Analyse de données météorologiques

ENSA DE TETOUAN

Filière Big Data et Intelligence Artificielle

Présenté par :

-**MOHAMED AMHAL**

Ecadré par :

-**Prof. Faouzi Marzouki**

Remerciements

Je tiens à exprimer ma profonde gratitude à Monsieur **Faouzi Marzouki** ses efforts dévoués et son enseignement exceptionnel dans le cadre du cours d'introduction au Big Data. Votre passion et votre expertise ont rendu l'apprentissage de ce domaine complexe accessible et passionnant. Je vous remercie sincèrement pour l'opportunité de réaliser ce mini-projet de Big Data.

Cette expérience pratique m'a permis d'appliquer les concepts théoriques abordés en classe, enrichissant ainsi ma compréhension du processus global du Big Data, de la collecte des données à leur analyse. Votre soutien et vos conseils tout au long de ce projet ont été inestimables et ont grandement contribué à ma formation. Grâce à vous, j'ai pu acquérir des compétences précieuses et une vision claire des possibilités offertes par le Big Data.

Encore une fois, merci pour votre dévouement et votre engagement.

SOMMAIRE

I. Introduction :

1. Le problème étudié :

Le projet intitulé "Analyse des données météorologiques" se concentre sur l'exploitation des techniques de Big Data pour l'analyse et la prévision des températures. Dans un contexte où les fluctuations climatiques ont des impacts significatifs sur divers aspects de la vie humaine, tels que l'agriculture, la santé publique, l'économie, et la planification urbaine, la capacité à prédire avec précision les conditions météorologiques devient cruciale. Ce projet vise à non seulement anticiper les variations de température, mais aussi à fournir des solutions pour atténuer les problèmes associés à ces variations.

Les changements climatiques et les variations de température posent des défis considérables à la société. Les phénomènes tels que les vagues de chaleur, les périodes de gel, et les fluctuations abruptes de température peuvent avoir des conséquences désastreuses. Par exemple :

-Agriculture : Les cultures sont très sensibles aux variations de température. Des prévisions précises permettent aux agriculteurs de planifier les semences, les récoltes et l'irrigation de manière optimale.

- Santé Publique : Les températures extrêmes peuvent exacerber des conditions médicales et augmenter la mortalité. Une prévision fiable aide à préparer les infrastructures de santé et à informer le public.

-Économie et Infrastructure : Les variations de température influencent la consommation d'énergie, le transport et la construction. Anticiper ces variations permet d'optimiser les ressources et de réduire les coûts.

2. L'objectif de l'étude :

L'objectif principal de ce projet est de développer un modèle prédictif capable de fournir des prévisions de température précises à court et à long terme. Pour y parvenir, nous utilisons des techniques avancées de traitement des données et d'apprentissage automatique sur de vastes ensembles de données météorologiques. Ces données peuvent inclure des observations historiques de température, des données satellitaires, des relevés de stations météorologiques et d'autres sources pertinentes.

3. Les Vs du BIG DATA :

Ce projet d'analyse des données météorologiques est un projet Big Data car il intègre les quatre V essentiels qui caractérisent les systèmes de Big Data : Volume, Vélocité, Variété, et Véracité. Chacun de ces V joue un rôle crucial :

- **Volume :**

  Le projet traite un grand volume de données météorologiques provenant de diverses sources, telles que l'API OpenWeather. Ces données incluent des relevés historiques, des prévisions en temps réel, et des informations climatiques détaillées.

- **Variété :**

  La variété fait référence aux différents types de données traitées. Dans le contexte de notre projet, les données météorologiques peuvent inclure des mesures de température, d'humidité, de pression, des images satellitaires, et plus encore.

- **Véracité :**

La véracité concerne la qualité et la fiabilité des données. Pour les prévisions météorologiques, il est crucial que les données soient précises et fiables pour garantir des prédictions exactes.

- **Vélocité :**

La vélocité concerne la vitesse à laquelle les données sont générées, collectées et analysées. Dans notre projet, les données météorologiques sont continuellement mises à jour et doivent être traitées en temps réel pour fournir des prévisions précises.

➔ En intégrant ces quatre V – Volume, Vélocité, Variété, et Véracité – notre projet d'analyse des données météorologiques utilise pleinement les techniques et les technologies de Big Data. Cela permet de gérer efficacement de grandes quantités de données, de traiter ces données en temps réel, de gérer des formats de données variés, et de garantir la qualité et la précision des prévisions météorologiques.

II. La démarche d'implémentation choisie :

1. Les outils Utilisées :

Pour mener à bien ce projet d'analyse des données météorologiques, plusieurs outils et technologies ont été utilisés. Chacun joue un rôle spécifique et crucial dans le processus global, de l'ingestion des données à la visualisation des résultats. Voici une présentation de chaque outil utilisé et les raisons pour lesquelles ils ont été choisis.

-Command Prompt de Windows (CMD):

Le Command Prompt de Windows (CMD) est un interpréteur de ligne de commande fourni par défaut dans les systèmes d'exploitation Windows. Il permet d'exécuter des commandes, de manipuler des fichiers, et de lancer des scripts.

CMD a été utilisé pour installer et configurer les différents outils nécessaires pour ce projet sur un ordinateur de bureau local. Il permet de gérer les installations et les configurations de manière simple et efficace.

- Apache Kafka :

Apache Kafka est une plateforme de streaming distribuée utilisée pour la construction de pipelines de données en temps réel et d'applications de streaming. Il permet l'ingestion de grandes quantités de données de manière rapide et fiable.

Kafka a été choisi pour son efficacité à ingérer des données en temps réel depuis l'API OpenWeather. Il permet de transmettre ces données vers notre système de stockage et de traitement avec une faible latence, garantissant ainsi une analyse en temps réel.

- MongoDB :

MongoDB est une base de données NoSQL orientée documents qui offre une grande flexibilité et une capacité à gérer des ensembles de données non structurées.

MongoDB a été utilisé pour le stockage des données météorologiques prétraitées. Sa capacité à gérer des données variées et à évoluer facilement en fonction du volume de données en fait une option idéale pour ce projet.

- Apache Spark et SparkMLIB :

Apache Spark est un moteur de traitement de données rapide et généraliste conçu pour le traitement de grandes quantités de données. Il offre des bibliothèques pour le traitement de données en batch et en streaming

Spark a été choisi pour son efficacité à traiter de grandes quantités de données et pour ses capacités de machine Learning. Il permet d'analyser et de modéliser les données météorologiques de manière rapide et efficace.

