Bootcamp
Machine Learning

Day01 - Linear Regression

During this day you will learn the first concepts constituing the field of machine learning.

Notions of the day

Matrix operations, gradient descent, cost function, normal equation, MSE, RMSE R-score and learning rate.

General rules

- The version of Python to use is 3.7, you can check the version of Python with the following command: python -V
- The norm: during this bootcamp you will follow the Pep8 standards https://www.python.org/dev/peps/pep-0008/
- The function eval is never allowed.
- The exercices are ordered from the easiest to the hardest.
- Your exercises are going to be evaluated by someone else, so make sure that your variable names and function names are appropriate and civil.
- Your manual is the internet.
- You can also ask question in the dedicated channel in Slack: 42-ai.slack.com.
- If you find any issue or mistakes in the subject please create an issue on our dedicated repository on Github: https://github.com/42-AI/bootcamp_machine-larning/issues.

Helper

Ensure that you have the right Python version.

```
> which python
/goinfre/miniconda/bin/python
> python -V
Python 3.7.*
> which pip
/goinfre/miniconda/bin/pip
```

Exercice 00 - Linear Regression

Exercise 01 - Multiples Features and Linear Gradient Descent

Exercice 02 - Multiples Features and Normal Equation

Exercise 03 - Learning Rate and Quadratic Hypothesis

Exercise 00 - Linear Regression

Turnin directory :	ex00
Files to turn in :	linear_model.py
Authorize module :	numpy, matplotlib
Forbidden module :	sklearn
Forbidden function :	LinearRegression
Remarks :	Read the doc

Objectives:

- Reinforce the mathematical skills tackled in **Mathematical Delights**, especially the **matrix-matrix operations** and **linear gradient**.
- Be able to explain what is the **linear cost function** and the **gradient descent**.
- Be able to implement basic methods in order to perform a linear regressions.

Instructions:

You can find in the ressources a tiny data set called **are_blue_pills_magics.csv** which give you mathematical performance of patients in function of the quantity of the "blue pills" they took before the test.

As hypothesis function h, you will choose:

$$h(X) = \theta_0 x_0 + \theta_1 x_1$$

Where X is the vector (x_0, x_1) and $theta = (theta_0, \theta_1)$

You will model the data and plot 2 differents graphs:

- The graph with the data and the best hypothesis you find for the spacecraft piloting score versus the quantity of "blue pills" (see example figure 1),
- The cost function $J(\theta)$ in function of the θ values (see example in figure 2),

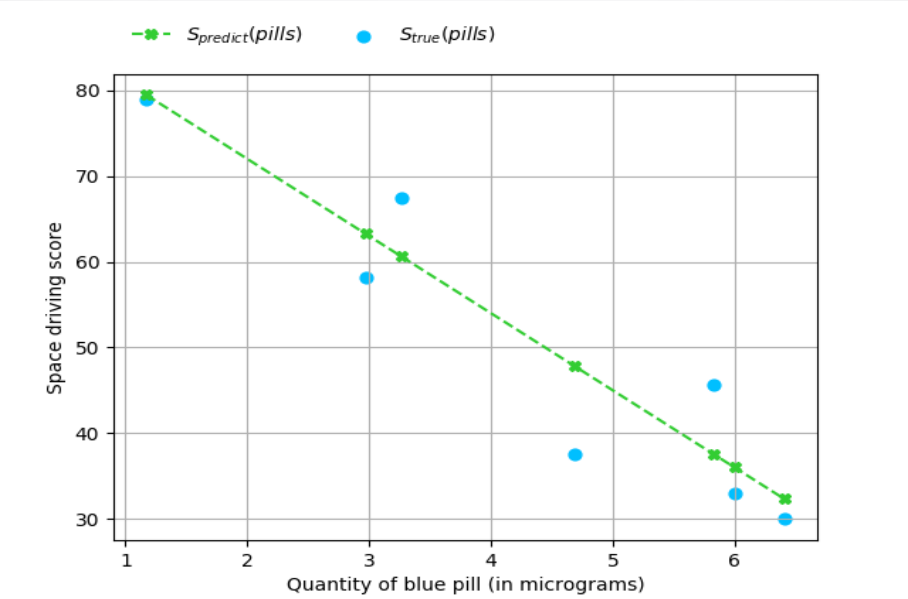


Figure 1: Evolution of the space driving score in function of the quantity of blue pill in micrograms.

