Week 3 Homework Parts 1 and 2

April 10, 2024

0.1 Time Series Exploring and Understanding

0.2 Origin of the Data and Ethical Assessment

The "Indian Summers" dataset likely originates from meteorological observations collected and compiled by a national weather service or a recognized meteorological organization. These data are typically sourced from a network of weather stations across various locations, focusing specifically on temperature records during the summer months in India.

0.3 Ethical Considerations:

Data Collection: The data collection process for weather data is generally considered ethical, as it involves systematic observations of natural phenomena without direct human intervention or privacy concerns. Data Use: The use of weather data for analysis and forecasting is inherently ethical, aimed at better understanding and preparing for climatic conditions. It supports a wide range of beneficial applications from public safety to agricultural planning. Accessibility: Data was sourced via Kaggle which is generally reliable as public domain.

0.4 Questions for Time-Series Analysis

0.5 Purpose of Generating Forecasts and the Difference Between Forecasting and Predictions

Purpose: The primary purpose of generating forecasts from this dataset is to anticipate future temperature trends during the summer months in India, which can aid in heatwave preparedness, public health planning, and resource management. ## Forecasting vs. Prediction: Forecasting involves the use of historical data to estimate future events in a continuous sequence. It is typically time-specific and uses patterns from past data to infer future conditions. Prediction often refers to estimating outcomes based on various inputs and can be used to determine specific events that may not necessarily follow a temporal sequence. Predictions can incorporate a range of data inputs and machine learning models to assess likely future events. ## Types of Forecasts Needed Descriptive Forecasts: These would analyze historical temperature trends, identifying patterns and anomalies during past summers. Predictive Forecasts: These are crucial for anticipating future temperature extremes, which could inform planning processes in multiple sectors. ## Usage of Forecasts by the Company Forecasts derived from this dataset could be used by utility companies to manage energy demand, by agricultural enterprises to plan irrigation and harvesting, and by health agencies to prepare for heat-related health issues. ## Costs Associated with Forecast Errors Economic Costs: Inaccurate temperature forecasts can lead to insufficient energy supply, resulting in power shortages or wasteful excesses. Health Costs: Underestimating temperatures may result in inadequate heatwave alerts, potentially leading to increased heatstroke incidents. Agricultural Costs: Farmers

relying on accurate weather forecasts might face crop damage if unexpected high temperatures are not accurately forecasted. ## Forecasting Horizon and Its Impact Short-Term Horizon: Typically a few days to weeks; more accurate but less useful for long-term planning. Long-Term Horizon: Extends over months to a season; crucial for strategic planning but generally less precise due to the increasing unpredictability of weather patterns over longer periods. ## Application of Time Series Forecasting Across Industries Retail: Used to plan and stock seasonal products effectively, anticipating consumer demand changes due to weather conditions. Energy: Forecasts are critical for anticipating energy demands, particularly for cooling systems during hot periods, ensuring adequate power supply. Government: Helps in urban planning, emergency preparedness, and resource allocation, especially in response to anticipated climatic conditions. Financial: Influences commodities trading, insurance, and investment decisions related to agricultural outputs and energy stocks. Agriculture: Used to schedule planting and harvesting, manage water resources, and plan crop rotations based on expected weather conditions. Education: Supports academic research and educational programs focused on climate science and environmental management

