A Little Book of Time Series with Python

Regi V. Mathew

(Praxis Tech School, Bangalore, India)

Preface

This booklet, *A Little Book of Time Series with Python*, grew out of my appreciation for the simplicity and brevity of *A Little Book of R for Time Series* (Avril Coghlan). I noticed that this slim volume was quite helpful for students as a quick reference for concept clarification, interview preparation etc.

Inspired by that spirit, this booklet adapts the same philosophy but in the context of Python. Over the past decade, Python has become the default language for data science, machine learning, and applied analytics. Yet the statistical foundations of time-series analysis remain as essential as ever. This booklet aims to help readers learn and apply these statistical principles using the Python ecosystem. While it mirrors the structure and teaching philosophy of the R counterpart, the examples, syntax, and workflow are fully rewritten and tailored for the tools used in today's Python-oriented analytics environment.

My hope is that this book serves as a compact, practical, and approachable guide for anyone beginning their journey into time series analysis.

Table of Contents

1.0 Introduction to Time Series

Time series data is a sequence of data points, typically indexed or listed in chronological order, collected at successive, often regular intervals over a period of time. The importance stems from the fact that most real-world phenomena are dynamic and time-dependent. It provides a narrative of past performance, revealing growth, decline, or stability over months, years, or even seconds. It allows for the study of autocorrelation, where current values are statistically dependent on past values—a feature absent in non-sequential data. It forms the essential input for building predictive models that can mimic real-world behavior.

1.1 Time Series Analysis Applications

Accurate Forecasting and Prediction:- This is arguably the most critical application of time series analysis. By modeling the existing trend and seasonality, analysts can forecast future values with a quantifiable level of confidence.

Business Planning: Forecasting demand, sales, revenue, and inventory needs allows organizations to optimize resource allocation and manage capacity proactively.

Financial Markets: Essential for predicting stock prices, market volatility, interest rates, and economic indicators like GDP.

Operations: Used in weather forecasting (meteorology) and monitoring IoT devices for predictive maintenance.

Understanding Underlying Causes & Patterns:- Analytics helps decompose the series to understand why variables behave as they do.

Identifying Seasonality: Revealing predictable peaks (e.g., retail sales spike during holidays) allows businesses to plan staffing, marketing, and procurement efficiently.

Trend Analysis: Determining if a variable is generally increasing, decreasing, or stable over the long term, reflecting fundamental changes (e.g., inflation, population growth).

Intervention Analysis: Assessing the impact of an external event (e.g., a policy change, a product launch, or a natural disaster) on the data series.

Data Cleaning and Noise Reduction: Time series methods are vital for making noisy data interpretable.

Smoothing: Techniques like Moving Averages or Exponential Smoothing filter out the high-frequency irregular component (noise), making the underlying trend and seasonality clearer and more reliable for analysis.

Anomaly Detection: By building a model of what "normal" time-dependent behavior looks like, analytics can flag significant outliers or anomalies instantly (e.g., sudden spikes in fraudulent transactions or equipment failure precursors).

Comparison and Evaluation: It allows for comparing time-dependent metrics across different periods or entities.

Benchmarking: Comparing current year performance against the same period last year (accounting for seasonality) gives a much truer measure of business health than a simple year-over-year comparison.

Model Validation: Time series models (like ARIMA or Prophet) provide diagnostic tools to ensure the chosen model accurately reflects the historical data structure before being used for forecasting.

1.2 Examples from Different Domains

Finance

- Daily stock prices (NIFTY index).
- Exchange rates (USD/INR over time).
- Credit card transactions per hour.

Example: forecasting stock volatility or portfolio risk.

Weather and Climate

- Daily maximum temperature of Bengaluru.
- Monthly rainfall totals in Kerala.
- Hourly wind speed from a weather station.

Example: predicting monsoon patterns or detecting climate change trends.

Sales and Business

- Weekly sales of an e-commerce product.
- Quarterly revenue of a company.
- Daily footfall in a retail store.

Example: demand forecasting for inventory management.

Healthcare

- ECG recordings (heartbeats per millisecond).
- Daily patient admissions in a hospital.
- Continuous glucose monitoring data.
- Example use: early detection of arrhythmia or disease outbreaks.

Internet of Things (IoT)

- Temperature readings from a smart home sensor every 10 seconds.
- Electricity consumption of an appliance per minute.
- Vibration data from a manufacturing machine.
- Example use: predictive maintenance to prevent equipment failure.

1.3 Dataset Types

Dataset types can be broadly categorized based on how observations are arranged across entities and time. The three major types are: cross-sectional, time series, and panel (longitudinal).

1.3.1 Cross-Sectional Data

Data collected at a single point in time (or over a very short period) across multiple entities (individuals, firms, countries, products, etc.). Each observation is independent of the others (no time ordering).

Examples:

- Household income survey of 10,000 families in India in 2025.
- Student test scores across schools in a district for the year 2024.
- Market share of companies in the smartphone industry in 2025 Q1.

1.3.2 Time Series Data

Data collected on a single entity (or variable) across multiple time periods, usually at equal intervals (daily, monthly, quarterly, annually, etc.). Observations are time-dependent, and order matters.

Examples:

- Daily closing price of TCS stock (2010–2025).
- Monthly rainfall in Mumbai from 2000–2025.
- Annual GDP of India from 1960–2025.
- Hourly electricity consumption in a city.

1.3.3 Panel Data (Longitudinal Data)

Data that tracks multiple entities (like people, firms, or countries) over multiple time periods. It combines both cross-sectional and time series dimensions.

Examples:

- Annual income of 1,000 households tracked from 2000–2025.
- Quarterly sales of 200 companies from 2015–2025.
- Health records of 500 patients measured monthly over 2 years.
- Country-level CO_2 emissions for 100 countries from 1990–2025.

The table below shows the datatypes for a chain store business.

First pane shows sales for Jan/2001 across different stores as an example of cross-section data. Second pane shows sales of one store (ST1042) across 20 months as an example of time series data. Note that sales for one month appear in this table. Third plane shows 5 months data for 4 stores as an example of cross section data.

Cross-section		
Store	**Month**	**Sales**
ST1035	Jan 01	126
ST1036	Jan 01	330
ST1037	Jan 01	106
ST1038	Jan 01	208
ST1039	Jan 01	315
ST1040	Jan 01	270
ST1041	Jan 01	**288**
ST1042	Jan 01	**326**
ST1043	Jan 01	**340**
ST1044	Jan 01	**178**
ST1045	Jan 01	143
ST1046	Jan 01	177
ST1047	Jan 01	163
ST1048	Jan 01	197
ST1049	Jan 01	339
ST1050	Jan 01	324
ST1051	Jan 01	111
ST1052	Jan 01	245
ST1053	Jan 01	108
ST1054	Jan 01	108

Time Series		
Store	**Month**	**Sales**
ST1042	Jan 01	**326**
ST1042	Feb 01	240
ST1042	Mar 01	255
ST1042	Apr 01	154
ST1042	May 01	193
ST1042	Jun 01	180
ST1042	Jul 01	223
ST1042	Aug 01	303
ST1042	Sep 01	186
ST1042	Oct 01	123
ST1042	Nov 01	358
ST1042	Dec 01	273
ST1042	Jan 02	247
ST1042	Feb 02	212
ST1042	Mar 02	348
ST1042	Apr 02	219
ST1042	May 02	230
ST1042	Jun 02	359
ST1042	Jul 02	150
ST1042	Aug 02	151

Panel		
Store	**Month**	**Sales**
ST1041	Jan 01	**288**
ST1041	Feb 01	330
ST1041	Mar 01	106
ST1041	Apr 01	208
ST1041	May 01	315
ST1042	Jan 01	**326**
ST1042	Feb 01	288
ST1042	Mar 01	326
ST1042	Apr 01	340
ST1042	May 01	178
ST1043	Jan 01	**340**
ST1043	Feb 01	177
ST1043	Mar 01	163
ST1043	Apr 01	197
ST1043	May 01	339
ST1044	Jan 01	**178**
ST1044	Feb 01	111
ST1044	Mar 01	245
ST1044	Apr 01	108
ST1044	May 01	108

1.4 Relevance of Classical Statistical Models in the Age of ML

The rise of advanced Machine Learning (ML) and Deep Learning (DL) has sparked debate about the relevance of traditional statistical approaches in the current era. Interestingly, statistical methods have enduring strengths that keep them relevant, even in the era of advanced ML and DL.

Interpretability: Statistical models are highly interpretable. The parameters (*p, d, q*) directly correspond to the data's autoregressive structure, differencing needs, and error dependencies. This transparency is valuable in domains like economics or finance, where stakeholders need to understand the model's logic.

For example, in forecasting GDP growth, policymakers may prefer Statistics because they can trace how lagged values and trends contribute to predictions.

Effectiveness for Small Datasets: Statistical models perform well with limited data, as it relies on parametric assumptions rather than large training sets. In contrast, DL models like LSTMs require substantial data to generalize effectively.

Example: For a small retail business with only a few years of sales data, Statistical can produce reliable forecasts without overfitting.

Robustness for Linear and Stationary Patterns: Statistical models excels at capturing linear relationships and short-term dependencies in stationary or near-stationary time series. Many real-world time series, such as monthly sales or temperature data, exhibit patterns that Statistics can model effectively.

Example: SARIMA is particularly adept at handling seasonality, common in retail, energy, or climate data.

Computational Efficiency: Statistical model is computationally lightweight compared to DL models, which require significant resources for training and deployment. This makes Statistical models practical for real-time applications for resource-constrained environments.

Example: In embedded systems for IoT (devices monitoring sensor data), Statistical models can run efficiently with minimal computational overhead.

Theoretical Rigor: Statistics is grounded in statistical theory, with well-established methods for model selection (e.g., AIC, BIC), diagnostics (e.g., residual analysis), and uncertainty quantification (e.g., confidence intervals). This rigor appeals to researchers and practitioners who prioritize statistical validity.

Handling Stationarity: Statistical models explicitly address non-stationarity through differencing, making it suitable for time series with trends. Many ML/DL models assume stationarity implicitly or require reprocessing to achieve it, which can introduce complexity.

Comparing ARIMA and ML/DL Approaches

Aspect	Statistical Models	ML/DL
Interpretability	High: Parameters directly explain relationships.	Low to medium: Often a "black box," though techniques like SHAP improve interpretability.
Data Requirements	Small to moderate datasets suffice.	Large datasets needed for training to avoid overfitting.
Complexity Handling	Limited to linear patterns and short-term dependencies.	Excels at non-linear patterns and long-term dependencies.
Computational Cost	Low: Fast to train and deploy.	High: Requires GPUs/TPUs and extensive tuning.
Scalability	Poor for large-scale, multivariate series.	High: Can handle thousands of series with shared models.
Uncertainty Quantification	Strong: Provides confidence intervals via statistical theory.	Variable: Probabilistic DL models (e.g., DeepAR) offer uncertainty estimates.
Domain Suitability	Economics, finance, small-scale forecasting.	High-dimensional, non-linear, or multimodal data (e.g., IoT, finance, social media).

Given the strengths of these approaches, it is no surprise that there is space for both. Below are few such areas.

1.5 Leveraging Statistics and ML

Given the strengths and weaknesses of both these approaches, it makes sense to make a choice given the objective or combine both judiciously. Given below are few situations where such choices were made.

Finance: ARIMA is used for short-term forecasting of stock indices or interest rates, where linear trends dominate. However, DL models like LSTMs are preferred for high-frequency trading, where non-linear patterns and external factors (e.g., news sentiment) are critical.

Retail: SARIMA is effective for forecasting seasonal sales patterns (e.g., holiday sales). For large retailers with thousands of products, DL models like DeepAR scale better by learning cross-series patterns.

Energy: ARIMA is used for short-term load forecasting in small grids with stable patterns. Transformers or LSTMs are preferred for smart grids with complex, multivariate data from IoT sensors.

Epidemiology: During the COVID-19 pandemic, ARIMA was used for short-term forecasting of case counts in regions with limited data. ML/DL models were applied to integrate mobility data, social media trends, and other exogenous variables.

1.5.1 Future Outlook

The relevance of ARIMA and statistical methods will persist, but their role is evolving:

Hybrid Models: Combining ARIMA with ML/DL (e.g., ARIMA for trend, LSTM for residuals) is becoming common, leveraging the strengths of both.

Automated Machine Learning (AutoML): Tools like Prophet (developed by Meta) blend statistical principles with ML, offering ARIMA-like interpretability with greater flexibility.

Explainable AI: As ML/DL models improve in interpretability, they may encroach further on ARIMA's domain, but statistical methods will remain a fallback for their simplicity and reliability.

Domain-Specific Niches: ARIMA will continue to dominate in fields valuing interpretability and small data, while ML/DL will lead in big data and non-linear applications.

2.0 Building Blocks

The objective of this section is to introduce several foundational topics that will be referenced repeatedly as we progress through time series analysis. Addressing these concepts in advance ensures a smooth and uninterrupted discussion of the main subject. While some of the topics involve straightforward coding principles, others require familiarity with basic statistical ideas. We will not cover statistical fundamentals in depth here, so you may wish to review them as needed.

2.1 Correlation and Partial Correlation

2.1.1 Correlation

Correlation is a statistical measure that quantifies the linear relationship between two variables, indicating how closely they move together, ranging from -1 (perfect negative correlation) to +1 (perfect positive correlation), with 0 meaning no linear relationship. It is crucial to remember that **correlation does not imply causation**; a strong correlation simply suggests an association, but it doesn't prove that one variable causes the other, as other factors could be involved.

Correlation measures the **strength and direction** of a linear relationship between two variables. The most common measure is the **Pearson correlation coefficient**:

$$r = \frac{Cov(X,Y)}{\sigma_X \sigma_Y}$$

σ_X = standard deviation of X. Cov(X,Y) is the covariance between X and Y.

Values range from –1 to +1.

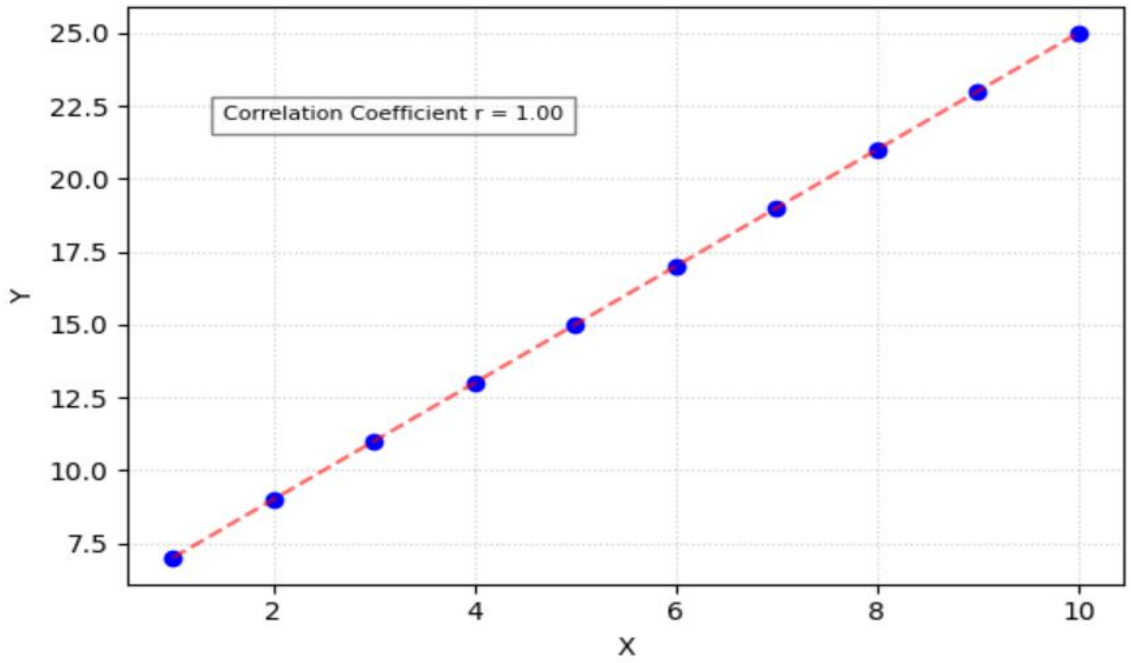
+1 = perfect positive linear relationship
–1 = perfect negative linear relationship
0 = no linear relationship

If you are familiar with simple regression, best way to understand correlation coefficient is to connect it with r-square (coefficient of determination i.e. proportion variation explained by linear regression). Correlation is the square root of r-square.

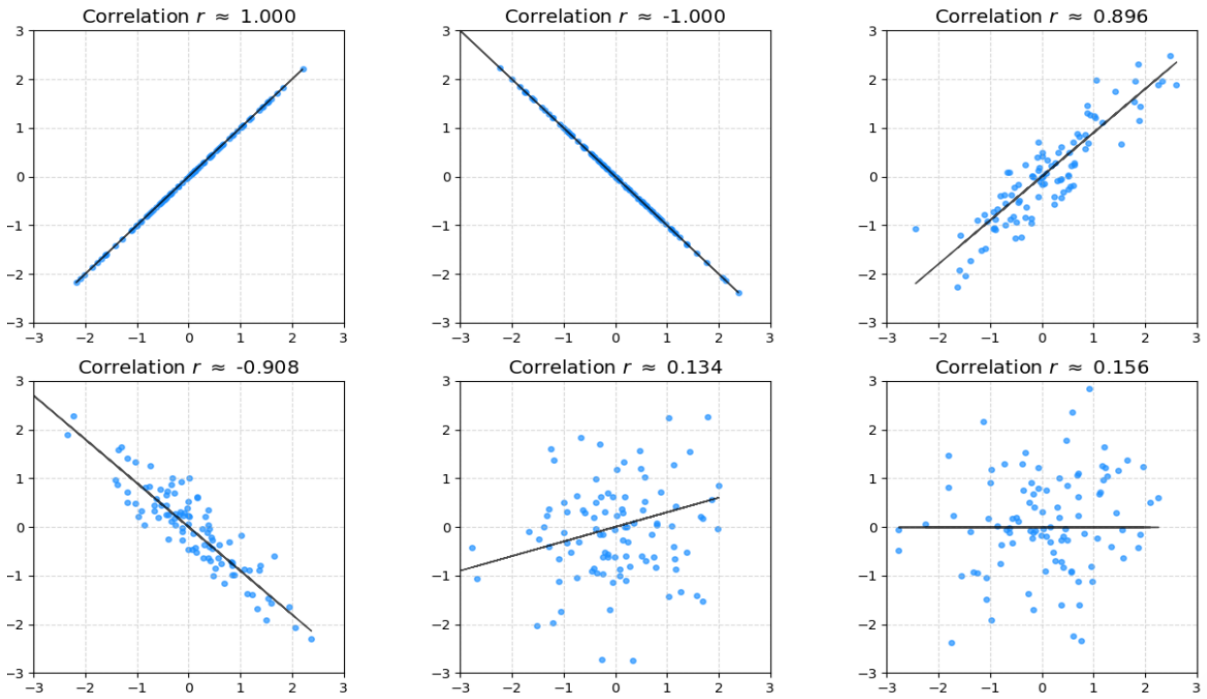
$$r^2 = 1 - \frac{\sum(Y-\hat{Y})^2}{\sum(Y-\bar{Y})^2} = 1\text{-}\frac{Variation\ not\ explained\ by\ regression}{Total\ Variation}$$

$$= Proportion\ explained\ by\ regression$$

Correlation coefficient is 1 when all data points are on a line. The logic is that when all data points are on a line Variation not explained by regression is zero (for all data points $Y=\hat{Y}$).



What does r=0.9 mean? To answer that question we must remember that $r = 0.9$ is the same as $r^2 = 0.81$. The latter tells us that 81 percent of the variation in Y is explained by the regression line. Hence, we see that r is the square root of r^2, and we are unable to interpret its meaning directly (never convert r value to a percent like 90% and try to interpret!!). Refer the plot to get an intuitive idea of various values of r.



Typical application of correlation is to indicate strength of relationship between two variables. Suppose we have data on ice cream sales and temperature over 30 summer days.

- As temperature increases, ice cream sales increase. If correlation = +0.85, that means high temperature and sales move strongly together.

But correlation can also be spurious:

- Ice cream sales and drowning accidents may also be correlated (both rise in summer). That doesn't mean ice cream causes drowning — a lurking variable (temperature) is driving both.

2.1.2 Partial Correlation

Partial correlation measures the strength of the linear relationship between two variables after removing the linear effect of other controlling variables. It helps isolate the true relationship between two variables by statistically controlling for the influence of third variables that might otherwise create a spurious correlation, thus providing a more accurate understanding of their unique connection.

For three variables X,Y,Z:

$$r_{XY.Z} = \frac{r_{XY} - r_{XZ}r_{YZ}}{\sqrt{(1 - r_{XZ}^2)(1 - r_{YZ}^2)}}$$

This formula shows how much of the correlation between X and Y remains after removing the effect of Z.

Ice cream sales and drowning accidents might show a correlation of +0.7. Once you control for temperature, the correlation between ice cream sales and drowning accidents may drop to near 0.

> Analysing data from 500 U.S. college students, researchers computed the partial correlation between SAT scores and freshman GPA, controlling for CLEP scores as a measure of prior knowledge. The raw correlation was $r = 0.52$, but partial correlation fell to near zero ($r \approx 0.05$), showing CLEP accounted for much of the apparent SAT-GPA link. This highlights biases in standardized testing (Brannick, M. T. Partial and Semipartial Correlation, USF faculty resource, echoed in studies like those in Educational Measurement: Issues and Practice).
>
> A cross-sectional study of 100 adults assessed the partial correlation between cardiorespiratory fitness and body weight, controlling for age. The zero-order correlation was negative ($r = -0.31$), but the partial correlation was weaker ($r = -0.25$), indicating age slightly inflated the fitness-weight link. This used real clinic data to refine health risk models (Laerd Statistics (2019)).
>
> A weight loss program finds a correlation between weekly exercise hours and weight loss. However, participants' diets (e.g., calorie intake) also affect weight loss. The raw correlation between exercise and weight loss was r=0.65. After controlling for diet, the partial correlation was r=0.3, showing exercise's direct effect is weaker when diet is consistent.

A consumer brand sees a correlation between monthly advertising budget and sales revenue. However, sales also vary by season (e.g., higher during holidays). The raw correlation might be r=0.8. After controlling for seasonality, the partial correlation is r=0.5, indicating advertising's direct impact is significant but partly driven by holiday peaks.

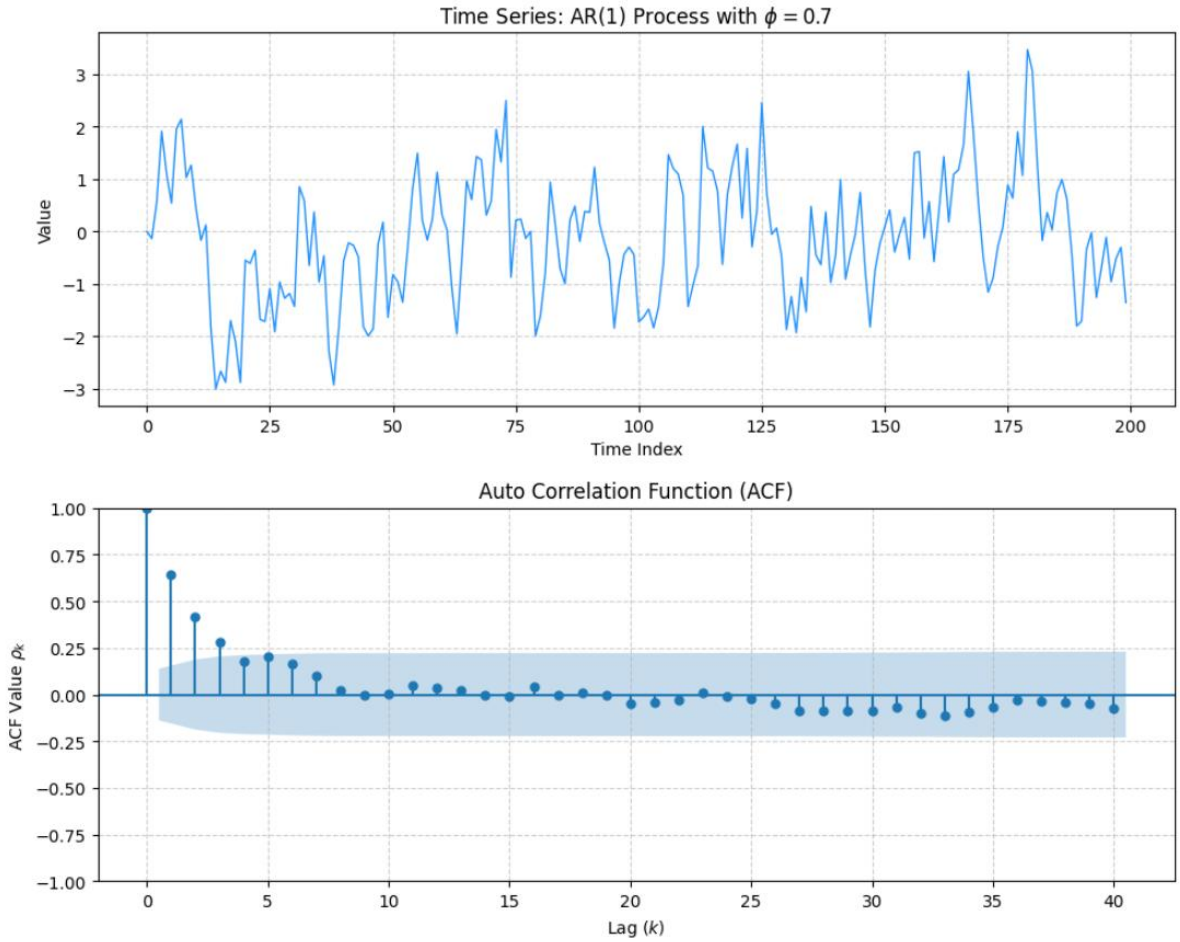
2.1.3 Autocorrelation Function

An autocorrelation function (ACF) measures the correlation between a time series and a delayed version of itself (lag). It helps in understanding the dependency structure of data over time, revealing patterns, periodicity, and trends. The ACF is typically plotted on a graph with lags on the x-axis and correlation coefficients on the y-axis, showing how correlation changes as the time separation (lag) increases.