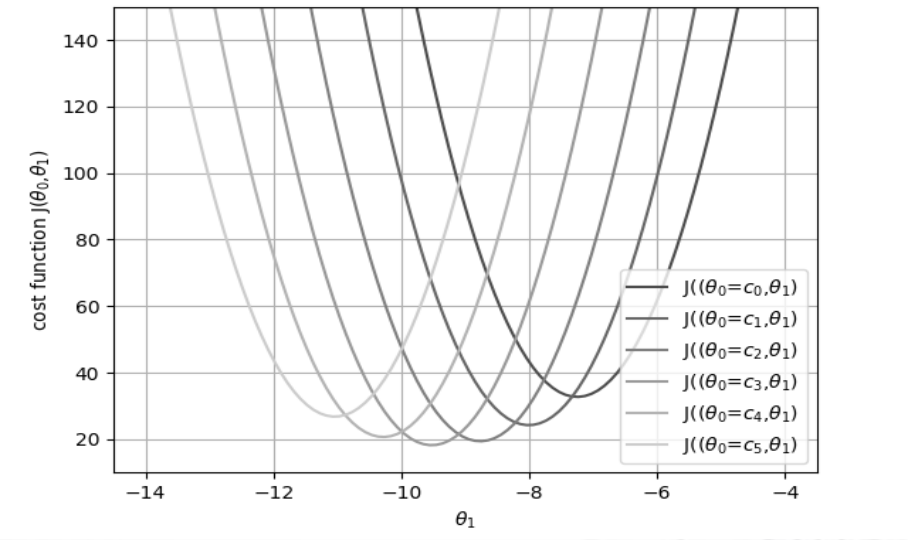


Figure 2: Evolution of the cost function J in fuction of θ_0 and θ_1.

Plus, you will calculate the mean values of the dataset and the MSE of the hypothesis you chose.
The MSE is given by the following formula:

$$MSE = \frac{1}{2M} \sum i = 1M \left(\hat{y}^{(i)} - y^{(i)} \right)^2$$

where $\hat{y}^{(i)}$ and $y^{(i)}$ are respectively the predicted output and the real output with the ith training samples $x^{(i)}$.

To do so you will implement a class called **MyLinearRegression** similar to **sklearn.linear_model.LinearRegression**, containing the following methods:

```
def predict(self, X):
    """
    Description:
        Predict using the linear model.
    Args:
        X: __array_like__ or __sparse matrix__, shape(number of training
examples, number of features)
        -> Samples(/training dataset) data from which we want to generate
predicted values.
    Returns:
        C: array, shape(number of the training examples,)
        -> Predicted values in the form of an array.
    Raises:
        This method should not raise any Exception.
    """

def mse(self, X, Y):
    """
    Description:
        Calculate the MSE (Mean Squared Error) of the set of predicted
values with respect to Y.
    Args:
        X: __array_like__ or __sparse matrix__, shape(number of training
examples, number of features)
        -> Samples(/training dataset) from which we want to generate
predicted values.
        Y: array, shape(number of training examples,)
        ->
    Return:
        mse: float.
        -> MSE of self.predict(X) with respect to Y.
    """
```

Examples

```python
import pandas as pd
import numpy as np
from sklearn.metrics import mean_squared_error
from mylinearregression import MyLinearRegression as MyLR

data = pd.read_csv("are_blue_pills_magics.csv")
Xpill = np.array(data[micrograms]).reshape(-1,1)
Yscore = np.array(data[Score]).reshape(-1,1)

linear_model_1 = MyLR([89.0, -8])
linear_model_2 = MyLR([89.0, -6])
Y_model1 = linear_model1.predict(Xpill)
Y_model2 = linear_model2.predict(Xpill)

print(linear_model_1.mse(Xpill, Yscore))
# 57.60304285714282
print(mean_squared_error(Yscore, Y_model1))
# 57.603042857142825
print(linear_model_2.mse(Xpill, Yscore))
# 232.16344285714285
print(mean_squared_error(Yscore, Y_model1))
# 232.16344285714285
```

Clarifications and hints

There is no method named **.mse** in the class LinearRegression of the module

sklearn.linear_model but there is a method named **.score**. The **.score** method correspond to

the R^2 score.

The metric MSE is available in the module sklearn.metrics.

Be sure to understand the underlying concept and be able to answer to those questions evaluators may ask:

- What is the hypothesis and what is it goal ?
- What is the cost function and what it is representing ?
- What is the linear gradient descent and what is it doing ?
- Can you explain the MSE and what it is spotting ?

Exercise 01 - Mutiples features and Linear Gradient Descent

Turnin directory :	ex01
Files to turn in :	multi_linear_model.py
Authorize module :	numpy, matplotlib
Forbidden module :	sklearn
Forbidden function :	LinearRegression
Remarks :	Read the doc

Objectives:

• Reinforce the mathematical skills tackled in **Mathematical Delights**, especially the vectorized form of **Root Mean Square** and **gradient descent** (If you were to lazy to do it in the exercise 00 cof cof).
• Be able to perform **linear cost function** and the **linear gradient descent** for a multiple features problem (and knowing what you are doing of course).
• Be able to implement vectorized methods **.fit** and **.rmse** in order to perform a full multi-linear regressions.
• Be able to visualize the different objects via graphics and extract basics informations based on its.

Instructions:

As you are able to perform a simple linear regression with one feature (well done!) it is time to dream bigger.
Lucky you are, we give you a new dataset with multiple features that you will find in the ressources attached.
The dataset is called **spacecraft_data.csv** which contains the prices of spacecrafts in function of multiples features (multiple features means you will need a multi-linear model but hold on for the moment). A description of the dataset is provided in the file **data_description.txt**.