[41]:
```
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm

# Load the dataset
df = pd.read_csv('Indian Summers - Over the years.csv', parse_dates=['Date'],␣
 ↪index_col='Date')

# Descriptive Statistics
print(df.describe())

# Check the data type of the date column
print(df.index.dtype)

# Print the first five rows
print(df.head())
```

```
            tempmax       tempmin          temp  feelslikemax  feelslikemin  \
count  13615.000000  13615.000000  13605.000000  13614.000000  13614.000000
mean      36.728248     25.821160     31.151510     40.212605     27.221324
std        4.115452      3.212167      3.074874      5.389016      4.907125
min        0.000000      0.000000     19.900000      0.000000      0.000000
25%       34.000000     23.700000     29.200000     36.500000     23.700000
50%       37.000000     26.000000     31.100000     40.000000     26.000000
75%       39.800000     28.100000     33.200000     43.700000     31.100000
max       50.000000     37.000000     40.500000     79.200000     43.300000

          feelslike           dew      humidity     windspeed       winddir  \
count  13604.000000  13605.000000  13605.000000  13605.000000  13600.000000
mean      33.704535     19.049607     54.638537     20.078552    205.236559
std        4.666616      5.966341     19.521510      9.886468     64.181345
min       19.900000    -10.300000      7.410000      0.000000      0.000000
25%       30.200000     15.000000     38.190000     14.800000    159.800000
```

```
50%       33.500000     20.300000     56.120000     19.500000    218.300000
75%       37.200000     24.000000     71.410000     24.100000    258.625000
max       48.500000     29.100000     99.040000    263.200000    360.000000

       sealevelpressure    cloudcover    visibility     moonphase
count      10631.000000  13605.000000  13605.000000  13650.000000
mean        1004.302446     37.120235      4.666645      0.500692
std            4.183785     24.684504      1.382413      0.308204
min          908.500000      0.000000      1.300000      0.000000
25%         1001.600000     16.700000      3.700000      0.250000
50%         1004.700000     36.700000      4.300000      0.500000
75%         1007.300000     54.000000      5.600000      0.760000
max         1026.200000    100.000000     12.300000      1.000000
datetime64[ns]
                City  tempmax  tempmin  temp  feelslikemax  feelslikemin  \
Date
2021-04-01  New Delhi     34.0     19.0  27.1          31.6          19.0
2021-04-02  New Delhi     33.9     16.0  25.8          31.8          16.0
2021-04-03  New Delhi     34.8     14.6  26.0          32.2          14.6
2021-04-04  New Delhi     36.8     16.9  27.1          34.2          16.9
2021-04-05  New Delhi     38.8     21.0  29.9          37.1          21.0

            feelslike  dew  humidity  windspeed  winddir  sealevelpressure  \
Date
2021-04-01       26.1  3.1     22.60       22.8    272.9            1002.8
2021-04-02       24.9  4.5     27.62       12.4    275.0            1006.2
2021-04-03       25.1  1.3     23.18       16.5    127.5            1008.8
2021-04-04       26.0  4.8     28.00       18.3    157.6            1009.5
2021-04-05       28.9  8.1     28.85       13.5    100.4            1007.8

            cloudcover  visibility              sunrise               sunset  \
Date
2021-04-01         0.0         3.1  2021-04-01 06:11:12  2021-04-01 18:39:13
2021-04-02         0.0         3.5  2021-04-02 06:10:04  2021-04-02 18:39:46
2021-04-03         1.4         3.5  2021-04-03 06:08:55  2021-04-03 18:40:19
2021-04-04         2.6         3.2  2021-04-04 06:07:47  2021-04-04 18:40:53
2021-04-05        38.4         3.1  2021-04-05 06:06:39  2021-04-05 18:41:26

            moonphase        conditions                            description
Date
2021-04-01       0.60             Clear  Clear conditions throughout the day.
2021-04-02       0.65             Clear  Clear conditions throughout the day.
2021-04-03       0.70             Clear  Clear conditions throughout the day.
2021-04-04       0.76             Clear  Clear conditions throughout the day.
2021-04-05       0.81  Partially cloudy     Partly cloudy throughout the day.
```

0.6 Data Interpretation

The dataset titled "Indian Summers - Over the years" comprises meteorological data from New Delhi, detailing daily weather metrics over several years. This comprehensive dataset captures a variety of atmospheric conditions, including temperature extremes, humidity levels, wind speeds, and atmospheric pressure, making it an invaluable resource for analyzing climate patterns, assessing environmental impacts, and aiding in urban planning and public health strategies.

0.6.1 Dataset Summary and Descriptive Statistics

The dataset consists of 13,650 entries, with measurements spanning several years. Each record includes both maximum and minimum values for temperature and 'feels like' temperature, dew point, humidity, wind speed and direction, sea-level pressure, cloud cover, visibility, and the lunar phase. These data points are supplemented with qualitative descriptions of the daily weather conditions.

- **Temperature Variations**: The average maximum temperature (`tempmax`) observed is approximately 36.7°C, with a recorded maximum of 50.0°C. The average minimum temperature (`tempmin`) is about 25.8°C, highlighting significant daily thermal variation which can impact energy consumption patterns and health.
- **Humidity and Dew Point**: The average humidity is 54.64%, with extremes being as low as 7.41% and as high as 99.04%, indicating days of extremely dry and very humid conditions, respectively. Dew point, a better measure of atmospheric moisture, averages at 19.05°C.
- **Wind and Atmospheric Pressure**: Average wind speeds are noted at 20.08 km/h, which suggests moderate breezy conditions typical for New Delhi. The sea-level pressure averages around 1004.3 hPa, within normal ranges but showing variability that could influence weather patterns.
- **Visibility and Cloud Cover**: Average visibility is around 4.7 km, potentially reduced during certain conditions like fog, dust storms, or pollution, which are common in urban settings like New Delhi. Cloud cover averages 37.12%, with days ranging from clear skies to completely overcast.

