**Machine learning canvas**

**Table des matières**

[INTRODUCTION 2](#_Toc195465611)

[PREDICTION TASK 2](#_Toc195465612)

[DECISIONS 3](#_Toc195465613)

[VALUE PROPOSITION 5](#_Toc195465614)

[DATA COLLECTION 7](#_Toc195465615)

[DATA SOURCES 9](#_Toc195465616)

[IMPACT SIMULATION 10](#_Toc195465617)

[MAKING PREDICTIONS 11](#_Toc195465618)

[API DETAILS & MLFLOW INTEGRATION 15](#_Toc195465619)

[PIPELINE DE PREDICTIONS 16](#_Toc195465620)

[BUILDING MODELS 17](#_Toc195465621)

[FEATURES 21](#_Toc195465622)

[MONITORING 23](#_Toc195465623)

[DEPLOIEMENT & STRATEGIES DE MISE EN PRODUCTION 26](#_Toc195465624)

# **INTRODUCTION**

La prédiction météorologique représente un **défi complexe de classification binaire** où **l'objectif est de prédire la probabilité de précipitations pour le lendemain avec une précision maximale.**

**Défis Méthodologiques**

* Haute variabilité des données météorologiques
* Interactions non-linéaires entre variables
* Nécessité de modèles robustes et adaptatifs

**Approche Algorithmique**

Pour ce projet de prédiction météo du lendemain, plusieurs algorithmes et outils de Machine Learning pourraient être efficaces, aussi certains d’entres eux ont été envisagés :

1. **Random Forest** : basé sur des arbres de décision, cet algorithme fonctionne bien pour des données avec des variables multiples (température, humidité, précipitations, vent, etc.).
2. **Logistic regression** : plus adapté aux données dont la variable à expliquer est binaire (1 : il va pleuvoir ; 2 : il ne va pas pleuvoir).
3. **Gradient Boosting Machines (GBM)** : inclut des modèles comme XGBoost, LightGBM,….. Ils sont performants sur des données tabulaires.
4. **Réseaux de neurones artificiels (ANN)** : utiles pour détecter des modèles complexes dans des données météorologiques historiques
5. **LSTM (Long Short-Term Memory)** : Conçu pour prédire des séquences temporelles, ce qui peut être un bon choix pour des données météorologiques.

**Dans l'implémentation actuelle**, seul le modèle **Random Forest** a été développé pour notre projet.

Les rubriques suivantes ont pour **objectif de** **décrire l'architecture et le fonctionnement de notre système de prédiction météorologique.**

# **PREDICTION TASK**

**Objectif :** Notre projet vise à prédire la présence de pluie pour le lendemain (RainTomorrow) à partir de données météorologiques actuelles. **Il s'agit d'une classification binaire avec deux résultats possibles** : "Yes" (pluie) ou "No" (pas de pluie).

**Entité cible :** Les prédictions seront effectuées pour des zones géographiques spécifiques identifiées par la variable "Location".

**Horizon temporel :** Nos prédictions portent sur un horizon de 24 heures, ce qui signifie que nous devrons attendre un jour pour évaluer la précision de nos prévisions.

1. **Type de tâche** : classification binaire ; on pourra prédire si ‘’RainTomorrow’’ sera "Yes" ou "No" (pluie ou pas).
2. **Entité sur laquelle les prédictions seront faites** : prédiction est faite sur la météo (spécifiquement la pluie) pour le lendemain dans une certaine "location" (région ou ville).
3. **Résultats possibles** : les résultats sont binaires "Yes (pluie)" ou "No (pas de pluie)" pour la variable ‘’RainTomorrow’’.
4. **Temps d'attente avant l'observation** : 24 heures

# **DECISIONS**

**1- Préparation des données :**

**► Nettoyage des données**

* Gestion des valeurs manquantes, conversion des variables catégorielles en variables numériques par encodage, et gestion des valeurs aberrantes.

**Gestion des valeurs manquantes**

* + Techniques utilisées :
    - Imputation par mode pour les variables catégorielles
    - Concernant les variables numériques, les lignes contenant des valeurs manquantes dans les colonnes critiques (notamment ‘’RainToday’’ et ‘’RainTomorrow’’) sont supprimées.
  + Objectif : Minimiser la perte d'information

**Encodage des Variables**

* + Label Encoding pour les variables catégorielles
  + Transformation binaire : 'Yes/No' → 1/0

df['RainTomorrow'] = df['RainTomorrow'].map({'Yes': 1, 'No': 0})

df['RainToday'] = df['RainToday'].map({'Yes': 1, 'No': 0})

lencoders = {}

for col in df.select\_dtypes(include=['object']).columns:

lencoders[col] = LabelEncoder()

df[col] = lencoders[col].fit\_transform(df[col])

**Standardisation des données**

* Les données numériques sont standardisées à l'aide de StandardScaler pour améliorer les performances du modèle

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

**Gestion des Valeurs Aberrantes**

* Méthode IQR (Interquartile Range)
* Calcul des seuils : Q1 - 1.5*IQR et Q3 + 1.5*IQR
* Remplacement des outliers par les valeurs seuils

def replace\_with\_thresholds(dataframe, column):

low\_limit, up\_limit = outlier\_thresholds(dataframe, column)

dataframe.loc[(dataframe[column] < low\_limit), column] = low\_limit

dataframe.loc[(dataframe[column] > up\_limit), column] = up\_limit

**► Sélection des features**

* Utilisation de variables pertinentes pour la prédiction, comme la température, l'humidité, la pression, etc. du jour actuel pour prédire ‘’RainTomorrow’’.
* On peut aussi inclure les observations météorologiques des jours précédents, car elles peuvent aider à améliorer les prévisions.

**Critères de Sélection**

* Corrélation avec la variable cible
* Importance prédictive
* Réduction de la dimensionnalité

**Features Supprimées**

* 'Date'
* 'Temp3pm'
* 'Pressure9am'
* 'Temp9am'
* 'Rainfall'

**2- Modélisation**

**► Choix du modèle :**

* Le modèle que nous avons retenu est un ‘’Random Forest Classifier’’ pour les avantages suivants :
* Gestion efficace des données multidimensionnelles (température, humidité, vent, etc.).
* Robustesse face aux valeurs aberrantes.
* Capacités de prédiction probabiliste.

**► Paramétrage du Modèle ‘’Random Forest Classifier’’**

* Hyperparamètres optimisés, permettant d’obtenir un niveau de performance acceptable, dans le cadre de notre projet MLOps :

params =

{

"n\_estimators": 10, # Nombre d'arbres

"max\_depth": 10, # Profondeur maximale des arbres

"random\_state": 42 # Reproductibilité

}

**► Stratégie de Validation**

* Split stratifié : 80% entraînement, 20% test
* Préservation de la distribution des classes

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42, stratify=data['RainTomorrow'])

**► Entraînement du modèle**

Le modèle est entraîné sur les données historiques disponibles. Il apprend les relations entre les différentes caractéristiques météorologiques (température, humidité, vitesse du vent, etc.) et la cible « RainTomorrow ».