The ACF calculates the correlation coefficient between a time series and its lagged versions. For example, the ACF at lag 1 measures the correlation between the series at time t and the series at time t-1. The table below shows the data and it's lagged values for clarity. The first correlation coefficient is the correlation between data (X_t) and itself (lag 0). Of course, it will be equal to 1. Second correlation coefficient is the correlation between the data and its lagged value by one period (X_{t-1}). The process continues till all correlations are examined with lagged values till maximum lag.

X(t)	X(t-1)	X(t-2)	X(t-3)	X(t-4)	X(t-5)
0.0000					
-0.1383	0.0000				
0.5509	-0.1383	0.0000			
1.9087	0.5509	-0.1383	0.0000		
1.1019	1.9087	0.5509	-0.1383	0.0000	
0.5372	1.1019	1.9087	0.5509	-0.1383	0.0000
1.9553	0.5372	1.1019	1.9087	0.5509	-0.1383
2.1361	1.9553	0.5372	1.1019	1.9087	0.5509
1.0258	2.1361	1.9553	0.5372	1.1019	1.9087
.	.	.	.	.	.
.	.	.	.	.	.
-0.1122	-0.7107	-1.2627	-0.0243	-0.3405	-1.7099
-0.9624	-0.1122	-0.7107	-1.2627	-0.0243	-0.3405
-0.5200	-0.9624	-0.1122	-0.7107	-1.2627	-0.0243
-0.3058	-0.5200	-0.9624	-0.1122	-0.7107	-1.2627
-1.3570	-0.3058	-0.5200	-0.9624	-0.1122	-0.7107

The results are visualized as an ACF plot, where the x-axis represents the lag (time delay) and the y-axis shows the correlation coefficient for each lag



It's a fundamental tool for analysing time series data and signal processing to identify patterns and dependencies.

Stationarity: By examining how the ACF decays over time, one can identify properties of the time series, such as whether it is stationary (statistical properties remain constant).

Periodicity: If a time series is periodic, the ACF plot will show large values at multiples of the period, indicating repeating patterns.

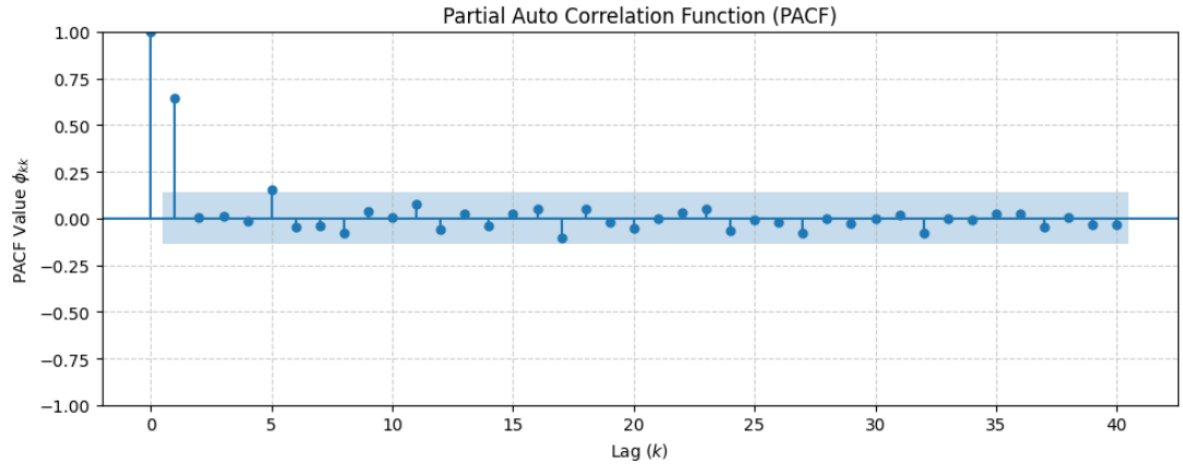
Trends: A slowly decaying ACF can indicate a trend in the data.

Noise and Randomness: A rapidly decaying ACF suggests that data points are less correlated with distant points, which can be a sign of a less complex or more random process.

2.1.4 Partial Autocorrelation Function

The partial autocorrelation function (PACF) measures the direct correlation between a time series and its lagged values, *after* removing the linear effects of all shorter lags. Unlike the autocorrelation function (ACF), which includes indirect effects from intermediate lags, the PACF isolates the direct relationship. It is a key tool in time series analysis, particularly

for identifying the order of an AR(p) process and for selecting appropriate ARIMA (details of ARIMA will be covered in the next section) models.



Note that ACF and PACF values is same for lags 0 and 1 (compared to correlation) as there are no intervening lags. However, partial correlation for lag 2 is different as the effect of lag 1 is removed. In other words, partial correlation between (X_t and X_{t-2}) is correlation between (X_t and X_{t-2}) after controlling for X_{t-1}.

$$r_{X_t X_{t-2}.X_{t-1}} = \frac{r_{X_t.X_{t-2}} - r_{X_t.X_{t-1}} r_{X_{t-2}.X_{t-1}}}{\sqrt{\left(1 - r^2_{X_t X_{t-1}}\right)\left(1 - r^2_{X_{t-2} X_{t-1}}\right)}}$$

2.2 QQ Plot

A QQ plot (Quantile-Quantile plot) is a graphical tool that compares the quantiles of two probability distributions to assess how well one distribution matches another, often used to check if a dataset follows a theoretical distribution like the normal distribution. If the data points on the QQ plot form a roughly straight line along a reference line, it indicates that the two distributions are similar

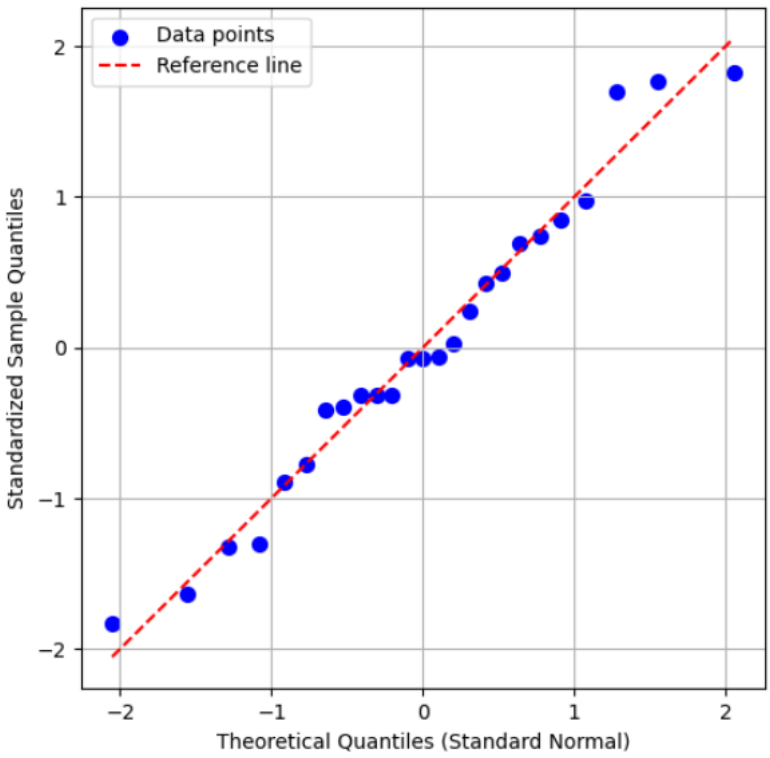
The table below outlines the logic for constructing a QQ plot to assess how closely the data adheres to a normal distribution.

Percentiles column shows the percentile position of each cell. Since there are 25 cells, percentile value of the first cell is 0.02 ((n-0.5)/25). Corresponding to this percentile value, z-score is -2.05 (theoretical z-score). This is theoretical z-score as it depends only on the position of the data. Or in other words, if this cell belongs to normal distribution, z-score of this data would be -2.05.

Actual z-score is calculated using the value of the cell ((x-mean)/std). If the data follows normal distribution, Actual z-score will be equal to theoretical z-score. This is tested using a plot connecting theoretical and actual scores.

By noting whether actual values coincides with theoretical, a judgement can be formed about whether the distribution is normal.

Rank	data	percentiles	Theoretical z-score	Actual z-score
1	80.87	0.02	-2.05	-1.83
2	82.75	0.06	-1.55	-1.63
3	85.75	0.1	-1.28	-1.32
4	85.88	0.14	-1.08	-1.31
5	89.87	0.18	-0.92	-0.89
6	90.92	0.22	-0.77	-0.78
7	94.38	0.26	-0.64	-0.42
8	94.56	0.3	-0.52	-0.40
9	95.31	0.34	-0.41	-0.32
10	95.34	0.38	-0.31	-0.32
11	95.37	0.42	-0.20	-0.31
12	97.66	0.46	-0.10	-0.07
13	97.66	0.5	0.00	-0.07
14	97.74	0.54	0.10	-0.07
15	98.62	0.58	0.20	0.03
16	100.68	0.62	0.31	0.24
17	102.42	0.66	0.41	0.42
18	103.14	0.7	0.52	0.50
19	104.97	0.74	0.64	0.69
20	105.43	0.78	0.77	0.74
21	106.48	0.82	0.92	0.85
22	107.67	0.86	1.08	0.97
23	114.66	0.9	1.28	1.70
24	115.23	0.94	1.55	1.76
25	115.79	0.98	2.05	1.82
Mean	**98.36**			
Std	**9.57**			



2.3 Model Validation in Time series Forecasting

In standard machine learning tasks (e.g., classification or regression using tabular data), basic method of model evaluation involves typical train-test method. This involves randomly sampling a portion of the dataset for training and the rest for testing. This is often done using functions like *train_test_split* from scikit-learn. The goal is to create independent and identically distributed (*i.i.d.*) subsets that represent the overall data distribution. In situations that involve taking critical decisions (tuning, selecting best modelling approach etc.), this approach would be extended to cross validation (CV).

However, when it comes to time series data, these approaches are not appropriate as observations are ordered chronologically and exhibit temporal dependencies (e.g., today's value influences tomorrow's). Applying a random train-test split disrupts this structure, leading to flawed model evaluation.

Temporal Splitting (No Shuffling): Here, time series data is split chronologically:

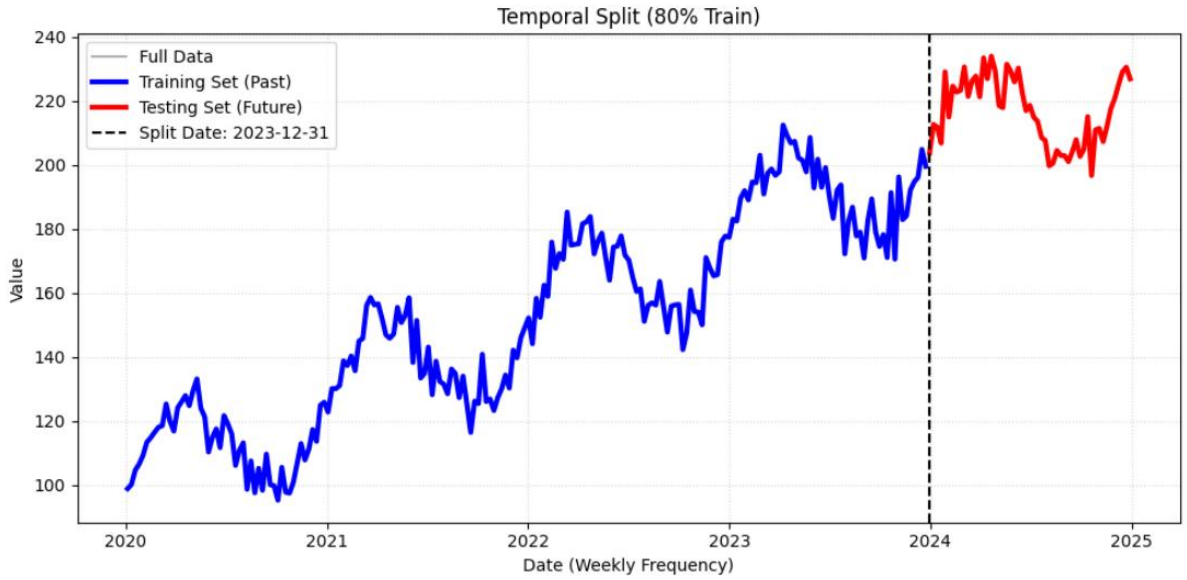
$$\text{Data} = [X_1, X_2, \ldots, X_T]$$

$$\text{Training Set} = [X_1, X_2, \ldots, X_{\text{Train End}}]$$

$$\text{Test Set} = [X_{\text{Train End}+1}, \ldots, X_T]$$

The training set consists of the earliest observations, and the test set consists of the latest observations. The size of the test set often depends on the required **forecast horizon** (e.g., if you need to forecast the next 12 months, the test set must be at least 12 months long).

```
def standard_split(data, split_ratio=0.8):
      # Determine the split index (e.g., 80% for training)
      train_size = int(len(data) * split_ratio)
      # Split the data chronologically
      train_set = data.iloc[:train_size]
      test_set = data.iloc[train_size:]

standard_split(ts_data, split_ratio=0.8)
```



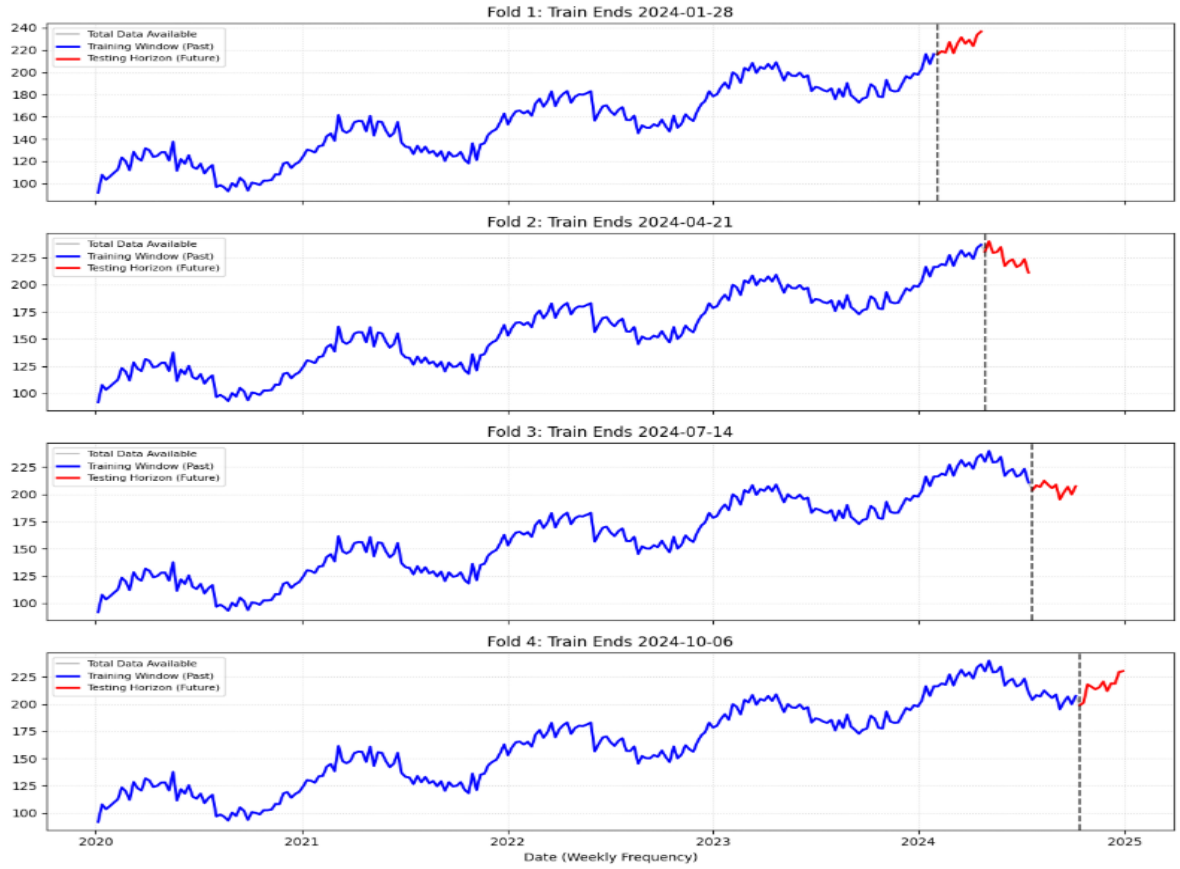
Time Series Cross-Validation: Standard K-Fold cross-validation (which involves shuffling and random splits) cannot be used. Instead, Rolling Origin or **Walk-Forward** Validation is employed to robustly evaluate model performance over multiple periods.

This method simulates how a model would be retrained over time:

Fold (Iteration)	Training Data (Origin)	Testing Data (Horizon)
1	$[X_1 \ldots X_{50}]$	$[X_{51}, X_{52}]$
2	$[X_1 \ldots X_{52}]$	$[X_{53}, X_{54}]$
3	$[X_1 \ldots X_{54}]$	$[X_{55}, X_{56}]$

..	(Training window grows)	(Testing window rolls forward)

```
def walk_forward_cv(data, n_splits=4, test_size=12):
    tscv = TimeSeriesSplit(n_splits=n_splits, test_size=test_size)
    for i, (train_index, test_index) in enumerate(tscv.split(data)):
        train_dates = data.index[train_index]
        test_dates = data.index[test_index]

walk_forward_cv(ts_data, n_splits=4, test_size=12)
```



This provides a more reliable estimate of model performance because it tests the model's accuracy on different, consecutive blocks of future data, accounting for potential shifts in the underlying time series dynamics.

2.4 Cost Function for Time Series Model Estimation

Time Series models are fundamentally regression problems. For such estimations, mean squared error (mse) function could be utilized.

$$mse = \frac{1}{n}\sum_{1}^{n}\left(y_t - \hat{y}_t(\theta)\right)^2$$

MSE penalizes large errors heavily (since errors are squared) and it is widely used for regression-based forecasting, and machine learning. θ is estimated by **minimizing *mse***.

However, many statistical time series models use Maximum Likelihood Estimation (MLE). MLE is preferred as it aligns with probabilistic framework, handles complex dependencies, supports efficient parameter estimation, and enables robust model diagnostics. MSE, while useful for simpler regression tasks or final forecast evaluation, does not capture the full structure of time series models.

If the model assumes errors are normally distributed with mean 0 and standard deviation of σ, ($\epsilon_t \sim N(0, \sigma^2)$), the log-likelihood is:

$$L(\theta) = -\frac{n}{2}\ln(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{1}^{n}(y_t - \hat{y}_t(\theta))^2$$

θ is estimated by maximizing likelihood L(θ) which is equivalent to **minimizing SSE (sum of squared errors)**.

2.5 Model Selection Metrics

Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC) are statistical measures used to select the best model among a set of candidate models. They are tools for achieving the balance between model fit (accuracy) and model complexity (simplicity). Both are rooted in the principle of parsimony; the simplest model that adequately explains the data is usually the best.

Akaike Information Criterion (AIC):- The AIC estimates the relative amount of information lost when a given model is used to represent the process that generated the data. It is a measure of the quality of a statistical model for a given set of data.

The general form of the AIC is:

$$AIC = -2ln(L)+2k$$

Where:

- **−2ln(L)** (Model Fit Term): Derived from the maximum value of the likelihood function (L) for the estimated model (refer likelihood function discussed above). **Lower** log-likelihood implies a worse fit, leading to a **higher** AIC.
- **2k** (Penalty Term): is the number of estimated parameters in the model (i.e., model complexity). More parameters lead to a **higher** AIC.

Bayesian Information Criterion (BIC):- The BIC is an adaptation of the AIC derived from a Bayesian perspective. It places a stronger penalty on models with more parameters, especially as the size of the dataset grows.

The general form of the BIC is:

$$BIC = -2ln(L)+kln(N)$$

Where:

- **$-2ln(L)$** (Model Fit Term): Same as in AIC, derived from the maximum likelihood
- **$kln(N)$** (Penalty Term): **k** is the number of estimated parameters. **N** is the number of observations/data points

Hannan-Quinn Information Criterion (HQIC):- The HQIC balances goodness-of-fit with model complexity. It serves a similar purpose to AIC and BIC but applies a different penalty for the number of parameters, making it a compromise between the two.

$$HQIC = -2ln(L)+2kln(ln(n))$$

Where:

- L: Likelihood of the model (how well it fits the data)
- k: Number of parameters in the model
- n: Number of observations in the time series

The term $2kln(ln(n))$ is the penalty for model complexity. Lower HQIC values indicate a better model, balancing fit and parsimony.

Suppose a data analyst is building an ARIMA(p,d,q) model to forecast stock prices. They need to determine the optimal values for the autoregressive order (p) and the moving average order (q). To do this, the analyst fits several candidate models, including ARIMA(1, 1, 0), ARIMA(2, 1, 1), ARIMA(3, 1, 0), and ARIMA(1, 1, 2) (The details of ARIMA terminology will be discussed in the next section).

Model	Parameters (k)	Log-Likelihood (ln(L))	AIC	BIC
ARIMA(1,1,0)	2	-100	204	**210**
ARIMA(2,1,1)	4	-95	206	220
ARIMA(3,1,0)	4	-96	208	222
ARIMA(1,1,2)	4	-94	204	218
ARIMA(1,1,1)	3	-96	**202**	213

Let's analyze the hypothetical results:

Model Fit: ARIMA(2,1,1) and ARIMA(1,1,2) have the best fit (highest ln(L)), but they are more complex (k=4).

AIC Selection: The model with the lowest AIC is ARIMA(1,1,1) (202). AIC favours this model as the best compromise for predicting future data.

BIC Selection: The model with the lowest BIC is ARIMA(1,1,0) (210). BIC heavily penalizes the extra parameter in ARIMA(1,1,1) and suggests the simplest model is the most likely true underlying process.

Broad conclusion is to use AIC for predictive tasks, smaller datasets, or when capturing complex patterns (e.g., seasonality in SARIMA) is critical. Use BIC for model parsimony, large datasets, or when identifying the true underlying process is the goal. In practice, compare both, validate with diagnostics (e.g., residual analysis), and consider time-series cross-validation for forecasting tasks to confirm the choice.

2.6 Model Evaluation Metrics

Root Mean Squared Error (RMSE):- In time series forecasting, RMSE serves as a key evaluation metric to quantify the accuracy of a model's predictions against actual observed values. It measures the average magnitude of the forecasting errors, emphasizing larger errors more heavily due to the squaring operation. This makes it particularly useful for assessing how well a model captures the underlying patterns in sequential data.

RMSE helps to compare different forecasting models (e.g., ARIMA vs. Prophet) on the same dataset. A lower RMSE indicates a better fit, meaning the model's predictions are closer to reality.

RMSE is calculated as follows:

$$RMSE = \sqrt{\frac{1}{n}\sum_{1}^{n}(y_t - \hat{y}_t)^2}$$

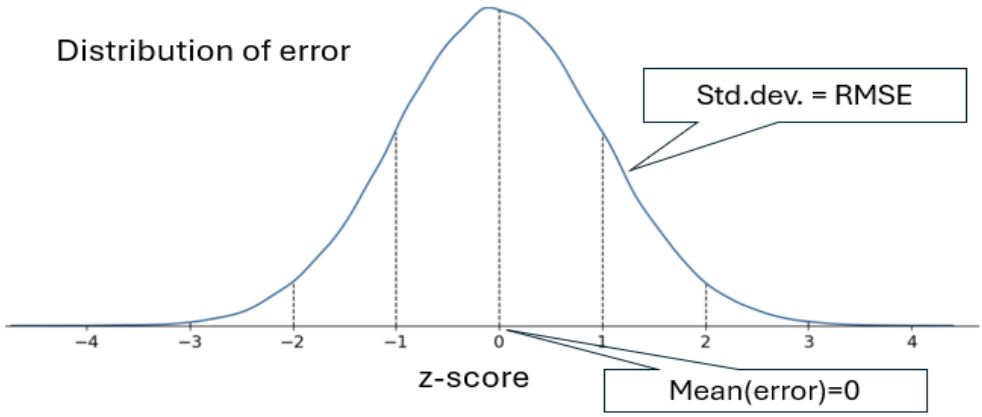
where y_t is the actual value, $\hat{y}_t$ the predicted value, and n is the number of observations.

The table illustrates the calculation of RMSE. The squared error is calculated first and square root of the mean of the squared error is the RMSE.

For an unbiased model, sum of error is zero and hence, RMSE can be considered as standard deviation of errors with mean 0. It is a reasonable assumption that the errors are normally distributed ($N(0, RMSE^2)$). Hence, applying the concepts of normal distribution, following conclusions can be drawn.

68% Prediction Interval: Roughly 68% of the future observations are expected to fall within the predicted value ±1×RMSE.

95% Prediction Interval: Roughly 95% of the future observations are expected to fall within the predicted value ±1.96×RMSE (often rounded to ±2×RMSE).



Mean Absolute Error (MAE): is a widely used metric and it measures the average magnitude of errors between predicted and actual values.

MAE is calculated as:-

$$MAE = \frac{1}{n}\sum_{1}^{n}|y_t - \hat{y}_t|$$

MAE is intuitive, expressed in the same units as the data, and robust to outliers since it does not square errors. It's ideal when you need a straightforward measure of average error magnitude, such as in models for stock price predictions.

Mean Absolute Percentage Error (MAPE):- measures the average percentage error relative to the actual values.

