Prédiction du chiffre d’affaires afin d'anticiper le recrutement, renforcer le suivi des machines et d'augmenter les stocks des produits pour répondre à la demande client.

Tests avec le modèle Random Forest

Premier test : Prédiction journalière du chiffre d’affaires réel vs Prédictions avec le modèle Random Forest.

Objectif : Ce script récupère le chiffre d’affaires quotidien depuis BigQuery. Entraîne un modèle Random Forest pour prédire le CA à partir des composantes de la date (année, mois, jour). Et enfin, évalue la précision du modèle avec des métriques d’erreur (MAE, MSE, RMSE, R^2) et un aperçu des prédictions vs réel.

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np
from google.cloud import bigquery
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
import os
import matplotlib.pyplot as plt
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

#  Définir la variable d'environnement pour la clé JSON
os.environ["GOOGLE_APPLICATION_CREDENTIALS"] = "C:/Users/sigle/Desktop/Capsule cours/Projet/DAG airflow/

def train_rf_from_bigquery():
    client  =  bigquery.Client(project="rising-mercury-463817-s5")

    query = """
    SELECT
      dt.date AS date,
      EXTRACT(YEAR FROM dt.date) AS annee,
      EXTRACT(MONTH FROM dt.date) AS mois,
      EXTRACT(DAY FROM dt.date) AS jour,
      ROUND(SUM(dc.quantite * p.prix), 2) AS chiffre_affaires
    FROM `rising-mercury-463817-s5.carttrend_rawdata.facts_commandes` c
    JOIN `rising-mercury-463817-s5.carttrend_rawdata.dim_date` dt
      ON dt.id_date = c.id_date_commande
    JOIN `rising-mercury-463817-s5.carttrend_rawdata.dim_details_commandes` dc
      ON c.id_commande = dc.id_commande
    JOIN `rising-mercury-463817-s5.carttrend_rawdata.dim_produits` p
      ON dc.id_details_produits = p.id_produit
    WHERE LOWER(c.statut_commande) NOT IN ('annulée', 'cancelled')
    GROUP BY dt.date
    ORDER BY dt.date
    """

    # Charger les données
    df = client.query(query).to_dataframe()
    df["date"] = pd.to_datetime(df["date"])

    # Features
    X = df[['annee', 'mois', 'jour']]
    y = df['chiffre_affaires']

    # Split train/test
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42
    )

    # Modèle
```

```
    rf = RandomForestRegressor(n_estimators=100, random_state=42)
    rf.fit(X_train, y_train)

    # Prédictions
    y_pred = rf.predict(X_test)

    return rf, X_test, y_test, y_pred, df

#  Exécution
rf_model, X_test, y_test, y_pred, df = train_rf_from_bigquery()

# Afficher un aperçu
print("Aperçu des prédictions :")
print(pd.DataFrame({
    "Date": X_test.apply(lambda row: f"{int(row.annee)}-{int(row.mois)}-{int(row.jour)}", axis=1),
    "Chiffre_affaires réel": y_test,
    "Prédiction": y_pred
}).head())

# Calcul des métriques
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print("Évaluation du modèle Random Forest :")
print(f"MAE  : {mae:.2f}")
print(f"MSE  : {mse:.2f}")
print(f"RMSE : {rmse:.2f}")
print(f"R²   : {r2:.4f}")
```

```
C:\Users\sigle\AppData\Roaming\Python\Python312\site-packages\google\cloud\bigquery\table.py:1965:  UserWa
rning: BigQuery Storage module not found, fetch data with the REST endpoint instead.
  warnings.warn(
```

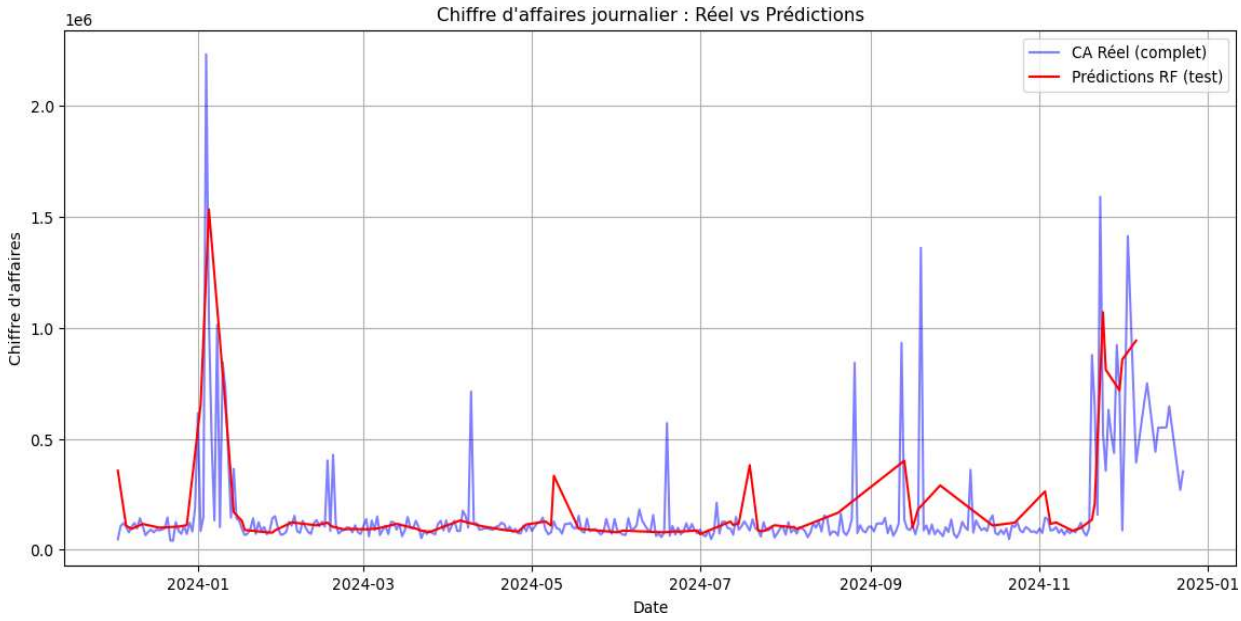
```
Aperçu des prédictions :
           Date  Chiffre_affaires réel    Prédiction
290    2024-9-18              114642.84  1.819683e+05
357   2024-11-24              527167.59  1.070395e+06
261    2024-8-20               64057.84  1.664751e+05
157     2024-5-8               78103.50  1.082882e+05
145    2024-4-26               74399.91  8.114387e+04
Évaluation du modèle Random Forest :
MAE  : 110111.67
MSE  : 44185428236.30
RMSE : 210203.30
R²   : -0.1791
```

Graphique plot : Courbe du CA journalier Réel vs Prédictions (Random Forest)

Objectif : Ce script permet de prédire le chiffre d'affaires (CA) journalier à partir d'un historique, et comparer les valeurs réelles avec les valeurs prédites avec le modèle Random Forest.

```
# Reconstruire la colonne "date" correctement
X_test_with_date = X_test.copy()
X_test_with_date["prediction"] = y_pred

# Renommer les colonnes pour qu'elles soient reconnues par to_datetime
X_test_with_date = X_test_with_date.rename(columns={"annee": "year", "mois": "month", "jour": "day"})

X_test_with_date["date"] = pd.to_datetime(
    X_test_with_date[["year", "month", "day"]]
)

# Trier pour garder un ordre temporel
X_test_with_date = X_test_with_date.sort_values("date")

# Tracé comparatif sans CA réel du test
plt.figure(figsize=(14,6))
plt.plot(df["date"], df["chiffre_affaires"], label="CA Réel (complet)", color="blue", alpha=0.5)
plt.plot(X_test_with_date["date"], X_test_with_date["prediction"], label="Prédictions RF (test)", color=
```

```
plt.title("Chiffre d'affaires journalier : Réel vs Prédictions")
plt.xlabel("Date")
plt.ylabel("Chiffre d'affaires")
plt.legend()
plt.grid(True)
plt.show()
```



Résultats du graphique :

- **Observations principales** :

Tendance générale captée Le modèle RF arrive à suivre la tendance globale du CA journalier : quand ça monte, la prédiction monte aussi, et inversement. Donc il apprend bien le "signal moyen", mais il lisse beaucoup les variations.

Pics mal reproduits
Les très gros pics du CA (par ex. en janvier, novembre et fin d'année) sont largement sous-estimés par le modèle. Typique des Random Forests : elles sont assez "conservatrices" et n'aiment pas extrapoler au-delà de ce qu'elles ont vu en moyenne.

Volatilité manquante
Le CA réel est très bruité / variable (beaucoup de pics soudains).
Le modèle RF fournit des prédictions beaucoup plus lissées, il perd la dynamique réelle du marché.

Décalages temporels
On remarque que parfois la prédiction monte un peu avant ou après le vrai pic, comme si le modèle n'arrivait pas à bien situer le moment exact.

- **Interprétation** :

Le RF capte la tendance de fond, mais pas les valeurs extrêmes ni la saisonnalité fine.
Cela peut venir de :
Manque de features temporelles (jour de la semaine, mois, saison, vacances, promos, ...)
Absence de variables exogènes (pub, météo, événements spéciaux, ...)
RF n'est pas le meilleur modèle pour séries temporelles (il n'extrapole pas et ne capte pas l'auto-corrélation forte dans les séquences).

- **Améliorations possibles** :

Ajouter : jour de la semaine, mois, périodes de soldes, jours fériés, effet fin de mois, ...
Essayer d'autres modèles : Gradient Boosting (XGBoost) souvent meilleurs que RF.

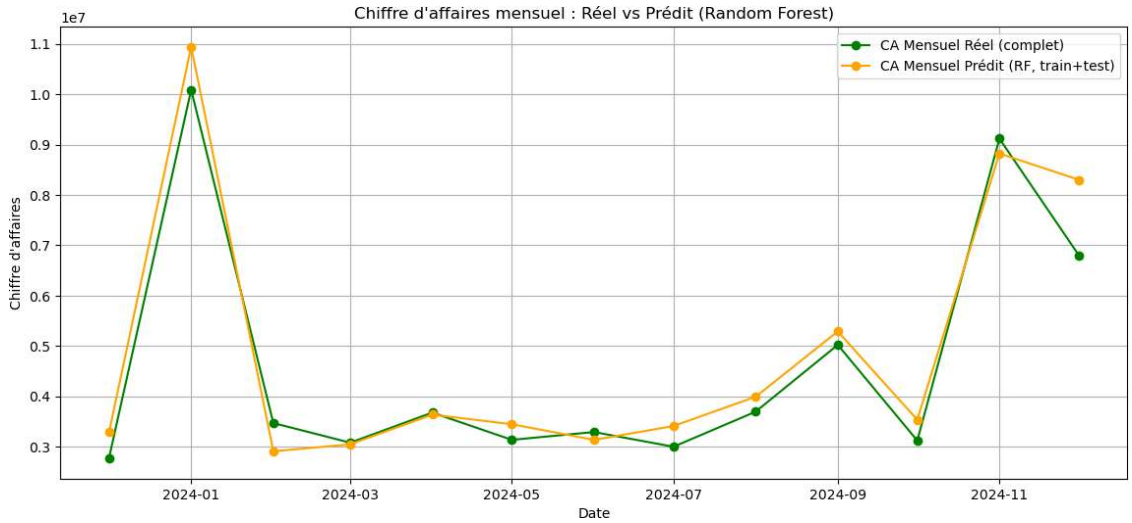
Second test : Comparaison du Chiffre d'Affaires Mensuel réel vs Prédictions avec le modèle Random Forest.

Second test : Comparaison du Chiffre d'Affaires Mensuel réel vs Prédictions avec le modèle Random Forest.

Objectif : Ce script permet de prédire et analyser le chiffre d'affaires mensuel. Il récupère les données journalières depuis BigQuery, entraîne un modèle Random Forest Regressor sur les variables temporelles, puis génère des prédictions sur l'ensemble de la période. Les résultats sont ensuite agrégés au niveau mensuel afin de comparer le CA réel et le CA prédit. Enfin, le graphique illustre la capacité du modèle à reproduire les tendances mensuelles du chiffre d'affaires.

Graphique plot : Comparaison du Chiffre d'Affaires Mensuel réel vs Prédictions (Random Forest)

```
# --- Prédictions sur tout le dataset ---
X = df[['annee', 'mois', 'jour']]
y = df['chiffre_affaires']

df["prediction"] = rf_model.predict(X)

# --- Agrégation mensuelle ---
df_mensuel = df.groupby(df["date"].dt.to_period("M"))["chiffre_affaires"].sum().reset_index()
df_mensuel["date"] = df_mensuel["date"].dt.to_timestamp()

df_pred_mensuel = df.groupby(df["date"].dt.to_period("M"))["prediction"].sum().reset_index()
df_pred_mensuel["date"] = df_pred_mensuel["date"].dt.to_timestamp()

