Time Series Analysis

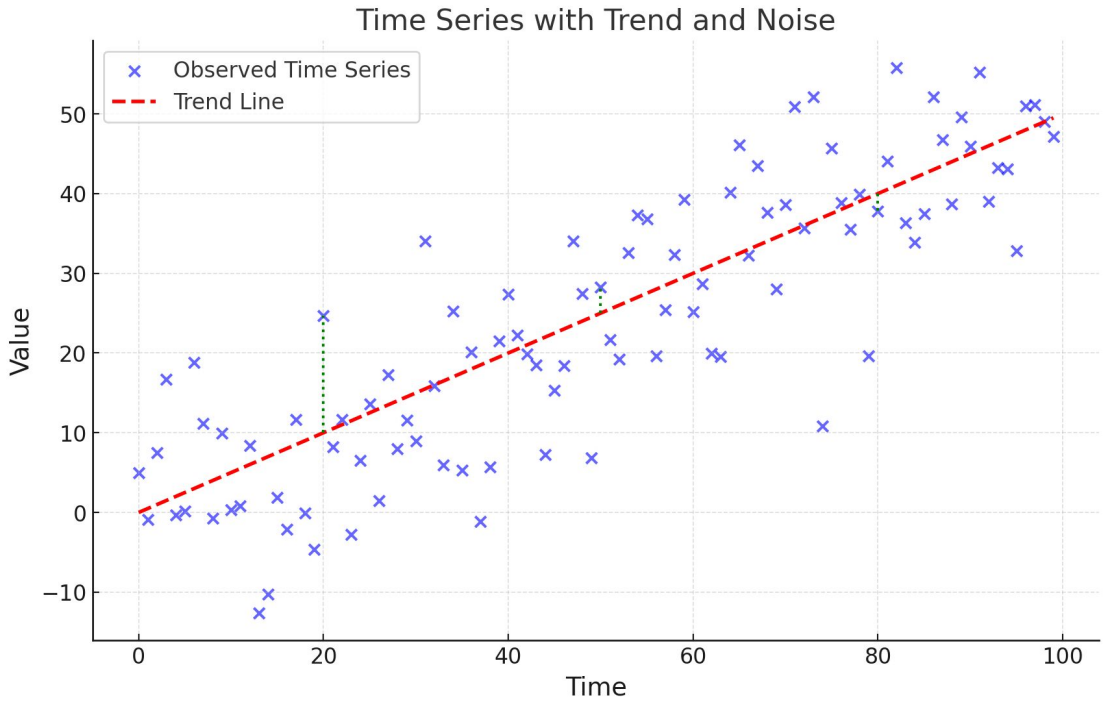
Shabab Akhter
External Lecturer
RUC

Contents

Contents

Introduction to Time Series

Time series analysis involves examining data points collected at regular time intervals to uncover patterns, trends, and relationships over time. Unlike other types of data, time series data is time-dependent, and the order of observations is crucial.



They are mostly used for applications such as:

Finance: Stock price prediction, portfolio optimization, identify trends, volatility, etc.

Weather Forecasting: Predicting daily temperatures and rainfall, etc.

Healthcare: Monitoring patient vitals like heart rate or glucose levels over time detects abnormalities and predicts potential health risks for timely interventions.

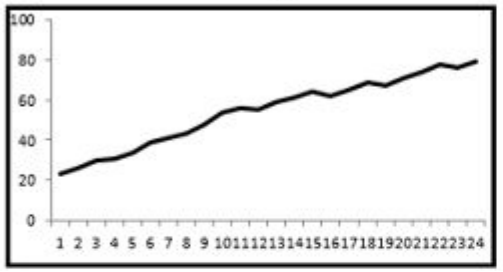
Key Components of Time Series

Trend: The long-term movement or direction in a time series data. It indicates whether the data is generally increasing, decreasing, or remaining stable over time.

E.g: A steady rise in global temperatures due to climate change or increasing population over time.

Types:

- Upward
- Downward
- Flat

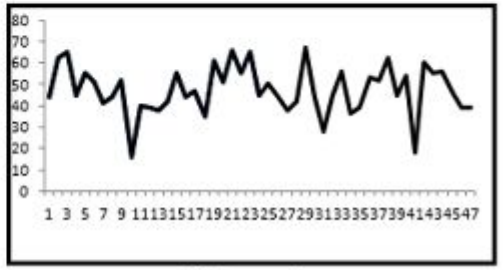


Noise: Random variations in the data that cannot be explained by the trend or seasonal components. Noise is unpredictable and often caused by external factors.

E.g: In a time series of monthly sales data, a sudden, unexplained drop in sales in a particular month due to a supply chain disruption is considered noise, as it doesn't follow any pattern of trend or seasonality.

Types:

- White
- Gaussian
- Autoregressive (AR)

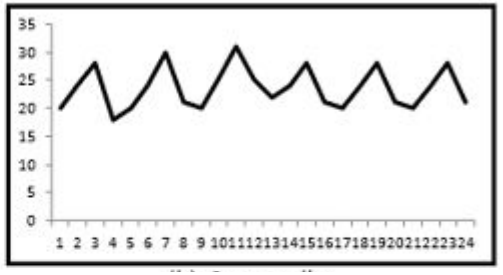


Seasonality: The repeating fluctuations or patterns in time series data at regular intervals due to seasonal factors.

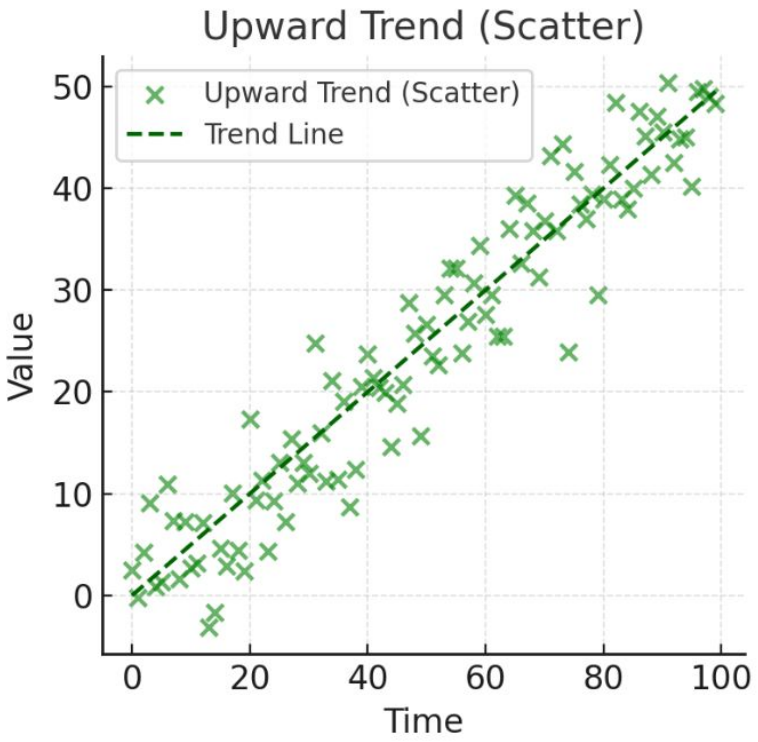
E.g: Higher sales of retail products during holiday seasons or increased energy consumption during winter months.

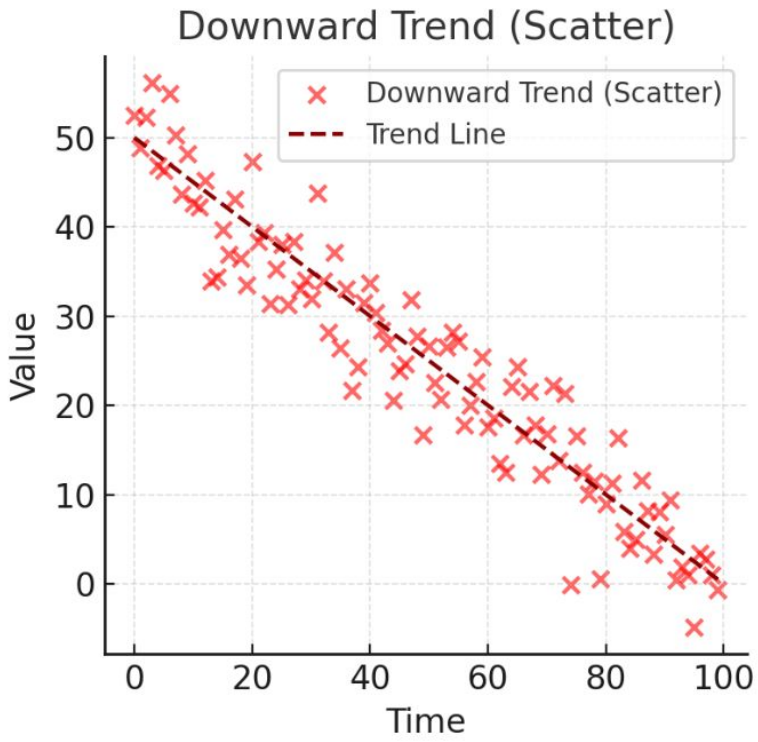
Types:

