# **W9D5 -- Time Series ML: Weather Forecasting**

JTC Program: Tech Pathways Cohort: S25 Lesson Plan: Time Series ML - Weather Forecasting Type: Lesson Plan Week / Day: W9D5 Version Date: 06/04/2025

## **Focus Concepts**

* Understanding what makes time series data unique and its applications in weather forecasting
* Preparing and preprocessing weather time series data for analysis
* Exploring techniques for visualizing time series patterns and seasonality in weather data
* Building forecasting models for weather prediction using various approaches
* Evaluating forecast accuracy with appropriate time series metrics
* Applying machine learning to weather time series data for practical applications

## **Learning Objectives**

By the end of this session, fellows will be able to:

* Explain the unique characteristics of time series data in the context of weather forecasting
* Properly preprocess weather time series data, handling missing values and timestamps
* Create effective visualizations to identify patterns in weather data
* Implement both classical time series models and machine learning approaches for forecasting
* Evaluate forecast accuracy using appropriate time series metrics
* Apply time series forecasting concepts to predict temperature

## **Out-of-Scope Objectives**

* Advanced deep learning techniques for time series (LSTM, GRU)
* Multivariate forecasting with complex dependencies
* Ensemble methods for forecast combination
* Production deployment of weather forecasting models
* Advanced statistical concepts behind time series models
* Image-based weather forecasting (satellite imagery)

## **Required Competencies**

* Understanding of basic machine learning concepts (from W9D1-W9D4)
* Familiarity with Python, NumPy, Pandas, and Matplotlib
* Experience with basic data preprocessing and feature engineering
* Comfort with basic statistics concepts
* Understanding of regression modeling concepts

## **Technical Requirements**

* Python 3.x installed
* Jupyter Notebook or code editor
* Required libraries: NumPy, Pandas, Matplotlib, scikit-learn, statsmodels
* Weather dataset (will be provided)

## **Prerequisites**

* Completion of W9D1: ML Foundations
* Completion of W9D2: Basic Regression
* Completion of W9D3: Model Evaluation
* Completion of W9D4: Feature Engineering
* Understanding of the machine learning workflow

## **Assigned Reading & Pre-Class Learning**

Estimated Time: 20 minutes

Resources:

* [Time Series Forecasting with Python (Machine Learning Mastery)](https://machinelearningmastery.com/time-series-forecasting-methods-in-python-cheat-sheet/) - Overview of time series forecasting methods - 10 minutes
* [Introduction to Weather Forecasting with Machine Learning](https://towardsdatascience.com/weather-forecasting-with-machine-learning-using-python-55e90c346647) - Tutorial on weather forecasting approaches - 10 minutes

## **Before-Class Mini Quiz Questions (5 questions)**

1. What makes time series data different from regular cross-sectional data?
   * A) Time series data always has more features
   * \*B) Time series data points have a chronological order and are often correlated with previous values
   * C) Time series data is always collected at equal intervals
   * D) Time series data is easier to predict
2. Which of these is a common challenge in weather time series data?
   * A) Too few data points to analyze
   * B) Lack of seasonality and patterns
   * \*C) Missing values and irregular recording intervals
   * D) Absence of historical records
3. What is a lag feature in time series analysis?
   * A) A feature that is always late or delayed
   * \*B) A feature created from past values of the target variable
   * C) A feature that measures the time between events
   * D) A feature that can be removed from the analysis
4. Why is using a simple train-test split often inappropriate for time series?
   * A) Time series data needs to be normalized first
   * \*B) It can lead to data leakage because future data shouldn't be used to predict past data
   * C) It's too computationally expensive
   * D) Time series data cannot be split
5. Which metric is commonly used to evaluate time series forecasts?
   * A) AUC-ROC
   * B) Confusion matrix
   * \*C) Mean Absolute Error (MAE)
   * D) F1-Score

## **Key Terms**

* **Time Series**: Sequential data points collected or recorded at specific time intervals
* **Seasonality**: Regular patterns that repeat at fixed intervals in time series data
* **Trend**: Long-term increase or decrease in the data
* **Stationarity**: Property where statistical properties (mean, variance) don't change over time
* **Lag Feature**: Feature created using past values of a variable
* **Autocorrelation**: Correlation of a signal with a delayed copy of itself
* **Forecasting Horizon**: How far into the future you're trying to predict
* **Moving Average**: Average calculated over a specific window of time periods
* **Exponential Smoothing**: Method that applies exponentially decreasing weights to past observations
* **ARIMA**: AutoRegressive Integrated Moving Average, a classical time series model
* **Resampling**: Converting time series data from one frequency to another
* **Cross-Validation**: Technique for assessing model performance with time series data
* **Feature Engineering**: Creating new features from existing time series data
* **Cyclical Features**: Features that capture repeating patterns (daily, weekly, seasonal)
* **Rolling Statistics**: Statistics calculated over a moving window of time
* **Time Series Decomposition**: Breaking a time series into trend, seasonality, and residual components
* **MAE**: Mean Absolute Error, a common metric for evaluating forecasts
* **RMSE**: Root Mean Squared Error, a forecast evaluation metric that penalizes large errors
* **MAPE**: Mean Absolute Percentage Error, a scale-independent forecast evaluation metric