MAPE is calculated as:-

$$MAPE = \frac{100}{n}\sum_{1}^{n}\left|\frac{y_t - \hat{y}_t}{y_t}\right|$$

Time	y_t	$\hat{y}_t$	error	error²
31 01 2020	0.4967	-0.6305	1.1272	1.2705
29 02 2020	0.2094	0.3566	-0.1471	0.0217
31 03 2020	0.7943	0.1050	0.6893	0.4751
30 04 2020	2.0790	0.6172	1.4619	2.1371
31 05 2020	1.2212	1.7422	-0.5210	0.2715
30 06 2020	0.6207	0.9910	-0.3703	0.1371
31 07 2020	2.0137	0.4651	1.5486	2.3980
31 08 2020	2.1770	1.6850	0.4921	0.2421
30 09 2020	1.0544	1.8280	-0.7735	0.5984
31 10 2020	1.2807	0.8450	0.4357	0.1898
30 11 2020	0.4330	1.0431	-0.6100	0.3721
31 12 2020	-0.1626	0.3008	-0.4634	0.2148
31 01 2021	0.1281	-0.2208	0.3489	0.1217
28 02 2021	-1.8236	0.0338	-1.8574	3.4500
31 03 2021	-3.0014	-1.6752	-1.3262	1.7587
30 04 2021	-2.6633	-2.7067	0.0434	0.0019
31 05 2021	-2.8771	-2.4106	-0.4666	0.2177
30 06 2021	-1.6997	-2.5978	0.8981	0.8065
31 07 2021	-2.0978	-1.5668	-0.5310	0.2820
31 08 2021	-2.8808	-1.9154	-0.9654	0.9320
30 09 2021	-0.5509	-2.6010	2.0501	4.2030
31 10 2021	-0.6114	-2.3560	1.7446	3.0437
30 11 2021	-0.3605	-2.1415	1.7811	3.1721
31 12 2021	-1.6771	-1.9537	0.2766	0.0765
31 01 2022	-1.7183	-1.7892	0.0708	0.0050
			mse	1.0560
			RMSE	**1.0276**

MAPE expresses errors as percentages, making it scale-invariant and easy to interpret across datasets with different magnitudes. It's particularly useful in business contexts, like inventory management, where relative errors matter. However, MAPE has limitations: it becomes undefined or skewed when actual values are zero or very small.

2.7 Stationarity and Dickey-Fuller Test

The concept of stationarity is fundamental to classical time series analysis. A time series is considered stationary if its statistical properties, such as the mean, variance, and autocorrelation, are constant over time.

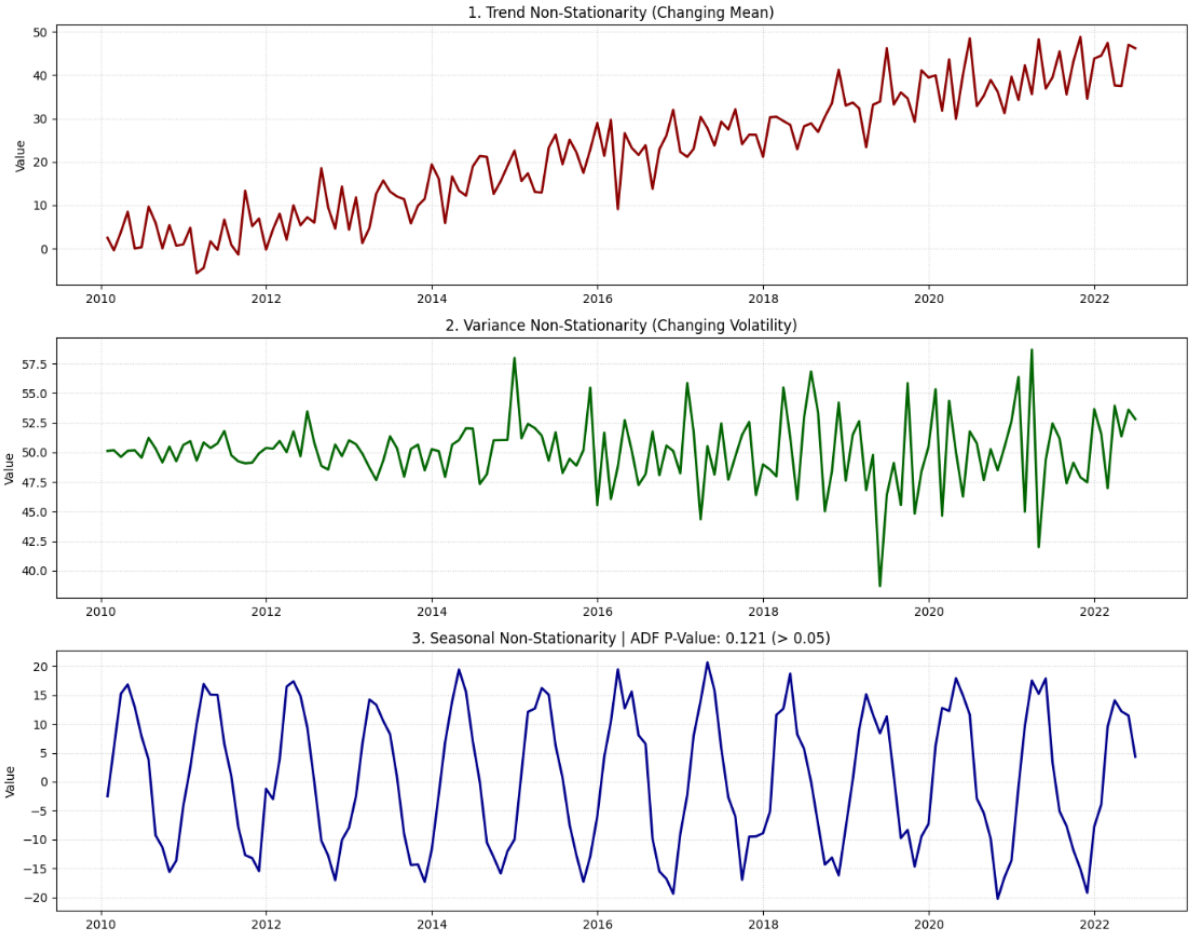
In simpler terms, if two different segments of a stationary time series are examined, they should look statistically the same, regardless of when the data was collected.

Stationarity is very important as even the most powerful time series models rely on the assumption that the pattern observed in the past will hold true in the future. If the properties of the series are constantly changing (non-stationary), any model built on historical data will quickly become unreliable for forecasting. Stationarity simplifies the mathematical modelling process, allowing usage of simpler, more robust statistical models.

Real-world data is often non-stationary. The most common forms of non-stationarity are:

Type	Description	How to Identify	How to Treat (Achieve Stationarity)
Trend Non-Stationarity	The mean changes over time (e.g., constantly increasing sales or population).	The series gradually moves up or down over time, indicating a persistent long-term direction.	Differencing (calculate the difference between consecutive observations: $Y_t = X_t - X_{t-1}$).
Variance Non-Stationarity	The variance (or volatility/spread) changes over time (e.g., small fluctuations become large, or vice versa).	The amplitude of the cycles or random noise visibly increases or decreases over the duration of the series.	Mathematical Transformations (e.g., Log or Square Root transformations) or using specialized GARCH/ARCH models (Generalized/Autoregressive Conditional Heteroskedasticity).
Seasonal Non-Stationarity	The pattern repeats at fixed intervals (e.g., yearly spikes in holiday sales).	Clear, predictable peaks and troughs repeating every k periods.	Seasonal Differencing (calculate the difference between observations k periods apart: $Y_t = X_t - X_{t-k}$).

The charts below show three types of non-stationarity increasing mean (trend), variance and seasonality.



2.7.1 Identifying non-stationarity (Dickey-Fuller Test)

The Dickey-Fuller Test (DF Test) and its more widely used successor, the Augmented Dickey-Fuller Test (ADF Test), are used to statistically determine if a time series is non-stationary (specifically, if it has a unit root or random walk behaviour).

Unit Root:- The most common form of non-stationarity is the presence of a trend, which often arises from a process called a Random Walk. A simple random walk is defined as:

$$Y_t = Y_{t-1} + \epsilon_t$$

where ϵ_t is white noise (random shock).

In this model, the current value (Y_t) is the previous value (Y_{t-1}) plus a random shock. If you start at $Y_0=0$, the value at time t is the sum of all past shocks:

$$Y_t = \sum_{i=1}^{t} \epsilon_i$$

Because of this structure, the variance of the series grows over time ($\text{Var}(Y_t)=t\sigma^2$), violating the constant variance requirement for stationarity.

Stationary AR(1) Process:- Now consider a stable autoregressive process of order 1 (AR(1)):

$$Y_t = \phi Y_{t-1} + \epsilon_t$$

If the coefficient ϕ is strictly less than 1 (i.e., $-1<\phi<1$), the series is stationary. Any shock ϵ_t eventually dissipates, and the series tends to revert to its mean. The critical question is: Is $\phi=1$?

> If $\phi<1$, the series is stationary.
> If $\phi=1$, the series is a random walk (non-stationary) and is said to have a Unit Root.

The Dickey-Fuller test is specifically designed to test this hypothesis. Instead of testing $\phi=1$, it is mathematically simpler to test an equivalent hypothesis by subtracting Y_{t-1} from both sides of the AR(1) equation:

$$\begin{aligned}\Delta Y_t &= Y_t - Y_{t-1} = \phi Y_{t-1} - Y_{t-1} + \epsilon_t \\ &= (\phi - 1) Y_{t-1} + \epsilon_t\end{aligned}$$

Let $\gamma=(\phi-1)$. The regression equation becomes:

$$\Delta Y_t = \gamma Y_{t-1} + \epsilon_t$$

The DF test then boils down to testing the coefficient γ.

> **Null Hypothesis (H_0):** $\gamma=0$ (This means $\phi=1$). The series is non-stationary (has a unit root).
> **Alternative Hypothesis (H_a):** $\gamma<0$ (This means $\phi<1$). The series is stationary.

If we find statistical evidence to reject the null hypothesis, we conclude the series is stationary.

2.7.2 Augmented Dickey-Fuller Test (ADF Test)

The basic DF test makes a crucial assumption: that the error term (ϵ_t) is white noise (uncorrelated). However, in real-world data, the error term might be serially correlated (autocorrelated).

The Augmented Dickey-Fuller (ADF) Test fixes this by adding lagged difference terms (ΔY_{t-1}, ΔY_{t-2},…) to the regression equation to "soak up" any residual autocorrelation:

$$\Delta Y_t = \alpha + \beta t + \gamma Y_{t-1} + \sum_{i=1}^{p} \delta_i \Delta Y_{t-i} + \varepsilon_t$$

- δ_i is the regression coefficient that measures the linear relationship between the current change in the series (ΔY_t) and the change in the series that occurred i periods ago (ΔY_{t-i}).
- *p* is the number of lagged terms added (determined by information criteria like AIC or BIC).

ADF test can be run in three different forms, depending on whether you include the α (drift) and βt (trend) terms. The choice of which form to use **depends on how the time series visually appears** (does it look like it drifts? does it have a clear trend?).

The ADF test calculates a t-statistic for the coefficient γ. Because the Y_{t-1} term is non-stationary under the null hypothesis, the calculated test statistic does not follow a standard t-distribution. Dickey and Fuller (1979) calculated specific critical values for this distribution.

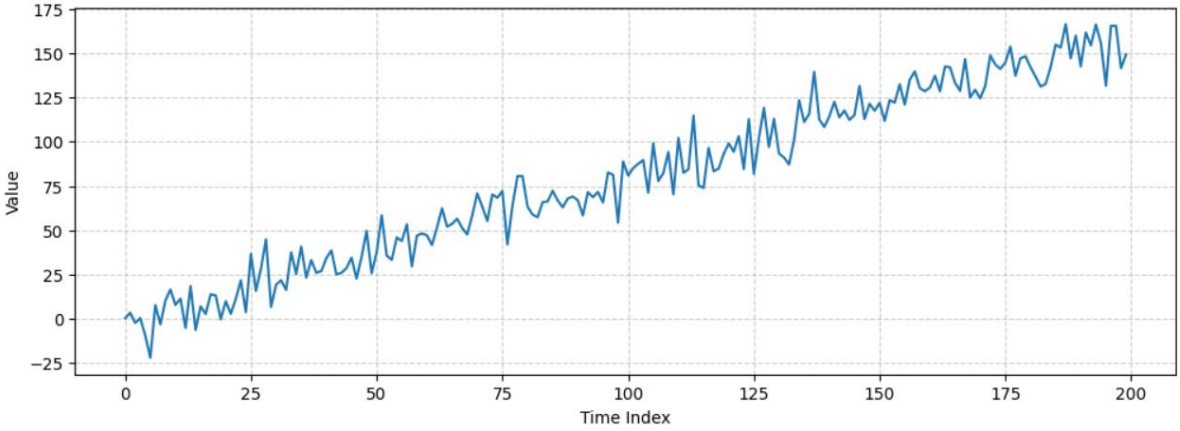
If the calculated ADF test statistic is less than (more negative than) the critical value (typically at the 5% level), we reject H_0 and conclude the series is stationary.

If the calculated ADF test statistic is greater than (less negative than) the critical value, we fail to reject H_0 and conclude the series is non-stationary (has a unit root).

The resulting p-value (which is what Python code typically report) simplifies this:

p-value < 0.05: Reject H_0. Series is stationary.
p-value ≥ 0.05: Fail to reject H_0. Series is non-stationary.

The code below illustrates Augmented Dickey-Fuller test using Statsmodels library.



```
from statsmodels.tsa.stattools import adfuller
result = adfuller(series_trend, regression='ct', autolag='AIC')
print('Results of ADF test: ADF: ',result[0],'p_value: ',
result[1], 'used_lag:  ',result[2])

Results of ADF test: ADF:  -13.120384446909593 p_value:
9.592505795544926e-21 used_lag:   0
```

Since p-value < 0.05, H_0 is rejected (Series is non-stationary).
In the adfuller() method, r*egression* options are chosen to reflect different forms as discussed above.

regression: {'c', 'ct', 'ctt', 'n'}

If the series has a clear slope (upward or downward trend): Start with *regression*='ct'. If you reject the null (it's stationary), stop. If you fail to reject, try differencing the data.

If the series fluctuates around a non-zero value, but has no clear slope: Use *regression*='c' ($\beta = 0$).

If the series has a zero mean and no trend (typically after differencing or detrending): Use *regression*='n' ($\alpha, \beta = 0$).

As discussed above, *autolag* options are chosen to decide which method is chosen to decide the lag length (0,1,2,,,,*maxlag*).

autolag: {'AIC', 'BIC', 't-stat', none}

If 'AIC' (default) or 'BIC', then the number of lags is chosen to minimize the corresponding information criterion.

't-stat' based choice of maxlag: Starts with maxlag and drops a lag until the t-statistic on the last lag length is significant using a 5%-sized test.

If none, then the number of included lags is set to maxlag.

2.7.3 Treating non-stationarity in time series data

Correcting non-stationarity is a crucial step in preparing time series data for classic forecasting models like ARIMA. The choice of correction method depends on the type of non-stationarity present (trend, seasonality, or changing variance).

Here are the primary methods for correcting non-stationarity.

Type of Non-Stationarity	Correction Method	Mathematical Operation	Rationale/Goal
Trend (Changing Mean)	Differencing	$Y_t = X_t - X_{t-1}$	Removes the linear or non-linear trend by calculating the change between consecutive observations.
Seasonality (Repeating Pattern)	Seasonal Differencing	$Y_t = X_t - X_{t-s}$ (where s is the season length, e.g., s=12 for monthly data)	Removes the seasonal component by subtracting the value from the same period in the previous cycle.
Changing Variance (Heteroskedasticity)	Log Transformation (or Box-Cox)	$Y_t = \log(X_t)$	Compresses the data, which stabilizes the variance, especially when variance is proportional to the level of the series.

2.8 Understanding Behaviour of Residual

Analysing the behavior of residuals is one of the most important steps in time series modeling. If the residuals do not behave like random noise (constant variance (homoscedasticity) and no autocorrelation), the model is misspecified and its forecasts, confidence intervals, and inference will be unreliable or invalid.

2.8.1 Randomness (Autocorrelation)

Ljung-Box (L-B) Q:- is a statistical test used to check for **autocorrelation** (serial correlation) in the **residuals** of a fitted model.

> **Null Hypothesis(H_0):-** The data (residuals) are independently distributed, and the autocorrelations for the specified lags are all zero. (The model is adequate).
>
> **Alternative Hypothesis(H_1):-** The data (residuals) are **not** independently distributed; at least one autocorrelation coefficient for the specified lags is not zero. (The model is inadequate).

The Ljung-Box statistic, denoted as Q (or Q*), is calculated using the sample autocorrelations ($\hat{\rho}$) of the residuals up to a specified number of lags (h):

$$Q = n(n+2)\sum_{k=1}^{h}\frac{\hat{\rho}_k^2}{n-k}$$

where:

n is the number of observations (residuals).
h is the number of lags being tested.
$\hat{\rho}_k$ is the sample autocorrelation of the residuals at lag k.

The primary consequences of significant autocorrelation in the residuals are:

- Inefficient Parameter Estimates: This means they have a larger variance than necessary, making them less precise.
- Biased Standard Errors: The estimated standard errors of the model parameters will be biased (usually underestimated). This is the most serious issue.
- Unreliable Hypothesis Tests: Because the standard errors are incorrect, the t-tests and p-values used to determine the statistical significance of the model parameters become unreliable and misleading. You might incorrectly conclude that a parameter is statistically significant when it is not, or vice versa.
- Suboptimal Forecasts: The forecasts generated by the model will be inefficient and may not capture all the available information in the time series, leading to less accurate predictions than a correctly specified model. This is because the correlation in the residuals implies there's a predictable pattern in the errors that the model has failed to capture.
- Model Misspecification: Persistent autocorrelation often indicates that the ARIMA model's orders (p,d,q) have been incorrectly specified, meaning the model is incomplete and has not fully captured the underlying time series structure.

The general approach to fixing residual autocorrelation is to refine the model specification to ensure the residuals are white noise.

Model Re-specification:- The most common and preferred fix is to adjust the AR and MA orders.

Incorporate Seasonality:- If the residual autocorrelation occurs at fixed, high lags (e.g., lag 4 for quarterly data, lag 12 for monthly data), it indicates seasonal patterns have been missed. The model should be extended to a Seasonal ARIMA (SARIMA) model, which has additional parameters (P,D,Q) for the seasonal component, such as ARIMA(p,d,q)$(P,D,Q)_m$, where m is the seasonal period.

Data Transformation:- Sometimes, the residual autocorrelation is a symptom of other violations, such as non-constant variance (heteroskedasticity).

If the variance of the residuals changes over time, apply a logarithmic or Box-Cox transformation to the original time series before modeling to stabilize the variance, which can sometimes also reduce residual autocorrelation.

2.8.2 Normality of Residuals

The Jarque-Bera (JB) Test is a statistical test that determines if sample data has the skewness and kurtosis matching a normal distribution. It is a goodness-of-fit test commonly used for checking the normality assumption of residuals in regression models.

Null Hypothesis (H_0):- The data is normally distributed (i.e., Skewness = 0 and Excess Kurtosis = 0).

Alternative Hypothesis (H_1):- The data is *not* normally distributed.

The test statistic, JB, is calculated based on the sample skewness (S) and sample kurtosis (K):

$$JB = \frac{n}{6}\left[S^2 + \frac{(K-3)^2}{4}\right]$$

where:

n is the sample size.
S is the sample skewness coefficient.
K is the sample kurtosis coefficient. (Note that K−3 is the excess kurtosis).

The main impacts of non-normal residuals are primarily on inference and the calculation of prediction intervals, rather than on the parameter estimates themselves (assuming a large enough sample size).

Inaccurate Prediction Intervals: This is the most significant impact. ARIMA models often assume a normal distribution for the innovations to construct Prediction Intervals (PIs). If the true distribution is non-normal (e.g., heavy-tailed like the Student's t-distribution), the calculated PIs will be incorrect.

Invalid Hypothesis Tests: The standard t-statistics used for testing the significance of the coefficients rely on the normality assumption. With non-normal residuals, these tests may be invalid or misleading.

Possible Fixes for Non-normality:

The fixes depend on the underlying cause of the non-normality, which can sometimes be a sign of a deeper problem with the model specification (though not always).

Check for Model Misspecification:- Non-normal residuals can sometimes be a symptom that the model has not fully captured the data's structure.

Check for Remaining Autocorrelation: Verify that the residuals still satisfy the white noise assumption (zero mean, constant variance, and no autocorrelation). Significant autocorrelation (often checked with a Ljung-Box test or residual ACF/PACF plots) means the model is misspecified, and fixing this will often improve normality.

Identify Outliers or Structural Breaks: Large, infrequent deviations (outliers, level shifts, temporary changes) can create heavy-tailed or skewed residuals.

Transform the Data:- A non-linear transformation of the original time series can sometimes stabilize the variance and improve the normality of the residuals.

2.8.3 Constant Variance

Heteroskedasticity(H) test is a diagnostic test to check the assumption that the residuals (errors) have a constant variance over time (a property called homoscedasticity).

Null Hypothesis (H_0):- The model is homoskedastic (constant error variance).

Alternative Hypothesis (H_1):- The model is heteroskedastic (non-constant error variance).

The test (White Test) proceeds in the following steps:

- Estimate the Original Model: Run the original model and obtain the squared residuals (ε^2). The squared residuals serve as a proxy for the error variance.
- Run the Auxiliary Regression: Regress the squared residuals on all the original explanatory variables, their squares, and their cross-products (interactions).
- Calculate the Test Statistic: The test statistic is calculated as the sample size (n) multiplied by the R-squared value (R^2) from the auxiliary regression:

$$W = n \cdot R_{aux}^2$$

Violating the Homoscedasticity Assumption primarily affects the efficiency of the parameter estimation and the accuracy of uncertainty measures. While the estimated model coefficients (AR and MA terms) are still consistent (they converge to the true values as the sample size increases), they are not efficient (they do not have the minimum possible variance). This means the estimates are not the "best" available.

Incorrect Standard Errors and t-statistics: The standard errors of the model coefficients will be biased (usually underestimated). This leads to:

Invalid Hypothesis Tests: t-statistics based on the incorrect standard errors can be misleading, potentially causing you to incorrectly conclude that a parameter is statistically significant when it is not (Type I error).

Inaccurate Prediction Intervals (PIs): Since the PIs are calculated using the estimated residual variance, the forecasts for periods with higher-than-average volatility will have PIs that are too narrow (underestimated risk), and for periods with lower volatility, the PIs may be too wide. This misrepresents the true forecast uncertainty.

Possible Fixes for Heteroscedasticity:

The solutions involve either correcting the data transformation or modifying the model to explicitly account for the time-varying variance.

Data Transformation If the variance of the original series appears to increase with the level of the series, a simple transformation can often stabilize the variance.

Log Transformation ($\ln(Y_t)$): This is the most common and effective transformation, especially for financial or economic data where volatility tends to be proportional to the series' magnitude.

Box-Cox Transformation: A more general approach that can help stabilize variance and improve normality simultaneously. It involves estimating an optimal transformation parameter.

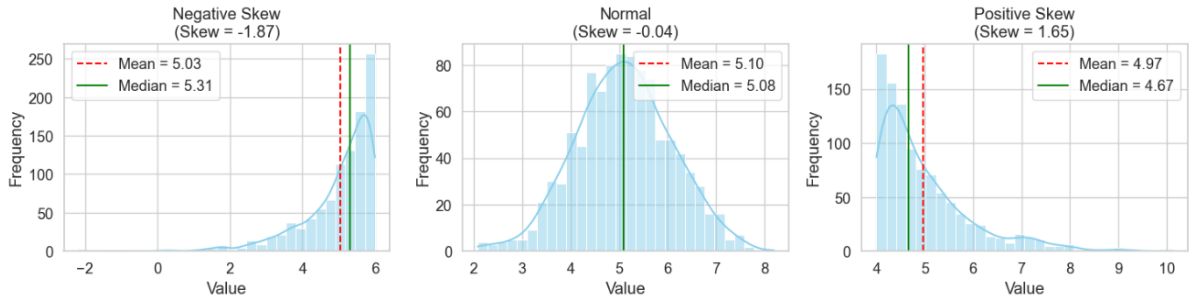
Explicitly Modeling Volatility: If simple transformations are insufficient, you must model the residual variance directly. This leads to combined ARIMA-type models.

ARIMA-GARCH/ARIMA-ARCH Models: The most robust solution is to combine the ARIMA model for the mean (level) of the series with a Generalized Autoregressive Conditional Heteroscedasticity (GARCH) model for the variance (volatility) of the residuals.

2.8.4 Skewness and Kurtosis

Skew and **Kurtosis** values for the **model residuals** (the error terms) are crucial for assessing the **normality assumption** of the residuals, which is a key diagnostic check for the model's validity and the reliability of its statistical inferences.

Skewness measures the **asymmetry** of the residual distribution around its mean. The figure below illustrates various levels of skewness.



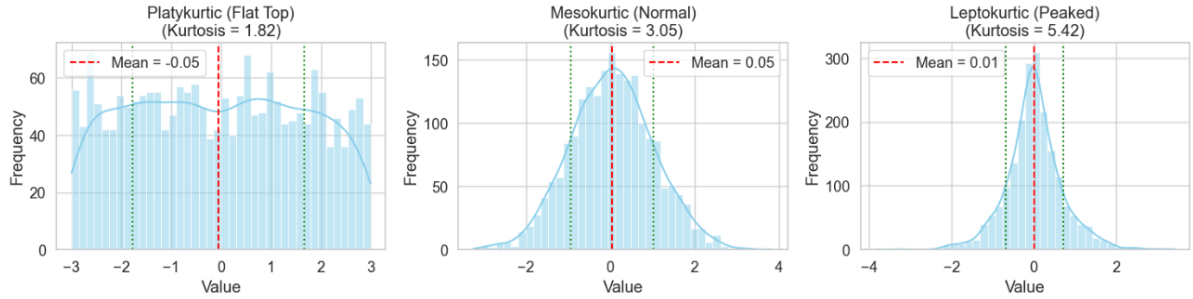
Value Range	Interpretation

Close to 0	Symmetric (ideal for a normal distribution)
Positive Skew (> 0)	Right-Skewed. The right tail is longer, meaning there are a few extreme positive errors (the model under-predicted a few large values)
Negative Skew (< 0)	Left-Skewed. The left tail is longer, meaning there are a few extreme negative errors (the model over-predicted a few large values)

Kurtosis measures the **"tailedness"** of the residual distribution—specifically, how thick or thin the tails are compared to a normal distribution.

