Université Sultan Moulay Slimane
Ecole Nationale des Sciences Appliquées Khouribga
Département Mathématiques et Informatique

Rapport projet Big Data

Analyse de la relation entre les données du trafic aérien et des conditions météorologiques

---

IID 3
2023-2024

Réalisé par :
Chaimaa KHALIL & Siham HAFSI

Encadré par :
Mme. Nassima SOUSSI

PLAN :

1. Introduction :

Dans le ballet aérien incessant qui façonne notre monde moderne, l'aviation reste non seulement une prouesse technologique, mais également une danse délicate entre la maîtrise humaine et les forces naturelles. Les conditions météorologiques sont l'un des acteurs omniprésents dans cette scène aérienne, exerçant une influence incontestable sur la sécurité, l'efficacité et la planification des vols. Ce projet incarne une exploration en profondeur de cette relation complexe et essentielle entre les conditions météorologiques et le trafic aérien, offrant un regard neuf sur la manière dont ces deux domaines interagissent et impactent l'aviation moderne.

2. Description du projet :

Ce projet a pour objectif principal d'obtenir une compréhension approfondie de la relation entre le trafic aérien et les conditions météorologiques, en utilisant des données massives (Big Data) et des techniques d'analyse avancées.

1. Analyser l'influence des conditions météorologiques sur le trafic aérien : En étudiant les données météorologiques en corrélation avec les données de trafic aérien, le projet vise à comprendre comment les conditions météorologiques, telles que les tempêtes, les brouillards épais, etc., affectent les opérations aériennes. Cela peut avoir un impact sur la sécurité et l'efficacité des vols.

2. Identifier les pays les plus touchés : Le projet a pour but de déterminer quels pays européens sont les plus touchés par des conditions météorologiques défavorables en relation avec le trafic aérien. Cette information peut être utile pour les autorités aéroportuaires et les compagnies aériennes dans la gestion des opérations.

3. Optimisation et prise de décisions informées : Les résultats de ce projet peuvent être utilisés pour optimiser les opérations aériennes, améliorer la sécurité des vols

3. Collecte de données :

On a fait la collecte des données à partir de deux sources de données différentes :

3.1. OpenWeatherMap API

La collecte de données s'effectue à travers une méthode systématique et exhaustive, utilisant une série de requêtes à **l'API OpenWeatherMap** pour obtenir des informations météorologiques spécifiques pour chaque pays européen. Les données récoltées sont structurées en plusieurs colonnes clés :

- **Country** : Cette colonne identifie le pays européen concerné, permettant ainsi une analyse météorologique individualisée pour chaque nation.

- **Description** : Cette rubrique offre une description détaillée des conditions météorologiques actuelles dans chaque pays, offrant un aperçu succinct mais informatif des phénomènes météorologiques dominants.

- **Temperature (K)** : La température en Kelvin, représentant la mesure numérique de la chaleur ou du froid dans chaque région. Cette donnée est essentielle pour comprendre le spectre thermique dans ces pays européens.

- **Pressure (hPa)** : Indiquant la pression atmosphérique en hectopascals (hPa), cette valeur offre un indice crucial des fluctuations barométriques, une variable pertinente dans l'étude des conditions météorologiques.

- **Humidity (%)** : La colonne de l'humidité exprime en pourcentage la quantité de vapeur d'eau présente dans l'atmosphère. Cette mesure joue un rôle significatif dans la perception et le ressenti climatique.

- **Wind Speed (m/s)** : Cette rubrique présente la vitesse du vent en mètres par seconde, une donnée essentielle pour évaluer les conditions météorologiques potentiellement impactantes sur le trafic aérien.

Ces données météorologiques, minutieusement collectées et organisées, constituent une base solide pour les analyses ultérieures, offrant une perspective complète des variations météorologiques dans les pays européens et leur potentiel impact sur le trafic aérien.

- Nous avons développé un script Python pour recueillir des données météorologiques en pour les pays européens en utilisant l'API OpenWeatherMap. Voici comment fonctionne le script :

Importation des Bibliothèques :
Nous commençons par importer les bibliothèques nécessaires : requests pour effectuer des requêtes HTTP vers l'API OpenWeatherMap et Nominatim de Geopy pour obtenir les coordonnées géographiques des pays.

Initialisation de l'API et du Géolocalisateur :
Nous utilisons une clé d'API OpenWeatherMap pour autoriser les requêtes à l'API. Ensuite, nous initialisons le géolocalisateur Nominatim pour obtenir les coordonnées géographiques des pays.
Liste des Pays Européens : Une liste de noms de pays européens est définie dans la variable countries.

Préparation pour la Collecte des Données :

Nous initialisons une liste vide nommée weather_data pour stocker les données météorologiques collectées.

Boucle pour la Collecte de Données :
Le script itère à travers chaque pays dans la liste des pays européens. Pour chaque pays, il utilise le géolocalisateur Nominatim pour obtenir les coordonnées géographiques (latitude et longitude).
En utilisant ces coordonnées, une requête est faite à l'API OpenWeatherMap pour obtenir les données météorologiques actuelles du pays.
Si la requête est réussie (code de statut HTTP 200), les données météorologiques telles que la description, la température, la pression, l'humidité et la vitesse du vent sont extraites de la réponse JSON de l'API.
Ces données sont structurées sous forme de dictionnaire et ajoutées à la liste weather_data.

Conversion des Données en DataFrame :
Les données météorologiques collectées sont converties en un DataFrame Pandas pour une meilleure manipulation et analyse.

