La Régression Linéaire

1. Qu'est-ce que la régression linéaire ?

La régression linéaire est une méthode **d'apprentissage supervisé** utilisée pour prédire une valeur numérique en fonction d'autres variables. Elle permet de trouver une relation entre une variable dépendante Y (ce qu'on veut prédire) et une ou plusieurs variables indépendantes X (les facteurs influençant la prédiction).

Formule de la régression linéaire simple :

$$Y=aX+b$$

où :

- Y est la valeur à prédire (ex: prix d'une maison),
- X est la variable explicative (ex: surface de la maison),
- a est le coefficient de régression (pente de la droite),
- b est l'ordonnée à l'origine (valeur de Y quand $X = 0$).

Dans le cas de plusieurs variables explicatives, on parle de **régression linéaire multiple** :

$$Y = a_1X_1 + a_2X_2 + ... + a_nX_n + b$$

où chaque X_i représente une caractéristique influençant la prédiction.

Exercice 1 : Prédiction de prix avec la régression linéaire

Objectif :

On va entraîner un modèle de régression linéaire pour prédire le prix d'une maison en fonction de ses caractéristiques (surface, nombre de chambres, etc.).

1. Chargement des données en ligne

On utilise le dataset "California Housing" qui est disponible en ligne via Scikit-learn.

2. Préparation des données

- Séparer les variables explicatives et la variable cible.
- Diviser les données en un ensemble d'entraînement et un ensemble de test.

3. Entraînement du modèle

- Utiliser la régression linéaire pour ajuster un modèle aux données d'entraînement.

4. Faire une vraie prédiction

- Utiliser le modèle entraîné pour prédire le prix d'une maison avec certaines caractéristiques réelles.

5. Évaluer le modèle

- Comparer les prédictions aux vraies valeurs avec des métriques comme l'erreur absolue moyenne (MAE) et l'erreur quadratique moyenne (RMSE).

Code : Exercice 1 - Régression Linéaire sur les prix des maisons

```
# Importation des bibliothèques nécessaires
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.datasets import fetch_california_housing

# Chargement du dataset en ligne
data = fetch_california_housing()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['PRICE'] = data.target  # Ajout de la colonne des prix des maisons

# Affichage des premières lignes du dataset
print(df.head())

# Séparation des variables explicatives (X) et de la variable cible (y)
X = df.drop('PRICE', axis=1)
y = df['PRICE']

# Division des données en ensemble d'entraînement (80%) et de test (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Création et entraînement du modèle de régression linéaire
model = LinearRegression()
model.fit(X_train, y_train)

# Prédiction des prix sur l'ensemble de test
y_pred = model.predict(X_test)

# Évaluation du modèle
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
```

```
# Affichage des résultats
print(f'MAE (Erreur absolue moyenne) : {mae:.2f}')
print(f'RMSE (Racine de l\'erreur quadratique moyenne) : {rmse:.2f}')
print(f'R2 Score (Coefficient de détermination) : {r2:.2f}')

# Visualisation des résultats
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.xlabel('Prix réel')
plt.ylabel('Prix prédit')
plt.title('Régression Linéaire : Prédiction des prix immobiliers')
plt.show()
```

1. **Chargement des données** :
   - On utilise fetch_california_housing() pour récupérer un dataset de maisons en Californie avec leur prix et différentes caractéristiques.
2. **Préparation des données** :
   - On définit les variables explicatives (surface, nombre de chambres, etc.) et la variable cible (prix des maisons).
   - On divise les données en un ensemble d'entraînement et de test (80% - 20%).
3. **Entraînement du modèle** :
   - On utilise LinearRegression() pour ajuster un modèle sur les données d'entraînement.
4. **Faire une vraie prédiction** :
   - On prédit les prix sur l'ensemble de test et on compare avec les vrais prix.
5. **Évaluation du modèle** :
   - On utilise MAE, RMSE et R^2 pour voir si notre modèle est performant.
6. **Visualisation** :
   - On affiche un nuage de points comparant les prix réels et les prix prédits.