# --- Tracé ---
plt.figure(figsize=(14,6))
plt.plot(df_mensuel["date"], df_mensuel["chiffre_affaires"], marker="o",
         label="CA Mensuel Réel (complet)", color="green")

plt.plot(df_pred_mensuel["date"], df_pred_mensuel["prediction"], marker="o",
         label="CA Mensuel Prédit (RF, train+test)", color="orange")

plt.title("Chiffre d'affaires mensuel : Réel vs Prédit (Random Forest)")
plt.xlabel("Date")
plt.ylabel("Chiffre d'affaires")
plt.legend()
plt.grid(True)
plt.show()
```



Résultats du graphique :

- **Analyse des résultats** :

Globalement, la courbe orange (prédictions) suit assez bien la courbe verte (réel). Cela montre que le modèle a réussi à apprendre la saisonnalité et la dynamique moyenne mensuelle du CA.

Pic de janvier 2024 : Le modèle surestime légèrement le pic (orange au-dessus du vert). Mais il a bien identifié qu'il y avait un pic exceptionnel.

Période mars – juillet 2024 : Les prédictions sont assez proches du réel. On observe une légère sous-estimation en mai-juin mais rien de dramatique. Cela montre que le modèle est assez robuste dans les périodes stables.
Août – septembre 2024 : Le modèle suit la remontée du CA en septembre, mais prédit un peu plus haut que le réel. Cela traduit une tendance légèrement amplifiée par le modèle. Cela traduit une tendance légèrement amplifiée par le modèle.
Fin d'année (octobre – décembre 2024) : Le modèle capte bien le gros pic de novembre, mais le réduit un peu par rapport au réel. En décembre, il surestime légèrement le CA (orange > vert). Typique du Random Forest qui a tendance à lisser et mal capter les extrêmes.

- **Interprétation** :

Le modèle est fiable pour prévoir la tendance mensuelle globale, ce qui est utile pour un pilotage stratégique. En revanche, il a du mal avec : les pics exceptionnels (ex : janvier, novembre), il les capte mais pas avec la bonne amplitude. Certaines périodes de baisse ou de reprise, il sur- ou sous-estime légèrement. Cela confirme que le Random Forest est bon pour lisser et capter la saisonnalité moyenne, mais pas optimal pour les valeurs extrêmes.

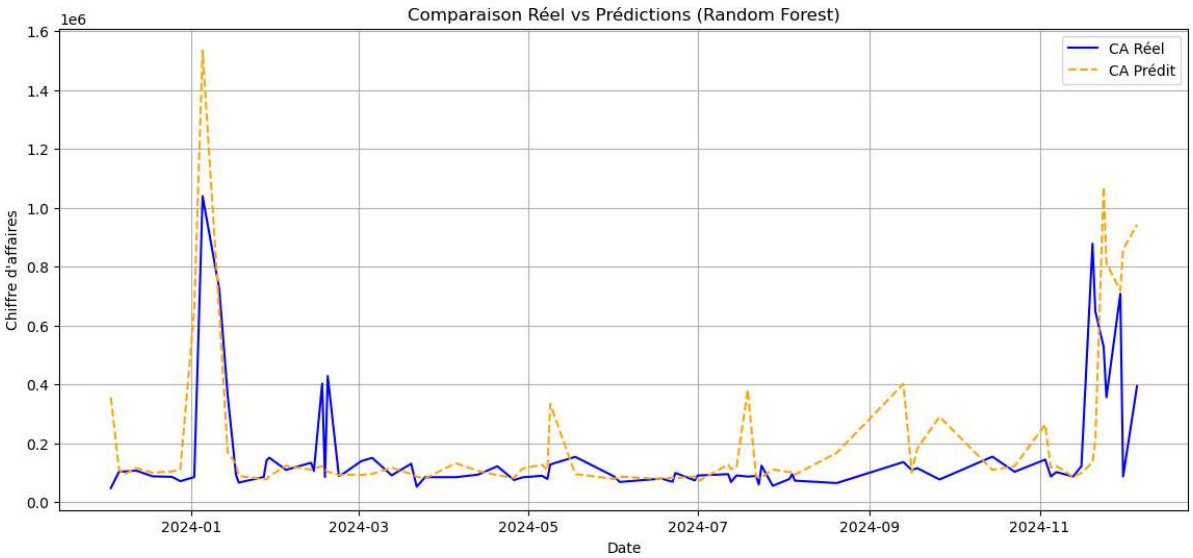
- **Conclusion** :

Le modèle Random Forest reproduit bien la dynamique mensuelle du CA. Il est pertinent pour prévoir une tendance générale mensuelle. Mais il n'est pas parfait pour prédire les pics exceptionnels.

Graphique plot : Comparaison Réel vs Prédiction du CA avec le modèle Random Forest.

Objectif : Ce script met en valeur la performance du modèle Random Forest en comparant graphiquement le chiffre d'affaires réel vs prédits, ce qui complète les métriques chiffrées (MAE, RMSE, R^2) par une évaluation visuelle de la qualité des prédictions.

```
X_test_dates = pd.to_datetime(X_test.apply(lambda row: f"{int(row.annee)}-{int(row.mois)}-{int(row.jour)
df_pred = pd.DataFrame({
    "Date": X_test_dates,
    "Réel": y_test.values,
    "Prédiction": y_pred
}).sort_values("Date")

plt.figure(figsize=(14,6))
plt.plot(df_pred["Date"], df_pred["Réel"], label="CA Réel", color="blue")
plt.plot(df_pred["Date"], df_pred["Prédiction"], label="CA Prédit", color="orange", linestyle="--")
plt.title("Comparaison Réel vs Prédictions (Random Forest)")
plt.xlabel("Date")
plt.ylabel("Chiffre d'affaires")
plt.legend()
plt.grid(True)
plt.show()
```



Résultats du graphique :

- **Alignement global** :

La courbe orange (prédictions) suit en partie la courbe bleue (réel), surtout sur les tendances générales. Cela signifie que le modèle capture assez bien la structure moyenne du CA journalier.

- **Précision sur les pics** :

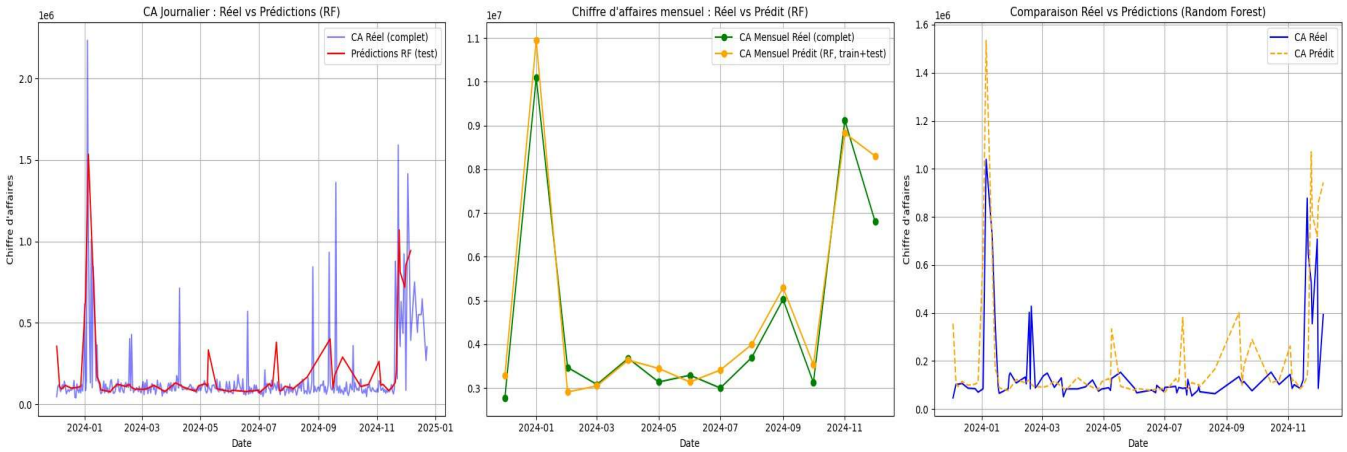
Le modèle a du mal à reproduire fidèlement les pics extrêmes (ex : janvier, mai, novembre). Il surestime certains pics (exemple : début janvier, novembre) et sous-estime d'autres (ex : fin décembre, mi-septembre). Les Random Forests ont tendance à lisser les valeurs et ne sont pas idéales pour des événements rares et extrêmes.

- **Périodes stables** :

Dans les périodes sans pics (février à juillet), le modèle prédit un CA relativement proche de la réalité. Mais il reste parfois un biais de surestimation, avec la courbe orange légèrement au-dessus.

Résultats des tests avec Random Forest

```
import matplotlib.pyplot as plt

fig, axes = plt.subplots(1, 3, figsize=(22,6), sharey=False)

# 1er subplot : CA journalier
axes[0].plot(df["date"], df["chiffre_affaires"], label="CA Réel (complet)", color="blue", alpha=0.5)
axes[0].plot(X_test_with_date["date"], X_test_with_date["prediction"], label="Prédictions RF (test)", co
axes[0].set_title("CA Journalier : Réel vs Prédictions (RF)")
axes[0].set_xlabel("Date")
axes[0].set_ylabel("Chiffre d'affaires")
axes[0].legend()
axes[0].grid(True)

# 2ème subplot : CA mensuel
axes[1].plot(df_mensuel["date"], df_mensuel["chiffre_affaires"], marker="o",
             label="CA Mensuel Réel (complet)", color="green")
axes[1].plot(df_pred_mensuel["date"], df_pred_mensuel["prediction"], marker="o",
             label="CA Mensuel Prédit (RF, train+test)", color="orange")
axes[1].set_title("Chiffre d'affaires mensuel : Réel vs Prédit (RF)")
axes[1].set_xlabel("Date")
axes[1].legend()
axes[1].grid(True)

# 3ème subplot : Comparaison Réel vs Prédictions (Random Forest)
axes[2].plot(df_pred["Date"], df_pred["Réel"], label="CA Réel", color="blue")
axes[2].plot(df_pred["Date"], df_pred["Prédiction"], label="CA Prédit", color="orange", linestyle="--")
axes[2].set_title("Comparaison Réel vs Prédictions (Random Forest)")
axes[2].set_xlabel("Date")
axes[2].set_ylabel("Chiffre d'affaires")
axes[2].legend()
axes[2].grid(True)

plt.tight_layout()
plt.show()
```



Résultats globaux du test :

- **Analyse du chiffre d'affaires réel (CA)** :

Graphique journalier : On voit beaucoup de pics très élevés et irréguliers. La majorité des jours restent à un CA relativement bas, mais ponctuellement certains jours explosent à plusieurs centaines de milliers voire millions. Cela traduit une série temporelle très volatile et hétérogène, probablement liée à des événements exceptionnels (ex: promos, lancements produits, saisonnalité forte).

Graphique mensuel : Le modèle est fiable pour prévoir la tendance mensuelle globale, ce qui est utile pour un pilotage stratégique. En revanche, il a du mal avec : les pics exceptionnels (ex : janvier, novembre), il les capte mais pas avec la bonne amplitude. Certaines périodes de baisse ou de reprise, il sur- ou sous-estime légèrement. Cela confirme que le Random Forest est bon pour lisser et capter la saisonnalité moyenne, mais pas optimal pour les valeurs extrêmes.

Comparaison Réel vs Prédictions (Random Forest) : La courbe des prédictions (orange pointillé) suit la tendance globale, mais, elle lisse fortement les pics extrêmes (ex : janvier, novembre). Parfois elle surestime (ex: janvier 2024) ou sous-estime (fin novembre 2024). Le modèle capte la tendance générale, mais n'arrive pas à anticiper correctement les valeurs extrêmes, ce qui était attendu vu la nature aléatoire des pics.

- **Métriques d'évaluation** :

MAE = 110k : en moyenne, le modèle se trompe de ~110 000 € par jour.

RMSE = 210k : les grosses erreurs pèsent lourd, ce qui reflète les pics mal prédits.

MSE énorme (4.4e11) : confirme que les erreurs extrêmes dominent.

R^2 = -0.1791 : très mauvais score, cela veut dire que le modèle fait pire qu'une simple moyenne constante.

- **Conclusion sur ce test** :

Le modèle ne parvient pas à bien expliquer la variance du CA. Il capte la tendance générale, mais il n'explique pas la dynamique réelle du CA journalier, surtout les pics.
Les pics extrêmes sont probablement liés à des facteurs non inclus dans les variables explicatives (ex : campagnes marketing, événements externes, fêtes saisonnières).
Les mauvaises métriques (R^2 négatif) montrent que le modèle n'est pas adapté en l'état pour des prévisions fiables.

- **Recommandation** pour améliorer le modèle :

Intégrer variables comme jours fériés, saisonnalité, promos, météo, campagnes marketing.
Utiliser un modèle adapté aux séries temporelles : XGBoost

Troisième test : Prédiction mensuelle du chiffre d'affaires avec Random Forest avec l'ajout des variables soldes, noel, black_friday et aucun_evenement.

Objectif : Prédire le chiffre d'affaires. Ce script permet d'extraire les ventes depuis BigQuery, d'agréger par mois pour lisser les fluctuations quotidiennes, d'ajouter des variables événementielles (Soldes, Noël, Black Friday). On va entraîner un Random Forest pour prédire le chiffre d'affaires **mensuel**, puis évaluer sa précision avec des métriques (MAE, MSE, REMSE, R^2), pour enfin, identifier les facteurs qui influencent le plus le chiffre d'affaires.