**► Stockage du modèle**

Le modèle entraîné est sauvegardé au format pickle pour être utilisé en production :

with open("../model/rfc.pkl", "wb") as file:

pickle.dump(rfc, file)

**3. Prédictions**

**► Prédiction binaire**

* Le modèle retourne une valeur de probabilité (par exemple, 0.8) qui indique la probabilité que la pluie survienne le lendemain. Cela peut être converti en une prédiction binaire.
* Si la probabilité est supérieure à un certain seuil (souvent 0.5), la prévision sera Yes (pluie).
* Si elle est inférieure, la prévision sera No (pas de pluie).

**► Seuil ajustable**

* Le seuil pour transformer la probabilité en prédiction binaire peut être ajusté selon la sensibilité désirée du modèle (plus ou moins conservateur pour prédire la pluie).

**4. Transformation en valeur finale pour l'utilisation**

**► Affichage des prévisions :** L'application renvoie à l'utilisateur une information claire et interprétable

**► Paramètres de l'application**

* Interface utilisateur (UI) : **avec streamlit** ou une application mobile
* Actualisation régulière
* Modèle dynamique : réentraînement périodique avec de nouvelles données
* Personnalisation des notifications : notification sur une app ou e-mail

**► Paramètres spécifiques du processus :**

* Modèle de machine Learning : choix d'un modèle et de ses hyperparamètres
* Seuil de décision : probabilité à partir de laquelle on détermine s'il va pleuvoir
* Métrique d'évaluation : accuracy, précision, rappel et AOC-ROC
* Fréquence de mise à jour des données : chaque jour ou plusieurs fois par jour en fonction des besoins de précision

# **VALUE PROPOSITION**

1. **Qui est l’utilisateur final ?**

* Hôpital ou service d’un hôpital :
  + Gestionnaire d’évènement en plein air : besoin de prévoir les conditions météorologiques pour ajuster leurs événements.
  + Grand public : pour sortir avec un parapluie, planifier des voyages.

1. **Objectif des utilisateurs**

* Prédire les conditions météorologiques du lendemain (prédiction de pluie).
* Prendre des décisions basées sur ces prévisions (transport, prévoir des vêtements appropriés, prévoir des solution de repli, …..).

1. **Bénéfices des utilisateurs vis-à-vis du système**

* Précision accrue des prévisions : fournir des précisions plus précises que les approches traditionnelles.
* Alerte en temps réel :faciliter la prise de décision grâce à des alertes sur des pluies prévues ou pas, sur une interface adaptée.
* Personnalisation des prévisions : obtenir des prévisions specifique a une région (en sélectionnant ‘’location’’ dans les données.

1. **Flux de travail / Interface**

**a) Flux de travail**

**► Chargement des données météo :** données météorologiques fournies dans le fichier pour diverses localisations en Australie.

**► Prétraitement des données :** nettoyage et transformation des données brutes, encodage et normalisation.

**► Sélection des caractéristiques pertinentes :** des variables comme la température, l'humidité, la pression atmosphérique, et la pluie du jour actuel sont extraites pour prédire si la pluie va survenir le lendemain

**► Prédiction via un modèle de machine learning** : le modèle de classification utilise les données prétraitées pour prédire la probabilité de pluie le lendemain ‘’**RainTomorrow**’’.

**► Résultats et interprétation** : Le système renvoie une prédiction binaire (pluie ou pas de pluie) accompagnée d'une probabilité (80% de chance qu'il pleuve)

**► Retour à l'utilisateur** : Les prévisions sont ensuite transmises à l'utilisateur via une interface (application web ou mobile)

**► Mise à jour continue des prévisions** : L'application actualise les prévisions chaque jour avec de nouvelles données météorologiques et les utilisateurs peuvent consulter les prévisions à tout moment.

**b) Interface proposée**

**► Interface utilisateur**

* Application web : **Utilisation de Streamlit.**
* Entrée : Région(location), date (automatique pour le lendemain).
* Sortie : Prévision de pluie (oui/non), probabilité de pluie (%), prévision détaillée pour d'autres paramètres météorologiques (température, humidité, etc.).

**► Notification et alertes personnalisées** : notification par e-mail ou SMS en cas de prévision de pluie (ex : il y a 70% de chance qu'il pleuve demain, veuillez ....).

**► Tableau de bord analytique**:

* Tableau de bord : pour une analyse approfondie des données passées et de prévisions futures
* Graphiques : Afficher les tendances de la météo pour plusieurs jours, avec des courbes montrant l'évolution des probabilités de pluie.
* Filtres : L'utilisateur peut filtrer par localisation ou ajuster la plage temporelle (par exemple, prévisions sur 3 jours).

**c) Exemple de flux utilisateur**

* Sélection de la région : l'utilisateur sélectionne une ville comme *Sydney* dans le menu déroulant.
* Consultation des prévisions : le système affiche *=>* "Pluie prévue demain avec une probabilité de 75 %".
* Prise de décision : L'utilisateur peut alors décider de reporter ou de maintenir une activité prévue pour le lendemain, en fonction de la prévision.
* Notification (optionnelle) : l'utilisateur peut configurer une alerte pour recevoir une notification automatique si la probabilité dépasse un certain seuil.

# **DATA COLLECTION**

1. **Stratégie de ‘’Train’’ initial**

**a) Préparation des données**

* Nettoyage des données historiques : traiter les valeurs manquantes de certaines colonnes par correction ou imputation pour éviter les biais dans l'entraînement.
* Sélection des caractéristiques (features) : conservation des variables ayant un impact direct sur les précipitations (température, humidité, direction du vent, pression, etc.)

**b) Division du jeu de données : division effectuée de façon chronologique**

* 80 % des données pour l'entraînement initial du modèle (données anciennes)
* 20 % des données pour tester les performances du modèle (données récentes)

**c) Sélection et entraînement du modèle**

* Choix de l'algorithme : Actuellement, seul le modèle Random Forest est implémenté pour prédire la variable cible ‘’RainTomorrow’’. Les autres algorithmes (Logistic Regression, GBM, ANN, LSTM) pourraient être testés ultérieurement.
* Hyperparamétrage : effectuer une recherche d'hyperparamètres (par exemple avec une grille de recherche ou Random Search) pour trouver la configuration optimale. Ensuite, mesurer les performances avec des métriques telles que l'accuracy, le rappel, la précision, F1-Score, ROC AUC et PR AUC, pour évaluer la capacité du modèle à prédire correctement la pluie ou l'absence de pluie.

