ECGR_4105_Assignment_1

February 23, 2024

Tyler Floyd

ID#: 801285803

Problem 1:

Let the first three columns of the data set be separate explanatory variables X1, X2, X3. Let the fourth column be the dependent variable Y. (Note: You cannot use the built-in function from ML libraries for gradient descent, you have to implement it yourself.)

Develop a code that runs linear regression with a gradient decent algorithm for each explanatory variable in isolation. In this case, you assume that in each iteration, only one explanatory variable (either X1, X2, or X3) is explaining the output. You need to do three different training, one per each explanatory variable. For the learning rate, explore different values between 0.1 and 0.01 (your choice). Initialize your parameters to zero (theta to zero).

Report the linear model you found for each explanatory variable. Plot the final regression model and loss over the iteration per each explanatory variable. Which explanatory variable has the lower loss (cost) for explaining the output (Y)? Based on your training observations, describe the impact of the different learning rates on the final loss and number of training iterations.

[877]:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

[878]:

```python
InpData=pd.read_csv('https://raw.githubusercontent.com/VoluSign/UNCC/main/D3.
↪csv')
InpData.head(8)#First eight rows of data
```

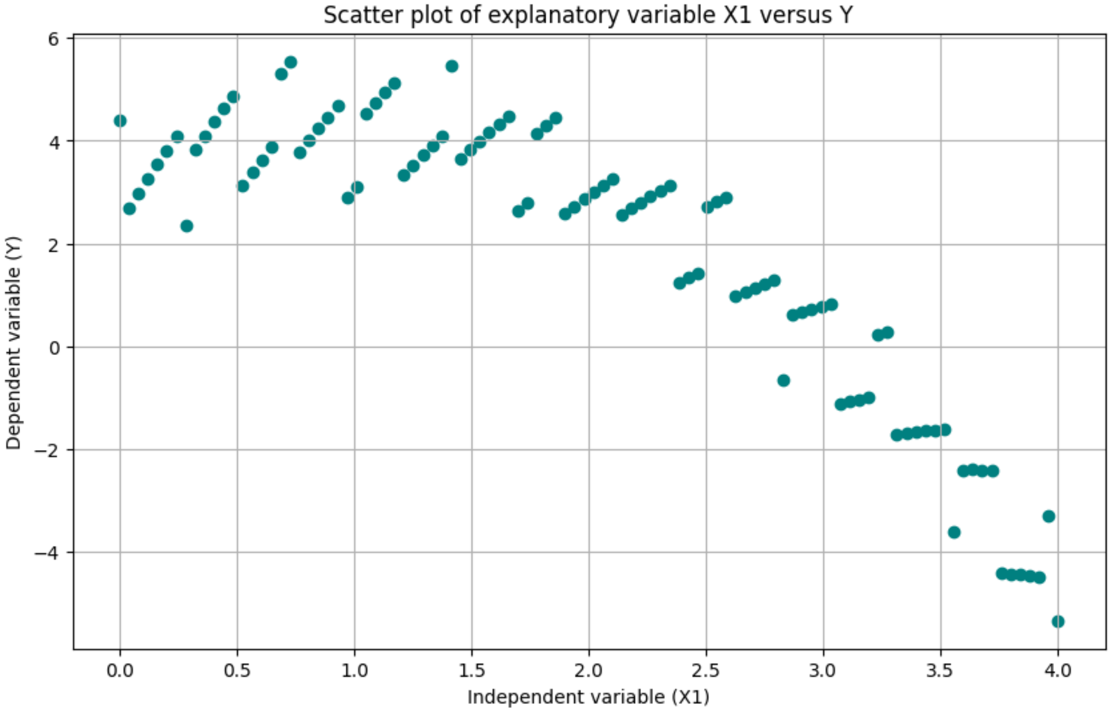
[878]:

```
         X1        X2        X3         Y
0  0.000000  3.440000  0.440000  4.387545
1  0.040404  0.134949  0.888485  2.679650
2  0.080808  0.829899  1.336970  2.968490
3  0.121212  1.524848  1.785455  3.254065
4  0.161616  2.219798  2.233939  3.536375
5  0.202020  2.914747  2.682424  3.815420
6  0.242424  3.609697  3.130909  4.091200
7  0.282828  0.304646  3.579394  2.363715
```

Ploting Explanitory Variable vs. Dependent Variable

```
[879]: plt.scatter(InpData.X1,InpData.Y, color='teal')
       plt.grid()
       plt.xlabel('Independent variable (X1)')
       plt.ylabel('Dependent variable (Y)')
       plt.title('Scatter plot of explanatory variable X1 versus Y')
```

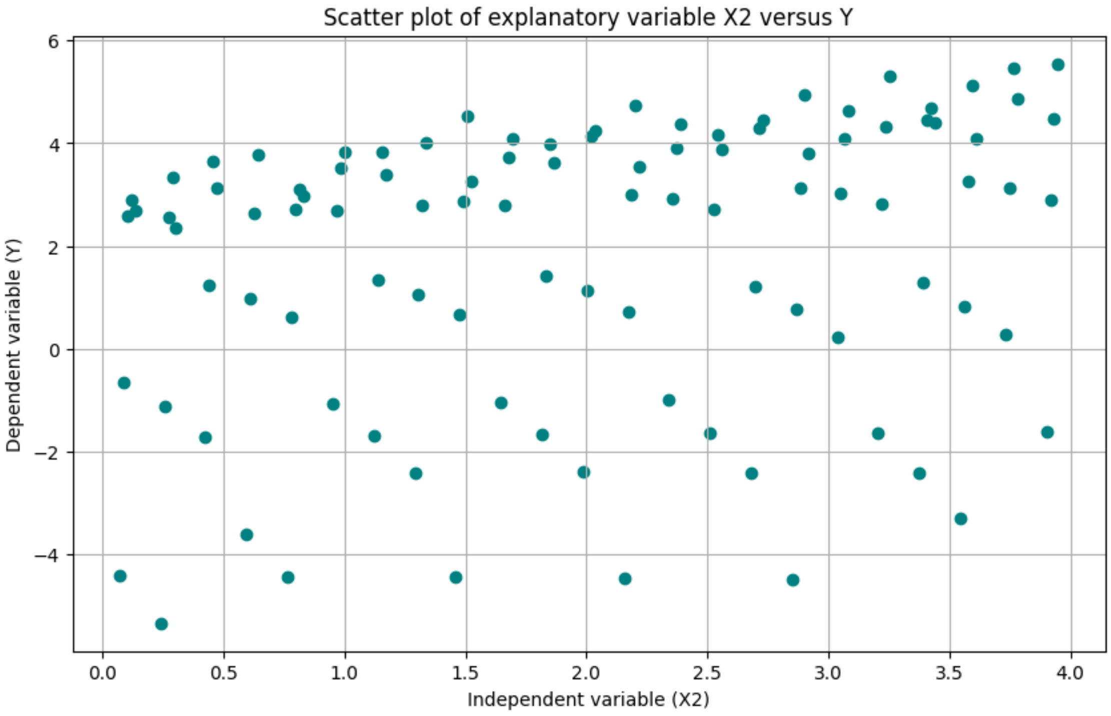
```
[879]: Text(0.5, 1.0, 'Scatter plot of explanatory variable X1 versus Y')
```



Note: Visable trend leans toward exponential or log function. Performing linear regression estimation.

