MACHINE LEARNING LAB - TUTORIAL 1

Juan Fernando Espinosa

303158

EXERCISE SHEET 1

1. First step is to read the database and transform it into a dataframe to make it possible to edit (add/erase,etc) columns of the database.

```
import pandas as pd
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
from google.colab import drive
drive.mount('/content/drive')
!ls "/content/drive/My Drive/MASTER DATA ANALYTICS/SEMESTER 2/Lab ML/Grades.csv"
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remou
'/content/drive/My Drive/MASTER DATA ANALYTICS/SEMESTER 2/Lab ML/Grades.csv'
```

```
grades=pd.read_csv('/content/drive/My Drive/MASTER DATA ANALYTICS/SEMESTER 2/Lab ML/Grades.csv')
grade = pd.DataFrame(grades)
```

2. Check the columns for a better understanding of the database distribution.

```
grade.columns
grade.drop("Final Grade", axis=1)
grade = grade.round()
```

3. Check if the database does not have empty fields.

```
check = grade.empty
print(check)
```

```
False
```

1. 1. PYTHON AND NUMPY

1. 1. 1. **Grading Program**

1. 1. 1. 1. Sum of the grades for each student

the sum of the grades of each students considering the courses: English, Math. Science, German and Sports are allocated in a new column called **sum**

```
grade['sum']=grade[['English', ' Maths', 'Science', 'German','Sports' ]].sum(axis=1)
grade
grade[['First Name','Last Name','English',' Maths', 'Science', 'German','Sports','sum']]
```

	First Name	Last Name	English	Maths	Science	German	Sports	sum
0	Robyn	Hobgood	61.0	25.0	21.0	69.0	8.0	184.0
1	Eddy	Swearngin	100.0	13.0	100.0	52.0	100.0	365.0
2	Leoma	Bridgman	83.0	100.0	79.0	100.0	20.0	382.0
3	Arnetta	Peart	88.0	100.0	87.0	88.0	42.0	405.0
4	Maryland	Colby	100.0	100.0	100.0	19.0	89.0	408.0
5	Sherron	Sherron	92.0	56.0	94.0	-57.0	78.0	263.0
6	Glendora	Christopher	78.0	100.0	26.0	100.0	100.0	404.0
7	Darlena	Gunn	100.0	65.0	100.0	23.0	79.0	367.0
8	Aldo	Armas	100.0	83.0	100.0	100.0	92.0	475.0
9	Tiny	Jack	94.0	33.0	83.0	31.0	100.0	341.0
10	Carlton	Elms	100.0	37.0	6.0	34.0	12.0	189.0
11	Lauretta	Herbert	51.0	-0.0	68.0	100.0	56.0	275.0
12	Almeta	Dimond	80.0	100.0	69.0	100.0	80.0	429.0
13	Phoebe	Schill	100.0	70.0	100.0	47.0	77.0	394.0
14	Krystyna	Paris	19.0	74.0	87.0	59.0	100.0	339.0
15	Miyoko	Laffoon	100.0	100.0	100.0	35.0	95.0	430.0
16	Rebecca	Duck	71.0	98.0	52.0	20.0	-14.0	227.0
17	Elwanda	Hyland	46.0	75.0	43.0	45.0	77.0	286.0
18	Gretchen	Kerrick	69.0	76.0	-15.0	57.0	84.0	271.0
19	Winnifred	Colonna	84.0	100.0	86.0	4.0	100.0	374.0
20	Gidget	Casseus	100.0	100.0	100.0	100.0	63.0	463.0
21	Elaina	Mcdougal	100.0	100.0	80.0	100.0	-37.0	343.0
22	Shoshana	Goldberger	55.0	100.0	100.0	100.0	100.0	455.0
23	Argentina	Nelson	100.0	100.0	40.0	100.0	100.0	440.0
24	Lyle	Millsaps	100.0	72.0	100.0	-17.0	100.0	355.0
25	Janay	Julius	42.0	100.0	56.0	76.0	100.0	374.0
26	Devorah	Heyden	1.0	19.0	50.0	84.0	66.0	220.0
27	Thelma	Romberger	72.0	61.0	100.0	76.0	51.0	360.0
28	Armanda	Hendley	35.0	67.0	70.0	71.0	-2.0	241.0
29	Raymon	Myerson	50.0	27.0	62.0	73.0	13.0	225.0

1. 1. 1. 2. Average of the grades of each student

Similar to the previous exercise, a new column is created called **Average**

```
grade['Average']=grade[['English', ' Maths', 'Science', 'German','Sports' ]].mean(axis=1)
grade.Average = grade.Average.round()
grade[['First Name','Last Name','English',' Maths', 'Science', 'German','Sports','sum','Average']]
```

	First Name	Last Name	English	Maths	Science	German	Sports	sum	Average
0	Robyn	Hobgood	61.0	25.0	21.0	69.0	8.0	184.0	37.0
1	Eddy	Swearngin	100.0	13.0	100.0	52.0	100.0	365.0	73.0
2	Leoma	Bridgman	83.0	100.0	79.0	100.0	20.0	382.0	76.0
3	Arnetta	Peart	88.0	100.0	87.0	88.0	42.0	405.0	81.0
4	Maryland	Colby	100.0	100.0	100.0	19.0	89.0	408.0	82.0
5	Sherron	Sherron	92.0	56.0	94.0	-57.0	78.0	263.0	53.0
6	Glendora	Christopher	78.0	100.0	26.0	100.0	100.0	404.0	81.0
7	Darlena	Gunn	100.0	65.0	100.0	23.0	79.0	367.0	73.0
8	Aldo	Armas	100.0	83.0	100.0	100.0	92.0	475.0	95.0
9	Tiny	Jack	94.0	33.0	83.0	31.0	100.0	341.0	68.0
10	Carlton	Elms	100.0	37.0	6.0	34.0	12.0	189.0	38.0
11	Lauretta	Herbert	51.0	-0.0	68.0	100.0	56.0	275.0	55.0
12	Almeta	Dimond	80.0	100.0	69.0	100.0	80.0	429.0	86.0
13	Phoebe	Schill	100.0	70.0	100.0	47.0	77.0	394.0	79.0
14	Krystyna	Paris	19.0	74.0	87.0	59.0	100.0	339.0	68.0
15	Miyoko	Laffoon	100.0	100.0	100.0	35.0	95.0	430.0	86.0
16	Rebecca	Duck	71.0	98.0	52.0	20.0	-14.0	227.0	45.0
17	Elwanda	Hyland	46.0	75.0	43.0	45.0	77.0	286.0	57.0
18	Gretchen	Kerrick	69.0	76.0	-15.0	57.0	84.0	271.0	54.0
19	Winnifred	Colonna	84.0	100.0	86.0	4.0	100.0	374.0	75.0
20	Gidget	Casseus	100.0	100.0	100.0	100.0	63.0	463.0	93.0
21	Elaina	Mcdougal	100.0	100.0	80.0	100.0	-37.0	343.0	69.0
22	Shoshana	Goldberger	55.0	100.0	100.0	100.0	100.0	455.0	91.0
23	Argentina	Nelson	100.0	100.0	40.0	100.0	100.0	440.0	88.0
24	Lyle	Millsaps	100.0	72.0	100.0	-17.0	100.0	355.0	71.0
25	Janay	Julius	42.0	100.0	56.0	76.0	100.0	374.0	75.0
26	Devorah	Heyden	1.0	19.0	50.0	84.0	66.0	220.0	44.0
27	Thelma	Romberger	72.0	61.0	100.0	76.0	51.0	360.0	72.0
28	Armanda	Hendley	35.0	67.0	70.0	71.0	-2.0	241.0	48.0
29	Raymon	Myerson	50.0	27.0	62.0	73.0	13.0	225.0	45.0