**d) Validation croisée**

Une validation croisée avec la méthode de K-fold cross-validation sur le jeu de données d'entraînement permettra d'évaluer la robustesse du modèle.

**2. Mise à jour continue**

**a) Taux de collecte des données**

* Source de données: Source de données météorologiques en temps réel : via des API comme OpenWeatherMap, BOM - Bureau of Meteorology en Australie. Les données peuvent être collectées quotidiennement ou plusieurs fois par jour (horaire).
* Fréquence de mise à jour : mise à jour quotidienne. Chaque jour, les nouvelles données collectées pour la journée seront ajoutées à l'ensemble de données.

**b) Rétention sur les entités de production :**

* Suivi des entités (villes ou localisations): la variable ‘’location’’ qui représente les différentes localisations en Australie, lesquelles localisations doivent être surveillées dans la phase de production
* Rétention des entités : conserver les données pour chaque localisation dans le système afin de suivre les tendances historiques propres à chaque entité géographique.
* Enrichissement des données: Si de nouvelles localisations apparaissent ou deviennent importantes pour les prévisions, elles devront être ajoutées dans le processus de mise à jour pour maintenir la précision du modèle sur une échelle géographique plus large.

**c) Mise à jour du modèle**

Le modèle est mis à jour à intervalles réguliers selon le schéma de pipeline suivant :

* Training Pipeline : exécution hebdomadaire (chaque lundi à 00:00)
* Prediction Pipeline : exécution quotidienne (chaque jour à 06:00)

Cette approche assure que le modèle reste performant à long terme en intégrant régulièrement les nouvelles données météorologiques.

**3. Coûts et contraintes pour observer les résultats**

**a) Coût de collecte des données**

* API météo : possibilité d'avoir des coûts associés à l'accès aux données via certaines APIs.
* Infrastructure : Stocker et traiter les données météorologiques en continu nécessite une infrastructure de serveur ou de cloud (AWS, Google Cloud, etc.), ce qui peut entraîner des coûts de stockage et de traitement.

**b) Coûts liés à la mise à jour du modèle**

* Temps de calcul : Réentraîner un modèle régulièrement peut exiger beaucoup de ressources de calcul, surtout avec de grandes quantités de données. Il est important de prévoir des coûts liés à l'infrastructure nécessaire pour maintenir et exécuter ces modèles.
* Déploiement en production : Si le modèle est déployé dans une application utilisée par de nombreux utilisateurs, il faudra également prévoir des coûts liés à l'hébergement de l'application, ainsi qu'à la maintenance et à la mise à jour du modèle en production.

**c) Contraintes d'observation des résultats**

* Temps d'observation : Pour évaluer la performance des prédictions, il faut attendre au moins 24 heures (ou plus) après l'émission des prévisions pour comparer les résultats (prédictions de pluie vs observations réelles). Cela impose un délai avant de pouvoir ajuster ou améliorer les performances du modèle.
* Contraintes de validation: Mettre en place un système de suivi des performances nécessite la collecte des résultats réels pour valider la précision du modèle.

# **DATA SOURCES**

1. **APIs pour les données météorologiques**

Pour obtenirdes données en temps réel et historiques sur les différentes entités (villes, régions, etc.), ainsi que les résultats observés (précipitations, température, vent, etc.).

* OpenWeatherMap API : [OpenWeatherMap API](https://openweathermap.org/api)
* WeatherAPI (anciennement Weatherstack) [WeatherAPI](https://www.weatherapi.com/docs/)
* Bureau of Meteorology (BOM) – Australie [Bureau of Meteorology API](http://www.bom.gov.au/data/)
* NOAA (National Oceanic and Atmospheric Administration) [NOAA API](<https://www.ncdc.noaa.gov/cdo-web/webservices/v2>)

1. **Sites web pour l'exploration des données météo**

* World Meteorological Organization (WMO) [WMO](<https://public.wmo.int/en>)
* Meteostat [Meteostat](<https://meteostat.net/en/>)

1. **Bases de données météorologiques (tables SQL ou autres formats)**

* Global Historical Climatology Network (GHCN): [GHCN](<https://www.ncei.noaa.gov/products/land-based-station/global-historical-climatology-network>)
* NOAA Climate Data Online (CDO): [NOAA CDO](<https://www.ncdc.noaa.gov/cdo-web/>)

1. **Méthodes pour intégrer les données dans un système de machine learning**

* **Bases de données SQL** : pour structurer les données météorologiques en utilisant des bases de données relationnelles, comme **MySQL**, **PostgreSQL**, ou **SQLite.**
* **Accès via API :** en intégrant directement les **APIs météorologiques** mentionnées plus haut dans un pipeline de données, nous pouvons automatiser la collecte de données météo en temps réel. Également, des **scripts Python** peuvent être utilisés pour appeler les APIs quotidiennement et stocker les données dans une base de données ou système de fichier (par exemple, CSV, Parquet)
* **Pipelines de données** : Outils comme **Apache Airflow, ETL pipelines**

# **IMPACT SIMULATION**

1. **Déploiement des modèles**

**Modèle pré-entraîné**

* Utiliser les données historique du fichier de base téléchargé pour entrainer initialement le modèle.
* Déployer le modèle dans une plateforme de production, où il recevra de nouvelles données météorologiques (via des APIs comme OpenWeatherMap ou BOM) pour générer des prédictions.

**Infrastructure de déploiement**

* + Plateformes cloud comme AWS SageMaker, Google Cloud AI, ou Azure ML permettent de déployer des modèles en production.
  + Via API REST personnalisée qui reçoit les données météo actuelles et renvoie une prédiction sur la base du modèle déployé.

**Interface utilisateur**

* + Intégrer le modèle dans une application web ou mobile où l'utilisateur final peut obtenir des prévisions météorologiques.

1. **Données d’essaies pour évaluer les performances**

**Données de validation**

* Evaluation du modèle sur des données de test

**Métriques de performance**

* Précision (accuracy) : proportion de prédictions correctes.
* Rappel : prédictions vraies positives sur toutes les occurrences de pluie.
* Précision : % de prédictions correctes parmi celles prédites comme étant de la pluie.
* F1-score : Une mesure qui combine précision et rappel pour évaluer globalement les performances du modèle.
* Courbe ROC et AUC : Pour évaluer la capacité de distinction entre pluie et non-pluie
* Matrice de confusion : Pour visualiser les vrais positifs, faux positifs, vrais négatifs et faux négatifs

**Exemple de Métriques**

metrics\_rfc =

{"Accuracy": 0.85, # Précision globale

"Precision": 0.82, # Précision des prédictions positives

"Recall": 0.78, # Capture des événements réels

"F1-Score": 0.80, # Équilibre précision-recall

"ROC AUC": 0.88, # Capacité de discrimination

"PR AUC": 0.85 # Performance sur données déséquilibrées

}

1. **Valeurs de coût/gain pour les décisions correctes ou incorrectes**

**Prédictions correctes**

* Gain lié à une bonne prédiction de pluie (vrai positif)

🡺Valeur estimée : Gain modéré à élevé selon le contexte

* Gain lié à une bonne prédiction de l’absence de pluie (vrai négatif)

🡺Valeur estimée : Gain modéré.

