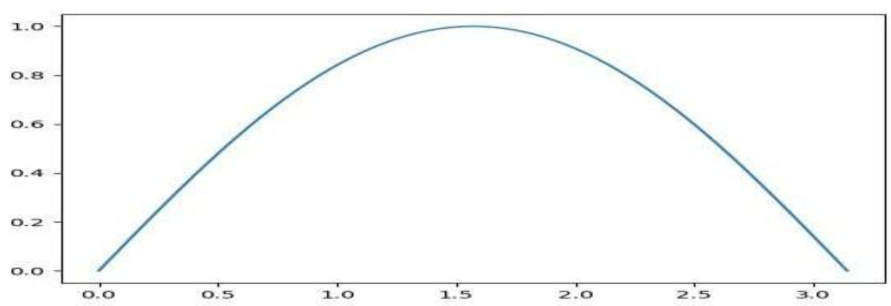

import matplotlib.pyplot as plt

import numpy as np

x=np.linspace(0,1*np.pi,10000)

y=np.sin(x) fig, ax=plt.subplots()

ax.plot(x,y)  plt.show() **Output:**



# LAB-10

## AIM: Visualization Tools (Area Plots, Histograms, Bar Charts)

**Code**: import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

df=pd.read_csv("abss.csv") df

import pandas as pd import

matplotlib import matplotlib.pyplot

as plt import numpy as np

df=pd.read_csv("abss.csv")

plt.plot(df.age,df.price,) plt.show()

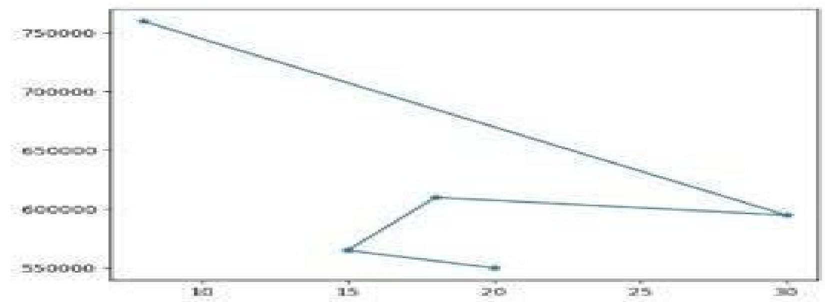
plt.plot(df.age,df.price,marker="*") plt.show()

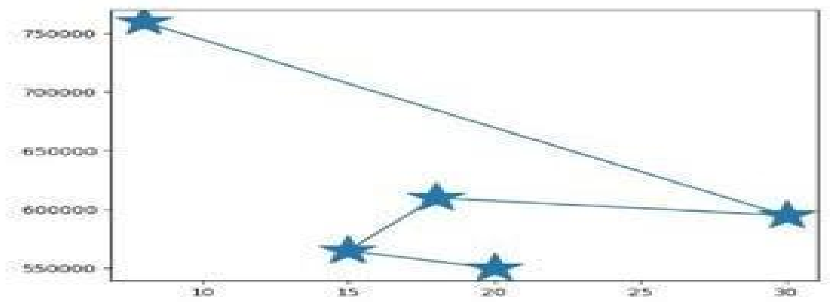
plt.plot(df.age,df.price,marker="*",ms="30")
plt.show()

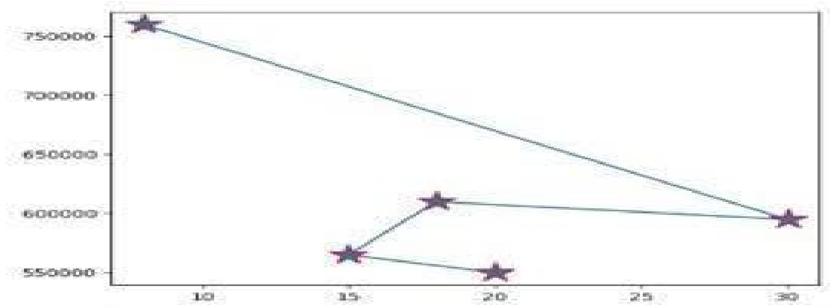
plt.plot(df.age,df.price,marker="*",ms="20",mec="red") plt.show()

plt.plot(df.age,df.price,marker="*",ms="20",mfc="green") plt.show()

