```
####   6  #######
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
%matplotlib inline
```

```
import pandas as pd
data = pd.read_csv(r"C:\Users\Abhay\anaconda3\salary_pred_data1.csv")
data.head()
```

	Level of city	Years of experiance	Age	Level of education	Job profile	Salary
0	2	11	34	4	3	43068
1	4	14	28	1	4	48856
2	1	13	55	3	2	41910
3	4	19	47	1	7	65494
4	2	10	24	2	6	55048

```
data.shape
```

```
(1000, 6)
```

```
data.describe()
```

	Level of city	Years of experiance	Age	Level of education	Job profile	Salary
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000
mean	2.503000	10.110000	39.110000	2.477000	4.933000	51630.020000
std	1.116798	5.466789	11.617007	1.104849	2.559504	14046.124478
min	1.000000	1.000000	20.000000	1.000000	1.000000	18586.000000
25%	2.000000	5.000000	29.000000	1.750000	3.000000	40681.000000
50%	2.000000	10.000000	39.000000	2.000000	5.000000	51780.000000
75%	4.000000	15.000000	49.000000	3.000000	7.000000	62940.000000
max	4.000000	19.000000	59.000000	4.000000	9.000000	85464.000000

```
x = data.iloc[:,:-1]
x.head()
```

	Level of city	Years of experiance	Age	Level of education	Job profile
0	2	11	34	4	3
1	4	14	28	1	4
2	1	13	55	3	2
3	4	19	47	1	7
4	2	10	24	2	6

```
y = data.iloc[:,5]
y.head()
```

```
0    43068
1    48856
2    41910
3    65494
4    55048
Name: Salary, dtype: int64
```

```
#Distribution of target variable
sns.kdeplot(data.Salary).set_title('Salary ($)')
plt.figure(figsize = (10, 3))
sns.boxplot(data.Salary)
```

```
c:\python\python38\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass 1
  warnings.warn(
<AxesSubplot:xlabel='Salary'>
```

```
#plt.rcParams['axes.facecolor'] = 'w'
data.plot(kind="scatter",x="Years of experiance",y="Salary",color="blue",figsize=(8,6))
plt.title("Years of Experience vs Salary")
plt.xlabel("Years of Experience")
plt.ylabel("Salary")
plt.show()
```

```
# Splitting the dataset into the Training set and Test set
# Allocating half of the dataset set train and the other half to test on the model
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 1/10, random_state =
```

```
# Fitting Simple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(x_train, y_train)
```

```
LinearRegression()
```

```
# Predicting the Test set results
y_pred = regressor.predict(x_test)
```

```
#Converting series to array
y_t=y_test.values
```

```
#Combining two arrays
y_final = y_t, np.round(y_pred,2)
y_final
```

```
(array([24312, 42378, 53306, 42746, 42494, 42052, 56560, 54362, 70470,
        39602, 65642, 28608, 49856, 51062, 43472, 64066, 33960, 35890,
        35790, 69360, 30142, 52792, 60906, 55004, 79446, 48460, 65862,
        30778, 39740, 48856, 18586, 27208, 38508, 33864, 55868, 79774,
        47494, 31962, 60766, 67044, 60752, 38918, 64810, 58244, 56206,
        56918, 51796, 36264, 57696, 49274, 51740, 25678, 66846, 51154,
        41916, 36188, 64142, 35778, 52384, 45590, 37568, 67004, 64668,
        67370, 72540, 79210, 48078, 60110, 44958, 58588, 37782, 67854,
        53550, 30542, 33808, 31772, 29716, 43486, 25006, 77216, 55274,
        34648, 40884, 72206, 78184, 27682, 34300, 70100, 61010, 63190,
        55500, 66110, 70948, 35064, 36662, 35556, 49194, 42452, 58784,
        52872], dtype=int64),
 array([31869.61, 39659.51, 46573.89, 34829.13, 37649.78, 43297.71,
        54500.67, 53507.54, 67925.01, 43394.23, 73861.12, 29363.9 ,
        52523.16, 56150.72, 34493.67, 54680.37, 38630.5 , 40368.58,
        32838.08, 68661.63, 36600.92, 60810.74, 59324.53, 61199.79,
        69233.79, 42656.27, 58781.77, 34337.26, 31022.83, 49640.94,
        29282.08, 34878.23, 47583.61, 32789.39, 54063.58, 71450.15,
        46091.57, 35802.68, 65085.29, 63901.33, 64024.35, 46138.35,
        72877.2 , 58816.4 , 59900.74, 56407.02, 42451.28, 39932.99,
        53919.33, 43386.13, 41726.43, 34660.04, 69324.79, 48348.66,
        35625.98, 40891.05, 70749.39, 33273.62, 54349.84, 36146.91,
        45226.58, 63158.6 , 58860.41, 63952.78, 64260.02, 74741.03,
        54570.38, 57854.18, 46494.17, 54228.47, 42399.21, 65949.17,
        44595.05, 35536.23, 36213.86, 40349.65, 35571.37, 38004.17,
        34908.23, 69294.27, 51059.63, 43238.64, 50230.8 , 63747.79,
        73017.82, 36218.73, 35707.75, 72789.22, 59195.35, 57635.75,
        62715.39, 60401.33, 62618.02, 34036.79, 44063.47, 29362.57,
```

```
result = pd.DataFrame(list(y_final))
result
```