## **Lesson Schedule & Detailed Script**

### **6:30 PM – 6:45 PM: Interactive Check-In**

**Instructor Script:** "Welcome to Week 9, Day 5! Throughout this week, we've built our machine learning foundation, learned about regression, model evaluation, and feature engineering. Today, we'll apply these concepts to time series data, specifically focusing on weather forecasting. This topic brings together everything we've learned so far and adds the dimension of time, which introduces new challenges and opportunities. By the end of today, you'll understand how to build models that can predict future weather conditions based on historical patterns."

**Admin Tasks:**

* Take attendance
* Ensure everyone has required libraries installed
* Check for any issues with previous assignments

**Prompting Questions:**

* "What patterns have you noticed in weather that might make it predictable?"
* "Why do you think forecasting the weather is challenging despite having lots of historical data?"

**Poll Questions:**

* "On a scale of 1-5, how familiar are you with time series data?"
* "What weather variable would you be most interested in predicting: temperature, precipitation, or wind speed?"

### **6:45 PM – 7:05 PM: Session 1 – Introduction to Time Series Data and Weather Forecasting**

**Objective:** Understand the unique characteristics of time series data and how they apply to weather forecasting.

**Instructor Script:** "Time series data has special properties that make it different from the data we've worked with so far. Let's explore what makes time series unique and why weather data is a perfect example of this type of data."

#### **What Makes Time Series Data Unique:**

1. **Temporal Ordering**: Observations have a specific chronological order
2. **Dependency**: Current values often depend on previous values (autocorrelation)
3. **Seasonality**: Regular patterns that repeat at fixed intervals
4. **Trends**: Long-term directional movements in the data
5. **Non-Stationarity**: Statistical properties often change over time

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from datetime import datetime, timedelta

# Create a simple synthetic weather dataset for demonstration

np.random.seed(42)

# Generate dates for one year with daily frequency

dates = pd.date\_range(start='2023-01-01', end='2023-12-31', freq='D')

# Create a temperature time series with trend, seasonality, and noise

# Base temperature (yearly average)

base\_temp = 15

# Yearly trend (warming slightly over the year)

trend = np.linspace(0, 2, len(dates))

# Seasonality (high in summer, low in winter)

seasonality = 15 \* np.sin(np.linspace(0, 2\*np.pi, len(dates)))

# Day-to-day random variations

noise = np.random.normal(0, 3, len(dates))

# Combine components

temperatures = base\_temp + trend + seasonality + noise

# Create a pandas DataFrame

weather\_df = pd.DataFrame({

'date': dates,

'temperature': temperatures

})

# Set date as index

weather\_df.set\_index('date', inplace=True)

# Visualize the time series

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

plt.plot(weather\_df.index, weather\_df['temperature'])

plt.title('Synthetic Daily Temperatures Over a Year')

plt.xlabel('Date')

plt.ylabel('Temperature (°C)')

plt.grid(True)

plt.tight\_layout()

plt.show()

# Show the first few rows

print("Sample of the weather time series data:")

print(weather\_df.head())

#### **Common Weather Data Sources and Formats:**

1. **CSV Files**: Historical records often stored in simple tabular format
2. **JSON**: Common format for API responses from weather services
3. **APIs**: Weather services like OpenWeatherMap, NOAA, Weather Underground
4. **Databases**: Structured storage for large weather datasets
5. **Sensors**: Direct readings from IoT devices and weather stations

#### **Key Weather Variables for Forecasting:**

1. **Temperature**: Daily high, low, average (°C or °F)
2. **Humidity**: Relative humidity percentage
3. **Pressure**: Atmospheric pressure (hPa, inHg)
4. **Wind**: Speed and direction
5. **Precipitation**: Rainfall, snowfall amounts
6. **Cloud Cover**: Percentage of sky covered
7. **Solar Radiation**: Direct and diffuse radiation

#### **The Importance of Timestamps and Data Frequency:**

# Example: Different sampling frequencies

# Resample to different frequencies

monthly\_avg = weather\_df.resample('M').mean()

weekly\_avg = weather\_df.resample('W').mean()

daily = weather\_df

# Visualize the different frequencies

plt.figure(figsize=(14, 8))

plt.subplot(3, 1, 1)

plt.plot(daily.index, daily['temperature'])

plt.title('Daily Temperature')

plt.ylabel('Temperature (°C)')

plt.subplot(3, 1, 2)

plt.plot(weekly\_avg.index, weekly\_avg['temperature'])

plt.title('Weekly Average Temperature')

plt.ylabel('Temperature (°C)')

plt.subplot(3, 1, 3)

plt.plot(monthly\_avg.index, monthly\_avg['temperature'])

plt.title('Monthly Average Temperature')

plt.ylabel('Temperature (°C)')

plt.tight\_layout()

plt.show()

**Key Points About Time Frequency:**

* Higher frequency (e.g., hourly) captures more detail but increases noise
* Lower frequency (e.g., monthly) smooths patterns but may miss important variations
* Choice of frequency affects model selection and performance
* Weather phenomena occur at different time scales (daily cycles, seasonal patterns, etc.)