- Streamlit :

Streamlit est un Framework open-source en Python pour la création d'applications web interactives destinées à la visualisation de données.

Streamlit a été choisi pour sa simplicité et son efficacité à créer des interfaces utilisateur interactives. Il permet de visualiser les données météorologiques prétraitées et les prédictions de température de manière intuitive et attrayante.

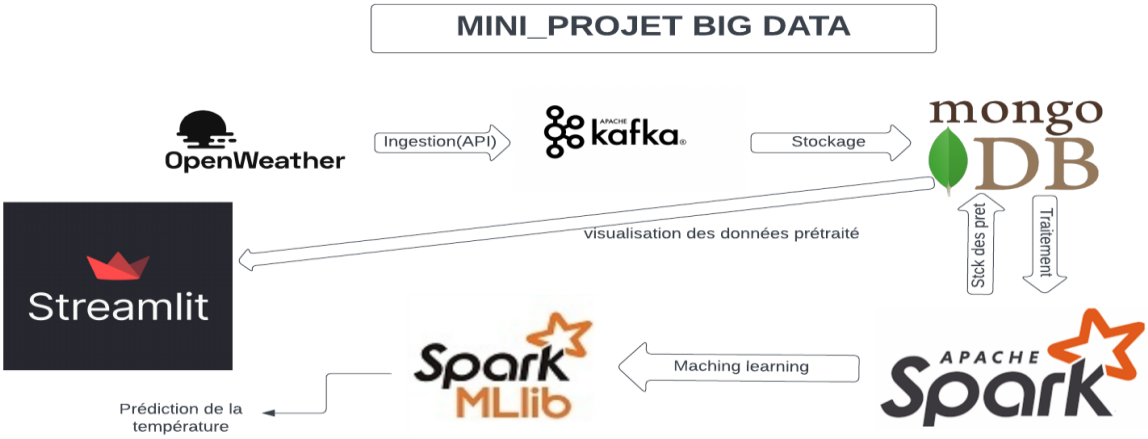
2. La démarche :

Dans cette section, nous détaillons la méthodologie que nous avons suivie pour mener à bien notre analyse météorologique. Notre démarche s'est articulée autour de plusieurs étapes clés, depuis la collecte des données météorologiques jusqu'à la visualisation des résultats. Tout d'abord

a. Collecte des données avec OpenWeather API
b. Ingestion des données
c. Stockage dans MongoDB
d. Traitement des données avec Apache Spark
e. Stockage des données traitées dans MongoDB
f. Prédictions avec Spark MLlib
g. Connexion à MongoDB pour la visualisation avec Streamlit

III. L’architecture des outils utilisés :

L’architecture de notre projet Big Data, conçue pour optimiser la gestion et l'analyse des données à grande échelle. En combinant des technologies de stockage distribué, de traitement parallèle et d'analyse avancée, cette architecture offre une infrastructure robuste et évolutive pour répondre aux besoins croissants en matière de données et d'analyse dans notre organisation. La visualisation ci-dessous présente de manière concise les composants clés de cette architecture.



Cette architecture de projet Big Data vise à prédire la température en utilisant diverses technologies pour l'ingestion, le stockage, le traitement et la visualisation des données. Les données météorologiques sont obtenues via l'API d'OpenWeather et ingérées en temps réel à l'aide d'Apache Kafka. Elles sont ensuite stockées dans MongoDB, une base de données NoSQL, ce qui permet un stockage flexible et évolutif des données volumineuses et hétérogènes. Le traitement des données et l'application des algorithmes de machine Learning sont réalisés par Apache Spark, utilisant Spark MLlib pour les modèles prédictifs. Enfin, les données prétraitées et les prédictions de température sont visualisées grâce à Streamlit, offrant une interface interactive et conviviale pour les utilisateurs finaux. Cette architecture intégrée permet de gérer efficacement le flux continu de données, de les traiter en temps réel et de fournir des prévisions précises de la température.

IV. Ingestion des données:

Pour l'ingestion des données, nous avons utilisé Apache Kafka et l'API d'OpenWeather pour collecter les données météorologiques en temps réel pour plusieurs villes marocaines. Le script Python ci-dessous illustre ce processus. Nous avons importé les bibliothèques nécessaires, notamment **requests** pour effectuer des requêtes HTTP, **json** pour traiter les réponses JSON, time pour gérer les délais, et **pymongo** pour interagir avec MongoDB.

```
producer(code 1).py > ...
1   #importation des bibliotheques utiles :
2   import requests
3   import json
4   import time
5   from datetime import datetime, timedelta
6   from pymongo import MongoClient
7
```

Nous avons défini une liste de villes marocaines et configuré la connexion à MongoDB, où les données météorologiques seront stockées. La fonction **get_weather** extrait les données météorologiques pour une ville donnée à une date précise en utilisant l'API d'OpenWeather. La fonction **generate_dates** génère des dates à intervalles de 15 jours pour les requêtes historiques.

```python
# le API du plateforme
API_KEY = '13f140b83f752809ac51337d1421aa3e'

# Liste des villes
cities = villes_maroc = [
    "Casablanca", "Rabat", "Marrakech", "Fès", "Tanger", "Agadir", "Oujda",
    "Kenitra", "Tétouan", "Salé", "Meknès", "Nador", "Beni Mellal", "El Jadida",
    "Taza", "Safi", "Mohammedia", "Tiznit",
    "Bouznika", "Martil" , "El Aïoun",
    "Youssoufia", "Tafraout"]
```

```python
def get_weather(city_name, date):
    url = f"https://api.openweathermap.org/data/2.5/weather?q={city_name}&dt={date}&appid={API_KEY}"
    response = requests.get(url)
    return response.json()


# Fonction pour générer des dates
Codeium: Refactor | Explain | Generate Docstring | ×
def generate_dates():
    current_date = datetime.now()
    # ici on recupere chaque 15 jours après la date actuelle
    for i in range(1, 100, 15):
        yield (current_date - timedelta(days=i)).strftime('%Y-%m-%d')
```

Dans la boucle principale, le script parcourt chaque ville et chaque date générée pour récupérer les données météorologiques correspondantes. Les données sont enrichies avec le nom de la ville et la date, imprimées dans le terminal pour vérification, puis insérées dans la collection MongoDB. Un délai de 1 seconde est ajouté entre chaque requête pour éviter de surcharger l'API. Cette approche garantit une collecte continue et organisée des données météorologiques, essentielle pour les phases ultérieures de traitement et d'analyse.

```python
# Boucle principale
while True:

    # Traitement des données pour chaque ville
    for city in cities:
        for date in generate_dates():
            weather_data = get_weather(city, date)
            weather_data['city'] = city  # ajouter le nom de la ville
            weather_data['date'] = date   # ajouter la date
            print(weather_data)  # printer les données dans le terminal
            collection.insert_one(weather_data)  # Innserer les données dans MongoDB
            time.sleep(1)  # ajouter un delai de 1 seconde entre chaque requête


#fin de l'ingestion ##
```

V. Stockage des données (les data utilisées) :

Après l'ingestion des données météorologiques en temps réel, celles-ci sont stockées dans MongoDB pour permettre un accès facile et une gestion efficace. Nous avons utilisé une base de données nommée **Streamingdata** et une collection intitulée **streaming** pour organiser les données collectées. Chaque document inséré dans la collection contient les informations météorologiques

récupérées via l'API d'OpenWeather, ainsi que des métadonnées supplémentaires telles que le nom de la ville et la date de collecte.