	0	1	2	3	4	5	6	7	
0	24312.00	42378.00	53306.00	42746.00	42494.00	42052.00	56560.00	54362.00	70470
1	31869.61	39659.51	46573.89	34829.13	37649.78	43297.71	54500.67	53507.54	67925

2 rows × 100 columns

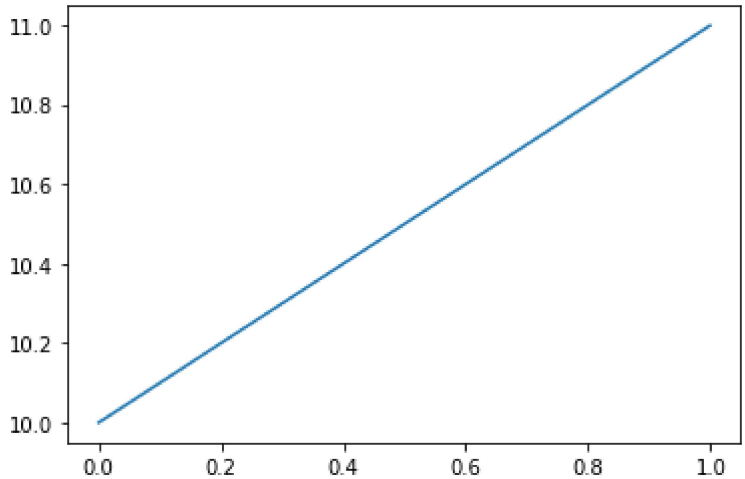
```
result = pd.DataFrame(list(y_final))
result = result.transpose()
result.columns = ["Actual Salary", "Predicted Salary"]
result.head(100)
```

	Actual Salary	Predicted Salary
0	24312.0	31869.61
1	42378.0	39659.51
2	53306.0	46573.89
3	42746.0	34829.13
4	42494.0	37649.78
95	35556.0	29362.57
97	42452.0	38117.80

```
#### 1a #####
```

```
import numpy
import matplotlib.pyplot as plt
x = numpy.arange(0, 1, 0.001)
y=[]
for i in x:
 a = 1*i + 10
 y.append(a)
plt.plot(x,y)
```

```
[<matplotlib.lines.Line2D at 0x2967f1ea760>]
```

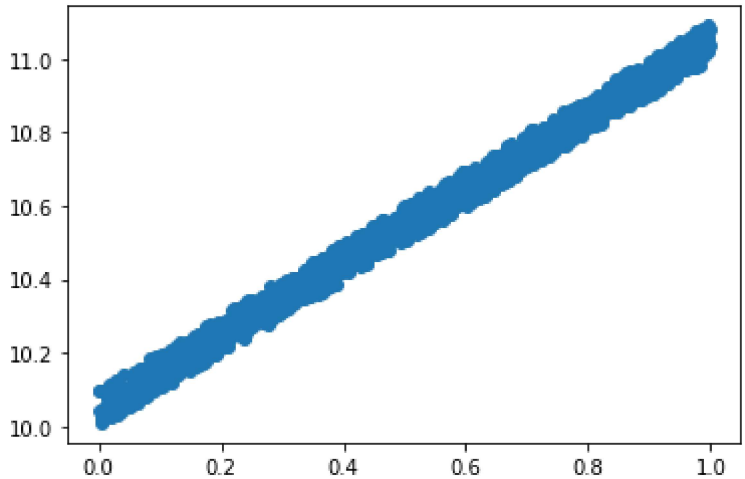



```
#### 1b #####
```

```
import numpy
import matplotlib.pyplot as plt
import random
x = numpy.arange(0, 1, 0.001)
```

```
y=[]
for i in x:
    a = 1*i + 10
    y.append(a)
r = []
for j in range(0, 1000):
    b=random.uniform(0, 1)
    r.append(b)
y1 =[]
for k in range(0, 1000):
    c=y[k] +(r[k]*0.1)
    y1.append(c)
plt.scatter(x, y1)
```

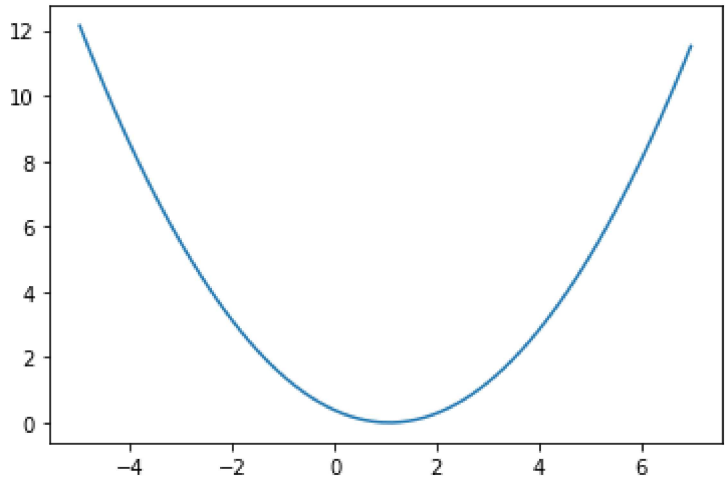
```
<matplotlib.collections.PathCollection at 0x2967f2e2d00>
```

