Prédiction des temps d'arrêt machines afin de planifier des contrôles préventifs plus fréquents et ainsi limiter les retards de livraison liés aux pannes.

Premier test : Modèle Random forest sur la logistique

Objectif : Ce script se connecte à Google BigQuery, extrait les données liées aux interventions sur les machines et aux entrepôts, prépare les données (nettoyage + encodage), entraîne un modèle de Random Forest pour prédire le temps d'arrêt des machines, puis évalue ses performances à l'aide de métriques (MAE, MSE, RMSE, R^2).

```
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

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

def train_rf_from_bigquery():
    """
    Fonction principale :
    - Connexion à BigQuery
    - Extraction des données
    - Préparation (nettoyage + encodage)
    - Entraînement du modèle Random Forest
    - Prédictions et résumé des résultats

    Returns:
        rf: modèle RandomForestRegressor entraîné
        X_test: features de test
        y_test: cibles de test
        y_pred: prédictions du modèle
        df_summary: résumé des temps réels vs prédits par localisation
    """

    # --- Connexion à BigQuery avec le projet défini ---
    client = bigquery.Client(project="projet-carttrend-468110p")

    # --- Requête SQL : sélection des colonnes utiles + jointure avec la table des entrepôts ---
    query = """
    SELECT DISTINCT
          c.id_intervention_machine,
          c.etat_machine,
          c.temps_d_arret,
          c.volume_traite,
          c.mois,
          c.id_machine,
          c.id_entrepot,
          d.localisation
    FROM `projet-carttrend-468110p.carttrend_rawdata.facts_entrepots_machine`  c
    JOIN `projet-carttrend-468110p.carttrend_rawdata.dim_entrepots` d ON c.id_entrepot=d.id_entrepot
    """

    # --- Exécution de la requête et chargement des données dans un DataFrame pandas ---
    df = client.query(query).to_dataframe()

    # --- Suppression des colonnes non nécessaires à la modélisation ---
    df = df.drop(columns=['id_intervention_machine'])

    # --- Définition des colonnes catégorielles à encoder ---
    cat_cols = ['localisation', 'etat_machine', 'id_machine', 'id_entrepot']

    # Sauvegarde de la localisation (utile pour le résumé par entrepôt après entraînement)
```

```
    entrepot_col = df['localisation']

    # --- Encodage des variables catégorielles (One-Hot Encoding) ---
    df_encoded = pd.get_dummies(df, columns=cat_cols, drop_first=True)

    # --- Séparation features (X) et cible (y = temps d'arrêt) ---
    X = df_encoded.drop(columns=['temps_d_arret']) y = df_encoded['temps_d_arret']

    # --- Découpage en données d'entraînement et de test (80/20) ---
    X_train, X_test, y_train, y_test, entrepot_train, entrepot_test = train_test_split( X, y, entrepot_col,
        test_size=0.2, random_state=42
    )

    # --- Entraînement du modèle Random Forest ---
    rf = RandomForestRegressor(n_estimators=100, random_state=42) rf.fit(X_train, y_train)

    # --- Prédictions sur le jeu de test ---
    y_pred = rf.predict(X_test)

    # --- Construction d'un DataFrame résultats (comparaison réel vs prédiction) ---
    df_results = pd.DataFrame({
        "localisation": entrepot_test, "temps_d_arret réel": y_test, "Prédiction": y_pred
    })

    # --- Agrégation des résultats par localisation (moyennes) ---
    df_summary = df_results.groupby("localisation").agg(
        temps_d_arret_moyen=('temps_d_arret réel', 'mean'), prediction_moyenne=('Prédiction', 'mean')
    ).reset_index()

    # Retourne le modèle, les données de test/prédiction, et le résumé
    return rf, X_test, y_test, y_pred, df_summary


# --- Programme principal ---
if __name__ == "__main__":
    # Entraînement et récupération des résultats
    rf_model, X_test, y_test, y_pred, df_summary = train_rf_from_bigquery() print(df_summary)

    # --- Évaluation du modèle avec des métriques de régression ---
    mae = mean_absolute_error(y_test, y_pred)   # Erreur absolue moyenne
    mse = mean_squared_error(y_test, y_pred)   # Erreur quadratique moyenne
    rmse = np.sqrt(mse)                        # Racine de MSE (plus interprétable)
    r2 = r2_score(y_test, y_pred)              # Coefficient de détermination R²

    # Affichage des résultats
    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(
```

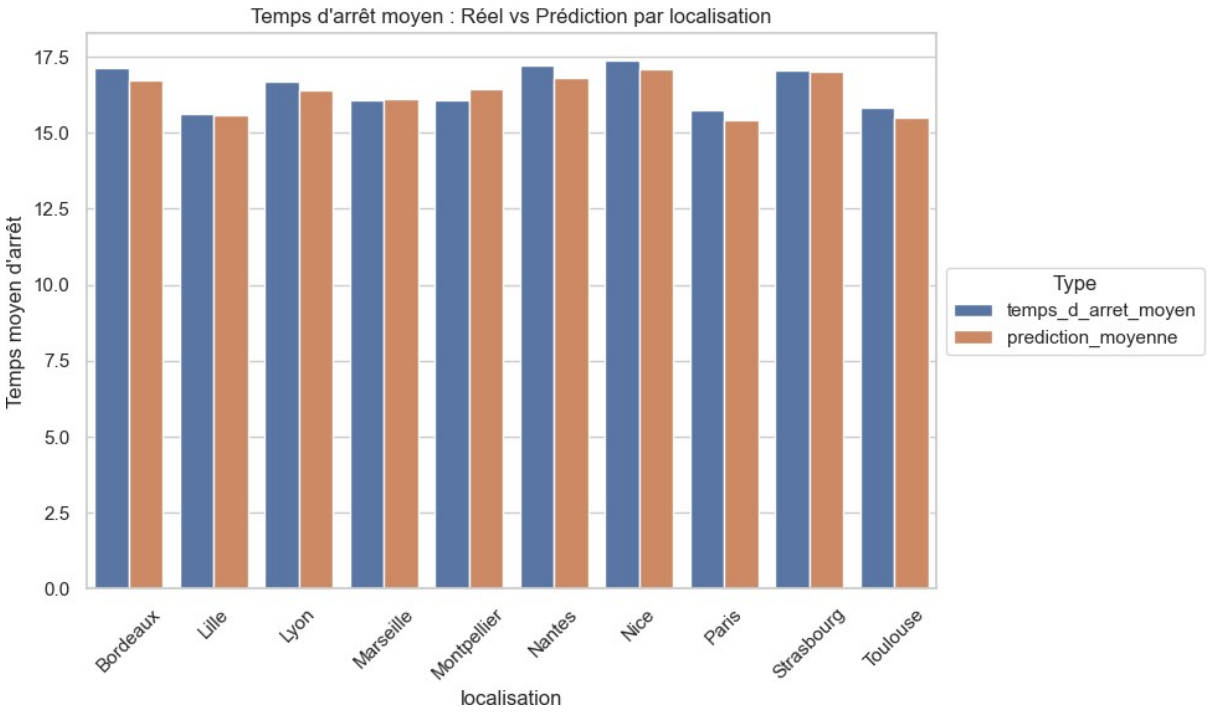
	localisation	temps_d_arret_moyen	prediction_moyenne
0	Bordeaux	17.137845	16.719288
1	Lille	15.614243	15.600536
2	Lyon	16.680693	16.401235
3	Marseille	16.068807	16.112242
4	Montpellier	16.062338	16.419901
5	Nantes	17.218232	16.791918
6	Nice	17.391705	17.098874
7	Paris	15.729545	15.425545
8	Strasbourg	17.03913	16.995513
=9	Toulouse	15.826923	15.515883

```
Évaluation du modèle Random Forest :
MAE  : 1.95
MSE  : 11.98
RMSE : 3.46
R²   : 0.9551
```

Graphique plot : Comparaison Réel vs Prédiction du temps d'arrêt des machines par localisation

Objectif : Ce script génère un graphique comparant le temps d'arrêt moyen réel et le temps d'arrêt moyen prédit par le modèle, regroupés par localisation d'entrepôt. Il permet de visualiser les performances du modèle Random Forest en fonction des entrepôts.