Réponse :

```
MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup  Latitude
\
0  8.3252      41.0  6.984127   1.023810       322.0  2.555556
37.88
1  8.3014      21.0  6.238137   0.971880      2401.0  2.109842
37.86
2  7.2574      52.0  8.288136   1.073446       496.0  2.802260
37.85
3  5.6431      52.0  5.817352   1.073059       558.0  2.547945
37.85
4  3.8462      52.0  6.281853   1.081081       565.0  2.181467
37.85

   Longitude  PRICE
0    -122.23  4.526
1    -122.22  3.585
2    -122.24  3.521
3    -122.25  3.413
4    -122.25  3.422
```

```
MAE (Erreur absolue moyenne) : 0.53
RMSE (Racine de l'erreur quadratique moyenne) : 0.75

R2 Score (Coefficient de détermination) : 0.58
```

Interprétation des résultats

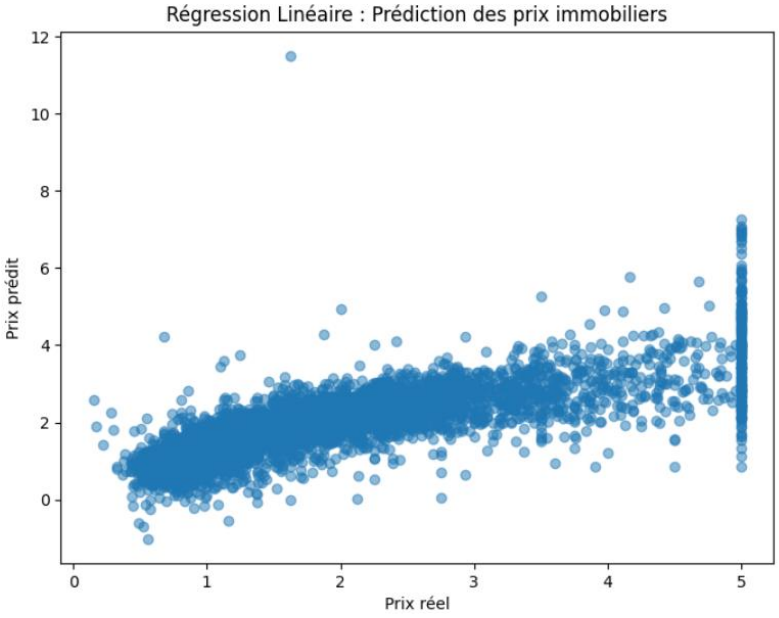
Données du dataset
Chaque ligne représente une zone géographique avec plusieurs caractéristiques :

- **MedInc** : Revenu médian des habitants.
- **HouseAge** : Âge moyen des maisons.
- **AveRooms** : Nombre moyen de pièces par maison.
- **AveBedrms** : Nombre moyen de chambres.
- **Population** : Nombre d'habitants dans la zone.
- **AveOccup** : Nombre moyen d'occupants par maison.
- **Latitude / Longitude** : Localisation de la zone.
- **PRICE** : Prix médian des maisons dans la zone (cible du modèle).

Performance du modèle
1. MAE (0.53)
En moyenne, le modèle se trompe de **0.53 unité** (comme les prix sont normalisés entre 0 et 5, l'erreur est plutôt faible).

2. RMSE (0.75)
L'erreur moyenne est de **0.75 unité**, ce qui est un peu plus sévère car RMSE pénalise les grandes erreurs.

3. R^2 Score (0.58)
58% des variations du prix sont expliquées par le modèle. C'est correct, mais il reste **42% d'imprécision** → il manque peut-être des variables importantes.



Interprétation du graphique

- Le nuage de points montre la corrélation entre les prix réels (axe X) et les prix prédits (axe Y).
- On remarque une tendance globale positive, mais avec des erreurs significatives et une saturation des prédictions autour de 5.
- Cela indique que le modèle a des difficultés à bien estimer les prix élevés et pourrait être amélioré.

Comment interpréter ce type de graphe ?

1. **Comprendre les axes** : Ici, l'axe X représente les prix réels et l'axe Y les prix prédits.
2. **Rechercher la tendance** : Une bonne régression devrait avoir les points proches d'une diagonale Y=X.

3. **Observer la dispersion** : Plus les points sont éparpillés, plus le modèle a des erreurs.
4. **Identifier les biais** : Si les points sont systématiquement sous ou surestimés, le modèle est biaisé.
5. **Détecter les outliers** : Des points très éloignés de la tendance peuvent indiquer des erreurs majeures.
6. **Vérifier les saturations** : Ici, les prédictions sont plafonnées à 5, ce qui montre une limitation du modèle.

7. **Évaluer la précision** : Si les points sont trop dispersés verticalement, la prédiction est peu fiable.
8. **Comparer avec d'autres modèles** : Un modèle plus performant aurait une tendance plus proche d'une ligne droite.

9. **Améliorer le modèle** : Essayer d'autres algorithmes ou ajouter des **features** peut réduire l'erreur.
10. **Utiliser des métriques** : Croiser l'analyse visuelle avec MAE, RMSE et R^2 pour valider les résultats.

