High Integrity Systems Project Time Series Analysis

Assignment 2

Chitra Khatri
Hellyben Shah
Aniket Nighot
Mehjabeen Jahangeer Khan
Hardikumar Kunt
Parth Patel

November 5, 2024

1 Summary of Modern Time Series Forecasting with Python- Chapter 2

A) Understanding the time series dataset

The initial step in analyzing a new dataset is understanding its origin and the data generation process, even before conducting Exploratory Data Analysis (EDA). This dataset, sourced from the London Data Store and enriched by Jean-Michel D for upload on Kaggle, consists of energy consumption readings from 5,567 London households involved in the Low Carbon London project from November 2011 to February 2014, with readings taken every half hour.

The dataset includes various metadata, such as:

- **Acorn Classification**: Households are categorized into demographic types using the Acorn system, which groups types into broader categories like Affluent Achievers and Financially Stretched. Detailed classifications are available in an accompanying user guide.
- **Customer Groups**: There are two groups of households—one with dynamic time-of-use (dToU) energy pricing throughout 2013 and another with flat-rate tariffs, with dToU prices communicated a day in advance via smart meters or text messages.
- **Additional Enrichment**: The dataset has been enhanced with weather data and information on UK bank holidays.

B) Preparing a data model

First, we need to understand where our data is coming from. Then, we can look through the files to see what information each contains and build a mental model of how they relate. Excel is a great tool for this first pass, but if a file is too large, we can read it in Python and save a sample to view in Excel. Just be careful—Excel can sometimes mess with formats, especially dates, so we shouldn't save any changes Excel makes to the file.

C) Wide, Compact and expanded format of data

- **Compact Form:** In the compact form, each time series is stored in a single row. The time dimension is managed as an array within that row. Two additional columns— `start_datetime` and `frequency`—help recreate the timeline. This format is efficient in memory and faster to process, but it only works with regularly sampled data.
- **Expanded Form:** In the expanded form, each time step in the series is spread across rows. Thus, if a series has n steps, it occupies n rows. Time series identifiers and metadata are repeated across all rows. This format allows for easy handling of each time point individually but can take up more space due to duplication.
- **Wide Format:** In the wide format, each time series is represented by separate columns for each series, with dates as the index or in one of the columns. While this format is common in traditional literature, it

becomes unwieldy with many time series, as it doesn't accommodate additional metadata and complicates database management.

D) Converting the London Smart Meters Dataset into Time Series Format

Preprocessing Steps

- **Global End Date:** Identify the maximum date across all data blocks to set a consistent end date for the time series.
- **Reshape Data:** Transform the half-hourly blocks data from a wide format into a long format with each row containing a date and a single half-hourly block, simplifying handling.

Expanded Form Conversion

The process for converting into expanded form includes:

1. Find the start date.
2. Create a standardized DataFrame covering dates from the start date to the global end date.
3. Left merge each DataFrame by identifier (LCLid) with the standard DataFrame, filling missing data with `np.nan`.
4. After obtaining all individual DataFrames:
    - Concatenate them into one DataFrame.
    - Add an "offset" column to represent half-hourly blocks numerically.
    - Create timestamps by adding a 30-minute offset to the date column.

Compact Form Conversion

The process for converting into compact form includes:

1. Find the start date and time series identifiers.
2. Generate a standard DataFrame based on the start and global end dates.
3. Left merge each LCLid DataFrame with the standard DataFrame, filling missing data with `np.nan`.
4. Sort by date and extract the time series data as an array.
5. Compile the data for each LCLid into a DataFrame with additional metadata like start date, identifier, and series length, using a 30-minute frequency.

The **compact form** is preferred due to its efficiency and lower memory requirements.

E) Handling Missing Data

Handling missing values systematically is important to minimize bias and improve the quality of analysis.

Identify the Missingness Pattern

- **MCAR (Missing Completely at Random):** Data is missing without any relation to other data.
- **MAR (Missing at Random):** Missingness is related to observed data.
- **MNAR (Missing Not at Random):** Missingness is related to unobserved data, which can be more challenging to address.

Decide Whether to Drop or Impute

- **Drop Missing Data:** If only a small percentage of values are missing and are MCAR, it might be reasonable to drop them.
- **Impute Missing Data:** If missing values are numerous or meaningful, proceed to imputation. Choose an imputation method based on data type (categorical or continuous) and amount of missing data.

