```
from google.colab import drive
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

drive.mount("/content/drive")
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

```
test = pd.read_csv("/content/drive/MyDrive/Course Work/Sem 4/Data
Analysis and Visualization/Homework 6/DailyDelhiClimateTest.csv")

train = pd.read_csv("/content/drive/MyDrive/Course Work/Sem 4/Data
Analysis and Visualization/Homework 6/DailyDelhiClimateTrain.csv")

train
```


```
{"summary":"{\n  \"name\": \"train\",\n  \"rows\": 1462,\n
\"fields\": [\n    {\n      \"column\": \"date\",\n
\"properties\": {\n        \"dtype\": \"object\",\n
\"num_unique_values\": 1462,\n        \"samples\": [\n
\"2015-06-12\",\n          \"2016-01-12\",\n          \"2014-02-18\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"meantemp\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
7.348102725432476,\n        \"min\": 6.0,\n        \"max\":
38.71428571428572,\n        \"num_unique_values\": 617,\n
\"samples\": [\n          19.0,\n          20.666666666666668,\n
28.714285714285715\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"humidity\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 16.769652268485302,\n        \"min\":
13.428571428571429,\n        \"max\": 100.0,\n
\"num_unique_values\": 897,\n        \"samples\": [\n          67.25,\
n          67.18181818181819,\n          76.125\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"wind_speed\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
4.561602164272007,\n        \"min\": 0.0,\n        \"max\": 42.22,\n
\"num_unique_values\": 730,\n        \"samples\": [\n
5.171428571428572,\n          6.483333333333333,\n          16.4375\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"meanpressure\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
180.2316683392097,\n        \"min\": -3.0416666666666665,\n
\"max\": 7679.333333333333,\n        \"num_unique_values\": 626,\n
\"samples\": [\n          1003.0625,\n          998.8125,\n
1012.8571428571428\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"train"}
```

