NAME : SAKSHI SHARMA

ROLL NO. : CSC/17/37

EXAM ROLL NO. : 17044570019

SUBJECT : Machine Learning Practical

COURSE : B.Sc.(H) Computer Science

SEMESTER : 6 th

FACULTY : Dr. Megha Gupta

COLLEGE : Mata Sundri College for Women

.

Q1.

CODE:

```
num1 = int(input("Enter First number "));
num2 = int(input("Enter Second Number"));

print("Addition : {0}".format(num1+num2));
print("Subtraction : {0} ".format(num1-num2));
print("Multiplication : {0} ".format(num1*num2));
print("Division : {0} ".format(int(num1/num2)));
print("Modulous : {0} ".format(num1%num2));
print("Exponent : {0} ".format(num1**num2));
```

OUTPUT:

```
Enter First number 12

Enter Second Number 15
Addition : 27
Subtraction : -3
Multiplication : 180
Division : 0
Modulous : 12
```

Exponent : 15407021574586368

Q2.

CODE:

```
print("True AND True : {0}".format( True and True));
print("True AND False : {0}".format(True and False));
print("False AND True : {0}".format(False and True));
print("False AND False : {0}".format(False and False));

print("True OR True : {0}".format(True or True));
print("True OR False : {0}".format(True or False));
print("False OR True : {0}".format(False or True));
print("False OR False : {0}".format(False or False));

var1 = True;
var2 = True;
if(var1==True and var2==True):
 res = False
else:
  res = True
print("True XOR True : {0}".format(res));

var1 = True;
var2 = False;
if(var1==True and var2==False):
 res = True;
else:
```

```
   res = False
print("True XOR False : {0}".format(res));

var1 = False;
var2 = True;
if(var1==False and var2==True):
 res = True;
else:
    res = False
print("False XOR True : {0}".format(res));

var1 =  False;
var2 = False;
if(var1== False and var2== False):
 res = False;
else:
   res = True
print("False XOR False : {0}".format(res));

var1 = True;
print("NOT True : {0}".format(not var1));
var2 = False;
print("NOT False : {0}".format(not var2));

num1 = int(input("Enter First number "));
num2 = int(input("Enter Second Number"));

print("First > Second : {0} ".format(num1>num2));
print("First >= Second : {0} ".format(num1>=num2));
print("First < Second : {0} ".format(num1<num2));
```

```
print("First <= Second : {0} ".format(num1<=num2));
print("First == Second : {0} ".format(num1==num2));
print("First != Second : {0} ".format(num1!=num2));
```

OUTPUT:

```
True AND True : True
True AND False : False
False AND True : False
False AND False : False
True OR True : True
True OR False : True
False OR True : True
False OR False : False
True XOR True : False
True XOR False : True
False XOR True : True
False XOR False : False
NOT True : False
NOT False : True

Enter First number 100

Enter Second Number -1
First > Second : True
First >= Second : True
First < Second : False
First <= Second : False
First == Second : False
```

First != Second : True

Q3.

CODE:

```
str1 = 'Hello'
print("String in \'\' : ",str1)
str2 = "World"
print("String in \"\" : ",str2)
varint = 10234567
print("Integer : " ,varint)
varfloat = 126.4356789101112131415
print("Float : ",varfloat)
a=b=c=1
print("Muliline Assignment : ", a,",",b,",",c)
a,b,c = 1,2,"John"
print("Multiline Assignment : ", a,",",b,",",c)
str = "Hello World!"
print(str[0],str[5])
print(str[2:5])
print(str[2:])
print(str*2)
print(str+"Python")

print(format(1234567891011121314151617 18,"d"))
print(format(12345.67891011121314151617 19,"f"))
print(format(123,"b"))
```

OUTPUT:

String in " : Hello

String in "" : World

Integer : 10234567

Float : 126.4356789101112

Mulilline Assignment : 1 , 1 , 1

Multiline Assignment : 1 , 2 , John

H

llo

llo World!

Hello World!Hello World!

Hello World!Python

12345678910111212131415161718

12345.678910

1111011

Q4.

CODE:

```
a1 = [1,3,5,"Hello"]
print(a1)

import array as arr

a2 = arr.array('d',[1.1,3.5,6.5,7.8,4.5])
```

```
print(a2[0],a2[1],a2[-2],a2[-1])

import numpy as np

a3 = np.array([[1,2,3],[5,6,7]])
print("\n 2-D integer array \n ",a3)

a4 = np.array([[1.0,2,3.1],[5.2,6.5,7.1]])
print("\n 2-D floating array \n",a4)

a5 = np.ones((1,5),dtype = np.int32)
print("\n 1's array \n",a5)

a6 = np.ones((3,3))
print("\n 1's array \n ",a6)

a7 = np.zeros((2,3))
print("\n 0's array \n",a7)

a8 = np.zeros((4,3),dtype = np.int32)
print("\n 0's array \n ",a8)

a9 = np.array([[1,2,3],[3,4,5]],dtype = complex)
print("\n Complex \n ",a9)

a10 = np.array([[1,4,5,12],[-5,8,9,0],[-6,7,11,19]])
print("\n Array \n", a10)
print("\n 1st element of first row \n " ,a10[0][0])
print("\n 3rd element of second row \n " ,a10[1][2])
print("\n last element of last row \n " ,a10[2][3])
```

```
print("\n last element of last row using negative indexing \n " ,a10[-1][-1])

print("\n First row \n " ,a10[0])
print("\n Third row \n " ,a10[2])
print("\n Last row \n " ,a10[-1])
print("\n First column \n " ,a10[:,0])
print("\n Third column \n " ,a10[:,2])
print("\n Last column \n " ,a10[:,-1])

a11 = np.random.rand(5)
print("\n 1-D matrix using rand : \n",a11)

a12 = np.random.rand(4,5)
print("\n 2-D matrix using rand : \n",a12)

a13 = np.random.rand(3,2,2)
print("\n 3-D matrix using rand: \n",a13)

a14 = np.random.uniform(0.5,5.3,10)
print("\n 1-D matrix using uniform : \n",a14)

a15 = np.random.uniform(0,5,(3,2))
print("\n 2-D matrix using uniform : \n",a15)

a16 = np.random.uniform(0,9,(4,3,3))
print("\n 2-D matrix using uniform : \n",a16)

a17 = np.array([1,2,3,4])
d1 = np.diag(a17)
print("\n Diagonal matrix: \n",d1)
```

```
a18 = np.array([[1,4],[-5,8]])
print("\n Diagonal matrix: \n",a18)

d2 = np.diag(a18)
print("\n Main Diagonal element: \n",d2)

d3 = np.diag(a18,1)
print("\nElements above main Diagonal : \n",d3)

d4 = np.diag(a18,-1)
print("\n Elements below main diagonal: \n",d4)
```

OUTPUT:

```
[1, 3, 5, 'Hello']
1.1 3.5 7.8 4.5

 2-D integer array
  [[1 2 3]
 [5 6 7]]

 2-D floating array
 [[1.  2.  3.1]
 [5.2 6.5 7.1]]

 1's array
 [[1 1 1 1 1]]
```

1's array

[[1. 1. 1.]

[1. 1. 1.]

[1. 1. 1.]]

0's array

[[0. 0. 0.]

[0. 0. 0.]]

0's array

[[0 0 0]

[0 0 0]

[0 0 0]

[0 0 0]]

Complex

[[1.+0.j 2.+0.j 3.+0.j]

[3.+0.j 4.+0.j 5.+0.j]]

Array

[[1 4 5 12]

[-5 8 9 0]

[-6 7 11 19]]

1st element of first row

1

3rd element of second row

9

last element of last row

19

last element of last row using negative indexing

19

First row

[1 4 5 12]

Third row

[-6 7 11 19]

Last row

[-6 7 11 19]

First column

[1 -5 -6]

Third column

[5 9 11]

Last column

[12 0 19]

1-D matrix using rand :

[0.84327912 0.20932831 0.85016348 0.43521895 0.45891993]

2-D matrix using rand :

[[0.87442443 0.96230926 0.34772231 0.46740351 0.70036926]

[0.2479808 0.238508   0.20416215 0.84837374 0.37750896]

[0.53377317 0.00774319 0.57375937 0.31987769 0.5773546]
[0.98842077 0.02554143 0.23201988 0.71468806 0.27893878]]

3-D matrix using rand:
[[[0.06860538 0.6844907]
[0.8960919 0.8985415]]

[[0.47066975 0.68900775]
[0.16725069 0.78594411]]

[[0.33161108 0.60426012]
[0.17012185 0.29969946]]]

1-D matrix using uniform :
[2.68238264 3.65871632 1.3839039 0.81738855 0.63355333 3.58277876
5.09553466 3.12347016 1.88688421 0.75620667]

2-D matrix using uniform :
[[3.10346641 0.65627858]
[0.65797177 4.20019602]
[2.17632571 4.66942905]]

2-D matrix using uniform :
[[[3.6548372 6.73618542 1.44721911]
[2.10805213 7.36191572 5.66385406]
[6.43156129 3.57869314 2.7004472]]

[[2.7021788 2.28989542 3.6911749]
[5.31786224 0.36091903 7.88847726]
[7.85225214 7.12800146 4.64329904]]

[[3.2162187 1.78659324 3.03452665]
 [7.35019446 0.86087748 7.4113086]
 [7.42161113 1.87972915 3.69024989]]

[[3.81259641 3.92989337 4.18452357]
 [4.52965047 8.76123401 6.25220314]
 [3.17483915 7.7330704 6.48972392]]]

Diagonal matrix:
[[1 0 0 0]
[0 2 0 0]
[0 0 3 0]
[0 0 0 4]]

Diagonal matrix:
[[1 4]
[-5 8]]

Main Diagonal element:
[1 8]

Elements above main Diagonal :
[4]

Elements below main diagonal:
[-5]

~~~~~~~~~~**R Commands**~~~~~~~~~~
~~~~~~~~~~

concatenation

```
> X<-c(1,3,2,5)
> X
[1] 1 3 2 5
```

Length

```
> length(X)
[1] 4
```

```
> x = c(5,7,9,10)
> x
[1]  5  7  9 10

X = c("aa","bb","cc","dd")
>X
[1] "aa" "bb" "cc" "dd"

X = c(TRUE,FALSE,TRUE,FALSE)
>X
[1]  TRUE FALSE  TRUE FALSE
```

