DDA LAB

SIMRAN KAUR

311443

EXERCISE 0

Exercise 1: Pandas and Numpy

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 σ = 0.01. Perform the following operations:

Import neccesaary libraries:

In [231…

```
import numpy as np
np.random.seed(3116)
import pandas as pd
import matplotlib.pyplot as plt
```

In [232…

```
n = 100
m = 20
A = np.random.rand(n, m)                    # random matrix of size n*m
v = np.random.normal(2, 0.01, (m, 1))        # vector v follows normal distribution with
```

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.

In [233…

```
c = np.zeros((n, 1))
for i in range(n):
    s = 0
    for k in range(m):
        s += A[i, k]*v[k, 0]
    c[i, 0] = s
```

In [234…

```
c
```

Out[234…

```
array([[17.85029831],
       [17.39078046],
       [23.92257886],
       [16.10361252],
       [21.91389217],
```

```
       [20.71572109],
       [20.70414083],
       [21.73074791],
       [18.03265788],
       [18.42550164],
       [20.09508694],
       [21.55314702],
       [22.78257233],
       [22.97044322],
       [15.74132265],
       [17.06115464],
       [18.59471626],
       [16.62189777],
       [23.46061155],
       [24.11633879],
       [22.57835999],
       [19.64265353],
       [17.77751012],
       [18.46844961],
       [22.38443743],
       [23.10163005],
       [18.91813174],
       [18.7712037 ],
       [17.55031908],
       [16.55344324],
       [16.87807418],
       [21.7068468 ],
       [16.59520652],
       [19.28456719],
       [18.64078492],
       [18.59105926],
       [18.54718473],
       [20.11876054],
       [19.92635347],
       [17.52525877],
       [21.97329414],
       [19.6983184 ],
       [23.73853907],
       [19.09608305],
       [20.31795672],
       [22.58078777],
       [21.20143414],
       [17.32665291],
       [18.94688536],
       [21.29038177],
       [20.84622189],
       [23.19800458],
       [17.11366091],
       [24.14144434],
       [17.03363191],
       [20.43472884],
       [22.69827618],
       [21.27844204],
       [22.18278192],
       [21.16529422],
       [13.44535602],
       [19.32608122],
       [17.9124977 ],
       [18.48819489],
       [17.4209562 ],
       [16.00307368],
       [14.73277605],
       [17.16858432],
       [22.83904105],
```

```
       [20.11802772],
       [21.72056128],
       [17.45342202],
       [20.77493631],
       [19.74142791],
       [19.73543866],
       [14.75017129],
       [20.08152742],
       [12.46945761],
       [24.34692527],
       [18.95912442],
       [17.80099456],
       [20.25183613],
       [17.54689172],
       [16.71506422],
       [14.97795403],
       [21.3643909 ],
       [20.24115154],
       [21.37708409],
       [21.7120462 ],
       [21.10772141],
       [20.41733953],
       [17.21649377],
       [21.9592206 ],
       [21.70470071],
       [24.84749528],
       [16.7223465 ],
       [15.93213613],
       [20.33825518],
       [19.42761292],
       [18.96149313]])
```

Find mean and standard deviation of the new vector c.

In [235…

```
np.mean(c)
```

Out[235…

```
19.577200874901536
```

In [238…

```
m = 0
for i in range(c.shape[0]):
    m = m + c[i, 0]
mean = m/c.shape[0]
mean
```

Out[238…

```
19.577200874901536
```

In [239…

```
np.std(c)
```

Out[239…

```
2.575225023396272
```

In [245…

```
st = 0
for i in range(c.shape[0]):
    st = st + (c[i, 0] - mean)**2
st = np.sqrt(st/c.shape[0])
st
```

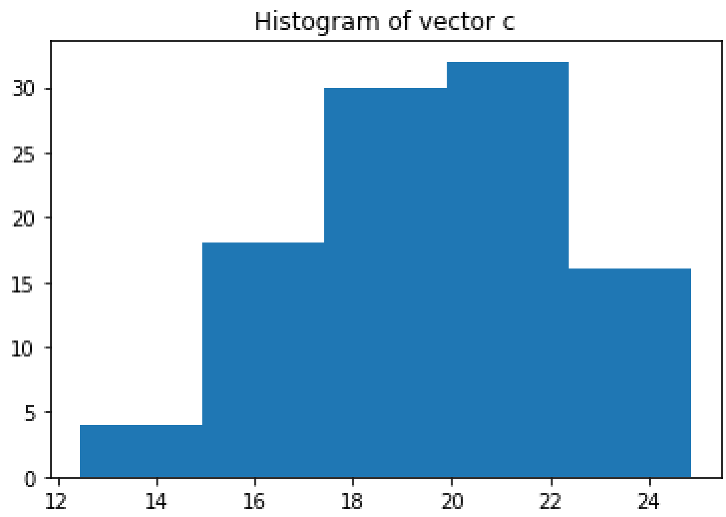
Out[245…

```
2.575225023396272
```

Plot the histogram of vector c using 5 bins.

