DDA_Tutorial_1

April 29, 2022

Farjad Ahmed - 1747371 Tutorial_1 Submission for Distributed Data Analytics Lab - Group 1

This notebook contains explanations for questions in the tutorial and code descriptions as comments. The notebook can be run without any particular configuration. Required explanations and descriptions can be found for each section followed by the code and graphics.

Descriptions and Explanations can be directly accessed by searching for '[EXPLANATION]'.

```
[ ]: # Importing the libraries
     import numpy as np
     import math
     import matplotlib.pyplot as plt
     import pandas as pd
```

Matrix Multiplication: Create a numpy matrix A of dimensions n × m, where n = 100 and m = 20. Initialize Matrix A with random values. Create a numpy vector v of dimensions m × 1. Initialize the vector v with values from a normal distribution using μ = 2 and standard deviation = 0.01. Perform the following operations: 2. Iteratively multiply (element-wise) each row of the matrix A with vector v and sum the result of each iteration in another vector c. This operation needs to be done with for-loops, not numpy built-in operations.

```
[ ]: #Setting variables for the data
     n = 100
     m = 20
     u = 2
     sigma = 0.01

     np.random.seed(0) #setting the seed for the random number generator
     A = np.random.randn(n,m) #generating a random matrix
     v = np.random.normal(loc=u, scale = sigma, size=(m,1)) #generating a random␣
      ↪vector
     c = np.zeros((A.shape[0],v.shape[1])) #creating a matrix of zeros
     myList=[] #creating an empty list

     for i in range(A.shape[0]): #for loop to iterate through the rows of the matrix
         sumBin = 0 # initializing the sum variable to 0
         for j in range(A.shape[1]): #for loop to iterate through the columns of the␣
      ↪matrix
             product = A[i,j]*v[j,0] #calculating the product of the row and column
```

```
        sumBin = sumBin + product #adding the product to the sum
    c[i,0] = sumBin #setting the value of the row to the sum
```

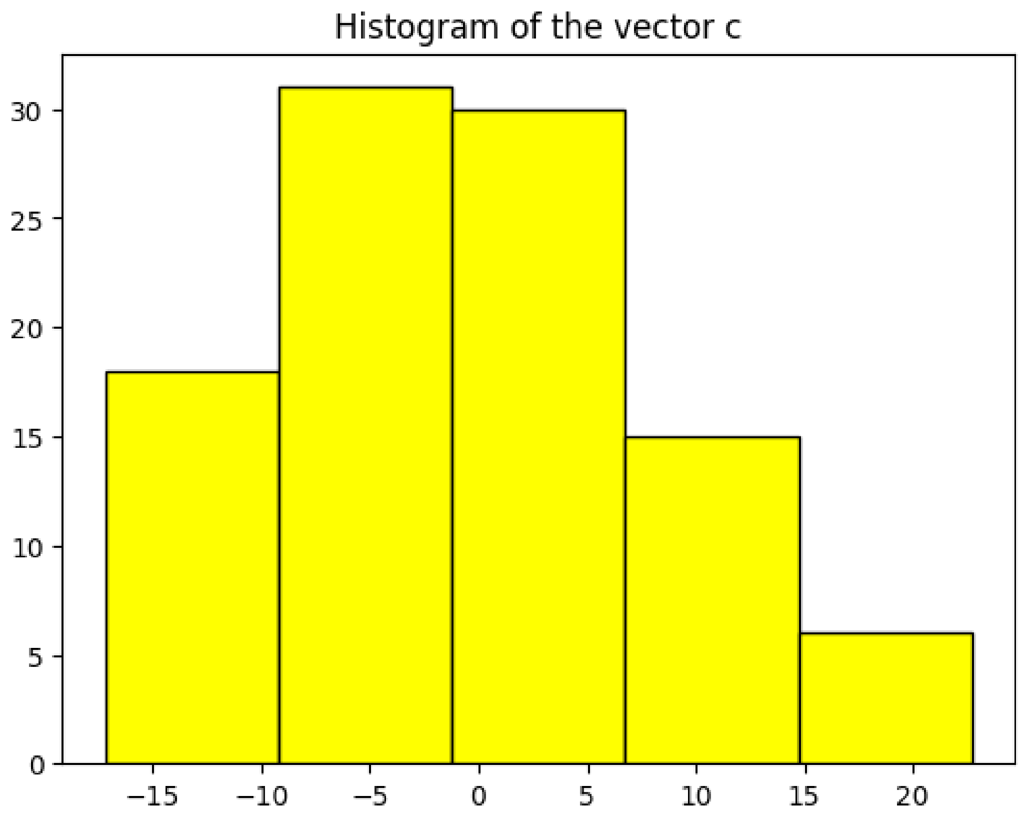
2. Find mean and standard deviation of the new vector c.

```
[ ]: # Mean and standard deviation of the c
     print('Mean :', np.mean(c, axis=0), 'Standard Deviation:', np.std(c, axis=0)) #␣
      ↪printing the mean and standard deviation of c
```

```
Mean : [-0.63158641] Standard Deviation: [8.64465701]
```

3. Plot the histogram of vector c using 5 bins.

```
[ ]: # Plotting the histogram
     plt.hist(c, bins = 5, color='yellow', edgecolor='black', linewidth=1,␣
      ↪align='mid') #plotting the histogram
     plt.title('Histogram of the vector c')
     plt.show()
```



Grading Program: This task puts you in the position that I end up at the end of every semester. Which is, grading your work and issuing the grades. In this task you are required to use the 'Grades.csv' File that has been provided on learnweb.

- Read the data from the csv.
- Compute the sum for all subjects for each student.

[ ]:
```
# Grading Program
df = pd.read_csv('Grades.csv') #reading the csv file
colList = df.columns #creating a list of the columns
subList = colList[2:-1] #creating a list of the subjects
calculated_total = df[subList].sum(axis=1) #calculating the total of the␣
↪subjects
df['Calculated_Totals'] = calculated_total #adding the calculated total to the␣
↪dataframe
print(df.head()) #printing the first 5 rows of the dataframe
```

```
  First Name  Last Name  English   Maths  Science  German  Sports  \
0      Robyn    Hobgood    60.95   24.77    20.60   69.32    8.36
1       Eddy  Swearngin   100.00   12.99   100.00   52.24  100.00
2      Leoma   Bridgman    83.37  100.00    78.69  100.00   19.50
3    Arnetta      Peart    87.75  100.00    86.93   87.90   41.73
4   Maryland      Colby   100.00  100.00   100.00   18.87   88.72

   Final Grade  Calculated_Totals
0       184.00             184.00
1       365.23             365.23
2       381.56             381.56
3       404.31             404.31
4       407.59             407.59
```

- Compute the average of the point for each student. (total points are 500).

[ ]:
```
# Calculating Average Point
student_average = calculated_total/500 #calculating the average point
df['Student_Average'] = student_average #adding the average point to the␣
↪dataframe
print(df.head()) #printing the first 5 rows of the dataframe
```

```
  First Name  Last Name  English   Maths  Science  German  Sports  \
0      Robyn    Hobgood    60.95   24.77    20.60   69.32    8.36
1       Eddy  Swearngin   100.00   12.99   100.00   52.24  100.00
2      Leoma   Bridgman    83.37  100.00    78.69  100.00   19.50
3    Arnetta      Peart    87.75  100.00    86.93   87.90   41.73
4   Maryland      Colby   100.00  100.00   100.00   18.87   88.72

   Final Grade  Calculated_Totals  Student_Average
0       184.00             184.00          0.36800
1       365.23             365.23          0.73046
2       381.56             381.56          0.76312
3       404.31             404.31          0.80862
4       407.59             407.59          0.81518
```

