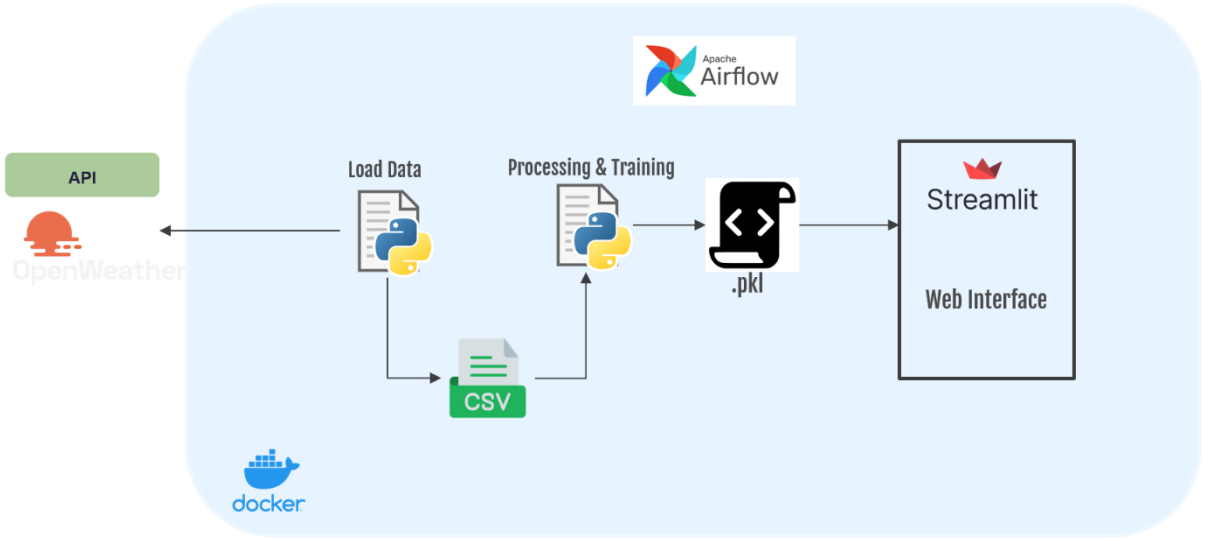
Université Chouaib Doukkali
Ecole Nationale des Sciences Appliquées d'El Jadida
Département Télécommunications, Réseaux et Informatique

Projet3 :

Mise en place d'un pipeline d'entrainement continu en utilisant Apache Airflow et Streamlit

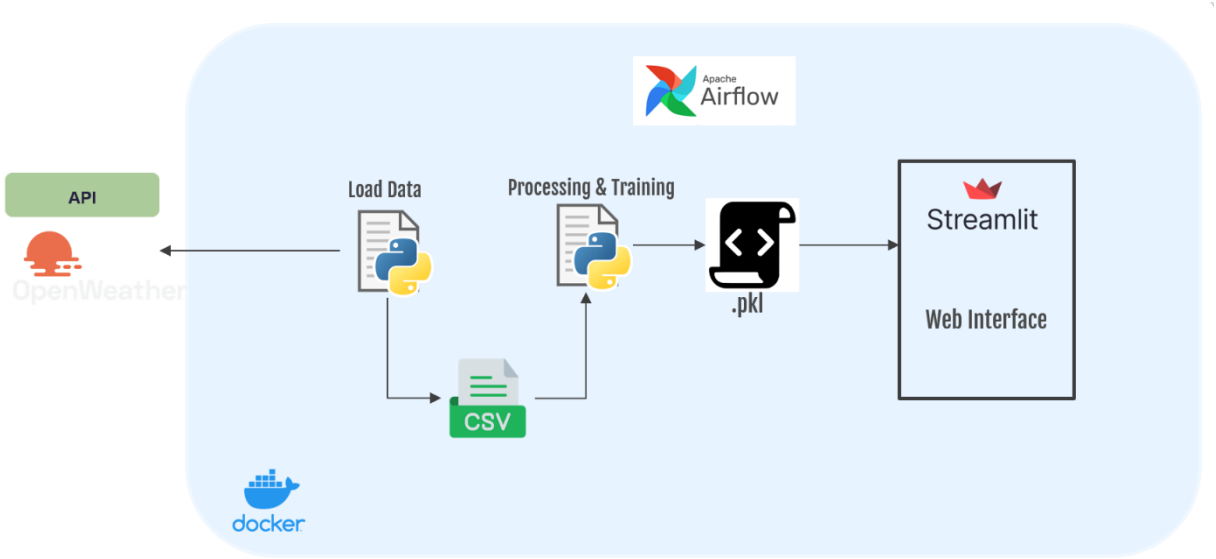


Professeur : F.KALLOUBI
Réalisé par : O.OUHAYOU

Table des matières

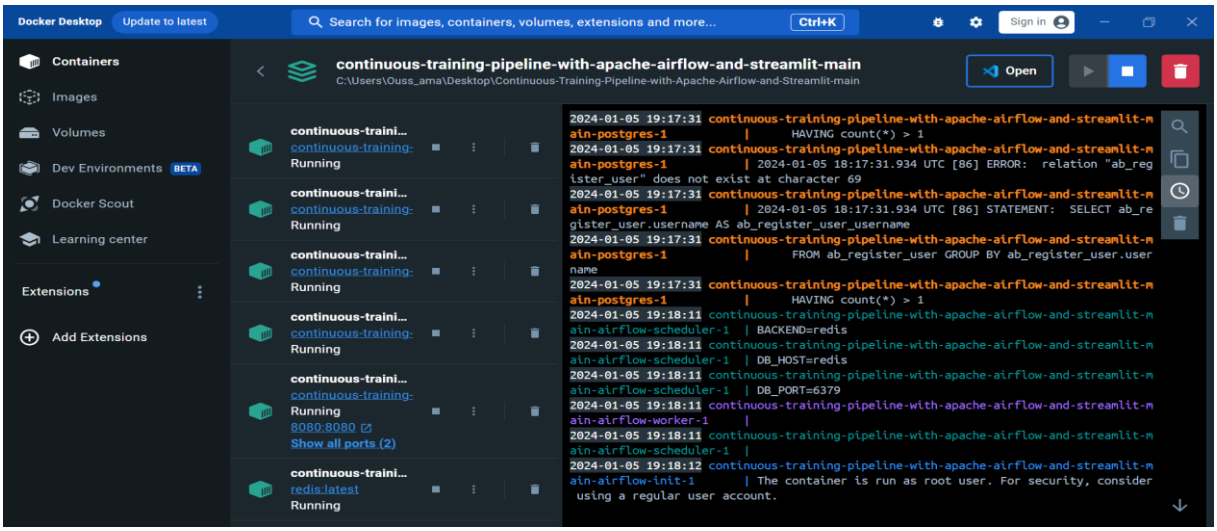
1) Objectif du projet :

