Université IBN TOFAIL – Kénitra
ECOLE NATIONALE DES SCIENCES APPLIQUEES

Cours Intelligence Artificielle

CH3: Machine Learning - Apprentissage Supervisé: Régressions

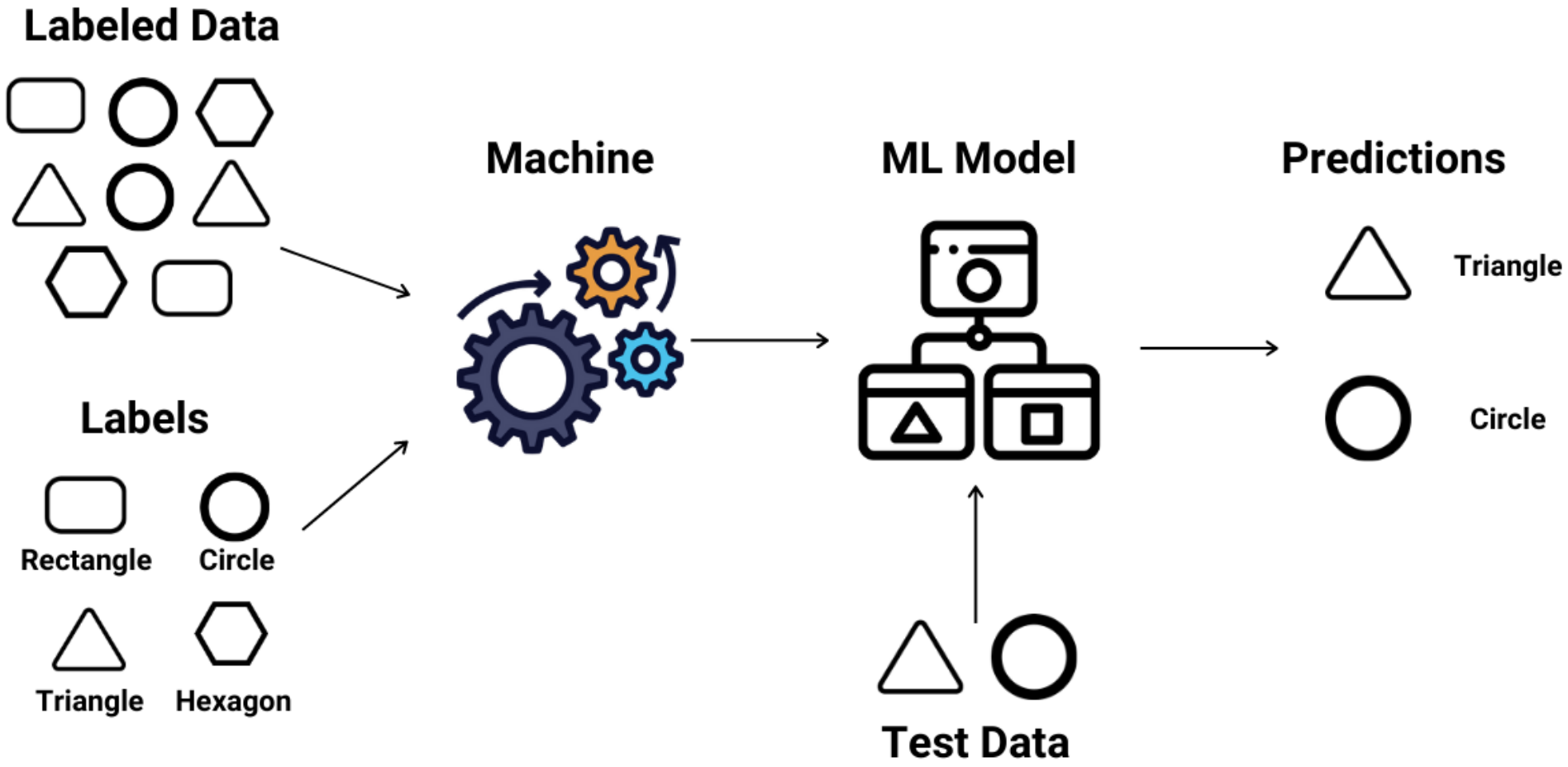
Par:

Hassan EL FADIL

elfadil.h@gmail.com

1. Définition de l'Apprentissage Supervisé

- L'apprentissage supervisé (Supervised Learning) consiste à apprendre à partir d'exemples connus pour faire des prédictions ou des classifications sur des données nouvelles.
- Les algorithmes apprennent à partir d'un ensemble de données étiquetées.
- Chaque exemple d'entraînement a une entrée et une sortie connue (label ou cible).
- **Exemples** :
  - *Prédire le prix d'une maison (régression).*
  - *Identifier si un e-mail est un spam ou non (classification).*
- **Algorithmes courants** :
  - *Régression linéaire, Régression logistique.*
  - *Arbres de décision.*
  - *Machines à Vecteurs de Support (SVM: Support Vector Machine).*
  - *Forêts aléatoires (Random Forest).*


Labeled Data
Labels
Rectangle
Circle
Triangle
Hexagon
Machine
ML Model
Predictions
Triangle
Circle
Test Data

2. Régression linéaire

2.1. Exemple introductif:

Considérons le petit tableau ci-contre qui indique le salaire d'un employé dans une entreprise en fonction des années d'expérience:

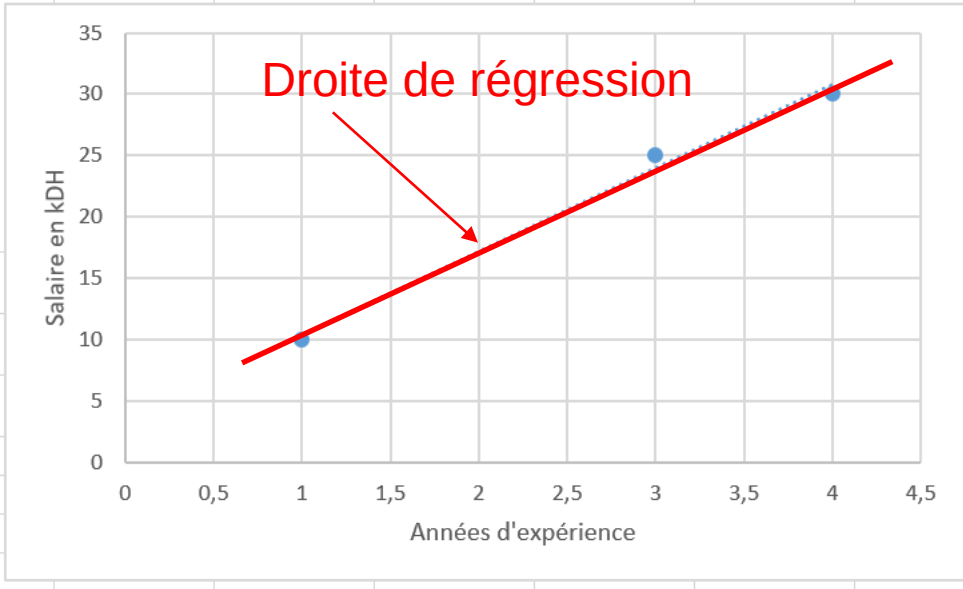
Question: Que serait le salaire pour 6 ans ?

Années d'expérience (x)	Salaire (y) en kDH
1	10
3	25
4	30

Idée:

Chercher la meilleure droite $y = ax + b$ (appellée droite de régression) qui :

- Passe au plus près de l'ensemble des points.
- Minimise l'erreur quadratique.



