

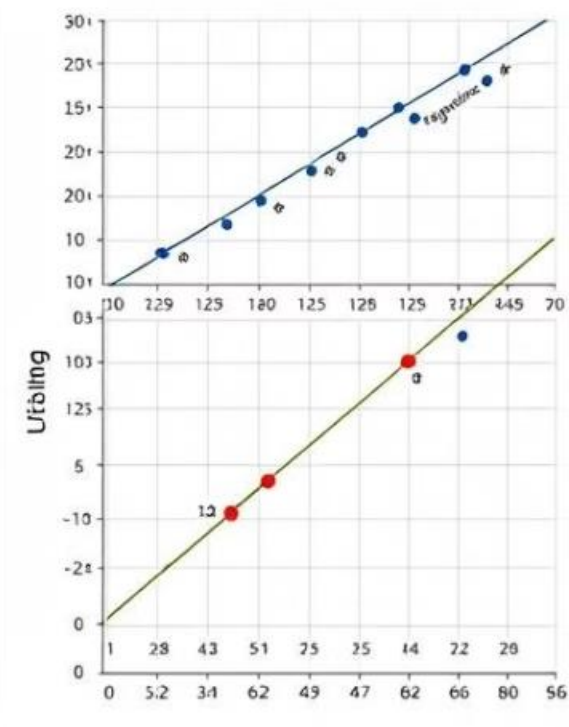
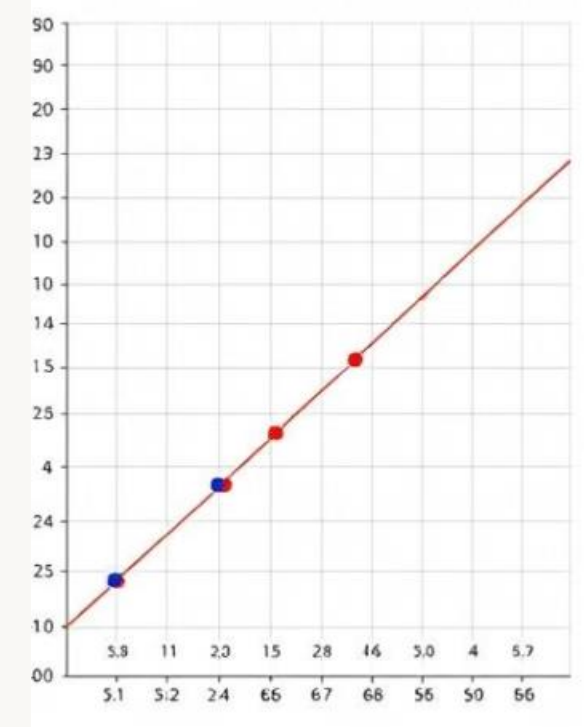


# La Régression Linéaire en Machine Learning

Réalisé par : BADR BERNANE (ISITD)

# Etapes pour résoudre un problème d'apprentissage supervisé (LR pour notre cas)

- 1 Dataset
- 2 Modél
- 3 Fonction Cout
- 4 Algorithme de minimisation



# 1 – DATASET :

Notre dataset représente l'ensemble des facteurs qui influence l'apparition du diabetes ou non.

Features						Target y
$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	
Glucose	BloodPressure	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
148	72	0	33.6	0.627	50	1
85	66	0	26.6	0.351	31	0
183	64	0	23.3	0.672	32	1
89	66	94	28.1	0.167	21	0
137	40	168	43.1	2.288	33	1
116	74	0	25.6	0.201	30	0
78	50	88	31	0.248	26	1
115	0	0	35.3	0.134	29	0
197	70	543	30.5	0.158	53	1
125	96	0	0	0.232	54	1