1. 1. 1. 3. Standard deviation for the grades for each student

the new column is called **STD**

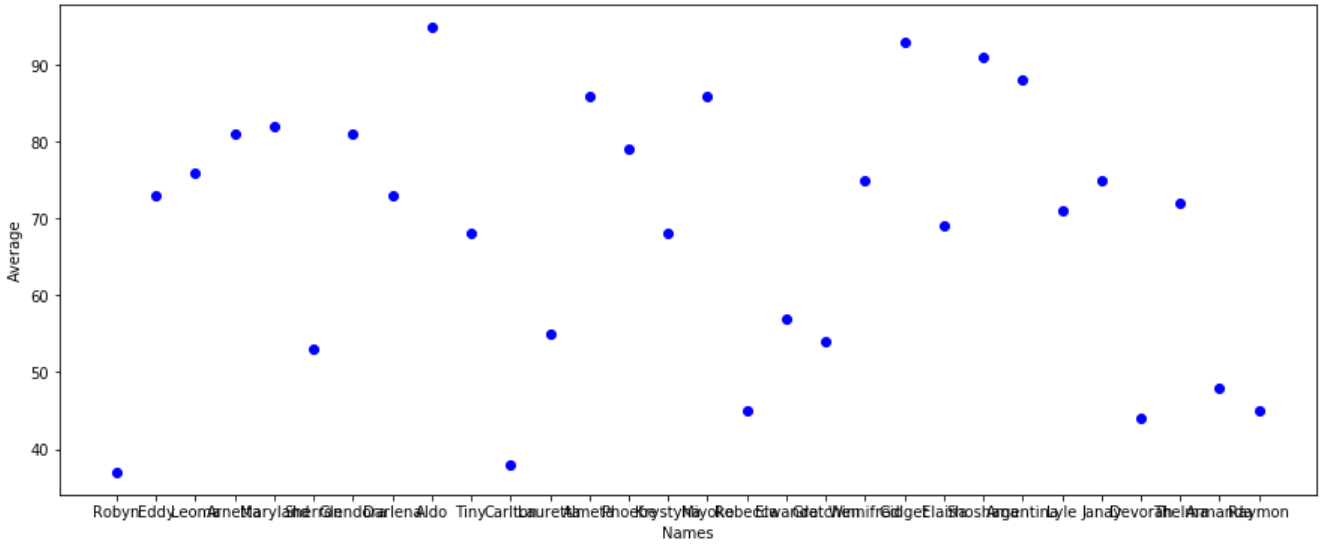
```
grade['STD']=grade[['English', ' Maths', 'Science', 'German','Sports' ]].std(axis=1)
grade
grade[['First Name','Last Name','English',' Maths', 'Science', 'German','Sports','sum','Average', 'STD']]
```

	First Name	Last Name	English	Maths	Science	German	Sports	sum	Average	STD
0	Robyn	Hobgood	61.0	25.0	21.0	69.0	8.0	184.0	37.0	26.649578
1	Eddy	Swearngin	100.0	13.0	100.0	52.0	100.0	365.0	73.0	39.458839
2	Leoma	Bridgman	83.0	100.0	79.0	100.0	20.0	382.0	76.0	32.959066
3	Arnetta	Peart	88.0	100.0	87.0	88.0	42.0	405.0	81.0	22.449944
4	Maryland	Colby	100.0	100.0	100.0	19.0	89.0	408.0	82.0	35.317135
5	Sherron	Sherron	92.0	56.0	94.0	-57.0	78.0	263.0	53.0	63.117351
6	Glendora	Christopher	78.0	100.0	26.0	100.0	100.0	404.0	81.0	32.081147
7	Darlena	Gunn	100.0	65.0	100.0	23.0	79.0	367.0	73.0	31.848077
8	Aldo	Armas	100.0	83.0	100.0	100.0	92.0	475.0	95.0	7.549834
9	Tiny	Jack	94.0	33.0	83.0	31.0	100.0	341.0	68.0	33.611010
10	Carlton	Elms	100.0	37.0	6.0	34.0	12.0	189.0	38.0	37.285386
11	Lauretta	Herbert	51.0	-0.0	68.0	100.0	56.0	275.0	55.0	36.180105
12	Almeta	Dimond	80.0	100.0	69.0	100.0	80.0	429.0	86.0	13.718601
13	Phoebe	Schill	100.0	70.0	100.0	47.0	77.0	394.0	79.0	22.309191
14	Krystyna	Paris	19.0	74.0	87.0	59.0	100.0	339.0	68.0	31.236197
15	Miyoko	Laffoon	100.0	100.0	100.0	35.0	95.0	430.0	86.0	28.591957
16	Rebecca	Duck	71.0	98.0	52.0	20.0	-14.0	227.0	45.0	43.701259
17	Elwanda	Hyland	46.0	75.0	43.0	45.0	77.0	286.0	57.0	17.210462
18	Gretchen	Kerrick	69.0	76.0	-15.0	57.0	84.0	271.0	54.0	39.933695
19	Winnifred	Colonna	84.0	100.0	86.0	4.0	100.0	374.0	75.0	40.288956
20	Gidget	Casseus	100.0	100.0	100.0	100.0	63.0	463.0	93.0	16.546903
21	Elaina	Mcdougal	100.0	100.0	80.0	100.0	-37.0	343.0	69.0	59.664060
22	Shoshana	Goldberger	55.0	100.0	100.0	100.0	100.0	455.0	91.0	20.124612
23	Argentina	Nelson	100.0	100.0	40.0	100.0	100.0	440.0	88.0	26.832816
24	Lyle	Millsaps	100.0	72.0	100.0	-17.0	100.0	355.0	71.0	50.665570
25	Janay	Julius	42.0	100.0	56.0	76.0	100.0	374.0	75.0	25.984611
26	Devorah	Heyden	1.0	19.0	50.0	84.0	66.0	220.0	44.0	33.889526
27	Thelma	Romberger	72.0	61.0	100.0	76.0	51.0	360.0	72.0	18.452642
28	Armanda	Hendley	35.0	67.0	70.0	71.0	-2.0	241.0	48.0	31.791508
29	Raymon	Myerson	50.0	27.0	62.0	73.0	13.0	225.0	45.0	24.728526

1. 1. 1. 4. Plot the average points of all the students

For plotting the graph, the columns used are: *first name* and average

```
rng = np.random.RandomState(0)
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 15
fig_size[1] = 6
colors1 = 'blue'
plt.rcParams["figure.figsize"] = fig_size
plt.ylabel('Average')
plt.xlabel('Names')
plt.scatter(grade['First Name'], grade['Average'], c=colors1)
plt.show()
```



1. 1. 1. 5. Assign a grade based on the following rubric

In consideration to the image shared, several conditions have to me met to assign each student to the ideal grade.

```
conditions = [
    (grade['Average'] >= 96.0),
    (grade['Average'] >= 90.0) & (grade['Average'] < 96.0),
    (grade['Average'] >= 86.0) & (grade['Average'] < 90.0),
    (grade['Average'] >= 80.0) & (grade['Average'] < 86.0),
    (grade['Average'] >= 76.0) & (grade['Average'] < 80.0),
    (grade['Average'] >= 70.0) & (grade['Average'] < 76.0),
    (grade['Average'] >= 66.0) & (grade['Average'] < 70.0),
    (grade['Average'] >= 60.0) & (grade['Average'] < 66.0),
    (grade['Average'] >= 56.0) & (grade['Average'] < 60.0),
    (grade['Average'] >= 0.0) & (grade['Average'] <= 55.0)]
gradingSystem = ['A+', 'A', 'A-', 'B+', 'B', 'B-', 'C+', 'C', 'D', 'F']
grade['finalGrade'] = np.select(conditions, gradingSystem)

grade[['First Name','Last Name','STD','finalGrade']]
```

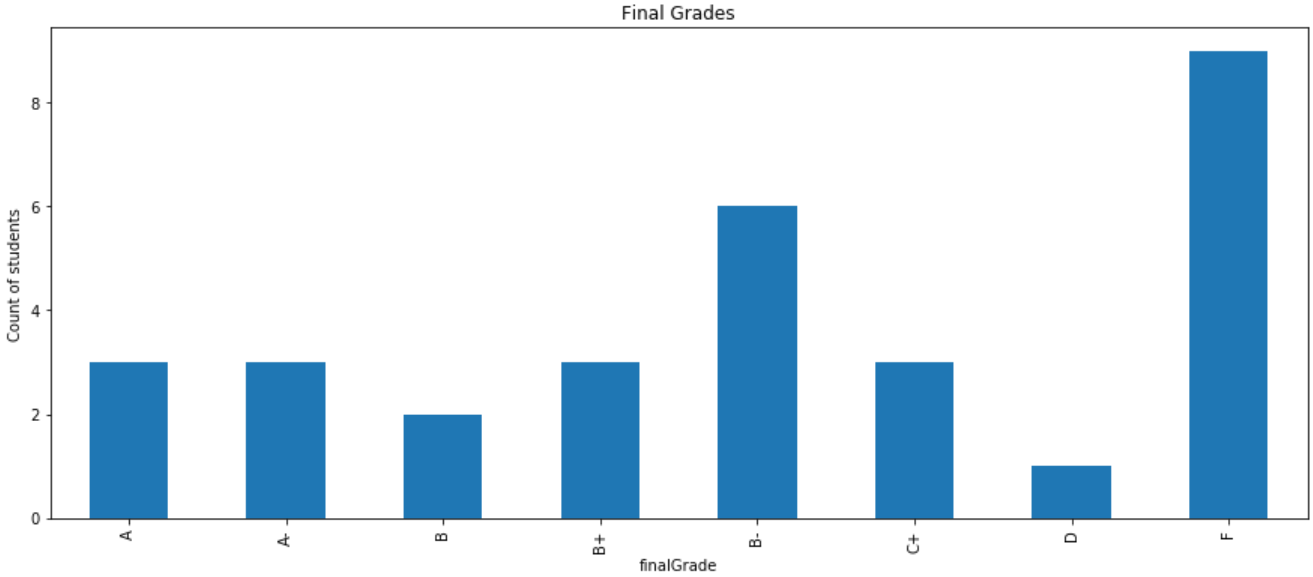
	First Name	Last Name	STD	finalGrade
0	Robyn	Hobgood	26.649578	F
1	Eddy	Swearngin	39.458839	B-
2	Leoma	Bridgman	32.959066	B
3	Arnetta	Peart	22.449944	B+
4	Maryland	Colby	35.317135	B+
5	Sherron	Sherron	63.117351	F
6	Glendora	Christopher	32.081147	B+
7	Darlena	Gunn	31.848077	B-
8	Aldo	Armas	7.549834	A
9	Tiny	Jack	33.611010	C+
10	Carlton	Elms	37.285386	F
11	Lauretta	Herbert	36.180105	F
12	Almeta	Dimond	13.718601	A-
13	Phoebe	Schill	22.309191	B
14	Krystyna	Paris	31.236197	C+
15	Miyoko	Laffoon	28.591957	A-
16	Rebecca	Duck	43.701259	F
17	Elwanda	Hyland	17.210462	D
18	Gretchen	Kerrick	39.933695	F
19	Winnifred	Colonna	40.288956	B-
20	Gidget	Casseus	16.546903	A
21	Elaina	Mcdougal	59.664060	C+
22	Shoshana	Goldberger	20.124612	A
23	Argentina	Nelson	26.832816	A-
24	Lyle	Millsaps	50.665570	B-
25	Janay	Julius	25.984611	B-
26	Devorah	Heyden	33.889526	F
27	Thelma	Romberger	18.452642	B-
28	Armanda	Hendley	31.791508	F
29	Raymon	Myerson	24.728526	F