Part One: single linear regression

As a starter, you will try to fit the data with a single linear regression and see what we get.
Thus, your hypothesis h(X) would be given by:

$$h(X) = \theta_0 + \theta_1 \cdot X$$

You are expected to:

• Add the method **.fit** in the class **MyLinearRegression** you started to create in the previous exercise (fit by hand is over), following the prototype:

```
def fit(self, X, Y, alpha = 0.005, n_cycle=10000):
    """
    Description:
        Fit the linear model by performing a gradient descent on the cost
function.
    Args:
        X: __array_like__ or __sparse matrix__, shape(number of training
examples, number of features)
        -> Samples(/training dataset) data from which we want to generate
predicted values.
        Y: array, shape(number of training examples,)
        -> Target values of the training dataset.
        alpha:
        -> learning rate.
        n_cycle: integer
        -> number of cycling in the descent gradient the method will
perform.
    Returns:
        None
    Raises:
        This method should not raise any Exception.
    """

def rmse(self, X, Y):
    """
    Description:
        Calculate the RMSE (Root Mean Squared Error) of the set of predicted
values with respect to Y.
    Args:
        X: __array_like__ or __sparse matrix__, shape(number of training
examples, number of features)
        -> Samples(/training dataset) from which we want to generate
predicted values.
        Y: array, shape(number of training examples,)
        ->
    Return:
        rmse: float.
        -> RMSE of self.predict(X) with respect to Y.
    """
```

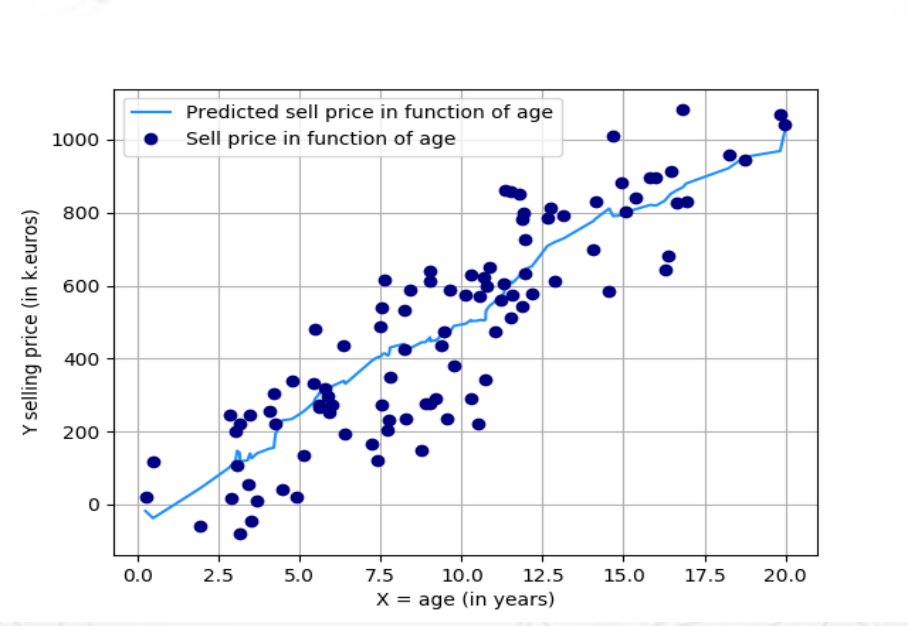
I strongly encourage you to code the **.fit** and **.rmse** methods in a vectorized way...

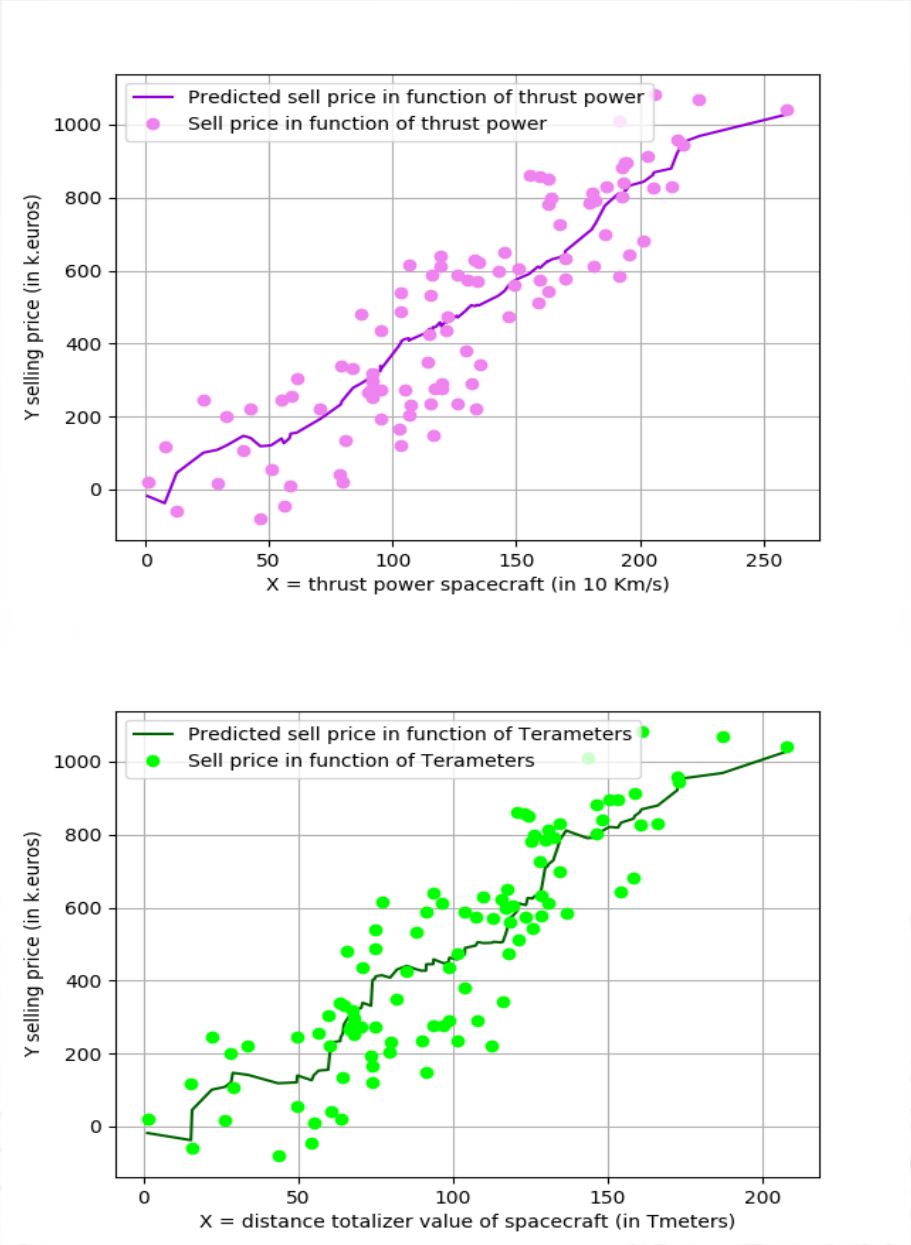
You will calcultate the score of your fits for the differents features:
Examples

```
>>>import pandas as pd
>>>import numpy as np
>>>form mylinearregression import MyLinearRegression as MyLR

>>>data = pd.read_csv("spacecraft_data.csv")
>>>[...]
>>>myRL_age = MyLR(2)
>>>myRL_thrust = MyLR(2)

>>>print(myLR_age)
[theta] = [[0.0]
[ 0.0]]

>>>myLRage.fit(X[:,0].reshape(-1,1),Y)
>>>print(myLRage)
[theta] = [[798.215..]
[-19.672..]]

>>>RMSE_age = myLRage.rmse(X[:,0].reshape(-1,1),Y)
>>> print(RMSE_age)
131.935...
```