y	$x_1$	$x_2$	$x_3$	...	$x_n$
$y^{(1)}$	$x_1^{(1)}$	$x_2^{(1)}$	$x_3^{(1)}$	...	$x_n^{(1)}$
$y^{(2)}$	$x_1^{(2)}$	$x_2^{(2)}$	$x_3^{(2)}$	...	$x_n^{(2)}$
$y^{(3)}$	$x_1^{(3)}$	$x_2^{(3)}$	$x_3^{(3)}$	...	$x_n^{(3)}$
...	...	...	...	...	...
$y^{(m)}$	$x_1^{(m)}$	$x_2^{(m)}$	$x_3^{(m)}$	...	$x_n^{(m)}$

vecteur target  $y \in \mathbb{R}^{m \times 1}$

$$y = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \dots \\ y^{(m)} \end{pmatrix}$$

matrice features  $X \in \mathbb{R}^{m \times n}$

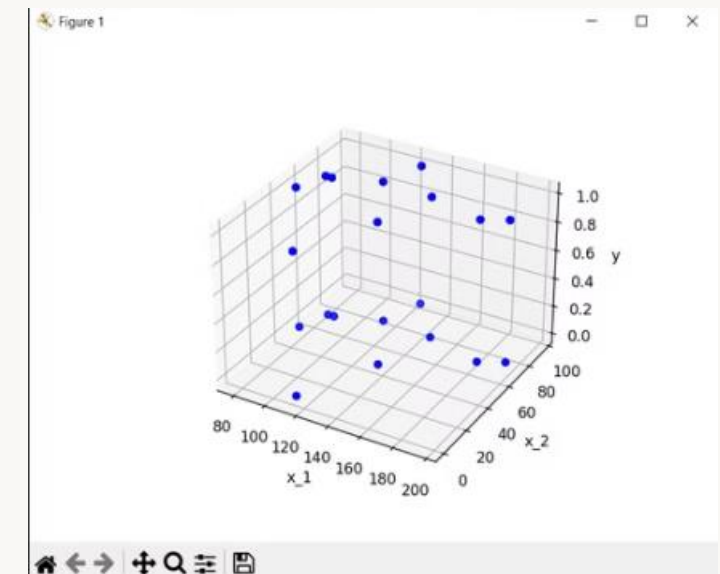
$$X = \begin{pmatrix} x_1^{(1)} & \dots & x_n^{(1)} \\ \vdots & \ddots & \vdots \\ x_1^{(m)} & \dots & x_n^{(m)} \end{pmatrix}$$

Outcome est une fonction de ( Glucose , BloodPressure, Insulin, BMI, DiabetesPedigreeFunction et Age).

# 1 – DATASET :

```
1 from mpl_toolkits.mplot3d import Axes3D
2 import numpy as np
3 import pandas as pd
4 import matplotlib.pyplot as plt
5
6
7 data = pd.read_csv('C:\sokar.csv')
8
9 x = data[['Glucose', 'BloodPressure', 'BMI', 'Age']].values
10 y = data['Outcome'].values.reshape(-1, 1)
11 #print(x.shape) # Pour vérifier Les dimensions de x
12 #print(y.shape) # Pour vérifier Les dimensions de y
```

```
1 """la premiere etape c'est la dataset sous sa forme matricielle"""
2 """la matrice X"""
3 X = np.hstack((x, np.ones((x.shape[0], 1))))
4 #print(X)
5 """maintenant theta qui contient les paramètres (a et b pour un model simple) """
6 theta = np.random.randn(X.shape[1], 1)
7 #print(theta.shape)
8 #print(theta)
9 #plt.scatter(x[:,0], y) # afficher les résultats. x_1 en abscisse et y en ordonnée
10 #plt.title('badr')
11 #plt.show()
12
13 fig = plt.figure()
14 ax = fig.add_subplot(111, projection='3d')
15 ax.scatter(x[:,0], x[:,1], y, c='b', marker='o') # affiche en 3D la variable x_1, x_2, et la target y
16 # affiche les noms des axes
17 ax.set_xlabel('x_1')
18 ax.set_ylabel('x_2')
19 ax.set_zlabel('y')
20 plt.show()
```