1. 1. 1. 6. Plot the histogram of the final grades

```
grade['count'] = grade[['finalGrade']].count(axis=1)
```

```
studentsGrades = grade.pivot_table(index=['finalGrade'], aggfunc='size')
print(studentsGrades)

fig, ax = plt.subplots(figsize=(15,6))
plt.xlabel('Final Grade')
plt.ylabel('Count of students')
grade.groupby(['finalGrade']).count()['count'].plot(kind='bar',title="Final Grades")
```

```
finalGrade
A     3
A-    3
B     2
B+    3
B-    6
C+    3
D     1
F     9
dtype: int64
<matplotlib.axes._subplots.AxesSubplot at 0x7fed24caba58>
```



1. 1. 2. Matrix Multiplication

First, intialization of the matrix **A** considering the mean and standard deviation. Moreover, adding a random vector **v**

```
import numpy as np
A = np.random.normal(loc =5, scale =0.01, size=(100, 20))
v = np.random.normal(loc =5, scale =0.01, size=(20, 1))
```

1. 1. 2. 1. Iterative multiply (element-wise) each row of matrix A with vector v and sum the result of each iteration in another vector **C**

```
c = []
for i in range(len(A)):
  rowA = A[i]
  for j in range(len(v[0])):
    result = 0
    for k in range(len(v)):
      numA = rowA[k]
      numB = v[k,j]
      result += numA * numB
  c.append(result)
print('vector c:',[c])
```

```
vector c: [[500.0403788702787, 499.7953044098863, 499.79150877827095, 499.6504220113477, 499.5720262057038, 499.705466
```

1. 1. 2. 2. Find the mean and Standard deviation of vector **v**

Mean:

```
def mean(xs: [float]):
    return sum(xs) / len(xs)

print('Mean:',mean(c))
```

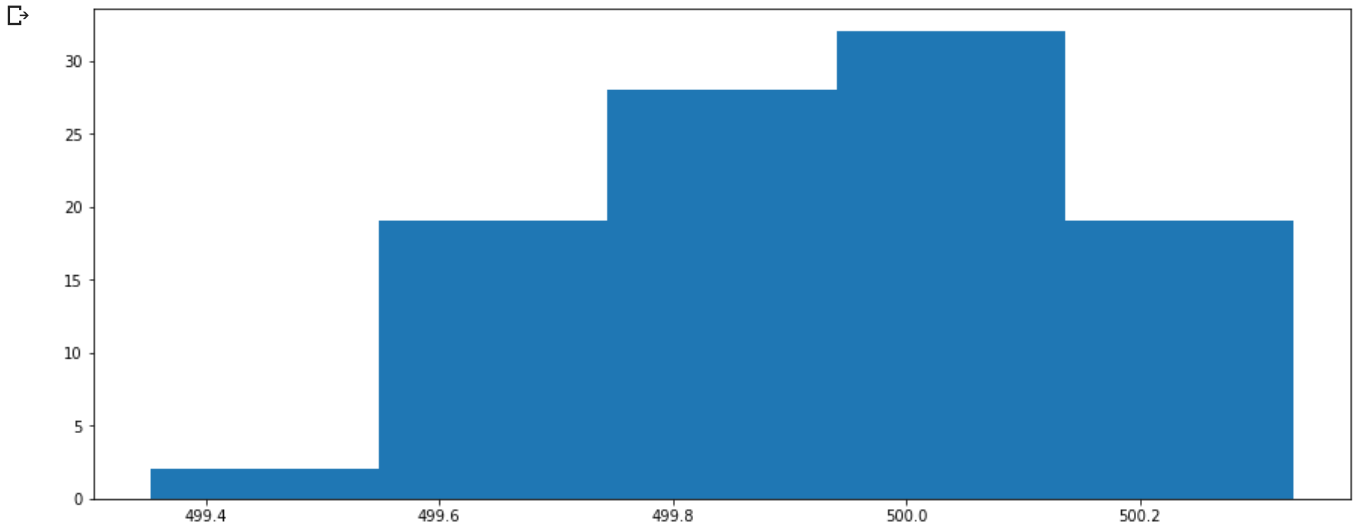
```
Mean: 499.93294711232124
```

Standard deviation:

```
import math
suma = 0

for i in range(len(c)):
  numerator = (c[i] - mean(c))**2
  suma += numerator
  denominator = len(c) - 1
  division = suma / denominator
  total = math.sqrt(division)
print('Standard Deviation:',total)
```

1. 2. 2. 3. Plot histogram of vector **c** with 5 bins

```
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 15
fig_size[1] = 6
plt.rcParams["figure.figsize"] = fig_size
plt.hist(c, 5)
plt.show()
```



1. 2. Linear Regression through exact form

1. 2. 1. OLS

---

1. 2. 1. 1. Implement Matrix A with random values.