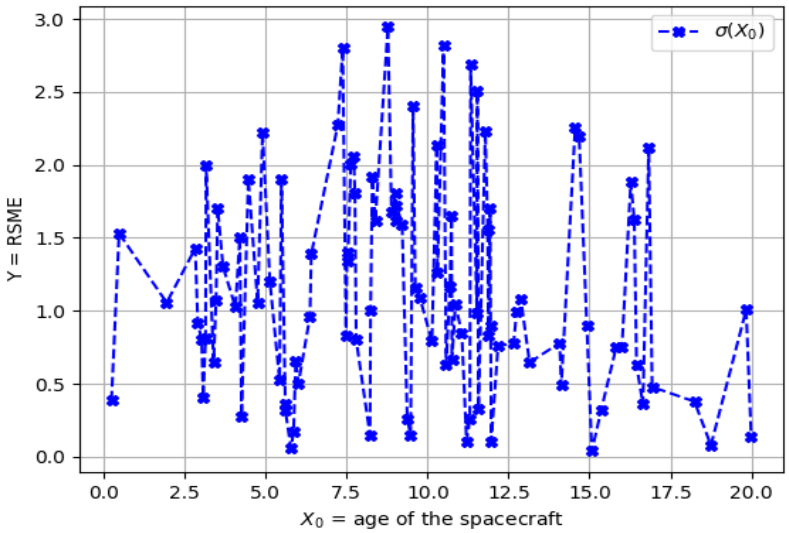
plot 6 differents graphs:

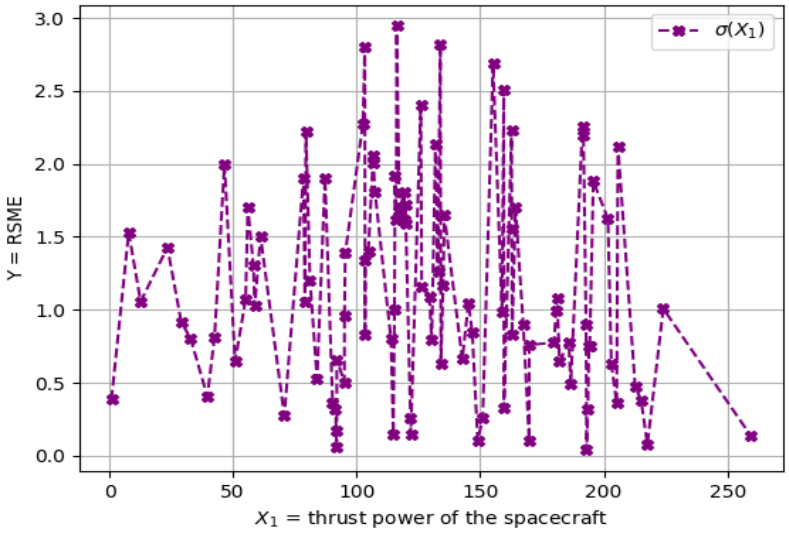
- The graph with the data, the hypothesis $h_\theta^{LGD}(X)$ obtained via linear gradient desent and $h_\theta^{NE}(X)$ (see examples figures 1a to 1c),