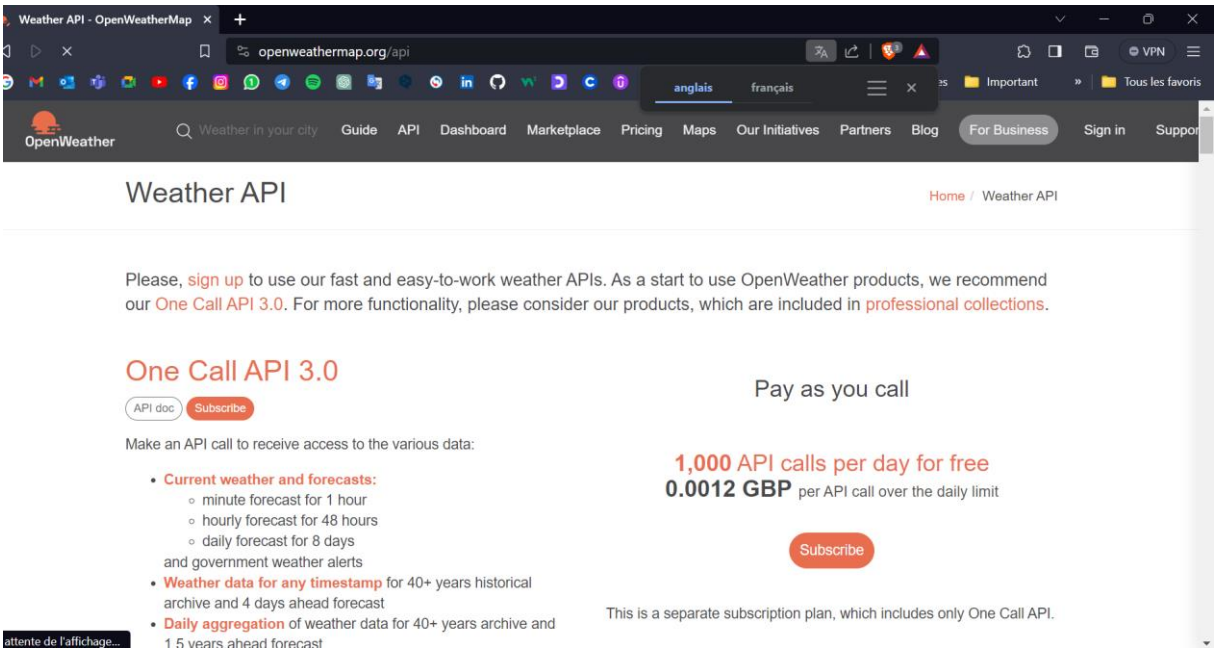
L'objectif de ce projet est de mettre en place un pipeline d'entraînement continu en utilisant Apache Airflow et Streamlit ce projet couvre plusieurs aspects liés à la collecte de données météorologiques via l'API **OpenWeatherMap** à l'application des techniques de prétraitement sur les données collectées (nettoyage, gestion des valeurs manquantes, conversion de types).
à l'entraînement de modèles prédictifs sur ces données, à la mise en oeuvre d'une application web pour consommer le modèle, et à l'automatisation du processus à l'aide d'Apache Airflow.
L'objectif global de ce projet est d'introduire et de mettre en pratique plusieurs concepts et compétences.

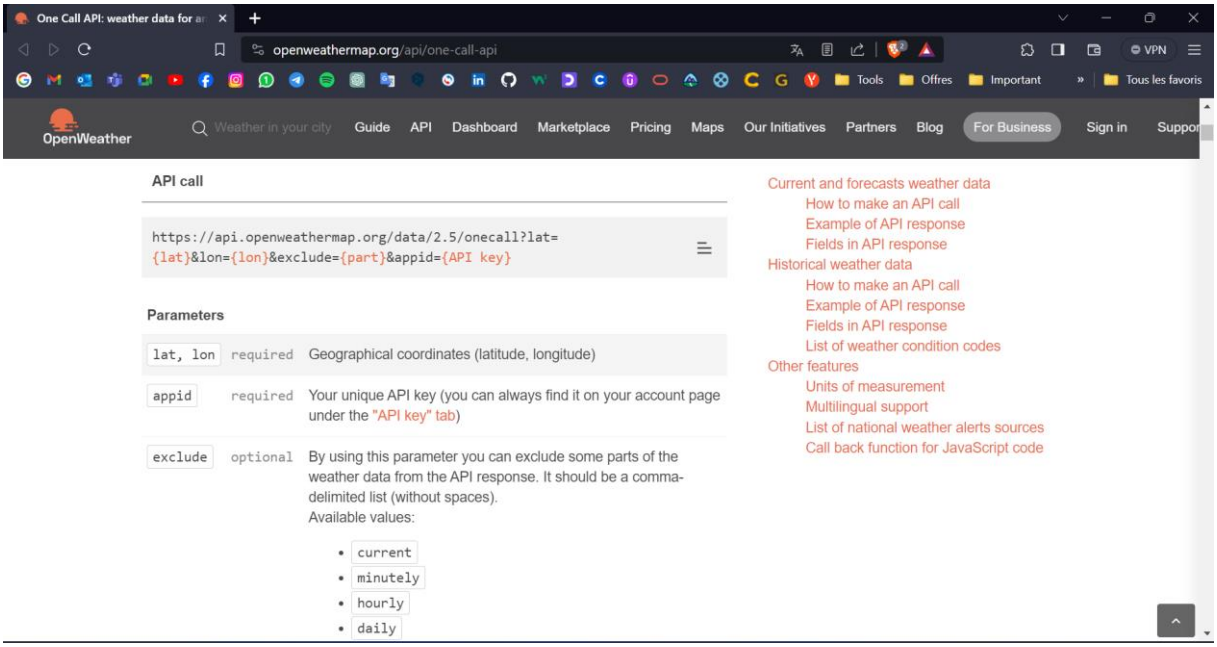


2) Installation et configuration :