#### **Real-World Applications of Weather Forecasting:**

1. **Agriculture**: Planning planting, harvesting, and irrigation
2. **Energy**: Predicting demand and renewable energy production
3. **Transportation**: Route planning and safety measures
4. **Emergency Management**: Preparing for extreme events
5. **Tourism and Recreation**: Planning outdoor activities
6. **Retail**: Inventory planning for seasonal goods
7. **Public Health**: Heat wave warnings, air quality forecasts

### **7:05 PM – 7:30 PM: Session 2 – Data Preparation for Weather Time Series**

**Objective:** Learn how to properly prepare weather time series data for analysis and forecasting.

**Instructor Script:** "Before we can build forecasting models, we need to properly prepare our weather data. Time series data requires specific preprocessing steps to handle timestamps, missing values, and to extract useful features that capture temporal patterns."

#### **Loading and Examining Weather Data:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

# Load the sample weather dataset (assuming CSV format with dates and temperatures)

# Note: In a real session, you would provide an actual CSV file

# For this example, we'll continue with our synthetic data

# Display basic information about the dataset

print("Dataset Information:")

print(f"Total observations: {len(weather\_df)}")

print(f"Date range: {weather\_df.index.min()} to {weather\_df.index.max()}")

print(f"Temperature range: {weather\_df['temperature'].min():.1f}°C to {weather\_df['temperature'].max():.1f}°C")

print(f"Missing values: {weather\_df['temperature'].isna().sum()}")

# Add some missing values for demonstration

weather\_df.loc[weather\_df.index[10:15], 'temperature'] = np.nan

weather\_df.loc[weather\_df.index[100:102], 'temperature'] = np.nan

# Basic statistics

print("\nBasic Statistics:")

print(weather\_df['temperature'].describe())

# Check for missing values

print("\nMissing Values:")

print(f"Number of missing temperature values: {weather\_df['temperature'].isna().sum()}")

# Visualize with missing values

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

plt.plot(weather\_df.index, weather\_df['temperature'])

plt.title('Temperature Time Series with Missing Values')

plt.xlabel('Date')

plt.ylabel('Temperature (°C)')

plt.grid(True)

plt.tight\_layout()

plt.show()

#### **Handling Missing Values in Weather Data:**

# Different methods for handling missing values

# Method 1: Forward fill (use previous day's value)

filled\_forward = weather\_df.copy()

filled\_forward['temperature'] = filled\_forward['temperature'].fillna(method='ffill')

# Method 2: Linear interpolation (connect the dots)

filled\_interpolated = weather\_df.copy()

filled\_interpolated['temperature'] = filled\_interpolated['temperature'].interpolate(method='linear')

# Method 3: Mean by time period (e.g., same day of week)

# For demonstration, we'll just use the global mean

filled\_mean = weather\_df.copy()

filled\_mean['temperature'] = filled\_mean['temperature'].fillna(filled\_mean['temperature'].mean())

# Visualize the different methods

plt.figure(figsize=(14, 8))

plt.subplot(3, 1, 1)

plt.plot(filled\_forward.index, filled\_forward['temperature'])

plt.title('Forward Fill')

plt.ylabel('Temperature (°C)')

plt.subplot(3, 1, 2)

plt.plot(filled\_interpolated.index, filled\_interpolated['temperature'])

plt.title('Linear Interpolation')

plt.ylabel('Temperature (°C)')

plt.subplot(3, 1, 3)

plt.plot(filled\_mean.index, filled\_mean['temperature'])

plt.title('Mean Fill')

plt.ylabel('Temperature (°C)')

plt.tight\_layout()

plt.show()

# We'll proceed with the interpolated data for the rest of the examples

weather\_df = filled\_interpolated

**Key Points About Handling Missing Values:**

* Forward or backward fill works well for small gaps
* Interpolation is good for short gaps with clear trends
* For longer gaps, consider using historical averages for that day/month
* Weather data often has seasonal patterns that can guide imputation
* Some advanced techniques use nearby weather stations to fill gaps

#### **Feature Engineering for Weather Forecasting:**

# Create time-based features

weather\_features = weather\_df.copy()

# Extract date components

weather\_features['day\_of\_year'] = weather\_features.index.dayofyear

weather\_features['month'] = weather\_features.index.month

weather\_features['day\_of\_week'] = weather\_features.index.dayofweek

weather\_features['is\_weekend'] = (weather\_features.index.dayofweek >= 5).astype(int)

# Create lag features (previous days)

for lag in [1, 2, 3, 7]: # Previous day, 2 days ago, 3 days ago, 1 week ago

weather\_features[f'temp\_lag\_{lag}'] = weather\_features['temperature'].shift(lag)