Le modèle fonctionne mais n'est pas parfait.
Il peut être amélioré en testant d'autres modèles (Ex: Random Forest, Gradient Boosting) ou en ajoutant des variables influentes.

Dans nos résultats actuels :

1. Le modèle a été entraîné sur un dataset existant et a fait des prédictions **sur l'ensemble des données**.
2. On a évalué la qualité des prédictions (MAE, RMSE, R^2), mais **on n'a pas encore utilisé le modèle pour prédire un prix à partir d'une nouvelle entrée spécifique.**

Comment faire une vraie prédiction ?

Ajoutons maintenant une vraie prédiction :

1. On entre **nos propres caractéristiques** d'une maison.
2. Le modèle **calcule et affiche** le prix estimé.

Ajoutez cette partie à la fin de votre code :

```
# Exemple : prédire le prix d'une maison avec vos propres caractéristiques
import numpy as np

# Définissez les caractéristiques de votre maison (modifiez selon vos
besoins)
maison_custom = np.array([[8.0, 30, 6, 1, 1000, 2, 37.85, -122.25]])  #
Exemple

# Prédire le prix
prix_pred = model.predict(maison_custom)

print(f"Le prix estimé pour cette maison est : {prix_pred[0]:.2f} (en
unités du dataset)")
```

NB : Paramètres

a. **8.0** : Nombre de chambres (par exemple, 8 chambres).
b. **30** : Nombre de fenêtres (cela dépend de la façon dont le modèle a été formé, mais cela pourrait représenter le nombre de fenêtres dans la maison).
c. **6** : Nombre de salles de bains (par exemple, 6 salles de bains).
d. **1** : Nombre d'étages ou niveau de la maison (par exemple, une maison de un étage).
e. **1000** : Taille de la maison en pieds carrés (par exemple, 1000 pieds carrés).
f. **2** : Nombre de garages ou espaces de stationnement (par exemple, 2 garages).
g. **37.85** : Latitude de l'emplacement de la maison (par exemple, 37.85 degrés de latitude).
h. **-122.25** : Longitude de l'emplacement de la maison (par exemple, -122.25 degrés de longitude).

Comment savoir ce qu'il faut inclure ?

Examinez la documentation ou les colonnes de votre jeu de données. Par exemple :

```
import pandas as pd
data = pd.read_csv('chemin_vers_votre_fichier.csv')
print(data.columns)
```

- **On définit une maison** avec ses caractéristiques (MedInc, HouseAge, AveRooms, etc.).
- **Le modèle utilise ces données** pour prédire un prix.
- **Le résultat s'affiche enfin clairement** : un vrai prix estimé pour nos maison!

```
Le prix estimé pour cette maison est : 4.02 (en unités du dataset)
```

Exercice 2 : Classification avec KNN sur le dataset Iris

Après la régression linéaire, on va travailler sur un autre type de problème : la **classification**. Ici, on va utiliser l'algorithme des **K plus proches voisins (KNN)** pour classer des fleurs en fonction de leurs caractéristiques.

Objectif :

- Utiliser un modèle KNN pour classifier des fleurs en trois catégories.
- Évaluer le modèle avec une matrice de confusion et un rapport de classification.

```
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns

# Chargement du dataset Iris
data_iris = load_iris()
X_iris = data_iris.data
y_iris = data_iris.target

# Séparation en train et test
X_train, X_test, y_train, y_test = train_test_split(X_iris, y_iris, test_size=0.2,
random_state=42)

# Entraînement du modèle KNN avec k=3
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

# Prédiction
y_pred = knn.predict(X_test)

# Évaluation
print(classification_report(y_test, y_pred))

# Matrice de confusion
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, cmap='Blues', fmt='d')
plt.xlabel('Prédit')
plt.ylabel('Réel')
plt.title('Matrice de Confusion')
plt.show()
```

Métriques d'Évaluation

1. MAE (Mean Absolute Error - Erreur Absolue Moyenne)

C'est la moyenne des erreurs en valeur absolue entre les prédictions et les vraies valeurs.

Exemple : Si on prédit le prix d'une maison à **320 000€** alors qu'elle vaut **300 000€**, l'erreur est **20 000€**.

2. RMSE (Root Mean Squared Error - Racine de l'Erreur Quadratique Moyenne)

Comme le MAE, mais il punit davantage les grosses erreurs.

Exemple : Si une maison est à **300 000€**, et on prédit **400 000€**, l'erreur sera plus forte qu'avec MAE.

3. Précision (Precision)

Parmi toutes les prédictions positives du modèle, combien sont correctes ?