**Prédictions incorrectes**

* Coût lié à une fausse alerte de pluie (faux positif) : annulation d’évènement

🡺Valeur estimée: Coût faible à modéré selon le contexte

* Coût lié à une absence de prévision de pluie (faux négatif) : dommages matériels

🡺Valeur estimée : Coût potentiellement élevé

1. **Contrainte d’équité**

**Equité géographique**

* Problème :localisations sous représentées dans les données historiques, ce qui pourrait entraîner des prévisions moins précises pour ces régions.
* Solution : S’assurer que les données couvrent de manière équitable différentes régions géographiques (urbaines, rurales, côtières, etc.).

**Equité sociale**

* Problème : Les prévisions météorologiques peuvent avoir des impacts différents selon les groupes sociaux.
* Solution : Concevoir le modèle et les stratégies d’intervention en tenant compte de ces disparités pour s'assurer que certaines communautés ne soient pas plus affectées par les erreurs de prédiction.

**Equité temporelle**

* Problème : Les prévisions pourraient être plus fiables à certaines périodes de l'année que d'autres (par exemple, saison sèche vs saison des pluies), introduisant un biais temporel
* Solution : Analyser la performance du modèle à différentes périodes de l'année et ajuster les modèles pour éviter de privilégier certaines périodes.

# **MAKING PREDICTIONS**

1. **Quand faisons nous des prédictions en temps réel ou pas batch ?**
2. **Prédictions en temps réel**

**Cas** **d’usage** :

* Un système automatisé d’alerte météorologiques reçues
* Un service qui alerte immédiatement sur des conditions critiques (orage, pluie imminente)

**Données** :

* Mise à jour des données en continu à partir de sources telles que des APIs météorologiques

**Fréquence** :

* Les prédictions peuvent être générées chaque minute ou chaque heure, selon la disponibilité des données et les besoins des utilisateurs

**Exemple** :

* Un modèle de prédiction qui traite les données d'une API météo chaque fois qu'une nouvelle observation est reçue et donne immédiatement la prédiction de pluie pour le lendemain.

**Contraintes** :

* Les calculs doivent être rapides, ce qui nécessite une infrastructure capable de traiter les données en quelques secondes ou minutes

1. **Prédictions par batches (lots)**

**Cas d’usage :**

* Les applications où les prédictions n'ont pas besoin d'être mises à jour en temps réel, mais plutôt à des intervalles réguliers ( prévisions journalières, hebdomadaires, ou mensuelles)
* Utilisé lorsque des ensembles de données plus volumineux sont traités en une seule fois, permettant une analyse approfondie des tendances météorologiques

**Données :**

* Les données météorologiques historiques ou agrégées sont utilisées pour prédire les conditions futures.

**Fréquence :**

* Les prédictions sont faites à des intervalles prédéfinis, par exemple, une fois par jour, une fois par semaine, ou selon un programme planifié

**Exemple :**

* Un système de prévision météo pour les agriculteurs qui génère des prédictions hebdomadaires à partir de données météorologiques collectées au cours des sept derniers jours.

**Contraintes :**

* Moins de contraintes sur la rapidité de traitement par rapport aux systèmes en temps réel, mais il faut tout de même maintenir un bon équilibre entre les performances du modèle et le temps de calcul disponible

1. **Temps disponible pour le feature engineering, la prédiction et le post-traitement**
2. **Prédictions en temps réel**

**Feature Engineering** :

* La transformation des données en caractéristiques utiles (features) doit être rapide. Les données reçues via des API météo doivent être prétraitées instantanément.

**🡺Temps estimé** : Quelques millisecondes à secondes

**Prédiction**:

* L'algorithme de machine learning doit être capable de générer une prédiction en quelques millisecondes à secondes.

**🡺temps estmé**: Millisecondes à secondes

**Post-traitement** :

* Cela inclut l'interprétation des résultats, la génération de visualisation ou l'envoi d'alertes et se faire rapidement

**🡺Temps estimé**: Millisecondes à secondes

1. **Prédictions par batches**

**Feature Engineering :**

* Les données historiques peuvent être agrégées et traitées en amont, donc le processus peut être plus lent et plus complexe

**🡺Temps estimé :** Quelques minutes à heures selon la taille du batch et la complexité du pipeline de données

**Prédiction**:

* le modèle est appliqué sur un ensemble de données plus important et peut être exécuté en parallèle ou distribué sur plusieurs machines

**🡺Temps estimé :** Minutes à heures selon la taille des données et le modèle

**Post-traitement :**

* Les résultats doivent être agrégés, analysés et interprétés pour des rapports plus approfondis. Des visualisations ou des rapports peuvent être générés automatiquement.

**🡺Temps estimé :** Minutes à heures (génération de rapports, stockage des résultats)

1. **Objectif de calcul**

Il dépend des contraintes de l’application et des ressources disponibles.

1. **Prédictions en temps réel**

**Objectif principale** :

* Faible latence et haute disponibilité
* Le système doit être capable de traiter les requêtes en quelques secondes pour permettre à l'utilisateur final de recevoir des prédictions immédiatement

**Infrastructures** :

* Le système doit être capable de traiter les requêtes en quelques secondes pour permettre à l'utilisateur final (ou à un système automatisé) de recevoir des prédictions immédiatement
* Optimisation du code pour réduire la charge de calcul

1. **Prédictions par batches**

**Objectif principal** :

* Traitement de grandes quantités de données pour obtenir des résultats fiables et précis sur une période donnée
* Pas de contraintes strictes sur la latence, mais l'efficacité du calcul est importante pour réduire les coûts et éviter les goulots d'étranglement dans le traitement des données

**Infrastructures** :

* Utiliser des solutions comme Apache Spark pour le traitement distribué des données, ou des instances cloud avec des capacités de calcul parallélisées
* Les batchs peuvent être traités la nuit ou pendant les périodes de faible activité

**Prédictions générées via une API RESTful construite avec FastAPI.**

Détails sur le fonctionnement de l'API et l'orchestration des prédictions.

**Types de prédictions :**

* + Prédiction automatique : basée sur les données du jour via une requête GET à **l'endpoint /predict.**
  + Prédiction manuelle : via une requête POST avec un fichier CSV contenant les données. Cette fonctionnalité n'est pas visible dans le code fourni et doit être vérifiée.