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np

# Style graphique plus propre (fond blanc avec grille)
sns.set(style="whitegrid")

# --- Préparation des données pour le graphique ---
# Transformation du DataFrame df_summary pour passer d'un format large à un format long #
Cela permet de comparer facilement "temps_d_arret_moyen" et "prediction_moyenne"
df_summary_plot = df_summary.melt(
    id_vars="localisation",                          # colonne qui reste fixe
    value_vars=["temps_d_arret_moyen", "prediction_moyenne"],  # colonnes à comparer
    var_name="Type",                                 # nouvelle colonne indiquant si c'est réel ou prédi
    value_name="Temps"                               # nouvelle colonne contenant les valeurs
)

# --- Création du graphique ---
plt.figure(figsize=(10, 6))  # définir la taille du graphique
sns.barplot(
    data=df_summary_plot,
    x="localisation", y="Temps",
    hue="Type"   # différencie les barres entre réel et prédiction
)

# --- Personnalisation du graphique ---
plt.title("Temps d'arrêt moyen : Réel vs Prédiction par localisation")
plt.xticks(rotation=45)
plt.ylabel("Temps moyen d'arrêt")

# --- Déplacer la légende à droite ---
plt.legend(loc="center left", bbox_to_anchor=(1, 0.5), title="Type")

# --- Affichage ---
plt.tight_layout()
plt.show()
```



Résultats du graphique :

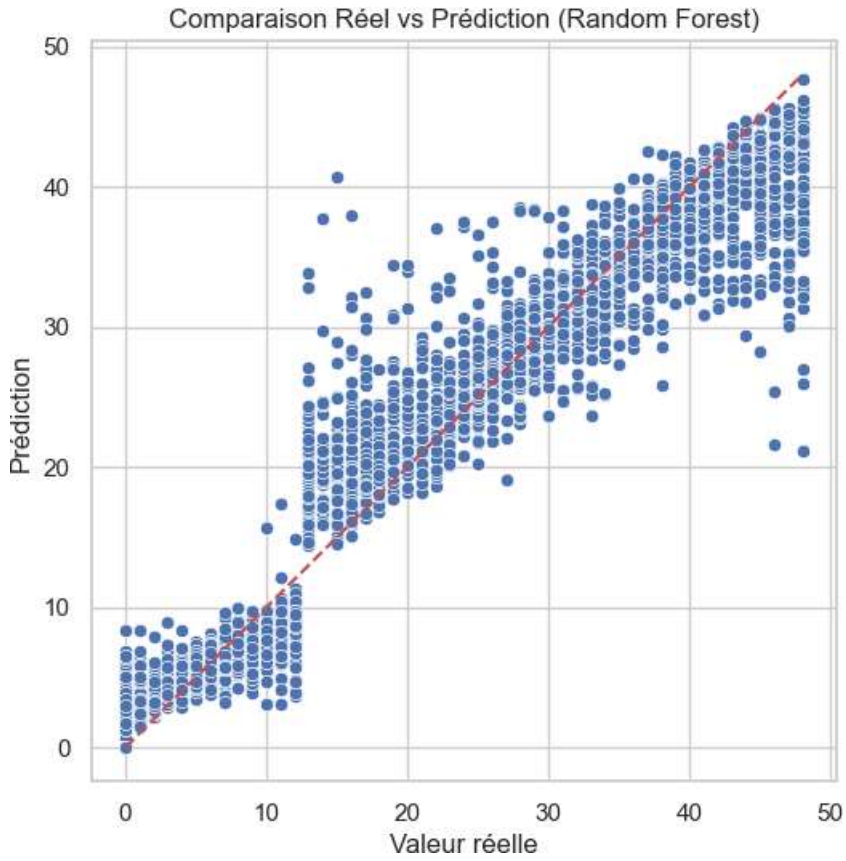
- Le modèle arrive bien à capturer la tendance des temps d'arrêt par localisation. • La différence entre réel et prédiction est faible (pas de gros écart visible).
- Le modèle ne semble pas sur-apprendre sur certaines villes (pas d'écarts systématiques en faveur du réel ou du prédit).
- Dans certaines localisations (ex : Nantes, Lyon), le modèle est légèrement en sous-estimation (prédiction plus basse que le réel).

Dans d'autres (ex : Toulouse, Marseille), il est plutôt en sur-estimation.

Graphique Scatterplot : Scatterplot Réel vs Prédiction

Objectif : Ce script génère un graphique de type scatterplot comparant les valeurs réelles du temps d'arrêt avec les valeurs prédites par le modèle Random Forest. Il permet d'évaluer la qualité des prédictions et de voir si elles s'alignent correctement avec les valeurs observées.

```
# --- Création du graphique ---
plt.figure(figsize=(6, 6))  # définir la taille du graphique (carré pour bien comparer)

# Nuage de points : chaque point représente (valeur réelle, prédiction)
sns.scatterplot(x=y_test,  y=y_pred)

# Ajout d'une ligne rouge en pointillés représentant la "parfaite prédiction"
# (si le modèle prédisait exactement la vérité, tous les points seraient sur cette ligne)
plt.plot(
    [y_test.min(),  y_test.max()],  # borne  min  et  max  de  l'axe  X
    [y_test.min(), y_test.max()], # borne min et max de l'axe Y 'r--
    '  # couleur rouge + ligne pointillée
)

# --- Personnalisation du graphique ---
plt.xlabel("Valeur réelle")   # nom de l'axe X
plt.ylabel("Prédiction")      # nom de l'axe Y
plt.title("Comparaison Réel vs Prédiction (Random Forest)")  # titre

# --- Affichage ---
plt.show()
```



Résultats du graphique :

- **Bonne tendance générale** :

La majorité des points sont proches de la diagonale : le modèle prédit bien les valeurs.
Cela confirme que le modèle Random Forest capture correctement la structure des données.

- **Dispersion plus forte pour certaines plages** :

Entre 10 et 20, on voit des grappes avec une dispersion plus large : le modèle a un peu plus de mal dans cette zone. Pour les valeurs élevées (35–50), on observe quelques sous-estimations ou surestimations.

- **Pas de biais systématique** :

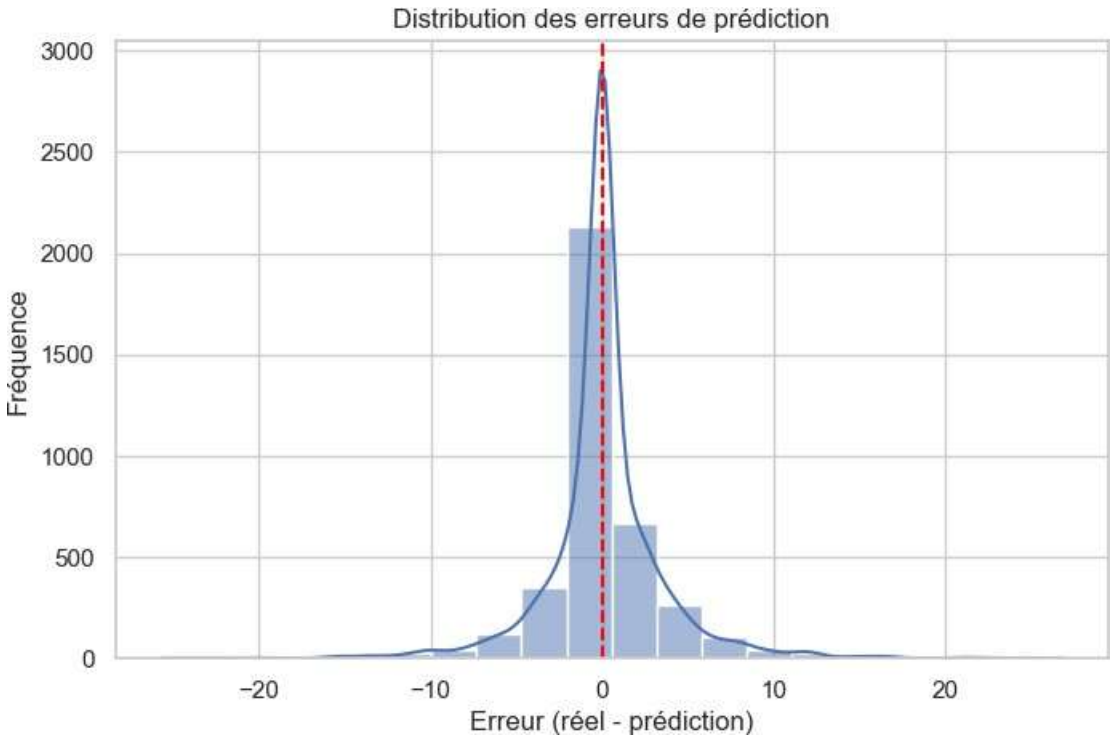
La diagonale est bien suivie sur l'ensemble de l'échelle : pas de sur-prédiction ou sous-prédiction systématique. Les erreurs semblent aléatoires et non dues à un problème structurel.

Graphique plot : Distribution des erreurs

Objectif : Ce script génère un histogramme représentant la distribution des erreurs de prédiction du modèle Random Forest. L'erreur est calculée comme la différence entre la valeur réelle et la valeur prédite (réel - prédiction). Cela permet d'analyser si le modèle tend à surestimer ou sous-estimer les temps d'arrêt.