In *statsmodels*, the reported Kurtosis is typically the Excess Kurtosis added to 3 (the value for a normal distribution). By convention, a normal distribution has a kurtosis of 3 (or an excess kurtosis of 0).

The figure below illustrates three types kurtosis viz. Platykurtic, Mesokurtic and Leptokurtic.



Value Range	Implication for Residuals
Close to 3 - Mesokurtic (Matches the normal distribution).	The peakedness and tail weight are consistent with the normality assumption.
Greater than 3 - Leptokurtic (Higher peak and fat tails).	The distribution has more frequent extreme outliers (very large positive or negative errors) than a normal distribution. This is common in financial time series.
Less than 3 - Platykurtic (Lower/flatter peak and thin tails).	The distribution has fewer extreme outliers than a normal distribution.

References

Laerd Statistics (2019). Partial Correlation in SPSS Statistics (tutorial based on real dataset from fitness research; aligns with studies like those in Journal of Applied Physiology).

Dickey, D.A.I. and Fuller, W.A. (1979) Distribution of the Estimators for Autoregressive Time Series with a Unit Root. Journal of the American Statistical Association, 74, 427-431.

3.0 Decomposition of Time Series

The main objective of decomposing a time series is to break down the observed data into its underlying, interpretable components so that each part can be understood, modeled, and forecasted separately (often more easily and accurately than the raw series). The objective of this exercise are to:-

Understand the structure of the data: Separate the series into meaningful components: Trend, Seasonality, Cycles, and Irregular/ Residual fluctuations. Makes it easier to see what is really driving the series (e.g., long-term growth, recurring seasonal patterns, business cycles).

Improve visual interpretation: Plotting the components side-by-side reveals patterns that are hidden in the raw data. Helps in presentations, reports, and exploratory data analysis.

Simplify modeling and forecasting: Each component can be modeled separately with the most appropriate method:

- Trend - polynomial or nonparametric regression
- Seasonality - seasonal dummies, Fourier terms, SARIMA
- Residuals - ARIMA, machine learning.

Often yields better forecasts than modeling the raw series directly (especially with strong seasonality).

Seasonal adjustment: Remove the seasonal component to obtain a seasonally adjusted series. This is essential in economics and official statistics (e.g., unemployment rates, GDP, retail sales) to compare underlying trends month-to-month or quarter-to-quarter.

Detect anomalies and outliers: After removing trend and seasonality, irregular spikes or drops become much more visible. Useful for fraud detection, sensor monitoring, demand forecasting, etc.

3.1 Decomposition of Time Series

Time series data is often a mixture of several components. **Decomposition** means separating a series into interpretable parts:

$$y_t = T_t + S_t + C_t + \varepsilon_t$$

where:

T_t: **Trend** (long-term progression)

S_t: **Seasonality** (systematic repeating patterns, e.g., monthly, quarterly)

C_t: **Cyclical** (longer-term economic or business cycles)

ε_t: **Irregular / Residual** (random noise)

Trend represents the slow-moving changes in a time series. It is responsible for making the series gradually increase or decrease over time.

Seasonality component represents the seasonal pattern in the series. The cycles occur repeatedly over a fixed period of time.

Cyclicality refers to irregular, long-term fluctuations in data, occurring over periods of at least two years and not at a fixed frequency, unlike the predictable patterns of seasonality. These cycles are often driven by economic factors like the business cycle, leading to periods of expansion and recession, and are characterized by a series of peaks and troughs that vary in duration and amplitude.

Residuals represent the behaviour that cannot be explained by the trend and seasonality components. They correspond to random errors, also termed white noise.

Following few examples in various fields of economics and business.

Economics & Finance

- GDP growth → long-term upward trend + business cycles + irregular shocks.
- Stock prices → separating trend from short-term volatility.

Retail & Sales Forecasting

- Sales data → strong seasonality (holidays, festivals) + long-term growth trend.
- Helps in inventory management and promotion planning.

Weather & Climate

- Temperature → annual seasonality + long-term climate trend.
- Decomposition used to detect climate change effects.

Energy & Utilities

- Electricity demand → daily and weekly cycles + upward trend in consumption.
- Helps in grid load balancing and pricing.

Healthcare & Epidemiology

- Disease incidence → seasonal cycles (flu in winter) + trend (long-term increase/decrease).
- Useful for resource allocation.

Operations & Quality Control

- Sensor/IoT data → decompose signals to detect abnormal machine behaviour.

Visualising Components of a Time Series

The classical method uses simple moving averages to estimate the trend, then calculates seasonality by averaging the detrended values for each seasonal period (e.g., each month).

- **Additive Model**

$$y_t = T_t + S_t + C_t + \varepsilon_t$$

Used when the magnitude of the seasonal fluctuations **does not** depend on the level of the time series (i.e., seasonality is constant).

- **Multiplicative Model**

$$y_t = T_t \times S_t \times C_t \times \varepsilon_t$$

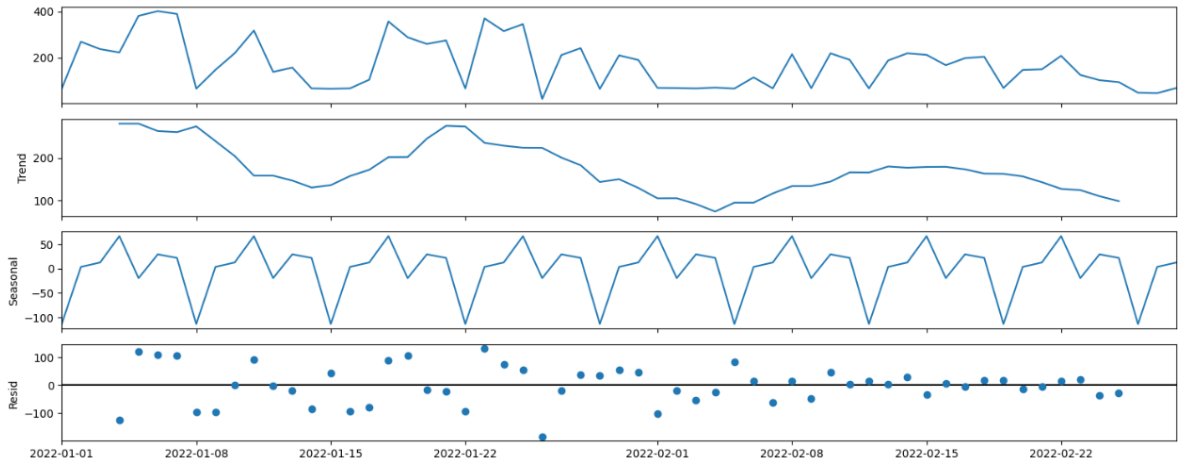
Used when the magnitude of the seasonal fluctuations **increases** as the time series trend increases (common in sales, economic data).

seasonal_decompose function from statsmodels.tsa.seasonal is used to decompose a time series into its seasonal, trend, and residual components. It allows specifying the type of seasonal component through the model parameter, which can be either 'additive' or 'multiplicative' as discussed above.

Illustration of decomposition below follows additive approach first. The data used was daily energy consumption of HVAC AHU Blower in KWh (https://www.kaggle.com/datasets/vitthalmadane/energy-consumption-time-series-dataset/data). This data is recorded with help of IoT device and it shows 59 days of data starting 01/Jan/2022.

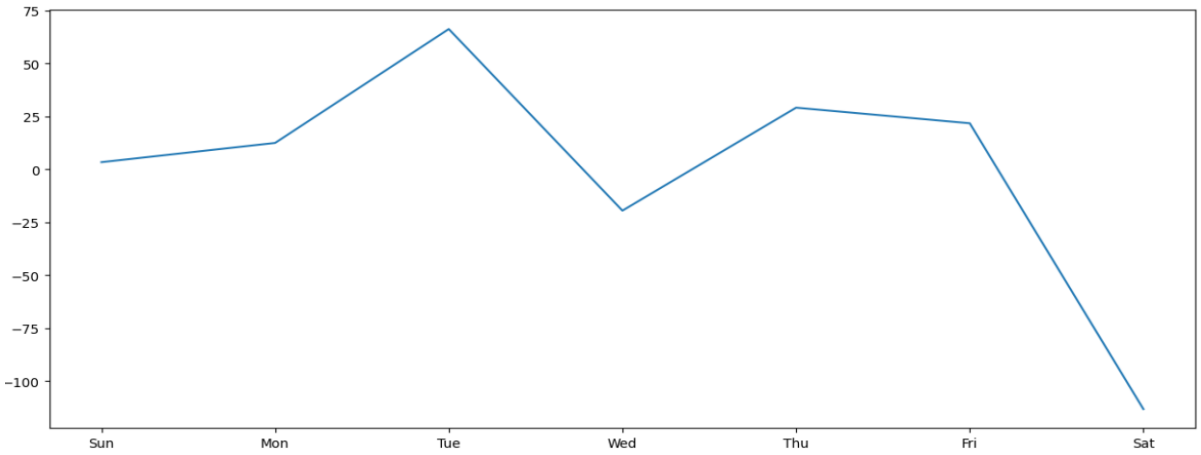
```
from statsmodels.tsa.seasonal import seasonal_decompose
decom = seasonal_decompose(df,model='additive', period=7)
decom_df = pd.DataFrame({'y':df['Consumption'], 'trend':decom.trend,
'seasonality':decom.seasonal, 'residual':decom.resid})
decom_df['day'] =decom_df.index.strftime('%a')
plt.rc("figure", figsize=(15,6))
decom.plot()
plt.show()
```

Decomposition plot is shown below. The top panel illustrates It is split into components to understand it better.



The trend component reveals a gradual decline in energy consumption over time, which may be attributable to changes in ambient temperature. The seasonal pattern exhibits a clear

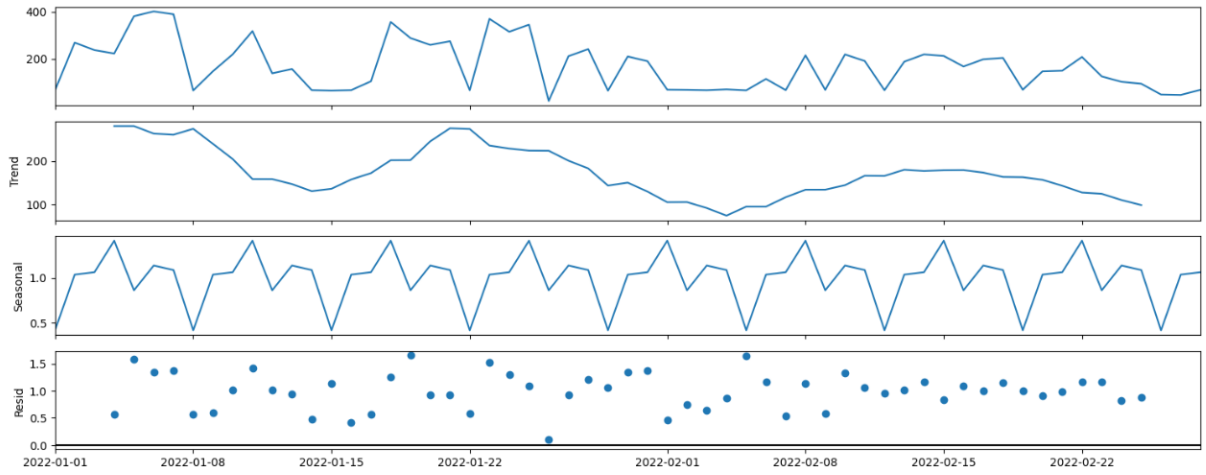
weekly cycle, with the lowest consumption occurring on Saturdays and the highest on Tuesdays (as illustrated in the weekly profile below).



3.2 Multiplicative Approach of Decomposition

Multiplicative Approach assumes the data as multiplication of components. The code is modified as below.

```
from statsmodels.tsa.seasonal import seasonal_decompose
decom = seasonal_decompose(df,model='multiplicative', period=7)
plt.rc("figure", figsize=(15,6))
decom.plot()
plt.show()
```



Note that the pattern of the components are quite similar to additive approach. Quantity of Trend is same while other components are dissimilar. This is because other components are created by multiplying Trend with these to obtain original dataset.

Summarizing, for 04/Jan/2022:

Additive

222.020 (Observed) = 280.4077(Trend) + 66.1780(Seasonal) + -124.5657 (Residual)

Multiplicative

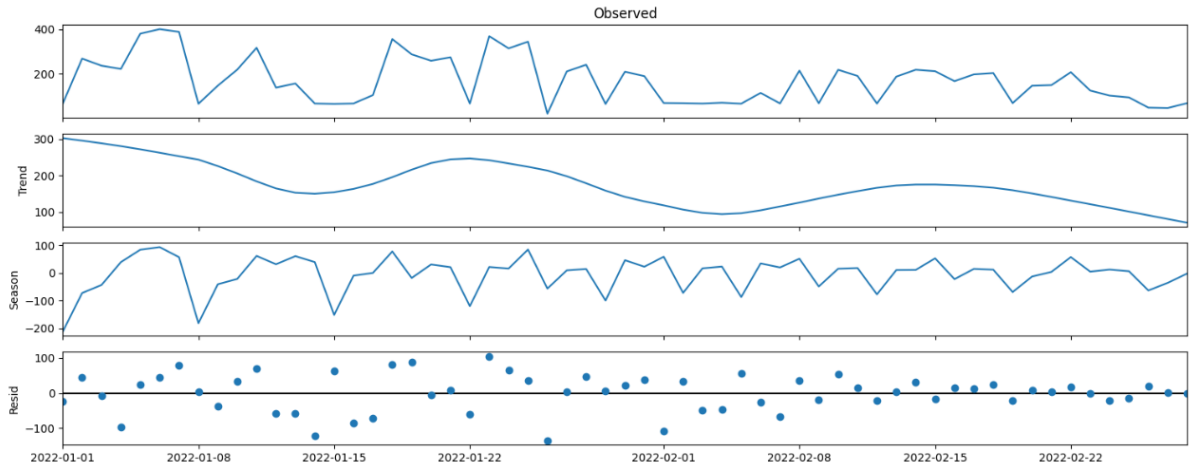
222.020 (Observed) = 280.4077(Trend) * 1.4072(Seasonal) * 0.5626 (Residual)

STL Decomposition (Seasonal-Trend decomposition using Loess)

STL is a robust and modern method introduced by Cleveland et al. (1990). It is often preferred over classical methods because:

- **Robustness:** It is less sensitive to outliers, as it uses **Loess (Locally Estimated Scatterplot Smoothing)** for trend and seasonality estimation.
- **Flexibility:** The seasonal component can vary over time, and the user has greater control over the smoothing parameters.
- **Additive Only:** By definition, STL works under the additive framework. If the data is multiplicative, it must first be transformed using a logarithm ($\ln(y_t)$).

```
from statsmodels.tsa.seasonal import STL
plt.figure(figsize=(15, 6))
decom2 = STL(df, seasonal=7).fit()
decom2_df = pd.DataFrame({'y':df['Consumption'], 'trend':decom2.trend,
'seasonality':decom2.seasonal, 'residual':decom2.resid})
decom2.plot()
plt.show()
```


Observed
Trend
Season
Resid
2022-01-01
2022-01-08
2022-01-15
2022-01-22
2022-02-01
2022-02-08
2022-02-15
2022-02-22

When compared with the **seasonal_decompose** approach, a few interesting differences become evident. The trend component appears smoother and extends from the very first data point, since it is estimated using the LOESS method. The seasonal pattern is not constant over time, which is often closer to reality, as there is no inherent reason to assume that seasonality remains fixed throughout the series.

For 04/Jan/2022:

222.02 (Observed) = 280.25(Trend) + 39.70(Seasonal) + -97.93 (Residual)

References:-

Cleveland, R. B., Cleveland, W. S., McRae, J., & Terpenning, I. (1990). STL: A Seasonal-Trend Decomposition Procedure Based on Loess. Journal of Official Statistics, 6(3), 209-228.

4.0 Simple Forecasting Approaches

These simple approaches assume that past patterns will continue into the future and are particularly useful for data smoothing and short-term predictions when data is relatively stable and free of complex trends or seasonality. Two of the most straightforward methods are the **moving average** and **exponential smoothing**. They are easy to implement, require minimal computational resources, and serve as building blocks for more advanced techniques.

4.1 Moving Average

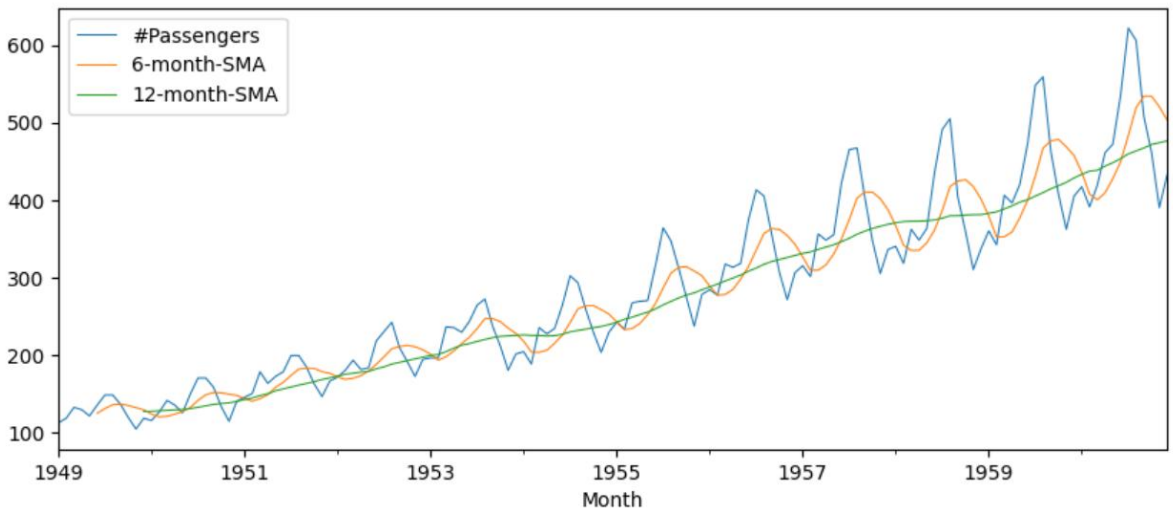
The moving average method smooths out short-term fluctuations in data by averaging a fixed number of past observations to generate a forecast. Imagine you're tracking daily temperatures instead of reacting to a single unusually hot day, you average the last few days' readings to get a more stable prediction.

For a simple moving average (SMA), the forecast for the next period is the arithmetic mean of the most recent k observations, where k is the window size (e.g., 3-day or 5-period average). As new data arrives, the oldest value drops out, and the newest is added—hence the "moving" aspect. The formula is:

$$y_{t+1} = \frac{1}{k}\sum_{i=0}^{k-1} y_{t-i}$$

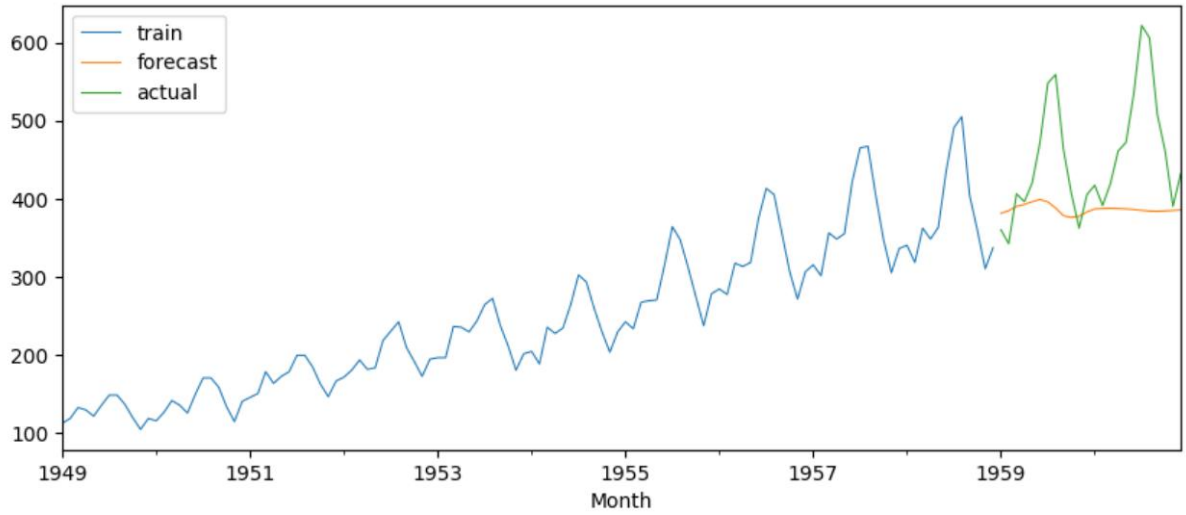
Using python, rolling window method can be used to create moving average in a one line code.

```
airline['6-month-SMA']=airline['#Passengers'].rolling(window=6).mean()
airline['12-month-SMA']=airline['#Passengers'].rolling(window=12).mean()
```



Applying this approach for forecasting would involve creating forecast one period at a time and updating the data. The code below shows how to forecast for 24 months using a window size of 12.

```
train = airline2[0:120].copy()
test = airline2[120:]

window_size = 12
sma = train['#Passengers'].rolling(window_size).mean()
for i in range(24):
    forecast = sma.iloc[-1]
    last_date = train.index[-1]
    last_date1 = last_date + pd.DateOffset(months=1)
    train.loc[last_date1, '#Passengers'] = forecast
    sma = train['#Passengers'].rolling(window_size).mean()
```



Note that forecast performance is not very impressive. This is because of presence of trend in the data and long period of forecast.

Advantage of moving average is that it is intuitive, reduces noise, and is unbiased for stationary data (data without trends). Limitation is that it treats all past data equally, so it lags behind sudden changes and doesn't adapt well to trends. This method is ideal for quick, rough estimates, like inventory planning in retail.

4.2 Simple Exponential Smoothing

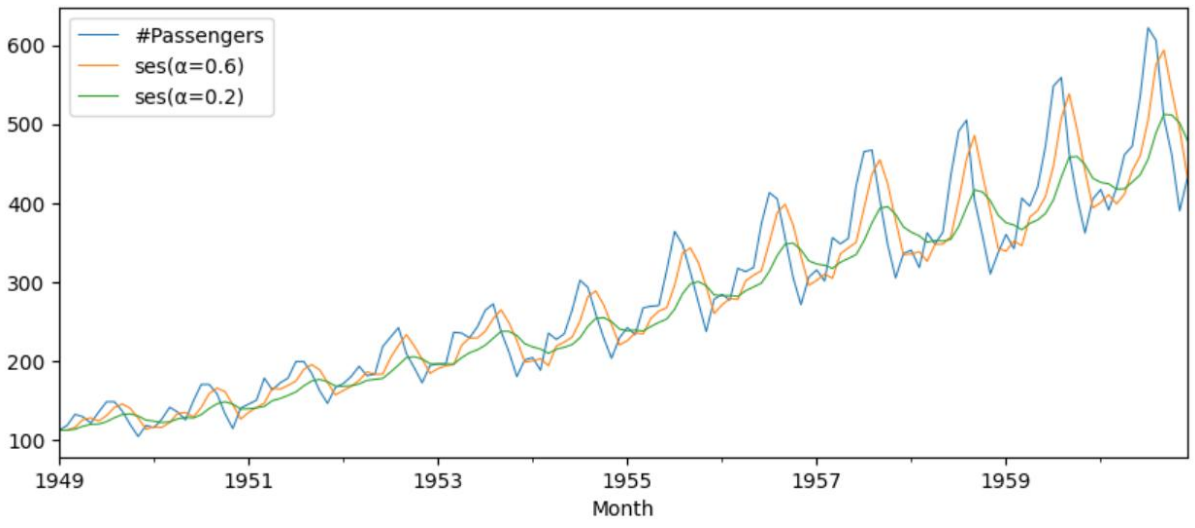
Exponential smoothing builds on the moving average idea but gives more weight to recent observations, making it more responsive to new information. It's like a weighted average where "fresh" data has a bigger voice, decaying the influence of older data exponentially.

The basic (simple) exponential smoothing formula is:

$$\hat{y}_{t+1} = \alpha y_t + (1 - \alpha)\hat{y}_t$$

Here, α (alpha) is the smoothing parameter between 0 and 1—higher values (e.g., 0.8) emphasize recent data, while lower ones (e.g., 0.2) smooth more aggressively. The forecast starts with an initial value (often the first observation) and updates iteratively.

```
alpha=0.2
fit02 = SimpleExpSmoothing(airline2).fit(smoothing_level=alpha,
optimized=False)
fitted02 = fit02.fittedvalues
alpha=0.6
fit06 = SimpleExpSmoothing(airline2).fit(smoothing_level=alpha,
optimized=False)
fitted06 = fit06.fittedvalues
```



It's adaptive, computationally simple, and performs well for data with mild trends or level shifts. Like moving averages, the basic version assumes no trends or seasonality; extensions (e.g., Holt's method) address this but add complexity. Exponential smoothing shines in

scenarios like demand forecasting for perishable goods, where recent sales are more indicative.

In practice, both methods are often evaluated using metrics like Root Mean Squared Error (RMSE) to fine-tune parameters. While they may not capture intricate patterns, they demystify forecasting and provide reliable baselines before diving into models like ARIMA or neural networks. Experimenting with real datasets can reveal their strengths in your specific context.

4.3 Holt-Winters Method

Holt (1957) extended simple exponential smoothing to allow the forecasting of data with a trend. This method involves a forecast equation and two smoothing equations (one for the level and one for the trend).