First, initilization of a random matrix **A** with the dimensions, mean, and standard deviation assigned.

```
A1 = np.random.normal(loc =2, scale =1, size=(100, 2))
A1
```

```
array([[ 1.73483847,  0.42629941],
       [ 1.58559368,  4.5971687 ],
       [ 0.64502245,  1.48951226],
       [ 3.35072478,  1.04611393],
       [ 2.21690047,  2.27124323],
       [ 1.58451597,  3.42105232],
       [ 2.64862495,  2.45219363],
       [ 3.02302654,  2.12393135],
       [ 1.72418318,  2.88883572],
       [ 2.10457478,  1.91703428],
       [ 0.01790543,  1.76753532],
       [ 3.09635772,  0.98455494],
       [ 1.38667694,  2.08772543],
       [ 1.58580937,  2.82174651],
       [ 1.17865527,  2.65693076],
       [ 2.84662048,  2.15909501],
       [ 2.7388454 ,  1.79400961],
       [ 0.51064713,  2.23230393],
       [ 0.37377388,  1.21765409],
       [ 2.54338432,  1.97955845],
       [-0.79695679,  1.36509756],
       [ 3.11033528,  3.33738723],
       [ 0.91170547,  1.03221129],
       [ 1.08202574,  1.72864125],
       [ 1.9499339 ,  2.98280352],
       [ 1.35329111,  1.93638184],
       [ 0.52210626,  2.01761051],
       [ 2.20678721, -0.57834232],
       [ 2.55156486,  1.72784432],
       [ 3.91156888,  2.41122276],
       [ 0.0182901 ,  2.81097915],
       [ 3.63312295,  1.51373963],
       [ 3.05748837,  2.45342584],
       [ 3.47392632,  1.21857677],
       [ 2.87375607,  3.87431705],
       [ 1.6089463 , -0.58781669],
       [ 2.7626967 ,  2.1123174 ],
       [ 0.52151635,  0.89187329],
       [ 1.51687903,  1.36064309],
       [ 3.22819418,  2.30865453],
       [ 3.89678915,  3.84973919],
       [ 2.27474702,  2.51209163],
       [ 2.50736606,  1.29431997],
       [ 0.69744532,  2.39016471],
       [ 2.46131117,  3.07193924],
       [ 1.5762919 ,  1.93726138],
       [ 2.93527366,  2.23267347],
       [ 2.07953417,  1.724407  ],
       [ 2.99359239,  1.68985209],
       [ 0.10646535,  1.61874245],
       [ 2.50162955,  2.56983621],
       [ 1.89193763,  3.10153339],
       [ 0.8999854 ,  0.15746181],
       [ 0.74205537,  0.80088093],
       [ 2.18386065,  2.48088607],
       [ 1.45291705,  3.72356403],
       [ 3.69355976,  0.97830561],
       [ 0.67971406,  2.87661381],
       [ 1.60976753,  2.93083248],
       [ 0.0880314 ,  0.99446886],
       [ 3.58309309,  2.80723817],
       [ 1.85345259,  1.55266663],
       [ 1.72778169,  3.18353349],
       [ 1.44093447,  2.40733249],
       [ 1.9057501 ,  1.4467888 ],
       [ 1.65947935,  1.96890372],
       [ 0.87368045,  3.1104558 ],
       [ 1.92748109,  3.89404072],
       [ 1.37131707, -0.0651976 ],
       [ 1.85976902,  1.99611763],
       [ 0.70610761,  1.63781766],
       [ 1.32226134,  1.28155429],
       [ 2.01193111,  2.6740449 ],
       [ 1.70257697,  1.70573435],
       [ 0.16209221,  2.09004428],
       [ 0.68849595,  0.46453758],
       [ 4.76188239,  1.56286532],
       [ 1.73342418,  1.14543912],
       [ 3.25670512,  2.05439954],
       [ 1.63167452,  1.98138221],
       [ 1.81336906,  2.33834237],
       [ 0.93118986,  2.24105463],
       [ 1.75778313,  1.20523   ],
       [ 2.03036546,  1.51996896],
       [ 2.28437195,  1.11741174],
```

```
[ 2.26457193,  1.11741174],
[ 0.71005763,  2.02644715],
[ 0.51565846,  1.52390047],
[ 0.77016854, -0.09292015],
[ 2.67526089,  2.47552681],
[ 2.66198886,  1.82082437],
[ 0.51320077,  1.33642559],
[ 1.73927535,  1.21895658],
[ 2.19147012,  1.6916339 ],
[ 3.75528166,  2.77572255],
[ 2.99760984,  2.42637627],
[ 2.26732417,  2.46930555],
[ 1.07200652,  2.35144638],
[ 2.44299343,  3.36245751],
[ 2.38875636,  0.77054581],
[ 2.23223558,  2.5102674 ]])
```

1. 2. 1. 2. Implement Learn Simple LIN-REG to find values of $Beta_0$ and $Beta_1$.

In order to solve the linear regression is important to bear in mind the formula to get the Least Squares.

> $$y - pred = \sum_{i=0}^{n} \frac{(x-\overline{x})(y-\overline{y})}{(x-\overline{x})^2}$$

- The first step made was to transpose matrix **A1** with the goal of making it easier to iterate with the values in the columns in matrix A1.
- Second, finding the mean of each column in matrix **A1**, values representing $\overline{x}$ and $\overline{y}$.
- Third, splitting the formula in 2: Numerator and denominator for a cleaner calculation.
- Finally, Applying the following formulas by using the outputs of the previous step: numerator and denominator.

> $$y = B_0 + B1x$$

> $$B_0 = \overline{y} - B_1\overline{x}$$

> $$B_1 = \overline{y} - B_0\overline{x}$$

```
# Transpose of A1
A2 = A1.T

# Mean of each column in Matrix A1
MeanA = mean(A1)
print('Mean A:', MeanA)


# Calculus of the numerator of the formula
numerator = (A2[0] - MeanA[0])*(A2[1] - MeanA[1])
totalNum = 0
for i in numerator:
  totalNum += i

# Calculus of the denominator of the formula
denominator = (A2[0] - MeanA[0])**2
totalDen = 0
for i in denominator:
    totalDen += i

# Calculus of the slope and y-intercept for the Linear Regression
beta1 = totalNum / totalDen
beta0 = MeanA[1] - (beta1*(MeanA[0]))
print('Beta0:', beta0)
print('Beta1:', beta1)
```

```
Mean A: [1.87596392 1.97176264]
Beta0: 1.6617787159156667
Beta1: 0.165239812522874
```

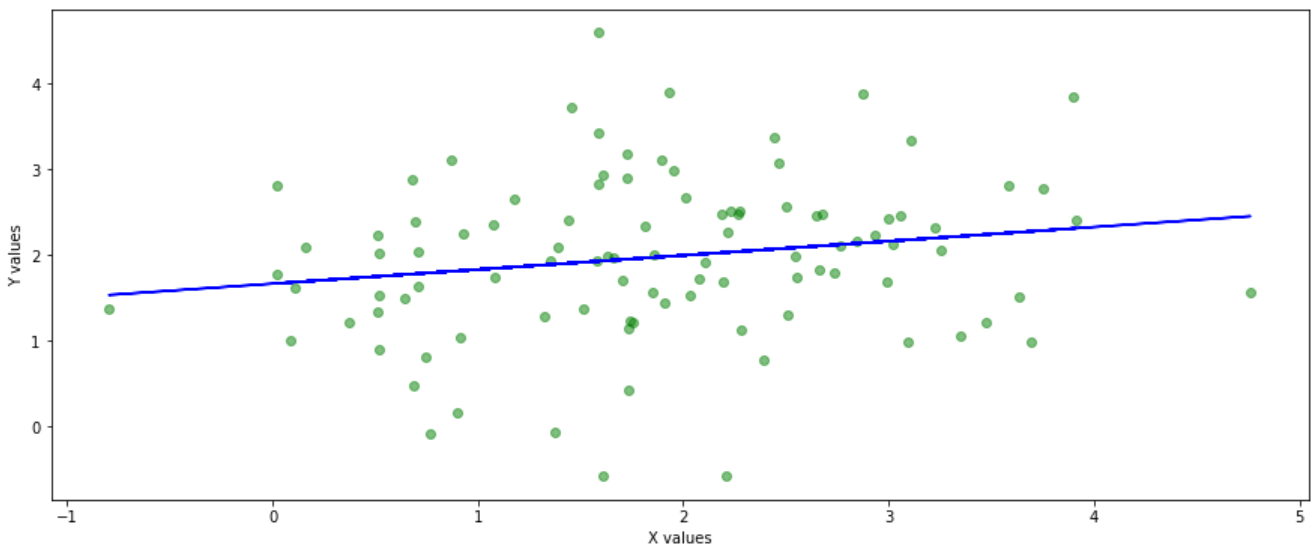
1. 2. 1. 3. SIMPLE-LINREG for all data points (predictions)

```
y_prediction = []
f = beta1*A2[0] + beta0
y_prediction = f
print("y_prediction:", y_prediction)
```

```
y_prediction: [1.9484431  1.92378192 1.7683621  2.21545185 2.02809893 1.92360384
 2.09943701 2.16130305 1.94668242 2.00953826 1.66473741 2.17342028
 1.89091295 1.92381756 1.85653949 2.13215375 2.11434502 1.74615795
 1.72354104 2.08204706 1.53008972 2.17572993 1.81242876 1.84057245
 1.98398543 1.88539629 1.74805146 2.02642782 2.08339881 2.30812562
 1.66480097 2.26211527 2.16699752 2.23580965 2.13663763 1.9276407
 2.1182862  1.74795398 1.91242752 2.19520492 2.30568342 2.03765749
 2.07609541 1.77702445 2.06848531 1.92224489 2.14680278 2.00540055
 2.15643936 1.67937103 2.07514751 1.97440214 1.81049213 1.78439581
 2.02263944 1.90185846 2.27210184 1.77409454 1.9277764  1.67632501
 2.25384835 1.96804287 1.94727704 1.89987846 1.9766845  1.93599077
 1.80614551 1.98027533 1.88837489 1.9690866  1.77845581 1.88026893
 1.99422984 1.94311221 1.6885628  1.77554566 2.44863127 1.9482094
 2.19991606 1.93139631 1.96141948 1.81564835 1.95223447 1.99727592
 2.03924791 1.77910851 1.74698602 1.78904122 2.10383832 2.10164526
 1.74657992 1.94917625 2.02389683 2.28230075 2.1571032  2.03643094
 1.83891687 2.06545849 2.05649637 2.0306329 ]
```

1. 2. 1. 4. Plot the training data and the predicted line.

```
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 15
fig_size[1] = 6
plt.rcParams["figure.figsize"] = fig_size

plt.scatter(A2[0], A2[1], color='green', alpha=0.5)
plt.plot(A2[0],y_prediction, c ='blue')
plt.xlabel('X values')
plt.ylabel('Y values')
plt.show()
```