```
# --- Calcul des erreurs ---
errors = y_test - y_pred   # différence entre valeurs réelles et prédites

# --- Création du graphique ---
plt.figure(figsize=(8, 5))  # définir la taille du graphique
sns.histplot(errors, bins=20, kde=True)  # histogramme + courbe de densité (kde=True)

# Ajout d'une ligne verticale rouge en pointillés à 0
# (permet de voir si les erreurs sont centrées autour de zéro)
plt.axvline(0, color='red', linestyle='--')

# --- Personnalisation du graphique ---
```

```
plt.xlabel("Erreur (réel - prédiction)")  # axe X = valeur de l'erreur
plt.ylabel("Fréquence")                   # axe Y = nombre d'occurrences
plt.title("Distribution des erreurs de prédiction")  # titre

# --- Affichage ---
plt.show()
```



Résultats du graphique :

- **Distribution centrée sur 0** :

La majorité des erreurs sont proches de 0 : le modèle est globalement non biaisé. Le modèle n'a pas tendance à systématiquement surestimer ou sous-estimer.

- **Forme symétrique et pic élevé** :

Les erreurs sont concentrées autour de 0 avec une forte densité : très bonnes prédictions dans la plupart des cas.
La courbe ressemble à une distribution normale, mais avec des queues un peu plus longues (quelques cas difficiles).

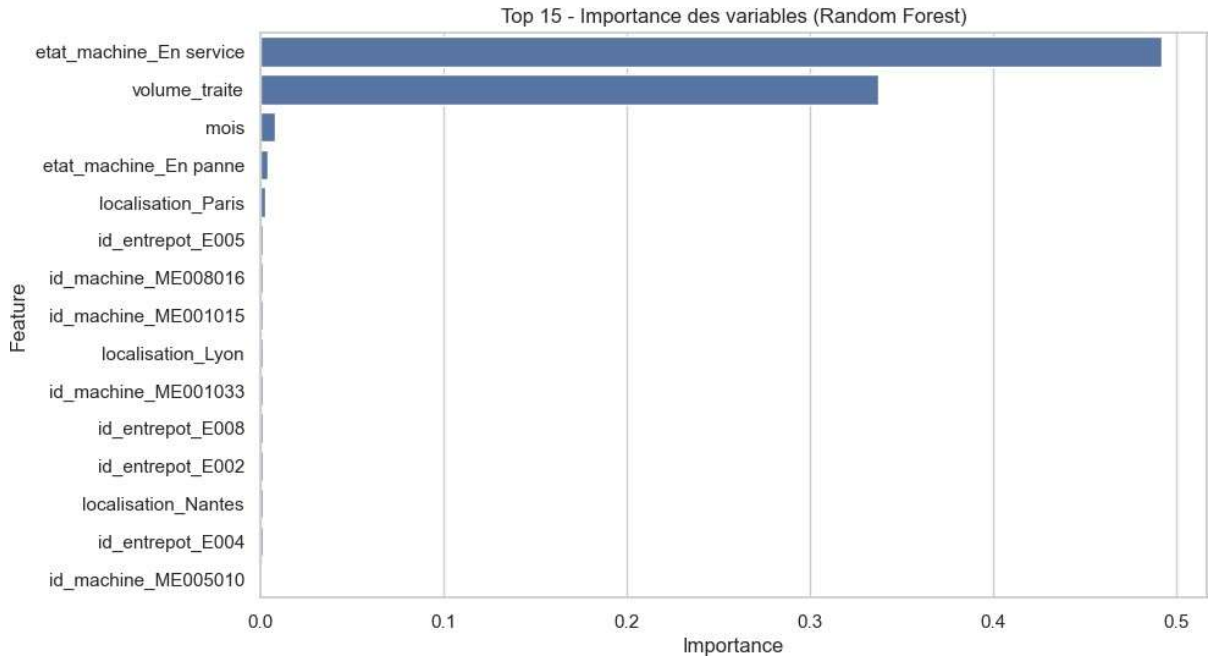
- **Présence d'outliers** :

Quelques erreurs vont jusqu'à -20 et +20 : rares cas où le modèle se trompe fortement.
Cela peut correspondre à des observations atypiques ou mal représentées dans l'entraînement.

Graphique plot : Importance des variables

Objectif : Ce script génère un graphique en barres représentant l'importance relative des variables utilisées par le modèle Random Forest pour prédire le temps d'arrêt. Cela permet d'identifier les facteurs les plus influents sur la prédiction.

```
importances = pd.Series(rf_model.feature_importances_, index=X_test.columns).sort_values(ascending=False
plt.figure(figsize=(10, 6))
sns.barplot(x=importances[:15],  y=importances.index[:15])
plt.title("Top 15 - Importance des variables (Random Forest)")
plt.xlabel("Importance")
plt.ylabel("Feature") plt.show()
```



Résultats du graphique :

- **Variables dominantes** :

Etat_machine_En service (0.48) et volume_traite (0.32) expliquent à elles seules plus de 80 % de l'importance totale. Cela veut dire que le modèle base principalement ses prédictions sur l'état des machines et le volume traité.

- **Variables secondaires** :

Mois et etat_machine_En panne ont une petite importance mais restent prises en compte. Cela suggère qu'il existe une saisonnalité ou des comportements liés aux pannes.

- **Variables négligeables** :

Les identifiants (id_machine, id_entrepot) et certaines localisations (Paris, Lyon, Nantes, Toulouse) ont une importance très faible : le modèle ne les considère presque pas.
Cela indique que les effets spécifiques à un site ou une machine précise n'apportent pas beaucoup de valeur ajoutée.

Second test : Modèle XGB sur la logistique

Objectif : Ce script se connecte à Google BigQuery, extrait les données liées aux interventions des machines et aux entrepôts, prépare les données (nettoyage + encodage), entraîne un modèle XGBoost Regressor pour prédire le temps d'arrêt des machines, puis évalue ses performances à l'aide de métriques de régression (MAE, MSE, RMSE, R^2).

```
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
import xgboost as xgb
import os

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

def train_xgb_from_bigquery(): """
    Fonction principale :
    - Connexion à BigQuery
    - Extraction des données
    - Préparation (nettoyage + encodage)
    - Entraînement du modèle XGBoost
    - Prédictions et résumé des résultats par localisation
```

```
Returns:
    xgb_model: modèle XGBRegressor entraîné X_test: features de test
    y_test: cibles de test
    y_pred: prédictions du modèle
    df_summary: résumé des temps réels vs prédits par localisation
"""

# --- Connexion à BigQuery ---
client = bigquery.Client(project="projet-carttrend-468110p")

# --- Requête SQL pour récupérer les données machines + entrepôts ---
query = """
SELECT DISTINCT
      c.id_intervention_machine, c.etat_machine,
      c.temps_d_arret, c.volume_traite, c.mois,
      c.id_machine, c.id_entrepot, d.localisation
FROM  `projet-carttrend-468110p.carttrend_rawdata.facts_entrepots_machine`  c
JOIN `projet-carttrend-468110p.carttrend_rawdata.dim_entrepots` d ON c.id_entrepot=d.id_entrepot """

# --- Exécuter la requête et charger les résultats dans un DataFrame pandas ---
df = client.query(query).to_dataframe()

# --- Suppression de colonnes inutiles pour la modélisation ---
df = df.drop(columns=['id_intervention_machine'])

# --- Colonnes catégorielles à encoder ---
cat_cols = ['localisation', 'etat_machine', 'id_machine', 'id_entrepot']

# Sauvegarde de la localisation pour analyse après modélisation
localisation_col = df['localisation']

# --- Encodage des variables catégorielles (One-Hot Encoding) ---
df_encoded = pd.get_dummies(df, columns=cat_cols, drop_first=True)

# --- Séparation features (X) et cible (y = temps d'arrêt) ---
X = df_encoded.drop(columns=['temps_d_arret']) y = df_encoded['temps_d_arret']

# --- Split train/test (80% apprentissage, 20% test) ---
X_train, X_test, y_train, y_test, loc_train, loc_test = train_test_split( X, y, localisation_col,
    test_size=0.2, random_state=42
)

# --- Définition et entraînement du modèle XGBoost ---
xgb_model = xgb.XGBRegressor(
    n_estimators=200,         # nombre d'arbres
    learning_rate=0.1,        # taux d'apprentissage
    max_depth=6,              # profondeur max des arbres
    random_state=42,          # reproductibilité
    objective='reg:squarederror'  # fonction de perte adaptée à la régression
)
xgb_model.fit(X_train, y_train)

# --- Faire les prédictions sur le jeu de test ---
y_pred = xgb_model.predict(X_test)

# --- Construction d'un DataFrame de comparaison réel vs prédiction ---
df_results = pd.DataFrame({ "localisation": loc_test,
    "temps_d_arret réel": y_test, "Prédiction": y_pred
})

# --- Calcul des moyennes par localisation ---
df_summary = df_results.groupby("localisation").agg(
    temps_d_arret_moyen=('temps_d_arret réel', 'mean'), prediction_moyenne=('Prédiction', 'mean')
   ).reset_index()

   return xgb_model, X_test, y_test, y_pred, df_summary

# --- Programme principal ---
if __name__ == "__main__":
    # Entraînement et récupération des résultats
    xgb_model, X_test, y_test, y_pred, df_summary = train_xgb_from_bigquery()
    print(df_summary)
```

```
    # --- Évaluation du modèle avec métriques ---
    mae = mean_absolute_error(y_test, y_pred)   # Erreur absolue moyenne
    mse = mean_squared_error(y_test, y_pred)    # Erreur quadratique moyenne
    rmse = np.sqrt(mse)                         # Racine carrée de MSE
    r2 = r2_score(y_test, y_pred)               # Coefficient de détermination R²