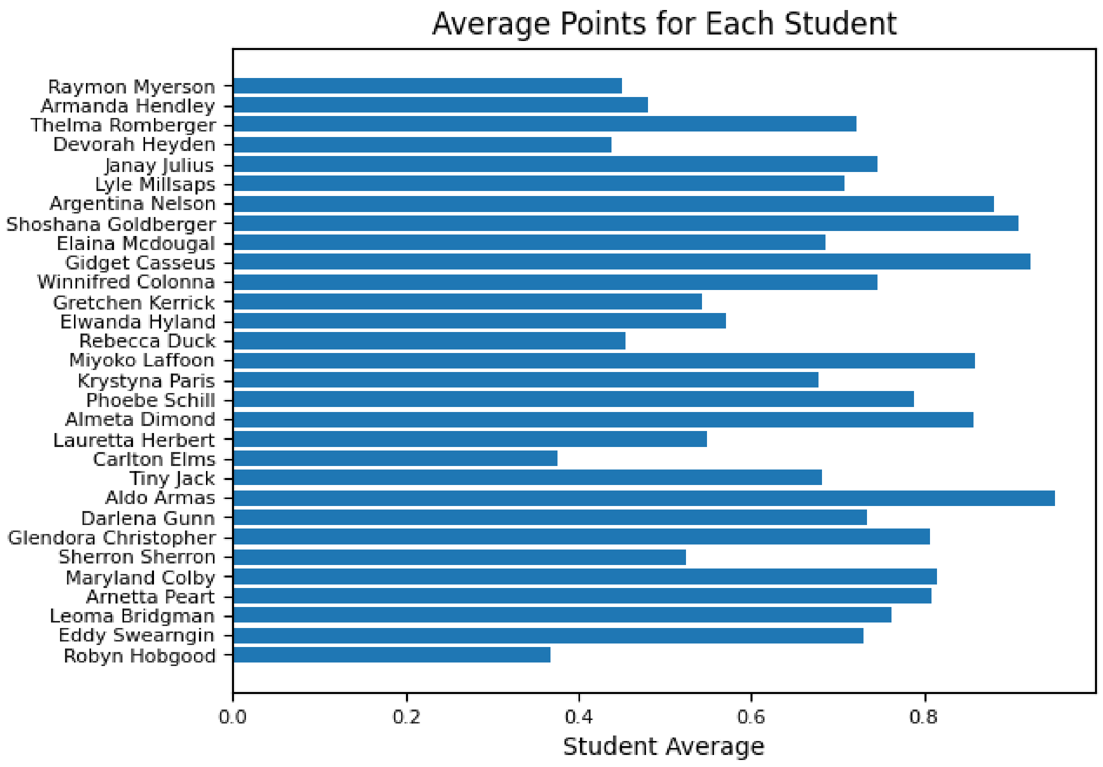
- Compute the standard deviation of point for each student.

```
[ ]: # Standard Deviation
     # Calculating standard deviation across row for each student
     std = df[subList].std(axis=1) #calculating the standard deviation of the␣
      ↪subjects
     df['Standard_Deviation'] = std #adding the standard deviation to the dataframe
     print(df.head()) #printing the first 5 rows of the dataframe
```

```
  First Name  Last Name  English   Maths  Science  German  Sports  \
0      Robyn    Hobgood    60.95   24.77    20.60   69.32    8.36
1       Eddy  Swearngin   100.00   12.99   100.00   52.24  100.00
2      Leoma   Bridgman    83.37  100.00    78.69  100.00   19.50
3    Arnetta      Peart    87.75  100.00    86.93   87.90   41.73
4   Maryland      Colby   100.00  100.00   100.00   18.87   88.72

   Final Grade  Calculated_Totals  Student_Average  Standard_Deviation
0       184.00             184.00          0.36800           26.724368
1       365.23             365.23          0.73046           39.430848
2       381.56             381.56          0.76312           33.186278
3       404.31             404.31          0.80862           22.535389
4       407.59             407.59          0.81518           35.360266
```

- Plot the average points for all the students (in one figure).

```
[ ]: # Plot for Average Points
     # Using matplotlib code example to plot student average points
     # Reference: https://matplotlib.org/stable/gallery/lines_bars_and_markers/
      ↪bar_label_demo.html

     nameList=df['First Name'] + ' ' + df['Last Name'] #creating a list of the names
     fig, ax = plt.subplots() # creating a figure and axes
     ax.barh(nameList, df['Student_Average'], align='center') #plotting the bar chart
     ax.set_yticks(nameList, labels=nameList) #setting the y ticks to the names
     ax.set_xlabel('Student Average') #setting the x label
     ax.set_title('Average Points for Each Student') #setting the title
     ax.tick_params(axis='x', labelsize=8) #setting the tick size
     ax.tick_params(axis='y', labelsize=8) #setting the tick size
     plt.show()
```



- For each student assign a grade based on the following rubric.

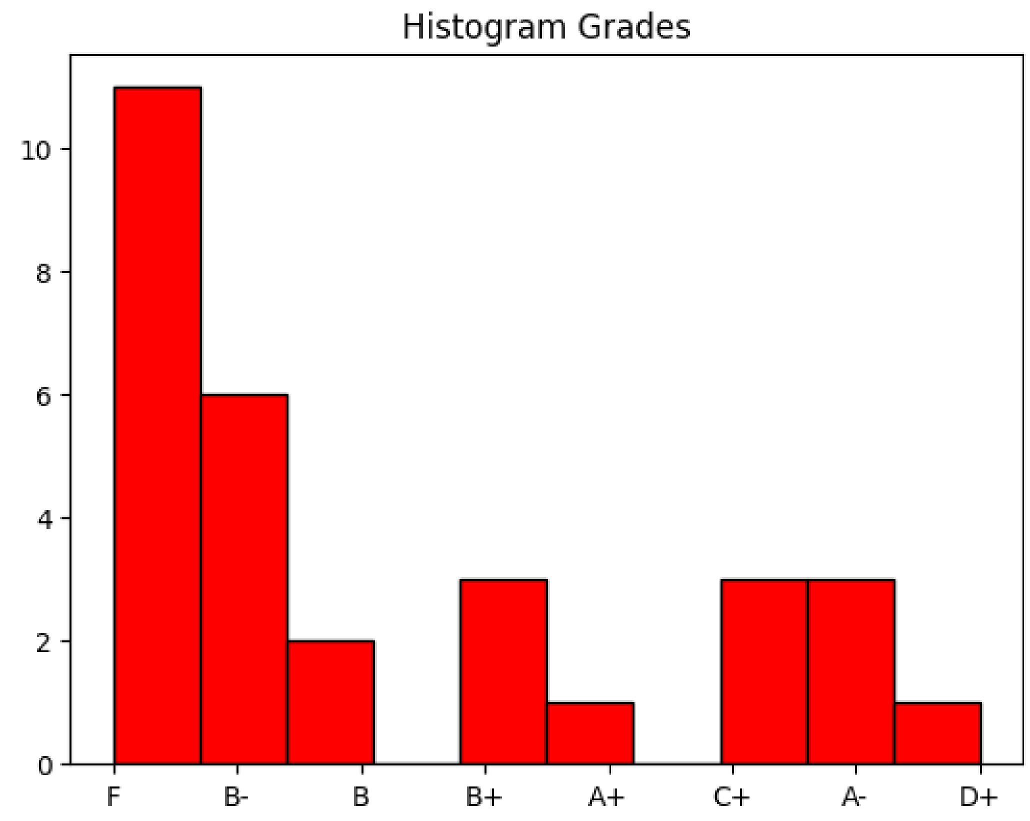
```
[ ]: # Grade Assignment
     # Grades are assigned to the students by rounding the average to nearest upper␣
      ↪integer to allow for all grades to be assigned

     grades = [] #creating an empty list
     gpa = df['Student_Average']*100 #calculating the percentage
     for marks in gpa: #for loop to iterate through the student averages
         marks = math.ceil(marks) #rounding the marks to the nearest upper integer
         if marks <= 100 and marks >= 96:
             grades.append('A+')
         elif marks <= 90 and marks >= 90:
             grades.append('A')
         elif marks <= 89 and marks >= 86:
             grades.append('A-')
         elif marks <= 85 and marks >= 80:
             grades.append('B+')
         elif marks <= 79 and marks >= 76:
             grades.append('B')
         elif marks <= 75 and marks >= 70:
             grades.append('B-')
```

```
    elif marks <= 69 and marks >= 66:
        grades.append('C+')
    elif marks <= 60 and marks >= 65:
        grades.append('C')
    elif marks <= 59 and marks >= 56:
        grades.append('D+')
    else:
        grades.append('F')

df['Grades'] = grades #adding the grades to the dataframe
print(df.head()) #printing the first 5 rows of the dataframe
```

```
  First Name  Last Name  English   Maths  Science  German  Sports  \
0      Robyn    Hobgood    60.95   24.77    20.60   69.32    8.36
1       Eddy  Swearngin   100.00   12.99   100.00   52.24  100.00
2      Leoma   Bridgman    83.37  100.00    78.69  100.00   19.50
3    Arnetta      Peart    87.75  100.00    86.93   87.90   41.73
4   Maryland      Colby   100.00  100.00   100.00   18.87   88.72

   Final Grade  Calculated_Totals  Student_Average  Standard_Deviation Grades
0       184.00             184.00          0.36800           26.724368      F
1       365.23             365.23          0.73046           39.430848     B-
2       381.56             381.56          0.76312           33.186278      B
3       404.31             404.31          0.80862           22.535389     B+
4       407.59             407.59          0.81518           35.360266     B+
```

- Plot the histogram of the final grades

```
[ ]: # Grades Histogram
     plt.title('Histogram Grades') #setting the title of the histogram
     plt.hist(grades, color='red', edgecolor='black', linewidth=1, align='mid')␣
       ↪#plotting the histogram
     plt.show() #showing the histogram
```