Choose an Imputation Method

- **Forward Fill:** In forward fill, each missing value is filled with the most recent non-missing value preceding it in the series.
  X $= x_1$,x_2....,x_n where x_i may be missing for some indices i
  For each missing x_i, the forward fill method replaces it with the last observed non-missing value.
  x_{i-k} for k = maxj<i | x_j is not missing
  Suppose X =2,Nan,3,Nan,Nan,5, Forward filling would proceed as: 2,2,3,3,3,5
- **Backward Fill:** Backward fill fills in missing values with the next available non-missing value in the series.
  X $= x_1$,x_2....,x_n where x_i may be missing for some indices i
  For each missing x_i, the backward fill method replaces it with the next observed non-missing value.
  x_{i+k} for k = minj>i | x_j is not missing
  Suppose X =2,Nan,3,Nan,Nan,5, Forward filling would proceed as: 2,3,3,5,5,5
- **Mean imputation:** This method replaces missing values with the mean of the observed data.
  For the dataset X $= x_1$,x_2....,$x_n$with a missing value x_i

$$\bar{x} = \frac{1}{n}\sum_{j=1}^{n} x_j$$

- **Linear Interpolation:** Linear interpolation is a method used to estimate unknown values that fall within the range of known values in a dataset. It assumes that the change between two known values is linear, which allows for the estimation of intermediate values based on the values surrounding them.

$$x_{i+1} = x_i + \frac{x_{i+2} - x_i}{2}$$

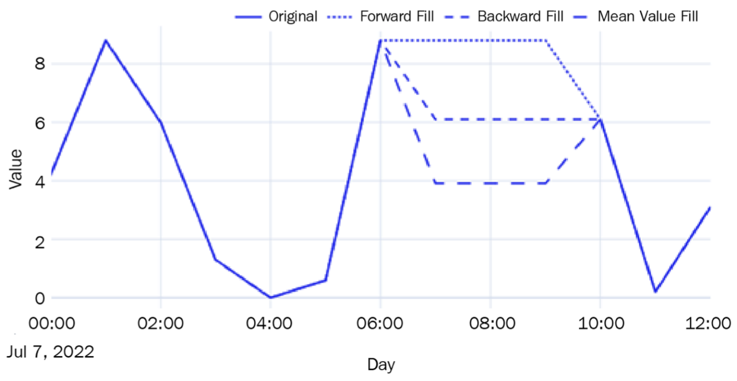

Figure 1: Imputed missing values using forward, backward, and mean

- **Nearest Neighbor Interpolation:** Nearest neighbor interpolation is the simplest form of interpolation. It assigns the value of the nearest known data point to the unknown value.

  y = y_i where i = arg min|x-x_j|. Here y_i is the value of nearest known point(x_i,y_i)

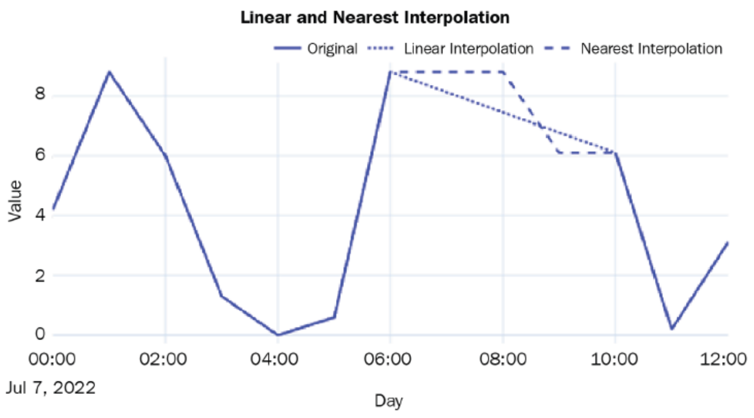

Figure 2: Imputed missing values using Linear and Nearest Interpolation

- **Spline, Polynomial and Other Interpolation:** Spline or Polynomial interpolation , an order needs to be provided. The higher the order, the more flexible the function that is used to fit the observed points.

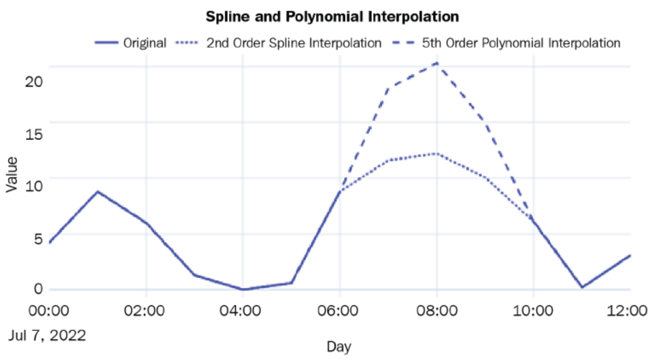

Figure 3: Imputed missing values using Spline and Polynomial Interpolation

F) Handling longer periods of Missing data

Decide Whether to Drop or Impute

- **Drop Missing Data:** If only a small percentage of values are missing and are MCAR, it might be reasonable to drop them.
- **Impute Missing Data:** If missing values are numerous or meaningful, proceed to imputation. Choose an imputation method based on data type (categorical or continuous) and amount of missing data.