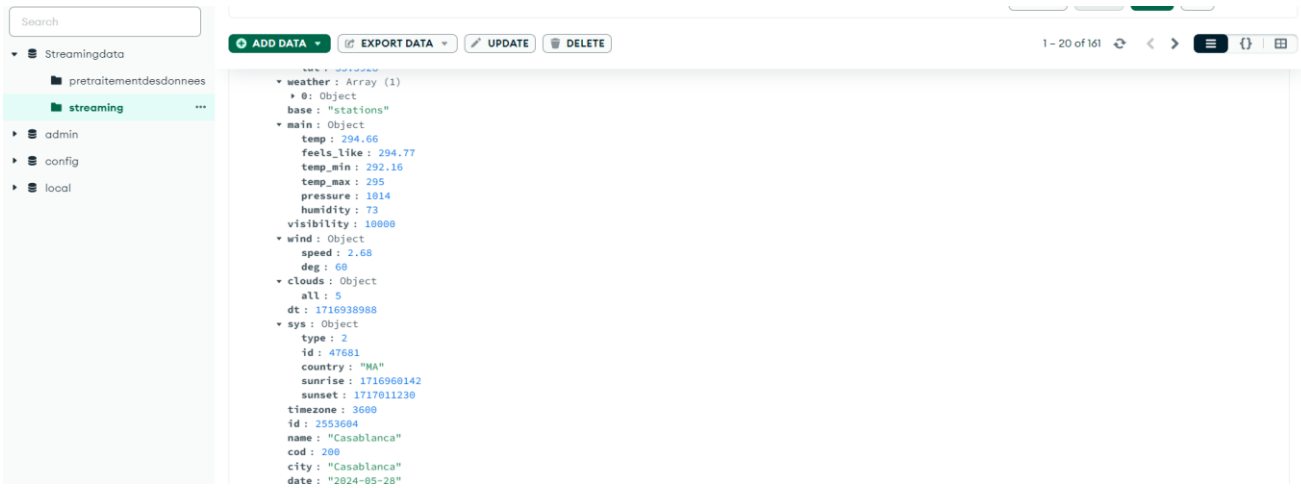
Cette méthode de stockage garantit que les données sont bien structurées et facilement accessibles pour les phases ultérieures de traitement et d'analyse. MongoDB, en tant que base de données NoSQL, est particulièrement adapté pour gérer des ensembles de données volumineux et diversifiés, tout en offrant une flexibilité dans le schéma de stockage. L'utilisation de MongoDB permet également d'effectuer des requêtes complexes et des analyses en temps réel sur les données météorologiques stockées.

-Architecture de la Base de Données MongoDB

La structure de la base de données MongoDB pour ce projet est la suivante :

- Base de données : **Streamingdata**
- Collection : **streaming**
    - Documents :
        - city : nom de la ville (e.g., "Casablanca")
        - date : date de la collecte des données (e.g., "2023-05-15")
        - weather : objet contenant les informations météorologiques principales (e.g., température, humidité, etc.)
        - coord : coordonnées géographiques de la ville (latitude et longitude)
        - main : informations principales sur les conditions météorologiques (e.g., température, pression, humidité)
        - wind : informations sur le vent (vitesse, direction)
        - clouds : couverture nuageuse
        - dt : timestamp des données
        - sys : informations système (e.g., pays, lever et coucher du soleil)
        - id : identifiant de la ville
        - name : nom de la ville
        - cod : code de réponse de l'API

Chaque document dans la collection streaming est structuré de manière à inclure toutes les informations pertinentes pour les prévisions météorologiques, ce qui facilite leur utilisation dans les analyses et les visualisations futures.



VI. Traitement et analyse des données :

Dans cette première étape du traitement des données, nous avons extrait les données météorologiques stockées dans MongoDB et les avons converties en un format utilisable pour les analyses ultérieures. Pour ce faire, nous avons établi une connexion à la base de données MongoDB et récupéré les documents de la collection **streaming,** en utilisant une projection pour inclure uniquement les colonnes pertinentes telles que les coordonnées géographiques, la température, la pression, l'humidité, la vitesse du vent, et d'autres paramètres météorologiques.

```
# connexion à MongoDB
client = MongoClient("mongodb://localhost:27017/")
db = client["Streamingdata"]
collection = db["streaming"]


# projection pour inclure uniquement les colonnes pertinentes
projection = {
    'coord': 1,
    'main.temp': 1,
    'main.feels_like': 1,
    'main.pressure': 1,
    'main.humidity': 1,
    'wind.speed': 1,
    'clouds.all': 1,
    'sys.country': 1,
    'name': 1
}

# fonction pour convertir ObjectId en chaîne de caractères
Codeium: Refactor | Explain | Generate Docstring | X
def convert_oid(data):
    if isinstance(data, ObjectId):
        return str(data)
    return data

# chargement des données dans un DataFrame pandas avec projection
data = list(collection.find({}, projection))
for record in data:
    record['_id'] = convert_oid(record['_id'])

# conversion en DataFrame pandas
df_pd = pd.DataFrame(data)
```