Help

```
> ?c
```

Mathematical operations

```
>X<-c(1,3,2,5)
>Y<-c(5,7,9,10)
>X+Y
[1]  6 10 11 15
```

Matrix

```
X = matrix(c(1,3,2,5),2,2)
>X
     [,1] [,2]
[1,]    1    2
[2,]    3    5
```

Row major matrix

```
>X = matrix(c(1,3,2,5),2,2,byrow = TRUE)
>X
     [,1] [,2]
```

```
[1,]    1    3
[2,]    2    5
```

Reordering arguments

```
X = matrix(c(1,3,2,5),nrow=2,ncol=2,byrow=TRUE)
>X
     [,1] [,2]
[1,]    1    3
[2,]    2    5
```

Square root

```
X = matrix(c(1,2,3,4),2,2)
>sqrt(X)
         [,1]     [,2]
[1,] 1.000000 1.732051
[2,] 1.414214 2.000000
```

Square

```
X^2
     [,1] [,2]
[1,]    1    9
[2,]    4   16
```

Rnorm

```
>rnorm(50,mean=1,sd=1.6)
 [1]  3.09063069 -0.52410352  0.97169871  1.61441864 -1.41472582  0.53687677  4.55909375 -0.39733449  1.04087385
[10]  0.12390157  1.03481918  0.47663212  5.75755134 -1.04735086  1.36354885  1.35993712 -0.77368416  1.09682469
[19]  2.21645723  0.07229656  0.67813928  0.37801434 -0.05199163  0.14864582  0.62728425  1.58952084  3.71922495
[28] -1.44058539  0.27529446 -2.30497610  1.06115919  1.38605404  3.93732172  3.54361986  1.59832978  0.73944230
[37]  1.07902901 -1.83719062  2.89059555  1.34087756  0.56254991 -0.33043075  0.94541943  1.70698435  0.85533442
[46]  3.50232998 -0.04204808 -0.81215786 -1.25564594  1.59510990
```

Correlation

```
>X = rnorm(50)
>Y=X+rnorm(50,mean = 50,sd=.1)
>cor(X,Y)
[1] 0.9931883
```

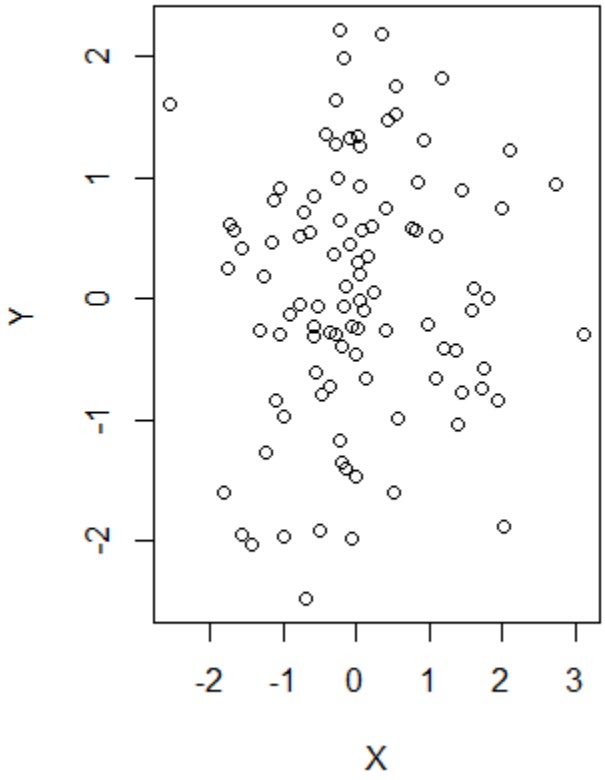
set.seed

```
>set.seed(102)
>Y=rnorm(10)
>mean(Y)
[1] 0.7637291
>set.seed(102)
```

```
>Y=rnorm(10)
>mean(Y)
[1] 0.7637291
>Y=rnorm(10)
>mean(Y)
[1] -0.1792108
```

Plot

```
>X=rnorm(100)
>Y=rnorm(100)
>plot(X,Y)
```



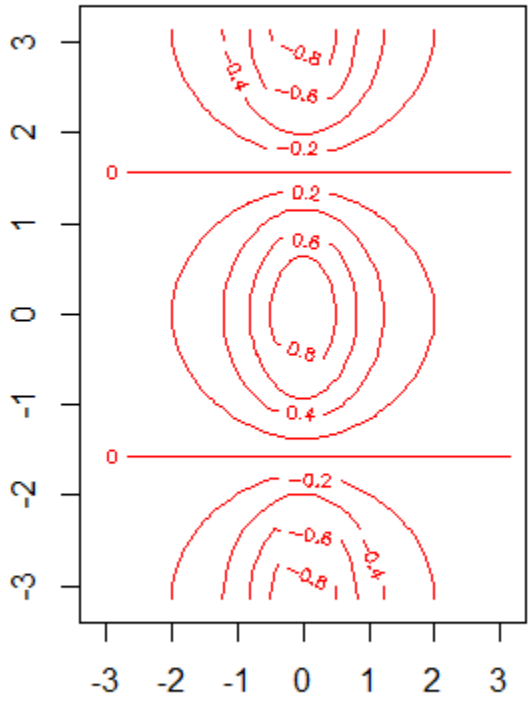
Pdf

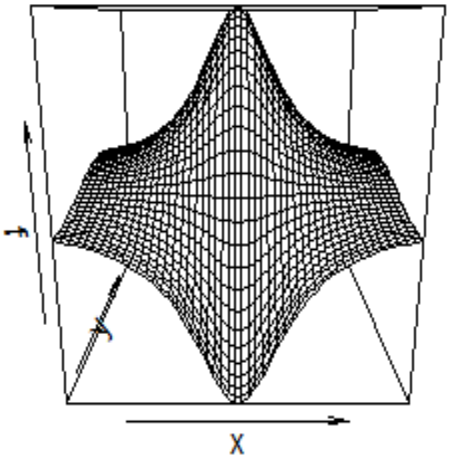
```
>pdf("G1.pdf")
>X=rnorm(100)
>Y=rnorm(100)
>plot(X,Y)
>dev.off()
RStudioGD
        2
```

>

Contour plotting

```
>x = seq(-pi,pi,length=50)
>y=x
>f=outer(x,y,function(x,y) cos(y)/(1+(x^2)))
>contour(x,y,f)
>contour(x,y,f,col="red")
```





Perspective rotated by theta

```
>persp(x,y,f,theta = 30)
```

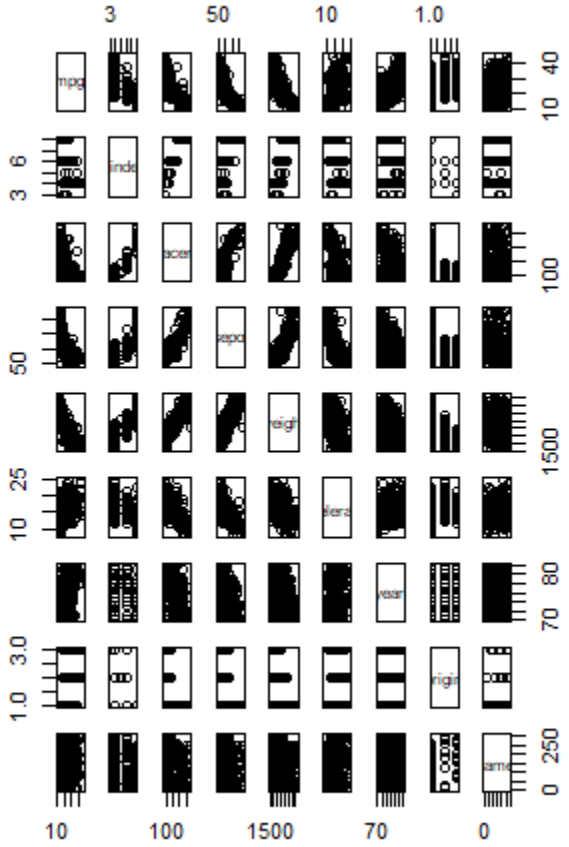
```
> a[c(1,3),c(2,4)]
     [,1] [,2]
[1,]    5   13
[2,]    7   15

> a[1:3,2:4]
     [,1] [,2] [,3]
```

Omitting auto

```
>tdata=na.omit(Auto)
> dim(tdata)
[1] 392   9
```

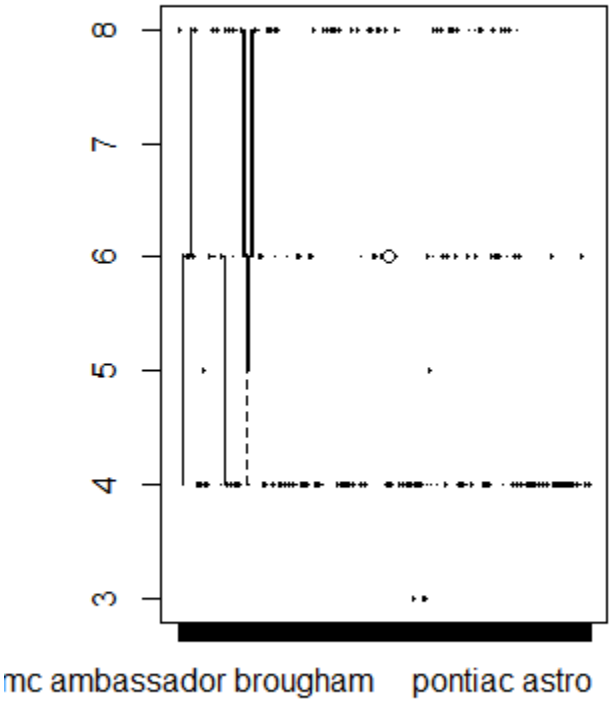
Plotting Auto

```
> plot(Auto)
```



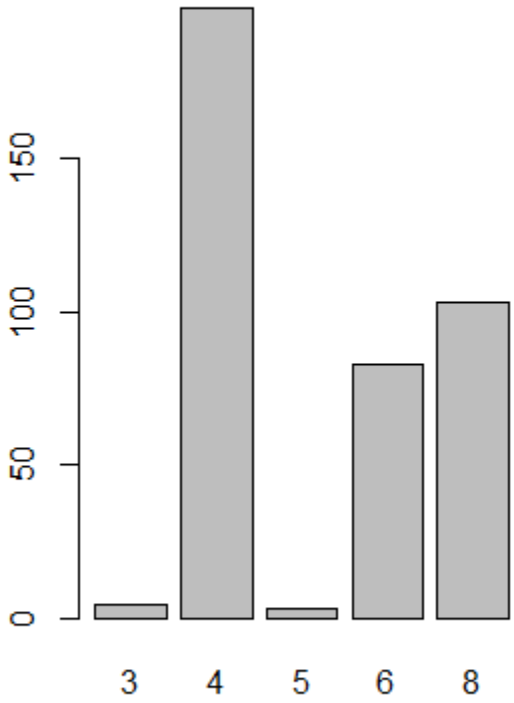
Plotting names,cylinders,mpg

```
> plot(tdata$name,tdata$cylinders,tdata$mpg)
```



