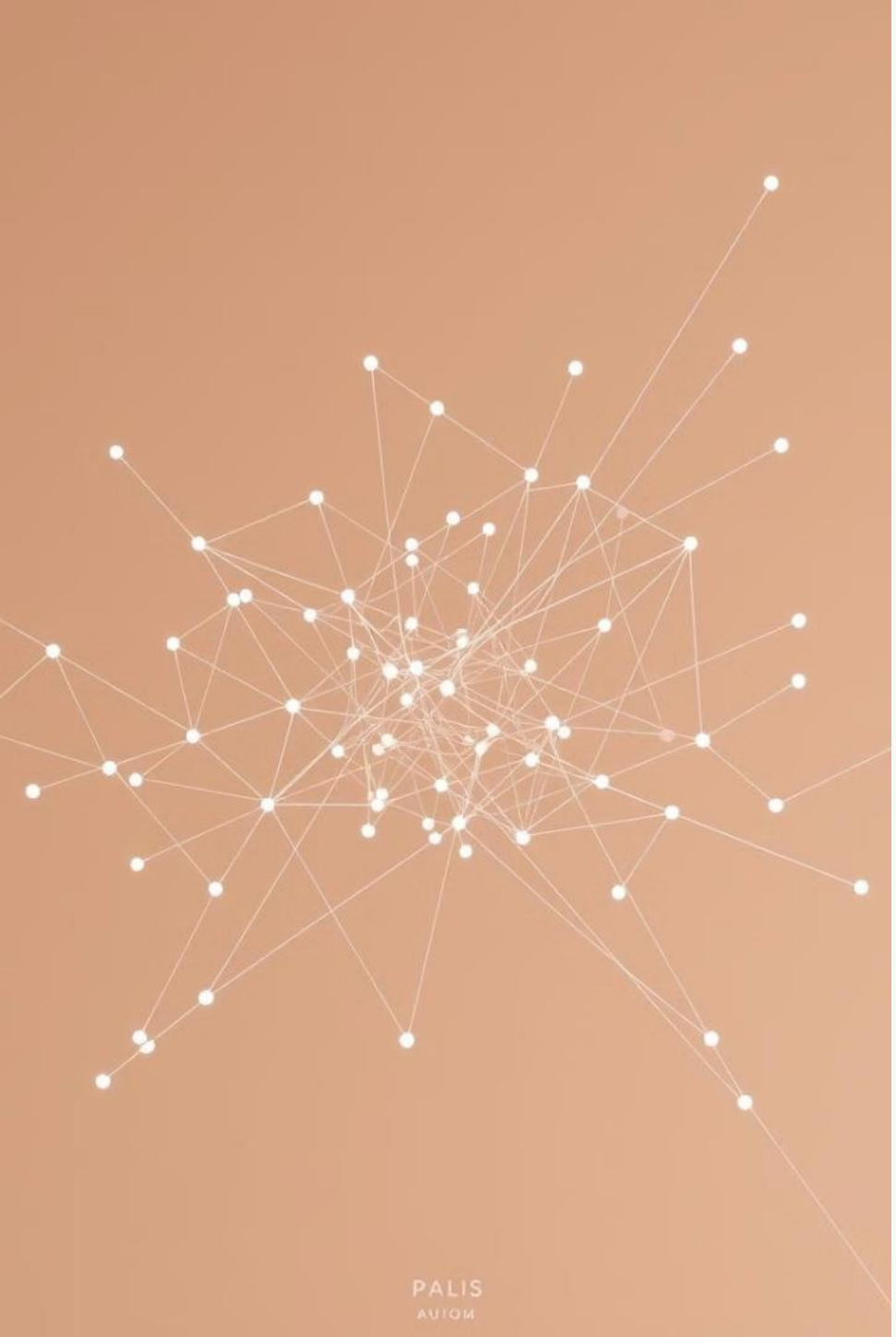
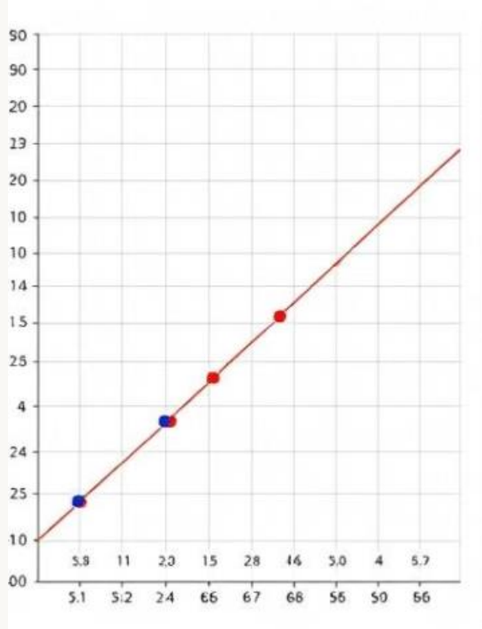