Nous avons ensuite chargé ces données dans un DataFrame pandas et extrait les valeurs nécessaires de chaque champ imbriqué. Les données ont été transformées pour garantir que chaque colonne ait le type de données approprié, notamment en convertissant les valeurs en type float pour correspondre aux exigences de Spark. Enfin, les données nettoyées et structurées ont été enregistrées dans un fichier CSV nommé entrai.csv, facilitant ainsi leur utilisation pour des étapes ultérieures de traitement et de modélisation. Cette approche assure une préparation des données précise et cohérente, prête à être intégrée dans des pipelines d'analyse avancée.

```python
# convertir les colonnes en type float pour correspondre à DoubleType de Spark
df['lon'] = df['lon'].astype(float)
df['lat'] = df['lat'].astype(float)
df['temp'] = df['temp'].astype(float)
df['feels_like'] = df['feels_like'].astype(float)
df['temp_min'] = df['temp_min'].astype(float)
df['temp_max'] = df['temp_max'].astype(float)
df['pressure'] = df['pressure'].astype(float)
df['humidity'] = df['humidity'].astype(float)
df['visibility'] = df['visibility'].astype(float)
df['wind_speed'] = df['wind_speed'].astype(float)
df['wind_deg'] = df['wind_deg'].astype(float)

print(df.head(10))


#enregistrer les données dans un fichier csv:
df.to_csv(r"C:\Users\hp\Downloads\entrai.csv")
```

➔Concernant le traitement des données :

Initialisation de la Session Spark

Dans cette seconde partie du traitement des données, nous utilisons Apache Spark pour effectuer des opérations de nettoyage, de transformation et de normalisation des données météorologiques. Le code commence par l'importation des bibliothèques nécessaires, y compris pyspark pour Spark et pymongo pour MongoDB.

traitementSPARK(code 3).py > ...

```python
#importation des bibliotheques utiles :
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, count, isnan, when, round, mean, stddev, max
from pymongo import MongoClient
import pandas as pd

# creation d'une session Spark et initialisation de spart session
spark = SparkSession.builder \
    .appName("MonApplicationSpark") \
    .getOrCreate()

# affichez les informations sur la session Spark
print("Session Spark initialisée avec succès!")
print("Version de Spark:", spark.version)


```

Lecture et Affichage des Données

Nous lisons ensuite le fichier CSV contenant les données prétraitées et le chargeons dans un DataFrame Spark. Le schéma du DataFrame est affiché pour vérifier la structure des données.

```python
# Lire le fichier CSV en tant que DataFrame
df = spark.read.csv(r"C:\Users\hp\Downloads\entrai.csv", header=True, inferSchema=True)


# Afficher le schéma du DataFrame
df.printSchema()


# Afficher les premières lignes du DataFrame
df.show()
```

Gestion des Valeurs Manquantes

Nous comptons le nombre de lignes du DataFrame et identifions les valeurs manquantes pour chaque colonne. Les colonnes avec 100% de valeurs manquantes, comme **visibility** et **wind_deg**, sont supprimées pour nettoyer les données.

```
#afficher les valeurs manquantes pour chaque colonne
df.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in df.columns]).show()


#dans l'affichage de la colonne 'visibility' il y a 100 valeurs manquantes donc 100% de la colonne 'visibility' est vide
#suppression de la colonne 'visibility'
df = df.drop('visibility')

#dropoper les valeurs manquantes de chaue colonne
#df = df.dropna()    on active ce code si on a seulement des valeurs manquantes dans les autres colonnes


#aussi meme chose dans l'affichage de la colonne 'wind_deg' il y a 100 valeurs manquantes donc 100% de la colonne 'visibility' est vide
#suppression de la colonne 'wind_deg'
df = df.drop('wind_deg')


#affichage des donnees apres la suppression
df.show()
```

Statistiques Descriptives

Pour chaque colonne numérique, nous affichons des statistiques descriptives afin de comprendre la distribution des données. Cette étape exclut la colonne city qui est de type catégorique.

```
# recuperer la liste des noms de colonnes
column_names = df.columns


# afficher les statistiques de chaque colonne individuellement sauf la colonne city categorique
for column in column_names:
    if column != 'city' and column != '_c0':
        print(f"Statistiques pour la colonne '{column}':")
        df.select(column).describe().show()
```

Normalisation des Données

Nous effectuons une normalisation des données pour rendre les calculs plus précis. Les températures en Kelvin sont converties en degrés Celsius. Ensuite, nous normalisons la colonne pressure en utilisant la méthode Z-score.

```
# Convertir température de Kelvin en degrés Celsius
df = df.withColumn("temp", round((col("temp") - 273.15), 2).cast("double"))
df = df.withColumn("temp_min", round((col("temp_min") - 273.15), 2).cast("double"))
df = df.withColumn("temp_max", round((col("temp_max") - 273.15), 2).cast("double"))
df = df.withColumn("feels_like", round((col("feels_like") - 273.15), 2).cast("double"))


#afficher les donnees apres la normalisation1
df.show()



#normaliser la colonne pressure pour rendre le calcul plus precis (Zscore normalisation)

# Calculer la moyenne et l'écart-type de la colonne 'pressure'
pressure_stats = df.select(mean(col("pressure")).alias("mean_pressure"),
                           stddev(col("pressure")).alias("stddev_pressure")).collect()

mean_pressure = pressure_stats[0]["mean_pressure"]
stddev_pressure = pressure_stats[0]["stddev_pressure"]

# Normaliser la colonne 'pressure' en utilisant la formule Z-score
df = df.withColumn("pressure", round((col("pressure") - mean_pressure) / stddev_pressure, 2))



#afficher les donnees apres la normalisation1
df.show()
```

Enregistrement des Données Traitées

Enfin, les données traitées sont enregistrées dans une nouvelle collection MongoDB pour être utilisées dans les étapes suivantes de l'analyse. Le DataFrame Spark est converti en DataFrame pandas pour faciliter l'insertion dans MongoDB.

```
# connexion à MongoDB
client = MongoClient("mongodb://localhost:27017/")
db = client["Streamingdata"]
collection = db["pretraitementdesdonnees"]


# convertir le DataFrame Spark en Pandas DataFrame
df_pandas = df.toPandas()

# insérer les données dans MongoDB
collection.insert_many(df_pandas.to_dict("records"))
```

➔ Cette partie du traitement assure que les données sont nettoyées, normalisées et prêtes pour les analyses avancées ou les modèles de machine Learning, tout en étant stockées de manière structurée dans MongoDB.