Sauvegarde des Données dans un Fichier CSV :
Enfin, le DataFrame est enregistré dans un fichier CSV nommé "weather_data.csv" à l'aide de la fonction to_csv() de Pandas, avec l'option index=False pour éviter d'écrire l'index du DataFrame dans le fichier CSV.
Ce script permet une collecte automatisée et organisée de données météorologiques pour les pays européens, fournissant ainsi une base solide pour des analyses approfondies des conditions météorologiques et de leur impact potentiel sur divers domaines, notamment le trafic aérien.

```python
import requests
from geopy.geocoders import Nominatim
api_key = "5e236ccc7a89f7789ce24aeaba494c52"
# Obtenir les coordonnées géographiques de chaque pays européen à partir de GeoPy
geolocator = Nominatim(user_agent="my-custom-user-agent")
countries = ["Albania", "Andorra", "Austria", "Belarus", "Belgium", "Bosnia and Herzegovina", "Bulgaria", "Croatia",
             "Cyprus", "Czech Republic", "Denmark", "Estonia", "Finland", "France", "Germany", "Greece", "Hungary",
             "Iceland", "Ireland", "Italy", "Kosovo", "Latvia", "Liechtenstein", "Lithuania", "Luxembourg", "Malta",
             "Moldova", "Monaco", "Montenegro", "Netherlands", "North Macedonia", "Norway", "Poland", "Portugal",
             "Romania", "Russia", "San Marino", "Serbia", "Slovakia", "Slovenia", "Spain", "Sweden", "Switzerland",
             "Ukraine", "United Kingdom", "Vatican City"]
import pandas as pd
weather_data = []

for country in countries:
    location = geolocator.geocode(country)
    if location:
        lat, lon = location.latitude, location.longitude
        url = f"http://api.openweathermap.org/data/2.5/weather?lat={lat}&lon={lon}&appid={api_key}"
        response = requests.get(url)
        if response.status_code == 200:
            data = response.json()
            weather = {
                'Country': country,
                'Description': data['weather'][0]['description'] if 'weather' in data else 'N/A',
                'Temperature (K)': data['main']['temp'] if 'main' in data else 'N/A',
                'Pressure (hPa)': data['main']['pressure'] if 'main' in data else 'N/A',
                'Humidity (%)': data['main']['humidity'] if 'main' in data else 'N/A',
                'Wind Speed (m/s)': data['wind']['speed'] if 'wind' in data else 'N/A'
            }
            weather_data.append(weather)
        else:
            print(f"Failed to retrieve data for {country}")

df = pd.DataFrame(weather_data)
df.to_csv("weather_data.csv", index=False)
```

	A	B	C	D	E	F	G
1	Country	Description	Temperature	Pressure (hP	Humidity (%)	Wind Speed (m/s)	
2	Albania	broken clouc	297.28	1010	92	1.52	
3	Andorra	broken clouc	271.97	1019	76	0.47	
4	Austria	broken clouc	268.77	1020	83	2.02	
5	Belarus	overcast clou	269.18	1007	61	6.66	
6	Belgium	moderate ra	280.2	1000	92	7.2	
7	Bosnia and H	overcast clou	272.86	1017	97	1.07	
8	Bulgaria	broken clouc	277.05	1013	81	1.54	
9	Croatia	overcast clou	278.18	1017	94	0.45	
10	Cyprus	overcast clou	277.58	1019	77	0.83	
11	Czech Repub	overcast clou	276.22	1014	73	3.91	
12	Denmark	overcast clou	278.35	1002	95	3.19	
13	Estonia	overcast clou	258.67	1021	72	4.36	
14	Finland	clear sky	244.35	1031	97	0.76	
15	France	light rain	282.76	1012	86	8.64	

3.2. OpenSky API

Le processus de collecte de données à partir de l'API OpenSky a été réalisé avec une attention particulière aux colonnes suivantes :

- **time** : Cette colonne fournit l'horodatage des observations des avions, permettant ainsi une chronologie précise des événements.

- **icao24** : L'identifiant unique de l'aéronef (ICAO 24-bit address) utilisé pour suivre et identifier un avion spécifique.

- **lat** et **lon** : Ces colonnes représentent respectivement la latitude et la longitude actuelles de l'aéronef, fournissant ainsi des informations cruciales sur sa position géographique.

- **velocity** : La vitesse actuelle de l'avion, exprimée en unités appropriées, permettant de déterminer la rapidité de déplacement de l'aéronef.

- **heading** : L'orientation ou la direction vers laquelle pointe l'avion au moment de l'observation, exprimée en degrés.

- **vertrate** : Le taux vertical de changement d'altitude, indiquant si l'avion est en montée, descente ou maintien de l'altitude.

- **callsign** : L'indicatif d'appel de l'aéronef, permettant de l'identifier sur les communications radio et dans les systèmes de contrôle aérien.

- **onground** : Un indicateur déterminant si l'aéronef est au sol (True) ou en vol (False).

- **alert**, **spi**, **squawk** : Ces colonnes fournissent des informations relatives aux indications d'urgence, à l'identification spécial (SPI), et au code de transpondeur (Squawk) émis par l'aéronef.

- **baroaltitude** et **geoaltitude** : Respectivement, l'altitude de l'aéronef mesurée par rapport à la pression atmosphérique (barométrique) et l'altitude géométrique par rapport au niveau de la mer.

- **lastposupdate** et **lastcontact** : Ces colonnes donnent des indications temporelles importantes, la première concernant la dernière mise à jour de la position de l'aéronef et la seconde concernant le dernier contact avec l'aéronef.

La collecte de ces données a été effectuée avec une attention minutieuse pour assurer la qualité, la précision et la cohérence des informations obtenues, fournissant ainsi une base solide pour des analyses approfondies des mouvements et des paramètres des avions dans l'espace aérien.

- on a fait un script Python pour manipuler et nettoyer les données extraites de l'API OpenSky, fournissant des informations sur les vols et leur pays d'origine. Voici le processus détaillé :

Chargement et Nettoyage des Données :
Le script charge initialement un fichier CSV (states_2022-06-27-00.csv) dans un DataFrame Pandas donnees. Il identifie ensuite certaines lignes à supprimer en fonction des préfixes dans

la colonne 'icao24' en utilisant la méthode startswith(). Ces lignes sont supprimées du DataFrame, et les données modifiées sont sauvegardées dans le même fichier CSV.

Correspondance entre les Préfixes 'icao24' et les Pays :
Le script crée un dictionnaire nommé correspondance_pays, associant des préfixes 'icao24' à des noms de pays.

Assignation des Pays aux Vols :
Une fonction trouver_pays() est définie pour trouver le pays correspondant à partir de la colonne 'icao24'. Cette fonction itère à travers le dictionnaire de correspondance pour trouver le pays correspondant au préfixe 'icao24' de chaque vol.

Ajout d'une Nouvelle Colonne 'Country' :
Une nouvelle colonne 'Country' est ajoutée au DataFrame en utilisant la fonction trouver_pays() pour mapper les préfixes 'icao24' aux noms de pays correspondants.

Filtrage des Vols sans Pays Correspondant :
Le script identifie et affiche les lignes où la colonne 'Country' a la valeur 'Autre' (les vols pour lesquels aucun pays correspondant n'a été trouvé). Ces lignes sont supprimées du DataFrame, et les données modifiées sont sauvegardées dans un nouveau fichier CSV nommé 'air.csv'.

