databricks   Toronto's Weather Forecast (Python)   Import Notebook

Toronto 10-Day Weather Forecast: ETL and Visualisation

Introduction

In this notebook, we extract, transform, load (ETL) and analyse the 10-day weather forecast for Toronto using data from the Weather API. The analysis involves visualising various weather-related parameters to gain insights into temperature trends, precipitation, air quality and more.

Extracting Data from Weather API

The first step involves extracting the 10-day weather forecast data for Toronto using the WeatherAPI. We import the necessary libraries, define the API endpoint URL and set the query parameters (including the API key, location and forecast duration). A GET request is sent to retrieve the forecast data and the data is processed to focus on hourly forecasts using a pandas DataFrame.

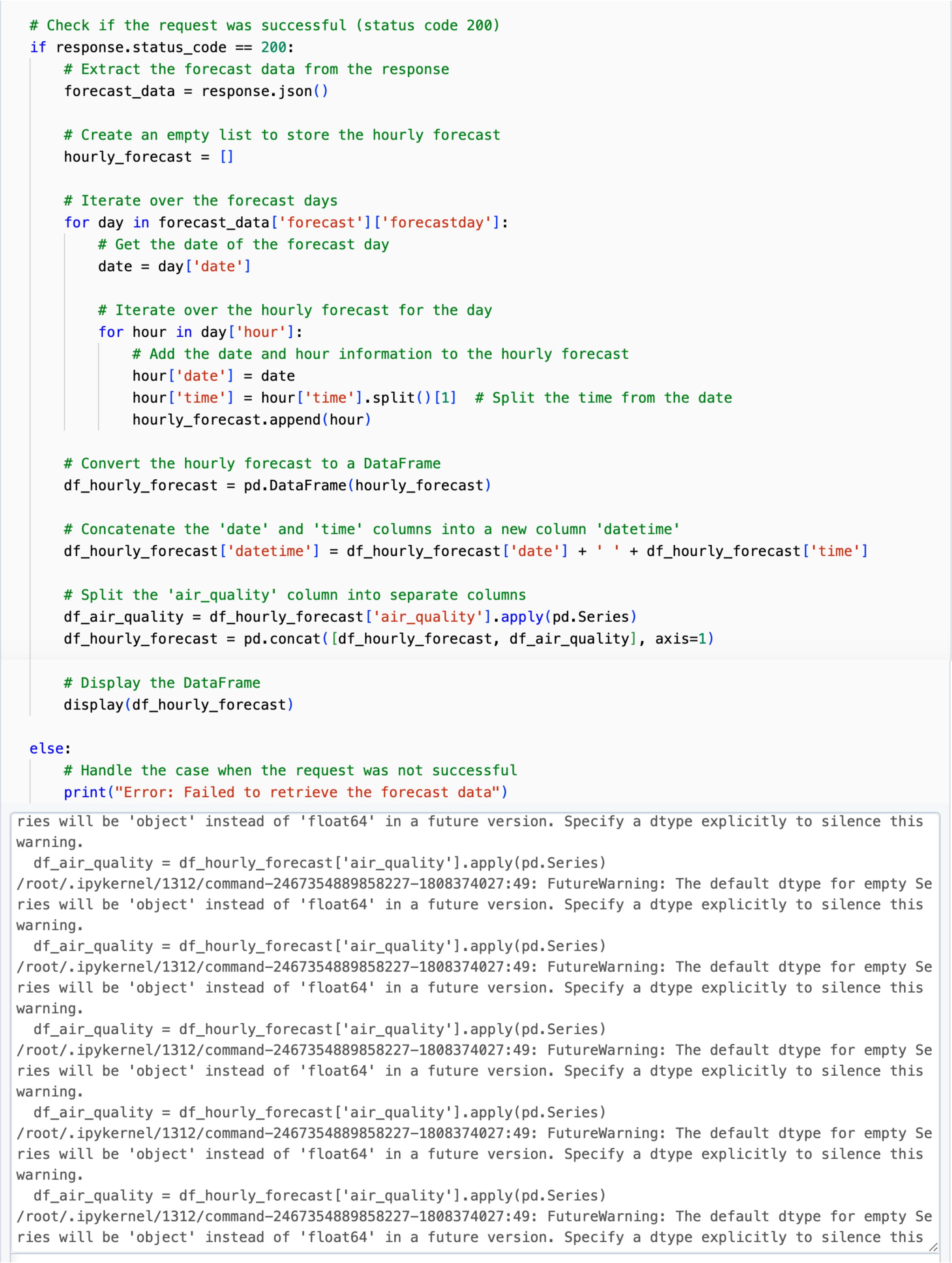
Data Transformation and Loading

Datetime Creation: A new column 'datetime' has been created by concatenating the date and time columns, which helps in aligning the data for time-series analysis.

Air Quality Data: The 'air_quality' column, which contains nested JSON data, was split into individual columns for each air quality parameter (e.g., CO, O3, NO2).

