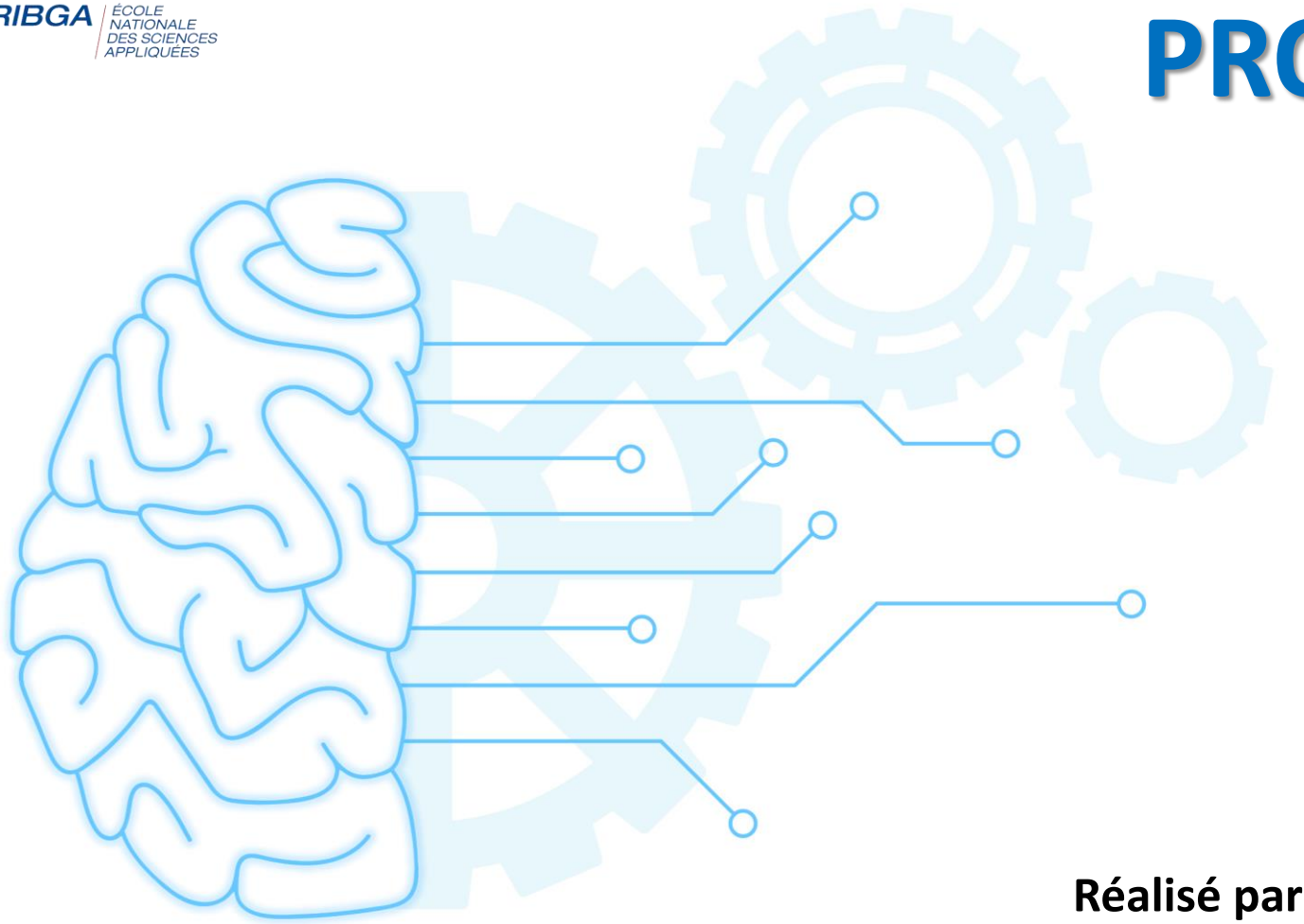
ENSA KHOURIBGA ÉCOLE NATIONALE DES SCIENCES APPLIQUÉES

PROJET ACADEMIQUE



Modèle de prédiction des prix des ordinateurs en utilisant la régression linéaire multiple.