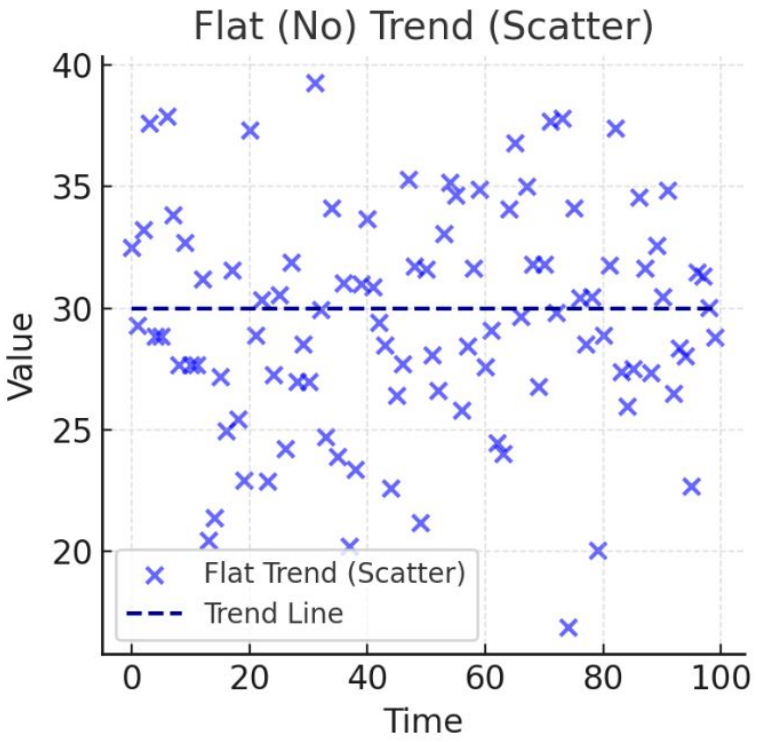
- Annual
- Monthly
- Weekly
- Daily



Looking deeper into trends

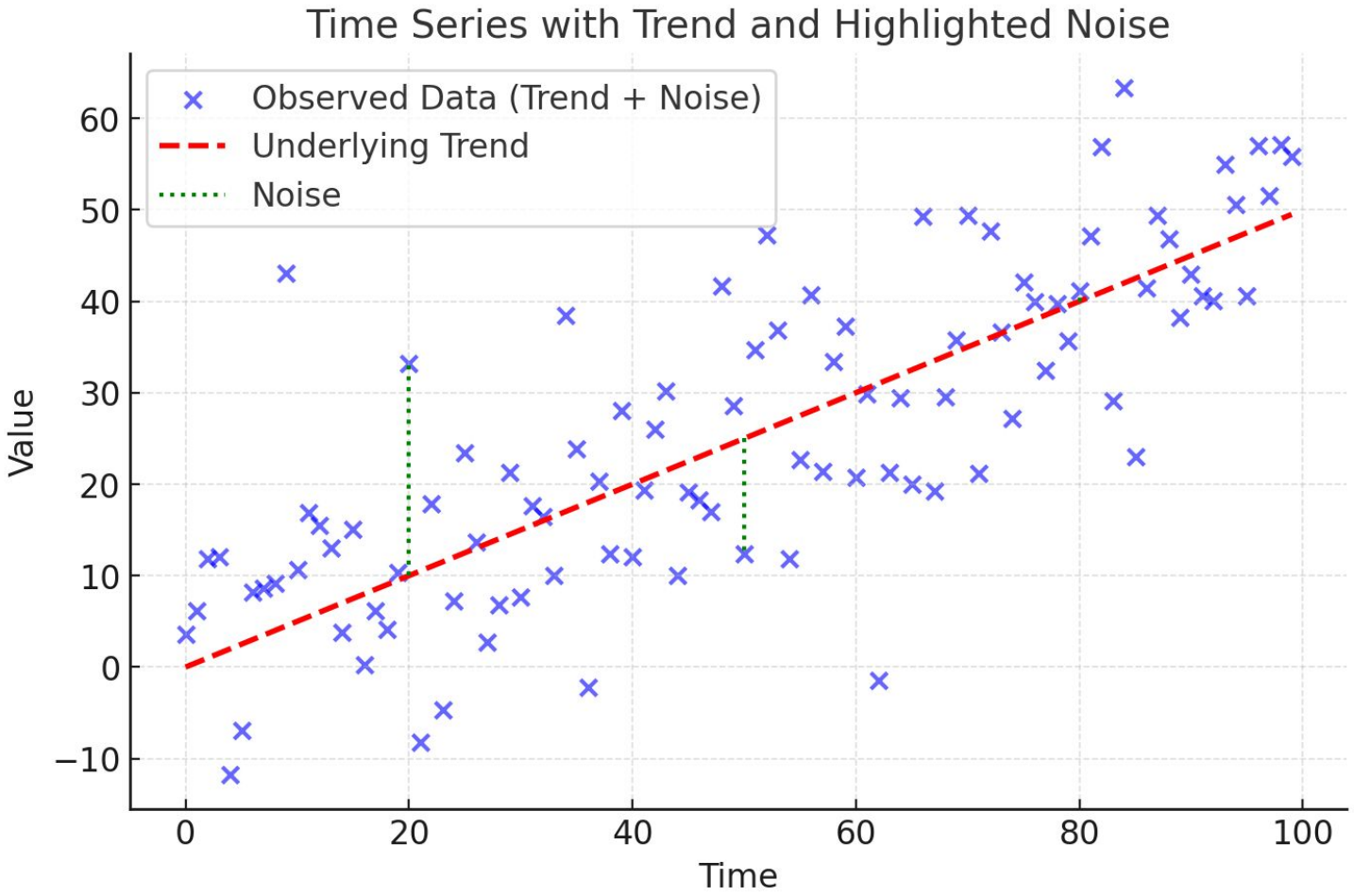






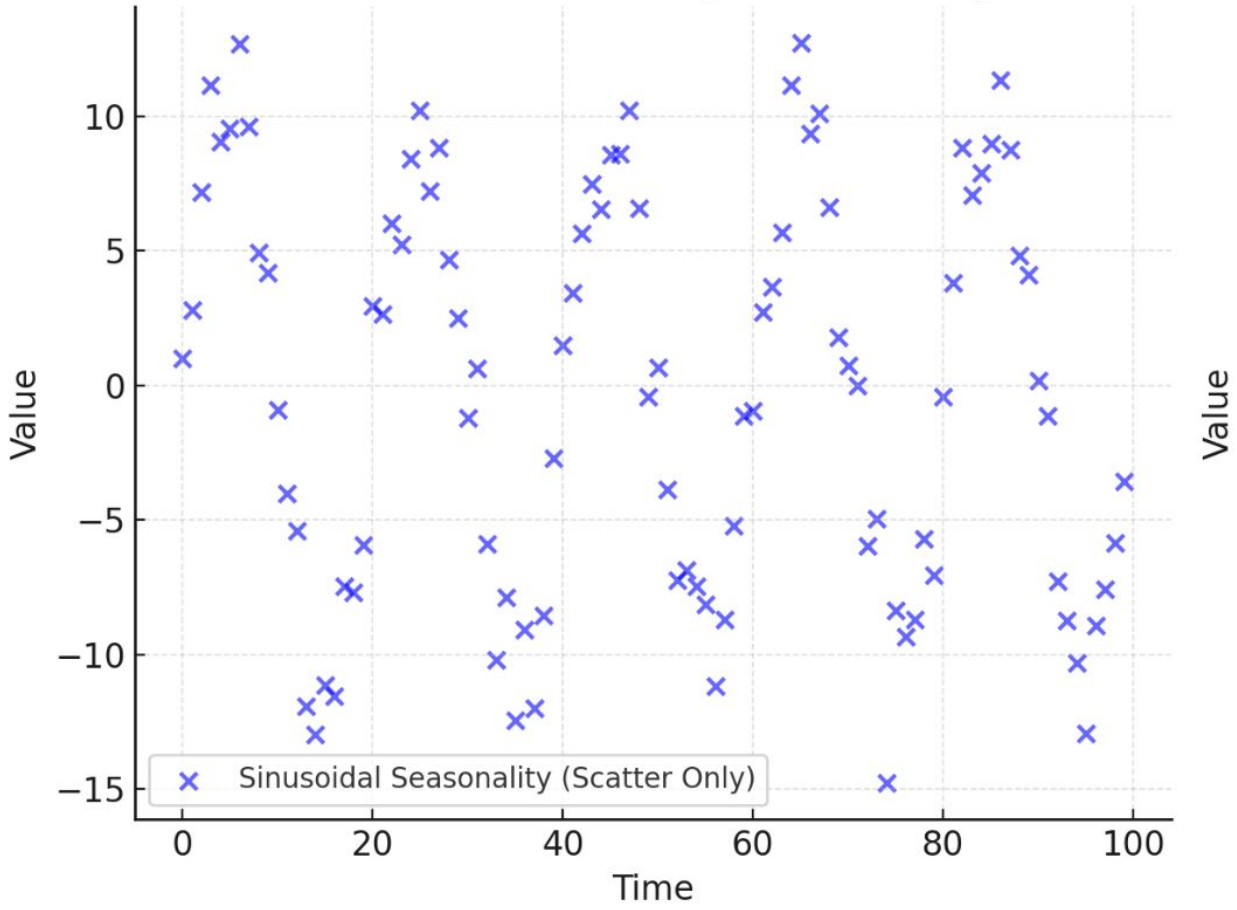
Simply seeing a lot of data points have no value. In order to make a decision, it is important to understand the pattern in the data and here we need to know the trend.