Our data, saved in compact form, needs to be expanded using the compacttoexpanded function which makes it easier to work with time series analysis.

- **Imputing with the Previous Day's Data:** It involves using data from the same hour on the preceding day to fill gaps.

  For example, if data is missing for 10 AM on a Monday, we can impute it by using the value from 10 AM on Sunday. This approach works well when daily patterns are stable and consistent, capturing short-term trends effectively. However, it has limitations in cases where weekly or seasonal variations are significant, as it does not account for differences across weekdays, weekends, or seasonal cycles.

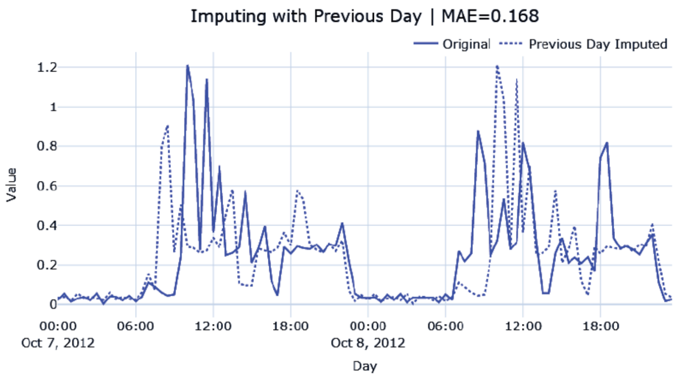

Figure 4: Imputing with Previous Day

- **Hourly Average Profile Imputation:** It involves calculating the average value for each hour across multiple days and using this average to fill missing hourly data.This approach is particularly effective for datasets that exhibit daily patterns or regular hourly fluctuations, such as energy consumption, traffic counts, or temperature readings.

  For example, missing data at 10 AM on Monday would be replaced with the average of all 10 AM values across days. While it effectively captures typical hourly trends, it may miss specific weekly or seasonal variations.

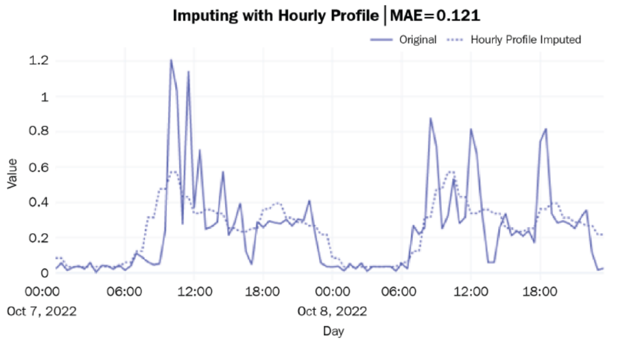

Figure 5: Imputing with an Hourly Profile

- **Hourly Average for Each Weekday :** It imputes missing data by averaging the value for a specific hour across the same weekday over multiple weeks. This method is ideal for data with both daily and weekly patterns, such as retail traffic or office energy use.

  For example, Monday mornings tend to be similar week-to-week. For instance, a missing value at 10 AM on a Monday would be filled with the average of all 10 AM readings from past Mondays. This approach captures both hourly and weekly trends but may miss longer-term seasonal variations.

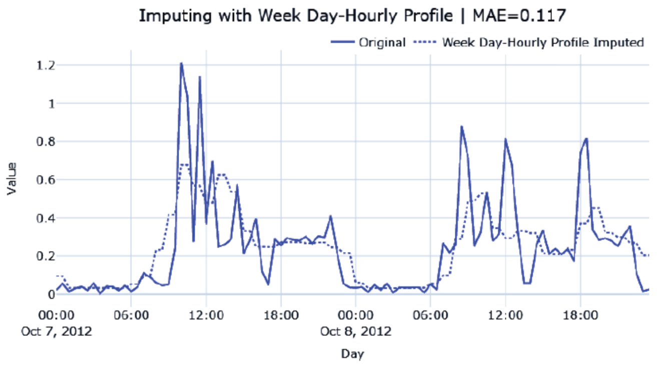

Figure 6: Imputing the hourly average for each weekday

- **Seasonal Interpolation:** While seasonal interpolation generally works well by calculating and using seasonal profiles, it can fall short in time series with a trend, as the simple seasonal profile may not account for trends. To handle this, we can:

  1. Calculate the seasonal profile, similar to how we calculated averages previously.
  2. Subtract the seasonal profile and apply one of the previously discussed interpolation techniques.
  3. Add the seasonal profile back to the interpolated series.