2.2. Détermination des coefficients a et b:

On cherche à trouver les valeurs de ***a*** et ***b*** qui minimisent l'**erreur quadratique totale**, c'est-à-dire :

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y_i})^2 = \frac{1}{N}\sum_{i=1}^{N}(y_i - (ax_i + b))^2$$

Les valeurs optimales de ***a*** et ***b*** peuvent être calculées avec les formules suivantes :

$$a = \frac{N\sum(x_i y_i) - \sum x_i \sum y_i}{N\sum x_i^2 - (\sum x_i)^2}$$

$$b = \frac{\sum y_i - a\sum x_i}{N}$$

x	y
1	10
3	25
4	30

- $\sum x_i = 1 + 3 + 4 = 8$
- $\sum y_i = 10 + 25 + 30 = 65$
- $\sum x_i^2 = 1 + 9 + 16 = 26$
- $\sum (x_i y_i) = 10 + 75 + 120 = 205$
- $N = 3$

$$a = \frac{N\sum(x_i y_i) - \sum x_i \sum y_i}{N\sum x_i^2 - (\sum x_i)^2} = \frac{3 \times 205 - 8 \times 65}{3 \times 26 - 8^2} = 6{,}7857$$

$$b = \frac{\sum y_i - a\sum x_i}{N} = \frac{65 - 6{,}785714286 \times 8}{3} = 3{,}5714$$

Le salaire estimé y en fonction de nombre d'années d'expérience x s'écrit alors comme suit:

$$y = 6{,}7857x + 3{,}5714 \text{ (en kDH)}$$

Pour une ancienneté de 6 ans le salaire estimé est:

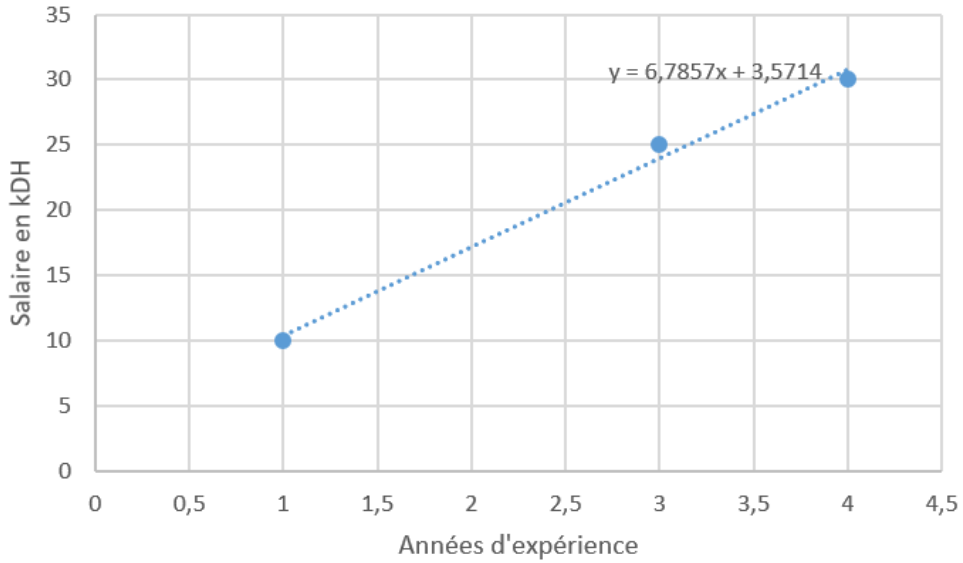
$$y = 6{,}7857 \times 6 + 3{,}5714 = 44{,}28 \text{ kDH}$$

2.3. Approche automatique avec Python ou Excel:

Pour des jeux de données plus grands, vous pouvez utiliser des outils comme **Python** (avec la bibliothèque **numpy** ou **sklearn**) ou **Excel** pour effectuer ces calculs rapidement.

2.3.1. Sous Excel:

- Créer le tableau
- Crée un graphique: Insertion > Graphiques > Nuage de points.
- Ajouter une courbe de tendance (ou faites un clic droit > "Ajouter une courbe de tendance").
- Dans le menu qui apparaît :Choisissez Linéaire comme type de courbe.
- Cochez l'option Afficher l'équation sur le graphique pour voir l'équation de la ligne.



- **Formule pour la pente a :** fonction **PENTE** . Dans une cellule **=PENTE(B2:B4; A2:A4)**
- **Formule pour l'ordonnée à l'origine b :** fonction **ORDONNEE.ORIGINE** **=ORDONNEE.ORIGINE(B2:B4, A2:A4)**
- où : B2:B4 correspond aux y.; A2:A4 correspond aux x.

2.3.2. Sous Python:

- **Jupyter Notebook** est un environnement interactif de développement open source qui permet d'écrire, exécuter et documenter du code dans des **notebooks** (carnets de notes) à la fois pour des projets de **science des données**, de **machine learning** et d'autres applications de programmation en **Python** (et d'autres langages).
- Peut être utilisé localement en installant **Anaconda** : Allez sur le site officiel d'Anaconda : https://www.anaconda.com/products/distribution
- On peut également utiliser des notebooks interactifs en ligne (tout et dans le cloud) pour exécuter le code Python en utilisant **Google Colab** : https://colab.research.google.com/
- C'est cette dernière option que nous allons utiliser pour exécuter le code Python relatif à la régression linéaire. Il faut disposer d'un compte **Gmail**.

2.4. Application 1: Régression linéaire pour « loyer des maisons »:

1. **Préparation du fichier. csv : Créer le tableau Excel ci-contre. Nommer le: LoyersMaisons**

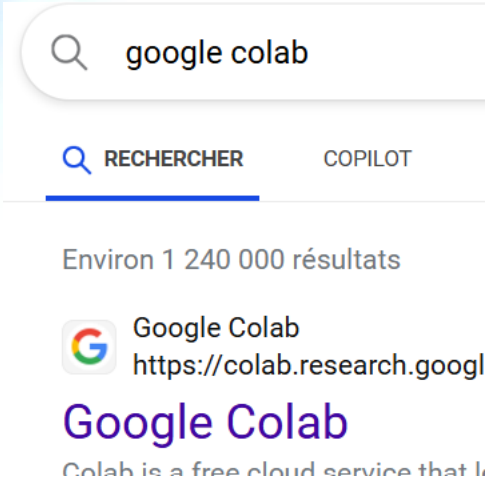
Surface	Loyer
50	900
55	950
60	1000
65	1100
70	1150
75	1200
80	1250
85	1300
90	1350
95	1400
100	1500
105	1600
110	1700
115	1750
120	1800
125	1850
130	1900
135	1950
140	2000
145	2050

Surface	Loyer
150	2100
155	2150
160	2200
165	2250
170	2300
175	2350
180	2400
185	2450
190	2500
195	2550
200	2600
205	2650
210	2700
215	2750
220	2800
225	2850
230	2900
235	2950
240	3000
245	3050

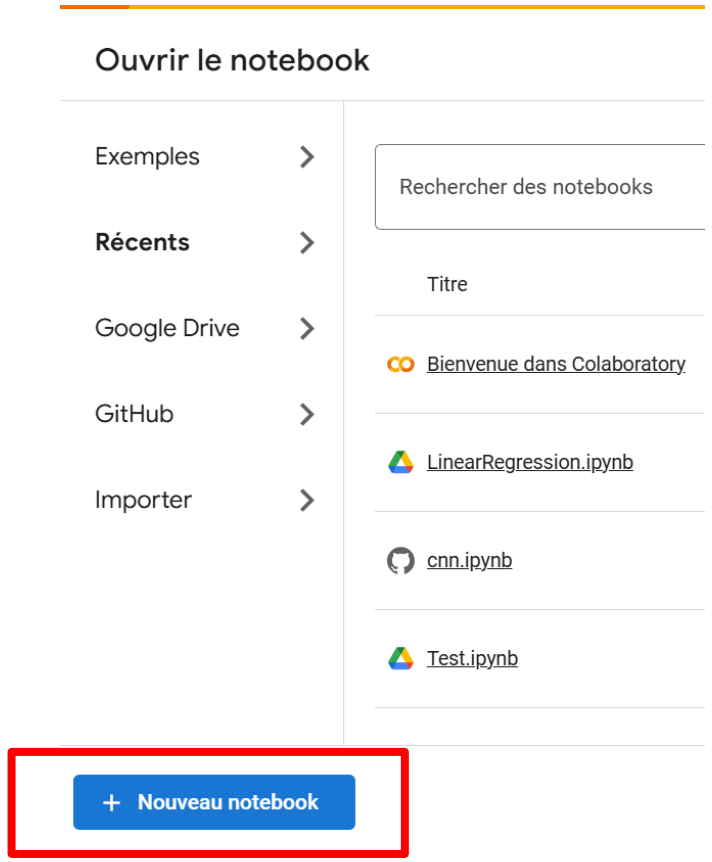
Surface	Loyer
250	3100
255	3150
260	3200
265	3250
270	3300
275	3350
280	3400
285	3450
290	3500
295	3550
300	3600