Looking deeper into noise



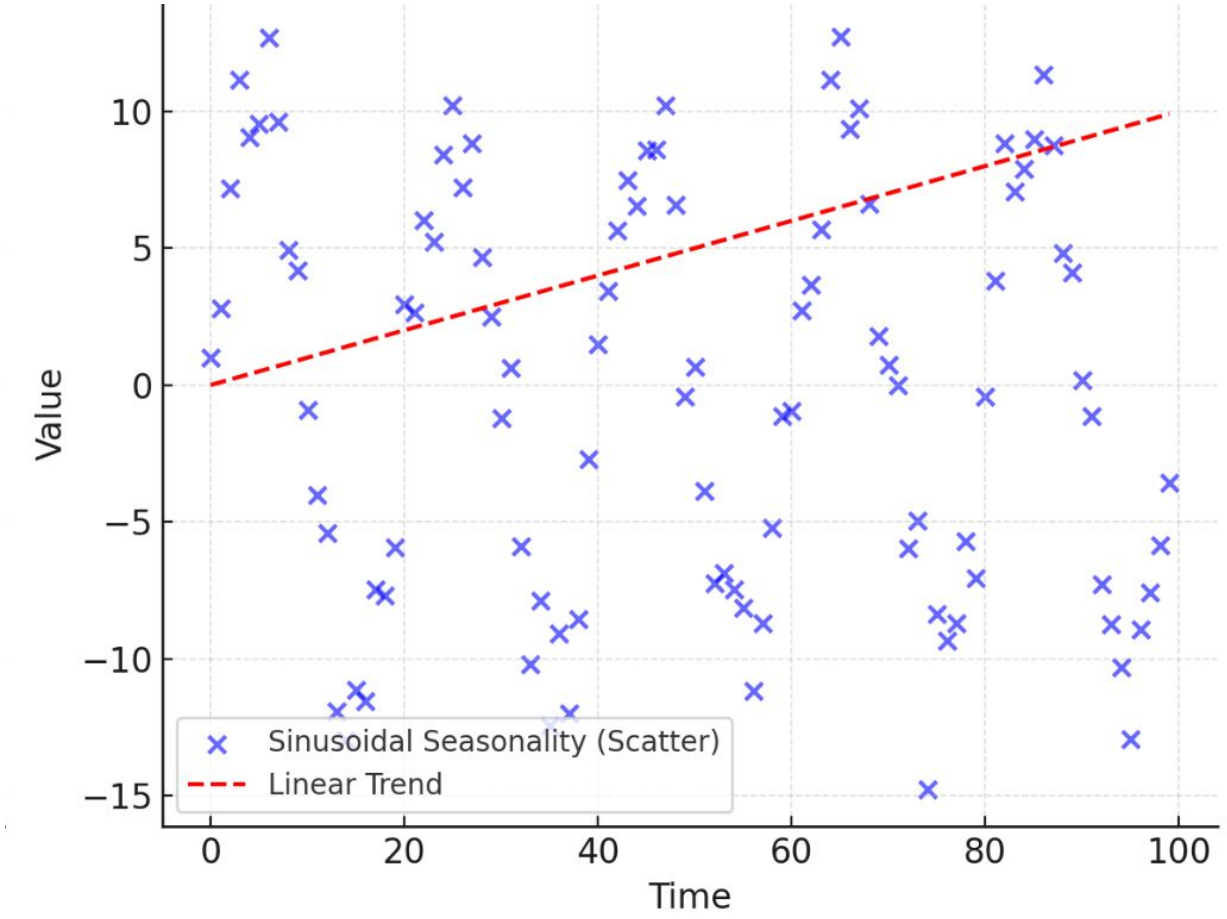
Real-World Representation: Most real-world data combines an underlying pattern with noise.
Understanding Trends: Identifying the trend helps reveal the systematic behavior of the data.
Managing Noise: Recognizing and accounting for noise is crucial for accurate predictions and decision-making.
Applications: This is common in areas like stock markets, climate data, and economic indicators, where distinguishing trend from noise leads to actionable insights.

Looking deeper into seasonality



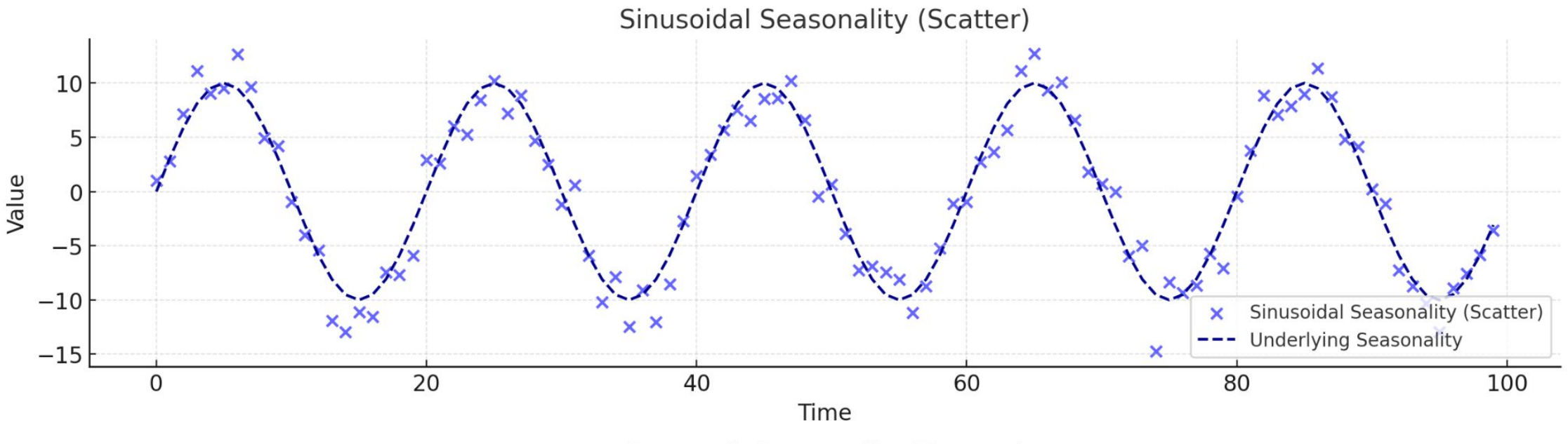
Do you see any trend or is this just noise?

Looking deeper into seasonality



Does the trend line give an accurate representation?

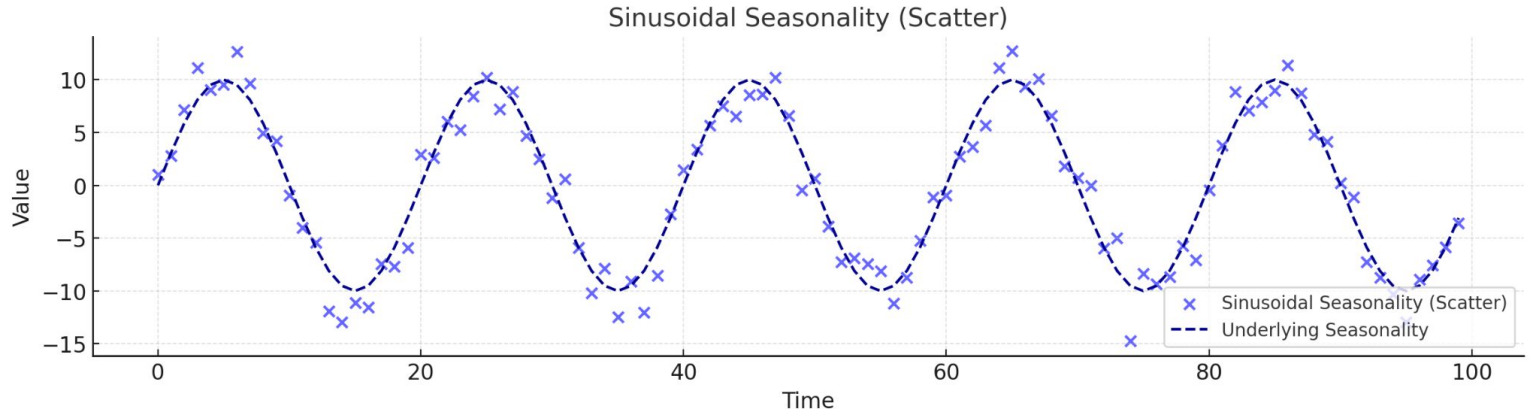
Looking deeper into seasonality

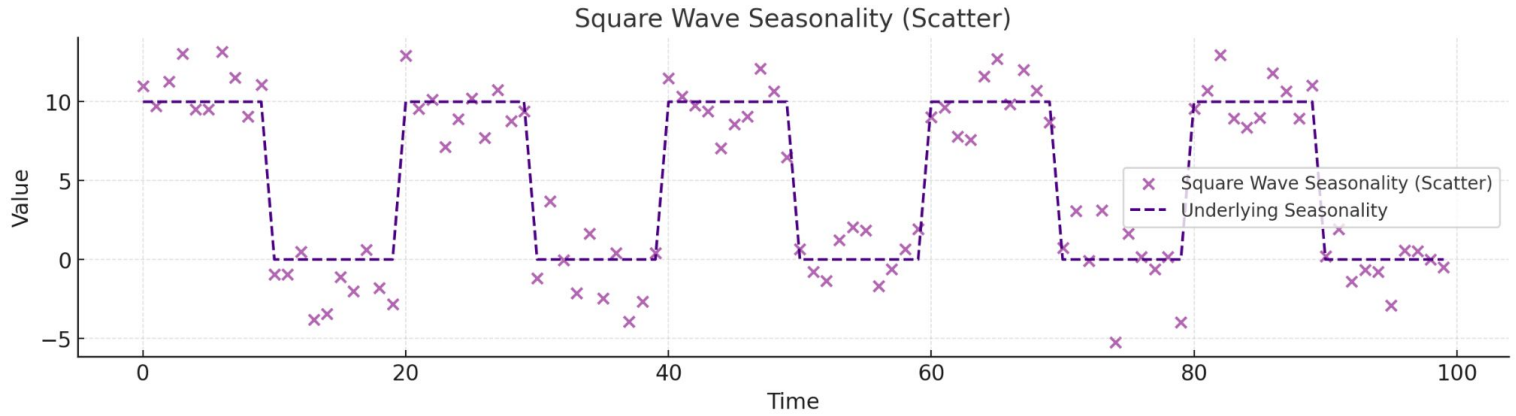


Some times trends are not linear & if they are recurring, then we can say it is a *seasonal behavior*

Seasonality

Some more examples





Exercise

1. Let’s see some real life examples
2. Complete the following tasks

1. Load the dataset from Kaggle. Use the "hour.csv" file.
2. Visualize the structure of the dataset using appropriate libraries and plots.