Exemple : Si un modèle dit que **10 e-mails** sont du **SPAM**, mais que **seulement 8 sont vraiment des SPAM**, la précision est **bonne, mais pas parfaite**.

4. Rappel (Recall ou Sensibilité)

Parmi tous les cas positifs réels, combien le modèle en a détecté ?

Exemple : S'il y a **20 vrais e-mails SPAM**, mais que le modèle n'en détecte que **15**, le rappel n'est pas parfait.

5. F1-Score

C'est un équilibre entre **précision et rappel**.

Exemple : Si on veut éviter **les faux positifs et les faux négatifs**, il faut un bon **F1-score**.

Quand les utiliser ?

- **MAE et RMSE** → Pour prédire des chiffres (ex : prix, température).
- **Précision et Rappel** → Pour classifier (ex : SPAM, détection de fraude).
- **F1-score** → Quand il faut équilibrer précision et rappel.

Exercice 1 : Classification des avis de produits

Contexte :

Un magasin en ligne souhaite analyser les avis de ses clients pour déterminer si un avis sur un produit est positif ou négatif. Pour ce faire, vous devez construire un modèle d'apprentissage automatique qui classe les avis en deux catégories : positif (1) ou négatif (0).

Objectif :

Utiliser des *features* extraites des commentaires de clients (tels que la fréquence de certains mots-clés comme "bon", "excellent", "mauvais", "horrible") pour entraîner un modèle de classification. Vous devrez également évaluer la performance de votre modèle en mesurant sa précision.

Étapes à suivre :

1. **Préparation des données** :
   - Collecter un ensemble d'avis clients et leurs étiquettes (1 pour un avis positif, 0 pour un avis négatif).
   - Nettoyer et prétraiter les avis (par exemple, suppression des stop words, normalisation des textes).
2. **Extraction des features** :
   - Utiliser des techniques de vectorisation pour transformer les avis textuels en *features*. Par exemple, vous pouvez utiliser le *CountVectorizer* ou le *TF-IDF Vectorizer* pour convertir chaque mot ou combinaison de mots en une représentation numérique.
3. **Séparation des données** :
   - Divisez les données en un ensemble d'entraînement et un ensemble de test (par exemple, 80% pour l'entraînement et 20% pour le test).
4. **Entraînement du modèle** :
   - Entraîner un modèle de classification (par exemple, un modèle Naive Bayes, SVM, ou un arbre de décision) en utilisant les données d'entraînement.
5. **Évaluation du modèle** :
   - Tester le modèle avec les données de test et calculer la précision, la précision, le rappel et la F-mesure pour évaluer la performance.

6. **Amélioration** :

    - Tenter de tester différentes techniques de vectorisation ou des modèles de classification pour améliorer la précision du modèle.

Exercice 2 : Classification des tweets

Contexte :

Vous avez une base de données de tweets et vous souhaitez prédire si un tweet est lié à un sujet politique (1) ou non-politique (0). Les tweets peuvent contenir des hashtags, des mentions et des liens, mais vous devez vous concentrer sur le texte du tweet pour déterminer s'il appartient à la catégorie politique ou non.

Objectif :

Extraire des *features* du texte des tweets (comme la fréquence de certains mots-clés tels que "élection", "gouvernement", "loi") pour entraîner un modèle qui pourra classer les tweets en deux catégories : politique (1) ou non-politique (0).

Étapes à suivre :

1. **Préparation des données** :

    - Recueillir un ensemble de tweets et leurs étiquettes (1 pour un tweet politique, 0 pour un tweet non-politique).
    - Nettoyer les données en supprimant les mentions (@), les hashtags (#), les liens URL et les caractères non pertinents.

2. **Extraction des features** :

    - Utiliser des techniques comme le *CountVectorizer* ou *TF-IDF Vectorizer* pour extraire des *features* des tweets. Vous pouvez extraire les mots-clés spécifiques, comme "élection", "gouvernement", "loi", ou toute autre expression fréquemment utilisée dans les tweets politiques.

3. **Séparation des données** :

    - Divisez les données en un ensemble d'entraînement et un ensemble de test.

4. **Entraînement du modèle** :
    - Choisissez un modèle de classification approprié, comme Naive Bayes, SVM, ou un modèle de régression logistique, et entraînez-le sur les données d'entraînement.
5. **Évaluation du modèle** :
    - Utilisez l'ensemble de test pour évaluer la performance du modèle. Vous pouvez mesurer la précision du modèle ainsi que d'autres métriques comme le rappel et la F-mesure.

6. **Amélioration** :
    - Expérimenter avec différentes méthodes d'extraction des *features* et ajuster les hyperparamètres du modèle pour améliorer la précision et la généralisation.