OpenAI

Weather Data Analysis

```
import requests
import pandas as pd
import datetime
import matplotlib.pyplot as plt
import seaborn as sns
import folium  # For geospatial visualization

# Set your OpenWeatherMap API key
api_key = 'fb365aa6104829b44455572365ff3b4e'

# Set the location for which you want to retrieve weather data
lat = 18.184135
lon = 74.610764

# Construct the API URL
api_url = f"http://api.openweathermap.org/data/2.5/forecast?lat={lat}&lon={lon}&appid=
{api_key}"

# Send a GET request to the API
response = requests.get(api_url)
weather_data = response.json()

# Extract relevant weather attributes using list comprehension
timestamps = [pd.to_datetime(item['dt'], unit='s') for item in weather_data['list']]
temperatures = [item['main']['temp'] for item in weather_data['list']]
humidity = [item['main']['humidity'] for item in weather_data['list']]
wind_speed = [item['wind']['speed'] for item in weather_data['list']]
weather_description = [item['weather'][0]['description'] for item in weather_data['list']]

# Create a DataFrame with the extracted weather data
weather_df = pd.DataFrame({
    'Timestamp': timestamps,
    'Temperature': temperatures,
    'Humidity': humidity,
    'Wind Speed': wind_speed,
    'Description': weather_description
})

# Set the Timestamp as the DataFrame's index
weather_df.set_index('Timestamp', inplace=True)

# Convert temperature from Kelvin to Celsius if needed
weather_df['Temperature'] = weather_df['Temperature'].apply(lambda x: x - 273.15 if x > 200 else
x)
```

```
# Handling missing values
weather_df.fillna(method='ffill', inplace=True)  # Forward-fill missing values

# Convert all numeric columns to numeric types if they're not already
numeric_columns = ['Temperature', 'Humidity', 'Wind Speed']
weather_df[numeric_columns] = weather_df[numeric_columns].apply(pd.to_numeric,
errors='coerce')

# Calculate daily, monthly, and seasonal means (excluding non-numeric columns)
weather_daily = weather_df[numeric_columns].resample('D').mean()
weather_monthly = weather_df[numeric_columns].resample('M').mean()
weather_seasonal = weather_df[numeric_columns].resample('Q').mean()

# Display the cleaned and preprocessed data
print("Daily Weather Data:\n", weather_daily)
print("Monthly Weather Data:\n", weather_monthly)
print("Seasonal Weather Data:\n", weather_seasonal)

# Maximum and minimum temperatures
max_temp = weather_df['Temperature'].max()
min_temp = weather_df['Temperature'].min()
print(f"Maximum Temperature: {max_temp:.2f}°C")
print(f"Minimum Temperature: {min_temp:.2f}°C")

# Visualization
# Plot mean daily temperature
plt.figure(figsize=(10, 6))
weather_daily['Temperature'].plot(color='red', linestyle='-', marker='o')
plt.title('Mean Daily Temperature')
plt.xlabel('Date')
plt.ylabel('Temperature (°C)')
plt.grid(True)
plt.show()

# Plot mean daily humidity
plt.figure(figsize=(10, 6))
weather_daily['Humidity'].plot(kind='bar', color='blue')
plt.title('Mean Daily Humidity')
plt.xlabel('Date')
plt.ylabel('Humidity (%)')
plt.grid(True)
plt.show()

# Plot Temperature vs. Wind Speed Scatter Plot
plt.figure(figsize=(10, 6))
plt.scatter(weather_df['Temperature'], weather_df['Wind Speed'], color='green')
plt.title('Temperature vs. Wind Speed')
plt.xlabel('Temperature (°C)')
plt.ylabel('Wind Speed (m/s)')
plt.grid(True)
```

plt.show()

# Temperature vs Humidity Heatmap
heatmap_data = weather_df[['Temperature', 'Humidity']]
sns.heatmap(heatmap_data.corr(), annot=True, cmap='coolwarm')
plt.title('Temperature vs Humidity Heatmap')
plt.show()

# Scatter plot: Temperature vs. Humidity
plt.figure(figsize=(10, 6))
plt.scatter(weather_df['Temperature'], weather_df['Humidity'])
plt.xlabel('Temperature (°C)')
plt.ylabel('Humidity (%)')
plt.title('Temperature vs Humidity Scatter Plot')
plt.show()

# Geospatial Visualization (Map)
# Creating a folium map centered on the specified latitude and longitude
weather_map = folium.Map(location=[lat, lon], zoom_start=10)