Contents

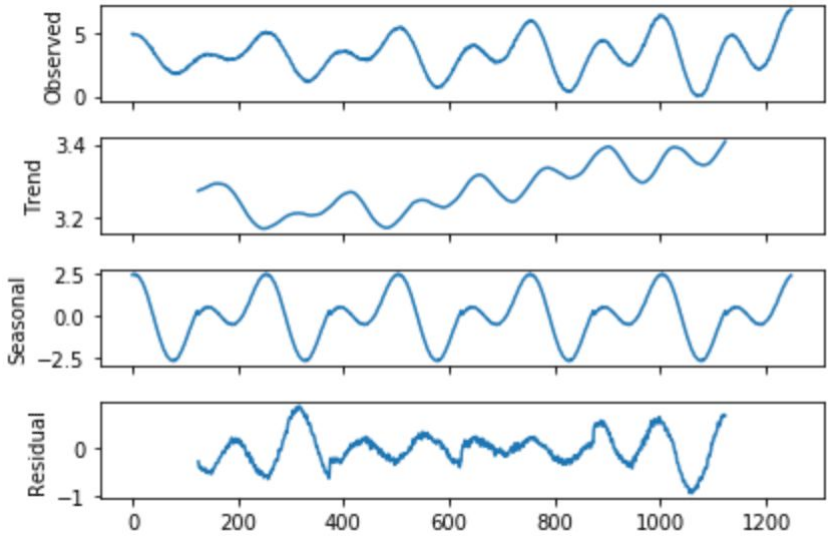
Autocorrelation

What is Autocorrelation?

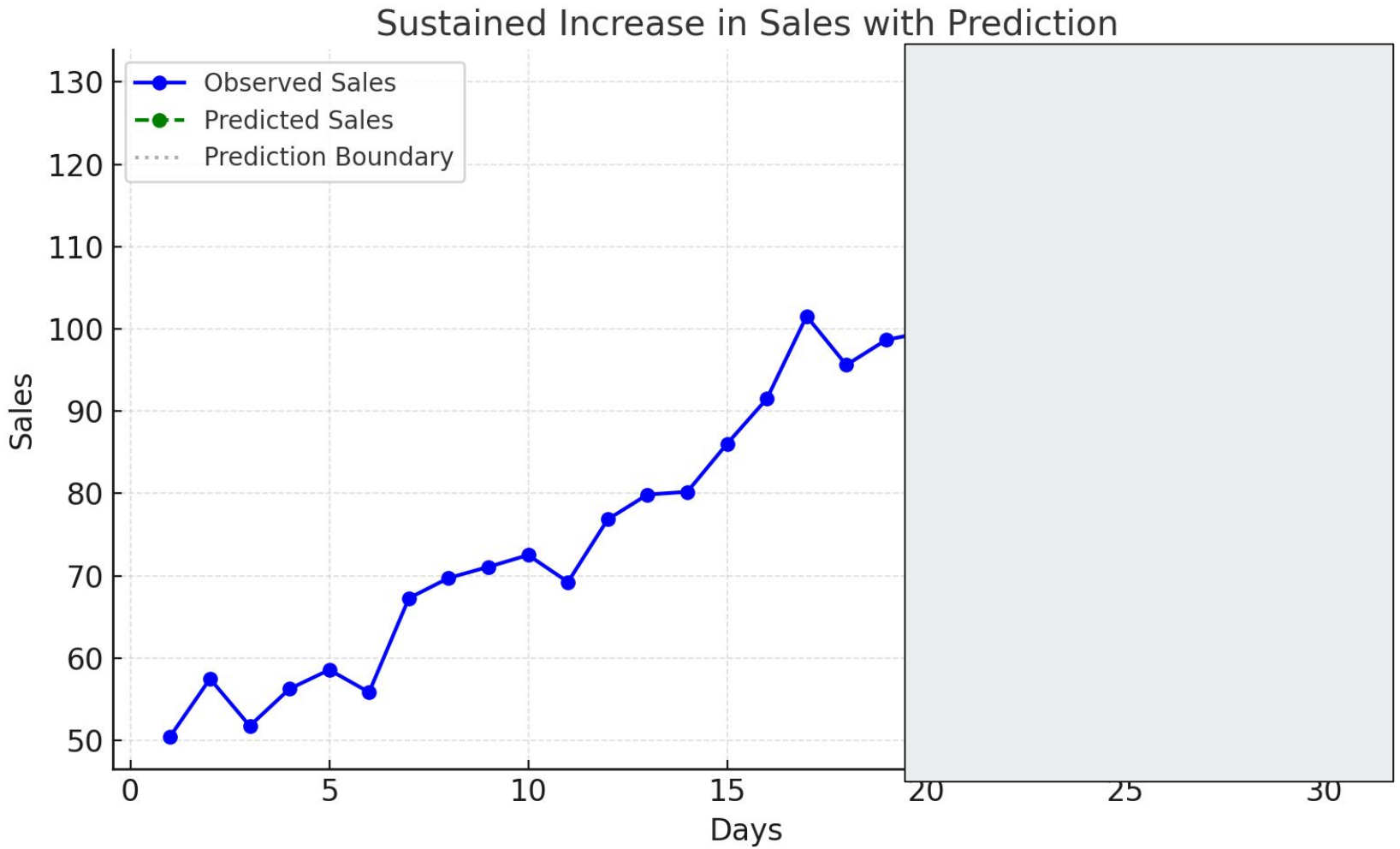
- Autocorrelation measures the correlation of a time series with its own past values over different time lags.
- It helps identify repeating patterns or dependencies over time.

Why it is important?

- Detects trends, cycles, or seasonality in the data.
- Used to build models like ARIMA, where past values are used to predict future ones.