Holt (1957) and Winters (1960) extended Holt's method to capture seasonality. The Holt-Winters seasonal method comprises the forecast equation and three smoothing equations — one for the level ℓ_t, one for the trend b_t, and one for the seasonal component s_t, with corresponding smoothing parameters α, β^* and γ.

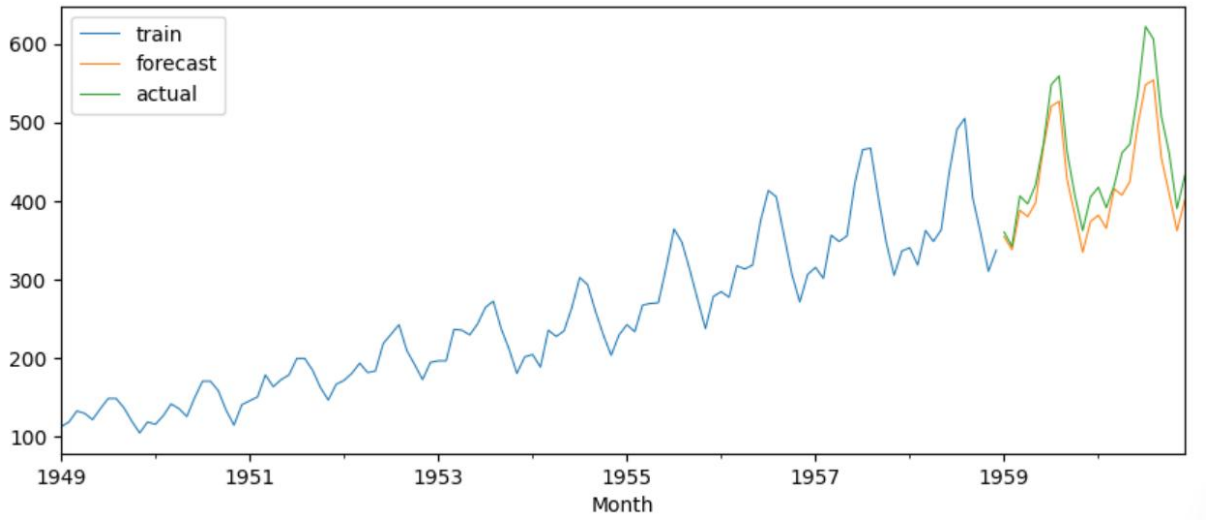
There are two versions:

- **Additive model:** suitable when seasonal variations are roughly constant.
- **Multiplicative model:** suitable when seasonal variations grow/shrink proportionally with the trend.

Formula governing additive model is provided below.

$$\begin{aligned}\hat{y}_{t+h|t} &= \ell_t + hb_t + s_{t+h-m(k+1)} \\ \ell_t &= \alpha(y_t - s_{t-m}) + (1-\alpha)(\ell_{t-1} + b_{t-1}) \\ b_t &= \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)b_{t-1} \\ s_t &= \gamma(y_t - \ell_{t-1} - b_{t-1}) + (1-\gamma)s_{t-m},\end{aligned}$$

```
train = airline2[0:120]
test = airline2[120:]
hw = ExponentialSmoothing(train['#Passengers'], trend='add',
         seasonal='add', seasonal_periods=12).fit(optimized=True)
fcast1 = hw.forecast(len(test))
```



Advantages:- Captures trend and seasonality simultaneously — more realistic than simple or double exponential smoothing. It is applicable without complex statistical assumptions, no need for stationarity like in ARIMA. This method is comparatively easy to interpret as it decomposes series into understandable parts — level, trend, and seasonal.

References:-

Holt, C. C. (1957). *Forecasting seasonals and trends by exponentially weighted averages* (ONR Memorandum No. 52). Carnegie Institute of Technology, Pittsburgh USA. Reprinted in the *International Journal of Forecasting*, 2004.

Winters, P. R. (1960). Forecasting sales by exponentially weighted moving averages. Management Science, 6(3), 324–342.

5.0 Classic Approaches – ARMA and ARIMA

The ARIMA (AutoRegressive Integrated Moving Average) model is one of the most influential developments in time series modeling, integrating multiple ideas that evolved over decades. The concept of autocorrelation and autoregression first appeared in Yule's paper on sunspot numbers. He introduced the Autoregressive (AR) model to explain how present values in a time series depend on past values:

$$X_t = \phi_1 X_{t-1} + \phi_2 X_{t-2} + \ldots + \phi_p X_{t-p} + \varepsilon_t$$

where ε_t is white noise.

During 1938, Herman Wold (Wold's Theorem) proved that any stationary time series can be decomposed into two parts. A deterministic part (predictable) and a stochastic part represented as a Moving Average (MA) process:

$$X_t = \mu + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \ldots$$

This provided the theoretical basis for MA models, expressing a series as a combination of current and past random shocks.

During 1930s–1940s, Slutsky, Kendall, and others explored how random shocks filtered through systems could produce smooth and cyclical patterns-a key intuition behind MA models.

The real synthesis came in the 1970s through the work of George Box and Gwilym Jenkins, who formalized the ARIMA framework in their seminal book 'Time Series Analysis: Forecasting and Control' (1970).

They combined:

- AR (dependence on past observations),
- MA (dependence on past shocks), and
- Integration (I) to handle non-stationarity (differences between consecutive observations).

5.1 Autoregressive (AR) Process

In an autoregression model, the variable of interest is forecasted using a linear combination of *past values of the variable*. The term *auto*regression indicates that it is a regression of the variable against itself. An **Autoregressive process of order *p*, denoted AR(*p*)**, models the current value of a series as a linear function of its *p* past values, plus a random error term. Equation:

$$X_t = c + \phi_1 X_{t-1} + \phi_2 X_{t-2} + \ldots + \phi_p X_{t-p} + \varepsilon_t$$

where,

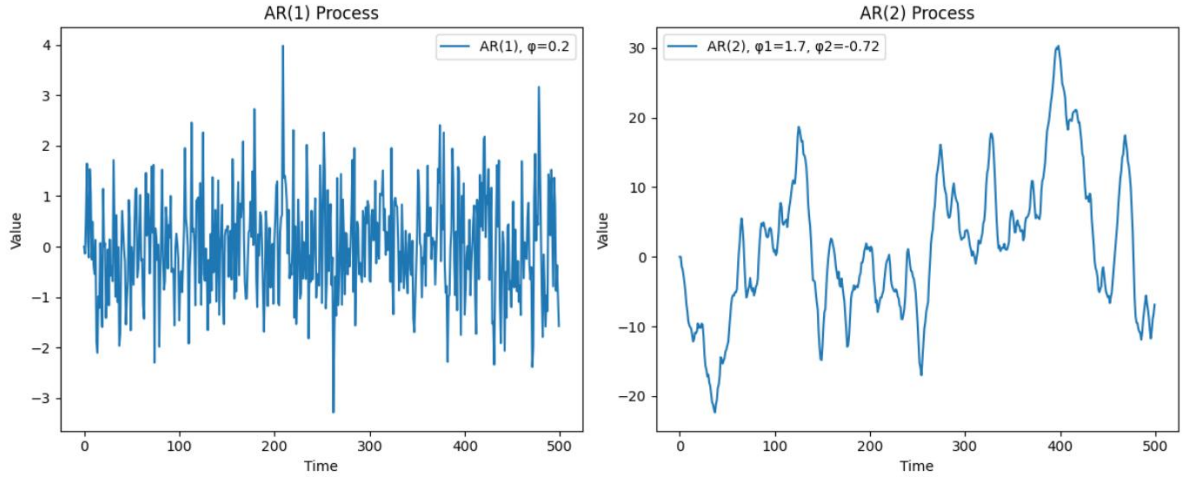
X_t value of the series at time t.
ϕ_i coefficient of the i^{th} lagged value.

ε_t White noise error (random shock) at time t
c constant term (drift/mean)

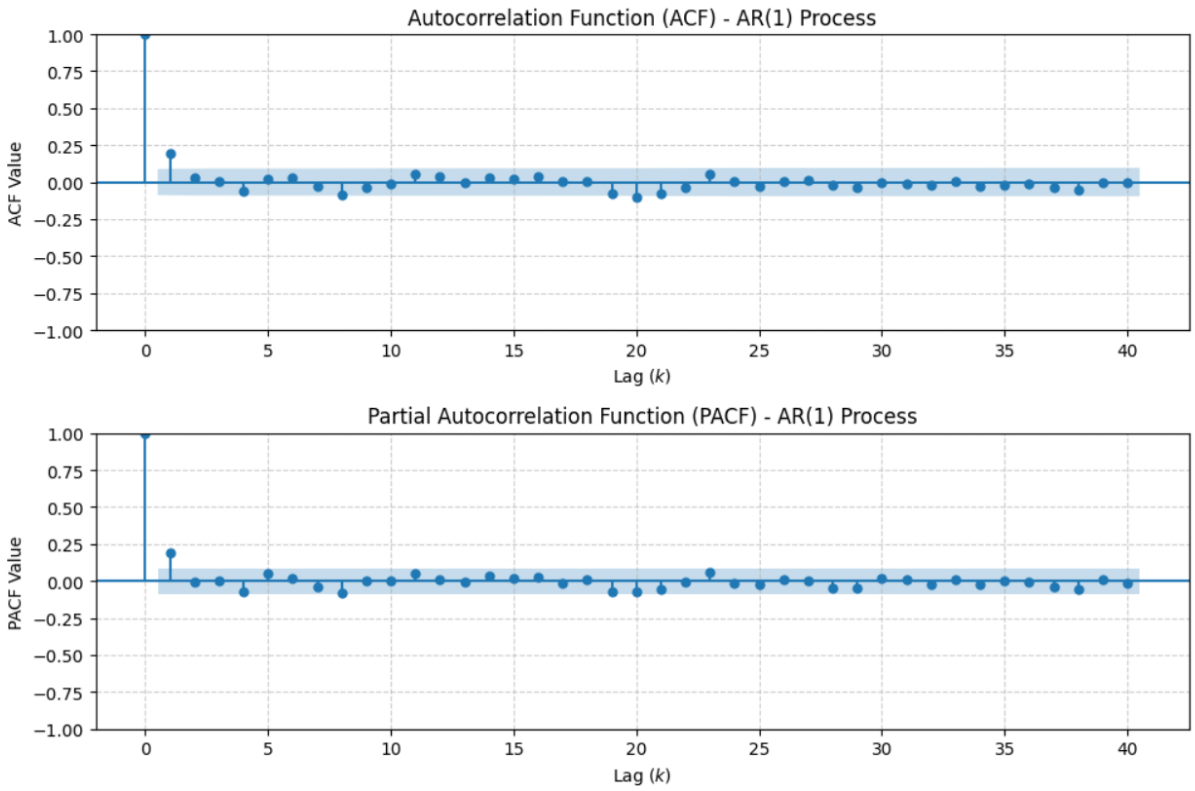
What is Special about AR?

- Long Memory/Decay: AR processes are characterized by an Autocorrelation Function (ACF) that decays exponentially or sinusoidally. This indicates that a shock to the system has a long-lasting, but diminishing, effect on future values.
- Identification: The lag *p* is easily identified by the Partial Autocorrelation Function (PACF), which cuts off sharply after lag *p*.

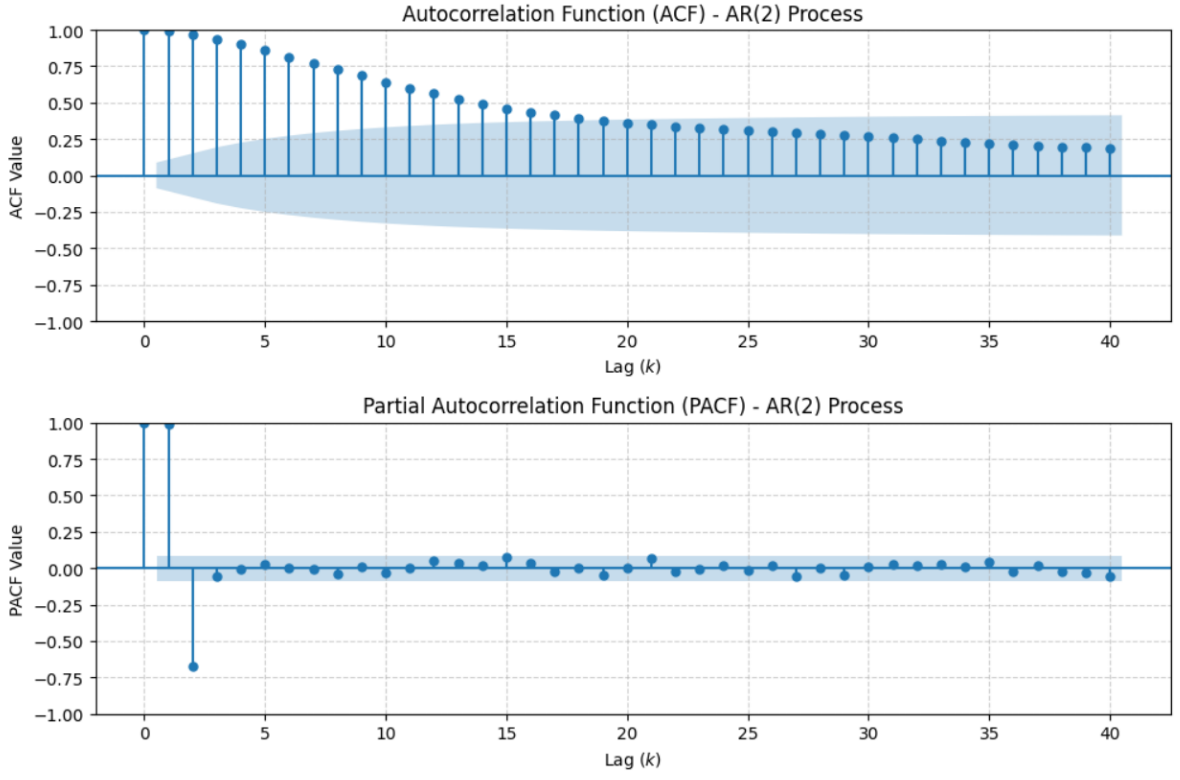
Autoregressive models are remarkably flexible at handling a wide range of different time series patterns. The two series in the figure show series from an AR(1) and an AR(2) model. Changing the parameters $\phi_1,\ldots,\phi_p$ results in different time series patterns. The variance of the error term ε_t will only change the scale of the series, not the patterns.



As discussed above, the characteristics of AR process can be detected by observing the behaviour of ACF and PACF plots. ACF plot of AR(1) process below shows sinusoidal behaviour with a sharp drop. PACF plot cuts off sharply after *p*=1.



ACF plot of AR(2) process below shows sinusoidal behaviour with a sharp drop. PACF plot cuts off sharply after p=2.



AR process is quite versatile as the choice of ϕ and c can provide different types of time series.

- when $\phi_1=0$ and $c=0$, X_t is equivalent to white noise
- when $\phi_1=1$ and $c=0$, X_t is equivalent to random walk
- when $\phi_1=1$ and $c\neq0$, X_t is equivalent to random walk with drift
- when $\phi_1<0$, X_t tends to oscillate around its mean
-

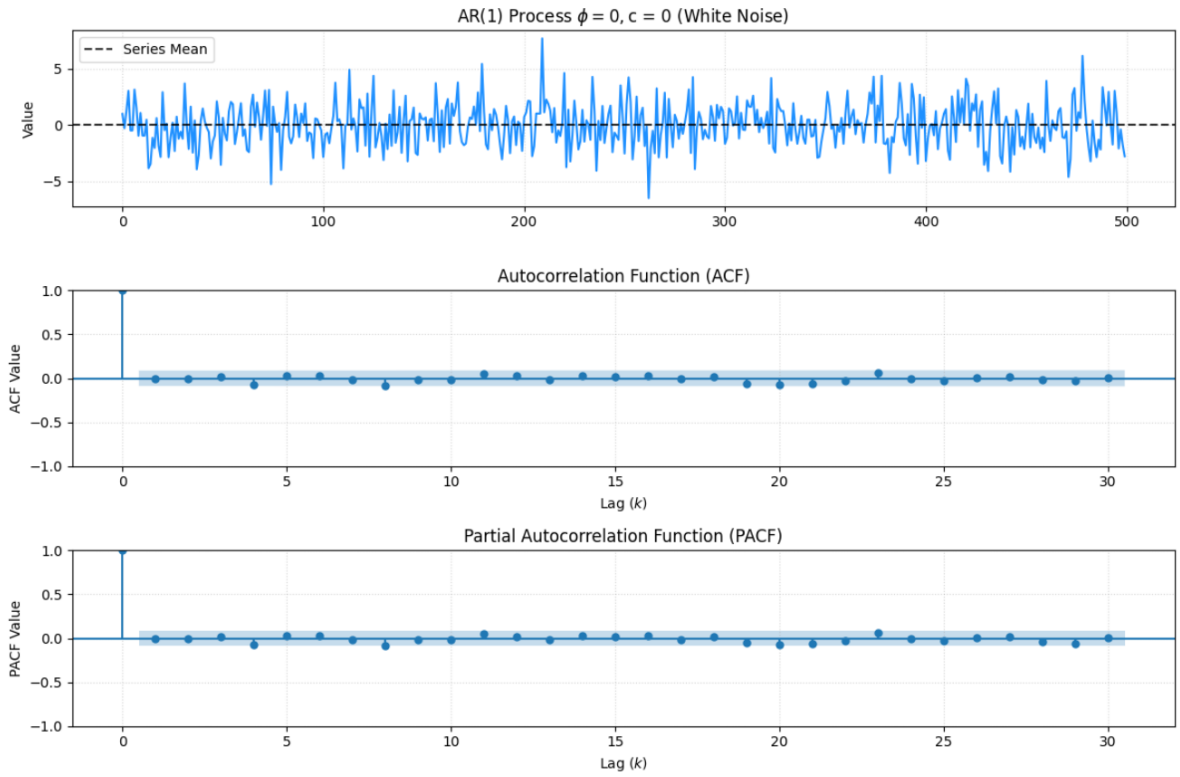
5.1.1 White noise

White noise in time series refers to a sequence of random variables that are independently and identically distributed (i.i.d.) with the following characteristics:

- Zero Mean: The expected value (mean) of the series is zero.
- Constant Variance: The variance of the series is constant over time.
- No Autocorrelation: The observations are uncorrelated with each other, meaning there is no predictable pattern or relationship between values at different time points.

Examples:-

- Stock Price Returns (Residuals) after accounting for trends or patterns in stock price movements (e.g., daily returns), the remaining fluctuations often resemble white noise because they are unpredictable and random.
- In audio or communication systems, background "hiss" or static noise (like the sound from an untuned radio or TV) is often modeled as white noise because it contains random fluctuations across all frequencies.
- In economic time series (e.g., GDP growth or unemployment rates), unexpected shocks or residuals after modeling trends and seasonality can approximate white noise.



5.1.2 Random walk

Random walk in time series is a process where the current value of a variable is composed of the previous value plus a random step (noise), which is typically drawn from a probability distribution (often normal with mean zero). Mathematically, a random walk can be expressed as:

$$X_t = X_{t-1} + \epsilon_t$$

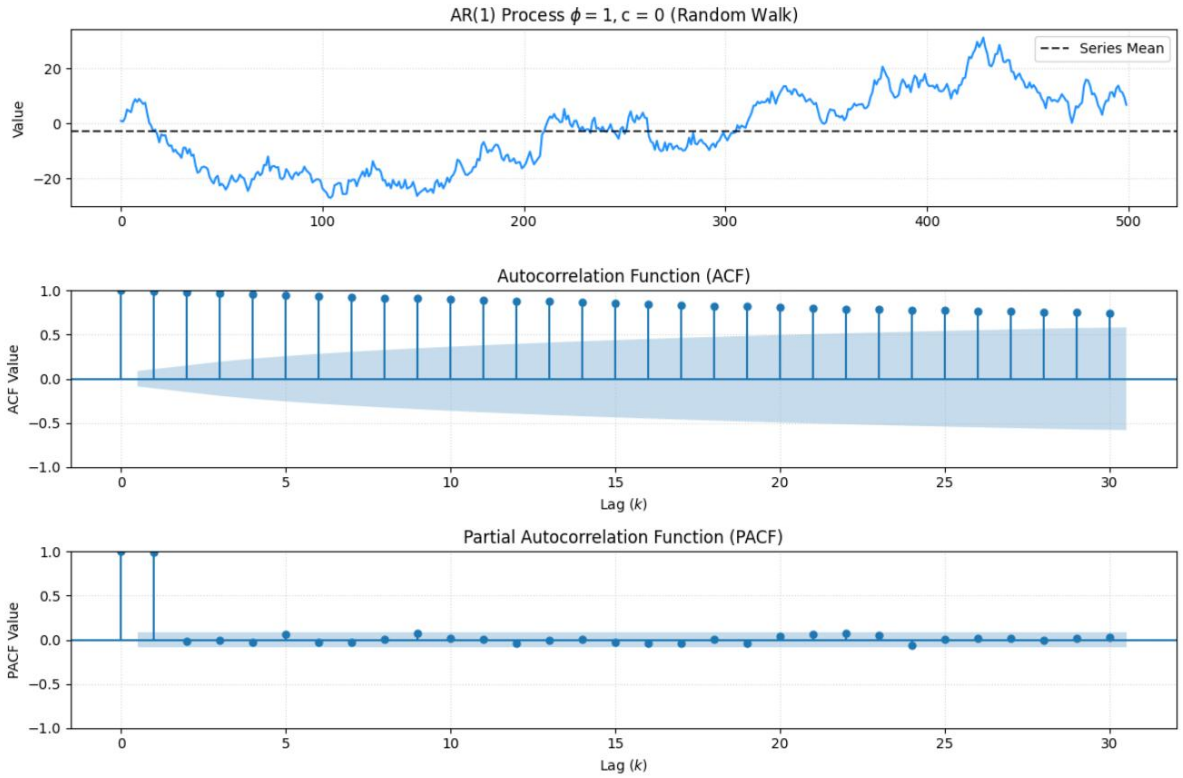
Key Characteristics of a Random Walk:

- **Non-stationary:** The mean and variance of a random walk are not constant over time. The variance increases as time progresses.
- **Unpredictable:** Future values cannot be predicted with certainty, as they depend on random shocks.
- **No mean reversion:** Unlike some time series that tend to revert to a mean, a random walk has no tendency to return to a specific value.
- **Unit root:** A random walk has a unit root and differencing the series (i.e.,$X_t - X_{t-1}$) results in a stationary process.

Examples:-

- Stock prices are often modeled as random walks (or geometric random walks for returns). The price at any given time is influenced by the previous price plus random market fluctuations driven by news, trading, or other unpredictable factors.

- Currency exchange rates, such as USD/EUR, tend to follow a random walk in the short term. Their movements are influenced by unpredictable economic events, policies, or market sentiment.
- Animal Foraging Paths: The movement paths of animals searching for food in an unpredictable environment can resemble a random walk, as they make random turns or steps in response to local conditions.
- Certain economic time series, like GDP or consumer price indices, may exhibit random walk-like behavior over short periods, especially when shocks (e.g., policy changes, natural disasters) introduce unpredictable changes.



5.1.3 Random walk with drift

Random walk with drift is a time series model where the value at each time step is the previous value plus a random step (usually drawn from a normal distribution) and a constant term called the drift. Mathematically, it can be expressed as:

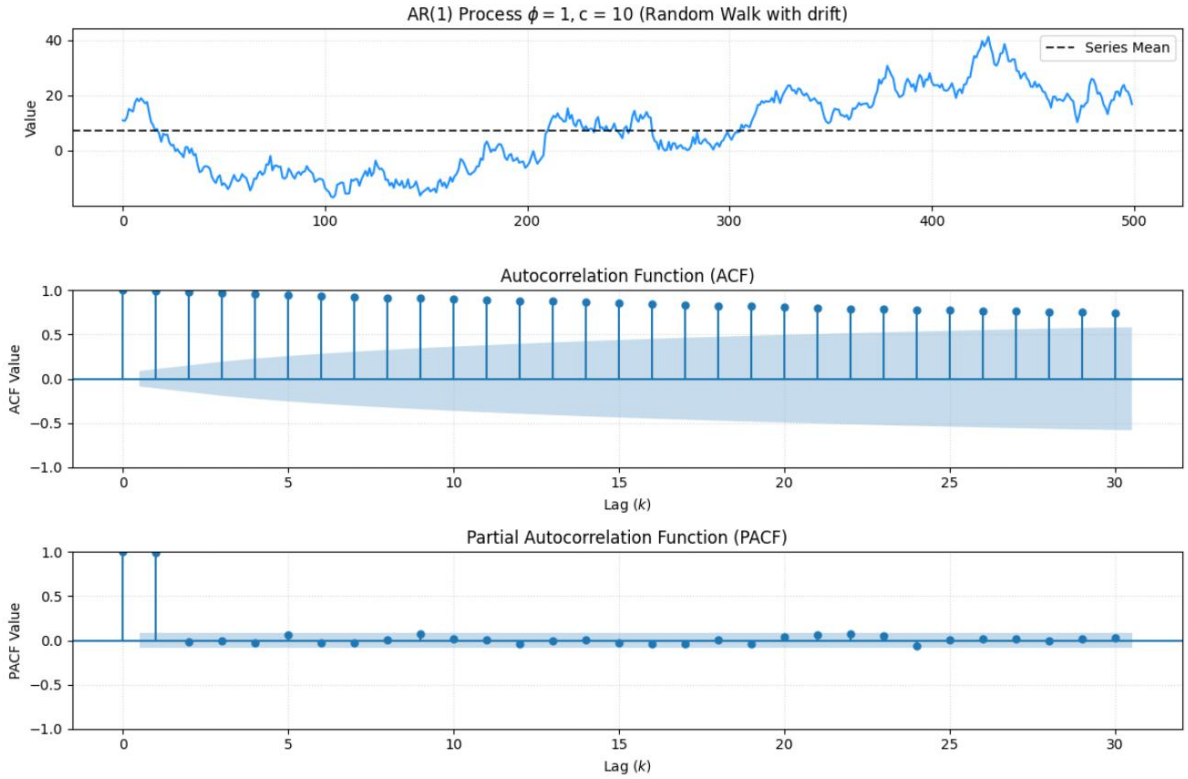
$$X_t = X_{t-1} + \mu + \epsilon_t$$

Key Characteristics:

- Non-stationary: The series has no fixed mean; it trends upward or downward due to the drift (μ).
- Random fluctuations: The noise term (ϵ_t) introduces randomness, causing the series to wander unpredictably around the trend.
- Unit root: A random walk with drift has a unit root, meaning it doesn't revert to a mean over time.