```
test
```

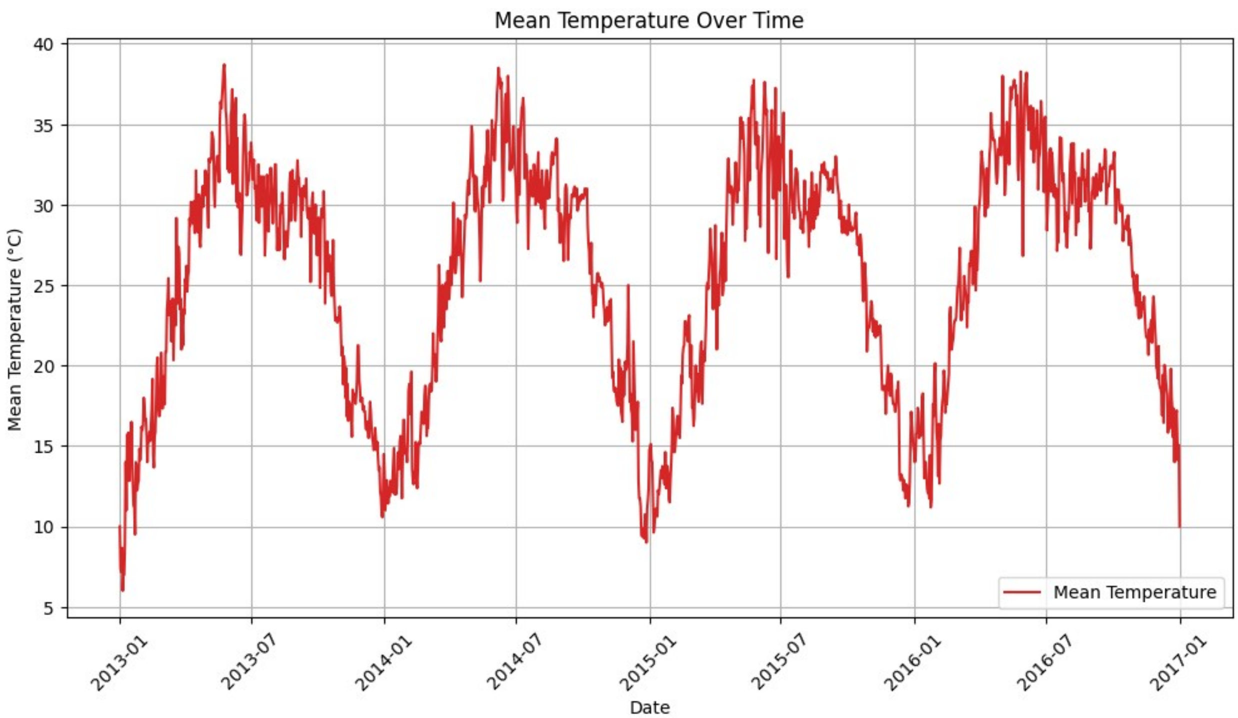

```
{"summary":"{\n  \"name\": \"test\",\n  \"rows\": 114,\n  \"fields\": [\n    {\n      \"column\": \"date\",\n      \"properties\": {\n        \"dtype\": \"object\",\n        \"num_unique_values\": 114,\n        \"samples\": [\n          \"2017-03-22\",\n          \"2017-01-05\",\n          \"2017-02-10\"\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"meantemp\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 6.360071848767034,\n        \"min\": 11.0,\n        \"max\": 34.5,\n        \"num_unique_values\": 105,\n        \"samples\": [\n          16.125,\n          19.9375,\n          20.785714285714285\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"humidity\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 19.068082852306677,\n        \"min\": 17.75,\n        \"max\": 95.83333333333331,\n        \"num_unique_values\": 109,\n        \"samples\": [\n          49.0,\n          72.11111111111111,\n          74.94444444444444\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"wind_speed\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 3.588049465473026,\n        \"min\": 1.3875000000000002,\n        \"max\": 19.314285714285717,\n        \"num_unique_values\": 109,\n        \"samples\": [\n          10.187500000000002,\n          9.772222222222222,\n          3.3000000000000003\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"meanpressure\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 89.47469204777894,\n        \"min\": 59.0,\n        \"max\": 1022.809523809524,\n        \"num_unique_values\": 109,\n        \"samples\": [\n          1011.75,\n          1016.7777777777778,\n          1014.3333333333334\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe","variable_name":"test"}
```


```
train["date"] = pd.to_datetime(train["date"])

test["date"] = pd.to_datetime(test["date"])
```

Basic Time Series Plot

```
# Plot Mean Temperature Over Time
plt.figure(figsize=(12, 6))
plt.plot(train["date"], train["meantemp"], label="Mean Temperature",
color="tab:red")

# Formatting
plt.xlabel("Date")
plt.ylabel("Mean Temperature (°C)")
```

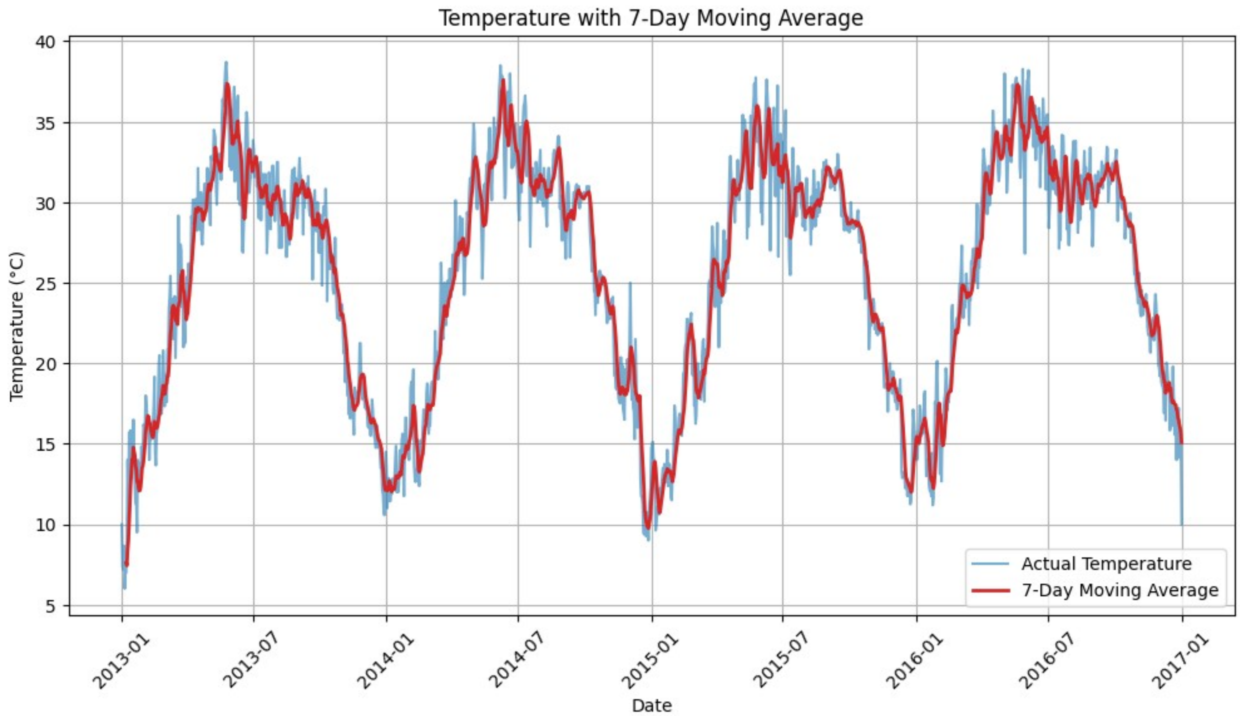
```
plt.title("Mean Temperature Over Time")
plt.legend()
plt.grid(True)
plt.xticks(rotation=45)