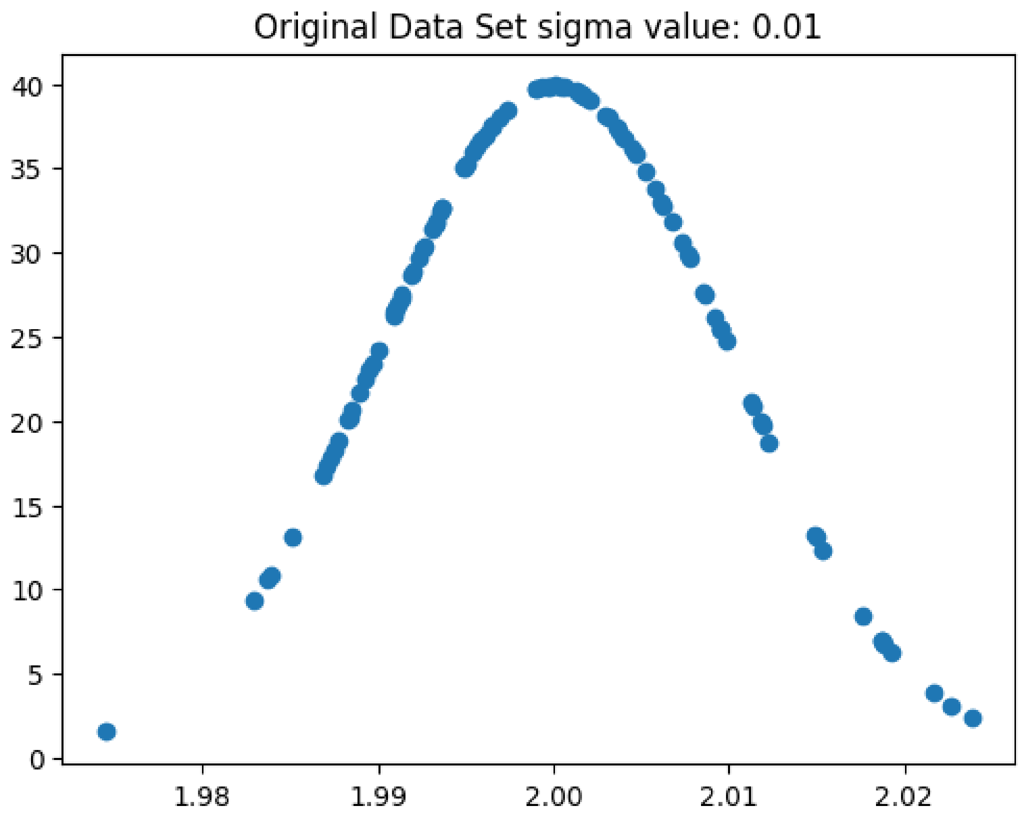
- Exercise 2: Linear Regression through exact form. (10 Points) In this exercise, you will implement linear regression that was introduced in the introduction Machine Learning Lecture. The method we are implementing here today is for a very basic univariate linear regression.
- Generate 3 sets of simple data, each consisting of a matrix A with dimensions 100 × 2. Initialize the sets of data with normal distribution µ = 2 and   = [0.01, 0.1, 1] so that each dataset has a different . You may assume that the first column of A represents the predictor data (X), whereas the second column of matrix A represents the target data (Y).
- Implement LEARN-SIMPLE-LINREG algorithm and train it using matrix A to learn values of b0 and b1.
- Implement LEARN-SIMPLE-LINREG algorithm and train it using matrix A to learn values of b0 and b1.
- The following section define the necessary variable and the functions for this exercise.

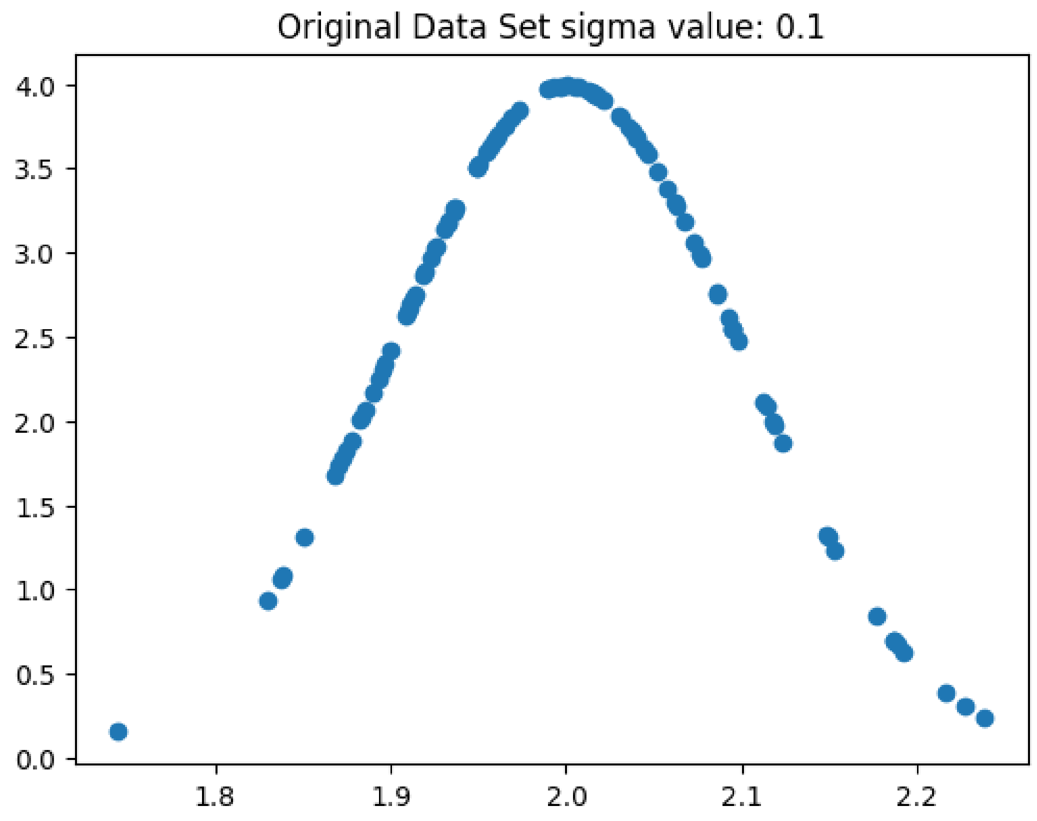
[ ]:
```
sigmaList = [0.01, 0.1, 1] #creating a list of the standard deviations
rows = 100 #setting the number of rows
cols = 2  #setting the number of columns
u = 2 #setting the mean

def GENERATE_DATA(rows, cols, u, sigma): #function to generate the data
```

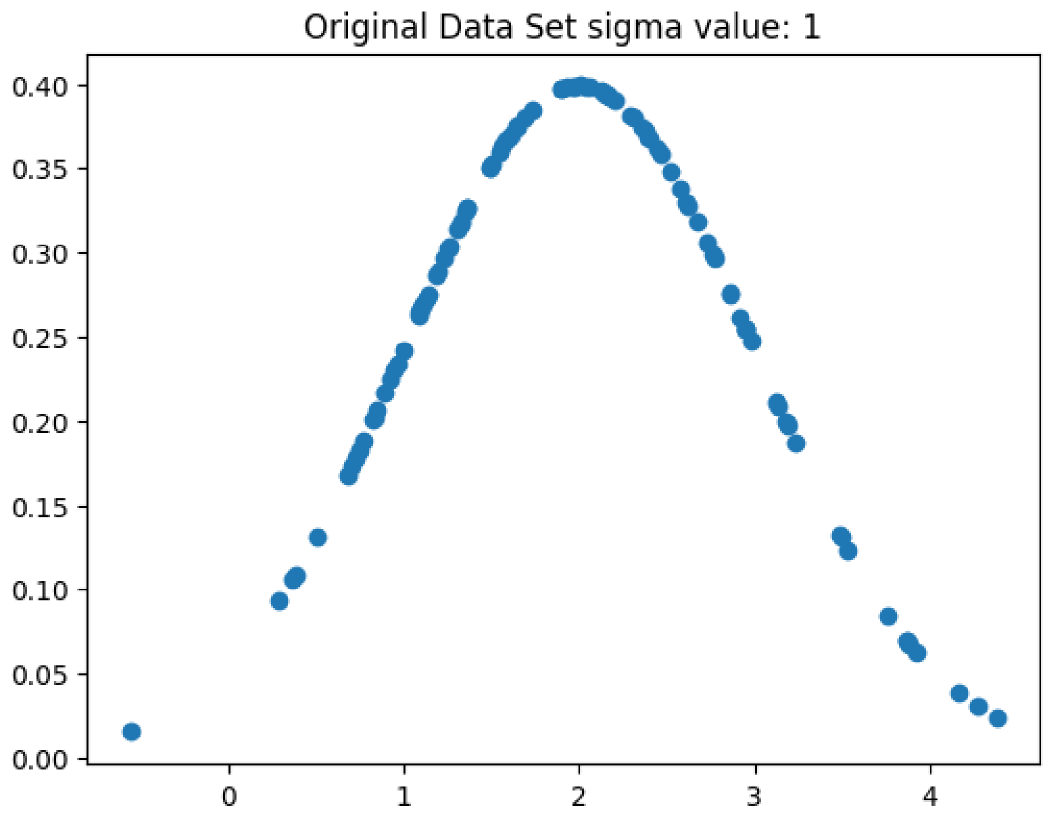
```
    np.random.seed(0) #setting the seed for the random number generator for␣
 ↪reproducibility
    A = np.random.normal(loc=u, scale = sigma, size=(rows, cols)) #generating a␣
 ↪random matrix
    return A[:,0], A[:,1] #returning the first column and the second column of␣
 ↪the matrix


def LEARN_SIMPLE_LINREG(X, Y): #function to learn the simple linear regression
    xmean = np.mean(X) #calculating the mean of the x values
    ymean = np.mean(Y) #calculating the mean of the y values
    b1 = sum((X-xmean)*(Y-ymean))/sum((X-xmean)**2) #calculating the slope
    b0 = ymean - (b1 * xmean) #calculating the y-intercept
    return(b0,b1) #returning the y-intercept and the slope


def PREDICT_SIMPLE_LINREG(b0, b1, x): #function to predict the simple linear␣
 ↪regression
    y = b0 + b1*x #calculating the y value
    return y #returning the y value
```