    # --- Affichage des résultats ---
    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}")
```

```
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(
```

	localisation	temps_d_arret_moyen	prediction_moyenne
0	Bordeaux	17.137845	16.351803
1	Lille	15.614243	15.221300
2	Lyon	16.680693	15.989435
3	Marseille	16.068807	16.162937
4	Montpellier	16.062338	16.213079
5	Nantes	17.218232	16.552317
6	Nice	17.391705	16.818026
7	Paris	15.729545	15.630764
8	Strasbourg	17.03913	16.565577
9	Toulouse	15.826923	15.201498

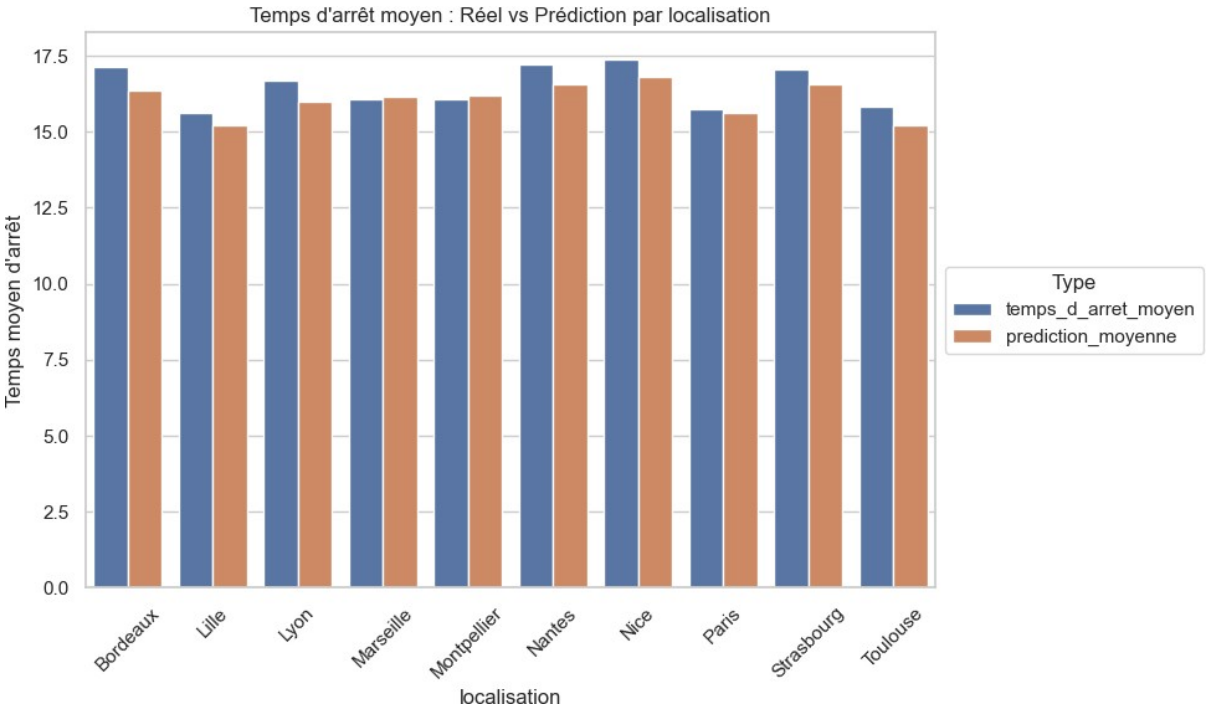
```
Évaluation du modèle XGBoost :
MAE  : 4.36
MSE  : 41.63
RMSE : 6.45
R²   : 0.8441
```

Graphique plot : Comparaison Réel vs Prédiction du temps d'arrêt des machines par localisation

Objectif : Ce script génère un graphique en barres comparant le temps d'arrêt moyen réel et le temps d'arrêt moyen prédit par le modèle XGBoost, regroupés par localisation d'entrepôt. Il permet de visualiser la performance du modèle selon les sites.

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np

# Style graphique propre (fond blanc + grille)
sns.set(style="whitegrid")

# --- Préparation des données ---
# Transformation du DataFrame df_summary en format long (melt)
# pour avoir une colonne "Type" (réel/prédiction) et une colonne "Temps"
df_summary_plot = df_summary.melt(
    id_vars="localisation",                          # colonne fixe
    value_vars=["temps_d_arret_moyen", "prediction_moyenne"],  # colonnes à comparer
    var_name="Type",                                 # nom de la nouvelle colonne catégorielle
    value_name="Temps"                               # nom de la colonne contenant les valeurs
)

# --- Création du graphique ---
plt.figure(figsize=(10, 6))
```

```
sns.barplot(
    data=df_summary_plot,
    x="localisation", y="Temps",
    hue="Type"
)

# --- Personnalisation ---
plt.title("Temps d'arrêt moyen : Réel vs Prédiction par localisation")
plt.xticks(rotation=45)
plt.ylabel("Temps moyen d'arrêt")

# --- Déplacer la légende à droite ---
plt.legend(loc="center left", bbox_to_anchor=(1, 0.5), title="Type")

# ---  Affichage ---
plt.tight_layout()
plt.show()
```



Résultats du graphique :

- Pour toutes les localisations, les prédictions sont proches des valeurs réelles.

Cela montre que le modèle XGBoost ne sous-estime ni ne surestime systématiquement une ville particulière. • Les écarts entre réel et prédiction sont visibles mais faibles (ex: Nantes, Lille).

On est probablement dans la marge d'erreur acceptable d'un modèle de régression.

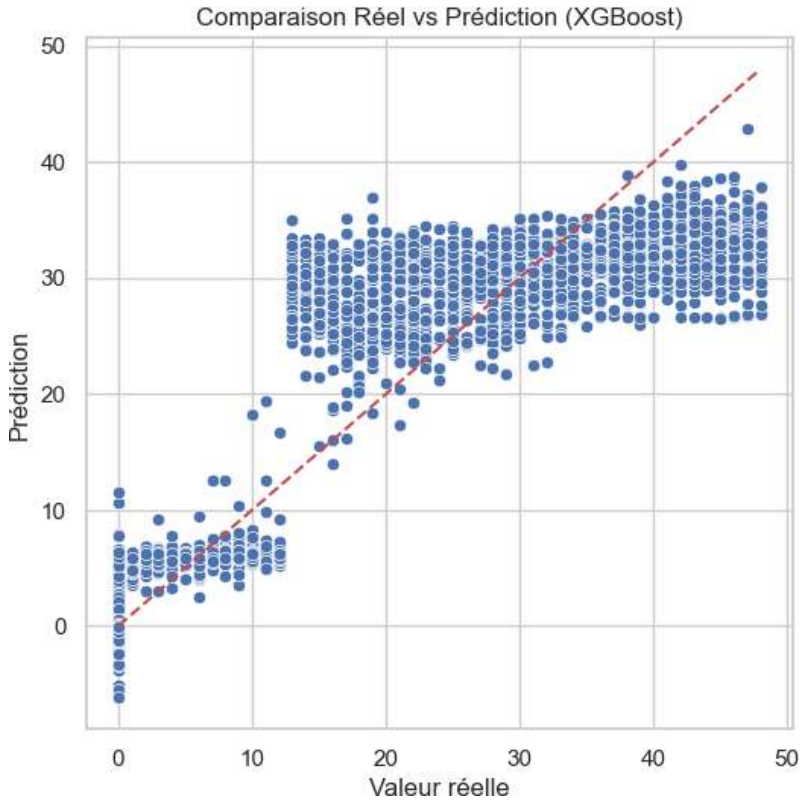
- Contrairement à la Random Forest (où l'importance des localisations était très faible), XGBoost semble mieux capter les effets spécifiques à chaque ville.

Cela confirme que XGBoost peut détecter des relations non linéaires plus fines entre localisation et temps d'arrêt.

Graphique Scatter plot : Scatter plot Réel vs Prédiction

Objectif : Ce script trace un scatterplot (nuage de points) comparant les valeurs réelles du temps d'arrêt avec les valeurs prédites par le modèle XGBoost.
Il permet de visualiser la qualité des prédictions : plus les points sont proches de la diagonale, plus le modèle est précis.

```
# --- Création du graphique ---
plt.figure(figsize=(6, 6))  # définir la taille du graphique (carré = meilleure comparaison)

# Chaque point = une prédiction vs sa valeur réelle
sns.scatterplot(x=y_test,  y=y_pred)

# Ajout de la diagonale rouge (référence de prédiction parfaite)
plt.plot(
    [y_test.min(),  y_test.max()],  # borne min et max de l'axe X
    [y_test.min(), y_test.max()], # borne min et max de l'axe Y 'r-
    -'  # couleur rouge, ligne pointillée
)

# --- Personnalisation ---
plt.xlabel("Valeur réelle")  # nom axe X
plt.ylabel("Prédiction")     # nom axe Y
plt.title("Comparaison Réel vs Prédiction (XGBoost)")  # titre du graphique

# --- Affichage ---
plt.show()
```



Résultats du graphique :

- **Alignement global correct** :

Beaucoup de points suivent la diagonale: le modèle capture bien la tendance générale.

- **Sous-estimation sur les fortes valeurs** :

Pour des valeurs réelles > 30, le modèle sous-estime systématiquement (les points passent sous la diagonale). Cela veut dire que XGBoost a du mal à prédire les arrêts très longs.

- **Bonne précision sur les petites valeurs** :

Entre 0 et 15, les prédictions collent assez bien. Peu de biais apparent dans cette zone.