# Create rolling window features

weather\_features['temp\_rolling\_mean\_7d'] = weather\_features['temperature'].rolling(window=7).mean()

weather\_features['temp\_rolling\_std\_7d'] = weather\_features['temperature'].rolling(window=7).std()

# Create cyclical features for day of year (handles seasonality better)

days\_in\_year = 365.25

weather\_features['day\_of\_year\_sin'] = np.sin(2 \* np.pi \* weather\_features['day\_of\_year'] / days\_in\_year)

weather\_features['day\_of\_year\_cos'] = np.cos(2 \* np.pi \* weather\_features['day\_of\_year'] / days\_in\_year)

# Drop NaN values created by lag and rolling features

weather\_features = weather\_features.dropna()

# Show the first few rows of the engineered features

print("Weather data with engineered features:")

print(weather\_features.head())

# Visualize some of the engineered features

plt.figure(figsize=(14, 10))

plt.subplot(3, 1, 1)

plt.plot(weather\_features.index, weather\_features['temperature'], label='Temperature')

plt.plot(weather\_features.index, weather\_features['temp\_lag\_1'], label='1-Day Lag', alpha=0.7)

plt.title('Original Temperature vs 1-Day Lag')

plt.legend()

plt.subplot(3, 1, 2)

plt.plot(weather\_features.index, weather\_features['temperature'], label='Temperature')

plt.plot(weather\_features.index, weather\_features['temp\_rolling\_mean\_7d'], label='7-Day Rolling Avg', alpha=0.7)

plt.title('Original Temperature vs 7-Day Rolling Average')

plt.legend()

plt.subplot(3, 1, 3)

plt.plot(weather\_features.index, weather\_features['day\_of\_year\_sin'], label='Sine')

plt.plot(weather\_features.index, weather\_features['day\_of\_year\_cos'], label='Cosine')

plt.title('Cyclical Features for Day of Year')

plt.legend()

plt.tight\_layout()

plt.show()

**Key Feature Engineering Concepts for Weather Time Series:**

1. **Lag Features**: Previous values of target variable
   * Captures short-term dependencies and autocorrelation
   * Examples: Yesterday's temperature, last week's temperature
2. **Calendar Features**: Time-related information
   * Captures seasonal and weekly patterns
   * Examples: Month, day of week, is weekend, holiday flag
3. **Cyclical Features**: Transformed calendar features
   * Represents cyclical patterns more effectively
   * Examples: sin(day of year), cos(month)
4. **Rolling Window Features**: Statistics over time windows
   * Captures local trends and variability
   * Examples: 7-day average temperature, 30-day max precipitation
5. **Weather Interactions**: Combinations of variables
   * Captures relationships between weather components
   * Examples: Temperature × Humidity (heat index), Wind × Temperature (wind chill)

#### **Data Splitting Strategies for Time Series:**

# Traditional train-test split (not recommended for time series)

from sklearn.model\_selection import train\_test\_split

# Wrong way (random split)

X = weather\_features.drop('temperature', axis=1)

y = weather\_features['temperature']

X\_train\_wrong, X\_test\_wrong, y\_train\_wrong, y\_test\_wrong = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

print("Traditional train-test split (wrong for time series):")

print(f"Training set: {len(X\_train\_wrong)} samples")

print(f"Testing set: {len(X\_test\_wrong)} samples")

# Correct way: temporal split

split\_date = '2023-10-01' # Use the last ~3 months as test set

train = weather\_features.loc[weather\_features.index < split\_date]

test = weather\_features.loc[weather\_features.index >= split\_date]

X\_train = train.drop('temperature', axis=1)

y\_train = train['temperature']

X\_test = test.drop('temperature', axis=1)

y\_test = test['temperature']

print("\nTemporal train-test split (correct for time series):")

print(f"Training set: {len(X\_train)} samples from {X\_train.index.min()} to {X\_train.index.max()}")

print(f"Testing set: {len(X\_test)} samples from {X\_test.index.min()} to {X\_test.index.max()}")

# Visualize the train-test split

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

plt.plot(train.index, train['temperature'], label='Training Data')

plt.plot(test.index, test['temperature'], label='Testing Data')

plt.axvline(x=pd.to\_datetime(split\_date), color='r', linestyle='--', label='Split Date')

plt.title('Time Series Train-Test Split')

plt.xlabel('Date')

plt.ylabel('Temperature (°C)')

plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.show()

**Key Points About Time Series Data Splitting:**

* Always split data chronologically
* Use older data for training, newer data for testing
* Consider the forecast horizon when determining split ratio
* For time series cross-validation, use expanding window or rolling window approaches
* Ensure the test set includes all seasonal patterns if possible

### **7:30 PM – 7:50 PM: Capstone Work Session**

**Activity:** Work on captone project

### **7:50 PM – 8:00 PM: Break**

10-minute break

### **8:00 PM – 8:35 PM: Session 3 – Visualizing Weather Time Series Data**

**Objective:** Learn effective techniques for visualizing time series patterns in weather data.

**Instructor Script:** "Visualization is particularly important for time series data because it helps us identify patterns, seasonality, trends, and anomalies. Let's explore various techniques for visualizing weather time series data to gain insights that will inform our forecasting approaches."