IID2
Le 08/05/2023

Réalisé par :
Chaimaa KHALIL
Siham HAFSI
Sofia FENNICH
Marouane SENNAH

Encadré par :
Pr. Abdelghani GHAZDALI
Pr. Hamza KHALFI

Sommaire

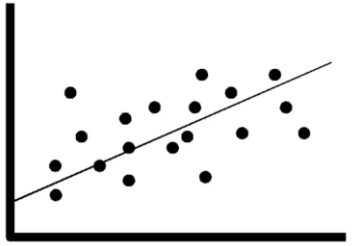
Introduction

Le Machine Learning est une branche de l'intelligence artificielle qui permet aux ordinateurs d'apprendre à partir de données sans être explicitement programmés.
Notre projet académique consiste à utiliser le Machine Learning pour développer un modèle de prédiction des prix des ordinateurs en utilisant la régression linéaire multiple.
Pour atteindre cet objectif, deux algorithmes d'optimisation différents ont été comparés :
La descente de gradient (GD) et la descente de gradient stochastique (SGD).
L'optimisation avec ces algorithmes permettra de trouver les coefficients optimaux du modèle de la régression linéaire multiple.
Ces coefficients représentent les poids associés à chaque caractéristique de l'ordinateur qui permettent de prédire son prix.

Régression linéaire multiple

En **machine Learning**, la **régression linéaire multiple** est utilisée comme méthode d'apprentissage supervisé pour prédire des valeurs continues à partir d'un ensemble de données d'entraînement. Elle est couramment utilisée pour la prédiction de prix, de quantités, de scores, etc.

Le processus d'apprentissage dans la **régression linéaire multiple** consiste à trouver les coefficients optimaux de l'équation linéaire en minimisant l'erreur quadratique moyenne entre les valeurs prédites et les valeurs réelles dans l'ensemble de données d'entraînement.



Régression linéaire multiple

Récolter des données (X, y)

01

Donner à la machine un modèle linéaire $F(X) = X.\theta$

02

Créer la Fonction Coût

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

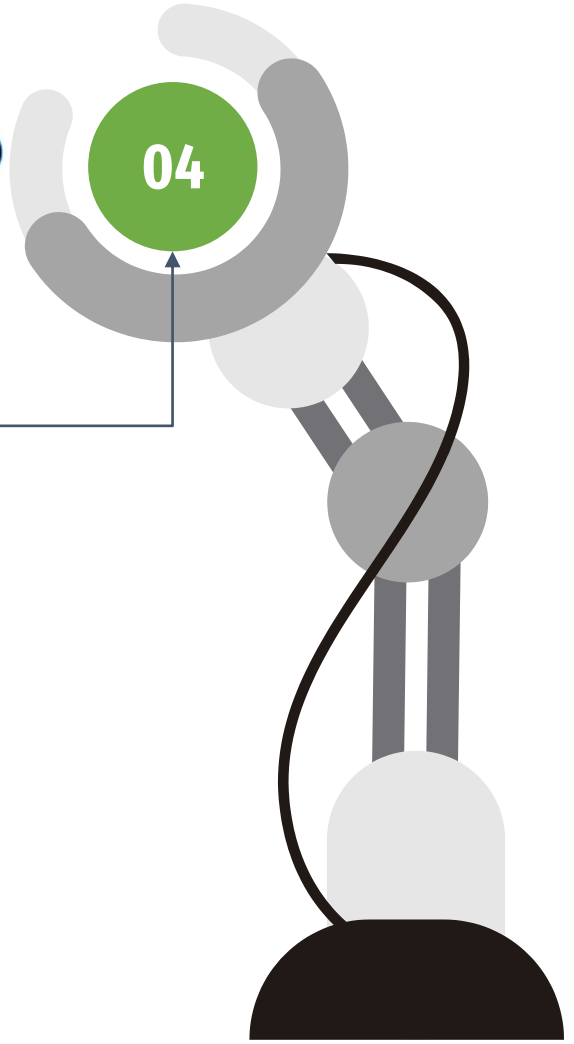
03

Calculer le gradient

$$\textbf{Gradient}: \frac{\partial J(\theta)}{\partial \theta} = \frac{1}{m} X^T . (F(X) - Y)$$

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

04



Pour atteindre nos objectifs, deux algorithmes d'optimisation différents ont été comparés

01

La descente de gradient (GD)

02

la descente de gradient stochastique (SGD)