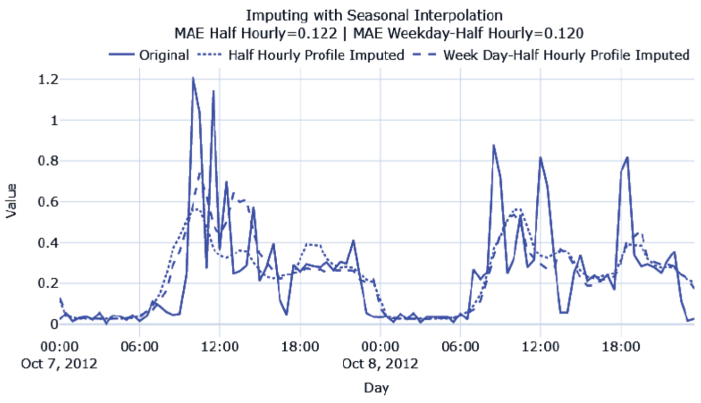

Figure 7: Imputing with Seasonal Interpolation

2 Summary of Modern Time Series Forecasting with Python- Chapter 3

Chapter 3 explores how to analyze and visualize time series data to uncover patterns and insights through Exploratory Data Analysis (EDA). This process includes using visualization techniques and decomposition methods to understand data structure and prepare it for modeling.

Components of a Time Series

- **Trend:** Long-term movements in data, like Tesla's increasing revenue over years. Trends can be linear or nonlinear.
- **Seasonality:** Regular patterns that repeat over fixed periods (e.g., retail sales spikes during holidays). Sunspots, which peak every 11 years, are an example.
- **Cyclical:** Patterns similar to seasonality but irregular in frequency, such as economic recessions.
- **Irregular (Residual):** Random, unpredictable variations that remain after removing trend, seasonal, and cyclical components.

Visualization Techniques for Time Series

- **Line Charts:** Basic plots of data over time. Adding a rolling average can help to smooth high-variation data, as seen with household energy consumption.
- **Seasonal Plots:** These plots visualize seasonality by showing trends over defined seasonal periods (e.g., average monthly consumption across years).
- **Seasonal Box Plots:** Represent seasonality with box plots, showing median, interquartile range, and outliers, to highlight data variability.
- **Calendar Heatmaps:** Show time series data patterns over two time dimensions (e.g., day and month), helping to visualize complex seasonal patterns.

- **Autocorrelation Plots:** Display correlations between a time series and its lagged versions, helpful in identifying dependencies in data. Partial autocorrelation removes indirect correlations for clearer analysis.

Decomposing a Time Series

Time series decomposition breaks down data into its components (trend, seasonality, and residuals). Two methods are discussed:

- **Detrending:** Removing trends using moving averages or LOESS (Locally Estimated Scatterplot Smoothing).
  - Moving Averages: Used for smoothing data to detect trends. Moving averages smooth data by taking averages over a window
  - LOESS: LOESS fits a weighted polynomial regression for a more accurate trend line.
- **Deseasonalizing:** Identifying seasonality through period-adjusted averages or Fourier series.
  - Period-adjusted averages: Period-adjusted averages compute the mean for each season (e.g., each month)
  - Fourier series: Decompose seasonality into sine and cosine waves to capture periodic patterns, with parameters like cycle length (P) and complexity level (N) to control accuracy.Fourier series approximate the seasonality with sine and cosine waves.

3 Handling of outliers in time series

In time series data outlier can have two different meanings as shown in the figure 8

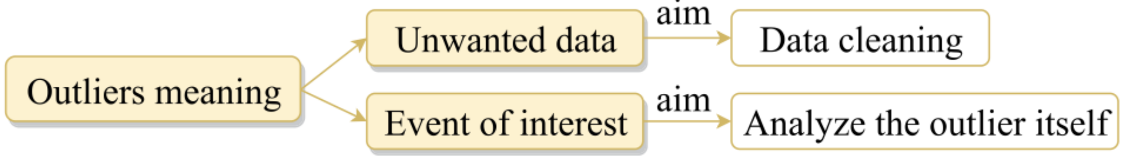

Figure 8: Meaning of the outliers in time series data [1]

For example, sensor transmission errors are eliminated to obtain more accurate predictions, because the principal aim is to make predictions. Nevertheless, in recent years and, especially in the area of time series data, many researchers have aimed to detect and analyze unusual but interesting phenomena. Fraud detection is an example of this, because the main objective is to detect and analyze the outlier itself [1].

Detection Methods

Outlier detection techniques in time series data vary depending on the input data type, the outlier type, and the nature of the method.