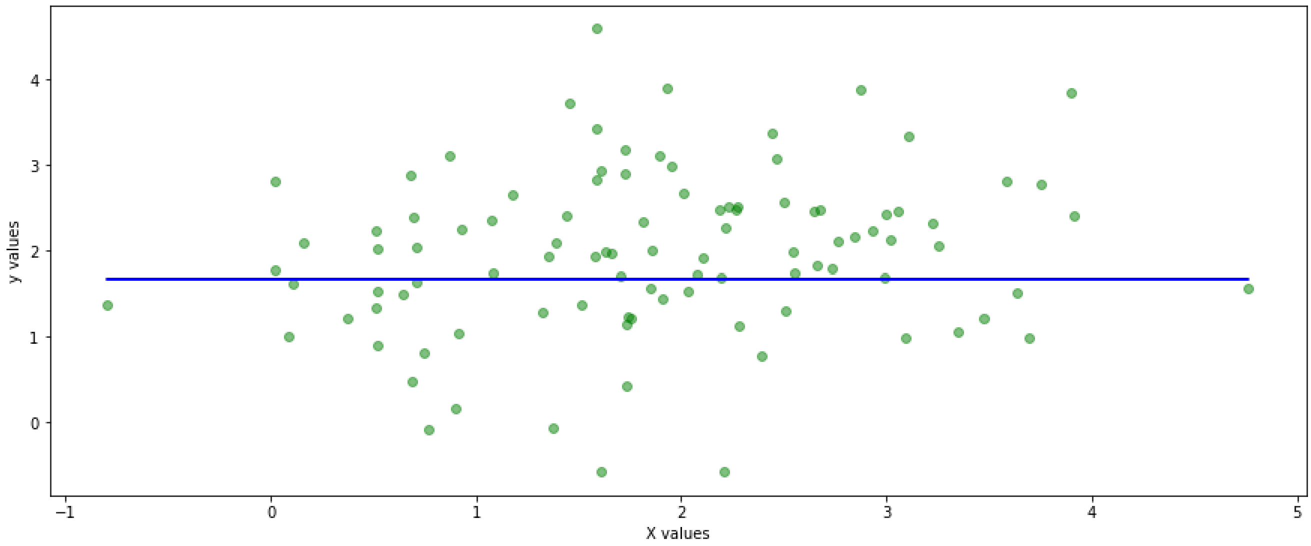
Observations: considering the slope and y-intercept above calculated ($Beta_0$ and $Beta_1$) the relationship between the 2 variables is as follow: For every aditional increase in value in the X axis the mean of the y-values will decrease in exactly the value of B_1 considering the error B_0.

1. 2. 1. 5. $B_1 = 0$ and re-run the predicted line

```
y1_prediction = []
f1 = 0*A2[0] + beta0
y1_prediction = f1
print("y1_prediction:", y1_prediction)
```

```
y1_prediction: [1.66177872 1.66177872 1.66177872 1.66177872 1.66177872 1.66177872
 1.66177872 1.66177872 1.66177872 1.66177872 1.66177872 1.66177872
 1.66177872 1.66177872 1.66177872 1.66177872 1.66177872 1.66177872
 1.66177872 1.66177872 1.66177872 1.66177872 1.66177872 1.66177872
 1.66177872 1.66177872 1.66177872 1.66177872 1.66177872 1.66177872
 1.66177872 1.66177872 1.66177872 1.66177872 1.66177872 1.66177872
 1.66177872 1.66177872 1.66177872 1.66177872 1.66177872 1.66177872
 1.66177872 1.66177872 1.66177872 1.66177872 1.66177872 1.66177872
 1.66177872 1.66177872 1.66177872 1.66177872 1.66177872 1.66177872
 1.66177872 1.66177872 1.66177872 1.66177872 1.66177872 1.66177872
 1.66177872 1.66177872 1.66177872 1.66177872 1.66177872 1.66177872
 1.66177872 1.66177872 1.66177872 1.66177872 1.66177872 1.66177872
```

```
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 15
fig_size[1] = 6
plt.rcParams["figure.figsize"] = fig_size

plt.scatter(A2[0], A2[1], color='green', alpha=0.5)
plt.xlabel('X values')
plt.ylabel('y values')
plt.plot(A2[0],y1_prediction, c ='blue')
plt.show()
```



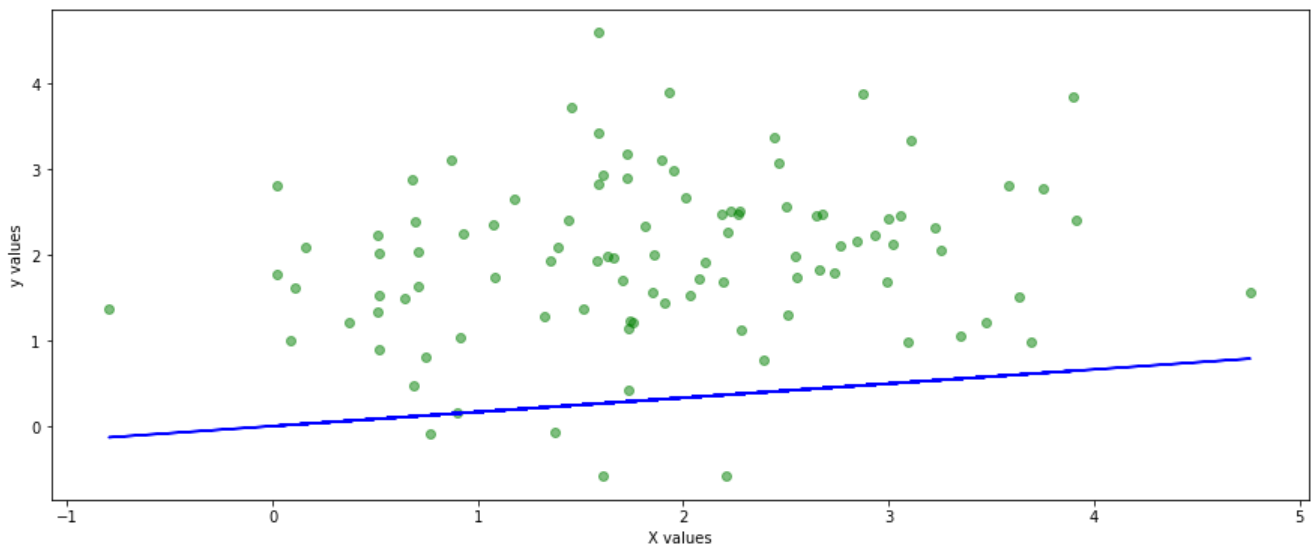
Observation: A slope equals to zero means that the mean of the dependant variable is not going to change no matter how much the values of the X-axis increases or decreases. It means that there is a high p-value to conclude insignificance of the scenario (changes in the predictor value does not change the result in the response variable).

1. 2. 1. 6. $B_0 = 0$ and re-run the predicted line

```
y2_prediction = []
f2 = beta1*A2[0] + 0
y2_prediction = f2
print("y2_prediction:", y2_prediction)
```

```
y2_prediction: [ 0.28666438  0.2620032   0.10658339  0.55367313  0.36632022  0.26182512
  0.43765829  0.49952434  0.28490371  0.34775954  0.00295869  0.51164157
  0.22913424  0.26203884  0.19476078  0.47037504  0.4525663   0.08437924
  0.06176233  0.42026835 -0.13168899  0.51395122  0.15065004  0.17879373
  0.32220671  0.22361757  0.08627274  0.3646491   0.4216201   0.64634691
  0.00302225  0.60033656  0.5052188   0.57403093  0.47485891  0.26586198
  0.45650749  0.08617526  0.25064881  0.5334262   0.64390471  0.37587877
  0.4143167   0.11524573  0.4067066   0.26046618  0.48502407  0.34362184
  0.49466065  0.01759231  0.4133688   0.31262342  0.14871342  0.12261709
  0.36086072  0.24007974  0.61032312  0.11231582  0.26599769  0.01454629
  0.59206963  0.30626416  0.28549832  0.23809974  0.31490579  0.27421206
  0.14436679  0.31849661  0.22659618  0.30730788  0.11667709  0.21849022
  0.33245112  0.2813335   0.02678409  0.11376694  0.78685255  0.28643069
  0.53813734  0.26961759  0.29964076  0.15386964  0.29045576  0.33549721
```

```
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 15
fig_size[1] = 6
plt.rcParams["figure.figsize"] = fig_size
plt.scatter(A2[0], A2[1], color='green', alpha=0.5)
plt.xlabel('X values')
plt.ylabel('y values')
plt.plot(A2[0],y2_prediction, c ='blue')
plt.show()
```



Observations: The y-intercept is the value at which the fitted line cross the y-axis. The constant guarantees that the result will not have positive or negative bias. It absorbs the bias existent in the relationship between the independant and dependant variable. If B_0 is forced to be zero, the regression must to go through the origin, if not some bias has been bearing into account when calculating the *slope* and *y-predictions*. As we can see in the image, without a y-intercept the predictions for a few datapoints tend to be high and low for others. The regression does not capture the esence of the dataset and could infere **underfitting** (not capturing the underlying structure of the data).