Ce script permet donc de nettoyer et d'assigner des pays aux données de vols provenant de l'API OpenSky, facilitant ainsi une analyse plus précise des origines géographiques des vols aériens.

```python
import pandas as pd

# Charger le fichier CSV dans un DataFrame
nom_fichier = 'states_2022-06-27-00.csv'
donnees = pd.read_csv(nom_fichier)

# Identifier les lignes à supprimer en fonction de la colonne 'icao24'
indices_a_supprimer = donnees[donnees['icao24'].str.startswith(('a', '8', 'e', '7', 'c','0','9','b','f','507'))].index

# Supprimer les lignes correspondantes
donnees = donnees.drop(indices_a_supprimer)

# Enregistrer les données modifiées dans le même fichier CSV
donnees.to_csv(nom_fichier, index=False)
```

```python
import pandas as pd

# Charger le fichier CSV dans un DataFrame
nom_fichier = 'states_2022-06-27-00.csv'
donnees = pd.read_csv(nom_fichier)

# Créer un dictionnaire de correspondance entre les préfixes 'icao24' et les pays
correspondance_pays = {
    '39': 'France',
    '501': 'Albania',
    '440': 'Austria',
    '510': 'Belarus',
    '44c': 'Belgium',
    '44f': 'Belgium',
    '44b': 'Belgium',
    '451': 'Bulgaria',
    '452': 'Bulgaria',
    '457': 'Bulgaria',
    '450': 'Bulgaria',
    '501c': 'Croatia',
    '501d': 'Croatia',
    '501f': 'Croatia',
    '501e': 'Croatia',
    '49d': 'Czech Republic',
    '49f': 'Czech Republic',
    '45f': 'Denmark',
    '45a': 'Denmark',
    '45d': 'Denmark',
    '511': 'Estonia',
    '460': 'Finland',
    '461': 'Finland',
    '464': 'Finland',
    '3c': 'Germany',
    '3f': 'Germany',
    '3d': 'Germany',
    '3e': 'Germany',
    '469': 'Greece',
    '46a': 'Greece',
    '46b': 'Greece',
    '46c': 'Greece',
    '46d': 'Greece',
    '468': 'Greece',
    '470': 'Hungary',
    '471': 'Hungary',
    '4cc': 'Iceland',
    '4ca': 'Ireland',
    '30': 'Italy',
    '31': 'Italy',
    '502d': 'Latvia',
    '502c': 'Latvia',
    '503c': 'Lithuania',
    '503d': 'Lithuania',
    '406': 'United Kingdom',
    '1': 'Russia'
}

# Fonction pour trouver le pays correspondant à partir de la colonne 'icao24'
def trouver_pays(code):
    for prefixe, pays in correspondance_pays.items():
        if code.startswith(prefixe):
            return pays
    return 'Autre'  # Si aucun pays correspondant n'est trouvé

# Ajouter une nouvelle colonne 'country' basée sur la correspondance 'icao24' -> pays
donnees['Country'] = donnees['icao24'].apply(trouver_pays)

# Enregistrer les données modifiées dans le même fichier CSV
donnees.to_csv(nom_fichier, index=False)
```

```python
# Afficher les lignes où la colonne 'country' a la valeur 'Autre'
autres_pays = donnees[donnees['Country'] == 'Autre']
print(autres_pays)
```

```
             time  icao24        lat        lon    velocity     heading  \
26     1656288010  4677ec        NaN        NaN         NaN         NaN
53     1656288010  485a82  45.352936  17.245200  221.751622  317.538545
57     1656288010  474801  47.446541  19.247501         NaN         NaN
64     1656288010  4a4d79  44.559104  26.003433   64.652329   84.062584
66     1656288010  48e304  54.356825  18.480086         NaN         NaN
...           ...     ...        ...        ...         ...         ...
357168 1656291590  474805  47.439430  19.273734         NaN         NaN
357179 1656291590  48e304  54.356827  18.480074         NaN         NaN
357181 1656291590  425850  53.357712  -2.279526         NaN         NaN
357188 1656291590  474801  47.446541  19.247501         NaN         NaN
357197 1656291590  425851  53.361340  -2.271855         NaN         NaN

        vertrate  callsign  onground  alert    spi  squawk  baroaltitude  \
26           NaN    RU7B13     False  False  False     NaN           NaN
53        3.2512    TFL15D     False  False  False  5507.0      11529.06
57           NaN    TXLU00      True  False  False  5424.0           NaN
64       -3.2512    HYS236     False  False  False  4153.0        342.90
```

```python
indices_a_supprimer = donnees[donnees['Country'] == 'Autre'].index

# Supprimer les lignes correspondantes
donnees = donnees.drop(indices_a_supprimer)

# Enregistrer les données modifiées dans le même fichier CSV ou dans un nouveau fichier
donnees.to_csv('air.csv', index=False)
```

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	time	icao24	lat	lon	velocity	heading	vertrate	callsign	onground	alert	spi	squawk	baroaltitude	geoaltitude	lastposupda	lastcontact	country
2	1656288000	34718e	40.8729401	1.92297363	187.631064	245.543044	8.128	IBE2601	False	False	False	417	5509.26	5814.06	1656287999	1656288000	Spain
3	1656288000	406471	51.9085693	1.83830261	169.170746	286.065061	-0.65024	TOM891	False	False	False	3541	5478.78	5570.22	1656287999	1656288000	United Kingc
4	1656288000	4d21ea	48.9276581	8.00579364	213.518431	159.993305	-0.32512	RYR2MW	False	False	False	2515	11887.2	12291.06	1656287999	1656288000	Malta
5	1656288000	4ca9cc	47.5651703	8.3886795	221.441104	303.726985	0	RYR8KU	False	False	False	6607	11582.4	12031.98	1656287999	1656288000	Ireland
6	1656288010	152019	46.3748932	52.0438441	241.480364	36.7722484	0	AFL2976	False	False	False		10660.38	11087.1	1656288010	1656288010	Russia
7	1656288010	49f0d4						TXLU00	False	False	False					1656288009	Czech Repul
8	1656288010	50840d	51.0500793	7.81187315	135.29975	216.652044	-5.20192	CAI88WY	False	False	False	3202	2491.74	2628.9	1656288010	1656288010	Ukraine
9	1656288010	4cc542	43.1719867	15.9007481	224.227368	13.3985872	0	BBD624	False	False	False	7750	10370.82	10896.6	1656287989	1656288009	Iceland
10	1656288010	343652	40.4615409	-3.5651415				P1	True	False	False				1656288010	1656288010	Spain
11	1656288010	400fe0	48.0287933	-0.756958	245.443707	358.799007	-0.32512	EZY93TB	False	False	False	5364	10980.42	11132.82	1656288010	1656288010	United Kingc
12	1656288010	4bb186	35.7466278	27.8621292	257.029011	193.897176	0	THY44D	False	True	False	2361	10660.38	11163.3	1656288009	1656288010	Switzerland
13	1656288010	471f7c	47.6514293	17.1189528	250.626642	122.254711	0	WZZ93JJ	False	False	False	2270	11285.22	11803.38	1656288010	1656288010	Hungary
14	1656288010	39c425	47.0206604	-72.368523	279.164448	71.2979399	-0.32512	AFR159	False	False	False	2370	11269.98	11750.04	1656288010	1656288010	France
15	1656288010	4.07E+05	41.6637873	-75.162197	210.311981	236.271062	-0.32512	BAW171	False	False	False	724	8534.4	8968.74	1656288010	1656288010	United Kingc
16	1656288010	4ca92b	53.2427673	-5.6814226	140.42154	315.593717	-5.85216	RYR82UG	False	False	False	2764	2171.7	2186.94	1656288009	1656288010	Ireland
17	1656288010	4b5ccd						TEST1234	True	True	False	1543				1656288010	Switzerland
18	1656288010	5.03E+63	44.5710677	12.0728358	224.11167	307.632744	0.32512	CXI3FU	False	False	False	5535	10988.04	11521.44	1656288010	1656288010	Lithuania
19	1656288010	4952ac	43.103648	-80.743365	209.583374	267.467684	0	TAP243	False	False	False	7001	12192	12740.64	1656288010	1656288010	Portugal
20	1656288010	4d0113	42.295166	-100.32571	289.593592	67.2157174	0	CLX43X	False	False	False	5726	10058.4	10485.12	1656288010	1656288010	Luxembourg
21	1656288010	4d224f	39.4151917	12.5425654	239.556968	148.255465	-14.95552	RYR739T	False	False	False	1000	8206.74	8900.16	1656288010	1656288010	Malta
22	1656288010	3.95E+11	42.4101128	-69.637352	258.261648	238.272867	0	AFR016	False	False	False	7364	12192	12748.26	1656288010	1656288010	France
23	1656288010	4ca9cc	47.5760612	8.36425781	219.742272	303.057607	0	RYR8KU	False	False	False	6607	11582.4	12031.98	1656288009	1656288010	Ireland