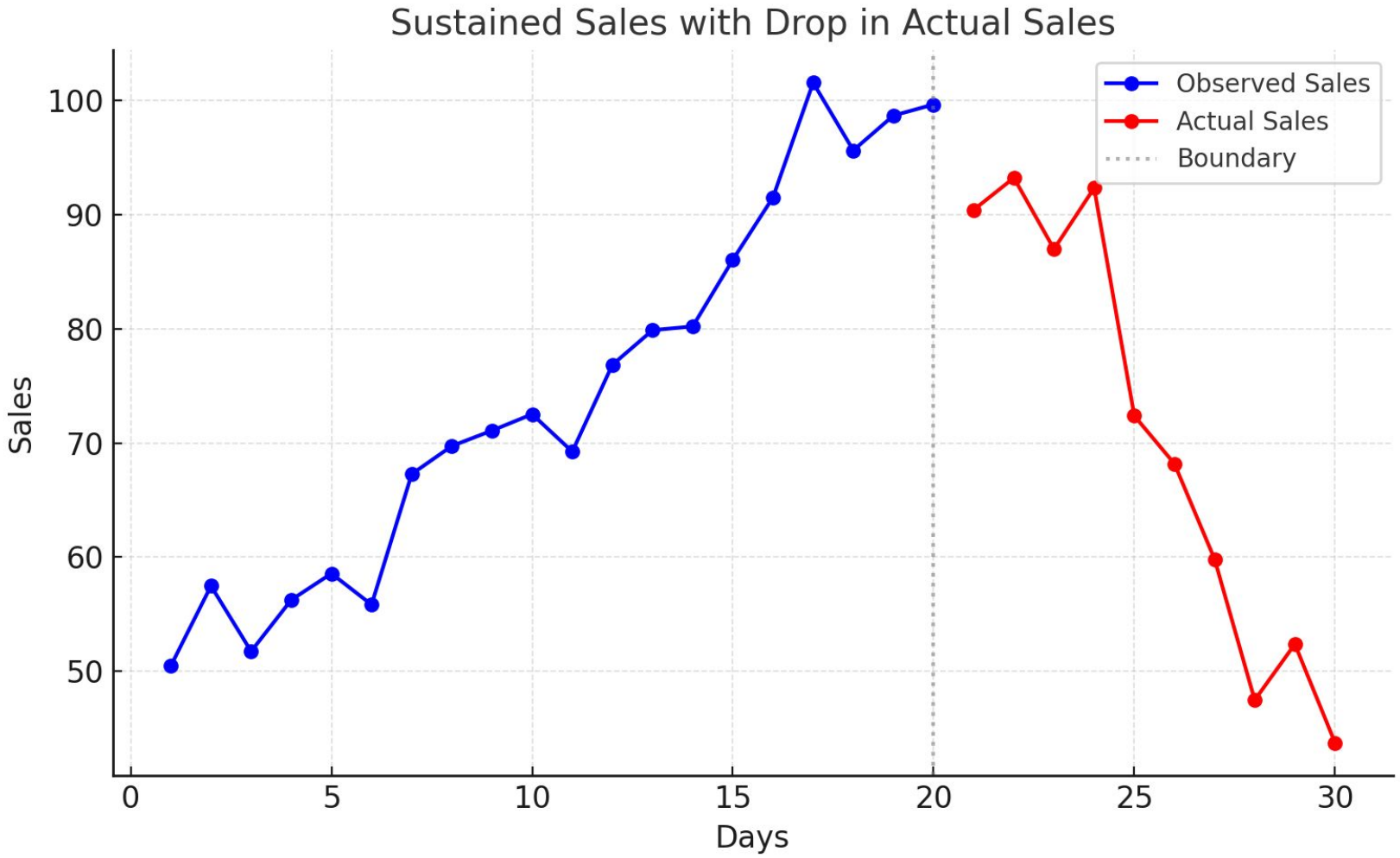


Let's take a real life example



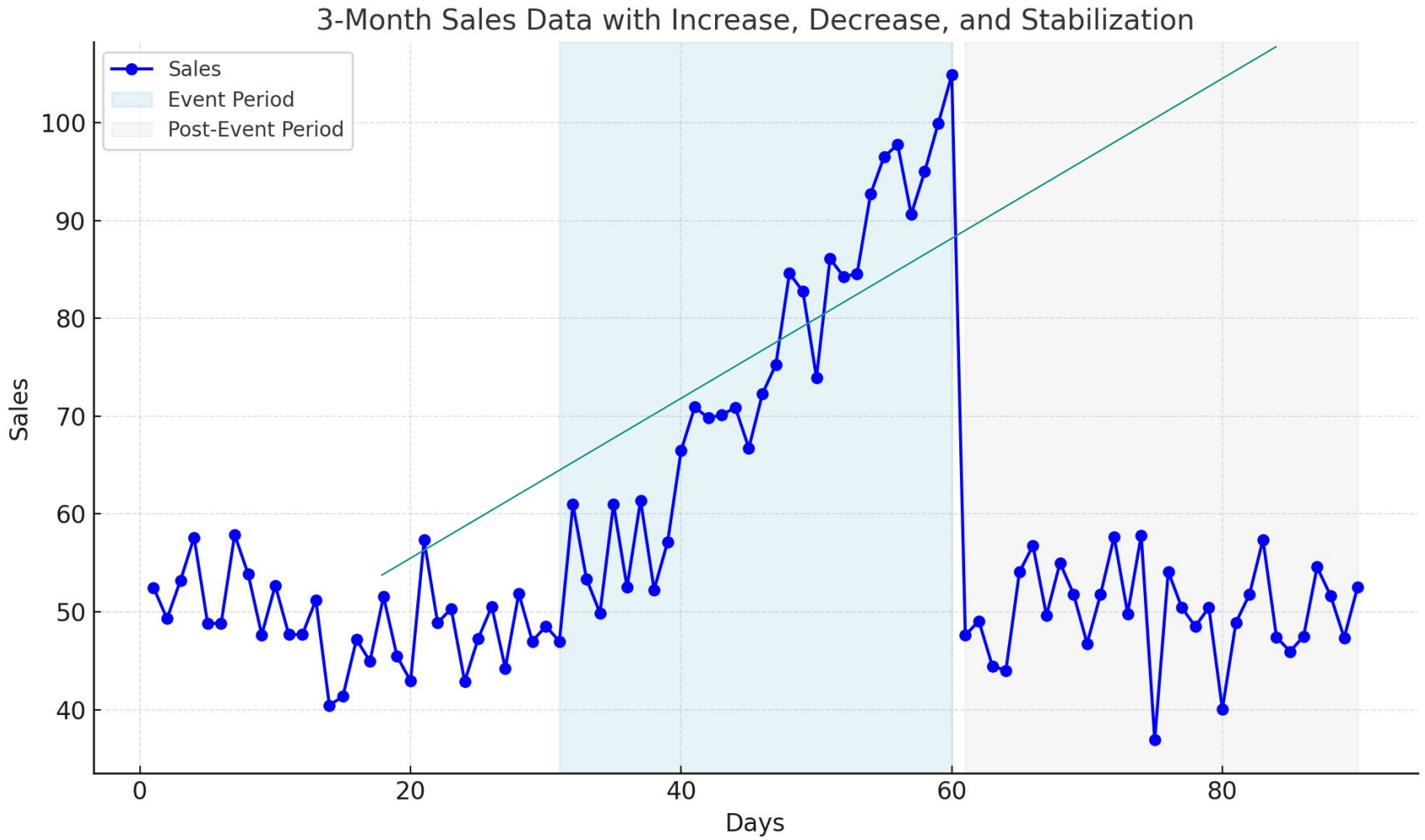
What do you think the sales will be in day 20 - 30?

Reality is actually different



Why do you think this happened?

Autocorrelation can mislead us



Following the prediction would mean that we overstock and hence **lose A LOT of money!**

Autocorrelation is very important

What is it?

1. If sales are **high** on one day, they often stay **high** for a few days.
2. Similarly, if sales are **low** on one day, they tend to stay **low** for the next few days.

This pattern happens because yesterday's sales influence today's sales. For instance:

- If it's **cold weather**, people might crave more chocolate, leading to consistently high sales for a few days.
- If it's a holiday season, chocolates sell more over multiple days.
- On the flip side, if chocolates run out of stock for a day, it might take a while to restock, keeping sales low for subsequent days.

Why Is It a Problem for Predictions?

When using past data to predict future sales:

1. **Overconfidence in Patterns**:
   - If we don't account for autocorrelation, the model might **overestimate its ability to predict**.
   - This could lead to **overstocking or understocking**.
2. **Missed Dynamics**:
   - If your model assumes sales are independent each day, it misses the fact that **yesterday's sales affect today's**.
   - Example: If Valentine's week shows high sales every day, the model might underpredict sales for Tuesday after seeing high sales on Monday.

It can be fixed using ARIMA models, which we will learn about later.

Feature Engineering

Feature engineering involves transforming raw time series data into a format that can be effectively used for building machine learning models. The goal is to extract meaningful features that capture underlying patterns, trends, and dependencies in the data.

Lag features refer to past observations in the time series, shifted by one or more time steps. These features capture the temporal dependencies that help predict future values.

Lagging a feature shifts the data by a specified time window.

```
df['lag_1'] = df['value'].shift(1)
```

Use Cases:

- Predict future values based on historical trends.
- Common in models like ARIMA and SARIMA.

Rolling statistics are used to smooth out short-term fluctuations and highlight longer-term trends or cycles. Common techniques include rolling mean, rolling sum, and rolling standard deviation.

A rolling window of a specific size is applied to calculate the statistics over time.

```
df['rolling_mean'] =
df['value'].rolling(window=7).mean()
```

Use Cases:

- Smoothing noisy data.
- Identifying underlying trends in data.

Exponential Moving Average (EMA) gives more weight to the most recent observations, making it more sensitive to recent changes in the data.

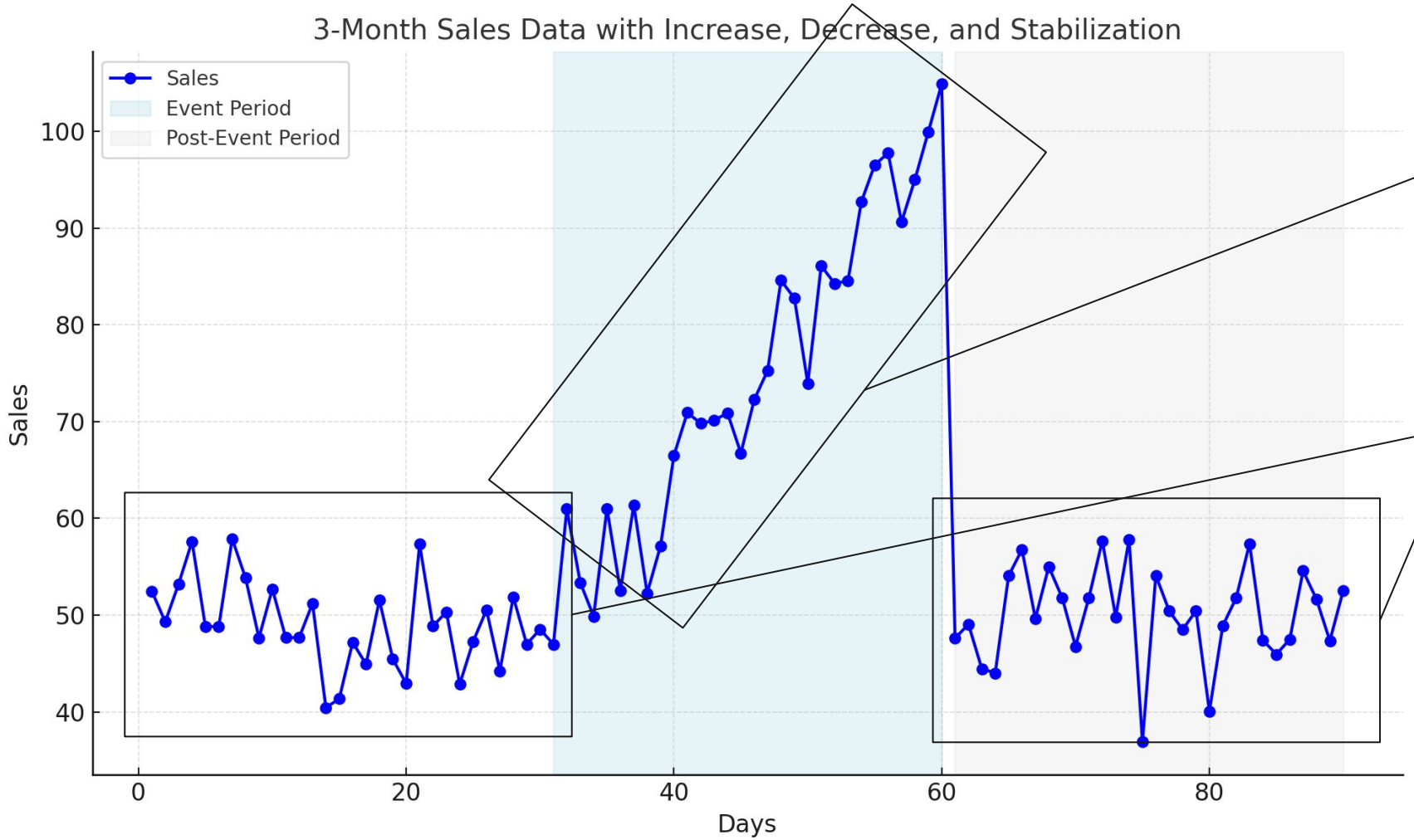
EMA is calculated using a smoothing factor that controls the weight given to recent data.

```
df['ema'] = df['value'].ewm(span=12,
adjust=False).mean()
```

Use Cases:

- Capturing short-term trends in volatile data.
- Often used in financial time series for trend analysis and forecasting.