In [246…

```
plt.hist(x = c, bins = 5)
plt.title('Histogram of 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.

In [263…

```
grades_file = pd.read_csv('Grades.csv')
grades_file
```

Out[263…

	First Name	Last Name	English	Maths	Science	German	Sports	Final Grade
0	Robyn	Hobgood	60.95	24.77	20.60	69.32	8.36	184.00
1	Eddy	Swearngin	100.00	12.99	100.00	52.24	100.00	365.23
2	Leoma	Bridgman	83.37	100.00	78.69	100.00	19.50	381.56
3	Arnetta	Peart	87.75	100.00	86.93	87.90	41.73	404.31
4	Maryland	Colby	100.00	100.00	100.00	18.87	88.72	407.59
5	Sherron	Sherron	92.06	55.91	93.93	-56.74	77.71	262.87
6	Glendora	Christopher	78.26	100.00	25.60	100.00	100.00	403.86
7	Darlena	Gunn	100.00	64.53	100.00	23.21	79.01	366.75
8	Aldo	Armas	100.00	83.49	100.00	100.00	92.32	475.81
9	Tiny	Jack	94.35	33.09	82.57	31.13	100.00	341.14
10	Carlton	Elms	100.00	36.52	5.54	33.82	12.07	187.95
11	Lauretta	Herbert	50.73	-0.10	67.76	100.00	55.98	274.37
12	Almeta	Dimond	80.37	100.00	69.02	100.00	79.62	429.01
13	Phoebe	Schill	100.00	70.37	100.00	47.00	77.37	394.74
14	Krystyna	Paris	18.75	73.80	87.00	59.30	100.00	338.85

	First Name	Last Name	English	Maths	Science	German	Sports	Final Grade
15	Miyoko	Laffoon	100.00	100.00	100.00	34.98	94.55	429.53
16	Rebecca	Duck	70.79	97.81	52.25	19.76	-13.93	226.68
17	Elwanda	Hyland	45.69	74.86	43.10	45.00	76.72	285.37
18	Gretchen	Kerrick	68.70	75.87	-14.87	57.32	84.28	271.30
19	Winnifred	Colonna	83.79	100.00	85.66	3.94	100.00	373.39
20	Gidget	Casseus	100.00	100.00	100.00	99.52	62.59	462.11
21	Elaina	Mcdougal	100.00	100.00	80.29	100.00	-36.81	343.48
22	Shoshana	Goldberger	55.10	100.00	100.00	100.00	100.00	455.10
23	Argentina	Nelson	100.00	100.00	40.44	100.00	100.00	440.44
24	Lyle	Millsaps	100.00	71.88	100.00	-17.33	100.00	354.55
25	Janay	Julius	41.70	100.00	55.90	75.53	100.00	373.13
26	Devorah	Heyden	0.84	18.61	50.14	83.58	65.68	218.85
27	Thelma	Romberger	72.41	61.00	100.00	76.38	51.37	361.16
28	Armanda	Hendley	35.40	66.92	69.67	71.08	-2.34	240.73
29	Raymon	Myerson	49.83	27.36	61.90	72.97	13.11	225.17

Compute the sum for all subjects for each student.

The final grade in the dataframe gives the sum of marks of each student. We will simply display that for sum of all subjects for each student.

In [264…

```
sum_marks = grades_file.iloc[:, [0, 1, -1]]
sum_marks
```

Out[264…

	First Name	Last Name	Final Grade
0	Robyn	Hobgood	184.00
1	Eddy	Swearngin	365.23
2	Leoma	Bridgman	381.56
3	Arnetta	Peart	404.31
4	Maryland	Colby	407.59
5	Sherron	Sherron	262.87
6	Glendora	Christopher	403.86
7	Darlena	Gunn	366.75
8	Aldo	Armas	475.81
9	Tiny	Jack	341.14
10	Carlton	Elms	187.95
11	Lauretta	Herbert	274.37
12	Almeta	Dimond	429.01
13	Phoebe	Schill	394.74

	First Name	Last Name	Final Grade
14	Krystyna	Paris	338.85
15	Miyoko	Laffoon	429.53
16	Rebecca	Duck	226.68
17	Elwanda	Hyland	285.37
18	Gretchen	Kerrick	271.30
19	Winnifred	Colonna	373.39
20	Gidget	Casseus	462.11
21	Elaina	Mcdougal	343.48
22	Shoshana	Goldberger	455.10
23	Argentina	Nelson	440.44
24	Lyle	Millsaps	354.55
25	Janay	Julius	373.13
26	Devorah	Heyden	218.85
27	Thelma	Romberger	361.16
28	Armanda	Hendley	240.73
29	Raymon	Myerson	225.17

Compute the average of the point for each student.

Average of marks of each student is the total marks divided by the total number of subjects.

In [265…

```
grades_file['average marks'] = grades_file['Final Grade']/5
grades_file.loc[:, ['First Name', 'Last Name', 'average marks']]
```

Out[265…

	First Name	Last Name	average marks
0	Robyn	Hobgood	36.800
1	Eddy	Swearngin	73.046
2	Leoma	Bridgman	76.312
3	Arnetta	Peart	80.862
4	Maryland	Colby	81.518
5	Sherron	Sherron	52.574
6	Glendora	Christopher	80.772
7	Darlena	Gunn	73.350
8	Aldo	Armas	95.162
9	Tiny	Jack	68.228
10	Carlton	Elms	37.590
11	Lauretta	Herbert	54.874
12	Almeta	Dimond	85.802
13	Phoebe	Schill	78.948

	First Name	Last Name	average marks
14	Krystyna	Paris	67.770
15	Miyoko	Laffoon	85.906
16	Rebecca	Duck	45.336
17	Elwanda	Hyland	57.074
18	Gretchen	Kerrick	54.260
19	Winnifred	Colonna	74.678
20	Gidget	Casseus	92.422
21	Elaina	Mcdougal	68.696
22	Shoshana	Goldberger	91.020
23	Argentina	Nelson	88.088
24	Lyle	Millsaps	70.910
25	Janay	Julius	74.626
26	Devorah	Heyden	43.770
27	Thelma	Romberger	72.232
28	Armanda	Hendley	48.146
29	Raymon	Myerson	45.034

Compute the standard deviation of point for each student.

In [270…