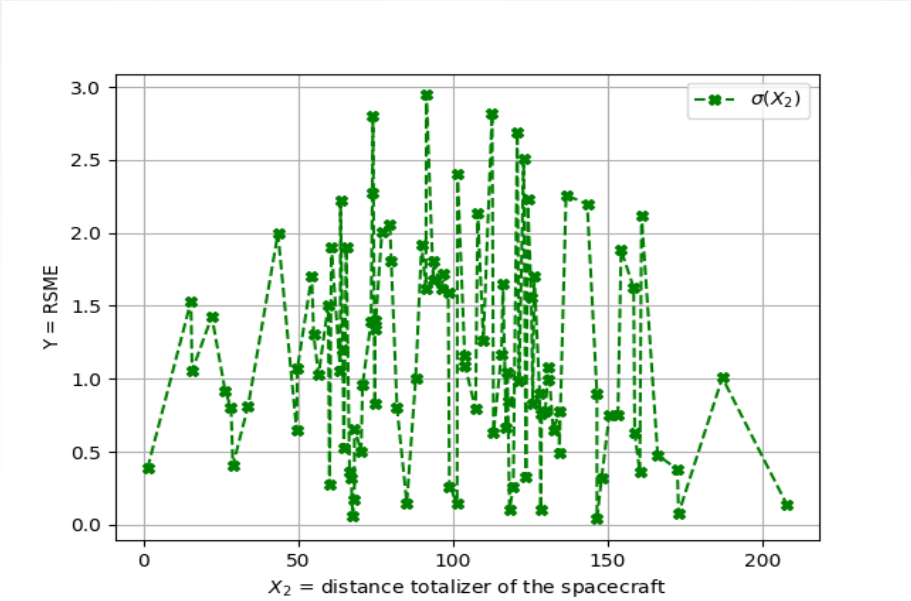




- Plot of standard deviation σ with respect to the theta's (figures 2a, 2b and 2c).


3.0
2.5
2.0
1.5
1.0
0.5
0.0
Y = RSME
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
X_0 = age of the spacecraft
$\sigma(X_0)$


3.0
2.5
2.0
1.5
1.0
0.5
0.0
Y = RSME
0
50
100
150
200
250
X_1 = thrust power of the spacecraft
$\sigma(X_1)$



Standard deviation is identical to the RMSE which is given by the following formula:

$$\sigma = RMSE = \sqrt{\frac{1}{M} \sum i = 1M \left(\hat{y}^{(i)} - y^{(i)}\right)^2}$$

where $\hat{y}^{(i)}$ and $y^(i)$ are respectively the predicted output and the output with the ith training samples $x^{(i)}$.

Are the fits with a single variable are precised ? Why ? (What did I say a the beginning ?)
What the purpose to represent the RMSE in function of a feature ?

Part Two: Multilinear Regression (A New Hope)

Now, it is time for your first multilenear regression !

As you might expected, the formula of the hyphothesis change a little and is given by:

$$h(X) = X\theta = \begin{bmatrix} x_0^{(1)} & \cdots & x_N^{(1)} \\ \vdots & \ddots & \vdots \\ x_0^{(m)} & \cdots & x_N^{(m)} \end{bmatrix} \cdot \begin{bmatrix} \theta_0 \\ \vdots \\ \theta_N \end{bmatrix} = \sum i = 1N\theta_i x_i$$

where X is the training dataset matrix, θ the parameter's vector, m is the number of training samples and N the number of features.

But It should not change the methods you coded, hopefully !

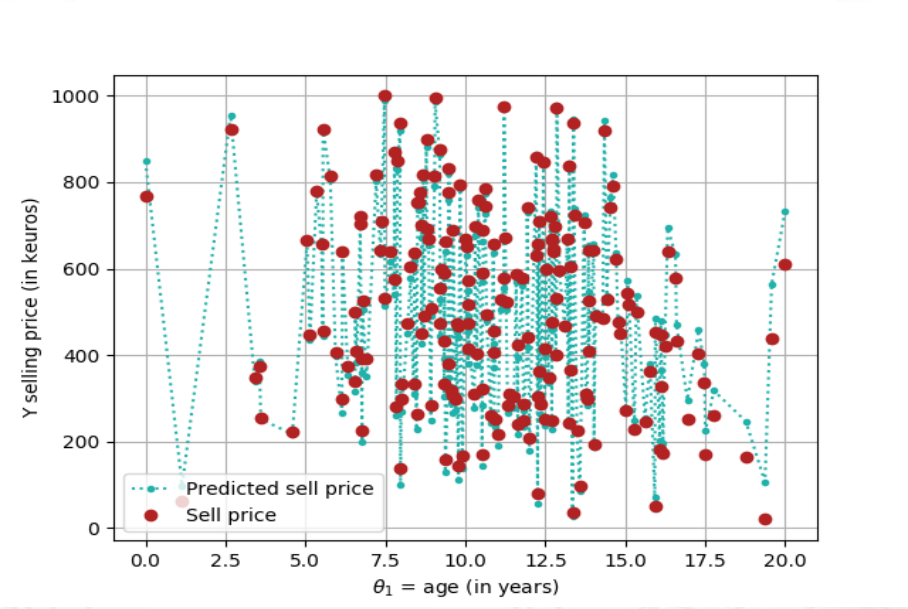
For this part, you are expected to:

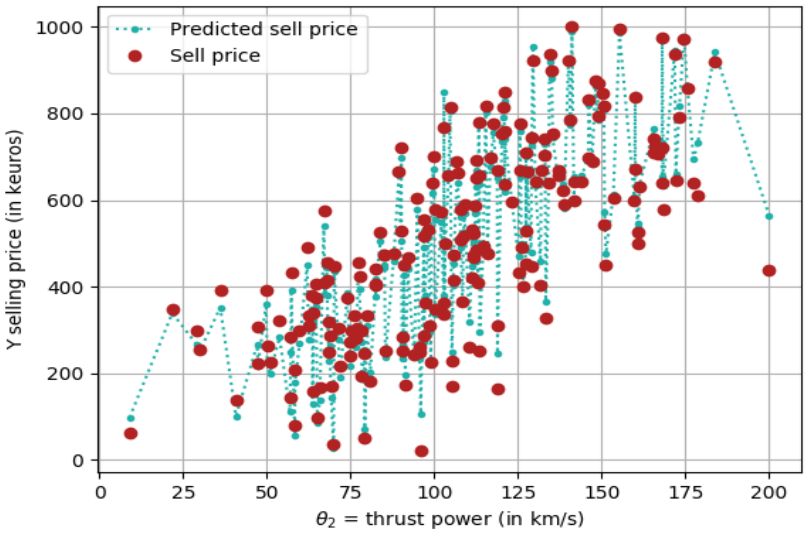
- Calculate the RMSE and compare it with the ones calculated in the previous part.

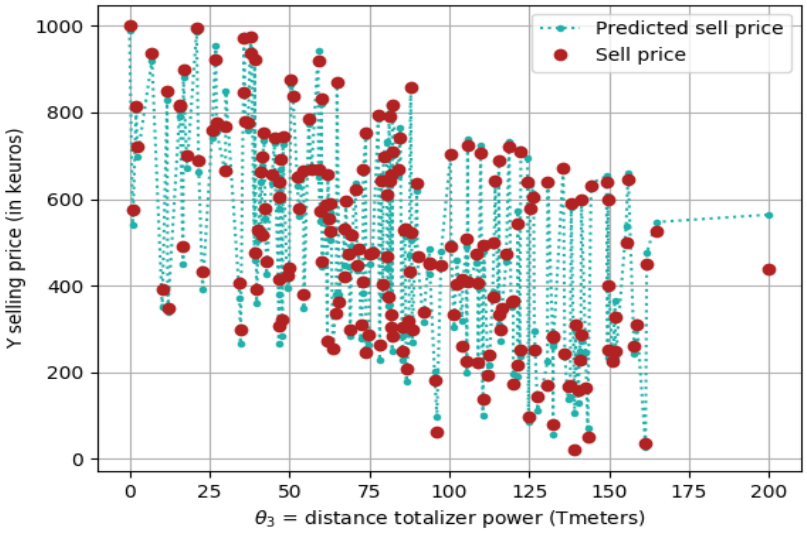
Examples

```
>>>import pandas as pd
>>>import numpy as np
>>>form mylinearregression import MyLinearRegression as MyLR

>>>data = pd.read_csv("spacecraft_data.csv")
>>>[...]
>>>my_lreg = MyLR(4)
>>>my_lreg.fit(X,Y)
>>>RMSE = myLR.score(X,Y))
>>>print(RMSE)
27.5049...
```

- Plot the output and predicted output on the same graph in function of the age, thrust power and distance (see figures 3a, 3b and 3c).







Questions:

Be sure to understand the underlying concept and be able to answer to those questions evaluators may ask:

- What is the learning rate ?
- What allows the linear gradient descent ?
- What information standard deviation gives you ?

Exercise 02 - Mutiples features and Normal Equation

Turnin directory :	ex02
Files to turn in :	normal_equation_model.py
Authorize module :	numpy, matplotlib
Forbidden module :	sklearn
Forbidden function :	LinearRegression
Remarks :	Read the doc

Objectives:

- Reinforce the mathematical skills tackled in **Mathematical Delights**, especially the vectorized form of **matrix-matrix oparations** (intermediate manipulation).
- Be able to perform **linear cost function** and what one names **normal equation** for a multiple features problem (and still knowing what you are doing of course obviously).
- Be able to implement vectorized methods **normalequation** and **rscore** (for R-score value) in order to perform a full linear regression.

Instructions:

We continue to **work** play with the same dataset (**spacraft_data.csv**) given in the previous exercise.
You will code a new method named **normalequation** which will allows you to perform a fit of the dataset based on the method called **normalequation**.

The normal equation is:

$$\theta_{NE} = \left(X^T X\right)^{-1} \cdot X^T Y$$

where X is the training dataset matrix, θ_{NE} the parameter's hypothesis vector obtained with the normal equation, Y are the output vector.

For folks who do not understand what the meaning of the superscripts T and -1 over a matrix, they respectively tell you we take the transpose and the inverse of the concerned matrix. Do not worry, for those who do not know what it means or do not remember how to do, well you can still do the exercise because numpy will run the

maths for you, but you will not understand what happens behind the scene for the moment. Fortunaly, the mathematic workshops will be coming soon, what a wonderful opportunity to learn to do transposition and inversion of matrices, lucky you.