#### **Time Series Plotting Fundamentals:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

# Continue with our weather dataset

# Add more synthetic weather variables for demonstration

np.random.seed(42)

# Add humidity (correlated with temperature but with different patterns)

humidity = 70 + 15 \* np.cos(np.linspace(0, 2\*np.pi, len(weather\_df.index))) + np.random.normal(0, 5, len(weather\_df.index))

weather\_df['humidity'] = np.clip(humidity, 0, 100) # Clip to valid humidity range (0-100%)

# Add precipitation (more sporadic)

precipitation\_prob = 0.2 + 0.3 \* np.sin(np.linspace(0, 2\*np.pi, len(weather\_df.index))) # Higher in certain seasons

precipitation = np.random.binomial(1, precipitation\_prob, len(weather\_df.index))

precipitation\_amount = precipitation \* np.random.exponential(3, len(weather\_df.index))

weather\_df['precipitation'] = precipitation\_amount

# Basic time series plot

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

plt.plot(weather\_df.index, weather\_df['temperature'])

plt.title('Temperature Over Time')

plt.xlabel('Date')

plt.ylabel('Temperature (°C)')

plt.grid(True)

plt.tight\_layout()

plt.show()

# Multiple variables on the same plot (with dual y-axis)

fig, ax1 = plt.subplots(figsize=(14, 6))

# Temperature on the primary y-axis

color = 'tab:red'

ax1.set\_xlabel('Date')

ax1.set\_ylabel('Temperature (°C)', color=color)

ax1.plot(weather\_df.index, weather\_df['temperature'], color=color)

ax1.tick\_params(axis='y', labelcolor=color)

# Humidity on the secondary y-axis

ax2 = ax1.twinx()

color = 'tab:blue'

ax2.set\_ylabel('Humidity (%)', color=color)

ax2.plot(weather\_df.index, weather\_df['humidity'], color=color)

ax2.tick\_params(axis='y', labelcolor=color)

fig.tight\_layout()

plt.title('Temperature and Humidity Over Time')

plt.grid(True)

plt.show()

#### **Visualizing Temperature Trends Over Time:**

# Resample to different frequencies for trend visualization

daily = weather\_df['temperature']

weekly = weather\_df['temperature'].resample('W').mean()

monthly = weather\_df['temperature'].resample('M').mean()

# Plot the trends

plt.figure(figsize=(14, 10))

plt.subplot(3, 1, 1)

plt.plot(daily.index, daily)

plt.title('Daily Temperature')

plt.ylabel('Temperature (°C)')

plt.grid(True)

plt.subplot(3, 1, 2)

plt.plot(weekly.index, weekly)

plt.title('Weekly Average Temperature')

plt.ylabel('Temperature (°C)')

plt.grid(True)

plt.subplot(3, 1, 3)

plt.plot(monthly.index, monthly)

plt.title('Monthly Average Temperature')

plt.ylabel('Temperature (°C)')

plt.grid(True)

plt.tight\_layout()

plt.show()

# Rolling statistics for trend visualization

rolling\_mean = weather\_df['temperature'].rolling(window=30).mean()

rolling\_std = weather\_df['temperature'].rolling(window=30).std()

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

plt.plot(weather\_df.index, weather\_df['temperature'], alpha=0.5, label='Daily Temperature')

plt.plot(rolling\_mean.index, rolling\_mean, label='30-Day Moving Average')

plt.fill\_between(

rolling\_mean.index,

rolling\_mean - rolling\_std,

rolling\_mean + rolling\_std,

alpha=0.2,

label='±1 Standard Deviation'

)

plt.title('Temperature Trend with 30-Day Moving Average')

plt.xlabel('Date')

plt.ylabel('Temperature (°C)')

plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.show()

#### **Identifying Seasonality and Patterns:**

# Decompose the time series into trend, seasonality, and residual

from statsmodels.tsa.seasonal import seasonal\_decompose

# Decompose the time series (daily data might be too noisy, so we'll use weekly)

decomposition = seasonal\_decompose(weekly, model='additive', period=52) # 52 weeks per year

# Plot the decomposition

fig = decomposition.plot()

fig.set\_size\_inches(14, 10)

plt.tight\_layout()

plt.show()

# Visualize seasonal patterns by month

# Group by month and calculate statistics

monthly\_stats = weather\_df.copy()

monthly\_stats['month'] = monthly\_stats.index.month

monthly\_stats = monthly\_stats.groupby('month')['temperature'].agg(['mean', 'std', 'min', 'max'])

monthly\_stats.index = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']

# Plot monthly patterns

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

plt.bar(monthly\_stats.index, monthly\_stats['mean'], yerr=monthly\_stats['std'], alpha=0.7)

plt.title('Average Monthly Temperature with Standard Deviation')

plt.xlabel('Month')

plt.ylabel('Temperature (°C)')

plt.grid(True, axis='y')

plt.tight\_layout()

plt.show()