**Orchestration :** Utilisation de MLflow pour le suivi des runs de prédiction, imbriqués dans un run de déploiement de modèle parent.

**Logique de l'API :**

* + Récupération du run de déploiement actuel (et configuration de MLflow).
  + Démarrage d'un run MLflow imbriqué pour la prédiction spécifique.
  + Exécution de la fonction predict\_weather() utilisant le modèle MLflow.
  + Retour des résultats (prédiction, probabilité, identifiants de run MLflow).

**Exemple de réponse (GET /predict):**

{

"status": "success",

"message": "Daily prediction successfully completed",

"run\_id": "...",

"deployment\_run\_id": "...",

"model\_version": "...",

"prediction": "No",

"probability": 0.15

}

**Orchestration des prédictions via Airflow *(Cf script prediction\_dag.py)* :**

* **Orchestration des prédictions** : Le fichier prediction\_dag.py met en place un DAG (Directed Acyclic Graph) Airflow pour automatiser le processus de prédiction.
* **Étapes principales du DAG** :
  + Extraction des données brutes depuis une source définie.
  + Préparation des données pour le modèle (nettoyage, transformations).
  + Chargement du modèle entraîné depuis un chemin spécifié.
  + Génération des prédictions et stockage des résultats.
* **Exemple de définition d'un DAG dans le fichier :**

with DAG(

dag\_id="prediction\_pipeline",

schedule\_interval="@daily",

start\_date=datetime(2023, 1, 1),

catchup=False,

) as dag:

* **Planification** : Le DAG est configuré pour s'exécuter quotidiennement (@daily), garantissant une mise à jour régulière des prédictions.

# **API DETAILS & MLFLOW INTEGRATION**

**Descriptif :**

* Utilisation de MLflow pour le monitoring et le suivi des performances (via predict\_api.py)
* Logging (via predict\_api.py)
* Variables d’environnement (via predict\_api.py)
* Exposition d’une API

**Framework API :** FastAPI

**Endpoints :**

* + **/predict (GET)** : Effectue une prédiction automatique basée sur les données actuelles.

**MLflow :**

* **Suivi des runs** : Chaque requête de prédiction crée un run MLflow imbriqué dans le run de déploiement du modèle.
* **Tags MLflow** : Utilisation de tags pour identifier le type de pipeline, la date de prédiction, l'endpoint utilisé et la version du modèle.
* **Gestion des runs** : Utilisation de mlflow.start\_run() et mlflow.end\_run() pour gérer les runs MLflow et assurer un suivi correct.
* **Fonction get\_deployment\_run()** issue de utils.mlflow\_run\_manager : Récupère les informations du run de déploiement du modèle courant depuis MLflow.

**Gestion des erreurs :**

* L'API capture les exceptions et retourne des erreurs HTTP 500 avec des détails.

# **PIPELINE DE PREDICTIONS**

Cette section détaille précisément comment les prédictions sont orchestrées ainsi que la structure et les composants du pipeline **(Cf script prediction\_dag.py).**

Le pipeline de prédiction est orchestré via **Airflow** et suit les étapes suivantes :

1. **Extraction des données** :
   * Source : Les données brutes sont extraites depuis un chemin ou une base de données spécifiée.
   * Exemple dans le script :

extract\_data = PythonOperator(

task\_id="extract\_data",

python\_callable=extract\_data\_function,

)

1. **Préparation des données** :
   * Nettoyage et transformation des données pour les rendre exploitables par le modèle.
   * Exemple dans le script :

preprocess\_data = PythonOperator(

task\_id="preprocess\_data", python\_callable=preprocess\_function,

)

1. **Chargement du modèle** :
   * Le modèle est chargé depuis un chemin défini (par exemple, un répertoire local ou un stockage distant).
   * Exemple dans le script :

load\_model = PythonOperator(

task\_id="load\_model",

python\_callable=load\_model\_function,

)

1. **Génération des prédictions** :
   * Les prédictions sont effectuées sur les données préparées.
   * Exemple dans le script :

predict = PythonOperator(

task\_id="predict",

python\_callable=predict\_function,

)

1. **Stockage des résultats** :

* Les résultats sont sauvegardés dans un emplacement spécifié (base de données ou fichier).
* Exemple dans le script : save\_predictions = PythonOperator(

task\_id="save\_predictions",

python\_callable=save\_predictions\_function,

)

1. **Dépendances entre tâches** :
   * Les tâches sont exécutées séquentiellement selon leur dépendance logique.
   * Exemple dans le script :

extract\_data >> preprocess\_data >> load\_model >> predict >> save\_predictions

# **BUILDING MODELS**

Diagramme des DAGs incluant le workflow des données :

Une image contenant texte, capture d’écran, diagramme, logiciel

Le contenu généré par l’IA peut être incorrect.

*Diagramme des DAGs montrant le workflow des données avec 3 pipelines :*

*Splitting phase, Training Pipeline (hebdomadaire), et Prediction Pipeline (quotidien)*

1. **Nombre de modèles de production nécessaire**

**Option1 (un seul modèle global) :** Ce modèle capture les tendances globales des conditions météorologiques de tout le pays, en s’appuyant sur les variables telles que la température, les précipitations, la vitesse du vent, etc.

Un modèle unique (RandomForestClassifier) a été utilisé, sans segmentation par région.

* **Avantages :** Moins de complexité en termes de gestion des modèles et des coûts informatiques réduits. L'approche actuelle dans train\_model.py utilise cette stratégie.
* **Inconvénients :** Le modèle pourrait manquer de précision dans certaines régions spécifiques ou pour certaines saisons.

**Option 2 (plusieurs modèles régionaux) :** Nous pourrions prendreen compte de la diversité climatique.

Par exemple :

* Modèle 1 : Régions Côtières (Est, Ouest, Sud)
* Modèle 2 : Régions intérieurs arides
* Modèle 3 : Régions tropicales du nord
* **Avantage** : Améliore la précision en s’adaptant aux variations climatiques régionales.
* **Inconvénient** : Nécessite une infrastructure plus complexe pour gérer plusieurs modèles et entraîner chacun d'eux séparément.

1. **Quand procéder à la mise à jour**
2. **Mises à jours périodique**

* **Mise à jour hebdomadaire** : **Cf Diagramme DAG ==** Le pipeline d'entraînement (DAG 2) est exécuté de façon hebdomadaire (chaque lundi à 00:00). Cette fréquence permet de maintenir le modèle à jour avec les dernières tendances météorologiques.
* **Mise à jour saisonnières** : Si les modèles sont saisonniers, une mise à jour à chaque changement de saison (tous les 3 mois) peut être efficace pour capturer les tendances de la nouvelle saison.