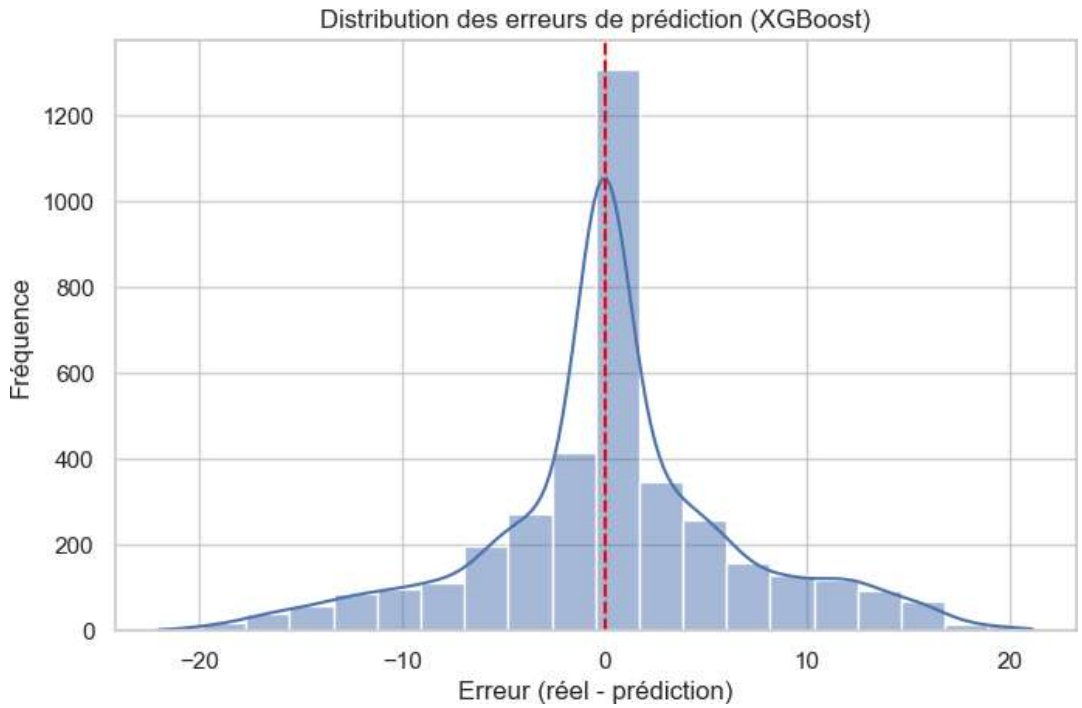
- **Cluster visible** :

On observe un regroupement dense entre 20 et 35 : signe que la majorité des arrêts moyens se situent dans cette plage. Le modèle est calibré dans cette zone, mais perd en précision quand on sort de cette "zone fréquente".

Graphique plot : Distribution des erreurs

Objectif : Ce script visualise la distribution des erreurs de prédiction du modèle XGBoost. L'erreur est calculée comme (valeur réelle - prédiction).
Cela permet d'analyser si le modèle a tendance à sous-estimer (erreur > 0) ou surestimer (erreur < 0) les temps d'arrêt.

```
# --- Calcul des erreurs ---
errors = y_test - y_pred  # différence entre réel et prédit

# --- Création du graphique ---
plt.figure(figsize=(8, 5))  # taille de la figure
sns.histplot(errors, bins=20, kde=True)  # histogramme + courbe de densité (kde=True)

# Ajout d'une ligne verticale rouge en 0 (erreur nulle)
plt.axvline(0, color='red', linestyle='--')

# --- Personnalisation ---
plt.xlabel("Erreur (réel - prédiction)")  # axe X = erreur
plt.ylabel("Fréquence")                   # axe Y = fréquence des erreurs
plt.title("Distribution des erreurs de prédiction (XGBoost)")  # titre

# --- Affichage ---
plt.show()
```



Résultats du graphique :

- **Distribution centrée autour de 0** :

Le pic principal est bien autour de zéro: le modèle ne présente pas de biais global fort.

- **Queue asymétrique à droite** :

Il y a quelques grosses sous-estimations (erreurs positives > 10).
Cela confirme ce qu'on voyait dans le scatter plot : XGBoost a du mal avec les grandes valeurs.

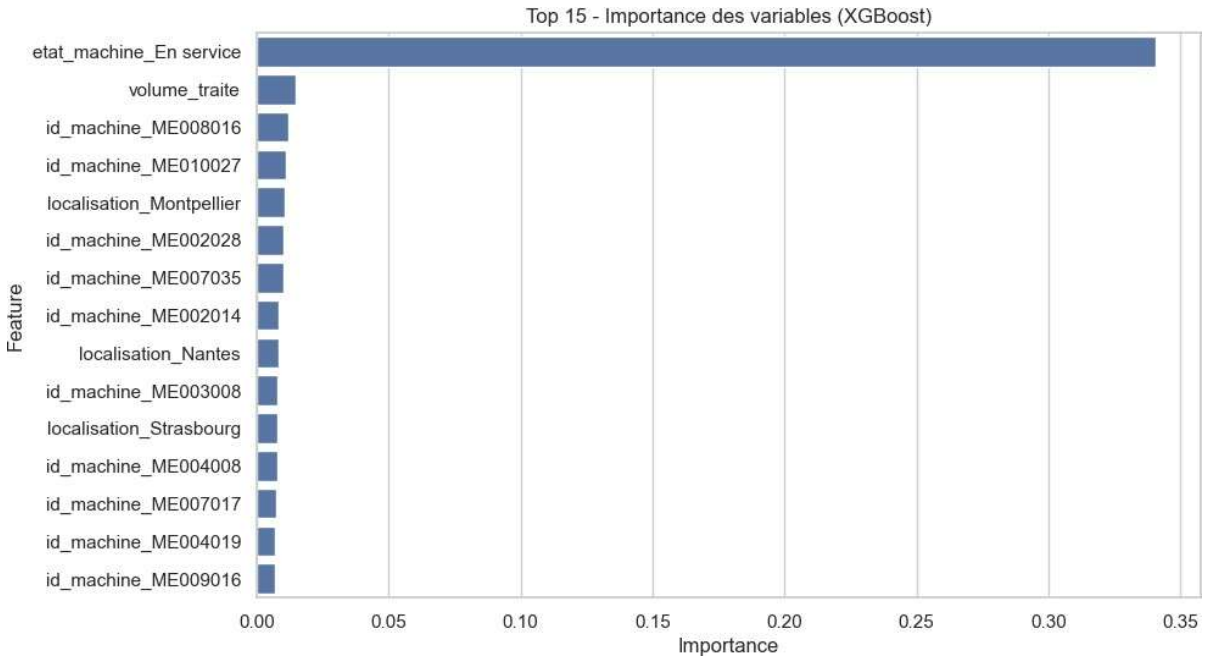
- **Variabilité faible pour la majorité des prédictions** :

La majorité des erreurs se situe entre -5 et +5 : le modèle est globalement précis.

Graphique plot : Importance des variables

Objectif : Ce script affiche les 15 variables (features) les plus importantes utilisées par le modèle XGBoost pour prédire le temps d'arrêt des machines.
L'importance des variables aide à comprendre quelles caractéristiques influencent le plus le modèle.

```
# --- Récupération des importances ---
# feature_importances_ donne le poids de chaque variable dans le modèle
importances = pd.Series(xgb_model.feature_importances_, index=X_test.columns)

# Tri décroissant (les plus importantes en premier)
importances = importances.sort_values(ascending=False)

# --- Création du graphique ---
plt.figure(figsize=(10, 6))  # taille de la figure

# Barplot des 15 variables les plus importantes
sns.barplot(x=importances[:15],  y=importances.index[:15])

# --- Personnalisation ---
plt.title("Top 15 - Importance des variables (XGBoost)")  # titre
plt.xlabel("Importance")    # axe X = score d'importance
plt.ylabel("Feature")       # axe Y = noms des variables

# --- Affichage ---
plt.show()
```



Résultats du graphique :

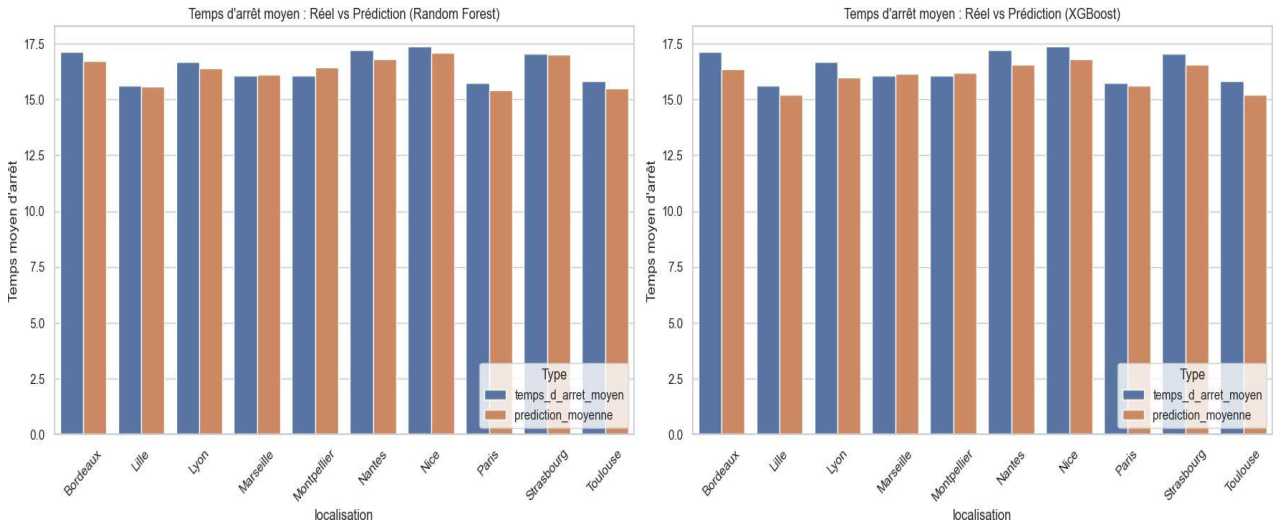
- **Etat_machine_En service domine largement** : c'est la variable la plus déterminante pour prédire le temps d'arrêt.
- **Volume_traite arrive en 2e position** : ce qui est logique, car plus le volume traité est important, plus le temps d'arrêt peut varier.
- **Certaines localisations (ex: Montpellier, Nantes)** : impact géographique, peut-être lié à l'utilisation des machines.
- **Plusieurs id_machine spécifiques** : certaines machines sont plus représentatives des temps d'arrêt.