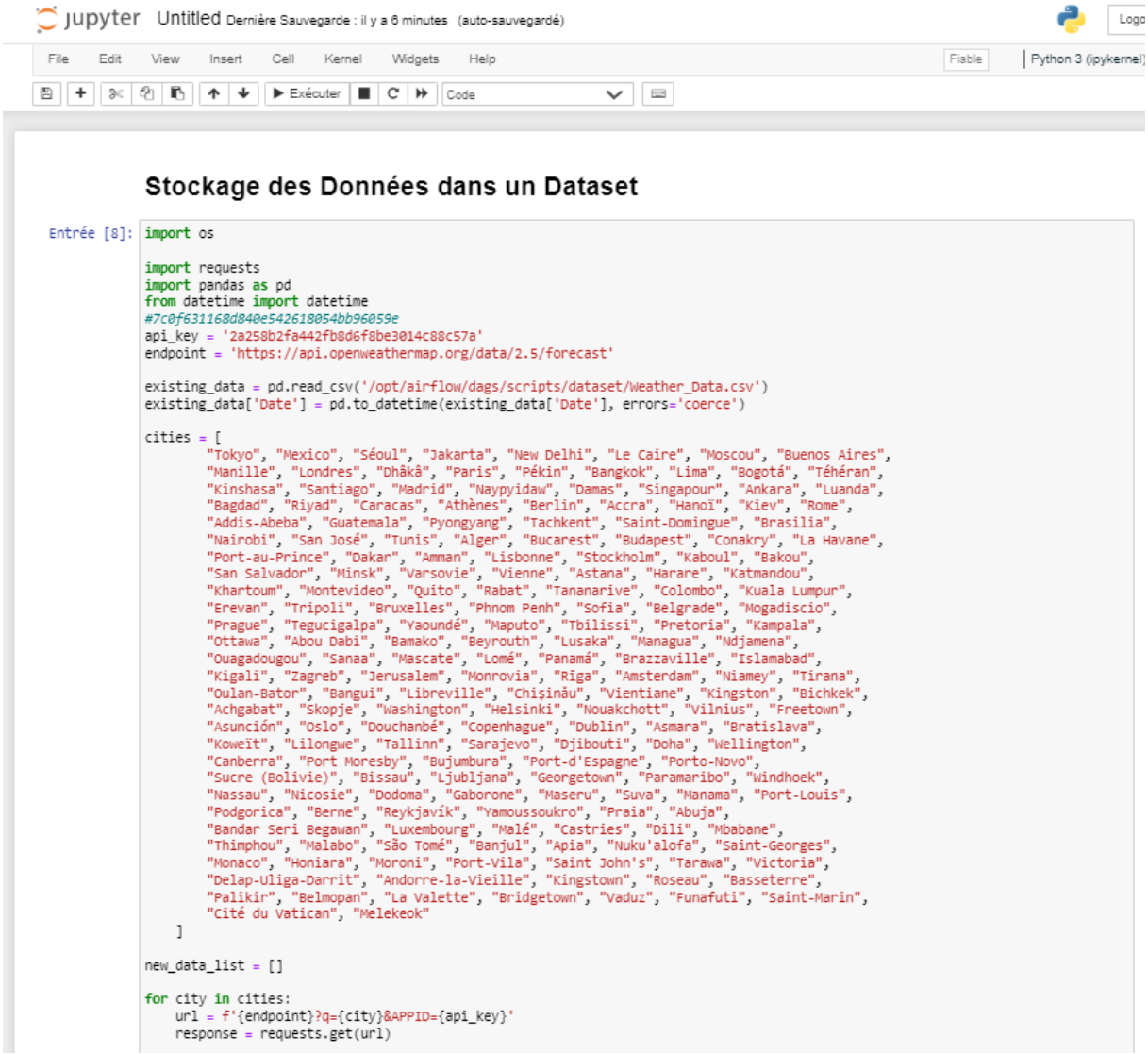
```
C:\Windows\System32\cmd.e
Microsoft Windows [version 10.0.22621.2861]
(c) Microsoft Corporation. Tous droits réservés.

C:\Users\Ouss_ama\Desktop\Continuous-Training-Pipeline-with-Apache-Airflow-and-Streamlit-main>docker-compose up -d
[+] Building 1.8s (8/8)                                                                    docker:default
 => [airflow-init internal] load .dockerignore                                                        0.1s
 => => transferring context: 2B                                                                       0.0s
 => [airflow-init internal] load build definition from Dockerfile                                     0.1s
[+] Building 2.1s (8/8)                                                                    docker:default
 => [airflow-init internal] load .dockerignore                                                        0.1s
[+] Building 3.5s (26/32)                                                                  docker:default
 => [airflow-init internal] load .dockerignore                                                        0.1s
 => => transferring context: 2B                                                                       0.0s
 => [airflow-init internal] load build definition from Dockerfile                                     0.1s
 => => transferring dockerfile: 121B                                                                  0.0s
 => [airflow-scheduler internal] load metadata for docker.io/apache/airflow:2.6.1                     1.8s
 => [airflow-init internal] load build context                                                        0.0s
 => => transferring context: 37B                                                                      0.0s
 => [airflow-webserver 1/3] FROM docker.io/apache/airflow:2.6.1@sha256:213aa1ed99052d9db4006c6813b3ffdbd0b529557f  0.0s
 => CACHED [airflow-init 2/3] ADD requirements.txt .                                                  0.0se
 => CACHED [airflow-init 3/3] RUN pip install -r requirements.txt                                     0.0s
 => [airflow-init] exporting to image                                                                 0.0se
 => => exporting layers                                                                               0.0s
 => => writing image sha256:62ba7737a42cb26b3cff8bb50051a6d83e5eaa10533c2df432fdb009f2ed65d1          0.0se
```


Docker Desktop
Update to latest
Search for images, containers, volumes, extensions and more...
Ctrl+K
Sign in
Containers
Images
Volumes
Dev Environments
BETA
Docker Scout
Learning center
Extensions
Add Extensions
continuous-training-pipeline-with-apache-airflow-and-streamlit-main
C:\Users\Ouss_ama\Desktop\Continuous-Training-Pipeline-with-Apache-Airflow-and-Streamlit-main
Open
continuous-traini...
continuous-training-
Running
8080:8080
Show all ports (2)
redis:latest
2024-01-05 19:17:31 continuous-training-pipeline-with-apache-airflow-and-streamlit-main-postgres-1 | HAVING count(*) > 1
2024-01-05 19:17:31 continuous-training-pipeline-with-apache-airflow-and-streamlit-main-postgres-1 | 2024-01-05 18:17:31.934 UTC [86] ERROR: relation "ab_register_user" does not exist at character 69
2024-01-05 19:17:31 continuous-training-pipeline-with-apache-airflow-and-streamlit-main-postgres-1 | 2024-01-05 18:17:31.934 UTC [86] STATEMENT: SELECT ab_register_user.username AS ab_register_user_username
2024-01-05 19:17:31 continuous-training-pipeline-with-apache-airflow-and-streamlit-main-postgres-1 | FROM ab_register_user GROUP BY ab_register_user.username
2024-01-05 19:17:31 continuous-training-pipeline-with-apache-airflow-and-streamlit-main-postgres-1 | HAVING count(*) > 1
2024-01-05 19:18:11 continuous-training-pipeline-with-apache-airflow-and-streamlit-main-airflow-scheduler-1 | BACKEND=redis
2024-01-05 19:18:11 continuous-training-pipeline-with-apache-airflow-and-streamlit-main-airflow-scheduler-1 | DB_HOST=redis
2024-01-05 19:18:11 continuous-training-pipeline-with-apache-airflow-and-streamlit-main-airflow-scheduler-1 | DB_PORT=6379
2024-01-05 19:18:11 continuous-training-pipeline-with-apache-airflow-and-streamlit-main-airflow-worker-1 |
2024-01-05 19:18:11 continuous-training-pipeline-with-apache-airflow-and-streamlit-main-airflow-scheduler-1 |
2024-01-05 19:18:12 continuous-training-pipeline-with-apache-airflow-and-streamlit-main-airflow-init-1 | The container is run as root user. For security, consider using a regular user account.


Weather API - OpenWeatherMap
openweathermap.org/api
anglais
français
OpenWeather
Weather in your city
Guide
API
Dashboard
Marketplace
Pricing
Maps
Our Initiatives
Partners
Blog
For Business
Sign in
Weather API
Home / Weather API
Please, sign up to use our fast and easy-to-work weather APIs. As a start to use OpenWeather products, we recommend our One Call API 3.0. For more functionality, please consider our products, which are included in professional collections.
One Call API 3.0
API doc
Subscribe
Make an API call to receive access to the various data:
Current weather and forecasts:
minute forecast for 1 hour
hourly forecast for 48 hours
daily forecast for 8 days
and government weather alerts
Weather data for any timestamp for 40+ years historical archive and 4 days ahead forecast
Daily aggregation of weather data for 40+ years archive and 1.5 years ahead forecast
Pay as you call
1,000 API calls per day for free
0.0012 GBP per API call over the daily limit
Subscribe
This is a separate subscription plan, which includes only One Call API.