Looking deeper into lag

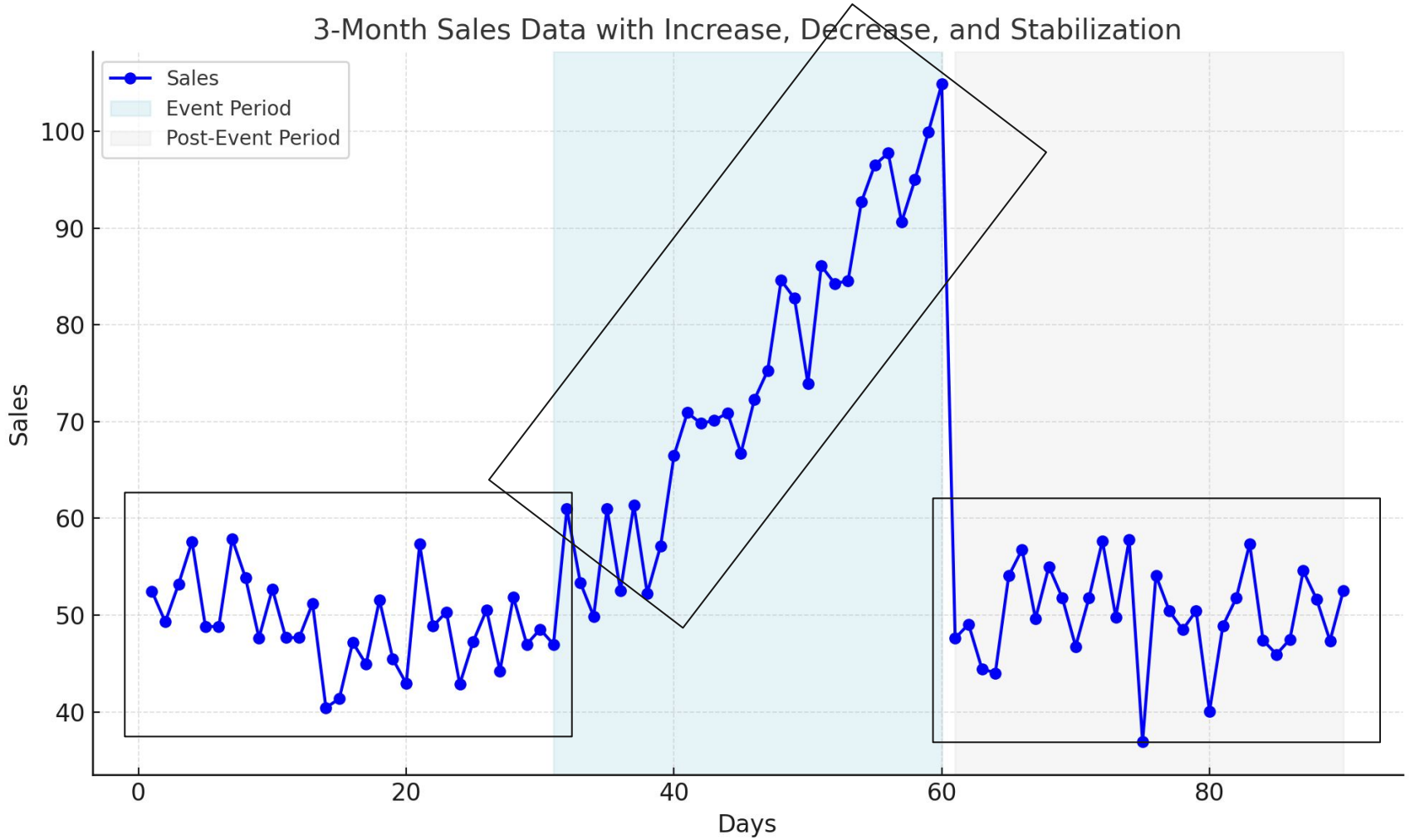


Lagged value of this period in previous year will help to predict this part

Lagged value from 2 months ago will help us predict this part

Lagged values give us more insights into the data and hence help us predict better

Looking deeper into rolling statistics



Rolling statistics of preceding weeks help us in both scenarios

But there will be some errors at the start and then it will pick up

We need to implement both rolling statistics & lagged features to get all the insights

Exercise

1. Let’s see some real life examples
2. Complete the following tasks

3. Clean and pre-process the dataset as required and prepare the data for modelling.
4. Create the lag and rolling windows features for the "cnt" column.

Contents

Linear models

Linear models are statistical models that assume a linear relationship between the dependent variable and one or more independent variables. They are used to predict future outcomes based on past data.

Types of Linear Models:

- **Simple Linear Regression:** A method that models the relationship between a single independent variable and a dependent variable using a straight line.
- **Multiple Linear Regression:** A method that models the relationship between two or more independent variables and a dependent variable using a linear equation.

The general form of the linear model equation is:

$$y=\beta 0+\beta 1x1+\beta 2x2+...+\beta nxn+\epsilon$$

where:

- y: Dependent variable (target).
- x1,x2,...,xn: Independent variables (predictors).
- β0,β1,...,βn: Model coefficients (weights).
- ϵ: Error term.

Simple Linear Regression

Simple linear regression models the relationship between a single independent variable and a dependent variable by fitting a straight line, assuming a linear relationship.

They are mostly used for applications such as:

Predictive Analytics: Estimating future sales, revenue, or stock prices based on past trends.

Trend Analysis: Analyzing the effect of time or other continuous variables on outcomes, like temperature or population growth.

Risk Assessment: Evaluating the relationship between a risk factor (e.g., advertising spend) and outcomes (e.g., sales growth).

Pros

- Easy to implement
- Efficient to train and predict, even on large datasets.
- Low complexity

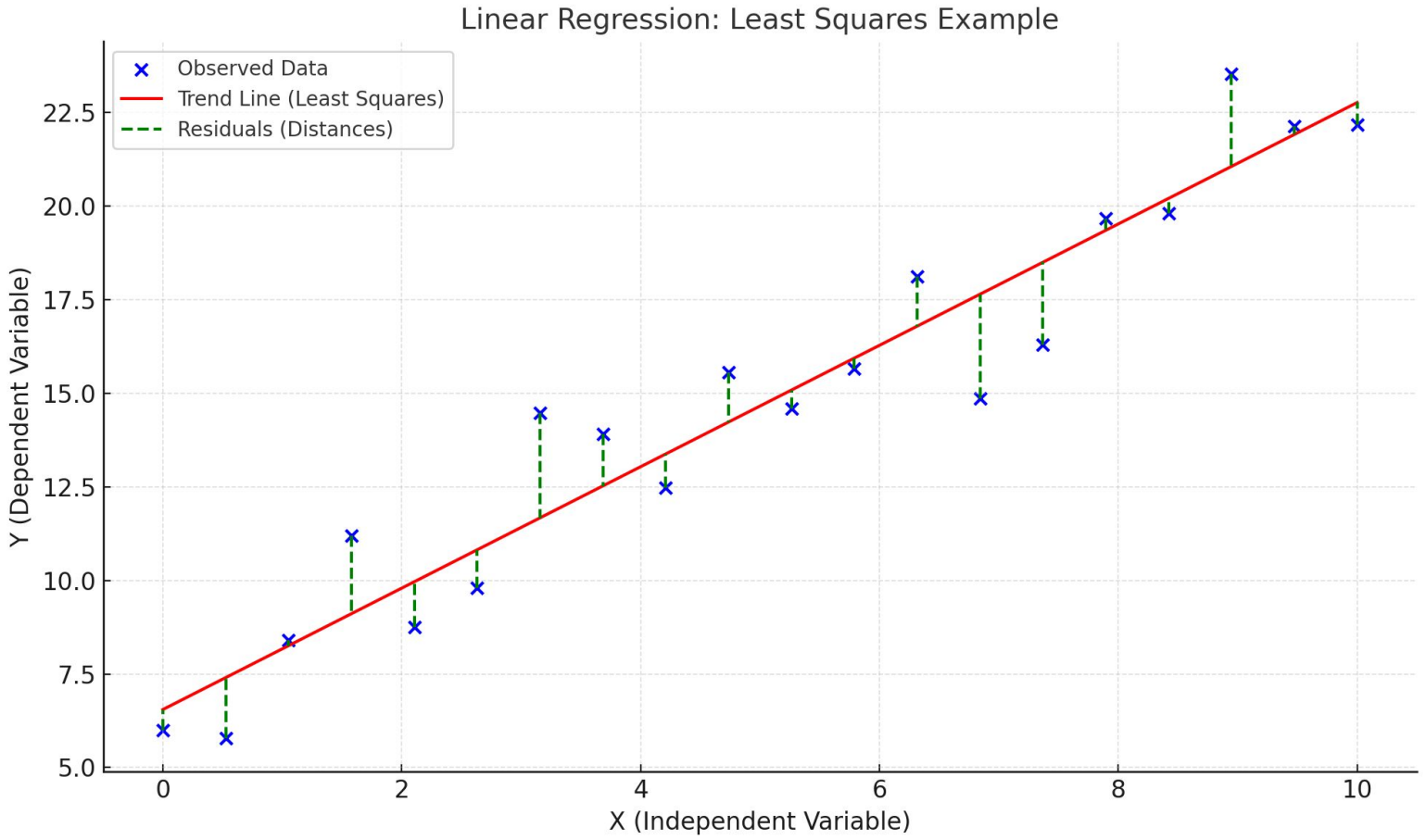
Cons

- Assumes linearity
- Sensitive to outliers
- Limited to one predictor
- Risk of underfitting

Key Features:

- **Single Independent Variable:** Focuses on the impact of one predictor on the target variable.
- **Linear Relationship:** Assumes a constant, linear relationship between the predictor and the outcome.