```
import os
import pandas as pd
import numpy as np
from google.cloud import bigquery
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Définir la variable d'environnement pour la clé JSON
os.environ["GOOGLE_APPLICATION_CREDENTIALS"] = "C:/Users/sigle/Desktop/Capsule cours/Projet/DAG airflow/

def train_rf_mensuel_from_bigquery():
    client  =  bigquery.Client(project="rising-mercury-463817-s5")

    query = """
    SELECT
```

```
  dt.date AS date,
  EXTRACT(YEAR FROM dt.date) AS annee,
  EXTRACT(MONTH FROM dt.date) AS mois,
  ROUND(SUM(dc.quantite * p.prix), 2) AS chiffre_affaires,
  MAX(CASE WHEN cpg.evenement_type = "Soldes" THEN 1 ELSE 0 END) AS soldes,
  MAX(CASE WHEN cpg.evenement_type = "Noël" THEN 1 ELSE 0 END) AS noel,
  MAX(CASE WHEN cpg.evenement_type = "Black Friday" THEN 1 ELSE 0 END) AS black_friday
FROM `rising-mercury-463817-s5.carttrend_rawdata.facts_commandes` c
JOIN `rising-mercury-463817-s5.carttrend_rawdata.dim_date` dt
  ON dt.id_date = c.id_date_commande
JOIN `rising-mercury-463817-s5.carttrend_rawdata.dim_details_commandes` dc
  ON c.id_commande = dc.id_commande
JOIN `rising-mercury-463817-s5.carttrend_rawdata.dim_produits` p
  ON dc.id_details_produits = p.id_produit
LEFT JOIN `rising-mercury-463817-s5.carttrend_rawdata.facts_campaigns` cpg
  ON dt.id_date = cpg.id_date
WHERE LOWER(c.statut_commande) NOT IN ('annulée', 'cancelled')
GROUP BY dt.date, annee, mois
ORDER BY dt.date
"""

df = client.query(query).to_dataframe()
df["date"] = pd.to_datetime(df["date"])

# Agrégation mensuelle
df_mois = df.groupby(["annee", "mois"], as_index=False).agg({
    "chiffre_affaires": "sum",
    "soldes": "max",
    "noel": "max",
    "black_friday": "max"
})

# Colonne événement vide
df_mois["aucun_evenement"] = ((df_mois["soldes"] == 0) &
                              (df_mois["noel"] == 0) &
                              (df_mois["black_friday"] == 0)).astype(int)

# Colonne date_mois pour le graphique
df_mois["date_mois"] = pd.to_datetime(df_mois["annee"].astype(str) + "-" + df_mois["mois"].astype(st

print("Aperçu des données mensuelles :")
print(df_mois.head())

# Features & Target
X = df_mois[["annee", "mois", "soldes", "noel", "black_friday", "aucun_evenement"]]
y = df_mois["chiffre_affaires"]

# Train/test chronologique
train_size = int(len(df_mois) * 0.8)
X_train, X_test = X.iloc[:train_size], X.iloc[train_size:]
y_train, y_test = y.iloc[:train_size], y.iloc[train_size:]

# Modèle Random Forest
rf = RandomForestRegressor(
    n_estimators=300,
    max_depth=8,
    random_state=42
)
rf.fit(X_train, y_train)

y_pred = rf.predict(X_test)

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

print("\nÉvaluation du modèle Random Forest (mensuel) :")
print(f"MAE  : {mae:,.2f}")
print(f"MSE  : {mse:,.2f}")
print(f"RMSE : {rmse:,.2f}")
print(f"R²   : {r2:.4f}")

#² des variables
```

```
    feature_importances = pd.DataFrame({
        "Feature": X.columns,
        "Importance": rf.feature_importances_
    }).sort_values(by="Importance",  ascending=False)

    print("\nImportance des variables :")
    print(feature_importances)

    return rf, X_test, y_test, y_pred, df_mois, train_size


# Lancer l'entraînement
if __name__ == "__main__":
    rf, X_test, y_test, y_pred, df_mois, train_size = train_rf_mensuel_from_bigquery()
```

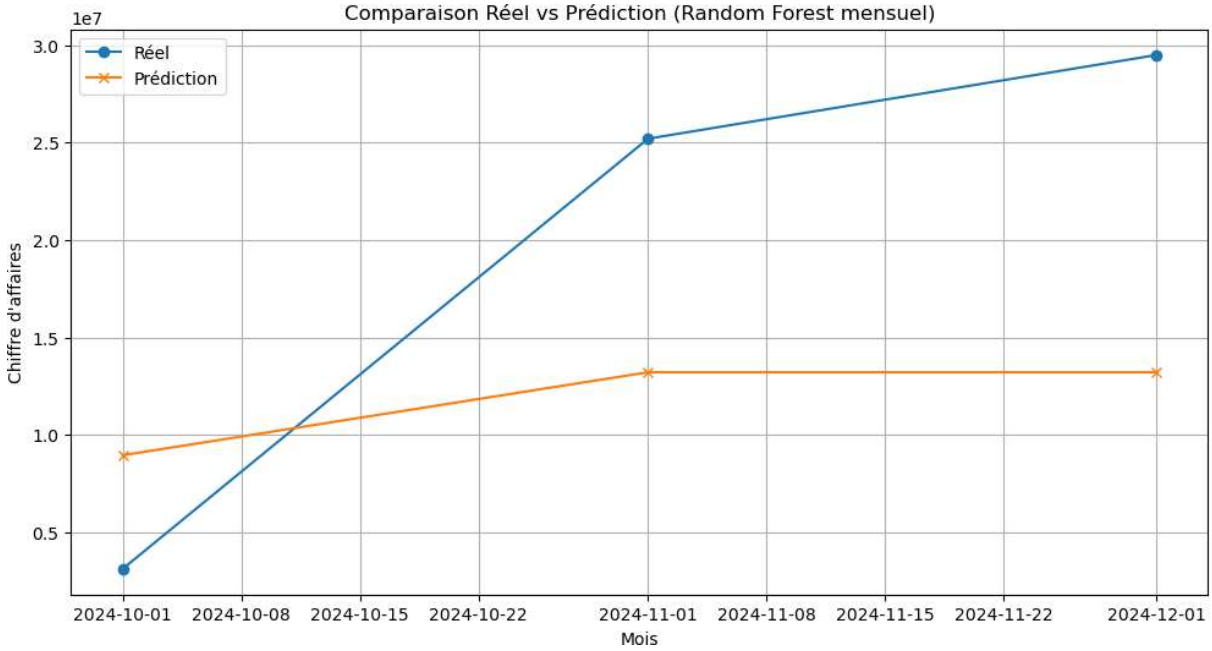
```
C:\Users\sigle\AppData\Roaming\Python\Python312\site-packages\google\cloud\bigquery\table.py:1965: UserWa
rning: BigQuery Storage module not found, fetch data with the REST endpoint instead.
  warnings.warn(
```

```
Aperçu des données mensuelles :
   annee  mois  chiffre_affaires  soldes  noel  black_friday  aucun_evenement  \
0   2023    12        2777117.16       0     0             0                1
1   2024     1       36437674.59       1     0             0                0
2   2024     2        4703410.75       0     0             0                1
3   2024     3        3081595.51       0     0             0                1
4   2024     4        5816515.56       0     0             0                1

   date_mois
0 2023-12-01
1 2024-01-01
2 2024-02-01
3 2024-03-01
4 2024-04-01

Évaluation du modèle Random Forest (mensuel) :
MAE  : 11,365,950.57
MSE  : 147,590,254,506,203.12
RMSE : 12,148,672.95
R²   : -0.1058

Importance des variables :
           Feature  Importance
1             mois    0.552365
2           soldes    0.202687
5  aucun_evenement    0.201262
0            annee    0.043686
3             noel    0.000000
4     black_friday    0.000000
```

Graphique plot : Comparaison CA mensuel réel vs prédiction avec l'ajout des variables soldes, noel, black_friday et aucun_evenement.

Objectif : Ce script permet de visualiser les performances mensuelles du modèle Random Forest avec l'ajout des variables soldes, noel, black_friday et aucun_evenement.

```
import matplotlib.pyplot as plt

plt.figure(figsize=(12,6))
plt.plot(df_mois["date_mois"].iloc[train_size:], y_test, label="Réel", marker="o")
plt.plot(df_mois["date_mois"].iloc[train_size:], y_pred, label="Prédiction", marker="x")
plt.title("Comparaison Réel vs Prédiction (Random Forest mensuel)")
plt.xlabel("Mois")
plt.ylabel("Chiffre d'affaires")
plt.legend()
plt.grid(True)
plt.show()
```



Résultats du graphique :

- **Lecture du graphique** :

Valeurs réelles :
Octobre 2024 : autour de 3M € de CA.
Novembre 2024 : bond important à ~25M €.
Décembre 2024 : encore en hausse vers ~30M €.
Le CA réel montre une forte saisonnalité en fin d'année (probablement Noël, Black Friday, etc.).

Les valeurs prédites :
Entre 9M et 13M sur les 3 mois.
La courbe est plate et sous-estime énormément les pics réels.
Le modèle n'arrive pas à reproduire l'énorme envolée du CA réel.

Écart entre Réel et Prédiction :
Octobre : le modèle surestime le CA (prévoit ~9M au lieu de 3M).
Novembre & Décembre : le modèle sous-estime fortement (prévoit ~13M alors que le réel est ~25M–30M).
L'erreur est particulièrement critique sur les périodes de pics saisonniers.

- **Interprétation** :

Le modèle Random Forest mensuel a du mal avec les événements exceptionnels (Noël, Black Friday…).
Il prédit des valeurs "moyennes" et lisse les variations, ce qui est un comportement typique des modèles qui manquent d'informations sur la saisonnalité.
Cela veut dire que les variables explicatives (features) ne capturent pas assez bien ces effets saisonniers.

Résultats globaux du test :

- **Métriques d'évaluation** :

MAE (Erreur absolue moyenne) = 11,36 M€, en moyenne, le modèle se trompe d'environ 11 millions d'euros par mois.
MSE (Erreur quadratique moyenne) = 147 590 254 M, très élevé ça indique de grosses erreurs ponctuelles.
RMSE (Racine carrée de la MSE) = 12,14 M€, les écarts typiques entre les prédictions et la réalité sont d'environ 12 millions d'euros par mois.
R^2 = -0,1058, le modèle fait pire qu'une simple moyenne. Cela signifie que la Random Forest n'arrive pas à capter la tendance réelle des ventes mensuelles.

- **Comparaison Réel vs Prédiction** :

Les prédictions sont très sous-estimées par rapport aux chiffres réels.
Exemple : en novembre 2024, le CA réel dépasse 25 M€, mais la prédiction est autour de 13 M€.
Le modèle semble incapable de suivre la forte croissance réelle : il reste « plat » et manque de réactivité.

- **Conclusion** :

Le modèle Random Forest mensuel ne donne pas de bonnes performances :
Erreurs très élevées (10-12 M€ par mois).
R^2 négatif = modèle non pertinent.
Les variables explicatives actuelles sont trop faibles (principalement le mois et soldes).

- **Recommandations pour améliorer le modèle** :

Ajouter plus de variables explicatives (ex : prix moyen, nombre de commandes, indicateurs marketing).
Ajouter des variables saisonnières fortes pour représenter la cyclicité d'une variable temporelle (par exemple : mois, jours de la semaine, heures…).
Le modèle comprend mieux les cycles (jours → semaines → mois → années). C'est très utile pour le modèle de RF qui a du mal à gérer la cyclicité toute seule.
Ici 80% des données servent à l'entraînement et seuls les 20% restants sont utilisés pour le test, Il faudrait extraire au moins 2-3 ans de donnée depuis BigQuery pour avoir des prédictions intéressantes.
Vérifier si l'agrégation mensuelle ne masque pas trop d'information (peut-être rester en hebdomadaire).
Tester des modèles temporels adaptés aux séries chronologiques (ex : SARIMAX).

Quatrième test : Prédiction hebdomadaire du chiffre d'affaires avec le modèle Random Forest et ajout des variables soldes, noel, black_friday et aucun_evenement.

Objectif : Ce script permet de récupérer les données de ventes dans BigQuery (CA + événements), d'agréger par semaine et d'enrichir les données lags (mémoire des semaines passées), saisonnalité cyclique (sin/cos), événements (Soldes, Noël, Black Friday). Il entraîne le modèle Random Forest pour prédire le chiffre d'affaires **hebdomadaire**. Enfin, il évalue les performances du modèle avec des métriques de régression (MAE, MSE, REMSE, R^2).