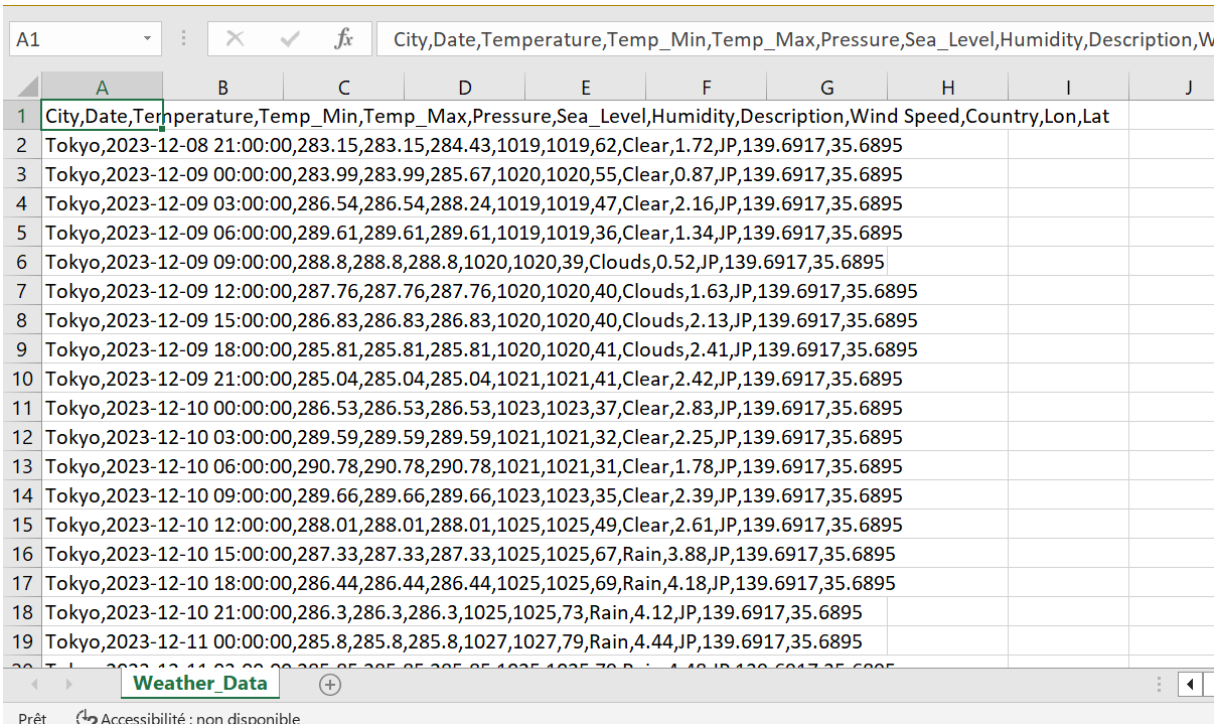

One Call API: weather data for
openweathermap.org/api/one-call-api
OpenWeather
Weather in your city
Guide
API
Dashboard
Marketplace
Pricing
Maps
Our Initiatives
Partners
Blog
For Business
Sign in
API call
https://api.openweathermap.org/data/2.5/onecall?lat={lat}&lon={lon}&exclude={part}&appid={API key}
Parameters
lat, lon required Geographical coordinates (latitude, longitude)
appid required Your unique API key (you can always find it on your account page under the "API key" tab)
exclude optional By using this parameter you can exclude some parts of the weather data from the API response. It should be a comma-delimited list (without spaces).
Available values:
current
minutely
hourly
daily
Current and forecasts weather data
How to make an API call
Example of API response
Fields in API response
Historical weather data
How to make an API call
Example of API response
Fields in API response
List of weather condition codes
Other features
Units of measurement
Multilingual support
List of national weather alerts sources
Call back function for JavaScript code

3) Stockage des Données dans un Dataset:



```
        "Palikir", "Belmopan", "La Valette", "Bridgetown", "Vaduz", "Funafuti", "Saint-Marin",
        "Cité du Vatican", "Melekeok"
    ]

new_data_list = []

for city in cities:
    url = f'{endpoint}?q={city}&APPID={api_key}'
    response = requests.get(url)

    if response.status_code == 200:
            weather_data = response.json()

            if 'list' in weather_data:
                city_forecast = weather_data['list']

                for forecast in city_forecast:
                    timestamp = forecast['dt']
                    date = datetime.utcfromtimestamp(timestamp).strftime('%Y-%m-%d %H:%M:%S')

                    forecast_date = pd.to_datetime(date)

                    if forecast_date > existing_data['Date'].max():
                        city_info = {
                            'City': city,
                            'Date': date,
                            'Temperature': forecast['main']['temp'],
                            'Temp_Min': forecast['main']['temp_min'],
                            'Temp_Max': forecast['main']['temp_max'],
                            'Pressure': forecast['main']['pressure'],
                            'Sea_Level': forecast['main']['sea_level'] if 'sea_level' in forecast['main'] else None,
                            'Humidity': forecast['main']['humidity'],
                            'Description': forecast['weather'][0]['main'],
                            'Wind Speed': forecast['wind']['speed'],
                            'Country': weather_data['city']['country'],
                            'Lon': weather_data['city']['coord']['lon'],
                            'Lat': weather_data['city']['coord']['lat'],
                        }
                        new_data_list.append(city_info)
            else:
                print(f"No forecast data for {city}")
    else:
        print(f"Error {city}: {response.status_code}")
```

```
        "Asuncion", "Oslo", "Douchanbe", "Copenhague", "Dublin", "Asmara", "Bratislava",
        "Koweït", "Lilongwe", "Tallinn", "Sarajevo", "Djibouti", "Doha", "Wellington",
        "Canberra", "Port Moresby", "Bujumbura", "Port-d'Espagne", "Porto-Novo",
        "Sucre (Bolivie)", "Bissau", "Ljubljana", "Georgetown", "Paramaribo", "Windhoek",
        "Nassau", "Nicosie", "Dodoma", "Gaborone", "Maseru", "Suva", "Manama", "Port-Louis",
        "Podgorica", "Berne", "Reykjavík", "Yamoussoukro", "Praia", "Abuja",
        "Bandar Seri Begawan", "Luxembourg", "Malé", "Castries", "Dili", "Mbabane",
        "Thimphou", "Malabo", "São Tomé", "Banjul", "Apia", "Nuku'alofa", "Saint-Georges",
        "Monaco", "Honiara", "Moroni", "Port-Vila", "Saint John's", "Tarawa", "Victoria",
        "Delap-Uliga-Darrit", "Andorre-la-Vieille", "Kingstown", "Roseau", "Basseterre",
        "Palikir", "Belmopan", "La Valette", "Bridgetown", "Vaduz", "Funafuti", "Saint-Marin",
        "Cité du Vatican", "Melekeok"
    ]

new_data_list = []

for city in cities:
    url = f'{endpoint}?q={city}&APPID={api_key}'
    response = requests.get(url)

    if response.status_code == 200:
            weather_data = response.json()

            if 'list' in weather_data:
                city_forecast = weather_data['list']

                for forecast in city_forecast:
                    timestamp = forecast['dt']
                    date = datetime.utcfromtimestamp(timestamp).strftime('%Y-%m-%d %H:%M:%S')

                    forecast_date = pd.to_datetime(date)

                    if forecast_date > existing_data['Date'].max():
                        city_info = {
                            'City': city,
                            'Date': date,
                            'Temperature': forecast['main']['temp'],
                            'Temp_Min': forecast['main']['temp_min'],
                            'Temp_Max': forecast['main']['temp_max'],
                            'Pressure': forecast['main']['pressure'],
                            'Sea_Level': forecast['main']['sea_level'] if 'sea_level' in forecast['main'] else None,
                            'Humidity': forecast['main']['humidity'],
                            'Description': forecast['weather'][0]['main'],
                            'Wind Speed': forecast['wind']['speed'],
                            'Country': weather_data['city']['country'],
                            'Lon': weather_data['city']['coord']['lon'],
                            'Lat': weather_data['city']['coord']['lat'],
                        }
                        new_data_list.append(city_info)
            else:
                print(f"No forecast data for {city}")
    else:
        print(f"Error {city}: {response.status_code}")

    new_data_df = pd.DataFrame(new_data_list)

    if not new_data_df.empty:
        new_data_df.to_csv(os.path.abspath('/opt/airflow/dags/scripts/dataset/Weather_Data.csv'), index=False, mode='a', header=
    else:
        print("No new data to add.")
```