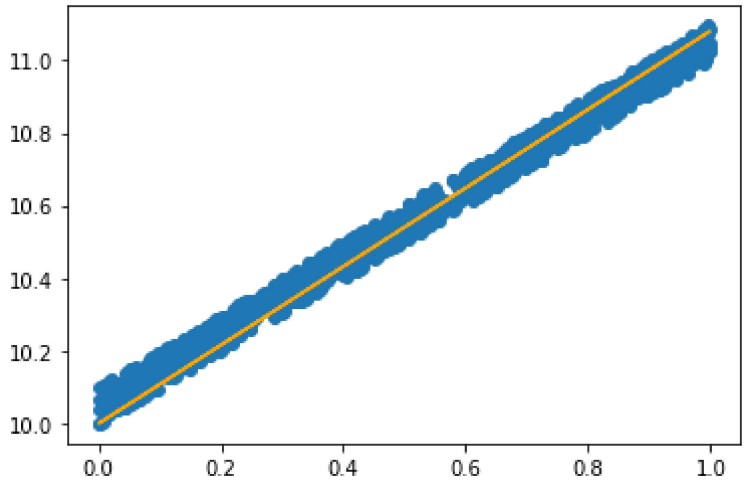



```
####   1c #####
```

```
import numpy as np
import random
import matplotlib.pyplot as plt
x=[]
N=1000
x = np.random.rand(N)
z = np.random.rand(N)

y_cor=x+10+0.1*z
w = []

for i in range(1000):

    k = random.uniform(-5, 7)
    w.append(k)
w.sort()
error=[]
for i in range(1000):
```

```
        y_pred=[]
        error.append(0)
        for j in range(1000):
            y_pred.append(w[i]*x[j]+10)
            error[i]+=(y_cor[j]-y_pred[j])**2
        error[i]/=1000

    plt.figure(1)
    plt.plot(w, error)

    min_in=error.index(min(error))

    y_best=[]
    for i in range(1000):
        y_best.append(w[min_in]*x[i] + 10)
    plt.figure(2)
    plt.scatter(x,y_cor)
    plt.plot(x,y_best, color='orange')
    plt.show()
```

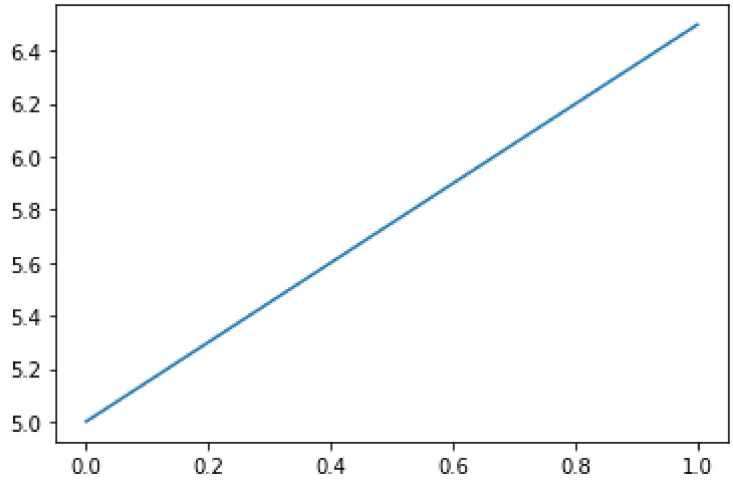





```
    #####    2a  ######


    import numpy as np
```

```
import matplotlib.pyplot as plt
x = np.arange(0, 1, 0.001)
y=[]
for i in x:
    a = 1.5*i + 5
    y.append(a)
plt.plot(x,y)
```

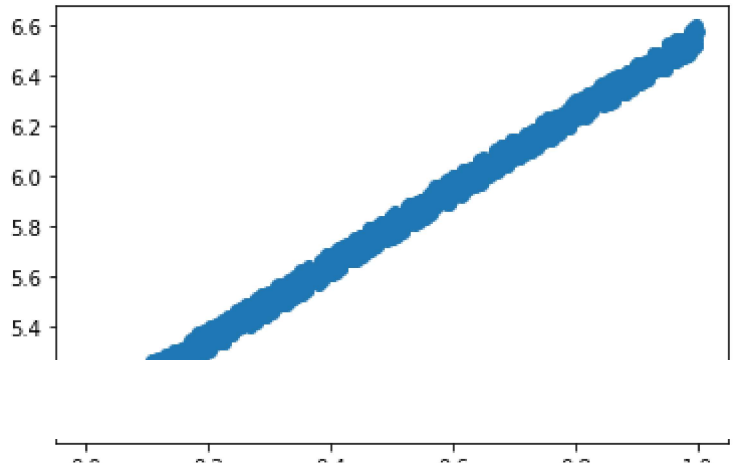
```
[<matplotlib.lines.Line2D at 0x2967f40b310>]
```




```
#### 2b ####
```

```
import numpy
import matplotlib.pyplot as plt
import random
x = numpy.arange(0, 1, 0.001)
y=[]
for i in x:
    a = 1.5*i + 5
    y.append(a)
r = []
for j in range(0, 1000):
    b=random.uniform(0, 1)
    r.append(b)
y1 =[]
for k in range(0, 1000):
    c=y[k] +(r[k]*0.1)
    y1.append(c)

plt.scatter(x, y1)
```