1. **Mises à jour basées sur la performance**

* Les modèles doivent également être mis à jour lorsque leurs performances diminuent. En surveillant des métriques telles que la précision, le modèle peut être réentraîné lorsqu'une dégradation de la performance est détectée

🡺**Fréquence** : **Cf Diagramme DAG ==** Le système MLflow indiqué dans notre diagramme permet de suivre les métriques du modèle et peut déclencher un réentraînement si nécessaire.

Une surveillance continue des performances permet de déclencher une mise à jour quand nécessaire, au lieu d'une mise à jour programmée.

1. **Mises à jour événementielles**

Dans le cas de changements météorologiques importants et soudains, une mise à jour immédiate pourrait être nécessaire pour ajuster les prévisions aux nouvelles conditions.

1. **Temps disponible pour cela**
2. **Temps pour la mise en forme des données (feature engineering)**

**Features Engineering** / **Cf script prepare\_data.py :** le processus comprend le nettoyage, l'encodage des variables catégorielles et la gestion des valeurs aberrantes.

🡺**Temps estimé** : Pour un traitement en lot, cela pourrait prendre \*\*quelques heures à une journée\*\*, en fonction de la quantité de données à traiter et de la complexité du pipeline de données.

**Optimisation** : Si le système est en temps réel, le features engineering doit être rapide (quelques secondes)

1. **Temps pour l’entraînement et la mise à jour du modèle**

* **Entraînement du modèle / Cf script train\_model.py**: montre un modèle RandomForestClassifier avec un nombre limité d'arbres (n\_estimators=10), ce qui se traduit par un temps d'entraînement relativement court.
* Pour des mises à jour en batch, les modèles peuvent être réentraînés pendant des périodes où l'utilisation du système est faible (par exemple, la nuit).

**Le DAG 2** s'exécute à minuit (00:00) chaque lundi, ce qui correspond à cette stratégie.

🡺 **Temps estimé :** Avec la configuration actuelle (RandomForest avec 10 arbres, profondeur max de 10), il est estimé à quelques minutes, voire quelques heures selon le volume de données.

1. **Post-traitement et analyse**

* Après l'entraînement, il est nécessaire d'évaluer la performance du modèle, de valider les résultats et de déployer le modèle en production. Le script évalue le modèle avec diverses métriques (accuracy, precision, recall, F1-score, ROC AUC, PR AUC) qui sont stockées dans **metrics\_rfc.**

🡺Temps estimé : quelques minutes pour le calcul des métriques et leur enregistrement dans MLflow **comme indiqué dans le Diagramme des DAGs.**

1. **Infrastructure et Déploiement**

Après avoir décrit la modélisation et les prédictions, cette section explique comment ces éléments sont intégrés dans un système fonctionnel (via Docker, Airflow, MLflow, etc.).

En d’autres termes, cette section est dédiée à l'infrastructure permettant de détailler comment ces modèles et pipelines sont déployés dans un environnement opérationnel.

Détail de l'architecture technique déployée via Docker :

* **Orchestration des services** :
  + Airflow (2.8.1) avec CeleryExecutor pour la gestion des workflows
  + PostgreSQL (v13) pour le stockage des métadonnées d'Airflow et MLflow
  + Redis pour le broker Celery
  + MLflow (v2.21.0rc0) avec backend PostgreSQL et stockage d'artefacts local
  + Application FastAPI sur Uvicorn (**port 8000**)

**Configuration des volumes** :

volumes:

- ./airflow/dags:/opt/airflow/dags

- ./api/data/models:/app/api/data/models

- ./raw\_data/:/app/raw\_data

Persistance des modèles entraînés et datasets via des mounts host-container.

**Tracking des Modèles / Intégration de MLflow :**

environment:

MLFLOW\_TRACKING\_URI=postgresql://${DB\_USER}:${DB\_PASSWORD}@db/${DB\_NAME}

command: >

mlflow server

--backend-store-uri postgresql://${DB\_USER}:${DB\_PASSWORD}@db/${DB\_NAME}

--default-artifact-root http://**mlflow:5000**/api/2.0/mlflow-artifacts/artifacts

Centralisation du tracking des expériences et versioning des modèles via PostgreSQL.

**API et Intégration de Services / Exposition des prédictions :**

* Endpoint REST sur le **port 8000** via Uvicorn
* Variables d'environnement critiques (1) :

MLFLOW\_TRACKING\_URI: http://**mlflow:5000**

MODEL\_PATH: /app/api/data/models

* Workflow de déploiement automatisé via Docker build context :

app:

build:

context: ./api

**Pipeline de Données :**

* Intégration des chemins d'accès containerisés :

INITIAL\_DATA\_PATH: /app/raw\_data/initial\_dataset/weatherAUS.csv

PREPARED\_DATA\_PATH: /app/api/data/prepared\_data

* Orchestration via Airflow avec monitoring santé (1) :

airflow-scheduler:

healthcheck:

test: ["CMD-SHELL", "airflow jobs check --job-type SchedulerJob"]

**Gestion des Environnements / Reproductibilité :**

* Versioning des dépendances via \_PIP\_ADDITIONAL\_REQUIREMENTS (1) :

environment: &airflow-common-env

\_PIP\_ADDITIONAL\_REQUIREMENTS: ${\_PIP\_ADDITIONAL\_REQUIREMENTS:- pymongo}

* Isolation des composants via réseaux Docker
* Configuration centralisée des identifiants DB via variables d'environnement

**Monitoring et Scaling / Partie opérationnelle :**

* Healthchecks automatisés pour PostgreSQL :

healthcheck:

test: ["CMD-SHELL", "pg\_isready -U ${DB\_USER} -d ${DB\_NAME}"]

* Flower (**port 5555**) pour le monitoring des workers Celery
* Logs persistants via volume mount /opt/airflow/logs

# **FEATURES**

1. **Sources de données brutes**

La source principale des données est un fichier CSV (weatherAUS.csv), qui contient des données historiques météorologiques australiennes :

* Stations météorologiques locales
* Données historiques
* Données satellitaires
* Données sur les phénomènes climatiques globaux
* Sources publiques et API

1. **Représentations d'entrées disponibles au moment de la prédiction**
2. **Variables météorologiques instantanées (features en temps réel)**

**Cf script** **prepare\_data.py**, les features suivantes sont utilisées après le nettoyage et la transformation des données :

* Température actuelle
* Vitesse du vent actuelle
* Vitesse du vent actuelle
* Pression atmosphérique
* Humidité
* Couverture nuageuse actuelle
* Direction du vent
* RainToday (présence de pluie aujourd'hui, encodée en 0/1)

Ce sont les caractéristiques disponibles immédiatement au moment de la prédiction.