	A	B	C	D	E	F	G	H	I	J
1	City,Date,Temperature,Temp_Min,Temp_Max,Pressure,Sea_Level,Humidity,Description,Wind Speed,Country,Lon,Lat									
2	Tokyo,2023-12-08 21:00:00,283.15,283.15,284.43,1019,1019,62,Clear,1.72,JP,139.6917,35.6895									
3	Tokyo,2023-12-09 00:00:00,283.99,283.99,285.67,1020,1020,55,Clear,0.87,JP,139.6917,35.6895									
4	Tokyo,2023-12-09 03:00:00,286.54,286.54,288.24,1019,1019,47,Clear,2.16,JP,139.6917,35.6895									
5	Tokyo,2023-12-09 06:00:00,289.61,289.61,289.61,1019,1019,36,Clear,1.34,JP,139.6917,35.6895									
6	Tokyo,2023-12-09 09:00:00,288.8,288.8,288.8,1020,1020,39,Clouds,0.52,JP,139.6917,35.6895									
7	Tokyo,2023-12-09 12:00:00,287.76,287.76,287.76,1020,1020,40,Clouds,1.63,JP,139.6917,35.6895									
8	Tokyo,2023-12-09 15:00:00,286.83,286.83,286.83,1020,1020,40,Clouds,2.13,JP,139.6917,35.6895									
9	Tokyo,2023-12-09 18:00:00,285.81,285.81,285.81,1020,1020,41,Clouds,2.41,JP,139.6917,35.6895									
10	Tokyo,2023-12-09 21:00:00,285.04,285.04,285.04,1021,1021,41,Clear,2.42,JP,139.6917,35.6895									
11	Tokyo,2023-12-10 00:00:00,286.53,286.53,286.53,1023,1023,37,Clear,2.83,JP,139.6917,35.6895									
12	Tokyo,2023-12-10 03:00:00,289.59,289.59,289.59,1021,1021,32,Clear,2.25,JP,139.6917,35.6895									
13	Tokyo,2023-12-10 06:00:00,290.78,290.78,290.78,1021,1021,31,Clear,1.78,JP,139.6917,35.6895									
14	Tokyo,2023-12-10 09:00:00,289.66,289.66,289.66,1023,1023,35,Clear,2.39,JP,139.6917,35.6895									
15	Tokyo,2023-12-10 12:00:00,288.01,288.01,288.01,1025,1025,49,Clear,2.61,JP,139.6917,35.6895									
16	Tokyo,2023-12-10 15:00:00,287.33,287.33,287.33,1025,1025,67,Rain,3.88,JP,139.6917,35.6895									
17	Tokyo,2023-12-10 18:00:00,286.44,286.44,286.44,1025,1025,69,Rain,4.18,JP,139.6917,35.6895									
18	Tokyo,2023-12-10 21:00:00,286.3,286.3,286.3,1025,1025,73,Rain,4.12,JP,139.6917,35.6895									
19	Tokyo,2023-12-11 00:00:00,285.8,285.8,285.8,1027,1027,79,Rain,4.44,JP,139.6917,35.6895									

Weather_Data

Prêt Accessibilité : non disponible

4) Prétraitement des données

Prétraitement des données

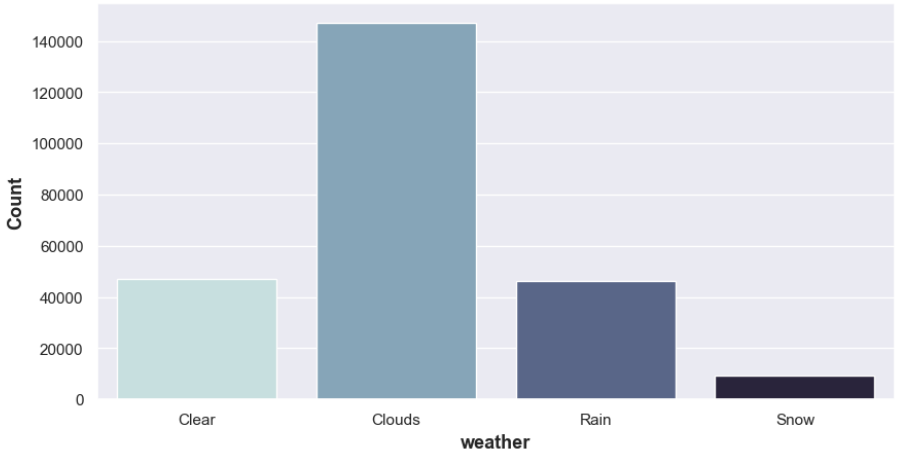
Entrée [1]:
```
import seaborn as sns
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Entrée [2]:
```
# Chargement du jeu de données
data = pd.read_csv("Weather_Data.csv")
data.head()
data.tail()
data.info()
```

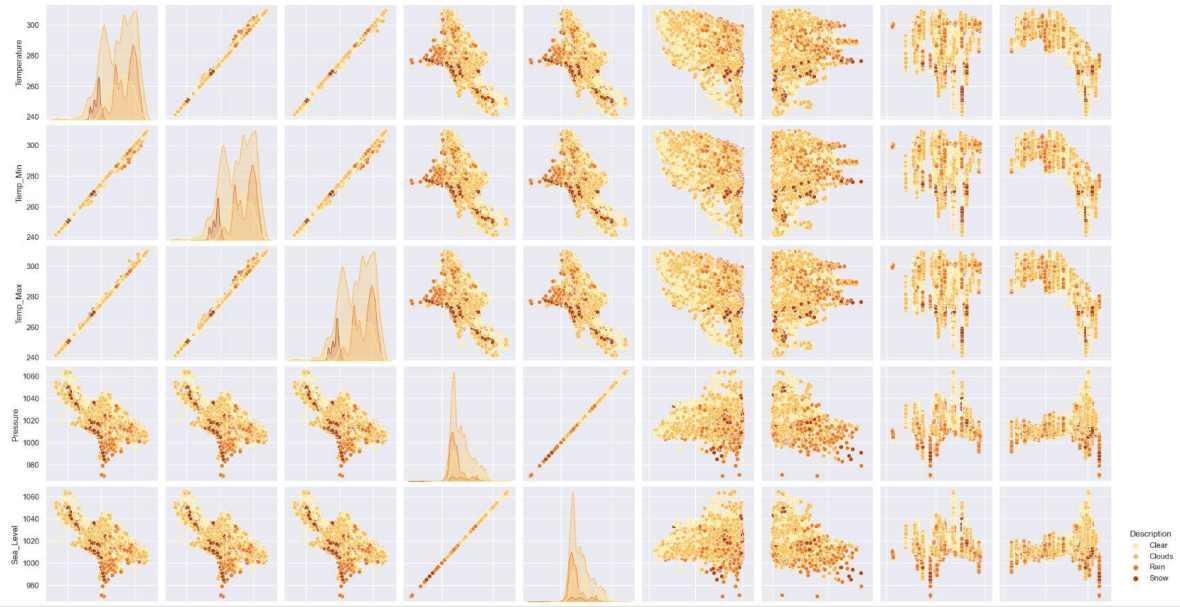
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 249850 entries, 0 to 249849
Data columns (total 13 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   City         249850 non-null  object
 1   Date         249850 non-null  object
 2   Temperature  249850 non-null  float64
 3   Temp_Min     249850 non-null  float64
 4   Temp_Max     249850 non-null  float64
 5   Pressure     249850 non-null  int64
 6   Sea_Level    249850 non-null  int64
 7   Humidity     249850 non-null  int64
 8   Description  249850 non-null  object
 9   Wind Speed   249850 non-null  float64
 10  Country      249163 non-null  object
 11  Lon          249850 non-null  float64
 12  Lat          249850 non-null  float64
dtypes: float64(6), int64(3), object(4)
memory usage: 24.8+ MB
```