- [EXPLANATION] Data can be plotted to see how normal distribution looks with increasing the standard deviation values. As evident in the section below, with the increase in the standard deviation value, the spread of the data is increasing further away about the mean point.

[ ]:
```
# Lets have a look at the data
import scipy.stats as stats
for sig in sigmaList: #for loop to iterate through the standard deviations
    X,Y = GENERATE_DATA(rows, cols, u, sig) #generating the data
    plt.title('Original Data Set'+' '+'sigma value: '+str(sig))
    plt.plot(X, stats.norm.pdf(X, u, sig), 'o') #plotting the original data
    plt.show()
```


Original Data Set sigma value: 0.01
0
5
10
15
20
25
30
35
40
1.98
1.99
2.00
2.01
2.02


Original Data Set sigma value: 0.1
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
1.8
1.9
2.0
2.1
2.2

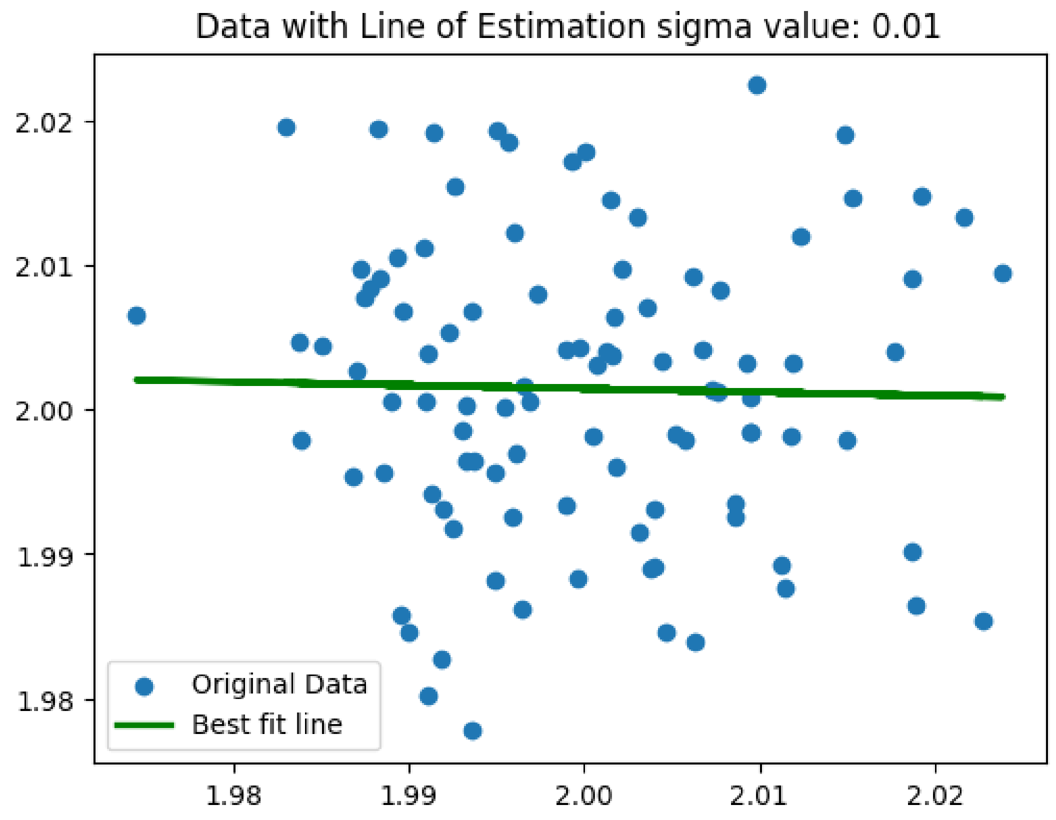


- [EXPLANATION] In the section below, the earlier implemented algorithms are being called, with a for loop to iterate over the provided std. deviation values. The original data and the predicted data is then presented in the graphs generated for each data set as per the std. deviation value.

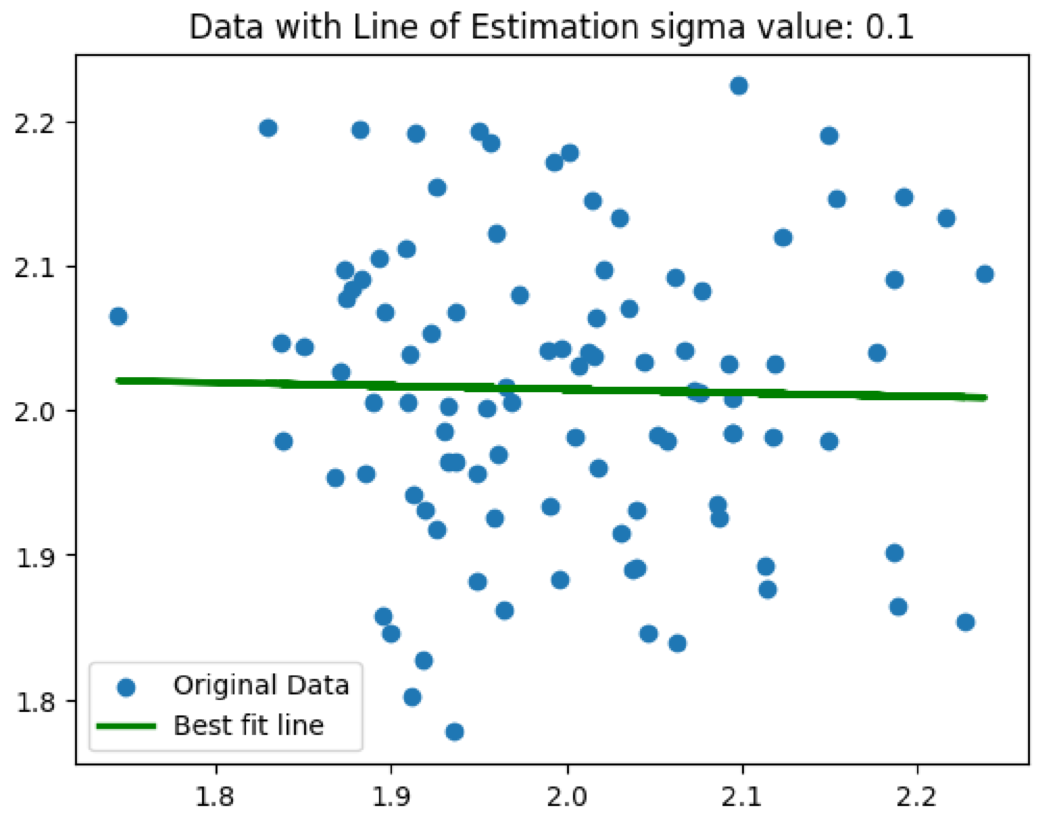
[ ]:
```
v1=np.zeros((3,2)) #creating a matrix of zeros
i1 = 0
for sig in sigmaList: #for loop to iterate through the standard deviations
    X,Y = GENERATE_DATA(rows, cols, u, sig) #generating the data
    b0, b1 = LEARN_SIMPLE_LINREG(X,Y) #learning the simple linear regression
    v1[i1,:] = np.array((b0, b1)) # creating a vector of the y-intercept and␣
 ↪the slope to compare later with numpy.linalg.lstsq
    print('Sigma:', sig, 'B0:', b0, 'B1:', b1) #printing the slope and␣
 ↪y-intercept
    y = PREDICT_SIMPLE_LINREG(b0, b1, X) #predicting the y value
    plt.figure() #creating a figure
    plt.title('Data with Line of Estimation'+' '+'sigma value: '+str(sig))␣
 ↪#setting the title
    plt.scatter(X,Y, label='Original Data') #plotting the data
    plt.plot(X,y,'g', label='Best fit line', linewidth=2.4) #plotting the line␣
 ↪of estimation
```

```
    plt.legend() #plotting the legend
    plt.show() #showing the figure
    i1 += 1
```