1. **Données historiques (lag features)**

Les valeurs historiques jouent un rôle clé dans la prédiction des événements météorologiques futurs.

Les principales caractéristiques sont :

* **Dans notre implémentation actuelle**, seules les données du jour même sont utilisées (pas de lag features explicites dans les scripts)
* **Dans une implémentation future**, le pipeline pourrait être amélioré en incluant des variables comme :
  + Température d'hier, température de la semaine dernière, moyenne des températures des 7 derniers jours
  + Précipitations cumulées sur les 3 derniers jours
  + Direction du vent sur les dernières 24h
  + Pression atmosphérique sur la dernière semaine

Ces variables historiques sont dérivées des enregistrements passés et utilisées pour créer des séries temporelles.

1. **Caractéristiques dérivées**

Certaines représentations d'entrée ne proviennent pas directement de sources de données brutes, mais sont dérivées des données existantes par des techniques de feature engineering

**Dans notre implémentation actuelle**, les caractéristiques dérivées incluent:

* Traitement des valeurs aberrantes avec la méthode IQR
* Encodage des variables catégorielles comme 'Location' et 'RainToday'

**Dans une implémentation future**, d'autres caractéristiques dérivées pourraient être ajoutées:

* **Tendances** : Variation de la température sur les dernières heures ou jours
* **Anomalies** : Par exemple, une température anormalement élevée par rapport à la moyenne historique
* **Rolling averages** (moyennes mobiles) : Moyenne glissante de la température sur plusieurs jours

1. **Caractéristiques exogènes (données environnementales externes)**

En plus des données brutes immédiates, d'autres sources de données exogènes pourraient être utilisées, bien que non présentes dans notre implémentation actuelle.

1. **Processus d'extraction et de transformation des données**

Cf script **prepare\_data.py**, le processus comprend:

* **Collection de données brutes**: Chargement du fichier CSV
* **Nettoyage et prétraitement**:
  + Encodage des variables binaires ('Yes'/'No' → 1/0)
  + Remplacement des valeurs manquantes (mode pour catégorielles)
  + Encodage des variables catégorielles avec LabelEncoder
  + Traitement des valeurs aberrantes avec la méthode IQR
* **Feature engineering**:
  + Suppression des colonnes non pertinentes ('Date', 'Temp3pm', 'Pressure9am', 'Temp9am', 'Rainfall')
* **Utilisation des données en temps réel**:
  + Le pipeline de prédiction quotidien **(DAG 3)** s'exécute chaque jour à 06:00 selon le diagramme

1. **Représentation finale au moment de la prédiction**

Lorsque toutes ces données brutes sont transformées, le modèle utilise une représentation d’entrée qui inclut :

* **Variables en temps réel** : Données météorologiques actuelles après prétraitement
* **Données encodées** : Variables catégorielles transformées par LabelEncoder
* **Données normalisées** : D'après train\_model.py, les données sont standardisées avec StandardScaler pour certaines prédictions
* **Données nettoyées des valeurs aberrantes** : Utilisation des seuils IQR pour limiter l'impact des valeurs extrêmes

Ces représentations permettent au modèle de faire des prédictions précises basées à la fois sur l'instantanéité des conditions météorologiques actuelles et sur l'historique des données.

# **MONITORING**

Pour quoi faire ?

* Suivi des performances du modèle via Mlflow
* Détection de dérive de données
* Réentraînement automatique hebdomadaire via DAG 2
* Suivi automatisé via Airflow
* Logs centralisés via Airflow
* Alertes via Airflow

1. **Mesures pour quantifier la création de valeur pour les utilisateurs finaux**
2. **Précision des prévisions météorologiques**

* **Mesure :** Exactitude (%)
* **Méthode / Cf** **train\_model.py** : le modèle est évalué avec plusieurs métriques, notamment l'accuracy, la precision, le recall et le F1-score
* **Objectif :** Atteindre une exactitude élevée (idéalement > 85 %), en particulier pour des paramètres critiques comme les précipitations

1. **Impact sur la prise de décision des utilisateurs finaux**

* **Mesure :** Réduire des erreurs dans la planification
* **Exemple :** Agriculteurs pouvant planifier l'irrigation en fonction des prévisions de pluie, ce qui optimise l'utilisation de l'eau.
* **Méthode :** Réaliser des enquêtes avant/après l'utilisation du système ML pour quantifier la réduction des erreurs de planification

1. **Satisfaction des utilisateurs**

* **Mesure :** Score de satisfaction des utilisateurs (sur 10 ou en pourcentage)
* **Exemple** : Un agriculteur qui évite des pertes de récolte en anticipant des précipitations excessives
* **Objectif** : Améliorer ce score de satisfaction au fil du temps

1. **Réduction des coûts liés aux mauvaises décisions météorologiques**

* **Mesure** : Coût évité
* **Exemple** : Un transporteur économisant sur les coûts de carburant en ajustant ses itinéraires selon les prévisions de vent ou de tempête
* **Objectif**: Quantifier les économies réalisées et montrer la valeur ajoutée du système

1. **Temps de réaction amélioré**

* **Mesure** : Gain de temps de reaction (en % ou en heure)
* **Objectif** : Minimiser le délai entre la prévision et l’action des utilisateurs finaux

1. **Mesures pour quantifier la création de valeur pour les entreprises**

L'objectif est de démontrer l'impact en termes de réduction des coûts, augmentation de la productivité, et optimisation des opérations

1. **Réduction des coûts opérationnels**

* Mesure :coût évité
* Mesurer les \*\*économies\*\* réalisées par les entreprises grâce aux prédictions précises
* Optimisation des itinéraires logistiques pour éviter les conditions météorologiques défavorables

1. **Augmentation de la productivité**

* Mesure : Amélioration de la productivité (%)
* Les prédictions météorologiques permettent aux entreprises d'améliorer la gestion des ressources (personnel, matériel) en fonction des conditions attendues

1. **Temps de calcul et de prédiction optimisé**

* Mesure : Temps de traitement et de prédiction (en secondes ou minutes)
* Mesurer la rapidité avec laquelle le système peut fournir une prévision après réception des données d'entrée (surtout pour les modèles en temps réel)

🡺**Objectif** : Réduire ce délai pour permettre aux utilisateurs de réagir rapidement aux conditions météorologiques changeantes

1. **Stabilité et performance du système**

* Mesure : Temps d’arrêt du système (% de disponibilité)
* Mesurer la disponibilité du système ML en production et sa robustesse face aux charges de travail croissantes.