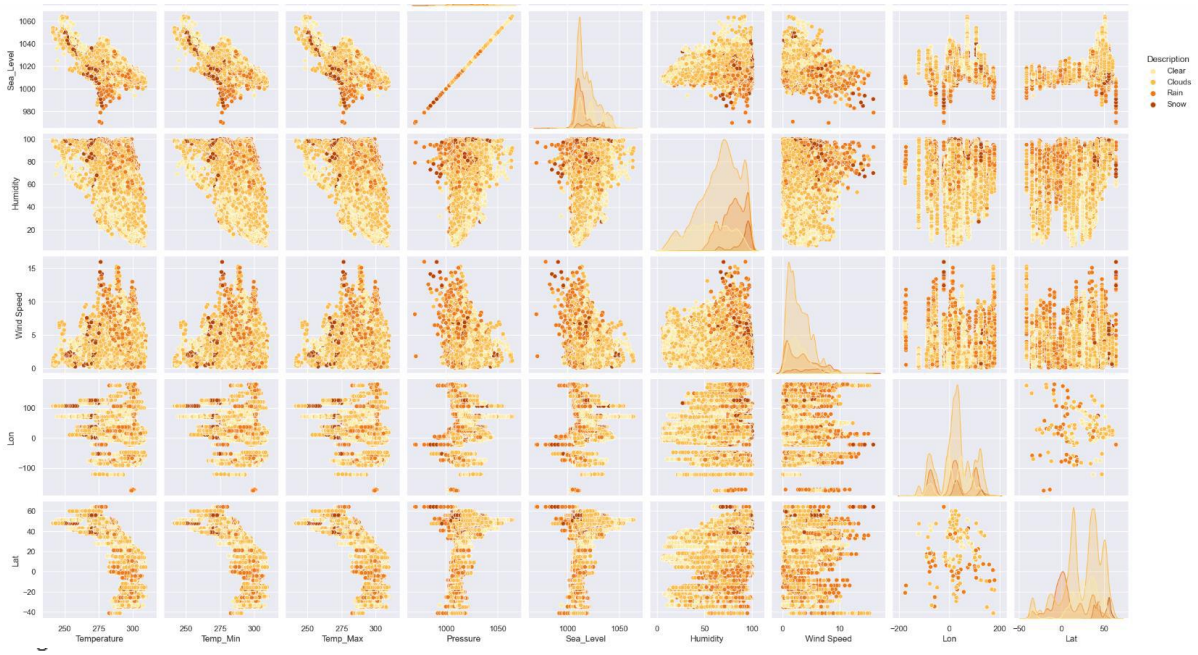
Entrée [3]:
```
plt.figure(figsize=(10,5))
sns.set_theme()
sns.countplot(x = 'Description',data = data,palette="ch:start=.2,rot=-.3")
plt.xlabel("weather",fontweight='bold',size=13)
plt.ylabel("Count",fontweight='bold',size=13)
plt.show()
```

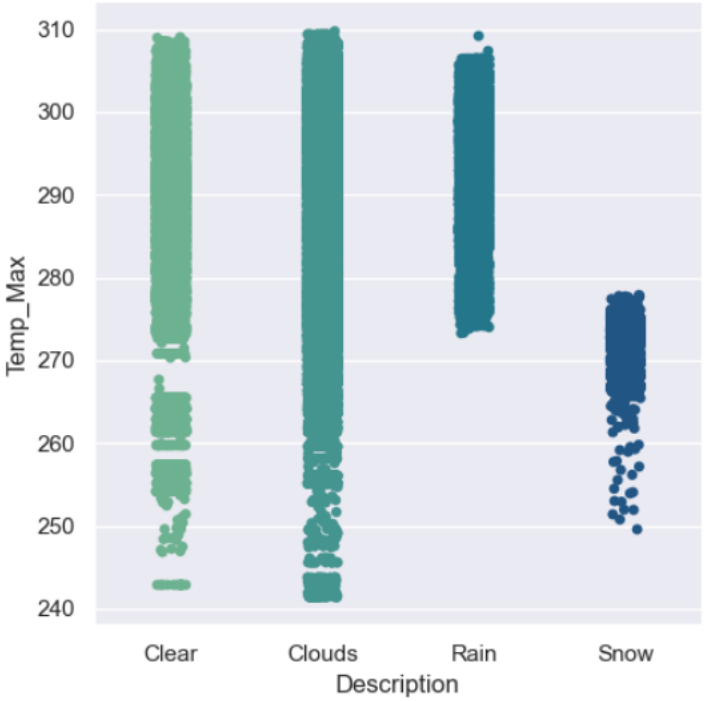
Entrée [3]:
```
plt.figure(figsize=(10,5))
sns.set_theme()
sns.countplot(x = 'Description',data = data,palette="ch:start=.2,rot=-.3")
plt.xlabel("weather",fontweight='bold',size=13)
plt.ylabel("Count",fontweight='bold',size=13)
plt.show()
```



Entrée [4]:
```
plt.figure(figsize=(14,8))
sns.pairplot(data.drop('Date',axis=1),hue='Description',palette="YlOrBr")
plt.show()
```



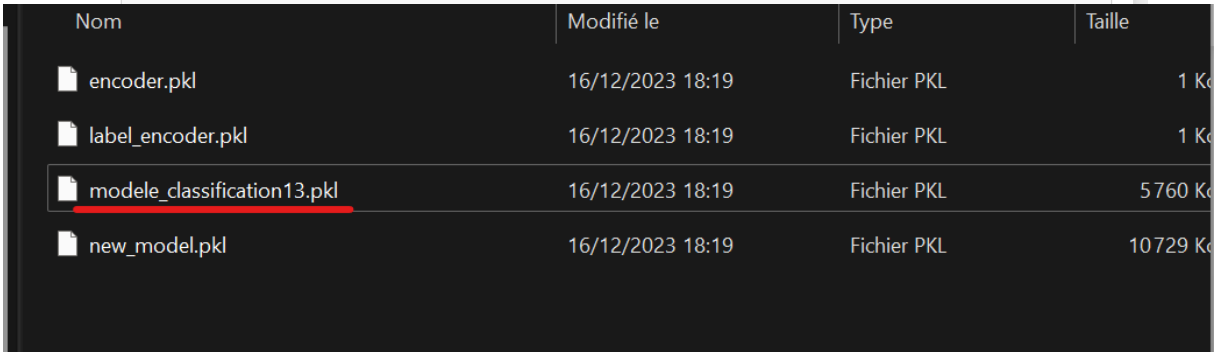

Description
Clear
Clouds
Rain
Snow
Sea_Level
Humidity
Wind Speed
Lon
Lat
Temperature
Temp_Min
Temp_Max
Pressure
Sea_Level
Humidity
Wind Speed
Lon
Lat


Temp_Max
310
300
290
280
270
260
250
240
Clear
Clouds
Rain
Snow
Description

Entrée [ ]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from imblearn.under_sampling import RandomUnderSampler
from collections import Counter
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score
import pickle



# Encode the target variable 'Description'
label_encoder = LabelEncoder()
data['Description'] = label_encoder.fit_transform(data['Description'])

# Separate features and target
X = data.drop('Description', axis=1)
y = data['Description']

# Split the dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=42)


# Specify the merge parameters
new_data = pd.merge(X_train, y_train, how='inner', on=None, left_index=True, right_index=True, validate=None)


# Undersample the majority class using RandomUnderSampler
undersampler = RandomUnderSampler(sampling_strategy='not minority', random_state=42)
X_train_balanced, y_train_balanced = undersampler.fit_resample(X_train, y_train)

# Display class distribution after balancing
print("Class distribution after balancing:", Counter(y_train_balanced))

# Scale numerical features
scaler = StandardScaler()
X_train_balanced_scaled = scaler.fit_transform(X_train_balanced)
```

```
# Scale numerical features
scaler = StandardScaler()
X_train_balanced_scaled = scaler.fit_transform(X_train_balanced)
X_test_scaled = scaler.transform(X_test)