Comparaisons des résultats des deux modèles

Graphique plot : Comparaison des performances de Random Forest et XGBoost pour la prédiction du temps d'arrêt moyen des machines

```
import matplotlib.pyplot as plt
import seaborn as sns

# --- Entraînement et récupération des résumés ---
rf_model, X_test_rf, y_test_rf, y_pred_rf, df_summary_rf = train_rf_from_bigquery()
xgb_model, X_test_xgb, y_test_xgb, y_pred_xgb, df_summary_xgb = train_xgb_from_bigquery()

# --- Transformation pour les graphiques ---
df_summary_rf_plot = df_summary_rf.melt(
    id_vars="localisation",
    value_vars=["temps_d_arret_moyen", "prediction_moyenne"], var_name="Type",
    value_name="Temps"
)

df_summary_xgb_plot = df_summary_xgb.melt(
    id_vars="localisation",
    value_vars=["temps_d_arret_moyen", "prediction_moyenne"], var_name="Type",
    value_name="Temps"
)

# --- Création des sous-graphiques côte à côte ---
sns.set(style="whitegrid")
fig, axes = plt.subplots(1, 2, figsize=(18, 6))

# --- Graphique Random Forest ---
sns.barplot(
    data=df_summary_rf_plot,
    x="localisation",
    y="Temps", hue="Type",
    ax=axes[0]
)
axes[0].set_title("Temps d'arrêt moyen : Réel vs Prédiction (Random Forest)")
axes[0].set_xticklabels(axes[0].get_xticklabels(), rotation=45)
axes[0].set_ylabel("Temps moyen d'arrêt")
axes[0].legend(loc="lower right", title="Type")

# --- Graphique XGBoost ---
sns.barplot(
    data=df_summary_xgb_plot,
    x="localisation",
    y="Temps", hue="Type",
    ax=axes[1]
)
axes[1].set_title("Temps d'arrêt moyen : Réel vs Prédiction (XGBoost)")
axes[1].set_xticklabels(axes[1].get_xticklabels(), rotation=45)
axes[1].set_ylabel("Temps moyen d'arrêt")
axes[1].legend(loc="lower right", title="Type")

# --- Ajustement ---
plt.tight_layout() plt.show()
```

```
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(
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(
C:\Users\sigle\AppData\Local\Temp\ipykernel_33608\1333259877.py:36: UserWarning: set_ticklabels() should only be
used with a fixed number of ticks, i.e. after set_ticks() or using a FixedLocator.
  axes[0].set_xticklabels(axes[0].get_xticklabels(),   rotation=45)
C:\Users\sigle\AppData\Local\Temp\ipykernel_33608\1333259877.py:49: UserWarning: set_ticklabels() should only be
used with a fixed number of ticks, i.e. after set_ticks() or using a FixedLocator.
  axes[1].set_xticklabels(axes[1].get_xticklabels(),   rotation=45)
```



Résultats :

- **Random Forest (à gauche)** :

Les prédictions (barres oranges) sont très proches des valeurs réelles (barres bleues). Les écarts sont faibles sur presque toutes les villes. Quelques légères sous-estimations apparaissent (Bordeaux, Paris, Toulouse), mais globalement la qualité prédictive est excellente. Malgré ces écarts, la tendance générale est bien captée.

- **XGBoost (à droite)** :

Le modèle a tendance à sous-estimer plus régulièrement les temps d'arrêt réels. Les écarts sont visibles notamment pour Lyon, Nantes, Nice, Strasbourg et Toulouse. Même si certaines villes sont bien prédites (Marseille, Montpellier, Paris), le modèle reste moins précis que Random Forest.

- **Comparaison générale** :

Random Forest : donne une bonne approximation, mais tend à sous-estimer légèrement. Les écarts sont visibles notamment pour Bordeaux, Lyon, Nantes, Nice, Strasbourg et Toulouse. Même si certaines villes sont bien prédites (Lille, Marseille, Montpellier), le modèle reste moins précis que Random Forest.

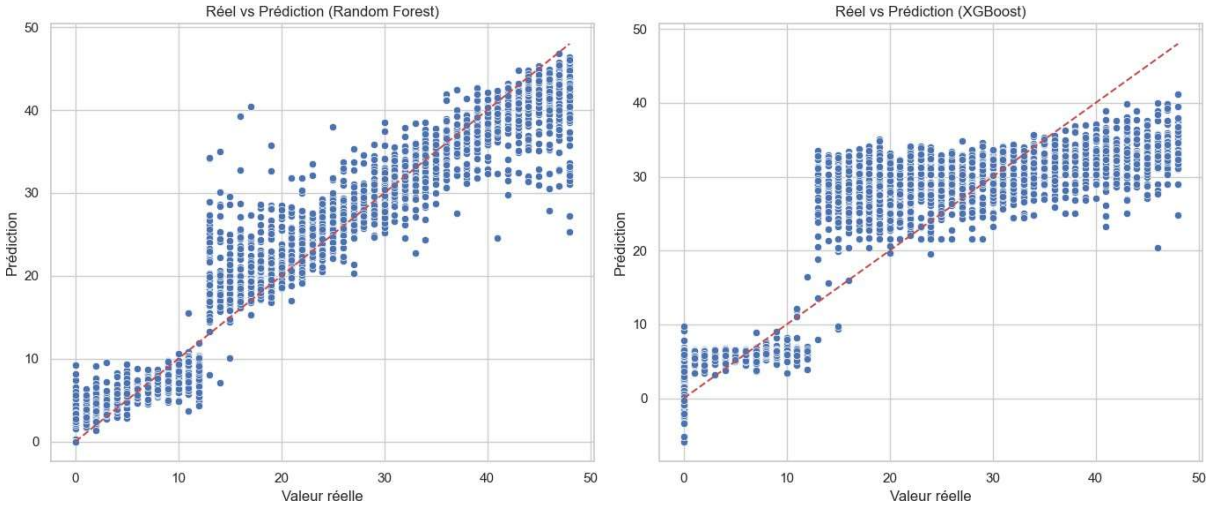
- **Conclusion** :

Les deux modèles sont capables de bien prédire le temps d'arrêt moyen en fonction de la localisation. Toutefois, Random Forest surpasse XGBoost en réduisant les écarts entre prédiction et réalité, ce qui en fait un modèle plus fiable pour ce cas d'étude.