4. Ingestion des données :

L'ingestion de données représente une étape cruciale dans la collecte, le traitement et l'acheminement des informations. Pour cette phase, nous avons opté pour Apache Kafka, une plateforme distribuée de streaming de données, reconnue pour sa capacité à gérer efficacement de grands volumes de données, garantissant ainsi une ingestion robuste et évolutive.

Fiabilité et Scalabilité :
Apache Kafka offre une architecture distribuée, permettant une ingénierie de données hautement fiable et évolutive. Cette plateforme est conçue pour gérer des flux massifs de données provenant de multiples sources sans perte d'informations, assurant une ingestion fiable et sans interruption.

Parallélisme et Partitions :
Kafka utilise des partitions pour distribuer et paralléliser les données. Cette fonctionnalité assure une ingestion rapide et efficace, en permettant le traitement simultané de plusieurs messages sur différentes partitions, augmentant ainsi les performances globales du système.

Sécurité et Durabilité :
La robustesse de Kafka inclut des mécanismes de sécurité avancés pour protéger les données transitant à travers la plateforme. De plus, la réplication des données et la persistance sur disque garantissent une durabilité élevée, préservant les données même en cas de défaillance matérielle.

-on a fait un script Python pour gérer les flux de données en utilisant Apache Kafka, en se concentrant sur la production (écriture) et la consommation (lecture) de données à partir de deux sources distinctes vers deux sujets Kafka distincts.

Production de Données
Le premier bloc de code est dédié à la production de données dans Kafka. Il lit deux fichiers CSV différents (weather_data.csv et air.csv) et envoie leurs contenus respectifs vers deux sujets Kafka distincts (meteo et vol). Ces données, converties en chaînes de caractères CSV, sont produites et envoyées vers les sujets correspondants à l'aide du producteur Kafka.

```python
from confluent_kafka import Producer
import csv

# Configuration de Kafka
bootstrap_servers = 'localhost:9092'
topic1 = 'meteo'
topic2 = 'vol'

# Configuration du producteur Kafka
conf = {'bootstrap.servers': bootstrap_servers}
producer = Producer(conf)

def produce_messages_from_csv(file_path, topic):
    with open(file_path, newline='') as csvfile:
        reader = csv.reader(csvfile)
        for row in reader:
            producer.produce(topic, value=','.join(row).encode('utf-8'))
            producer.flush()

# Envoyer le contenu du premier fichier CSV au premier topic Kafka
file1_path = 'weather_data.csv'
produce_messages_from_csv(file1_path, topic1)
print("Tous les messages du fichier 1 ont été envoyés avec succès")

# Envoyer le contenu du deuxième fichier CSV au deuxième topic Kafka
file2_path = 'air.csv'
produce_messages_from_csv(file2_path, topic2)
print("Tous les messages du fichier 2 ont été envoyés avec succès")
```

```
Tous les messages du fichier 1 ont été envoyés avec succès
Tous les messages du fichier 2 ont été envoyés avec succès
```

Consommation de Données

Le deuxième bloc de code se concentre sur la consommation de données à partir des sujets Kafka précédemment peuplés. Il utilise un consommateur Kafka pour lire les messages des sujets meteo et vol. Les données sont récupérées sous forme de chaînes CSV, séparées en lignes individuelles et stockées dans des DataFrames pandas.

DataFrame pour les Données Météorologiques (df_meteo) :
Les données météorologiques sont lues à partir du sujet meteo. Le script attend de recevoir 47 lignes de données avant de créer un DataFrame pandas (df_meteo) contenant les informations des données météorologiques.