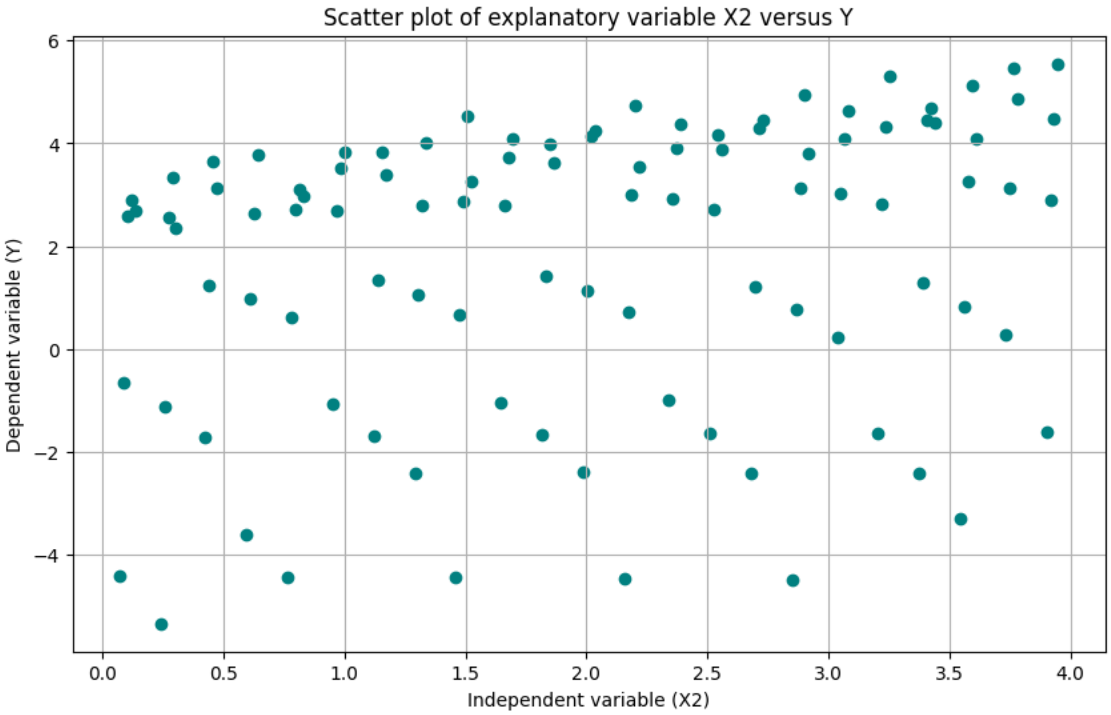
```
[880]: plt.scatter(InpData.X2,InpData.Y, color='teal')
       plt.grid()
       plt.xlabel('Independent variable (X2)')
       plt.ylabel('Dependent variable (Y)')
       plt.title('Scatter plot of explanatory variable X2 versus Y')
```

```
[880]: Text(0.5, 1.0, 'Scatter plot of explanatory variable X2 versus Y')
```



[881]:
```
plt.scatter(InpData.X2,InpData.Y, color='teal')
plt.grid()
plt.xlabel('Independent variable (X2)')
plt.ylabel('Dependent variable (Y)')
plt.title('Scatter plot of explanatory variable X2 versus Y')
```

[881]:
```
Text(0.5, 1.0, 'Scatter plot of explanatory variable X2 versus Y')
```



Cost Function

[882]:
```
#Cost function for linear regression
def computeCost(X, Y, theta):
  m = len(X)
  predictions = X.dot(theta)
  errors = np.subtract(predictions, Y)
  sqrErrors = np.square(errors)
  J = 1 / (2 * m) * np.sum(sqrErrors)
  return J
```

Gradiant Descent

[883]:
```
# Gradient descent function minimizes the cost function output to find the␣
 ↪optimal theta
def gradientDescent(X, y, theta, alpha, iterations):
  m = len(y)
  cost_history = np.zeros(iterations)

  for i in range(iterations):
      predictions = X.dot(theta)
      errors = predictions - y
      sum_delta = (alpha / m) * X.T.dot(errors)
      theta = theta - sum_delta
      cost_history[i] = computeCost(X, y, theta)
  return theta, cost_history
```

Plotting Functions

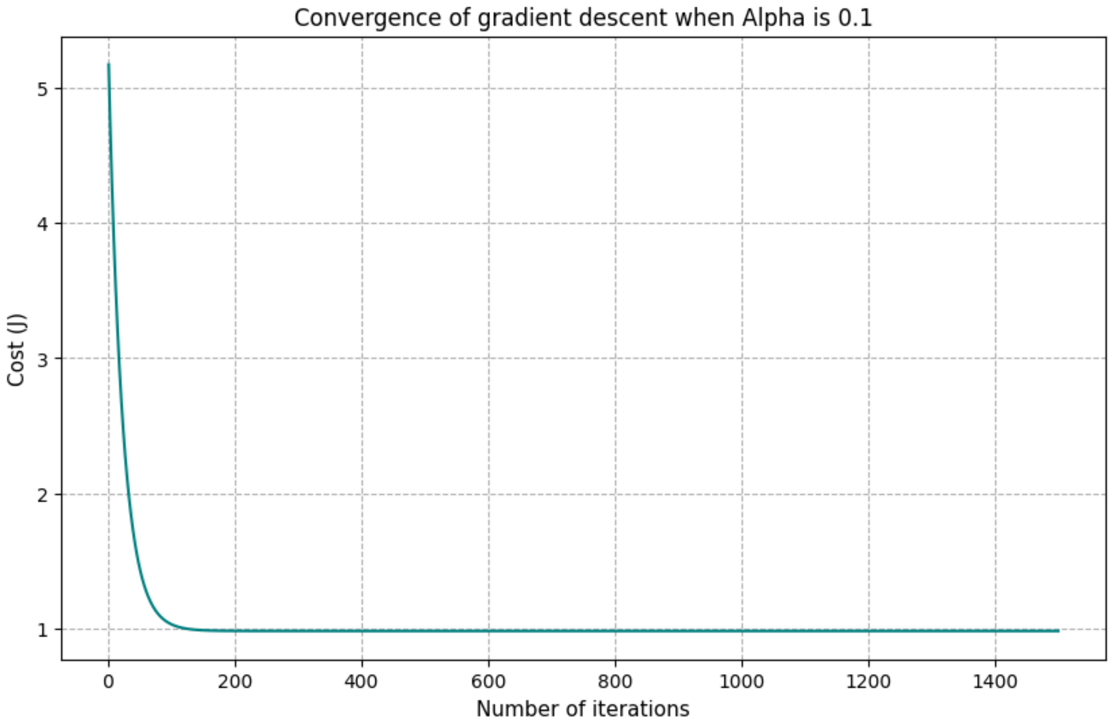
[884]:
```
# Plot the convergence of gradient descent with respect to the number of␣
 ↪iterations
def plotConvFigure(iterations, cost_history, alpha):
  plt.figure(figsize=(10, 6))
  plt.plot(range(1, iterations + 1), cost_history, color='teal')
  plt.grid(linestyle='--')
  plt.xlabel('Number of iterations', fontsize=11)
  plt.ylabel('Cost (J)', fontsize=11)
  plt.title('Convergence of gradient descent when Alpha is '+ str(alpha),␣
 ↪fontsize=12)
  plt.show()
```