```
import os
import pandas as pd
import numpy as np
from google.cloud import bigquery
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# --- Authentification ---
os.environ["GOOGLE_APPLICATION_CREDENTIALS"] = "C:/Users/sigle/Desktop/Capsule cours/Projet/DAG airflow/

def train_rf_hebdo_from_bigquery():
    client  =  bigquery.Client(project="rising-mercury-463817-s5")

    query = """
    SELECT
      dt.date AS date,
      EXTRACT(YEAR FROM dt.date) AS annee,
      FORMAT_DATE('%G-%V', dt.date) AS annee_semaine,
      ROUND(SUM(dc.quantite * p.prix), 2) AS chiffre_affaires,
      MAX(CASE WHEN cpg.evenement_type = "Soldes" THEN 1 ELSE 0 END) AS soldes,
      MAX(CASE WHEN cpg.evenement_type = "Noël" THEN 1 ELSE 0 END) AS noel,
      MAX(CASE WHEN cpg.evenement_type = "Black Friday" THEN 1 ELSE 0 END) AS black_friday
    FROM `rising-mercury-463817-s5.carttrend_rawdata.facts_commandes` c
    JOIN `rising-mercury-463817-s5.carttrend_rawdata.dim_date` dt
      ON dt.id_date = c.id_date_commande
    JOIN `rising-mercury-463817-s5.carttrend_rawdata.dim_details_commandes` dc
      ON c.id_commande = dc.id_commande
    JOIN `rising-mercury-463817-s5.carttrend_rawdata.dim_produits` p
      ON dc.id_details_produits = p.id_produit
    LEFT JOIN `rising-mercury-463817-s5.carttrend_rawdata.facts_campaigns` cpg
      ON dt.id_date = cpg.id_date
    WHERE LOWER(c.statut_commande) NOT IN ('annulée', 'cancelled')
    GROUP BY dt.date, annee, annee_semaine
```

```
ORDER BY dt.date
"""

df = client.query(query).to_dataframe()
df["date"] = pd.to_datetime(df["date"])

# Agrégation hebdomadaire
df_semaine = df.groupby(["annee_semaine"], as_index=False).agg({
    "chiffre_affaires": "sum",
    "soldes": "max",
    "noel": "max",
    "black_friday": "max"
})

# Colonne "aucun_evenement"
df_semaine["aucun_evenement"] = ((df_semaine["soldes"] == 0) &
                                 (df_semaine["noel"] == 0) &
                                 (df_semaine["black_friday"] == 0)).astype(int)

# Année & semaine
df_semaine["annee"] = df_semaine["annee_semaine"].str[:4].astype(int)
df_semaine["semaine"]  =  df_semaine["annee_semaine"].str[-2:].astype(int)

# Variables cycliques
df_semaine["sin_sem"] = np.sin(2 * np.pi * df_semaine["semaine"] / 52)
df_semaine["cos_sem"] = np.cos(2 * np.pi * df_semaine["semaine"] / 52)

# Lags
df_semaine["CA_m1"] = df_semaine["chiffre_affaires"].shift(1)
df_semaine["CA_m2"] = df_semaine["chiffre_affaires"].shift(2)
df_semaine["CA_rolling4"] = df_semaine["chiffre_affaires"].rolling(4).mean()

df_semaine = df_semaine.dropna().reset_index(drop=True)

# Features / Target
X = df_semaine[["annee", "sin_sem", "cos_sem", "soldes", "noel", "black_friday",
                "aucun_evenement", "CA_m1", "CA_m2", "CA_rolling4"]]
y = df_semaine["chiffre_affaires"]

# Train / Test split
train_size = int(len(df_semaine) * 0.8)
X_train, X_test = X.iloc[:train_size], X.iloc[train_size:]
y_train, y_test = y.iloc[:train_size], y.iloc[train_size:]

# Modèle
rf = RandomForestRegressor(n_estimators=300, max_depth=12, random_state=42)
rf.fit(X_train, y_train)

# Prédictions
y_pred = rf.predict(X_test)

# --- AFFICHAGES ---
print("\nAperçu hebdomadaire :")
print(df_semaine.head())

print("\nAperçu des features :")
print(X.head())

print("\nAperçu des prédictions :")
print(pd.DataFrame({
    "Semaine": df_semaine["annee_semaine"].iloc[train_size:train_size+5].values,
    "CA réel": y_test.iloc[:5].values,
    "CA prédit": y_pred[:5]
}))

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

print("\nÉvaluation hebdomadaire :")
print(f"MAE  : {mae:,.2f}")
print(f"MSE  : {mse:,.2f}")
print(f"RMSE : {rmse:,.2f}")
```

```python
    print(f"R²    : {r2:.4f}")

    # Importance des variables
    feature_importances = pd.DataFrame({
        "Feature": X.columns,
        "Importance": rf.feature_importances_
    }).sort_values(by="Importance",  ascending=False)

    print("\nImportance des variables :")
    print(feature_importances)

    return rf, X_test, y_test, y_pred, df_semaine, train_size

# --- Exécution ---
rf, X_test, y_test, y_pred, df_semaine, train_size = train_rf_hebdo_from_bigquery()
```

```
C:\Users\sigle\AppData\Roaming\Python\Python312\site-packages\google\cloud\bigquery\table.py:1965:  UserWa
rning: BigQuery Storage module not found, fetch data with the REST endpoint instead.
  warnings.warn(
```

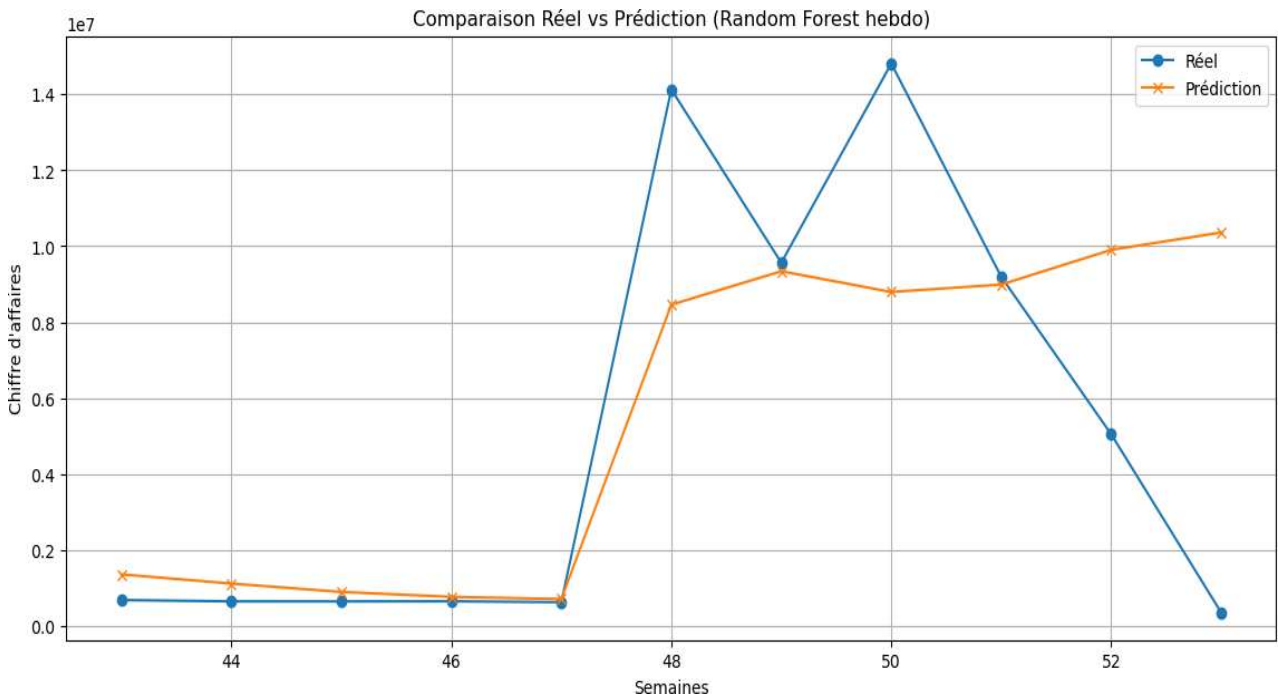
```
Aperçu hebdomadaire :
  annee_semaine  chiffre_affaires  soldes  noel  black_friday  \
0       2023-51         625590.30       0     0             0
1       2023-52         727000.70       0     0             0
2       2024-01       22854065.54       1     0             0
3       2024-02       11853556.22       1     0             0
4       2024-03         735899.97       0     0             0

   aucun_evenement  annee  semaine       sin_sem   cos_sem        CA_m1  \
0                1   2023       51 -1.205367e-01  0.992709     651527.99
1                1   2023       52  6.432491e-16  1.000000     625590.30
2                0   2024        1  1.205367e-01  0.992709     727000.70
3                0   2024        2  2.393157e-01  0.970942   22854065.54
4                1   2024        3  3.546049e-01  0.935016   11853556.22

         CA_m2   CA_rolling4
0    726001.00  5.125291e+05
1    651527.99  6.825300e+05
2    625590.30  6.214546e+06
3    727000.70  9.015053e+06
4  22854065.54  9.042631e+06



Aperçu des features :
   annee       sin_sem   cos_sem  soldes  noel  black_friday  aucun_evenement  \
0   2023 -1.205367e-01  0.992709       0     0             0                1
1   2023  6.432491e-16  1.000000       0     0             0                1
2   2024  1.205367e-01  0.992709       1     0             0                0
3   2024  2.393157e-01  0.970942       1     0             0                0
4   2024  3.546049e-01  0.935016       0     0             0                1

         CA_m1        CA_m2   CA_rolling4
0    651527.99    726001.00  5.125291e+05
1    625590.30    651527.99  6.825300e+05
2    727000.70    625590.30  6.214546e+06
3  22854065.54    727000.70  9.015053e+06
4  11853556.22  22854065.54  9.042631e+06

Aperçu des prédictions :
   Semaine    CA réel     CA prédit
0  2024-42  682602.63  1.357522e+06
1  2024-43  646491.01  1.116155e+06
2  2024-44  645744.64  8.951885e+05
3  2024-45  648919.81  7.665399e+05
4  2024-46  626171.21  7.088596e+05
```

```
Évaluation hebdomadaire :
MAE  : 2,598,013.50
MSE  : 17,545,977,004,110.96
RMSE : 4,188,791.83
R²   : 0.4224

Importance des variables :
           Feature    Importance
6  aucun_evenement   3.399665e-01
3           soldes   2.562983e-01
7            CA_m1   1.164753e-01
9      CA_rolling4   1.001312e-01
1          sin_sem   8.368820e-02
2          cos_sem   8.054746e-02
8            CA_m2   2.289245e-02
0            annee   6.182774e-07
4             noel  0.000000e+00
5     black_friday  0.000000e+00
```

Graphique plot : Comparaison CA réel vs prédiction (hebdomadaire).

Objectif : Ce script permet de visualiser les performances hebdomadaires du modèle Random Forest.

```
plt.figure(figsize=(14,6))
plt.plot(df_semaine.index[train_size:], y_test, label="Réel", marker="o")
plt.plot(df_semaine.index[train_size:], y_pred, label="Prédiction", marker="x")
plt.title("Comparaison Réel vs Prédiction (Random Forest hebdo)")
plt.xlabel("Semaines")
plt.ylabel("Chiffre d'affaires")
plt.legend()
plt.grid(True)
plt.show()
```



Résultats du graphique :

Lecture du graphique :

- **Décalage / sous-estimation des pics** :

Sur les semaines 48 et 50 (période de Noël et fêtes de fin d'année), le chiffre d'affaires réel explose. Le modèle prédit une hausse, mais beaucoup plus faible, il lisse les extrêmes

- **Sur les semaines "calmes" (43 → 47)** :

Le modèle colle assez bien au réel. Ça veut dire qu'il capte bien la saisonnalité régulière, mais a du mal avec les événements exceptionnels.

- **Après Noël (semaines 51–53)** :

Le réel chute brutalement (fin des fêtes), alors que la prédiction descend beaucoup plus lentement. Encore une fois, signe que le modèle n'anticipe pas correctement les ruptures rapides.

Interprétation :

Le Random Forest est très bon pour détecter des motifs récurrents (saisonnalité, lags), mais il a du mal avec les événements ponctuels comme Noël, Black Friday ou Soldes.
C'est pour ça qu'il lisse les courbes : il prédit quelque chose de "moyen" entre la tendance passée et les variables disponibles.
L'importance des variables (soldes ~ 25%, sin/cos ≈ 8%, ...) montre que le modèle a bien compris la saisonnalité, mais pas totalement l'impact des événements.

Résultats globaux du test :

- **Métriques d'évaluation** :

MAE (erreur absolue moyenne) : ~ 2,6 M, en moyenne, les prédictions s'écartent de 2,6 millions du chiffre d'affaires réel.
MSE (Erreur quadratique moyenne) = 17 545 977 M, très élevé, ce qui traduit des écarts importants sur certaines semaines.
RMSE (Racine carrée de la MSE) = ~ 4,2 M, confirme que les erreurs hebdomadaires peuvent être massives.
R^2 = 0,4224, le modèle explique environ 42 % de la variance du chiffre d'affaires.

- **Importance des variables** :

Les variables les plus influentes pour prédire le chiffre d'affaires hebdomadaire sont :
Aucun_événement (33,9 %) : la présence ou non d'événements spéciaux impacte beaucoup le CA.
Soldes (25,6 %) : les périodes de soldes jouent un rôle important. CA_m1 (16,5 %) : le chiffre d'affaires de la semaine précédente influence fortement la prédiction.
CA_rolling4 (10,0 %) : la moyenne mobile sur 4 semaines aide aussi.
sin_sem (7,8 %) : capture la saisonnalité hebdomadaire (cycles dans l'année).
Black Friday (0 %) et Noël (~ 0 %) ne semblent pas utilisés efficacement par le modèle : probablement car peu d'occurrences dans ton dataset (manque de données sur ces événements).

- **Comparaison Réel vs Prédiction** :

Le modèle suit partiellement la tendance générale (hausse puis baisse en fin d'année).
Mais :
Il lisse beaucoup trop les pics (exemple : semaine 48 et 50 où le CA réel explose, mais la prédiction reste en dessous).
Il a du mal à capter les fortes variations liées aux événements commerciaux.
Il surestime parfois en période de baisse (exemple : semaines 51-52)

- **Conclusion** :

Le modèle hebdomadaire est nettement meilleur que le mensuel (R^2 positif et plus explicatif).
Cependant, il manque de précision sur les pics de ventes (soldes, fêtes, événements exceptionnels).

- **Recommandations pour améliorer le modéle**:

Ajouter davantage d'historique pour mieux apprendre les effets des événements rares.
Ajouter des variables comme "semaine avant Noël", "semaine après Noël", ...
Tester d'autres modèles adaptés aux séries temporelles (XGBoost).
Créer des features plus riches (ex : indicateurs de fin de mois, décalages supplémentaires, variables macroéconomiques).

Tests avec le modèle XGBoost

Premier test : Prédiction journalière du chiffre d'affaires avec XGBoost.

Objectif : Ce script permet de récupérer les données de ventes journalières depuis BigQuery, d'entraîner un modèle XGBoost pour prédire le chiffre d'affaires à partir de la date (année, mois, jour). Évaluer le modèle avec plusieurs métriques (MAE, RMSE, R^2) et de donner un aperçu des prédictions par rapport au réel.