La descente de gradient (GD)

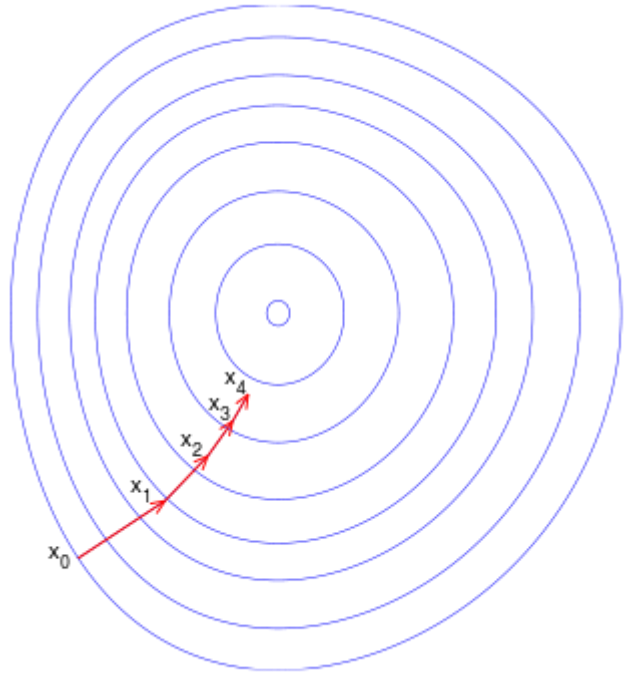
Considérez le problème :

$$\min_x \frac{1}{n} \sum_{i=1}^{n} f_i(x)$$

Rappelons que la descente de gradient se déroule comme suit :

$$x^{(k)} = x^{(k-1)} - t_k \cdot \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(x^{(k-1)})$$

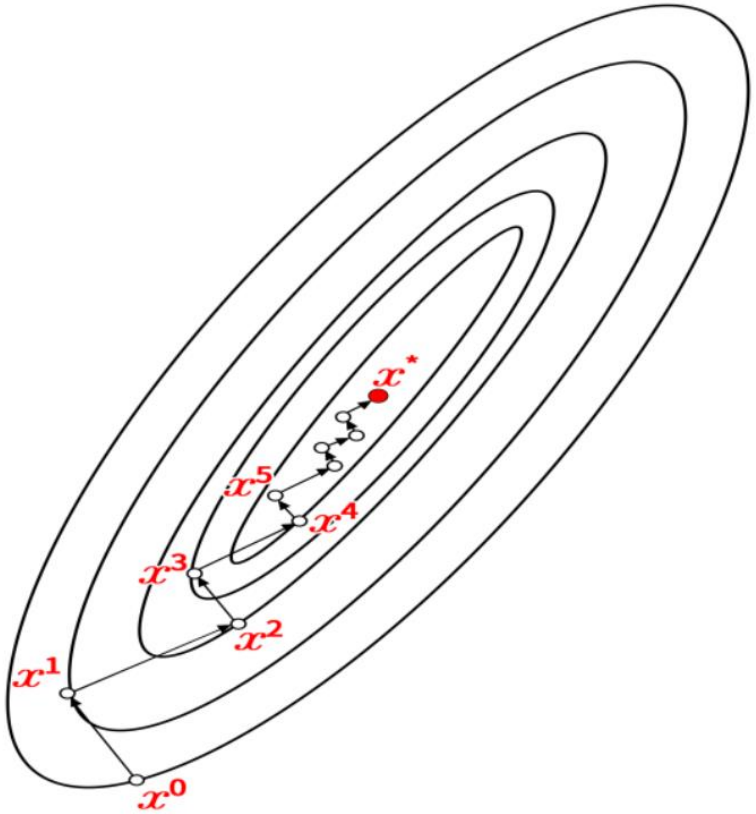
La descente de gradient est plus appropriée lorsque l'ensemble des données est relativement petit et que le calcul de la dérivée de la fonction de coût est réalisable en un temps raisonnable.



la descente de gradient stochastique (SGD)

La descente de gradient stochastique remplace le gradient moyen par le gradient d'un gradient choisi au hasard.