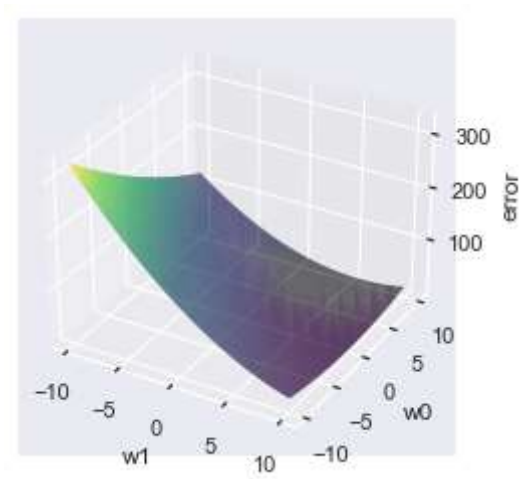
```
<matplotlib.collections.PathCollection at 0x2967f549610>
```




```
####  2c ####
```

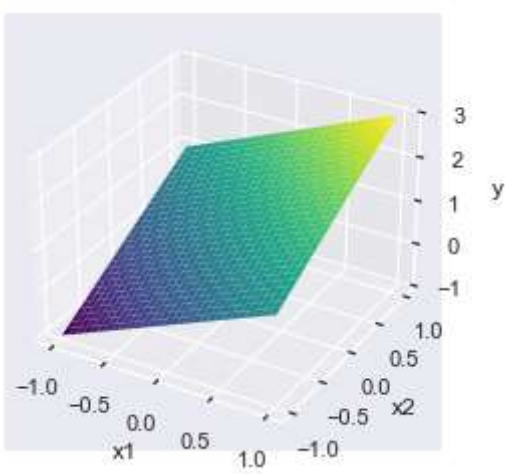
```
import numpy as np
import matplotlib.pyplot as plt
import random
import pandas as pd
from mpl_toolkits.mplot3d.axes3d import Axes3D
import seaborn as sns
%matplotlib inline
sns.set()

x = np.arange(0, 1, 0.001)
y=[]
for i in x:
    a = 1*i + 10
    y.append(a)
r = []
for j in range(0, 1000):
    b=random.uniform(0, 1)
    r.append(b)
y1 =[]
for k in range(0, 1000):
    c=y[k] +(r[k]*0.1)
    y1.append(c)
def J(w0, w1, x, y1):
    J = 0
    for i in range(1000):
        J += ((w0 + w1*x[i]) - y1[i] )**2
    return J/2000
fig = plt.figure()
ax = fig.add_subplot(1,1,1,projection='3d')
w0 = np.linspace(-10,10,1000)
w1 = np.linspace(-10,10,1000)
aa0, aa1 = np.meshgrid(w0, w1)
ax.plot_surface(aa0, aa1, J(aa0,aa1,x,y1), rstride=1, cstride=1,cmap='viridis', edgecolor=
ax.set_xlabel('w1')
ax.set_ylabel('w0')
ax.set_zlabel('error')
```

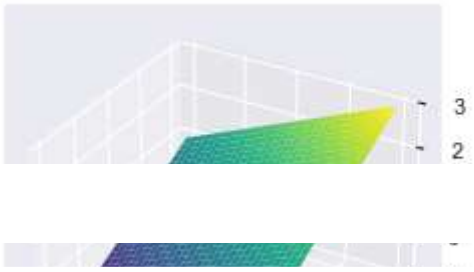
```
plt.show()
```




```
####  3a #####


import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d.axes3d import Axes3D
import random

import numpy
import pandas as pd
from mpl_toolkits.mplot3d.axes3d import Axes3D
import seaborn as sns
# %matplotlib inline


x1 = np.linspace(-1, 1, 30)
x2 = np.linspace(-1, 1, 30)
w0=1
w1=1
w2=1
p=[]
X1,X2 = np.meshgrid(x1, x2)
y = 1 + X1 + X2
for i in x1:

    for j in x2:

        a = 1 + i*w1 + j*w2
        p.append(a)
errorp = []
for i in range(900) :
    errorp.append(p[i]+random.random()*0.1)
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_surface(X1, X2, y, rstride=1, cstride=1,cmap='viridis', edgecolor='none')
ax.set_xlabel('x1')
```

```
ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('y')
plt.show()

x1 = np.linspace(-1, 1, 30)
x2 = np.linspace(-1, 1, 30)
w0=1
w1=1
w2=1
p=[]
X1,X2 = np.meshgrid(x1, x2)
y = 1 + X1 + X2
for i in x1:
    for j in x2:

        a = 1 + i*w1 + j*w2
        p.append(a)
errorp = []
for i in range(900) :
    errorp.append(p[i]+random.random()*0.1)
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_surface(X1, X2, y, rstride=1, cstride=1,cmap='viridis', edgecolor='none')
ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('y')
plt.show()
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.scatter(X1, X2, errorp,'.')
plt.show()
```