The inner workings of a Linear model



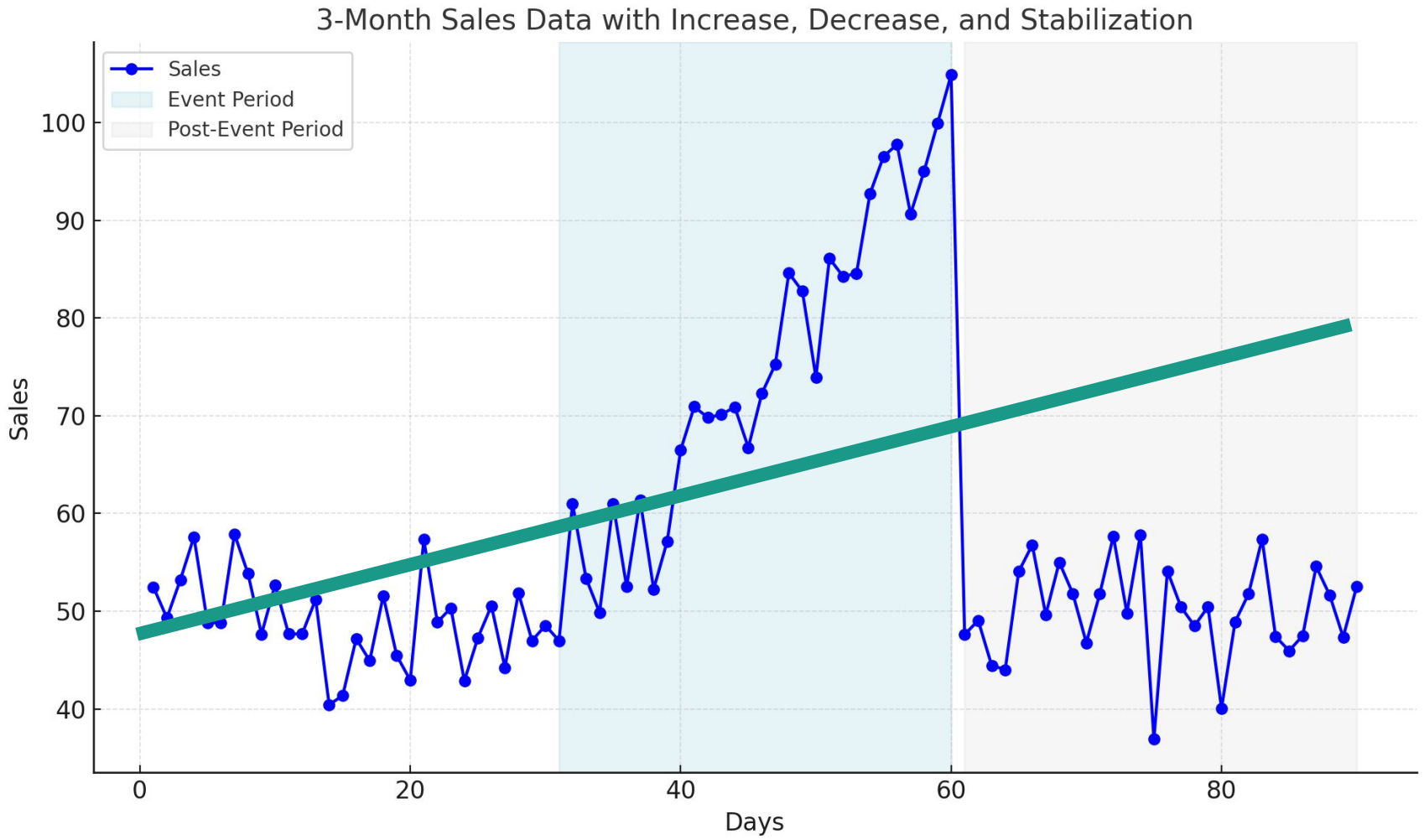
1. Observed Data (Blue Points):
2. Trend Line (Red Line):
3. Residuals (Distances) (Green Dashed Lines):
    - The vertical green lines show the differences between the observed values (yyy) and the predicted values (y^\hat{y}y^) on the trend line.
    - These distances are squared and summed to determine the least squares criterion, which is minimized to find the optimal line.

Exercise

1. Let’s see some real life examples
2. Complete the following tasks

5. Implement linear regression with appropriate evaluation metrics.

But linear models do not always work



Do you think this is a good model? Why or why not?

Contents

Why are more advanced modelling techniques neeeded?

Imagine you are tasked with forecasting sales using historical data, including marketing spend, customer traffic, and seasonality.

If you used a simple mode like linear regression, it would assume a linear relationship between marketing spend and sales, meaning increased spend always results in proportional sales growth and would fail to capture non-linearity, interactions and seasonality

As a result, linear regression might produce inaccurate forecasts for certain periods, especially when there are sharp increases or decreases due to these factors.

The solution?

> XGBoost can model non-linear relationships and feature interactions. It accounts for diminishing returns on marketing, handles seasonality, and improves accuracy by combining multiple factors in a complex way.

XGBOOST: A Powerful Forecasting Method

XGBoost (Extreme Gradient Boosting) is an advanced machine learning algorithm that uses an ensemble of decision trees to make predictions, combining the strengths of multiple weak models to form a strong predictor.

It is mostly used for applications such as:

Predictive Analytics: Forecasting outcomes like sales, stock prices, or customer behavior by learning complex relationships and interactions within the data.

Classification Problems: Identifying categories, such as detecting fraud, classifying customer segments, or diagnosing diseases based on patient data.

Ranking Problems: Predicting the rank of items, such as recommending the most relevant products to customers or ranking search engine results.

Pros

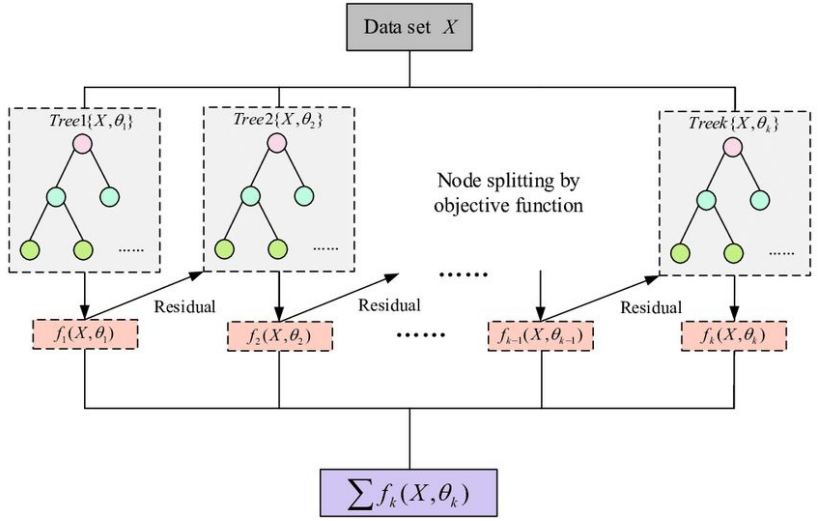
- High accuracy
- Handle missing data
- Feature importance
- Regularization
- Parallelization

Cons

- Complexity
- Overfitting risk
- Requires Feature Engineering

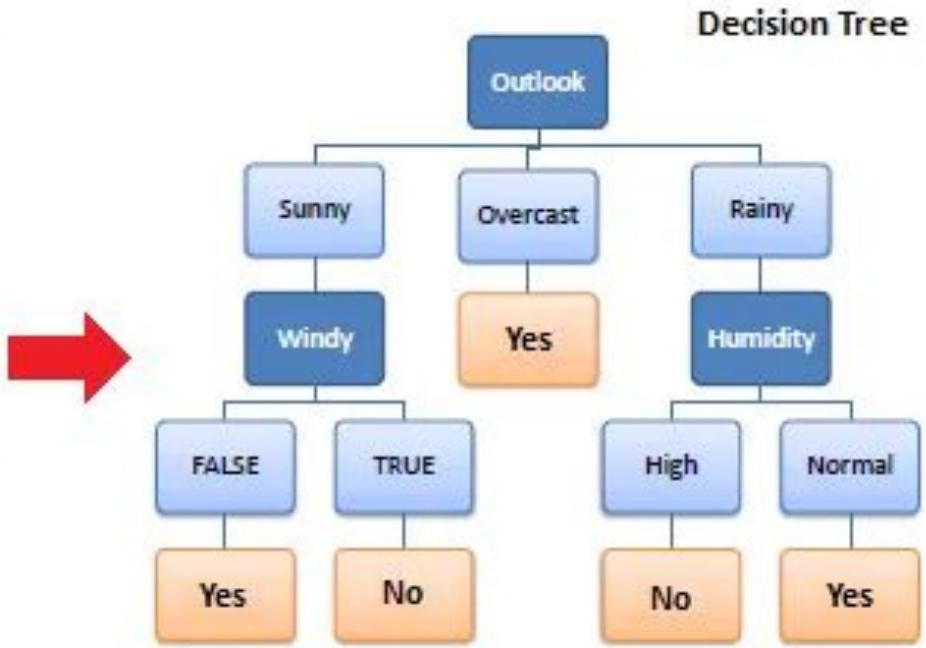
Key Features:

- **Ensemble Learning**
- **Gradient Boosting**
- **Tree Pruning**
- **Cross-validation**