DataFrame pour les Données de Vol (df_vol) :
Les données de vol sont extraites du sujet vol. Le script collecte 312 272 lignes de données avant de créer un DataFrame pandas (df_vol) pour stocker les informations des données de vol.

```python
from confluent_kafka import Consumer, KafkaError
import pandas as pd
import io

import uuid  # Module pour générer un UUID

# Générez un ID de groupe unique
group_id = 'spring-boot-avro-consum-' + str(uuid.uuid4())

# Configuration Kafka avec l'ID de groupe dynamique
conf = {
    'bootstrap.servers': 'localhost:9092',
    'group.id': group_id,
    'auto.offset.reset': 'earliest'
}

# Créez une instance du consommateur Kafka
consumer = Consumer(conf)

# Abonnez-vous au sujet
topic = 'meteo'  # Changez le sujet au besoin
consumer.subscribe([topic])

# Initialisez une liste pour stocker les lignes CSV
csv_lines = []

# Continuellement récupérer de nouveaux messages
while True:
    msg = consumer.poll(1.0)

    if msg is None:
        continue
    if msg.error():
        if msg.error().code() == KafkaError._PARTITION_EOF:
            print('Reached end of partition')
        else:
            print('Error while consuming: {}'.format(msg.error()))
    else:
        # Décodez le message en une chaîne CSV
        csv_string = msg.value().decode('utf-8')

        # Divisez la chaîne CSV en lignes
        lines = csv_string.split('\n')

        # Ajoutez chaque ligne CSV à la liste
        for line in lines:
            csv_lines.append(line.split(','))

    if len(csv_lines) >= 47:  # Si vous avez collecté suffisamment de lignes
        break

# Créez un DataFrame avec les lignes CSV
df_meteo = pd.DataFrame(csv_lines[1:], columns=csv_lines[0])

# Affichez le DataFrame
print(df_meteo)
# Fermez le consommateur Kafka une fois que vous avez terminé
consumer.close()
```

```
                Country             Description Temperature (K)  \
0               Albania          overcast clouds          303.43
1               Andorra               few clouds           273.6
2               Austria          overcast clouds          272.08
```

```python
from confluent_kafka import Consumer, KafkaError
import pandas as pd
import io

import uuid  # Module pour générer un UUID

# Générez un ID de groupe unique
group_id = 'spring-boot-avro-consum-' + str(uuid.uuid4())

# Configuration Kafka avec l'ID de groupe dynamique
conf = {
    'bootstrap.servers': 'localhost:9092',
    'group.id': group_id,
    'auto.offset.reset': 'earliest'
}

# Créez une instance du consommateur Kafka
consumer = Consumer(conf)

# Abonnez-vous au sujet
topic = 'vol'  # Changez le sujet au besoin
consumer.subscribe([topic])

# Initialisez une liste pour stocker les lignes CSV
csv_lines = []

# Continuellement récupérer de nouveaux messages
while True:
    msg = consumer.poll(1.0)

    if msg is None:
        continue
    if msg.error():
        if msg.error().code() == KafkaError._PARTITION_EOF:
            print('Reached end of partition')
        else:
            print('Error while consuming: {}'.format(msg.error()))
    else:
        # Décodez le message en une chaine CSV
        csv_string = msg.value().decode('utf-8')

        # Divisez la chaine CSV en lignes
        lines = csv_string.split('\n')

        # Ajoutez chaque ligne CSV à la liste
        for line in lines:
            csv_lines.append(line.split(','))

    if len(csv_lines) >= 312272:  # Si vous avez collecté suffisamment de lignes
        break

# Créez un DataFrame avec les lignes CSV
df_vol = pd.DataFrame(csv_lines[1:], columns=csv_lines[0])

# Affichez le DataFrame
print(df_vol)
# Fermez le consommateur Kafka une fois que vous avez terminé
consumer.close()
```

```
            time  icao24               lat               lon  \
0     1656288000  34718e   40.87294006347656     1.9229736328125
1     1656288000  406471    51.9085693359375  1.8383026123046875
```

En complément à l'extraction des données de vol à partir du sujet Kafka dédié, une analyse supplémentaire a été réalisée pour identifier le nombre de vols par pays.

À l'aide de la bibliothèque Pandas, les données de vol extraites ont été regroupées par pays, comptabilisant le nombre de vols pour chaque pays. Cette opération a été exécutée en utilisant la méthode groupby() sur le DataFrame df_vol, permettant ainsi d'obtenir une agrégation du nombre de vols par pays.

Le résultat de cette agrégation est stocké dans un nouveau DataFrame nommé opensky_df, où chaque ligne correspond à un pays et affiche le nombre total de vols enregistrés pour ce pays. Ce DataFrame a été trié par ordre décroissant en fonction du nombre de vols, permettant ainsi de mettre en évidence les pays ayant enregistré le plus grand nombre de vols dans l'espace aérien analysé.

```
opensky_df = df_vol.groupby('country').size().reset_index(name='num_flights')
opensky_df = opensky_df.sort_values(by='num_flights', ascending=False)
print(opensky_df)
```

```
                      country  num_flights
32             United Kingdom        54578
30                Switzerland        51428
10                    Germany        33106
25                     Russia        28593
14                    Ireland        24169
28                      Spain        19078
9                      France        11277
5              Czech Republic         9037
20                      Malta         8518
12                    Hungary         7740
24                   Portugal         7338
6                     Denmark         7018
23                     Poland         6965
16  Kingdom of the Netherlands         6041
1                     Austria         5240
13                    Iceland         4677
19                 Luxembourg         4274
8                     Finland         3881
29                     Sweden         3465
7                     Estonia         2368
11                     Greece         2110
27                   Slovakia         1938
18                  Lithuania         1934
2                     Belarus         1352
...
17                     Latvia          516
22                     Norway          244
21                    Moldova          188
0                     Albania           31
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

5. Traitement de données : (PySpark)

- **Première analyse : analyse des Données avec Spark ML : Régression Linéaire pour Corrélation entre Données Météorologiques et Vols :**

La régression linéaire est une technique statistique utilisée pour modéliser la relation entre une variable dépendante continue et une ou plusieurs variables indépendantes. Spark ML, le module de machine learning de Spark, offre des outils pour exécuter des régressions linéaires sur de larges ensembles de données distribuées.

Dans ce contexte, nous appliquons la régression linéaire pour identifier la corrélation entre les données météorologiques et le nombre de vols en Europe. L'objectif est d'évaluer comment les variations des conditions météorologiques, telles que la température, la pression atmosphérique, l'humidité et la vitesse du vent, peuvent influencer le nombre de vols aériens dans la région.

- Le script commence par l'initialisation d'une session Spark, suivie du chargement des données météorologiques et des données de vol dans des DataFrames Spark distincts.

Ensuite, ces DataFrames sont joints en fonction de la colonne "country" pour créer un DataFrame consolidé nommé EuropeWeather_data.

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("myApp").getOrCreate()


# Chargement des données météorologiques dans un DataFrame Spark
openweather_data = spark.createDataFrame(df_meteo)


# Chargement des données de vol dans un DataFrame Spark
opensky_data = spark.createDataFrame(opensky_df)


# Jointure des deux DataFrames sur la colonne "country"
EuropeWeather_data= opensky_data.join(openweather_data, 'country')


# Partitionnement en 100 partitions
EuropeWeather_data = EuropeWeather_data.repartition(100)
```