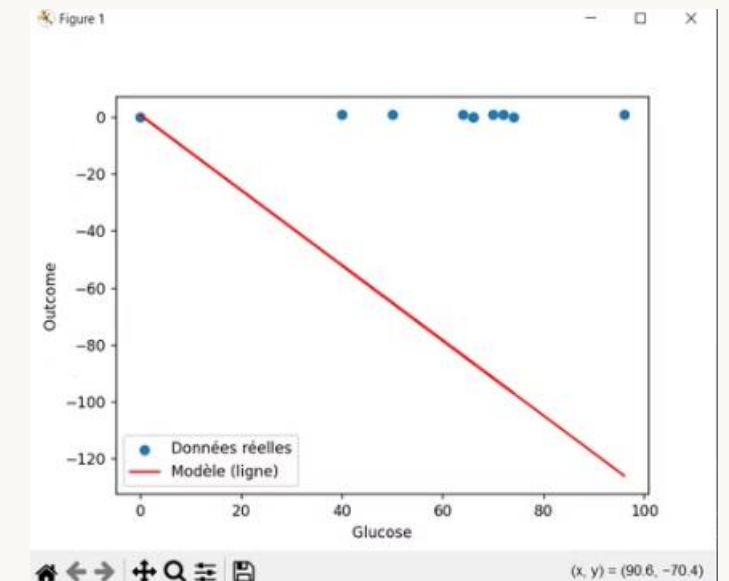
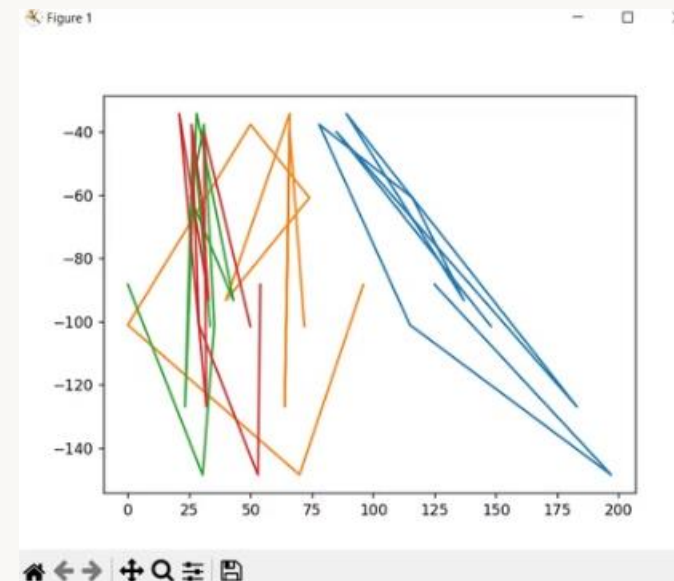


- La première image illustre l'importation du dataset sous forme de fichier CSV et la sélection de quatre features. (Lorsqu'on dépasse trois dimensions, il devient difficile pour un humain de comprendre les graphes)
- La deuxième image montre la transformation du dataset en une matrice, où l'on parle de X, la matrice des features.
- La troisième image représente un graphe 3D montrant la relation entre la target Y, le Glucose et la BloodPressure. Le graphe semble un peu vide car une petite portion du dataset a été utilisée, afin de faciliter la compréhension des concepts.

## 2 – Model :

Pour notre model on a :  $F = X.\theta$

```
1  """le modèle F = X.θ """
2  def model(X, theta):
3      return X.dot(theta)
4
5  #print(model(X, theta)) #pour tester est ce que tout sa marche bien
6  #plt.plot(x, model(X, theta))
7  #plt.show()
8
```