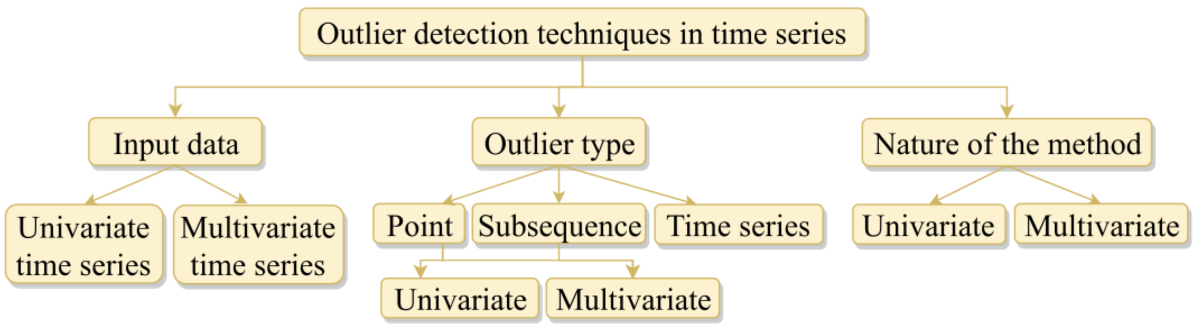

Figure 9: Taxonomy of outlier detection techniques in the time series [1]

Input Data

- **Univariate Time Series**: A single sequence (One Variable) of observations over time.
- **Multivariate Time Series**: A set of multiple correlated sequences (Multiple Variables) observed simultaneously.

Outlier Type

- **Point outliers**: Behaves unusually in a specific time instant when compared either to the other values in the time series (global outlier) or to its neighboring points (local outlier). They can be univariate or multivariate.

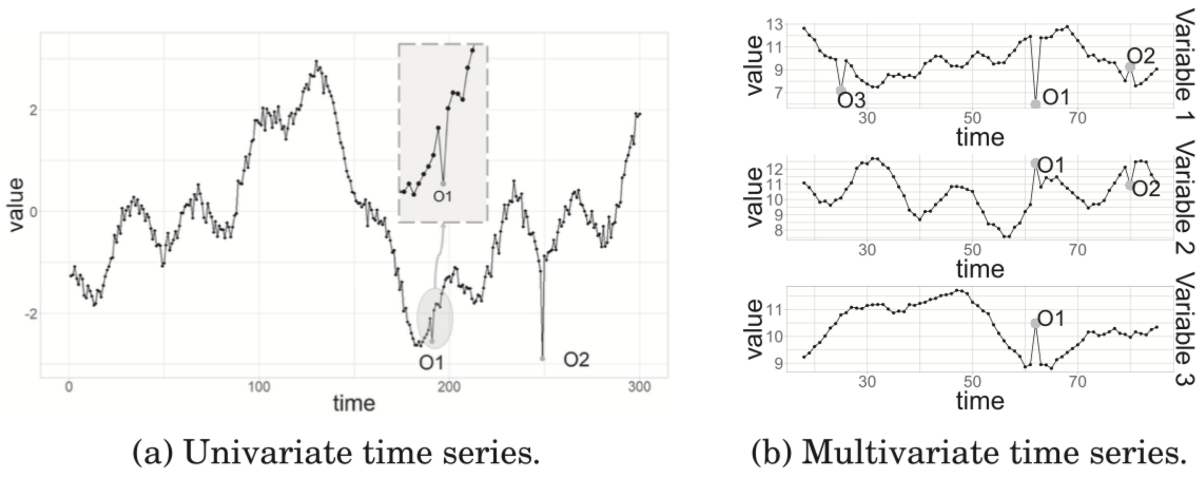

Figure 10: Point outliers in time series [1]

- **Subsequence outliers**: Points in time whose joint behavior is unusual, although each observation individually is not necessarily a point outlier. Subsequence outliers can also be global or local and can affect one (univariate subsequence outlier) or more (multivariate subsequence outlier) time-dependent variables.
- **Outlier timeseries**: Entire time series can also be outliers,but they can only be detected when the input data are a multivariate time series.