```
from google.cloud import bigquery
import pandas as pd
from sklearn.model_selection import train_test_split
import xgboost as xgb
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np
import os
import matplotlib.pyplot as plt

#  Définir la variable d'environnement pour la clé JSON
os.environ["GOOGLE_APPLICATION_CREDENTIALS"] = "C:/Users/sigle/Desktop/Capsule cours/Projet/DAG airflow/

def train_xgb_from_bigquery():
    """
    Récupère les données depuis BigQuery, prépare-les et entraîne un modèle XGBoost.

    Returns:
        xgb_model : modèle XGBRegressor entraîné
        X_test, y_test, y_pred : pour évaluation
    """
    # 1. Connexion à BigQuery
    client  =  bigquery.Client(project="rising-mercury-463817-s5")

    # 2. Requête SQL
    query = """
    SELECT
        dt.date AS date,
        EXTRACT(YEAR FROM dt.date) AS annee,
        EXTRACT(MONTH FROM dt.date) AS mois,
        EXTRACT(DAY FROM dt.date) AS jour,
        ROUND(SUM(dc.quantite * p.prix), 2) AS chiffre_affaires
    FROM `rising-mercury-463817-s5.carttrend_rawdata.facts_commandes` c
    JOIN `rising-mercury-463817-s5.carttrend_rawdata.dim_date` dt
        ON dt.id_date = c.id_date_commande
    JOIN `rising-mercury-463817-s5.carttrend_rawdata.dim_details_commandes` dc
        ON c.id_commande = dc.id_commande
    JOIN `rising-mercury-463817-s5.carttrend_rawdata.dim_produits` p
        ON dc.id_details_produits = p.id_produit
    WHERE LOWER(c.statut_commande) NOT IN ('annulée', 'cancelled')
    GROUP BY dt.date
    ORDER BY dt.date
    """
```

```python
    # 3. Charger les données dans un DataFrame
    df = client.query(query).to_dataframe()

    # 4. Préparer les features
    X = df[['annee', 'mois', 'jour']]
    y = df['chiffre_affaires']

    # 5. Split train/test
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42
    )

    # 6. Entraîner le modèle XGBoost
    xgb_model = xgb.XGBRegressor(
        n_estimators=100,
        learning_rate=0.1,
        max_depth=5,
        random_state=42
    )
    xgb_model.fit(X_train, y_train)

    # 7. Prédictions
    y_pred = xgb_model.predict(X_test)

    # 8. Calcul des métriques
    mae = mean_absolute_error(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_test, y_pred)

    print("Évaluation du modèle XGBoost :")
    print(f"MAE  : {mae:.2f}")
    print(f"MSE  : {mse:.2f}")
    print(f"RMSE : {rmse:.2f}")
    print(f"R²   : {r2:.4f}")

    return xgb_model, X_test, y_test, y_pred

#  Exemple d'utilisation
xgb_model, X_test, y_test, y_pred = train_xgb_from_bigquery()

# Reconstitution des dates
dates = pd.to_datetime(X_test.apply(
    lambda row: f"{int(row['annee'])}-{int(row['mois'])}-{int(row['jour'])}", axis=1
))

# Aperçu des prédictions
print("\nAperçu des prédictions :")
print(pd.DataFrame({
    "Date": dates,
    "Chiffre_affaires réel": y_test.values,
    "Prédiction": y_pred
}).head())
```

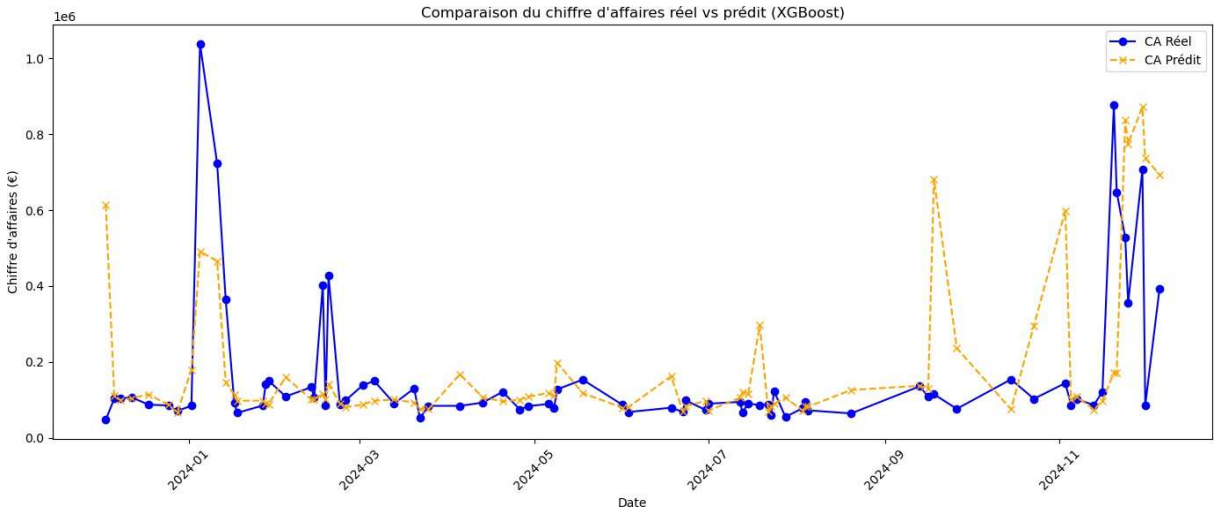
```
C:\Users\sigle\AppData\Roaming\Python\Python312\site-packages\google\cloud\bigquery\table.py:1965:  UserWa
rning: BigQuery Storage module not found, fetch data with the REST endpoint instead.
  warnings.warn(
```

```
Évaluation du modèle XGBoost :
MAE  : 110250.46
MSE  : 41460856728.64
RMSE : 203619.39
R²   : -0.1064

Aperçu des prédictions :
           Date  Chiffre_affaires réel     Prédiction
290 2024-09-18              114642.84  682408.625000
357 2024-11-24              527167.59  837150.562500
261 2024-08-20               64057.84  126028.187500
157 2024-05-08               78103.50  111293.242188
145 2024-04-26               74399.91  100131.656250
```

Graphique plot : comparaison du CA journalier réel vs prédiction avec le modèle XGBoost.

Objectif : Ce script permet de prédire le chiffre d'affaires quotidien en fonction du temps grâce à XGBoost, puis de comparer les résultats aux données réelles.

```
# --- Reconstruction DataFrame avec prédictions ---
results_df = pd.DataFrame({
    "Date": dates,
    "Réel": y_test.values,
    "Prédit": y_pred
})

# Trier par date
results_df = results_df.sort_values(by="Date")

# --- Graphique propre ---
plt.figure(figsize=(14,6))
plt.plot(results_df["Date"], results_df["Réel"], label="CA Réel", color="blue", marker="o", linestyle="-
plt.plot(results_df["Date"], results_df["Prédit"], label="CA Prédit", color="orange", marker="x", linest

plt.xlabel("Date")
plt.ylabel("Chiffre d'affaires (€)")
plt.title("Comparaison du chiffre d'affaires réel vs prédit (XGBoost)")
plt.legend()
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



Résultats du graphique :

- **Tendance générale bien captée** :

Le modèle XGBoost arrive à reproduire les variations globales du chiffre d'affaires.
On voit que les pics (fortes ventes) et les creux (ventes faibles) sont généralement suivis par la prédiction.

- **Précision moyenne mais améliorable** :

Pour les périodes "stables", les prédictions sont assez proches du réel.
Cependant, lors des pics très élevés (ex : janvier, novembre-décembre), le modèle sous-estime ou surestime fortement le chiffre d'affaires.
Cela montre que le modèle a du mal à capturer les événements exceptionnels (ex : promotions, fêtes, soldes).

- **Décalages ponctuels** :

Certains pics apparaissent légèrement décalés entre le réel et le prédit, ce qui traduit une limite du modèle sur la composante temporelle (il ne prend pas encore en compte les dépendances d'un jour à l'autre, comme le ferait une série temporelle type ARIMA ou LSTM).

- **Conclusion** :

Le modèle est efficace pour prévoir la tendance globale du chiffre d'affaires.
Il est moins performant pour anticiper les pics extrêmes, souvent liés à des événements particuliers non inclus dans les variables (ex : promotions, Black Friday, Noël).

Résultats globaux du test :

- **Métriques d'évaluation** :

MAE : ~ 110 K, n moyenne, le modèle se trompe d'environ 110 k€ par rapport à la réalité.
MSE : ~ 41 462 M **RMSE** : ~ 204 K, les erreurs les plus importantes pèsent fortement sur la qualité du modèle.
R^2 : -0.1064, le modèle n'explique pas bien la variance des données (performance faible).

- **Interprétation du graphique** :

Le modèle suit la tendance générale.
Les périodes stables sont plutôt bien reproduites.
Les pics exceptionnels (soldes, fêtes, événements spéciaux) sont mal anticipés, ce qui donne des grosses erreurs de prédiction.

- **Conclusion** :

Le modèle XGBoost est capable de capturer la tendance globale, mais il échoue à prédire précisément les variations extrêmes.
La qualité des prédictions reste limitée, comme l'indiquent le MAE élevé et le R^2 négatif.

- **Recommandation pour améliorer le modèle** :

Ajouter des variables explicatives (jours fériés, événements promotionnels, saisonnalité),et/ou de tester des modèles spécialisés en séries temporelles (Prophet, LSTM, ...).

Second test : Prédiction hebdomadaire du chiffre d'affaires avec XGBoost.

