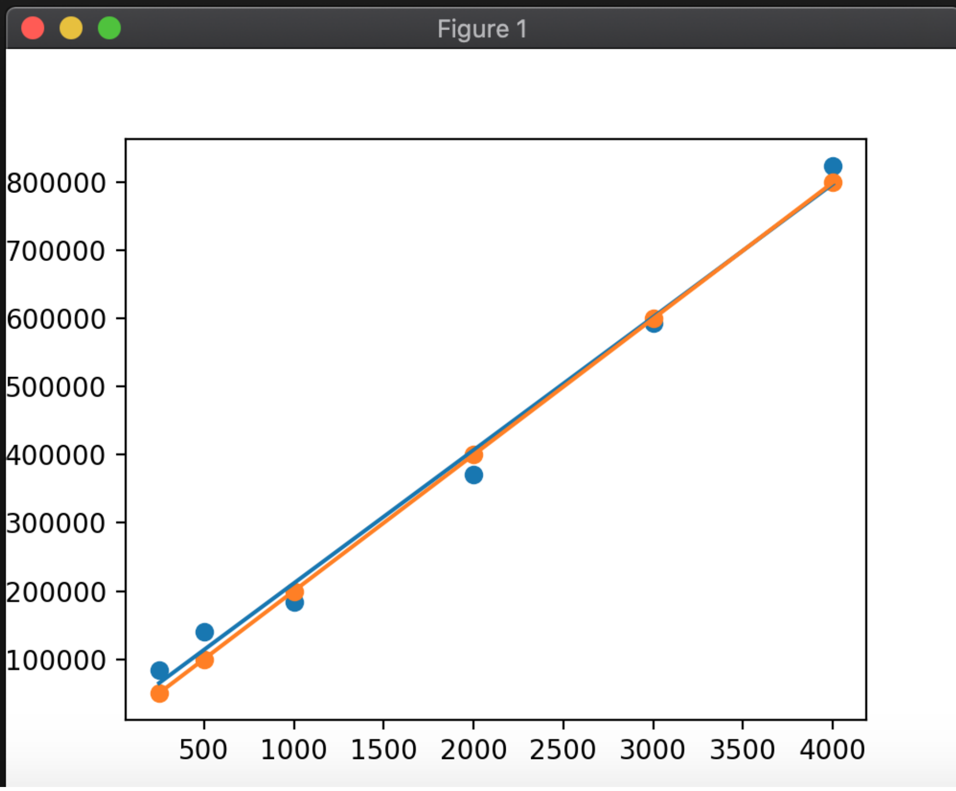
Raul Rodriguez

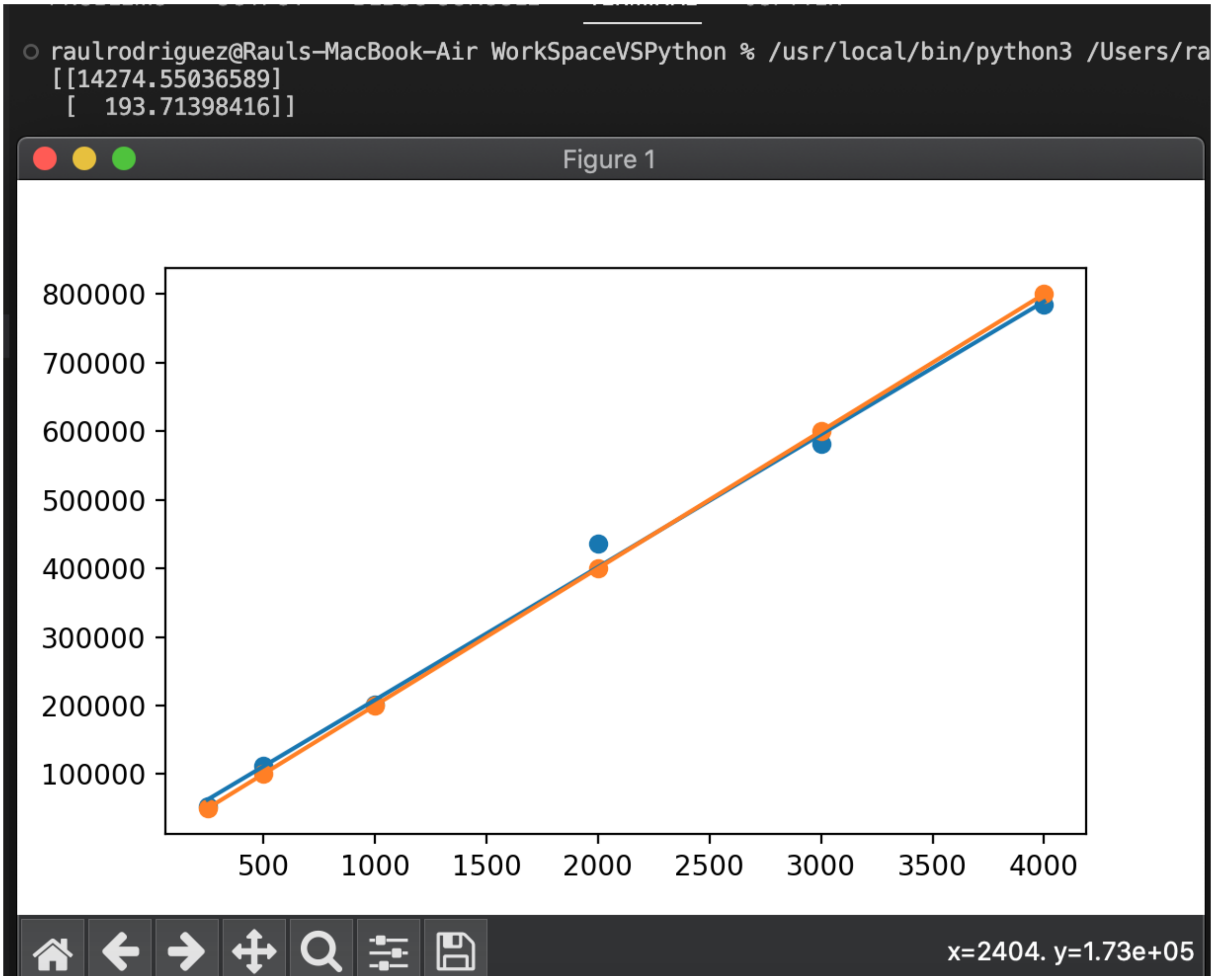
```python
'''Exercise 1
Given the following house square feet: 250, 500, 1000, 2000, 3000, 4000 and the following house
prices: 50000, 100000, 200000, 400000, 600000, 800000. Create noise derived from a standard nor-
mal (Gaussian) distribution function (i.e., one with 0 mean and 1 standard deviation). Multiply
the noise derived by 30000 and then add it to the dependent variable. Use Ordinary Least
Squares to find the regression line parameters, estimate the SSE (Sum of Squared Error), and
find the Correlation Coefficient. Plot the data points with and without noise, along with the
regression lines with and without noise, and predict house prices for the following square feet:
200, 1250, 2710, 5100
Note 1: The following line of code produces 6 numbers from a normal distribution with 0
mean and 1 standard deviation: randomNumbers = np.random.normal(0.0, 1.0, 6)
Note 2: Declare your arrays as: np.float64() instead of np.array or use e.g., np.array([250.]) in
at least one element so as to be treated as a float array and multiplying it with an int array will
also result in a float array
Note 3: Do not use any built-in functions, apart from np.mean(), to estimate the: regression
line, SSE, correlation coefficient; you can use them, though, for verification with your own results'''
import numpy as np
from matplotlib import pyplot as plt
from scipy import stats
x = np.float64([250, 500, 1000, 2000, 3000, 4000] )
y = np.float64([50000, 100000, 200000, 400000, 600000, 800000])
randomNumbers = np.random.normal(0.0, 1.0, 6)
noise=randomNumbers*30000
y2=y+noise
#w0=y-intercept, w1=slope
w1=((np.mean(x*y2))-(np.mean(x)*np.mean(y2)))/((np.mean(x**2))-(np.mean(x)**2))
w0=np.mean(y2)-(w1*np.mean(x))
noisemodel=w0+(w1*x)
SSE=sum((noisemodel-y2)**2)
r=(sum((x-np.mean(x))*(y2-np.mean(y2))))/((sum((x-np.mean(x))**2))*(sum((y-np.mean(y))**2)))**0.5
print(SSE)
print(r)
p=np.float64([200,1250,2710,5100])
prediction=w0+(w1*p)
print("Predection price is",prediction)
plt.scatter(x,y2)
plt.scatter(x,y)
plt.plot(x,noisemodel)
plt.plot(x,y)
plt.show()
```

```
/usr/local/bin/python3 /Users/raulrodriguez/Documents/WorkSpaceVSPython/HW9_1.py
raulrodriguez@Rauls-MacBook-Air WorkSpaceVSPython % /usr/local/bin/python3 /Users/raulrodri
3947551062.95107
0.9773919673956528
Predection price is [  54976.01217077  260228.32532386  545626.77980339 1012820.14021851]
```



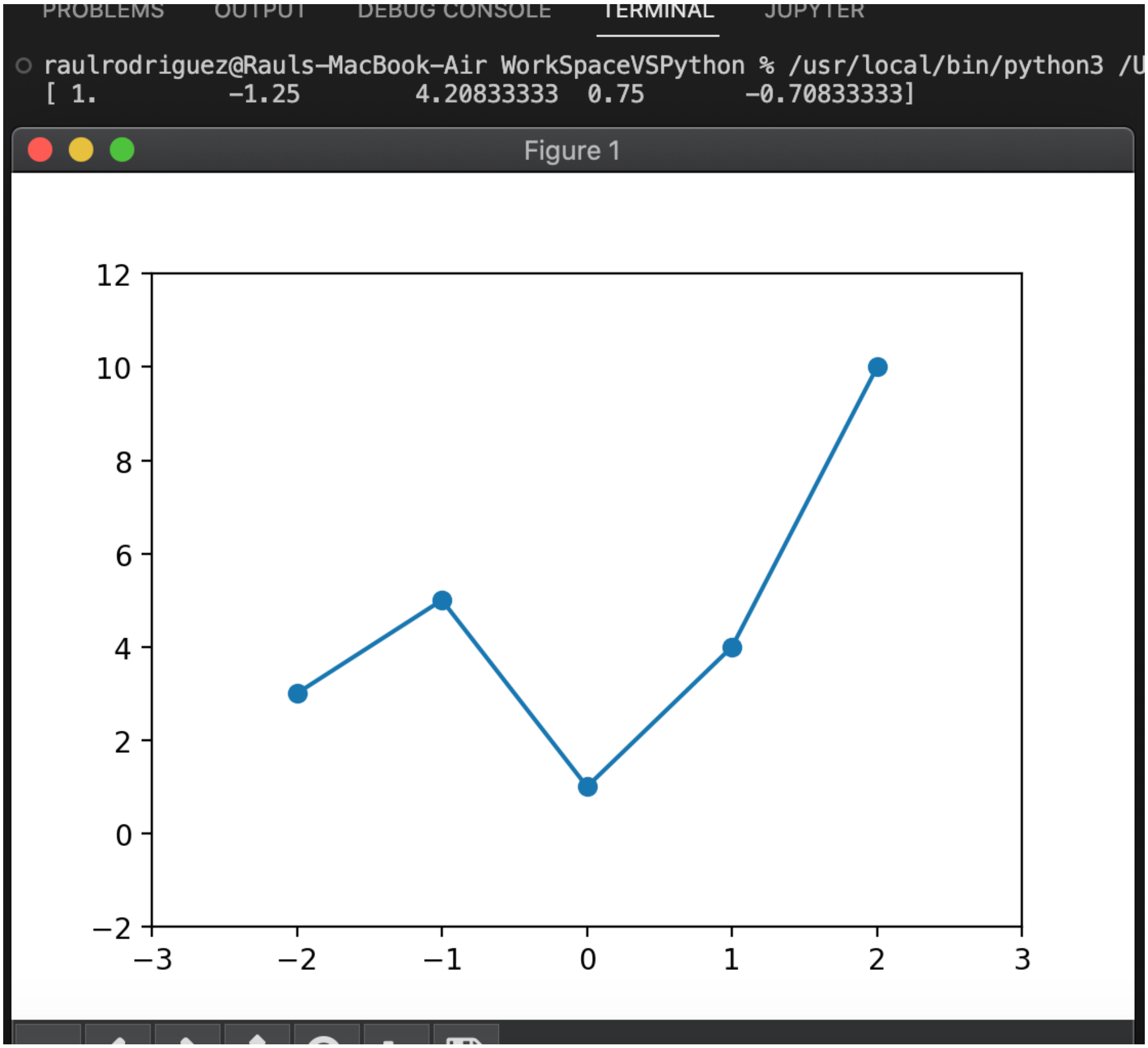
HW9_2.py > ...

```
'''Exercise 2
Based on Ex. 1, find the Least Squares Regression Line using the Matrix Algebra method.
Verify whether you get the same regression line parameters as in Ex. 1. Plot the data points, after
noise addition, along with the regression line
Note: You can use the transpose(), np.matmul(), and np.linalg.inv() functions to get the transpose,
multiply matrices, and get the inverse of a matrix, respectively'''
import numpy as np
from matplotlib import pyplot as plt
x = np.array([250, 500, 1000, 2000, 3000, 4000] )
y = np.array([50000, 100000, 200000, 400000, 600000, 800000])
randomNumbers = np.random.normal(0.0, 1.0, 6)
noise=randomNumbers*30000
y2=y+noise
n=len(x)
x1=np.ones((n),dtype=int).reshape(n,1)
x2=np.array(x).reshape(n,1)
xArr=np.hstack((x1,x2))
yArr=np.array(y2).reshape(n,1)
xArrT=xArr.transpose()
yArrT=yArr.transpose()
xMul=np.matmul(xArrT,xArr)
xyMul=np.matmul(xArrT,yArr)
xMulInv=np.linalg.inv(xMul)
A=np.matmul(xMulInv,xyMul)
print(A)
b=A[0]
m=A[1]
noisemodel=b+(m*x2)
plt.scatter(x,y2)
plt.scatter(x,y)
plt.plot(x,noisemodel)
plt.plot(x,y)
plt.show()
```


raulrodriguez@Rauls-MacBook-Air WorkSpaceVSPython % /usr/local/bin/python3 /Users/ra
[[14274.55036589]
 [193.71398416]]
Figure 1
800000
700000
600000
500000
400000
300000
200000
100000
500
1000
1500
2000
2500
3000
3500
4000
x=2404. y=1.73e+05

HW9_3.py > ...

```python
'''Exercise 3
Perform Polynomial Curve Fitting on the following dataset: (-2, 3), (-1, 5), (0, 1), (1, 4), (2,
10). Print the coefficients and plot the graph of the polynomial function as shown in the Figure
below
Note 1: Do not use any built-in functions such as polyfit() or polyval(). You can use the
functions np.matmul() and np.linalg.inv() to multiply matrices and get the inverse of a matrix,
respectively
Note 2: Your algorithm should be able to work with any dataset not just a dataset of 5 data
points'''
import numpy as np
from matplotlib import pyplot as plt
x=[-2,-1,0,1,2]
y=[3,5,1,4,10]
n=len(x)
xArr=np.ones((n,n),dtype=float)
for row in range(xArr.shape[0]):
    for col in range(xArr.shape[1]):
        if col==0:
            xArr[row][col]=1
        else:
            xArr[row][col]=x[row]**col
xArrInv=np.linalg.inv(xArr)
c=np.matmul(xArrInv,y)
print(c)
plt.scatter(x,y)
plt.plot(x,y)
plt.ylim(-2,12)
plt.xlim(-3,3)
plt.show()
```


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
JUPYTER
raulrodriguez@Rauls-MacBook-Air WorkSpaceVSPython % /usr/local/bin/python3 /U
[1.   -1.25   4.20833333 0.75   -0.70833333]
Figure 1
12
10
8
6
4
2
0
−2
−3
−2
−1
0
1
2
3