Data Display: The transformed DataFrame was displayed to verify the structure and content of the data.

```python
#Import the necessary libraries
import pandas as pd
import requests

# Define the API endpoint URL
url = "https://api.weatherapi.com/v1/forecast.json"

# Define the query parameters
params = {
    "key": "74402b963fbf4ad19a444545240308",
    "q": "toronto:canada",
    "days": 10,
    "aqi": "yes",
    "alerts": "yes"
}

# Send a GET request to the API endpoint
response = requests.get(url, params=params)
```



```python
# Check if the request was successful (status code 200)
if response.status_code == 200:
    # Extract the forecast data from the response
    forecast_data = response.json()

    # Create an empty list to store the hourly forecast
    hourly_forecast = []

    # Iterate over the forecast days
    for day in forecast_data['forecast']['forecastday']:
        # Get the date of the forecast day
        date = day['date']

        # Iterate over the hourly forecast for the day
        for hour in day['hour']:
            # Add the date and hour information to the hourly forecast
            hour['date'] = date
            hour['time'] = hour['time'].split()[1]  # Split the time from the date
            hourly_forecast.append(hour)

    # Convert the hourly forecast to a DataFrame
    df_hourly_forecast = pd.DataFrame(hourly_forecast)

    # Concatenate the 'date' and 'time' columns into a new column 'datetime'
    df_hourly_forecast['datetime'] = df_hourly_forecast['date'] + ' ' + df_hourly_forecast['time']

    # Split the 'air_quality' column into separate columns
    df_air_quality = df_hourly_forecast['air_quality'].apply(pd.Series)
    df_hourly_forecast = pd.concat([df_hourly_forecast, df_air_quality], axis=1)

    # Display the DataFrame
    display(df_hourly_forecast)

else:
    # Handle the case when the request was not successful
    print("Error: Failed to retrieve the forecast data")
```

```
ries will be 'object' instead of 'float64' in a future version. Specify a dtype explicitly to silence this
warning.
  df_air_quality = df_hourly_forecast['air_quality'].apply(pd.Series)
/root/.ipykernel/1312/command-2467354889858227-1808374027:49: FutureWarning: The default dtype for empty Se
ries will be 'object' instead of 'float64' in a future version. Specify a dtype explicitly to silence this
warning.
  df_air_quality = df_hourly_forecast['air_quality'].apply(pd.Series)
/root/.ipykernel/1312/command-2467354889858227-1808374027:49: FutureWarning: The default dtype for empty Se
ries will be 'object' instead of 'float64' in a future version. Specify a dtype explicitly to silence this
warning.
  df_air_quality = df_hourly_forecast['air_quality'].apply(pd.Series)
/root/.ipykernel/1312/command-2467354889858227-1808374027:49: FutureWarning: The default dtype for empty Se
ries will be 'object' instead of 'float64' in a future version. Specify a dtype explicitly to silence this
warning.
  df_air_quality = df_hourly_forecast['air_quality'].apply(pd.Series)
/root/.ipykernel/1312/command-2467354889858227-1808374027:49: FutureWarning: The default dtype for empty Se
ries will be 'object' instead of 'float64' in a future version. Specify a dtype explicitly to silence this
warning.
  df_air_quality = df_hourly_forecast['air_quality'].apply(pd.Series)
/root/.ipykernel/1312/command-2467354889858227-1808374027:49: FutureWarning: The default dtype for empty Se
ries will be 'object' instead of 'float64' in a future version. Specify a dtype explicitly to silence this
```