```
grades_file['Standard deviation'] = 0.0
m = grades_file['average marks'].mean()         # mean of the marks for the entire c
l = len(grades_file)
idx = 0
for r in grades_file.iterrows():
    std = np.sqrt(np.square(r[1]['average marks'] - m)/l)    # for each student the
    grades_file['Standard deviation'][idx] = std                     # from the mean
    idx += 1


grades_file
```

```
C:\Users\simra\AppData\Local\Temp/ipykernel_5432/2809124357.py:7: SettingWithCopyWar
ning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  grades_file['Standard deviation'][idx] = std                    # from the mean of
the entire class
```

Out[270…

	First Name	Last Name	English	Maths	Science	German	Sports	Final Grade	average marks	Standard deviation
0	Robyn	Hobgood	60.95	24.77	20.60	69.32	8.36	184.00	36.800	5.792507
1	Eddy	Swearngin	100.00	12.99	100.00	52.24	100.00	365.23	73.046	0.825077
2	Leoma	Bridgman	83.37	100.00	78.69	100.00	19.50	381.56	76.312	1.421364
3	Arnetta	Peart	87.75	100.00	86.93	87.90	41.73	404.31	80.862	2.252077

	First Name	Last Name	English	Maths	Science	German	Sports	Final Grade	average marks	Standard deviation
4	Maryland	Colby	100.00	100.00	100.00	18.87	88.72	407.59	81.518	2.371846
5	Sherron	Sherron	92.06	55.91	93.93	-56.74	77.71	262.87	52.574	2.912582
6	Glendora	Christopher	78.26	100.00	25.60	100.00	100.00	403.86	80.772	2.235645
7	Darlena	Gunn	100.00	64.53	100.00	23.21	79.01	366.75	73.350	0.880580
8	Aldo	Armas	100.00	83.49	100.00	100.00	92.32	475.81	95.162	4.862888
9	Tiny	Jack	94.35	33.09	82.57	31.13	100.00	341.14	68.228	0.054565
10	Carlton	Elms	100.00	36.52	5.54	33.82	12.07	187.95	37.590	5.648273
11	Lauretta	Herbert	50.73	-0.10	67.76	100.00	55.98	274.37	54.874	2.492661
12	Almeta	Dimond	80.37	100.00	69.02	100.00	79.62	429.01	85.802	3.153993
13	Phoebe	Schill	100.00	70.37	100.00	47.00	77.37	394.74	78.948	1.902630
14	Krystyna	Paris	18.75	73.80	87.00	59.30	100.00	338.85	67.770	0.138184
15	Miyoko	Laffoon	100.00	100.00	100.00	34.98	94.55	429.53	85.906	3.172981
16	Rebecca	Duck	70.79	97.81	52.25	19.76	-13.93	226.68	45.336	4.234054
17	Elwanda	Hyland	45.69	74.86	43.10	45.00	76.72	285.37	57.074	2.090998
18	Gretchen	Kerrick	68.70	75.87	-14.87	57.32	84.28	271.30	54.260	2.604762
19	Winnifred	Colonna	83.79	100.00	85.66	3.94	100.00	373.39	74.678	1.123038
20	Gidget	Casseus	100.00	100.00	100.00	99.52	62.59	462.11	92.422	4.362635
21	Elaina	Mcdougal	100.00	100.00	80.29	100.00	-36.81	343.48	68.696	0.030879
22	Shoshana	Goldberger	55.10	100.00	100.00	100.00	100.00	455.10	91.020	4.106666
23	Argentina	Nelson	100.00	100.00	40.44	100.00	100.00	440.44	88.088	3.571358
24	Lyle	Millsaps	100.00	71.88	100.00	-17.33	100.00	354.55	70.910	0.435099
25	Janay	Julius	41.70	100.00	55.90	75.53	100.00	373.13	74.626	1.113544
26	Devorah	Heyden	0.84	18.61	50.14	83.58	65.68	218.85	43.770	4.519965
27	Thelma	Romberger	72.41	61.00	100.00	76.38	51.37	361.16	72.232	0.676462
28	Armanda	Hendley	35.40	66.92	69.67	71.08	-2.34	240.73	48.146	3.721020
29	Raymon	Myerson	49.83	27.36	61.90	72.97	13.11	225.17	45.034	4.289191

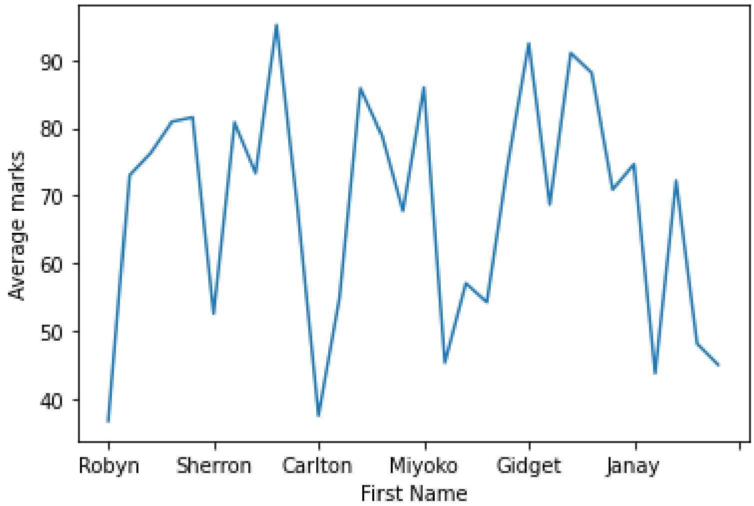
Plot the average points for all the students

In [271…

```
grades_file.plot('First Name', 'average marks',ylabel = 'Average marks', kind = 'lin
```

Out[271…

```
<AxesSubplot:xlabel='First Name', ylabel='Average marks'>
```



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

Firstly all the students are assigned grade F and then for each partition the grades are updated as shown below:

In [275...