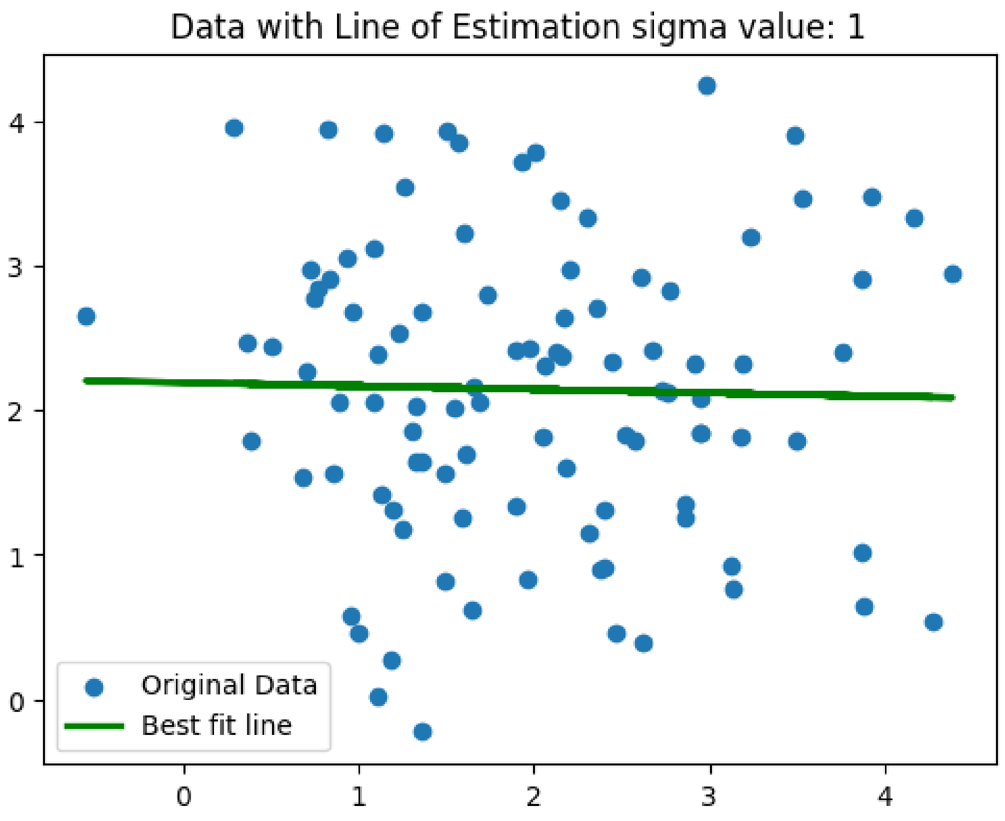
```
Sigma: 0.01 B0: 2.0491493939501475 B1: -0.023860917890473884
```



```
Sigma: 0.1 B0: 2.0619974174729507 B1: -0.023860917890475914
```



```
Sigma: 1 B0: 2.1904776527009386 B1: -0.023860917890475678
```

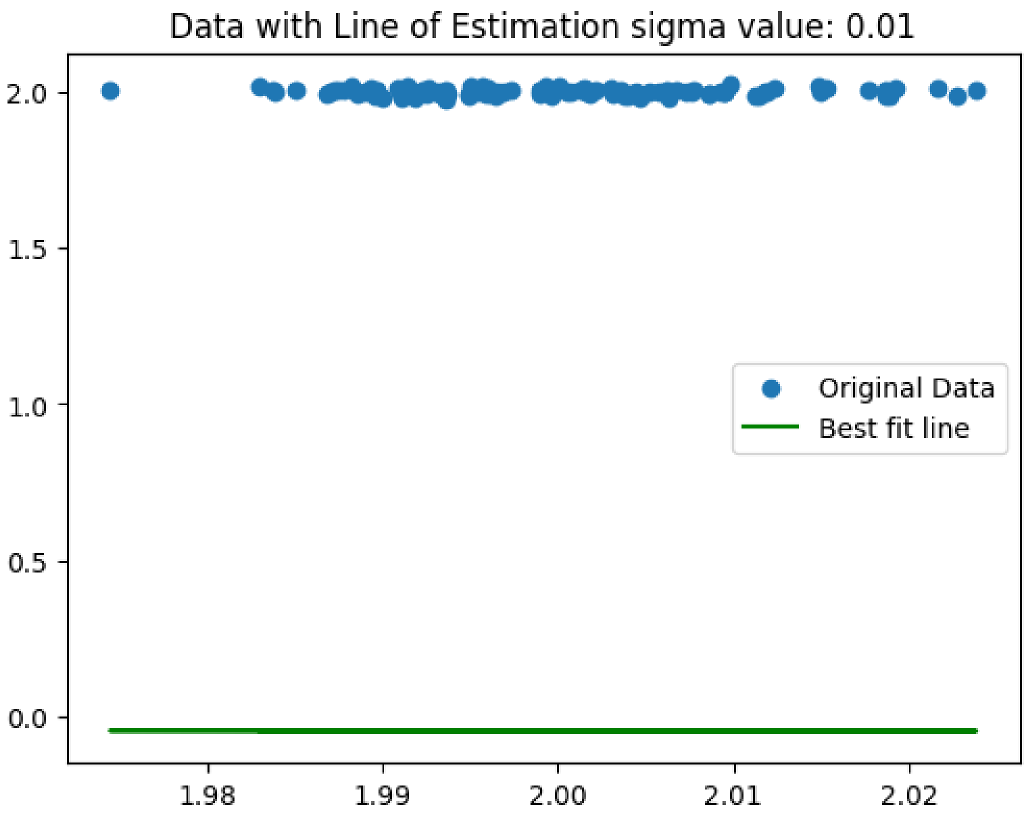


- Put b0 to zero and rerun the program to generate the predicted line. Comment on the change you see for the varying values of .
- [EXPLANATION] As evident in the next section, the value of b0 = 0, hence this translates to the y-intercept to be 0. As per the definition of y-intercept, which is the point where the graph intersects the y-axis. Now as evident from the values and the plots, as standard deviation increases the spread of the data is also increasing around the mean point. The relation standard deviation to b1 is by (b1 = sum((X-xmean)*(Y-ymean))/sum((X-xmean)**2)). Moreover, due to b0 being 0, that is the intercept, the predicted line is further from the actual data cloud exactly by the shift factor equal to b0.