# Box plot of temperatures by month

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

sns.boxplot(x=weather\_df.index.month, y=weather\_df['temperature'])

plt.xticks(range(0, 12), ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'])

plt.title('Temperature Distribution by Month')

plt.xlabel('Month')

plt.ylabel('Temperature (°C)')

plt.grid(True, axis='y')

plt.tight\_layout()

plt.show()

#### **Creating Correlation Plots Between Weather Variables:**

# Calculate correlation between variables

corr = weather\_df[['temperature', 'humidity', 'precipitation']].corr()

# Correlation heatmap

plt.figure(figsize=(10, 8))

sns.heatmap(corr, annot=True, cmap='coolwarm', vmin=-1, vmax=1)

plt.title('Correlation Between Weather Variables')

plt.tight\_layout()

plt.show()

# Scatter plot with regression line

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

sns.regplot(x='temperature', y='humidity', data=weather\_df, scatter\_kws={'alpha':0.3})

plt.title('Relationship Between Temperature and Humidity')

plt.xlabel('Temperature (°C)')

plt.ylabel('Humidity (%)')

plt.grid(True)

plt.tight\_layout()

plt.show()

# Pairplot to show all relationships

plt.figure(figsize=(14, 10))

sns.pairplot(weather\_df[['temperature', 'humidity', 'precipitation']])

plt.suptitle('Relationships Between Weather Variables', y=1.02)

plt.tight\_layout()

plt.show()

#### **Interactive Visualizations (Description Only):**

**Instructor Script:** "For interactive visualizations, libraries like Plotly and Bokeh are excellent choices. They allow users to zoom, pan, hover for details, and explore the data dynamically. While we won't implement them in detail here, these tools are valuable for exploratory data analysis and creating dashboards for weather data."

Examples of interactive visualization features:

* Zooming into specific time periods
* Hovering to see exact values
* Toggling between different weather variables
* Selecting time ranges for detailed analysis
* Animating changes over time

### **8:35 PM – 9:05 PM: Session 4 – Time Series Forecasting Models**

**Objective:** Explore different approaches to forecasting weather time series data.

**Instructor Script:** "Now that we understand our weather data and have visualized its patterns, let's build models to forecast future values. We'll explore both classical time series approaches and machine learning methods, and discuss the strengths and limitations of each."

#### **Classical Time Series Forecasting Approaches:**

1. **Moving Averages for Temperature Prediction:**

# Simple Moving Average forecast

def simple\_moving\_average(series, window):

"""Forecast using the average of the last 'window' observations."""

return series.rolling(window=window).mean()

# Create forecasts with different window sizes

ma\_7 = simple\_moving\_average(weather\_df['temperature'], 7) # 7-day moving average

ma\_30 = simple\_moving\_average(weather\_df['temperature'], 30) # 30-day moving average

# Plot the results

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

plt.plot(weather\_df.index, weather\_df['temperature'], alpha=0.5, label='Actual Temperature')

plt.plot(ma\_7.index, ma\_7, label='7-Day Moving Average')

plt.plot(ma\_30.index, ma\_30, label='30-Day Moving Average')

plt.title('Simple Moving Average Forecasts')

plt.xlabel('Date')

plt.ylabel('Temperature (°C)')

plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.show()

**Key Points About Moving Averages:**

* Simple to implement and understand
* Smooth out noise and short-term fluctuations
* Limited predictive power (just averages recent values)
* Different window sizes capture different patterns
* Lagging indicator (always behind actual changes)

1. **Exponential Smoothing Methods:**

from statsmodels.tsa.holtwinters import ExponentialSmoothing

# Create train and test sets

train\_size = int(len(weather\_df) \* 0.8)

train = weather\_df.iloc[:train\_size]['temperature']

test = weather\_df.iloc[train\_size:]['temperature']

# Simple Exponential Smoothing (SES)

ses\_model = ExponentialSmoothing(

train,

trend=None,

seasonal=None,

).fit()

# Triple Exponential Smoothing (Holt-Winters) - includes trend and seasonality

hw\_model = ExponentialSmoothing(

train,

trend='add',

seasonal='add',

seasonal\_periods=365, # Annual seasonality

).fit()

# Generate forecasts

ses\_forecast = ses\_model.forecast(len(test))

hw\_forecast = hw\_model.forecast(len(test))

# Plot the results

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

plt.plot(train.index, train, label='Training Data')

plt.plot(test.index, test, label='Actual Test Data')

plt.plot(test.index, ses\_forecast, label='Simple Exponential Smoothing')

plt.plot(test.index, hw\_forecast, label='Holt-Winters')

plt.title('Exponential Smoothing Forecasts')

plt.xlabel('Date')

plt.ylabel('Temperature (°C)')

plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.show()

**Key Points About Exponential Smoothing:**

* Assigns exponentially decreasing weights to past observations
* More recent observations have more influence
* Variations handle trend and seasonality (Holt-Winters)
* More adaptive to recent changes than simple moving averages
* Good for short to medium-term forecasts