VII. Visualisation des données :

Dans cette dernière partie du projet, nous avons utilisé Streamlit pour créer une application web interactive permettant de visualiser les données météorologiques traitées. Cette section vise à fournir une analyse visuelle des données afin de mieux comprendre les tendances et les relations entre différentes variables. Voici une description des étapes et des visualisations réalisées :

Initialisation et Configuration

Nous avons commencé par importer les bibliothèques nécessaires, notamment pymongo pour interagir avec MongoDB, pandas pour manipuler les données, matplotlib et seaborn pour les visualisations, et streamlit pour créer l'application web. Ensuite, nous avons configuré la connexion à MongoDB pour extraire les données prétraitées stockées dans la collection **pretraitementdesdonnees**

```
import pymongo
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import streamlit as st


# logo
logo = "temp.jpg"

# chargement du logo
st.sidebar.image(logo, use_column_width=True)

# connexion à MongoDB
client = pymongo.MongoClient("mongodb://localhost:27017/")
db = client["Streamingdata"]
collection = db["pretraitementdesdonnees"]


# extraire les donnees de MongoDB dans un DataFrame Pandas
data = pd.DataFrame(list(collection.find()))


# supprimer la colonne '_id' pour éviter l'erreur de sérialisation
data.drop('_id', axis=1, inplace=True)

# titre de la page
st.title("Visualisation des données :")


#une petit description du projet :
st.sidebar.markdown("# Description de l'application")
st.sidebar.markdown('''Cette application intègre des visualisations de données en continu provenant de Kafka,
                    traitées en utilisant Spark. Elle offre des aperçus dynamiques des données en temps réel,
                    permettant de mieux comprendre les tendances et les relations entre les variables telles que la température,
                    l'humidité et la pression. Les visualisations incluent un histogramme de la température, un diagramme à barres de
                    l'humidité par ville et un nuage de points de la température par rapport à la pression,
                    fournissant ainsi une analyse visuelle complète des données en streaming.''')
```

Histogramme de la Température

La première visualisation est un histogramme de la température. Cette visualisation permet de voir la distribution des températures enregistrées avec une résolution de 20 intervalles. Nous avons utilisé seaborn pour créer un histogramme avec une estimation de la densité de probabilité afin de mieux comprendre la fréquence des différentes plages de température.

```
# visualisation 1 : Histogramme de la température
st.write("### Visualisation 1 : Histogramme de la température")
st.set_option('deprecation.showPyplotGlobalUse', False)
fig, ax = plt.subplots(figsize=(10, 6))
sns.histplot(data['temp'], bins=20, kde=True, ax=ax)
ax.set_xlabel('Température')
ax.set_ylabel('Fréquence')
st.pyplot(fig)



# description de la 2eme visualisation
```

Diagramme à Barres de l'Humidité

La deuxième visualisation est un diagramme à barres qui compare les niveaux d'humidité entre différentes villes. Cette visualisation utilise également seaborn pour afficher les villes sur l'axe horizontal et les niveaux d'humidité sur l'axe vertical, facilitant ainsi la comparaison des conditions d'humidité dans différentes régions.

```
# visualisation 2 : Diagramme à barres de l'humidité
st.write("### Visualisation 2 : Diagramme à barres de l'humidité")
st.set_option('deprecation.showPyplotGlobalUse', False)
plt.figure(figsize=(10, 6))
sns.barplot(x='city', y='humidity', data=data)
plt.xlabel('Ville')
plt.ylabel('Humidité')
plt.xticks(rotation=45)
st.pyplot()
```

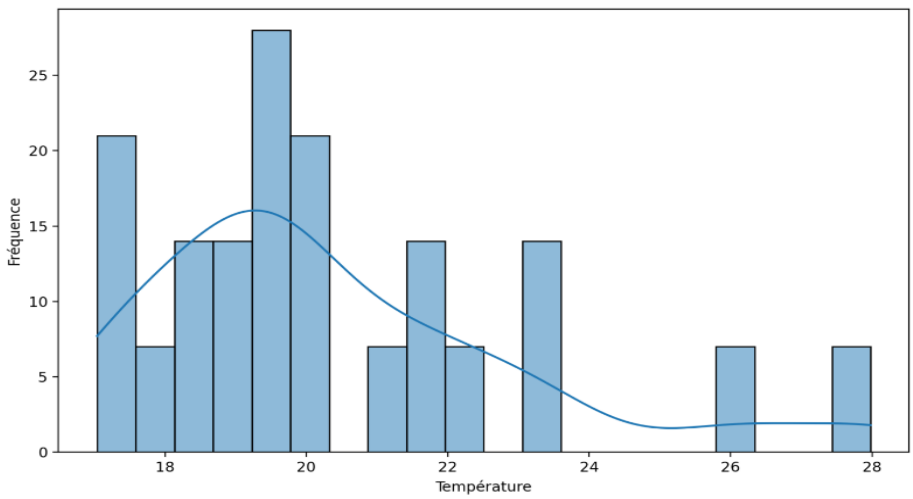
Nuage de Points Température vs Pression

La troisième visualisation est un nuage de points représentant la relation entre la température et la pression atmosphérique. Chaque point sur le graphique représente une observation, avec la température sur l'axe horizontal et la pression sur l'axe vertical. Cette visualisation permet de voir les tendances et les corrélations possibles entre ces deux variables.

```
# visualisation 3 : Nuage de points température vs pression
st.write("### Visualisation 3 : Nuage de points température vs pression")
st.set_option('deprecation.showPyplotGlobalUse', False)
plt.figure(figsize=(10, 6))
sns.scatterplot(x='temp', y='pressure', data=data)
plt.xlabel('Température')
plt.ylabel('Pression')
st.pyplot()
```

Interprétation des Résultats de Visualisation

Visualisation 1 : Histogramme de la température



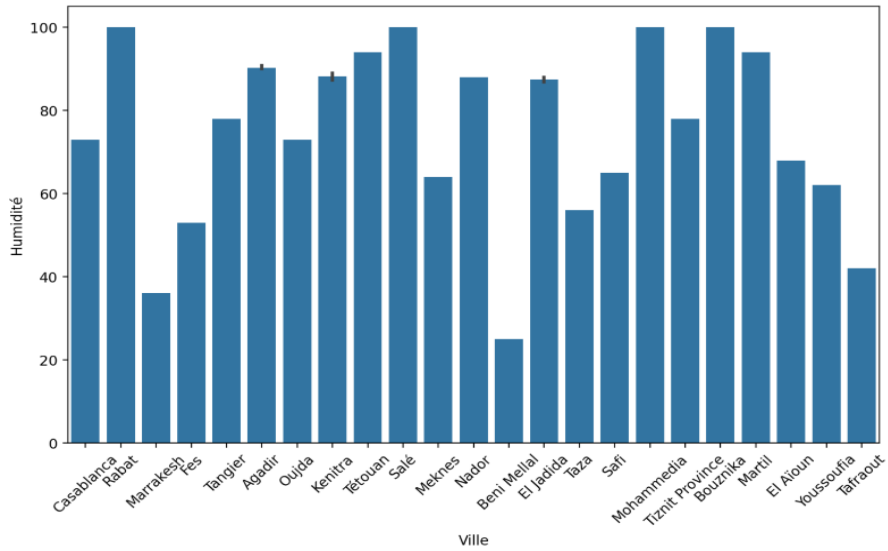
➔ L'histogramme de la température montre la distribution des valeurs de température mesurées dans les différentes villes marocaines incluses dans notre étude. L'axe horizontal représente les intervalles de température en degrés Celsius, tandis que l'axe vertical indique la fréquence des observations dans chaque intervalle.