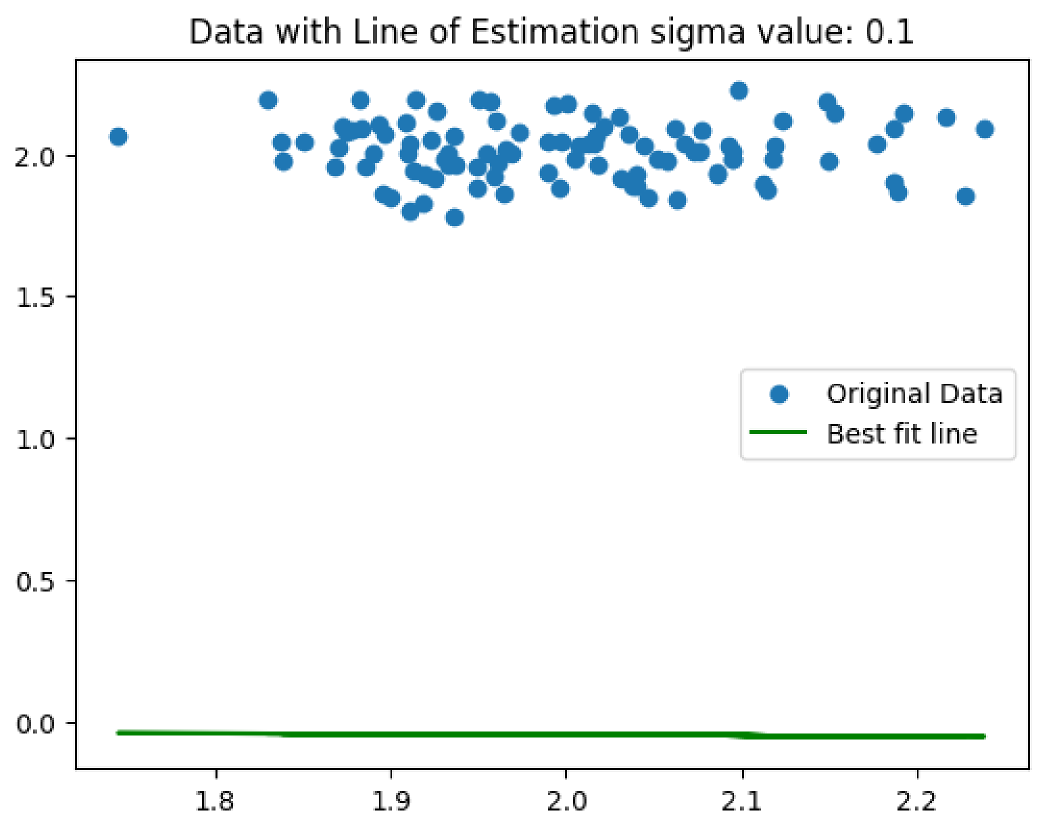
```
[ ]: b0 = 0 # setting the y-intercept to 0
     for sig in sigmaList: #for loop to iterate through the standard deviations
         X,Y = GENERATE_DATA(rows, cols, u, sig) #generating the data
         _, b1 = LEARN_SIMPLE_LINREG(X,Y) #learning the simple linear regression
         print('Sigma:', sig, 'B0:', b0, 'B1:', b1) #printing the slope and␣
      ↪y-intercept
         y = PREDICT_SIMPLE_LINREG(b0, b1, X) #predicting the y value
         plt.figure() #creating a figure
         plt.title('Data with Line of Estimation'+' '+'sigma value: '+str(sig))␣
      ↪#setting the title
         plt.scatter(X,Y, label='Original Data') #plotting the data
```

```
    plt.plot(X,y,'g', label='Best fit line') #plotting the best fit line
    plt.legend() #plotting the legend
    plt.show() #showing the plot
```

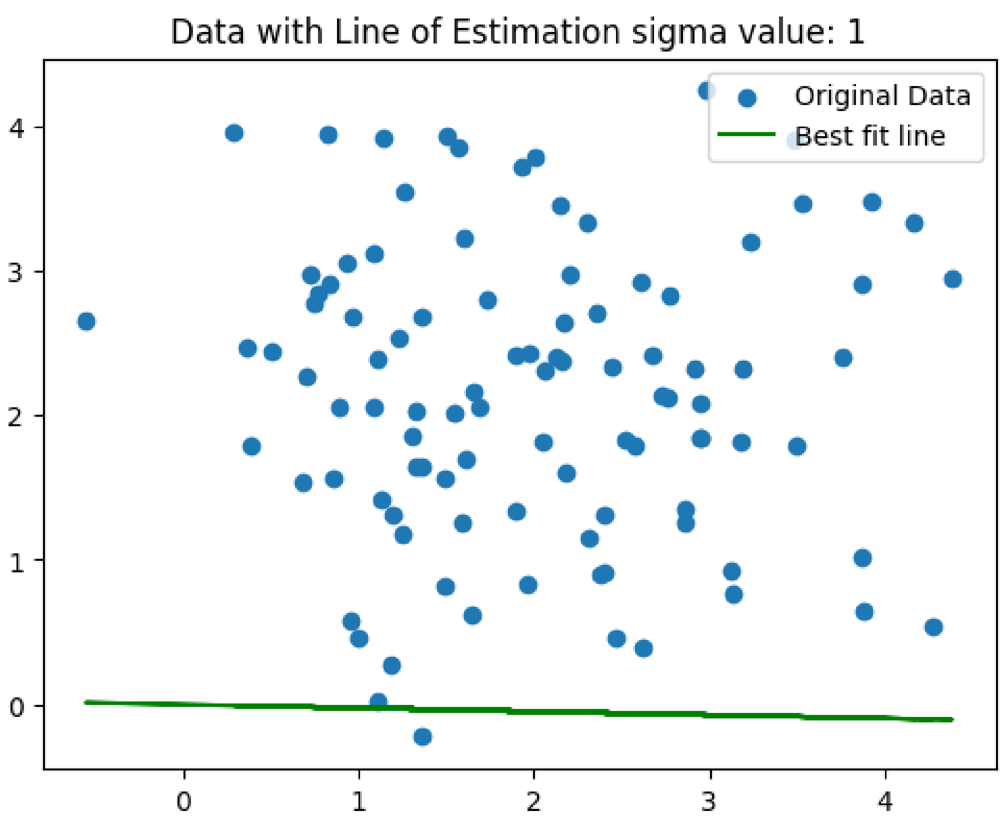
```
Sigma: 0.01 B0: 0 B1: -0.023860917890473884
```



```
Sigma: 0.1 B0: 0 B1: -0.023860917890475914
```



```
Sigma: 1 B0: 0 B1: -0.023860917890475678
```

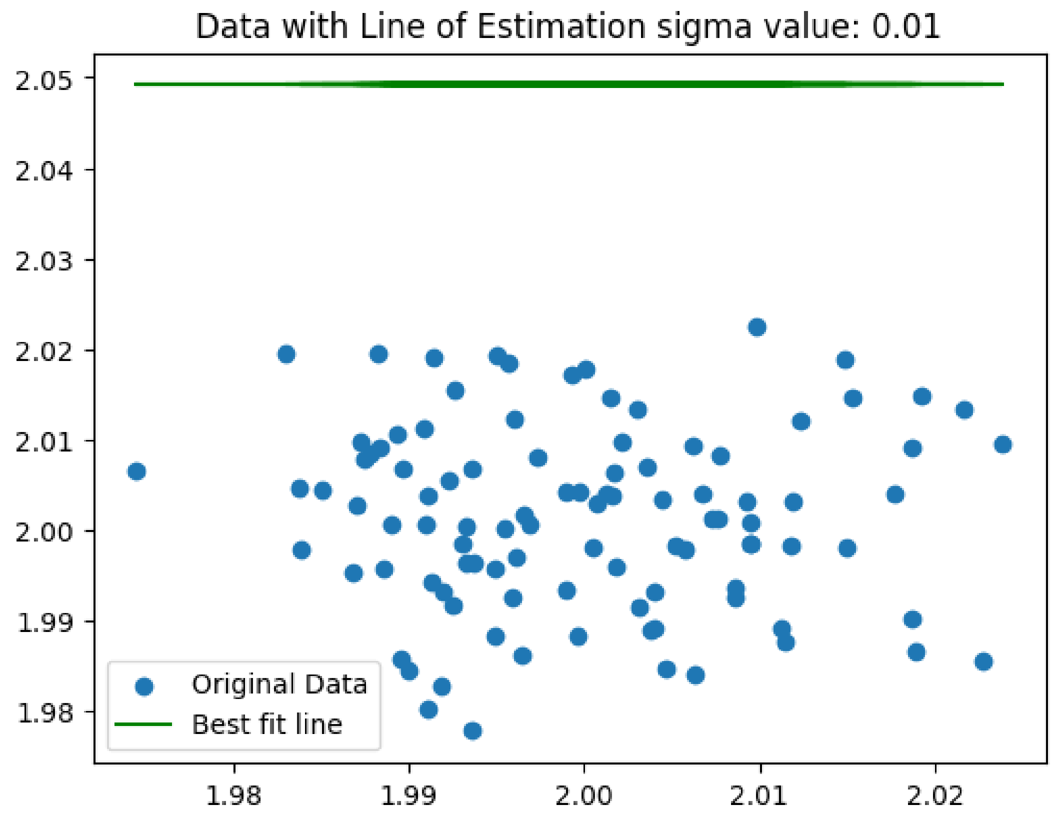


- Put b1 to zero and rerun the program to generate the predicted line. Comment on the change you see for the varying values of .
- [EXPLANATION] The equation in our case looks like, Y = b1*X + b0. This means if b1 = 0, the relationship of X and Y is vanished. This means the output from our implementaion has no direct correlation other than the value of b0 to the input, i.e (b0 = ymean - (b1 xmean)).* In such a case varying values of standard deviation have no impact on the value of prediction 'y' since y is not equals to a constant that is b0, the equation looks like y = b0. Hence, in all three cases there is a line with 0 gradient (straight, horizontal line) that is only determined through a scalar value b0.