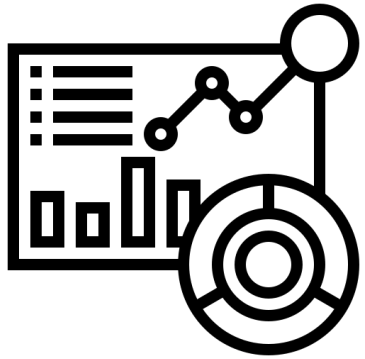
$$x^{(k)} = x^{(k-1)} - t_k \cdot \nabla f_{i_k}(x^{(k-1)})$$



la descente de gradient stochastique est souvent préférable pour les ensembles de données volumineux, car elle peut être calculée plus rapidement sur des échantillons de données plus petits.

Dataset

laptop-data.csv



Un ensemble de données disponible sur Kaggle, qui contient des informations sur des ordinateurs portables de différentes marques, modèles et configurations. Les données ont été collectées à partir de différentes sources en ligne, telles que des sites d'e-commerce et des sites de fabricants d'ordinateurs portables.

Le dataset comprend 14 colonnes :

Company – Product – TypeName – Inches – CPU – RAM – Memory – GPU – OpSys – Weight – Price – dtype – Speed.

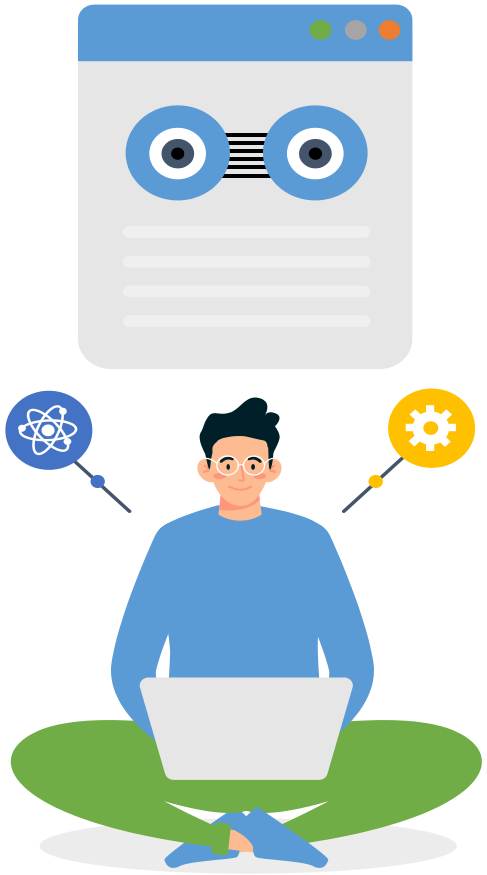
	Unnamed: 0	Company	TypeName	Inches	ScreenResolution	Cpu	Ram	Memory	Gpu	OpSys	Weight	Price
0	0	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 2.3GHz	8GB	128GB SSD	Intel Iris Plus Graphics 640	macOS	1.37kg	71378.6832
1	1	Apple	Ultrabook	13.3	1440x900	Intel Core i5 1.8GHz	8GB	128GB Flash Storage	Intel HD Graphics 6000	macOS	1.34kg	47895.5232
2	2	HP	Notebook	15.6	Full HD 1920x1080	Intel Core i5 7200U 2.5GHz	8GB	256GB SSD	Intel HD Graphics 620	No OS	1.86kg	30636.0000
3	3	Apple	Ultrabook	15.4	IPS Panel Retina Display 2880x1800	Intel Core i7 2.7GHz	16GB	512GB SSD	AMD Radeon Pro 455	macOS	1.83kg	135195.3360
4	4	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 3.1GHz	8GB	256GB SSD	Intel Iris Plus Graphics 650	macOS	1.37kg	96095.8080