🡺**Objectif** : Garantir une disponibilité proche de 100 %, surtout lors des périodes météorologiques critiques

1. **Retour sur investissement (ROI)**

* Mesure : ROI (%)
* Le **retour sur investissement** (ROI) est une mesure essentielle pour déterminer si la mise en place du système de ML est financièrement rentable pour l'entreprise

1. **Mesures pour la performance du modèle ML**

**Précision de la prédiction**

**a)** **Précision de la prédiction**

* **Mesure :** Cf train\_model.py, le modèle est évalué avec plusieurs métriques:
* Accuracy: Pourcentage global de prédictions correctes
* Precision: Proportion de vrais positifs parmi les cas prédits positifs
* Recall: Proportion de vrais positifs détectés parmi tous les cas réellement positifs
* F1-score: Moyenne harmonique entre précision et recall
* ROC AUC: Mesure la capacité du modèle à distinguer les classes
* PR AUC: Performance sur données déséquilibrées
* **Objectif :** Réduire au maximum ces erreurs pour que les prédictions soient aussi fiables que possible

**b)** **Robustesse du modèle**

* **Mesure :** Robustesse des prévisions face aux outliers ou aux conditions exceptionnelles
* Le script prepare\_data.py inclut un traitement des valeurs aberrantes, ce qui améliore la robustesse du modèle
* **Objectif :** Vérifier si le modèle reste performant même lors d'événements météorologiques inhabituels ou extrêmes

**c)** **Temps de réentraînement du modèle**

* **Mesure :** Temps de réentraînement (en heures ou minutes)
* **Méthode :** Mesurer le temps d'exécution du DAG 2 (Training Pipeline) qui s'exécute hebdomadairement
* **Objectif** : Minimiser le temps de réentraînement pour s'assurer que le modèle reste à jour en permanence

**d) Impact de la performance sur l'utilisateur final**

* **Mesure :** Impact de la précision des prédictions sur l'utilisateur final (en % ou en satisfaction)
* **Méthode :** Relier la précision des prédictions à la satisfaction des utilisateurs finaux ou aux économies réalisées
* **Exemple :** Un modèle avec une précision accrue de 10% pourrait réduire les coûts de transport d'une entreprise de 5%

**Les mesures pour quantifier la création de valeur** et **mesurer l'impact d'un système de machine learning dans la production** doivent inclure à la fois **des métriques techniques** (précision, erreurs, disponibilité) et **des métriques métier** (réduction des coûts, gain de productivité, satisfaction des utilisateurs).

Ces mesures permettent de **démontrer l'efficacité du système ML** à la fois pour les utilisateurs finaux et pour les entreprises, **tout en garantissant un retour sur investissement positif.**

# **DEPLOIEMENT & STRATEGIES DE MISE EN PRODUCTION**

1. **Conteneurisation**

* **Docker pour environnement reproductible**

Nous utilisons Docker pour assurer la reproductibilité de notre environnement de

déploiement. Docker permet de packager l’application avec toutes ses dépendances,

garantissant ainsi que l’exécution est cohérente quel que soit l’environnement

(développement, test, production).

* **Kubernetes pour orchestration des DAGs (non implémenté à ce jour)**

Nous utiliserons Kebernetes pour orchestrer l’exécution des DAGs (Directed Acyclic

Graphs) de notre pipeline de Machine Learning. Kubernetes pemettra de gérer le

déploiement, la mise à l’échelle et l’ordonnancement des tâches définies dans nos DAGs.

1. **Monitoring Proactif**

* **Suivi des performances du modèle via MLflow**

MLflow est intégré pour suivre les métriques de performance de notre modèle au fil du temps. Ceci nous permet de suivre les métriques spécifiques suivantes : précision, rappel et F1-score. Mlflow nous aide à visualiser et à comparer les performances des différentes versions du modèle.

* **Détection de dérive de données entre les distributions des données d'entraînement et de prédiction**

Décrire notre approche pour détecter la dérive des données :

Préciser les outils que nous utilisons pour comparer les distributions des données d’entraînement et des données de prédiction.

Expliquez comment nous définissons des seuils pour déclencher des alertes en cas de dérive significative.

* **Réentraînement automatique hebdomadaire via le DAG 2**

Préciser comment le DAG 2 est configuré pour réentraîner automatiquement le modèle chaque semaine.

Détailler les étapes du DAG 2 (collecte des nouvelles données, prétraitement, entraînement du modèle, évaluation, déploiement) et comment il s’intègre avec MLflow pour le suivi des expériences.

1. **Monitoring & Déploiement**

**Suivi & gestion post-déploiement** ---- **Cf script prediction\_dag.py) :**

* **Suivi automatisé** (via Airflow)

Expliquer comment Airflow permet de surveiller l’état d’exécution de chaque tâche dans nos DAGs.

Détailler comment nous utilisons l’interface d’Airflow pour suivre les succès, les échecs et la durée des tâches.

* **Logs centralisés** (via Airflow) : Les journaux d'exécution sont centralisés et accessibles via l’interface Airflow pour faciliter le diagnostic des problèmes.
* **Alertes** (via Airflow) :

Décrire comment nous avons intégré des notifications (e-mail ou Slack) en cas d’échec d’une tâche critique. Préciser les types d’événements qui déclenchent ces alertes.

**Suivi & gestion post-déploiement** ---- **Cf script predict\_api.py) :**

Informations sur l'utilisation de MLflow pour le monitoring et le suivi des performances des prédictions.

* **MLflow :** Le suivi des runs MLflow permet de monitorer les performances des prédictions au fil du temps.
* **Logging :** Utilisation de ce module pour enregistrer les erreurs et les informations importantes.
* **Variables d'environnement :** L'API dépend de variables d'environnement pour la configuration de MLflow et d'autres paramètres.
* **Infrastructure:** Dépendances sur un registre de modèles MLflow et potentiellement d'autres services (base de données, etc.).

**Axes d'Amélioration**

* Intégration de modèles d'ensembleplus complexes (augmenter n\_estimators au-delà de 10)
* Features engineering avancé incluant des lag features et des agrégations temporelles
* Incorporation de données satellitaires comme sources complémentaires
* Modèles probabilistes plus sophistiqués pour mieux quantifier l'incertitude des prédictions

**Considérations Éthiques et Environnementales**

* Équité géographique des prédictions grâce à une représentation équilibrée des différentes régions
* Impact carbone du machine learning (optimisation des ressources de calcul)
* Transparence algorithmique et interprétabilité des prédictions (avantage du RandomForest)

**Conclusion Technique**

Un système de prédiction météorologique efficace nécessite une approche multidimensionnelle combinant :

* Des prétraitement rigoureux (comme implémenté dans prepare\_data.py)
* Des algorithmes adaptatifs (RandomForest avec potentiel d'optimisation)
* Un monitoring continu via MLflow et une orchestration des workflows par DAGs
* Une amélioration itérative basée sur les métriques de performance collectées