# Show plot
plt.show()
```



Rolling Average for Smoothing

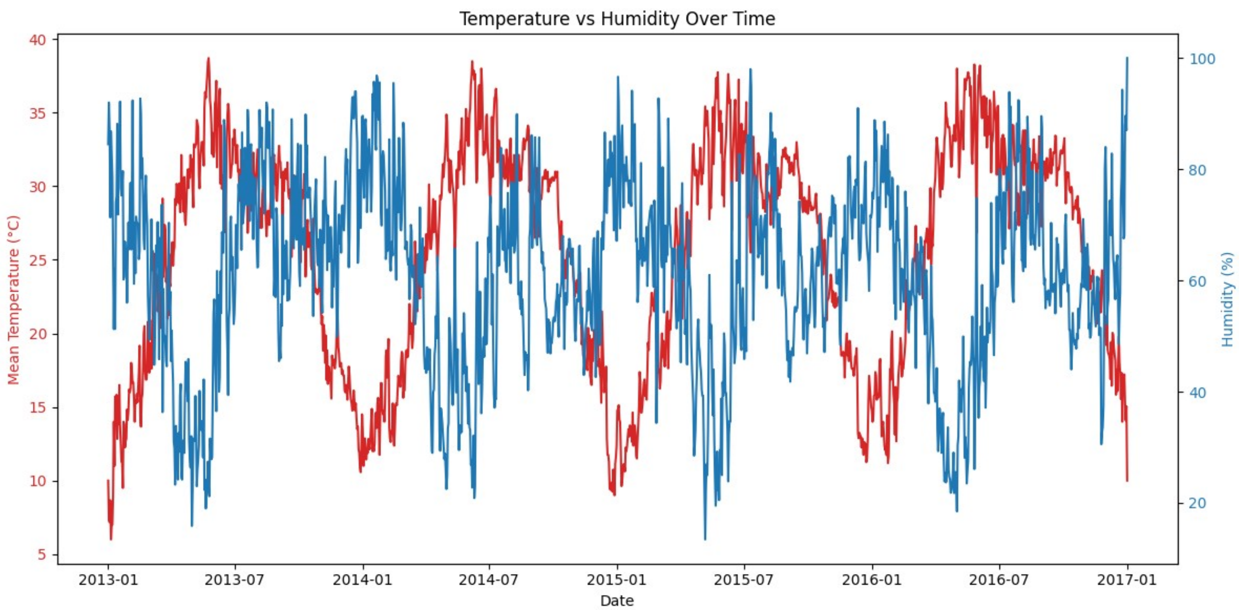
```
# Compute 7-day Moving Average
train["temp_rolling_avg"] = train["meantemp"].rolling(window=7).mean()

# Plot original vs smoothed data
plt.figure(figsize=(12, 6))
plt.plot(train["date"], train["meantemp"], label="Actual Temperature",
color="tab:blue", alpha=0.6)
plt.plot(train["date"], train["temp_rolling_avg"], label="7-Day Moving
Average", color="tab:red", linewidth=2)

