

The background features a complex network of thin grey lines and dots, forming a web-like structure. Scattered throughout are various triangles of different sizes and orientations, some with solid dots at their vertices. The overall aesthetic is minimalist and technical, suggesting a focus on data or mathematics.

PREMIER MODELE IA

Régression linéaire simple, multiple et polynomiale
Décembre 2020 - Ludivine, Christelle

Rappels sur les régressions

Régressions linéaires simple et multiple, régression polynomiale

01

Equations

Équations matricielles des du modèle, fonction de coût, gradient et descente de gradient

02

Représentation Graphique

Présentation des résultats sous forme graphique

03

SOMMAIRE

04

Evaluation des modèles

Performance du modèle et pertinence des résultats

05

Résultats avec Scikit-Learn

Utilisation du module sklearn de python

06

Conclusion

Évolution, difficultés rencontrées

01

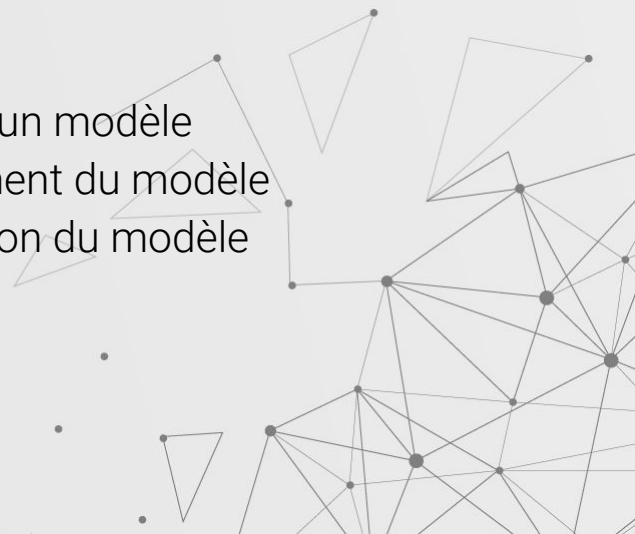
Rappels

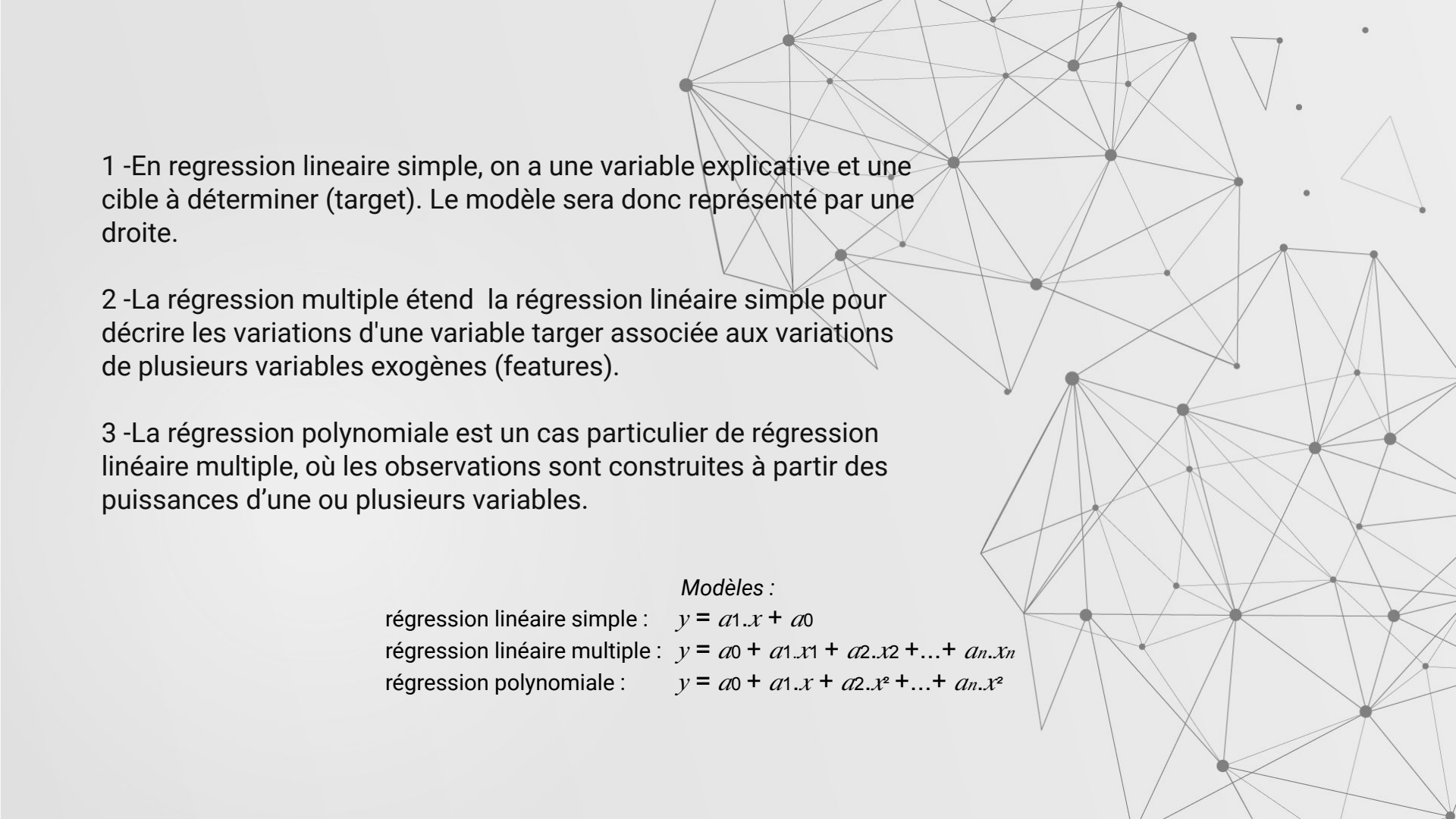




La régression est l'opération consistant à expérimenter pour faire passer une droite ou une courbe mathématique le plus près possible des points d'un graphique.

- Choix d'un modèle
- Ajustement du modèle
- Evaluation du modèle





1 -En regression lineaire simple, on a une variable explicative et une cible à déterminer (target). Le modèle sera donc représenté par une droite.

2 -La régression multiple étend la régression linéaire simple pour décrire les variations d'une variable target associée aux variations de plusieurs variables exogènes (features).