[885]:
```
# Plot the linear regression fit
def plotLinReg(X, Y, theta):
  XInt = np.column_stack((np.ones(len(X)), X))
  plt.scatter(X, Y, color='red', marker='o', label='Input Data')
  plt.plot(X, XInt.dot(theta), color='teal', label='Linear Regression')
  plt.grid(linestyle='--')
  plt.xlabel('X', fontsize=11)
  plt.ylabel('Y', fontsize=11)
  plt.title('Linear Regression Fit', fontsize=12)
  plt.legend()
```

Explanitory Variable X1 in Isolation

[886]:
```
# Gradient descent for linear regression with explanatory variable X1
InpDataCopy=InpData.copy();#Keeps from altering original data
m = InpDataCopy.X1.size
X_0 = np.ones((m, 1))
X_0 = pd.DataFrame(X_0)
InpDataCopy.insert(0, "intercept",X_0)
X = InpDataCopy.loc[:,['intercept','X1']]
y = InpDataCopy.Y
Xarray = X.loc[:,['intercept','X1']].values
yarray = y.values
theta = np.zeros(2)
alpha=0.1;
iterations = 1500;
theta, cost_history = gradientDescent(Xarray, yarray, theta, alpha, iterations)
print('Final value of theta =', theta)
print("Final loss for X1:", cost_history[-1])
plotConvFigure(iterations, cost_history, alpha)
```

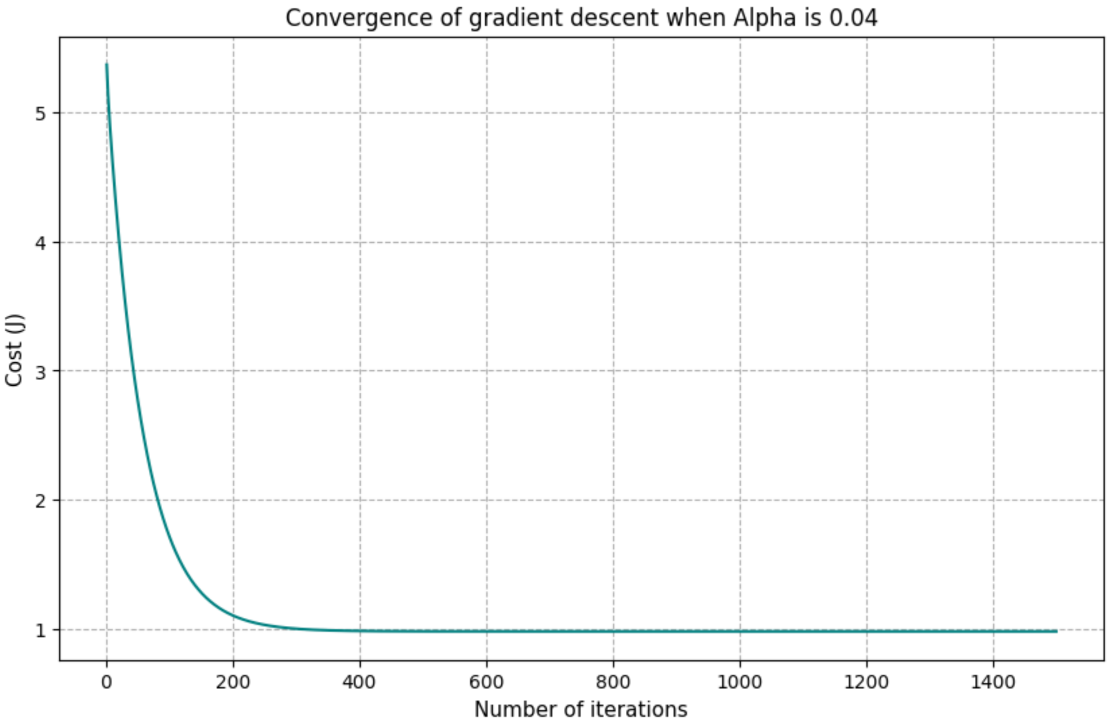
```
Final value of theta = [ 5.92794892 -2.03833663]
Final loss for X1: 0.9849930825405946
```



A high learning rate and could cause divergence.