0.6.2 Time-Series Analysis Objectives

Given the dataset, several pertinent questions for time-series analysis arise:

1. **Trend Analysis**: How have temperature and humidity trends changed over the years? This could indicate broader climatic shifts or urban heat island effects.
2. **Seasonality**: What are the seasonal patterns in weather conditions? Understanding this helps predict energy demands, agricultural impacts, and health advisories.
3. **Forecasting**: Can we develop predictive models to forecast extreme weather conditions, such as heatwaves or high pollution days? These forecasts could be critical for preparing healthcare services and public advisories.

0.6.3 Usage of Forecasts

The forecasts derived from this dataset could serve multiple stakeholders: - **Government and Urban Planners**: To enhance infrastructure resilience against heatwaves and plan cooling centers. - **Health Departments**: To prepare for and mitigate the impacts of heat-related illnesses. - **Agricultural Sector**: To better schedule planting and harvesting by predicting optimal weather

conditions. - **Energy Sector**: To manage and adjust energy supply in response to anticipated demand due to temperature fluctuations.

0.6.4 Forecast Accuracy and Horizon

The accuracy of forecasts generally decreases as the horizon extends. Short-term forecasts are typically more accurate and useful for immediate planning, such as disaster management during expected heatwaves. Long-term forecasts, while less precise, are vital for strategic planning and policy formulation to adapt to changing climate conditions.

0.6.5 Costs Associated with Forecast Errors

Inaccurate forecasts can lead to significant costs: - **Health Risks**: Under-preparation for heatwaves can result in increased mortality and morbidity. - **Economic Costs**: Incorrect energy supply management can lead to inefficiencies and increased operational costs. - **Agricultural Losses**: Poor timing in agricultural practices due to inaccurate weather predictions can lead to crop failures.

0.6.6 Conclusion

The "Indian Summers" dataset provides a rich source of information for detailed climatic analysis. Through careful examination and modeling, it can yield forecasts that help mitigate the adverse effects of extreme weather, guide urban and agricultural planning, and enhance public health strategies. Such analyses underscore the importance of robust time-series studies in understanding and adapting to our changing environment.

[111]:
```
# Check the data type of the index to confirm it's datetime
print(df.index.dtype)
```

```
datetime64[ns]
```

[113]:
```
# Print the first five rows of the DataFrame
print(df.head())
```

```
                City  tempmax  tempmin  temp  feelslikemax  feelslikemin  \
Date
2021-04-01  New Delhi     34.0     19.0  27.1          31.6          19.0
2021-04-02  New Delhi     33.9     16.0  25.8          31.8          16.0
2021-04-03  New Delhi     34.8     14.6  26.0          32.2          14.6
2021-04-04  New Delhi     36.8     16.9  27.1          34.2          16.9
2021-04-05  New Delhi     38.8     21.0  29.9          37.1          21.0

            feelslike  dew  humidity  windspeed  winddir  sealevelpressure  \
Date
2021-04-01       26.1  3.1     22.60       22.8    272.9            1002.8
2021-04-02       24.9  4.5     27.62       12.4    275.0            1006.2
2021-04-03       25.1  1.3     23.18       16.5    127.5            1008.8
2021-04-04       26.0  4.8     28.00       18.3    157.6            1009.5
2021-04-05       28.9  8.1     28.85       13.5    100.4            1007.8
```

```
            cloudcover  visibility             sunrise              sunset  \
Date
2021-04-01         0.0         3.1  2021-04-01 06:11:12  2021-04-01 18:39:13
2021-04-02         0.0         3.5  2021-04-02 06:10:04  2021-04-02 18:39:46
2021-04-03         1.4         3.5  2021-04-03 06:08:55  2021-04-03 18:40:19
2021-04-04         2.6         3.2  2021-04-04 06:07:47  2021-04-04 18:40:53
2021-04-05        38.4         3.1  2021-04-05 06:06:39  2021-04-05 18:41:26

            moonphase        conditions                          description
Date
2021-04-01       0.60             Clear  Clear conditions throughout the day.
2021-04-02       0.65             Clear  Clear conditions throughout the day.
2021-04-03       0.70             Clear  Clear conditions throughout the day.
2021-04-04       0.76             Clear  Clear conditions throughout the day.
2021-04-05       0.81  Partially cloudy     Partly cloudy throughout the day.
```

[117]:
```
# Remove NaN values from temp
df.dropna(subset=['temp'], inplace=True)
```

[121]:
```
# Confirm missing values in the 'temp' column or the correct column for␣
↪temperature
print(df['temp'].isnull().sum())
```

```
0
```