**Output:**

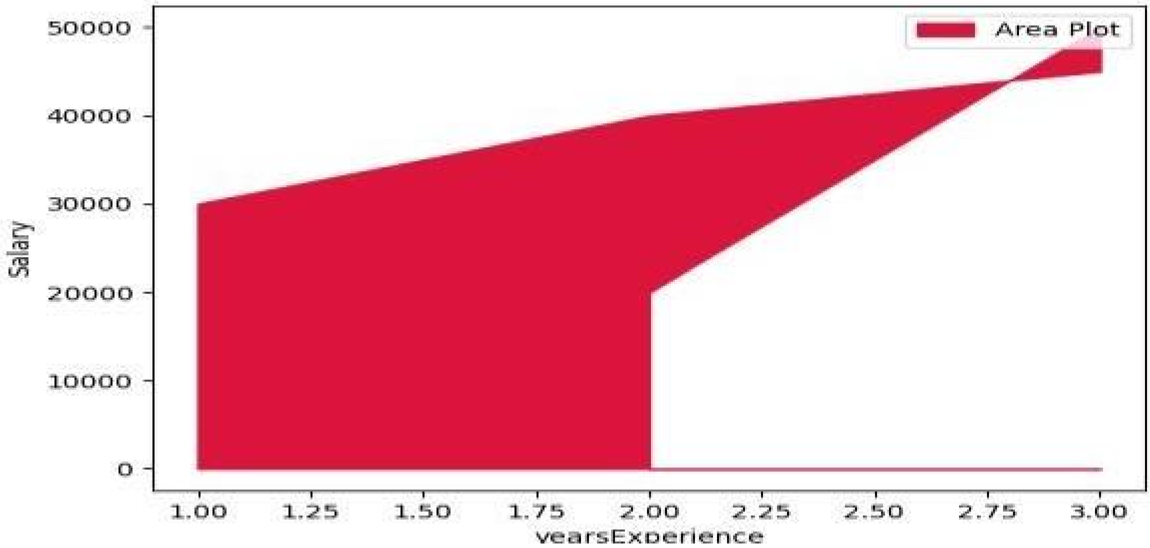


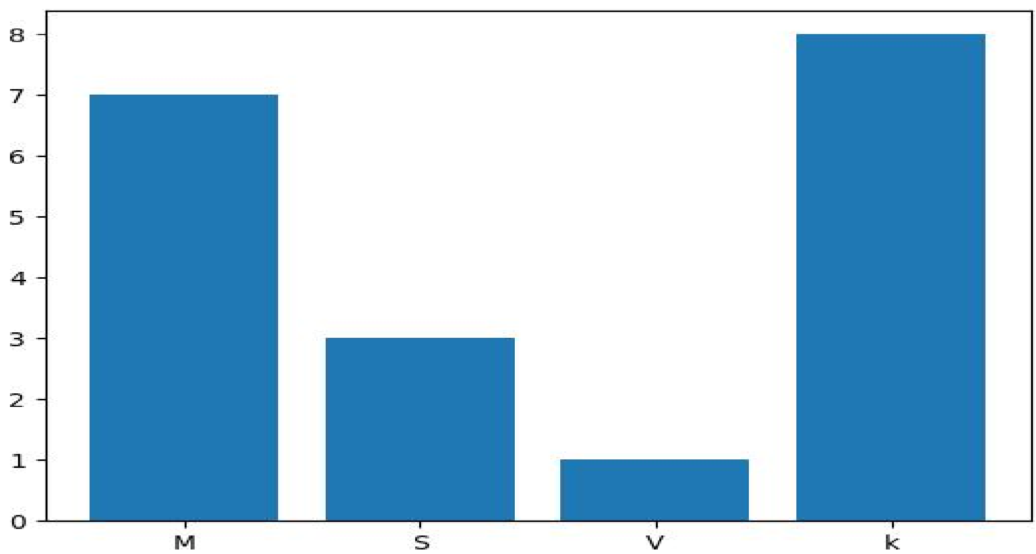
**#Area plots**

```
d2=df.head()
print(d2['yearsExperience'])
x_area=d2['yearsExperience']
y_area=d2['salary']
plt.xlabel('yearsExperience')
plt.ylabel('Salary')
plt.fill_between(x_area,y_area,label="Area
Plot",color="crimson") plt.legend() plt.show()
```