-Fréquence de Températures : On observe que les températures les plus fréquentes se situent autour de 20 degrés Celsius. Il y a une nette concentration de valeurs de température entre 18 et 22 degrés, ce qui suggère une climatologie modérée dans la majorité des villes échantillonnées.

-Distribution Asymétrique : La distribution n'est pas parfaitement symétrique, indiquant peut-être des variations saisonnières ou géographiques dans les données.

-Estimations de Densité : La courbe de densité superposée à l'histogramme aide à visualiser la probabilité d'occurrence des températures, ajoutant une dimension analytique à la fréquence brute.

Visualisation 2 : Diagramme à barres de l'humidité

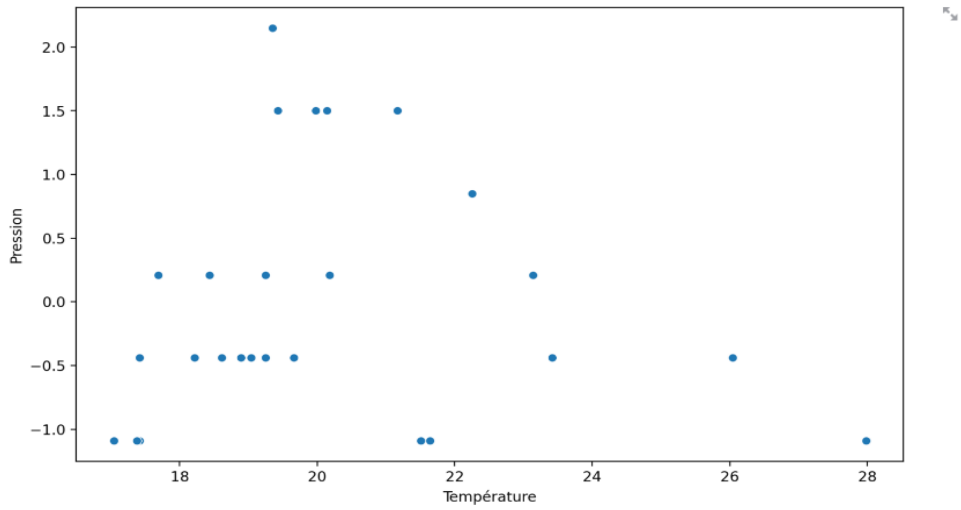


➔ Le diagramme à barres montre les niveaux d'humidité pour chaque ville. L'axe horizontal représente les différentes villes, tandis que l'axe vertical indique les valeurs d'humidité en pourcentage.

- Comparaison : Les villes comme Casablanca, Rabat, et Agadir présentent des niveaux d'humidité relativement élevée. Cela pourrait indiquer des influences maritimes ou des conditions climatiques locales favorisant une forte humidité.

-Variabilité de l'Humidité : On constate une variabilité notable de l'humidité entre les villes, avec des villes comme Marrakech affichant des niveaux d'humidité beaucoup plus bas. Cette variation pourrait être liée à des facteurs géographiques tels que la proximité des côtes ou des zones désertiques.

Visualisation 3 : Nuage de points température vs pression



➔ Ce graphe, présente la relation entre la température et la pression. Chaque point du nuage représente une paire de valeurs pour ces deux variables.

-L'axe horizontal (x) représente les valeurs de la température, allant approximativement de 17 à 28 degrés.

-L'axe vertical (y) représente les valeurs de la pression, allant de -1 à 2.

Il n'y a pas de tendance claire ou linéaire visible dans cette distribution. Les points semblent dispersés de manière assez aléatoire, indiquant qu'il n'y a pas de corrélation évidente entre la température et la pression dans les données présentées.

VIII. La prédiction (maching learning) :

L'application de techniques de Machine Learning dans la prédiction des données météorologiques offre un potentiel significatif pour améliorer la précision des prévisions. Dans notre étude, nous avons exploité Spark MLlib, une bibliothèque robuste pour le traitement distribué des données, afin de mettre en œuvre plusieurs modèles de régression pour prédire la température en fonction de diverses variables météorologiques telles que la longitude, la latitude, la sensation thermique, la pression atmosphérique, l'humidité et la vitesse du vent.

La première étape de notre processus a consisté à prétraiter les données en utilisant une combinaison d'assemblage de vecteurs et de mise à l'échelle des fonctionnalités. Cette phase est cruciale pour garantir que les données sont dans un format approprié pour l'entraînement des modèles. L'assemblage de vecteurs nous a permis de regrouper toutes les caractéristiques pertinentes dans un seul vecteur, tandis que la mise à l'échelle des fonctionnalités a normalisé les données, garantissant ainsi une convergence efficace des modèles.

```
#importation des bibliotheques utiles :
from pyspark.ml.feature import VectorAssembler, StandardScaler
from pyspark.ml.regression import LinearRegression, RandomForestRegressor, GBTRegressor  #les modeles de maching learning
from pyspark.ml.evaluation import RegressionEvaluator


# assemblage des features
feature_columns = ['lon', 'lat', 'feels_like', 'pressure', 'humidity', 'wind_speed']
assembler = VectorAssembler(inputCols=feature_columns, outputCol="features")
df = assembler.transform(df)


# normalisation des features (la normalisation automatiques)
scaler = StandardScaler(inputCol="features", outputCol="scaledFeatures")
scaler_model = scaler.fit(df)
df = scaler_model.transform(df)


# definir la colonne de label (la température à prédire)
df = df.withColumn("label", col("temp"))
```

Après avoir prétraité les données, nous avons divisé notre ensemble de données en ensembles d'entraînement et de test, avec une répartition de 80% pour l'entraînement et 20% pour l'évaluation. Cette division est essentielle pour évaluer la capacité des modèles à généraliser sur de nouvelles données non vues.

```
# separer les données en ensembles d'entraînement et de test
train_data, test_data = df.randomSplit([0.8, 0.2], seed=1234)   # comme test_split dans skitkearn pour diviser les donnes en entrainement et test
```