# Save the LabelEncoder
with open('label_encoder.pkl', 'wb') as encoder_file:
    pickle.dump(label_encoder, encoder_file)

# Train different models on the balanced and scaled dataset
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000, random_state=42),
    "SVM": SVC(kernel='linear', random_state=42),
    "KNN": KNeighborsClassifier(n_neighbors=5),
    "Naive Bayes": GaussianNB(),
    "Decision Tree": DecisionTreeClassifier(criterion='entropy', random_state=42),
    "Random Forest": RandomForestClassifier(n_estimators=40, random_state=42),
    "XGBoost": XGBClassifier()
}

results = {}

for model_name, model in models.items():
    model.fit(X_train_balanced_scaled, y_train_balanced)
    y_pred = model.predict(X_test_scaled)
    accuracy = accuracy_score(y_test, y_pred)
    results[model_name] = accuracy

# Find the best model
best_model_name = max(results, key=results.get)
best_accuracy = results[best_model_name]

print("Results:")
for model, accuracy in results.items():
    print(f"{model}: {accuracy:.2%}")

print(f"Best Model: {best_model_name} with Accuracy: {best_accuracy:.2%}")

# Save the best model as a pickle file with a new name
new_file_name = 'dags\scripts\pickle_files\modele_classification13.pkl'
with open(new_file_name, 'wb') as file:
    pickle.dump(models[best_model_name], file)
```



5) Développement de l'application web

weather_dag.py streamlit_app.py

```python
import streamlit as st
import pickle
import pandas as pd
from sklearn import preprocessing
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
from matplotlib.ticker import MaxNLocator
import filter
import visualization as viz

# Load the pre-trained model
with open('/opt/airflow/dags/scripts/pickle_files/modele_classification13.pkl', 'rb') as model_file:
    model = pickle.load(model_file)

label_encoder = preprocessing.LabelEncoder()

# num_samples = 400
# nouvelles_donnees1 = pd.DataFrame({
#     'Temperature': np.random.uniform(270, 300, num_samples),
#     'Temp_Min': np.random.uniform(270, 300, num_samples),
#     'Temp_Max': np.random.uniform(270, 300, num_samples),
#     'Pressure': np.random.uniform(990, 1020, num_samples),
```

```python
# Main Streamlit app
def main():
    st.title("Weather Prediction App")

    # Sidebar navigation
    page = st.sidebar.selectbox("Choose a page", ["Home", "Prediction"])

    if page == "Home":

        data = pd.read_csv("/opt/airflow/dags/scripts/Weather_Data.csv")

        cities = filter.get_unique_cities_and_countries(data)
        city_selected = filter.create_city_country_dropdown(cities)

        # Call the function to get filtered records
        filtered_records = filter.filter_records_by_city(data, city_selected)

        # Display the selected city in the map
        viz.map(filtered_records)

        # Dataset features
        all_features = ['Temperature', 'Temp_Min', 'Temp_Max', 'Pressure', 'Humidity', 'Wind Speed']
```

```
        viz.create_viz(filtered_records,all_features)


    elif page == "Prediction":

        st.write("Make a weather prediction:")
        # User input for each feature
        temperature = st.slider("Temperature", min_value=0.0, max_value=400.0)
        temp_min = st.slider("Minimum Temperature", min_value=0.0, max_value=400.0)
        temp_max = st.slider("Maximum Temperature", min_value=0.0, max_value=400.0)
        pressure = st.slider("Pressure", min_value=900, max_value=1100)
        sea_level = st.slider("Sea Level", min_value=900, max_value=1100)
        humidity = st.slider("Humidity", min_value=0, max_value=100)
        wind_speed = st.slider("Wind Speed", min_value=0.0, max_value=20.0)
        lon = st.number_input("Longitude")
        lat = st.number_input("Latitude")
        # Create a DataFrame with user input
        user_data = pd.DataFrame({
            'Temperature': [temperature],
            'Temp_Min': [temp_min],
            'Temp_Max': [temp_max],
            'Pressure': [pressure],
            'Sea_Level': [sea_level],
```

```
            'Pressure': [pressure],
            'Sea_Level': [sea_level],
            'Humidity': [humidity],
            'Wind Speed': [wind_speed],
            'Lon': [lon],
            'Lat': [lat]
        })
        # Load the label_encoder
        with open('/opt/airflow/dags/pickle_files/label_encoder.pkl', 'rb') as label_encoder_file:
            label_encoder1 = pickle.load(label_encoder_file)
        # Make a prediction
        if st.button("Predict"):
            # scaler = preprocessing.StandardScaler()
            # processed_data = scaler.fit_transform(nouvelles_donnees1)
            print(user_data)
            prediction = model.predict(user_data)
            print("Raw predictions:",prediction)
            st.success(f"Prediction: {prediction[0]}")

if __name__ == "__main__":
    main()
```

6) Mise en place avec Apache Airflow

weather_dag.py

```python
from datetime import datetime, timedelta
from pytz import timezone
from airflow import DAG
from airflow.operators.bash import BashOperator

# Set the timezone to Casablanca
casablanca_tz = timezone('Africa/Casablanca')

default_args = {
    'owner': 'OUHAYOU_OUSSAMA',
    'depends_on_past': False,
    'start_date': datetime.now(casablanca_tz),
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 0,
    'retry_delay': timedelta(minutes=5),
}