3 -La régression polynomiale est un cas particulier de régression linéaire multiple, où les observations sont construites à partir des puissances d'une ou plusieurs variables.

Modèles :

régression linéaire simple : $y = a_1.x + a_0$

régression linéaire multiple : $y = a_0 + a_1.x_1 + a_2.x_2 + \dots + a_n.x_n$

régression polynomiale : $y = a_0 + a_1.x + a_2.x^2 + \dots + a_n.x^2$

02

Equations

Équations matricielles des du modèle, fonction de coût, gradient et descente de gradient





Modèle

$$y = X \cdot \theta$$

Fonction de coût

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

Le Gradient

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

Descente de gradient

$$\theta = \theta - \frac{\partial J(\theta)}{\partial \theta}$$

Modèles :

régression linéaire simple : $y = a_1 \cdot x + a_0$

régression linéaire multiple : $y = a_0 + a_1 \cdot x_1 + a_2 \cdot x_2 + \dots + a_n \cdot x_n$

régression polynomiale : $y = a_0 + a_1 \cdot x + a_2 \cdot x^2 + \dots + a_n \cdot x^2$

En python :

```
# Création du modèle (model(X,theta))
def modelmult(X, theta):
    return X.dot(theta)

# Fonction du coût (fonction_cout(X,y,theta))
def fonction_cout(X, y, theta):
    return 1/(2*m) * np.sum((modelmult(X, theta) - y)**2)

# Le gradient (gradient(X,y,theta))
def gradient(X,y,theta):
    return 1/m * X.T.dot(modelmult(X, theta) - y)

# Descente du gradient (descente_gradient(X,y,theta,alpha,n_iterations))
def descente_gradient(X,y,theta,alpha,n_iterations):
    for i in range(n_iterations):
        theta = theta - alpha*gradient(X, y, theta)
    return theta
```


Code python

Présentation des résultats



1-Utilisez les bibliothèques de Python pour récupérer le contenu du jeu de données.