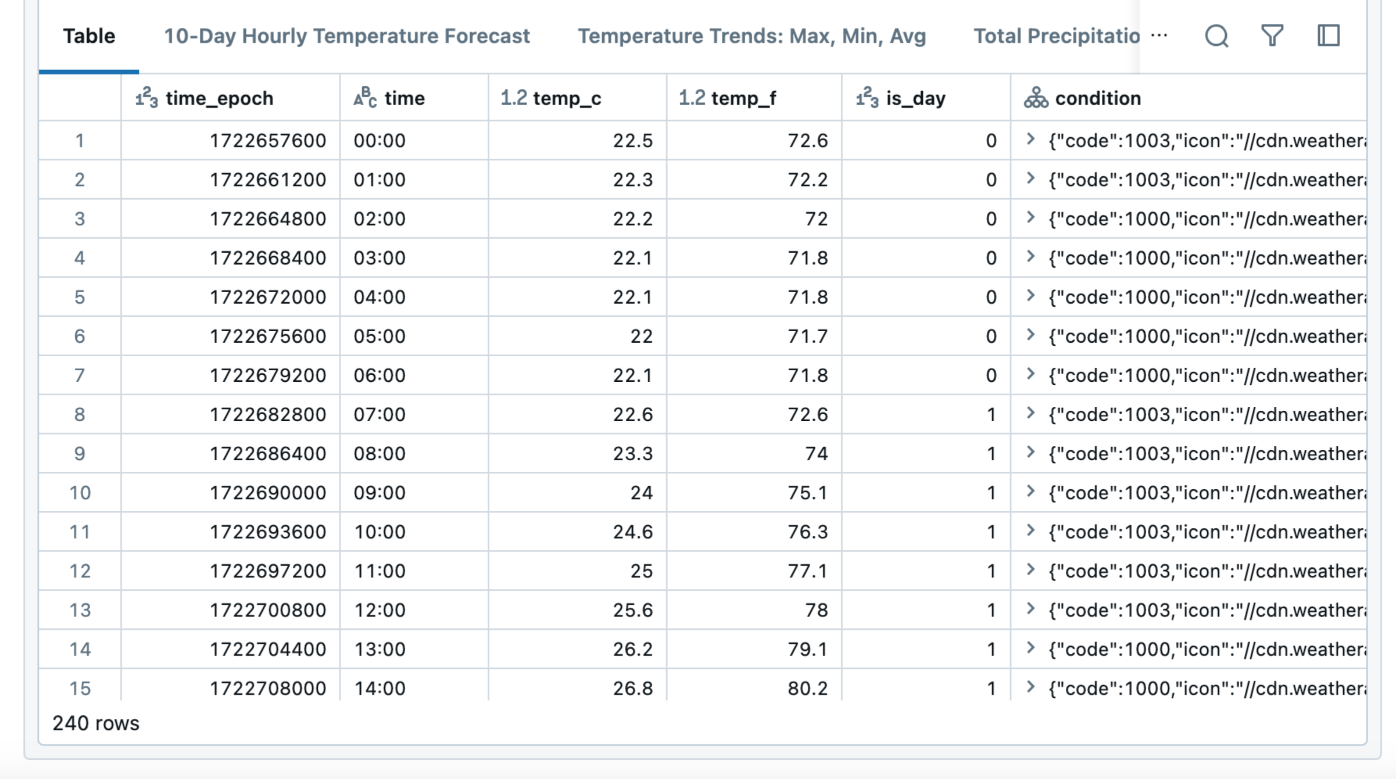


Table
10-Day Hourly Temperature Forecast
Temperature Trends: Max, Min, Avg
Total Precipitatio ···

	time_epoch	time	temp_c	temp_f	is_day	condition
1	1722657600	00:00	22.5	72.6	0	{"code":1003,"icon":"//cdn.weather
2	1722661200	01:00	22.3	72.2	0	{"code":1003,"icon":"//cdn.weather
3	1722664800	02:00	22.2	72	0	{"code":1000,"icon":"//cdn.weather
4	1722668400	03:00	22.1	71.8	0	{"code":1000,"icon":"//cdn.weather
5	1722672000	04:00	22.1	71.8	0	{"code":1000,"icon":"//cdn.weather
6	1722675600	05:00	22	71.7	0	{"code":1000,"icon":"//cdn.weather
7	1722679200	06:00	22.1	71.8	0	{"code":1000,"icon":"//cdn.weather
8	1722682800	07:00	22.6	72.6	1	{"code":1003,"icon":"//cdn.weather
9	1722686400	08:00	23.3	74	1	{"code":1003,"icon":"//cdn.weather
10	1722690000	09:00	24	75.1	1	{"code":1003,"icon":"//cdn.weather
11	1722693600	10:00	24.6	76.3	1	{"code":1003,"icon":"//cdn.weather
12	1722697200	11:00	25	77.1	1	{"code":1003,"icon":"//cdn.weather
13	1722700800	12:00	25.6	78	1	{"code":1003,"icon":"//cdn.weather
14	1722704400	13:00	26.2	79.1	1	{"code":1000,"icon":"//cdn.weather
15	1722708000	14:00	26.8	80.2	1	{"code":1000,"icon":"//cdn.weather

240 rows

Visualization of Weather Data

10-Day Hourly Temperature Forecast: This visualization shows temperature trends over the 10-day period, highlighting hourly variations.

Temperature Trends: Max, Min, Avg: This combo line and bar graph displays the daily temperature extremes and averages, providing a clear overview of the temperature trends.

Total Precipitation: This Area graph presents the expected total precipitation for each day, helping to identify potential rainy days.

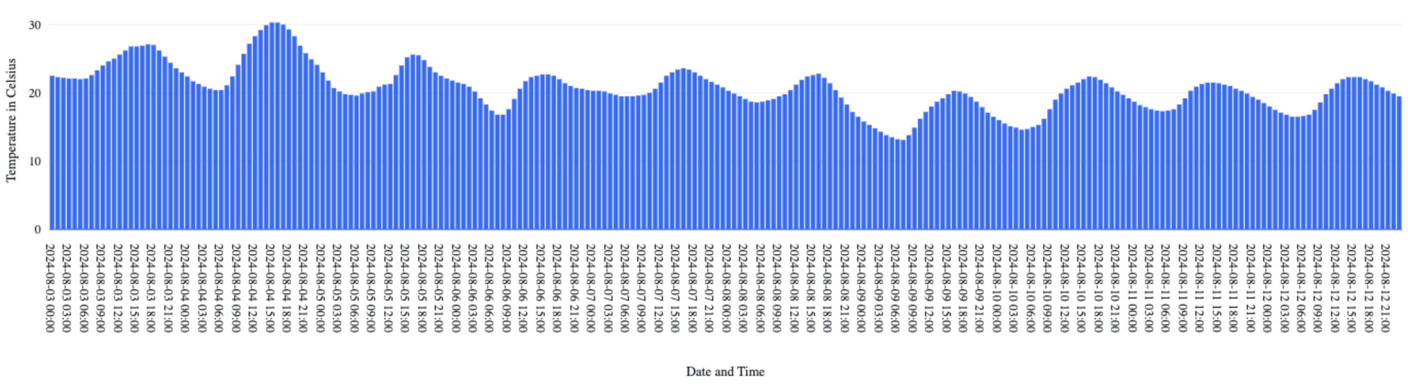
Average UV Index: This line graph shows the daily average UV index, which is crucial for assessing sun exposure risks.