Examples:-

- Stock prices often exhibit a random walk with drift, where the drift represents the expected long-term growth rate (e.g., due to economic expansion or company performance), and the random component reflects daily price fluctuations driven by market noise.
- Global average temperatures can resemble a random walk with drift, where the drift represents long-term warming due to climate change, and random fluctuations arise from weather variability.
- Certain economic metrics, like nominal GDP or consumer price indices, can behave like a random walk with drift. The drift captures long-term growth or inflation, while random shocks (e.g., policy changes, economic events) add volatility.



5.1.4 Oscillating around the mean

Oscillating around the mean refers to a process where the values fluctuate above and below a fixed or long-term average (mean) over time, without a consistent upward or downward trend. This behavior is characteristic of a stationary time series, where the statistical properties (mean, variance, autocorrelation) remain constant over time. The oscillations are typically driven by random noise or cyclical patterns, but the series tends to revert to its mean rather than wandering indefinitely.

$$X_t = \mu + \epsilon_t$$

Key Characteristics:-

- **Stationarity:** The time series has a constant mean and variance, and the oscillations do not exhibit a trend or drift.

- **Mean-reverting:** Values tend to return to the mean after deviating, unlike a random walk where deviations accumulate.
- **Random or cyclical fluctuations:** The oscillations can be random (e.g., white noise) or follow a periodic pattern (e.g., seasonal or cyclical components).

Examples:-

- Daily temperatures in a stable climate oscillate around a seasonal average. For example, in a temperate region, summer temperatures might fluctuate around a mean of 25°C due to weather patterns, without a long-term trend in the short term.
- Economic Indicators with Stable Dynamics: Certain economic metrics, like unemployment rates in a stable economy, may oscillate around a long-term average due to short-term economic cycles or random shocks, without a persistent trend.
- Stock Market Returns: While stock prices may follow a random walk with drift, daily or weekly returns (percentage changes) often oscillate around a mean (e.g., a small positive average return). These returns are typically stationary, with fluctuations driven by market noise.

5.2 Moving Average (MA) Process

The MA process, denoted as MA(q), models a time series as a linear combination of a fixed number of past error terms (random shocks) plus a constant mean. It assumes the time series is stationary, meaning it oscillates around a constant mean with stable variance. The "q" in MA(q) refers to the order of the moving average, indicating how many lagged error terms are included.

Mathematically, an MA(q) process is expressed as:

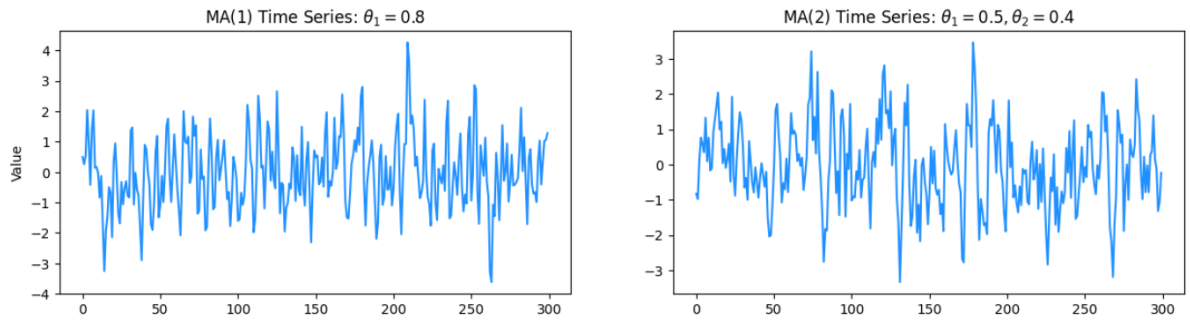
$$X_t = \mu+\varepsilon_t+\theta_1\varepsilon_{t-1}+\theta_2\varepsilon_{t-2}+\ldots+\theta_q\varepsilon_{t-q}$$

where
ε_t is white noise. We refer to this as an MA(q) model, a moving average model of order q.

Of course, we do not observe the values of ε_t, so it is not really a regression in the usual sense. Each value of X_t can be thought of as a weighted moving average of the past few forecast errors (although the coefficients will not normally sum to one).

> **Clarity of terms**: - MA term here should not be confused with the moving average smoothing we discussed in the previous section. A moving average model is an **average of past values** used for smoothing or forecasting future values. MA term here is termed so because it is literally a moving (sliding in time) **weighted average of error terms**.

Figure below shows MA(1) and MA(2) models. Changing the parameters of $\theta_1,\ldots,\theta_q$ results in different time series patterns. As with autoregressive models, the variance of the error term ε_t will only change the scale of the series, not the patterns.



Key Features of the MA Process:

Stationarity: The MA process is inherently stationary, as it depends only on random shocks and not on past values of the series itself, ensuring a constant mean and variance.

Short-term memory: The MA(q) process has a finite memory, as it only considers the past q error terms. Beyond q lags, the autocorrelation of the series drops to zero.

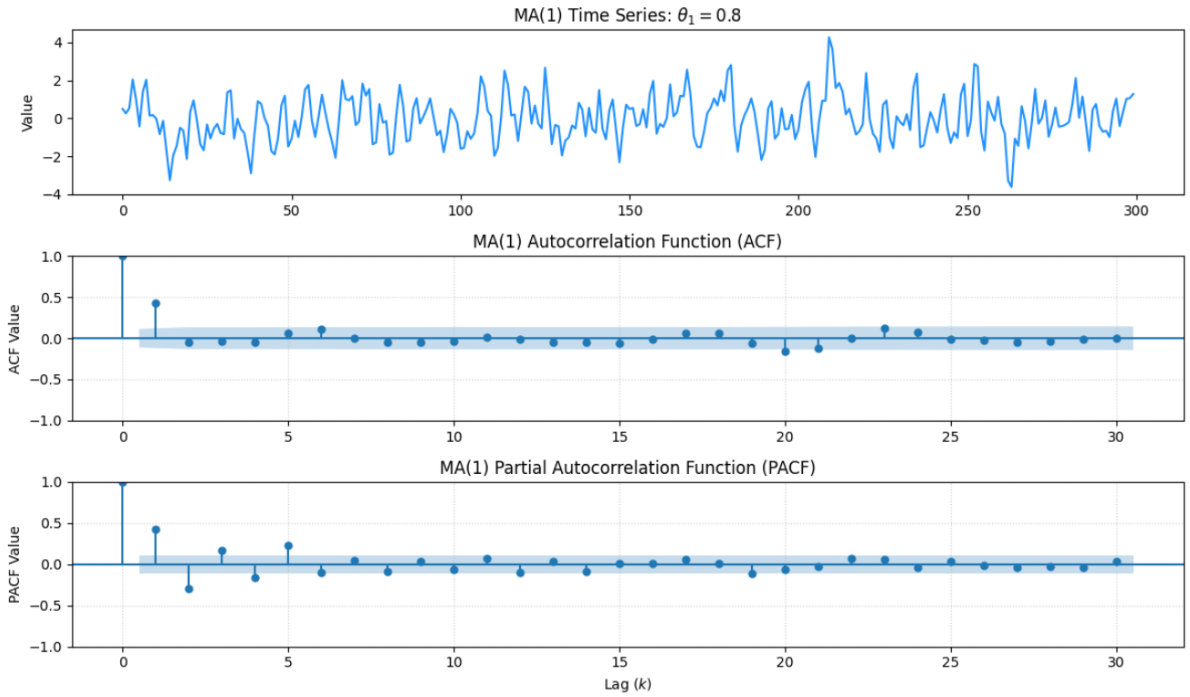
Examples:-

- Daily stock returns often exhibit short-term dependencies due to random market shocks (e.g., news or trading activity). An MA(1) or MA(2) model might capture how recent shocks affect current returns.
- Economic Indicators: Metrics like monthly retail sales may experience random shocks (e.g., supply chain disruptions) that affect sales for a few months. An MA process can model these temporary effects.
- Daily temperature anomalies (deviations from the seasonal average) may depend on recent random weather events. An MA process can model how these shocks persist for a few time steps.

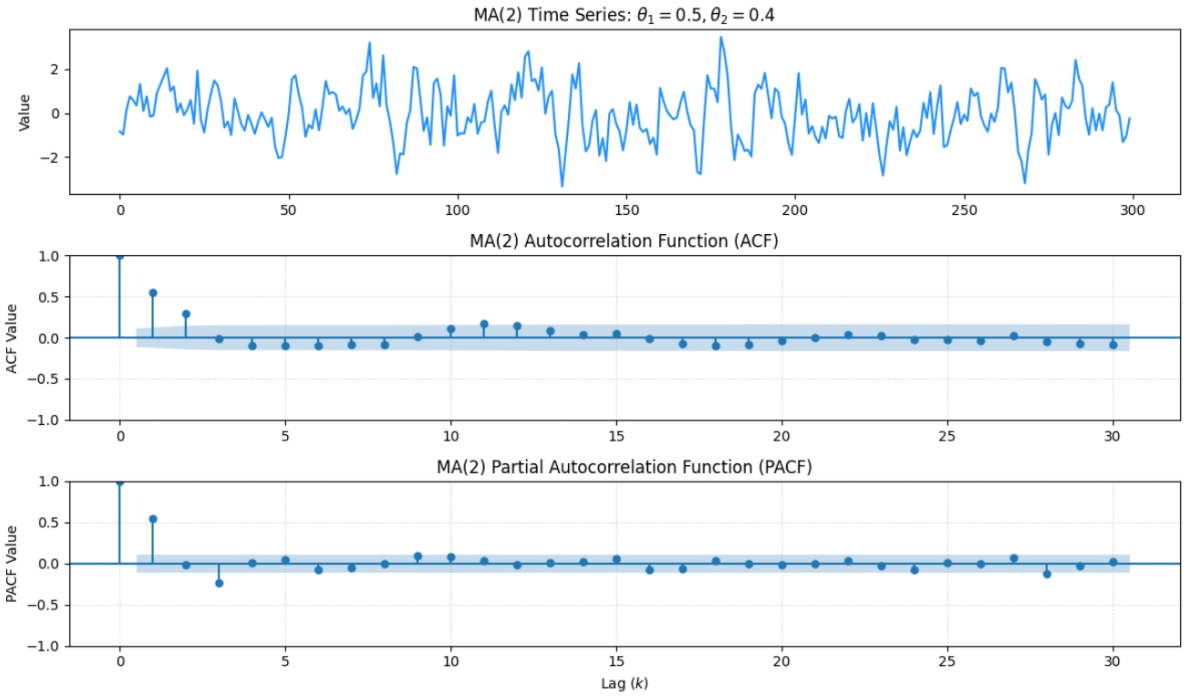
The ACF (Autocorrelation Function) and PACF (Partial Autocorrelation Function) are crucial diagnostic tools for identifying the order q of an MA(q) process. These functions reveal the theoretical patterns in the autocorrelation structure of the time series, helping us distinguish MA processes from other models like AR.

- ACF pattern cuts off after lag q and significant autocorrelations are observed from lag 1 to lag q (zero autocorrelations (theoretically) after lag q). The ACF "cuts off" abruptly at lag q.

- PACF pattern tails off after lag q and significant partial autocorrelations at lag 1 to lag q. After lag q, it gradually decays to zero ("tails off" exponentially).



ACF pattern above cuts off after lag 1 and PACF pattern tails off after lag 1.



ACF pattern cuts off after lag 2 and significant autocorrelations are observed on lag 1 and lag 2 (zero autocorrelations (theoretically) after lag 2). The ACF "cuts off" abruptly at lag 2.

PACF pattern tails off after lag 2 and significant partial autocorrelations at lag 1 and lag 2. After lag 2, it gradually decays to zero ("tails off" exponentially).

5.3 ARMA Process

The ARMA process is a fundamental model in time series analysis, used to describe and predict stationary data — that is, data whose statistical properties (mean, variance, autocorrelation) remain constant over time.

This process **combines AR and MA**, to get a **richer and more flexible structure** that captures both:

- The *momentum or persistence* in the series (via AR terms), and
- The *shock or sudden effect* from past errors (via MA terms).

This combination allows ARMA models to approximate a wide range of stationary processes with fewer parameters.

Mathematically, an ARMA(p, q) model combines both:

$$X_t = c + \sum_{i=1}^{p} \phi_i X_{t-i} + \epsilon_t + \sum_{j=1}^{q} \theta_j \epsilon_{t-j}$$

Characteristics of ARMA Process:

Stationarity: ARMA assumes the time series is stationary (mean, variance, and autocorrelation structure do not change with time). Non-stationary data are first differenced or transformed (leading to ARIMA models).

Linearity: The model is linear in parameters — the output is a weighted sum of past values and past errors.

Short-term dependence: ARMA captures correlations between nearby observations efficiently; autocorrelation dies down as the lag increases.

Parsimonious modeling: Combines AR and MA parts to model complex autocorrelation with fewer parameters than using AR or MA alone.

White noise residuals: After fitting a proper ARMA model, residuals (errors) should appear random (no autocorrelation).

5.3 ARIMA Process

While ARMA(p, q) models are mathematically elegant and work well for stationary processes, most real-world time series are not stationary. ARMA assumes that the series fluctuates around a constant mean and variance — a strong assumption that rarely holds in practice.

Most of the economic, financial, and environmental data show Trends (upward or downward e.g., GDP, population, temperature), Seasonality (repeating patterns e.g., monthly sales, electricity demand) and changing variance (volatility clustering e.g., stock returns). In such cases, the mean is not constant over time, variance may increase or decrease with time and autocorrelation structure changes with time. Thus, fitting a stationary ARMA model directly gives biased and unstable results.

For example, predicting inflation next month using ARMA may work, but predicting GDP 2 years ahead fails because of underlying trends.

5.3.1 ARIMA Improves Over ARMA

The ARIMA (AutoRegressive Integrated Moving Average) model extends ARMA by adding an "I" (Integration) term that represents differencing.

Mathematically:

ARIMA(p,d,q)

where;

p: AR order
d: number of differences needed to make the series stationary
q: MA order
The model is written as:

$$(1 - \sum_{i=1}^{p} \phi_i L^i)(1 - L)^d X_t = (1 + \sum_{j=1}^{q} \theta_j L^j)\varepsilon_t$$

Here L is the lag operator, and $(1-L)^d$ represents differencing.

Differencing removes trend and ARIMA performs differencing to handle non-stationary data (or multiple differences, if required).

$$X_t' = X_t - X_{t-1}$$

This converts a non-stationary series (like GDP or temperature) into a stationary one suitable for ARMA modeling. Differencing ensures the model captures growth patterns while keeping statistical stability.

Example: If GDP shows a steady upward trend, ARIMA(d=1) uses differences (ΔGDP) instead of raw GDP to model changes.

It is adaptable to seasonality by applying seasonal extensions like SARIMA (Seasonal ARIMA) add terms that explicitly model periodic patterns (monthly, quarterly, etc.).

Example:

$$\text{SARIMA}(p,d,q)(P,D,Q)_s$$

- handles both trend (via d) and seasonality (via D).

In summary, ARIMA accounts for both trend and random shocks. It provides more stable parameter estimates, lower forecast errors and wider applicability in business and science.

Real-Life Examples Illustrating the Difference

Application	Why ARMA Fails	How ARIMA Solves It
Monthly sales	Seasonal pattern and growth trend violate stationarity	ARIMA with differencing (and seasonal terms) removes trend and seasonality
GDP data	Rising over time; not stationary	ARIMA(d=1) captures changes rather than levels
Electricity demand	Strong daily/weekly cycles	SARIMA models seasonal periodicity
Stock prices	Non-stationary random walk	ARIMA(0,1,0) captures random walk nature
Temperature series	Trend due to climate change	ARIMA with differencing and seasonal terms adjusts for both trend and cycles

5.3.2 Identifying *p, d* and *q* of ARIMA process

Identifying the appropriate orders, ***p***,***d*** and ***q***, for an **ARIMA(*p*,*d*,*q*)** model is a critical step in the Box-Jenkins methodology. The core principle is to find a **parsimonious model**—the simplest model that adequately captures the statistical structure of the data.

Symbol	Meaning	Interpretation
p	Order of **AutoRegressive (AR)** part	Number of lagged values of the series (X_{t-1}, X_{t-2}, ...) used to predict (X_t)
d	Order of **Integration (I)**	Number of times the series is differenced to achieve stationarity
q	Order of **Moving Average (MA)** part	Number of lagged forecast errors (ε_{t-1},ε_{t-2},...) used to predict (X_t)

Classical (Manual) Identification Methods

These methods form the historical and conceptual foundation of ARIMA modeling — rooted in **Box–Jenkins methodology (1970s)**.

- Identify *d* (degree of differencing) to make the series *stationary in mean*. Plot the time series to identify if it shows trend or changing mean. If it's non-stationary, difference once (or twice) to flatten the trend. After this step test the series using **Augmented Dickey-Fuller (ADF)** test to check if the series is stationary or not. Note that most real-world series require ***d* = 0, 1, or 2**.
- Identification of *p* and *q* is done using **Autocorrelation Function (ACF)** and the **Partial Autocorrelation Function (PACF)** plots of the *stationary* time series.

The Rules of Identification

The shape of the ACF and PACF plots provides unique signatures for pure AR, pure MA, and mixed ARMA processes

Process	**AR(*p*)**	**MA(*q*)**	**ARMA(*p*,*q*)**
ACF	**Tails off** (Decays exponentially or sinusoidally)	**Cuts off** sharply after lag *q*	**Tails off**
PACF	**Cuts off** sharply after lag *p*	**Tails off**	**Tails off**

Since for ARMA, both ACF and PACF tails off, accurate identification *p* and *q* is trickier.

Modern and Industry-Standard Approaches

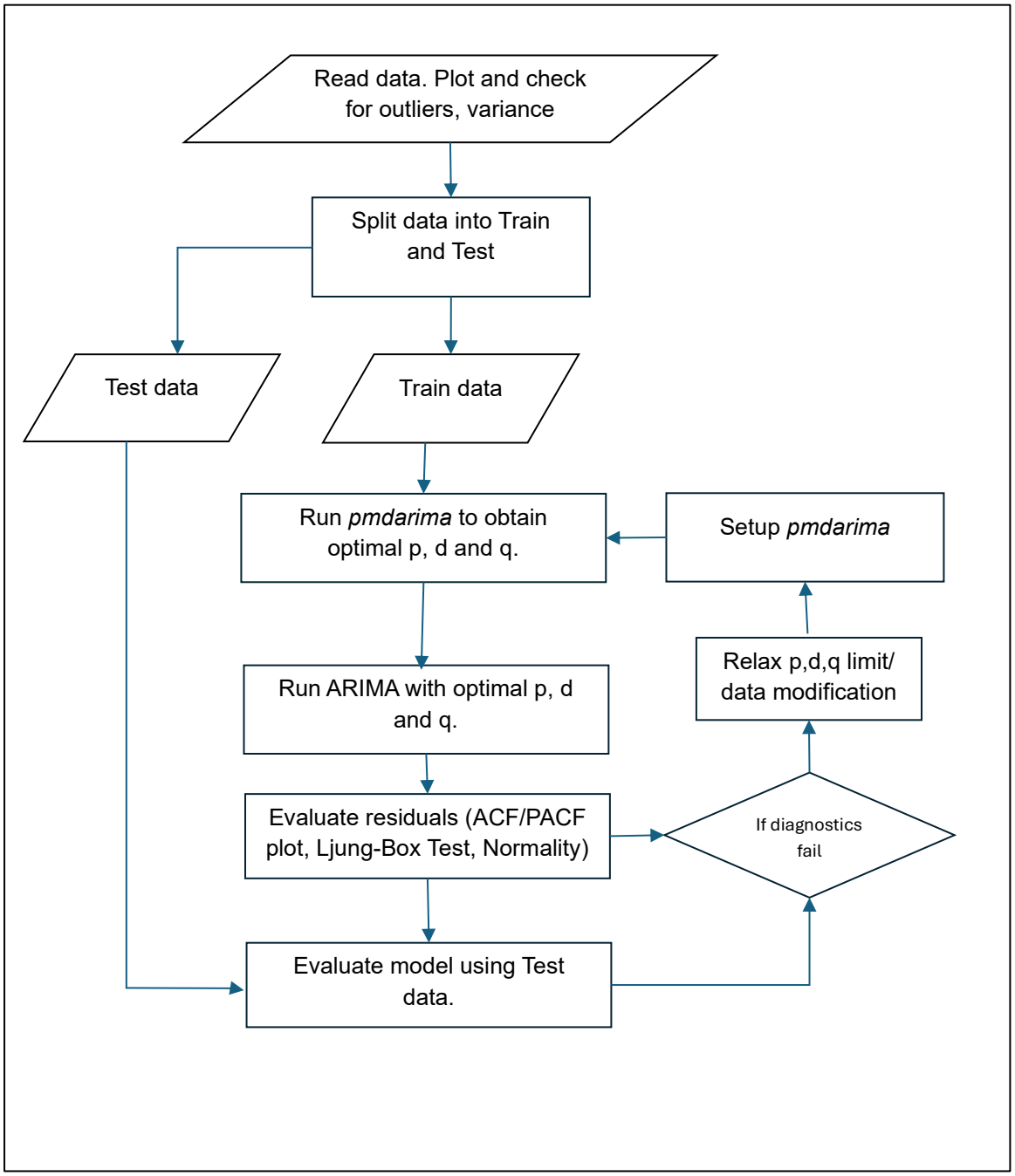
Due to the subjectivity of the classical approach, current practice heavily favours objective, automated methods based on information criteria. The most common and efficient practice is to use automated algorithms, best known through the *auto.arima* function (available in the *pmdarima* library in Python and the forecast package in R).

The broad approach is to define a reasonable grid of candidate models (e.g., *p* from 0 to 5, *q* from 0 to 5, with the predetermined *d*). Then fit every model in the grid (e.g., 36 total models). Select the (*p,d,q*) combination that yields the lowest AIC or BIC value.

Following are the steps of automated selection:

- Unit Root Testing: The function automatically uses unit root tests (like ADF or KPSS) to determine the optimal *d* order.

- Model Search: It performs a guided, stepwise search across a wide range of (p,d,q) parameters, often starting with a simple model (like (1,d,1)).



- Optimization: At each step, it calculates the AIC (or BIC) for the candidate models and moves toward the model with the lowest score.
- Final Output: It returns the single set of (p,d,q) parameters that minimizes the selected criterion, eliminating the need for manual plot interpretation and grid testing.

5.3.3 The Final Validation Step

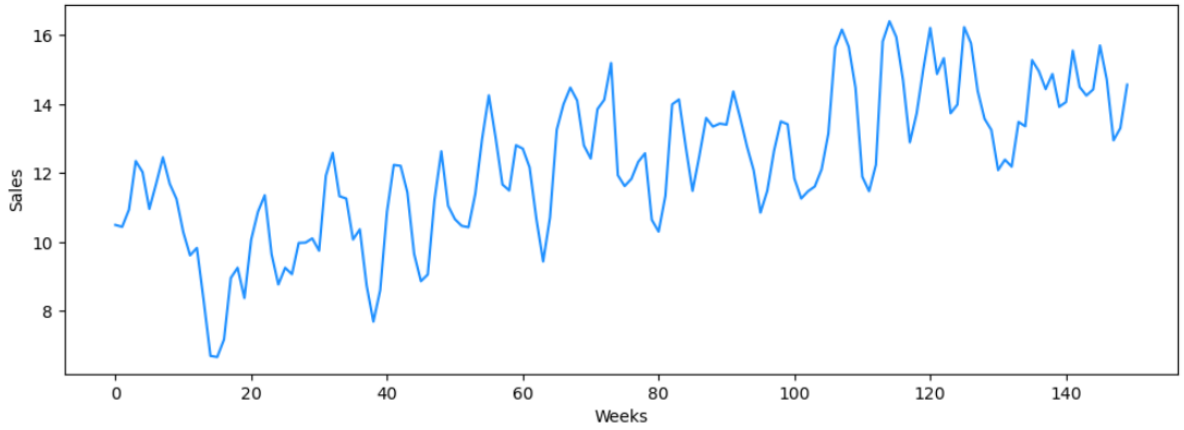
Regardless of the method used to select (*p,d,q*), the resulting model must undergo rigorous diagnostic checking:

- **Residual Analysis**: The residuals (ϵ_t) of the chosen ARIMA model must be uncorrelated (white noise).
- **Formal Testing**: A test like the Ljung-Box Test is applied to the residuals. If the test fails to reject the null hypothesis (high p-value), the model is considered adequate, and the orders (*p,d,q*) are confirmed. If it fails, the analyst return to the identification step and try a different model.

Issue Addressed	**Method**	**Details**
Underfitting / Correlated residuals.	**Increase *p* and *q* (Order)**	Systematically test higher orders for the AR and MA components. If pmdarima failed to find a good model, manually check models with slightly higher orders than suggested (e.g., if it chose ARIMA(1,1,1), try ARIMA(2,1,1) or ARIMA(1,1,2)).
Overfitting / Lack of parsimony.	**Change Information Criterion**	If the forecast is poor but residuals are fine, the model might be overfitting. Switch the objective function in *pmdarima* from **AIC** (tends to select complex models) to **BIC** (Bayesian Information Criterion, which heavily penalizes complexity).
Series not truly stationary.	**Optimize Integration Order (*d*)**	If the residuals drift or show a slow decay in the ACF, the series might be under-differenced. If *d*=1, try increasing to *d*=2 (though rarely needed). Conversely, if *d*=1 was applied unnecessarily, reduce it to *d*=0.