2 Ouvrir Google Colab et créer un Nouveau Notebook:

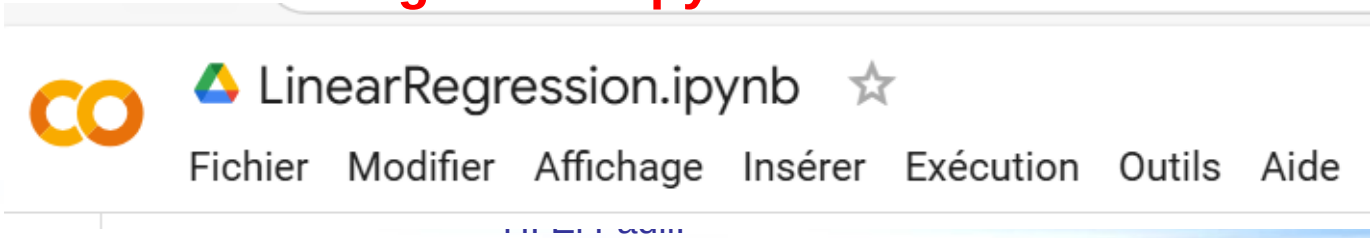
Voir comment utiliser Google Colab:
https://www.youtube.com/watch?v=SRPCqXcuxKo



Cliquer sur Nouveau notebook



3 Renommer le Notebook: LinearRegression.ipynb

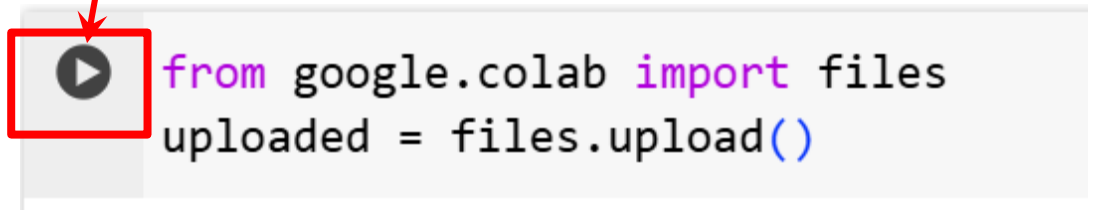


4 Charger le fichier .csv directement depuis votre ordinateur dans Colab

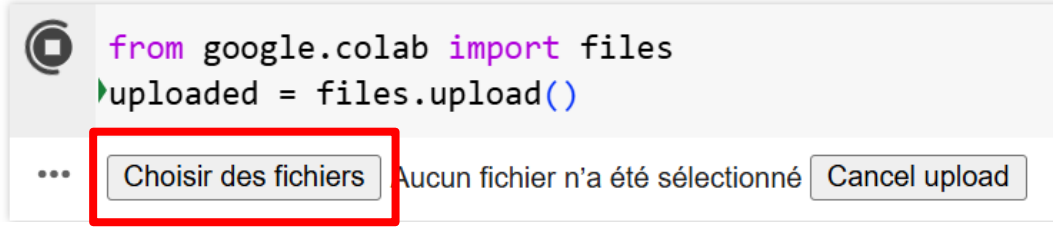
- Pour cela ajoutez la cellule suivante dans votre notebook Colab pour téléverser le fichier :

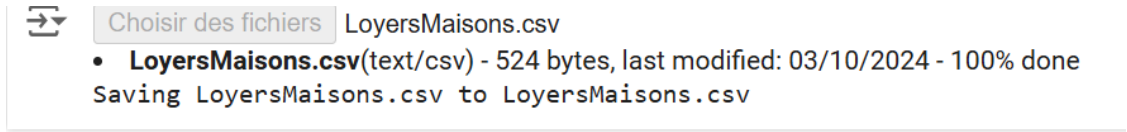
```
from google.colab import files
uploaded = files.upload()
```

Exécuter le code



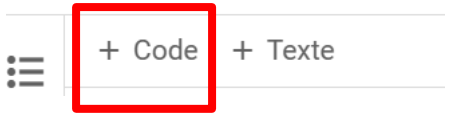
- Une boîte de dialogue s'ouvrira pour que vous sélectionniez LoyersMaisons.csv depuis votre ordinateur.
- Une fois téléversé, le fichier sera disponible temporairement dans l'espace de travail de Colab.





5 Charger et visualiser les données

- Ajoutez une cellule pour charger et afficher les données avec pandas :




```
import pandas as pd
# Charger le fichier CSV
data = pd.read_csv('LoyersMaisons.csv', sep=";")
# Afficher les données du fichiers
print(data)
```

```
import pandas as pd
# Charger le fichier CSV
data = pd.read_csv('LoyersMaisons.csv', sep=";")
# Afficher les données du fichiers
print(data)
```

```
   Surface  Loyer
0       50    900
1       55    950
2       60   1000
3       65   1100
4       70   1150
5       75   1200
6       80   1250
7       85   1300
8       90   1350
```

Remarque:

- Dans le cas où le fichier existe déjà dans le Drive utiliser la commande suivante:

```
data=pd.read_csv('/content/drive/MyDrive/Colab Notebooks/LoyersMaisons.csv',sep=";")
```

6 Préparer les données pour la régression

```
# Variable indépendante : Surface
x = data[['Surface']]
# Variable dépendante : Loyer
y = data['Loyer']
```

0 s

```
# Variable indépendante : Surface
X = data[['Surface']]
# Variable dépendante : Loyer
y = data['Loyer']
```

7 Importer les bibliothèques pour la régression:

Ajoutez une cellule pour importer les outils nécessaires :

```
# Import Librairies
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

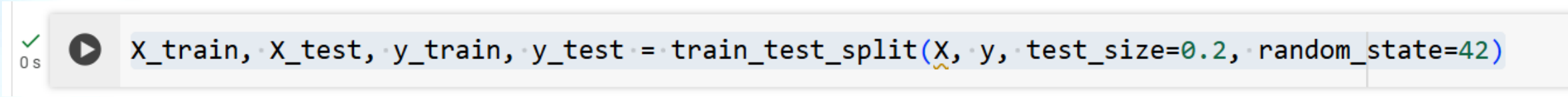
4 s

```
# Import Librairies
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

7 Diviser les données en ensembles d'entraînement et de test :

- Créez un ensemble d'entraînement et un ensemble de test :

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```



- Divise les données de manière aléatoire en deux sous-ensembles.
- 80 % des données vont dans l'ensemble d'entraînement (X_train, y_train).
- 20 % des données vont dans l'ensemble de test (X_test, y_test).
- La division est répétable grâce à random_state=42.

8 Effectuer la régression linéaire :

- Entraînez un modèle de régression linéaire sur les données :

```
# Initialiser le modèle
model = LinearRegression()
# Entraîner le modèle
model.fit(X_train, y_train)
# Coefficients du modèle
print(f"Coefficient a (pente) : {model.coef_[0]}")
print(f"Ordonnée à l'origine b : {model.intercept_}")
```

```
[6]  # Initialiser le modèle
     model = LinearRegression()
     # Entraîner le modèle
     model.fit(X_train, y_train)
     # Coefficients du modèle
     print(f"Coefficient a (pente) : {model.coef_[0]}")
     print(f"Ordonnée à l'origine b : {model.intercept_}")

     Coefficient a (pente) : 10.717714940098661
     Ordonnée à l'origine b : 434.5853153629316
```

9 Tester et évaluer le modèle :

- Faites des prédictions sur l'ensemble de test et évaluez la performance :

```
# Faire des prédictions
y_pred = model.predict(X_test)
# Calculer l'erreur quadratique moyenne (MSE)
mse = mean_squared_error(y_test, y_pred)
print(f"Erreur quadratique moyenne : {mse}")
```

```
[14] # Faire des prédictions
     y_pred = model.predict(X_test)
     # Calculer l'erreur quadratique moyenne (MSE)
     mse = mean_squared_error(y_test, y_pred)
     print(f"Erreur quadratique moyenne : {mse}")
```