[887]:
```python
# Perform linear regression for X1 with a different learning rate
theta = np.zeros(2)
alpha=.04;
theta, cost_history = gradientDescent(Xarray, yarray, theta, alpha, iterations)
print('Final value of theta =', theta)
print("Final loss for X1 :", cost_history[-1])
plotConvFigure(iterations, cost_history, alpha)
```

```
Final value of theta = [ 5.92793966 -2.03833303]
Final loss for X1 : 0.9849930825515203
```

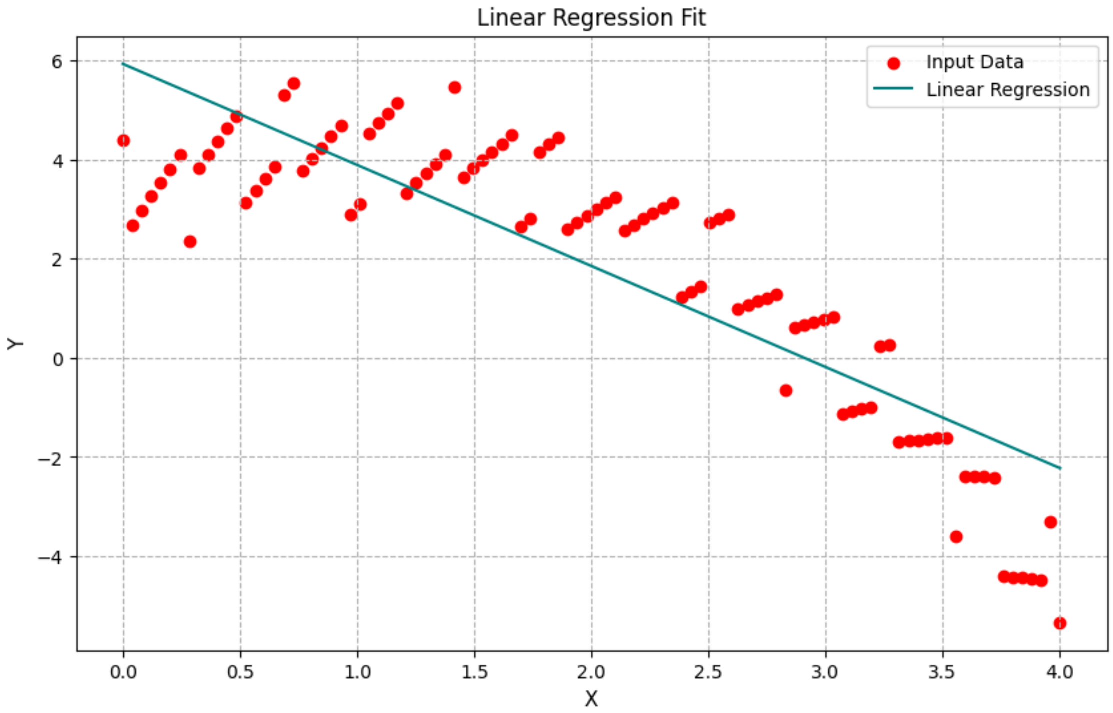


Note that its possible to do itteration to find the lowest loss for alpha

[888]:

```
# Plot linear regression for X1
plotLinReg(X.X1.values, InpDataCopy.Y.values, theta)
display(f'Final X1 Linear Regression Model: {theta[0]} + x1*{theta[1]}')
```

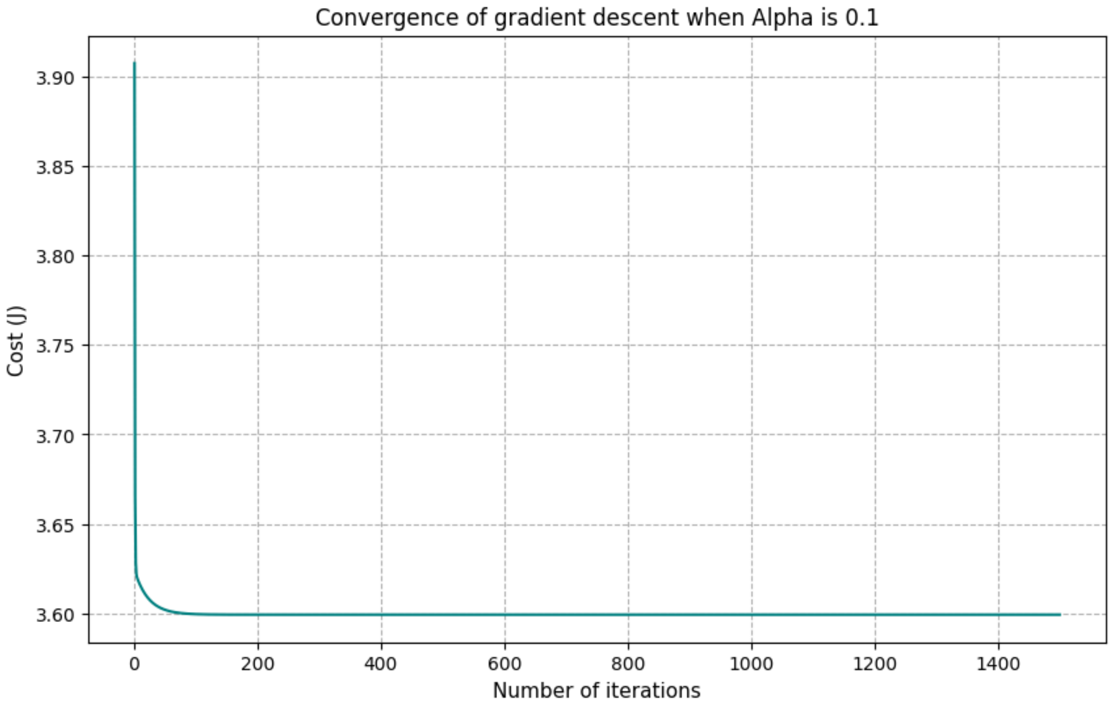
```
'Final X1 Linear Regression Model: 5.927939663897304 + x1*-2.0383330318833175'
```



Explanitory Variable X2 in Isolation

[889]:
```
# Since X1 and X2 may be of different lengths, reinserting the intercepts is a precaution
InpDataCopy=InpData.copy()
m = InpDataCopy.X2.size
X_0 = np.ones((m, 1))
X_0 = pd.DataFrame(X_0)
InpDataCopy.insert(0, "intercept",X_0)
X = InpDataCopy.loc[:,['intercept','X2']]
y = InpDataCopy.Y
Xarray = X.loc[:,['intercept','X2']].values
yarray = y.values
theta = np.zeros(2)
alpha=.1#This is the found alpha value that resulted in the lowest final loss
iterations = 1500;
theta, cost_history = gradientDescent(Xarray, yarray, theta, alpha, iterations)
print('Final value of theta =', theta)
print("Final loss for X2:", cost_history[-1])
plotConvFigure(iterations, cost_history, alpha)
```

