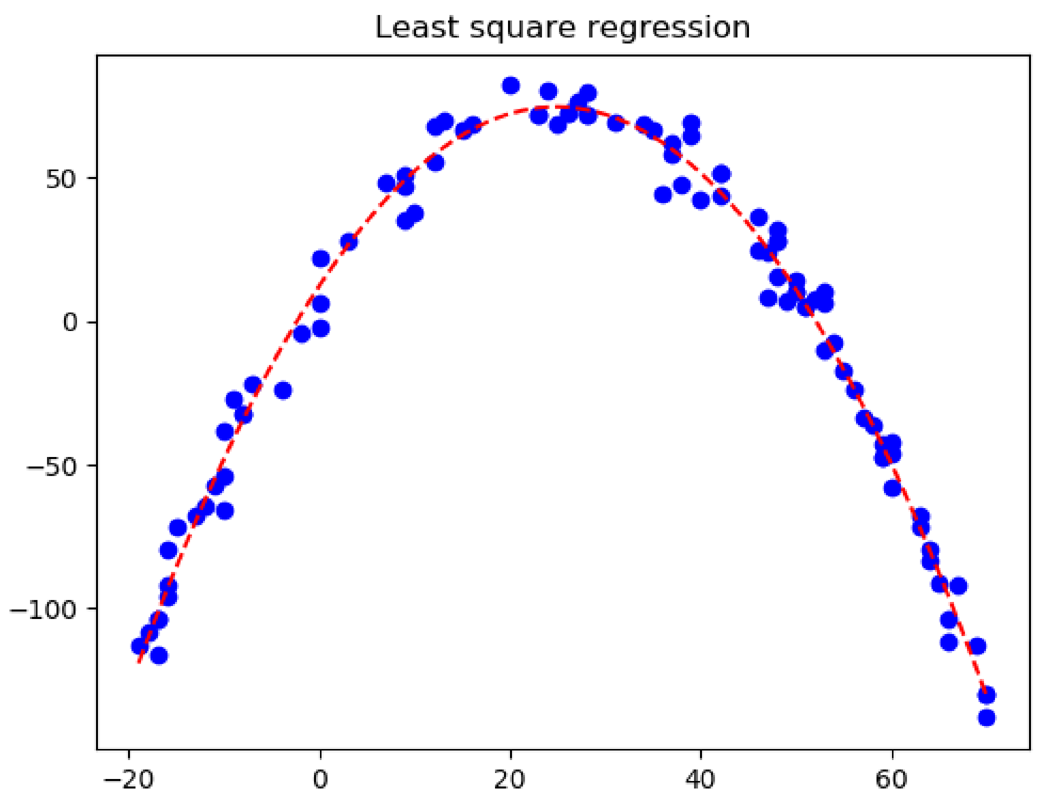
CS428 (Advanced Operating Systems) / MA500 (Geometric Foundations of Data Analysis – Classical Techniques) – Assignment 1

Name: Taidgh Murray

Student ID: 15315901

Part 1 Outputs:

Generated graph showcasing the data points in blue, and the line of best fit in red.



```
b0 values are [12.34270521  5.00610241 -0.10068145]

SSR/SSTo =  0.5234696113226226

1 - SSE/SSTo =  0.52772458233925
```

Both these values are = to R^2. For the most part they are similar values

Part 1 Code:

```python
#Student ID: 15315901
#Name: Taidgh Murray
#CS428/MA500 Homework 1

import math
import pylab
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

# Open the data file in a read format
f = open('data.txt', 'r')

# yi = b0+b1*xi+b2*xi^2+error(i)

# Initialise x & y arrays, along with error and beta value

Xs, Ys= [], []
b0 = [1.0, 1.0, 1.0]

def func(x, a, b, c):
    return a +  b*x + c*x**2


# For-loop
# Iterates through lines in file
# If the number is an x, it's added to the Xi array
# If the number is a y, it's added to the Yi Array
for l in f:
    # Split the lines up by their = sign
    # print(l)
    sp = l.split('=')

    # Splits the x value by its ',' character
    xVal= sp[1].split(',')

    # Adds corresponding x & y values in the line ot the arrays
    Xs.append(float(xVal[0]))
    Ys.append(float(sp[2]))

# Closes file
f.close()

# Changing X and Y data to numpy arrays
Xs = np.array(Xs)
Ys = np.array(Ys)
# Creating an arbitrary sigma array filled with ones to indicate error values
sigma = np.ones((100), dtype=float)