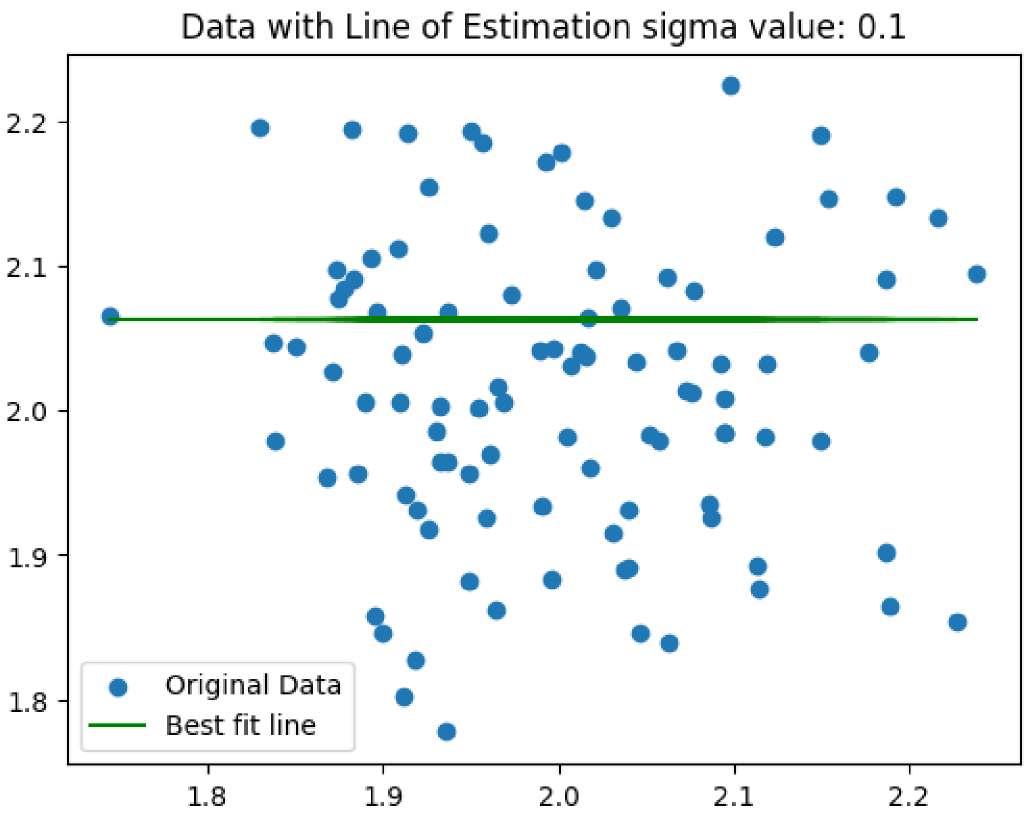
```
[ ]: b1 = 0 # setting the slope to 0
     for sig in sigmaList: #for loop to iterate through the standard deviations
         X,Y = GENERATE_DATA(rows, cols, u, sig) #generating the data
         b0, _ = LEARN_SIMPLE_LINREG(X,Y) #learning the simple linear regression
         print('Sigma:', sig, 'B0:', b0, 'B1:', b1) #printing the slope and␣
      ↪y-intercept
         y = PREDICT_SIMPLE_LINREG(b0, b1, X) #predicting the y value
         plt.figure() #creating a figure
         plt.title('Data with Line of Estimation'+' '+'sigma value: '+str(sig))␣
      ↪#setting the title
         plt.scatter(X,Y, label='Original Data') #plotting the data
```

```
    plt.plot(X,y,'g', label='Best fit line') #plotting the best fit line
    plt.legend() #plotting the legend
    plt.show() #showing the plot
```

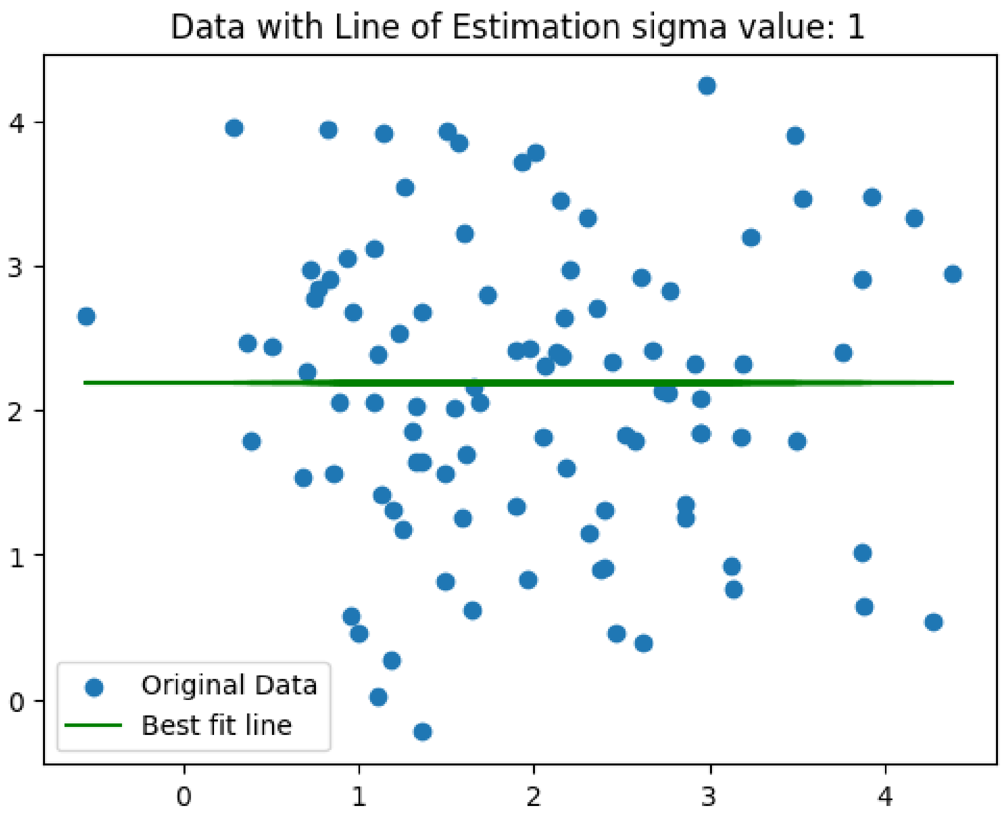
```
Sigma: 0.01 B0: 2.0491493939501475 B1: 0
```



```
Sigma: 0.1 B0: 2.0619974174729507 B1: 0
```



```
Sigma: 1 B0: 2.1904776527009386 B1: 0
```

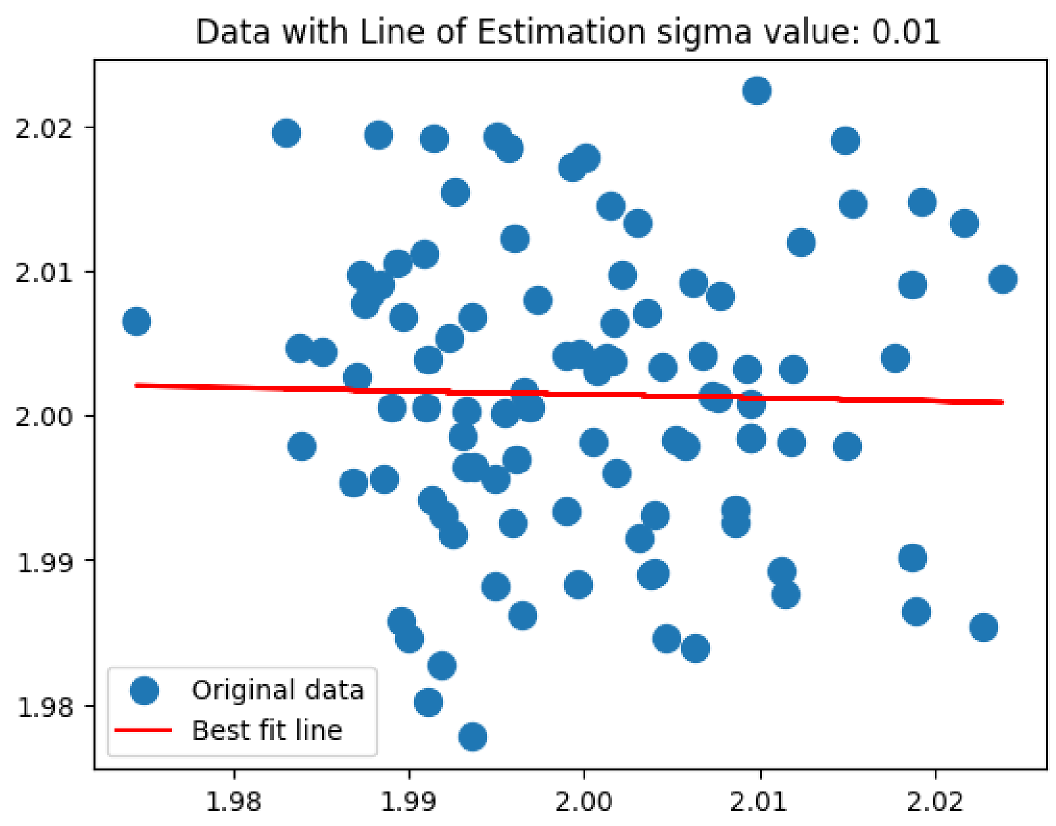


- Use numpy.linalg lstsq to replace step 2 for learning values of b0 and b1 . Explain the difference between your values and the values from numpy.linalg lstsq.
- [EXPLANATION] In this section, first numpy.linalg.lstsq() is used then the results obtained from manual implementation and numpy.linalg.lstsq() are compared.
- [EXPLANATION] There seems to be negligible difference in the output, which is due to higher decimal positions only, since both of the algorithms use sum of squares to find the error values for the calculation of the gradient and the intercept.

