

## But du TP :

- savoir charger et manipuler une base de données à partir du bibliothèque Scikits-learn et pandas
- Créer une base de données et la manipuler

## Introduction :

Ce TP nous allons avoir une idée générale sur les bibliothèques Pandas et Scikits-learn et leurs utilités dans la machine Learning et les différents algorithmes d'apprentissage que ces deux librairies couvrent. On va essayer d'exécuter le code pour charger et créer (load) une base de données, connaître ces classes et ses attributs et la manipuler

## Manipulation du TP :

Des installations préalables : on doit installer les bibliothèques pandas et scikits-learn

```
Anaconda Prompt (anaconda)

(base) C:\Users\G I E>conda install scikit-learn
Collecting package metadata (current_repodata.json): done
Solving environment: done

# All requested packages already installed.

(base) C:\Users\G I E>
(base) C:\Users\G I E>
```

```
Sélection Anaconda Prompt (anaconda)

(base) C:\Users\G I E>conda install pandas
Collecting package metadata (current_repodata.json): done
Solving environment: done

# All requested packages already installed.

(base) C:\Users\G I E>_
```

## I -Base de Données du cancer du sein

Maintenant on va charger la base de données du cancer du sein dans la variable 'breast' en exécutant le code déjà existé dans la fiche de TP1 :

```
Entrée [25]: from sklearn.datasets import load_breast_cancer
breast = load_breast_cancer()
```

Visualiser son type avec commande 'type()'

```
Entrée [26]: print(type(breast))

<class 'sklearn.utils.Bunch'>
```

La variable breast est de type class

Visualiser les attributs de cette classe avec la commande 'dir()'

```
Entrée [27]: print(dir(breast))

['DESCR' 'data' 'feature_names' 'filename', 'target', 'target_names']
```

```
Entrée [18]: type(breast.target)
```

```
Out[18]: numpy.ndarray
```

```
Entrée [15]: type(breast.filename)
```

```
Out[15]: str
```

```
Entrée [16]: type(breast.target_names)
```

```
Out[16]: numpy.ndarray
```

On obtient une liste d'attributs (sous classes de la classe) où chaque attribut est de type spécifique

## La liste des caractéristiques

```
Entrée [7]: print(breast.feature_names)
```

```
[ 'mean radius' 'mean texture' 'mean perimeter' 'mean area'
'mean smoothness' 'mean compactness' 'mean concavity'
'mean concave points' 'mean symmetry' 'mean fractal dimension'
'radius error' 'texture error' 'perimeter error' 'area error'
'smoothness error' 'compactness error' 'concavity error'
'concave points error' 'symmetry error' 'fractal dimension error'
'worst radius' 'worst texture' 'worst perimeter' 'worst area'
'worst smoothness' 'worst compactness' 'worst concavity'
'worst concave points' 'worst symmetry' 'worst fractal dimension']
```

On visualise Les informations

On obtient Une liste de sous listes des valeurs numériques

Entrée [25]: `print(breast.data[:2])`

```
[[1.799e+01 1.038e+01 1.228e+02 1.001e+03 1.184e-01 2.776e-01 3.001e-01
 1.471e-01 2.419e-01 7.871e-02 1.095e+00 9.053e-01 8.589e+00 1.534e+02
 6.399e-03 4.904e-02 5.373e-02 1.587e-02 3.003e-02 6.193e-03 2.538e+01
 1.733e+01 1.846e+02 2.019e+03 1.622e-01 6.656e-01 7.119e-01 2.654e-01
 4.601e-01 1.189e-01]
 [2.057e+01 1.777e+01 1.329e+02 1.326e+03 8.474e-02 7.864e-02 8.690e-02
 7.017e-02 1.812e-01 5.667e-02 5.435e-01 7.339e-01 3.398e+00 7.408e+01
 5.225e-03 1.308e-02 1.860e-02 1.340e-02 1.389e-02 3.532e-03 2.499e+01
 2.341e+01 1.588e+02 1.956e+03 1.238e-01 1.866e-01 2.416e-01 1.860e-01
 2.750e-01 8.902e-02]]
```

les noms des deux buts d'apprentissage ce sont deux classes ( deux types du cancer du sein)

```
Entrée [20]: print(breast.target_names)
```

```
['malignant' 'benign']
```

Entrée [21]: `T = breast.target`

Entrée [22]: `print(T)`

[illegible]

```
Entrée [23]: for i in [0,1]:
              print("la classe : %s, contient un nombre d'exemplaires: %s" % (i, len(T[ T == i]) ) )
```

la classe : 0, contient un nombre d'exemplaires: 212  
la classe : 1, contient un nombre d'exemplaires: 357

Le nombre d'échantillons malignes (classe 0) et bénigne (classe1)

La description de la base de données

```
Entrée [13]: print (breast.DESCR)
```

```
.. _breast_cancer_dataset:

Breast cancer wisconsin (diagnostic) dataset
-----

**Data Set Characteristics:**

: Number of Instances: 569

: Number of Attributes: 30 numeric, predictive attributes and the class

: Attribute Information:
  - radius (mean of distances from center to points on the perimeter)
  - texture (standard deviation of gray-scale values)
  - perimeter
  - area
  - smoothness (local variation in radius lengths)
  - compactness (perimeter^2 / area - 1.0)
  - concavity (severity of concave portions of the contour)
```

Dans la description on trouve une liste des caractéristiques , de statistiques , nombres d'échantillons , l'auteur et d'autre informations

## II- Lecture et manipulation base de données de type csv

on va créer un fichier csv et le remplir , ligne par ligne, les notes et puis l'enregistrer dans un dossier peut être traité par anaconda

id,Nom				
A	B	C	D	E
1	id,Noms,Anglais,Maths,Arabe,Allemand			
2	1,Sarra	15,17,10,20		
3	2,olfa	10,9,17,15		
4	3,Ahmed	11,9,6,20		
5	4,taheni	10,5,18,19		
6	6,rami	19,19,19,19		
7				

On exécute le code suivant qui sert à lire le contenu du fichier et d'afficher les données :

```
Entrée [2]: import csv
with open('liste1.csv',newline='') as g: #Ouvrir un fichier CSV
    T=[] #créer une liste vide
    lecture=csv.reader(g) #charger les lignes
    print('',end='\n')
    print('les lignes sont :',end='\n')
    for l in lecture: #Itération ligne par ligne
        print(l, end='\n') #afficher la ligne
        T.append(l) #ajouter le contenu la ligne l à chaque itération|
```

```
les lignes sont :
['id', 'Noms', 'Anglais', 'Maths', 'Arabe', 'Allemand ']
['1', 'Sarrah', '15', '17', '10', '20']
['2', 'olfa', '10', '9', '17', '15']
['3', 'Ahmed', '11', '9', '6', '20']
['4', 'taheni', '10', '5', '18', '19']
['6', 'rami', '19', '19', '19', '19']
['7', 'Asma', '11', '3', '19', '14']
['7', 'Asma', '11', '3', '19', '14']
```

En python3, on peut effectuer la boucle for dans un tableau ,ou chaine de caractères ou une liste c'est pour ça on a chargé les données du 'liste1.csv' dans la variable lecture (de type csv.reader) et on a effectué un boucle for : en itérant ligne par ligne on va afficher sa contenu et remplir la liste T par concaténation à la fin en utilisant ' append()'

```
Entrée [2]: type(lecture)
```

```
Out[2]: _csv.reader
```

```
Entrée [3]: type(g)
```

```
Out[3]: _io.TextIOWrapper
```

```
Entrée [4]: type(T)
```

```
Out[4]: list
```

Chaque ligne de lecture est une liste, T est une liste des listes

```
Entrée [13]: print(T)
```

```
[['id', 'Noms', 'Anglais', 'Maths', 'Arabe', 'Allemand '], ['1', 'Sarrah', '15', '17', '10', '20'], ['2', 'olfa', '10', '9', '17', '15'], ['3', 'Ahmed', '11', '9', '6', '20'], ['4', 'taheni', '10', '5', '18', '19'], ['6', 'rami', '19', '19', '19', '19']]
```

Maintenant on va reconstruire ce tableau dans le fichier csv vide 'liste2.csv'

```
Entrée [5]: with open('liste2.csv','w',newline='') as h:#Ouvrir le fichier liste2 en écriture, w=%writefile
write=csv.writer(h) # préparation à l'écriture
for j in T: # Pour chaque ligne du tableau...
    write.writerow(j) # Mettre dans la variable écrire cette nouvelle ligne
print(' ',end='\n')
print('longueur de T : ',len(T))
```

longueur de T : 6

Et si on affiche le contenu du liste2.csv on trouve le même tableau

```
Entrée [15]: with open('liste2.csv',newline='') as f: #Ouvrir un fichier CSV
              lecture=csv.reader(f) #charger les lignes
              for k in lecture: #Itération ligne par ligne
                  print(k, end='\n') #afficher la ligne

['id', 'Noms', 'Anglais', 'Maths', 'Arabe', 'Allemand']
['1', 'Sarrah', '15', '17', '10', '20']
['2', 'olfa', '10', '9', '17', '15']
['3', 'Ahmed', '11', '9', '6', '20']
['4', 'taheni', '10', '5', '18', '19']
['6', 'rami', '19', '19', '19', '19']
```

On peut ajouter une ligne au fichier liste1.csv

```
Entrée [16]: with open('liste1.csv','a',newline='') as g: #ouvrir le fichier pour l'ajout d'une ligne,
              #a=append

              write=csv.writer(g)
              write.writerow(['7','Asma','11','3','19','14']) #ajouter une nouvelle ligne
```

	A	B	C	D
1	id,Noms,Anglais,Maths,Arabe,Allemand			
2	1,Sarrah,15,17,10,20			
3	2,olfa,10,9,17,15			
4	3,Ahmed,11,9,6,20			
5	4,taheni,10,5,18,19			
6	6,rami,19,19,19,19			
7	7,Asma,11,3,19,14			
8				

On remarque que la ligne s'ajoute au tableau Excel dans le fichier liste1.csv

Maintenant on va calculer la moyenne, pour cela on construit la fonction moyenne qui prend en variable d'entrée une liste et retourne un réel

```
Entrée [4]: def moyenne(lis):
              m=0
              k=len(lis)
              for i in range(0,k):
                  m=m+int(lis[i])

              return m/k
```

Dans la variable moy de type liste on va mettre juste les moyennes après avoir les calculé par la fonction moyenne dont la variable d'entrée est 'l' la liste des notes en chaque ligne de T

```
Entrée [60]: moy=[]
              for i in range(1,len(T)):
                  l=T[i][2:]
                  print(l)
                  moy.append(moyenne(l))
                  T[i].append(moyenne(l))

              T[0].append('Moyenne')
```

```
['15', '17', '10', '20', 15.5]
['10', '9', '17', '15', 12.75]
['11', '9', '6', '20', 11.5]
['10', '5', '18', '19', 13.0]
['19', '19', '19', '19', 19.0]
['11', '3', '19', '14', 11.75]
```

dans ce code on affiche la colonne des moyennes et la liste T

```
Entrée [61]: print('',end='\n')
              print('colonne des moyennes : ',end='\n')
              print(moy)
              print(T)
```

```
colonne des moyennes :
[15.4, 12.6, 11.4, 13.0, 19.0, 11.6]
[['id', 'Noms', 'Anglais', 'Maths', 'Arabe', 'Allemand', 'Moyenne', 'Moyenne'], ['1', 'Sarrah', '15', '17', '10', '20', 15.5, 15.4], ['2', 'Olfa', '10', '9', '17', '15', 12.75, 12.6], ['3', 'Ahmed', '11', '9', '6', '20', 11.5, 11.4], ['4', 'Taheni', '10', '5', '18', '19', 13.0, 13.0], ['6', 'Rami', '19', '19', '19', '19', 19.0, 19.0], ['7', 'Asma', '11', '3', '19', '14', 11.75, 11.6]]
```

après avoir créé un fichier csv vide nommé 'liste3' et le fermé on exécute ce code

```
Entrée [62]: with open('liste3.csv','a',newline='') as f:
              ecrire=csv.writer(f)
              for i in T:
                  ecrire.writerow(i)
```

2	id,Noms,Anglais,Maths,Arabe,Allemand,Moyenne
3	1,Sarrah,15,17,10,20,15.5
4	2,Olfa,10,9,17,15,12.75
5	3,Ahmed,11,9,6,20,11.5
6	4,Taheni,10,5,18,19,13.0
7	6,Rami,19,19,19,19,19.0
8	7,Asma,11,3,19,14,11.75

La colonne des moyennes s'y ajoute

## Objectif :

Le but du TP N2 est de traiter les données de la base de données afin d'extraire les caractéristiques nécessaires pour la phase d'apprentissage. Pour cela on va se servir des bibliothèques sous python3 et algorithmes adéquates pour assurer ces objectifs. Par la suite on va essayer différent modèle d'entraînement et choisir le plus performant pour la problématique.

## Manipulation :

On va exécuter les instructions notées dans la fiche de TP et visualiser le résultat et découvrir l'utilité de quelque commandes et algorithmes.

### • Importer la data base, visualisation, information et description statistique :

On va se servir de la bibliothèques pandas qui renferme des commandes convenables pour importer, visualiser la base de données et avoir une répartition statistique des informations et leurs descriptions

Entrée [27]: 

```
import pandas as pd
data = pd.read_csv('data.csv')
```

Entrée [28]: 

```
data.head()
```

Out[28]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

La commande `____.head()` permet de donner un aperçu sur la base de données donc il s'agit des informations les passagers de Titanic leurs id, âge, sexe et s'ils ont survécu..