```
list_grades = ['A+', 'A', 'A-', 'B+', 'B', 'B-', 'C+', 'C', 'D']
grades_file['Grade'] = 'F'
c = 0
while c <= 40:        # c is used to update scale as well as to assign grades
    for i, val in enumerate(grades_file['average marks']):
        if (val >= 95 - c)  and (val < 100 - c):
            grades_file.loc[i, 'Grade'] = list_grades[c//5]
    c += 5
```

In [276...

```
grades_file.loc[:, ['First Name', 'Last Name', 'average marks', 'Grade']]
```

Out[276...

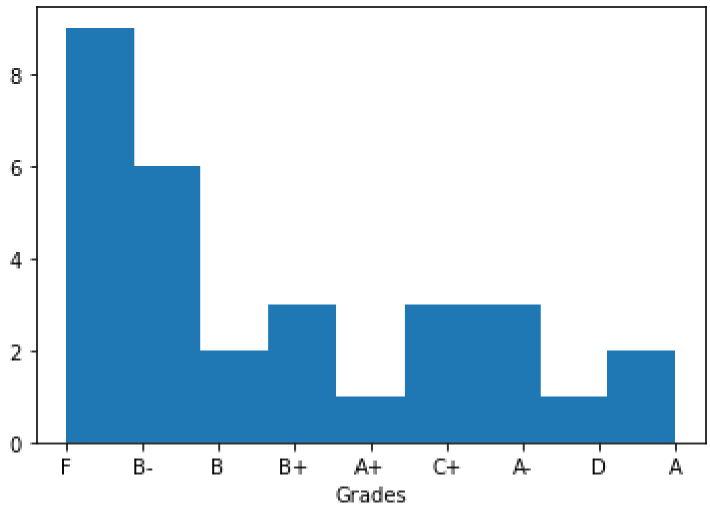
	First Name	Last Name	average marks	Grade
0	Robyn	Hobgood	36.800	F
1	Eddy	Swearngin	73.046	B-
2	Leoma	Bridgman	76.312	B
3	Arnetta	Peart	80.862	B+
4	Maryland	Colby	81.518	B+
5	Sherron	Sherron	52.574	F
6	Glendora	Christopher	80.772	B+
7	Darlena	Gunn	73.350	B-
8	Aldo	Armas	95.162	A+
9	Tiny	Jack	68.228	C+
10	Carlton	Elms	37.590	F
11	Lauretta	Herbert	54.874	F
12	Almeta	Dimond	85.802	A-
13	Phoebe	Schill	78.948	B
14	Krystyna	Paris	67.770	C+

	First Name	Last Name	average marks	Grade
15	Miyoko	Laffoon	85.906	A-
16	Rebecca	Duck	45.336	F
17	Elwanda	Hyland	57.074	D
18	Gretchen	Kerrick	54.260	F
19	Winnifred	Colonna	74.678	B-
20	Gidget	Casseus	92.422	A
21	Elaina	Mcdougal	68.696	C+
22	Shoshana	Goldberger	91.020	A
23	Argentina	Nelson	88.088	A-
24	Lyle	Millsaps	70.910	B-
25	Janay	Julius	74.626	B-
26	Devorah	Heyden	43.770	F
27	Thelma	Romberger	72.232	B-
28	Armanda	Hendley	48.146	F
29	Raymon	Myerson	45.034	F

Plot the histogram of the final grades.

In [277…

```
plt.hist(x = grades_file['Grade'], bins = 9)
plt.xlabel('Grades')
plt.show()
```



Exercise 2: Linear Regression through exact form.

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; a matrix A with dimensions 100 × 2. Initialize it with normal distribution μ = 2 and σ = [0.01, 0.1, 1].

In [278...

```
A_1 = np.random.normal(2, 0.01, (100,2))
A_2 = np.random.normal(2, 0.1, (100,2))
A_3 = np.random.normal(2, 1, (100,2))
```

Implement LEARN-SIMPLE-LINREG algorithm and train it using matrix A to learn values of β0 and β1.

In [279...

```
def LEARN_SIMPLE_LINREG(A):
    n = A.shape[0]
    X = A[:, 0]         # features
    Y = A[:, 1]               # targets
    X_avg = np.sum(X)/n
    Y_avg = np.sum(Y)/n
    beta_1 = np.sum(np.multiply((X - X_avg), (Y - Y_avg)))/(np.sum((X - X_avg)**2))
    beta_0 = Y_avg - beta_1*X_avg
    return beta_0, beta_1
```

Implement PREDICT-SIMPLE-LINREG and calculate the points for each training example in matrix A.

In [280...

```
def PREDICT_SIMPLE_LINREG(A, b0, b1):
    X = A[:, 0]
    Y_pred = b0 + np.dot(b1, X)
    return Y_pred
```

Plot the training data (use plt.scatter) and your predicted line (use plt.plot).

For the first dataset:

In [290...

