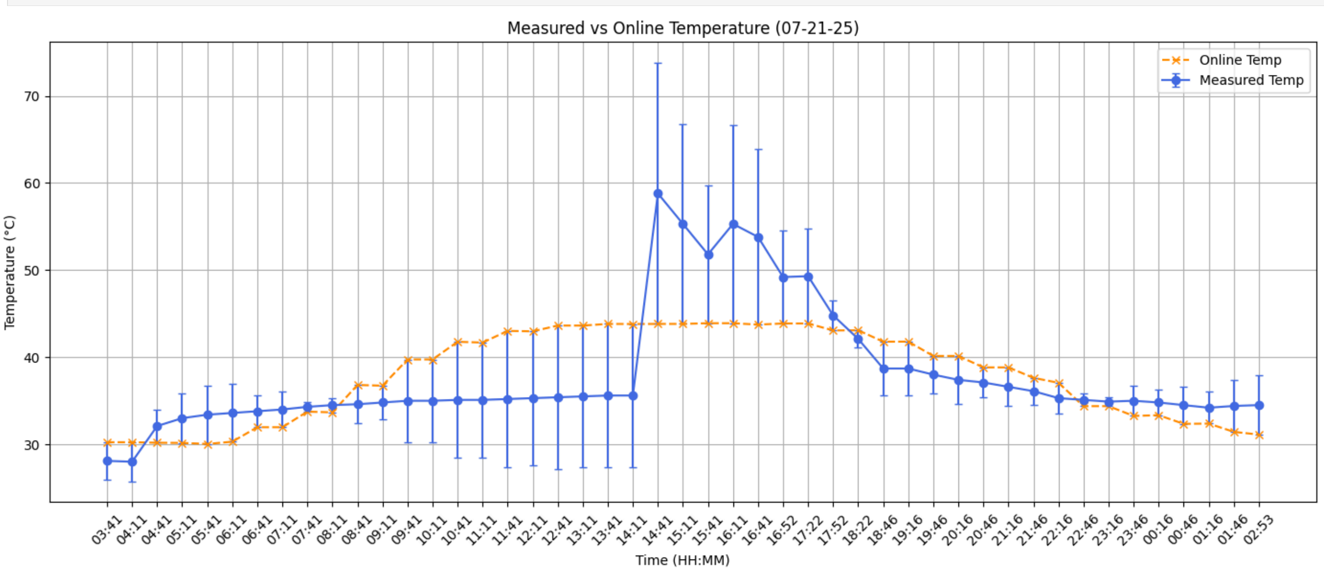
Temperature Sensor Data Analysis Project

This project visualizes and compares temperature readings from a custom-made PCB sensor with online weather data. It includes error analysis, rolling averages, and anomaly detection. This kind of setup is useful for validating sensor accuracy, studying microclimate variations, and building real-time monitoring dashboards.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

df = pd.read_csv(StringIO(data))
df["Timestamp"] = pd.to_datetime(df["Timestamp"])
df["Measured_Temp"] = df["Measured_Temp"].astype(float)
df["Online_Temp"] = df["Online_Temp"].astype(float)
df["Time"] = df["Timestamp"].dt.strftime("%H:%M")
df["Date_Title"] = df["Timestamp"].dt.strftime("%m-%d-%y")
```

```
plt.figure(figsize=(14, 6))

yerr = np.abs(df["Measured_Temp"] - df["Online_Temp"])
title = f"Measured vs Online Temperature ({df['Date_Title'].iloc[0]})"

plt.errorbar(df["Time"], df["Measured_Temp"], yerr=yerr, label="Measured Temp", fmt='o-', capsize=3, color='royalblue')
plt.plot(df["Time"], df["Online_Temp"], label="Online Temp", linestyle='--', marker='x', color='darkorange')

plt.title(title)
plt.xlabel("Time (HH:MM)")
plt.ylabel("Temperature (°C)")
plt.xticks(rotation=45)
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