Entrée [29]: 

```
data.info()
```

`____.info()` permet de donner des informations concernant les colonnes de notre data base La colonne 'Name' contient 891 composants et de type 'object'. L'espace mémoire de la data base est 83.7 KiloByte

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
#   Column      Non-Null Count  Dtype
---  ---
0   PassengerId  891 non-null    int64
1   Survived     891 non-null    int64
2   Pclass       891 non-null    int64
3   Name         891 non-null    object
4   Sex          891 non-null    object
5   Age          714 non-null    float64
6   SibSp        891 non-null    int64
7   Parch        891 non-null    int64
8   Ticket       891 non-null    object
9   Fare         891 non-null    float64
10  Cabin        204 non-null    object
11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```



Entrée [30]: data.describe()

Out[30]:

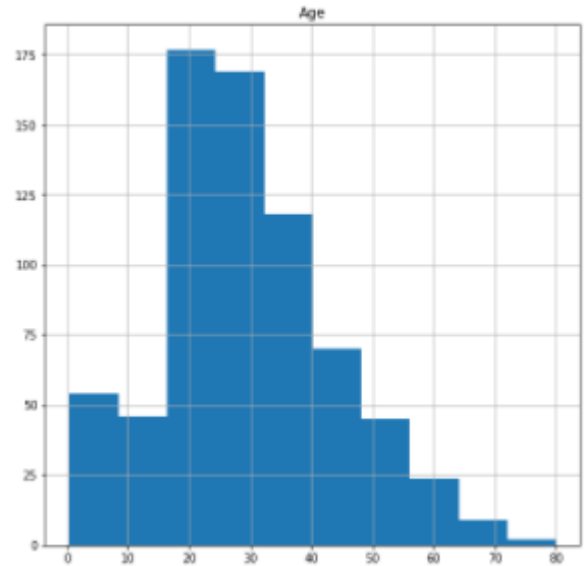
	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

\_\_\_\_.describe() permet de calculer les données statistiques des variables

Entrée [31]: 

```
import matplotlib.pyplot as plt
data.hist(figsize=(30,30))
plt.show()
```

On a visualisé les données en utilisant l'histogramme ça facilite la lecture des données par exemple : la plupart des passagers sont âgés de 20 à 30 ans



## • Tri des valeurs :

Entrée [32]: data.sort\_values(by=['Sex'],ascending=True )

Out[32]:

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
383	384	1	Holverson, Mrs. Alexander Oskar (Mary Aline To...	female	35.0	1	0	113789	52.0000	NaN	S
218	219	1	Bazzani, Miss. Albina	female	32.0	0	0	11813	76.2917	D15	C
609	610	1	Shutes, Miss. Elizabeth W	female	40.0	0	0	PC 17582	153.4625	C125	S
216	217	1	Honkanen, Miss. Ellina	female	27.0	0	0	STON/O2. 3101283	7.9250	NaN	S
215	216	1	Newell, Miss. Madeleine	female	31.0	1	0	35273	113.2750	D36	C
...	...	...	...	...	...	...	...	...	...	...	...
371	372	0	Wiklund, Mr. Jakob Alfred	male	18.0	1	0	3101267	6.4958	NaN	S
372	373	0	Beavan, Mr. William Thomas	male	19.0	0	0	323951	8.0500	NaN	S
373	374	0	Ringhini, Mr. Sante	male	22.0	0	0	PC 17760	135.6333	NaN	C
360	361	0	Skoog, Mr. Wilhelm	male	40.0	1	4	347088	27.9000	NaN	S
890	891	0	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

On peut trier les éléments des colonnes dans l'ordre croissant ou décroissant, ici on a ordonné les composants de la colonne sexe d'une manière croissante et donc ceci est imposé aux éléments des autres colonnes, on remarque que les informations des passagers femelles sont avancées par rapport aux autres de type male .



```
Entrée [33]: data.sort_values(by=['Cabin'], ascending=False)[:20]
```

On remarque que les éléments chaque colonne suivent l'ordonnement de la colonne 'Cabin' dont ses éléments sont descendants alphabétiquement. 20 éléments sont visualisés

Parch	Ticket	Fare	Cabin	Embarked
0	113784	35.5000	T	S
2	PP 9549	16.7000	G6	S
1	347054	10.4625	G6	S
1	347054	10.4625	G6	S
1	PP 9549	16.7000	G6	S
1	230136	39.0000	F4	S
1	230136	39.0000	F4	S
0	383121	7.7500	F38	Q
0	C.A. 29395	10.5000	F33	S
0	248733	13.0000	F33	S
0	C.A. 34260	10.5000	F33	S

- **Corrélation :**

```
Entrée [34]: data.corr()
```

Out[34]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
PassengerId	1.000000	-0.005007	-0.035144	0.036847	-0.057527	-0.001652	0.012658
Survived	-0.005007	1.000000	-0.338481	-0.077221	-0.035322	0.081629	0.257307
Pclass	-0.035144	-0.338481	1.000000	-0.369226	0.083081	0.018443	-0.549500
Age	0.036847	-0.077221	-0.369226	1.000000	-0.308247	-0.189119	0.096067
SibSp	-0.057527	-0.035322	0.083081	-0.308247	1.000000	0.414838	0.159651
Parch	-0.001652	0.081629	0.018443	-0.189119	0.414838	1.000000	0.216225
Fare	0.012658	0.257307	-0.549500	0.096067	0.159651	0.216225	1.000000

Comme on a un nombre énorme de caractéristiques on cherche à trouver une relation mathématiquement dite corrélation entre ces données ceci est assuré par la commande `__.corr()`. On voit une matrice carrée de diagonale 1 est symétrique . les variables ayant une corrélation plus proche de -1 ou 1 sont corrélés : par exemple {Pclass,Fare} et {SibSp,Parch}.

On peut se concentrer sur la corrélation de la variable 'Survived' avec les autres variables.

On constate que 'survived' et 'Pclass' sont corrélés de valeur -0.3384.

Ceci est bien traduit si on revient aux données la plupart de la classe 3 sont non survived 0

Tant que la plupart de la classe 1 sont survived 1

```
Entrée [35]: data.corr()["Survived"]
```

```
Out[35]: PassengerId    -0.005007
Survived      1.000000
Pclass        -0.338481
Age           -0.077221
SibSp         -0.035322
Parch         0.081629
Fare          0.257307
Name: Survived, dtype: float64
```

- Remplacement des valeurs :

En visualisant les valeurs de la colonne Age on remarque qu'il y a des données de types 'NaN' (valeurs manquantes) et ne sont pas numériques(réels) et donc ne peuvent pas être manipulées dans le programme d'entraînement, C'est pourquoi on va les remplacer par la moyenne d'âge

```
Age
22.0
38.0
26.0
35.0
35.0
NaN
54.0
2.0
27.0
14.0
4.0
58.0
20.0
39.0
14.0
55.0
2.0
NaN
31.0
NaN
```

```
: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
#   Column                Non-Null Count  Dtype
---  -
0   PassengerId            891 non-null    int64
1   Survived               891 non-null    int64
2   Pclass                 891 non-null    int64
3   Name                   891 non-null    object
4   Sex                    891 non-null    object
5   Age                    714 non-null    float64
6   SibSp                  891 non-null    int64
7   Parch                  891 non-null    int64
8   Ticket                 891 non-null    object
9   Fare                   891 non-null    float64
10  Cabin                  204 non-null    object
11  Embarked               889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

```
Entrée [46]: Av_Age=data.Age.mean()
print(Av_Age)

29.69911764705882
```

```
Entrée [47]: data['Age']=data.Age.fillna(Av_Age)
```

```
Entrée [49]: data.head(20)
```

On a bien les valeurs NAN sont remplacés par 29.69

Et par suite les nombres d'élément de type réel dans la colonne Age(vecteur) est transformé de 714 à 891

```
Entrée [50]: data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
#   Column                Non-Null Count  Dtype
---  -
0   PassengerId            891 non-null    int64
1   Survived               891 non-null    int64
2   Pclass                 891 non-null    int64
3   Name                   891 non-null    object
4   Sex                    891 non-null    object
5   Age                    891 non-null    float64
6   SibSp                  891 non-null    int64
7   Parch                  891 non-null    int64
8   Ticket                 891 non-null    object
9   Fare                   891 non-null    float64
10  Cabin                  204 non-null    object
11  Embarked               889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

	Name	Sex	Age
0	Braund, Mr. Owen Harris	male	22.000000
1	Allen, Mr. William Henry	male	35.000000
2	Heikkinen, Miss. Laina	female	26.000000
3	McCarthy, Mr. Timothy J	male	54.000000
4	Moran, Mr. James	male	29.699118
5	McCarthy, Mr. Timothy J	male	54.000000
6	son, Master. Gosta Leonard	male	2.000000
7	car W (Elisabeth Vilhelmina Berg)	female	27.000000
8	rs. Nicholas (Adele Achem)	female	14.000000
9	strom, Miss. Marguerite Rut	female	4.000000
10	Bonnell, Miss. Elizabeth	female	58.000000
11	ndercock, Mr. William Henry	male	20.000000
12	ndersson, Mr. Anders Johan	male	39.000000
13	iss. Hulda Amanda Adolfina	female	14.000000
14	tt, Mrs. (Mary D Kingcome)	female	55.000000
15	Rice, Master. Eugene	male	2.000000
16	illiams, Mr. Charles Eugene	male	29.699118
17	a, Mrs. Julius (Emelia Maria Vande...	female	31.000000
18	Masselmani, Mrs. Fatima	female	29.699118

## • Principe Component Analysis :

Ce principe consiste à réduire les caractéristiques et garder celles qui sont utiles . on va l'utiliser pour transformer les données de Fare et Classes en un seul vecteur puisqu'ils sont corrélés(  
**corr{Pclass, Fare} = -0.549**

```
Entrée [54]: from sklearn.decomposition import PCA
p = PCA(n_components=1) #appliquer Le PCA
C = p.fit_transform(l)
print('C= \n',C[:10])

C=
[[-24.95953384]
 [ 39.0895218 ]
 [-24.2845627 ]
 [ 20.90699914]
 [-24.15956804]
 [-23.75128549]
 [ 19.66955204]
 [-11.13512486]
 [-21.07639985]
 [-2.1304629 ]]
```

```
Entrée [51]: l = data.loc[:,['Fare','Pclass']]
print('l= \n',l[:10])

l=
   Fare  Pclass
0  7.2500      3
1 71.2833      1
2  7.9250      3
3 53.1000      1
4  8.0500      3
5  8.4583      3
6 51.8625      1
7 21.0750      3
8 11.1333      3
9 30.0708      2
```

On affecte les deux colonnes 'Fare ' et 'Pclasse' dans la variable 'l' puis on applique le **PCA** , on obtient le vecteur C qui est un mixage entre les deux vecteurs corrélés, chaque classe est représenté par sa moyenne.

```
Entrée [57]: data['ADD_C']=C[:,0] #ajouter une colonne à la base de données
data.head()
```

Out[57]:

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	ADD_C	
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S	-24.959534
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C	39.089522
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S	-24.284563
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S	20.906999
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S	-24.159568

On ajoute le vecteur obtenu dans la position 0

## Conversion en donnée logique utiles :

```
Entrée [83]: GT = data["Survived"]
D = data.drop(["Name", "Ticket", "Cabin", "Embarked", "PassengerId", "Survived"],
axis=1)# axis=1 afin d'éliminer les colonnes
```

```
Entrée [84]: print(GT)
```

```
0      0
1      1
2      1
3      1
4      0
..
886    0
887    1
888    0
889    1
890    0
Name: Survived, Length: 891, dtype: int64
```

On va se baser sur le vecteur 'survived' comme un vecteur vérité terrain et le mettre dans var GT, puis on va se débarrasser des autres vecteurs par la commande `__.drop`

On va convertir les données de la colonne 'sexe' en 0 et 1 (valeur logique), pour être utile dans le programme d'apprentissage

Entrée [85]: `DB= pd.get_dummies(D, columns=["Sex"])`  
`print(DB)`

	Pclass	Age	SibSp	Parch	Fare	ADD_C	Sex_female	Sex_male
0	3	22.0	1	0	7.2500	-24.959534	0	1
1	1	38.0	1	0	71.2833	39.089522	1	0
2	3	26.0	0	0	7.9250	-24.284563	1	0
3	1	35.0	1	0	53.1000	20.906999	1	0
4	3	35.0	0	0	8.0500	-24.159568	0	1
...	...	...	...	...	...	...	...	...
886	2	27.0	0	0	13.0000	-19.200533	0	1
887	1	19.0	0	0	30.0000	-2.192013	1	0
888	3	NaN	1	2	23.4500	-8.760226	1	0
889	1	26.0	0	0	30.0000	-2.192013	0	1
890	3	32.0	0	0	7.7500	-24.459555	0	1

[891 rows x 8 columns]

Maintenant on va remplacer tous les valeurs NAN ,là où ils apparaissent dans chaque colonne, par la médiane en se basant sur la fonction Imputer

Entrée [86]: `from sklearn.impute import SimpleImputer`  
`import numpy as np`  
`imputer = SimpleImputer(missing_values=np.nan, strategy='median')`  
`New = imputer.fit_transform(DB)`  
`print(New)`

[	3.	22.	1.	...	-24.95953384	0.
	1.					
[	1.	38.	1.	...	39.0895218	1.
	0.					
[	3.	26.	0.	...	-24.2845627	1.
	0.					
...						
[	3.	28.	1.	...	-8.76022639	1.
	0.					
[	1.	26.	0.	...	-2.19201334	0.
	1.					
[	3.	32.	0.	...	-24.45955521	0.
	1.					

On injecte les nouvelles valeurs dans la variable X : on obtient une nouvelle data base 'X', qui dérive de la base 'data', dont ses données sont tous réels

Entrée [87]: `# réinjecter Les nouvelles valeurs`  
`X = pd.DataFrame(New, columns=DB.columns)`  
`print(X)`

	Pclass	Age	SibSp	Parch	Fare	ADD_C	Sex_female	Sex_male
0	3.0	22.0	1.0	0.0	7.2500	-24.959534	0.0	1.0
1	1.0	38.0	1.0	0.0	71.2833	39.089522	1.0	0.0
2	3.0	26.0	0.0	0.0	7.9250	-24.284563	1.0	0.0
3	1.0	35.0	1.0	0.0	53.1000	20.906999	1.0	0.0
4	3.0	35.0	0.0	0.0	8.0500	-24.159568	0.0	1.0
...	...	...	...	...	...	...	...	...
886	2.0	27.0	0.0	0.0	13.0000	-19.200533	0.0	1.0
887	1.0	19.0	0.0	0.0	30.0000	-2.192013	1.0	0.0
888	3.0	28.0	1.0	2.0	23.4500	-8.760226	1.0	0.0
889	1.0	26.0	0.0	0.0	30.0000	-2.192013	0.0	1.0
890	3.0	32.0	0.0	0.0	7.7500	-24.459555	0.0	1.0

[891 rows x 8 columns]

X est une base de données numériques dont chaque vecteur est de longueur 891

Entrée [90]: `X.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 8 columns):
#   Column      Non-Null Count  Dtype
---  ---
0   Pclass      891 non-null    float64
1   Age         891 non-null    float64
2   SibSp       891 non-null    float64
3   Parch      891 non-null    float64
4   Fare        891 non-null    float64
5   ADD_C       891 non-null    float64
6   Sex_female  891 non-null    float64
7   Sex_male    891 non-null    float64
dtypes: float64(8)
memory usage: 55.8 KB
```

Finalement on normalise les données des attribues en se servant du module `StandardScaler`. DS la nouvelle base de

Entrée [25]: `from sklearn.preprocessing import StandardScaler`  
`S= StandardScaler()`  
`XN = S.fit_transform(X)`  
`DS = pd.DataFrame(XN, columns=X.columns)`  
`DS.head()`

Out[25]:

	Pclass	Age	SibSp	Parch	Fare	ADD_C	Sex_female	Sex_male
0	0.827377	-0.592481	0.432793	-0.473674	-0.502445	-0.502531	-0.737695	0.737695
1	-1.566107	0.638789	0.432793	-0.473674	0.786845	0.787022	1.355574	-1.355574
2	0.827377	-0.284663	-0.474545	-0.473674	-0.488854	-0.488941	1.355574	-1.355574
3	-1.566107	0.407926	0.432793	-0.473674	0.420730	0.420938	1.355574	-1.355574
4	0.827377	0.407926	-0.474545	-0.473674	-0.486337	-0.486425	-0.737695	0.737695

### • Base de données d'apprentissage / base de données de Test :

Maintenant on prépare les variables d'entrée de notre programme d'entraînement :  
 Donc on divise notre base de données prétraité en deux parties : l'une pour l'apprentissage et l'autre pour le test. Sans oublier de préparer les matrices Target dont leur composantes sont 0 et 1

Entrée [26]: `from sklearn import model_selection`  
`X_app, X_val, Y_app, Y_val = model_selection.train_test_split(DS, GT, test_size=0.4)`

Entrée [100]:

```
type(X_app)
X_app.head()
X_app.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 534 entries, 228 to 815
Data columns (total 8 columns):
#   Column      Non-Null Count  Dtype
---  ---
0   Pclass      534 non-null    float64
1   Age         534 non-null    float64
2   SibSp       534 non-null    float64
3   Parch      534 non-null    float64
4   Fare        534 non-null    float64
5   ADD_C       534 non-null    float64
6   Sex_female  534 non-null    float64
7   Sex_male    534 non-null    float64
dtypes: float64(8)
memory usage: 37.5 KB
```

Entrée [101]: `X_val.info()`

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 357 entries, 285 to 218
Data columns (total 8 columns):
#   Column      Non-Null Count  Dtype
---  ---
0   Pclass      357 non-null    float64
1   Age         357 non-null    float64
2   SibSp       357 non-null    float64
3   Parch      357 non-null    float64
4   Fare        357 non-null    float64
5   ADD_C       357 non-null    float64
6   Sex_female  357 non-null    float64
7   Sex_male    357 non-null    float64
dtypes: float64(8)
memory usage: 25.1 KB
```

Division des données : 60% (parmi 891 exemplaires) pour l'apprentissage et 40% pour le teste

Y\_app vecteur Target, issue de 60% de GT

```
Entrée [104]: type(Y_app) #Y_app est le target de l'apprentissage !
               print(Y_app)

228    0
474    0
817    0
589    0
764    0
..
842    1
813    0
730    1
509    1
815    0
Name: Survived, Length: 534, dtype: int64
```

- **Choix du modèle d'entraînement :**

On va maintenant exécuter trois modèles d'apprentissage et calculer à chaque fois entre le résultat obtenu et celui estimé

- ♦ **Régression linéaire :**

```
Entrée [105]: #Regression_Lineaire
               from sklearn.linear_model import LinearRegression
               import numpy as np
               from sklearn.model_selection import cross_val_score
               LR = LinearRegression()
               LR.fit(X_app, Y_app)
               S = cross_val_score(LR, X_app, Y_app, scoring="neg_mean_squared_error", cv=10)
               LR_rmse = np.sqrt(-S)
               print("Moyenne", LR_rmse.mean())
               print("Ecart-type", LR_rmse.std())

Moyenne 0.39319417835118786
Ecart-type 0.024430577573850587
```

- ♦ **Decision Tree Regressor :**

```
Entrée [30]: #decision_lineaire
              from sklearn.tree import DecisionTreeRegressor
              TR = DecisionTreeRegressor()
              TR.fit(X_app, Y_app)
              TRsc = cross_val_score(TR, X_app, Y_app, scoring="neg_mean_squared_error", cv=10)
              TR_rmse = np.sqrt(-TRsc)
              print(TR_rmse)
              print("Moyenne", TR_rmse.mean())
              print("Ecart-type", TR_rmse.std())

[0.52704628 0.40156887 0.33487299 0.50945914 0.55439922 0.4955581
 0.47772171 0.43568225 0.55371814 0.41208169]
Moyenne 0.4702108370271242
Ecart-type 0.0686513065998808
```



## ◆ Random Forest Regressor :

```
Entrée [31]: #Random_Forest_Regression
from sklearn.ensemble import RandomForestRegressor
RFR = RandomForestRegressor()
RFR.fit(X_app,Y_app)
Fsc = cross_val_score(RFR, X_app,Y_app, scoring="neg_mean_squared_error", cv=10)
RFR_rmse = np.sqrt(-Fsc)
print(RFR_rmse)
print("Moyenne", RFR_rmse.mean())
print("Ecart-type", RFR_rmse.std())

[0.37293409 0.32246347 0.30510445 0.40323465 0.41748346 0.40763803
 0.41497611 0.32724524 0.37689745 0.35728434]
Moyenne 0.37052612940804036
Ecart-type 0.03913172570930308
```

La régression linéaire a la plus faible moyenne d'erreur 0.039 et le plus faible écart type 0.024 entre les résultats et l'estimé . donc c'est la plus efficace



## Objectif :

Développer l'algorithme du code source de la regression linéaire Simple

## Manipulation :

On commence par importer les bibliothèques numpy, pandas, matplotlib et scipy

```
Entrée [2]: from numpy import *  
from numpy.random import *  
from math import *  
from matplotlib.pyplot import *  
from scipy.misc import *  
import pandas as pd
```

On lit le fichier ex1data1 à partir de son path

```
[3]: #path=  
data = pd.read_csv('ex1data1.txt', header= None, names=['population', 'profit'])
```

On affiche les 10 premiers lignes :

on calcule les données statistiques de chacune des variables :

```
Entrée [4]: print('data='), data.head(10)  
data=
```

```
Out[4]: (None,  
        population  profit  
0         6.1101    17.5920  
1         5.5277     9.1302  
2         8.5186    13.6620  
3         7.0032    11.8540  
4         5.8598     6.8233  
5         8.3829    11.8860  
6         7.4764     4.3483  
7         8.5781    12.0000  
8         6.4862     6.5987  
9         5.0546     3.8166)
```

```
Entrée [6]: print('data.describe'), data.describe()  
data.describe
```

```
Out[6]: (None,  
        population  profit  
count    97.000000    97.000000  
mean      8.159800     5.839135  
std       3.869884     5.510262  
min       5.026900    -2.680700  
25%      5.707700     1.986900  
50%      6.589400     4.562300  
75%      8.578100     7.046700  
max      22.203000    24.147000)
```

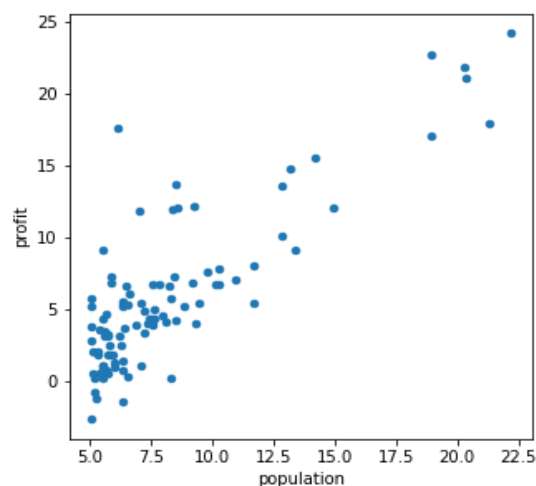
```
[7]: import matplotlib.pyplot as plt  
data.plot(kind='scatter', x='population', y='profit', figsize=(5,5))
```

```
: [7]: <matplotlib.axes._subplots.AxesSubplot at 0x1733231ec88>
```

On trace le graphe profil en fonction de la population :

On remarque que le nuage des points est assimilé à une droite linéaire, le profil et la population sont linéairement dépendants.

On peut s'assurer en cherchant une corrélation.



Effectivement, le profit et la population sont corrélés

Entrée [8]: `data.corr()`

Out[8]:

	population	profit
population	1.000000	0.837873
profit	0.837873	1.000000

On ajoute une colonne ones (biais) pour pouvoir effectuer la multiplication de  $X(n \times 2)$  et  $\mu(1 \times 2)$

```
Entrée [9]: #ajouter une ligne
data.insert(0, 'Ones', 1)
print('new data = \n'), data.head(10)

new data =
```

```
Out[9]: (None,
      Ones  population  profit
0      1      6.1101  17.5920
1      1      5.5277   9.1302
2      1      8.5186  13.6620
3      1      7.0032  11.8540
4      1      5.8598   6.8233
5      1      8.3829  11.8860
6      1      7.4764   4.3483
7      1      8.5781  12.0000
8      1      6.4862   6.5987
9      1      5.0546   3.8166)
```

On sépare les colonnes :

```
Entrée [10]: #séparer les données d'entrée et les sorties
cols=data.shape[1] #nbre de colonne
print(cols)
x=data.iloc[:,0:cols-1]
y=data.iloc[:,cols-1:cols]

3
```

On a x de taille  $97 \times 2$  et y de taille  $97 \times 1$

1]: `x.head(10)`

`y.head(10)`

1]:

	Ones	population
0	1	6.1101
1	1	5.5277
2	1	8.5186
3	1	7.0032
4	1	5.8598
5	1	8.3829
6	1	7.4764
7	1	8.5781
8	1	6.4862
9	1	5.0546

	profit
0	17.5920
1	9.1302
2	13.6620
3	11.8540
4	6.8233
5	11.8860
6	4.3483
7	12.0000
8	6.5987
9	3.8166

On convertit x et y en matrice

```
17]: x=np.matrix(x)
      print(x.shape)
      print(x)
```

(97, 2)

```
[[ 1.    6.1101]
 [ 1.    5.5277]
 [ 1.    8.5186]
 [ 1.    7.0032]
 [ 1.    5.8598]
 [ 1.    8.3829]
 [ 1.    7.4764]
 [ 1.    8.5781]
 [ 1.    6.4862]
 [ 1.    5.0546]
 [ 1.    5.7107]
 [ 1.   14.164 ]
 [ 1.    5.734 ]
 [ 1.    8.4084]
 [ 1.    5.6407]
```

```
[19]: y=np.matrix(y)
      print(y.shape)
      y
```

(97, 1)

```
[19]: matrix([[17.592 ],
               [ 9.1302 ],
               [13.662 ],
               [11.854 ],
               [ 6.8233 ],
               [11.886 ],
               [ 4.3483 ],
               [12.     ],
               [ 6.5987 ],
               [ 3.8166 ],
               [ 3.2522 ],
               [15.505 ],
               [ 3.1551 ],
               [ 7.2258 ],
               [ 0.71618],
               [ 3.5129 ],
               [ 5.3048 ],
               [ 0.56077],
               [ 3.6518 ],
               [ 5.3893 ]]
```

On définit mu :

```
[24]: m=np.array([0,0])
      mu=np.matrix(m)
```

```
[25]: print('mu.shape \n', mu.shape)
      print('mu= \n',mu)
```

```
mu.shape
(1, 2)
mu=
[[0 0]]
```

On définit la fonction **computeCost** qui permet de calculer l'écart de l'erreur.

```
[6]: def computeCost(x,y,mu):
      L=np.dot(x,mu.T)
      z=np.power((L-y),2)
      return np.sum(z)/(2*len(x))

      #calcul d'erreur
```

la valeur d'erreur pour mu=[0,0] :

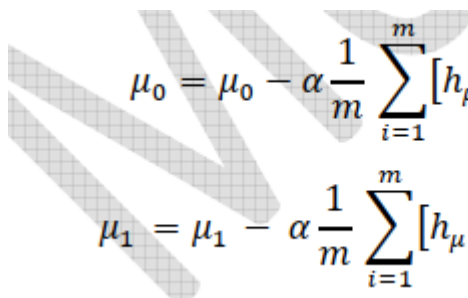
```
[31]: print('computeCost(x,y,mu)=', computeCost(x,y,mu))

computeCost(x,y,mu)= 32.072733877455676
```

## On calcule le gradient descent

```
[475]: def gradientDescent(n,x,y,mu):  
    er_vect=np.zeros(n)  
    alpha=0.01  
    temp=np.zeros(mu.shape)  
    mu_val=np.zeros(mu.shape)  
  
    for i in range(n):  
        er_vect[i]=computeCost(x,y,mu)  
        error=(np.dot(x,mu.T))-y  
  
        term=np.multiply(error,x[:,1])  
        temp[0,1]=mu[0,1]-(alpha/len(x))*np.sum(term)  
        temp[0,0]=mu[0,0]-(alpha/len(x))*np.sum(error)  
  
        mu=temp  
    return mu , er_vect
```

Le principe de cette algorithme est inspiré de ces équations


$$\mu_0 = \mu_0 - \alpha \frac{1}{m} \sum_{i=1}^m [h_{\mu}(X^i) - Y^i]$$
$$\mu_1 = \mu_1 - \alpha \frac{1}{m} \sum_{i=1}^m [h_{\mu}(X^i) - Y^i] X^i$$

Après exécution de la gradient descent on affiche la nouvelle mu, le vecteur d'erreur et l'erreur minimale correspondant à mu nouveau

```
:rée [485]: n=10000  
  
[mu1,cost1]=gradientDescent(n,x,y,mu)  
  
print("mu=", mu1)  
print("le vecteur de l'erreur,", cost1)  
print("la valeur minimale de l'erreur est", computeCost(x,y,mu1) )  
  
mu= [[-3.89578082  1.19303364]]  
le vecteur de l'erreur, [32.07273388  6.73719046  5.93159357 ...  4.47697138  4.47697138  
 4.47697138]  
la valeur minimale de l'erreur est 4.476971375975178
```

On remarque que l'erreur a diminué progressivement de 32.072 jusqu'à 4.47 , ce qui montre que la valeur du nouveau mu est meilleure que la valeur initiale

Si on calcule la meilleure mu qu'on peut obtenir directement on trouve

```
Entrée [464]: mu_best = np.linalg.inv(x.T.dot(x)).dot(x.T).dot(y)
              mu_best.T
```

```
Out[464]: matrix([[ -3.89578088,  1.19303364]])
```

On compare l'erreur en nouveau mu (mu1) et meilleure mu (mu\_best)

```
Entrée [494]: c1=computeCost(x,y,mu1)
              print(c1)
```

```
4.476971375975178
```

```
Entrée [495]: c_best=computeCost(x,y,mu_best)
              print(c2)
```

```
4.476971375975179
```

```
Entrée [496]: err=c_best-c1
              print(err)
              print(round(err))
```

```
8.881784197001252e-16
0.0
```

avec un taux de différence d'ordre  
1 exposant -17

on peut dire qu'on a obtenu un mu  
optimal pour 10000 itérations

Maintenant on souhaite tracer une ligne d'ajustement

on calcule min et max de la population

```
Entrée [48]: a=data.population.max()
              a
```

```
Out[48]: 22.203000000000003
```

```
Entrée [49]: b=data.population.min()
              b
```

```
Out[49]: 5.0269
```