[127]:
```
# Confirm the type of the index
print(df.index)

# If not set as a DatetimeIndex, set it
if not isinstance(df.index, pd.DatetimeIndex):
    df.index = pd.to_datetime(df.index)
```

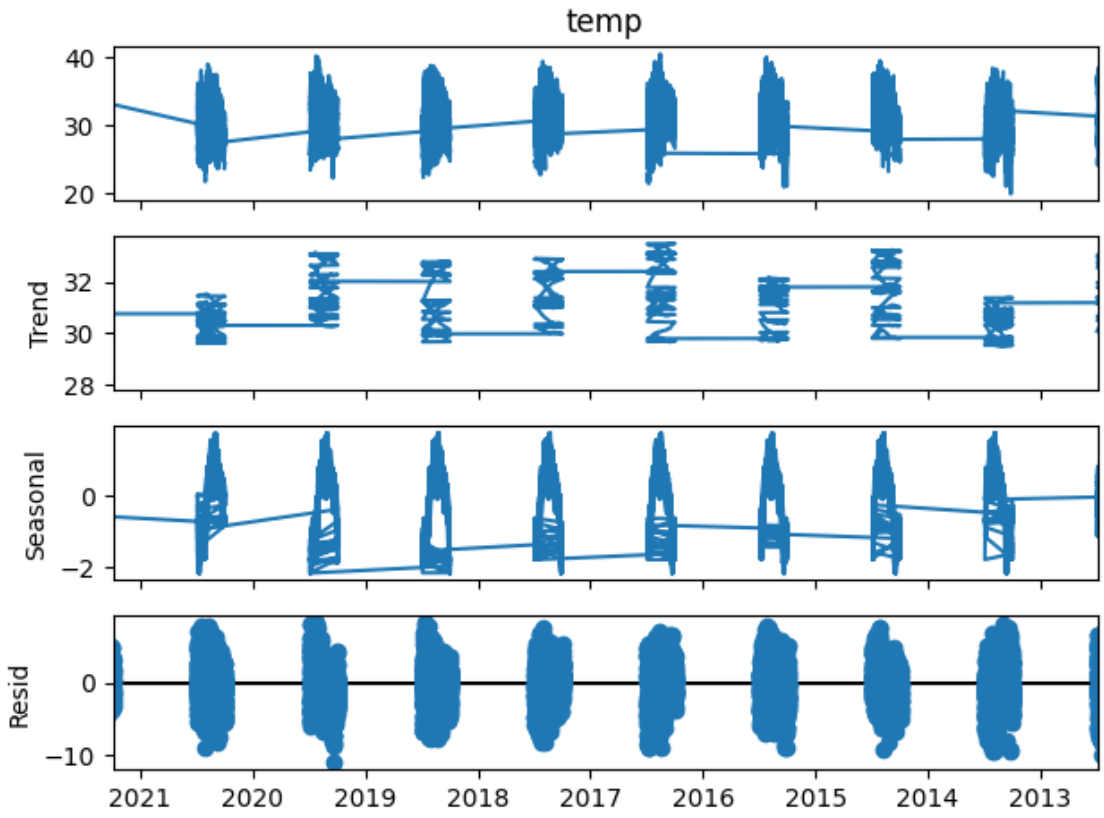
```
DatetimeIndex(['2021-04-01', '2021-04-02', '2021-04-03', '2021-04-04',
               '2021-04-05', '2021-04-06', '2021-04-07', '2021-04-08',
               '2021-04-09', '2021-04-10',
               …
               '2012-06-21', '2012-06-22', '2012-06-23', '2012-06-24',
               '2012-06-25', '2012-06-26', '2012-06-27', '2012-06-28',
               '2012-06-29', '2012-06-30'],
              dtype='datetime64[ns]', name='Date', length=13650, freq=None)
```

[131]:
```
from statsmodels.tsa.seasonal import seasonal_decompose

# Perform seasonal decomposition
decomposition = seasonal_decompose(df['temp'], model='additive', period=365)

# Plot the decomposed components
fig = decomposition.plot()
```

```
plt.show()
```



Based on the decomposition plot provided, we can observe and interpret the following components of the "temp" time series data:

1. **Observed**: The top panel represents the actual observed temperatures over time. The fluctuations indicate seasonal changes, with peaks typically representing warmer summer months and troughs indicating cooler seasons.

2. **Trend**: The second panel shows the trend component extracted from the data. There is a relatively stable trend across the years without a clear long-term increase or decrease, which suggests that over the period captured by the dataset, the average temperature hasn't significantly changed. However, there are intervals of increases and decreases that may correspond to shorter-term climate variations or cycles.