5.4 Case1:- Building a model to forecast sales for 25 weeks

The data contains 150 weeks of sales. As a first step, the data is checked for missing values and it is plotted.



The data does not suggest any outliers and it doesn't look seasonal. However, it seems there is a mild increasing trend. As part of the exploratory process, test for stationarity.

Test for Stationarity:-

Null and Alternative Hypotheses:

Null Hypothesis (H_0): The series has a unit root (non-stationary)

Alternative Hypothesis (H_1): The series is stationary

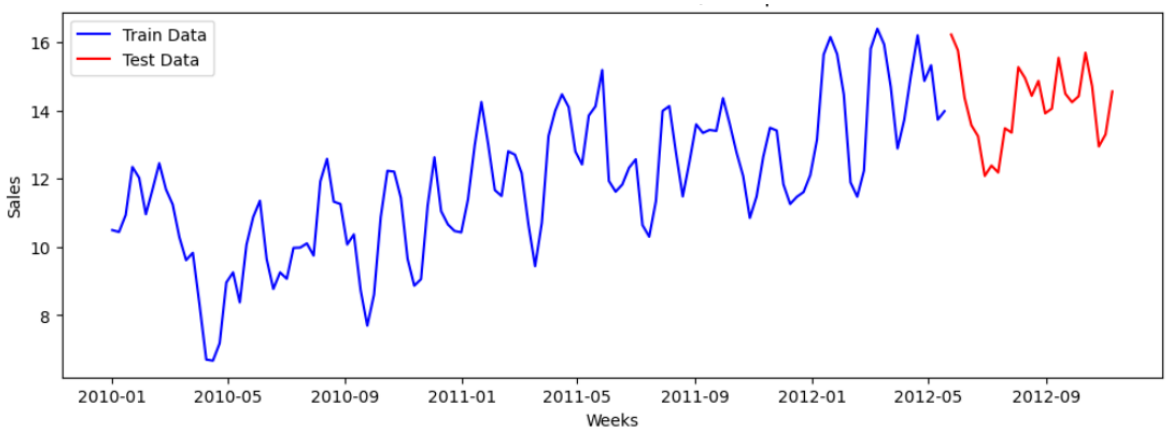
```
adf_result = adfuller(df, regression='c')
print(f"Initial ADF Test P-value: {adf_result[1]:.4f} (Expected <= 0.05 for
Stationarity")

df_diff = df.diff(periods=1)
df_diff = df_diff.dropna()

adf_result = adfuller(df_diff, regression='c')
print(f"ADF Test P-value: {adf_result[1]:.4f} (Expected <= 0.05 for
Stationarity)")

/****/
Initial ADF Test P-value: 0.3508 (Expected <= 0.05 for Stationarity
ADF Test P-value: 0.0000 (Expected <= 0.05 for Stationarity)
```

Note that the data is not stationarity and differencing by one period (d=1) was able to make it stationary.

Train-Test split:-

Since the requirement is to forecast for 25 weeks, the data is split into train and test based on this. The latest 25 weeks chosen as Test and the rest as Train.

```
train = df[:-25]
test = df[-25:]
```



5.4.1 Model building process

Next step is to build an ARIMA model for forecasting. As we discussed, the specification of ARIMA need to be decided (tuning) before ARIMA can be estimated. The required three parameters are *p*(order of AR), *q*(order of MA) and *d*(number of periods of differencing).

Exploratory analysis showed that one period differencing (*d*=1) is required to make the series stationary. Even though *d* was determined, tuning process would include *d* to let the system determine the best *d*. It also would act as a cross check.

To tune the ARIMA, *pmdarima* library is applied as illustrated below.

- *start* and *max* of *p* and *q* refers to minimum and maximum values.
- *d*=None results in the algorithm testing the data for stationarity and d is decided based on it. *max_d* puts an upper limit.
- *stepwise=True* directs the algorithm to use the stepwise algorithm outlined in Hyndman and Khandakar (2008) to identify the optimal model parameters. The stepwise algorithm can be significantly faster than fitting all (or a random subset of) hyper-parameter combinations and is less likely to over-fit the model.
- As we discussed, AIC and BIC are the most commonly used to select the best model from a set of candidate models by balancing goodness of fit with model complexity. Here BIC is used as it penalize complexity more heavily than AIC.

```
model_auto = pm.auto_arima(train,
    start_p=1, max_p=3,
    start_q=1, max_q=3,
    d=None, max_d=2, # Let auto_arima estimate the differencing order 'd'
    seasonal=False,
    stepwise=True, trace=True,
    suppress_warnings=True, error_action='ignore',
    information_criterion='bic')

/****/
Performing stepwise search to minimize bic
 ARIMA(1,1,1)(0,0,0)[0] intercept   : BIC=404.220, Time=0.20 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : BIC=407.526, Time=0.02 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : BIC=404.320, Time=0.03 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : BIC=399.624, Time=0.03 sec
 ARIMA(0,1,0)(0,0,0)[0]             : BIC=402.849, Time=0.02 sec
 ARIMA(0,1,2)(0,0,0)[0] intercept   : BIC=394.553, Time=0.07 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : BIC=inf, Time=0.25 sec
 ARIMA(0,1,3)(0,0,0)[0] intercept   : BIC=inf, Time=0.29 sec
 ARIMA(1,1,3)(0,0,0)[0] intercept   : BIC=inf, Time=0.28 sec
 ARIMA(0,1,2)(0,0,0)[0]             : BIC=390.985, Time=0.05 sec
 ARIMA(0,1,1)(0,0,0)[0]             : BIC=394.871, Time=0.02 sec
 ARIMA(1,1,2)(0,0,0)[0]             : BIC=382.289, Time=0.05 sec
 ARIMA(1,1,1)(0,0,0)[0]             : BIC=399.475, Time=0.04 sec
 ARIMA(2,1,2)(0,0,0)[0]             : BIC=376.113, Time=0.08 sec
 ARIMA(2,1,1)(0,0,0)[0]             : BIC=371.418, Time=0.06 sec
 ARIMA(2,1,0)(0,0,0)[0]             : BIC=388.251, Time=0.03 sec
 ARIMA(3,1,1)(0,0,0)[0]             : BIC=375.948, Time=0.07 sec
 ARIMA(1,1,0)(0,0,0)[0]             : BIC=399.574, Time=0.03 sec
 ARIMA(3,1,0)(0,0,0)[0]             : BIC=375.841, Time=0.04 sec
 ARIMA(3,1,2)(0,0,0)[0]             : BIC=380.376, Time=0.12 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : BIC=374.578, Time=0.12 sec

Best model:  ARIMA(2,1,1)(0,0,0)[0]
```

The steps above shows the combination of parameters evaluated by algorithm and the corresponding BIC values. The lowest BIC is chosen to decide the best model. The extra term (`(0,0,0)[0]`) is relevant only for a seasonal model.

Note that some of the values of *p* and *q* are 0 although *start* values were 1. This is decided by the choice `stepwise=True`. This directs the algorithm to use parameters around the values set for tuning.

5.4.2 Estimate the best model and forecast

Next step is to estimate the ARIMA model using the optimal parameters (2,1,1). For this, ARIMA class from *statsmodels* library will be deployed.

```
from statsmodels.tsa.arima.model import ARIMA, ARIMAResults

model = ARIMA(train, order=(2,1,1))
results = model.fit()
results.summary()

/****/
```

SARIMAX Results

Dep. Variable:	sales	**No. Observations:**	125
Model:	ARIMA(2, 1, 1)	**Log Likelihood**	-172.029
Date:	Sun, 26 Oct 2025	**AIC**	352.058
Time:	10:39:27	**BIC**	363.340
Sample:	01-01-2010	**HQIC**	356.641
	- 05-18-2012		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.9556	0.107	8.943	0.000	0.746	1.165
ar.L2	-0.5164	0.087	-5.954	0.000	-0.686	-0.346
ma.L1	-0.8169	0.068	-11.978	0.000	-0.951	-0.683
sigma2	0.9306	0.111	8.348	0.000	0.712	1.149

Ljung-Box (L1) (Q):	0.11	**Jarque-Bera (JB):**	1.55
Prob(Q):	0.74	**Prob(JB):**	0.46
Heteroskedasticity (H):	0.88	**Skew:**	-0.14
Prob(H) (two-sided):	0.68	**Kurtosis:**	3.47

This model will be evaluated using two broad approaches.

First will be **Model Diagnostics** (Statistical Adequacy Checking). This is to ensure that the fitted ARIMA model is statistically valid i.e., it meets the theoretical assumptions of the model. Even if the model fits well numerically, if its assumptions are violated (e.g., residuals still correlated), the model is statistically invalid; its parameters, forecasts, and confidence intervals become unreliable.

Second approach is **Forecast Evaluation** (Predictive Performance Assessment). This is to check how well the model generalizes to unseen data i.e., how accurate the forecasts are. A model can pass all diagnostic tests but still perform poorly in forecasting due to overfitting or poor parameterization. This step ensures the model is practically useful, not just statistically sound.

The result contains detailed information for making important decisions regarding model performance.

5.4.3 Model Diagnostics (Statistical Adequacy Checking)

AIC/BIC/HQIC:- These are statistical metrics used to evaluate and compare the goodness-of-fit of models), while penalizing model complexity to prevent overfitting. There is no benchmark to compare against any expected values. Hence, the application is to compare between models. As the model quality improves, it should exhibit lower values.

Coefficients:-

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.9556	0.107	8.943	0.000	0.746	1.165
ar.L2	-0.5164	0.087	-5.954	0.000	-0.686	-0.346
ma.L1	-0.8169	0.068	-11.978	0.000	-0.951	-0.683
sigma2	0.9306	0.111	8.348	0.000	0.712	1.149

The coefficient table provides the two coefficients of Auto Regressive (*ar.L1* and *ar.L2*) and Moving Average (*ma.L1*) terms under 'coef'. Std.err, z and P values are useful for testing hypothesis for significance of coefficients.

Null Hypothesis H_0: ar.L1 = 0
Alternate Hypothesis H_1: ar.L1 # 0

Since the p<=significance level, we will reject H_0 to conclude that *ar.L1* is significant.

[0.025 0.975]:- The values under [0.025 0.975] represent the **95% Confidence Interval (CI)** for the corresponding coefficient. We are 95% confident that the true

coefficient of *ar.L1* lies somewhere between the lower bound (the value under 0.025) and the upper bound (the value under 0.975).

sigma2:-The σ^2 term represents the variance of the residuals(ϵ_t). It quantifies the noise or unexplained variability in the time series. The hypothesis testing connected to σ^2 is about testing whether the true variance is statistically different from zero.

In practice, it is mostly observed to be significant, indicating that the model has a non-zero amount of residual variance. The primary concern is usually not the significance of σ^2 itself, but rather its magnitude (how much unexplained variance remains). This may be compared to the variance of the original time series data (Var(Y)) and should be as low as possible. Hence, it may be used to compare different models (like AIC and BIC) using the same data.

Ljung-Box (L1) (Q):

The Ljung-Box test for autocorrelation assumption that residuals are not autocorrelated.

Null Hypothesis H_0 : The residuals are independently distributed.
Alternative Hypothesis H_1: The residuals are not independently distributed; they exhibit serial correlation.

Since the Prob(Q)>significance level, we will accept **H_0** to conclude that residuals doesn't exhibit any autocorrelation.

Jarque-Bera (JB):

Jarque-Bera tests for normality assumption of residuals.

Null Hypothesis H_0:- Residuals are Normally distributed
Alternative Hypothesis H_1:- Residuals are Not Normally distributed.

Since the Prob(JB)>significance level, we will accept **H_0** to conclude that residuals are normally distributed.

Heteroskedasticity (H):

Heteroskedasticity (H) tests for homoscedasticity assumption (constant variance) of residuals.

Null Hypothesis H_0:- No heteroskedasticity
Alternative Hypothesis H_1:- There is heteroskedasticity

Since the Prob(H)>significance level, we will accept **H_0** to conclude that residuals are not heteroscedastic (homoscedastic).

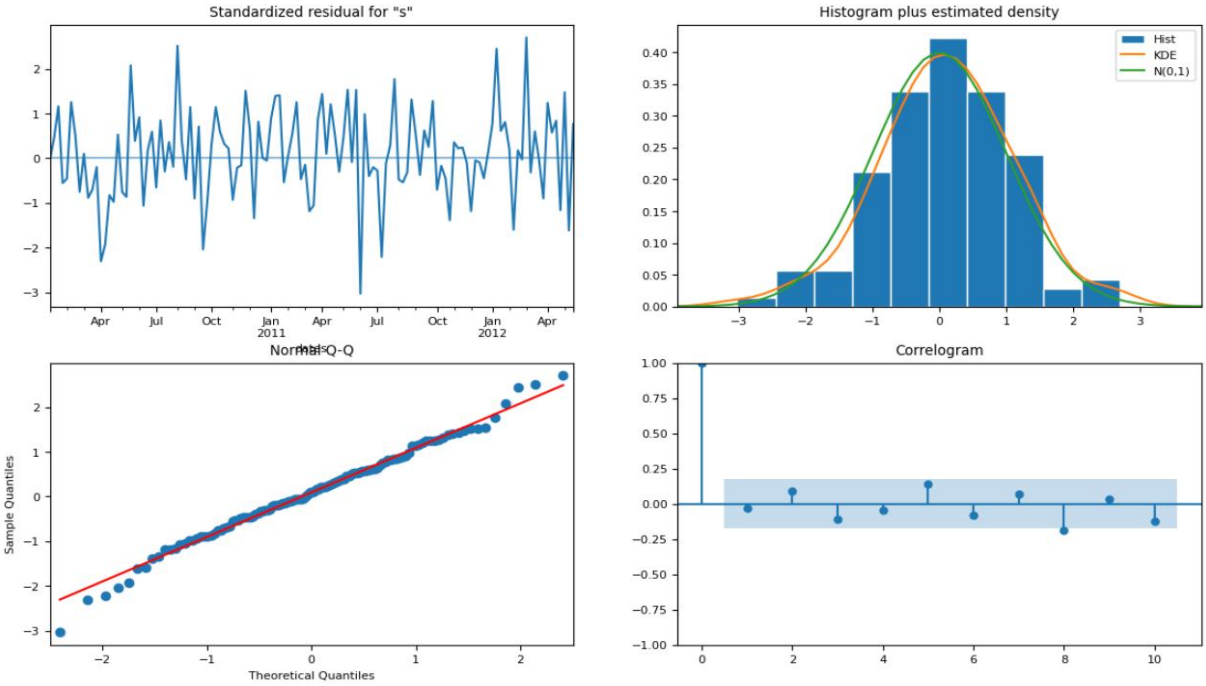
Skew and Kurtosis:-

The value of skew for the case is -0.14 indicating minor negative skewness (since the value is close to zero, it is very much normally distributed).

The value kurtosis of 3.47 indicates that it is slightly leptokurtic (note that kurtosis value for a normal distribution is 3).

Model diagnostics section is closed with a visual evaluation of behaviour of residuals. ARIMA class got *plot_diagnostics* method that provides these visuals provided below.

```
plt.rcParams.update({'font.size': 8})
results.plot_diagnostics(figsize=(14,8))
plt.show()
```



Standardized Residuals Plot (Top Left): Shows the residuals over time. It should look like random scatter around zero, with no trends, cycles, or bursts of volatility (heteroscedasticity).

Histogram and KDE (Top Right): Compares the distribution of the residuals against a theoretical normal distribution (the dotted red line). This is a visual check for the **normality assumption** (related to Skewness and Kurtosis). In an ideal situation, it should coincide normal distribution.

Normal Q-Q Plot (Bottom Left): Plots the residuals' quantiles against the quantiles of a standard normal distribution. For normal residuals, the points should fall closely along the straight diagonal line. This is another check for normality.

Correlogram/ACF Plot (Bottom Right): Plots the **Autocorrelation Function (ACF)** of the residuals. For a well-specified ARIMA model, the residuals should be uncorrelated, meaning virtually all spikes should fall within the blue shaded area (the confidence bounds), confirming the **white noise assumption**.

5.4.4 Forecast Evaluation (Predictive Performance Assessment)

The next step in the model building process is to evaluate the model performance on Train and Test data. For this purpose, two appropriate methods (*predict* and *forecast*) of fitted model object (*results*) are available.

results.predict method:- For any time point t within the specified range, the model computes the predicted value $\hat{Y}_t$ using the actual observed value from the previous time step, Y_{t-1}.

$$\hat{Y}_t = \text{constant} + \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \ldots. \theta_1 \varepsilon_{t-1} + \ldots..$$

This is useful for seeing how well the model **fits** the past data, as it is always given the "correct" recent history. Hence, *predict* defaults to using the actual historical data point by point, essentially providing a fit.

results.forecast method:- The *forecast* method is a specialized version of *predict* designed for **out-of-sample** (future) forecasting. It implicitly enforces the dynamic nature of prediction:

Step-1 (T+1):- The model predicts the first future step, T+1, using the last available observed data point, Y_T.

Step-2 (T+2):- For the second step, T+2, the model cannot use the observed value Y_{T+1} (because it hasn't happened yet). Instead, it uses its own prediction from the previous step, Y_{T+1}, as the input for the AR component.

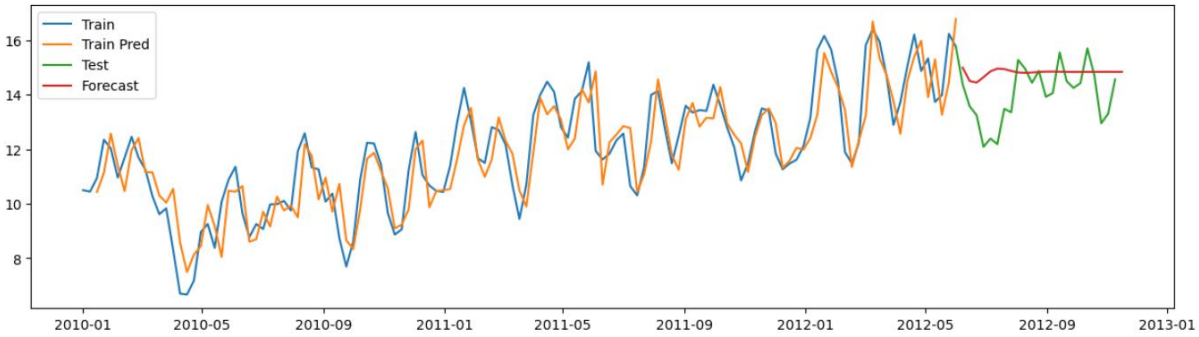
$$Y_{T+2} = \text{constant} + \phi_1 Y_{T+1} + \phi_2 Y_T + \ldots\ldots$$

Subsequent Steps:- This process continues. For forecast(24), the prediction for the 24th step will use the prediction from the 23rd step, the prediction from the 22nd step, and so on, until it hits the boundary of the observed data (Y_T).

Hence, forecast is a fully dynamic prediction where, for future time points, the model relies entirely on its own projected outputs to feed its AR components, accurately simulating real-world forecasting.

```
train_pred = results.predict(start=train.index[2], end=train.index[-1])
test_pred = results.forecast(24)
```

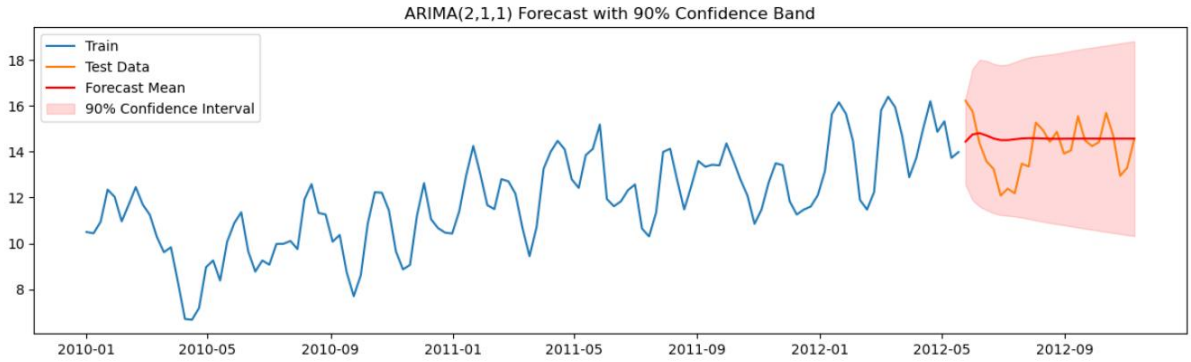
The performance is evaluated visually first and then using quantitative metrics.



The figure shows that predicted values of Train is close to the observed data as expected (predicted values are based on previous observed values). Forecasting of the unseen data (Test) shows the forecasted values with some error.

The figure below shows the visualization of 90% confidence interval band around forecasted values.

```
forecast_result = results.get_forecast(steps=len(test), alpha=0.1)
test_pred = forecast_result.predicted_mean
conf_int = forecast_result.conf_int()
# returns DataFrame with lower and upper bounds
lower_ci = conf_int.iloc[:, 0]
upper_ci = conf_int.iloc[:, 1]
```



Quantitative Metrics:-

	MAE	MAPE(%)	RMSE
Train	0.83	7.33	1.34
Test	0.93	6.86	1.17

Note that performance of the forecast is quite similar between Train and Test dataset. In general, RMSE is commonly used for evaluation in the industry/ business setting. This can be

interpreted as the standard deviation of error distribution. For an unbiased model, the errors would be normally distributed with mean 0 and standard deviation RMSE. Hence, properties of normal distribution can be used for interpretation of RMSE.

- For the test data, 68% of observations got errors within -1.17 and 1.17.
- 87% of observations got errors within -1.75 and 1.75.

5.5 ARIMA with Seasonality

SARIMA, or Seasonal AutoRegressive Integrated Moving Average, is the standard statistical model used to forecast time series data that exhibits both non-stationary and seasonal patterns. It extends the non-seasonal ARIMA model by adding specific terms to account for the seasonality observed in the data.

The complete SARIMA model is denoted as:

$$\text{SARIMA}(p,d,q)\times(P,D,Q)_m$$

Here (p,d,q) is the Non-Seasonal Order (familiar ARIMA components) and $(P,D,Q)_m$ is the Seasonal Order (added components for seasonality). These terms specifically capture the correlation and pattern that repeats every m periods.

P (Seasonal AR Order): The number of seasonal lags (e.g., lag 12 for monthly data) to include in the model.

D (Seasonal Differencing): The number of times the data is seasonally differenced to remove the seasonal pattern and achieve stationarity.

Q (Seasonal MA Order): The number of seasonal lagged forecast errors in the model.

m (Seasonal Period): The number of periods in a season (e.g., m=12 for monthly data, m=4 for quarterly data).

5.5.1 Modeling Process for SARIMA

The general procedure for using SARIMA involves identification, estimation, and diagnostic checking.

Data Preprocessing and Visualization: Plot the data to visually confirm the existence of a trend and a recurring seasonal pattern. Determine the Seasonal Period (m): Identify the number of observations in one cycle (e.g., 12 months, 7 days, 4 quarters).

Plot ACF and PACF function to develop an insight into seasonality of data. Note that this step is more of a cross check as data source should provide the expected seasonality (weather dependent process would have 12 month seasonality).

Automated Model Identification and Estimation: (auto_arima step) This step replaces the stationarity test and manual determination of paaramters (p,q,P,Q) using ACF/PACF plots. Call auto_arima and pass the time series data and the determined seasonal period (m) to the function.

m (Seasonal Period): Must be set (e.g., m=12).

stepwise=True (Recommended): Uses a faster, heuristic approach to search the parameter space, reducing computation time compared to an exhaustive search.

seasonal=True: Enables the search for seasonal orders (P,D,Q).

trace=True: Allows you to see the models being tested and their corresponding AIC/BIC scores.

information_criterion (e.g., 'aic', 'bic'): Specifies the metric used to rank and select the best model.

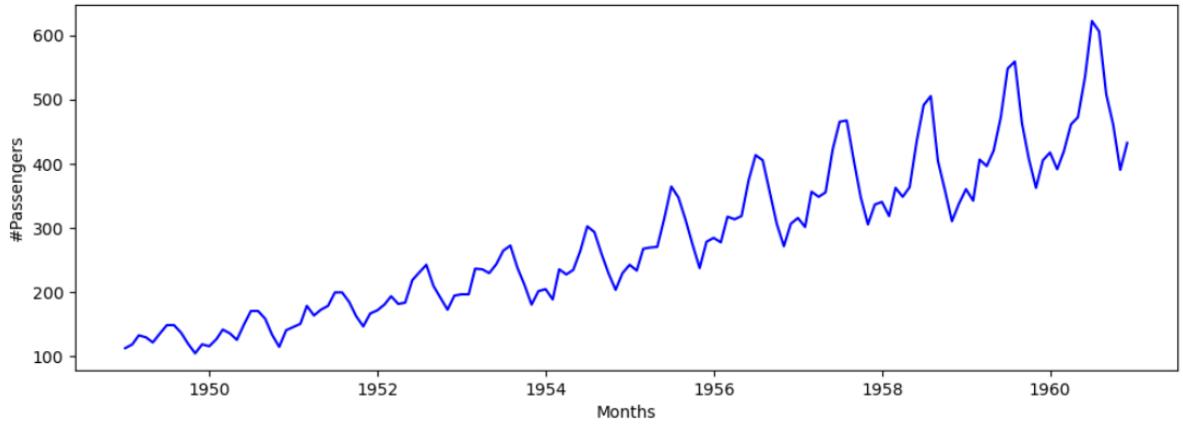
Output: The function returns the single best-fitting SARIMA model, automatically providing the optimal $(p,d,q)\times(P,D,Q)_m$ parameters.

Diagnostic Checking and Forecasting: The automatic identification doesn't guarantee a perfect model, so diagnostic checks are still essential.