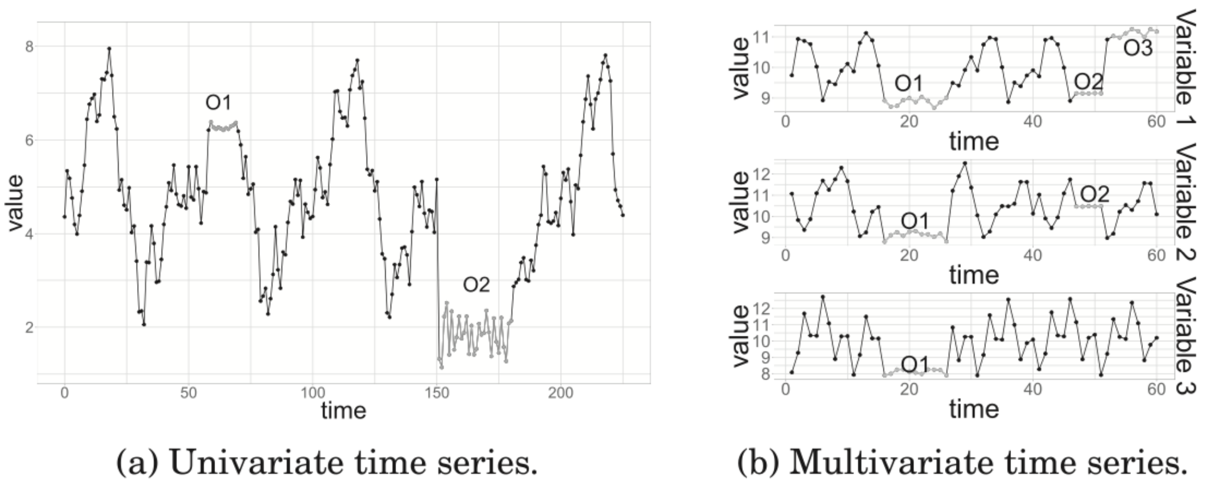

Figure 11: Subsequence outliers in time series [1]

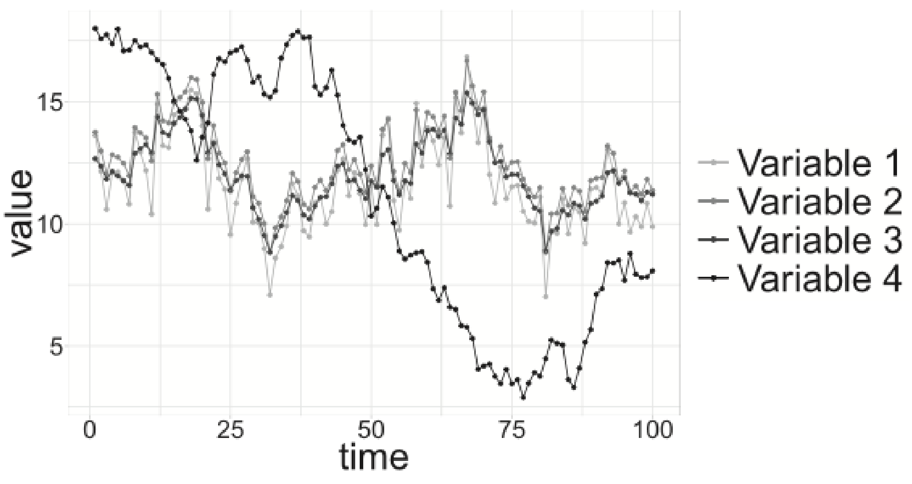

Figure 12: Outlier time series [1]

Statistical Methods

- **Standard Deviation Method/Z-score Analysis**: Points beyond $\mu \pm 3\sigma$ are considered outliers. This can be easily visualized by the famous bell shaped graph for normally distributed data.

  Note: For seasonal data, we must deseasonalize the data using any of the techniques we discussed earlier and then apply the outliers to the residuals. If we don't do that, we may flag a seasonal peak as an outlier. Another key assumption here is the normal distribution. [2]

- **Interquartile Range (IQR) Method/Boxplot Method**: Identifies outliers based on $Q1 - 1.5 * IQR$ and $Q3 + 1.5 * IQR$. This can be visualized by boxplot as well. This method is slightly robust than SD.