```
b0_A1, b1_A1 = LEARN_SIMPLE_LINREG(A_1)
Y_1_pred = PREDICT_SIMPLE_LINREG(A_1, b0_A1, b1_A1)
print(f'beta0 = {b0_A1} and beta1 = {b1_A1}')
print('-----------------------------')
print(f'Y_predicted = {Y_1_pred}')
```

```
beta0 = 1.923684237848655 and beta1 = 0.03837470190577874
-----------------------------
Y_predicted = [1.99995191 2.0009827  1.9996912  2.00006963 2.00054336 2.00154301
 2.00074331 2.00046842 2.00062101 2.00035024 2.00048925 2.00059017
 1.9997046  2.00030794 2.00086557 2.00044604 2.00070651 2.00002639
 2.00037979 1.99956866 2.0002913  2.00013733 2.00082964 2.00091804
 2.0009935  1.99965261 2.00041153 2.00066764 2.00063236 2.00024714
 2.00124337 2.00067998 2.00042023 1.99979451 2.00079582 2.00039211
 2.00072855 2.00027175 2.00033195 2.000918   2.00124402 2.00044738
 2.00044442 2.00213268 1.99988451 2.00075523 2.0000039  2.00060307
 2.00037822 2.00029989 2.00025786 2.00040262 1.99994936 2.0006458
 2.00030275 1.99960359 2.00073735 2.00074339 2.00110239 2.00016394
 2.00066464 1.99993387 2.00100082 2.00051055 2.00083211 2.00054143
 2.00031249 2.000533   2.00064803 2.00083993 1.99998697 2.00062777
 2.00011888 2.00079006 2.00035448 2.00015412 2.00095413 2.00018341
 2.00046183 2.00092143 2.00155915 2.00012415 2.00029083 2.00122548
 2.00007761 2.00039272 2.00059908 2.00028319 2.00070645 2.00030839
 2.00073094 2.00093737 1.99985126 2.00062223 2.00066359 2.0006622
 2.00065997 2.00043068 2.00050888 2.00076014]
```

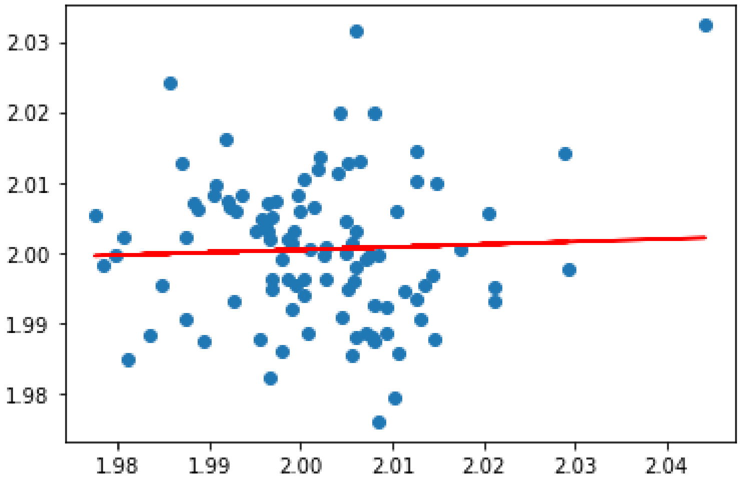
In [283...

```
X_1 = A_1[:, 0]
```

```
Y_1 = A_1[:, 1]
plt.scatter(X_1, Y_1)
plt.plot(X_1, Y_1_pred, 'r')
```

Out[283…

```
[<matplotlib.lines.Line2D at 0x24302df0d90>]
```



For the second dataset:

In [291…

```
b0_A2, b1_A2 = LEARN_SIMPLE_LINREG(A_2)
Y_2_pred = PREDICT_SIMPLE_LINREG(A_2, b0_A2, b1_A2)
print(f'beta0 = {b0_A2} and beta1 = {b1_A2}')
print('-----------------------------')
print(f'Y_predicted = {Y_2_pred}')
```

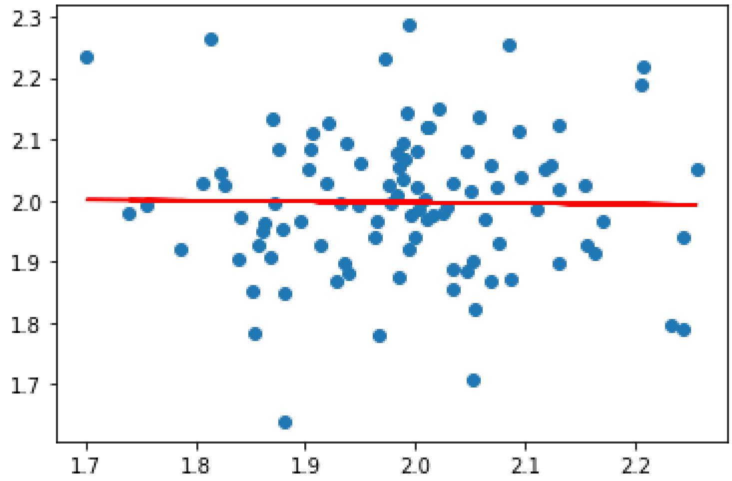
```
beta0 = 2.02628840091477 and beta1 = -0.014729069654950263
-----------------------------
Y_predicted = [1.99341134 1.99599065 1.998023   1.99489795 1.99871423 1.99826652
 1.99659908 1.99570436 2.00044655 1.99809795 1.99379935 1.996146
 1.99681993 1.99541782 1.99580475 1.99323492 1.99508658 1.99430314
 1.99774218 2.00123335 1.9973672  1.99876856 1.99837054 1.99520326
 1.99874573 1.99716191 1.99489783 1.99574457 1.99695922 1.99938477
 1.99718211 1.99968102 1.99734039 1.99555843 1.9988796  2.00066766
 1.99666954 1.99706927 1.9937867  1.9968819  1.99441469 1.99902407
 1.99859511 1.99323803 1.9999913  1.9969301  1.99633658 1.99683156
 1.99604843 1.99581262 1.99943797 1.99899407 1.99782845 1.99642048
 1.99861172 1.99604654 1.99778596 1.99706194 1.99650744 1.99759882
 1.99668673 1.99917735 1.99664171 1.99798005 1.99704878 1.99632148
 1.99819716 1.99885004 1.99645767 1.99454423 1.99607425 1.9995669
 1.99787888 1.9961319  1.99824492 1.99694224 1.99543443 1.99857536
 1.99921045 1.99677775 1.99558179 1.99681784 1.99865845 1.99590273
 1.99698683 1.99707842 1.99305514 1.99724704 1.99732372 1.99773421
 1.99490605 1.99699604 1.99691655 1.99757683 1.99501031 1.99632665
 1.9945203  1.99602363 1.99892098 1.99671063]
```

In [292…