Temperature and Windspeed Scatter Plot: A scatter plot was created to explore the relationship between temperature and wind speed.

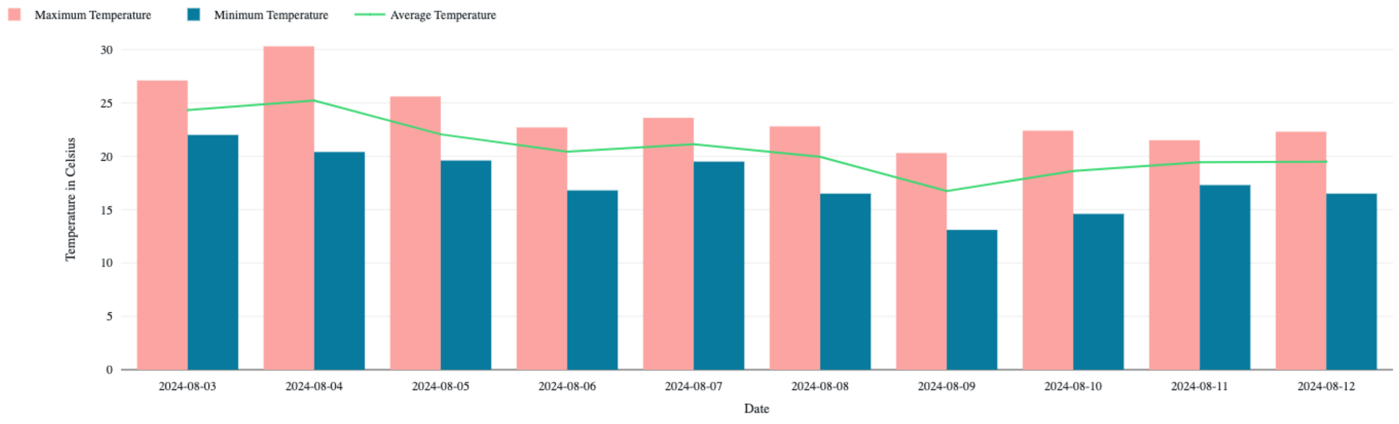
Heat Index Heatmap: This heatmap visualizes the heat index across the 10 days, illustrating the perceived temperature due to humidity and air temperature.

Air Quality Indices Boxplot: A boxplot was created for air quality indices (CO, O3, NO2, SO2, PM2.5, PM10), allowing for a comparison of pollution levels over the forecast period.