# Formatting
plt.xlabel("Date")
plt.ylabel("Temperature (°C)")
plt.title("Temperature with 7-Day Moving Average")
plt.legend()
plt.grid(True)
plt.xticks(rotation=45)
```

```
plt.show()
```



Comparing Two Climate Variables

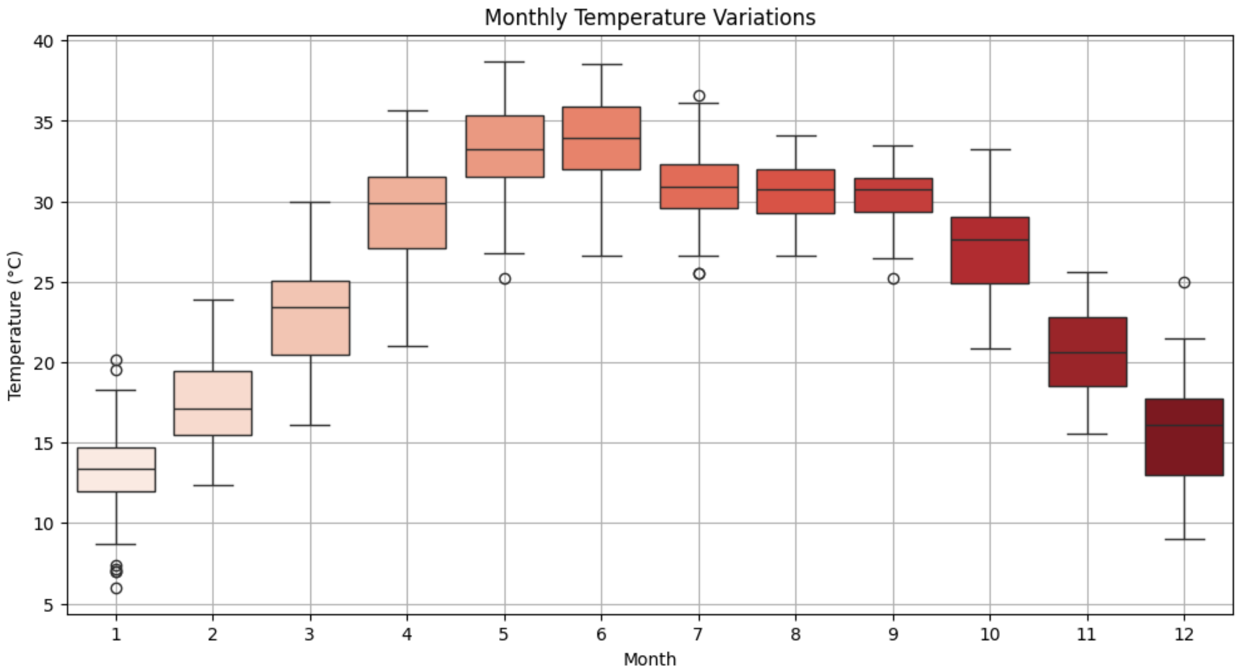
```
fig, ax1 = plt.subplots(figsize=(12, 6))

# Plot Temperature on the left axis
ax1.set_xlabel("Date")
ax1.set_ylabel("Mean Temperature (°C)", color="tab:red")
ax1.plot(train["date"], train["meantemp"], label="Temperature",
color="tab:red")
ax1.tick_params(axis="y", labelcolor="tab:red")

# Create second y-axis for Humidity
ax2 = ax1.twinx()
ax2.set_ylabel("Humidity (%)", color="tab:blue")
ax2.plot(train["date"], train["humidity"], label="Humidity",
color="tab:blue")
ax2.tick_params(axis="y", labelcolor="tab:blue")