```
X_2 = A_2[:, 0]
Y_2 = A_2[:, 1]
plt.scatter(X_2, Y_2)
plt.plot(X_2, Y_2_pred, 'r')
```

Out[292…

```
[<matplotlib.lines.Line2D at 0x24302f2e310>]
```



For the third dataset:

In [293…

```
b0_A3, b1_A3 = LEARN_SIMPLE_LINREG(A_3)
Y_3_pred = PREDICT_SIMPLE_LINREG(A_3, b0_A3, b1_A3)
print(f'beta0 = {b0_A3} and beta1 = {b1_A3}')
print('-----------------------------')
print(f'Y_predicted = {Y_3_pred}')
```

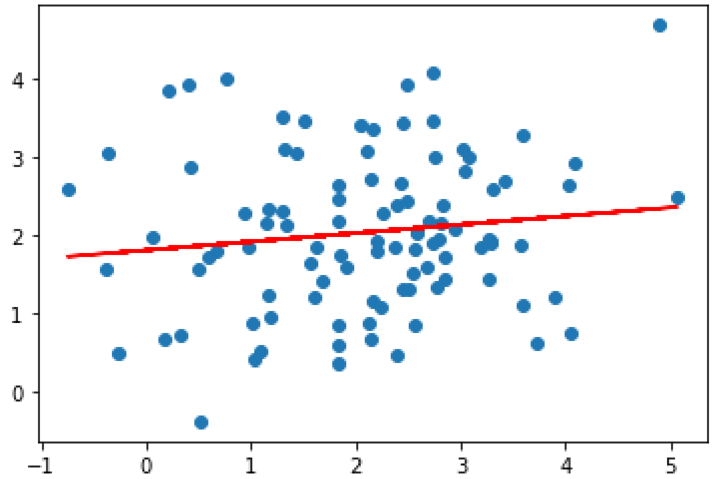
```
beta0 = 1.808230231537007 and beta1 = 0.10916908808287593
-----------------------------
Y_predicted = [2.07533362 1.72651341 2.11969467 2.11666537 1.93417398 2.10326062
 2.08037912 2.08526105 1.8428322  2.10722859 2.04741597 2.24739028
 2.06993549 2.04419254 1.97383957 2.01699452 2.09936595 1.88150931
 1.89202332 2.16823381 2.2148579  1.95106634 1.96422482 2.16394707
 2.00948327 2.00789559 1.86556012 1.95297903 2.11378632 1.9110164
 2.05130539 2.14082455 2.10600211 2.2495001  1.93682294 1.93549653
 2.0389442  2.1673048  2.04219972 2.20039477 2.11061466 1.98260496
 2.08011081 1.85521138 1.91805181 2.00880269 2.08198481 1.99152886
 1.95091319 2.0077893  2.10670164 2.06663726 1.76952938 2.16415728
 2.08917519 2.25535727 1.8139084  2.01071593 2.19825355 2.04301551
 2.11834466 2.06819308 2.1441392  1.93597323 2.10831354 1.91467395
 1.92711034 2.08863211 1.92137034 2.0741417  2.07520936 2.04851537
 2.04228659 2.36133728 2.13040748 1.87379555 2.34222796 1.77862993
 2.03227777 2.13840766 1.95378461 2.16654351 1.83199334 2.05514734
 2.00925992 1.82780393 2.00938812 2.09075284 1.85122384 1.76759016
 2.2345567  2.19975195 2.11547758 2.03864834 2.15602309 1.98486798
 2.18143224 1.97948497 1.86216744 2.08799992]
```

In [294…

```
X_3 = A_3[:, 0]
Y_3 = A_3[:, 1]
plt.scatter(X_3, Y_3)
plt.plot(X_3, Y_3_pred, 'r')
```

Out[294…