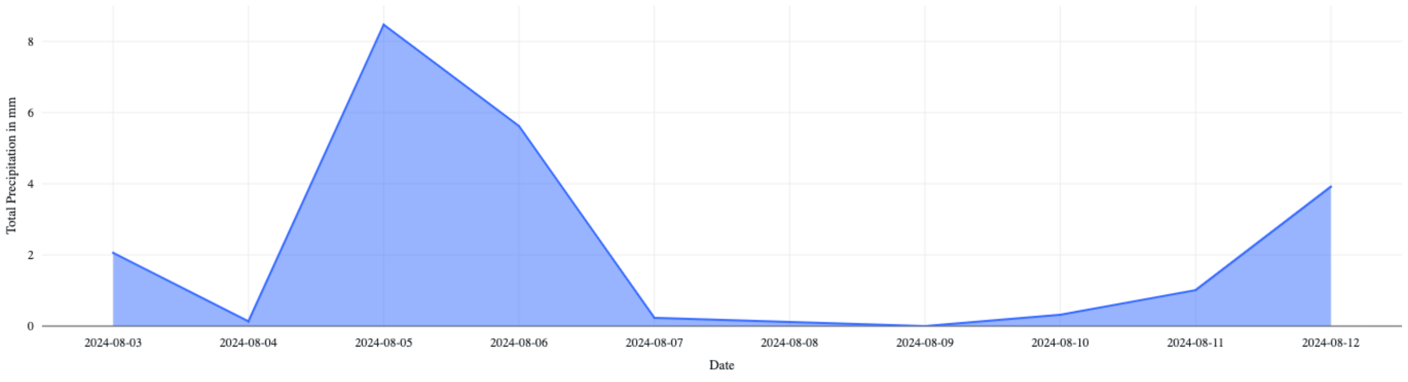
10-Day Hourly Temperature Forecast



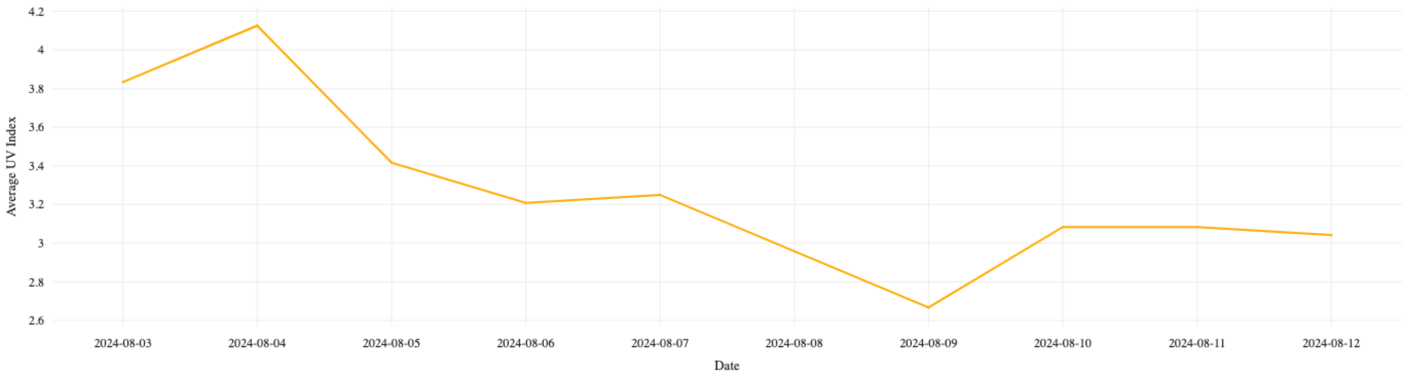
Temperature Trends: Max, Min, Avg



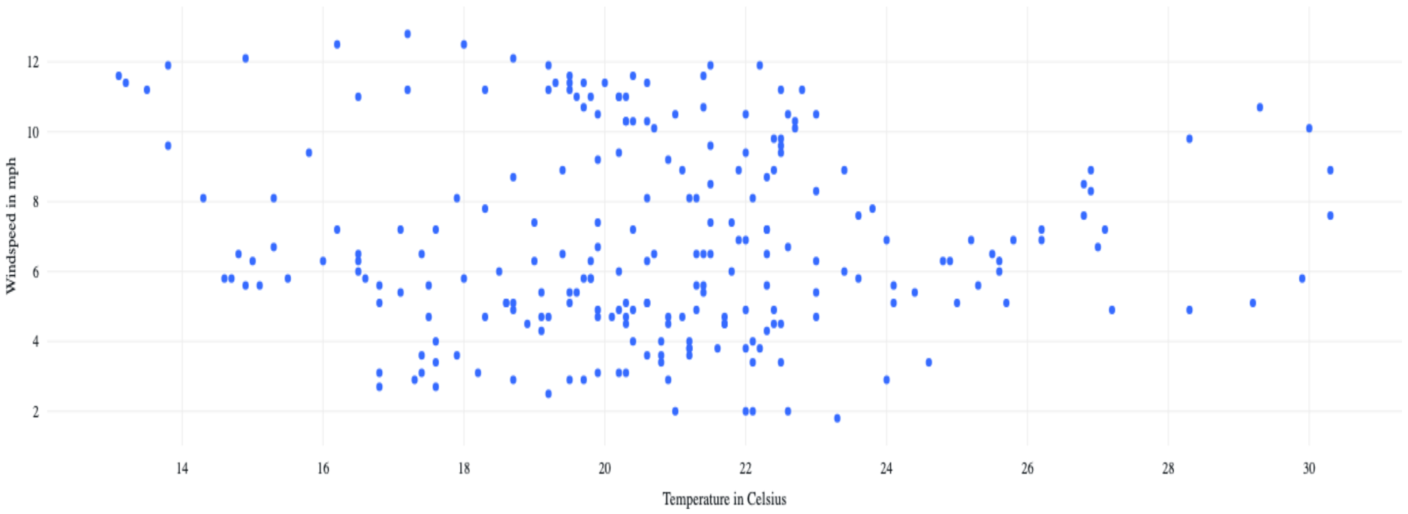
Total Precipitation



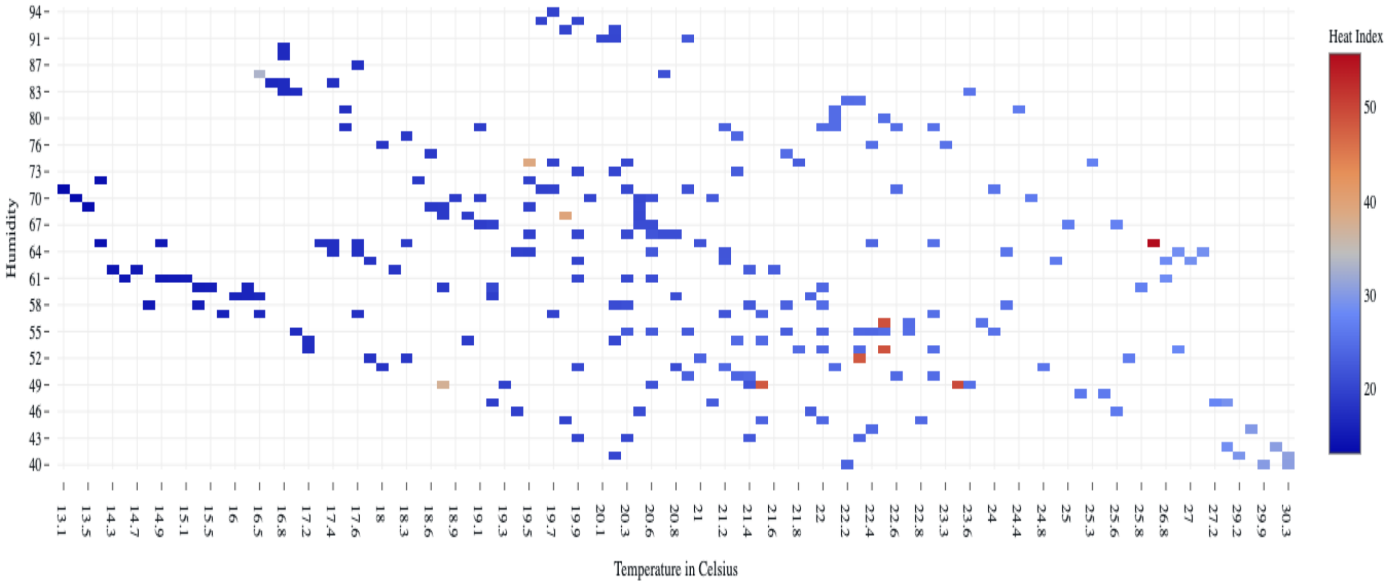
Average UV Index



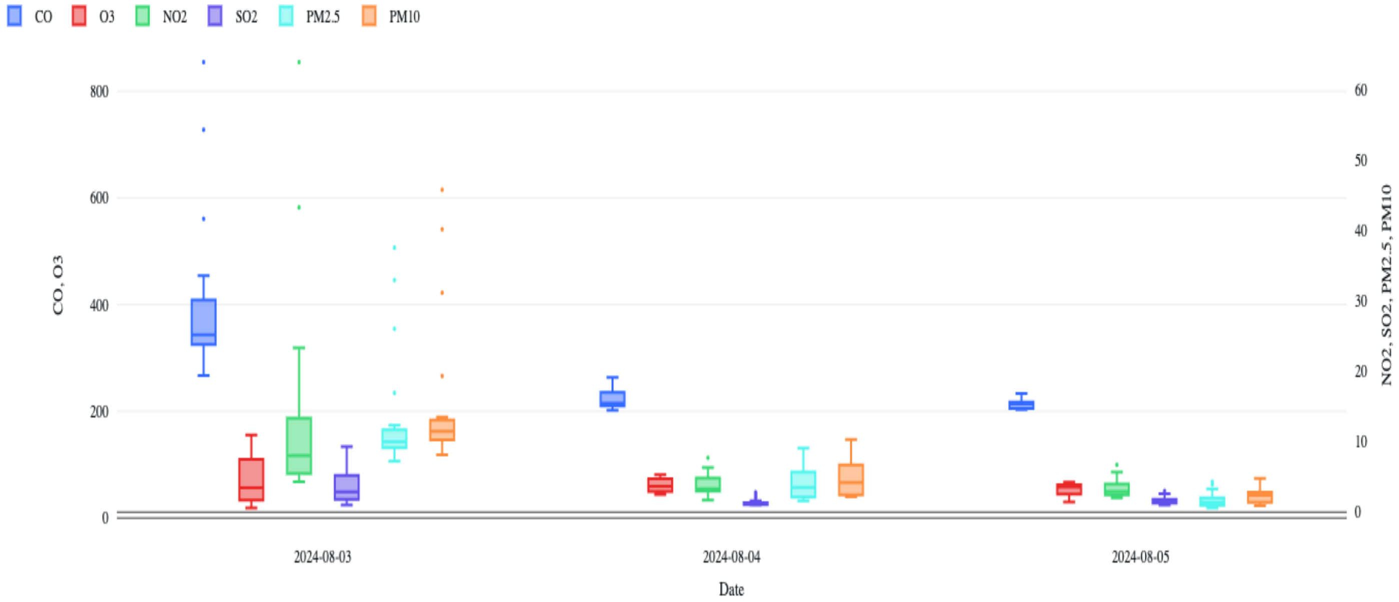
Temperature and Windspeed Scatter Plot



Heat Index Heatmap

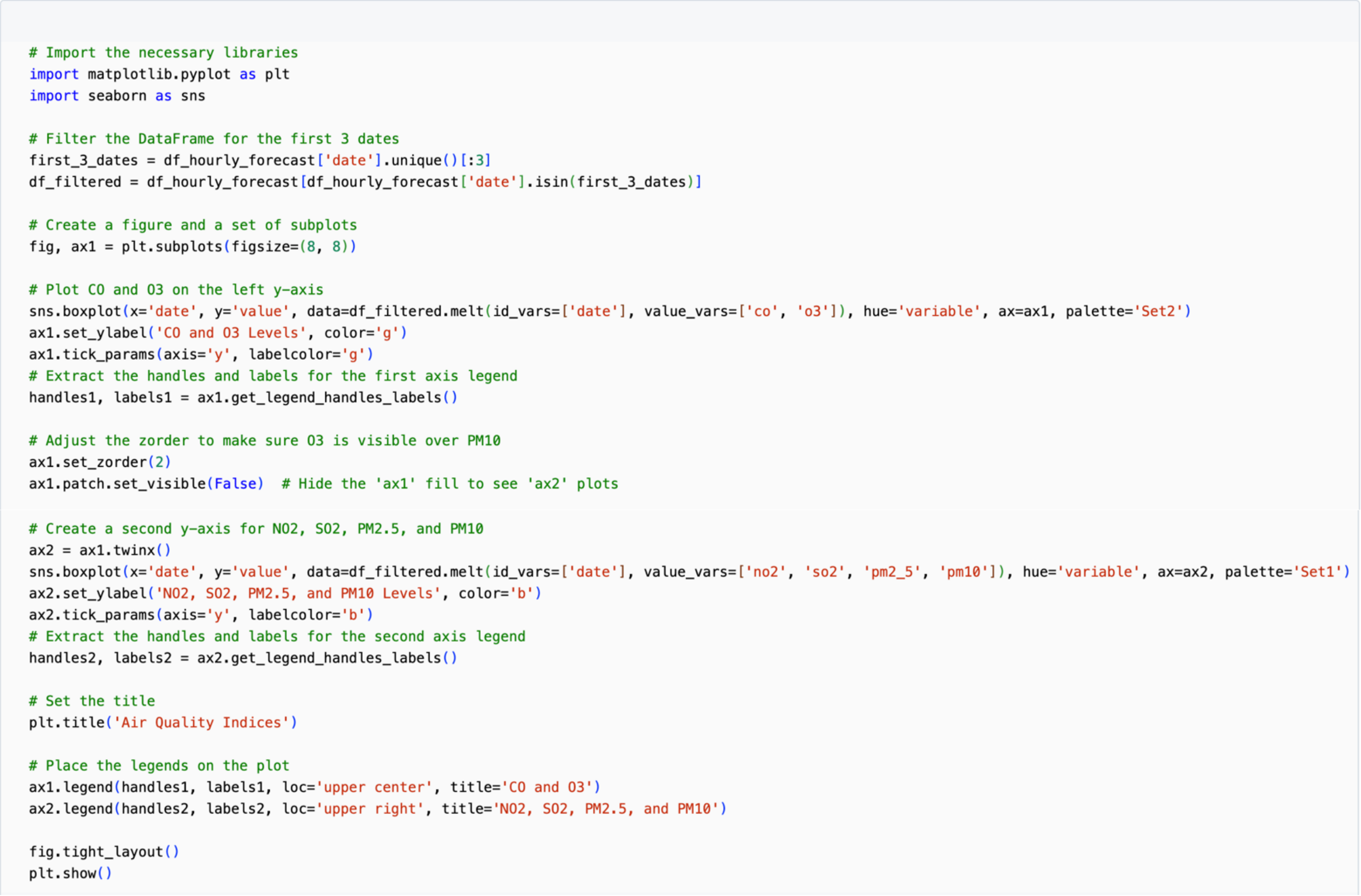


Air Quality Indices Boxplot

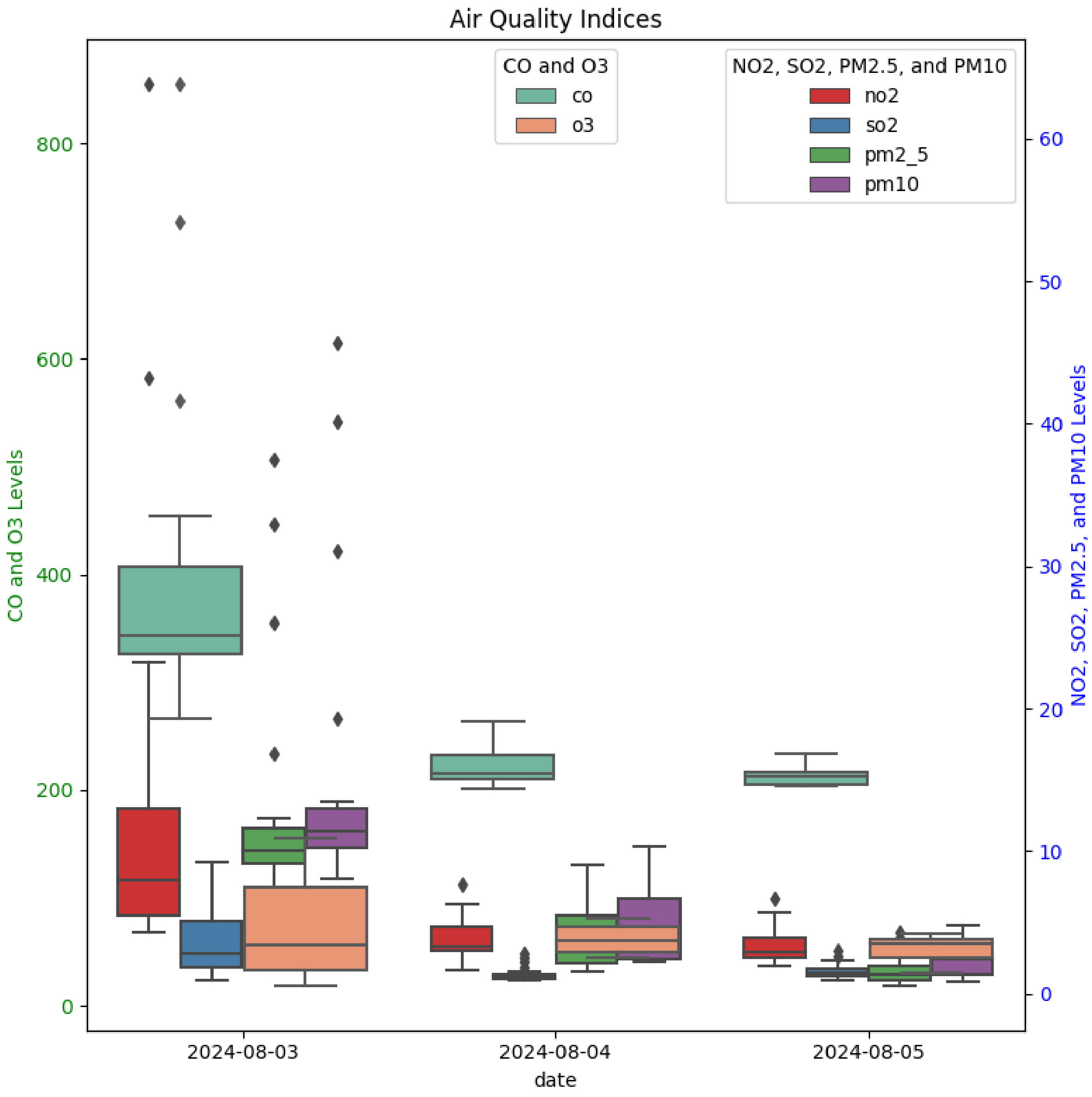


Advanced Visualization: Dual Y-Axis Boxplot of Air Quality Indices

Here, a dual y-axis boxplot has been created to analyse air quality indices over the first three days. CO and O3 levels are plotted on the left y-axis, while NO2, SO2, PM2.5 and PM10 levels