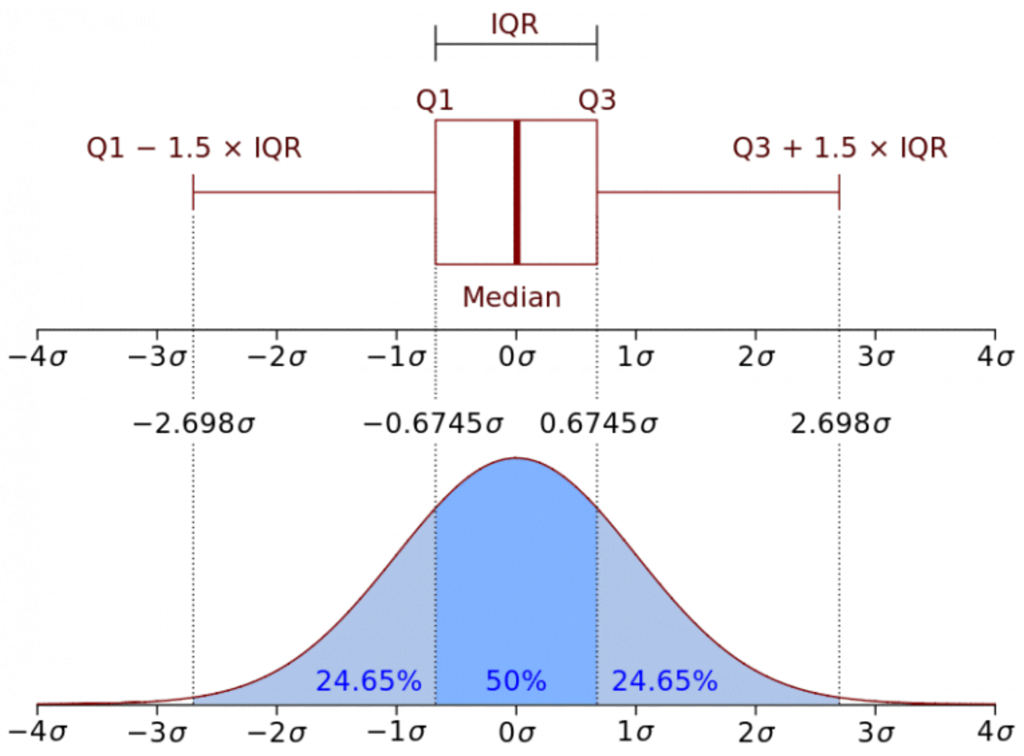

Figure 13: Bell Graph

Machine Learning Approaches

- **Isolation Forest**: An unsupervised algorithm that isolates anomalies by randomly partitioning the data. works on the assumption that the outlier points fall in the outer periphery and are easier to fall into a leaf node of a tree. i.e outlier will have short branches.The "anomaly score" of any point is determined by the depth of the tree to be traversed before reaching that particular point.
- **Clustering-based Methods**: Techniques like k-means or DBSCAN can be used to identify points that do not fit well into any cluster.

Time Series Specific Methods

- **Seasonal-Trend Decomposition**: Decompose the series and analyze residuals for outliers.
- **Generalized ESD (S-ESD)**: Adapts the Extreme Studentized Deviate test for time series data.

Deep Learning Approaches

- **Recurrent Neural Networks (RNNs)**: Particularly LSTM networks, which are effective for capturing temporal dependencies in sequential data.
- **Autoencoders**: Used for reconstructing input data and identifying anomalies based on reconstruction error.

Treatment Strategies

1. **Removal**

2. **Imputation**
3. **Transformation**
4. **Winsorization**
5. **Contextual Analysis**

Considerations

- **Domain Knowledge**: Incorporate domain expertise to distinguish between true anomalies and important events.
- **Seasonality**: Deseasonalize data before applying outlier detection methods to avoid flagging seasonal peaks as outliers.
- **Model Selection**: Some modern forecasting methods may be robust to outliers, potentially reducing the need for extensive outlier treatment.
- **When to Use Statistical Methods**: Best for simple, low-dimensional time series with clear point outliers when data is normally distributed
- **When to Use Machine Learning Algorithms**: Suitable for multivariate data and when contextual (e.g., seasonal variations) dependencies are significant and shows complex behaviour (e.g., trends or seasonal cycles)
- **When to Use Deep Learning Approaches**: Ideal for high-dimensional datasets exhibiting non-linear relationships with complex patterns (e.g., shapelet, seasonal, and trend outliers).

4 Time Series Visualization

Time Series Visualization in Python

Line Plot

A line plot is commonly used to visualize time series data, where time is on the x-axis and the observation values are on the y-axis.

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

# Load the data
sales_data = pd.read_csv("sales_data.csv")
sales_data["date"] = pd.to_datetime(sales_data["date"])
sales_data.set_index("date", inplace=True)
```

```
# Create a line plot
plt.plot(sales_data.index, sales_data["sales"])
plt.xlabel("Date")
plt.ylabel("Sales (USD)")
plt.title("Sales Over Time")
plt.show()
```

Box Plot

Box plots can be useful to see the distribution of values grouped by time intervals.

```
import seaborn as sns

sales_data["year"] = sales_data.index.year
sns.boxplot(data=sales_data, x="year", y="sales")
plt.title("Sales Distribution by Year")
plt.show()
```

Heatmap

A heatmap can compare observations between time intervals, showing total sales by month and year.

```
sales = sales_data.groupby([sales_data.index.year, sales_data.index.month]).sum()
sales_month_year = sales.reset_index().pivot(index="year", columns="month", values="sales")

sns.heatmap(sales_month_year, cbar_kws={"label": "Total Sales"})
plt.title("Sales Over Time")
plt.xlabel("Month")
plt.ylabel("Year")
plt.show()
```

Autocorrelation Plot

This plot shows whether the elements of a time series are correlated.

```
from pandas.plotting import autocorrelation_plot

autocorrelation_plot(sales_data["sales"])
plt.title("Autocorrelation of Sales Data")
plt.show()
```

Time Series Visualization in MATLAB

Basic Time Series Plot

MATLAB provides straightforward plotting capabilities for time series data.

```
% Load the data
data = readtable('sales_data.csv');
data.date = datetime(data.date);

% Create a time series plot
figure;
plot(data.date, data.sales);
xlabel('Date');
ylabel('Sales (USD)');
title('Sales Over Time');
```

Subplots for Multiple Time Series

You can visualize multiple time series in a single figure using subplots.