Factor

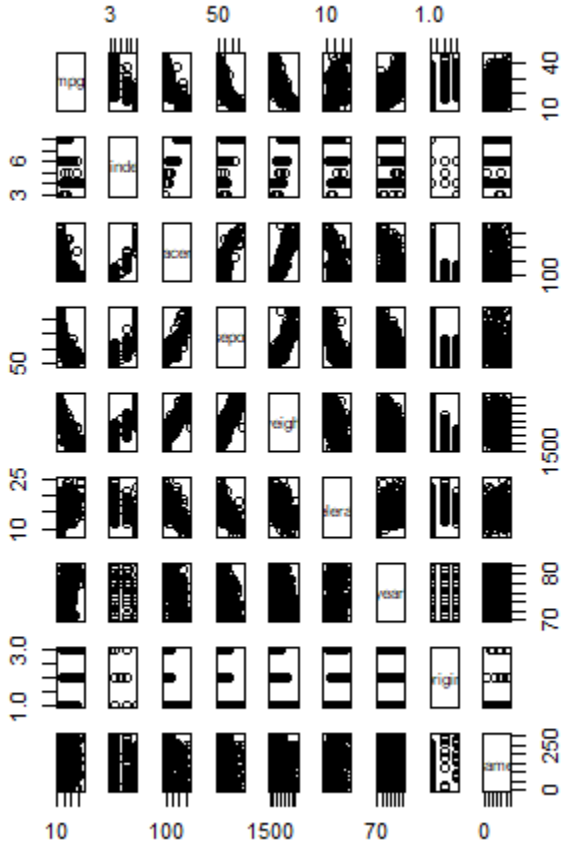
```
> cylinders = as.factor(tdata$cylinders)
> plot(cylinders)
```



Histogram

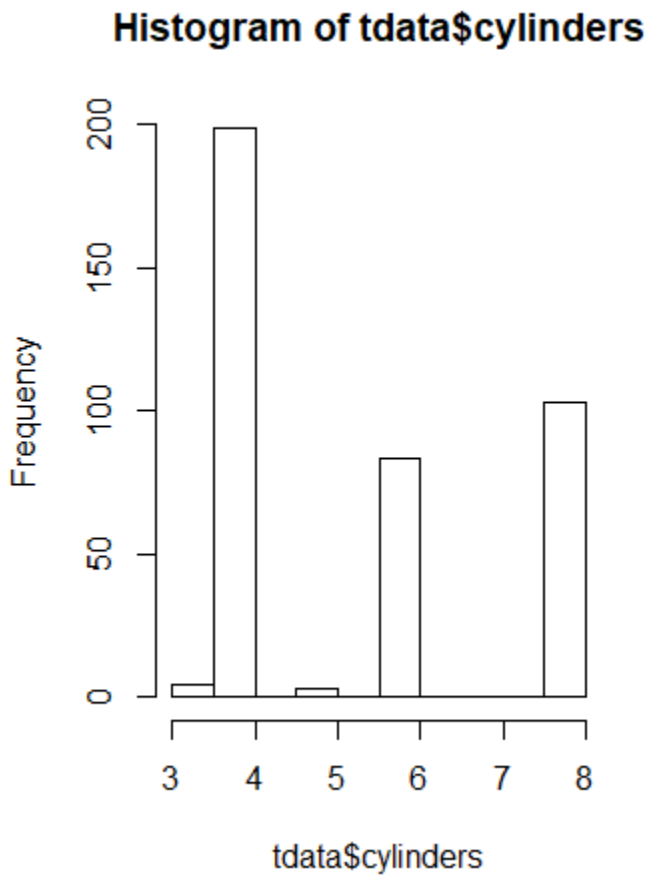
>hist(cylinders)
Error in hist.default(cylinders) : 'x' must be numeric
>hist(tdata$cylinders)

Pairs
> pairs(tdata)



Summary

```
> summary(tdata)
      mpg          cylinders      displacement     horsepower        weight      acceleration        year      
 Min.   : 9.00   Min.   :3.000   Min.   : 68.0   Min.   : 46.0   Min.   :1613   Min.   : 8.00   Min.   :70.00  
 1st Qu.:17.00   1st Qu.:4.000   1st Qu.:105.0   1st Qu.: 75.0   1st Qu.:2225   1st Qu.:13.78   1st Qu.:73.00  
 Median :22.75   Median :4.000   Median :151.0   Median : 93.5   Median :2804   Median :15.50   Median :76.00  
 Mean   :23.45   Mean   :5.472   Mean   :194.4   Mean   :104.5   Mean   :2978   Mean   :15.54   Mean   :75.98  
 3rd Qu.:29.00   3rd Qu.:8.000   3rd Qu.:275.8   3rd Qu.:126.0   3rd Qu.:3615   3rd Qu.:17.02   3rd Qu.:79.00  
 Max.   :46.60   Max.   :8.000   Max.   :455.0   Max.   :230.0   Max.   :5140   Max.   :24.80   Max.   :82.00  

     origin                  name     
 Min.   :1.000   amc matador      :  5  
 1st Qu.:1.000   ford pinto       :  5  
 Median :1.000   toyota corolla   :  5  
 Mean   :1.577   amc gremlin      :  4  
 3rd Qu.:2.000   amc hornet       :  4  
 Max.   :3.000   chevroletchevette:  4  
                 (Other)          :365
```



Q. Matrix operations.

CODE:

```
import numpy as np;

x = np.matrix([[10,20,30],[40,50,60],[70,80,90]],np.int32)
print("Matrix 1 is : ",x)
y = np.matrix([[1,2,3],[4,5,6],[7,8,9]],np.int32)
print("Matrix 2 is : ",y)
print("Sum of 2 matrix is : ", x+y)
print("Sum of 2 matrix using function is : ", sum(x,y))
print("Difference of 2 matrix is : ",x-y)
print("Difference of 2 matrix using function is : ",np.subtract(x,y))
print("Element multiplication is : ",x*y)
print("Element multiplication using function is : ",np.multiply(x,y))
print("Division of 2 matrix is : ", x/y)
print("Division of 2 matrix using function is : ", np.divide(x,y))
```

```
print("Transpose of x matrix is : ",np.transpose(x))
print("Transpose of x matrix is : ",x.T)
a = np.matrix([[1,2,3],[4,5,6]])
print("Matrix a is : ",a)
b = np.matrix([[1,4],[2,5],[3,6]])
print("Matrix b is : ",b)
print("Matrix multiplication is : ",np.dot(a,b))
```

OUTPUT:

```
Matrix 1 is :  [[10 20 30]
 [40 50 60]
 [70 80 90]]
Matrix 2 is :  [[1 2 3]
 [4 5 6]
 [7 8 9]]
Sum of 2 matrix is :  [[11 22 33]
 [44 55 66]
 [77 88 99]]
Sum of 2 matrix using function is :  [[121 152 183]
 [124 155 186]
 [127 158 189]]
Difference of 2 matrix is :  [[ 9 18 27]
 [36 45 54]
 [63 72 81]]
Difference of 2 matrix using function is :  [[ 9 18 27]
 [36 45 54]
 [63 72 81]]
Element multiplication is :  [[ 300  360  420]
 [ 660  810  960]
 [1020 1260 1500]]
Element multiplication using function is :  [[ 10  40  90]
 [160 250 360]
 [490 640 810]]
Division of 2 matrix is :  [[10. 10. 10.]
 [10. 10. 10.]
 [10. 10. 10.]]
Division of 2 matrix using function is :  [[10. 10. 10.]
 [10. 10. 10.]
 [10. 10. 10.]]
Transpose of x matrix is :  [[10 40 70]
 [20 50 80]
 [30 60 90]]
Transpose of x matrix is :  [[10 40 70]
 [20 50 80]
 [30 60 90]]
Matrix a is :  [[1 2 3]
 [4 5 6]]
Matrix b is :  [[1 4]
 [2 5]
```

```
 [3 6]]
Matrix multiplication is :  [[14 32]
 [32 77]]
```

Q5 : Text file

CODE:

```
import numpy as np

tarr = np.random.rand(3,4)
print("Shape of the matrix is : " ,tarr.shape)

print("Length of row and column is : ",len(tarr))
print("Length of first row is : ", len(tarr[:,1]))
print("length of first column is : " ,len(tarr[1,:]))
print("Size of the matrix is : ",tarr.size)

td=np.loadtxt("q5.txt");
print("Text file read is : ", td);

from io import StringIO

c=StringIO(u"123\n456\n789")
d=np.loadtxt(c)
print(d)

d=StringIO(u"M 21 72\n F 35 58")
d=np.loadtxt(d,dtype={'names' :('gender','age','weight'),'formats' : ('S1','i4','f4' )})
print("The text of different data types loaded is : ",d)

x = np.array([[1,2,3],[4,5,6],[7,8,9]],np.int32)
np.savetxt("a.txt",x)
print("a.txt saved")

age=32
global ml
ml="hello"
name="mary"
defchange_name(n_name):
 name=n_name
globals()["ml"]="world";
 print("local variable : ",locals())
print(name)
change_name("Mohan")
print(name)
print(ml)
```

OUTPUT:

```
Shape of the matrix is :  (3, 4)
Length of row and column is :  3
Length of first row is :  3
length of first column is :  4
Size of the matrix is :  12
Text file read is :  [[ 12.  57.  89.]
 [ 23.   5.   6.]
 [ 89. 765.  54.]]
[123. 456. 789.]
The text of different data types loaded is :  [(b'M', 21, 72.) (b'F', 35, 58.)]
a.txt saved
mary
local variable :  {'n_name': 'Mohan', 'name': 'Mohan'}
mary
world
```

Q6.