3. **Seasonal**: The third panel illustrates the seasonal component. There's a clear and consistent seasonal pattern repeated annually. This pattern likely represents the temperature increase during the summer months and decrease in the winter months, typical for New Delhi's climate with marked hot summers and cooler winters.

4. **Residual**: The bottom panel shows the residual component, which includes the noise and anomalies that are not explained by the trend or seasonality. The residuals appear to have some structure, with certain years showing more variability than others. These could be

random variations or they could reflect other variables not included in the model, such as unusual weather events, environmental changes, or even urban development that can affect local temperatures.

Patterns and Anomalies: The observed temperature data shows annual cyclic patterns that align with expected seasonal temperature changes. The trend component does not show a strong long-term upward or downward trend, indicating relative stability in average temperatures over this period. However, further analysis might be needed to determine if there are any subtle changes when viewing the data over a longer period or with a more granular breakdown, such as monthly averages. The seasonal pattern is consistent with known climate patterns for the region, confirming that temperatures peak during certain months of the year and dip in others. The residuals should ideally be random if the model has captured all the systematic information in the data. The presence of structure in the residuals suggests there may be additional factors or patterns affecting the temperature that the model hasn't captured. This could warrant a more complex model or the inclusion of additional variables.

In conclusion, the decomposition analysis of the "Indian Summers" dataset reveals expected seasonal variations in temperature and suggests that there may be additional factors influencing temperature variability that are worth investigating further. Understanding these patterns can be crucial for planning purposes in various sectors, including energy, agriculture, and public health.

0.7 The Importance of Standardization and Normalization Based on Time-Series Decomposition:

The seasonal decomposition of the dataset provided clear patterns of seasonality and trend, which are crucial for forecasting and predictive analytics. However, the decomposition output also suggested the presence of heteroscedasticity - a variance that changes over time. To address this:

Standardization could be applied to temperature measurements to mitigate the impact of heteroscedastic residuals, ensuring that the variance is consistent over time, which is particularly beneficial for regression-based models.

Normalization might be required if the objective includes comparing the temperature data with other scaled variables or incorporating this data into machine learning models, such as neural networks, that are sensitive to the scale of input data.

In the case of multivariate time-series forecasting, where temperature data might be used alongside other variables (like humidity or wind speed), normalizing or standardizing the dataset ensures that each variable contributes proportionately to the predictive model, preventing any single variable from dominating due to its scale.

If subsequent steps involve the comparison of temperature trends across different geographical locations or the integration of this dataset with other climatic data, then standardizing each variable to have a mean of zero and a standard deviation of one would allow for fair comparison and combination of data from different scales or units.

0.8 Conclusion:

In conclusion, while the date column serves as the structural backbone for the time-series analysis, the actual transformational needs, such as standardization or normalization, are directed at the quantitative attributes of the data. These transformations are imperative for preparing the dataset

for advanced analytical methods and predictive modeling, ensuring the accuracy of conclusions drawn and the effectiveness of models built. As demonstrated in the seasonal decomposition output, the approach to preprocessing should be informed by the underlying statistical characteristics of the dataset, the analytical goals, and the requirements of specific analytical methods to be employed in future steps.

0.9 Part 2: Time Series Feature Engineering

Feature engineering is a crucial step in the model-building process, especially in time-series data where the relationship between time and the other variables can reveal important patterns. Here are the main goals and benefits of performing feature engineering on time-series data:

0.9.1 1. Enhancing Model Performance

Improved Accuracy: By creating features that capture underlying patterns such as trends, seasonality, and cycles, models can learn from these structures, leading to more accurate predictions.

Reduced Overfitting: Effective features can help a model focus on the signal rather than the noise, which is particularly important in time-series data that often contains many irregularities.

0.9.2 2. Capturing Temporal Dynamics

Trend: Time-series data often have a trend component that reflects the increasing or decreasing pattern over time. Feature engineering can extract this trend to help models account for long-term directional movements in the data.

Seasonality: Many time-series datasets exhibit seasonal patterns. Features like month of the year, day of the week, or special events (holidays, sales seasons) can capture this periodicity and improve the model's ability to predict seasonal fluctuations.

Cyclic Changes: Apart from seasonality, time-series data might exhibit cycles that are not of fixed frequency. Engineering features that can capture such cycles (business cycles, macroeconomic cycles) can be beneficial for models to understand longer-term fluctuations.

0.9.3 3. Incorporating Domain Knowledge

Custom Features: Domain expertise can be used to create features that are not immediately apparent in the raw data. This might include, for instance, the impact of certain weather conditions on energy demand or the effect of specific holidays on sales data.

External Data: Time-series analysis can be enriched by bringing in additional data sources. Features crafted from economic indicators, weather data, or other relevant external variables can provide additional context that can significantly improve model predictions.