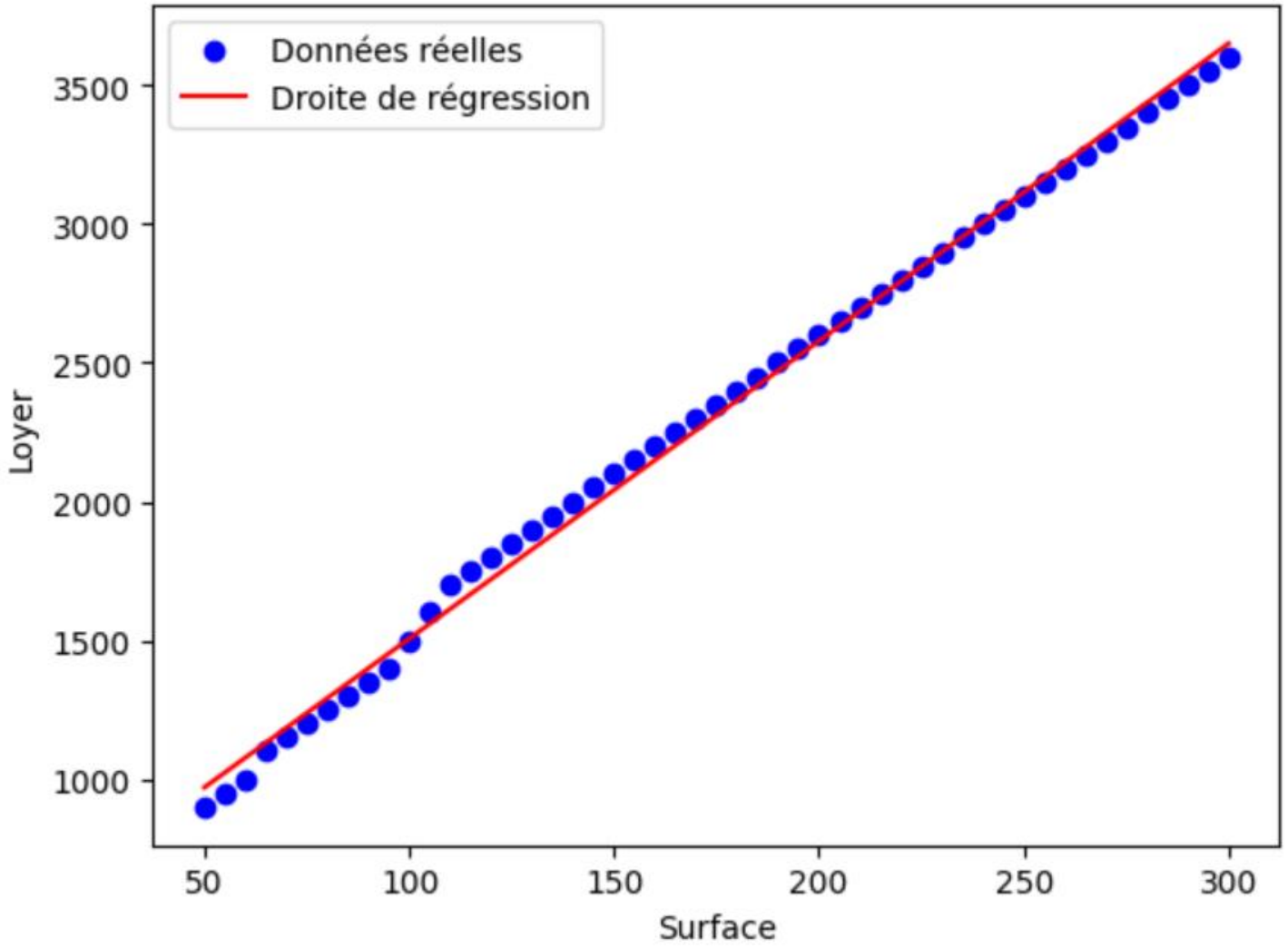
```
Erreur quadratique moyenne : 2272.9574315512086
```

10 Visualiser les résultats :

- Ajoutez une cellule pour tracer les points et la droite de régression :

```
import matplotlib.pyplot as plt
# Visualiser les données
plt.scatter(X, y, color='blue', label='Données réelles')
# Tracer la droite de régression
plt.plot(X, model.predict(X), color='red', label='Droite de
régression')
plt.xlabel('Surface')
plt.ylabel('Loyer')
plt.legend()
plt.show()
```

…Application 1: Régression linéaire



3. Régression logistique

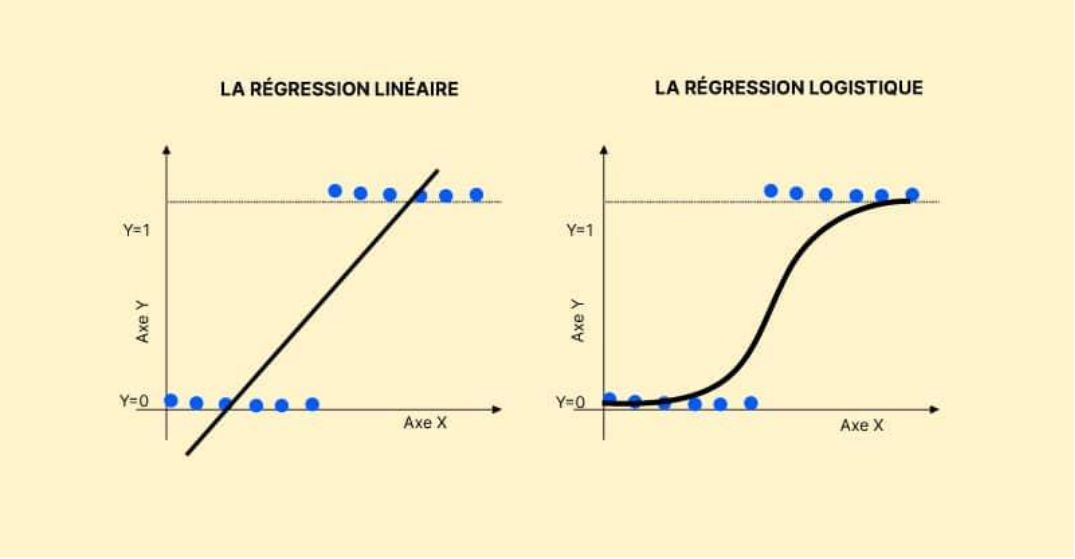
3.1. Introduction:

- La régression logistique est une méthode statistique pour prédire une variable catégorielle.
- **Exemple** :
  - Prédire si un patient est malade (1) ou en bonne santé (0).
  - Email spam ou pas spam.
  - Transaction frauduleuse ou non, …
- **Applications courantes** :
  - Diagnostic médical
  - Analyse marketing
  - Détection de fraude