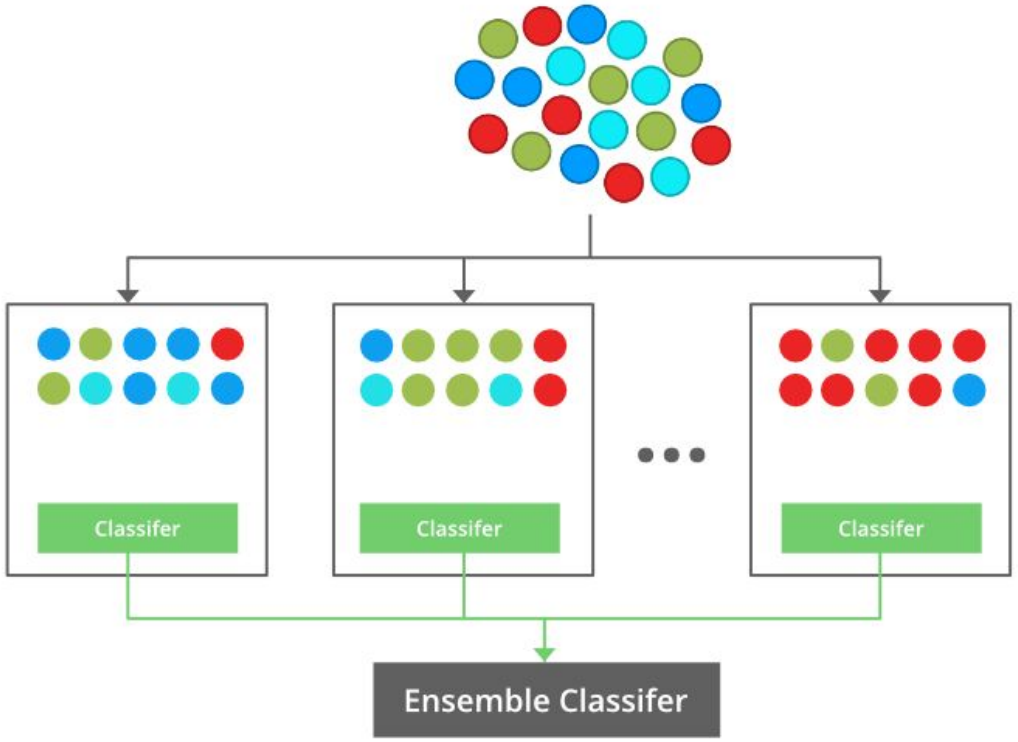


The building blocks of XGBoost is based on decision trees

Predictors				Target
Outlook	Temp	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No



We can augment decision trees - Bagging



Original Data

Bootstrapping

Aggregating

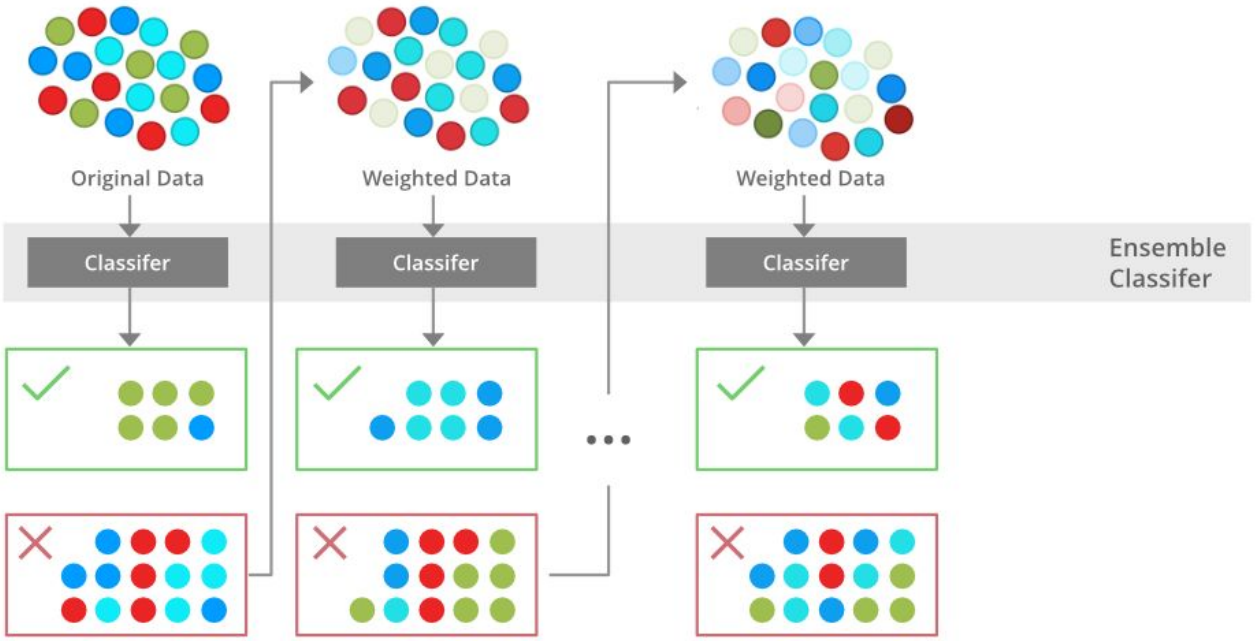
Bagging

A Bagging classifier is an ensemble meta-estimator that fits base classifiers each on random subsets of the original dataset and then aggregate their individual predictions to form a final prediction.

Such a meta-estimator can typically be used as a way to ***reduce the variance of a black-box estimator*** (e.g., a decision tree), by introducing randomization into its construction procedure and then making an ensemble out of it.

Bagging reduces overfitting (variance) by averaging or voting, however, this leads to an increase in bias, which is compensated by the reduction in variance though

We can augment even further - Boosting



Boosting is an ensemble modelling, technique that attempts to build a strong classifier from the number of weak classifiers.

It is done by building a model by using weak models in series.

1. Firstly, a model is built from the training data.
2. Then the second model is built which tries to correct the errors present in the first model.
3. This procedure is continued and models are added until either the complete training data set is predicted correctly or the maximum number of models are added.

Derived from boosting, we have the final method - ***Gradient Boosting*** *is a popular boosting algorithm. In gradient boosting, each predictor corrects its predecessor's error. In contrast to regular boosting, the weights of the training instances are not tweaked, instead, each predictor is trained using the residual errors of predecessor as labels.*

Hyperparameter Tuning

If we think of our machine learning model as a car engine, **hyperparameters** are like the knobs and switches used to control its performance. So, hyperparameter tuning is the process of **optimizing** these settings to improve the model's results. You may tweak parameters like learning rate or the depth of a decision tree to get the best fit for your data.

Why Tune Hyperparameters?

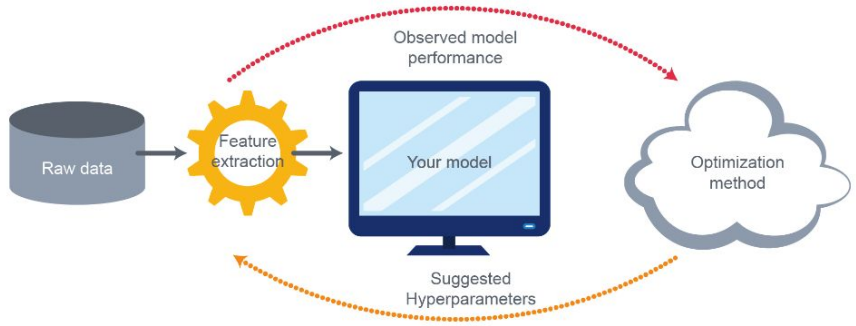
- **Improves Model Accuracy:** Ensures the model generalizes well to unseen data.
- **Prevents Overfitting/Underfitting:** Finds the right balance between model complexity and training data.
- **Optimizes Training Time:** Helps reduce unnecessary computation by selecting the best parameters.

Hyperparameter Tuning Methods:

- **Grid Search:** Exhaustive search over a range of hyperparameter values.
- **Random Search:** Randomly samples hyperparameter combinations to find the best results.
- **Bayesian Optimization:** Efficient search using probabilistic models to find optimal hyperparameters.

Key Hyperparameters to Tune:

- **Learning rate**
- **Number of estimators**
- **Max depth**
- **Subsample**



Hyperparameter Tuning

Some types of hyperparameters:

1. **General Parameters** — Guide the overall functioning of the model and relate to the booster being used.
2. **Booster Parameters** — Depends on the choice of the booster.
3. **Learning Task Parameters** — are used to define the optimization objective, i.e. the metric to be calculated at each step. They are also used to specify the learning task and the corresponding learning objective.

General approach:

1. Try parameters iteratively
2. Evaluate model performance at each iteration
3. Choose optimal parameters - *least runtime with best performance*

What to tune?
How much to tune?

It is a trial and error process.

You get better with experience.

```
param_grid = {
    'max_depth': [3, 4, 5],
    'learning_rate': [0.01, 0.1, 0.2],
    'n_estimators': [100, 200, 300]
}
```

Exercise

1. Let’s see some real life examples
2. Complete the following tasks

6. Implement XGBoost using hyperparameter tuning and evaluate using appropriate metrics.
7. Experiment tuning hyperparameters

Wrap Up

1. Background & fundamentals of Time Series
    a. What is time series analysis and where do we use it?
    b. Key concepts such as stationarity, trend, etc.
    c. Visualization
    d. Exercise - load time series and visualize
2. Exploratory analysis & feature engineering
    a. Autocorrelation
    b. Feature engineering with Pandas (lag, windows, rolling statistics, etc.)
    c. Exercise - feature engineering
3. Basis modelling - linear & ARIMA models
    a. Forecasting using simple linear models
    b. Forecasting using ARIMA models
    c. Exercise - implement linear model and ARIMA model and evaluate
4. Advanced modelling - XGBoost
    a. Why do we want more advanced modelling techniques?
    b. Forecasting with XGBoost
    c. Hyperparameter tuning
    d. Exercise - implement XGBoost

ARIMA (AutoRegressive Integrated Moving Average)

ARIMA is a forecasting method that models the relationship between past observations and future values by incorporating autoregression, differencing to achieve stationarity, and moving averages.

It is mostly used for applications such as:

Time Series Forecasting: Predicting future values of stock prices, sales, or demand based on historical data and trends.

Economic Forecasting: Modeling economic indicators such as inflation, GDP, or interest rates to predict future economic conditions.

Weather Prediction: Forecasting temperature, rainfall, or other weather conditions based on past data patterns.

Pros

- Effective for Stationary Data
- Versatile
- Easy to implement

Cons

- Assumes linearity
- Requires stationary data
- Sensitive to hyperparameters
- Limited to univariate data

Key Features:

- **Autoregressive (AR) Component:** The AR part captures the relationship between an observation and its previous values (lags).
- **Integrated (I) Component:** The I part represents the differencing of the series to make it stationary (i.e., removing trends and seasonality).
- **Moving Average (MA) Component:** The MA part models the relationship between an observation and the residual errors from a moving average model applied to lagged observations.