Scatter Plot (Réel vs Prédiction) Random Forest vs XGBoost

```
# --- Prédictions des modèles ---
rf_pred = rf_model.predict(X_test)
xgb_pred = xgb_model.predict(X_test)

fig, axes = plt.subplots(1, 2, figsize=(14, 6))

# --- Random Forest ---
sns.scatterplot(x=y_test, y=rf_pred, ax=axes[0])
axes[0].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--') axes[0].set_title("Réel
vs Prédiction (Random Forest)")
axes[0].set_xlabel("Valeur réelle")
```

```
axes[0].set_ylabel("Prédiction")

# --- XGBoost ---
sns.scatterplot(x=y_test, y=xgb_pred, ax=axes[1])
axes[1].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--') axes[1].set_title("Réel
vs Prédiction (XGBoost)")
axes[1].set_xlabel("Valeur réelle")
axes[1].set_ylabel("Prédiction")

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



Résultats :

- **Random Forest (à gauche)** :

Les points sont assez bien alignés sur la diagonale, le modèle prédit donc correctement la tendance générale. Mais on voit quelques zones plates (valeurs prédictives bloquées autour de 30 ou 35) : c'est un effet typique des arbres de décision (RF "moyenne" les feuilles et donne des prédictions par paliers).
Résultat : il y a du bruit mais la corrélation reste forte.

- **XGBoost (à droite)** :

On remarque des zones horizontales très marquées, pour certaines plages de valeurs réelles (par ex. entre 10 et 30), XGBoost donne presque toujours la même prédiction (autour de 30).
Cela indique une sous-estimation pour les petites valeurs (XGB surestime quand réel < 20) et une sous-prédiction pour les grandes valeurs (XGB plafonne vers 40 même quand réel : 50).
Le modèle semble avoir moins de finesse que le Random Forest sur les données actuelles.

- **Comparaison générale** :

Random Forest capture mieux la variabilité (malgré du bruit).
XGBoost est plus "rigide" il écrase la variabilité et donne des prédictions trop uniformes.

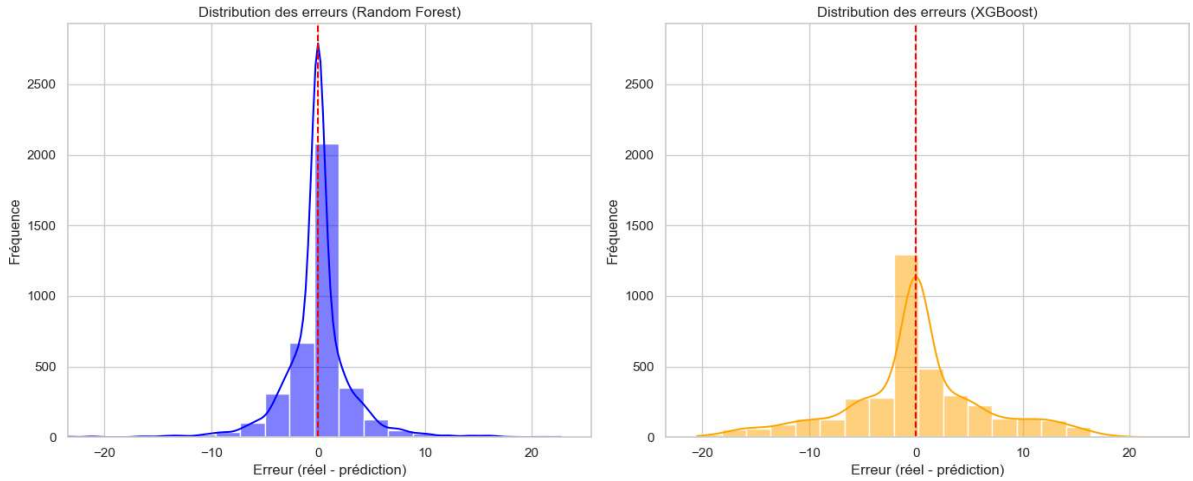
- **Conclusion** :

RF > XGB : Random Forest donne des prédictions plus proches de la diagonale et plus nuancées.
XGBoost pourrait être meilleur si on règle les hyperparamètres (n_estimators (nb d'arbres), max_depth (profondeur des arbres), learning_rate (vitesse d'apprentissage), ...).

Distribution des erreurs Random Forest vs XGBoost

```
# --- Prédictions des modèles ---
rf_pred = rf_model.predict(X_test)
xgb_pred = xgb_model.predict(X_test)

# --- Calcul des erreurs ---
errors_rf = y_test - rf_pred
```

```
errors_xgb = y_test - xgb_pred

fig, axes = plt.subplots(1, 2, figsize=(14, 6))

# --- Définir la même plage pour les erreurs
min_err = min(errors_rf.min(), errors_xgb.min()) max_err =
max(errors_rf.max(), errors_xgb.max())

# --- Random Forest ---
sns.histplot(errors_rf, bins=20, kde=True, ax=axes[0], color="blue")
axes[0].axvline(0, color='red', linestyle='--')
axes[0].set_xlim(min_err, max_err)  #  même échelle X
axes[0].set_title("Distribution des erreurs (Random Forest)")
axes[0].set_xlabel("Erreur (réel - prédiction)")
axes[0].set_ylabel("Fréquence")

# --- XGBoost ---
sns.histplot(errors_xgb, bins=20, kde=True, ax=axes[1], color="orange")
axes[1].axvline(0, color='red', linestyle='--')
axes[1].set_xlim(min_err, max_err)  # même échelle X
axes[1].set_title("Distribution des erreurs (XGBoost)") axes[1].set_xlabel("Erreur
(réel - prédiction)")
axes[1].set_ylabel("Fréquence")

#  --- uniformiser l'échelle Y aussi ---
max_y = max(axes[0].get_ylim()[1], axes[1].get_ylim()[1]) axes[0].set_ylim(0,
max_y)
axes[1].set_ylim(0, max_y)

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



Résultats :

- **Random Forest (à gauche)** :

La distribution est fortement centrée autour de 0. La majorité des erreurs se situent entre -5 et +5. Le pic est très marqué autour de 0, ce qui signifie que la plupart des prédictions sont très proches de la réalité. Quelques valeurs extrêmes apparaissent (jusqu'à environ +20 ou -20), mais elles sont rares. Indique une forte stabilité et une bonne précision globale.

- **XGBoost (à droite)** :

La distribution est également centrée autour de 0, mais elle est plus aplatie et plus étalée. Les erreurs sont plus dispersées : on observe un nombre non négligeable de cas avec des erreurs supérieures à ±10. Le pic central est moins haut que pour Random Forest, cela signifie qu'il y a moins de prédictions parfaitement proches de la réalité. Indique que XGBoost génère plus de variabilité et plus d'erreurs extrêmes que Random Forest.

- **Comparaison générale** :

Random Forest : Erreurs plus concentrées autour de 0, distribution plus fine et moins dispersée. Le modèle est plus précis et plus stable. XGBoost : Erreurs plus étalées, donc plus de prédictions éloignées de la valeur réelle. Le modèle est moins robuste que Random Forest sur ce jeu de données.

- **Conclusion** :

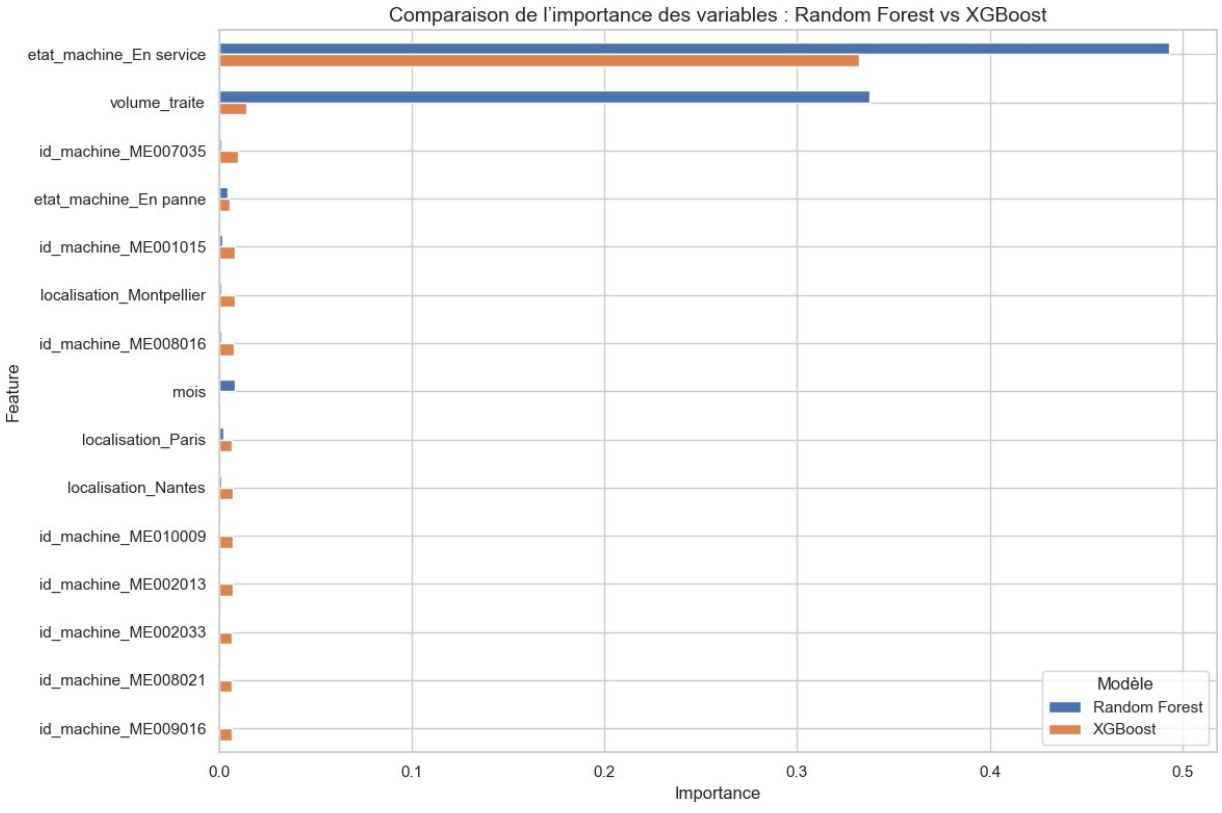
Random Forest est le modèle le plus performant dans ce cas, car il produit des erreurs plus faibles et plus concentrées autour de zéro. XGBoost, bien qu'efficace, présente une distribution d'erreurs plus large, ce qui traduit une moins bonne fiabilité dans les prédictions.

- **Recommandation** :

Pour prédire le temps d'arrêt moyen, Random Forest est à privilégier, car il offre une meilleure précision et réduit les risques d'erreurs extrêmes.