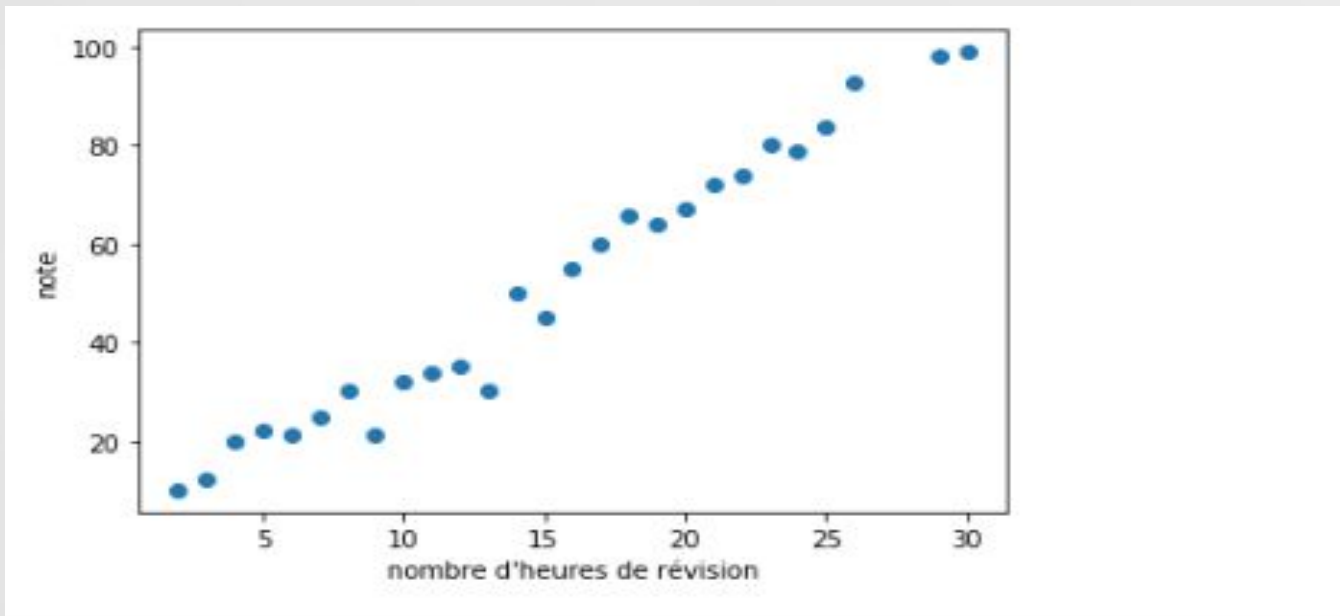
```
import pandas as pd    # pour la lecture de fichier
import numpy as np     # maths
import matplotlib.pyplot as plt # affichage des graphiques
```

2-Utilisez les bibliothèques de Python pour récupérer le contenu du jeu de données.

```
# Visualisation des données
nobs = len(reg_simple)
print("colonnes :", reg_simple.columns, " - nombre d'observations :", nobs)
plt.scatter(reg_simple["heure_rev"], reg_simple["note"])
plt.xlabel('nombre d'heures de révision')
plt.ylabel('note')
plt.show()
```

Nuage de points

Axe des ordonnées, notre "y" notes .
En abscisses le "x " nombres d'heures de révisions.



Création du modèle (model(X,theta))

```
a = np.array(reg_simple)
```

```
m = a.shape[0]
```

```
x = a[:,0]
```

```
x = x[:, np.newaxis]
```

```
y = a[:,1]
```

```
y = y[:, np.newaxis]
```

sert a coller 2 vecteurs

```
X = np.hstack((x, np.ones(x.shape)))
```

```
theta = np.random.randn(2,1)
```



Gradient et descente de gradient.

```
# Descente du gradient (descente_gradient(X,y,theta,alpha,n_iterations))
```

```
# 1° Initialiser avec x0(au hasard) -- ok
```

```
# 2° Répéter  $\theta(t+1) = \theta(t) - \alpha \times \nabla f(\theta(t))$ 
```

```
# 3° Jusqu'à convergence
```

```
def descente_gradient(X,y,theta,alpha,n_iterations):
```

```
    for i in range(n_iterations): #2°
```

```
        theta = theta - alpha*gradient(X, y, theta)
```

```
    return theta
```

```
[[3.31390942]  
 [0.95131246]]  
[[3.3111333 ]  
 [1.00517102]]  
[[-2.52998398e+57]  
 [-1.30403637e+56]]  
[[3.33070488]  
 [0.62546093]]
```