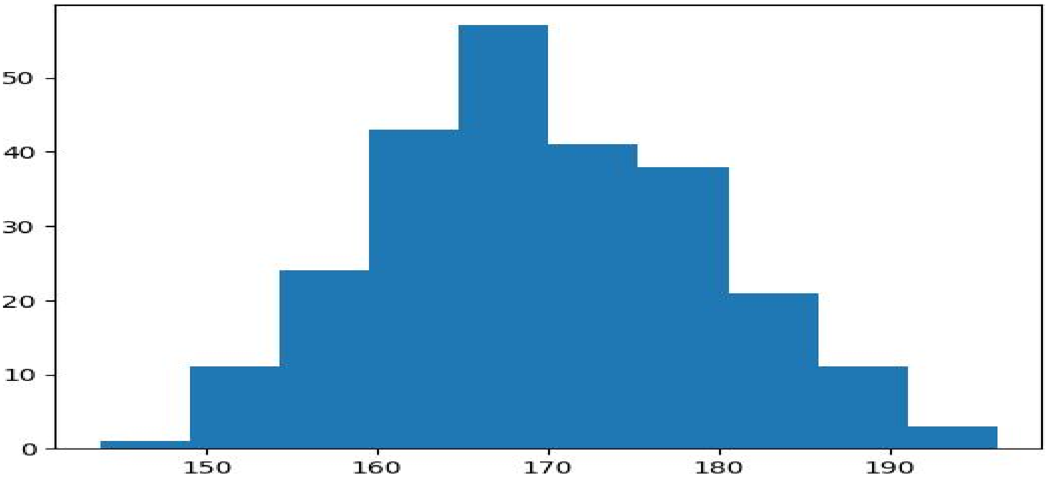

**#Bars Plots**

```
import matplotlib.pyplot
as pltimport numpy as np
x = np.array(["M", "S", "V",
"k"])y = np.array([7, 3, 1,
8]) plt.bar(x,y)
plt.show()
```



#**Histogram:**

```
import matplotlib.pyplot
as pltimport numpy as np
x = np.random.normal(170,
10, 250)plt.hist(x)
plt.show()
```

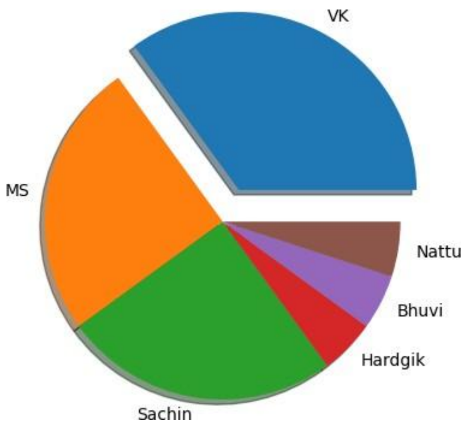


# LAB-11

**AIM: Specialized Visualization Tools (Pie Charts, Box Plots, Scatter Plots, Bubble Plots)**

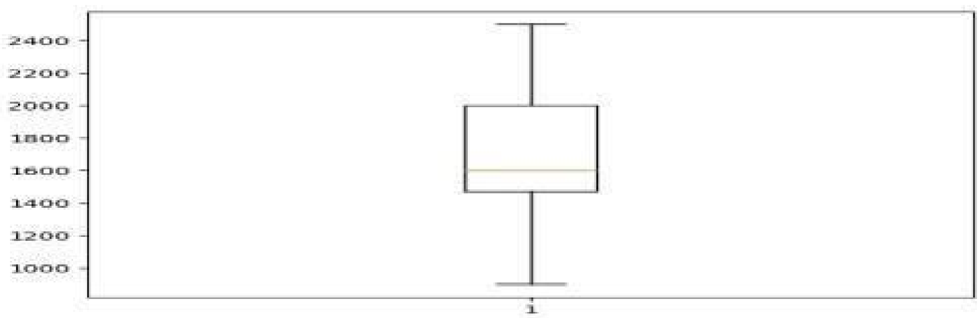
**#PIE Charts:**

```
import matplotlib.pyplot
as pltimport numpy as np
y = np.array([35, 25, 25, 5,5,5])
mylabels = ["VK", "MS",
"Sachin","Hardgik","Bhuvi","Nattu"]myexplode =
[0.2, 0, 0, 0,0,0]
plt.pie(y, labels = mylabels, explode = myexplode
```




**#Box plot :**

```
import pandas as pd import
matplotlib.pyplot as plt
import numpy as np data=
pd.read_csv("data.csv")
data.head() x = data.Volume
plt.boxplot(x)
plt.show()
```