Comparaison par localisation Random Forest vs XGBoost

```
# --- Récupération des importances ---
rf_importances = pd.Series(rf_model.feature_importances_, index=X_test.columns).sort_values(ascending=Fa
xgb_importances = pd.Series(xgb_model.feature_importances_, index=X_test.columns).sort_values(ascending=

# --- Construire un DataFrame combiné ---
df_importances = pd.DataFrame({
    "Random Forest": rf_importances,
    "XGBoost": xgb_importances
})

# --- Garder seulement le top 15 (moyenne RF+XGB) ---
top_features = df_importances.mean(axis=1).sort_values(ascending=False).head(15).index df_top
= df_importances.loc[top_features]

# --- Plot comparatif ---
ax = df_top.plot(kind="barh", figsize=(12, 8))
plt.title("Comparaison de l'importance des variables : Random Forest vs XGBoost", fontsize=14)
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.gca().invert_yaxis()  # pour que la feature la plus importante soit en haut
plt.legend(title="Modèle")
plt.tight_layout()
plt.show()
```



Résultats :

- **Variables dominantes** :

Les deux modèles apprennent surtout à partir de etat_machine_En service et volume_traite.
Ces deux variables portent la quasi-totalité du signal, ce qui veut dire qu'elles expliquent presque tout le phénomène.

- **Différences dans la pondération** :

Random Forest met plus de poids sur etat_machine_En service (~ 0.49) et volume_traite (~ 0.33).
XGBoost répartit un peu plus l'importance entre ces deux variables : etat_machine_En service (≈ 0.35) et volume_traite (≈ 0.05).

- **Variables secondaires** :

RF donne un petit rôle à la variable mois, mais XGBoost préfère donner un peu d'importance à certains identifiants (id_machine_*) et localisations (Montpellier, Paris).
Cela traduit que XGBoost cherche des patterns plus spécifiques / précis, alors que RF reste plus global.

- **Interprétation générale** :

Les deux modèles captent le même signal principal : état de la machine + volume traité.
Mais Random Forest est plus tranché : il considère que quasiment rien d'autre n'apporte d'information. XGBoost essaie d'exploiter des variables plus fines, mais leur poids reste faible.

- **Conclusion** :

Les deux modèles apprennent les mêmes drivers principaux (machine en service et volume).

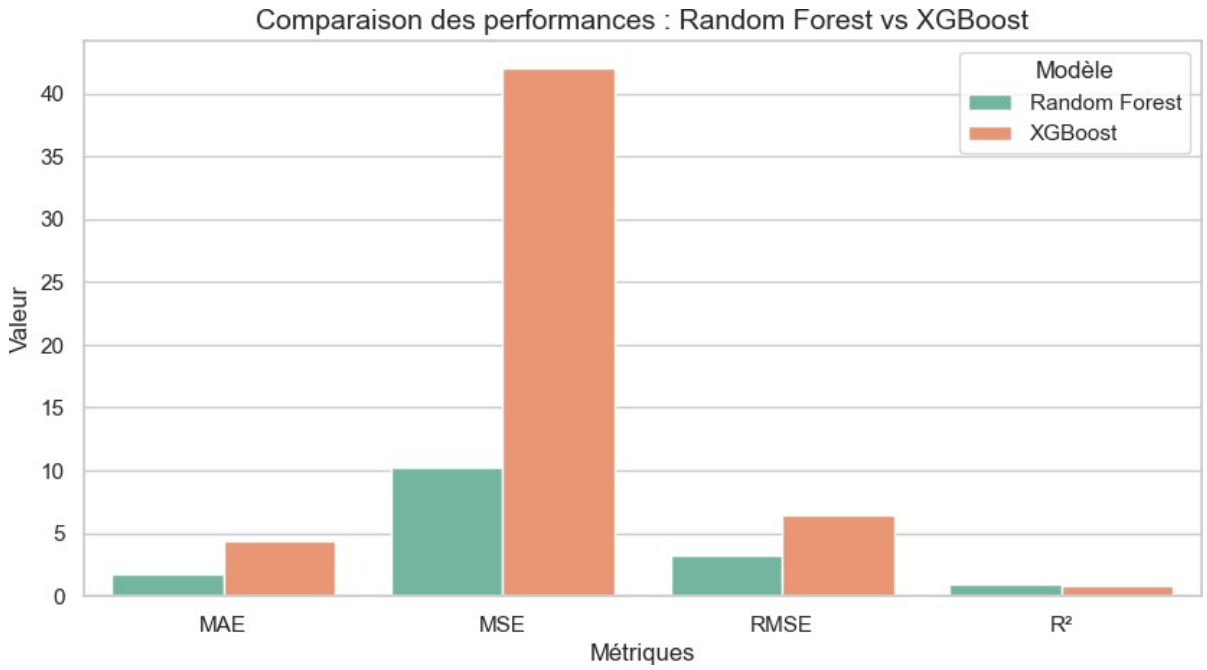
Random Forest est plus concentré et robuste.
XGBoost est plus dispersé, il essaie d'aller chercher des détails supplémentaires mais sans grand gain (ce qui explique ses erreurs plus larges dans les graphiques précédents).

Conclusion finale sur la comparaison Random Forest vs XGBoost

```
# Résultats
results = {
    "Modèle": ["Random Forest", "XGBoost"],
    "MAE": [1.79, 4.39],
    "MSE": [10.18, 42.09],
    "RMSE": [3.19, 6.49],
    "R²": [0.9613, 0.8400]
}

# Création DataFrame
df_results = pd.DataFrame(results)

# --- Affichage tableau stylisé ---
styled_table = (
    df_results.style
    .background_gradient(cmap="Blues", subset=["MAE","MSE","RMSE"]) # dégradé sur les erreurs
    .background_gradient(cmap="Greens",  subset=["R²"])              # dégradé vert pour R²
    .format(precision=3)                                            # 3 décimales
    .set_properties(**{"text-align":  "center"})                    # centrage
    .set_table_styles([dict(selector="th",  props=[("text-align",  "center")])])
)
display(styled_table)

# --- Graphique comparatif ---
plt.figure(figsize=(10,5))
df_plot = df_results.melt(id_vars="Modèle", var_name="Métrique", value_name="Valeur")

sns.barplot(
    data=df_plot,
    x="Métrique",
    y="Valeur",
    hue="Modèle",
    palette="Set2"
```

```
)

plt.title("Comparaison des performances : Random Forest vs XGBoost", fontsize=14) plt.ylabel("Valeur")
plt.xlabel("Métriques")
plt.legend(title="Modèle")
plt.show()
```

	Modèle	MAE	MSE	RMSE	R^2
0	Random Forest	1.790	10.180	3.190	0.961
1	XGBoost	4.390	42.090	6.490	0.840



Résultats globaux des tests :

- **Erreur absolue moyenne (MAE)** :

RF MAE = 1.79, en moyenne, l'erreur absolue est faible.
XGB MAE = 4.39, erreurs absolues plus du double de RF.
Random Forest fait 2,5 fois moins d'erreur moyenne que XGBoost.

- **Erreur quadratique moyenne (MSE/RMSE)** :

RF MSE = 10.18, variance des erreurs beaucoup plus faible.
XGB MSE = 42.09, erreurs bien plus dispersées. **RF RMSE** = 3.19, les grosses erreurs sont limitées. **XGB RMSE** = 6.49, grosses erreurs présentes.
Les erreurs de Random Forest sont beaucoup plus faibles et dispersées.

- **Pouvoir explicatif (R^2)** :

RF R^2 = 0.961, explique 96 % de la variance, excellent ajustement.
XGB R^2 = 0.840, n'explique que 84 % de la variance, beaucoup moins précis. Random Forest capte beaucoup mieux la relation.

- **Sur les moyennes par localisation** :

Les prédictions RF sont très proches des valeurs réelles (écart faible). Les prédictions XGB sont plus éloignées, donc moins précises.

- **Interprétation métier des résultats** :

Les variables, état de la machine et volume traité sont les principaux facteurs expliquant les arrêts, ce qui est cohérent d'un point de vue opérationnel. La localisation des entrepôts et l'identifiant des machines jouent un rôle secondaire, le problème est donc davantage lié à l'utilisation et l'usure des équipements qu'à leur emplacement géographique. Le modèle Random Forest peut déjà être utilisé pour détecter les machines à risque et planifier des interventions préventives, réduisant ainsi les retards logistiques.

- **Interprétation** :

Random Forest surpasse nettement XGBoost sur toutes les métriques (MAE, MSE, RMSE, R^2). Le modèle est plus précis, plus stable et généralise mieux sur nos données.
XGBoost n'est pas catastrophique (R^2=0.84 reste correct), mais il est nettement moins performant que Random Forest ici.
Il se peut que ses hyperparamètres ne soient pas optimisés (learning_rate, max_depth, n_estimators, ...).

Visualisation par ville : sur la plupart des localisations, RF colle beaucoup mieux aux valeurs réelles que XGB.

- **Recommandations** :

Choisir Random Forest, car il donne des résultats fiables et précis.
Optimiser XGBoost avec un GridSearchCV pour ajuster les paramètres (ex : n_estimators (nombre total d'arbres), max_depth (profondeur des arbres) ...)