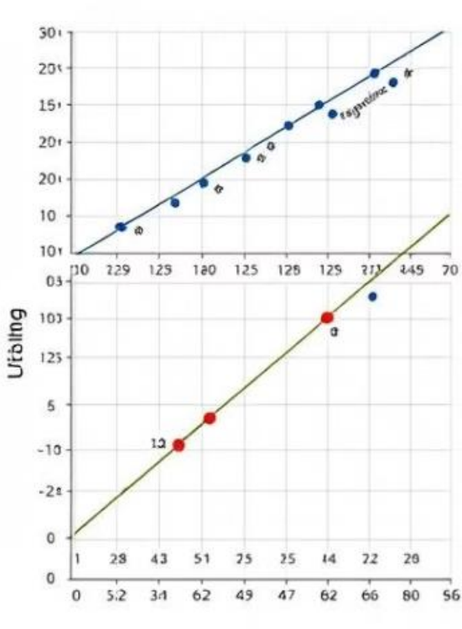
La Régression Linéaire en Machine Learning

Réalisé par : BADR BERNANE (ISITD)

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

1. Dataset
2. Modèl
3. Fonction Cout
4. Algorithme de minimisation



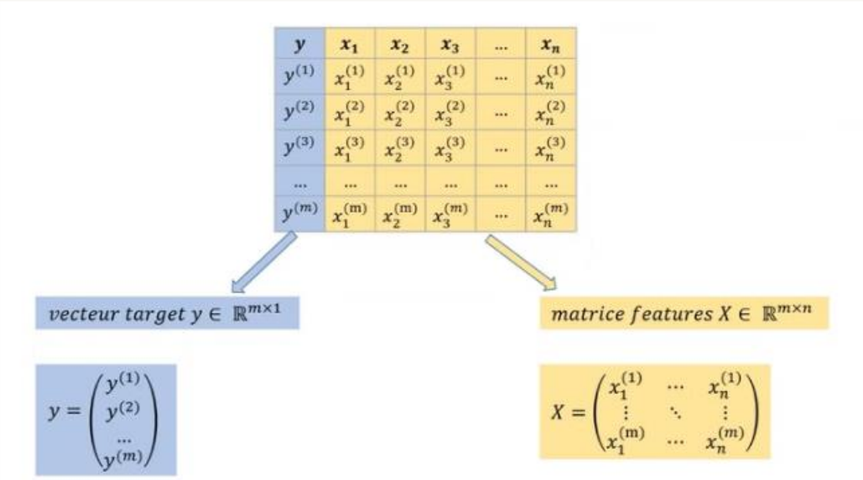


1 – DATASET :

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

Features

x_1	x_2	x_3	x_4	x_5	x_6	Target y
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

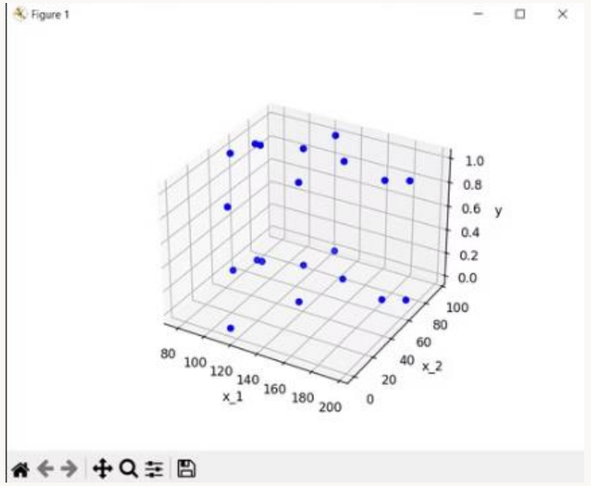


Outcome est une fonction de (Glucose , BloodPressure, Insulin, BMI, DiabetesPedigreeFunction et Age).