- Le graphe 1 represente l'affichage des resultats de notre modele par rapport a notre dataset x .
- Les résultats semblent illogiques, mais cela est normal, car nous développons un modèle qui traite plusieurs dimensions (>3). Contrairement à un modèle à une seule dimension où les relations sont plus intuitives et compréhensibles, la complexité augmente avec le nombre de dimensions. Le graphe 2 le confirme, car il montre que les choses sont plus faciles à interpréter lorsqu'on se concentre sur une seule variable.

```

45 """La fonction cout"""
    Comment Code
46 def fonction_cout(X, theta, y):
47     m = len(y)
48     return 1/(2*m) * np.sum((model(X, theta) - y)**2)
49     #print(fonction_cout(X, theta, y))

```

### 3 – Fonction Cout :

Formule :

$$J(\theta) = \frac{1}{2m} \sum (X \cdot \theta - y)^2$$

On mesure les erreurs du modele sur le Dataset X, y en implémenter l'Erreur Quadratique Moyenne.

Résultats:

```

PROBLEMS 5 OUTPUT DEBUG CONSOLE TERMINAL PORTS SEARCH ERROR JUPYTER NUGET COMMENTS
PS C:\Users\BERNANE> python -u "c:\Users\BERNANE\Desktop\controle_LR_surdataset_diabet.py"
c:\Users\BERNANE\Desktop\controle_LR_surdataset_diabet.py:7: SyntaxWarning: invalid escape sequence '\s'
  data = pd.read_csv('C:\sokar.csv')
872.4370585990048
-0.19413400844612072
PS C:\Users\BERNANE>

```



## 4- Algorithme de minimisation

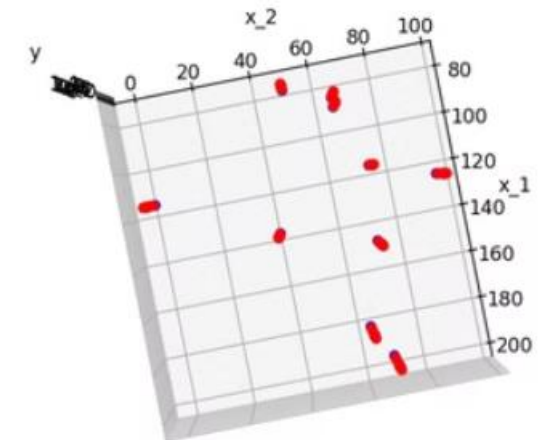
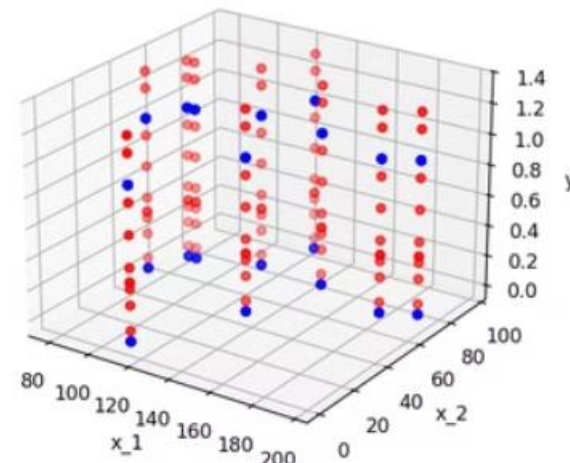
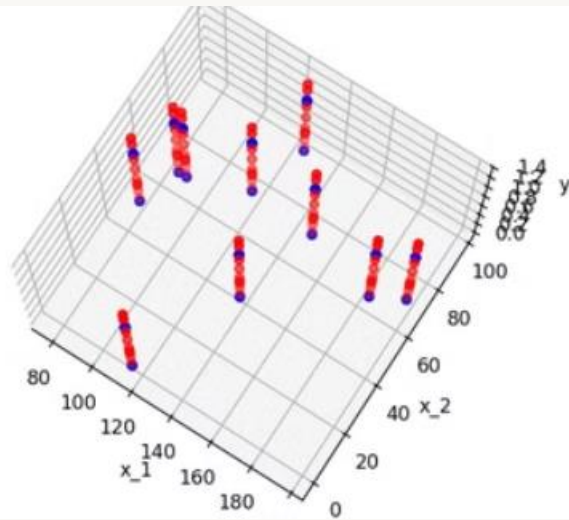
Pour ce algorithme on a deux methodes : les moindres carres et la descente de gradient.