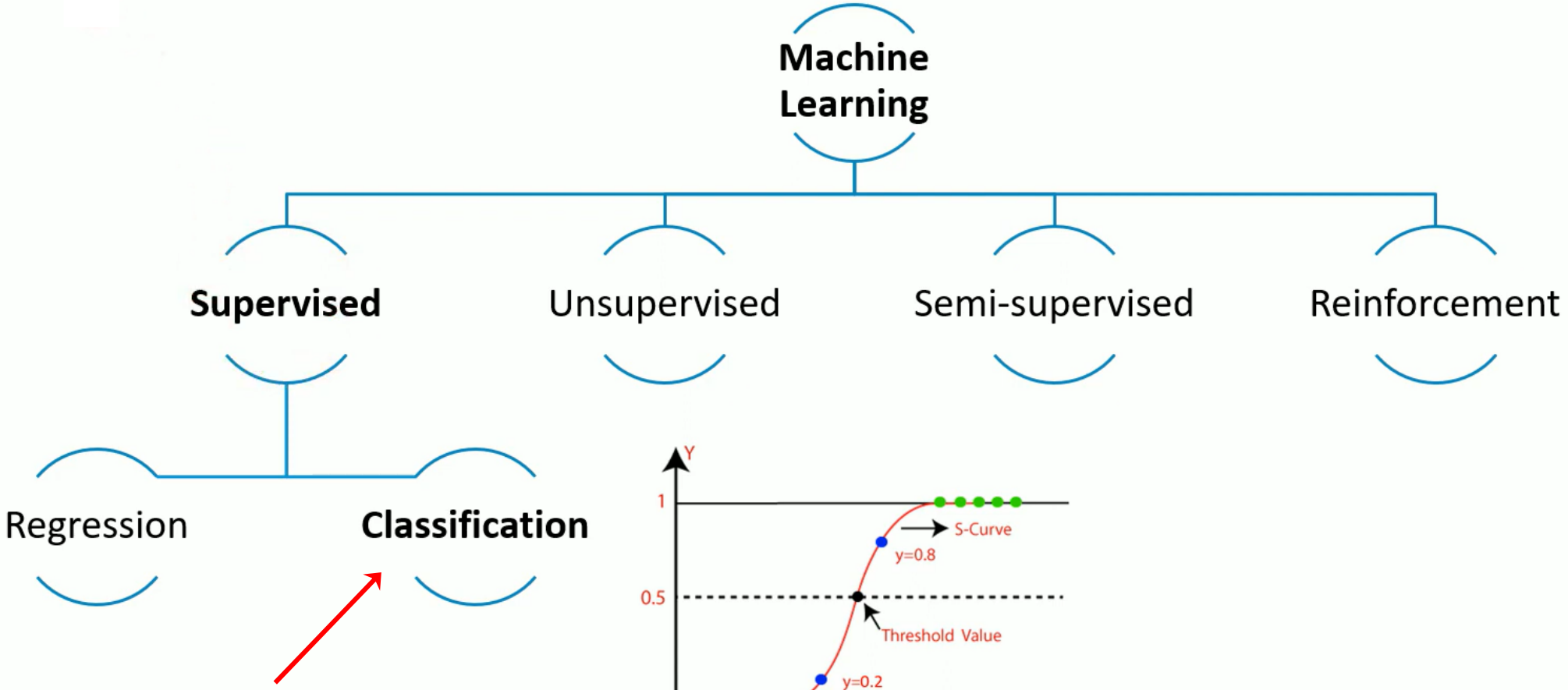
3.2. Différences avec la régression linéaire:

- Régression Linéaire :
  - Prédit des valeurs continues
  - Basée sur une fonction linéaire
  - Résultats non bornés

- Régression Logistique :
  - Prédit des probabilités (0 ou 1)
  - Utilise une fonction sigmoïde
  - Résultats bornés entre [0, 1]



… Régression logistique



Régressions Logistique: fait partie des algorithmes de classification binaire.

3.3. Fonction Sigmoïde:

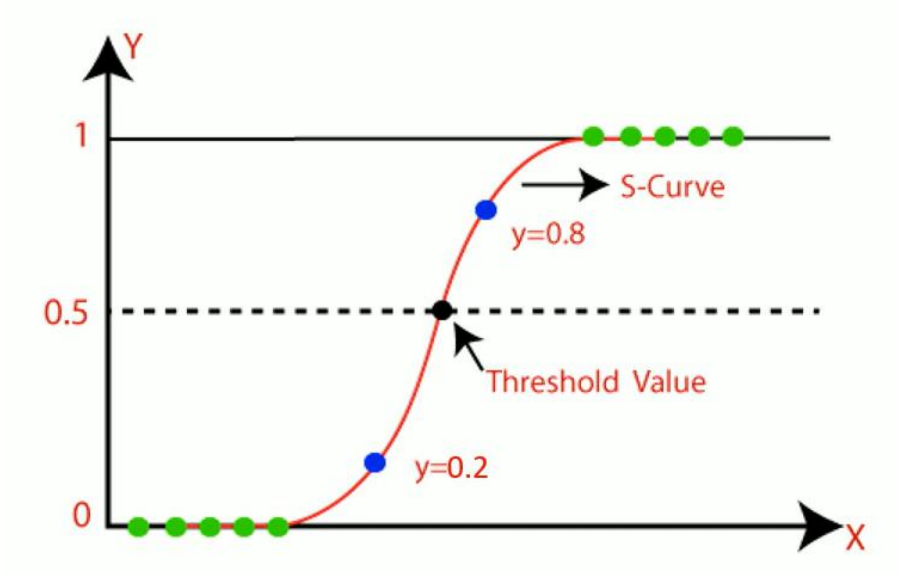
- Formule :

$$h_\theta(x) = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x)}}$$

Cette fonction produit une probabilité $h_\theta(x) \in [0, 1]$, utilisée pour prédire *y*

- But de l'algorithme de ML: Trouver les coefficients $\theta_0 \; et \; \theta_1$

- Si $h_\theta(x) > 0.5$ y = 1 (classe positive).
- Sinon, y = 0 (classe négative).

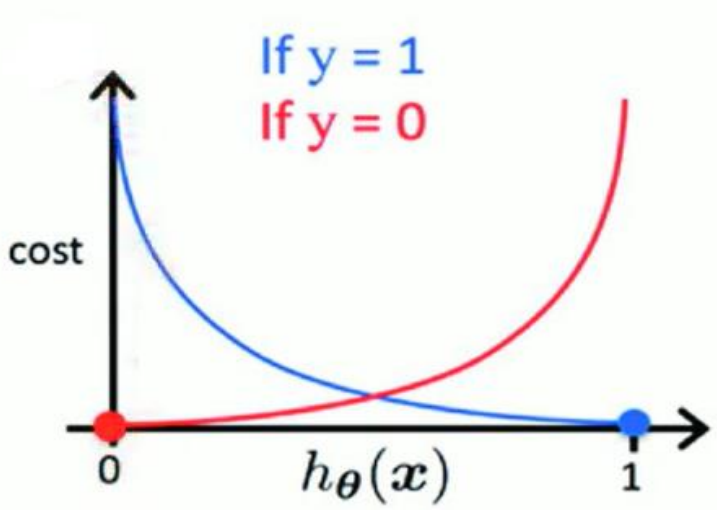


3.4. Fonction coût (Cost function):

- La régression logistique utilise la **log-vraisemblance** :

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right)$$

$$Cost(h_\theta(x), y) = \begin{cases} -\log\left(h_\theta(x)\right) & if\ y = 1 \\ -\log\left(1 - h_\theta(x)\right) & if\ y = 0 \end{cases}$$



- m est le nombre total d'exemples
- $y^{(i)}$ est la valeur réelle de l'exemple i. Classification binaire: $y^{(i)}$ prend uniquement les valeurs 0 ou 1
- $h_\theta(x)$ est la prédiction pour $x^{(i)}$.

- **But** : Minimiser J(θ) pour ajuster les paramètres θ_0 *et* θ_1.

3.5. Détermination des paramètres θ_0 et θ_1:

- Utilisation de l'algorithme Descente de gradient : « Gradient Descent »

1) Initialisation :

- Fixer des valeurs initiales pour θ_0 et θ_1 (souvent $\theta_0 = 0$ et $\theta_1 = 0$).
- Choisir un **taux d'apprentissage (learning rate)** α (une petite valeur, comme 0.01).

2) Calculer le gradient :

- Les gradients pour θ_0 et θ_1 sont donnés par :

$$\frac{\partial J(\theta)}{\partial \theta_0} = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)$$

$$\frac{\partial J(\theta)}{\partial \theta_1} = \frac{1}{m}\sum_{i=1}^{m}\left(\left(h_\theta(x^{(i)}) - y^{(i)}\right)\cdot x^{(i)}\right)$$

3) Mettre à jour θ_0 et θ_1 :

- En utilisant les formules suivantes :