1. **ARIMA Models for Temperature Forecasting:**

from statsmodels.tsa.arima.model import ARIMA

# Fit an ARIMA model

# Parameters: (p, d, q)

# p: order of the autoregressive model

# d: degree of differencing

# q: order of the moving average model

arima\_model = ARIMA(train, order=(2, 1, 2)).fit()

# Generate forecasts

arima\_forecast = arima\_model.forecast(len(test))

# Plot the results

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

plt.plot(train.index, train, label='Training Data')

plt.plot(test.index, test, label='Actual Test Data')

plt.plot(test.index, arima\_forecast, label='ARIMA Forecast')

plt.title('ARIMA Model Forecast')

plt.xlabel('Date')

plt.ylabel('Temperature (°C)')

plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.show()

**Key Points About ARIMA:**

* Combines autoregression (AR), differencing (I), and moving average (MA)
* Can capture complex time series patterns
* Order parameters (p, d, q) need to be selected carefully
* Good for stationary time series or after differencing
* Variants like SARIMA can handle seasonality

#### **Machine Learning Approaches:**

1. **Linear Regression for Simple Forecasting:**

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error

import numpy as np

# Use the feature-engineered dataset from earlier

# Alternatively, create basic features for this example

features\_df = weather\_df.copy()

features\_df['month'] = features\_df.index.month

features\_df['day\_of\_year'] = features\_df.index.dayofyear

features\_df['day\_of\_year\_sin'] = np.sin(2 \* np.pi \* features\_df['day\_of\_year'] / 365.25)

features\_df['day\_of\_year\_cos'] = np.cos(2 \* np.pi \* features\_df['day\_of\_year'] / 365.25)

# Add lag features

for lag in [1, 2, 3, 7]:

features\_df[f'temp\_lag\_{lag}'] = features\_df['temperature'].shift(lag)

# Drop rows with NaN values

features\_df = features\_df.dropna()

# Split data

train\_size = int(len(features\_df) \* 0.8)

train = features\_df.iloc[:train\_size]

test = features\_df.iloc[train\_size:]

# Prepare features and target for training

X\_train = train[['month', 'day\_of\_year\_sin', 'day\_of\_year\_cos',

'temp\_lag\_1', 'temp\_lag\_2', 'temp\_lag\_3', 'temp\_lag\_7']]

y\_train = train['temperature']

X\_test = test[['month', 'day\_of\_year\_sin', 'day\_of\_year\_cos',

'temp\_lag\_1', 'temp\_lag\_2', 'temp\_lag\_3', 'temp\_lag\_7']]

y\_test = test['temperature']

# Train a linear regression model

lr\_model = LinearRegression()

lr\_model.fit(X\_train, y\_train)

# Make predictions

lr\_predictions = lr\_model.predict(X\_test)

# Evaluate the model

mae = mean\_absolute\_error(y\_test, lr\_predictions)

rmse = np.sqrt(mean\_squared\_error(y\_test, lr\_predictions))

print(f"Linear Regression Model:")

print(f"Mean Absolute Error (MAE): {mae:.2f}°C")

print(f"Root Mean Squared Error (RMSE): {rmse:.2f}°C")

# Plot the results

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

plt.plot(train.index, train['temperature'], label='Training Data')

plt.plot(test.index, test['temperature'], label='Actual Test Data')

plt.plot(test.index, lr\_predictions, label='Linear Regression Predictions')

plt.title('Linear Regression Forecast')

plt.xlabel('Date')

plt.ylabel('Temperature (°C)')

plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.show()

# Feature importance

coefficients = pd.DataFrame({

'Feature': X\_train.columns,

'Coefficient': lr\_model.coef\_

})

coefficients = coefficients.sort\_values('Coefficient', key=abs, ascending=False)

plt.figure(figsize=(10, 6))

plt.barh(coefficients['Feature'], coefficients['Coefficient'])

plt.title('Linear Regression Coefficients')

plt.xlabel('Coefficient Value')

plt.ylabel('Feature')

plt.grid(True, axis='x')

plt.tight\_layout()

plt.show()

**Key Points About Linear Regression for Time Series:**

* Simple to implement and interpret
* Can incorporate multiple features, not just historical values
* Enables feature importance analysis
* May not capture complex non-linear patterns
* Good starting point for feature selection

1. **Random Forests for Temperature Prediction:**

from sklearn.ensemble import RandomForestRegressor

# Train a Random Forest model

rf\_model = RandomForestRegressor(n\_estimators=100, random\_state=42)

rf\_model.fit(X\_train, y\_train)

# Make predictions

rf\_predictions = rf\_model.predict(X\_test)

# Evaluate the model

mae = mean\_absolute\_error(y\_test, rf\_predictions)

rmse = np.sqrt(mean\_squared\_error(y\_test, rf\_predictions))

print(f"Random Forest Model:")

print(f"Mean Absolute Error (MAE): {mae:.2f}°C")

print(f"Root Mean Squared Error (RMSE): {rmse:.2f}°C")