Les données sont ensuite préparées pour la régression linéaire. Cela implique la conversion des colonnes de données en types numériques adéquats et la création d'un vecteur contenant les variables indépendantes à l'aide de la fonction VectorAssembler de Spark ML.

```
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression


# Création d'un vecteur contenant les variables indépendantes
assembler = VectorAssembler(
    inputCols=["Temperature (K)", "Pressure (hPa)", "Humidity (%)", "Wind Speed (m/s)"],
    outputCol="features")


from pyspark.sql.functions import col

EuropeWeather_data = EuropeWeather_data.withColumn("Temperature (K)", col("Temperature (K)").cast("float"))
EuropeWeather_data = EuropeWeather_data.withColumn("Pressure (hPa)", col("Pressure (hPa)").cast("float"))
EuropeWeather_data = EuropeWeather_data.withColumn("Humidity (%)", col("Humidity (%)").cast("float"))
EuropeWeather_data = EuropeWeather_data.withColumn("Wind Speed (m/s)", col("Wind Speed (m/s)").cast("float"))


# Transformation des données pour inclure le vecteur de variables indépendantes
transformed_data = assembler.transform(EuropeWeather_data)
```

Le jeu de données est divisé en ensembles d'entraînement et de test, puis un modèle de régression linéaire est entraîné sur l'ensemble d'apprentissage. Le modèle est évalué sur l'ensemble de test pour mesurer ses performances.

```
# Transformation des données pour inclure le vecteur de variables indépendantes
transformed_data = assembler.transform(EuropeWeather_data)
```

```
# Division des données en ensembles de formation et de test
(training_data, test_data) = transformed_data.randomSplit([0.7, 0.3])
```

```
# Entraînement du modèle de régression linéaire
lr = LinearRegression(featuresCol="features", labelCol="num_flights")
model = lr.fit(training_data)
```

```
# Évaluation du modèle sur l'ensemble de test
predictions = model.transform(test_data)
predictions.show()
```

```
+--------+-----------+---------------+---------------+-------------+------------+---------------+--------------------+-------------------+
| country|num_flights|    Description|Temperature (K)|Pressure (hPa)|Humidity (%)|Wind Speed (m/s)|            features|         prediction|
+--------+-----------+---------------+---------------+-------------+------------+---------------+--------------------+-------------------+
| Ireland|      24169|overcast clouds|         282.98|        980.0|        91.0|           8.33|[282.98001098632...|  26656.53242850257|
| Moldova|        188|overcast clouds|         277.14|       1007.0|        79.0|            2.9|[277.14001464843...| 18095.588088053744|
|  Greece|       2110|overcast clouds|         270.97|       1020.0|        84.0|           0.45|[270.97000122070...|  17196.39799945487|
|Bulgaria|       1240|     few clouds|          279.1|       1012.0|        70.0|           6.71|[279.10000610351...| 1094.8306969355326|
|  Norway|        244|      clear sky|         265.89|       1016.0|        63.0|          13.89|[265.89001464843...|-18138.466874960344|
|   Italy|       1036|overcast clouds|         279.47|       1014.0|        88.0|           0.89|[279.47000122070...| 18860.108741408098|
| Iceland|       4677|overcast clouds|         270.61|        988.0|        96.0|          11.14|[270.60998535156...|  18149.80440982897|
| Estonia|       2368|      clear sky|         261.13|       1020.0|        71.0|           4.38|[261.13000488281...|  7322.962616451434|
|  Poland|       6965|overcast clouds|         277.82|       1007.0|        88.0|           4.92|[277.82000732421...|  14542.72268251516|
| Belarus|       1352|overcast clouds|         270.86|       1008.0|        75.0|           5.83|[270.85998535156...| 10827.360527944402|
|  Latvia|        516|  broken clouds|         263.97|       1015.0|        58.0|           4.44|[263.97000122070...|  7317.976039196947|
+--------+-----------+---------------+---------------+-------------+------------+---------------+--------------------+-------------------+
```

Enfin, le modèle entraîné est utilisé pour prédire le nombre de vols en fonction de nouvelles données météorologiques simulées.

```
# Prédiction du nombre de vols en fonction des conditions météorologiques
new_data = spark.createDataFrame([
    (10.0, 1010.0, 80.0, 5.0),
    (20.0, 1000.0, 70.0, 10.0),
    (30.0, 990.0, 60.0, 15.0)
], ["Temperature (K)", "Pressure (hPa)", "Humidity (%)", "Wind Speed (m/s)"])

new_transformed_data = assembler.transform(new_data)
predictions = model.transform(new_transformed_data)
predictions.show()
```

```
+---------------+--------------+------------+---------------+--------------------+------------------+
|Temperature (K)|Pressure (hPa)|Humidity (%)|Wind Speed (m/s)|            features|        prediction|
+---------------+--------------+------------+---------------+--------------------+------------------+
|           10.0|        1010.0|        80.0|            5.0|[10.0,1010.0,80.0...|105876.39385384449|
|           20.0|        1000.0|        70.0|           10.0|[20.0,1000.0,70.0...| 95015.77269047697|
|           30.0|         990.0|        60.0|           15.0|[30.0,990.0,60.0,...| 84155.15152710956|
+---------------+--------------+------------+---------------+--------------------+------------------+
```

- **Deuxième analyse: nous allons utiliser les données de vol et les données météorologiques pour identifier les pays européens qui sont les plus touchés par les conditions météorologiques défavorables.**

L'indice de conditions météorologiques défavorables est un métrique composite utilisé pour évaluer l'impact des conditions météorologiques sur les vols aériens. Ce calcul combine plusieurs variables météorologiques telles que la température, la pression atmosphérique,

l'humidité et la vitesse du vent pour estimer le niveau de conditions météorologiques adverses dans une région donnée.

- Nous avons utilisé Spark pour réaliser plusieurs calculs visant à évaluer les conditions météorologiques défavorables par pays. Tout d'abord, nous avons regroupé les données par pays pour calculer le nombre total de vols dans chaque pays. Ensuite, nous avons calculé la moyenne des variables météorologiques pertinentes pour chaque pays.

```
from pyspark.sql.functions import sum
```

```
# Calcul du nombre de vols pour chaque pays
flight_counts = EuropeWeather_data.groupBy("country").agg(sum("num_flights").alias("total_flights"))
```

```
from pyspark.sql.functions import avg
```

```
# Calcul de la moyenne de chaque variable météorologique pour chaque pays
weather_averages = EuropeWeather_data.groupBy("country").agg(
    avg("Temperature (K)").alias("avg_temperature"),
    avg("Pressure (hPa)").alias("avg_pressure"),
    avg("Humidity (%)").alias("avg_humidity"),
    avg("Wind Speed (m/s)").alias("avg_wind_speed"))
```

