République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Alger 1 – BENYOUCEF Benkhedda

Faculté des Sciences
Département Informatique

Rapport

Module : Intelligence Artificielle

Réalisé par :

Moulay Abdellah Asma
Belabbas Rania

Section : M1 ISII groupe 3

Année universitaire : 2024/2025

Approche matricielle de la Régression Linéaire :

Nous avons un dataset qui représente le prix moyen en fonction de la superficie. Nous allons appliquer une régression linéaire, comme dans l'exemple précédent, mais en utilisant des matrices cette fois-ci, car c'est plus simple, rapide et sans itération. Cette méthode est particulièrement utile lorsque nous disposons de plusieurs caractéristiques

Lors de l'étape de **nettoyage,** nous avons constaté qu'il manquait des valeurs. Pour les remplacer, bous avons choisi de les remplir avec la moyenne :

```
d['Prix'].fillna(d['Prix'].mean(), inplace=True)
d['Superficie'].fillna(d['Superficie'].mean(), inplace=True)
```

Nous avons rencontré l'erreur suivante :

```
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Explication : indique qu'il y a un mélange de types de données dans l'une des colonnes. Plus précisément, la colonne contient à la fois des nombres et des chaînes de caractères. Cela empêche les opérations mathématiques (comme la moyenne dans ce cas).

Solution : pour cela nous avons utilisé la méthode : **« pd.to_numeric »** qui permet de convertir les colonnes en valeurs numériques en remplaçant les chaînes ou autres types invalides par **NaN**.

Par la suite, nous avons pu remplacer les valeurs manquantes et après vérification on a trouvé qu'il n'y avait plus de valeurs manquantes dans les 2 colonnes.

```
print(d.isnull().sum())
```

```
Prix          0
Superficie    0
dtype: int64
```

Tout d'abord pour ajouter une colonne de 1 pour inclure **b** dans le modèle nous utilisons la fonction '**hstack'** de la bibliothèque **Numpy** :

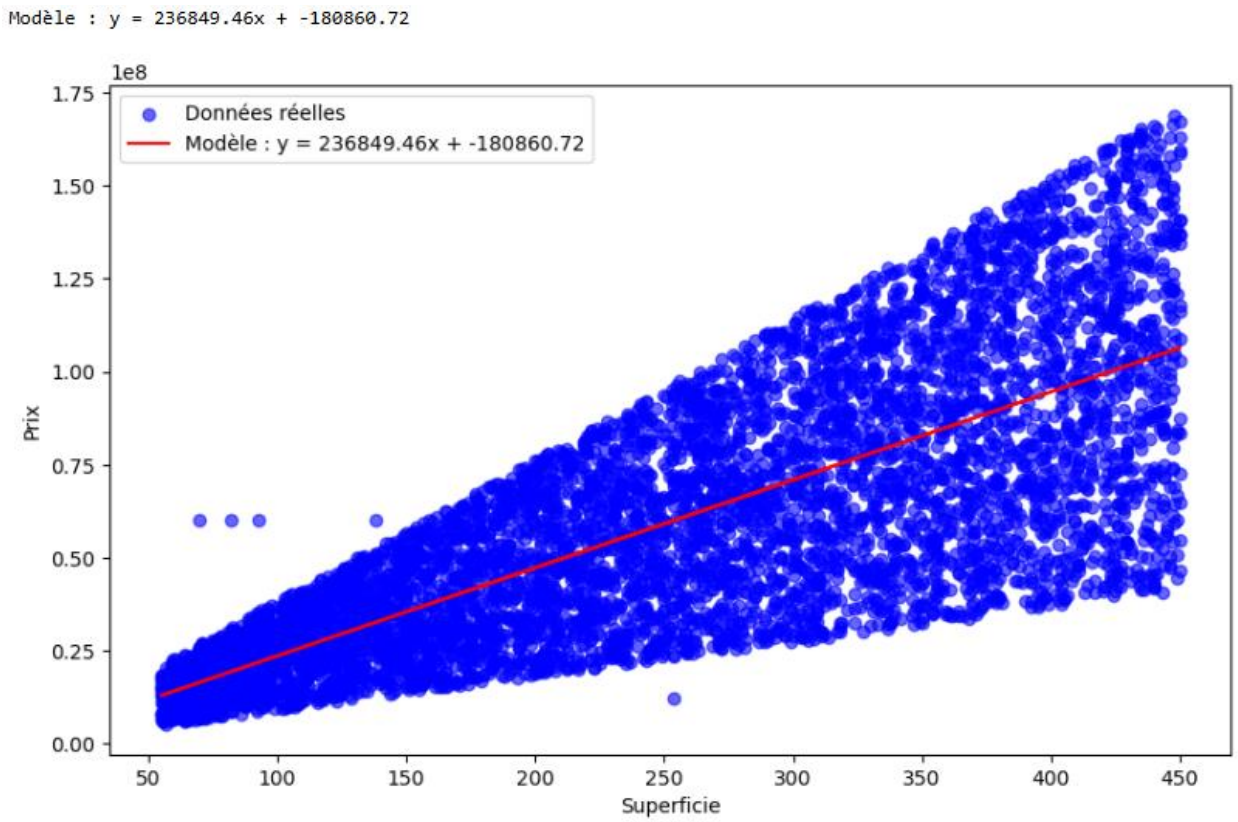
X = np.hstack((X, np.ones((X.shape[0], 1))))

Ensuite, nous avons calculer theta, qui contient les paramètres a et b (ou theta est un vecteur). Apres cela, nous avons effectué des prédictions avec le modèle en utilisant

Y_pred = X @ theta

(Note : '@' représente la multiplication matricielle)

Résultat :



La fonction du cout :

Nous avons calculé la **fonction de coût J(θ)**, qui mesure l'erreur quadratique moyenne entre les prédictions et les valeurs réelles. C'est la somme des erreurs au carré, divisée par 2m, ou **m** est le nombre d'échantillons.

```python
def fonction_cout(X, y, theta):
    m = len(y)
    y_pred = X @ theta
    cout = (1 / (2 * m)) * np.sum((y_pred - y) ** 2)
    return cout

# Calculer le coût initial
cout_initial = fonction_cout(X, y, theta)
print(f"Valeur de la fonction du coût : {cout_initial:.2f}")
```

Résultat :

Valeur de la fonction du coût : **269492357846213.59**

L'erreur est initialement élevée, donc on utilise la méthode descente de gradient pour optimiser les paramètres du modèle