0.9.4 4. Improving Model Interpretability

Intuitive Understanding: Engineered features can make model outputs more interpretable. For instance, instead of letting the model implicitly handle seasonality, creating explicit seasonal features allows us to see how different times of the year impact the target variable.

Diagnosis and Explanation: When a model prediction is incorrect, having well-defined features allows for easier diagnosis of what might have gone wrong and better explanations to stakeholders.

0.9.5 5. Simplifying Models

Dimensionality Reduction: Feature engineering can reduce the dimensionality of the data. Instead of using raw data with many lagged observations, we can use a smaller number of engineered features that summarize the important information.

Efficiency: Simplified models with fewer but more informative features can be more computationally efficient, reducing both training times and inference times.

0.9.6 6. Handling Missing Values and Anomalies

Robustness: Well-crafted features can make models more robust to gaps or anomalies in the data, by capturing the broader patterns that persist even when some data points are missing or are outliers.

In conclusion, feature engineering is vital for unlocking the full potential of time-series datasets. It enables the creation of more sophisticated, accurate, and interpretable models that can capture complex temporal dynamics and interactions within the data, ultimately leading to better decision-making. Whether for forecasting, anomaly detection, or causal inference, feature engineering is an indispensable tool in the time-series analyst's toolkit.

[138]:
```
# create new columns
df['Year'] = df.index.year
df['Month'] = df.index.month
df['Day'] = df.index.day

# Same day last week
df['temp_same_day_last_week'] = df['temp'].shift(7)
# Roughly the same day last month
df['temp_same_day_last_month'] = df['temp'].shift(30)
# Same day last year
df['temp_same_day_last_year'] = df['temp'].shift(365)

#Window Feature
df['temp_2_month_rolling_avg'] = df['temp'].rolling(window=60).mean()

# Expanding Feature
df['temp_max_up_to_date'] = df['temp'].expanding().max()
```

[161]:
```
# Check to ensure data converted
df.head()
```

[161]:

Date	City	tempmax	tempmin	temp	feelslikemax	feelslikemin	\
2021-04-01	New Delhi	34.0	19.0	27.1	31.6	19.0	
2021-04-02	New Delhi	33.9	16.0	25.8	31.8	16.0	
2021-04-03	New Delhi	34.8	14.6	26.0	32.2	14.6	
2021-04-04	New Delhi	36.8	16.9	27.1	34.2	16.9	
2021-04-05	New Delhi	38.8	21.0	29.9	37.1	21.0	

```
            feelslike  dew  humidity  windspeed  …        conditions  \
Date                                             …
2021-04-01       26.1  3.1     22.60       22.8  …             Clear
2021-04-02       24.9  4.5     27.62       12.4  …             Clear
2021-04-03       25.1  1.3     23.18       16.5  …             Clear
2021-04-04       26.0  4.8     28.00       18.3  …             Clear
2021-04-05       28.9  8.1     28.85       13.5  …  Partially cloudy

                                  description  Year  Month Day  \
Date
2021-04-01  Clear conditions throughout the day.  2021      4   1
2021-04-02  Clear conditions throughout the day.  2021      4   2
2021-04-03  Clear conditions throughout the day.  2021      4   3
2021-04-04  Clear conditions throughout the day.  2021      4   4
2021-04-05     Partly cloudy throughout the day.  2021      4   5

           temp_same_day_last_week  temp_same_day_last_month  \
Date
2021-04-01                     NaN                       NaN
2021-04-02                     NaN                       NaN
2021-04-03                     NaN                       NaN
2021-04-04                     NaN                       NaN
2021-04-05                     NaN                       NaN

           temp_same_day_last_year temp_2_month_rolling_avg  \
Date
2021-04-01                     NaN                      NaN
2021-04-02                     NaN                      NaN
2021-04-03                     NaN                      NaN
2021-04-04                     NaN                      NaN
2021-04-05                     NaN                      NaN

            temp_max_up_to_date
Date
2021-04-01                 27.1
2021-04-02                 27.1
2021-04-03                 27.1
2021-04-04                 27.1
2021-04-05                 29.9

[5 rows x 27 columns]
```

[187]:
```
# Columns with any NaN values
print(df.isnull().sum())
```

```
City                         0
```

```
tempmax                      0
tempmin                      0
temp                         0
feelslikemax                 0
feelslikemin                 0
feelslike                    0
dew                          0
humidity                     0
windspeed                    0
winddir                      0
sealevelpressure             0
cloudcover                   0
visibility                   0
sunrise                      0
sunset                       0
moonphase                    0
conditions                   0
description                  0
Year                         0
Month                        0
Day                          0
temp_same_day_last_week      7
temp_same_day_last_month    30
temp_same_day_last_year    365
temp_2_month_rolling_avg    59
temp_max_up_to_date          0
dtype: int64
```

0.9.7 Insights from Date Time Features

- **Temporal Trends**: The Year, Month, and Day features can help uncover any long-term trends (like global warming), seasonal effects (such as monsoon patterns), and even daily fluctuations due to diurnal cycles. Understanding these can allow a model to anticipate future values more accurately.
- **Calendar Effects**: Month and Day features can capture calendar effects, like increased temperatures in specific months due to climatic conditions or particular days that consistently show temperature anomalies.