CODE:

```
import numpy as np;

x = np.matrix([[10,20,30],[40,50,60],[70,80,90]],np.int32)
print("Matrix 1 is : ",x)
y = np.matrix([[1,2,3],[4,5,6],[7,8,9]],np.int32)
print("Matrix 2 is : ",y)
print("Sum of 2 matrix is : ", x+y)
print("Sum of 2 matrix using function is : ", sum(x,y))
print("Difference of 2 matrix is : ",x-y)
print("Difference of 2 matrix using function is : ",np.subtract(x,y))
print("Element multiplication is : ",x*y)
print("Element multiplication using function is : ",np.multiply(x,y))
print("Division of 2 matrix is : ", x/y)
print("Division of 2 matrix using function is : ", np.divide(x,y))
print("Transpose of x matrix is : ",np.transpose(x))
print("Transpose of x matrix is : ",x.T)
a = np.matrix([[1,2,3],[4,5,6]])
print("Matrix a is : ",a)
b = np.matrix([[1,4],[2,5],[3,6]])
print("Matrix b is : ",b)
print("Matrix multiplication is : ",np.dot(a,b))
```

OUTPUT:

```
Matrix 1 is :  [[10 20 30]
```

```
 [40 50 60]
 [70 80 90]]
Matrix 2 is :  [[1 2 3]
 [4 5 6]
 [7 8 9]]
Sum of 2 matrix is :  [[11 22 33]
 [44 55 66]
 [77 88 99]]
Sum of 2 matrix using function is :  [[121 152 183]
 [124 155 186]
 [127 158 189]]
Difference of 2 matrix is :  [[ 9 18 27]
 [36 45 54]
 [63 72 81]]
Difference of 2 matrix using function is :  [[ 9 18 27]
 [36 45 54]
 [63 72 81]]
Element multiplication is :  [[ 300  360  420]
 [ 660  810  960]
 [1020 1260 1500]]
Element multiplication using function is :  [[ 10  40  90]
 [160 250 360]
 [490 640 810]]
Division of 2 matrix is :  [[10. 10. 10.]
 [10. 10. 10.]
 [10. 10. 10.]]
Division of 2 matrix using function is :  [[10. 10. 10.]
 [10. 10. 10.]
 [10. 10. 10.]]
Transpose of x matrix is :  [[10 40 70]
 [20 50 80]
 [30 60 90]]
Transpose of x matrix is :  [[10 40 70]
 [20 50 80]
 [30 60 90]]
Matrix a is :  [[1 2 3]
 [4 5 6]]
Matrix b is :  [[1 4]
 [2 5]
 [3 6]]
Matrix multiplication is :  [[14 32]
 [32 77]]
```

Q7.

CODE:

```
import numpy as np;
```

```
m1 = np.array([[1,-2,3],[4,-5,6],[7,-8,9]],np.int32)
mat1 = np.matrix([[1,-2,3],[4,-5,6],[7,-8,9]],np.int32)
print("The matrix is : ",m1)
print("The absolute of matrix is : ",np.absolute(m1))

print("The negative of the matrix is : ",np.negative(m1))

c1 = np.array([10,11,-12])
print("The column to be added is : ",c1 )

print("Adding column using hstack  : ")
m2 = np.hstack((m1,np.atleast_2d(c1).T))
print(m2)

print("Adding column to matrix using hstack  : ")
mat2 = np.hstack((mat1,np.atleast_2d(c1).T))
print(mat2)

print("Adding column using column stack : ")
m3 = np.column_stack((m1,c1))
print(m3)

print("Adding column to matrix using column stack : ")
mat3 = np.column_stack((mat1,c1))
print(mat3)

r1 = np.array([13,14,15,-16])
print("Adding row to the matrix using vstack: ")
m2 = np.vstack((m2,r1))
print(m2)

print("Deleting 2nd row : ")
m4 = np.delete(m2,1,0)
print(m4)

print("Deleting last column : ")
m5 = np.delete(m2,-1,1)
print(m5)

print("Deleting first row  : ")
m6 = np.delete(m2,0,0)
print(m6)

print("Deleting last row : ")
m7 = np.delete(m2,-1,0)
print(m7)

print("Deleting last column : ")
m8 = np.delete(m2,-1,-1)
print(m8)
```

```
print("Deleting 2nd element of 1st column : ")
m9 = np.delete(m2,1)
print(m9)

print("Deleting first element : ")
m10 = np.delete(m2,0,None)
print(m10)

defmatmax(mat):
    max = mat[0][0]
    for i in range(4):
        for j in range(4):
            if mat[i][j]>max:
                max = mat[i][j]
    print(max)

print(" Maximum of matrix is : ")
matmax(m2)

defmatmin(mat):
    min = mat[0][0]
    for i in range(4):
        for j in range(4):
            if mat[i][j]<min:
                min = mat[i][j]
    print(min)

print(" Minimum of matrix is : ")
matmin(m2)

print("Maximum of matrix is : " ,m2.max())
print("Minimum of the matrix is : ", m2.min())
print("Maximum of each row is : " , m2.max(1))
print("Minimum of each row is : " , m2.min(1))
print("Maximum of each column is : " , m2.max(0))
print("Minimum of each column is : " , m2.min(0))


defcolmax(mat,c):
    max = mat[0][c]
    for i in range(4):
            if mat[i][c]>max:
                max = mat[i][c]
    print(max)

defcolmin(mat,c):
    min = mat[0][c]
    for i in range(4):
        if mat[i][c]<min:
                min = mat[i][c]
    print(min)
```

```
c = int(input("Enter the column number to find max and min : "))
print("The maximum of the column is : ")
colmax(m2,c)
print("The minimum of the column is : ")
colmin(m2,c)

defrowmax(mat,r):
    max = mat[r][0]
    for j in range(4):
        if mat[r][j]>max:
            max = mat[r][j]
    print(max)

defrowmin(mat,r):
    min = mat[r][0]
    for j in range(4):
        if mat[r][j]<min:
                min = mat[r][j]
    print(min)

r = int(input("Enter the row number to find max and min : "))
print("The maximum of the row is : ")
rowmax(m2,r)
print("The minimum of the row is : ")
rowmin(m2,r)

defsummat(mat):
matsum=0;
    for i in range(4):
        for j in range(4):
matsum= matsum+mat[i][j]
    print(matsum)

print("The sum of elements of matrix is : ")
summat(m2)


defrowsum(mat,r):
sumrow=0
    for i in range(4):
sumrow=sumrow + mat[r][i]
    print(sumrow)

r = int(input("Enter the row number to find the sum : "))
print("The sum of elements of the row is : ")
rowsum(m2,r)

defcolsum(mat,c):
sumcol=0
    for i in range(4):
```

```
sumcol=sumcol+ mat[i][c]
   print(sumcol)

c = int(input("Enter the column number to find the sum : "))
print("The sum of elements of the column is : ")
colsum(m2,c)
```

OUTPUT:

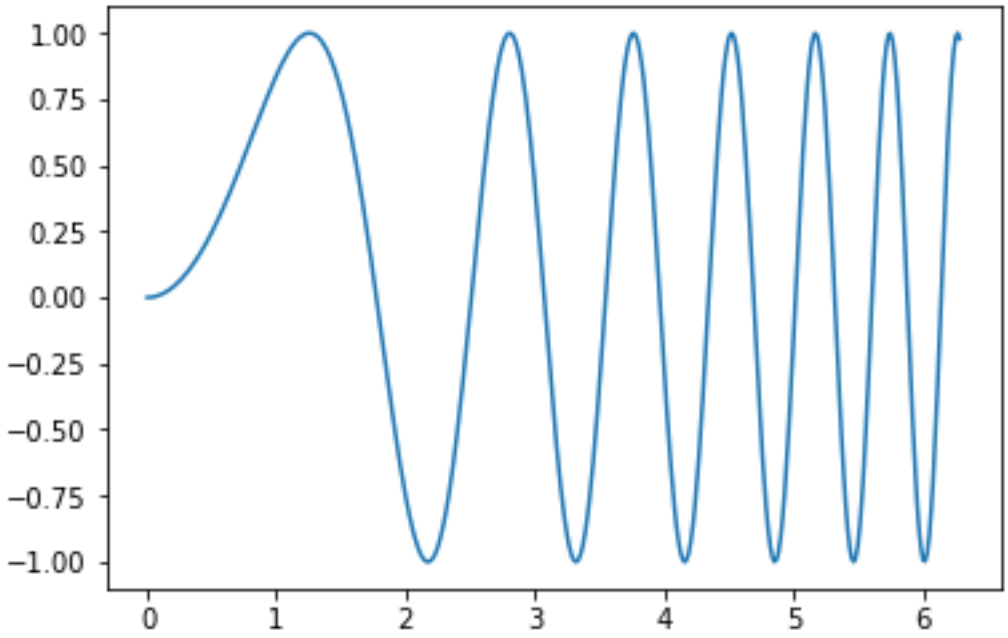
```
The matrix is :  [[ 1 -2  3]
 [ 4 -5  6]
 [ 7 -8  9]]
The absolute of matrix is :  [[1 2 3]
 [4 5 6]
 [7 8 9]]
The negative of the matrix is :  [[-1  2 -3]
 [-4  5 -6]
 [-7  8 -9]]
The column to be added is :  [ 10  11 -12]
Adding column using hstack  :
[[  1  -2   3  10]
 [  4  -5   6  11]
 [  7  -8   9 -12]]
Adding column to matrix using hstack  :
[[  1  -2   3  10]
 [  4  -5   6  11]
 [  7  -8   9 -12]]
Adding column using column stack :
[[  1  -2   3  10]
 [  4  -5   6  11]
 [  7  -8   9 -12]]
Adding column to matrix using column stack :
[[  1  -2   3  10]
 [  4  -5   6  11]
 [  7  -8   9 -12]]
Adding row to the matrix using vstack:
[[  1  -2   3  10]
 [  4  -5   6  11]
 [  7  -8   9 -12]
 [ 13  14  15 -16]]
Deleting 2nd row :
[[  1  -2   3  10]
 [  7  -8   9 -12]
 [ 13  14  15 -16]]
Deleting last column :
[[ 1 -2  3]
 [ 4 -5  6]
 [ 7 -8  9]
 [13 14 15]]
```

```
Deleting first row  :
[[  4  -5   6  11]
 [  7  -8   9 -12]
 [ 13  14  15 -16]]
Deleting last row :
[[  1  -2   3  10]
 [  4  -5   6  11]
 [  7  -8   9 -12]]
Deleting last column :
[[ 1 -2  3]
 [ 4 -5  6]
 [ 7 -8  9]
 [13 14 15]]
Deleting 2nd element of 1st column :
[  1   3  10   4  -5   6  11   7  -8   9 -12  13  14  15 -16]
Deleting first element :
[ -2   3  10   4  -5   6  11   7  -8   9 -12  13  14  15 -16]
 Maximum of matrix is :
15
 Minimum of matrix is :
-16
Maximum of matrix is :  15
Minimum of the matrix is :  -16
Maximum of each row is :  [10 11  9 15]
Minimum of each row is :  [ -2  -5 -12 -16]
Maximum of each column is :  [13 14 15 11]
Minimum of each column is :  [  1  -8   3 -16]

Enter the column number to find max and min : 1
The maximum of the column is :
14
The minimum of the column is :
-8

Enter the row number to find max and min : 2
The maximum of the row is :
9
The minimum of the row is :
-12
The sum of elements of matrix is :
50

Enter the row number to find the sum : 2
The sum of elements of the row is :
-4

Enter the column number to find the sum : 1
The sum of elements of the column is :
-1
```