1 – DATASET :

```python
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt


data = pd.read_csv('C:\sokar.csv')

x = data[['Glucose', 'BloodPressure', 'BMI', 'Age']].values
y = data['Outcome'].values.reshape(-1, 1)
#print(x.shape)  # Pour vérifier les dimensions de x
#print(y.shape)  # Pour vérifier les dimensions de y
```

```python
"""la premiere etape c'est la dataset sous sa forme matricille"""
"""la matrice X"""
X = np.hstack((x, np.ones((x.shape[0], 1))))
#print(X)
"""maintenant theta qui contient  les paramétres (a et b pour un model simple) """
theta = np.random.randn(X.shape[1], 1)
#print(theta.shape)
#print(theta)
#plt.scatter(x[:,0], y) # afficher les résultats. x_1 en abscisse et y en ordonnée
#plt.title('badr')
#plt.show()

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x[:,0], x[:,1], y, c='b', marker='o') # affiche en 3D la variable x_1, x_2, et la target y
# affiche les noms des axes
ax.set_xlabel('x_1')
ax.set_ylabel('x_2')
ax.set_zlabel('y')
plt.show()
```

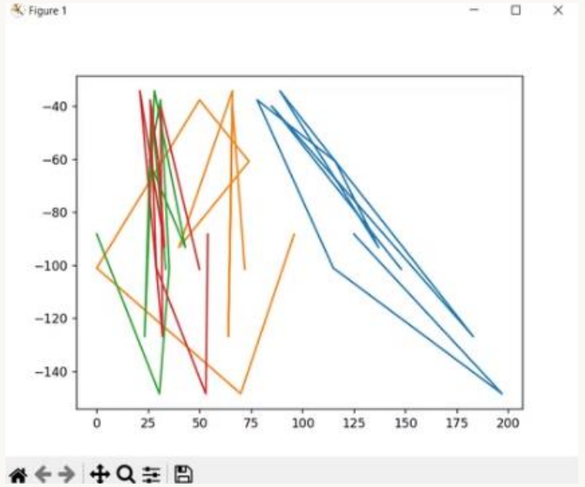


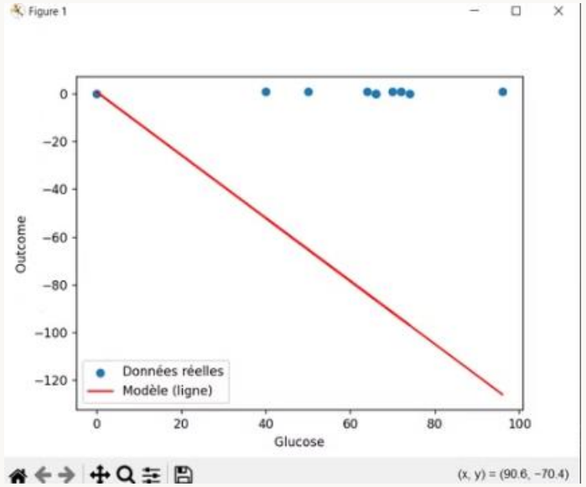
- La première image illustre l'importation du dataset sous forme de fichier CSV et la sélection de quatre features. (Lorsqu'on dépasse trois dimensions, il devient difficile pour un humain de comprendre les graphes)
- La deuxième image montre la transformation du dataset en une matrice, où l'on parle 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.θ**

```python
"""le modéle  F = X.θ """
def model(X, theta):
    return X.dot(theta)

#print(model(X, theta)) #pour tester est ce que tout sa marche bien
#plt.plot(x, model(X, theta))
#plt.show()
```





- 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 :

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

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

Résultats:

4- Algorithme de minimisation

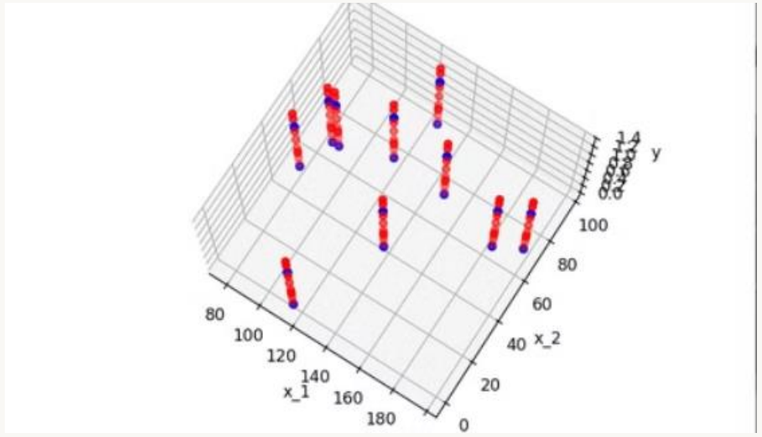
Pour ce algorithme on a deux methodes : les moindres carres et la descente de gradient.

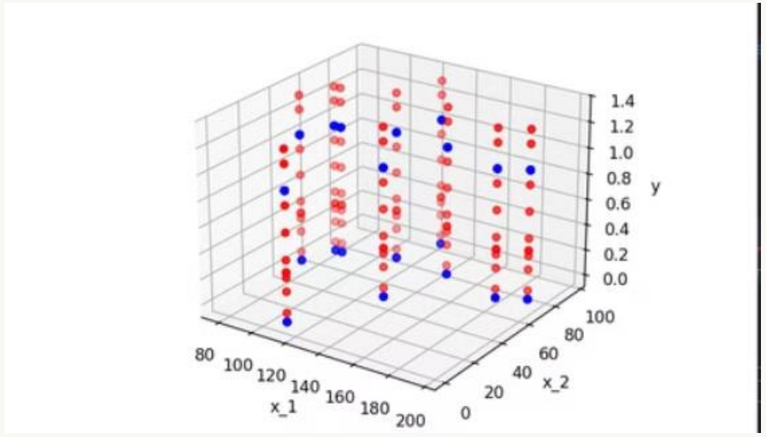
- Nous avons choisi la descente de gradient, car la méthode des moindres carrés (ou les équations normales) implique l'inversion de matrices. Bien que cela fonctionne parfaitement pour des cas simples, cela devient problématique pour de grands ensembles de données (par exemple, 1 million d'exemples). 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.

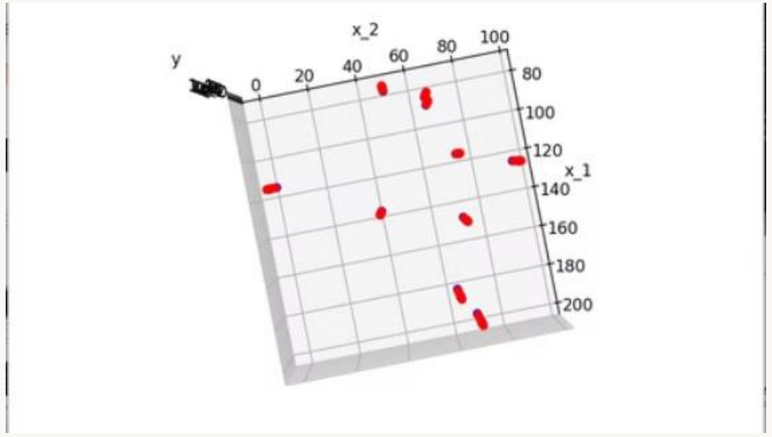
```python
"""gradient & Descente de gradient"""
# awalan le gradient
def gradient(X, theta, y):
    m = len(y)
    return 1/(2*m) * X.T.dot(model(X, theta) - y)

# maintenant la descente du gradient
def descente_du_gradient(X, theta, y, alpha, n_iter):
    cost_history = np.zeros(n_iter)
    for i in range(0, n_iter):
        theta = theta - alpha * gradient(X, theta, y)
        cost_history[i] = fonction_cout(X, theta, y)
    return theta, cost_history