0.9.8 Insights from Lag Features

- **Autocorrelation**: Lag features capture the autocorrelation in data. For instance, knowing yesterday's temperature can be a good predictor for today's temperature. Understanding the strength and duration of autocorrelation can help set the right lags in models like ARIMA (AutoRegressive Integrated Moving Average).
- **Historical Comparisons**: By comparing the same day across weeks, months, and years, we can measure how significant recent changes are compared to historical patterns. This can help in detecting and adjusting for outliers or breaks in the series.

0.9.9 Insights from Window Features

- **Moving Averages**: A 2-month rolling average smooths out short-term fluctuations and reveals the underlying trend. This can be particularly useful for dampening the noise and capturing the trajectory of the time series, which is beneficial for identifying sustained shifts in the mean level of the series.
- **Seasonality Adjustment**: Rolling averages can also help in adjusting for seasonality, especially when the window spans a full seasonal cycle, enabling the identification of anomalies that deviate from typical seasonal behavior.

0.9.10 Insights from Expanding Features

- **Cumulative Knowledge**: Expanding maximum values capture the most extreme events up to that point in time. Understanding how extremes are changing can be critical for risk management and preparedness, especially in scenarios where extremes have more impact than averages.
- **Benchmarking and Thresholds**: The expanding maximum can act as a benchmark for identifying extreme but rare events. It also helps in setting adaptive thresholds in anomaly detection systems.

0.9.11 Building a Better Forecasting Solution

With these insights, as a data scientist, you can enhance time series forecasting models in several ways:

1. **Feature Selection**: You can select features that have the most predictive power and exclude redundant or irrelevant features, thereby simplifying the model and improving performance.
2. **Model Specification**: You can better specify the structure of time series models. For instance, if you detect strong seasonal patterns, you may opt for a SARIMA model rather than a non-seasonal one.
3. **Anomaly Detection**: With a better understanding of historical patterns and extremes, you can design systems that more accurately flag anomalies, leading to timely interventions.
4. **Risk Management**: Knowledge of historical extremes can inform risk management strategies, allowing for more robust planning against potential adverse conditions predicted by the model.
5. **Data Imputation**: Lag features can be used to impute missing values based on historical patterns, which is particularly useful in time series where continuity is important.
6. **Hyperparameter Tuning**: Features like rolling averages or lags can inform the tuning of model hyperparameters, such as the length of the moving average window in ARIMA models.

In summary, the added knowledge from feature engineering enables the crafting of models that are more aligned with the actual dynamics of the system being modeled. This leads to more accurate and reliable forecasts, which are crucial for decision-making processes in various application domains.

[193]:
```
# 'Q' signifies the quarter frequency, and 'mean()' calculates the average.
df['Q'] = df['temp'].resample('Q').mean()
```

```
# Show the results
df[['temp', 'Q']]
```

```
/tmp/ipykernel_421/1633005760.py:2: FutureWarning: 'Q' is deprecated and will be
removed in a future version, please use 'QE' instead.
  df['Q'] = df['temp'].resample('Q').mean()
```

[193]:

Date	temp	Q
2021-04-01	27.1	NaN
2021-04-02	25.8	NaN
2021-04-03	26.0	NaN
2021-04-04	27.1	NaN
2021-04-05	29.9	NaN
…	…	…
2012-06-26	25.8	NaN
2012-06-27	25.5	NaN
2012-06-28	26.8	NaN
2012-06-29	26.7	NaN
2012-06-30	27.7	31.449817

[13650 rows x 2 columns]

[199]:
```
df.head()
```

[199]:

Date	City	tempmax	tempmin	temp	feelslikemax	feelslikemin	\
2021-04-01	New Delhi	34.0	19.0	27.1	31.6	19.0	
2021-04-02	New Delhi	33.9	16.0	25.8	31.8	16.0	
2021-04-03	New Delhi	34.8	14.6	26.0	32.2	14.6	
2021-04-04	New Delhi	36.8	16.9	27.1	34.2	16.9	
2021-04-05	New Delhi	38.8	21.0	29.9	37.1	21.0	