# Adding a marker for the location with weather information
folium.Marker(
  [lat, lon],
  popup=f"Temperature: {max_temp:.2f}°C\nHumidity: {weather_daily['Humidity'].mean():.2f}%",
  tooltip="Click for weather details"
).add_to(weather_map)

# Save the map to an HTML file
weather_map.save("weather_map.html")
print("Geospatial map saved as 'weather_map.html'.")
Explain this code in very detail

This code performs several steps to retrieve, process, analyze, and visualize weather data using Python. Here's a detailed breakdown of each part:

Step 1: Import Libraries

```python
import requests
import pandas as pd
import datetime
import matplotlib.pyplot as plt
import seaborn as sns
import folium
```

- `requests`: For sending HTTP requests to the OpenWeatherMap API to fetch weather data.
- `pandas`: For data manipulation and analysis, specifically to store and transform the weather data.

- `datetime`: For handling date and time information.
- `matplotlib.pyplot`: For creating various types of visualizations.
- `seaborn`: An extension of `matplotlib` with more advanced visualizations, used here for heatmaps.
- `folium`: For creating interactive geospatial visualizations, such as maps.

Step 2: Define API Key and Location

```python
api_key = 'fb365aa6104829b44455572365ff3b4e'
lat = 18.184135
lon = 74.610764
```

- `api_key`: The unique key to access the OpenWeatherMap API.
- `lat` and `lon`: Latitude and longitude of the location for which the weather

LEave loading and smaill small things, explain main code in detail

Certainly, let's go through the core sections of the code to explain the primary functionalities in detail:

1. Fetching and Parsing Weather Data

```python
# Construct the API URL
api_url = f"http://api.openweathermap.org/data/2.5/forecast?lat={lat}&lon={lon}&appid={api
_key}"

# Send a GET request to the API
response = requests.get(api_url)
weather_data = response.json()
```

- This section constructs a URL to fetch a 5-day weather forecast for a specific latitude and longitude. After sending a request to the OpenWeatherMap API, the JSON response is parsed into the `weather_data` dictionary.

2. Extracting and Organizing Relevant Data

```python
# Extract relevant weather attributes using list comprehension
timestamps = [pd.to_datetime(item['dt'], unit='s') for item in weather_data['list']]
temperatures = [item['main']['temp'] for item in weather_data['list']]
humidity = [item['main']['humidity'] for item in weather_data['list']]
```

```
wind_speed = [item['wind']['speed'] for item in weather_data['list']]
weather_description = [item['weather'][0]['description'] for item in weather_data['list']]
```

- Here, list comprehensions are used to extract specific weather data from each forecast entry (`item`) in `weather_data['list']`:
  - `timestamps`: Converts Unix timestamps to a `datetime` format.
  - `temperatures`: Retrieves temperature values in Kelvin.
  - `humidity`: Retrieves humidity percentages.
  - `wind_speed`: Retrieves wind speed values in meters per second.
  - `weather_description`: Retrieves a description of the weather condition (e.g., "clear sky").

3. Creating a DataFrame for Analysis

```python
# Create a DataFrame with the extracted weather data
weather_df = pd.DataFrame({
    'Timestamp': timestamps,
    'Temperature': temperatures,
    'Humidity': humidity,
    'Wind Speed': wind_speed,
    'Description': weather_description
})
weather_df.set_index('Timestamp', inplace=True)
```

- This step organizes the extracted data into a `DataFrame`, `weather_df`, for easy manipulation and analysis. The `Timestamp` column is set as the index, which helps in resampling and time-series operations.

4. Data Transformation and Cleaning

```python
# Convert temperature from Kelvin to Celsius if needed
weather_df['Temperature'] = weather_df['Temperature'].apply(lambda x: x - 273.15 if x > 20
0 else x)

# Handling missing values
weather_df.fillna(method='ffill', inplace=True)

# Convert all numeric columns to numeric types if they're not already
numeric_columns = ['Temperature', 'Humidity', 'Wind Speed']
weather_df[numeric_columns] = weather_df[numeric_columns].apply(pd.to_numeric, errors='coe
rce')
```

- **Temperature Conversion**: Temperatures are converted from Kelvin to Celsius, applying the formula `x - 273.15`.
- **Handling Missing Values**: Any missing values are forward-filled (using the last available data point).
- **Ensuring Numeric Data**: Numeric columns are explicitly converted to numeric types for consistent analysis and to handle potential non-numeric values.