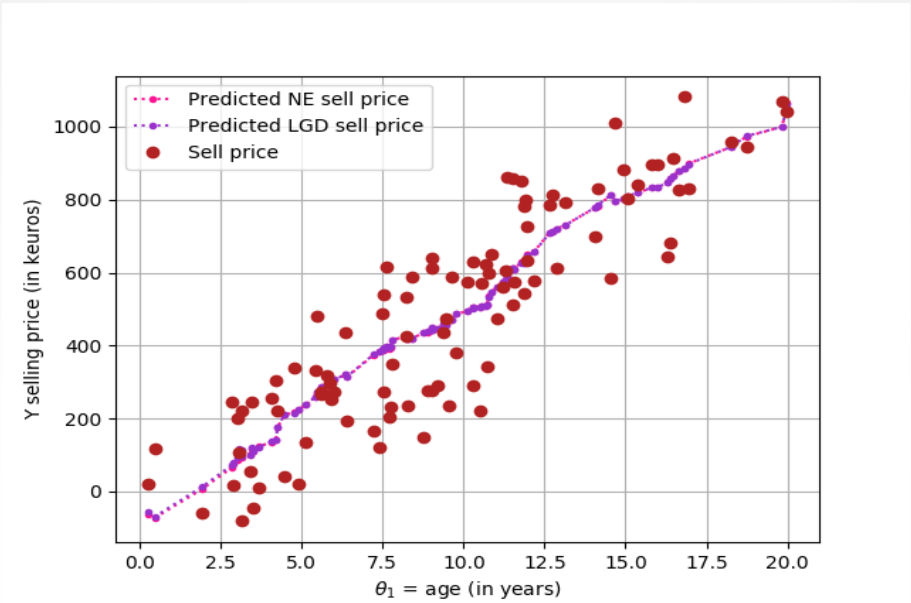
You are expected to:

• Add the methods **normalequation** and **rscore** in the class **MyLinearRegression** following the prototype:

```
def normalequation(self, X, Y))
    """
    Description:
        Perform the normal equation to get the theta parameters of the
hypothesis h and stock them in self.theta.
    Args:
        X: __array_like__ or __sparse matrix__, shape(number of training
examples, number of features)
        -> Samples(/training dataset) data from which we want to generate
predicted values.
        Y: array, shape(number of training examples,)
        -> Target values of the training dataset.
    Returns:
        None.
    Raises:
        This method should not raise any Exceptions.
    """

def rscore(self, X, Y):
    """
    Description:
        Calculate the R-score of the set of predicted values with respect to
Y.
    Args:
        X: __array_like__ or __sparse matrix__, shape(number of training
examples, number of features)
        -> Samples(/training dataset) from which we want to generate
predicted values.
        Y: array, shape(number of training examples,)
        ->
    Return:
        rscore: float.
        -> R-score of self.predict(X) with respect to Y.
    """
```

You will model the data and plot differents graphs (yes again):

• The graph with the data, the hypothesis $h_\theta^{LGD}(X)$ obtained via linear gradient desent and $h_\theta^{NE}(X)$ (see example figure 1) with respect to the feature of your choice,



• Calculate the R-score of the gradient descent model and of the normal equation with respect to the feature of your choice.

The formula for the R-score is given by the following formula:

$$R^2 = 1 - \frac{\frac{1}{N}\sum_i^N \left(y^{(i)} - \hat{y}^{(i)}\right)^2}{\frac{1}{N}\sum_i^N \left(y^{(i)} - \mu\right)^2}$$

where $\hat{y}^{(i)}$ and $y^(i)$ are respectively the predicted output and the output with the ith training samples $x^{(i)}$. μ is the mean value of the output y.

Example

```
>>>import pandas as pd
>>>import numpy as np
>>>from mylinearregression import MyLinearRegression as MyLR
>>>data = pd.read_csv("spacecraft_data.csv")
>>>[...]
>>>myLR_ne = MyLR(4)
>>>myLR_lgd = MyLR(4)
>>>myLR_lgd.fit(X,Y)
>>>myLR_ne.normalequation(X,Y)
>>>print("R2-score value: ",myLR_ne.rscore(X,Y))
0.787659...
>>>print("R2-score value: ",myLR_lgd.rscore(X,Y))
0.787619...
```

Be sure to understand the underlying concept and be able to answer to those questions evaluators may ask:

- What are the advantages and drawbacks of the linear gradient descent and normal equation ?
- In which case, the method normal equation cannot be used ?
- What is the information that the R-score give you and what are the informations that the R-score does not give you ?