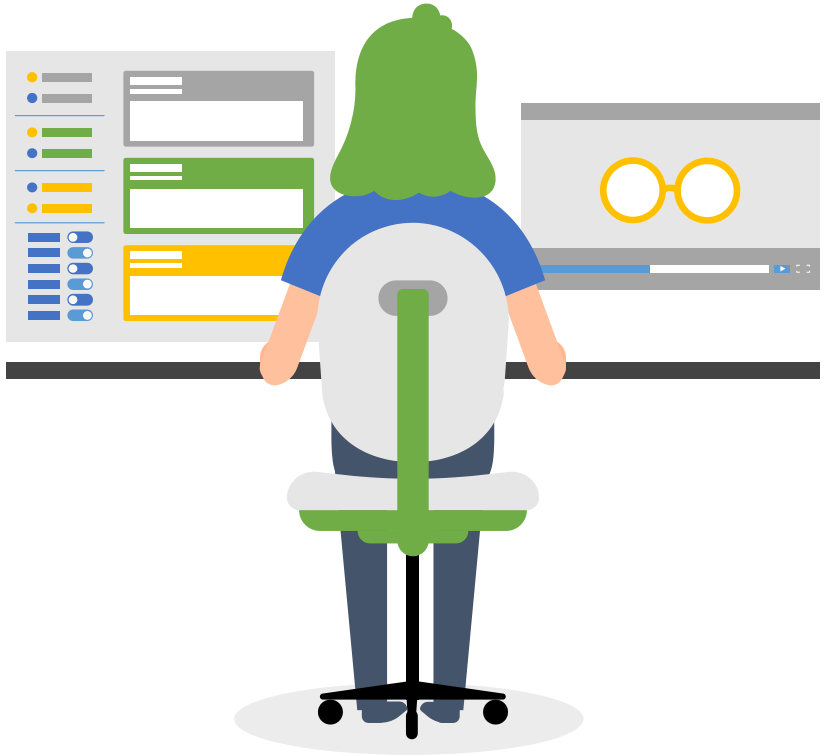
Après le processus de nettoyage nous allons travailler avec ces champs :

	Ram	Weight	Touchscreen	ppi	HDD	SSD
0	8	1.37	0	226.983005	0	128
1	8	1.34	0	127.677940	0	0
2	8	1.86	0	141.211998	0	256
3	16	1.83	0	220.534624	0	512
4	8	1.37	0	226.983005	0	256
...	...	...	...	...	...	...
1298	4	1.80	1	157.350512	0	128
1299	16	1.30	1	276.053530	0	512
1300	2	1.50	0	111.935204	0	0
1301	6	2.19	0	100.454670	1000	0
1302	4	2.20	0	100.454670	500	0

1302 rows × 6 columns

Implémentation du modèle :





```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import random
x= pd.read_csv('feathers.csv')
y= pd.read_csv('label.csv')
# Conversion en tableaux numpy
X = x.values
y = y.values
w = np.zeros((6,))
b = 0
```