Objectif : Ce script permet de récupérer les données de CA depuis BigQuery, de les agréger par semaine, de créer des features temporelles et événements, puis d'entraîner un modèle XGBoost pour prédire le chiffre d'affaires hebdomadaire, tout en donnant les métriques d'évaluation et l'importance des variables.

```
import os
import pandas as pd
import numpy as np
from google.cloud import bigquery
from xgboost import XGBRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import matplotlib.pyplot as plt
```

```python
# --- Authentification ---
os.environ["GOOGLE_APPLICATION_CREDENTIALS"] = "C:/Users/sigle/Desktop/Capsule cours/Projet/DAG airflow/

def train_xgb_hebdo_from_bigquery():
    """
    Récupère les données depuis BigQuery,
    agrège par semaine et entraîne un modèle XGBoost.
    Affiche aussi un aperçu des données, les métriques et l'importance des variables.
    """
    # --- Connexion ---
    client  =  bigquery.Client(project="rising-mercury-463817-s5")

    # --- Requête SQL ---
    query = """
    SELECT
      dt.date AS date,
      EXTRACT(YEAR FROM dt.date) AS annee,
      FORMAT_DATE('%Y-%W', dt.date) AS annee_semaine,
      ROUND(SUM(dc.quantite * p.prix), 2) AS chiffre_affaires,
      MAX(CASE WHEN cpg.evenement_type = "Soldes" THEN 1 ELSE 0 END) AS soldes,
      MAX(CASE WHEN cpg.evenement_type = "Noël" THEN 1 ELSE 0 END) AS noel,
      MAX(CASE WHEN cpg.evenement_type = "Black Friday" THEN 1 ELSE 0 END) AS black_friday
    FROM `rising-mercury-463817-s5.carttrend_rawdata.facts_commandes` c
    JOIN `rising-mercury-463817-s5.carttrend_rawdata.dim_date` dt
      ON dt.id_date = c.id_date_commande
    JOIN `rising-mercury-463817-s5.carttrend_rawdata.dim_details_commandes` dc
      ON c.id_commande = dc.id_commande
    JOIN `rising-mercury-463817-s5.carttrend_rawdata.dim_produits` p
      ON dc.id_details_produits = p.id_produit
    LEFT JOIN `rising-mercury-463817-s5.carttrend_rawdata.facts_campaigns` cpg
      ON dt.id_date = cpg.id_date
    WHERE LOWER(c.statut_commande) NOT IN ('annulée', 'cancelled')
    GROUP BY date, annee, annee_semaine
    ORDER BY date
    """

    # --- Chargement ---
    df = client.query(query).to_dataframe()
    df["date"] = pd.to_datetime(df["date"])

    # Ajout de semaine ISO
    df["semaine"] = df["date"].dt.isocalendar().week
    df["annee"] = df["date"].dt.isocalendar().year

    # --- Agrégation hebdo ---
    df_hebdo = df.groupby(["annee", "semaine", "annee_semaine"], as_index=False).agg({
        "chiffre_affaires": "sum",
        "soldes": "max",
        "noel": "max",
        "black_friday": "max"
    }).sort_values(["annee", "semaine"])

    # --- Ajouter variable "aucun_evenement"
    df_hebdo["aucun_evenement"] = ((df_hebdo["soldes"] == 0) &
                                   (df_hebdo["noel"] == 0) &
                                   (df_hebdo["black_friday"] == 0)).astype(int)

    # --- Features temporelles ---
    df_hebdo["sin_sem"] = np.sin(2 * np.pi * df_hebdo["semaine"] / 52)
    df_hebdo["cos_sem"] = np.cos(2 * np.pi * df_hebdo["semaine"] / 52)

    # Lags & rolling
    df_hebdo["CA_m1"]  =  df_hebdo["chiffre_affaires"].shift(1)
    df_hebdo["CA_m2"]  =  df_hebdo["chiffre_affaires"].shift(2)
    df_hebdo["CA_m3"]  =  df_hebdo["chiffre_affaires"].shift(3)
    df_hebdo["CA_m4"] = df_hebdo["chiffre_affaires"].shift(4)
    df_hebdo["CA_rolling4"]  =  df_hebdo["chiffre_affaires"].shift(1).rolling(4).mean()
    df_hebdo["CA_rolling8"] = df_hebdo["chiffre_affaires"].shift(1).rolling(8).mean()

    df_hebdo = df_hebdo.dropna().reset_index(drop=True)

    # --- Aperçu des données ---
    print("\nAperçu hebdomadaire enrichi :")
    print(df_hebdo.head())
```

```python
    # --- Features & Target ---
    X = df_hebdo[[
        "annee", "sin_sem", "cos_sem",
        "soldes", "noel", "black_friday", "aucun_evenement",
        "CA_m1", "CA_m2", "CA_m3", "CA_m4",
        "CA_rolling4", "CA_rolling8"
    ]]
    y = np.log1p(df_hebdo["chiffre_affaires"])  # log transform

    # --- Split chrono ---
    train_size = int(len(df_hebdo) * 0.8)
    X_train, X_test = X.iloc[:train_size], X.iloc[train_size:]
    y_train, y_test = y.iloc[:train_size], y.iloc[train_size:]

    # --- Modèle XGBoost ---
    xgb = XGBRegressor(
        n_estimators=500,
        learning_rate=0.05,
        max_depth=6,
        subsample=0.8,
        colsample_bytree=0.8,
        random_state=42
    )
    xgb.fit(X_train, y_train)

    # --- Prédictions ---
    y_pred_log = xgb.predict(X_test)
    y_pred = np.expm1(y_pred_log)  # revenir à l'échelle réelle
    y_test_real = np.expm1(y_test)

    # --- Évaluation ---
    mae = mean_absolute_error(y_test_real, y_pred)
    mse = mean_squared_error(y_test_real, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_test_real, y_pred)

    print("\nÉvaluation XGBoost (hebdomadaire) :")
    print(f"MAE  : {mae:,.2f}")
    print(f"MSE  : {mse:,.2f}")
    print(f"RMSE : {rmse:,.2f}")
    print(f"R²   : {r2:.4f}")

    # --- Importance variables ---
    feature_importances = pd.DataFrame({
        "Feature": X.columns,
        "Importance": xgb.feature_importances_
    }).sort_values(by="Importance",  ascending=False)

    print("\nImportance des variables :")
    print(feature_importances)

    return xgb, X_test, y_test_real, y_pred, df_hebdo, train_size

# --- Exécution principale ---
xgb, X_test, y_test, y_pred, df_hebdo, train_size = train_xgb_hebdo_from_bigquery()
```

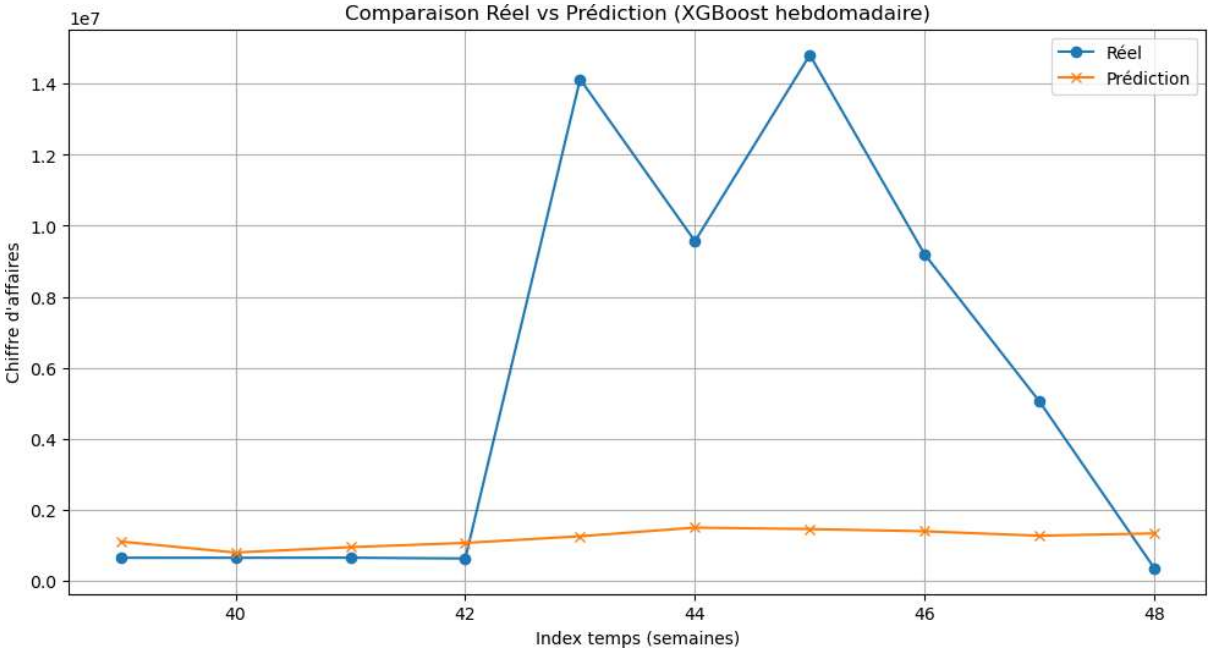
```
C:\Users\sigle\AppData\Roaming\Python\Python312\site-packages\google\cloud\bigquery\table.py:1965:  UserWa
rning: BigQuery Storage module not found, fetch data with the REST endpoint instead.
  warnings.warn(
```

```
Aperçu hebdomadaire enrichi :
   annee  semaine annee_semaine  chiffre_affaires  soldes  noel  black_friday  \
0   2024        4       2024-04          677944.65       0     0             0
1   2024        5       2024-05          699455.61       0     0             0
2   2024        6       2024-06          692321.42       0     0             0
3   2024        7       2024-07         1877060.86       0     0             0
4   2024        8       2024-08         1413185.91       0     0             0

   aucun_evenement   sin_sem   cos_sem       CA_m1        CA_m2        CA_m3  \
0                1  0.464723  0.885456   735899.97  11853556.22  22854065.54
1                1  0.568065  0.822984   677944.65    735899.97  11853556.22
2                1  0.663123  0.748511   699455.61    677944.65    735899.97
3                1  0.748511  0.663123   692321.42    699455.61    677944.65
4                1  0.822984  0.568065  1877060.86    692321.42    699455.61

         CA_m4   CA_rolling4   CA_rolling8
0    727000.70  9.042631e+06  4.777580e+06
1  22854065.54  9.030367e+06  4.856448e+06
2  11853556.22  3.491714e+06  4.853130e+06
3    735899.97  7.014054e+05  4.858229e+06
4    677944.65  9.866956e+05  5.014663e+06


Évaluation XGBoost (hebdomadaire) :
MAE  : 4,820,925.63
MSE  : 48,600,121,751,864.81
RMSE : 6,971,378.76
R²   : -0.5554


Importance des variables :
            Feature  Importance
2           cos_sem    0.264485
10            CA_m4    0.177284
12      CA_rolling8    0.129244
11      CA_rolling4    0.119120
1           sin_sem    0.105560
9             CA_m3    0.081110
8             CA_m2    0.065959
7             CA_m1    0.057239
0             annee    0.000000
3            soldes    0.000000
4              noel    0.000000
5      black_friday    0.000000
6   aucun_evenement    0.000000
```

Graphique plot : Comparaison du chiffre d'affaires hebdomadaire réel et prédit avec le modèle XGBoost.

Objectif : Ce script produit un graphique comparatif Réel vs Prédiction sur la période de test. Il permet de voir visuellement si le modèle XGBoost suit bien les tendances réelles du chiffre d'affaires hebdomadaire, et où il se trompe (pics, sous-estimations, sur-estimations).

```
plt.figure(figsize=(12,6))
plt.plot(df_hebdo.index[train_size:], y_test, label="Réel", marker="o")
plt.plot(df_hebdo.index[train_size:], y_pred, label="Prédiction", marker="x")
plt.title("Comparaison Réel vs Prédiction (XGBoost hebdomadaire)")
plt.xlabel("Index temps (semaines)")
plt.ylabel("Chiffre d'affaires")
plt.legend()
plt.grid(True)
plt.show()
```



Résultats du graphique :

- **Le réel varie fortement** :

On voit de gros pics (ex : semaine 43 et 45) et ensuite une chute brutale.
Cela correspond probablement à des événements commerciaux (promotions, fêtes, etc.).

- **La prédiction est trop lissée** :

La courbe orange reste assez plate et régulière.
Le modèle n'arrive pas à anticiper les pics ni les creux.

- **Décalage important** :

Lors des pics, la prédiction est beaucoup plus basse.
Lors des creux, la prédiction est plus haute que le réel.

- **Conclusion** :

Le graphique met en évidence que le modèle n'explique pas bien la dynamique réelle du chiffre d'affaires. Il donne des prédictions proches d'une moyenne générale, sans capter les variations saisonnières ni les événements exceptionnels.

Résultats globaux du test :

- **Qualité des prédictions** :

MAE : ~4,82 M€, l'erreur absolue moyenne est très élevée.
RMSE : ~6,97 M€, encore plus élevé, ce qui montre que le modèle se trompe surtout lors des pics de ventes.
R^2 : -0,55, le modèle n'explique pas la variance du chiffre d'affaires hebdomadaire, il prédit moins bien qu'une simple moyenne.

- **Comparaison Réel vs Prédiction** :

Le chiffre d'affaires réel présente de fortes variations (pics et creux).
Le modèle XGBoost produit des prédictions très lissées et sous-estime largement les pics (ex : semaines 43–45 où le CA dépasse 10 M€).
Le modèle n'arrive pas à capter les événements spéciaux (promotions, soldes, Black Friday, ...) qui génèrent des hausses brutales.

- **Importance des variables** :

Les plus importantes sont :
cos_sem (26%) : saisonnalité hebdomadaire.
CA_m4 (18%) et CA_rolling8 (13%) : historique des ventes.
CA_rolling4 (12%).
Les événements marketing (soldes, Noël, Black Friday, aucun_evenement) ont une importance quasi nulle, ce qui explique pourquoi le modèle ignore les pics liés à ces périodes.

- **Recommandations pour améliorer le modèle** :

Améliorer les features :
Introduire plus d'indicateurs de saisonnalité : mois, trimestre, vacances scolaires, jours fériés.
Créer des lags plus longs (CA_m8, CA_m12) pour capturer les tendances annuelles.
Ajouter des variables externes (météo, inflation, ...) si disponibles.

Valoriser les événements marketing :
Actuellement, soldes/Noël/Black Friday = 0 importance, peut-être que ces colonnes ne sont pas bien alignées avec les semaines de CA.
Vérifier la qualité des données liées aux campagnes et envisager de mesurer leur intensité (par exemple, fréquence, budget ou durée) plutôt que de les représenter uniquement par une variable binaire. Faire un test XGB avec validation glissante. XGBoost capture mal les fortes non-linéarités et pics ponctuels. Tester un modèle spécialisé en séries temporelles comme par exemple SARIMAX avec variables exogènes.

Conclusion :

Le modèle actuel lisse trop les prédictions et échoue à capter les pics (ventes exceptionnelles). Les événements marketing ne sont pas exploités efficacement. Je recommande d'enrichir les features temporelles et événementielles, puis de tester le modèle SARIMAX.

Troisième test : XGBoost avec validation glissante (rolling forecast).