On définit un vecteur X de 100 éléments dont les valeurs sont comprises entre la valeur minimale de la population et la valeur maximale.

```
Entrée [439]: X=np.linspace(a,b+1,num=100)
X
```

```
Out[439]: array([22.203      , 22.03960505, 21.8762101 , 21.71281515, 21.5494202 ,
                21.38602525, 21.2226303 , 21.05923535, 20.8958404 , 20.73244545,
                20.56905051, 20.40565556, 20.24226061, 20.07886566, 19.91547071,
                19.75207576, 19.58868081, 19.42528586, 19.26189091, 19.09849596,
                18.93510101, 18.77170606, 18.60831111, 18.44491616, 18.28152121,
                18.11812626, 17.95473131, 17.79133636, 17.62794141, 17.46454646,
                17.30115152, 17.13775657, 16.97436162, 16.81096667, 16.64757172,
                16.48417677, 16.32078182, 16.15738687, 15.99399192, 15.83059697,
                15.66720202, 15.50380707, 15.34041212, 15.17701717, 15.01362222,
                14.85022727, 14.68683232, 14.52343737, 14.36004242, 14.19664747,
                14.03325253, 13.86985758, 13.70646263, 13.54306768, 13.37967273,
                13.21627778, 13.05288283, 12.88948788, 12.72609293, 12.56269798,
                12.39930303, 12.23590808, 12.07251313, 11.90911818, 11.74572323,
                11.58232828, 11.41893333, 11.25553838, 11.09214343, 10.92874848,
                10.76535354, 10.60195859, 10.43856364, 10.27516869, 10.11177374,
                9.94837879, 9.78498384, 9.62158889, 9.45819394, 9.29479899,
```

```
Entrée [443]: f=mu1[0,0]+mu1[0,1]*X
```

```
Entrée [444]: f
```

```
Out[444]: array([21.78791103, 21.60371685, 21.41952267, 21.23532849, 21.05113432,
                20.86694014, 20.68274596, 20.49855178, 20.3143576 , 20.13016342,
                19.94596924, 19.76177506, 19.57758088, 19.3933867 , 19.20919252,
                19.02499834, 18.84080416, 18.65660998, 18.47241581, 18.28822163,
                18.10402745, 17.91983327, 17.73563909, 17.55144491, 17.36725073,
                17.18305655, 16.99886237, 16.81466819, 16.63047401, 16.44627983,
                16.26208565, 16.07789147, 15.8936973 , 15.70950312, 15.52530894,
                15.34111476, 15.15692058, 14.9727264 , 14.78853222, 14.60433804,
                14.42014386, 14.23594968, 14.0517555 , 13.86756132, 13.68336714,
                13.49917297, 13.31497879, 13.13078461, 12.94659043, 12.76239625,
                12.57820207, 12.39400789, 12.20981371, 12.02561953, 11.84142535,
                11.65723117, 11.47303699, 11.28884281, 11.10464863, 10.92045446,
                10.73626028, 10.5520661 , 10.36787192, 10.18367774, 9.99948356,
                9.81528938, 9.6310952 , 9.44690102, 9.26270684, 9.07851266,
                8.89431848, 8.7101243 , 8.52593012, 8.34173595, 8.15754177,
                7.97334759, 7.78915341, 7.60495923, 7.42076505, 7.23657087,
                7.05237669, 6.86818251, 6.68398833, 6.49979415, 6.31559997,
                6.13140579, 5.94721161, 5.76301744, 5.57882326, 5.39462908,
                5.2104349 , 5.02624072, 4.84204654, 4.65785236, 4.47365818,
                4.289464 , 4.10526982, 3.92107564, 3.73688146, 3.55268728])
```

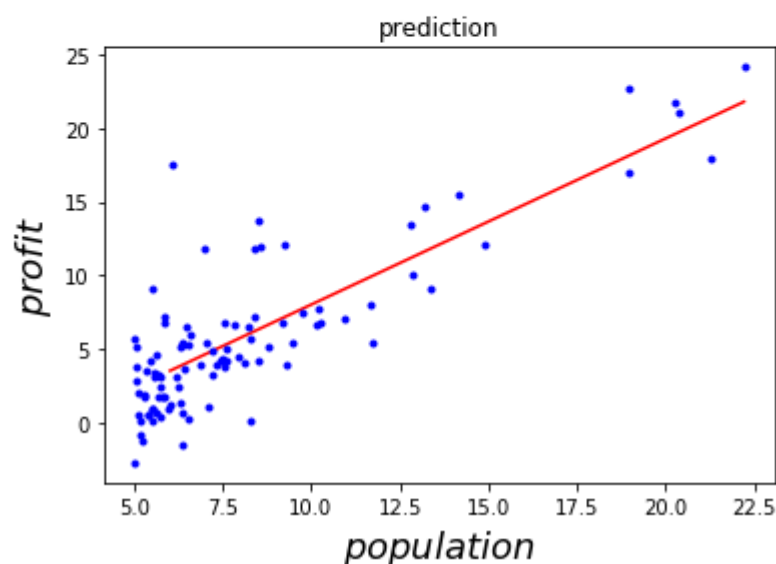
On écrit l'équation finale de la droite de régression

On trace la ligne de régression

```
In [473]: plt.plot(X,f,'r-')

plt.plot(data.population,data.profit,'b.')
plt.xlabel("$population$", rotation=0,fontsize=18)
plt.ylabel("$profit$", rotation=90, fontsize=18)
plt.title('prediction')

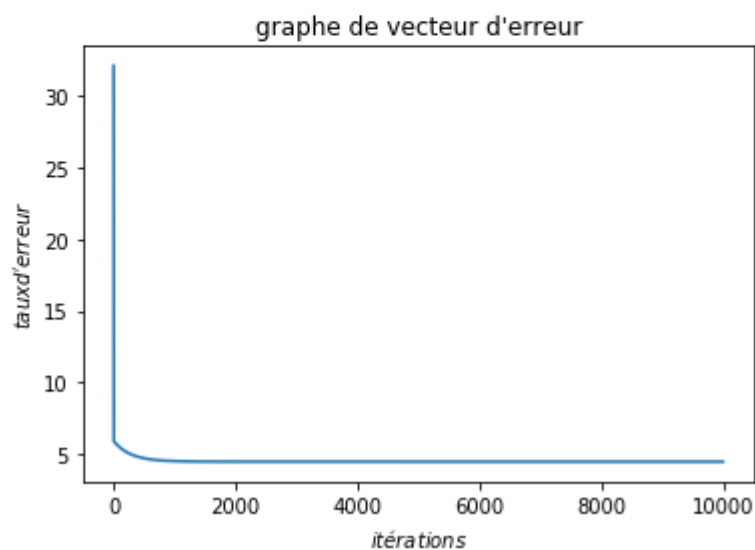
#plt.legend()
plt.show()
```



On trace le graphe du vecteur d'erreur :

```
In [497]: plt.plot(range(n),cost1)
plt.xlabel("$itérations$", rotation=0)
plt.ylabel("$taux d'erreur$", rotation=90)
plt.title("graphe de vecteur d'erreur")
```

Out[497]: Text(0.5, 1.0, "graphe de vecteur d'erreur")





### Objectif :

Développer l'algorithme du code source de la régression linéaire multiple

On va prendre l'exemple du prix d'un loyer en fonction de la surface et du nombre de chambre.

### Manipulation :

On commence par importer les bibliothèques nécessaires :

```
In [8]: from numpy import *  
        from numpy.random import *  
        from math import *  
        from matplotlib.pyplot import *  
        from scipy.misc import *  
        import pandas as pd  
        import numpy as np
```

On lit le fichier ex2data1 à partir de son path :

```
[3]: data = pd.read_csv('ex2data1.txt', header=None, names=['size', 'bedrooms', 'price'])
```

un aperçu sur les données du fichier lu.

on calcule les données statistiques de chacune des variables.

```
Entrée [6]: data.head()
```

Out[6]:

	size	bedrooms	price
0	2104	3	399900
1	1600	3	329900
2	2400	3	369000
3	1416	2	232000
4	3000	4	539900

```
Out [24]: data.describe()
```

Out[24]:

	size	bedrooms	price
count	47.000000	47.000000	47.000000
mean	2000.680851	3.170213	340412.659574
std	794.702354	0.760982	125039.899586
min	852.000000	1.000000	169900.000000
25%	1432.000000	3.000000	249900.000000
50%	1888.000000	3.000000	299900.000000
75%	2269.000000	4.000000	384450.000000
max	4478.000000	5.000000	699900.000000

### la normalisation des données :

```
Entrée [9]: data_norm=np.absolute(data-mean(data))/std(data)
```

on visualise les données normalisés :

```
In[10]: data_norm.head(10)
```

Out[10]:

	size	bedrooms	price
0	0.131415	0.226093	0.480890
1	0.509641	0.226093	0.084983
2	0.507909	0.226093	0.231097
3	0.743677	1.554392	0.876398
4	1.271071	1.102205	1.612637
5	0.019945	1.102205	0.327501
6	0.593589	0.226093	0.206242
7	0.729686	0.226093	1.143175
8	0.789467	0.226093	1.038076
9	0.644466	0.226093	0.791517

données statistiques de data\_norm

```
In[1]: data_norm.describe()
```

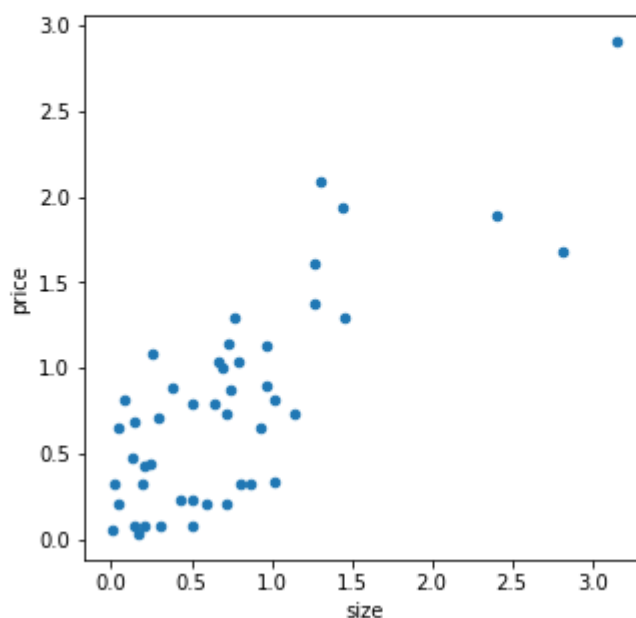
Out[1]:

	size	bedrooms	price
count	47.000000	47.000000	47.000000
mean	0.739008	0.760059	0.788536
std	0.680980	0.656880	0.621637
min	0.000866	0.226093	0.037084
25%	0.223455	0.226093	0.327501
50%	0.668173	0.226093	0.731696
75%	0.970718	1.102205	1.063740
max	3.150993	2.882690	2.906063

## représentation graphique de price en fonction de size

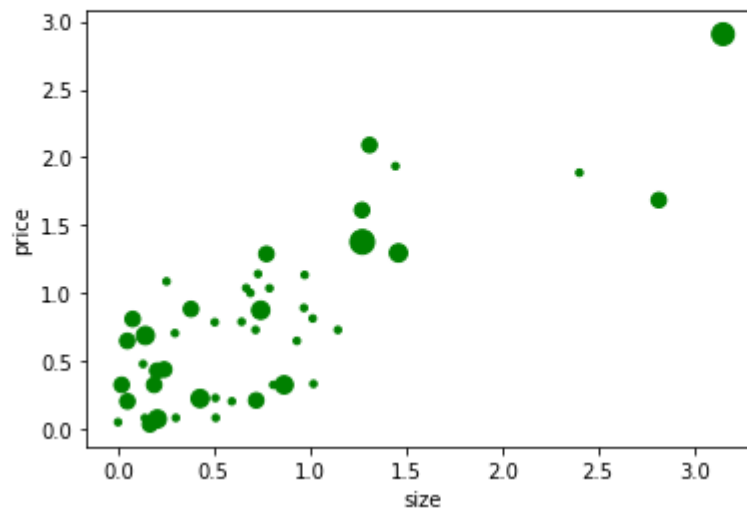
```
In[15]: import matplotlib.pyplot as plt  
data_norm.plot(kind='scatter',x='size',y='price',figsize=(5,5))
```

Out[15]: <matplotlib.axes.\_subplots.AxesSubplot at 0x2a818746188>



représentation graphique de **price** en fonction de **size** en distinguant les points (taille) selon **bedrooms** :

```
Intrée [88]: data_norm.plot.scatter(x='size',y='price',c="g",s=data_norm['bedrooms']*50)
Out[88]: <matplotlib.axes._subplots.AxesSubplot at 0x2a81a3d1a08>
```



ajout d'une ligne ones comme biais pour que x soit de taille 47x3

```
In [17]: #ajouter une ligne
data_norm.insert(0,'Ones',1)

new data =

*****
```

```
In [18]: data_norm.head()
```

```
Out[18]:
```

	Ones	size	bedrooms	price
0	1	0.131415	0.226093	0.480890
1	1	0.509641	0.226093	0.084983
2	1	0.507909	0.226093	0.231097
3	1	0.743677	1.554392	0.876398
4	1	1.271071	1.102205	1.612637

Séparation des entrées :

```
In [94]: #séparer les données d'entrée et les sorties
cols=data_norm.shape[1] #nbre de colonne
print(cols)
x=data_norm.iloc[:,0:cols-1]
y=data_norm.iloc[:,cols-1:cols]
```

```
[95]: x.head()
```

```
[95]:
```

	Ones	size	bedrooms
0	1	0.131415	0.226093
1	1	0.509641	0.226093
2	1	0.507909	0.226093
3	1	0.743677	1.554392
4	1	1.271071	1.102205

```
[96]: y.head()
```

```
[96]:
```

	price
0	0.480890
1	0.084983
2	0.231097
3	0.876398
4	1.612637

On convertit x et y en matrices et on définit mu

```
[97]: x=np.matrix(x)
```

```
[32]: y=np.matrix(y)
```

```
[33]: m=np.array([0,0,0])  
mu=np.matrix(m)
```

```
[99]: print('y.shape= ', y.shape)  
print('y= \n', y)
```

```
y.shape= (47, 1)
```

```
y=
```

	price
0	0.480890
1	0.084983
2	0.231097
3	0.876398
4	1.612637
5	0.327501
6	0.206242
7	1.143175
8	1.038076

```
[98]: print('x.shape= ', x.shape)  
print( x)
```

```
x.shape= (47, 3)
```

```
[1.00000000e+00 1.31415422e-01 2.26093368e-01]  
[1.00000000e+00 5.09640698e-01 2.26093368e-01]  
[1.00000000e+00 5.07908699e-01 2.26093368e-01]  
[1.00000000e+00 7.43677059e-01 1.55439190e+00]  
[1.00000000e+00 1.27107075e+00 1.10220517e+00]  
[1.00000000e+00 1.99450507e-02 1.10220517e+00]  
[1.00000000e+00 5.93588523e-01 2.26093368e-01]  
[1.00000000e+00 7.20605755e-01 2.26093368e-01]
```

```
[100]: print('mu.shape \n', mu.shape)  
print('mu= \n',mu)
```

```
mu.shape
```

```
(1, 3)
```

```
mu=
```

```
[[0.20117479 0.71039701 0.08206231]]
```

On définit la fonction de calcul d'erreur :

```
[38]: def computeCost(x,y,mu):  
    L=np.dot(x,mu.T)  
    z=np.power((L-y),2)  
    return np.sum(z)/(2*len(x))  
  
#calcul d'erreur
```

on calcule l'erreur lorsque mu=[0,0,0]

```
[10]: print('computeCost(x,y,mu)=', computeCost(x,y,mu))
```

```
computeCost(x,y,mu)= 0.5000000000000001
```

on définit la fonction du gradient descendant

```
rée [41]: def gradDescent(x,y,mu,iter):
    cost=np.zeros(iter)
    alpha=0.01
    param=mu.shape[1]
    temp=np.zeros(mu.shape)
    for i in range(iter):
        error=(np.dot(x,mu.T))-y
        for j in range(param):
            term=np.multiply(error,x[:,j])
            temp[0,j]=mu[0,j]-(alpha/len(x))*np.sum(term)
            mu=temp
            cost[i]=computeCost(x,y,mu)
    return mu, cost
```

on calcule nouveau mu , vecteur d'erreur et l'erreur à l'aide du gradient descendant