dag = DAG(
    'Weather_app',
    default_args=default_args,
    description='An Airflow DAG to fetch weather data and update the CSV file',
    schedule_interval='0 */3 * * *',  # Run every 3 hours
    max_active_runs=1,  # Ensure only one run at a time
```

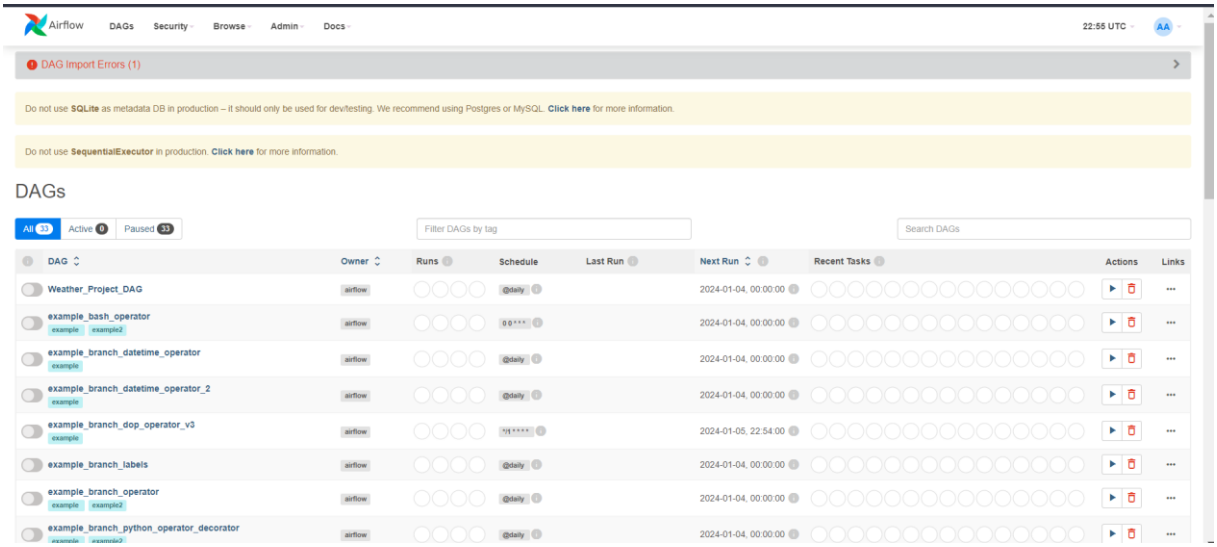
weather_dag
casablanca_tz: Any = timezone('Africa/Casablanca')

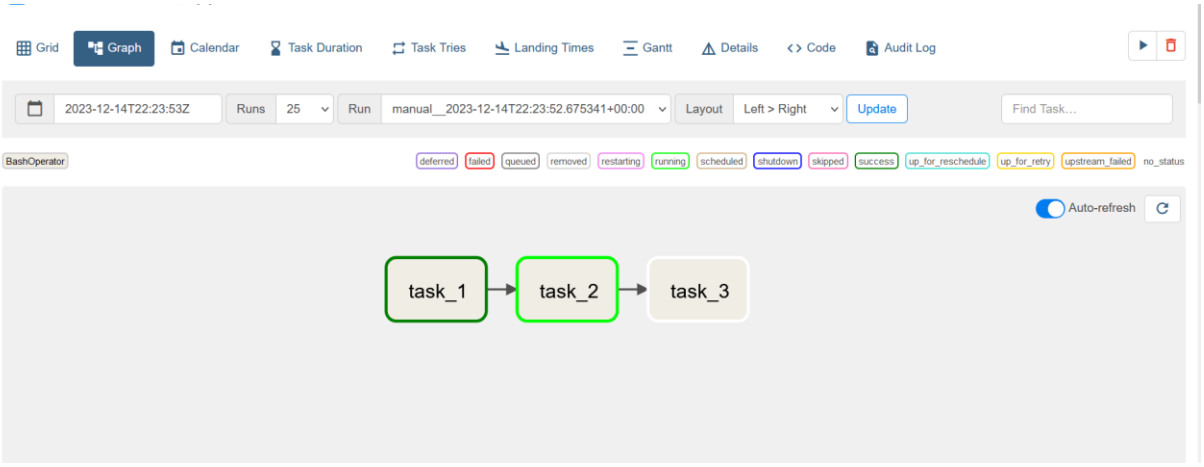
weather_dag.py

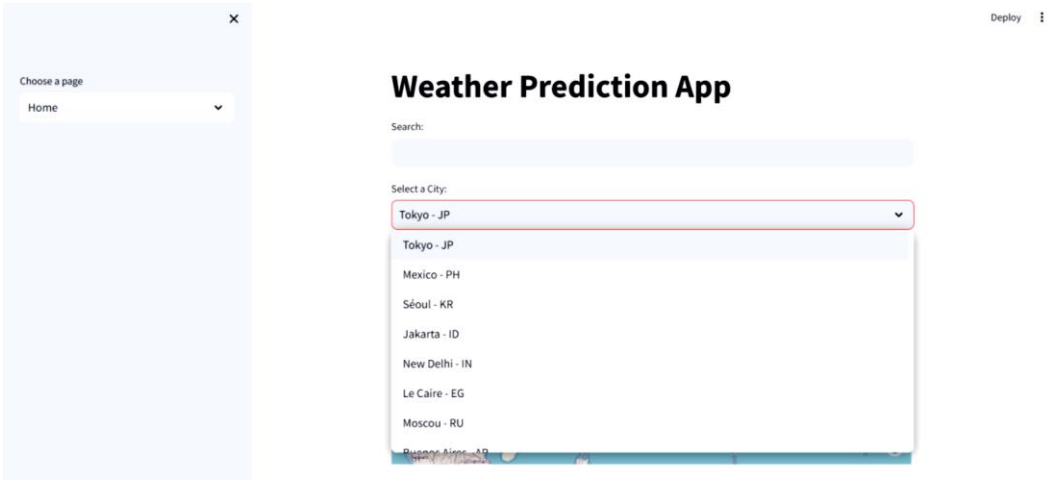
```python
dag = DAG(
    'Weather_app',
    default_args=default_args,
    description='An Airflow DAG to fetch weather data and update the CSV file',
    schedule_interval='0 */3 * * *',  # Run every 3 hours
    max_active_runs=1,  # Ensure only one run at a time
    catchup=False,  # Do not run backfill for the intervals between start_date and the curr
)



# Task to execute the Python script
execute_script_task_1 = BashOperator(
    task_id='task_1',
    bash_command='python /opt/airflow/dags/scripts/main.py',
    dag=dag
)

# Task to execute the Python script
execute_script_task_2 = BashOperator(
    task_id='task_2',
    bash_command='python /opt/airflow/dags/scripts/Projet_ML2.py',
    dag=dag
)
```

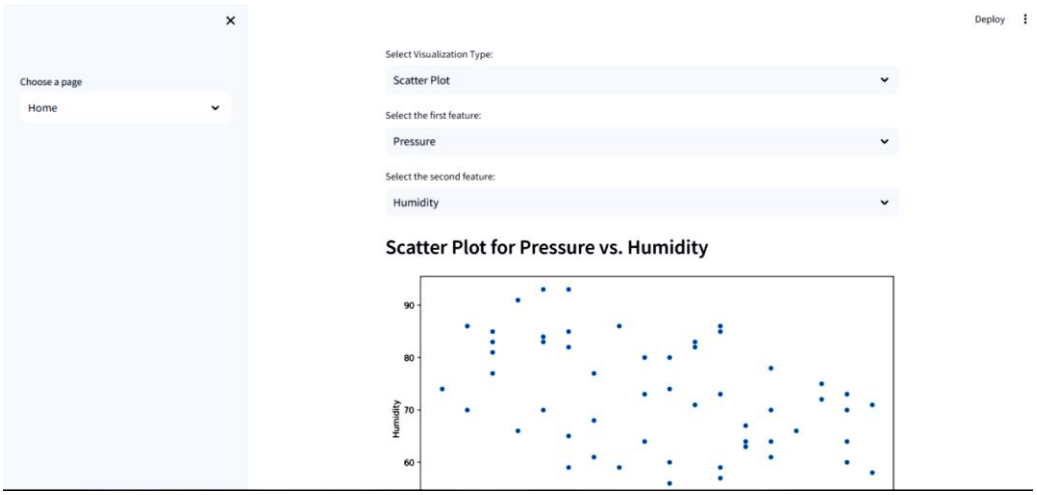
```python
)

execute_script_task_3 = BashOperator(
    task_id='task_3',
    bash_command='streamlit run --server.address 0.0.0.0 --server.enableWebsocketCompressio
    dag=dag
)

# Set task dependencies
execute_script_task_1 >> execute_script_task_2 >> execute_script_task_3
```

```
PS C:\Users\Ouss_ama\Desktop\2ITE-3\BigDataImage\dags> python weather_dag.py
WARNING:root:OSError while attempting to symlink the latest log directory
C:\Users\Ouss_ama\Desktop\2ITE-3\BigDataImage\dags\weather_dag.py:19 RemovedInAirflow3Warning: Param `schedule_interval` is deprecated and will be r
emoved in a future release. Please use `schedule` instead.
PS C:\Users\Ouss_ama\Desktop\2ITE-3\BigDataImage\dags> python weather_dag.py
WARNING:root:OSError while attempting to symlink the latest log directory
PS C:\Users\Ouss_ama\Desktop\2ITE-3\BigDataImage\dags>
```






Deploy
Choose a page
Home
Weather Prediction App
Search:
Select a City:
Tokyo - JP
Tokyo - JP
Mexico - PH
Séoul - KR
Jakarta - ID
New Delhi - IN
Le Caire - EG
Moscou - RU


Deploy
Choose a page
Home
Select Visualization Type:
Scatter Plot
Select the first feature:
Pressure
Select the second feature:
Humidity
Scatter Plot for Pressure vs. Humidity
Humidity
90
80
70
60