plt.title("Temperature vs Humidity Over Time")
fig.tight_layout()
plt.show()
```



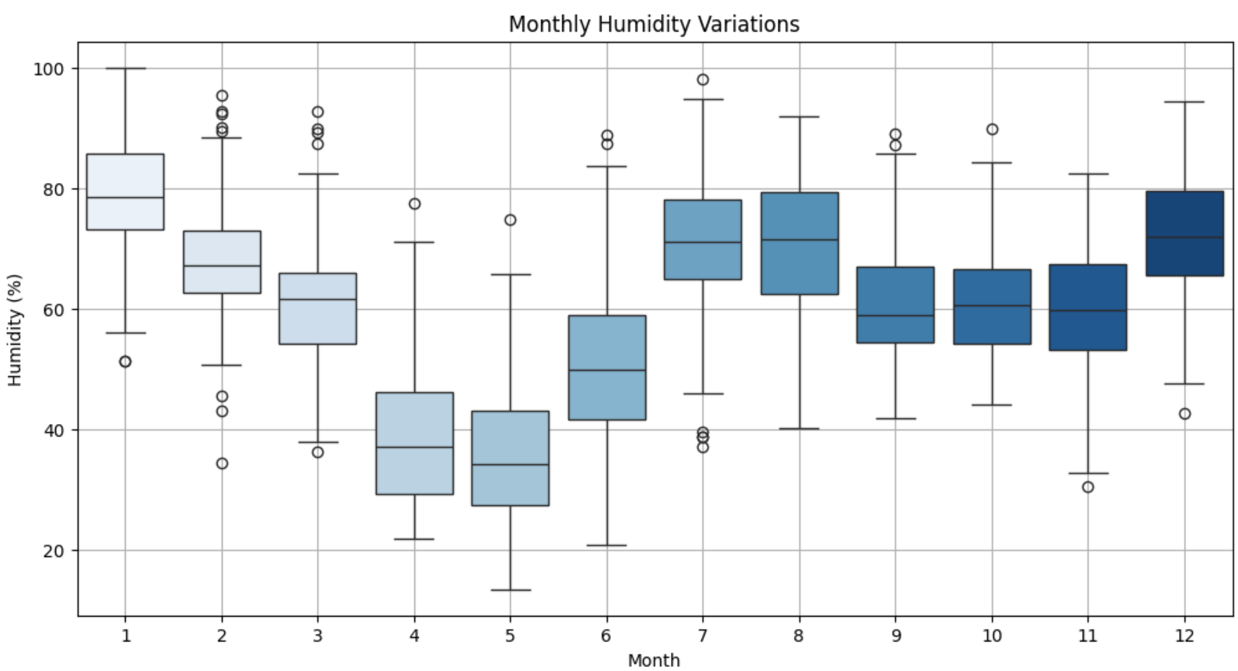
Seasonal Trend Analysis

```
# Extract month from date
train["month"] = train["date"].dt.month

# Boxplot for Temperature
plt.figure(figsize=(12, 6))
sns.boxplot(x="month", y="meantemp", data=train, palette="Reds")
plt.xlabel("Month")
plt.ylabel("Temperature (°C)")
plt.title("Monthly Temperature Variations")
plt.grid()
plt.show()

# Boxplot for Humidity
plt.figure(figsize=(12, 6))
sns.boxplot(x="month", y="humidity", data=train, palette="Blues")
plt.xlabel("Month")
plt.ylabel("Humidity (%)")
plt.title("Monthly Humidity Variations")
plt.grid()
plt.show()
```

```
<ipython-input-38-3ec6ef724a58>:6: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.boxplot(x="month", y="meantemp", data=train, palette="Reds")
```




```
<ipython-input-38-3ec6ef724a58>:15: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.boxplot(x="month", y="humidity", data=train, palette="Blues")
```



Detecting Weather Anomalies

```
# Function to detect anomalies using IQR
def detect_outliers_iqr(data, column):
    Q1 = data[column].quantile(0.25)
    Q3 = data[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    return data[(data[column] < lower_bound) | (data[column] >
upper_bound)]

# Detect outliers for temperature and humidity
temp_anomalies = detect_outliers_iqr(train, "meantemp")
humidity_anomalies = detect_outliers_iqr(train, "humidity")

# Plot anomalies on temperature chart
plt.figure(figsize=(12, 6))
plt.plot(train["date"], train["meantemp"], label="Mean Temperature",
color="tab:blue", alpha=0.6)
plt.scatter(temp_anomalies["date"], temp_anomalies["meantemp"],
color="red", label="Anomalies", zorder=3)
plt.xlabel("Date")
plt.ylabel("Temperature (°C)")
plt.title("Temperature Anomalies Detected")
plt.legend()
plt.grid(True)
plt.show()
```