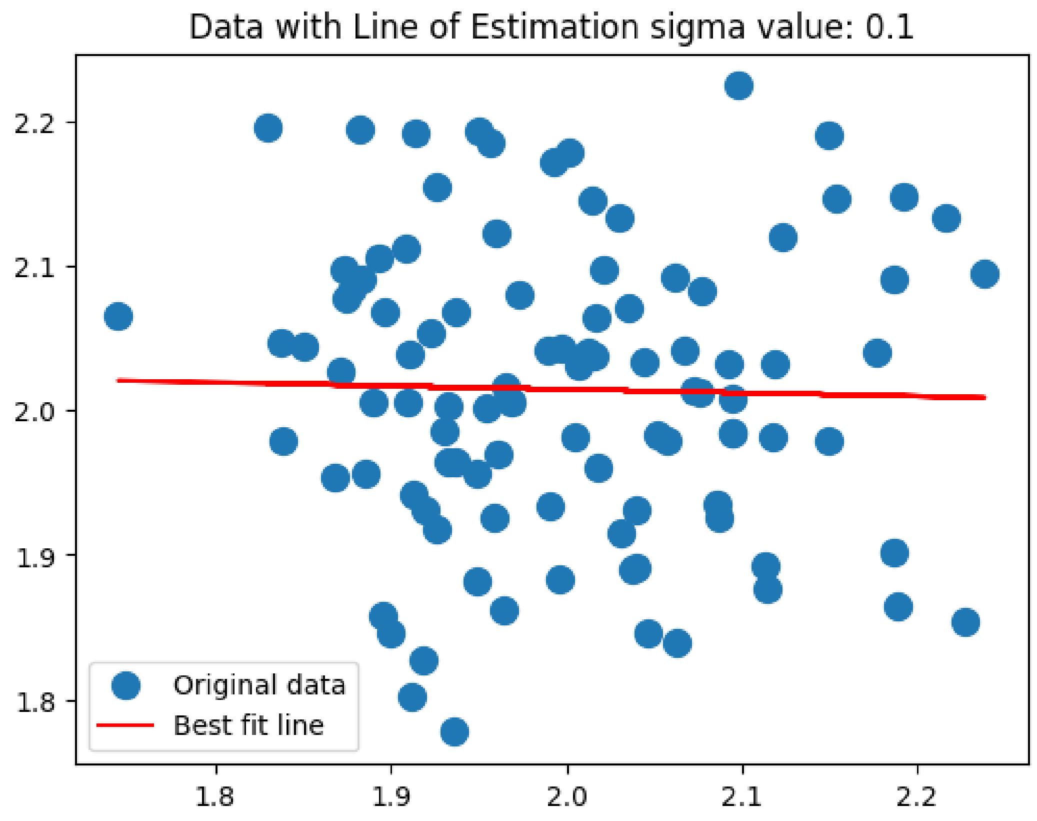
```
[ ]: # In this section  we will be using numpy.linalg.lstsq() to solve the linear␣
      ↪regression problem.
     # As per the documentation for numpy.linalg.lstsq(). It processes the data in␣
      ↪the form y=A_dash * p
     # where A_dash = [[X, 1]] and p = [m, c] where m is the slope(b1_dash) and␣
      ↪c(b0_dash) is the y-intercept.
     # In this case we will loop over the standard deviation list, generate A from␣
      ↪X, then apply the np.linalg.lstsq()
     # Reference: https://numpy.org/doc/stable/reference/generated/numpy.linalg.
      ↪lstsq.html
     v2=np.zeros((3,2)) #creating a matrix of zeros
     i2=0 #setting the index to 0
```

```
for sig in sigmaList: #for loop to iterate through the standard deviations
    x,y = GENERATE_DATA(rows, cols, u, sig) #generating the data
    A = np.vstack([x, np.ones(len(x))]).T # using vstack to stack the x values␣
 ↪and the ones values
    b1_dash, b0_dash = np.linalg.lstsq(A, y, rcond=None)[0] #calculating the␣
 ↪slope and y-intercept
    v2[i2,:] = np.array((b0_dash, b1_dash)) # creating a vector of the␣
 ↪y-intercept and the slope to compare with values from custom implementation
    print('Sigma:', sig, 'B0:', b0_dash, 'B1:', b1_dash) #printing the slope␣
 ↪and y-intercept
    plt.figure() #creating a figure
    plt.title('Data with Line of Estimation'+' '+'sigma value: '+str(sig))
    plt.plot(x, y, 'o', label='Original data', markersize=10)
    plt.plot(x, b1_dash*x + b0_dash, 'r', label='Best fit line')
    plt.legend()
    plt.show()
    i2 += 1
v3 = v2-v1  # subtracting the values from the custom implementation from the␣
 ↪numpy.linalg.lstsq() implementation
v3 = sum((v3)**2) #calculating the sum of the squared values
print('The difference in values from manual implementation and numpy.linalg.
 ↪lstsq() is: ' ,v3)
```

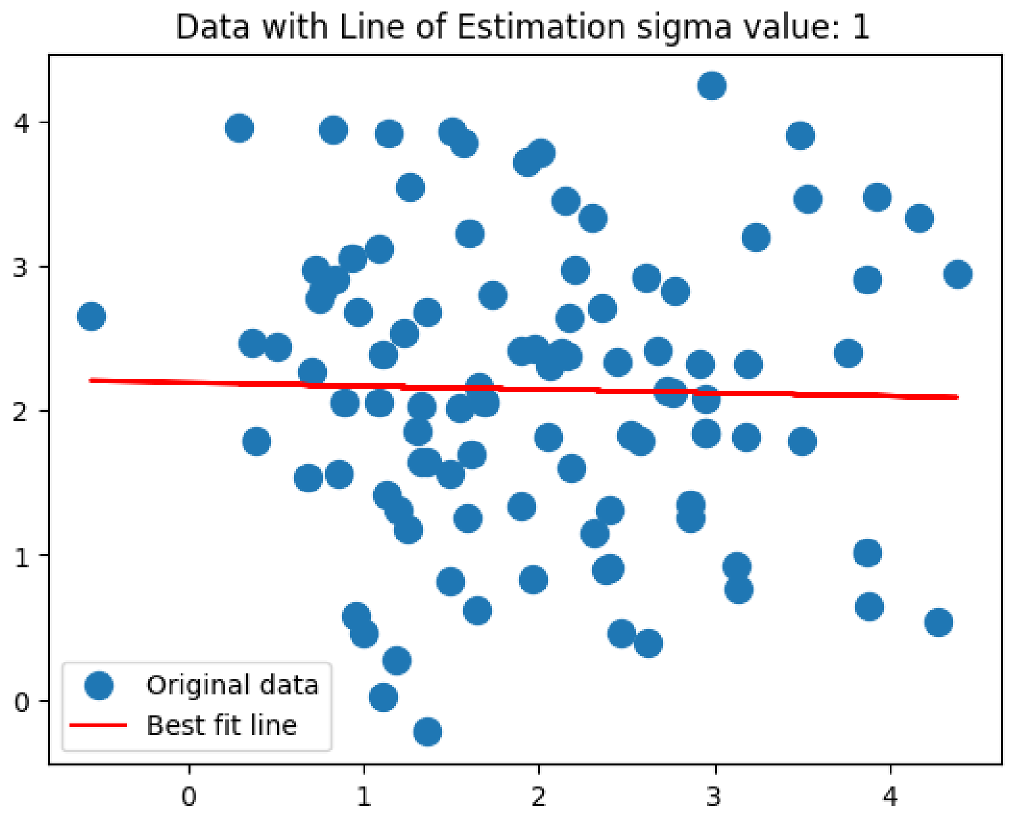
```
Sigma: 0.01 B0: 2.049149393950148 B1: -0.023860917890473943
```



```
Sigma: 0.1 B0: 2.06199741747295 B1: -0.023860917890474752
```



```
Sigma: 1 B0: 2.190477652700939 B1: -0.023860917890475508
```



```
The difference in values from manual implementation and numpy.linalg.lstsq() is:
[1.18329136e-30 1.38323900e-30]
```