03

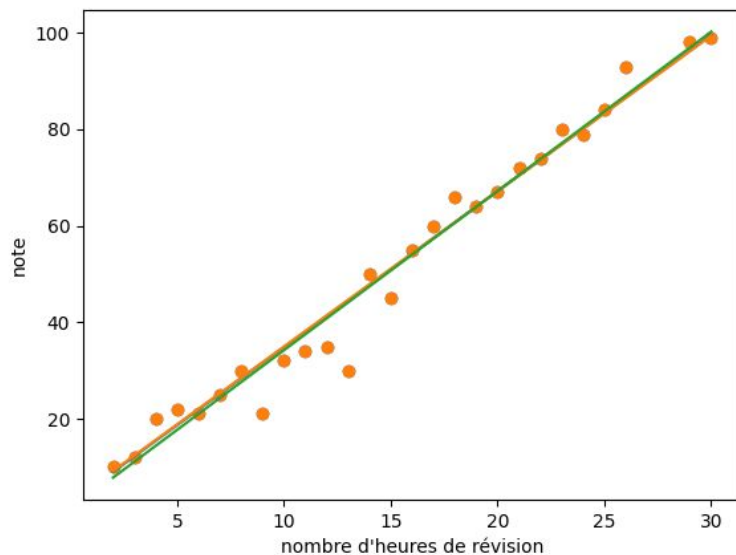
Graphiques

Présentation des résultats



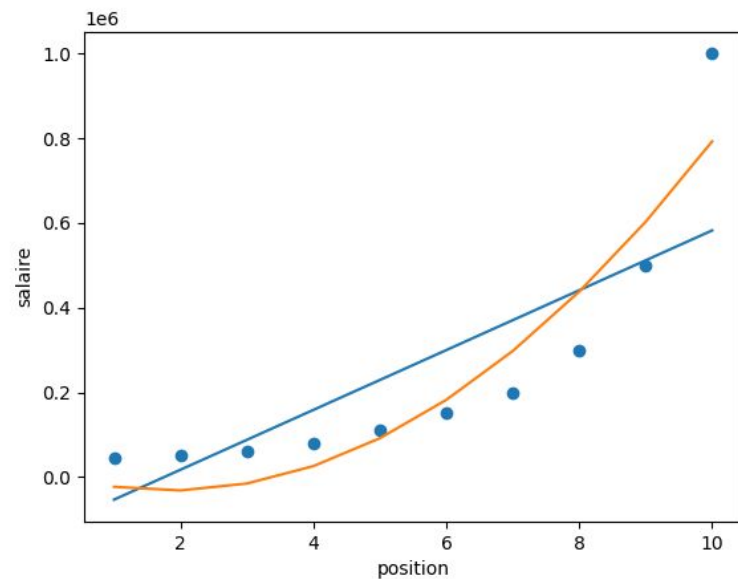
Révisions

Régression linéaire simple



Salaire

Régression linéaire polynomiale



04

Evaluation

Evaluation des modèles



Révisions

Régression linéaire simple

Utilisation du R^2

```
def r2(X, y, y_pred):  
    return 1 - (np.sum((y - y_pred)**2) / np.sum((y - np.mean(y))**2))  
  
y_pred = model(X, theta4)  
  
print(r2(X, y, y_pred)) # 0.9732961007203775
```

Immobilier Boston

Régression linéaire multiple

Erreur quadratique moyenne

```
theta1 = descente_gradient(X, y, theta, 0.001, 5000)  
theta2 = descente_gradient(X, y, theta, 0.0005, 3300)  
theta3 = descente_gradient(X, y, theta, 0.002, 1500)  
  
print(theta1, fonction_cout(X, y, theta1)) # 13.196  
print(theta2, fonction_cout(X, y, theta2)) # 14.115  
print(theta3, fonction_cout(X, y, theta3)) # 1050815795310118.4  
  
print(mean_squared_error(y, modelmult(X, theta1))) # 26.460  
print(mean_squared_error(y, modelmult(X, theta2))) # 28.035  
print(mean_squared_error(y, modelmult(X, theta3))) # 7594507303458618.0
```



05

ScikitLearn



Régression avec scikit learn:

Simple:

```
from sklearn import linear_model
```

```
import matplotlib.pyplot as plt
```

```
import numpy as np
```

```
import random
```

```
#entraînement des données
```

```
X = [i for i in range(10)]
```

```
Y = [random.gauss(x,0.75) for x in X]
```

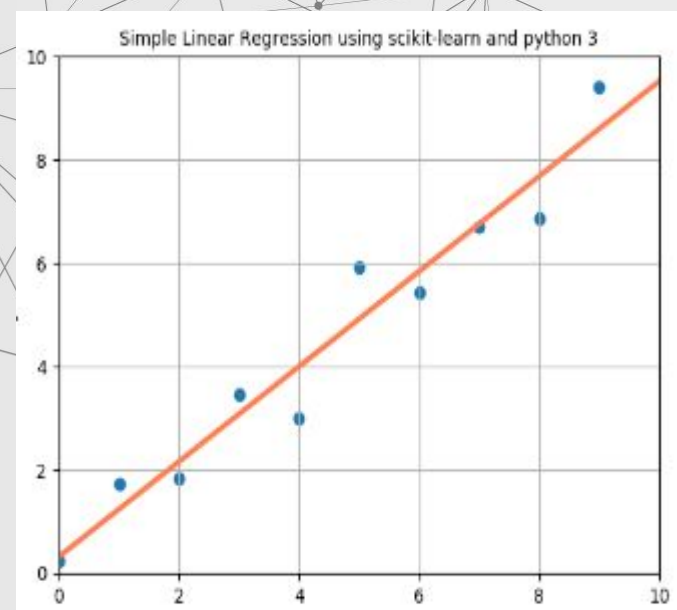
```
X = np.asarray(X)
```

```
Y = np.asarray(Y)
```

```
X = X[:,np.newaxis]
```

```
Y = Y[:,np.newaxis]
```

```
plt.scatter(X,Y)
```



Régression linéaire multiple avec le module scikit learn :

```
import pandas as pd  
from sklearn import linear_model  
import statsmodels.api as sm
```





06

CONCLUSION