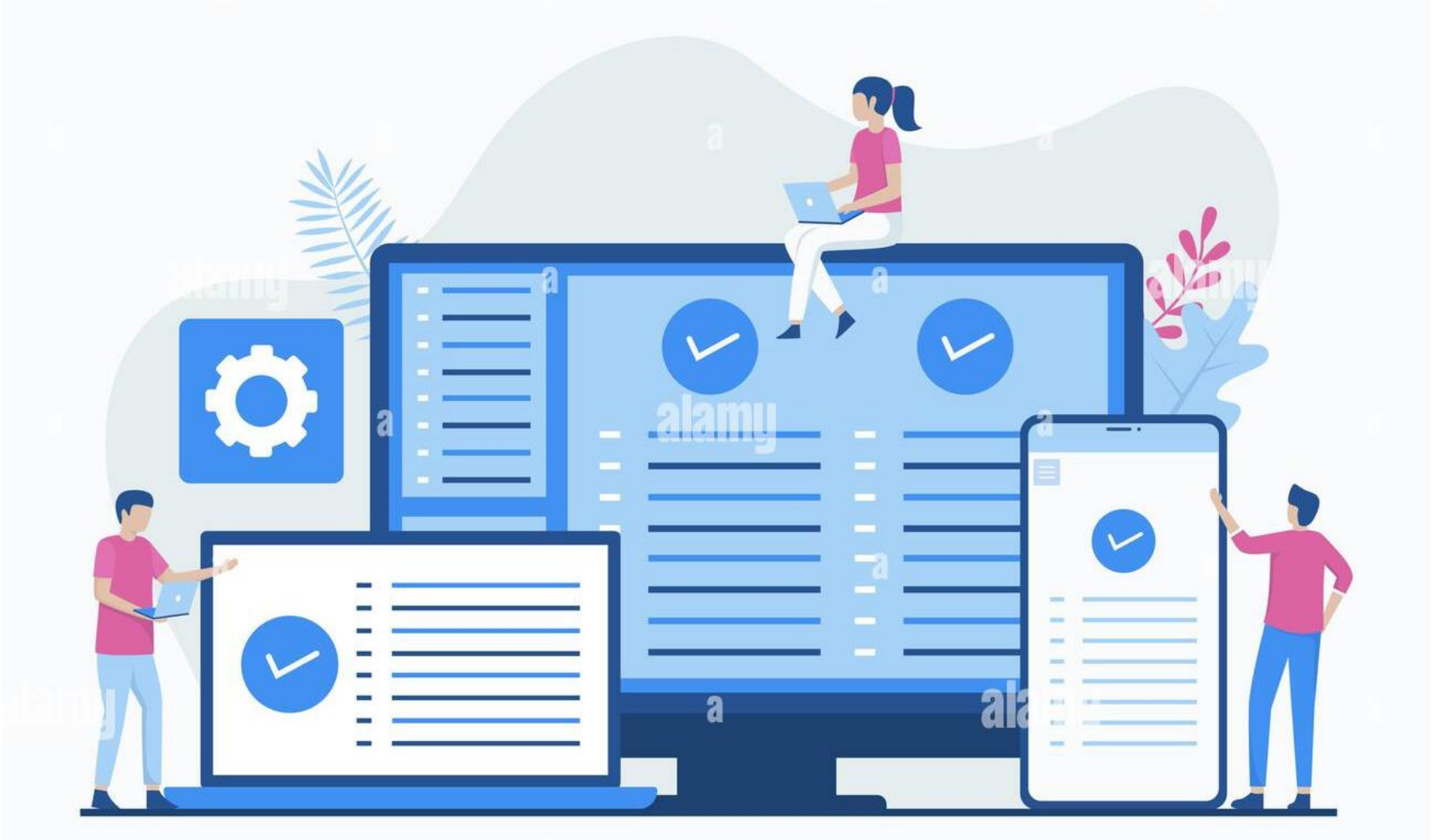
```python
def cost(X, y, w, b):
    m = len(y)
    total_cost = 0.0
    for i in range(m):
        f_wb_i = np.dot(X[i], w) + b
        total_cost += (f_wb_i - y[i])**2
    cost = total_cost / (2*m)
    return cost
```

GD VS SGD

```python
def compute_gradient(X, y, w, b):
    m,n = X.shape
    dj_dw = np.zeros((n,))
    dj_db = 0.

    for i in range(m):
        err = (np.dot(X[i], w) + b) - y[i]
        for j in range(n):
            dj_dw[j] = dj_dw[j] + err * X[i, j]
        dj_db = dj_db + err
    dj_dw = dj_dw / m
    dj_db = dj_db / m

    return dj_db, dj_dw

# Descente de gradient
alpha = 0.000000005
tolerance = 1e-6
max_iter = 10000
costs = []
iteration = 0
while iteration < max_iter:
    dj_db, dj_dw = compute_gradient(X, y, w, b)
    w = w - alpha * dj_dw
    b = b - alpha * dj_db
    cost_i = cost(X, y, w, b)
    costs.append(cost_i)
    if len(costs) > 1 and np.abs(costs[-1] - costs[-2]) < tolerance:
        break
    iteration += 1
```

```python
def compute_gradient_stochastique(X, y, w, b, i):
    dj_dw = np.zeros((len(w),))
    dj_db = 0.
    err = (np.dot(X[i], w) + b) - y[i]
    for j in range(len(w)):
        dj_dw[j] = err * X[i, j]
    dj_db = err

    return dj_db, dj_dw

# Descente de gradient stochastique

alpha = 0.000000005
tolerance = 1e-6
max_iter = 100000

costs = []
iteration = 0
while iteration < max_iter:
    shuffled_indices = list(range(len(y)))
    random.shuffle(shuffled_indices)
    for i in shuffled_indices:
        dj_db, dj_dw = compute_gradient_stochastique(X, y, w, b, i)
        w = w - alpha * dj_dw
        b = b - alpha * dj_db
    cost_i = cost(X, y, w, b)
    costs.append(cost_i)
    if len(costs) > 1 and np.abs(costs[-1] - costs[-2]) < tolerance:
        break
    iteration += 1
```

Plateforme



Merci de votre attention