1. 2. 1. 7. Use numpy.linalg lstsq and replace steps 2

```
v=[]
for i in A1:
  v+=[[i[0],1]]
B = np.linalg.lstsq(v, A1[:,1], rcond=None)[0]
print('Betas',B)
```

```
Betas [0.16523981 1.66177872]
```

Since the Numpy Linalg Lstsq minimizes X in regards to $||ax - b||_2^2$. It uses a Singular Value Decomposition to solve least squares. Basically it finds a solution where $|x|_2$ is minimized by finding the minimum norm solution. But also finds the minimum norm least squares for for x : $||ax - b||_2$. In addition, this formula also calculates the rank of the matrix, the residuals and the error.

1. 2. 2. OLS using real dataset

The first step is to import the database and add the columns' names because the file does not explicitly show them. Moreover, reading the description and head of the dataset gives a better understanding of the structure of it.

```
drive.mount('/content/drive')
!ls "/content/drive/My Drive/Colab Notebooks/auto-mpg.data"
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remou
'/content/drive/My Drive/Colab Notebooks/auto-mpg.data'
```

```
column_names = ['mpg', 'cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', 'year', 'origin', 'name']
missing_values = ['-','na','Nan','nan','n/a','?']
MPG=pd.read_csv('/content/drive/My Drive/Colab Notebooks/auto-mpg.data', delim_whitespace=True, names=column_names, na_valu
MPG.head()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
0	18.0	8	307.0	130.0	3504.0	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693.0	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150.0	3436.0	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150.0	3433.0	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140.0	3449.0	10.5	70	1	ford torino

Next, always check for missing or incongruent values in the dataset. By using *MPG.empty* a boolean answer is returned if empty fields appear in the dataset. Moreover, by using *.dtypes* the output facilitates to identify if the type is correct according to the values in the columns.

```
check = MPG.empty
print('checking missing values:',check)
print('Sum of errors:',MPG.isnull().sum())
```

```
checking missing values: False
Sum of errors: mpg             0
cylinders       0
displacement    0
horsepower      6
weight          0
acceleration    0
year            0
origin          0
name            0
dtype: int64
```

As we can see, the column horsepower has 6 incongruent values which make the dtype of the column to be an object. To debug the dataset, it is necessary to replace the values of those fields with the mean of the column.

```
MPG['horsepower'] = MPG['horsepower'].fillna((MPG['horsepower'].mean()))
```

```
print('Sum of errors:',MPG.isnull().sum())
```

```
Sum of errors: mpg             0
cylinders       0
displacement    0
horsepower      0
weight          0
acceleration    0
year            0
origin          0
name            0
dtype: int64
```

Moreover, as each column in the dataset has different range of values it is necessary to scale them and get an unique range of values.

```
def scale(value):
    new = (value-value.min())/(value.max()-value.min())
    return new

MPG_scale = MPG.copy()

MPG_scale ['displacement'] = scale(MPG_scale['displacement'])
MPG_scale['horsepower'] = scale(MPG_scale['horsepower'])
MPG_scale ['acceleration'] = scale(MPG_scale['acceleration'])
MPG_scale ['weight'] = scale(MPG_scale['weight'])
MPG_scale['mpg'] = scale(MPG_scale['mpg'])
```

1. 2. 2. 1. Split the database into a uni-variate case with Displacement and mpg as x and y.

```
MPG_univariate = MPG_scale[['displacement', 'mpg']]
print(MPG_univariate)
```

```
     displacement       mpg
0        0.617571  0.239362
1        0.728682  0.159574
2        0.645995  0.239362
3        0.609819  0.186170
4        0.604651  0.212766
..            ...       ...
393      0.186047  0.478723
394      0.074935  0.930851
395      0.173127  0.611702
396      0.134367  0.505319
397      0.131783  0.585106

[398 rows x 2 columns]
```

1. 2. 2. 2. Find B_0 and B_1.

```
# Calculus of the Mean for each column
mean_univariate = MPG_univariate.mean(axis = 0)
Mean_MPG_univariate = mean_univariate[0]
Mean_MPG_univariate1 = mean_univariate[1]
print('Mean x:', Mean_MPG_univariate)
print('Mean y:', Mean_MPG_univariate1)

# Split the formula in 2: numerator and denominator for an ease understanding
numerator = (MPG_univariate['displacement'] - Mean_MPG_univariate)*(MPG_univariate['mpg'] - Mean_MPG_univariate1)
totalNum = 0
for i in numerator:
  totalNum += i

denominator = (MPG_univariate['displacement'] - Mean_MPG_univariate)**2
totalDen = 0
for i in denominator:
    totalDen += i

# Calculus of the Betas
beta_mpg1 = totalNum / totalDen
beta_mpg0 = Mean_MPG_univariate1 - (beta_mpg1*(Mean_MPG_univariate))
```

```
print('Beta0:', beta_mpg0)
print('Beta1:', beta_mpg1)
```

```
Mean x: 0.3240978795787724
Mean y: 0.38602587405110644
Beta0: 0.5871156005961069
Beta1: -0.6204598647987309
```

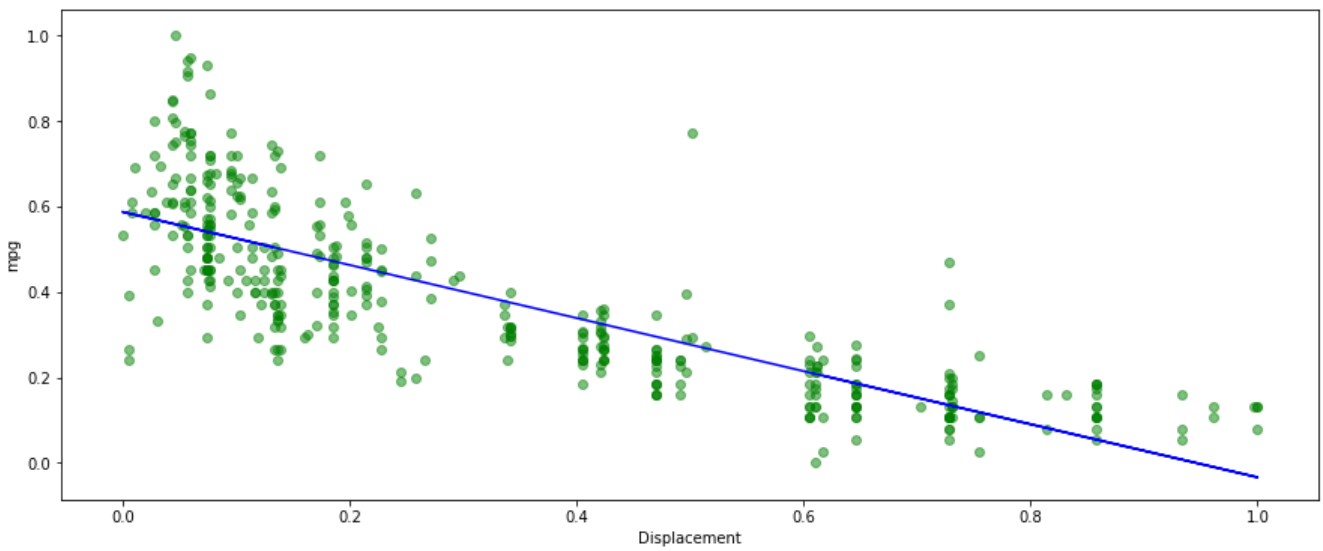
1. 2. 2. 3. $y - prediction$: calculate the points for each training example in A.

```
y_prediction3 = []
p = beta_mpg1*MPG_univariate['displacement'] + beta_mpg0
y_prediction3 = p
print("y_prediction:", y_prediction3)
```

```
y_prediction: 0      0.203938
1      0.134998
2      0.186302
3      0.208747
4      0.211954
         ...
393    0.471681
394    0.540621
395    0.479697
396    0.503746
397    0.505350
Name: displacement, Length: 398, dtype: float64
```

1. 2. 2. 4. Plot the training data and the predicted line.