are plotted on the right y-axis. Separate legends are provided for clarity. Adjustments are made to ensure that O3 levels are not obscured by PM10, ensuring all data points are visible.



```python
# Import the necessary libraries
import matplotlib.pyplot as plt
import seaborn as sns

# Filter the DataFrame for the first 3 dates
first_3_dates = df_hourly_forecast['date'].unique()[:3]
df_filtered = df_hourly_forecast[df_hourly_forecast['date'].isin(first_3_dates)]

# Create a figure and a set of subplots
fig, ax1 = plt.subplots(figsize=(8, 8))

# Plot CO and O3 on the left y-axis
sns.boxplot(x='date', y='value', data=df_filtered.melt(id_vars=['date'], value_vars=['co', 'o3']), hue='variable', ax=ax1, palette='Set2')
ax1.set_ylabel('CO and O3 Levels', color='g')
ax1.tick_params(axis='y', labelcolor='g')
# Extract the handles and labels for the first axis legend
handles1, labels1 = ax1.get_legend_handles_labels()

# Adjust the zorder to make sure O3 is visible over PM10
ax1.set_zorder(2)
ax1.patch.set_visible(False)  # Hide the 'ax1' fill to see 'ax2' plots

# Create a second y-axis for NO2, SO2, PM2.5, and PM10
ax2 = ax1.twinx()
sns.boxplot(x='date', y='value', data=df_filtered.melt(id_vars=['date'], value_vars=['no2', 'so2', 'pm2_5', 'pm10']), hue='variable', ax=ax2, palette='Set1')
ax2.set_ylabel('NO2, SO2, PM2.5, and PM10 Levels', color='b')
ax2.tick_params(axis='y', labelcolor='b')
# Extract the handles and labels for the second axis legend
handles2, labels2 = ax2.get_legend_handles_labels()

# Set the title
plt.title('Air Quality Indices')

# Place the legends on the plot
ax1.legend(handles1, labels1, loc='upper center', title='CO and O3')
ax2.legend(handles2, labels2, loc='upper right', title='NO2, SO2, PM2.5, and PM10')

fig.tight_layout()
plt.show()
```



Conclusion

This notebook demonstrates the end-to-end process of extracting, transforming, loading and analysing a 10-day weather forecast for Toronto. By utilizing various visualizations, we gain valuable insights into temperature trends, air quality and other weather-related factors. The dual y-axis boxplot is particularly useful for analyzing multiple air quality indices simultaneously, providing a clear picture of environmental conditions over the forecast period.