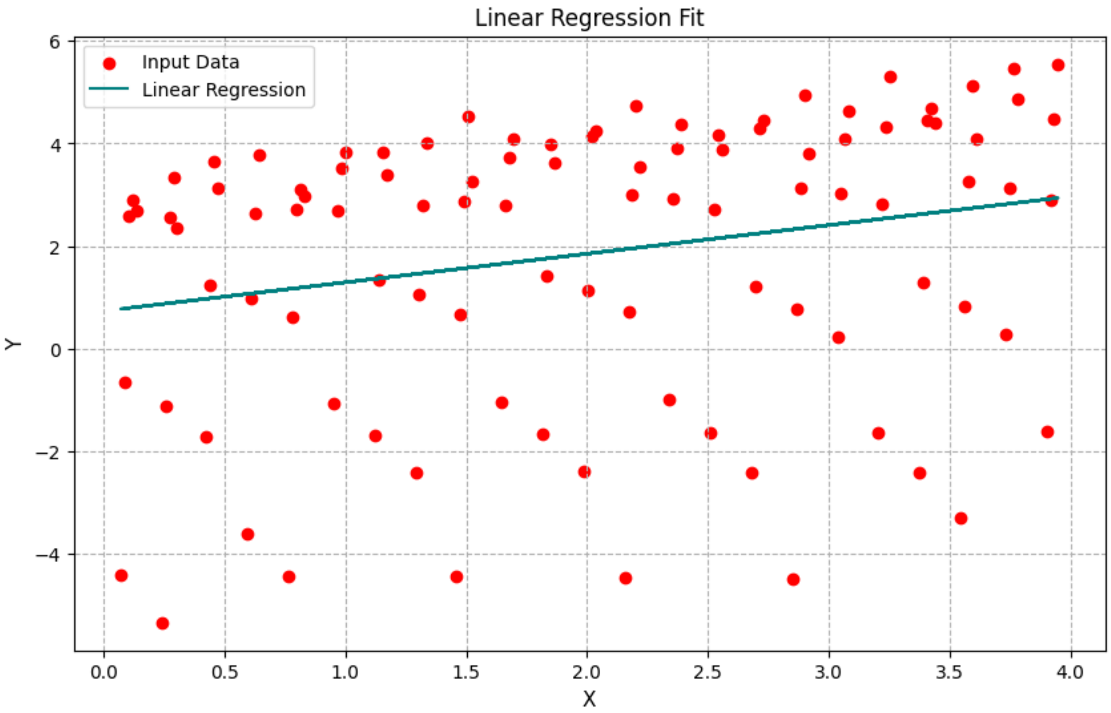
```
Final value of theta = [0.73606043 0.55760761]
Final loss for X2: 3.5993660181680425
```



[890]:

```
# Plot linear regression for X2
plotLinReg(X.X2.values, InpDataCopy.Y.values, theta)
display(f'Final X2 Linear Regression Model: {theta[0]} + x2*{theta[1]}')
```

```
'Final X2 Linear Regression Model: 0.7360604300947905 + x2*0.5576076103326044'
```

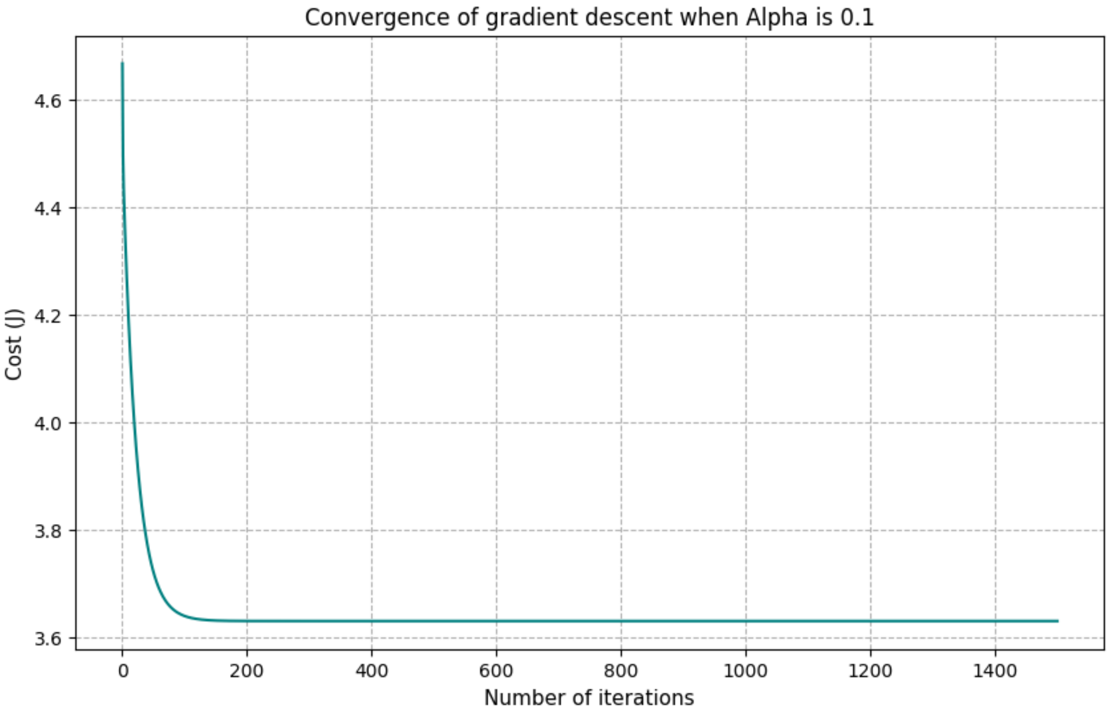


Explanitory Variable X3 in Isolation

[891]:

```
# Perform linear regression for X3
InpDataCopy=InpData.copy()
m = InpDataCopy.X3.size
X_0 = np.ones((m, 1))
X_0 = pd.DataFrame(X_0)
InpDataCopy.insert(0, "intercept",X_0)
X = InpDataCopy.loc[:,['intercept','X3']]
y = InpDataCopy.Y
Xarray = X.loc[:,['intercept','X3']].values
yarray = y.values
theta = np.zeros(2)
alpha=.1
iterations = 1500;
theta, cost_history = gradientDescent(Xarray, yarray, theta, alpha, iterations)
print('Final value of theta =', theta)
print("Final loss for X3:", cost_history[-1])
plotConvFigure(iterations, cost_history, alpha)
```

