Linear regression:
What is a linear function?
In general we have:

$$y = a * x + b$$

Where b is our bias, a our slope and y our output value.

However, in Machine Learning one uses different letters:

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 * x$$

We would call them Beta-hat. The beta hat indicated with 0 is still the same bias and the other one is also the same slope, just different names. But we change y to y-hat because we say it's a predictor.

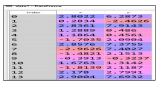
But how would one train a model like this and how can we optimize it?
We need:

- •a dataset

- •a training algorithm with a formula to minimize analytically (usually we minimize a cost function/error measure)

- •an error measure, for our wrong predictions and to look how good our model predicts. In general we look how much difference there is between the real and predicted values.

Our observed dataset contains x values and y values. We later want to predict a y value for any input x.



data1 - DataFrame

Index	x	y
0	2.8022	6.2873
1	0.2834	-2.4626
2	2.8361	7.9143
3	1.2889	0.486
4	1.1864	0.4561
5	-1.7035	2.0904
6	2.8576	7.3755
7	-2.9626	7.4027
8	-1.4821	2.1513
9	-0.3913	-0.3237
10	1.6763	1.3142
11	-1.8139	2.1187
12	2.178	2.7991
13	2.9004	7.6932

Now we have our data set we can move on to our training algorithm.
We now have to calculate our betas, bias and slope.
We do that by minimizing the error measure and solve after beta: Residual Sum of Squares(RSS)
Keep in mind: N is the size of the dataset, c.f. indices of our x,y columns above.

$$RSS = \sum_{n=0}^{N} (y_n - \hat{y}_n)^2$$

Itresults into a least squares problem by minimizing it after beta0 and beta1

$$\hat{\beta}_1 = \frac{\sum_{n=1}^{N} (x_n - \bar{x})(y_n - \bar{y})}{\sum_{n=1}^{N} (x_n - \bar{x})^2}$$
$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

Furthermore, I need to mention that xbar is the mean of all x. Which would be the sum over all x divided by x.

$$MEAN = \frac{1}{N} \sum_{n=0}^{N} x$$

And last we use the Mean Squared Error measure to see how good our model is by using the sum of dividing our predicted y_hat(x) from the real y-value we have in our dataset and then squaring it up.

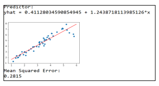
$$MSE = \frac{1}{N} \sum_{n=0}^{N} (y_n - \hat{y})$$

But since this would we a lot of work to do hand-written we instead can use Python to compute all those things.

Code

```
"""
@author: Chris
"""
#used for calculating the solution of the predictor
def predictor(beta0, beta1, x):
return beta0 + beta1 * x

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
#read dataset
dataset = pd.read_csv("tutorial3.dat", sep = " ",
names= ["x", "y"])
#compute ß1
xmean = np.mean(dataset["x"])
ymean = np.mean(dataset["y"])
numerator = 0
denominator = 0
#calcilate numerator and denominator of the formula
for i, row in dataset.iterrows():
numerator += ((row["x"] - xmean)) * (row["y"] - ymean)
denominator += (row["x"] - xmean)**2
#fraction
beta1 = numerator / denominator
#compute ß0 using the formula given
beta0 = ymean - beta1 * xmean

print("Predictor:")
print("yhat = " + str(beta0) + " + " + str(beta1) + "*x")

#plot our data
plt.scatter(dataset["x"],dataset["y"])
#plot predictor function
x = np.linspace(1,6, 100)
y = beta0 + beta1 * x
plt.plot(x,y, color = "red")
plt.show()

#Calculate Mean Squared error using the formula
MSE = 0
for i, row in dataset.iterrows():
MSE += (row["y"] - predictor(beta0, beta1, row["x"])) **2
MSE = MSE / len(dataset)
print("Mean Squared Error:")
print(np.around(MSE,4))
```

In the end we get the following result:



```
Predictor:
yhat = 0.41128034590854945 + 1.2438718113985126*x

Mean Squared Error:
0.2815
```

For this specific dataset we see that linear regression performs quite good.