- Nous avons choisi la descente de gradient, car la méthode des moindres carrés (ou les équations normales) implique l'inversion de matrices. Bien que cela fonctionne parfaitement pour des cas simples, cela devient problématique pour de grands ensembles de données (par exemple, 1 million d'exemples). Dans ces situations, même un ordinateur peut prendre des millions d'années pour effectuer les calculs nécessaires. La descente de gradient, en revanche, est plus efficace pour traiter de grands ensembles de données, car elle itère progressivement vers une solution sans nécessiter d'inversion de matrice.

```
1  """gradient & Descente de gradient"""
2  # awalan le gradient
3  def gradient(X, theta, y):
4      m = len(y)
5      return 1/(2*m) * X.T.dot(model(X, theta) - y)
6
7  # maintenant la descente du gradient
8  def descente_du_gradient(X, theta, y, alpha, n_iter):
9      cost_history = np.zeros(n_iter)
10     for i in range(0, n_iter):
11         theta = theta - alpha * gradient(X, theta, y)
12         cost_history[i] = fonction_cout(X, theta, y)
13     return theta, cost_history
14
```

# PHASE D'Entraînement du Modèle

```
1 """ entraînement """
2 theta_final, cost_history = descente_du_gradient(X, theta, y, alpha=0.0001, n_iter=1000)
3 #print(theta_final)
4 prediction = model(X, theta_final)
5 """plt.scatter(x[:,2], y)
6 plt.plot(x[:,2], prediction, color='red')
7 plt.show()"""
8
9 fig = plt.figure()
10 ax = fig.add_subplot(111, projection='3d')
11 ax.scatter(x[:,0], x[:,1], y, c='b', marker='o') # affiche en 3D la variable x_1, x_2, et la target y
12 ax.scatter(x[:,0], x[:,1], prediction, c='r', marker='o')
13 # affiche les noms des axes
14 ax.set_xlabel('x_1')
15 ax.set_ylabel('x_2')
16 ax.set_zlabel('y')
17 plt.show()
```



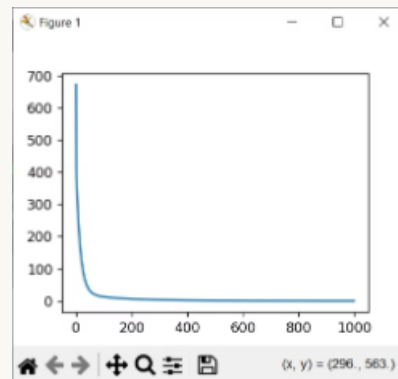
comme conclusion notre modèle travaille bien .



# Améliorations :

```
1 """ La courbe d'apprentissage """
2 plt.plot(range(len(cost_history)), cost_history)
3 plt.show()
```

## La courbe d'apprentissage



Cela nous aide à bien choisir le nombre d'itérations pour que le modèle fonctionne efficacement. Par exemple, à partir du graphe, nous pouvons observer qu'après 500 itérations, les résultats deviennent satisfaisants. Cela indique que 500 itérations sont suffisantes pour que la machine converge vers une solution optimale.

```
1 """coefficient de determination"""
2 def coef_determination(y, pred):
3     u = ((y - pred)**2).sum()
4     v = ((y - y.mean())**2).sum()
5     return 1 - u/v
6 print(coef_determination(y, prediction))
```

## Le coefficient de determination

```
PROBLEMS 5 OUTPUT DEBUG CONSOLE TERMINAL PORTS SEARCH ERROR JUPYTER NUGET COMMENTS
PS C:\Users\BERNAME> python -u "c:\Users\BERNAME\Desktop\controle_LR_surdataset_diabet.py"
c:\Users\BERNAME\Desktop\controle_LR_surdataset_diabet.py:7: SyntaxWarning: invalid escape sequence '\s'
  data = pd.read_csv('C:\sokar.csv')
6413.363280518358
-2.0283872349651992
PS C:\Users\BERNAME>
```

la performance de notre modele .