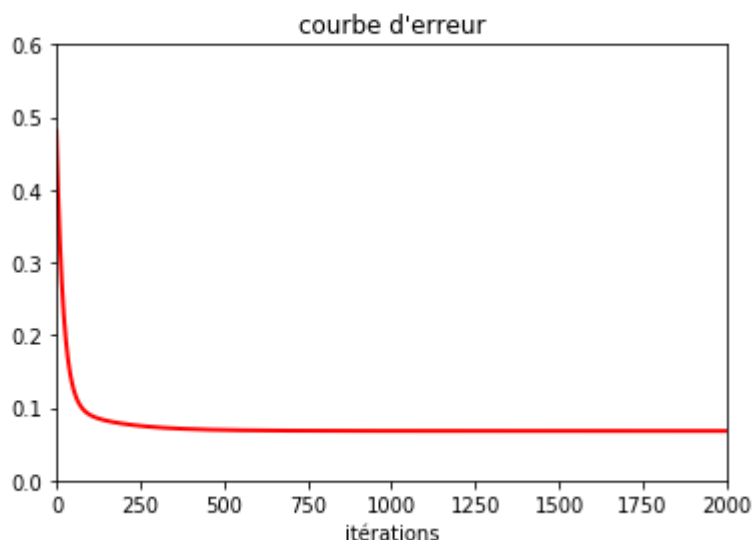
```
rée [42]: iter=10000
[mu,cost] =gradDescent(x,y,mu,iter)
print("mu=", mu)
print("le vecteur de l'erreur,", cost)
print("la valeur minimale de l'erreur est", computeCost(x,y,mu) )

mu= [[0.20117479 0.71039701 0.08206231]]
le vecteur de l'erreur, [0.48083899 0.46258817 0.44520386 ... 0.06802453 0.06802453 0.06802453]
la valeur minimale de l'erreur est 0.06802452954732971
```

**on obtient meilleure mu =[0.201 0.710 0.082] pour laquelle l'erreur vaut 0.06 : une amélioration de l'erreur**

on trace le courbe d'erreur en fonction des itérations

```
Entrée [51]: plt.plot(range(iter),cost,'r-',linewidth=2)
#_ =plt.axis([0,4000,0,0.5])
plt.xlim(0,2000)
plt.ylim(0,0.6)
plt.xlabel("itérations")
plt.title("courbe d'erreur")
plt.show()
```



## Corrélation des variables

```
In [85]: data_norm[['size', 'bedrooms', 'price']].corr()
```

```
Out[85]:
```

	size	bedrooms	price
size	1.000000	0.201143	0.795655
bedrooms	0.201143	1.000000	0.243247
price	0.795655	0.243247	1.000000

On remarque que size et price sont corrélés de l'ordre 0.79

Donc ils sont linéairement dépendants

Maintenant on souhaite tracer une ligne d'ajustement selon le size et le price

on calcule min et max de size

```
In [57]: a=data_norm['size'].min()
a
```

```
Out[57]: 0.0008659994861353915
```

```
In [58]: b=data_norm['size'].max()
b
```

```
Out[58]: 3.15099325527155
```

On définit un vecteur X de 100 éléments dont les valeurs sont comprises entre la valeur minimale de size et la valeur maximale.

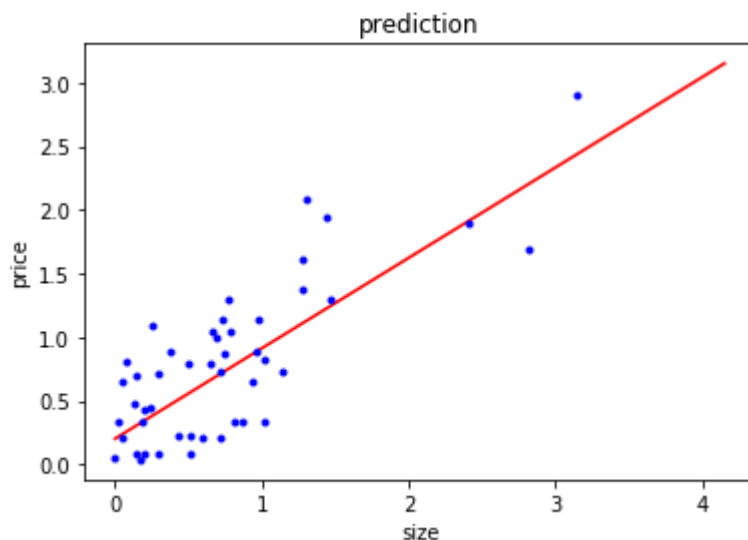
```
In [59]: X=np.linspace(a,b+1,100)
```

```
In [60]: X
```

```
Out[60]: array([8.65999486e-04, 4.27864768e-02, 8.47069541e-02, 1.26627431e-01,
1.68547909e-01, 2.10468386e-01, 2.52388863e-01, 2.94309341e-01,
3.36229818e-01, 3.78150295e-01, 4.20070773e-01, 4.61991250e-01,
5.03911727e-01, 5.45832205e-01, 5.87752682e-01, 6.29673159e-01,
6.71593637e-01, 7.13514114e-01, 7.55434591e-01, 7.97355069e-01,
8.39275546e-01, 8.81196023e-01, 9.23116501e-01, 9.65036978e-01,
1.00695746e+00, 1.04887793e+00, 1.09079841e+00, 1.13271889e+00,
1.17463936e+00, 1.21655984e+00, 1.25848032e+00, 1.30040080e+00,
1.34232127e+00, 1.38424175e+00, 1.42616223e+00, 1.46808271e+00,
1.50999319e+00, 1.55191367e+00, 1.59383415e+00, 1.63575463e+00,
1.67767511e+00, 1.71959559e+00, 1.76151607e+00, 1.80343655e+00,
1.84535703e+00, 1.88727751e+00, 1.92919799e+00, 1.97111847e+00,
2.01303895e+00, 2.05495943e+00, 2.09687991e+00, 2.13879939e+00,
2.18071987e+00, 2.22264035e+00, 2.26456083e+00, 2.30648131e+00,
2.34840179e+00, 2.39032227e+00, 2.43224275e+00, 2.47416323e+00,
2.51608371e+00, 2.55800419e+00, 2.59992467e+00, 2.64184515e+00,
2.68376563e+00, 2.72568611e+00, 2.76760659e+00, 2.80952707e+00,
2.85144755e+00, 2.89336803e+00, 2.93528851e+00, 2.97720899e+00,
3.01912947e+00, 3.06104995e+00, 3.10297043e+00, 3.14489091e+00,
3.18681139e+00, 3.22873187e+00, 3.27065235e+00, 3.31257283e+00,
3.35449331e+00, 3.39641379e+00, 3.43833427e+00, 3.48025475e+00,
3.52217523e+00, 3.56409571e+00, 3.60601619e+00, 3.64793667e+00,
3.68985715e+00, 3.73177763e+00, 3.77369811e+00, 3.81561859e+00,
3.85753907e+00, 3.89945955e+00, 3.94138003e+00, 3.98330051e+00,
4.02522099e+00, 4.06714147e+00, 4.10906195e+00, 4.15098243e+00,
4.19290291e+00, 4.23482339e+00, 4.27674387e+00, 4.31866435e+00,
4.36058483e+00, 4.40250531e+00, 4.44442579e+00, 4.48634627e+00,
4.52826675e+00, 4.57018723e+00, 4.61210771e+00, 4.65402819e+00,
4.69594867e+00, 4.73786915e+00, 4.77978963e+00, 4.82171011e+00,
4.86363059e+00, 4.90555107e+00, 4.94747155e+00, 4.98939203e+00,
5.03131251e+00, 5.07323299e+00, 5.11515347e+00, 5.15707395e+00,
5.19899443e+00, 5.24091491e+00, 5.28283539e+00, 5.32475587e+00,
5.36667635e+00, 5.40859683e+00, 5.45051731e+00, 5.49243779e+00,
5.53435827e+00, 5.57627875e+00, 5.61819923e+00, 5.66011971e+00,
5.70204019e+00, 5.74396067e+00, 5.78588115e+00, 5.82780163e+00,
5.86972211e+00, 5.91164259e+00, 5.95356307e+00, 5.99548355e+00,
6.03740403e+00, 6.07932451e+00, 6.12124499e+00, 6.16316547e+00,
6.20508595e+00, 6.24700643e+00, 6.28892691e+00, 6.33084739e+00,
6.37276787e+00, 6.41468835e+00, 6.45660883e+00, 6.49852931e+00,
6.54044979e+00, 6.58237027e+00, 6.62429075e+00, 6.66621123e+00,
6.70813171e+00, 6.75005219e+00, 6.79197267e+00, 6.83389315e+00,
6.87581363e+00, 6.91773411e+00, 6.95965459e+00, 7.00157507e+00,
7.04349555e+00, 7.08541603e+00, 7.12733651e+00, 7.16925699e+00,
7.21117747e+00, 7.25309795e+00, 7.29501843e+00, 7.33693891e+00,
7.37885939e+00, 7.42077987e+00, 7.46270035e+00, 7.50462083e+00,
7.54654131e+00, 7.58846179e+00, 7.63038227e+00, 7.67230275e+00,
7.71422323e+00, 7.75614371e+00, 7.79806419e+00, 7.83998467e+00,
7.88190515e+00, 7.92382563e+00, 7.96574611e+00, 8.00766659e+00,
8.04958707e+00, 8.09150755e+00, 8.13342803e+00, 8.17534851e+00,
8.21726899e+00, 8.25918947e+00, 8.30110995e+00, 8.34303043e+00,
8.38495091e+00, 8.42687139e+00, 8.46879187e+00, 8.51071235e+00,
8.55263283e+00, 8.59455331e+00, 8.63647379e+00, 8.67839427e+00,
8.72031475e+00, 8.76223523e+00, 8.80415571e+00, 8.84607619e+00,
8.88799667e+00, 8.92991715e+00, 8.97183763e+00, 9.01375811e+00,
9.05567859e+00, 9.09759907e+00, 9.13951955e+00, 9.18144003e+00,
9.22336051e+00, 9.26528099e+00, 9.30720147e+00, 9.34912195e+00,
9.39104243e+00, 9.43296291e+00, 9.47488339e+00, 9.51680387e+00,
9.55872435e+00, 9.60064483e+00, 9.64256531e+00, 9.68448579e+00,
9.72640627e+00, 9.76832675e+00, 9.81024723e+00, 9.85216771e+00,
9.89408819e+00, 9.93600867e+00, 9.97792915e+00, 1.00171343e+01,
1.00562771e+01, 1.00954199e+01, 1.01345627e+01, 1.01737055e+01,
1.02128483e+01, 1.02519911e+01, 1.02911339e+01, 1.03302767e+01,
1.03694195e+01, 1.04085623e+01, 1.04477051e+01, 1.04868479e+01,
1.05259907e+01, 1.05651335e+01, 1.06042763e+01, 1.06434191e+01,
1.06825619e+01, 1.07217047e+01, 1.07608475e+01, 1.08000003e+01,
1.08391431e+01, 1.08782859e+01, 1.09174287e+01, 1.09565715e+01,
1.09957143e+01, 1.10348571e+01, 1.10739999e+01, 1.11131427e+01,
1.11522855e+01, 1.11914283e+01, 1.12305711e+01, 1.12697139e+01,
1.13088567e+01, 1.13479995e+01, 1.13871423e+01, 1.14262851e+01,
1.14654279e+01, 1.15045707e+01, 1.15437135e+01, 1.15828563e+01,
1.16219991e+01, 1.16611419e+01, 1.17002847e+01, 1.17394275e+01,
1.17785703e+01, 1.18177131e+01, 1.18568559e+01, 1.18959987e+01,
1.19351415e+01, 1.19742843e+01, 1.20134271e+01, 1.20525699e+01,
1.20917127e+01, 1.21308555e+01, 1.21699983e+01, 1.22091411e+01,
1.22482839e+01, 1.22874267e+01, 1.23265695e+01, 1.23657123e+01,
1.24048551e+01, 1.24439979e+01, 1.24831407e+01, 1.25222835e+01,
1.25614263e+01, 1.26005691e+01, 1.26397119e+01, 1.26788547e+01,