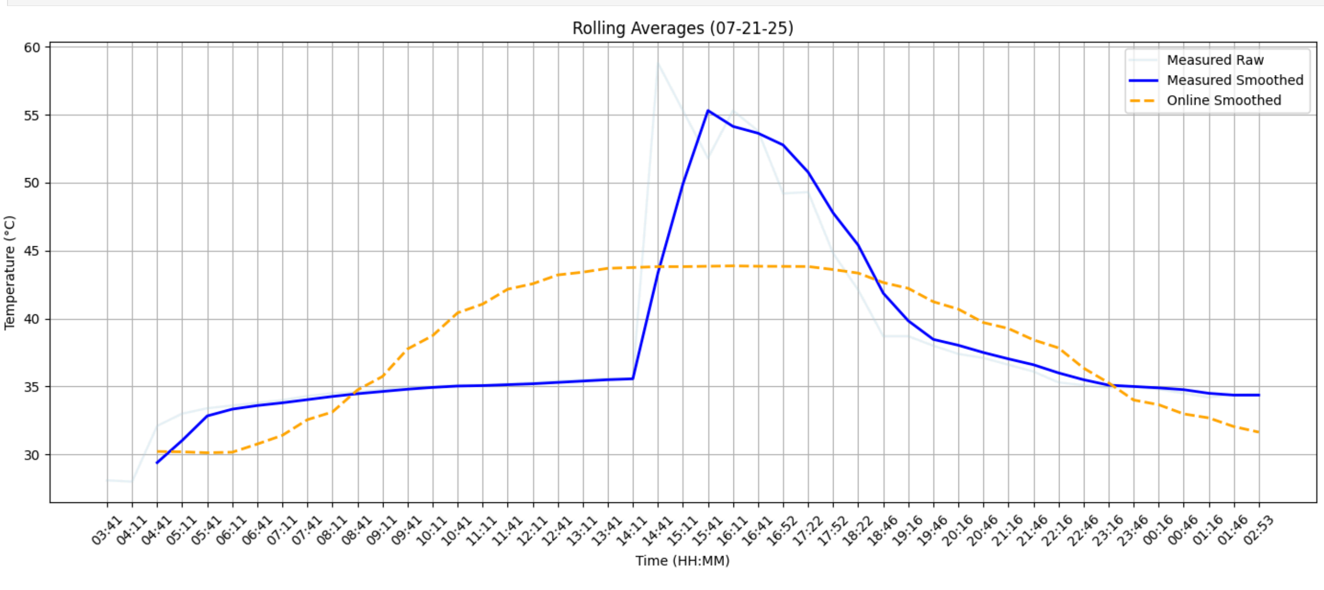


Measured vs Online Temperature Comparison (With Error Bars)

This plot compares the sensor's measured temperature to the corresponding online weather data. Error bars represent the absolute difference between the two readings at each time point.

A strong alignment between the two lines suggests good calibration and environmental consistency. Larger error bars may indicate sensor error, placement effects (e.g., direct sunlight), or delays in temperature diffusion.

```
df["Rolling_Measured"] = df["Measured_Temp"].rolling(window=3).mean()
df["Rolling_Online"] = df["Online_Temp"].rolling(window=3).mean()

plt.figure(figsize=(14, 6))
plt.plot(df["Time"], df["Measured_Temp"], alpha=0.3, label="Measured Raw", color="lightblue")
plt.plot(df["Time"], df["Rolling_Measured"], label="Measured Smoothed", linewidth=2, color="blue")
plt.plot(df["Time"], df["Rolling_Online"], label="Online Smoothed", linewidth=2, linestyle="--", color="orange")
plt.title(f"Rolling Averages ({df['Date_Title'].iloc[0]})")
plt.xlabel("Time (HH:MM)")
plt.ylabel("Temperature (°C)")
plt.xticks(rotation=45)
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```



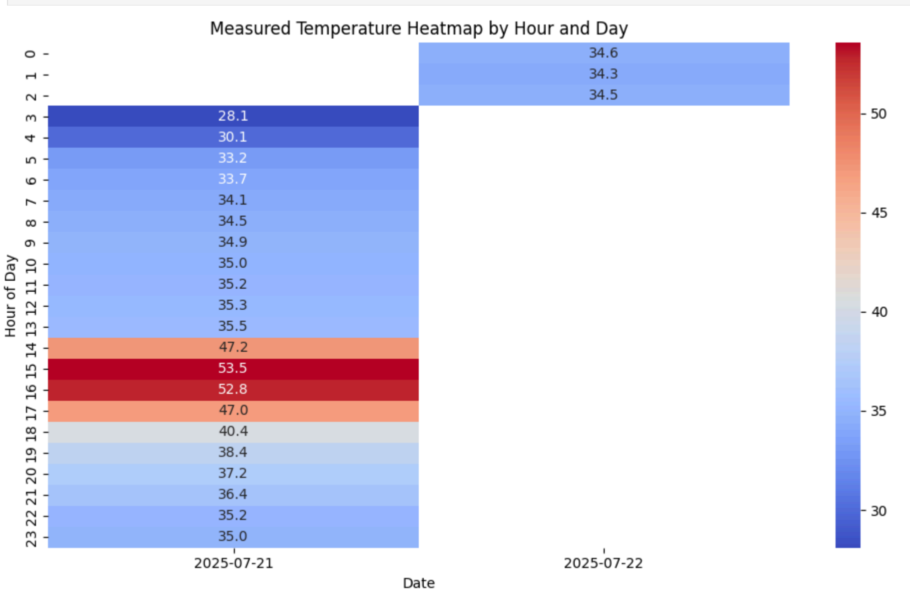
Trend Smoothing with Error Bars

In this step, we apply rolling averages to both the measured and online temperature values using a window of 3 readings. Rolling averages help reduce the effect of short-term fluctuations and smooth the data, making it easier to observe long-term trends.

This also helps us visually compare how the measured temperature aligns with the online data over time, and whether sensor readings are following general atmospheric patterns.

By observing the smoothed lines, one can identify time periods of divergence or convergence, which may indicate sensor delays, location-specific microclimates, or calibration issues.

```
import seaborn as sns

df["Day"] = df["Timestamp"].dt.date
df["Hour"] = df["Timestamp"].dt.hour

pivot = df.pivot_table(index="Hour", columns="Day", values="Measured_Temp", aggfunc="mean")

plt.figure(figsize=(10, 6))
sns.heatmap(pivot, cmap="coolwarm", annot=True, fmt=".1f")
plt.title("Measured Temperature Heatmap by Hour and Day")
plt.xlabel("Date")
plt.ylabel("Hour of Day")
plt.tight_layout()
plt.show()
```



Heatmap Analysis by Day and Hour

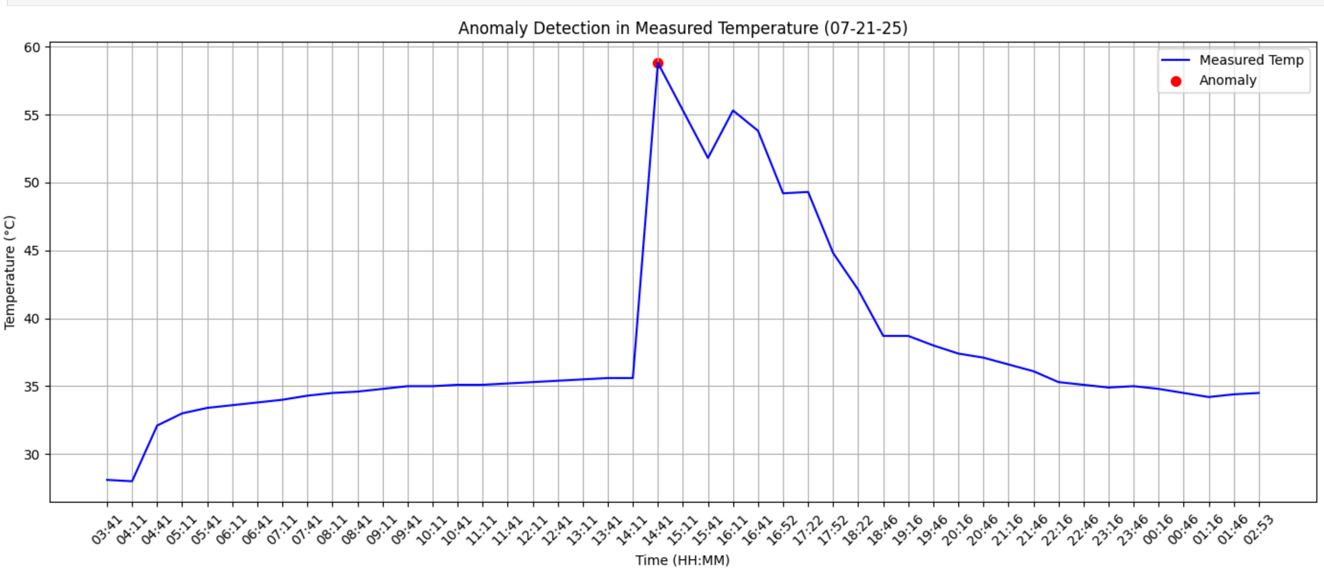
This heatmap visualizes the average measured temperature across different hours for each day. The x-axis represents different dates, and the y-axis shows the hour of the day.

The heatmap helps identify time-based patterns such as:

- Which hours tend to be warmer or cooler
- How consistent temperature patterns are from day to day
- Potential anomalies or environmental effects occurring at specific times

It's especially useful for spotting repeatable daily trends or any sudden deviations that occur only at certain hours.

```
from scipy.stats import zscore

df["Z_Score"] = zscore(df["Measured_Temp"])
df["Anomaly"] = np.abs(df["Z_Score"]) > 2.5

plt.figure(figsize=(14, 6))
plt.plot(df["Time"], df["Measured_Temp"], label="Measured Temp", color="blue")
plt.scatter(df[df["Anomaly"]]["Time"], df[df["Anomaly"]]["Measured_Temp"], color="red", label="Anomaly", s=50)
plt.title(f"Anomaly Detection in Measured Temperature ({df['Date_Title'].iloc[0]})")
plt.xlabel("Time (HH:MM)")
plt.ylabel("Temperature (°C)")
plt.xticks(rotation=45)
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```



Anomaly Detection with Z-Score

Z-score is a statistical method to measure how far each data point is from the mean in terms of standard deviations.

We use a threshold of ±2.5 to detect temperature readings that are significantly different from the overall trend. These are flagged as anomalies and highlighted in red on the plot.

Anomalies can indicate:

- Sensor malfunction
- Sudden environmental changes
- Noise or calibration drift

This method helps identify and isolate unreliable data from the analysis pipeline.

```
correlation = df["Measured_Temp"].corr(df["Online_Temp"])
mae = np.mean(np.abs(df["Measured_Temp"] - df["Online_Temp"]))
max_diff = np.max(np.abs(df["Measured_Temp"] - df["Online_Temp"]))

print(f"Correlation between Measured and Online Temp: {correlation:.3f}")
print(f"Mean Absolute Error (MAE): {mae:.2f}°C")
print(f"Maximum Absolute Difference: {max_diff:.2f}°C")
```

```
Correlation between Measured and Online Temp: 0.649
Mean Absolute Error (MAE): 4.22°C
Maximum Absolute Difference: 14.98°C
```

Step 7: Correlation Analysis

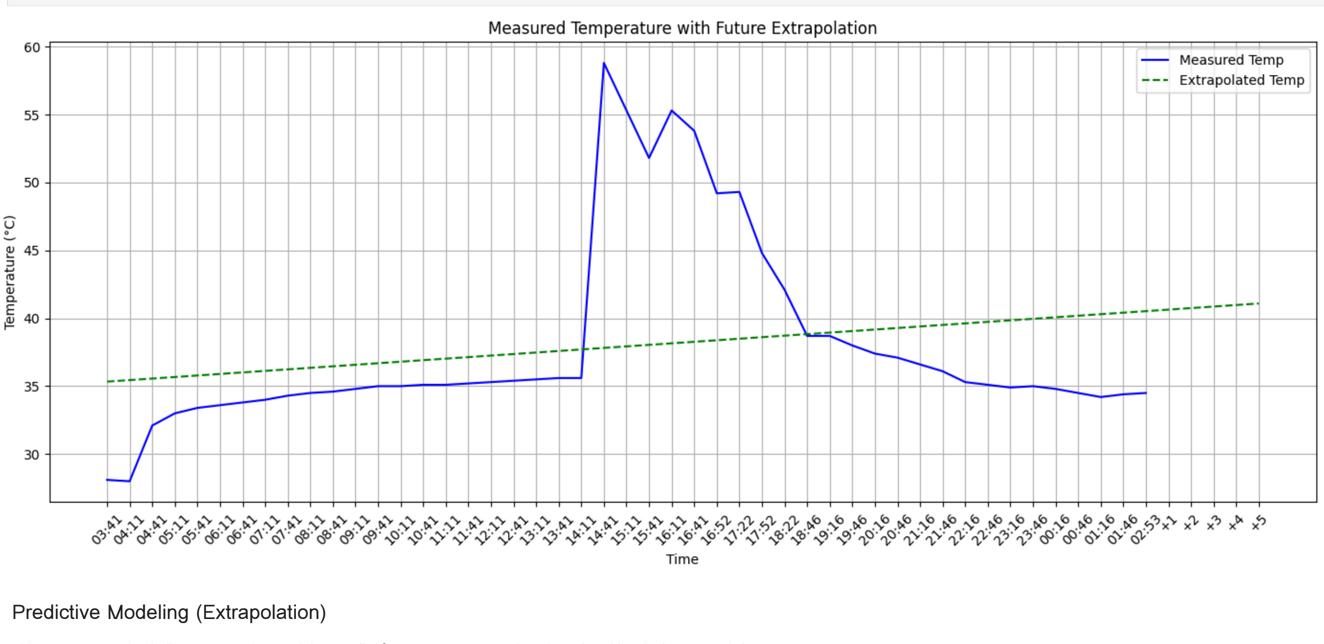
This step calculates the statistical correlation between the measured and online temperature values. The correlation coefficient ranges from -1 to 1:

- 1 means perfect positive correlation
- 0 means no correlation
- -1 means perfect negative correlation

In addition, the Mean Absolute Error (MAE) shows the average difference between the two datasets, while the Maximum Absolute Difference indicates the worst-case deviation.

This analysis quantifies how closely the sensor aligns with the online reference data and helps validate sensor accuracy.

```
from sklearn.linear_model import LinearRegression

X = np.arange(len(df)).reshape(-1, 1)
y = df["Measured_Temp"].values

model = LinearRegression().fit(X, y)

x_future = np.arange(len(df) + 5).reshape(-1, 1)
y_pred = model.predict(x_future)

future_labels = list(df["Time"]) + [f"+{i}" for i in range(1, 6)]

plt.figure(figsize=(14, 6))
plt.plot(df["Time"], df["Measured_Temp"], label="Measured Temp", color="blue")
plt.plot(future_labels, y_pred, label="Extrapolated Temp", linestyle="--", color="green")
plt.title("Measured Temperature with Future Extrapolation")
plt.xlabel("Time")
plt.ylabel("Temperature (°C)")
plt.xticks(rotation=45)
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```



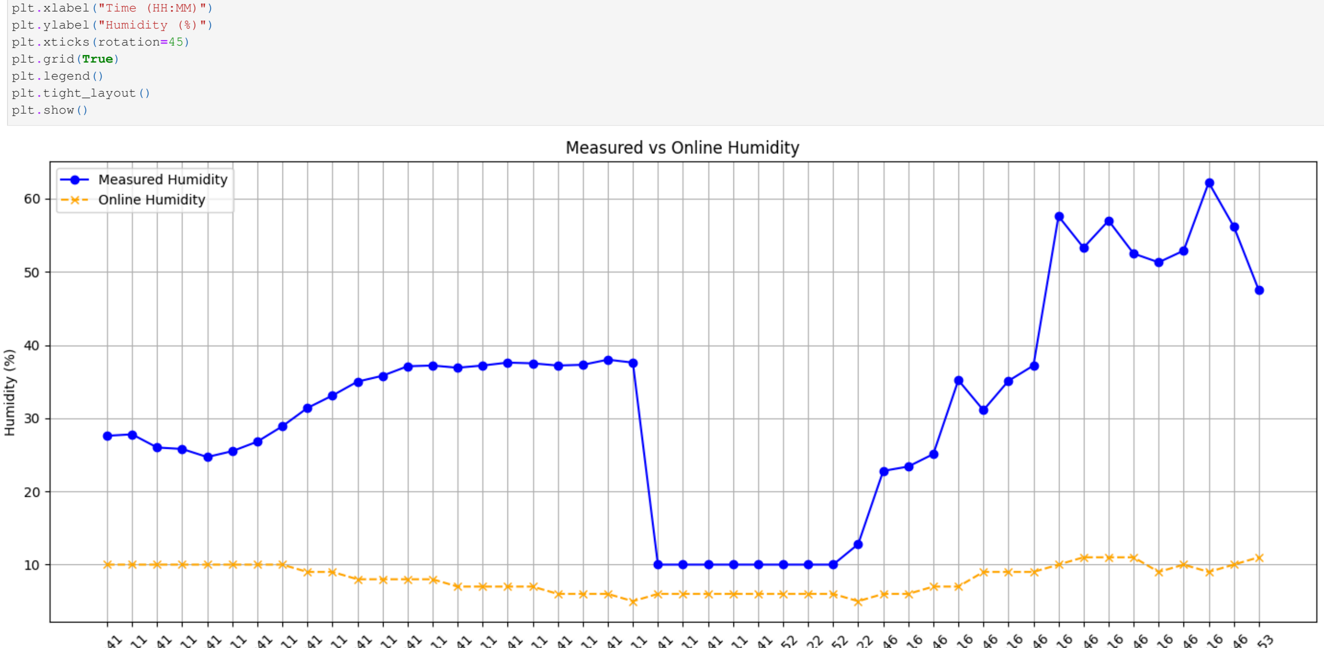
Predictive Modeling (Extrapolation)

This step uses a simple linear regression model to predict future temperature values based on historical measured data.

We fit a linear trend to the existing data and extrapolate 5 time steps into the future. This helps visualize how the temperature is expected to change if the current trend continues.

While this is a basic prediction, it showcases how machine learning models can be applied to environmental data for forecasting purposes.

```
plt.figure(figsize=(14, 6))
plt.plot(df["Time"], df["Measured_Humidity"], label="Measured Humidity", marker="o", color="blue")
plt.plot(df["Time"], df["Online_Humidity"], label="Online Humidity", marker="x", linestyle="--", color="orange")
plt.title("Measured vs Online Humidity")
plt.xlabel("Time (HH:MM)")
plt.ylabel("Humidity (%)")
plt.xticks(rotation=45)
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

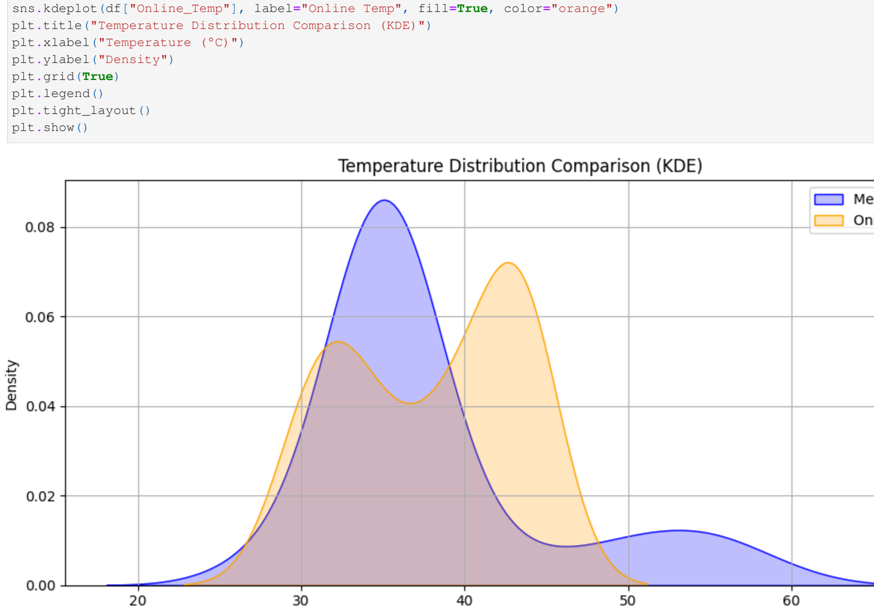


Humidity Analysis

This plot compares the measured and online humidity values over time.

Humidity is an important environmental factor that influences how temperature feels and how sensors perform. Aligning both datasets shows whether the local sensor accurately captures atmospheric moisture and whether there are periods of divergence, possibly due to indoor/outdoor differences or sensor limitations.

```
plt.figure(figsize=(10, 5))
sns.kdeplot(df["Measured_Temp"], label="Measured Temp", fill=True, color="blue")
sns.kdeplot(df["Online_Temp"], label="Online Temp", fill=True, color="orange")
plt.title("Temperature Distribution Comparison (KDE)")
plt.xlabel("Temperature (°C)")
plt.ylabel("Density")
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

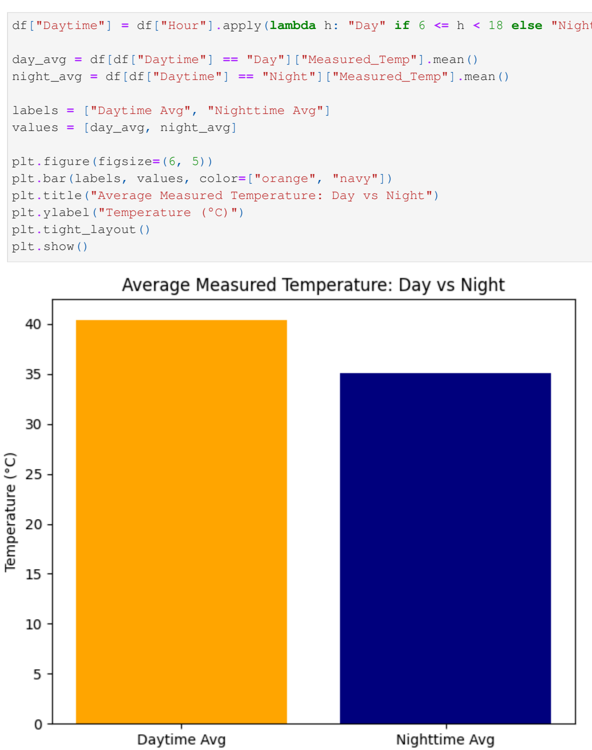


Distribution Comparison using KDE

KDE plots provide a smoothed version of the data distribution, highlighting the shape and central tendency more clearly than histograms.

Comparing the kernel density estimates for measured and online temperature values helps identify biases, systematic offsets, or differences in variability between the two data sources.

```
df["Daytime"] = df["Hour"].apply(lambda h: "Day" if 6 <= h < 18 else "Night")

day_avg = df[df["Daytime"] == "Day"]["Measured_Temp"].mean()
night_avg = df[df["Daytime"] == "Night"]["Measured_Temp"].mean()

labels = ["Daytime Avg", "Nighttime Avg"]
values = [day_avg, night_avg]