Q8. linespace.py

CODE:

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



x = np.linspace(0,2*m.pi,400)
x2 = np.square(x)
sinval = np.sin(x2)
plt.plot(x,sinval)
```

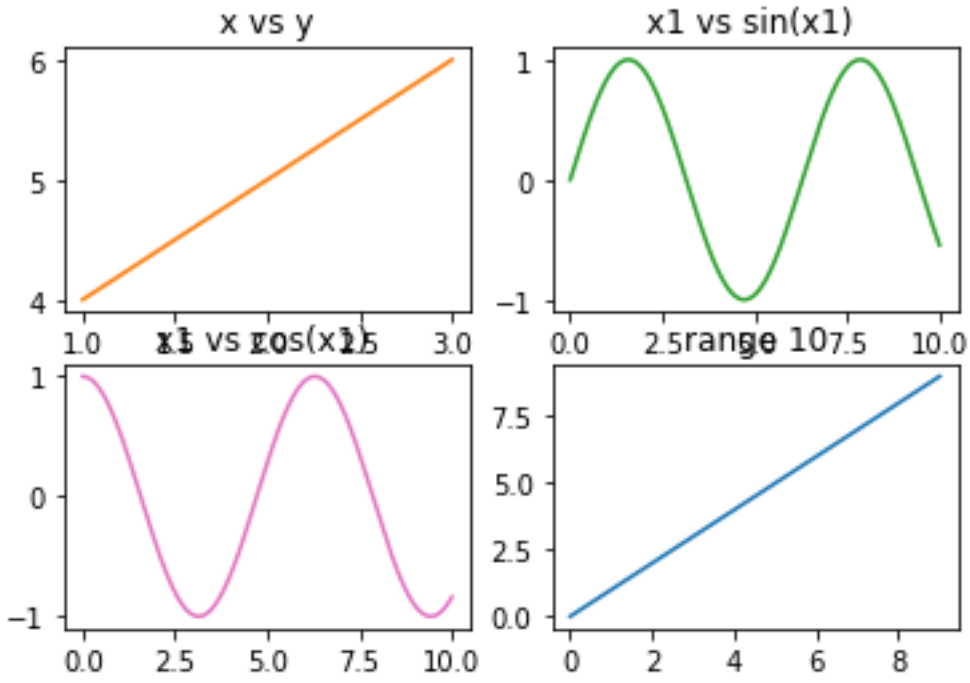


Q9. Colorsubplotting

CODE:

```
import numpy as np
import math as m

x = [1,2,3]
y = [4,5,6]
x1 = np.linspace(0,10,1000)

fig,a = plt.subplots(2,2)

a[0][0].plot(x,y,'tab:orange')
a[0][0].set_title("x vs y")
```

```
sinval = np.sin(x1)
a[0][1].plot(x1,sinval,'tab:green')
a[0][1].set_title("x1 vs sin(x1) ")

cosval = np.cos(x1)
a[1][0].plot(x1,cosval,'tab:pink')
a[1][0].set_title("x1 vs cos(x1) ")

a[1][1].plot(range(10))
a[1][1].set_title("range 10")

plt.show()
```



Q12. linear regression on the basis of sqft_living and price

CODE:

```
import pandas as pd
from sklearn.model_selection import train_test_split


data = pd.read_csv("kc_house_data.csv")
feature_cols = 'sqft_living'
x = data[feature_cols]
y = data.price

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)

from sklearn.linear_model import LinearRegression

linreg = LinearRegression()
x_train = x_train.to_frame()
y_train = y_train.to_frame()
linreg.fit(x_train,y_train)
y_pred = linreg.predict(x_test.to_frame())
print("Comparing values %.2f predicted %.2f",y_test,y_pred)
```

OUTPUT:

```
Comparing values %.2f predicted %.2f 6228      380000.0
13827     339100.0
7693      389000.0
19664     275000.0
```

10988 299990.0

20892 790000.0

8896 650000.0

8169 630000.0

19448 450000.0

20632 580000.0

1901 375000.0

16174 425000.0

854 327000.0

10915 332000.0

16948 542500.0

6680 535900.0

8288 650000.0

2105 310000.0

4133 523000.0

4825 415000.0

11725 617000.0

6646 265000.0

17061 272000.0

1885 340000.0

1815 477000.0

904 300000.0

995 291000.0

8797 715000.0

7907 3200000.0

7406 425000.0

18159 315000.0

10481 484000.0

8211 222900.0

16545 362000.0

16875 425000.0

2682 687000.0

16168 623000.0

2772 960000.0

12507 725000.0

17338 815000.0

2559 890000.0

9505 525000.0

11471 1180000.0

10737 1035000.0

16656 324900.0

6994 1295650.0

2488 425000.0

15577 320900.0

16399 206000.0

12351 1000000.0

14459 430000.0

18196 760000.0

10584 1060000.0

18637 222400.0

12934 327200.0

5552 540400.0

12510 260000.0

20532 550000.0

3161 780000.0

12199 1015000.0

Name: price, Length: 4323, dtype: float64 [[474232.30726418]

[615770.8524978]

[610220.32131217]

...

[499209.69759952]

[413176.46422222]

[965454.31719263]]

Q13. Multiple regression

CODE:

```
import pandas as pd
from sklearn.model_selection import train_test_split



data = pd.read_csv("kc_house_data.csv")
#feature_cols = 'sqft_living'
x = data[['sqft_living','bedrooms','bathrooms','floors','condition','grade']]
y = data['price']

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)

from sklearn.linear_model import LinearRegression

linreg = LinearRegression()
#x_train = x_train.to_frame()
y_train = y_train.to_frame()
linreg.fit(x_train,y_train)
#y_pred = linreg.predict(x_test.to_frame())
y_pred = linreg.predict(x_test)
print("Comparing values %.2f predicted %.2f",y_test,y_pred)
```

OUTPUT:

Comparing values %.2f predicted %.2f 7498 295000.0

13999 575000.0

12004 845000.0

4540 550000.0

2204 230000.0

5104 569950.0

10816 605000.0

13539 230000.0

12810 302000.0

20236 588000.0

10209 539000.0

15437 990000.0

3748 2450000.0

10484 405000.0

18590 194250.0

7203 995000.0

513 290000.0

18067 435000.0

21438 435000.0

13053 445500.0

18269 425000.0

3891 1225000.0

1388 267000.0

6294 265953.0

16483 627000.0

12716 455000.0

18185 825000.0

10326 150000.0

6437 152000.0

17772 270000.0

6904 349900.0

6713 286000.0

17672 690000.0

8989 454000.0

1203 510000.0

18085 525000.0

12082 325000.0

18147 435000.0

11215 1234580.0

2216 327000.0

16298 887000.0

17644 239950.0

16078 734000.0

4603 305000.0

673 865000.0

8754 1175000.0

11776 238000.0

3247 464000.0

14482 425000.0

6348 440000.0

19164 244000.0

14383 285000.0

9218 305000.0

9125 562000.0

9822 485000.0

11698 315000.0

4286 300000.0

16889 835000.0

18285 476500.0

16446 765000.0

Name: price, Length: 4323, dtype: float64 [[427327.96618747]

[549664.36941511]

[354218.34456478]

...

[846018.07772843]

[544468.90032674]

[1146968.79026704]]

Q14 and 15. Classification and Regularization.

CODE:

```
from pandas import read_csv
from matplotlib import pyplot
from scipy import optimize as op
from pandas.plotting import scatter_matrix
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

print("\n Reading the dataset \n")
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"
names = ['sepal-length','sepal-width','petal-length','petal-width','class']
dataset = read_csv(url,names=names)

print("\n Summarising the dataset \n")
```

```
print(dataset.shape)
print(dataset.head(10))

print("\n Obtaining Statistical Summary \n ")
print(dataset.describe())

print("\n Getting Class Distribution \n ")
print(dataset.groupby('class').size())

print("\n Visualising Dataset \n")
dataset.hist()
pyplot.show()

print("\n Multivariate Plots \n")
scatter_matrix(dataset)
pyplot.show()

print("\n Evaluating Algorithms \n ")
array = dataset.values
X = array[:,0:4]
Y = array[:,4]
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size = 0.20,random_state = 1)

print("\n Build Model \n ")
logreg = LogisticRegression()
logreg.fit(X_train,Y_train)
print('Accuracy of Logistic Regression classifier on training set:
{:.2f}'.format(logreg.score(X_train,Y_train)))
print('Accuracy of Logistic Regression classifier on test set: {:.2f}'.format(logreg.score(X_test,Y_test)))
```

```
print("\n Making Predictions \n ")
predictions = logreg.predict(X_test)
print(accuracy_score(Y_test,predictions))

print("\n Classification Metrics \n ")
print(classification_report(Y_test,predictions))

print("\n\n Regularisation \n\n ")

print("Data Setup")

import numpy as np
Species = ['Iris-setosa','Iris-versicolor','Iris-virginica']
m = dataset.shape[0] #number of examples
n=4 #features
k=3 #Number of classes

X=np.ones((m,n+1))
y=np.array((m,1))
X[:,1] = dataset['petal-length'].values
X[:,2] = dataset['petal-width'].values
X[:,3] = dataset['sepal-length'].values
X[:,4] = dataset['sepal-width'].values

print("Mean Normalization")
y = dataset['class'].values
for j in range(n):
    X[:,j] = (X[:,j]-X[:,j]-X[:,j].mean())
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.2,random_state = 4)
X = dataset.drop(['class'],axis =1)
```

```
y=dataset['class']
print(X_train.shape)
print(y_test.shape)

#Sigmoid function
def sigmoid(z):
    return 1.0/(1+np.exp(-z))

#Regularised cost function
def reglrCostFunction(theta,X,y,lambda_s = 0.1):
    m = len(y)
    h = sigmoid(X.dot(theta))
    J = (1/m)*(-y.T.dot(np.log(h))-(1-y).T.dot(np.log(1-h)))
    reg = (lambda_s/(2*m))*np.sum(theta**2)
    J  = J+reg
    return J

#regularised gradient function
def reglrGradient(theta,X,y,lambda_s=0.1):
    m,n = X.shape
    theta = theta.reshape((n,1))
    y = y.reshape((m,1))
    h = sigmoid(X.dot(theta))
    reg = lambda_s*theta/m
    gd  = ((1/m)*X.T.dot(h-y))
    gd = gd+reg
    return gd

#optimizing logistic regression(theta)
def logisticRegression(X,y,theta):
```

```
    result = op.minimize(fun = reglrCostFunction , x0 = theta , args = (X,y),method = 'TNC',jac =
reglrGradient)
    return result.x


all_theta = np.zeros((k,n+1))
i=0
for flower in Species:
    tmp_y = np.array(y_train == flower,dtype = int)
    optTheta = logisticRegression(X_train,tmp_y,np.zeros((n+1,1)))
    all_theta[i]=optTheta
    i+=1


#predictions and accuracy
Prob = sigmoid(X_test.dot(all_theta.T))
pred = [Species[np.argmax(Prob[i,:])] for i in range(X_test.shape[0])]
print("Test Accuracy ",accuracy_score(y_test,pred)
```

Reading the dataset

Summarising the dataset

(150, 5)

	sepal-length	sepal-width	...	petal-width	class
0	5.1	3.5	...	0.2	Iris-setosa
1	4.9	3.0	...	0.2	Iris-setosa
2	4.7	3.2	...	0.2	Iris-setosa
3	4.6	3.1	...	0.2	Iris-setosa
4	5.0	3.6	...	0.2	Iris-setosa
5	5.4	3.9	...	0.4	Iris-setosa

6	4.6	3.4	...	0.3	Iris-setosa
7	5.0	3.4	...	0.2	Iris-setosa
8	4.4	2.9	...	0.2	Iris-setosa
9	4.9	3.1	...	0.1	Iris-setosa

[10 rows x 5 columns]

Obtaining Statistical Summary

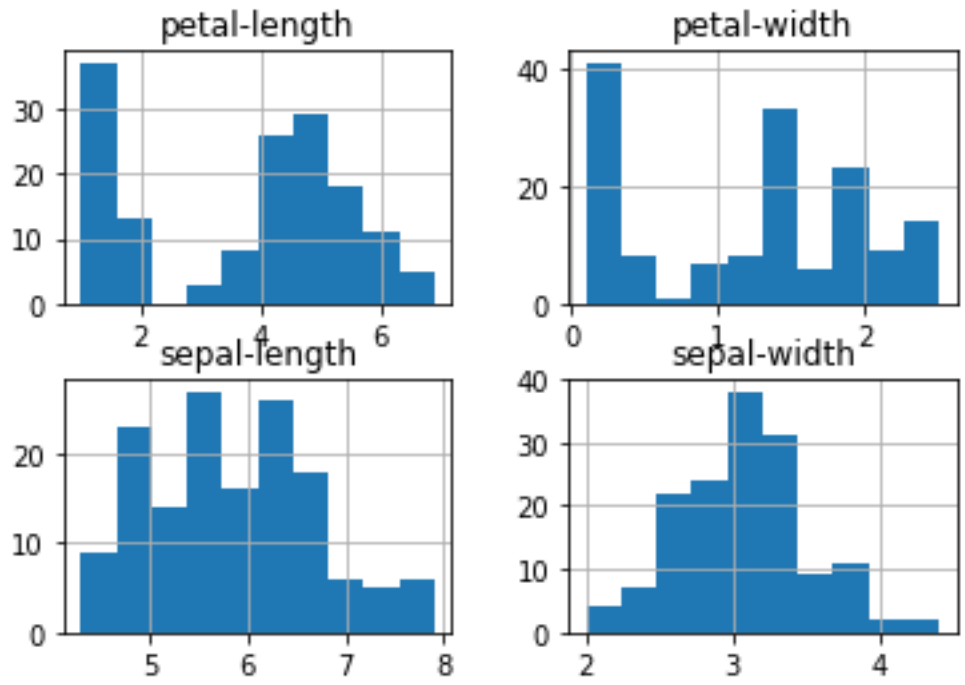
	sepal-length	sepal-width	petal-length	petal-width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Getting Class Distribution

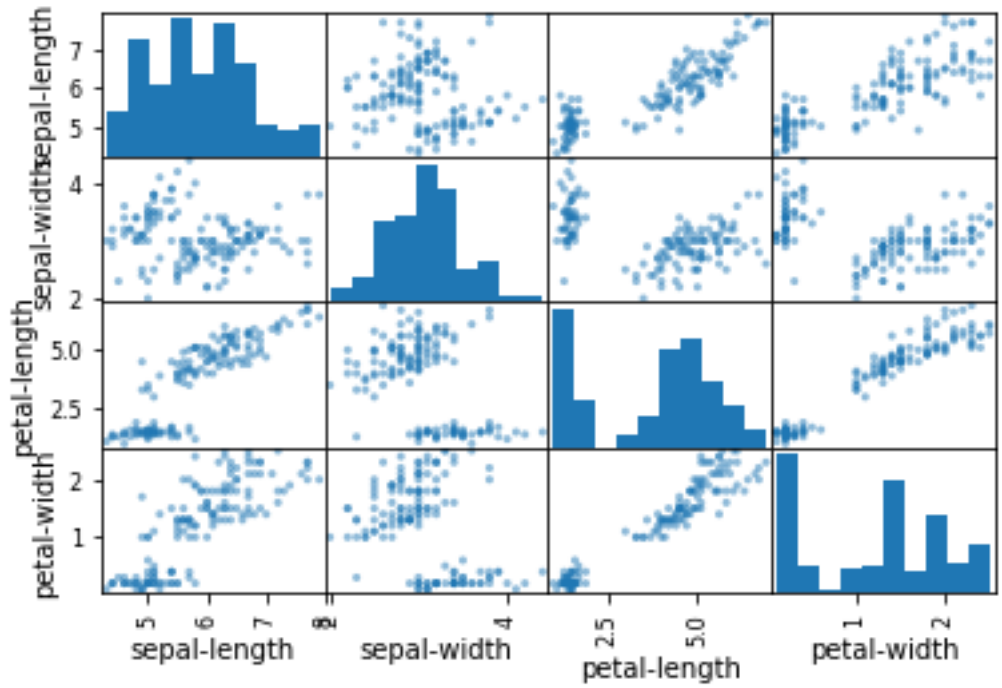
class	
Iris-setosa	50
Iris-versicolor	50
Iris-virginica	50

dtype: int64

Visualising Dataset



Multivariate Plots



Evaluating Algorithms

Build Model

Accuracy of Logistic Regression classifier on training set: 0.96

Accuracy of Logistic Regression classifier on test set: 0.83

Making Predictions

0.8333333333333334

Classification Metrics

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	11
Iris-versicolor	1.00	0.62	0.76	13
Iris-virginica	0.55	1.00	0.71	6
avg / total	0.91	0.83	0.84	30

Regularisation

Data Setup

Mean Normalization

(120, 5)

(30,)

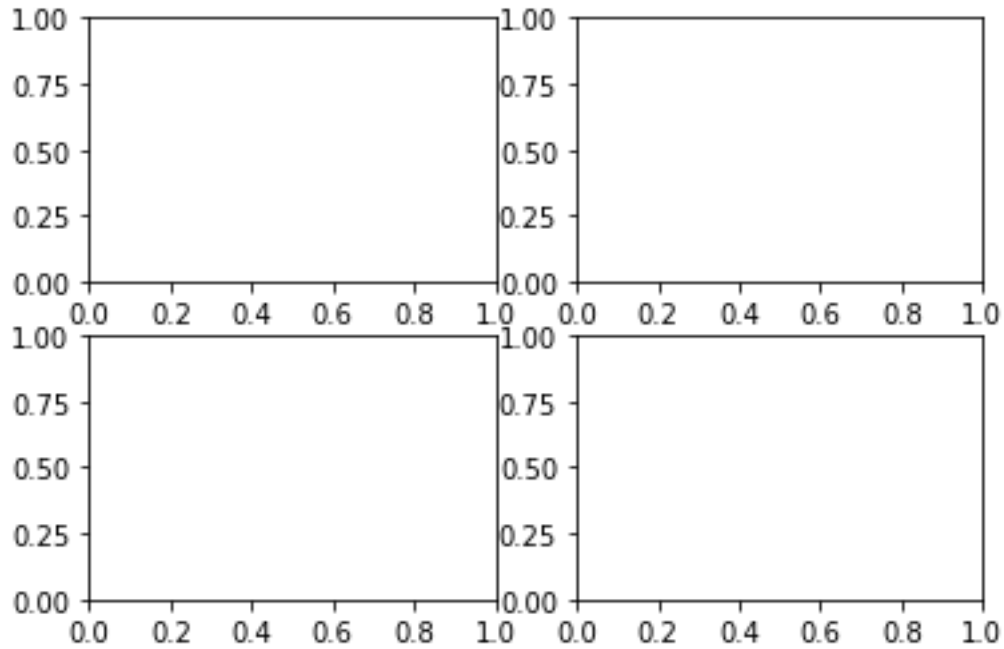
Test Accuracy 56.666666666666664 %

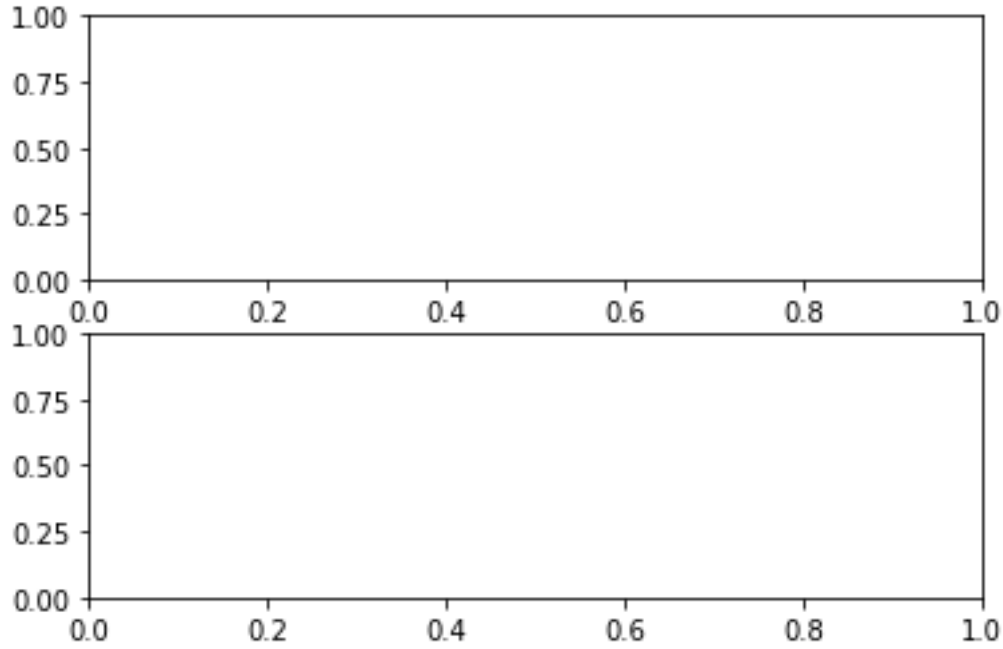
~~~~~~~~~~~EXTRA QUESTIONS~~~~~~~~~~~~~~~