# LAB-12

## AIM: Advanced Visualization Tools

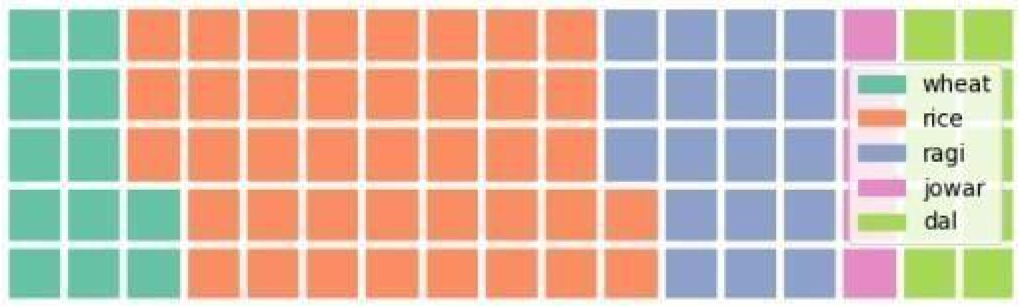
**(Waffle Charts, Word Clouds, Seaborn)**

**Code:**

**#Waffle charts**

```
# python program to generate Waffle Chart
# importing all necessary requirements
import pandas as pd import
matplotlib.pyplot as plt from pywaffle
import Waffle # creation of a dataframe
data ={'grossary': ['wheat',
'rice','ragi','jowar', 'dal'],
'stock': [12, 40, 18, 5, 10]}  df =
pd.DataFrame(data) # To plot the
waffle Chart
 fig = plt.figure(FigureClass =Waffle,
rows = 5,values = df.stock,
labels = list(df.grossary))
```

## Output:



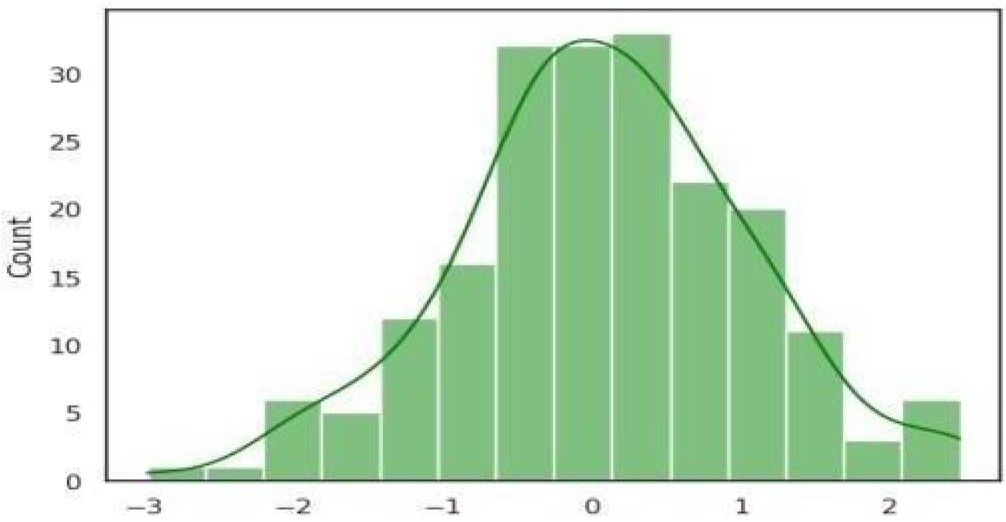

### #world cloud:

```
from wordcloud import WordCloud
import matplotlib.pyplot as plt
text="Hello"
wc=WordCloud().generate(text)
plt.imshow(wc)
plt.axis("off")
plt.show()
```

**Output:**


#SEABORN

```
import numpy as np
import seaborn as sns
sns.set(style="white")
# Generate a random univariate dataset rs =
np.random.RandomState(10) d =
rs.normal(size=200)
# Plot a simple histogram and kde sns.histplot(d,
kde=True, color="green")
```



**#maps**

# import the library import

folium

# Make an empty map m = folium.Map(location=[20,0],

tiles="OpenStreetMap", zoom_start=2)

# Import the pandas library import

pandas as pd

# Make a data frame with dots to show on the map data =

pd.DataFrame({

'lon':[-58, 20.5937, 145, 30.32, -4.03, -73.57, 36.82, -38.5],

'lat':[-34, 78.9629, -38, 59.93, 5.33, 45.52, -1.29, -12.97],

'name':['Buenos Aires', 'norway', 'melbourne', 'St Petersbourg', 'Abidjan',

'Montreal', 'Nairobi', 'Salvador'],

'value':[10, 12, 40, 70, 23, 43, 100, 43]

}, dtype=str)

# add marker one by one on the map for i

in range(0,len(data)):

folium.Marker( location=[data.iloc[i]['lat'],

data.iloc[i]['lon']], popup=data.iloc[i]['name'],

).add_to(m)

# Show the map again