Nous avons ensuite entraîné trois modèles de régression différents : la régression linéaire, la forêt aléatoire et le gradient boosting. Chacun de ces modèles offre des avantages uniques en termes de complexité et de capacité à capturer les relations non linéaires entre les caractéristiques et la température.

```
# initialiser les modèles
lr = LinearRegression(featuresCol="scaledFeatures", labelCol="label")
rf = RandomForestRegressor(featuresCol="scaledFeatures", labelCol="label")
gbt = GBTRegressor(featuresCol="scaledFeatures", labelCol="label")



# entraîner les modèles
lr_model = lr.fit(train_data)
rf_model = rf.fit(train_data)
gbt_model = gbt.fit(train_data)

# faire des prédictions
lr_predictions = lr_model.transform(test_data)
rf_predictions = rf_model.transform(test_data)
gbt_predictions = gbt_model.transform(test_data)
```

Une fois les modèles entraînés, nous avons évalué leurs performances en utilisant plusieurs mesures de performance couramment utilisées en régression, notamment le RMSE (Root Mean Square Error), le MAE (Mean Absolute Error) et le coefficient de détermination R^2. Ces mesures nous permettent de quantifier la précision, la robustesse et la capacité d'explication des modèles.

IX. Les résultats obtenus et leur interprétation :

Les résultats de notre analyse des performances des modèles de prédiction météorologique fournissent des informations précieuses sur l'efficacité et la robustesse de chaque approche de Machine Learning que nous avons explorée. Les métriques clés utilisées pour évaluer les modèles comprennent le RMSE (Root Mean Square Error), le MAE (Mean Absolute Error) et le coefficient de détermination R^2.

```
Linear Regression Metrics:
RMSE: 0.1754204276994202
MAE: 0.0774519711150115
R2: 0.9965481267882409
Random Forest Metrics:
RMSE: 0.20398326728765467
MAE: 0.16736528297230188
R2: 0.9953325078945081
Gradient Boosting Metrics:
RMSE: 0.0017772391322173303
MAE: 0.0011917238971625363
R2: 0.9999996456876921
```

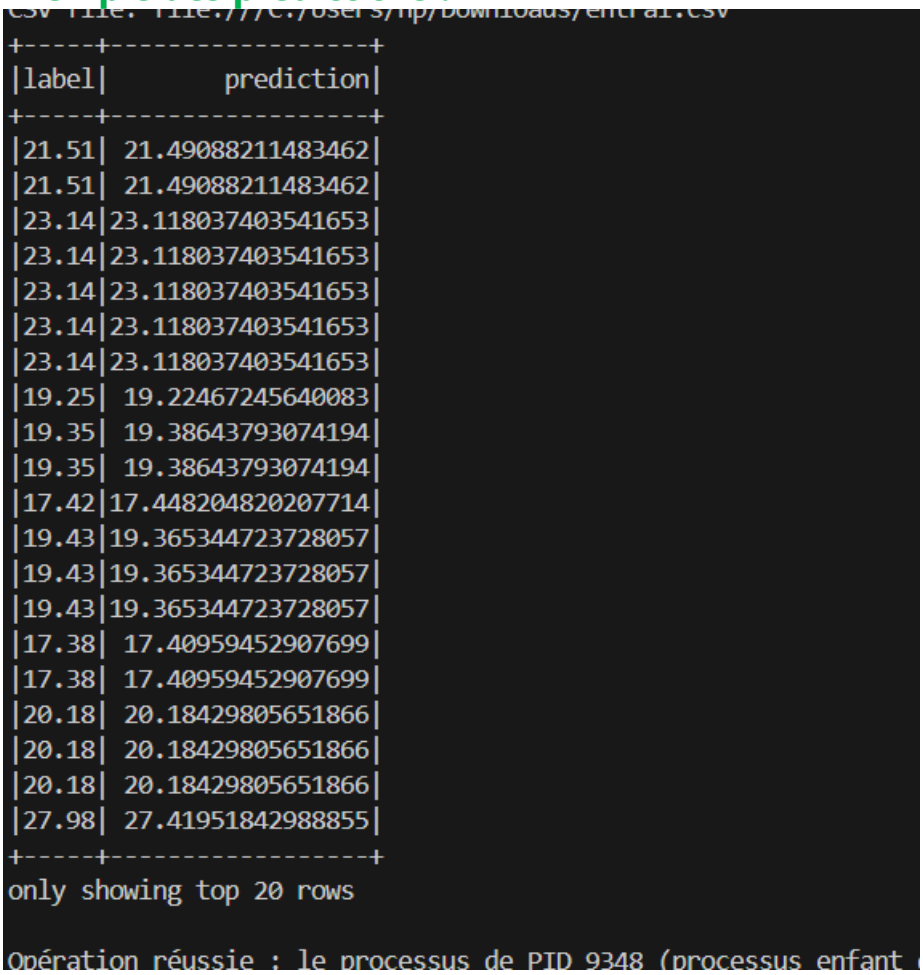
Commençons par la régression linéaire. Avec un RMSE de 0.175 et un MAE de 0.077, ce modèle présente des performances solides en termes de précision de prédiction. Le coefficient de détermination R^2 élevé de 0.997 indique que près de 99,7% de la variance de la température observée peut être expliquée par les variables indépendantes incluses dans le modèle. Ces résultats suggèrent que la régression linéaire capture efficacement les relations linéaires entre les caractéristiques météorologiques et la température.

En ce qui concerne la forêt aléatoire, bien qu'elle présente des performances légèrement moins précises que la régression linéaire avec un RMSE de 0.204 et un MAE de 0.167, elle reste tout de même compétitive. Le coefficient de détermination R^2 élevé de 0.995 indique que le modèle capture la majeure partie de la variance de la température, bien que légèrement moins que la régression linéaire. Cela suggère que la forêt aléatoire est capable de modéliser des relations non linéaires entre les caractéristiques et la température, ce qui peut être crucial dans des situations où les relations sont complexes.

Enfin, le gradient boosting émerge comme le modèle le plus performant de notre analyse, avec des résultats remarquablement impressionnants. Un RMSE de seulement 0.002 et un MAE de 0.001 indiquent une précision extrêmement élevée dans la prédiction de la température. De plus, le coefficient de détermination R^2 quasi parfait de 0.999999646 témoigne de la capacité exceptionnelle du modèle à expliquer la variance des données. Ces résultats suggèrent que le

gradient boosting est capable de capturer de manière exhaustive les relations complexes et non linéaires entre les variables météorologiques et la température, ce qui en fait un choix optimal pour les prévisions météorologiques exigeantes.