## Annexe :

```
1 from mpl_toolkits.mplot3d import Axes3D
2 import numpy as np
3 import pandas as pd
4 import matplotlib.pyplot as plt
5 data = pd.read_csv('C:\sokar.csv')
6 x = data[['Glucose', 'BloodPressure', 'BMI', 'Age']].values
7 y = data['Outcome'].values.reshape(-1, 1)
8 #print(x.shape) # Pour vérifier Les dimensions de x
9 #print(y.shape) # Pour vérifier Les dimensions de y
10 """la premiere etape c'est la dataset sous sa forme matricielle"""
11 """la matrice X"""
12 X = np.hstack((x, np.ones((x.shape[0], 1))))
13 #print(X)
14 """maintenant theta qui contient les paramètres (a et b pour un model simple) """
15 theta = np.random.randn(X.shape[1], 1)
16 #print(theta.shape)
17 #print(theta)
18 #plt.scatter(x[:,0], y) # afficher les résultats. x_1 en abscisse et y en ordonnée
19 #plt.title('badr')
20 #plt.show()
21 """fig = plt.figure()
22 ax = fig.add_subplot(111, projection='3d')
23 ax.scatter(x[:,0], x[:,1], y, c='b', marker='o') # affiche en 3D la variable x_1, x_2, et la target y
24 # affiche les noms des axes
25 ax.set_xlabel('x_1')
26 ax.set_ylabel('x_2')
27 ax.set_zlabel('y')
28 plt.show()"""
29 """le modèle F = X.θ """
30 def model(X, theta):
31     return X.dot(theta)
32 #print(model(X, theta)) #pour tester est ce que tout sa marche bien
33 #plt.scatter(x, y)
34 #plt.plot(x, model(X, theta))
35 #plt.show()
36 """La fonction cout"""
37 def fonction_cout(X, theta, y):
38     m = len(y)
39     return 1/(2*m) * np.sum((model(X, theta) - y)**2)
40 print(fonction_cout(X, theta, y))
41 """gradient & Descente de gradient"""
42 # awalan le gradient
43 def gradient(X, theta, y):
44     m = len(y)
45     return 1/(2*m) * X.T.dot(model(X, theta) - y)
46 # maintenant la descente du gradient
47 def descente_du_gradient(X, theta, y, alpha, n_iter):
48     cost_history = np.zeros(n_iter)
49     for i in range(0, n_iter):
50         theta = theta - alpha * gradient(X, theta, y)
51         cost_history[i] = fonction_cout(X, theta, y)
52     return theta, cost_history
53 """ entraînement """
54 theta_final, cost_history = descente_du_gradient(X, theta, y, alpha=0.0001, n_iter=1000)
55 #print(theta_final)
56 prediction = model(X, theta_final)
57 """plt.scatter(x[:,2], y)
58 plt.plot(x[:,2], prediction, color='red')
59 plt.show()"""
60
61 """fig = plt.figure()
62 ax = fig.add_subplot(111, projection='3d')
63 ax.scatter(x[:,0], x[:,1], y, c='b', marker='o') # affiche en 3D la variable x_1, x_2, et la target y
64 ax.scatter(x[:,0], x[:,1], prediction, c='r', marker='o')
65 # affiche les noms des axes
66 ax.set_xlabel('x_1')
67 ax.set_ylabel('x_2')
68 ax.set_zlabel('y')
69 plt.show()"""
70
71 """ La courbe d'apprentissage """
72 #plt.plot(range(len(cost_history)), cost_history)
73 #plt.show()
74 """coefficient de determination"""
75 def coef_determination(y, pred):
76     u = ((y - pred)**2).sum()
77     v = ((y - y.mean())**2).sum()
78     return 1 - u/v
79 print(coef_determination(y, prediction))
```