En combinant ces deux résultats, nous avons créé un DataFrame nommé EuropeWeather_data. À l'aide des valeurs moyennes des variables météorologiques et du nombre total de vols, nous avons calculé l'indice de conditions météorologiques défavorables pour chaque pays.

```
# Jointure des deux DataFrames
EuropeWeather_data = flight_counts.join(weather_averages, "country")
```

```
#Plus la valeur de l'indice est élevée, plus le pays est susceptible d'être touché par des conditions météorologiques défavorables :
from pyspark.sql.functions import col
```

```
# Calcul de l'indice de conditions météorologiques défavorables pour chaque pays
bad_weather_index = EuropeWeather_data.select(
    "country",
    ((col("avg_temperature") * col("avg_pressure") * (100 - col("avg_humidity")) * col("avg_wind_speed")) / col("total_flights")).alias("bad_weather_index")
)

# Affichage des 10 premiers pays les plus touchés par des conditions météorologiques défavorables
#Cela nous donne une liste des 10 premiers pays les plus touchés par des conditions météorologiques défavorables.
#Nous pouvons utiliser ces résultats pour identifier
#les pays qui ont besoin de mesures spéciales pour faire face à ces conditions météorologiques.
bad_weather_index.orderBy("bad_weather_index", ascending=False).show(10)
```

```
+---------+------------------+
|  country| bad_weather_index|
+---------+------------------+
|   Norway| 568996.8866538958|
|  Albania| 359452.9539939405|
|   Latvia|  96828.4948816966|
|  Moldova| 90404.10765559821|
| Bulgaria| 45852.43994082162|
|Lithuania| 39332.90554420793|
|  Belarus|29433.184236059715|
|  Ukraine|19036.399094394703|
|  Estonia|14287.208094842317|
| Slovakia| 9892.180450888856|
+---------+------------------+
only showing top 10 rows
```

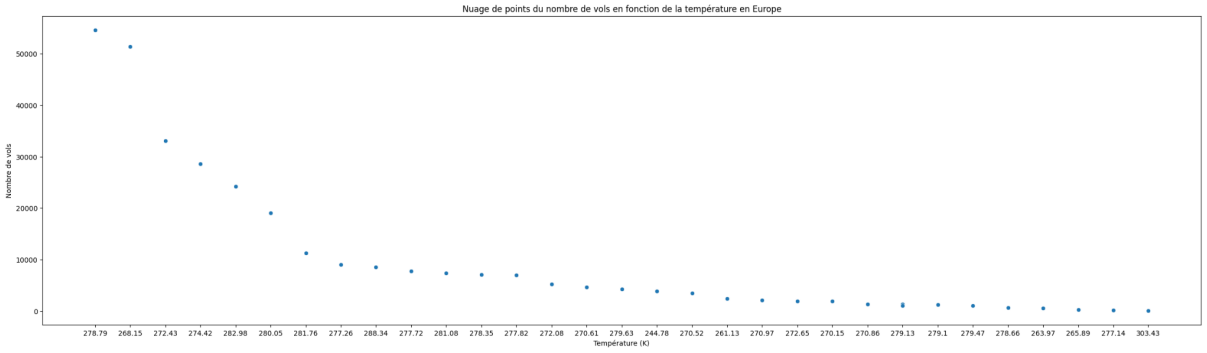
Nous affichons les 10 premiers pays les plus touchés par des conditions météorologiques défavorables. Ces informations sont précieuses pour prioriser les mesures à prendre face à ces conditions. Elles nous aident à cibler les régions nécessitant des précautions ou des ajustements spécifiques pour garantir la sécurité et l'efficacité des opérations aériennes dans des conditions météorologiques difficiles.

Cette analyse proactive nous permet d'identifier les régions où les conditions météorologiques ont le plus grand impact sur les vols, offrant ainsi des indications précieuses pour la gestion opérationnelle et la planification des compagnies aériennes et des autorités aéroportuaires.

6. Visualisation :

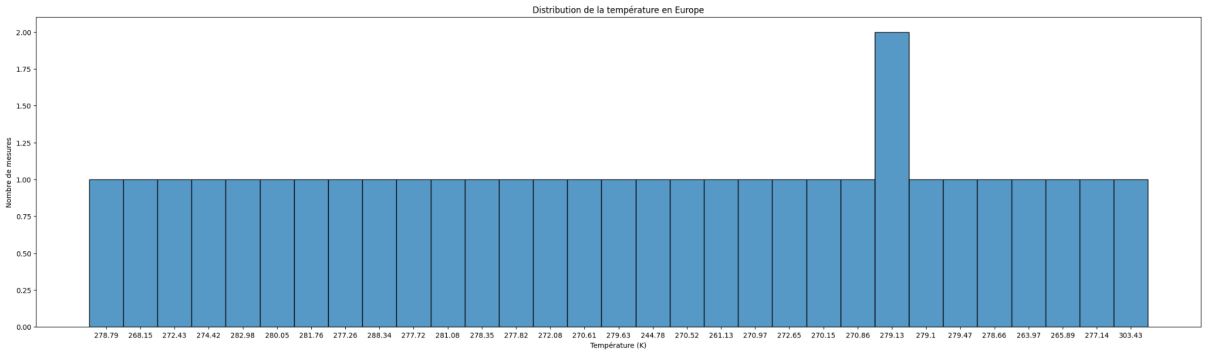
- **Nuage de Points : Nombre de Vols en Fonction de la Température**

Nous avons combiné les données sur le trafic aérien et les données météorologiques pour représenter graphiquement la relation entre la température et le nombre de vols en Europe. Ce nuage de points nous permet de visualiser comment la température influe sur l'activité aérienne dans différentes régions européennes.



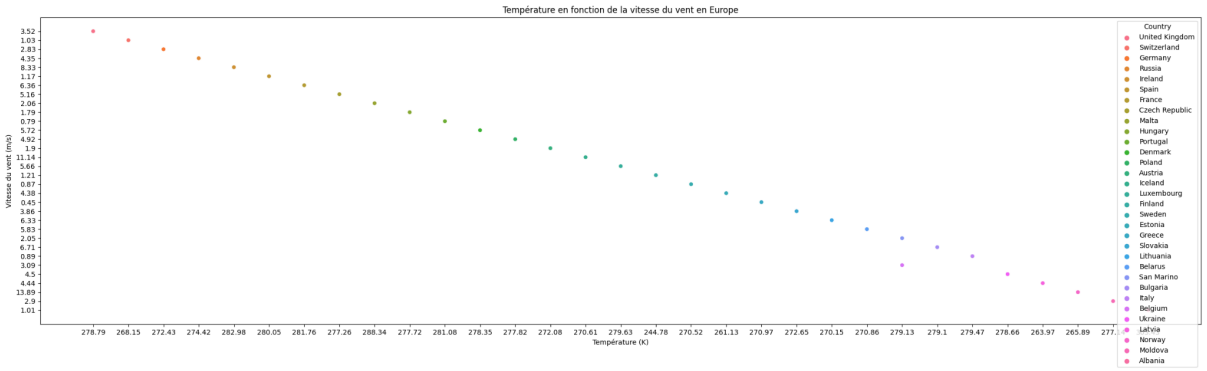
- **Histogramme : Distribution de la Température en Europe**

L'histogramme illustre la répartition des températures en Europe, fournissant une vue d'ensemble des variations de température dans les différentes régions étudiées. Cela nous aide à comprendre la diversité des conditions météorologiques dans les pays européens et leur impact potentiel sur le trafic aérien.