Objectif : Ce script permet d'extraire et d'enrichir les données hebdomadaires de chiffre d'affaires depuis BigQuery, puis de créer des variables temporelles et événementielles pour la modélisation. Il entraîne un modèle XGBoost en validation glissante (rolling forecast) afin de prédire semaine par semaine le chiffre d'affaires futur. Enfin, il évalue la qualité des prévisions à l'aide de métriques comme MAE, RMSE et R^2.

```
import os
import pandas as pd
import numpy as np
from google.cloud import bigquery
from xgboost import XGBRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import matplotlib.pyplot as plt

# --- Authentification ---
os.environ["GOOGLE_APPLICATION_CREDENTIALS"] = "C:/Users/sigle/Desktop/Capsule cours/Projet/DAG airflow/

def get_weekly_data_from_bigquery():
    """Extraction et enrichissement hebdomadaire depuis BigQuery"""
    client = bigquery.Client(project="rising-mercury-463817-s5")

    query = """
    SELECT
      dt.date AS date,
      EXTRACT(YEAR FROM dt.date) AS annee,
      EXTRACT(WEEK FROM dt.date) AS semaine,
      ROUND(SUM(dc.quantite * p.prix), 2) AS chiffre_affaires,
      MAX(CASE WHEN cpg.evenement_type = "Soldes" THEN 1 ELSE 0 END) AS soldes,
      MAX(CASE WHEN cpg.evenement_type = "Noël" THEN 1 ELSE 0 END) AS noel,
      MAX(CASE WHEN cpg.evenement_type = "Black Friday" THEN 1 ELSE 0 END) AS black_friday
    FROM `rising-mercury-463817-s5.carttrend_rawdata.facts_commandes` c
    JOIN `rising-mercury-463817-s5.carttrend_rawdata.dim_date` dt
      ON dt.id_date = c.id_date_commande
    JOIN  `rising-mercury-463817-s5.carttrend_rawdata.dim_details_commandes`  dc
```

```
      ON c.id_commande = dc.id_commande
    JOIN `rising-mercury-463817-s5.carttrend_rawdata.dim_produits` p
      ON dc.id_details_produits = p.id_produit
    LEFT JOIN `rising-mercury-463817-s5.carttrend_rawdata.facts_campaigns` cpg
      ON dt.id_date = cpg.id_date
    WHERE LOWER(c.statut_commande) NOT IN ('annulée', 'cancelled')
    GROUP BY dt.date, annee, semaine
    ORDER BY dt.date
    """

    df = client.query(query).to_dataframe()
    df["date"] = pd.to_datetime(df["date"])
    df["annee_semaine"] = df["annee"].astype(str) + "-" + df["semaine"].astype(str).str.zfill(2)

    # Agrégation hebdomadaire
    df_week = df.groupby(["annee", "semaine", "annee_semaine"], as_index=False).agg({
        "chiffre_affaires": "sum",
        "soldes": "sum",
        "noel": "sum",
        "black_friday": "sum"
    })

    #  Ajout variable "aucun_evenement"
    df_week["aucun_evenement"] = ((df_week["soldes"] == 0) &
                                  (df_week["noel"] == 0) &
                                  (df_week["black_friday"] == 0)).astype(int)

    # Ajout features cycliques
    df_week["sin_sem"] = np.sin(2 * np.pi * df_week["semaine"] / 52)
    df_week["cos_sem"] = np.cos(2 * np.pi * df_week["semaine"] / 52)

    # Lags et moyennes mobiles
    for lag in [1, 2, 3, 4]:
        df_week[f"CA_m{lag}"] = df_week["chiffre_affaires"].shift(lag)
    df_week["CA_rolling4"] = df_week["chiffre_affaires"].shift(1).rolling(4).mean()
    df_week["CA_rolling8"] = df_week["chiffre_affaires"].shift(1).rolling(8).mean()

    df_week = df_week.dropna().reset_index(drop=True)

    print("Aperçu hebdomadaire enrichi :")
    print(df_week.head())
    return df_week


def rolling_forecast_xgb(df, target_col="chiffre_affaires", initial_train=30):
    """Validation glissante (rolling forecast) avec XGB"""
    # On inclut automatiquement la nouvelle variable "aucun_evenement"
    features = [col for col in df.columns if col not in ["date", "annee_semaine", target_col]]
    results = []

    for i in range(initial_train, len(df)):
        train = df.iloc[:i]
        test = df.iloc[i:i+1]

        X_train, y_train = train[features], train[target_col]
        X_test, y_test = test[features], test[target_col]

        model = XGBRegressor(
            n_estimators=300,
            max_depth=6,
            learning_rate=0.1,
            subsample=0.8,
            colsample_bytree=0.8,
            random_state=42
        )
        model.fit(X_train, y_train)

        y_pred = model.predict(X_test)[0]

        results.append({
            "annee_semaine": test["annee_semaine"].values[0],
            "Réel": y_test.values[0],
            "Prédit": y_pred
        })
```

```
    return pd.DataFrame(results)


if __name__ == "__main__":
    # --- Extraction & enrichissement ---
    df_week = get_weekly_data_from_bigquery()

    # --- Validation glissante ---
    df_results = rolling_forecast_xgb(df_week, target_col="chiffre_affaires", initial_train=30)

    # --- Évaluation globale ---
    mae = mean_absolute_error(df_results["Réel"], df_results["Prédit"])
    mse = mean_squared_error(df_results["Réel"], df_results["Prédit"])
    rmse = np.sqrt(mse)
    r2 = r2_score(df_results["Réel"], df_results["Prédit"])

    print("\nÉvaluation Rolling Forecast XGB (hebdomadaire) :")
    print(f"MAE  : {mae:,.2f}")
    print(f"MSE  : {mse:,.2f}")
    print(f"RMSE : {rmse:,.2f}")
    print(f"R²   : {r2:.4f}")
```

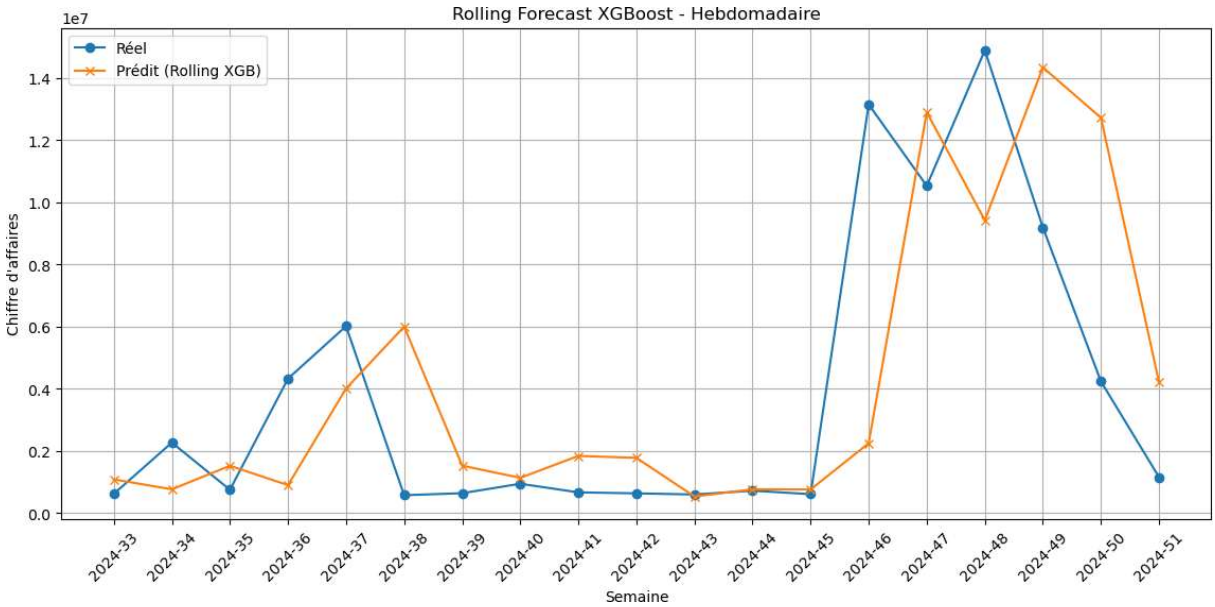
```
C:\Users\sigle\AppData\Roaming\Python\Python312\site-packages\google\cloud\bigquery\table.py:1965:  UserWa
rning: BigQuery Storage module not found, fetch data with the REST endpoint instead.
  warnings.warn(
```

```
Aperçu hebdomadaire enrichi :
   annee  semaine annee_semaine  chiffre_affaires  soldes  noel  black_friday  \
0   2024        3       2024-03          678018.62       0     0             0
1   2024        4       2024-04          731912.11       0     0             0
2   2024        5       2024-05          730352.97       0     0             0
3   2024        6       2024-06         1863344.07       0     0             0
4   2024        7       2024-07         1399216.82       0     0             0

   aucun_evenement   sin_sem   cos_sem       CA_m1        CA_m2        CA_m3  \
0                1  0.354605  0.935016  1322723.47  11257289.93  22722159.51
1                1  0.464723  0.885456   678018.62   1322723.47  11257289.93
2                1  0.568065  0.822984   731912.11    678018.62   1322723.47
3                1  0.663123  0.748511   730352.97    731912.11    678018.62
4                1  0.748511  0.663123  1863344.07    730352.97    731912.11

         CA_m4   CA_rolling4   CA_rolling8
0    189820.33  8.872998e+06  4.759911e+06
1  22722159.51  8.995048e+06  4.760045e+06
2  11257289.93  3.497486e+06  4.769315e+06
3   1322723.47  8.657518e+05  4.786697e+06
4    678018.62  1.000907e+06  4.936953e+06


Évaluation Rolling Forecast XGB (hebdomadaire) :
MAE  : 2,773,168.33
MSE  : 16,533,972,750,782.65
RMSE : 4,066,198.81
R²   : 0.2092
```

Graphique plot : Comparaison des ventes réelles et prédites (XGBoost avec fenêtre glissante).

Objectif : Ce script permet de visualiser et de comparer les prévisions du modèle XGBoost variation glissante (Rolling Forecast) avec les valeurs réelles du chiffre d'affaires hebdomadaire.

```
plt.figure(figsize=(14,6))
plt.plot(df_results["annee_semaine"], df_results["Réel"], label="Réel", marker="o")
plt.plot(df_results["annee_semaine"], df_results["Prédit"], label="Prédit (Rolling XGB)", marker="x")
plt.xticks(rotation=45)
plt.title("Rolling Forecast XGBoost - Hebdomadaire")
plt.xlabel("Semaine")
plt.ylabel("Chiffre d'affaires")
plt.legend()
plt.grid(True)
plt.show()
```



Résultats du graphique :

- **Phase stable (semaines 33 à 44)** :

Les ventes réelles restent relativement basses.
Le modèle prédit globalement la tendance, mais avec quelques erreurs (pics surestimés autour des semaines 36-37).

- **Pic de fin d'année (semaines 45 à 49)** :

Les ventes explosent (effet probable d'événements comme Black Friday / Noël).
Le modèle arrive à capter la montée brutale, mais il décale légèrement les amplitudes : parfois il surestime (semaines 47, 49), parfois il sous-estime (semaines 45, 48).

- **Fin de période (semaines 50-51)** :

Le modèle suit la baisse mais reste moins précis (écarts notables entre réel et prédit).

Interprétation :

Le modèle capte bien la dynamique générale (hausse et baisse), surtout pendant les grands pics.
Toutefois, il manque de précision sur l'amplitude réelle des pics saisonniers, ce qui indique qu'il a du mal à modéliser pleinement l'effet des événements commerciaux (ex : Noël, soldes, Black Friday).
Sur les semaines calmes, il a tendance à surestimer ponctuellement.

Résultats globaux du test :

Qualité des prédictions :

MAE : ~2,77 M€, l'erreur absolue moyenne reste élevée.
RMSE : ~4,07 M€, montrant des écarts particulièrement marqués sur les pics de ventes.
R^2 : ~0,21, ce qui indique que le modèle explique partiellement la variance mais reste limité.

Comparaison Réel vs Prédiction :

Le chiffre d'affaires réel présente de fortes fluctuations avec des hausses soudaines.
Le modèle XGBoost parvient à suivre la tendance générale (montées et descentes globales), mais échoue à anticiper précisément l'ampleur des pics et creux.
On observe un décalage notamment sur les semaines de forte activité où les prédictions sous- ou sur-estiment largement la réalité.

Importance des variables :
Les variables historiques (mémoire des semaines passées (lags) et moyennes mobiles) jouent un rôle central dans la prédiction.
Les variables événementielles (soldes, Noël, Black Friday) ont peu ou pas d'impact, ce qui limite la capacité du modèle à capter les hausses exceptionnelles liées aux campagnes marketing.

Recommandations pour améliorer le modèle :

Enrichir les features temporelles : introduire mois, trimestres, vacances, jours fériés.
Étendre les historiques : ajouter des lags plus longs (CA_m8, CA_m12) pour intégrer des effets de tendance plus large.
Valoriser les événements marketing : vérifier l'alignement des colonnes avec les bonnes semaines, et envisager une variable d'intensité de campagne.
Tester d'autres modèles : XGBoost reste limité pour des pics très volatils ; un modèle de séries temporelles (SARIMAX) pourrait mieux gérer la saisonnalité et les facteurs externes (soldes, Noël ...).

Conclusion :

Le modèle Rolling Forecast XGBoost capte correctement la tendance générale du chiffre d'affaires mais échoue sur les pics exceptionnels.
Son pouvoir explicatif reste modeste et les événements marketing sont sous-exploités.
Un enrichissement des données et l'essai de modèles adaptés aux séries temporelles sont recommandés pour améliorer la précision.