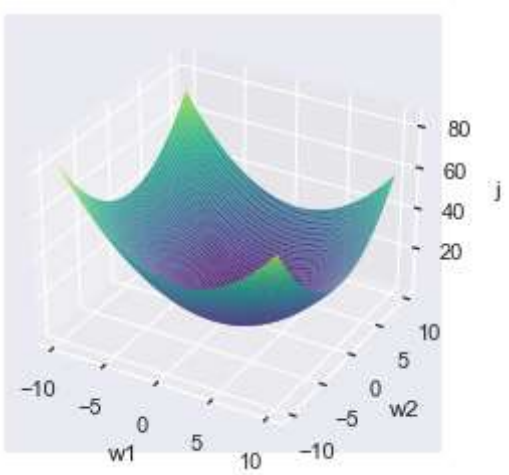


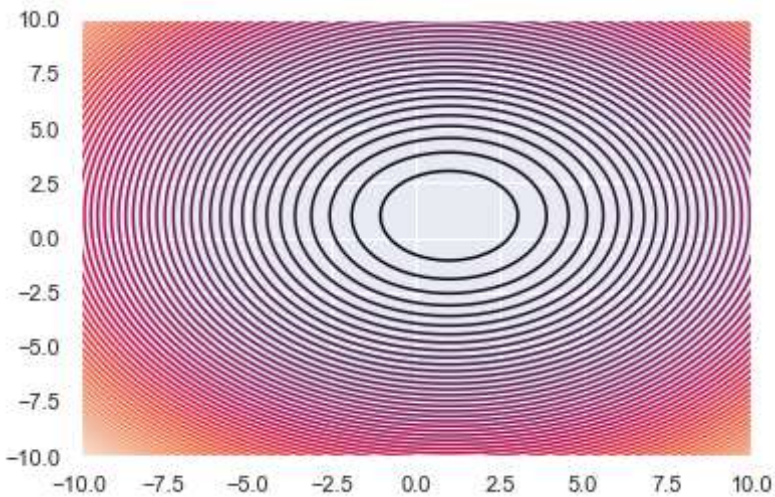



```
###  3b #####
```

```
sns.set()
x1 = np.linspace(-1, 1, 30)
x2 = np.linspace(-1, 1, 30)
w0=1
w1=1
w2=1
p=[]
for i in x1:
    for j in x2:
        a = 1 + i*w1 + j*w2
        p.append(a)
y1 = []
for i in range(900) :
    y1.append(p[i]+random.random()*0.1)
def J(w1, w2,x1,y1):
    J = 0
    p=[]
    for i in x1:
        for j in x2:
            a = 1 + i*w1 + j*w2
            p.append(a)
    for i in range(900):
        J += (p[i] - y1[i] )**2
    return J/900
fig = plt.figure()
ax = fig.add_subplot(1,1,1,projection='3d')
w1 = np.linspace(-10,10,100)
w2 = np.linspace(-10,10,100)
aa0, aa1 = np.meshgrid(w1, w2)
ax.plot_surface(aa0, aa1, J(aa0,aa1,x1,y1), rstride=1, cstride=1,cmap='viridis', edgecolor
ax.set_xlabel('w1')
```

```
ax.set_xlabel('w1')
ax.set_ylabel('w2')
ax.set_zlabel('j')
plt.show()
plt.contour(w1,w2,J(aa0,aa1,x1,y1),60)
w3=w1
w4=w2
plt.show()
```






```
####   3c #####


y = w0 + 1*X1 + 2*X2
print(y.shape)
# y = y.reshape(30,1)

# write your code here
xnew = [X1.reshape(900,1),X2.reshape(900,1)]
xnew = np.array(xnew).reshape(900,2)
ynew = y.reshape(900,1)
# xnew = xnew.T
print(xnew.shape)
w = np.array([5.,-6.,-4.])
eps = 0.00001
```

```
lr = 0.1
indexs = [1,2]
# w,w_gds,error1,epoch =  gradient_descent(w,eps,xnew,ynew,lr,indexs)
num=900
w = np.array([5.,-6.,-4.])
eps = 0.00001
lr = 0.1
X1 = X1.reshape(900,1)
X2 = X2.reshape(900,1)
y = w[0] + 1*X1 +2*X2
error1 = 1000001.
error2 = 1000000.
error_gd=[]
w1=[]
w2=[]
epoch=0
while abs(error1-error2)>eps:
    epoch+=1
   # print(error1)
    y_pred = w[0] + w[1]*X1 + w[2]*X2
    w1.append(w[1])
    w2.append(w[2])
    error1 = np.sum((y-y_pred)**2)/num
    error_gd.append(error1)
    del_error_1 = -(np.sum(np.dot((y-y_pred).T,X1)))/num
    del_error_2 = -(np.sum(np.dot((y-y_pred).T,X2)))/num
    w[1] = w[1] - lr * del_error_1
    w[2] = w[2] - lr * del_error_2
   # print(w)
    y_pred = w[0] + w[1]*X1 + w[2]*X2
    error2 = np.sum((y-y_pred)**2)/num
print(w)
print(epoch)
plt.contour(w3,w4,J(aa0,aa1,x1,y1),60)
# plt.show()
plt.plot(w1, w2, 'black')
plt.plot(w[1],w[2], 'orange', marker = 'X')
plt.show()
```

```
(30, 30)
(900, 2)
[5          0.98585506 1.98787576]
```

```
###   4  ####
```