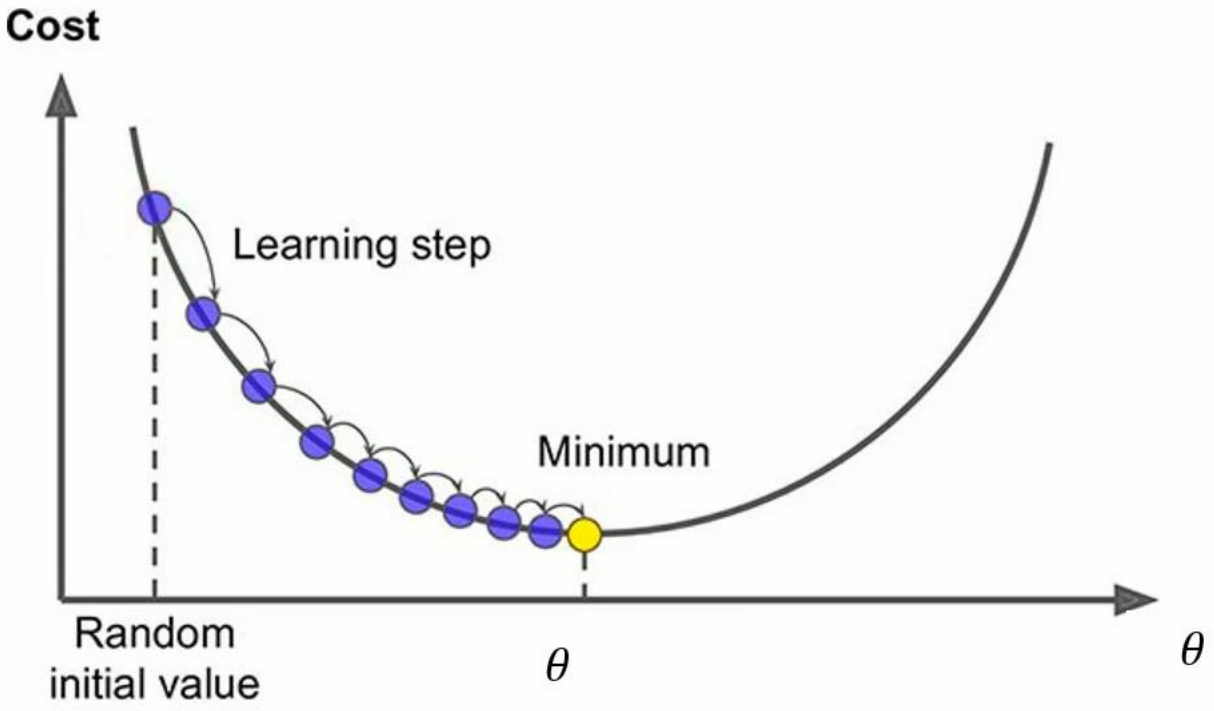
$$\theta_0 \leftarrow \theta_0 - \alpha\frac{\partial J(\theta)}{\partial \theta_0}$$

$$\theta_1 \leftarrow \theta_1 - \alpha\frac{\partial J(\theta)}{\partial \theta_1}$$

4) Répéter les étapes 2 et 3:

- Répéter les calculs du gradient et les mises à jour des paramètres jusqu'à ce que la convergence soit atteinte (i.e., les changements dans $J(\theta)$ sont minimes).

… Régression logistique



Autres Algorithmes possibles:

- **Conjugate gradient**
- **BFGS**: Broyden-Fletcher-Goldfarb-Shanno
- **L-BFGS**: Limited memory - BFGS

3.6. Application 2: Régression logistique pour « Fleurs »:

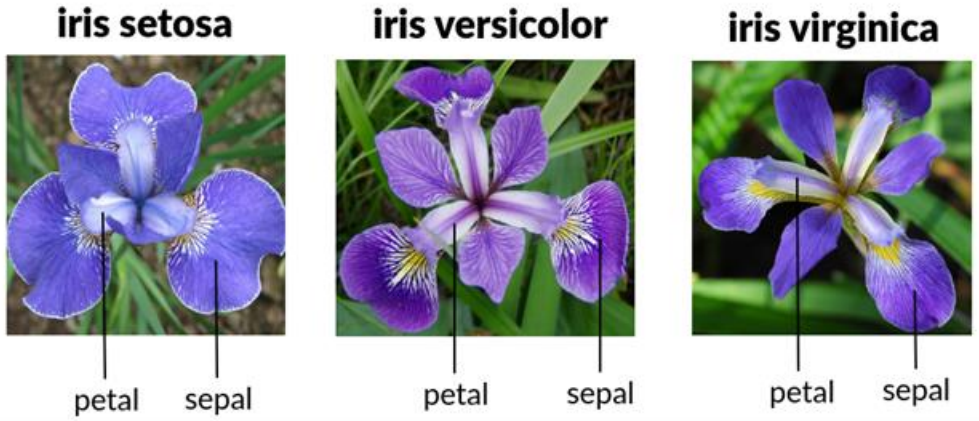
3.6.1. Définition du Problème :

- **Objectif** : Classifier les fleurs en deux catégories (setosa ou versicolor) à partir de leurs caractéristiques (longueur et largeur des sépales).
- **Question clé** : Peut-on prédire le type de fleur à partir des données disponibles ?
- **Type de problème** : Classification binaire.

3.6.2. Collecte et Exploration des Données :

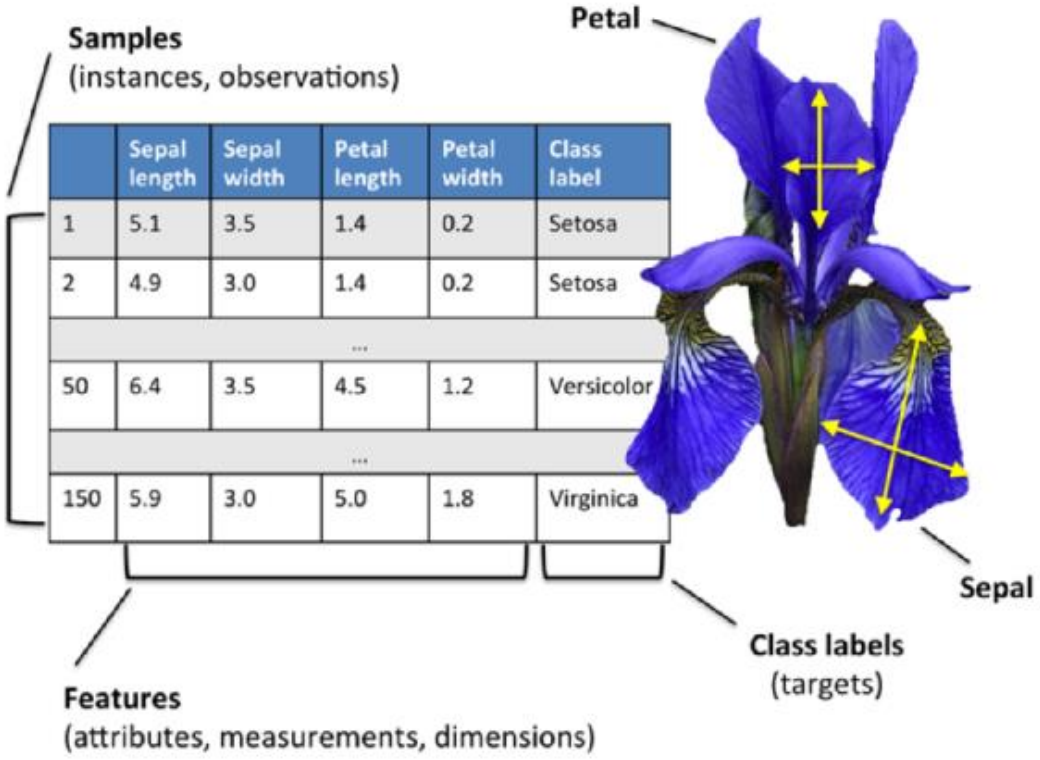
- **Données utilisées** : Jeu de données **Iris** (fourni par **scikit-learn**).
- **Étapes** :
  1. Charger le jeu de données.
  2. Comprendre ses caractéristiques (features) et ses classes (labels).
  3. Visualiser les données pour repérer les relations ou les éventuelles anomalies

… Application 2: Régression logistique



Le jeu de données comporte **150 échantillons**, avec **4 caractéristiques** (features) mesurées en centimètres :

1. **Sepal length** (*Longueur du sépale*)
2. **Sepal width** (*Largeur du sépale*)
3. **Petal length** (*Longueur du pétale*)
4. **Petal width** (*Largeur du pétale*)