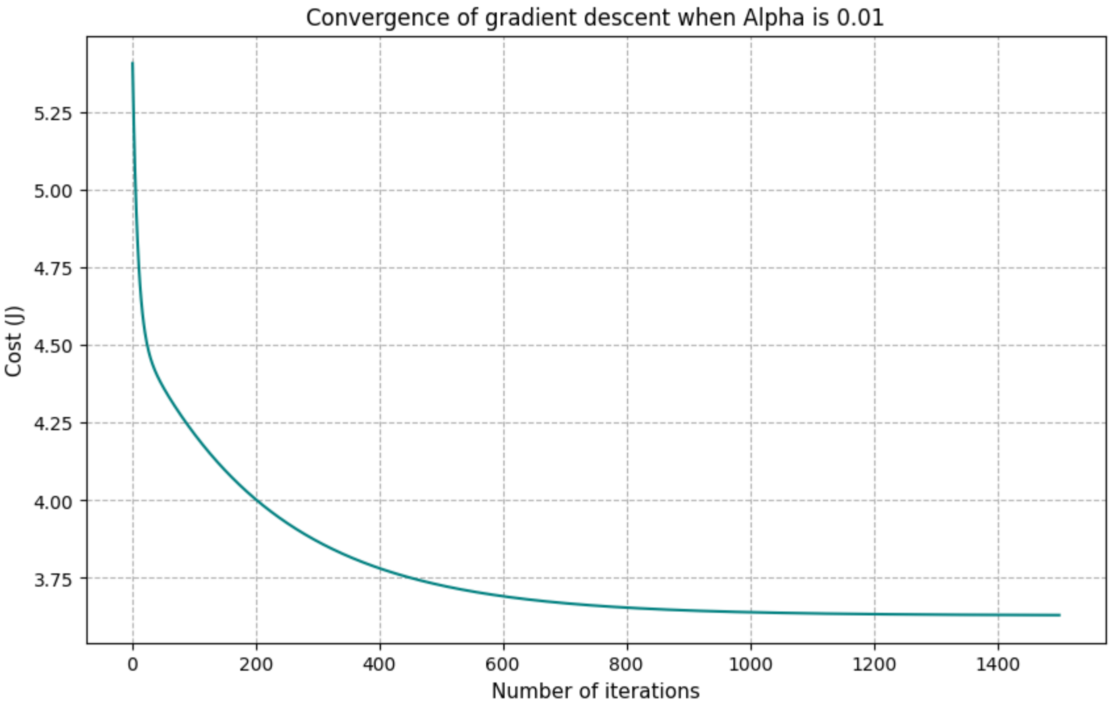
```
Final value of theta = [ 2.8714221  -0.52048288]
Final loss for X3: 3.6294511246079155
```



[892]:

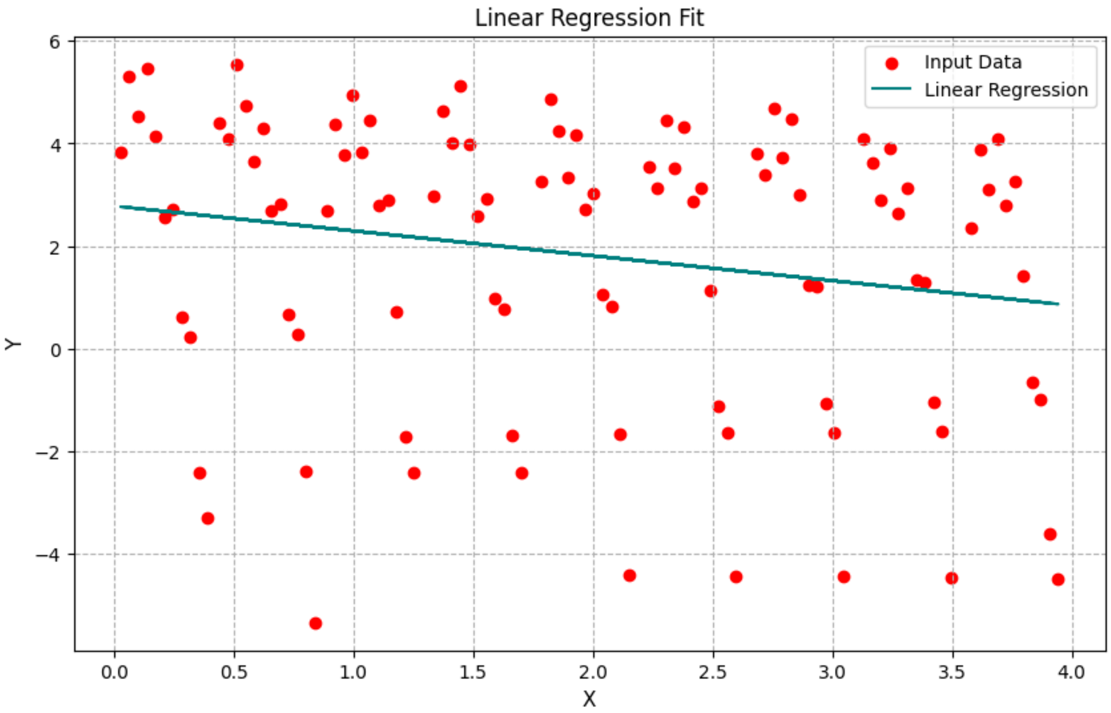
```
# Perform linear regression for X3 with a different learning rate
theta = np.zeros(2)
alpha=.01
theta, cost_history = gradientDescent(Xarray, yarray, theta, alpha, iterations)
print('Final value of theta =', theta)
print("Final loss for X3:", cost_history[-1])
plotConvFigure(iterations, cost_history, alpha)
```

```
Final value of theta = [ 2.78048129 -0.48451631]
Final loss for X3: 3.6305262475389664
```



[893]:
```
# Plot linear regression for X3
plotLinReg(X.X3.values, InpDataCopy.Y.values, theta)
display(f'Final X3 Linear Regression Model: {theta[0]} + x3*{theta[1]}')
```

```
'Final X3 Linear Regression Model: 2.78048129099449 + x3*-0.48451630947351204'
```



Which explanatory variable has the lower loss (cost) for explaining the output (Y)? X1 has lowest cost to explain output Y when fitting a linear regression model to the data.

Based on your training observations, describe the impact of the different learning rates on the final loss and number of training iterations.

Problem 2:

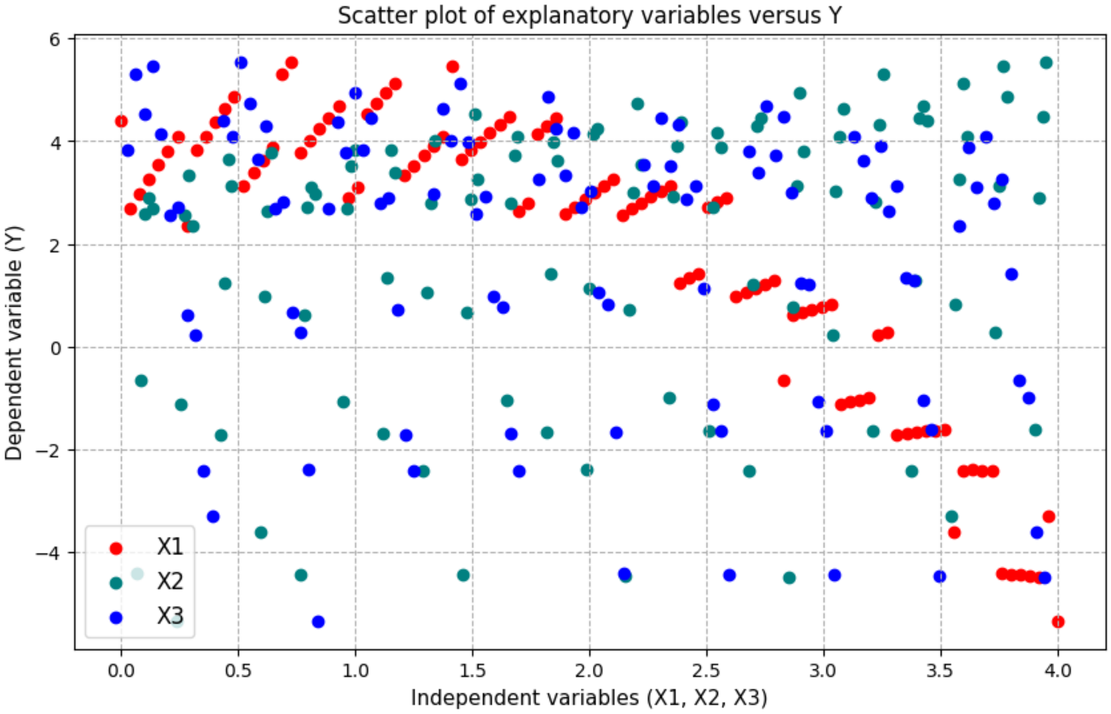
This time, run linear regression with gradient descent algorithm using all three explanatory variables. For the learning rate, explore different values between 0.1 and 0.01 (your choice). Initialize your parameters (theta to zero).

Report the final linear model you found the best. Plot loss over the iteration. Based on your training observations, describe the impact of the different learning rates on the final loss and number of training iterations. Predict the value of y for new (X1, X2, X3) values (1, 1, 1), for (2, 0, 4), and for (3, 2, 1).

Plotting All Explanitory Variables vs. Dependent Variable

[894]:

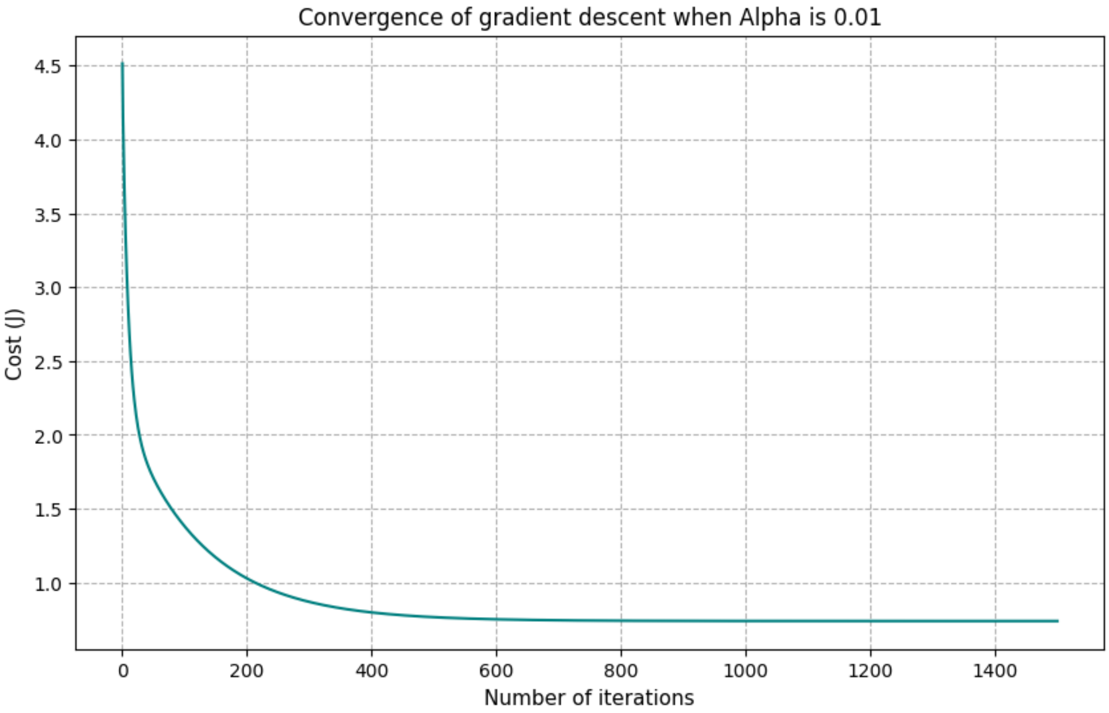
```
plt.figure(figsize=(10, 6))
plt.scatter(InpDataCopy.X1, InpDataCopy.Y, color='red', label='X1')
plt.scatter(InpDataCopy.X2, InpDataCopy.Y, color='teal', label='X2')
plt.scatter(InpDataCopy.X3, InpDataCopy.Y, color='blue', label='X3')
plt.grid(True, linestyle='--')
plt.xlabel('Independent variables (X1, X2, X3)', fontsize=11)
plt.ylabel('Dependent variable (Y)', fontsize=11)
plt.title('Scatter plot of explanatory variables versus Y', fontsize=12)
plt.legend(fontsize=12)
plt.show()
```



[895]:

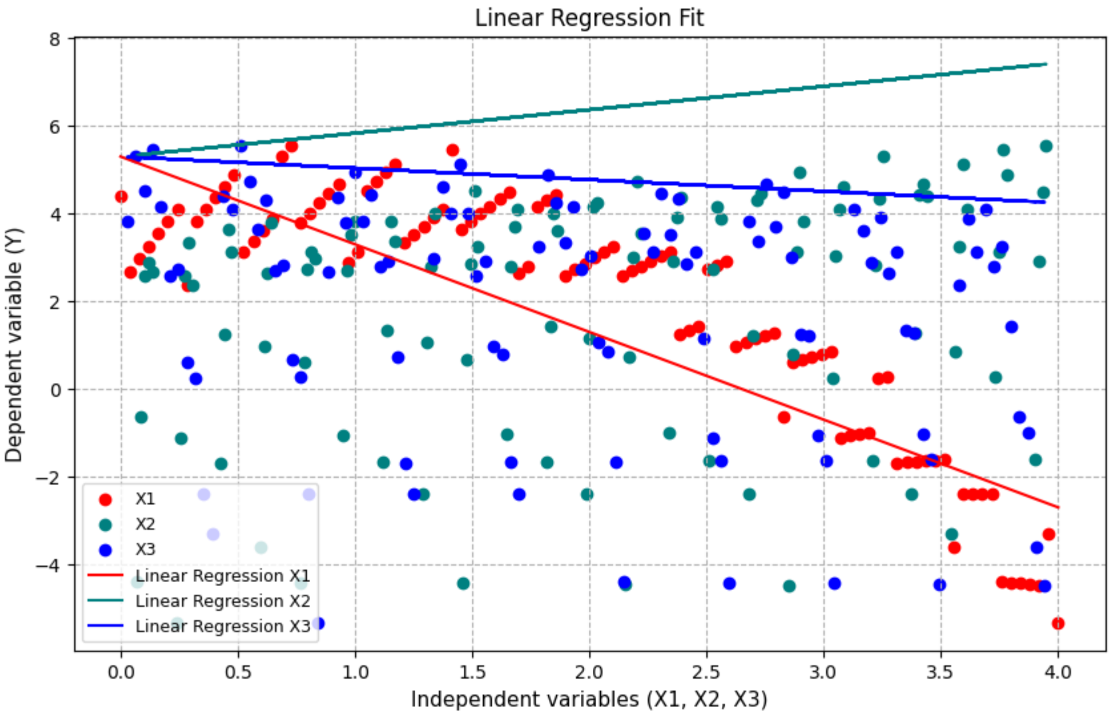
```
# Preparing the data and running gradient descent for linear regression with all explanatory variables (X1, X2, X3)
InpDataCopy=InpData.copy()
m = InpDataCopy.X1.size
X_0 = np.ones((m, 1))
X_0 = pd.DataFrame(X_0)
InpDataCopy.insert(0, "intercept",X_0)
X = InpDataCopy.loc[:,['intercept','X1','X2','X3']]
y = InpDataCopy.Y
Xarray = X.values
yarray = y.values
theta = np.zeros(X.shape[1])
iterations = 1500;
theta, cost_history = gradientDescent(Xarray, yarray, theta, 0.04, iterations)
print('Final value of theta =', theta)
print("Final loss:", cost_history[-1])
plotConvFigure(iterations, cost_history, alpha)
```

```
Final value of theta = [ 5.30120817 -2.0018886   0.53470473 -0.26370234]
Final loss: 0.7384731871693537
```



[896]:

```
# Scatter plot of all independent variables versus Y
plt.figure(figsize=(10, 6))
plt.scatter(InpDataCopy.X1, InpDataCopy.Y, color='red', label='X1')
plt.scatter(InpDataCopy.X2, InpDataCopy.Y, color='teal', label='X2')
plt.scatter(InpDataCopy.X3, InpDataCopy.Y, color='blue', label='X3')
plt.plot(InpDataCopy.X1, theta[0] + theta[1] * InpDataCopy.X1, color='red',␣
 ↪label='Linear Regression X1')
plt.plot(InpDataCopy.X2, theta[0] + theta[2] * InpDataCopy.X2, color='teal',␣
 ↪label='Linear Regression X2')
plt.plot(InpDataCopy.X3, theta[0] + theta[3] * InpDataCopy.X3, color='blue',␣
 ↪label='Linear Regression X3')
plt.grid(linestyle='--')
plt.xlabel('Independent variables (X1, X2, X3)', fontsize=11)
plt.ylabel('Dependent variable (Y)', fontsize=11)
plt.title('Linear Regression Fit', fontsize=12)
plt.legend(loc='lower left', fontsize=9)
plt.show()
display(f'Final Linear Regression Model: {theta[0]} + x1*{theta[1]}+␣
 ↪x2*{theta[2]}+ x3*{theta[3]}')
```



```
'Final Linear Regression Model: 5.301208174409086 + x1*-2.0018885958356587+ x2*0.
↪534704730324727+ x3*-0.2637023397841248'
```

By using a single intercept term, the model becomes simpler and easier to understand. It reduces the number of parameters that need to be estimated, making the interpretation of the model more straightforward. Estimating fewer parameters can lead to faster computational times, particularly when dealing with large datasets or complex models.

```
[897]: # Displaying the final multivariable regression model and predicting the output␣
        ↪for new input values
       display(f'Final Multivariable Regression Model: {theta[0]} + x1*{theta[1]} +␣
        ↪x2*{theta[2]} + x3*{theta[3]}')
       #Predicting the value of y for new (X1, X2, X3):
       #(1, 1, 1), for (2, 0, 4), and for (3, 2, 1)
       def discreteInp(x1,x2,x3):
         return theta[0]+x1*theta[1]+x2*theta[2]+x3*theta[3]

       display("Output at (1, 1, 1): "+str(discreteInp(1,1,1)))
       display("Output at (2, 0, 4): "+str(discreteInp(2,0,4)))
       display("Output at (3, 2, 1): "+str(discreteInp(3,2,1)))
       !jupyter nbconvert --to pdf /content/ECGR_4105_Assignment_1.ipynb
```

```
'Final Multivariable Regression Model: 5.301208174409086 + x1*-2.
↪0018885958356587 + x2*0.534704730324727 + x3*-0.2637023397841248'
```

```
'Output at (1, 1, 1): 3.5703219691140298'
```

```
'Output at (2, 0, 4): 0.2426216236012697'
```

```
'Output at (3, 2, 1): 0.10124950776743974'
```

```
[NbConvertApp] Converting notebook /content/ECGR_4105_Assignment_1.ipynb to pdf
[NbConvertApp] Support files will be in ECGR_4105_Assignment_1_files/
[NbConvertApp] Making directory ./ECGR_4105_Assignment_1_files
[NbConvertApp] Making directory ./ECGR_4105_Assignment_1_files
[NbConvertApp] Making directory ./ECGR_4105_Assignment_1_files
[NbConvertApp] Making directory ./ECGR_4105_Assignment_1_files
[NbConvertApp] Making directory ./ECGR_4105_Assignment_1_files
[NbConvertApp] Making directory ./ECGR_4105_Assignment_1_files
[NbConvertApp] Making directory ./ECGR_4105_Assignment_1_files
[NbConvertApp] Making directory ./ECGR_4105_Assignment_1_files
[NbConvertApp] Making directory ./ECGR_4105_Assignment_1_files
[NbConvertApp] Making directory ./ECGR_4105_Assignment_1_files
[NbConvertApp] Making directory ./ECGR_4105_Assignment_1_files
[NbConvertApp] Making directory ./ECGR_4105_Assignment_1_files
[NbConvertApp] Making directory ./ECGR_4105_Assignment_1_files
[NbConvertApp] Making directory ./ECGR_4105_Assignment_1_files
[NbConvertApp] Writing 62710 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 581409 bytes to /content/ECGR_4105_Assignment_1.pdf
```