```
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 15
fig_size[1] = 6
plt.rcParams["figure.figsize"] = fig_size
plt.scatter(MPG_univariate['displacement'], MPG_univariate['mpg'], color='green', alpha=0.5)
plt.xlabel('Displacement')
plt.ylabel('mpg')
plt.plot(MPG_univariate['displacement'],y_prediction3, c ='blue')
plt.show()
```



Observations: A vehicle *displacement* affects the Mile per Gallon a car needs. For each increase in displacement, the consumption of fuel will increase in 0,62 considering the scale previously created.

1. 2. 2. 5. Repeat the above but change the indepedant variable to ["horsepower","weight","acceleration"],

- x : horsepower | y : mpg

```
MPG_horsepower = MPG_scale[['horsepower', 'mpg']]
```

```
# Calculus of the Mean for each column
mean_horsepower = MPG_horsepower.mean(axis = 0)
Mean_MPG_horsepower_x = mean_horsepower[0]
Mean_MPG_horsepower_y = mean_horsepower[1]
print('Mean x:', Mean_MPG_horsepower_x)
print('Mean y:', Mean_MPG_horsepower_y)

numerator = (MPG_horsepower['horsepower'] - Mean_MPG_horsepower_x)*(MPG_horsepower['mpg'] - Mean_MPG_horsepower_y)
totalNum = 0
for i in numerator:
  totalNum += i

denominator_horsepower = (MPG_horsepower['horsepower'] - Mean_MPG_horsepower_x)**2
totalDen = 0
for i in denominator:
    totalDen += i


# Calculus of the Betas
beta_horsepower_1 = totalNum / totalDen
beta_horsepower_0 = Mean_MPG_horsepower_y - (beta_horsepower_1*(Mean_MPG_horsepower_x))
print('Beta0:', beta_horsepower_0)
print('Beta1:', beta_horsepower_1)

# Prediction of y for all datapoints in X.
y_prediction_horsepower = []
p1 = beta_horsepower_1*MPG_horsepower['horsepower'] + beta_horsepower_0
y_prediction_horsepower = p1
print("y_prediction:", y_prediction_horsepower)


#Plotting the graph considering the Betas found and the predictions.
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 15
fig_size[1] = 6
plt.rcParams["figure.figsize"] = fig_size

plt.scatter(MPG_horsepower['horsepower'], MPG_horsepower['mpg'], color='green', alpha=0.5)
plt.xlabel('Horsepower')
plt.ylabel('mpg')
plt.plot(MPG_horsepower['horsepower'],y_prediction_horsepower, c ='blue')
plt.show()
```

```
Mean x: 0.3177684117125112
Mean y: 0.38602587405110644
```

Observations: The dependence of MPG in regards to Horsepower is negative considering the pursue of the dataset. Each increase per unit in Horsepower in a vehicle, results in a 0.57 decrease in gallons per mile. Therefore, more fuel is required to address the demand.

- x : weight | y : mpg

```
MPG_weight = MPG_scale[['weight', 'mpg']]


# Calculus of the Mean for each column
mean_weight = MPG_weight.mean(axis = 0)
Mean_MPG_weight_x = mean_weight[0]
Mean_MPG_weight_y = mean_weight[1]
print('Mean x:', Mean_MPG_weight_x)
print('Mean y:', Mean_MPG_weight_y)

numerator = (MPG_weight['weight'] - Mean_MPG_weight_x)*(MPG_weight['mpg'] - Mean_MPG_weight_y)
totalNum = 0
for i in numerator:
  totalNum += i

denominator = (MPG_weight['weight'] - Mean_MPG_weight_x)**2
totalDen = 0
for i in denominator:
    totalDen += i


# Calculus of the Betas
beta_weight_1 = totalNum / totalDen
beta_weight_0 = Mean_MPG_weight_y - (beta_weight_1*(Mean_MPG_weight_x))
print('Beta0:', beta_weight_0)
print('Beta1:', beta_weight_1)


# Prediction of y for all datapoints in X.
y_prediction_weight = []
p2 = beta_weight_1*MPG_weight['weight'] + beta_weight_0
y_prediction_weight = p2
print("y_prediction:", y_prediction_weight)


#Plotting the graph considering the Betas found and the predictions.
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 15
fig_size[1] = 6
plt.rcParams["figure.figsize"] = fig_size

plt.scatter(MPG_weight['weight'], MPG_weight['mpg'], color='green', alpha=0.5)
plt.xlabel('Weight')
plt.ylabel('mpg')
plt.plot(MPG_weight['weight'],y_prediction_weight, c ='blue')
plt.show()
```

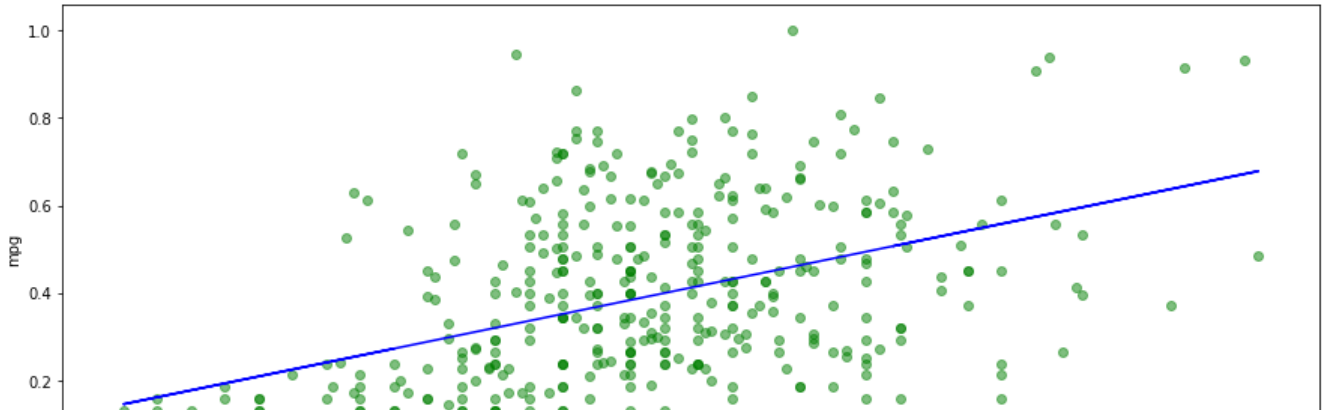
```
Mean x: 0.38486663541694865
Mean y: 0.38602587405110644
Beta0: 0.6631646911476656
Beta1: -0.720090523815655
y_prediction: 0      0.277088
1      0.238501
2      0.290972
3      0.291584
4      0.288317
         ...
393    0.422862
394    0.557611
395    0.523924
396    0.456550
397    0.437154
Name: weight, Length: 398, dtype: float64
```

Obervations: *weight* and *mpg* have a negative relationship. The heavier a vehicle is, the more gallons per mile it will need. In the presented graph, for each increase in 1 ton of a vehicle weight, the vehicle will need a 0,72 increase in gallons per mile. Therefore, the final output is an increase in fuel consumption.

- x : Acceleration | y : mpg

```
MPG_acceleration = MPG_scale[['acceleration', 'mpg']]

# Calculus of the Mean for each column
mean_acceleration = MPG_acceleration.mean(axis = 0)
print(mean_acceleration)
Mean_MPG_acceleration_x = mean_acceleration[0]
Mean_MPG_acceleration_y = mean_acceleration[1]
print('Mean x:', Mean_MPG_acceleration_x)
print('Mean y:', Mean_MPG_acceleration_y)

numerator = (MPG_acceleration['acceleration'] - Mean_MPG_acceleration_x)*(MPG_acceleration['mpg'] - Mean_MPG_acceleration_y
totalNum = 0
for i in numerator:
  totalNum += i

denominator = (MPG_acceleration['acceleration'] - Mean_MPG_acceleration_x)**2
totalDen = 0
for i in denominator:
    totalDen += i


# Calculus of the Betas
beta_acceleration_1 = totalNum / totalDen
beta_acceleration_0 = Mean_MPG_acceleration_y - (beta_acceleration_1*(Mean_MPG_acceleration_x))
print('Beta0:', beta_acceleration_0)
print('Beta1:', beta_acceleration_1)


# Prediction of y for all datapoints in X.
y_prediction_acceleration = []
p3 = beta_acceleration_1*MPG_acceleration['acceleration'] + beta_acceleration_0
y_prediction_acceleration = p3
print("y_prediction:", y_prediction_acceleration)


#Plotting the graph considering the Betas found and the predictions.
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 15
fig_size[1] = 6
plt.rcParams["figure.figsize"] = fig_size

plt.scatter(MPG_acceleration['acceleration'], MPG_acceleration['mpg'], color='green', alpha=0.5)
plt.xlabel('Acceleration')
plt.ylabel('mpg')
plt.plot(MPG_acceleration['acceleration'],y_prediction_acceleration, c ='blue')
plt.show()
```