Exercise 03 - Learning Rate and Quadratic Hypothesis

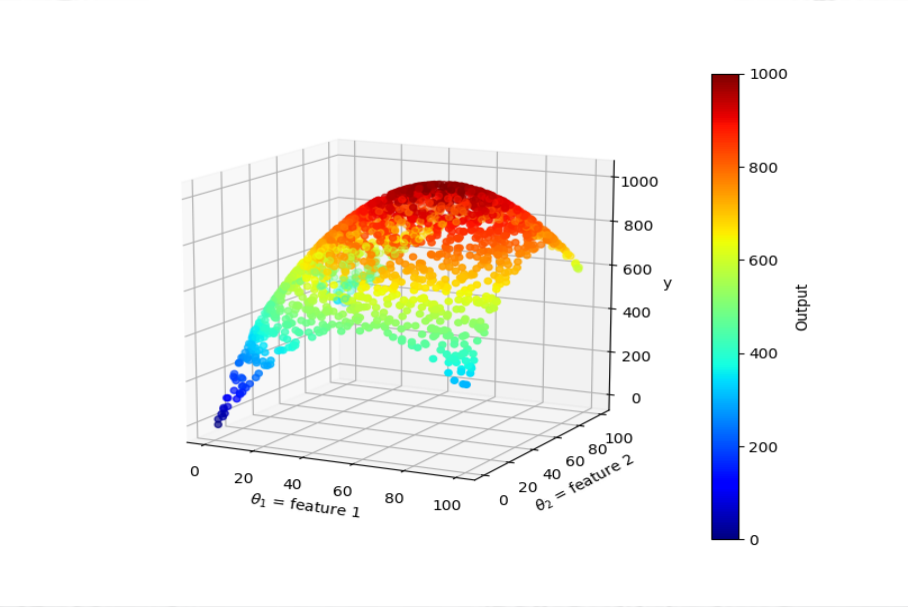
Turnin directory :	ex03
Files to turn in :	so_much_hyp.py
	alpha.py
Authorize module :	numpy, matplotlib
Forbidden module :	sklearn
Forbidden function :	LinearRegression
Remarks :	Read the doc

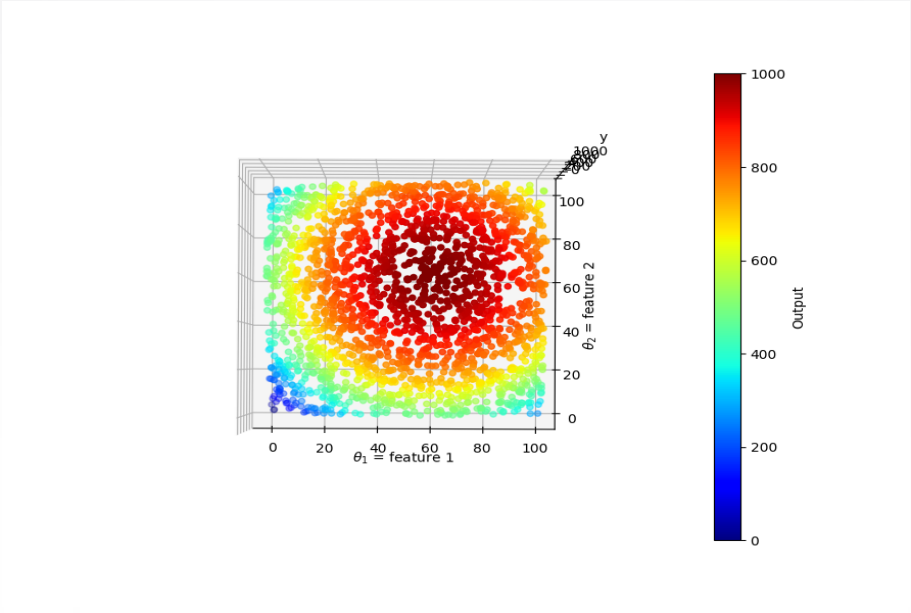
Objectives:

- Discover the non-linear dependency to elaborate more complex hypothesis and reinforce mathematical skills at the same time.
- Gain a deeper understanding of the learning rate.

Instructions:

For this last exercise we will play with a last dataset named **data.csv**. You will find a description of the dataset in the file **description_data.txt** (and 2 really nice graphical representations figure 1 and 2).





You will use the methods you code during the day to fit the data.
The hypothesis function you will need is a more complex one. This time you have to find out the best model to fit the data.
Do not worry, it is not too complicated.

You have to justify your hypothesis (know what and why you are doing).

Part One - Find the best hypothesis

For this part, you will write your code in the file **so_much_hyp.py**.

You are expected to:

- Find a good hypothesis and justify that your model is a good one.

hint: Take a look to the metrics with respect to the features (as you were asked in the exercise 1).

Here, what you can get:

```
>>>import pandas as pd
>>>import numpy as np
>>>data = pd.read_csv("data.csv")
[...]
>>>hypo.normalequation(X,Y)
>>>print("valeur du RMSE: ",hypo.rmse(X,Y))
valeur du RMSE: 1.4513e-11
```

Bonus question

Here a mathematical question, it is a bit challenging, more if you are not at ease with the linear algebra and the manipulation of the functions.

Even with the best hypothesis (if you have the same formula that the one I used to generate the data) you should have a difference between you predicted output and the real output of the order of 10^{-10} or even less. Can you explain where this trend come from ?

HINT: to see clearly the trend, you can visualize the difference between $y(\hat{X}) - y(X)$

Part Two - The learning rate α

For this part you will write your code in the file **alpha.py**.
We will focus on the concept of learning rate, in consequence, you will have to fit with the method **MyLinearRegression.fit** you codedpreviously.

You are expected to:

- Find a good learning rate α to fit the data with the hypothesis you have chosen in the previous part, you choice will be motivated by the convergence reached in the minimum cycle (see figure 3).

figure is missing

Be sure to understand the underlying concept and be able to answer to those questions evaluators may ask:

- What happens if you choose a learning rate too big ?
- How the metric and the θ's evolve with a slighly too big learning rate ?
- How the convergence evolves in function of the cycle ? Do you have any idea to speed up the convergence as the numbre of cycle increase ? (It is an open question, be creative and smart).