1.27179975e+01, 1.27571403e+01, 1.27962831e+01, 1.28354259e+01,
1.28745687e+01, 1.29137115e+01, 1.29528543e+01, 1.29919971e+01,
1.30311399e+01, 1.30702827e+01, 1.31094255e+01, 1.31485683e+01,
1.31877111e+01, 1.32268539e+01, 1.32659967e+01, 1.33051395e+01,
1.33442823e+01, 1.33834251e+01, 1.34225679e+01, 1.34617107e+01,
1.35008535e+01, 1.35399963e+01, 1.35791391e+01, 1.36182819e+01,
1.36574247e+01, 1.36965675e+01, 1.37357103e+01, 1.37748531e+01,
1.38139959e+01, 1.38531387e+01, 1.38922815e+01, 1.39314243e+01,
1.39705671e+01, 1.40097099e+01, 1.40488527e+01, 1.40879955e+01,
1.41271383e+01, 1.41662811e+01, 1.42054239e+01, 1.42445667e+01,
1.42837095e+01, 1.43228523e+01, 1.43619951e+01, 1.44011379e+01,
1.44402807e+01, 1.44794235e+01, 1.45185663e+01, 1.45577091e+01,
1.45968519e+01, 1.46359947e+01, 1.46751375e+01, 1.47142803e+01,
1.47534231e+01, 1.47925659e+01, 1.48317087e+01, 1.48708515e+01,
1.49099943e+01, 1.49491371e+01, 1.49882799e+01, 1.50274227e+01,
1.50665655e+01, 1.51057083e+01, 1.51448511e+01, 1.51839939e+01,
1.52231367e+01, 1.52622795e+01, 1.53014223e+01, 1.53405651e+01,
1.53797079e+01, 1.54188507e+01, 1.54579935e+01, 1.54971363e+01,
1.55362791e+01, 1.55754219e+01, 1.56145647e+01, 1.56537075e+01,
1.56928503e+01, 1.57319931e+01, 1.57711359e+01, 1.58102787e+01,
1.58494215e+01, 1.58885643e+01, 1.59277071e+01, 1.59668499e+01,
1.60059927e+01, 1.60451355e+01, 1.60842783e+01, 1.61234211e+01,
1.61625639e+01, 1.62017067e+01, 1.62408495e+01, 1.62799923e+01,
1.63191351e+01, 1.63582779e+01, 1.63974207e+01, 1.64365635e+01,
1.64757063e+01, 1.65148491e+01, 1.65539919e+01, 1.65931347e+01,
1.66322775e+01, 1.66714203e+01, 1.67105631e+01, 1.67497059e+01,
1.67888487e+01, 1.68279915e+01, 1.68671343e+01, 1.69062771e+01,
1.69454199e+01, 1.69845627e+01, 1.70237055e+01, 1.70628483e+01,
1.71019911e+01, 1.71411339e+01, 1.71802767e+01, 1.72194195e+01,
1.72585623e+01, 1.72977051e+01, 1.73368479e+01, 1.73759907e+01,
1.74151335e+01, 1.74542763e+01, 1.74934191e+01, 1.75325619e+01,
1.75717047e+01, 1.76108475e+01, 1.76499903e+01, 1.76891331e+01,
1.77282759e+01, 1.77674187e+01, 1.78065615e+01, 1.78457043e+01,
1.78848471e+01, 1.79239899e+01, 1.79631327e+01, 1.80022755e+01,
1.80414183e+01, 1.80805611e+01, 1.81197039e+01, 1.81588467e+01,
1.81979895e+01, 1.82371323e+01, 1.82762751e+01, 1.83154179e+01,
1.83545607e+01, 1.83937035e+01, 1.84328463e+01, 1.84719891e+01,
1.85111319e+01, 1.85502747e+01, 1.85894175e+01, 1.86285603e+01,
1.86677031e+01, 1.87068459e+01, 1.87459887e+01, 1.87851315e+01,
1.88242743e+01, 1.88634171e+01, 1.89025599e+01, 1.89417027e+01,
1.89808455e+01, 1.90199883e+01, 1.90591311e+01, 1.90982739e+01,
1.91374167e+01, 1.91765595e+01, 1.92157023e+01, 1.92548451e+01,
1.92939879e+01, 1.93331307e+01, 1.93722735e+01, 1.94114163e+01,
1.94505591e+01, 1.94897019e+01, 1.95288447e+01, 1.95679875e+01,
1.96071303e+01, 1.96462731e+01, 1.96854159e+01, 1.97245587e+01,
1.97637015e+01, 1.98028443e+01, 1.98419871e+01, 1.98811299e+01,
1.99202727e+01, 1.99594155e+01, 1.99985583e+01, 2.00376951e+01,
2.00768379e+01, 2.01159807e+01, 2.01551235e+01, 2.01942663e+01,
2.02334091e+01, 2.02725519e+01, 2.03116947e+01, 2.03508375e+01,
2.03899803e+01, 2.04291231e+01, 2.04682659e+01, 2.05074087e+01,
2.05465515e+01, 2.05856943e+01, 2.06248371e+01, 2.06639799e+01,
2.07031227e+01, 2.07422655e+01, 2.07814083e+01, 2.08205511e+01,
2.08596939e+01, 2.08988367e+01, 2.09379795e+01, 2.09771223e+01,
2.10162651e+01, 2.10554079e+01, 2.10945507e+01, 2.11336935e+01,
2.11728363e+01, 2.12119791e+01, 2.12511219e+01, 2.12902647e+01,
2.13294075e+01, 2.13685503e+01, 2.14076931e+01, 2.14468359e+01,
2.14859787e+01, 2.15251215e+01, 2.15642643e+01, 2.16034071e+01,
2.16425499e+01, 2.16816927e+01, 2.17208355e+01, 2.17599783e+01,
2.17991211e+01, 2.18382639e+01, 2.18774067e+01, 2.19165495e+01,
2.19556923e+01, 2.19948351e+01, 2.20339779e+01, 2.20731207e+01,
2.21122635e+01, 2.21514063e+01, 2.21905491e+01, 2.22296919e+01,
2.22688347e+01, 2.23079775e+01, 2.23471203e+01, 2.23862631e+01,
2.24254059e+01, 2.24645487e+01, 2.25036915e+01, 2.25428343e+01,
2.25819771e+01, 2.26211199e+01, 2.26602627e+01, 2.26994055e+01,
2.27385483e+01, 2.27776911e+01, 2.28168339e+01, 2.28559767e+01,
2.28951195e+01, 2.29342623e+01, 2.29734051e+01, 2.30125479e+01,
2.30516907e+01, 2.30908335e+01, 2.31299763e+01, 2.31691191e+01,
2.32082619e+01, 2.32474047e+01, 2.32865475e+01, 2.33256903e+01,
2.33648331e+01, 2.34039759e+01, 2.34431187e+01, 2.34822615e+01,
2.35214043e+01, 2.35605471e+01, 2.35996899e+01, 2.36388327e+01,
2.36779755e+01, 2.37171183e+01, 2.37562611e+01, 2.37954039e+01,
2.38345467e+01, 2.38736895e+01, 2.39128323e+01, 2.39519751e+01,
2.39911179e+01, 2.40302607e+01, 2.40694035e+01, 2.41085463e+01,
2.41476891e+01, 2.41868319e+01, 2.42259747e+01, 2.42651175e+01,
2.43042603e+01, 2.43434031e+01, 2.43825459e+01, 2.44216887e+01,
2.44608315e+01, 2.45000000e+01, 2.45391428e+01, 2.45782856e+01,
2.46174284e+01, 2.46565712e+01, 2.46957140e+01, 2.47348568e+01,
2.47739996e+01, 2.48131424e+01, 2.48522852e+01, 2.48914280e+01,
2.49305708e+01, 2.49697136e+01, 2.50088564e+01, 2.50479992e+01,
2.50871420e+01, 2.51262848e+01, 2.51654276e+01, 2.52045704e+01,
2.52437132e+01, 2.52828560e+01, 2.53219988e+01, 2.53611416e+01,
2.54002844e+01, 2.54394272e+01, 2.54785700e+01, 2.55177128e+01,
2.55568556e+01, 2.55959984e+01, 2.56351412e+01, 2.56742840e+01,
2.57134268e+01, 2.57525696e+01, 2.57917124e+01, 2.58308552e+01,
2.58699980e+01, 2.59091408e+01, 2.59482836e+01, 2.59874264e+01,
2.60265692e+01, 2.60657120e+01, 2.61048548e+01, 2.61439976e+01,
2.61831404e+01, 2.62222832e+01, 2.62614260e+01, 2.63005688e+01,
2.63397116e+01, 2.63788544e+01, 2.64179972e+01, 2.64571400e+01,
2.64962828e+01, 2.65354256e+01, 2.65745684e+01, 2.66137112e+01,
2.66528540e+01, 2.66919968e+01, 2.67311396e+01, 2.67702824e+01,
2.68094252e+01, 2.68485680e+01, 2.68877108e+01, 2.69268536e+01,
2.69659964e+01, 2.70051392e+01, 2.70442820e+01, 2.70834248e+01,
2.71225676e+01, 2.71617104e+01, 2.72008532e+01, 2.72399960e+01,
2.72791388e+01, 2.73182816e+01, 2.73574244e+01, 2.73965672e+01,
2.74357100e+01, 2.74748528e+01, 2.75139956e+01, 2.75531384e+01,
2.75922812e+01, 2.76314240e+01, 2.76705668e+01, 2.77097096e+01,
2.77488524e+01, 2.77879952e+01, 2.78271380e+01, 2.78662808e+01,
2.79054236e+01, 2.79445664e+01, 2.79837092e+01, 2.80228520e+01,
2.80619948e+01, 2.81011376e+01, 2.81402804e+01, 2.81794232e+01,
2.82185660e+01, 2.82577088e+01, 2.82968516e+01, 2.83359944e+01,
2.83751372e+01, 2.84142800e+01, 2.84534228e+01, 2.84925656e+01,
2.85317084e+01, 2.85708512e+01, 2.86099940e+01, 2.86491368e+01,
2.86882796e+01, 2.87274224e+01, 
```

On trace le graphe des prédictions qui contient la droite de régression

```
[76]: plt.figure()
plt.plot(X,f,'r')
plt.plot(data_norm['size'],data_norm['price'],'b.')
plt.xlabel('size')
plt.ylabel('price')
plt.title('prediction')
plt.show()
```



**Prédiction selon bedrooms**

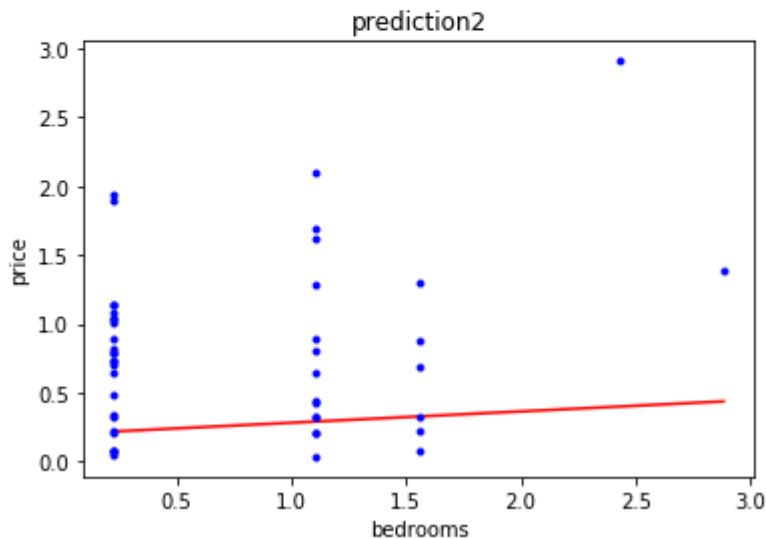
```
[77]: Y=np.linspace(data_norm['bedrooms'].min(),data_norm['bedrooms'].max(),100)
Y
```

```
[77]: array([0.22609337, 0.25292768, 0.279762 , 0.30659631, 0.33343062,
0.36026494, 0.38709925, 0.41393356, 0.44076788, 0.46760219,
0.49443651, 0.52127082, 0.54810513, 0.57493945, 0.60177376,
0.62860808, 0.65544239, 0.6822767 , 0.70911102, 0.73594533,
0.76277964, 0.78961396, 0.81644827, 0.84328259, 0.8701169 ,
0.89695121, 0.92378553, 0.95061984, 0.97745415, 1.00428847,
1.03112278, 1.0579571 , 1.08479141, 1.11162572, 1.13846004,
1.16529435, 1.19212867, 1.21896298, 1.24579729, 1.27263161,
1.29946592, 1.32630023, 1.35313455, 1.37996886, 1.40680318,
1.43362749, 1.46045181, 1.48727613, 1.51410045, 1.54092477,
1.56774909, 1.59457341, 1.62139773, 1.64822205, 1.67504637,
1.70187069, 1.72869501, 1.75551933, 1.78234365, 1.80916797,
1.83599229, 1.86281661, 1.88964093, 1.91646525, 1.94328957,
1.97011389, 1.99693821, 2.02376253, 2.05058685, 2.07741117,
2.10423549, 2.13105981, 2.15788413, 2.18470845, 2.21153277,
2.23835709, 2.26518141, 2.29200573, 2.31883005, 2.34565437,
2.37247869, 2.39930301, 2.42612733, 2.45295165, 2.47977597,
2.50659929, 2.53342361, 2.56024793, 2.58707225, 2.61389657,
2.64072089, 2.66754521, 2.69436953, 2.72119385, 2.74801817,
2.77484249, 2.80166681, 2.82849113, 2.85531545, 2.88213977,
2.90896409, 2.93578841, 2.96261273, 2.98943705, 3.01626137,
3.04308569, 3.06991001, 3.09673433, 3.12355865, 3.15038297,
3.17720729, 3.20403161, 3.23085593, 3.25768025, 3.28450457,
3.31132889, 3.33815321, 3.36497753, 3.39180185, 3.41862617,
3.44545049, 3.47227481, 3.49909913, 3.52592345, 3.55274777,
3.57957209, 3.60639641, 3.63322073, 3.66004505, 3.68686937,
3.71369369, 3.74051801, 3.76734233, 3.79416665, 3.82099097,
3.84781529, 3.87463961, 3.90146393, 3.92828825, 3.95511257,
3.98193689, 4.00876121, 4.03558553, 4.06240985, 4.08923417,
4.11605849, 4.14288281, 4.16970713, 4.19653145, 4.22335577,
4.25018009, 4.27700441, 4.30382873, 4.33065305, 4.35747737,
4.38430169, 4.41112601, 4.43795033, 4.46477465, 4.49159897,
4.51842329, 4.54524761, 4.57207193, 4.59889625, 4.62572057,
4.65254489, 4.67936921, 4.70619353, 4.73301785, 4.75984217,
4.78666649, 4.81349081, 4.84031513, 4.86713945, 4.89396377,
4.92078809, 4.94761241, 4.97443673, 4.99997537, 5.02591401,
5.05185265, 5.07779129, 5.10372993, 5.12966857, 5.15560721,
5.18154585, 5.20748449, 5.23342313, 5.25936177, 5.28530041,
5.31123905, 5.33717769, 5.36311633, 5.38905497, 5.41499361,
5.44093225, 5.46687089, 5.49280953, 5.51874817, 5.54468681,
5.57062545, 5.59656409, 5.62250273, 5.64844137, 5.67438001,
5.70031865, 5.72625729, 5.75219593, 5.77813457, 5.80407321,
5.83001185, 5.85595049, 5.88188913, 5.90782777, 5.93376641,
5.95970505, 5.98564369, 6.01158233, 6.03752097, 6.06345961,
6.08939825, 6.11533689, 6.14127553, 6.16721417, 6.19315281,
6.21909145, 6.24503009, 6.27096873, 6.29690737, 6.32284601,
6.34878465, 6.37472329, 6.40066193, 6.42660057, 6.45253921,
6.47847785, 6.50441649, 6.53035513, 6.55629377, 6.58223241,
6.60817105, 6.63410969, 6.66004833, 6.68598697, 6.71192561,
6.73786425, 6.76380289, 6.78974153, 6.81568017, 6.84161881,
6.86755745, 6.89349609, 6.91943473, 6.94537337, 6.97131201,
6.99725065, 7.02318929, 7.04912793, 7.07506657, 7.10100521,
7.12694385, 7.15288249, 7.17882113, 7.20475977, 7.23069841,
7.25663705, 7.28257569, 7.30851433, 7.33445297, 7.36039161,
7.38633025, 7.41226889, 7.43820753, 7.46414617, 7.49008481,
7.51602345, 7.54196209, 7.56790073, 7.59383937, 7.61977801,
7.64571665, 7.67165529, 7.69759393, 7.72353257, 7.74947121,
7.77540985, 7.80134849, 7.82728713, 7.85322577, 7.87916441,
7.90510305, 7.93104169, 7.95698033, 7.98291897, 8.00885761,
8.03479625, 8.06073489, 8.08667353, 8.11261217, 8.13855081,
8.16448945, 8.19042809, 8.21636673, 8.24230537, 8.26824401,
8.29418265, 8.32012129, 8.34605993, 8.37199857, 8.39793721,
8.42387585, 8.44981449, 8.47575313, 8.50169177, 8.52763041,
8.55356905, 8.57950769, 8.60544633, 8.63138497, 8.65732361,
8.68326225, 8.70920089, 8.73513953, 8.76107817, 8.78701681,
8.81295545, 8.83889409, 8.86483273, 8.89077137, 8.91671001,
8.94264865, 8.96858729, 8.99452593, 9.02046457, 9.04640321,
9.07234185, 9.09828049, 9.12421913, 9.15015777, 9.17609641,
9.20203505, 9.22797369, 9.25391233, 9.27985097, 9.30578961,
9.33172825, 9.35766689, 9.38360553, 9.40954417, 9.43548281,
9.46142145, 9.48736009, 9.51329873, 9.53923737, 9.56517601,
9.59111465, 9.61705329, 9.64299193, 9.66893057, 9.69486921,
9.72080785, 9.74674649, 9.77268513, 9.79862377, 9.82456241,
9.85050105, 9.87643969, 9.90237833, 9.92831697, 9.95425561,
9.98019425, 10.00613289, 10.03207153, 10.05801017, 10.08394881,
10.10988745, 10.13582609, 10.16176473, 10.18770337, 10.21364201,
10.23958065, 10.26551929, 10.29145793, 10.31739657, 10.34333521,
10.36927385, 10.39521249, 10.42115113, 10.44708977, 10.47302841,
10.49896705, 10.52490569, 10.55084433, 10.57678297, 10.60272161,
10.62866025, 10.65459889, 10.68053753, 10.70647617, 10.73241481,
10.75835345, 10.78429209, 10.81023073, 10.83616937, 10.86210801,
10.88804665, 10.91398529, 10.93992393, 10.96586257, 10.99180121,
11.01773985, 11.04367849, 11.06961713, 11.09555577, 11.12149441,
11.14743305, 11.17337169, 11.19931033, 11.22524897, 11.25118761,
11.27712625, 11.30306489, 11.32900353, 11.35494217, 11.38088081,
11.40681945, 11.43275809, 11.45869673, 11.48463537, 11.51057401,
11.53651265, 11.56245129, 11.58838993, 11.61432857, 11.64026721,
11.66620585, 11.69214449, 11.71808313, 11.74402177, 11.76996041,
11.79589905, 11.82183769, 11.84777633, 11.87371497, 11.89965361,
11.92559225, 11.95153089, 11.97746953, 12.00340817, 12.02934681,
12.05528545, 12.08122409, 12.10716273, 12.13310137, 12.15904001,
12.18497865, 12.21091729, 12.23685593, 12.26279457, 12.28873321,
12.31467185, 12.34061049, 12.36654913, 12.39248777, 12.41842641,
12.44436505, 12.47030369, 12.49624233, 12.52218097, 12.54811961,
12.57405825, 12.6000000, 12.62594175, 12.6518835, 12.67782525,
12.703767, 12.72970875, 12.7556505, 12.78159225, 12.807534,
12.83347575, 12.8594175, 12.88535925, 12.911301, 12.93724275,
12.9631845, 12.98912625, 13.015068, 13.04100975, 13.0669515,
13.09289325, 13.118835, 13.14477675, 13.1707185, 13.19666025,
13.222602, 13.24854375, 13.2744855, 13.30042725, 13.326369,
13.35231075, 13.3782525, 13.40419425, 13.430136, 13.45607775,
13.4820195, 13.50796125, 13.533903, 13.55984475, 13.5857865,
13.61172825, 13.63767, 13.66361175, 13.6895535, 13.71549525,
13.741437, 13.76737875, 13.7933205, 13.81926225, 13.845204,
13.87114575, 13.8970875, 13.92302925, 13.948971, 13.97491275,
14.0008545, 14.02679625, 14.052738, 14.07867975, 14.1046215,
14.13056325, 14.156505, 14.18244675, 14.2083885, 14.23433025,
14.260272, 14.28621375, 14.3121555, 14.33809725, 14.364039,
14.38998075, 14.4159225, 14.44186425, 14.467806, 14.49374775,
14.5196895, 14.54563125, 14.571573, 14.59751475, 14.6234565,
14.64939825, 14.67534, 14.70128175, 14.7272235, 14.75316525,
14.779107, 14.80504875, 14.8309905, 14.85693225, 14.882874,
14.90881575, 14.9347575, 14.96069925, 14.986641, 15.01258275,
15.0385245, 15.06446625, 15.090408, 15.11634975, 15.1422915,
15.16823325, 15.194175, 15.22011675, 15.2460585, 15.272,
15.29794175, 15.3238835, 15.34982525, 15.375767, 15.40170875,
15.4276505, 15.45359225, 15.479534, 15.50547575, 15.5314175,
15.55735925, 15.583301, 15.60924275, 15.6351845, 15.66112625,
15.687068, 15.71300975, 15.7389515, 15.76489325, 15.790835,
15.81677675, 15.8427185, 15.86866025, 15.894602, 15.92054375,
15.9464855, 15.97242725, 15.998369, 16.02431075, 16.0502525,
16.07619425, 16.102136, 16.12807775, 16.1540195, 16.17996125,
16.205903, 16.23184475, 16.2577865, 16.28372825, 16.30967,
16.33561175, 16.3615535, 16.38749525, 16.413437, 16.43937875,
16.4653205, 16.49126225, 16.517204, 16.54314575, 16.5690875,
16.59502925, 16.620971, 16.64691275, 16.6728545, 16.69879625,
16.724738, 16.75067975, 16.7766215, 16.80256325, 16.828505,
16.85444675, 16.8803885, 16.90633025, 16.932272, 16.95821375,
16.9841555, 17.01009725, 17.036039, 17.06198075, 17.0879225,
17.11386425, 17.139806, 17.16574775, 17.1916895, 17.21763125,
17.243573, 17.26951475, 17.2954565, 17.32139825, 17.34734,
17.37328175, 17.3992235, 17.42516525, 17.451107, 17.47704875,
17.5029905, 17.52893225, 17.554874, 17.58081575, 17.6067575,
17.63269925, 17.658641, 17.68458275, 17.7105245, 17.73646625,
17.762408, 17.78834975, 17.8142915, 17.84023325, 17.866175,
17.89211675, 17.9180585, 17.94399925, 17.969941, 17.99588275,
18.0218245, 18.04776625, 18.073708, 18.09964975, 18.1255915,
18.15153325, 18.177475, 18.20341675, 18.2293585, 18.2553,
18.28124175, 18.3071835, 18.33312525, 18.359067, 18.38500875,
18.4109505, 18.43689225, 18.462834, 18.48877575, 18.5147175,
18.54065925, 18.566601, 18.59254275, 18.6184845, 18.64442625,
18.670368, 18.69630975, 18.7222515, 18.74819325, 18.774135,
18.80007675, 18.8260185, 18.85196025, 18.877902, 18.90384375,
18.9297855, 18.95572725, 18.981669, 19.00761075, 19.0335525,
19.05949425, 19.085436, 19.11137775, 19.1373195, 19.16326125,
19.189203, 19.21514475, 19.2410865, 19.26702825, 19.29297,
19.31891175, 19.3448535, 19.37079525, 19.396737, 19.42267875,
19.4486205, 19.47456225, 19.500504, 19.52644575, 19.5523875,
19.57832925, 19.604271, 19.63021275, 19.6561545, 19.68209625,
19.708038, 19.73397975, 19.7599215, 19.78586325, 19.811805,
19.83774675, 19.8636885, 19.88963025, 19.915572, 19.94151375,
19.9674555, 19.99339725, 20.019339, 20.04528075, 20.0712225,
20.09716425, 20.123106, 20.14904775, 20.1749895, 20.20093125,
20.226873, 20.25281475, 20.2787565, 20.30469825, 20.33064,
20.35658175, 20.3825235, 20.40846525, 20.434407, 20.46034875,
20.4862905, 20.51223225, 20.538174, 20.56411575, 20.5900575,
20.61599925, 20.641941, 20.66788275, 20.6938245, 20.71976625,
20.745708, 20.77164975, 20.7975915, 20.82353325, 20.849475,
20.87541675, 20.9013585, 20.92730025, 20.953242, 20.97918375,
21.0051255, 21.03106725, 21.057009, 21.08295075, 21.1088925,
21.13483425, 21.160776, 21.18671775, 21.2126595, 21.23860125,
21.264543, 21.29048475, 21.3164265, 21.34236825, 21.36831,
21.39425175, 21.4201935, 21.44613525, 21.472077, 21.49801875,
21.5239605, 21.54990225, 21.575844, 21.60178575, 21.6277275,
21.65366925, 21.679611, 21.70555275, 21.7314945, 21.75743625,
21.783378, 21.80931975, 21.8352615, 21.86120325, 21.887145,
21.91308675, 21.9390285, 21.96497025, 21.990912, 22.01685375,
22.0427955, 22.06873725, 22.094679, 22.12062075, 22.1465625,
22.17250425, 22.198446, 22.22438775, 22.2503295, 22.27627125,
22.302213, 22.32815475, 22.3540965, 22.38003825, 22.40598,
22.43192175, 22.4578635, 22.48380525, 22.509747, 22.53568875,
22.5616305, 22.58757225, 22.613514, 22.63945575, 22.6653975,
22.69133925, 22.717281, 22.74322275, 22.7691645, 22.79510625,
22.821048, 22.84698975, 22.8729315, 22.89887325, 22.924815,
22.95075675, 22.9766985, 23.00264025, 23.028582, 23.05452375,
23.0804655, 23.10640725, 23.132349, 23.15829075, 23.1842325,
23.21017425, 23.236116, 23.26205775, 23.2879995, 23.31394125,
23.339883, 23.36582475, 23.3917665, 23.41770825, 23.44365,
23.46959175, 23.4955335, 23.52147525, 23.547417, 23.57335875,
23.5993005, 23.62524225, 23.651184, 23.67712575, 23.7030675,
23.72900925, 23.754951, 23.78089275, 23.8068345, 23.83277625,
23.858718, 23.88465975, 23.9106015, 23.93654325, 23.962485,
23.98842675, 24.0143685, 24.04031025, 24.066252, 24.09219375,
24.1181355, 24.14407725, 24.170019, 24.19596075, 24.2219025,
24.24784425, 24.273786, 24.29972775, 24.3256695, 24.35161125,
24.377553, 24.40349475, 24.4294365, 24.45537825, 24.48132,
24.50726175, 24.5332035, 24.55914525, 24.585087, 24.61102875,
24.6369705, 24.66291225, 24.688854, 24.71479575, 24.7407375,
24.76667925, 24.792621, 24.81856275, 24.8445045, 24.87044625,
24.896388, 24.92232975, 24.9482715, 24.97421325, 24.99998,
25.02592175, 25.0518635, 25.07780525, 25.103747, 25.12968875,
25.1556305, 25.18157225, 25.207514, 25.23345575, 25.2593975,
25.28533925, 25.311281, 25.33722275, 25.3631645, 25.38910625,
25.415048, 25.44098975, 25.4669315, 25.49287325, 25.518815,
25.54475675, 25.5706985, 25.59664025, 25.622582, 25.64852375,
25.6744655, 25.70040725, 25.726349, 25.75229075, 25.7782325,
25.80417425, 25.830116, 25.85605775, 2
```



On trace le graphe de prédiction selon bedrooms et la droite de regression

```
Entrée [79]: plt.figure()
plt.plot(Y,g,'r')
plt.plot(data_norm['bedrooms'],data_norm['price'],'b.')
plt.xlabel('bedrooms')
plt.ylabel('price')
plt.title('prediction2')
plt.show()
```



Prédiction selon bedrooms et size

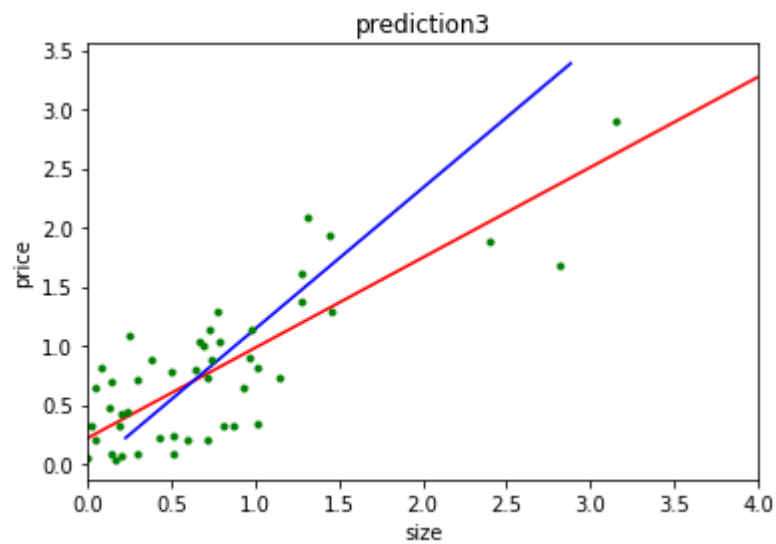
```
Entrée [80]: f_total=mu[0,0]+mu[0,1]*X+mu[0,2]*Y
```

```
Entrée [81]: f_total
```

```
Out[81]: array([0.22034373, 0.252326 , 0.28430827, 0.31629054, 0.3482728 ,
0.38025507, 0.41223734, 0.4442196 , 0.47620187, 0.50818414,
0.54016641, 0.57214867, 0.60413094, 0.63611321, 0.66809548,
0.70007774, 0.73206001, 0.76404228, 0.79602455, 0.82800681,
0.85998908, 0.89197135, 0.92395362, 0.95593588, 0.98791815,
1.01990042, 1.05188269, 1.08386495, 1.11584722, 1.14782949,
1.17981176, 1.21179402, 1.24377629, 1.27575856, 1.30774083,
1.33972309, 1.37170536, 1.40368763, 1.4356699 , 1.46765216,
1.49963443, 1.5316167 , 1.56359896, 1.59558123, 1.6275635 ,
.....])
```

on trace le graphe de prédiction avec droite de régression total

```
ée [142]: plt.figure()
plt.plot(X,f_total,'r',label='size')
plt.plot(Y,f_total,'b',label='bedrooms')
#plt.plot(X+Y,f_total,'c')
plt.plot(data_norm['size'],data_norm['price'],'g.')
plt.xlabel('size')
plt.ylabel('price')
plt.xlim(0,4)
plt.title('prediction3')
plt.show()
```



## Objectif :

Comme on a essayé dans les TP précédents de créer un modèle de prédiction et de classification linéaire, dans ce TP on va créer un autre modèle prédiction nommé : **Régression Logistique** qui est dédié pour les échantillons qui **ne sont pas linéairement dépendants**

Ceci en se basant sur la programmation en python.

## Manipulation :

On va étudier la relation existante entre les trois variables d'une base de données médicale contenant les valeurs : test1, test2 qui contient des valeurs numériques et résultat (0 ou 1) si test est positif ou négatif pour plusieurs échantillons.