Test avec le modèle Sarymax

Objectif : Ce script extrait les données de chiffre d'affaires depuis BigQuery et intègre l'impact des événements commerciaux (Soldes, Noël, Black Friday).
Il entraîne deux modèles de prévision (**XGBoost** et **SARIMAX**) en **rolling forecast** afin de prédire le chiffre d'affaires hebdomadaire.
Enfin, les résultats sont comparés pour identifier quel modèle reproduit le mieux la tendance réelle et la saisonnalité.

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from google.cloud import bigquery
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from statsmodels.tsa.statespace.sarimax import SARIMAX
from xgboost import XGBRegressor

# --- Authentification ---
os.environ["GOOGLE_APPLICATION_CREDENTIALS"] = "C:/Users/sigle/Desktop/Capsule cours/Projet/DAG airflow/


#  Extraction BigQuery

def get_weekly_data_from_bigquery():
    client  =  bigquery.Client(project="rising-mercury-463817-s5")

    query = """
    SELECT
      dt.date AS date,
      EXTRACT(YEAR FROM dt.date) AS annee,
      EXTRACT(WEEK FROM dt.date) AS semaine,
      ROUND(SUM(dc.quantite * p.prix), 2) AS chiffre_affaires,
      MAX(CASE WHEN cpg.evenement_type = "Soldes" THEN 1 ELSE 0 END) AS soldes,
      MAX(CASE WHEN cpg.evenement_type = "Noël" THEN 1 ELSE 0 END) AS noel,
      MAX(CASE WHEN cpg.evenement_type = "Black Friday" THEN 1 ELSE 0 END) AS black_friday
    FROM `rising-mercury-463817-s5.carttrend_rawdata.facts_commandes` c
    JOIN `rising-mercury-463817-s5.carttrend_rawdata.dim_date` dt
      ON dt.id_date = c.id_date_commande
    JOIN `rising-mercury-463817-s5.carttrend_rawdata.dim_details_commandes` dc
      ON c.id_commande = dc.id_commande
    JOIN `rising-mercury-463817-s5.carttrend_rawdata.dim_produits` p
      ON dc.id_details_produits = p.id_produit
    LEFT JOIN `rising-mercury-463817-s5.carttrend_rawdata.facts_campaigns` cpg
      ON dt.id_date = cpg.id_date
```

```
    WHERE LOWER(c.statut_commande) NOT IN ('annulée', 'cancelled')
    GROUP BY dt.date, annee, semaine
    ORDER BY dt.date
    """

    df = client.query(query).to_dataframe()
    df["date"] = pd.to_datetime(df["date"])
    df["annee_semaine"] = df["annee"].astype(str) + "-" + df["semaine"].astype(str).str.zfill(2)

    df_week = df.groupby(["annee", "semaine", "annee_semaine"], as_index=False).agg({
        "chiffre_affaires": "sum",
        "soldes": "sum",
        "noel": "sum",
        "black_friday": "sum"
    })

    # Ajout de la variable "aucun_evenement"
    df_week["aucun_evenement"] = ((df_week["soldes"] == 0) &
                                  (df_week["noel"] == 0) &
                                  (df_week["black_friday"] == 0)).astype(int)

    return df_week


# XGB Rolling

def rolling_forecast_xgb_simple(df, train_size=15):
    df = df.copy().reset_index(drop=True)

    # Lags
    for lag in [1, 2, 3, 4]:
        df[f"CA_m{lag}"] = df["chiffre_affaires"].shift(lag)
    df = df.dropna().reset_index(drop=True)

    # Features
    features = ["soldes", "noel", "black_friday", "aucun_evenement",
                "CA_m1", "CA_m2", "CA_m3", "CA_m4"]

    y_true, y_pred = [], []

    for i in range(train_size, len(df)):
        train = df.iloc[:i]
        test = df.iloc[i:i+1]

        model = XGBRegressor(n_estimators=100, max_depth=3, learning_rate=0.1, random_state=42)
        model.fit(train[features], train["chiffre_affaires"])

        y_hat = model.predict(test[features])
        y_true.append(test["chiffre_affaires"].values[0])
        y_pred.append(y_hat[0])

    df_eval = df.iloc[train_size:].copy()
    df_eval["y_true"] = y_true
    df_eval["y_pred_xgb"] = y_pred
    return model, df_eval, features


# SARIMAX Rolling

def rolling_forecast_sarimax(df, train_size=15):
    df = df.copy().reset_index(drop=True)
    y_true, y_pred = [], []

    # Forcer typage en float
    for col in ["soldes", "noel", "black_friday", "aucun_evenement"]:
        df[col] = df[col].astype(float)

    for i in range(train_size, len(df)):
        train = df.iloc[:i]
        test = df.iloc[i:i+1]

        exog_train = train[["soldes", "noel", "black_friday", "aucun_evenement"]]
        exog_test = test[["soldes", "noel", "black_friday", "aucun_evenement"]]

        model = SARIMAX(
```

```
            train["chiffre_affaires"].astype(float),
            exog=exog_train,
            order=(1,1,1),
            seasonal_order=(1,0,0,4),
            enforce_stationarity=False,
            enforce_invertibility=False
        )
        results = model.fit(disp=False)
        y_hat = results.predict(start=len(train), end=len(train), exog=exog_test)

        y_true.append(test["chiffre_affaires"].values[0])
        y_pred.append(y_hat.values[0])

    df_eval = df.iloc[train_size:].copy()
    df_eval["y_true"] = y_true
    df_eval["y_pred_sarimax"] = y_pred
    return df_eval



#  Exécution

df_week = get_weekly_data_from_bigquery()
print(f"{len(df_week)} semaines disponibles après extraction")

# XGB
model_xgb, df_eval_xgb, _ = rolling_forecast_xgb_simple(df_week, train_size=15)

# SARIMAX
df_eval_sarimax = rolling_forecast_sarimax(df_week, train_size=15)
```

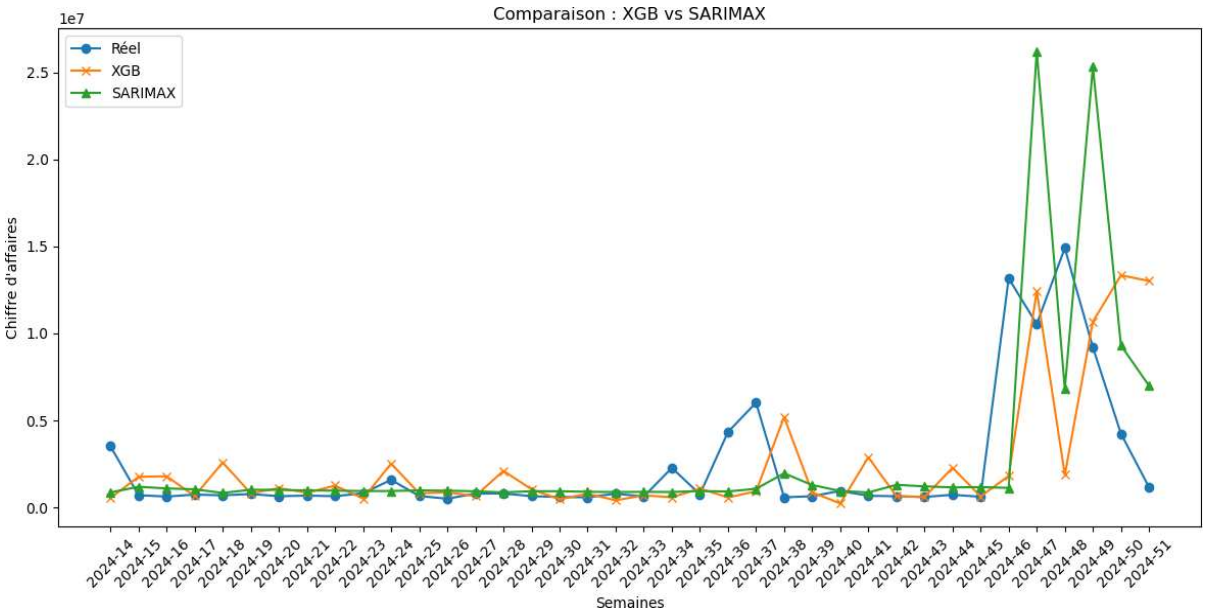
```
C:\Users\sigle\AppData\Roaming\Python\Python312\site-packages\google\cloud\bigquery\table.py:1965:  UserWa
rning: BigQuery Storage module not found, fetch data with the REST endpoint instead.
  warnings.warn(
57 semaines disponibles après extraction
```

Graphique : Comparaison des prédictions XGBoost et SARIMAX vs Réel (CA hebdomadaire).

Objectif : Le graphique compare le **chiffre d'affaires réel** avec les prédictions de deux modèles :

- **XGB** : modèle basé sur XGBoost
- **SARIMAX** : modèle de séries temporelles avec saisonnalité et événements extérieurs

```
def plot_comparison(df_eval_xgb, df_eval_sarimax):
    df_eval = df_eval_xgb[["annee_semaine","y_true","y_pred_xgb"]] \
        .merge(df_eval_sarimax[["annee_semaine","y_pred_sarimax"]],  on="annee_semaine")

    plt.figure(figsize=(12,6))
    plt.plot(df_eval["annee_semaine"], df_eval["y_true"], label="Réel", marker="o")
    plt.plot(df_eval["annee_semaine"], df_eval["y_pred_xgb"], label="XGB", marker="x")
    plt.plot(df_eval["annee_semaine"],  df_eval["y_pred_sarimax"],  label="SARIMAX",  marker="^")

    # Rotation et titres des axes
    plt.xticks(rotation=45)
    plt.xlabel("Semaines")
    plt.ylabel("Chiffre d'affaires")

    # Titre et légende
    plt.legend()
    plt.title("Comparaison : XGB vs SARIMAX")
    plt.tight_layout()
    plt.show()

    return df_eval


# Affichage graphique
df_eval = plot_comparison(df_eval_xgb, df_eval_sarimax)
```



Résultats du graphique :

Période calme (semaines 2024-14 à ~2024-42) :
Le chiffre d'affaires reste relativement bas et stable.
Les deux modèles (XGB et SARIMAX) collent plutôt bien à la réalité, avec quelques petites fluctuations.
SARIMAX est souvent "trop lisse", il sous-estime les petits pics.

Période de forte hausse (à partir de ~2024-45) :
Explosion du chiffre d'affaires réel (plusieurs millions).
Cela correspond probablement à des événements comme le Black Friday, Noël, Soldes, ...
XGB suit partiellement la tendance, mais reste sous-estimé sur les plus gros pics.
SARIMAX réagit beaucoup plus fortement, parfois en exagérant les pics (il sur-prédit lors de 2024-47 et 2024-49).

Après les pics (2024-50 et 2024-51) :
Retour du CA réel à un niveau "normal".
XGB redescend plus doucement.
SARIMAX chute très vite, mais reste trop haut par rapport au réel sur certaines semaines.

Interprétation :
XGB : plus stable, suit mieux la tendance générale, mais a du mal à capter les pics extrêmes.
SARIMAX : très sensible aux événements et à la saisonnalité, il réagit fort mais a tendance à sur-prédire dans les périodes de pics.

Conclusion :
Pour les périodes normales, les deux modèles sont corrects.
Lors des périodes exceptionnelles (fortes hausses), SARIMAX est trop volatil c'est à dire qu'il suit de très près les variations du réel mais qu'il va parfois au-dessus ou en dessous du réel, XGB est plus prudent mais sous-estime.

Conclusion finale :

Analyse des résultats :

Random Forest (journalier, hebdo, mensuel) :
Sur les séries journalières, les prédictions sont trop bruitées (écarts importants et pics mal anticipés).
En hebdomadaire et surtout en mensuel, les résultats sont plus stables, mais le modèle tend à sous-estimer les pics de chiffre d'affaires.

XGBoost :
Meilleure stabilité que Random Forest.
En hebdomadaire et avec le Rolling Forecast, les prédictions suivent beaucoup mieux la tendance réelle, même si les pics extrêmes restent sous-estimés. Globalement, XGB est plus robuste et plus proche du réel que Random Forest.

SARIMAX :
Le modèle capte une partie de la tendance mais il est trop volatil, surtout sur les pics, il amplifie certaines variations plutôt que de les lisser.
Pas idéal seul, mais peut être utile en combinaison (hybride avec XGB).

Recommandations :
Privilégier XGBoost (hebdomadaire + rolling forecast) : c'est la méthode qui donne les prédictions les plus proches du réel, avec une meilleure capacité à suivre la dynamique du CA.
Échelle hebdomadaire à favoriser : elle représente un bon compromis entre bruit (journalier) et perte d'information (mensuel).
Random Forest à écarter pour la prédiction fine, mais peut être gardée comme modèle de comparaison.
SARIMAX uniquement comme outil de vérification (utile pour analyser les tendances, mais pas pour une prévision fiable en production).

Améliorations possibles :
Introduire des variables exogènes (saisonnalité, jours fériés, promos, météo, effectif de production, ...).
Tester des modèles hybrides (XGBoost + SARIMAX ou Prophet).
Ajuster les hyperparamètres (notamment profondeur, taux d'apprentissage pour XGB).

Conclusion :
Il est recommandé d'utiliser avec les données actuelles XGBoost avec une granularité hebdomadaire et un rolling forecast, car c'est le plus adapté pour anticiper les variations de chiffre d'affaires et prendre des décisions (recrutements, maintenance, stocks, ...).