5. Calculating Daily, Monthly, and Seasonal Averages

```python
# Calculate daily, monthly, and seasonal means (excluding non-numeric columns)
weather_daily = weather_df[numeric_columns].resample('D').mean()
weather_monthly = weather_df[numeric_columns].resample('M').mean()
weather_seasonal = weather_df[numeric_columns].resample('Q').mean()
```

- **Resampling**: The `resample()` function groups data based on time intervals:
  - `weather_daily` calculates daily averages.
  - `weather_monthly` calculates monthly averages.
  - `weather_seasonal` calculates seasonal averages (in quarters).

This provides a summary of weather trends over various time periods.

6. Finding Maximum and Minimum Temperatures

```python
# Maximum and minimum temperatures
max_temp = weather_df['Temperature'].max()
min_temp = weather_df['Temperature'].min()
print(f"Maximum Temperature: {max_temp:.2f}°C")
print(f"Minimum Temperature: {min_temp:.2f}°C")
```

- This section calculates the maximum and minimum temperatures from the dataset, giving insights into the highest and lowest recorded temperatures in the forecast data.

7. Visualizations

The code then uses `matplotlib` and `seaborn` to visualize the data with different types of plots:

- **Mean Daily Temperature Line Plot**

  ```python
  plt.figure(figsize=(10, 6))
  weather_daily['Temperature'].plot(color='red', linestyle='-', marker='o')
  ```

  - Plots the daily average temperature over time using a line plot.
- **Mean Daily Humidity Bar Plot**

  ```python
  plt.figure(figsize=(10, 6))
  weather_daily['Humidity'].plot(kind='bar', color='blue')
  ```

  - Plots daily average humidity as a bar chart.
- **Scatter Plot of Temperature vs. Wind Speed**

```python
plt.figure(figsize=(10, 6))
plt.scatter(weather_df['Temperature'], weather_df['Wind Speed'], color='green')
```

- Shows the relationship between temperature and wind speed, helping to observe if higher temperatures correspond to higher/lower wind speeds.

- **Heatmap of Temperature vs. Humidity**

```python
heatmap_data = weather_df[['Temperature', 'Humidity']]
sns.heatmap(heatmap_data.corr(), annot=True, cmap='coolwarm')
```

- Displays a correlation heatmap to show the relationship between temperature and humidity. Values closer to `1` or `-1` indicate a strong relationship.

8. Geospatial Visualization Using Folium

```python
# Creating a folium map centered on the specified latitude and longitude
weather_map = folium.Map(location=[lat, lon], zoom_start=10)

# Adding a marker for the location with weather information
folium.Marker(
    [lat, lon],
    popup=f"Temperature: {max_temp:.2f}°C\nHumidity: {weather_daily['Humidity'].mean():.2
f}%",
    tooltip="Click for weather details"
).add_to(weather_map)
weather_map.save("weather_map.html")
```

- This section creates an interactive map centered on the specified location.
- A marker is added at the latitude and longitude, displaying the maximum temperature and mean humidity when clicked.
- The map is saved as an HTML file (`weather_map.html`) that can be viewed in a web browser, enabling geospatial visualization of weather data.

Explain all the graphs in the code in detail

Here's a detailed explanation of each graph used in the code, along with the purpose and insights it provides from the weather data.

---

1. Mean Daily Temperature Line Plot

python

```python
plt.figure(figsize=(10, 6))
weather_daily['Temperature'].plot(color='red', linestyle='-', marker='o')
plt.title('Mean Daily Temperature')
plt.xlabel('Date')
plt.ylabel('Temperature (°C)')
plt.grid(True)
plt.show()
```

- **Description**: This line plot displays the mean daily temperature over a given time period.
- **Purpose**: By plotting daily temperature averages, this graph reveals trends in temperature fluctuations from day to day.
- **Insights**:
  - Peaks and troughs indicate warmer and cooler days, respectively.
  - It's useful for identifying patterns, such as daily warming or cooling trends.
  - If the plot spans different seasons, you might observe broader seasonal patterns (e.g., temperatures rising in summer or falling in winter).
- **Visual Elements**:
  - Each point represents the average temperature for a specific day.
  - The line connects the points to show continuity in temperature change over time.