On commence par **importer les bibliothèques nécessaires** pour le déroulement du travail

```
In [10]: import numpy as np
         from numpy.random import *
         from math import *
         import pandas as pd
         import matplotlib.pyplot as plt
```

### ❖ Visualisation de la base de données

Lecture le fichier de la base de données à partir de son path :

```
In [2]: data = pd.read_csv('ex3data1.txt', header= None, names=['test1', 'test2', 'resultat'])
```

Affichage et aperçu sur les dix premières données du fichier lu en utilisant la commande `__.head()`

```
In [3]: print('data='), data.head(10)
```

data=

```
Out[3]: (None,
         test1    test2  resultat
0  34.623660  78.024693         0
1  30.286711  43.894998         0
2  35.847409  72.902198         0
3  60.182599  86.308552         1
4  79.032736  75.344376         1
5  45.083277  56.316372         0
6  61.106665  96.511426         1
7  75.024746  46.554014         1
8  76.098787  87.420570         1
9  84.432820  43.533393         1)
```

Les données numériques de la colonne résultat sont de type binaire

0 pour un résultat négatif

1 pour un résultat positif

```
In [5]: print('data.describe'), data.describe()
```

data.describe

```
Out[5]: (None,
         test1    test2  resultat
count  100.000000  100.000000  100.000000
mean    65.644274   66.221998    0.600000
std     19.458222   18.582783    0.492366
min     30.058822   30.603263    0.000000
25%     50.919511   48.179205    0.000000
50%     67.032988   67.682381    1.000000
75%     80.212529   79.360605    1.000000
max     99.827858   98.869436    1.000000)
```

Les données statistiques de chaque variable

Visualisés grâce à la commande `__.describe()`

## ❖ Séparation des résultat positifs et des résultats négatifs :

Pour séparer les résultat positifs et négatifs dans test1 et test2 on va appliquer la méthode suivante :

- Compter les 1 et 0 dans la colonne résultat en utilisant `value_counts()`

```
Entrée [3]: data.resultat.value_counts()
```

```
Out[3]: 1    60
        0    40
        Name: resultat, dtype: int64
```

Dans le cas où data est ordonné  
On va avoir de 60 premiers lignes les variables positives  
Et le reste des 40 lignes des variables négatifs

- Ordonner data selon les valeurs de résultat comme ça on va avoir les valeurs positive en haut et les négatifs en bas . créer une nouvelle base à partir de data ordonnée

```
Entrée [16]: data_sort= data.sort_values(by='resultat', ascending=False)
```

```
Entrée [17]: data_sort.head(100)
```

```
Out[17]:
```

	test1	test2	resultat
50	79.944818	74.163119	1
59	71.796462	78.453562	1
73	60.457886	73.094998	1
72	72.346494	96.227593	1
71	64.039320	78.031688	1
...	...	...	...
65	66.560894	41.092098	0
27	93.114389	38.800670	0
67	49.072563	51.883212	0
41	51.547720	46.856290	0
0	34.623660	78.024693	0

100 rows × 3 columns

- Extraire les valeurs positifs et négatifs dans test1 et test2 :

```
Entrée [6]: pos=data_sort.iloc[0:60,0:2]
```

```
Entrée [7]: pos.head(60)
```

```
Out[7]:
```

	test1	test2
50	79.944818	74.163119
59	71.796462	78.453562
73	60.457886	73.094998

```
Entrée [8]: neg=data_sort.iloc[60:100,0:2]
```

```
Entrée [13]: neg.head(40)
```

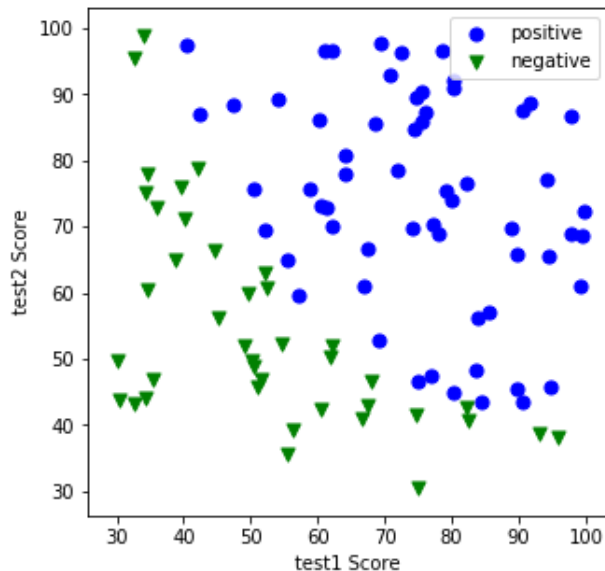
```
Out[13]:
```

	test1	test2
78	60.455556	42.508409
79	82.226662	42.719879
17	67.946855	46.678574
...	...	...

On trace le **graphe** qui illustre les cas **positifs et négatifs** :

```
Entrée [12]: fig, ax = plt.subplots(figsize=(5,5))
ax.scatter(pos['test1'], pos['test2'], s=50, c='b', marker='o', label='positive')
ax.scatter(neg['test1'], neg['test2'], s=50, c='g', marker='v', label='negative')
ax.legend()
ax.set_xlabel('test1 Score')
ax.set_ylabel('test2 Score')
```

Out[12]: Text(0, 0.5, 'test2 Score')



Ou bien on peut les séparer de cette manière :

```
rée [24]: positive= data[data['resultat'].isin([1])]
```

```
rée [25]: negative= data[data['resultat'].isin([0])]
```

```
Entrée [6]: def sigmoid(z):
#z=np.matrix(z)
return 1 / (1 + np.exp(-z))
```

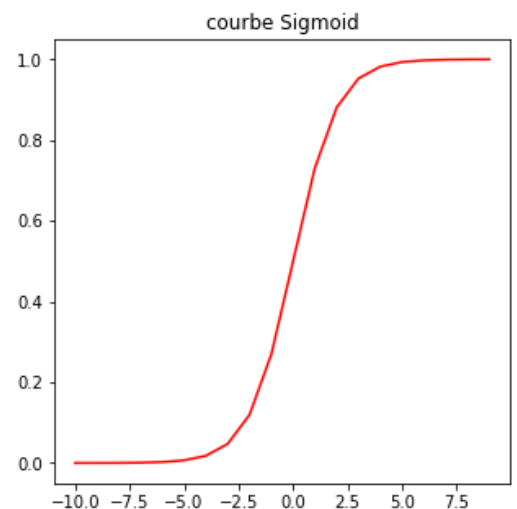
#### ❖ Courbe Sigmoïde :

on va estimer Y par la sigmoïde de la manière suivante

```
Entrée [7]: t=np.arange(-10,10,step=1)
fig,ax = plt.subplots(figsize=(5,5))
ax.set_title('courbe Sigmoid')
ax.plot(t,sigmoid(t),'r')
```

Out[7]: [<matplotlib.lines.Line2D at 0x1cadb1ec548>]

$$H_{\theta}(X) = \frac{1}{1 + e^{-(\theta_3 + X_2 \theta_2 + X_1 \theta_1)}}$$



## ❖ Insertion des uns et séparation des variables d'entrées et de sortie :

```
In [15]: #insertion de la colonne des uns : Le biais
data.insert(0, 'Ones', 1)
#séparation des variables d'apprentissage X et les labels y
cols = data.shape[1]
X = data.iloc[:,0:cols-1]
y = data.iloc[:,cols-1:cols]
#Les convertir en matrices et initialisation de theta
X = np.array(X.values)
y = np.array(y.values)
theta = np.zeros(3)
```

```
In [16]: X[0:11,:]
```

```
Out[16]: array([[ 1.          , 34.62365962, 78.02469282],
 [ 1.          , 30.28671077, 43.89499752],
 [ 1.          , 35.84740877, 72.90219803],
 [ 1.          , 60.18259939, 86.3085521 ],
 [ 1.          , 79.03273605, 75.34437644],
 [ 1.          , 45.08327748, 56.31637178],
 [ 1.          , 61.10666454, 96.51142588],
 [ 1.          , 75.02474557, 46.55401354],
 [ 1.          , 76.0987867 , 87.42056972],
 [ 1.          , 84.43281996, 43.53339331],
 [ 1.          , 95.86155507, 38.22527806]])
```

```
In [17]: y[0:11,:]
```

```
Out[17]: array([[0],
 [0],
 [0],
 [1],
 [1],
 [0],
 [1],
 [1],
 [1],
 [1],
 [0]], dtype=int64)
```

```
In [18]: X.shape, theta.shape, y.shape
```

```
Out[18]: ((100, 3), (3,), (100, 1))
```

## ❖ Fonction de l'erreur : Cost

On va créer la fonction qui calcule l'erreur entre  $H_{\theta}(X)$  et  $Y$  selon les équations suivantes :

$$J(\theta) = -\frac{1}{m} \sum [Y_i \log(h_{\theta}(X_i)) + (1 - Y_i) \log(1 - h_{\theta}(X_i))]$$

```
In [13]: def cost(theta, X, y):
theta = np.matrix(theta)
X = np.matrix(X)
y = np.matrix(y)
first = np.multiply(-y, np.log(sigmoid(X * theta.T)))
second = np.multiply((1 - y), np.log(1 - sigmoid(X * theta.T)))
return np.sum(first - second) / (len(X))
```

on calcule l'erreur pour  $\theta=(0,0,0)$

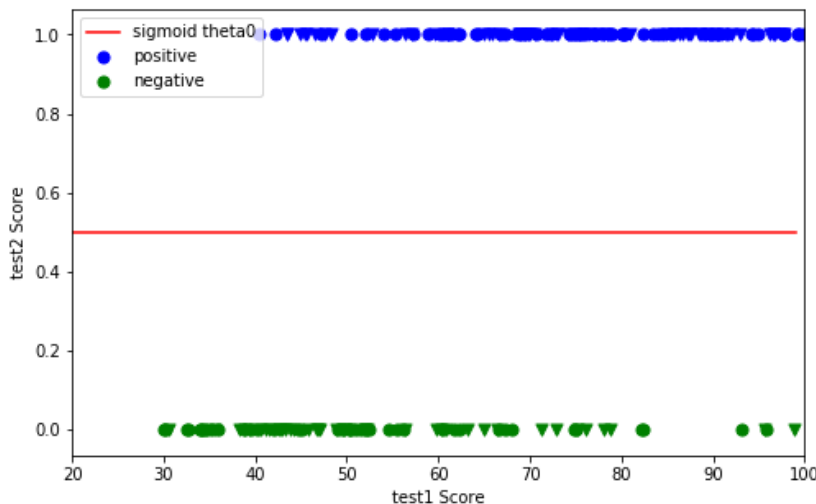
```
In [22]: cost(theta, X, y)
```

```
Out[22]: 0.6931471805599453
```

## ❖ visualisation de la courbe de prédiction pour $\theta = (0,0,0)$

```
Entrée [147]: fig, ax = plt.subplots(figsize=(8,5))
ax.scatter(positive['test1'],positive['resultat'], s=50, c='b', marker='o',label='positive')
ax.scatter(positive['test2'],positive['resultat'], s=50, c='b', marker='v')
ax.scatter(negative['test1'], negative['resultat'], s=50, c='g', marker='o',label='negative')
ax.scatter(negative['test2'], negative['resultat'], s=50, c='g', marker='v')
ax.plot(sigmoid(X * theta.T)[: ,0], 'r',label='sigmoid theta0')
ax.legend()
ax.set_xlim(20,100)
ax.set_xlabel('test1 Score')
ax.set_ylabel('test2 Score')
```

Out[147]: Text(0, 0.5, 'test2 Score')



La droite de la prédiction ne touche pas le nuage des points de la data

Ceci est attendu comme l'erreur de  $\theta$  initiale vaut 0.69

## ❖ Descente de gradient :

On va créer la fonction de descente de **gradient** selon l'équation suivante dont le rôle est de trouver  $\theta$  qui a l'erreur minimale possible

```
e [23]: def gradient(theta, X, y):
        theta = np.matrix(theta)
        X = np.matrix(X)
        y = np.matrix(y)
        parameters = int(theta.ravel().shape[1])
        grad = np.zeros(parameters)
        error = sigmoid(X * theta.T) - y
        for i in range(parameters):
            term = np.multiply(error, X[:,i])
            grad[i] = np.sum(term) / len(X)
        return grad
```

```
: [24]: gradient(theta, X, y)
```

```
it[24]: array([ -0.1          , -12.00921659, -11.26284221])
```



## ❖ Optimisation :

On veut optimiser la valeur de  $\theta$  pour cela on va utiliser `scipy.optimize`

```
[27]: import scipy.optimize as opt
      result = opt.fmin_tnc(func=cost, x0=theta, fprime=gradient, args=(X, y))
      result[0]
```

```
Out[27]: array([-25.16131872,  0.20623159,  0.20147149])
```

```
[28]: cost(result[0], X, y)
```

```
Out[28]: 0.20349770158947425
```

On a trouvé  $\theta$  minimale dont l'erreur égale à 0.2 l'erreur a diminué de 0.69 vers 0.2

## ❖ visualisation de la courbe de prédiction pour $\theta$ optimisé

```
In [83]: cost(result[0], X, y)
```

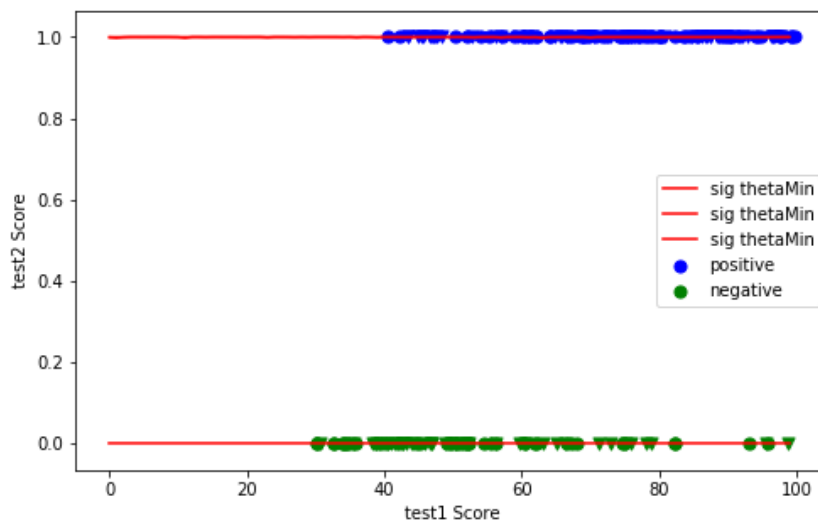
```
Out[83]: 0.20349770158947425
```

```
In [84]: theta_optimum=result[0]
      print(theta_optimum)
```

```
[-25.16131872  0.20623159  0.20147149]
```

```
[167]: fig, ax = plt.subplots(figsize=(8,5))
      ax.scatter(positive['test1'],positive['resultat'], s=50, c='b', marker='o', label='positive')
      ax.scatter(positive['test2'],positive['resultat'], s=50, c='b', marker='v')
      ax.scatter(negative['test1'], negative['resultat'], s=50, c='g', marker='o', label='negative')
      ax.scatter(negative['test2'], negative['resultat'], s=50, c='g', marker='v')
      #ax.plot(sigmoid(X *theta_optimum.T), 'r')
      D=sigmoid(X *theta_optimum.T)
      ax.plot(D, 'r',label='sig thetaMin')
      ax.legend()
      ax.set_xlabel('test1 Score')
      ax.set_ylabel('test2 Score')
```

```
[167]: Text(0, 0.5, 'test2 Score')
```



On remarque que la droite de la prédiction touche bien le nuage des points

C'est amélioré comme l'erreur a diminué pour  $\theta$  optimisé

### ❖ Prédiction et seuillage de $H_\theta(X)$

On construit la fonction **predict** qui calcule la probabilité qui n'est que la sigmoïde de  $h_\theta(X)$ . Cette fonction retourne un vecteur dont les composantes sont 0 ou 1 comme ça on construit Y prédite qu'on va la comparer avec Y réel

```
In [29]: def predict(theta, X):  
         probability = sigmoid(X * theta.T)  
         return [1 if x >= 0.5 else 0 for x in probability]
```

```
In [23]: theta_min = np.matrix(result[0])  
         predictions = predict(theta_min, X)  
         print(predictions)  
  
[0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0,  
1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0,  
1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1]
```

### ❖ Accuracy :

après avoir calculé Y estimé on va la comparer avec Y réel et voir les valeurs compatibles (justes)

puis on calcule la précision qui est le taux des bonnes réponses

```
In [30]: theta_min = np.matrix(result[0])  
         predictions = predict(theta_min, X)  
         correct = [1 if ((a == 1 and b == 1) or (a == 0 and b == 0)) else 0 for (a, b) in zip(predictions, y)]  
         accuracy = (sum(list(map(int, correct))) % len(correct))  
         print ("accuracy = {0}%".format(accuracy))  
  
accuracy = 89%
```

Pour  $\theta$  optimisé, On a obtenu une précision de 89%  
donc une erreur de 11% ce qui est le cas du taux d'erreur  
de  $\theta$  optimisé comme **cost**( $\theta_{min}, X, y$ )=0.2