```
% Assuming 'speed' and 'torque' are columns in the data
figure;
subplot(2, 1, 1);
plot(data.date, data.speed);
title('Speed Over Time');
ylabel('Speed (RPM)');

subplot(2, 1, 2);
plot(data.date, data.torque);
title('Torque Over Time');
ylabel('Torque (Nm)');
xlabel('Date');
```

Box Plot

MATLAB also supports box plots for visualizing distributions.

```
% Create a box plot for sales data by year
data.year = year(data.date);
boxplot(data.sales, data.year);
title('Sales Distribution by Year');
xlabel('Year');
```

```
ylabel('Sales␣(USD)');
```

Heatmap

You can create heatmaps to visualize data over time.

```
% Create a heatmap of sales by month and year
sales_by_month = groupsummary(data, 'year', 'month', 'sum', 'sales');
heatmap(sales_by_month.year, sales_by_month.month, sales_by_month.sum_sales);
title('Sales␣Heatmap');
xlabel('Year');
ylabel('Month');
```

5 Kalman Filter

The Kalman filter is essentially a set of mathematical equations that implement a predictor-corrector type estimator that is optimal in the sense that it minimizes the estimated error covariance—when some presumed conditions are met. Since the time of its introduction, the Kalman filter has been the subject of extensive research and application, particularly in the area of autonomous or assisted navigation.

Core Principles of the Kalman Filter

The Kalman Filter operates on several key principles, which give it the ability to estimate the hidden state of a dynamic system:

- **State Estimation in Dynamic Systems**: At each time step, the Kalman Filter estimates the system's current state and the uncertainty in that estimate. It then updates these estimates when new measurements become available, balancing the prediction with the new information.
- **Optimality Under Gaussian Assumptions**: The Kalman Filter is mathematically optimal when:
  - The system dynamics are **linear**,
  - The process and measurement noise are **Gaussian** with zero mean and known covariances. Under these assumptions, the Kalman Filter minimizes the mean squared error of the estimated state.
- **Recursive Process**: The Kalman Filter is recursive, meaning that it only requires the previous state estimate and measurement to generate the current estimate. This recursive property makes it highly efficient for real-time applications, as it avoids the need to store extensive historical data.
- **Minimizing the Covariance (Error Uncertainty)**: The Kalman Filter is designed to minimize the error covariance, which quantifies the uncertainty in the state estimate. This is done by adjusting the weight (Kalman Gain) between the prediction and the measurement at each time step.

Mathematical Foundation

The mathematical foundation of the Kalman Filter is rooted in **linear system modeling**, **stochastic processes**, and **probability theory**. Let's explore the model, key equations, and how these relate to the filter's operation.

System Model

The Kalman Filter estimates the **state** of a linear discrete-time dynamic system, which can be described by two equations:

State Transition Model:

$$x_k = Ax_{k-1} + Bu_k + w_k$$

Measurement Model:

$$z_k = Hx_k + v_k$$

Prediction and Update Steps

Prediction Step:

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_k$$

$$P_k^- = AP_{k-1}A^T + Q$$

Update Step:

- **Kalman Gain Calculation:**

$$K_k = P_k^- H^T(HP_k^- H^T + R)^{-1}$$

–

- **State Update with Measurement:**

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-)$$

–

- **Error Covariance Update:**

$$P_k = (I - K_k H)P_k^-$$

–

Here, K_k represents the Kalman gain, balancing the estimate's reliance on prediction versus the measurement.

Different Variants of the Kalman Filter

- **Extended Kalman Filter (EKF):** Designed for non-linear systems by linearizing the process and measurement functions around the current estimate.
- **Unscented Kalman Filter (UKF):** Utilizes a set of carefully chosen sample points to more accurately capture the mean and covariance, better suited for highly non-linear systems than EKF.
- **Ensemble Kalman Filter (EnKF):** Useful for large-scale systems, it propagates an ensemble of estimates instead of covariance, commonly applied in geophysical models.
- **Information Filter:** An alternative formulation using the information matrix (inverse of covariance), which can sometimes simplify computation, especially in multi-sensor networks.
- **Square-Root Kalman Filter:** Improves numerical stability by calculating square roots of covariance matrices instead of the covariance matrix directly.

These examples illustrate how to effectively visualize time series data using both Python and MATLAB, allowing for insights into trends, distributions, and correlations over time.

***NOTE**: The corresponding code for chapter 2 and chapter 3 is added in the shared Github repository*

References

[1] A. Blázquez-García, A. Conde, U. Mori, and J. A. Lozano, “A review on outlier/anomaly detection in time series data,” *ACM computing surveys (CSUR)*, vol. 54, no. 3, pp. 1–33, 2021.

[2] M. Joseph, *Modern Time Series Forecasting with Python: Explore industry-ready time series forecasting using modern machine learning and deep learning.* Packt Publishing Ltd, 2022.