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

class regression:
   # Constructor
   def __init__(self, name='reg'):
       self.name = name # Create an instance variable
   # def f(x):
#      return 1/x
   def grad_update(self,w_old,lr,y,x):
       # write your code here
       w = w_old + (2*lr)*(x@(y-(x.T@w_old)))/(y.shape[0])
       return w
   def error(self,w,y,x):
       return (np.sum(y - (x.T@w)))/(y.shape[0])# write your code here
   def mat_inv(self,y,x_aug):
       return (np.linalg.pinv(x_aug@x_aug.T))@(x_aug@y)# write your code here
       # by Gradient descent
   def Regression_grad_des(self,x,y,lr):
       # write your code here
       eps = 0.000001
       w_old = np.random.rand(x.shape[0],1)
       error1 = 100001.
       error2 = 100000.
       err = []
       while (error1 - error2)>eps:
           error1 = self.error(w_old,y,x)
           w_old = self.grad_update(w_old,lr,y,x)
           error2 = self.error(w_old,y,x)
           err.append(error1)
       w_pred = w_old
       return w_pred,err
##################################################################################
############################
# Generation of data
sim_dim=5
sim_no_data=1000
x=np.random.uniform(-1,1,(sim_dim,sim_no_data))
# print("1")
print(x.shape)
# print("1")
w=np.array([[1],[2],[3],[5],[9],[3]]) # W=[w0,w1,.....,wM]'
# print("2")
print(w.shape)
# print("2")
# # augment feat
x_aug=np.concatenate((np.ones((1,x.shape[1])), x),axis=0)
```

```
x_aug=np.concatenate((np.ones((1,x.shape[1])), x),axis=0)
# print("3")
print(x_aug.shape)
# print("3")
y=x_aug.T @ w # vector multiplication
# print("4")
print(y.shape)
# print("4")
## corrupted by noise
nois=np.random.uniform(0,1,y.shape)
y=y+0.1*nois
### the data (x_aug and y is generated)#####
##############################################################################
#################################
# by computation (Normal equation)
reg=regression()
w_opt=reg.mat_inv(y,x_aug)
# print("5")
print(w_opt)
# print("5")
# by Gradien descent
lr=0.01
w_pred,err=reg.Regression_grad_des(x_aug,y,lr)
# print("6")
print(w_pred)
# print("6")
plt.plot(err)

fx_name = r'$f(x)=\frac{1}{x}$'

x=np.setdiff1d(np.linspace(0.35,0,100),[0]) #to remove the zero
# y=f(x)
# plt.plot(x, y, label=fx_name)
# plt.legend(loc='upper left')
# plt.show()
```

```
(5, 1000)
(6, 1)
(6, 1000)
(1000, 1)
[[1.04998397]
 [2.0007265 ]
 [3.00021665]
 [4.99952871]
 [9.00004784]
 [3.00054264]]
[[1.0494223 ]


 [4.98691674]
```

```
#####   5 #######
```

```
import numpy as np
import numpy
import matplotlib.pyplot as plt
import random
x = np.linspace(-6,6,100)
x = np.array(x)
y=[]
for i in range(0,100):
    y.append((0.25*pow(x[i],3)) + (1.25*pow(x[i],2)) - 3*x[i] - 3)

r = []
for j in range(0, 100):
    b=random.uniform(0, 1)
    r.append(b)
y1 =[]
for k in range(0, 100):
    c=y[k] +(r[k]*5)
    y1.append(c)

plt.scatter(x,y1,color='orange')
x1 = np.ones((100,1))
#x = np.reshape(x,(10,1))
#x = np.append(x,axis=1)
x_transpose = np.transpose(x1)
x_transpose_dot_x = x_transpose.dot(x1)

temp_1 = np.linalg.inv(x_transpose_dot_x)

temp_2=x_transpose.dot(y1)

theta =temp_1.dot(temp_2)

print(theta)

y = 17.67+0*x

plt.plot(x,y,color='green')
plt.show()
#### for degree=1-----------------------------------------------------------------------
```

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

x = np.linspace(-6,6,100)
x = np.array(x)

y=[]
for i in range(0,100):
    y.append((0.25*pow(x[i],3)) + (1.25*pow(x[i],2)) - 3*x[i] - 3)

r = []
for j in range(0, 100):
    b=random.uniform(0, 1)
    r.append(b)

y1 =[]
for k in range(0, 100):
    c=y[k] +(r[k]*5)
    y1.append(c)

plt.scatter(x,y1,color='orange')

x_bias = np.ones((100,1))

x1 = np.reshape(x,(100,1))

x_new = np.append(x_bias,x1,axis=1)

x_new_transpose = np.transpose(x_new)
x_new_transpose_dot_x_new = x_new_transpose.dot(x_new)

temp_1 = np.linalg.inv(x_new_transpose_dot_x_new)

temp_2=x_new_transpose.dot(y1)

theta =temp_1.dot(temp_2)

print(theta)

ynew = 18.67+3.6*x

plt.plot(x,ynew,color='green')
plt.show()



#### for degree=2-----------------------------------------------------------------------
import numpy as np
import numpy
import matplotlib.pyplot as plt
import random

x = np.linspace(-6,6,100)
```

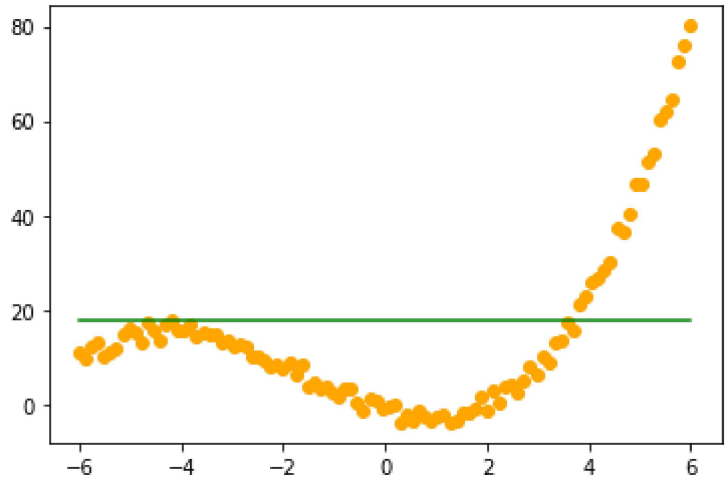
```
x = np.array(x)
z = np.linspace(0,1,100)
z = np.array(z)

y=[]
for i in range(0,100):
    y.append((0.25*pow(x[i],3)) + (1.25*pow(x[i],2)) - 3*x[i] - 3)

r = []
for j in range(0, 100):
    b=random.uniform(0, 1)
    r.append(b)

y1 =[]
for k in range(0, 100):
    c=y[k] +(r[k]*5)
    y1.append(c)

x2=[]
for i in range(0,100):
    x2.append(pow(x[i],2))

plt.scatter(x,y1,color='orange')

x_bias = np.ones((100,1))
x1 = np.reshape(x,(100,1))
x3 = np.reshape(x2,(100,1))

x_new = np.append(x_bias,x1,axis=1)
x_new = np.append(x_new,x3,axis=1)

x_new_transpose = np.transpose(x_new)
x_new_transpose_dot_x_new = x_new_transpose.dot(x_new)

temp_1 = np.linalg.inv(x_new_transpose_dot_x_new)
temp_2=x_new_transpose.dot(y1)

theta =temp_1.dot(temp_2)

print(theta)

ynew = (1.24*pow(x,2))+2.54*x-0.80

plt.plot(x,ynew,color='green')
plt.show()

#### for degree=3-----------------------------------------------------------------------
import numpy as np
import numpy
import matplotlib.pyplot as plt
import random
x = np.linspace(-6,6,100)
x = np.array(x)

y=[]
```

```
for i in range(0,100):
    y.append((0.25*pow(x[i],3)) + (1.25*pow(x[i],2)) - 3*x[i] - 3)

r = []
for j in range(0, 100):
    b=random.uniform(0, 1)
    r.append(b)

y1 =[]
for k in range(0, 100):
    c=y[k] +(r[k]*5)
    y1.append(c)

x2=[]
for i in range(0,100):
    x2.append(pow(x[i],2))

x4=[]
for i in range(0,100):
    x4.append(pow(x[i],3))

plt.scatter(x,y1,color='orange')

x_bias = np.ones((100,1))
x1 = np.reshape(x,(100,1))
x3 = np.reshape(x2,(100,1))
x5 = np.reshape(x4,(100,1))

x_new = np.append(x_bias,x1,axis=1)
x_new = np.append(x_new,x3,axis=1)
x_new = np.append(x_new,x5,axis=1)

x_new_transpose = np.transpose(x_new)
x_new_transpose_dot_x_new = x_new_transpose.dot(x_new)

temp_1 = np.linalg.inv(x_new_transpose_dot_x_new)
temp_2=x_new_transpose.dot(y1)

theta =temp_1.dot(temp_2)

print(theta)

ynew = (0.24*pow(x,3))+(1.25*pow(x,2))-2.97*x-0.64

plt.plot(x,ynew,color='green')
plt.show()

#### for degree=4-----------------------------------------------------------------------
import numpy as np
import numpy
import matplotlib.pyplot as plt
import random
x = np.linspace(-6,6,100)
x = np.array(x)
```

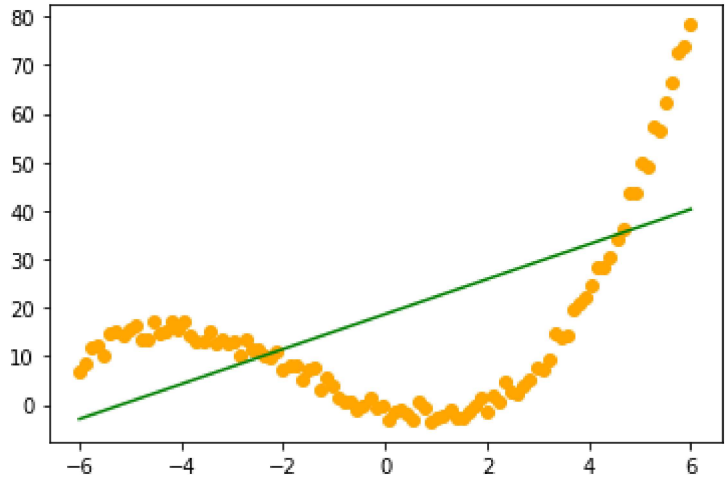
```
y=[]
for i in range(0,100):
    y.append((0.25*pow(x[i],3)) + (1.25*pow(x[i],2)) - 3*x[i] - 3)

r = []
for j in range(0, 100):
    b=random.uniform(0, 1)
    r.append(b)

y1 =[]
for k in range(0, 100):
    c=y[k] +(r[k]*5)
    y1.append(c)

x2=[]
for i in range(0,100):
    x2.append(pow(x[i],2))

x4=[]
for i in range(0,100):
    x4.append(pow(x[i],3))

x6=[]
for i in range(0,100):
    x6.append(pow(x[i],4))

plt.scatter(x,y1,color='orange')

x_bias = np.ones((100,1))
x1 = np.reshape(x,(100,1))
x3 = np.reshape(x2,(100,1))
x5 = np.reshape(x4,(100,1))
x7 = np.reshape(x6,(100,1))

x_new = np.append(x_bias,x1,axis=1)
x_new = np.append(x_new,x3,axis=1)
x_new = np.append(x_new,x5,axis=1)
x_new = np.append(x_new,x7,axis=1)

x_new_transpose = np.transpose(x_new)
x_new_transpose_dot_x_new = x_new_transpose.dot(x_new)

temp_1 = np.linalg.inv(x_new_transpose_dot_x_new)
temp_2=x_new_transpose.dot(y1)

theta =temp_1.dot(temp_2)

print(theta)

plt.plot(x,ynew,color='green')
plt.show()



# for degree 0 polynomial fitting
```

```
#degree=0
#X_1=data_transform(x,degree)
# print(X_1.shape)
#w_mat=reg.mat_inv(y,X_1)
# print(y.shape)
# print(w_mat.shape)
#y_pred=X_1.T @ w_mat
# print(y_pred.shape)
#plt.figure()
#plt.plot(x.T,y,'.')
#plt.plot(x.T,y_pred)
```

```
[14.54101895]
```

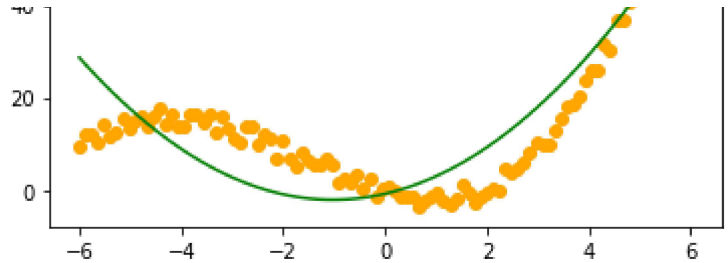



```
[14.52159884  2.45640624]
```

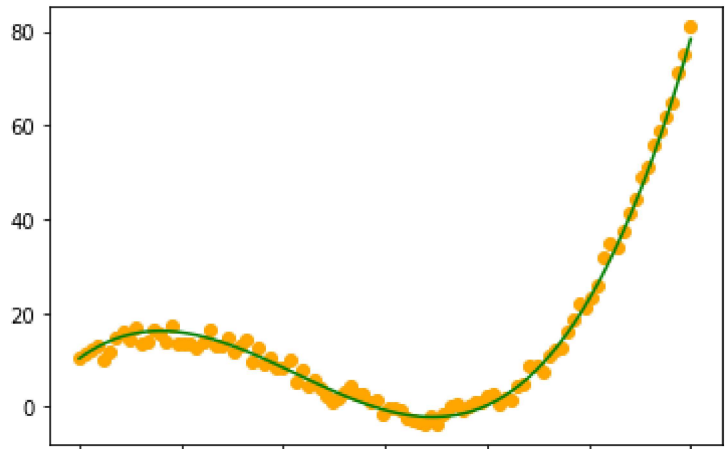


link to the google colab:

https://colab.research.google.com/drive/1ZuAgdd_9pEkJL4A2vflrFnS9l3_SlPFu?usp=sharing




```
[-0.48698395 -2.8404051   1.25183801  0.24205815]
```



```
     −6      −4      −2       0       2       4       6
⌈-7.07435928e-01 -2.88944252e+00  1.28208431e+00  2.47192670e-01
```