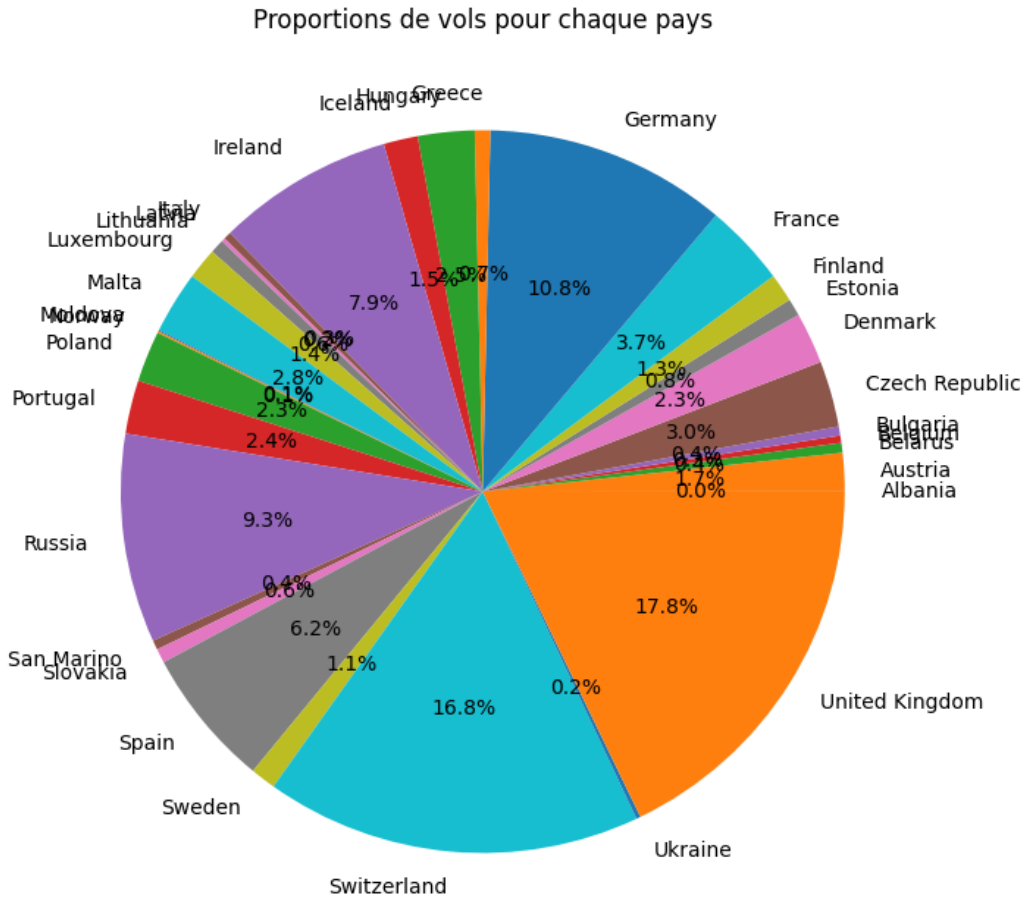
- **Nuage de Points : Température en Fonction de la Vitesse du Vent en Europe**

Ce graphique présente la relation entre la température et la vitesse du vent dans divers pays européens. En affichant ces variables météorologiques conjointement, nous pouvons observer comment la vitesse du vent varie en fonction de la température dans les différentes régions, offrant des insights sur les conditions météorologiques locales.



- **Diagramme Circulaire : Proportions de Vols pour Chaque Pays**

Ce diagramme circulaire met en évidence les proportions de vols pour chaque pays en Europe. En visualisant les proportions relatives, il offre une vue d'ensemble des contributions de chaque pays au trafic aérien global, soulignant les différences d'activité aérienne entre les nations européennes.



7. Conclusion :

La corrélation entre les conditions météorologiques et le trafic aérien est un domaine essentiel pour garantir la sécurité et l'efficacité des opérations aériennes. Notre étude approfondie de cette relation en Europe a révélé plusieurs aspects cruciaux :

Influence des Conditions Météorologiques sur les Opérations Aériennes
Nous avons constaté que les conditions météorologiques, telles que les températures extrêmes, les vents forts et les phénomènes atmosphériques, ont un impact significatif sur le nombre de vols. Les variations météorologiques ont des répercussions directes sur la sécurité, la régularité et la planification des vols.

Identification des Pays les Plus Touchés par les Conditions Défavorables
Grâce à notre analyse approfondie, nous avons pu déterminer les pays européens les plus impactés par des conditions météorologiques défavorables. Cette information est cruciale pour les autorités aéroportuaires et les compagnies aériennes, leur permettant de mettre en place des mesures spécifiques pour atténuer les effets néfastes de ces conditions sur les opérations aériennes.

Utilisation des Données et des Outils d'Analyse
L'intégration de données massives provenant de sources météorologiques et du trafic aérien a été essentielle pour réaliser une analyse approfondie. L'utilisation d'outils avancés tels que Apache Kafka pour l'ingestion des données, Spark ML pour les analyses prédictives et les visualisations diverses a été cruciale pour obtenir des insights pertinents.

Prise de Décisions Informatives pour l'Optimisation
Les résultats de notre analyse offrent des informations clés pour optimiser les opérations aériennes. En comprenant mieux l'impact des conditions météorologiques, les compagnies aériennes et les autorités aéroportuaires peuvent prendre des décisions éclairées pour améliorer la sécurité, la planification des vols et l'efficacité opérationnelle.

En somme, cette étude souligne l'importance vitale de comprendre la relation entre la météo et le trafic aérien. Ces connaissances permettent de développer des stratégies préventives et adaptatives pour garantir la sûreté des vols et maintenir des opérations aériennes fluides, même dans des conditions météorologiques difficiles.