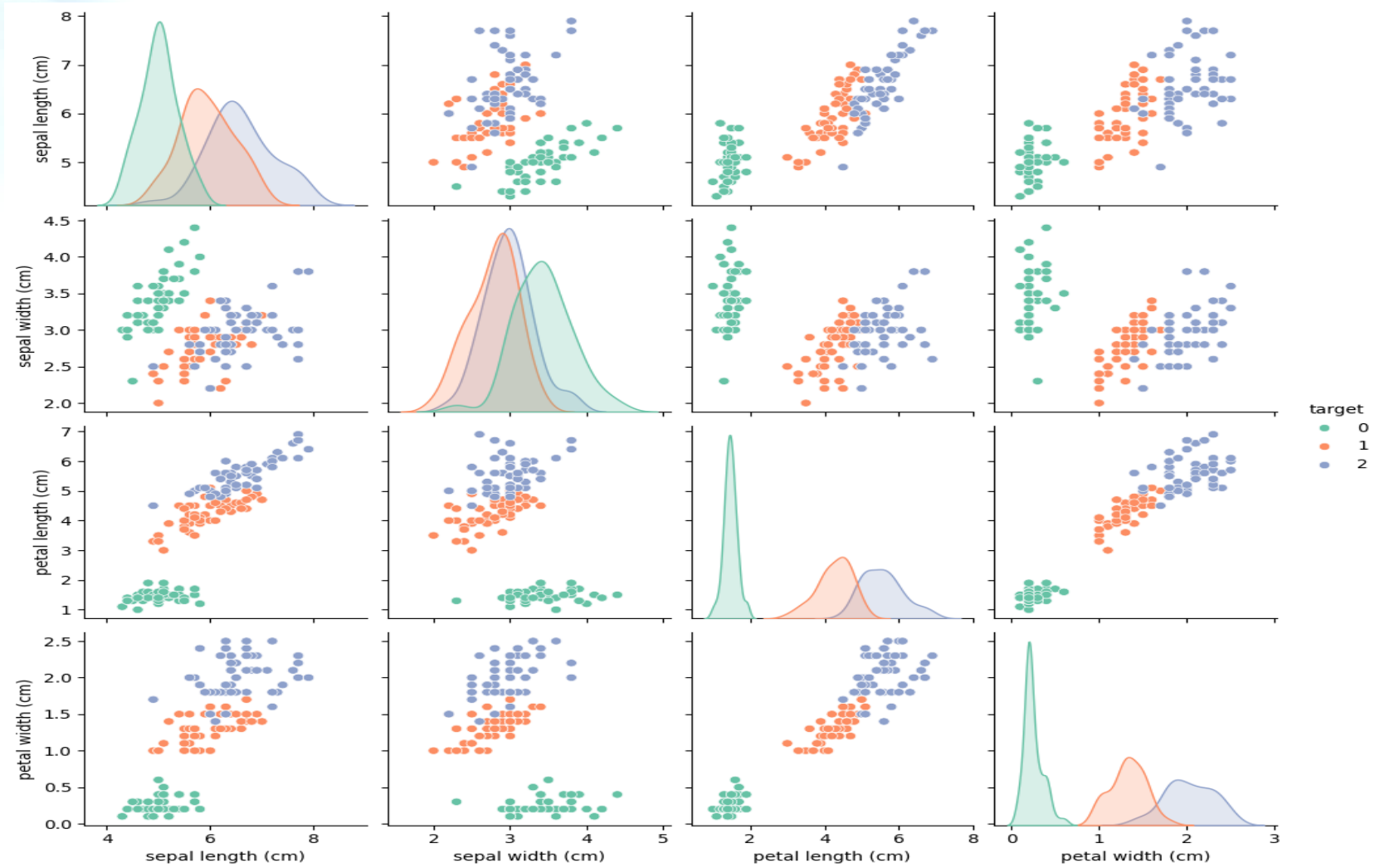
	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
…					
50	6.4	3.5	4.5	1.2	Versicolor
…					
150	5.9	3.0	5.0	1.8	Virginica

Ouvrir Google Colab et créer un Nouveau Notebook.
Renommer le RegressionLogistique.ipynb

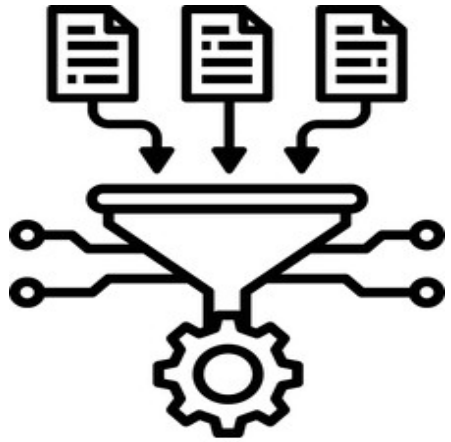
… Application 2: Régression logistique

```
from sklearn.datasets import load_iris
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Chargement des données
iris = load_iris()
data = pd.DataFrame(data=iris.data, columns=iris.feature_names)
data['target'] = iris.target

# Affichage d'un échantillon
print(data.head())

# Visualisation des données
sns.pairplot(data, hue='target', palette='Set2')
plt.show()
```

… Application 2: Régression logistique



3.6.3. Préparation des Données :



- **Nettoyage** : Pas nécessaire dans ce cas (jeu de données propre).
- **Sélection des données** :
  - On réduit les données à deux classes (setosa et versicolor) pour simplifier.
  - On sélectionne deux caractéristiques (longueur et largeur des sépales) pour visualiser facilement.
- **Division des données** :
  - 80 % pour l'entraînement.
  - 20 % pour les tests.

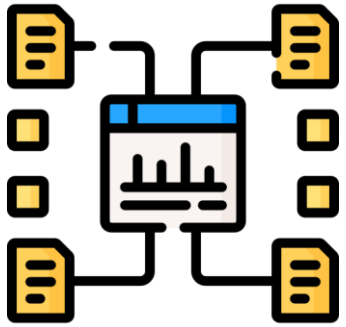
… Application 2: Régression logistique

```
from sklearn.model_selection import train_test_split


# Filtrer pour les deux premières classes
X = iris.data[iris.target != 2, :2]  # Deux caractéristiques
y = iris.target[iris.target != 2]    # Deux classes


# Division en ensembles d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print("Taille des données d'entraînement :", X_train.shape)
print("Taille des données de test :", X_test.shape)
```

```
Taille des données d'entraînement : (80, 2)
Taille des données de test : (20, 2)
```

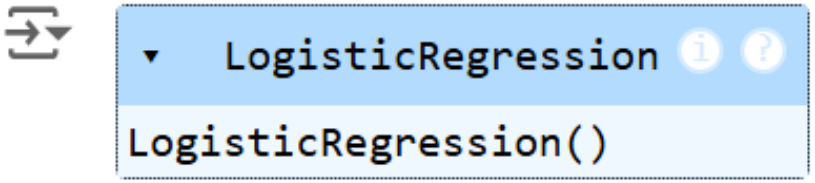
3.6.4. Modélisation :

- **Modèle choisi** : Régression logistique.
- **Pourquoi ?** :
  - Simple à comprendre et rapide à entraîner.
  - Adapté aux problèmes de classification binaire.
- **Entraînement du modèle** : Ajustement du modèle aux données d'entraînement.



… Application 2: Régression logistique

```
from sklearn.linear_model import LogisticRegression

# Création et entraînement du modèle
model = LogisticRegression()
model.fit(X_train, y_train)
```



3.6.5. Évaluation du Modèle :

- **Prédiction** : Utiliser les données de test pour évaluer les performances.
- **Métriques** :
  - Précision (accuracy).
  - Rapport de classification (precision, recall, F1-score).
  - Matrice de confusion.
- **Visualisation** : Matrice de confusion et frontière de décision.