```

```python
# Calculating the b0, b1 & b2 values
linear = np.linspace(Xs.min(), Xs.max(), 100)
B, _ = curve_fit(func, Xs, Ys, b0, sigma)
y = func(linear, *B)


# Plotting & drawing the graph

#Plotting the points
plt.plot(Xs,Ys, 'bo', label='Data')
#Plotting the line of best fit
plt.plot(linear, y, 'r--', label='Fit')
plt.title('Least square regression')
#Drawing graph
plt.show()

print(B)

# Defining yHat - The fitted value
def yHat(i):
    return b0[0] + b0[1]*Xs[i]

# Defining yMean - The sample mean
yTotal = 0

for i in Ys:
    yTotal += i
yMean = (1/len(Ys))*(yTotal)

# Defining SSTO - The Total Sum of Squares
SSTO = 0
for i in Ys:
    SSTO += (i - yMean)**2

# Defining SSE - The Error Sum of Squares
SSE = 0
count = 0
for j in Ys:
    SSE += (j - (yHat(count)))**2
    count+=1

# Defining SSR - The Regression Sum of Squares
count = 0
SSR = 0
for k in Ys:
    SSR += ((yHat(count)) - yMean)**2
    count+=1

# Showcasing that r^2 = SSR/SSTO = (SSE/SSTO)-1 (Which, they largely are)
print('SSR/SSTo = ', SSR/SSTO)
print('1 - SSE/SSTo = ', (SSE/SSTO)-1)
```

Part 2 Outputs:

All I’m able to provide is the code, I didn’t have enough time to finish the assignment.

Part 2 Code:

```python
# Assignment 1 - Question 2

#Student ID: 15315901
#Name: Taidgh Murray
#CS428/MA500 Homework 1

"""
Gonna level with you here, I've attended all but two lectures so far, and I still have no idea
what's going on in this subject. Even using packages, I can't really get the answer I need for
the assignment. I also ran out of time to finish it. So, apologies for the mess.
"""

import math
import pylab
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit, leastsq
from scipy.stats import t

# Defining data as np arrays
Xi1=np.array([7, 18, 5, 14, 11, 5, 23, 9, 16, 5])
Xi2=np.array([5.11, 16.70, 3.20, 7.00, 11.00, 4.00, 22.10, 7.00, 10.60, 4.80])
Yi=np.array([58, 152, 41, 93, 101, 38, 203, 78, 117, 44])
b0=np.array([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0])

# Model definition
def model(Beta, Xi1, Xi2, Error, i):
    return Beta[0] + Beta[1]*Xi1[i] + Beta[2]*Xi2[i] + Error[i]

# Defining the least square estimator
def leastSquareEstimator(b,x, x2):
    return b[0] + b[1]*x[1] + b[2]*x2[2]

# Defining the least square estimator
def func(x, x2, a, b, c):
    return a + b*x + c*x2


# Calculating the b0, b1 & b2 values


# T value with 12 degrees of freedom & a significange level of 0.05
T = t.ppf(0.975, 12)
P = 3

#MSR = (1/p-1)(yTranspose*y - bTranspose * xTranspose * y)


# Defining yHat - The fitted value
def yHat(i):
    return b0[0] + b0[1]*Xi1[i]

# Defining yMean - The sample mean
yTotal = 0

for i in Yi:
    yTotal += i

yMean = (1/len(Yi))*(yTotal)

# Defining SSTO - The Total Sum of Squares
```

```
SSTO = 0
for i in Yi:
    SSTO += (i - yMean)**2

# Defining SSE - The Error Sum of Squares
SSE = 0
count = 0
for j in Yi:
    SSE += (j - float(yHat(count)))**2
    count+=1

# Defining SSR - The Regression Sum of Squares
count = 0
SSR = 0
for k in Yi:
    SSR += (float(yHat(count)) - yMean)**2
    count+=1

# Defining MSR
MSR = (1/P-1) * (Yi.transpose()*Yi - b0.transpose()*Xi1.transpose()*Yi)

MSE = SSE/len(Yi)-P
print(MSE)

FStar = MSR/MSE
print(FStar)
```