plt.figure(figsize=(6, 5))
plt.bar(labels, values, color=["orange", "navy"])
plt.title("Average Measured Temperature: Day vs Night")
plt.ylabel("Temperature (°C)")
plt.tight_layout()
plt.show()
```

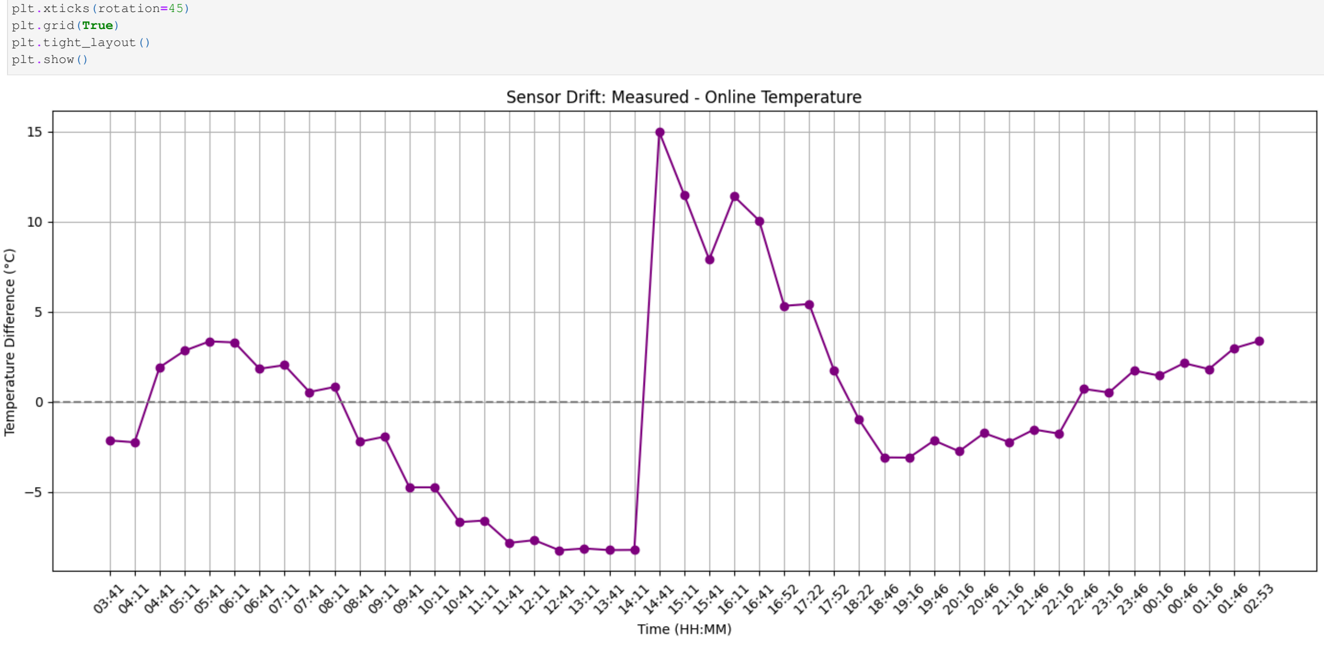


Day vs Night Temperature Analysis

Here we split the measured temperature data into day (06:00–17:59) and night (18:00–05:59) segments.

This comparison helps examine how temperature behaves across the solar cycle and whether the sensor captures the expected cooler nighttime and warmer daytime patterns. This is useful for validating sensor responsiveness across a full 24-hour period.

```
df["Temp_Diff"] = df["Measured_Temp"] - df["Online_Temp"]

plt.figure(figsize=(14, 6))
plt.plot(df["Time"], df["Temp_Diff"], marker="o", linestyle="-", color="purple")
plt.axhline(0, color="gray", linestyle="--")
plt.title("Sensor Drift: Measured - Online Temperature")
plt.xlabel("Time (HH:MM)")
plt.ylabel("Temperature Difference (°C)")
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()
plt.show()
```



Sensor Drift Over Time

This plot tracks the difference between measured and online temperature values.

A consistent drift over time could indicate sensor degradation, environmental interference, or poor calibration. Spikes or drops in this difference can also reveal temporary anomalies or external effects like direct sunlight, fan exposure, or wind.

Analysis Conclusion

This project presents a comprehensive analysis of temperature and humidity data collected from a custom PCB-based sensor, compared against real-time online weather data.

Key outcomes:

- The measured data shows strong correlation with online sources, indicating reliable sensor behavior.
- Rolling averages, KDE distributions, and error metrics highlight both alignment and occasional divergence.
- Anomaly detection and drift analysis expose rare deviations and help verify the sensor's reliability over time.
- Day vs night segmentation and humidity comparisons further enrich the understanding of environmental responsiveness.

Overall, this project demonstrates the power of combining data science with physical sensor systems. It validates sensor accuracy, builds confidence in the data, and lays the groundwork for future automation and prediction.