```

PHASE D'Entraînement du Modèle

```python
""" entrainemant """
theta_final, cost_history = descente_du_gradient(X, theta, y, alpha=0.0001, n_iter=1000)
#print(theta_final)
prediction = model(X, theta_final)
"""plt.scatter(x[:,2], y)
plt.plot(x[:,2], prediction, color='red')
plt.show()"""

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x[:,0], x[:,1], y, c='b', marker='o') # affiche en 3D la variable x_1, x_2, et la target y
ax.scatter(x[:,0], x[:,1], prediction, c='r', marker='o')
# affiche les noms des axes
ax.set_xlabel('x_1')
ax.set_ylabel('x_2')
ax.set_zlabel('y')
plt.show()
```





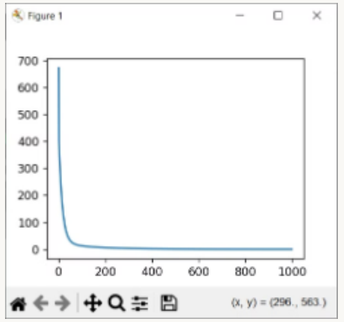


comme conclusion notre modéle travaille bien .

Améliorations :

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

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

Le coeficient de determination

```
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')
6413.363280518358
-2.0283872349651992
PS C:\Users\BERNANE>
```

la performance de notre modele .

Annexe :

```python
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
data = pd.read_csv('C:\sokar.csv')
x = data[['Glucose', 'BloodPressure', 'BMI', 'Age']].values
y = data['Outcome'].values.reshape(-1, 1)
#print(x.shape)  # Pour vérifier les dimensions de x
#print(y.shape)  # Pour vérifier les dimensions de y
"""la premiere etape c'est la dataset sous sa forme matricille"""
"""la matrice X"""
X = np.hstack((x, np.ones((x.shape[0], 1))))
#print(X)
"""maintenant theta qui contient  les paramétres (a et b pour un model simple) """
theta = np.random.randn(X.shape[1], 1)
#print(theta.shape)
#print(theta)
#plt.scatter(x[:,0], y) # afficher les résultats. x_1 en abscisse et y en ordonnée
#plt.title('badr')
#plt.show()
"""fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x[:,0], x[:,1], y, c='b', marker='o') # affiche en 3D la variable x_1, x_2, et la target y
# affiche les noms des axes
ax.set_xlabel('x_1')
ax.set_ylabel('x_2')
ax.set_zlabel('y')
plt.show()"""
"""le modéle  F = X.θ """
def model(X, theta):
    return X.dot(theta)
#print(model(X, theta)) #pour tester est ce que tout sa marche bien
#plt.scatter(x, y)
#plt.plot(x, model(X, theta))
#plt.show()
"""La fonction cout"""
def fonction_cout(X, theta, y):
    m = len(y)
    return 1/(2*m) * np.sum((model(X, theta) - y)**2)
print(fonction_cout(X, theta, y))
"""gradient & Descente de gradient"""
# awalan le gradient
def gradient(X, theta, y):
    m = len(y)
    return 1/(2*m) * X.T.dot(model(X, theta) - y)
# maintenant la descente du gradient
def descente_du_gradient(X, theta, y, alpha, n_iter):
    cost_history = np.zeros(n_iter)
    for i in range(0, n_iter):
        theta = theta - alpha * gradient(X, theta, y)
        cost_history[i] = fonction_cout(X, theta, y)
    return theta, cost_history
"""  entrainemant """
theta_final, cost_history = descente_du_gradient(X, theta, y, alpha=0.0001, n_iter=1000)
#print(theta_final)
prediction = model(X, theta_final)
"""plt.scatter(x[:,2], y)
plt.plot(x[:,2], prediction, color='red')
plt.show()"""

"""fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x[:,0], x[:,1], y, c='b', marker='o') # affiche en 3D la variable x_1, x_2, et la target y
ax.scatter(x[:,0], x[:,1], prediction, c='r', marker='o')
# affiche les noms des axes
ax.set_xlabel('x_1')
ax.set_ylabel('x_2')
ax.set_zlabel('y')
plt.show()"""

""" La courbe d'apprentissage """
#plt.plot(range(len(cost_history)), cost_history)
#plt.show()
"""coeficient de determination"""
def coef_determination(y, pred):
    u = ((y - pred)**2).sum()
    v = ((y - y.mean())**2).sum()
    return 1 - u/v
print(coef_determination(y, prediction))
```