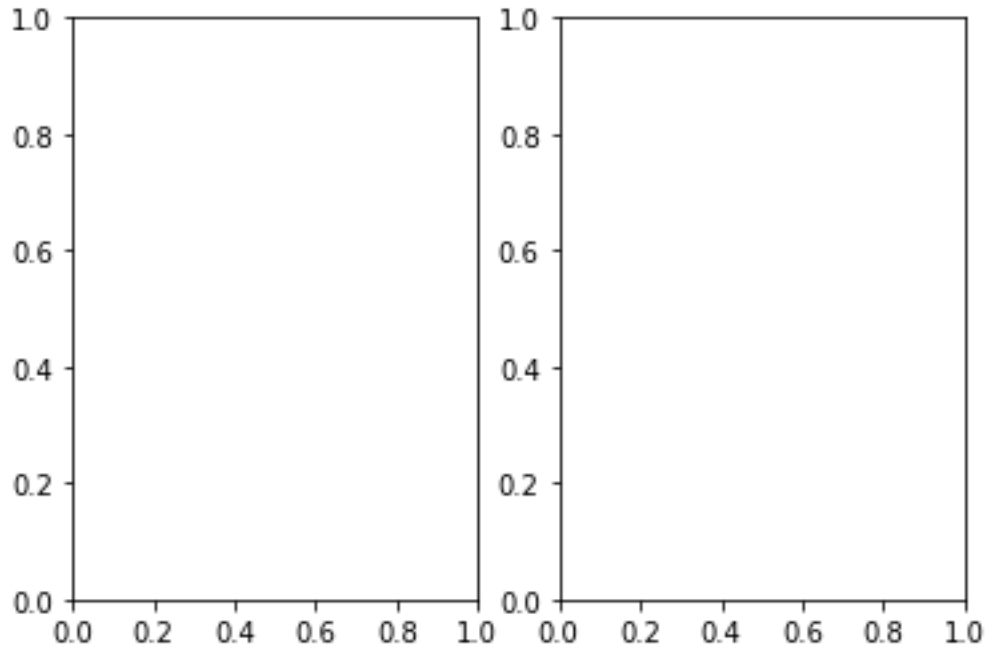
Q. Subplot.py

CODE:

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

plt.subplots(2,2)
plt.subplots(2,1)
plt.subplots(1,2)
```



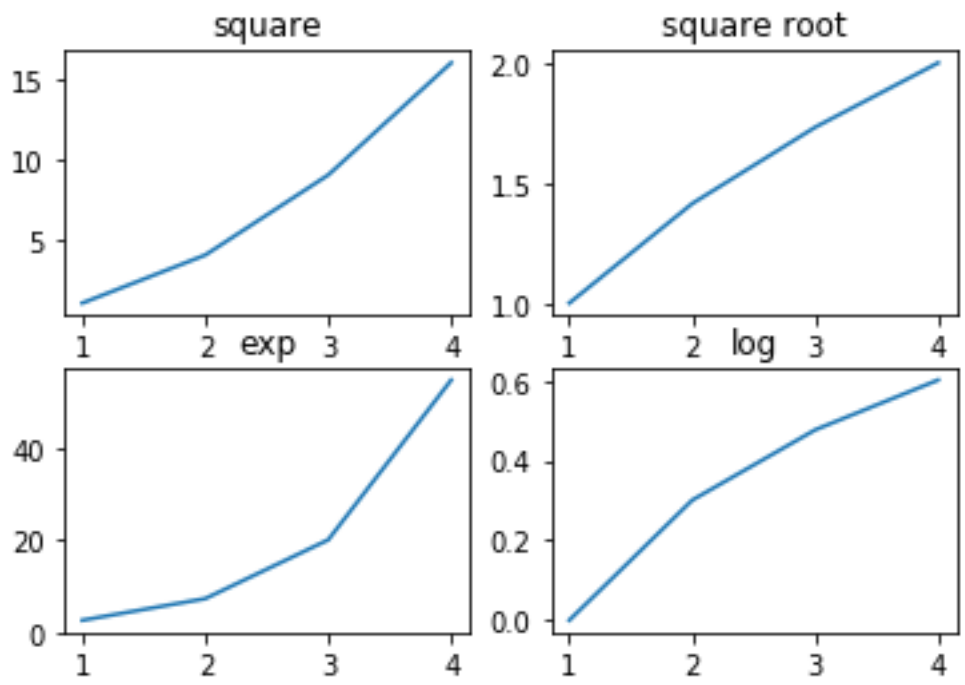




Q. Subplotting.py

CODE:

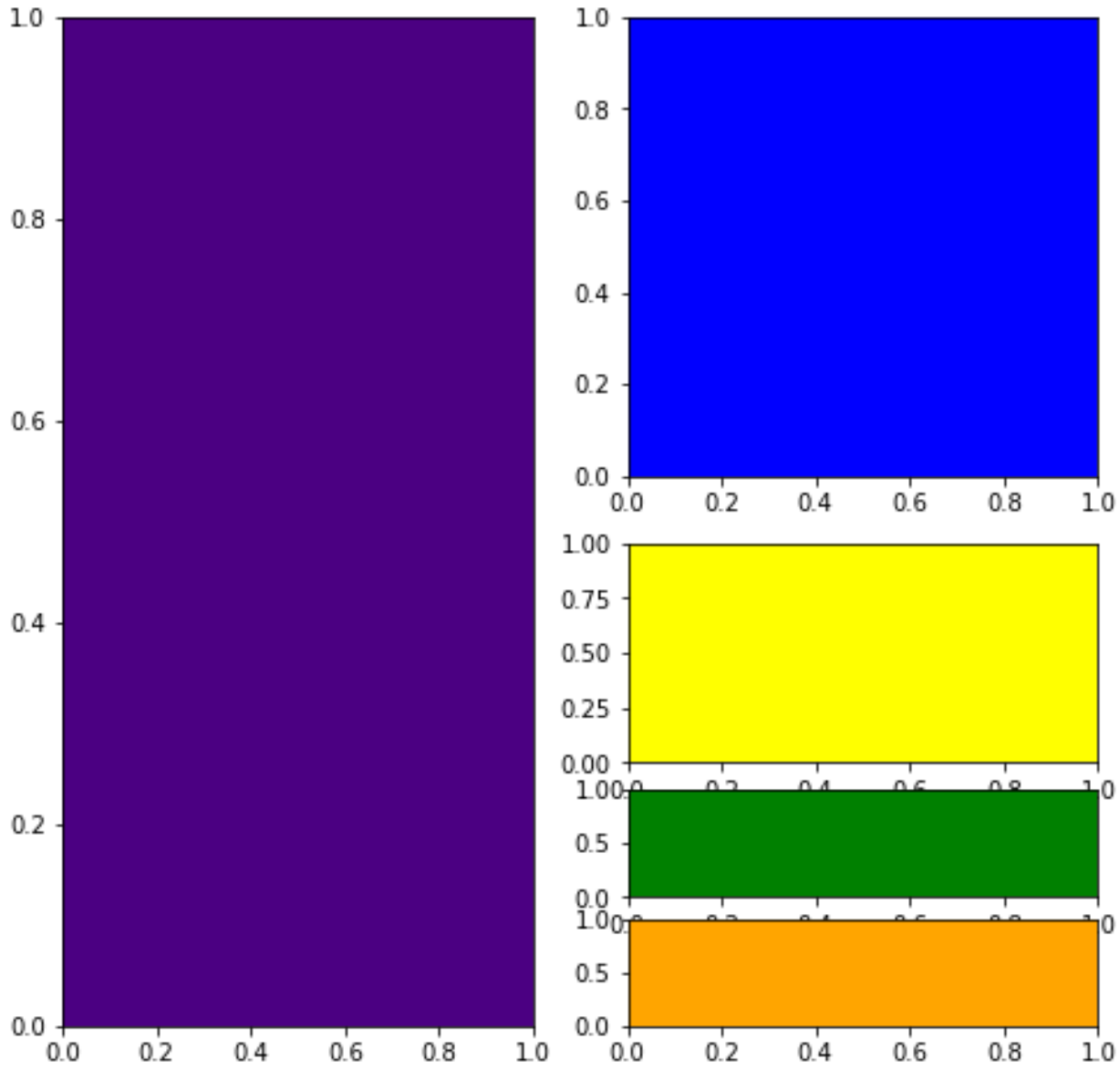
```
import matplotlib.pyplot as plt
fig,a = plt.subplots(2,2)

import numpy as np

x = np.arange(1,5)

a[0][0].plot(x,x*x)
a[0][0].set_title("square")

a[0][1].plot(x,np.sqrt(x))
a[0][1].set_title("square root")

a[1][0].plot(x,np.exp(x))
a[1][0].set_title("exp")

a[1][1].plot(x,np.log10(x))
a[1][1].set_title("log")

plt.show()
```



Q. addsubplot.py

CODE:

```
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(8,8))
ax1 = fig.add_subplot(1,2,1,fc = 'indigo')
ax2 = fig.add_subplot(2,2,2,fc = 'blue')
ax3 = fig.add_subplot(4,2,6,fc = 'yellow')
ax4 = fig.add_subplot(8,2,14,fc = 'green')
ax5 = fig.add_subplot(8,2,16,fc = 'orange')
```



Q. Commands on dataset : head,tail,dtypes,heatmap,corr,isnull,sqft_living vs price.

CODE:

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

```
#data = pd.read_csv(r"C:\Users\sam\Desktop\online notes\Megha mam\kc_house_data.csv")
data = pd.read_csv("kc_house_data.csv")
data.isnull().sum()

datatop = data.head()
print("Top of database is :\n ",datatop)

databottom = data.tail()
print("Bottom of database is \n: ",databottom)

datatypes = np.dtype(np.int16)
print("constructing a datatype \n: ",datatypes)

datacorr = data.corr()
print("Correlations \n",datacorr )

dataheat = sns.heatmap(datacorr)
print(dataheat)
plt.show()

plt.plot(data.sqft_living,data.price)
plt.show()
```

OUTPUT:

```
Top of database is :
           id            date    ...      sqft_living15  sqft_lot15
0  7129300520  20141013T000000    ...               1340        5650
1  6414100192  20141209T000000    ...               1690        7639
```

2 5631500400 20150225T000000 ... 2720 8062

3 2487200875 20141209T000000 ... 1360 5000

4 1954400510 20150218T000000 ... 1800 7503

[5 rows x 21 columns]

Bottom of database is

: id date ... sqft_living15 sqft_lot15

21608 263000018 20140521T000000 ... 1530 1509

21609 6600060120 20150223T000000 ... 1830 7200

21610 1523300141 20140623T000000 ... 1020 2007

21611 291310100 20150116T000000 ... 1410 1287

21612 1523300157 20141015T000000 ... 1020 1357

[5 rows x 21 columns]

constructing a datatype

: int16

Correlations

	id	price	...	sqft_living15	sqft_lot15
id	1.000000	-0.016762	...	-0.002901	-0.138798
price	-0.016762	1.000000	...	0.585379	0.082447
bedrooms	0.001286	0.308350	...	0.391638	0.029244
bathrooms	0.005160	0.525138	...	0.568634	0.087175
sqft_living	-0.012258	0.702035	...	0.756420	0.183286
sqft_lot	-0.132109	0.089661	...	0.144608	0.718557
floors	0.018525	0.256794	...	0.279885	-0.011269
waterfront	-0.002721	0.266369	...	0.086463	0.030703
view	0.011592	0.397293	...	0.280439	0.072575
condition	-0.023783	0.036362	...	-0.092824	-0.003406
grade	0.008130	0.667434	...	0.713202	0.119248
sqft_above	-0.010842	0.605567	...	0.731870	0.194050

sqft_basement -0.005151 0.323816 ... 0.200355 0.017276

yr_built 0.021380 0.054012 ... 0.326229 0.070958

yr_renovated -0.016907 0.126434 ... -0.002673 0.007854

zipcode -0.008224 -0.053203 ... -0.279033 -0.147221

lat -0.001891 0.307003 ... 0.048858 -0.086419

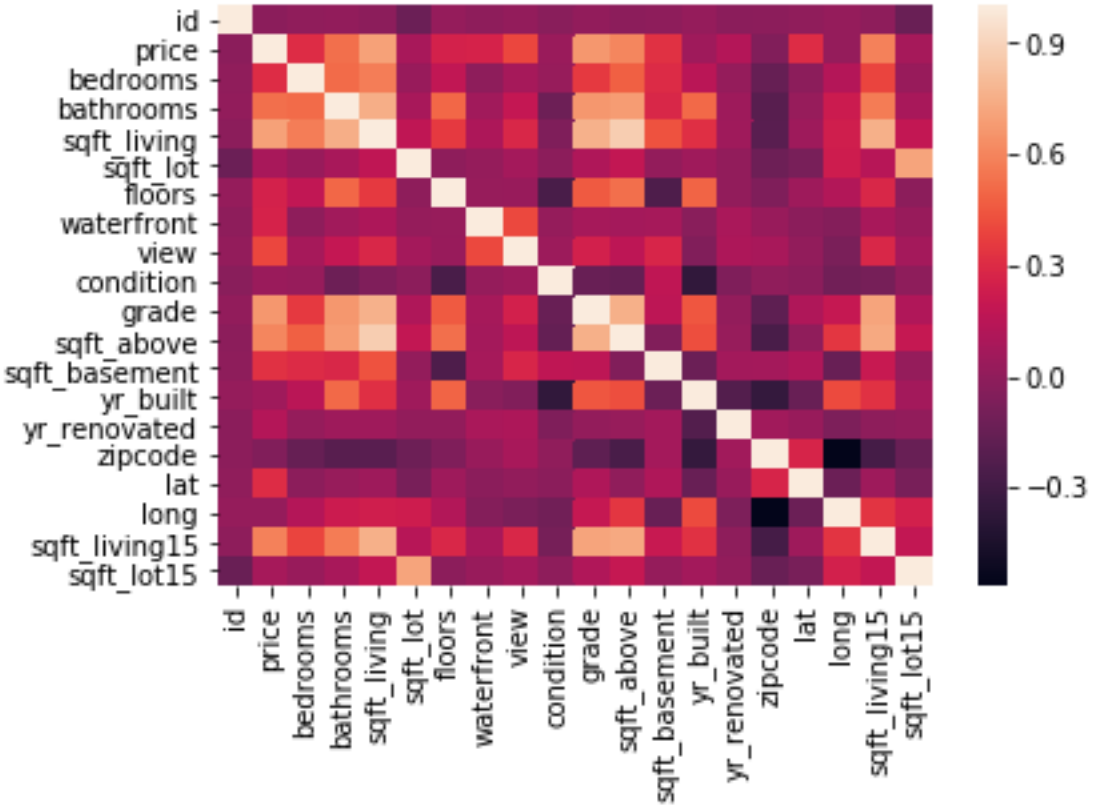
long 0.020799 0.021626 ... 0.334605 0.254451

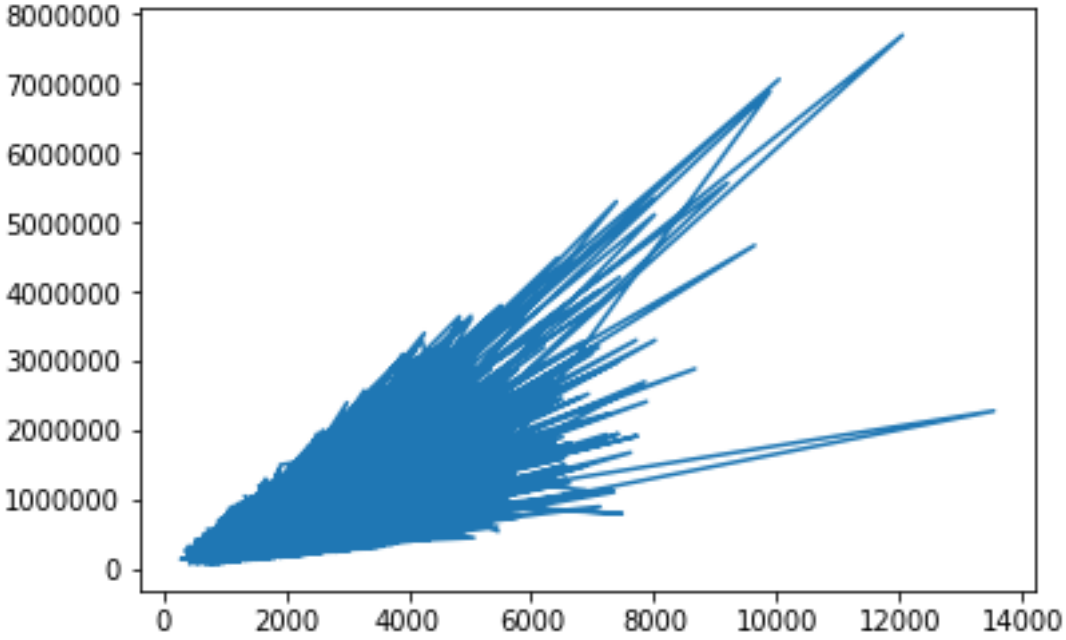
sqft_living15 -0.002901 0.585379 ... 1.000000 0.183192

sqft_lot15 -0.138798 0.082447 ... 0.183192 1.000000

[20 rows x 20 columns]

AxesSubplot(0.125,0.125;0.62x0.755)




8000000
7000000
6000000
5000000
4000000
3000000
2000000
1000000
0
0
2000
4000
6000
8000
10000
12000
14000