# Plot the results

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

plt.plot(train.index, train['temperature'], label='Training Data')

plt.plot(test.index, test['temperature'], label='Actual Test Data')

plt.plot(test.index, rf\_predictions, label='Random Forest Predictions')

plt.title('Random Forest Forecast')

plt.xlabel('Date')

plt.ylabel('Temperature (°C)')

plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.show()

# Feature importance

feature\_importance = pd.DataFrame({

'Feature': X\_train.columns,

'Importance': rf\_model.feature\_importances\_

})

feature\_importance = feature\_importance.sort\_values('Importance', ascending=False)

plt.figure(figsize=(10, 6))

plt.barh(feature\_importance['Feature'], feature\_importance['Importance'])

plt.title('Random Forest Feature Importance')

plt.xlabel('Importance')

plt.ylabel('Feature')

plt.grid(True, axis='x')

plt.tight\_layout()

plt.show()

**Key Points About Random Forests for Time Series:**

* Can capture non-linear relationships
* Less prone to overfitting than some other models
* Provides feature importance metrics
* May struggle with long-term trends
* Good for complex problems with many features

1. **Introduction to Deep Learning for Time Series (Brief Overview):**

**Instructor Script:** "Deep learning approaches like Long Short-Term Memory (LSTM) networks and other recurrent neural networks are particularly well-suited for time series forecasting. They can capture complex temporal dependencies and learn from large amounts of data. While implementing these models is beyond the scope of today's lesson, it's worth noting that they represent the state-of-the-art in many time series forecasting applications, including weather prediction."

**Key Deep Learning Approaches for Time Series:**

* Recurrent Neural Networks (RNNs)
* Long Short-Term Memory (LSTM) networks
* Gated Recurrent Units (GRUs)
* Temporal Convolutional Networks (TCNs)
* Transformer-based models

#### **Comparing Statistical vs. Machine Learning Approaches:**

**Instructor Script:** "Let's compare the statistical and machine learning approaches we've explored:"

**Statistical Approaches (Moving Averages, Exponential Smoothing, ARIMA):**

* Pros:
  + Often work well with limited data
  + Explicitly model time series components (trend, seasonality)
  + Typically require less preprocessing
  + Easier to interpret and understand
* Cons:
  + May not capture complex non-linear patterns
  + Limited ability to incorporate external variables
  + Often require stationarity assumptions
  + May struggle with very long-term forecasts

**Machine Learning Approaches (Linear Regression, Random Forests, Deep Learning):**

* Pros:
  + Can capture complex, non-linear relationships
  + Easily incorporate multiple input variables
  + Often more flexible and adaptable
  + Can outperform statistical methods with sufficient data
* Cons:
  + May require more data to perform well
  + Risk of overfitting with complex models
  + Some models are "black boxes" (less interpretable)
  + Often require more feature engineering

"The best approach often depends on your specific forecasting problem, data availability, and requirements for interpretability versus accuracy."

### **9:05 PM – 9:25 PM: Breakout #2: Capstone Working Session**

**Activity:** Work on capstone project.

### **9:25 PM – 9:30 PM: Wrap-Up & Final Questions**

**Instructor Script:** "Today we've explored time series data in the context of weather forecasting. We've learned how to prepare and visualize weather time series data, and we've implemented both classical and machine learning approaches to forecasting. These concepts are fundamental not just for weather prediction, but for any time-dependent data analysis, from finance to energy consumption to web traffic forecasting."

**Review Key Points:**

* Time series data has unique properties that require special handling
* Weather data typically exhibits strong seasonality and trends
* Feature engineering is crucial for effective time series forecasting
* Both statistical and machine learning approaches have their strengths
* Proper evaluation requires chronological train-test splits and appropriate metrics

**Prompting Question:** "What's one key insight about time series forecasting that you'll apply in your future projects?"

## **After-Class Quiz (5 questions)**

1. When creating lag features for time series forecasting, what are you doing?
   * A) Removing delays in the data collection
   * \*B) Using past values of the target variable as input features
   * C) Fixing missing timestamps in the data
   * D) Combining multiple time series into one
2. Why is it important to split time series data chronologically rather than randomly?
   * A) It's more computationally efficient
   * B) Random splits are too unpredictable
   * \*C) Using future data to predict the past would create data leakage
   * D) Time series data can only be processed sequentially
3. What is the advantage of using cyclical features (sin/cos transformations) for day of year or month?
   * A) They are easier to calculate than calendar features
   * \*B) They represent the circular nature of time, connecting December to January
   * C) They make the model run faster
   * D) They eliminate the need for lag features
4. Which of these is a key limitation of using moving averages for forecasting?
   * A) They require large amounts of data
   * B) They can only be used for temperature forecasting
   * \*C) They always lag behind actual changes in the data
   * D) They can't handle seasonal data
5. When evaluating a weather forecasting model, which metric would be most appropriate?
   * A) Accuracy
   * B) F1-score
   * \*C) Mean Absolute Error (MAE)
   * D) Log Loss