```
acceleration    0.450482
mpg             0.386026
dtype: float64
Mean x: 0.45048157453936316
Mean y: 0.38602587405110644
Beta0: 0.14626141593233313
Beta1: 0.5322403216245699
y_prediction: 0      0.272985
1      0.257145
2      0.241304
3      0.272985
4      0.225464
         ...
393    0.387037
394    0.672166
395    0.260313
396    0.482080
397    0.507424
Name: acceleration, Length: 398, dtype: float64
```



The relationship between *acceleration* and *mpg* is positive. For every one porcentual point in acceleration increased, the mean value of miles per gallon will increase in a value of 0,53. In conclusion, the more acceleration a vehicle has, the less fuel consumption it will require.

Sumary

1. The faster a car goes, the less fuel consumption it makes.
2. The heavier the car, the less miles per hour it will get.
3. The more power a car needs to be moved (horsepower) the more fuel it requires to work optimally, increasing the consumption of fuel per mile.
4. The more effort is done by the displacement of the car, it requires more fuel to work, therefore, the MPG will increase.

Final thought: The worst impact for a city (giving perspective of the dataset) depends on the weight of a car. It has the higher neccesity of fuel. On second place, the effort made by the engine requires sufficient fuel which speeds the fuel consumption. On the other hand, acceleration has a positive impact in the mpg-relationship. According to this analysis, decision-making could take place.

1. 2. 2. 6. use scikit learn to implement OLS Linear Model.

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()

#Displacement and MPG

X = MPG_scale['displacement'].values.reshape(-1,1)
y = MPG_scale['mpg'].values.reshape(-1,1)
regressor = LinearRegression()
regressor.fit(X, y) #training the algorithm
#Retrieving the intercept:
print('regressor Intercept displacement',regressor.intercept_)
#Retrieving the slope:
print('regressor Coef displacement',regressor.coef_)
#Retrieving the predictions in regards to X:
y_OLS_predictions_displacement = regressor.predict(X)


#Horsepower and MPG

X1 = MPG_scale['horsepower'].values.reshape(-1,1)
y1 = MPG_scale['mpg'].values.reshape(-1,1)
regressor1 = LinearRegression()
regressor1.fit(X1, y1) #training the algorithm
#Retrieving the intercept:
```

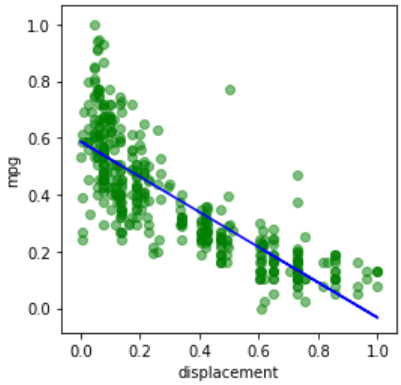
```
print('regressor Intercept horsepower',regressor.intercept_)
#Retrieving the slope:
print('regressor Coef horsepower',regressor.coef_)
#Retrieving the predictions in regards to X:
y_OLS_predictions_horsepower = regressor.predict(X1)

# Weight and MPG

X2 = MPG_scale['weight'].values.reshape(-1,1)
y2 = MPG_scale['mpg'].values.reshape(-1,1)
regressor2 = LinearRegression()
regressor2.fit(X2, y2) #training the algorithm
#Retrieving the intercept:
print('regressor Intercept weight',regressor.intercept_)
#Retrieving the slope:
print('regressor Coef weight',regressor.coef_)
#Retrieving the predictions in regards to X:
y_OLS_predictions_weight = regressor.predict(X2)


# Acceleration and MPG

X3 = MPG_scale['acceleration'].values.reshape(-1,1)
y3 = MPG_scale['mpg'].values.reshape(-1,1)
regressor3 = LinearRegression()
regressor3.fit(X3, y3) #training the algorithm
#Retrieving the intercept:
print('regressor Intercept acceleration',regressor.intercept_)
#Retrieving the slope:
print('regressor Coef acceleration',regressor.coef_)
#Retrieving the predictions in regards to X:
y_OLS_predictions_acceleration = regressor.predict(X3)
```

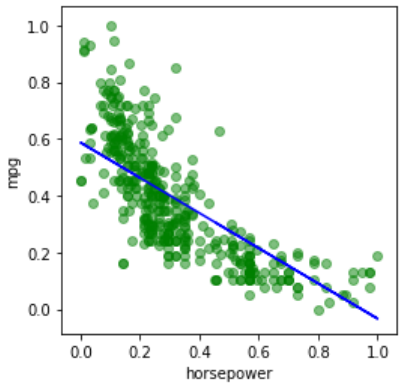
```
regressor Intercept displacement [0.5871156]
regressor Coef displacement [[-0.62045986]]
regressor Intercept horsepower [0.5871156]
regressor Coef horsepower [[-0.62045986]]
regressor Intercept weight [0.5871156]
regressor Coef weight [[-0.62045986]]
regressor Intercept acceleration [0.5871156]
regressor Coef acceleration [[-0.62045986]]
```

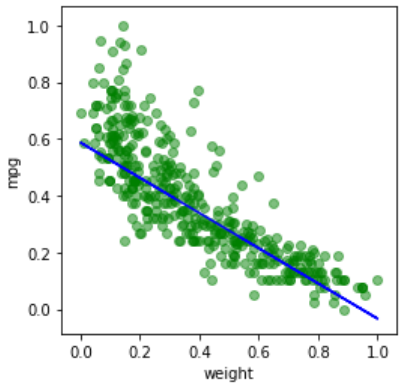
1. 2. 2. 7. Plot the training data (use plt.scatter) and your predicted line (use plt.plot).

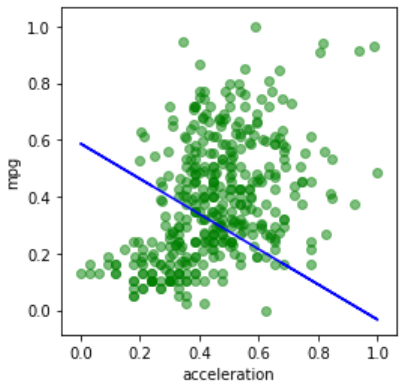
```
# Displacement
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 4
fig_size[1] = 4
plt.rcParams["figure.figsize"] = fig_size
plt.scatter(X, y, color='green', alpha=0.5)
plt.xlabel('displacement')
plt.ylabel('mpg')
plt.plot(X,y_OLS_predictions_displacement, c ='blue')
plt.show()

# Horsepower
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 4
fig_size[1] = 4
plt.rcParams["figure.figsize"] = fig_size
plt.scatter(X1, y1, color='green', alpha=0.5)
plt.xlabel('horsepower')
plt.ylabel('mpg')
plt.plot(X1,y_OLS_predictions_horsepower, c ='blue')
plt.show()

# Weight
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 4
fig_size[1] = 4
```

```
plt.rcParams["figure.figsize"] = fig_size
plt.scatter(X2, y2, color='green', alpha=0.5)
plt.xlabel('weight')
plt.ylabel('mpg')
plt.plot(X2,y_OLS_predictions_weight, c ='blue')
plt.show()

# Acceleration
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 4
fig_size[1] = 4
plt.rcParams["figure.figsize"] = fig_size
plt.scatter(X3, y3, color='green', alpha=0.5)
plt.xlabel('acceleration')
plt.ylabel('mpg')
plt.plot(X3,y_OLS_predictions_acceleration, c ='blue')
plt.show()
```









Bibliography

- Devanshbesain. (2017, August 3). Exploration and analysis - Auto-MPG. Retrieved from https://www.kaggle.com/devanshbesain/exploration-and-analysis-auto-mpg.

- Editor, M. B. (n.d.). Regression Analysis: How to Interpret the Constant (Y Intercept). Retrieved from https://blog.minitab.com/blog/adventures-in-statistics-2/regression-analysis-how-to-interpret-the-constant-y-intercept.

- Editor, M. B. (n.d.). Why You Need to Check Your Residual Plots for Regression Analysis: Or, To Err is Human, To Err Randomly is Statistically Divine. Retrieved from https://blog.minitab.com/blog/adventures-in-statistics-2/why-you-need-to-check-your-residual-plots-for-regression-analysis.

- Sarigoz, F. (n.d.). Fatih Sarigoz Data Science Blog. Retrieved from https://fatihsarigoz.com/tag/python-machine_learning-auto-mpg-dataset.html.

- Editor, M. B. (n.d.). Why You Need to Check Your Residual Plots for Regression Analysis: Or, To Err is Human, To Err Randomly is Statistically Divine. Retrieved from https://blog.minitab.com/blog/adventures-in-statistics-2/why-you-need-to-check-your-residual-plots-for-regression-analysis.

- sklearn.linear_model.LinearRegression¶. (n.d.). Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html.