… Application 2: Régression logistique

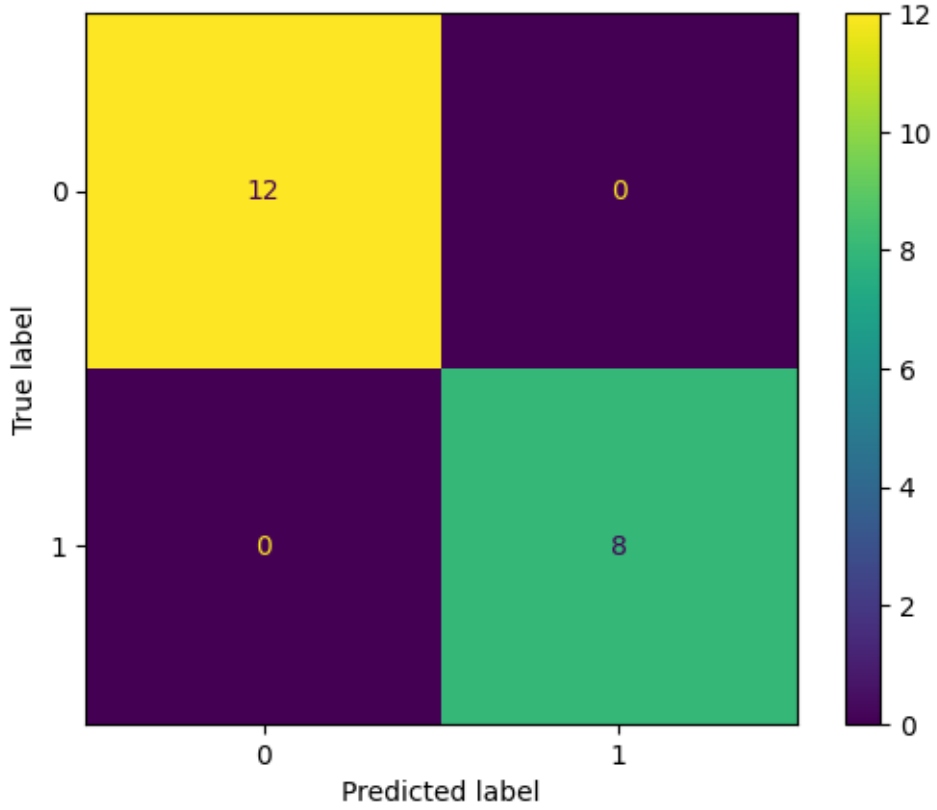
```
from sklearn.metrics import accuracy_score, classification_report, ConfusionMatrixDisplay

# Prédictions
y_pred = model.predict(X_test)

# Précision
print("Précision :", accuracy_score(y_test, y_pred))

# Rapport de classification
print("Rapport de classification :\n", classification_report(y_test, y_pred))

# Matrice de confusion
ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
plt.show()
```

… Application 2: Régression logistique

Précision et Rapport de classification

```
Précision : 1.0
Rapport de classification :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        12
           1       1.00      1.00      1.00         8

    accuracy                           1.00        20
   macro avg       1.00      1.00      1.00        20
weighted avg       1.00      1.00      1.00        20
```

Matrice de Confusion



3.6.6. Visualisation des Résultats :

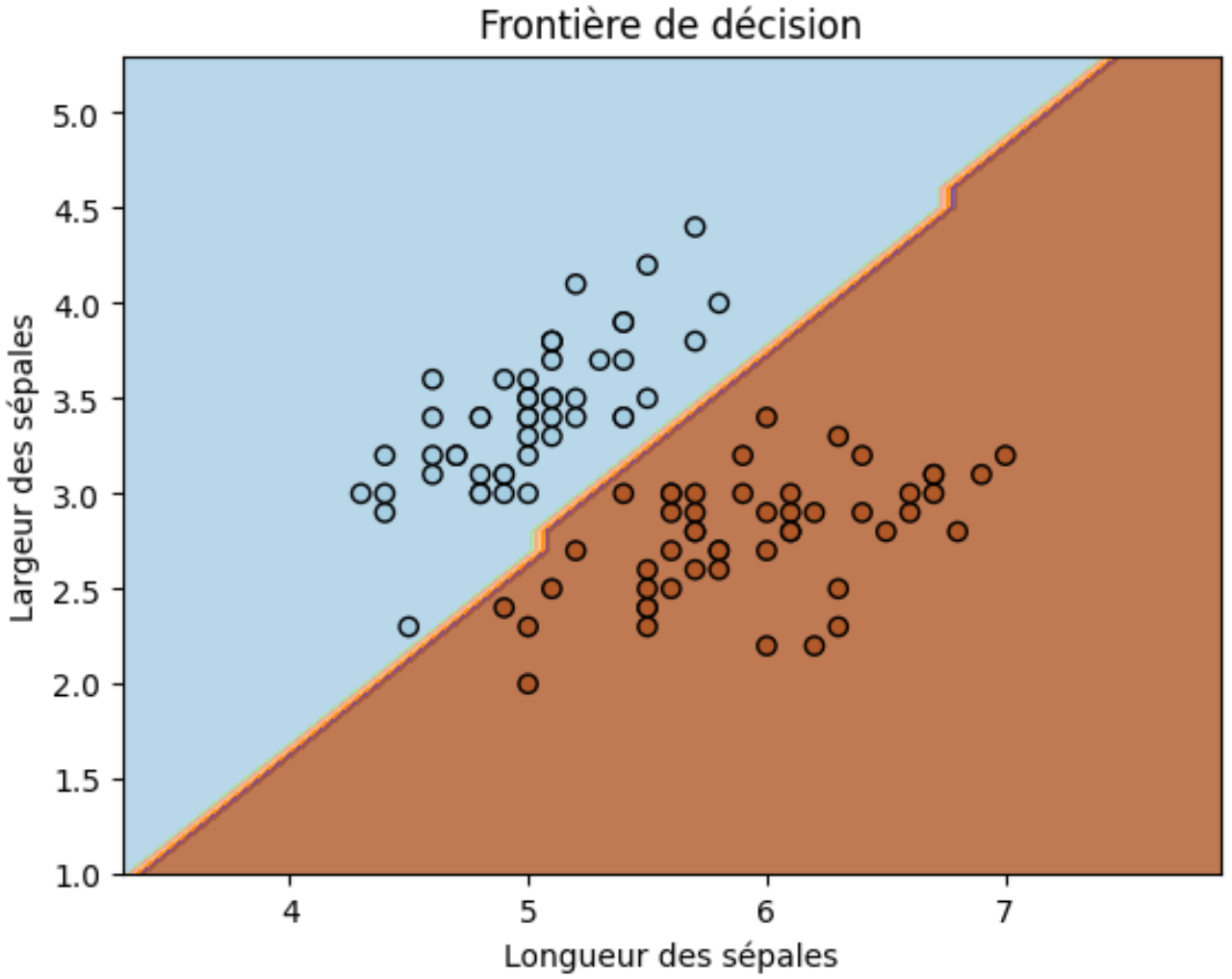
- Afficher la **frontière de décision** pour montrer comment le modèle sépare les deux classes.

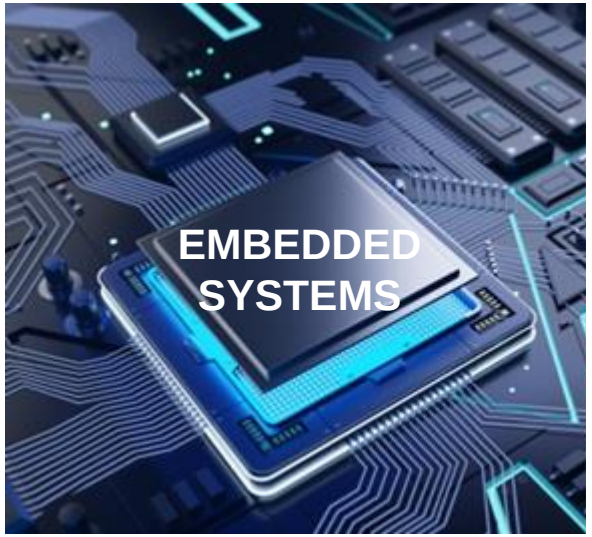
```
import numpy as np
# Frontière de décision
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1), np.arange(y_min, y_max, 0.1))

Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Graphique
plt.contourf(xx, yy, Z, alpha=0.8, cmap=plt.cm.Paired)
plt.scatter(X[:, 0], X[:, 1], c=y, edgecolor='k', cmap=plt.cm.Paired)
plt.xlabel("Longueur des sépales")
plt.ylabel("Largeur des sépales")
plt.title("Frontière de décision")
plt.show()
```

… Application 2: Régression logistique

La figure obtenue montre comment le modèle de régression logistique sépare deux classes (classification binaire) dans un espace à deux dimensions.

Les frontières de décision sont tracées en fonction des prédictions du modèle.





FIN !
Questions ?