Date	feelslike	dew	humidity	windspeed	…	\
2021-04-01	26.1	3.1	22.60	22.8	…	
2021-04-02	24.9	4.5	27.62	12.4	…	
2021-04-03	25.1	1.3	23.18	16.5	…	
2021-04-04	26.0	4.8	28.00	18.3	…	
2021-04-05	28.9	8.1	28.85	13.5	…	

Date	description	Year	Month	Day	\
2021-04-01	Clear conditions throughout the day.	2021	4	1	
2021-04-02	Clear conditions throughout the day.	2021	4	2	
2021-04-03	Clear conditions throughout the day.	2021	4	3	

```
2021-04-04  Clear conditions throughout the day.  2021      4    4
2021-04-05     Partly cloudy throughout the day.  2021      4    5

            temp_same_day_last_week temp_same_day_last_month  \
Date
2021-04-01                      NaN                       NaN
2021-04-02                      NaN                       NaN
2021-04-03                      NaN                       NaN
2021-04-04                      NaN                       NaN
2021-04-05                      NaN                       NaN

             temp_same_day_last_year temp_2_month_rolling_avg  \
Date
2021-04-01                       NaN                       NaN
2021-04-02                       NaN                       NaN
2021-04-03                       NaN                       NaN
2021-04-04                       NaN                       NaN
2021-04-05                       NaN                       NaN

            temp_max_up_to_date   Q
Date
2021-04-01                 27.1 NaN
2021-04-02                 27.1 NaN
2021-04-03                 27.1 NaN
2021-04-04                 27.1 NaN
2021-04-05                 29.9 NaN

[5 rows x 28 columns]
```

[201]:
```python
# Resample by year ('A' signifies annual frequency) and calculate the mean.
df['Yearly'] = df['temp'].resample('A').mean()

# Display the results, showing the original temperature and the new yearly mean
df[['temp', 'Yearly']]
```

```
/tmp/ipykernel_421/4056923344.py:2: FutureWarning: 'A' is deprecated and will be
removed in a future version, please use 'YE' instead.
  df['Yearly'] = df['temp'].resample('A').mean()
```

[201]:
```
             temp  Yearly
Date
2021-04-01  27.1     NaN
2021-04-02  25.8     NaN
2021-04-03  26.0     NaN
2021-04-04  27.1     NaN
2021-04-05  29.9     NaN
…           …     …
```

```
2012-06-26  25.8     NaN
2012-06-27  25.5     NaN
2012-06-28  26.8     NaN
2012-06-29  26.7     NaN
2012-06-30  27.7     NaN

[13650 rows x 2 columns]
```

[203]:
```
df.head()
```

[203]:
```
                City  tempmax  tempmin  temp  feelslikemax  feelslikemin  \
Date
2021-04-01  New Delhi     34.0     19.0  27.1          31.6          19.0
2021-04-02  New Delhi     33.9     16.0  25.8          31.8          16.0
2021-04-03  New Delhi     34.8     14.6  26.0          32.2          14.6
2021-04-04  New Delhi     36.8     16.9  27.1          34.2          16.9
2021-04-05  New Delhi     38.8     21.0  29.9          37.1          21.0

            feelslike  dew  humidity  windspeed  …  Year  Month  Day  \
Date                                             …
2021-04-01       26.1  3.1     22.60       22.8  …  2021      4    1
2021-04-02       24.9  4.5     27.62       12.4  …  2021      4    2
2021-04-03       25.1  1.3     23.18       16.5  …  2021      4    3
2021-04-04       26.0  4.8     28.00       18.3  …  2021      4    4
2021-04-05       28.9  8.1     28.85       13.5  …  2021      4    5

            temp_same_day_last_week temp_same_day_last_month  \
Date
2021-04-01                      NaN                      NaN
2021-04-02                      NaN                      NaN
2021-04-03                      NaN                      NaN
2021-04-04                      NaN                      NaN
2021-04-05                      NaN                      NaN

           temp_same_day_last_year  temp_2_month_rolling_avg  \
Date
2021-04-01                     NaN                       NaN
2021-04-02                     NaN                       NaN
2021-04-03                     NaN                       NaN
2021-04-04                     NaN                       NaN
2021-04-05                     NaN                       NaN

           temp_max_up_to_date   Q  Yearly
Date
2021-04-01                27.1 NaN     NaN
2021-04-02                27.1 NaN     NaN
2021-04-03                27.1 NaN     NaN
```

```
2021-04-04                27.1 NaN     NaN
2021-04-05                29.9 NaN     NaN

[5 rows x 29 columns]
```

[ ]: