Page de garde

Etude de cas - Analyse des ventes e-commerce 2019

Nom: Konate

Prénom: Issa Banzan

Bloc de competence: - Analyse des ventes e-commerce développer une solution d'intelligence artificielle

Objectif:
Préparer, analyser et modéliser les données de ventes e-commerce de l'année 2019 dans le but de réaliser des prédictions de la demande quotidienne à l'aide d'un modèle de machine learning.

1) Nombre total de commandes en 2019

Exécution de la première requête pour compter le nombre total de commandes enregistrées dans l'année 2019

2) Chiffre d'affaires total

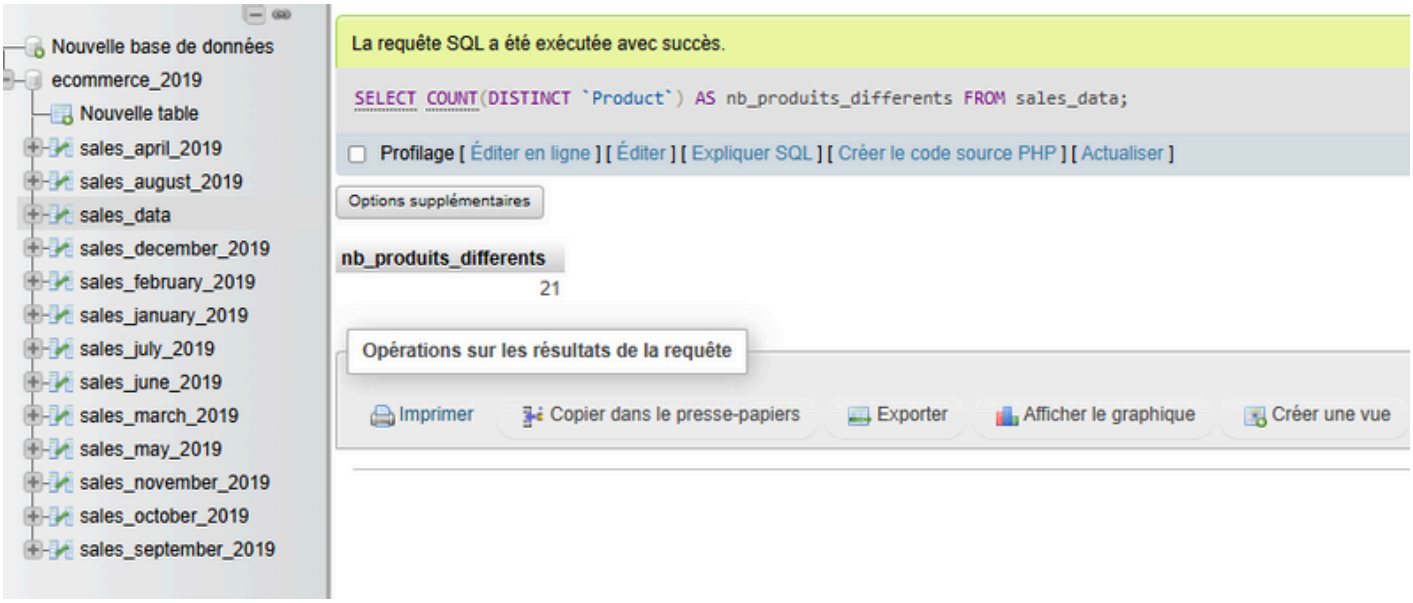
Le chiffre d'affaires a été calculé en multipliant la quantité commandée par le prix unitaire pour chaque ligne.

3) Produit le plus vendu

Le produit qui a généré le plus de ventes en quantité

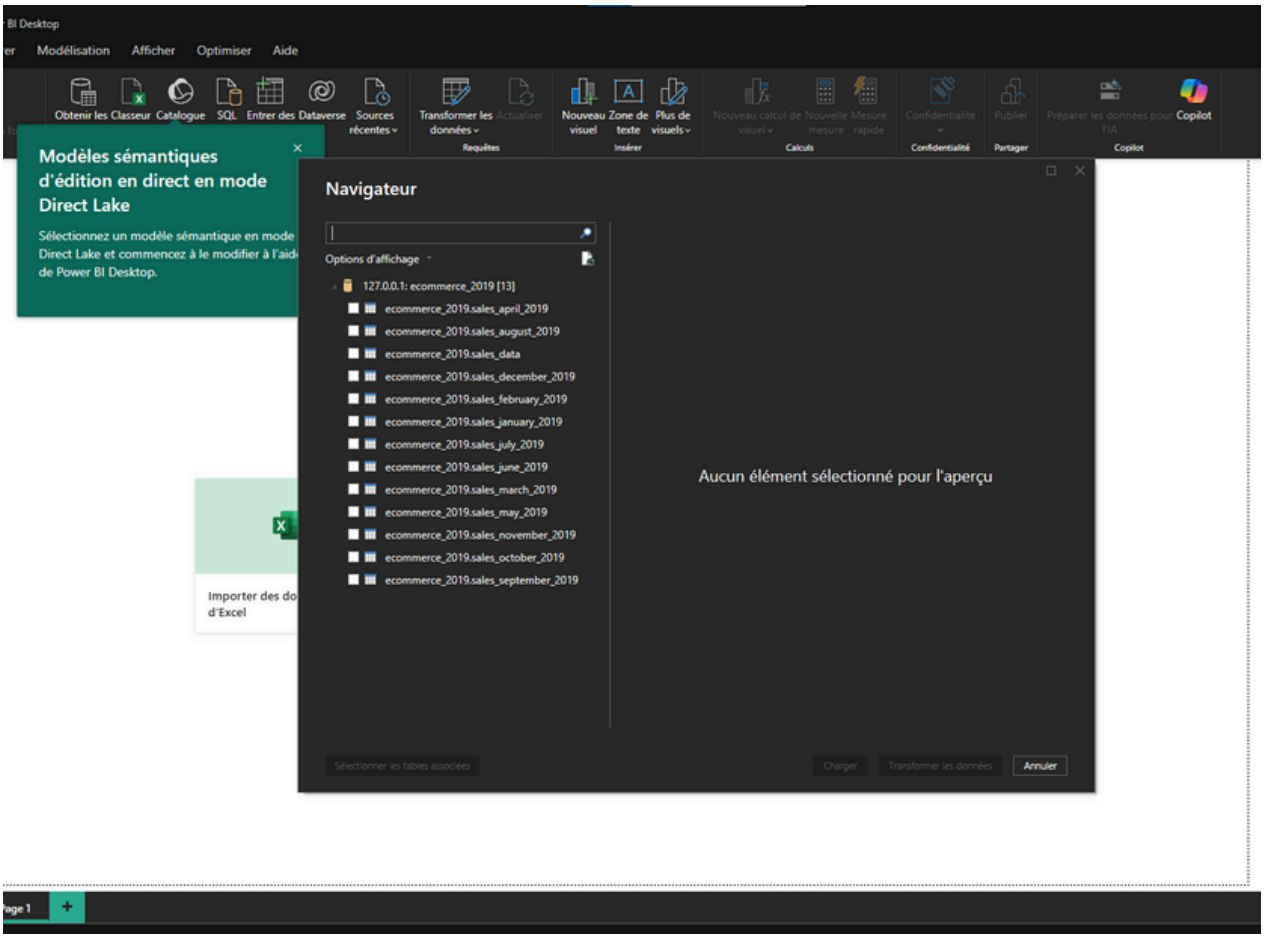
4) Nombre de produits différents

Pour connaître la diversité du catalogue produit :



Partie 2 : Dashboard Power BI

Dans cette partie, j'ai utilisé power BI pour visualiser les données e-commerce de l'année 2019.
Connexion à la base de données
Les tables mensuelles importées dans MySQL ont été chargées dans Power BI via une connexion directe à la base de données MySQL locale (localhost



Power BI permet ensuite :

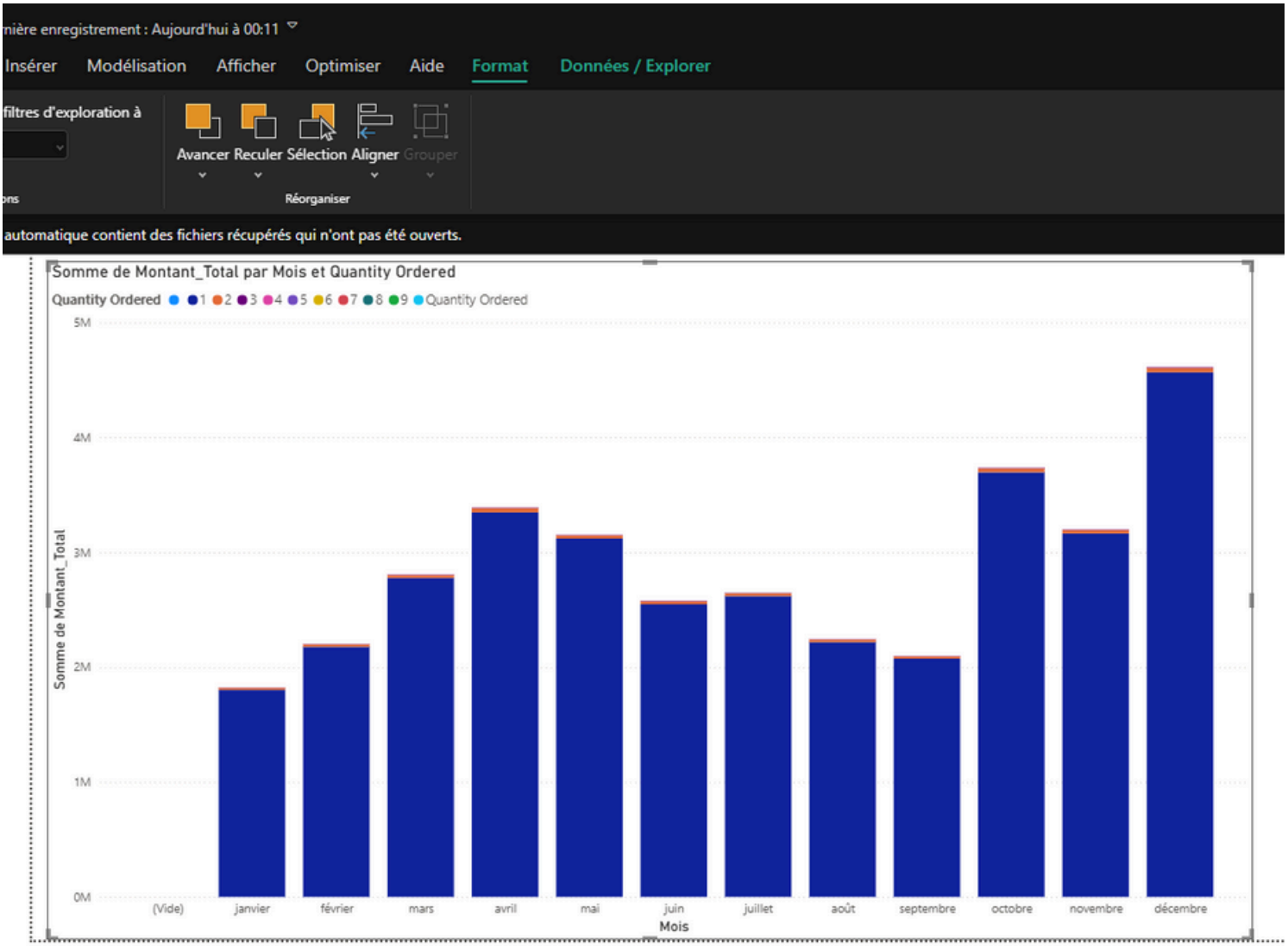
- Fusionner les tables (via Power Query ou une requête SQL)
- de créer un modèle de données relationnel
- de construire des visuels dynamiques : histogrammes, KPIs, séries temporelles, cartes, etc.

Objectif du dashboard

L'objectif du tableau de bord est de :

- Suivre les ventes par produit, par jour, par région
- Identifier les tendances mensuelles
- Mettre en avant les pics d'activité
- Fournir des indicateurs clés (KPI) comme le CA total, le nombre de commandes, etc.

Le Chiffre d’affaires mensuel ainsi que la quantité vendue mensuellement



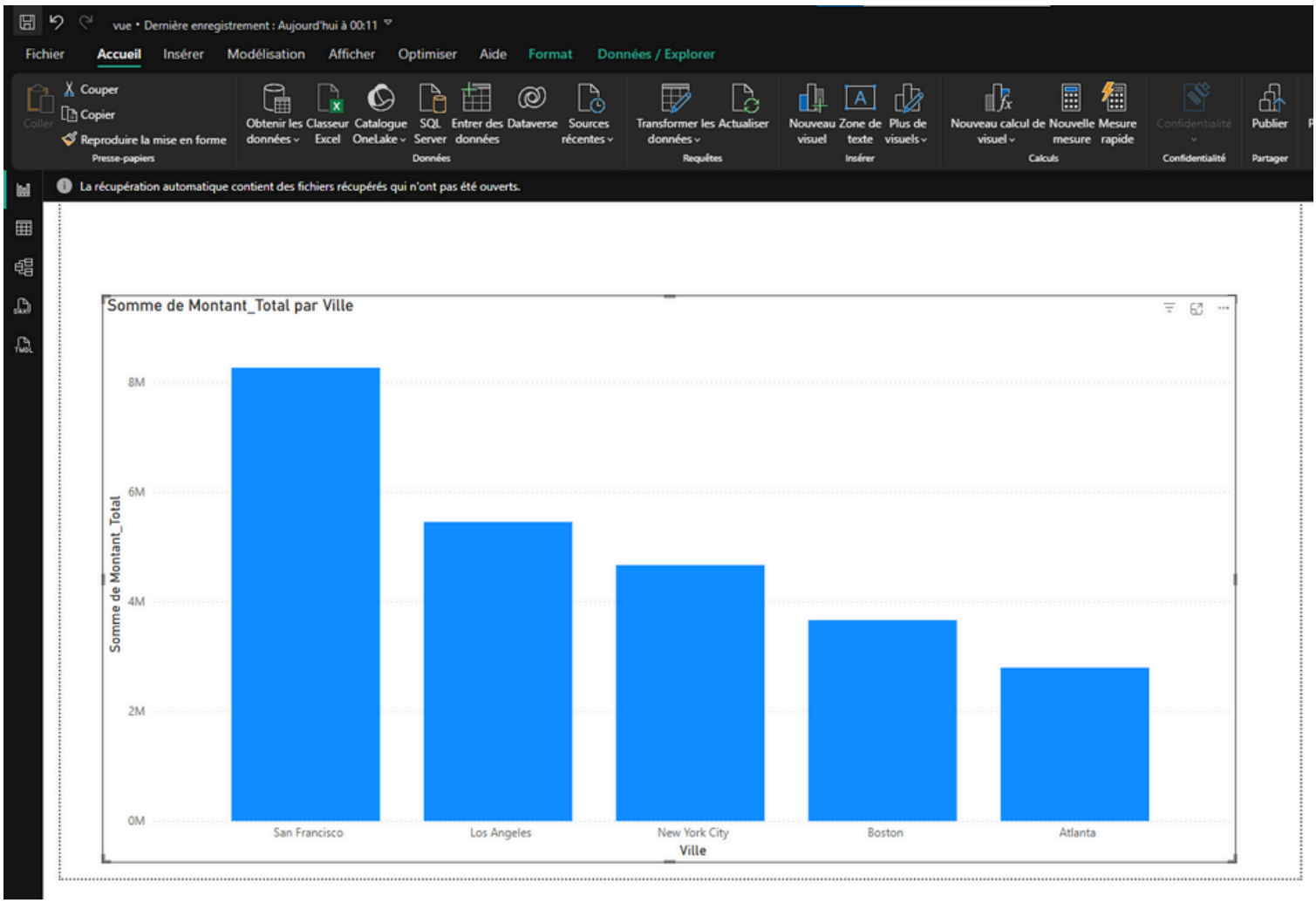
Ce graphique présente à la fois :

- Le chiffre d’affaires mensuel (barres bleues)
- La quantité totale de produits commandés chaque mois (points colorés)

Observations :

- Le mois de décembre est celui qui génère le chiffre d'affaires le plus élevé
- On observe également des pics en avril et octobre.
- Une baisse est visible en août, probablement liée à une baisse d’activité.

Analyse géographique: TOP 5 des villes sur le montant total d'achat



Le graphique ci-dessus met en évidence les 5 villes générant le plus de chiffre d'affaires en 2019 :

Résultats :

- San Francisco plus de 8M $ de chiffre d'affaires.
- Los Angeles et New York City.
- Boston et Atlanta

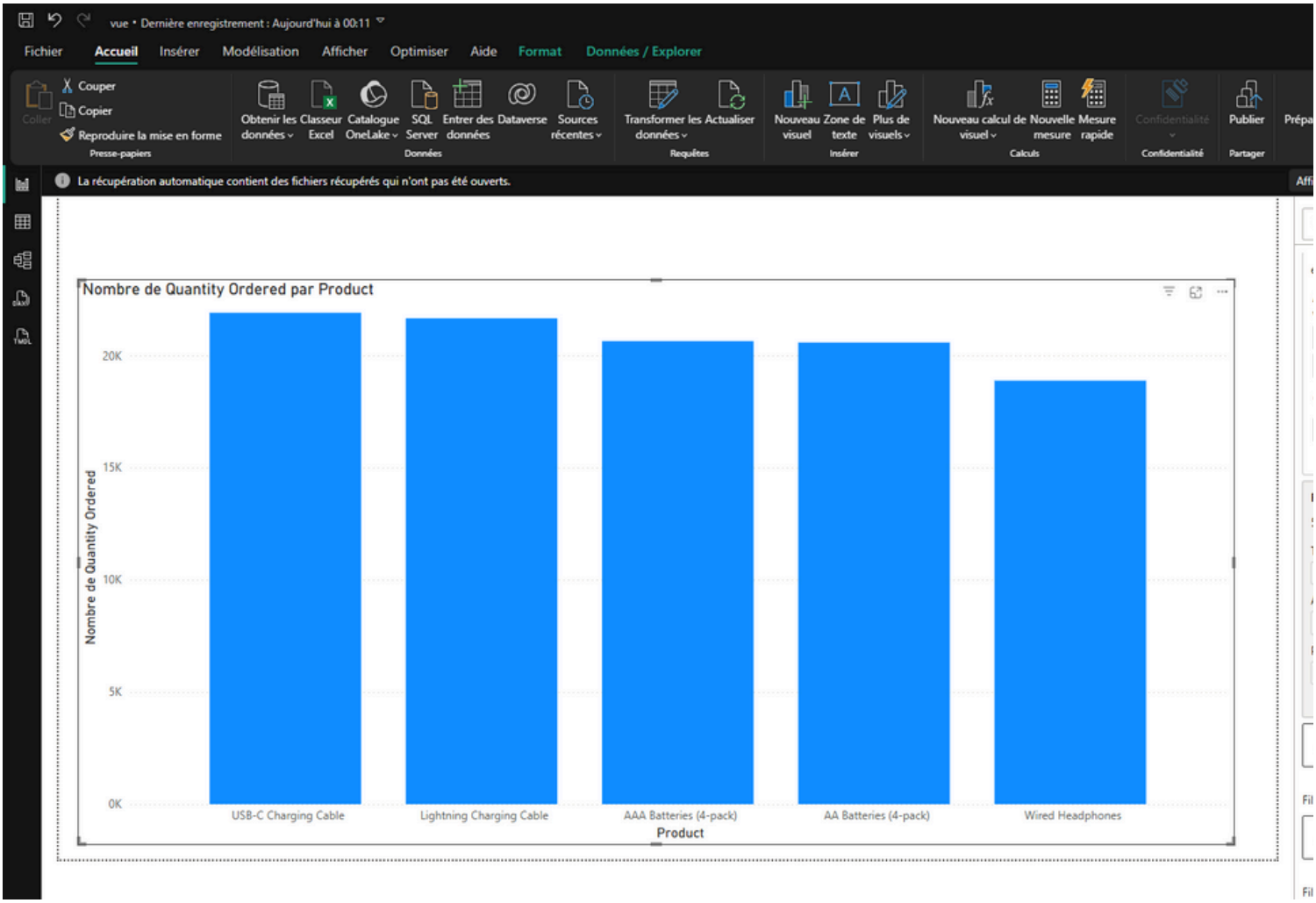
Interprétation :
Une forte densité de population
Un pouvoir d'achat plus élevé
Une meilleure accessibilité logistique
Le graphique ci-dessus met en évidence les 5 villes générant le plus de chiffre d'affaires en 2019 :
Résultats :

- San Francisco plus de 8M $ de chiffre d'affaires.
- Los Angeles et New York City.
- Boston et Atlanta.

Interprétation :

- Une forte densité de population
- Un pouvoir d'achat plus élevé
- Une meilleure accessibilité logistique

Analyse produit:les 5 produits les plus achetés par quantité et par montant total



Le graphique ci-dessus présente les 5 produits les plus achetés en 2019, classés par quantité totale vendue :

Résultats :

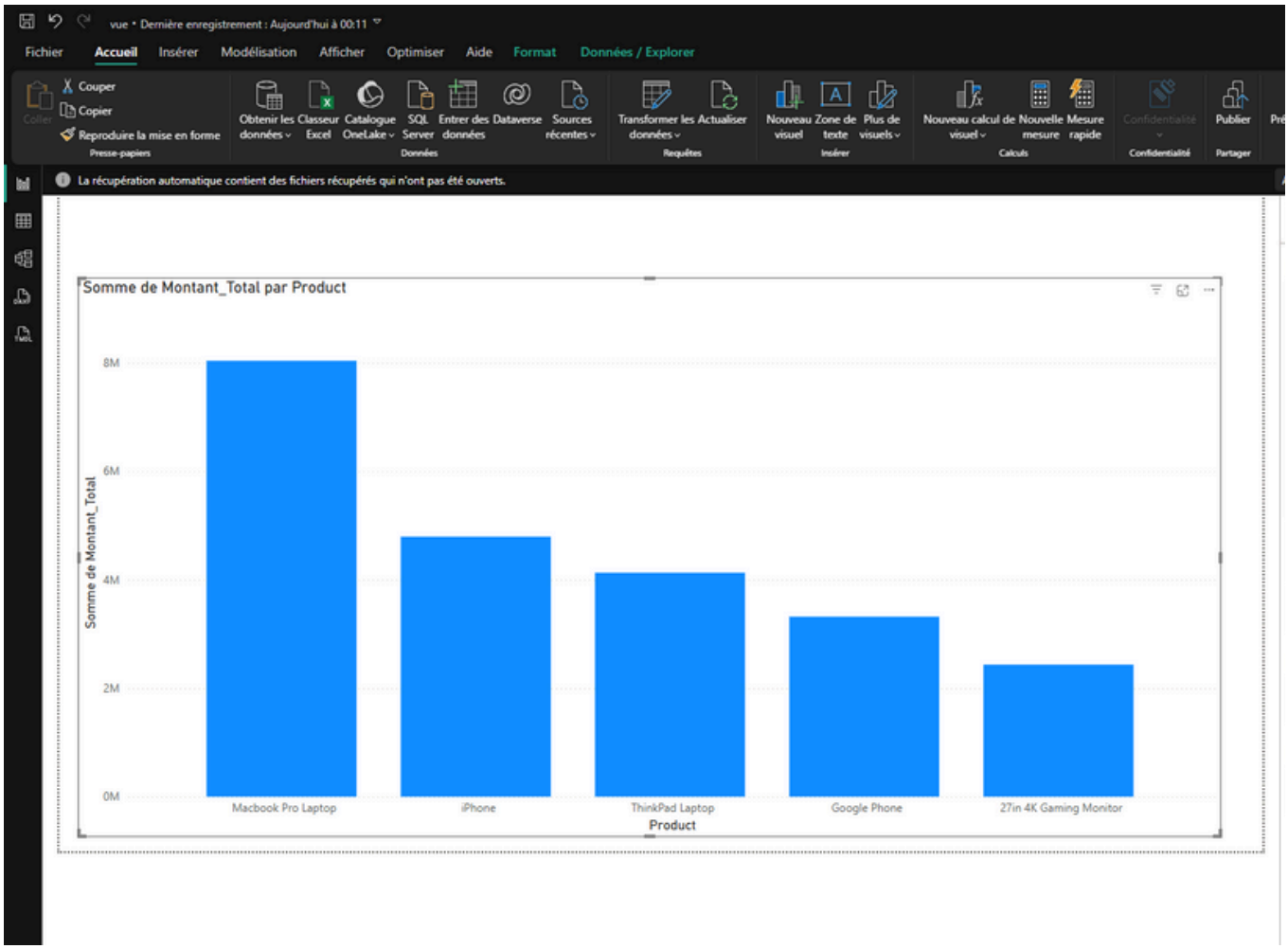
- USB-C Charging Cable et Lightning Charging Cable arrivent en tête.
- Les piles AAA et AA sont également très demandées.
- Wired Headphones complètent ce top 5.

Interprétation :

Ces produits sont :

- Des accessoires universels compatibles avec de nombreux appareils.
- Pas cher
- Essentiels au quotidien

Les 5 produits les plus achetés par montant total



Le graphique montre les produits ayant généré le plus de chiffre d'affaires sur l'année 2019 :

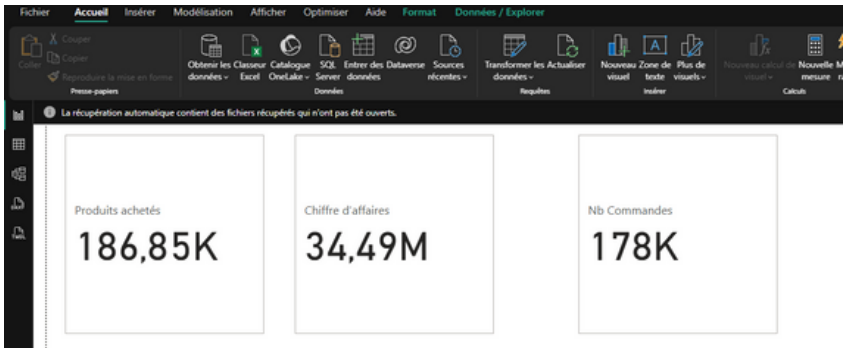
Résultats :

- Le Macbook Pro Laptop avec plus de 8 millions $.
- iPhone
- ThinkPad Laptop
- Google Phone
- 27in 4K Gaming Monitor

Interprétation :

- Ces produits ne sont pas achetés en grande quantité comme les câbles ou piles etc
- mais ils sont chers à l'unité
- Ils sont tres souvent achetés lors d'événements

Nombre total de produits achetés, Chiffre d'affaires et Nombre de commandes sur 2019



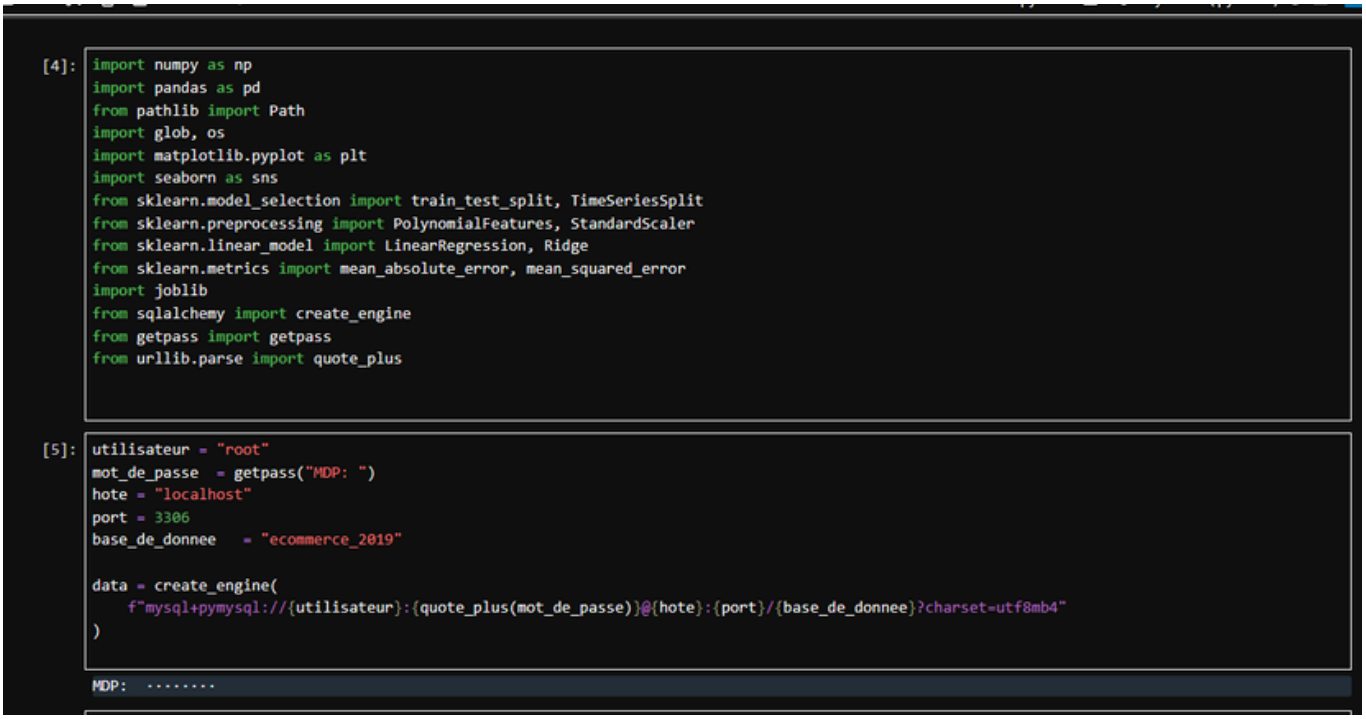
Partie 3 : Modélisation IA

1) Import des librairies

Les librairies nécessaires sont importées, surtout pour la connexion à MySQL, le traitement des données, la modélisation et la visualisation.

2) Connexion à la base MySQL

Une connexion est établie à la base ecommerce_2019 hébergée en local via SQLAlchemy.



```
[4]: import numpy as np
     import pandas as pd
     from pathlib import Path
     import glob, os
     import matplotlib.pyplot as plt
     import seaborn as sns
     from sklearn.model_selection import train_test_split, TimeSeriesSplit
     from sklearn.preprocessing import PolynomialFeatures, StandardScaler
     from sklearn.linear_model import LinearRegression, Ridge
     from sklearn.metrics import mean_absolute_error, mean_squared_error
     import joblib
     from sqlalchemy import create_engine
     from getpass import getpass
     from urllib.parse import quote_plus

[5]: utilisateur = "root"
     mot_de_passe  = getpass("MDP: ")
     hote = "localhost"
     port = 3306
     base_de_donnee   = "ecommerce_2019"

     data = create_engine(
         f"mysql+pymysql:///{utilisateur}:{quote_plus(mot_de_passe)}@{hote}:{port}/{base_de_donnee}?charset=utf8mb4"
     )

     MDP:  ·····
```

3) Fusion des données mensuelles

Pour faciliter l'analyse des ventes sur toute l'année 2019, j'ai regroupé les 12 fichiers mensuels dans une vue SQL appelée sales_2019, créée directement depuis Python à l'aide de SQLAlchemy.

ça permet d'éviter une table temporaire ou une concaténation manuelle.

[8]:
```
from sqlalchemy import text
with data.begin() as connexion:
    connexion.execute(text("""
    CREATE OR REPLACE VIEW sales_2019 AS
        SELECT * FROM sales_january_2019
        UNION ALL SELECT * FROM sales_february_2019
        UNION ALL SELECT * FROM sales_march_2019
        UNION ALL SELECT * FROM sales_april_2019
        UNION ALL SELECT * FROM sales_may_2019
        UNION ALL SELECT * FROM sales_june_2019
        UNION ALL SELECT * FROM sales_july_2019
        UNION ALL SELECT * FROM sales_august_2019
        UNION ALL SELECT * FROM sales_september_2019
        UNION ALL SELECT * FROM sales_october_2019
        UNION ALL SELECT * FROM sales_november_2019
        UNION ALL SELECT * FROM sales_december_2019;
    """))
donnees = pd.read_sql("SELECT * FROM sales_2019", data)
donnees
```

[8]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address
0	141234	iPhone	1	700	01/22/19 21:25	944 Walnut St, Boston, MA 02215
1	141235	Lightning Charging Cable	1	14.95	01/28/19 14:15	185 Maple St, Portland, OR 97035
2	141236	Wired Headphones	2	11.99	01/17/19 13:33	538 Adams St, San Francisco, CA 94016
3	141237	27in FHD Monitor	1	149.99	01/05/19 20:33	738 10th St, Los Angeles, CA 90001
4	141238	Wired Headphones	1	11.99	01/25/19 11:59	387 10th St, Austin, TX 73301
...	...	...	...	...	...	...
186845	319666	Lightning Charging Cable	1	14.95	12/11/19 20:58	14 Madison St, San Francisco, CA 94016
186846	319667	AA Batteries (4-pack)	2	3.84	12/01/19 12:01	549 Willow St, Los Angeles, CA 90001
186847	319668	Vareebadd Phone	1	400	12/09/19 06:43	273 Wilson St, Seattle, WA 98101
186848	319669	Wired Headphones	1	11.99	12/03/19 10:39	778 River St, Dallas, TX 75001
186849	319670	Bose SoundSport Headphones	1	99.99	12/21/19 21:45	747 Chestnut St, Los Angeles, CA 90001

4 Nettoyage des données

Une fois les données fusionnées dans la vue sales_2019 et importées avec Pandas, plusieurs opérations de nettoyage ont été effectuées :

Objectif :

- Corriger les formats (datetime, float, int)
- Supprimer les lignes vides ou corrompues

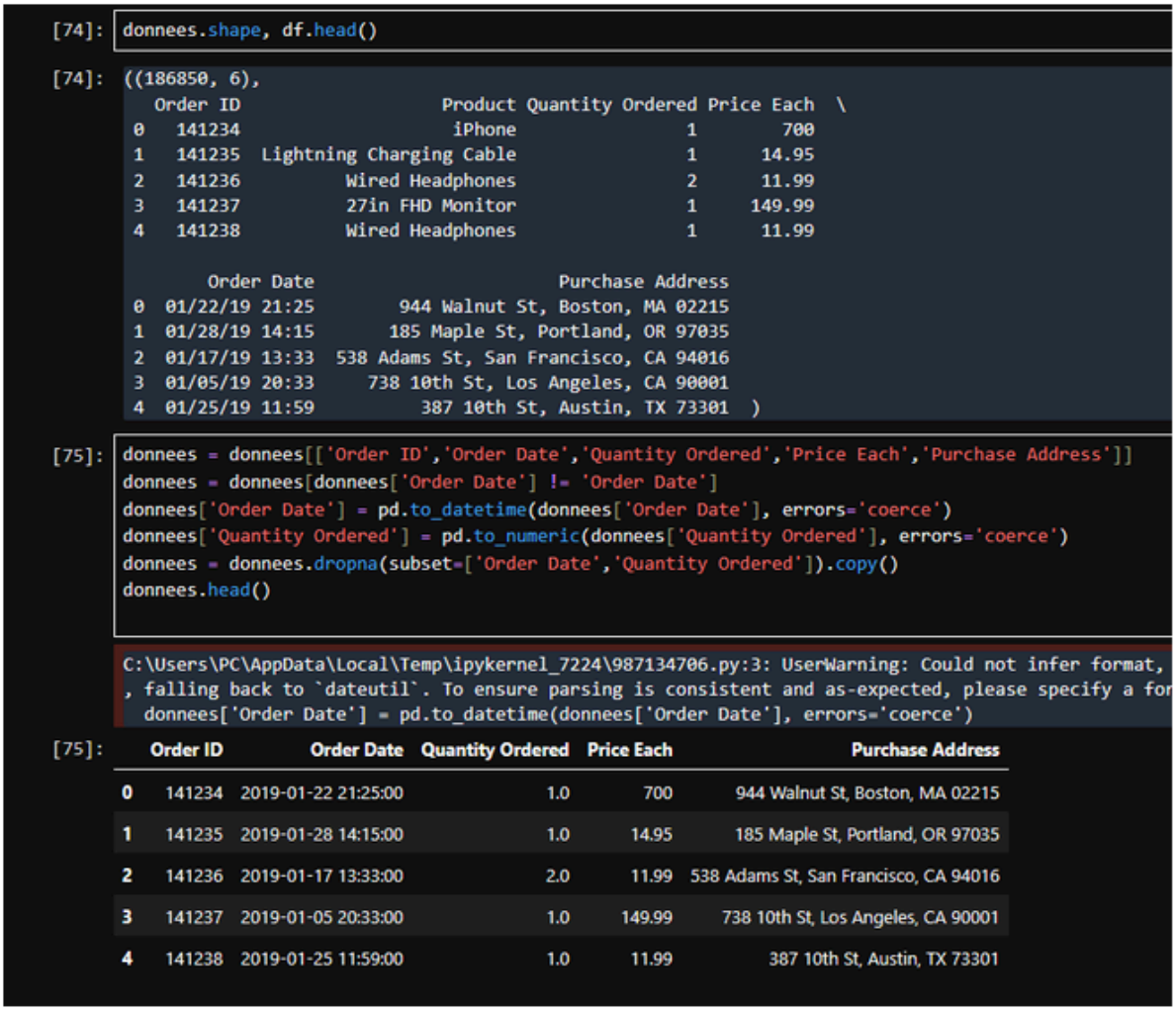
Étapes réalisées :

- Sélection des colonnes utiles
- Conversion du champ Order Date en type datetime
- Conversion des colonnes numériques
- Suppression des lignes avec valeurs manquantes (NaN)

Résultat après nettoyage :

186 850 lignes

- Lignes invalides supprimées
- Dates et quantités correctement formatées
- Données prêtes pour l'analyse ou la modélisation



```
[74]: donnees.shape, df.head()
```

```
[74]: ((186850, 6),
   Order ID                Product Quantity Ordered Price Each  \
0    141234                 iPhone                1        700
1    141235  Lightning Charging Cable                1      14.95
2    141236          Wired Headphones                2      11.99
3    141237          27in FHD Monitor                1     149.99
4    141238          Wired Headphones                1      11.99

       Order Date                     Purchase Address
0  01/22/19 21:25         944 Walnut St, Boston, MA 02215
1  01/28/19 14:15        185 Maple St, Portland, OR 97035
2  01/17/19 13:33  538 Adams St, San Francisco, CA 94016
3  01/05/19 20:33      738 10th St, Los Angeles, CA 90001
4  01/25/19 11:59           387 10th St, Austin, TX 73301  )
```

```
[75]: donnees = donnees[['Order ID','Order Date','Quantity Ordered','Price Each','Purchase Address']]
donnees = donnees[donnees['Order Date'] != 'Order Date']
donnees['Order Date'] = pd.to_datetime(donnees['Order Date'], errors='coerce')
donnees['Quantity Ordered'] = pd.to_numeric(donnees['Quantity Ordered'], errors='coerce')
donnees = donnees.dropna(subset=['Order Date','Quantity Ordered']).copy()
donnees.head()
```

```
C:\Users\PC\AppData\Local\Temp\ipykernel_7224\987134706.py:3: UserWarning: Could not infer format,
, falling back to `dateutil`. To ensure parsing is consistent and as-expected, please specify a for
  donnees['Order Date'] = pd.to_datetime(donnees['Order Date'], errors='coerce')
```

[75]:

	Order ID	Order Date	Quantity Ordered	Price Each	Purchase Address
0	141234	2019-01-22 21:25:00	1.0	700	944 Walnut St, Boston, MA 02215
1	141235	2019-01-28 14:15:00	1.0	14.95	185 Maple St, Portland, OR 97035
2	141236	2019-01-17 13:33:00	2.0	11.99	538 Adams St, San Francisco, CA 94016
3	141237	2019-01-05 20:33:00	1.0	149.99	738 10th St, Los Angeles, CA 90001
4	141238	2019-01-25 11:59:00	1.0	11.99	387 10th St, Austin, TX 73301

5) Modélisation IA

5.1 Préparation des données temporelles

L'objectif est de prédire le nombre de commandes par jour sur l'année 2019, à partir de la colonne Order Date.

Agrégation quotidienne des commandes :

Les données ont été regroupées par date, avec une somme des quantités commandées (Quantity Ordered)

Résultat :

Un DataFrame de 366 lignes, représentant les jours de l'année 2019

5.2 Encodage temporel

Pour simplifier la modélisation, la date a été remplacée par un index temporel j, de 0 à 365

5.3 Séparation en jeu d'entraînement / test

Les données ont été séparées en deux jeux :

- 80 % pour l'entraînement
- 20 % pour le test final

En respectant l'ordre chronologique

Résultat :

- Entraînement : 292 jours
- Test : 74 jours

[79]:
```
quotidien = (
    donnees
      .assign(Jour = donnees['Order Date'].dt.floor('D'))
      .groupby('Jour', as_index=False)['Quantity Ordered'].sum()
      .rename(columns={'Quantity Ordered': 'y'})
      .sort_values('Jour')
)

quotidien.head(), quotidien.shape
```

[79]:
```
(        Jour      y
 0 2019-01-01  343.0
 1 2019-01-02  368.0
 2 2019-01-03  330.0
 3 2019-01-04  330.0
 4 2019-01-05  355.0,
 (366, 2))
```

[80]:
```
import numpy as np
quotidien['j'] = np.arange(len(quotidien))
X = quotidien[['j']]
y = quotidien['y']
```

[81]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, shuffle=False
)
X_train.shape, X_test.shape
```

[81]:
```
((292, 1), (74, 1))
```

5.1 Régression Linéaire

Une régression linéaire simple a été utilisée afin de modéliser la tendance des commandes sur l'année.

- Entraînement du modèle
- Evaluation du modèle

Résultats :

- MAE (erreur absolue moyenne) : 213.67
- RMSE (erreur quadratique moyenne) : 245.69

ça veut dire qu'il une erreur moyenne de plus de 200 commandes, ce qui montre que le modèle linéaire est trop simple pour bien représenter la dynamique des ventes

5.2 Visualisation des prédictions

Le graphe ci-dessous montre la différence entre :

- Les valeurs réelles (bleu)
- Les valeurs prédites (orange)

La régression linéaire n'arrive pas à capturer les variations saisonnières

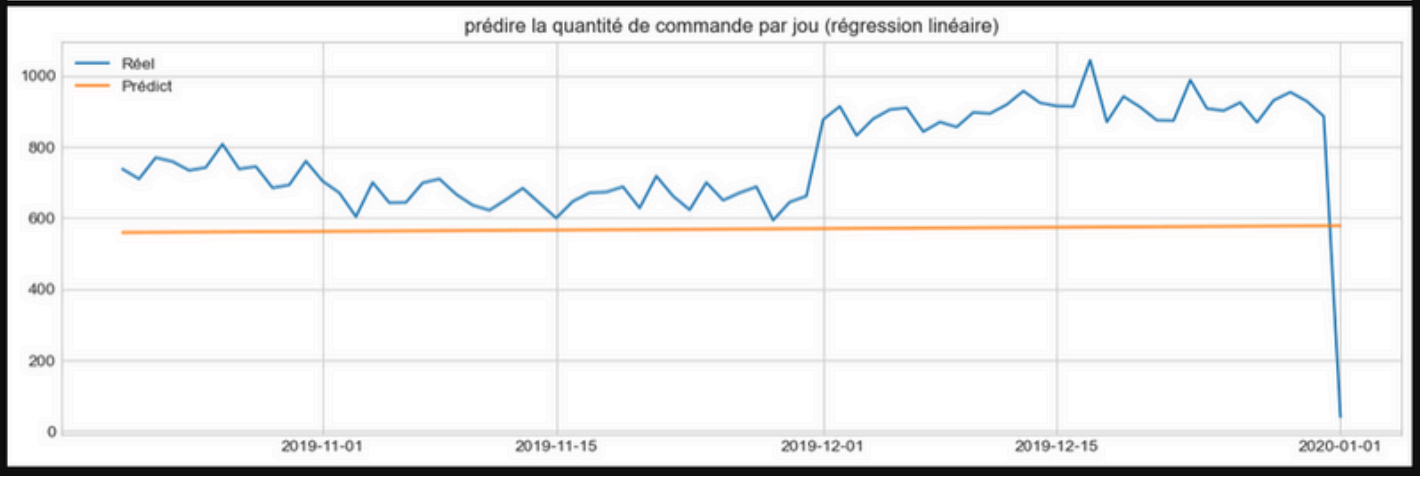
[82]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error

reg = LinearRegression()
reg.fit(X_train, y_train)
y_pred = reg.predict(X_test)

mae  = mean_absolute_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)
print(f"MAE: {mae:.2f} | RMSE: {rmse:.2f}")
```

```
MAE: 213.67 | RMSE: 245.69
C:\Users\PC\anaconda3\Lib\site-packages\sklearn\metrics\_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean squared error, use the function'root_mean_squared_error'.
  warnings.warn(
```

[86]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(12,4))
dates_test = quotidien['Jour'].iloc[-len(y_test):]
plt.plot(dates_test, y_test, label='Réel')
plt.plot(dates_test, y_pred, label='Prédict')
plt.title("prédire la quantité de commande par jou (régression linéaire)")
plt.legend()
plt.tight_layout()
plt.show()
```



6.1 Régression polynomiale

Pour améliorer les performances de la régression linéaire, une régression polynomiale de degré 2 a été testée.

Transformation des données :
Les données ont d'abord été transformées à l'aide de PolynomialFeatures de degré 2
Entraînement du modèle
Prédiction

Visualisation :
Le graphique suivant montre :

- En bleu : les données réelles (commandes par jour)
- En rouge : la courbe prédite par le modèle polynômial

6.2 Interprétation

Ce modèle permet de mieux capturer les variations et tendances au fil des jours que le modèle linéaire. On distingue plus clairement :

- Les variations cycliques des ventes
- Les hausses progressives

ça pourrait être amélioré avec un degré plus élevé ou remplacé par un modèle de séries temporelles

```
[84]: from sklearn.preprocessing import PolynomialFeatures
      from sklearn.linear_model import LinearRegression
      import matplotlib.pyplot as plt
      import numpy as np

      x = quotidien['j'].values.reshape(-1, 1)
      y = quotidien['y'].values.reshape(-1, 1)

      poly_features = PolynomialFeatures(degree=2, include_bias=False)
      x_poly = poly_features.fit_transform(x)

      reg = LinearRegression()
      reg.fit(x_poly, y)

      x_vals = np.linspace(x.min(), x.max(), 500).reshape(-1, 1)
      x_vals_poly = poly_features.transform(x_vals)
      y_vals = reg.predict(x_vals_poly)

      plt.figure(figsize=(10,6))
      plt.title("Une regression polynomiale", size=16)
      plt.scatter(x, y, s=20)
      plt.plot(x_vals, y_vals, c="red")
      plt.show()
```