```
[<matplotlib.lines.Line2D at 0x24302f991f0>]
```



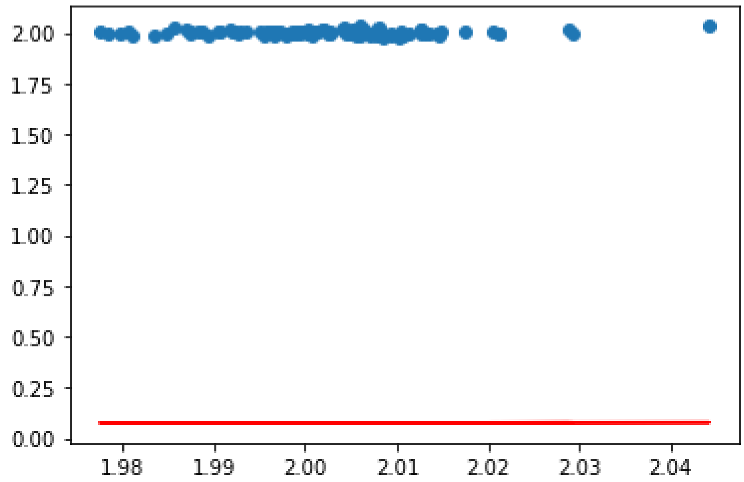
Put β0 to zero and rerun the program to generate the predicted line. Comment on the change you see for the varying values of σ

In [295…

```
Y1_pred = PREDICT_SIMPLE_LINREG(A_1, 0, b1_A1)
plt.scatter(X_1, Y_1)
plt.plot(X_1, Y1_pred, 'r')
```

Out[295…

```
[<matplotlib.lines.Line2D at 0x24304016700>]
```

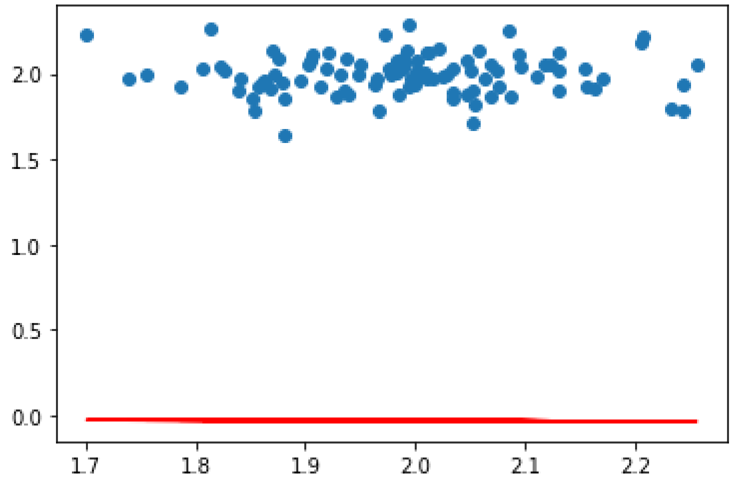


In [296…

```
Y2_pred = PREDICT_SIMPLE_LINREG(A_2, 0, b1_A2)
plt.scatter(X_2, Y_2)
plt.plot(X_2, Y2_pred, 'r')
```

Out[296…

```
[<matplotlib.lines.Line2D at 0x243040288b0>]
```

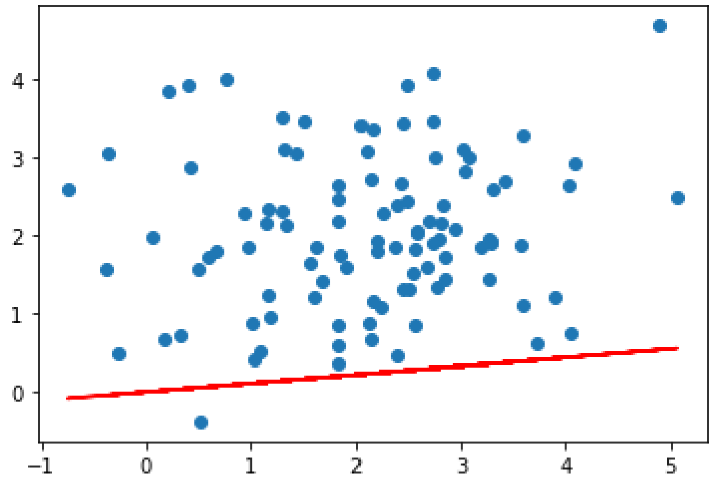


In [297…

```
Y3_pred = PREDICT_SIMPLE_LINREG(A_3, 0, b1_A3)
plt.scatter(X_3, Y_3)
plt.plot(X_3, Y3_pred, 'r')
```

Out[297…

```
[<matplotlib.lines.Line2D at 0x24302cefd60>]
```



Putting beta0 = 0 makes prediction over all dataset bad but as the variance increases from 0.01 to 1, the dataset is more scattered and hence we can observe that the predicted line moves slowly towards the datapoints with increasing values of variance.

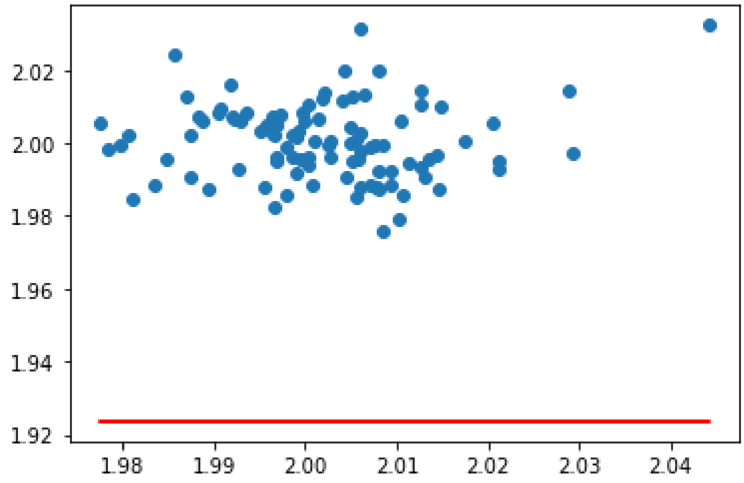
Put β1 to zero and rerun the program to generate the predicted line. Comment on the change you see for the varying values of σ

In [298…