---

2. Mean Daily Humidity Bar Plot

python

```python
plt.figure(figsize=(10, 6))
weather_daily['Humidity'].plot(kind='bar', color='blue')
plt.title('Mean Daily Humidity')
plt.xlabel('Date')
plt.ylabel('Humidity (%)')
plt.grid(True)
plt.show()
```

- **Description**: This bar plot shows the average daily humidity percentage.
- **Purpose**: It provides a clear view of daily humidity levels and makes it easy to compare humidity across days.
- **Insights**:
  - The height of each bar indicates the level of humidity on that specific day.
  - This plot helps to identify periods of high or low humidity, which could relate to changes in weather conditions (e.g., higher humidity might indicate rain or stormy conditions).
  - It's useful for quickly spotting days with unusually high or low humidity levels.
- **Visual Elements**:
  - Each bar represents the mean humidity for a single day, and the bar height corresponds to the humidity percentage.
  - The color blue emphasizes the humidity data, and a grid is added to help in visually assessing the values.

---

3. Scatter Plot of Temperature vs. Wind Speed

python

```python
plt.figure(figsize=(10, 6))
plt.scatter(weather_df['Temperature'], weather_df['Wind Speed'], color='green')
plt.title('Temperature vs. Wind Speed')
plt.xlabel('Temperature (°C)')
plt.ylabel('Wind Speed (m/s)')
plt.grid(True)
plt.show()
```

- **Description**: This scatter plot displays the relationship between temperature and wind speed.
- **Purpose**: It helps to analyze whether there is a correlation between temperature and wind speed, and if certain temperatures correspond to higher or lower wind speeds.
- **Insights**:
  - Each point represents a recorded temperature and its corresponding wind speed at that moment.
  - If a pattern (e.g., a clustering of points in a particular area) appears, it might indicate a relationship (positive, negative, or neutral) between temperature and wind speed.
  - For example, if warmer temperatures tend to have lower wind speeds, points may cluster towards the lower end of the y-axis as x-values increase.
- **Visual Elements**:
  - Each green dot represents a specific measurement of temperature and wind speed.
  - This scatter plot format is helpful for assessing the spread and clustering of data points.

---

4. Temperature vs. Humidity Heatmap

python

```python
heatmap_data = weather_df[['Temperature', 'Humidity']]
sns.heatmap(heatmap_data.corr(), annot=True, cmap='coolwarm')
plt.title('Temperature vs Humidity Heatmap')
plt.show()
```

- **Description**: This heatmap visualizes the correlation between temperature and humidity.
- **Purpose**: It shows the strength and direction of the relationship between temperature and humidity using correlation values.
- **Insights**:
  - The correlation coefficient (between -1 and 1) indicates how strongly temperature and humidity are related:
    - A value close to `` `1` `` suggests a strong positive correlation.
    - A value close to `` `-1` `` suggests a strong negative correlation.
    - A value close to `` `0` `` indicates no correlation.
  - This insight helps determine if temperature and humidity tend to rise and fall together or move in opposite directions.
  - For example, if the correlation is negative, we might expect humidity to decrease as temperature increases (or vice versa).
- **Visual Elements**:
  - The color scale (`` `coolwarm` ``) helps to visually interpret the correlation (warm colors for positive, cool for negative).
  - Annotations show exact correlation values, aiding in precise analysis.

5. Scatter Plot of Temperature vs. Humidity

```python
plt.figure(figsize=(10, 6))
plt.scatter(weather_df['Temperature'], weather_df['Humidity'])
plt.xlabel('Temperature (°C)')
plt.ylabel('Humidity (%)')
plt.title('Temperature vs Humidity Scatter Plot')
plt.show()
```

- **Description**: This scatter plot displays individual data points of temperature against humidity.
- **Purpose**: It shows the relationship and potential correlation between temperature and humidity in a more granular form.
- **Insights**:
  - Each point represents a measurement of temperature and humidity at a specific time.
  - Patterns can indicate trends, such as an inverse relationship where high temperatures correspond to lower humidity or direct relationships where both rise together.
  - The plot can reveal clustering, outliers, or data patterns that are more variable than the averaged correlation.
- **Visual Elements**:
  - Points are scattered across the plot, showing how temperature and humidity data pairs vary.
  - A grid and labeled axes help to interpret the spread and concentration of points across temperature and humidity values.

These graphs provide comprehensive insights into temperature, humidity, and wind speed over time, as well as relationships between these variables. They help identify weather trends and patterns, making the data more accessible for interpretation and analysis.