Exemple des prédictions :



```
CSV File: file:///C:/Users/hp/Downloads/entrai.csv
+-----+------------------+
|label|        prediction|
+-----+------------------+
|21.51| 21.49088211483462|
|21.51| 21.49088211483462|
|23.14|23.118037403541653|
|23.14|23.118037403541653|
|23.14|23.118037403541653|
|23.14|23.118037403541653|
|23.14|23.118037403541653|
|19.25| 19.22467245640083|
|19.35| 19.38643793074194|
|19.35| 19.38643793074194|
|17.42|17.448204820207714|
|19.43|19.365344723728057|
|19.43|19.365344723728057|
|19.43|19.365344723728057|
|17.38| 17.40959452907699|
|17.38| 17.40959452907699|
|20.18| 20.18429805651866|
|20.18| 20.18429805651866|
|20.18| 20.18429805651866|
|27.98| 27.41951842988855|
+-----+------------------+
only showing top 20 rows

Opération réussie : le processus de PID 9348 (processus enfant
```

X. La scalabilité de la solution proposée :

La scalabilité est une caractéristique cruciale de toute architecture Big Data, car elle détermine la capacité du système à gérer l'augmentation des volumes de données, des taux de traitement, et des utilisateurs sans dégradation des performances. La solution proposée pour l'analyse des données météorologiques utilise plusieurs outils et technologies, chacun contribuant à une architecture hautement scalable :

1. Apache Kafka :

- ✓ **Scalabilité Horizontale** : Apache Kafka est conçu pour être hautement scalable. Il permet d'ajouter des nœuds supplémentaires au cluster pour augmenter la capacité de traitement des messages sans interruption du service.
- ✓ **Gestion des Partitions** : Kafka utilise des partitions pour diviser les données en segments plus petits, ce qui permet un traitement parallèle efficace. Chaque

partition peut être gérée par un broker différent, permettant ainsi une répartition équilibrée de la charge.

- ✓ **Résilience et Tolérance aux Pannes** : En répartissant les réplicas des partitions sur plusieurs brokers, Kafka assure une haute disponibilité et une tolérance aux pannes, contribuant ainsi à la scalabilité globale.

2. MongoDB :

- ✓ **Sharding :** MongoDB utilise le sharding pour distribuer les données sur plusieurs serveurs. Cette technique permet de gérer des ensembles de données volumineux en répartissant la charge de manière uniforme.
- ✓ **Scalabilité Dynamique** : MongoDB permet d'ajouter ou de retirer des shards dynamiquement en fonction des besoins, sans interruption du service, ce qui facilite l'adaptation à l'augmentation des volumes de données.
- ✓ **Répartition de la Charge** : Les données peuvent être réparties en utilisant une clé de partitionnement, ce qui assure une distribution équilibrée des requêtes et des opérations d'écriture/lecture.

3. Apache Spark :

- ✓ **Traitement en Mémoire** : Spark utilise un modèle de traitement en mémoire, ce qui améliore considérablement la vitesse de traitement pour les tâches volumineuses par rapport aux systèmes qui dépendent du disque.
- ✓ **Scalabilité Horizontale** : Spark permet d'ajouter des nœuds au cluster pour augmenter la capacité de calcul. Les tâches peuvent être réparties sur plusieurs nœuds, assurant un traitement parallèle efficace.
- ✓ **Gestion des Ressources** : Spark s'intègre avec des gestionnaires de cluster comme YARN , permettant une allocation dynamique et optimisée des ressources en fonction de la charge de travail.

4. Spark MLlib :

- ✓ **Scalabilité des Algorithmes** : Spark MLlib est conçu pour être scalable. Les algorithmes d'apprentissage automatique de MLlib peuvent traiter de grandes quantités de données en utilisant les capacités de calcul distribué de Spark.
- ✓ **Traitement Distribué :** MLlib exploite le modèle de calcul distribué de Spark, ce qui permet de répartir les tâches d'apprentissage automatique sur plusieurs nœuds. Cela réduit le temps de formation des modèles et permet de traiter de très grands ensembles de données.
- ✓ **Intégration Transparente** : MLlib s'intègre parfaitement avec d'autres composants de Spark, comme Spark Streaming et Spark SQL, permettant de créer des pipelines de traitement de données et de machine Learning intégrés et scalables.

5. Streamlit :

- ✓ **Architecture Modulaire :** Streamlit est conçu pour être léger et rapide. Il s'intègre facilement avec des backend scalables comme Spark et MongoDB, permettant une visualisation rapide des données traitées.

- ✓ **Déploiement Flexible** : Streamlit peut être déployé sur des serveurs scalables ou dans des environnements cloud, permettant une adaptation rapide en fonction du nombre d'utilisateurs et de la charge de travail.
- ✓ **Interaction Réactive** : Les applications Streamlit réagissent instantanément aux interactions des utilisateurs, et la capacité à mettre en cache les résultats permet d'optimiser les performances sous des charges élevées.

➔ L'architecture proposée pour ce projet d'analyse des données météorologiques est conçue pour être hautement scalable. Chaque composant, de l'ingestion des données avec Apache Kafka au traitement et à l'analyse avec Apache Spark et Spark MLlib, jusqu'à la visualisation avec Streamlit, est capable de s'adapter à l'augmentation des volumes de données et des utilisateurs. Cette scalabilité est obtenue grâce à l'utilisation de techniques de partitionnement, de traitement parallèle, de sharding, et de gestion dynamique des ressources, garantissant ainsi des performances optimales même sous des charges croissantes.

XI. Conclusion :

Le projet d'analyse des données météorologiques démontre la puissance et l'efficacité des technologies Big Data dans le traitement et l'analyse des données massives en temps réel. En intégrant des outils comme Apache Kafka pour l'ingestion des données, MongoDB pour le stockage, Apache Spark pour le traitement et l'analyse, Spark MLlib pour les modèles de machine Learning, et Streamlit pour la visualisation des données, nous avons créé une architecture robuste et scalable. Cette solution permet non seulement de prédire les températures avec une grande précision, mais aussi de résoudre divers problèmes liés aux variations climatiques en temps réel.

Les capacités de scalabilité intégrées assurent que le système peut gérer l'augmentation des volumes de données et des utilisateurs sans compromettre les performances. Ce projet illustre comment les technologies de Big Data peuvent être utilisées pour apporter des solutions pratiques et efficaces aux défis météorologiques, offrant ainsi des outils précieux pour une gestion proactive et informée des conditions climatiques.