```
Y1_predict = PREDICT_SIMPLE_LINREG(A_1, b0_A1, 0)
plt.scatter(X_1, Y_1)
plt.plot(X_1, Y1_predict, 'r')
```

Out[298…

```
[<matplotlib.lines.Line2D at 0x24302deaaf0>]
```

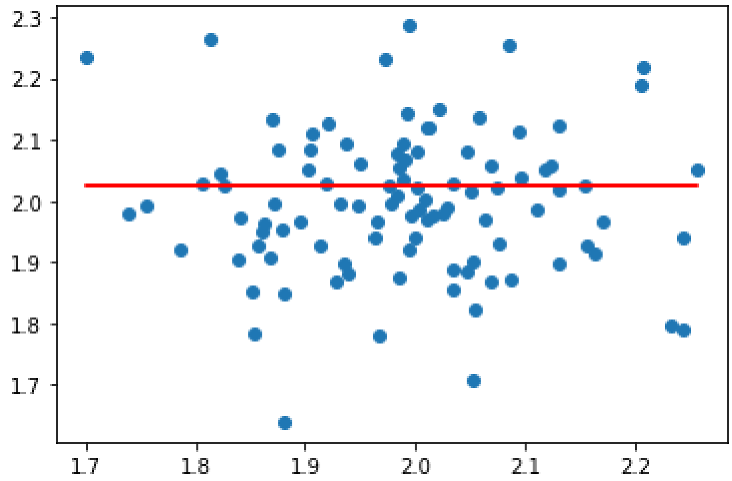


In [299…

```
Y2_predict = PREDICT_SIMPLE_LINREG(A_2, b0_A2, 0)
plt.scatter(X_2, Y_2)
plt.plot(X_2, Y2_predict, 'r')
```

Out[299…

```
[<matplotlib.lines.Line2D at 0x24302dc2730>]
```

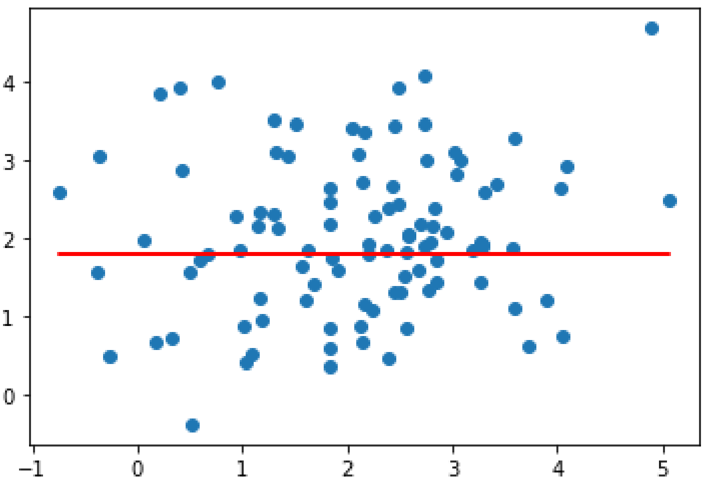


In [300…

```
Y3_predict = PREDICT_SIMPLE_LINREG(A_3, b0_A3, 0)
plt.scatter(X_3, Y_3)
plt.plot(X_3, Y3_predict, 'r')
```

Out[300…

```
[<matplotlib.lines.Line2D at 0x24302ab2e20>]
```



Putting beta1 = 0 again makes prediction over all dataset bad but as the variance increases from

0.01 to 1, the dataset is more scattered and hence we can observe that the predicted line moves slowly towards the datapoints with increasing values of variance.

Use numpy.linalg lstsq to replace step 2 for learning values of β0 and β1. Explain the difference between your values and the values from numpy.linalg lstsq.

For the first dataset:

In [303...
```
bias_column = np.ones(shape=(100,1))
A1 = np.append(bias_column, A_1, axis=1)
X1 = A1[:, :-1]
np.linalg.lstsq(X1, Y_1, rcond = None)[0]
```

Out[303...
```
array([1.92368424, 0.0383747 ])
```

In [302...
```
b0_A1, b1_A1
```

Out[302...
```
(1.923684237848655, 0.03837470190577874)
```

For the second dataset:

In [307...
```
A2 = np.append(bias_column, A_2, axis=1)
X2 = A2[:, :-1]
np.linalg.lstsq(X2, Y_2, rcond = None)[0]
```

Out[307...
```
array([ 2.0262884 , -0.01472907])
```

In [308...
```
b0_A2, b1_A2
```

Out[308...
```
(2.02628840091477, -0.014729069654950263)
```

For the third dataset:

In [309...
```
A3 = np.append(bias_column, A_3, axis=1)
X3 = A3[:, :-1]
np.linalg.lstsq(X3, Y_3, rcond = None)[0]
```

Out[309...
```
array([1.80823023, 0.10916909])
```

In [310...
```
b0_A3, b1_A3
```

Out[310...
```
(1.808230231537007, 0.10916908808287593)
```

The difference in the value of beta's is very small when compared with manually calculated and the one using library because the beta's calculated using library solves the system of equations approximatedly while the other one is the exact solution.