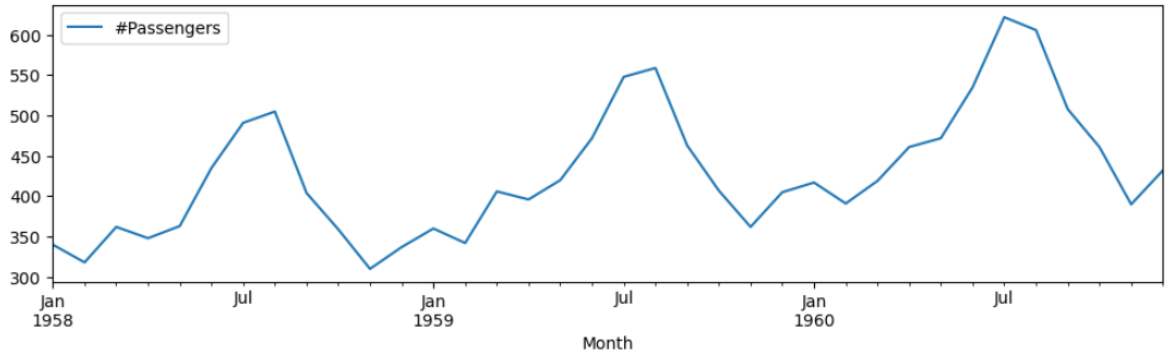
Forecasting: Use the final optimized SARIMA model to generate future predictions.

5.5.2 Case-2:- Building a model to forecast air passenger for 24 months

The data provides the number of air passengers for 144 months and the objective is to build a forecasting model for 24 months.



The plot shows distinct trend and seasonality (note the similar pattern repeats every 12-months). It can be observed that seasonal component too is increasing gradually indicating non-stationarity at Seasonal level.

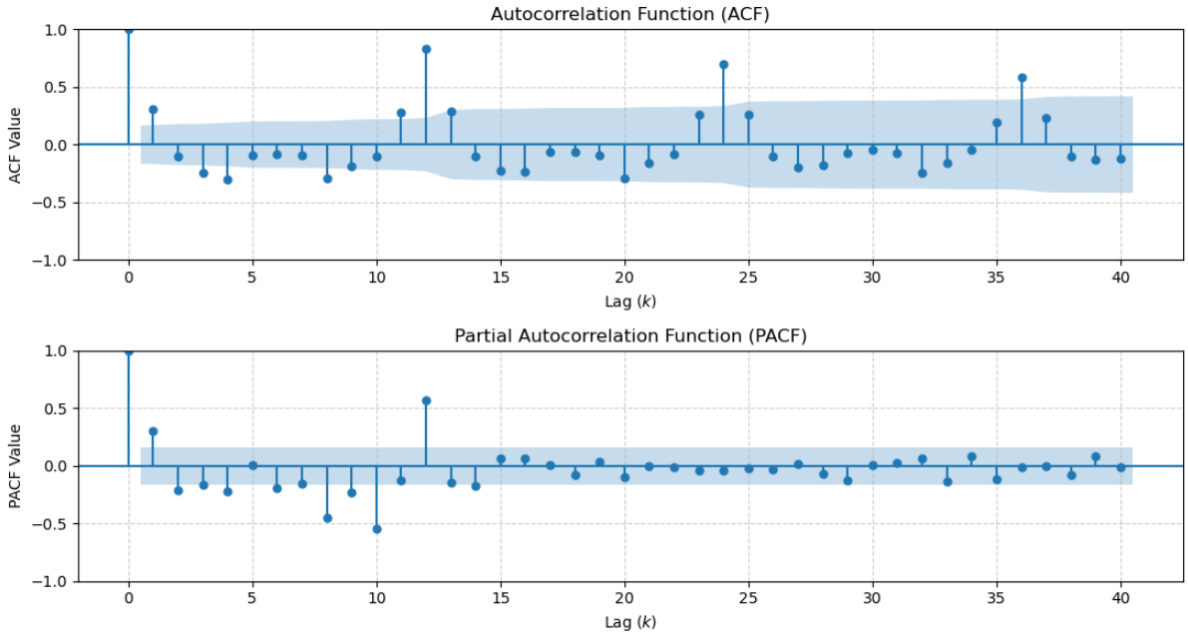


The closeup of 3 years data shows that the number of passengers spike every year during July every year.

ACF and PACF plots are provided below to cross-check the presence of seasonality. The

```
df1d = df['#Passengers'].diff()
df1d = df1d.dropna()

fig, axes = plt.subplots(2, 1, figsize=(11, 6), sharex=False)
plot_acf(df1d, ax=axes[0], lags=40, alpha=0.05, title='ACF')
axes[0].set_xlabel('Lag ($k$)')
axes[0].set_ylabel('ACF Value')
axes[0].grid(True, linestyle='--', alpha=0.6)

plot_pacf(df1d, ax=axes[1], lags=40, alpha=0.05, method='ywmle', title='PACF')
axes[1].set_xlabel('Lag ($k$)')
axes[1].set_ylabel('PACF Value')
axes[1].grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()
```

time series is made first order stationary by taking a one period difference.



Both ACF and PACF functions exhibit significant spikes at lag 12 indicating seasonality of period 12 (m=12).

Automated model identification and estimation:-

The data is split into Train and Test in preparation for developing a model. Since the objective is to forecast for 24 months, latest 24 months is selected as Test and the rest as Train.

```
train = df[:120]
test = df[120:]
```

For automated model building, *auto_arima* class is used as shown below.

```
from pmdarima import auto_arima
auto_model = auto_arima(train,
    start_p=0, d=None,  start_q=0,
    max_p=2,   max_d=2, max_q=2,
    start_P=0, D=None,  start_Q=0,
    max_P=2,   max_D=2, max_Q=2,
    max_order=6, m=12, stepwise=True, trace=True)
/****/
 Performing stepwise search to minimize aic
  ARIMA(0,0,0)(0,1,0)[12] intercept   : AIC=910.236, Time=0.12 sec
  ARIMA(1,0,0)(1,1,0)[12] intercept   : AIC=810.646, Time=0.37 sec
  ARIMA(0,0,1)(0,1,1)[12] intercept   : AIC=860.575, Time=0.31 sec
  ARIMA(0,0,0)(0,1,0)[12]             : AIC=1060.479, Time=0.04 sec
  ARIMA(1,0,0)(0,1,0)[12] intercept   : AIC=809.411, Time=0.14 sec
  ARIMA(1,0,0)(0,1,1)[12] intercept   : AIC=810.721, Time=0.29 sec
  ARIMA(1,0,0)(1,1,1)[12] intercept   : AIC=812.509, Time=0.74 sec
  ARIMA(2,0,0)(0,1,0)[12] intercept   : AIC=808.863, Time=0.16 sec
  ARIMA(2,0,0)(1,1,0)[12] intercept   : AIC=810.482, Time=0.43 sec
  ARIMA(2,0,0)(0,1,1)[12] intercept   : AIC=810.553, Time=0.35 sec
  ARIMA(2,0,0)(1,1,1)[12] intercept   : AIC=inf, Time=0.96 sec
  ARIMA(2,0,1)(0,1,0)[12] intercept   : AIC=809.672, Time=0.24 sec
  ARIMA(1,0,1)(0,1,0)[12] intercept   : AIC=809.448, Time=0.14 sec
  ARIMA(2,0,0)(0,1,0)[12]             : AIC=812.240, Time=0.07 sec

 Best model:  ARIMA(2,0,0)(0,1,0)[12] intercept
```

The objective of setting *d*=None and *D*=None in the *auto_arima* function is to instruct the algorithm to automatically determine the optimal differencing orders (including no differencing) for both the non-seasonal and seasonal components, respectively.

The *max_order* parameter imposes a constraint on the combined non-seasonal order and the combined seasonal order ie. $p+q+P+Q<=max\_order$. Note that for all the combinations evaluated *max_order* condition is met.

The optimal model is estimated using the parameters identified by *auto_arima*. A term trend='c' is required as optimal model parameters included an intercept.

```
model = SARIMAX(train, order=(2, 0, 0), seasonal_order=(0,1,0,12), trend='c')
results = model.fit()
results.summary()

/****/
```

SARIMAX Results

Dep. Variable:	#Passengers	**No. Observations:**	120
Model:	SARIMAX(2, 0, 0)x(0, 1, 0, 12)	**Log Likelihood**	-400.431
Date:	Thu, 30 Oct 2025	**AIC**	808.863
Time:	13:10:22	**BIC**	819.592
Sample:	01-01-1949	**HQIC**	813.213
	- 12-01-1958		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	4.2859	2.035	2.106	0.035	0.297	8.275
ar.L1	0.6783	0.100	6.816	0.000	0.483	0.873
ar.L2	0.1550	0.096	1.609	0.108	-0.034	0.344
sigma2	96.2826	11.855	8.121	0.000	73.046	119.519

Ljung-Box (L1) (Q):	0.01	**Jarque-Bera (JB):**	1.64
Prob(Q):	0.93	**Prob(JB):**	0.44
Heteroskedasticity (H):	1.41	**Skew:**	0.02
Prob(H) (two-sided):	0.31	**Kurtosis:**	3.60

The result of model estimation show that *ar.L2* is not significant. This needs correction and a new model will be built after removing *ar.L2.* Hence, the command for model will be modified as (1,0,0).

> How do we handle if *ar.L2* is significant and *ar.L1* is not? This can be handled by changing the command as ARIMA([2],0,0).

```
model = SARIMAX(train, order=(1, 0, 0), seasonal_order=(0,1,0,12), trend='c')
results = model.fit()
results.summary()

/****/
```

SARIMAX Results

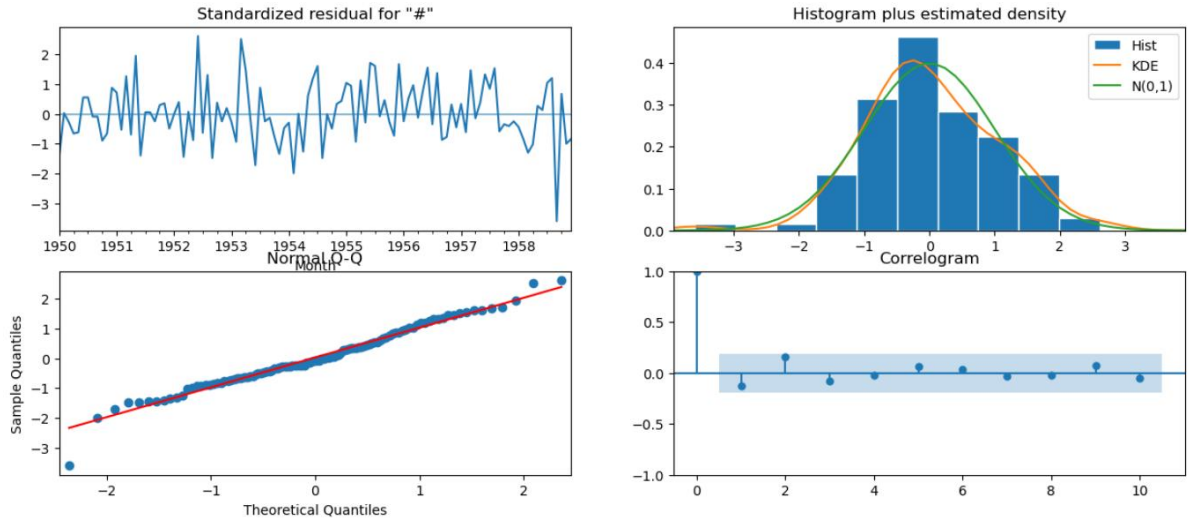
Dep. Variable:	#Passengers	**No. Observations:**	120
Model:	SARIMAX(1, 0, 0)x(0, 1, 0, 12)	**Log Likelihood**	-401.706
Date:	Thu, 30 Oct 2025	**AIC**	809.411
Time:	14:17:19	**BIC**	817.458
Sample:	01-01-1949	**HQIC**	812.674
	- 12-01-1958		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	5.3420	2.055	2.600	0.009	1.315	9.369
ar.L1	0.7981	0.066	12.088	0.000	0.669	0.928
sigma2	98.6491	11.688	8.440	0.000	75.741	121.557

Ljung-Box (L1) (Q):	1.85	**Jarque-Bera (JB):**	2.55
Prob(Q):	0.17	**Prob(JB):**	0.28
Heteroskedasticity (H):	1.29	**Skew:**	-0.06
Prob(H) (two-sided):	0.45	**Kurtosis:**	3.74

Now all coefficients are significant and the rest of model diagnostics can proceed. The results shows that residuals got no Autocorrelation (Ljung-Box L1), no Heteroskedasticity and normally distributed (Jarque-Bera).

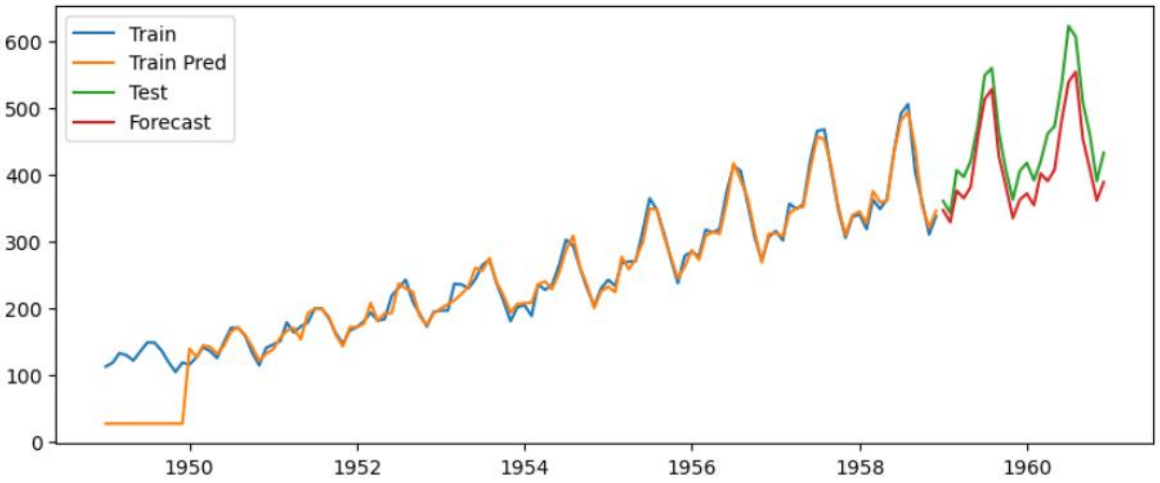
The behaviour of residuals is visualized below and it shows that residuals are evenly distributed around mean and the distribution is quite close to a normal distribution.



Forecast Evaluation (Predictive Performance Assessment)

The next step in the model building process is to evaluate the model performance on Train and Test data. For this purpose, two appropriate methods (*predict* and *forecast*) of fitted model object (*results*) are available.

```
train_pred = results.predict(start=train.index[2], end=train.index[-1])
test_pred = results.forecast(24)
```



Note that predicted values are almost coinciding observed values for the Train period indicating excellent fit. Test period shows that forecasted values are capturing seasonality and trend very well.

Quantitative Metrics:-

	MAE	MAPE(%)	RMSE
Train	17.1	0.11	33.34
Test	39.65	0.09	43.38

Note that performance of the forecast is quite similar between Train and Test dataset. RMSE can be interpreted as the standard deviation of error distribution and following interpretations can be drawn.

- For the test data, 68% of observations got errors within -43.38 and 43.38.
- 90% of observations got errors within -71.4 and 71.4.

5.6 ARIMA with Seasonality and External Variable

ARIMA is a powerful and widely used statistical model for time series forecasting. It is designed to capture patterns such as autocorrelation, trends, and short-term dependencies in data. However, many real-world time series are not purely autoregressive; they often exhibit seasonality (repeated patterns over fixed intervals) and are influenced by external factors (such as prices, holidays, or weather). Extension of ARIMA, namely SARIMAX, address these complexities.

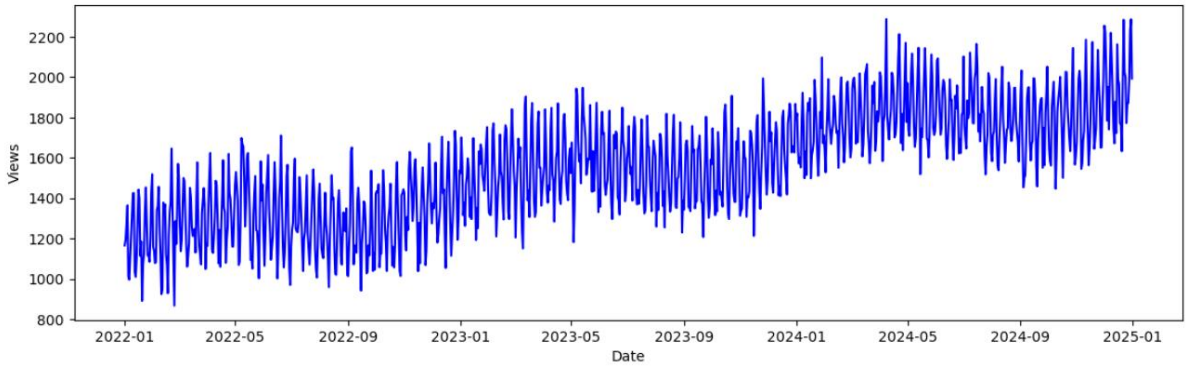
To use SARIMAX for forecasting, you must have or be able to accurately predict the future values of the exogenous variables (X_{future}). If you can't, the model can't generate a forecast.

5.6.1 Case-3:- Building a model with seasonality and external variables

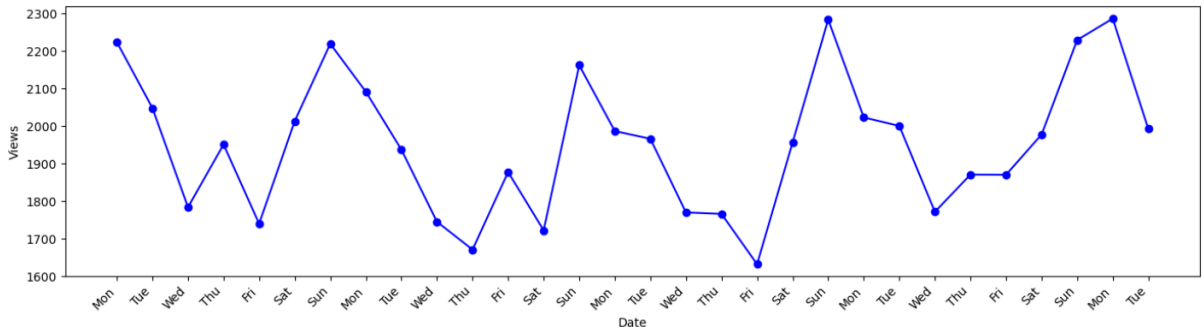
The data (Knight, Micheal (2023)) provides the page views for 1096 days and the objective is to build a forecasting model for 30 days. The table below shows the head() of dataframe. We will use 'page-views' as time series and 'ad_spend', 'weekend', 'day' and 'holiday' as external variables.

date	page_views	ad_spend	weekend	day	holiday
01 01 2022	1165.96277	194.9014	TRUE	Saturday	TRUE
02 01 2022	1191.48584	175.8521	TRUE	Sunday	FALSE
03 01 2022	1266.92827	269.4307	FALSE	Monday	FALSE
04 01 2022	1364.19386	295.6909	FALSE	Tuesday	FALSE
05 01 2022	1013.28252	242.9754	FALSE	Wednesday	FALSE

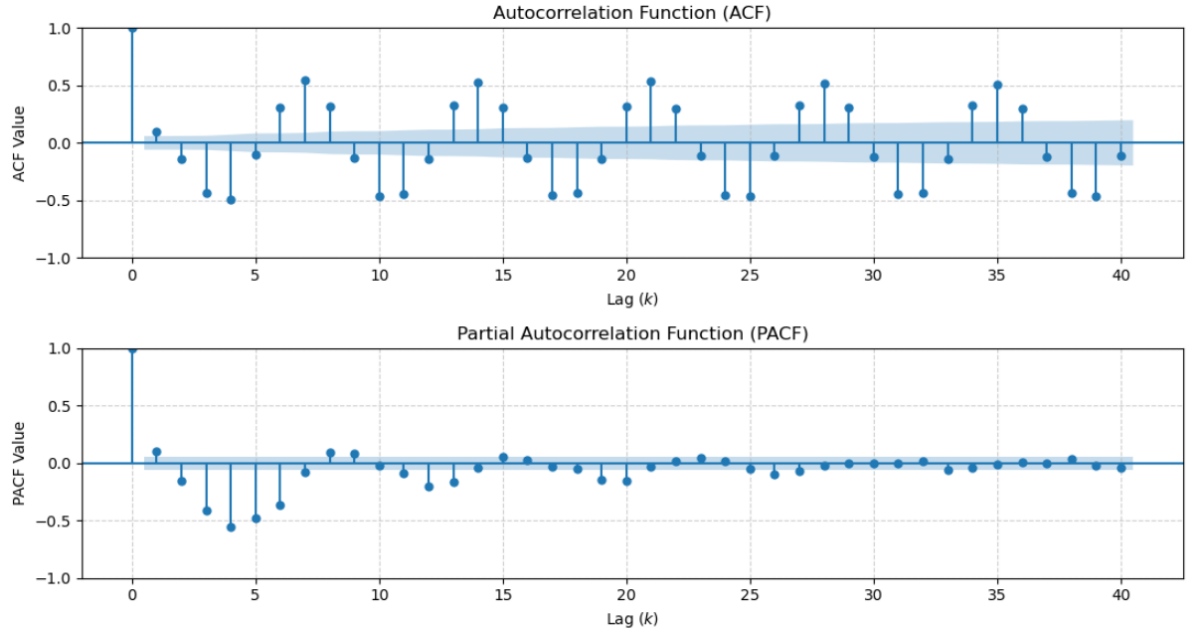
Plot of time series is shown below.



The plot shows distinct trend and seasonality. The closeup of 4 weeks (shown below) shows that the page views spike every week on Sunday and reach low on Friday and Wednesday.



The data is tested for non-stationarity and found that taking difference with one period removes it. This data was used to plot ACF and PACF functions to check the presence of seasonality.



Examining both ACF and PACF function, there seems to be seasonality with periods of 4 and 7. Since the data is at daily level, seasonality with period of 7 is quite logical (many behaviours repeat at weekly level like spikes on Sunday etc.). However, since the EDA is indicating 4 too, we will quantitatively determine the best seasonality.

Identify best period of seasonality: -

The data is split into Train and Test in preparation for developing a model. Since the objective is to forecast for 30 days, latest 30 days is selected as Test and the rest as Train.

First objective is to decide the period of seasonality. For this purpose, *auto_arima* class is used to tune the model and evaluate the performance.

```
train = df[:-30]
test = df[-30:]

from pmdarima import auto_arima
def arima_model(m_try):
    auto_model = auto_arima(y_train,
        start_p=0, d=None, start_q=0,
        max_p=2, max_d=2, max_q=2,
        start_P=0, D=None, start_Q=0,
        max_P=2, max_D=2, max_Q=2,
        max_order=6, m=m_try, stepwise=True, trace=False)

    train_pred = auto_model.predict_in_sample()
    test_pred = auto_model.predict(n_periods=30)

    rmse_train = np.sqrt(mean_squared_error(y_train, train_pred))
    rmse_test = np.sqrt(mean_squared_error(y_test, test_pred))
    print(auto_model.summary())
    print()
    print('Train RMSE:  ', rmse_train, 'm = ', m_try)
    print('Test RMSE:  ', rmse_test, 'm = ', m_try)

arima_model(7)

/****/
Train RMSE:   121.94325375591254 m =  7
Test RMSE:   292.94475324178785 m =  7
```

	m = 7	m = 4
Train	121.94	119.75
Test	292.94	155.54

Note that Test RMSE is lower for seasonality with period of 4. Now let's evaluate ARIMA model with External variables.

Automated model identification and estimation with external variables

Exactly as in the previous section, here we will be using auto_arima to tune SARIMA with External variables.

```
from pmdarima import auto_arima
def arima_model_x(m_try):
    auto_model = auto_arima(y_train, X=X_train,
        start_p=0, d=None, start_q=0,
        max_p=2, max_d=2, max_q=2,
        start_P=0, D=None, start_Q=0,
        max_P=2, max_D=2, max_Q=2,
        max_order=8, m=m_try, stepwise=True, trace=True,
information_criterion='bic', error_action='ignore')

    train_pred = auto_model.predict_in_sample(X=X_train)
    test_pred = auto_model.predict(n_periods=30, X=X_test)

    mse_train = root_mean_squared_error(y_train, train_pred)
    mse_test = root_mean_squared_error(y_test, test_pred)
    print(auto_model.summary())
    print()
    print('Train RMSE:  ', mse_train, 'm = ', m_try)
    print('Test RMSE:  ', mse_test, 'm = ', m_try)

arima_model_x(4)

/****/
 Best model:  ARIMA(1,1,1)(0,0,0)[4]
 Total fit time: 77.233 seconds
                               SARIMAX Results
 ==============================================================================
 Dep. Variable:                      y   No. Observations:                 1066
 Model:               SARIMAX(1, 1, 1)   Log Likelihood               -6279.703
 Date:                Fri, 31 Oct 2025   AIC                          12585.407
 Time:                        21:41:06   BIC                          12650.026
 Sample:                    01-01-2022   HQIC                         12609.892
                          - 12-01-2024
 Covariance Type:                  opg
 ================================================================================
                     coef    std err          z      P>|z|      [0.025      0.975]
 --------------------------------------------------------------------------------
 ad_spend          0.2434      0.041      5.937      0.000       0.163       0.324
 weekend          78.3550   8101.168      0.010      0.992   -1.58e+04    1.6e+04
 holiday         -61.6328     33.333     -1.849      0.064    -126.963       3.698
 day_Friday     -125.3254   1.22e+04     -0.010      0.992   -2.39e+04   2.37e+04
 day_Monday      231.7145   1.22e+04      0.019      0.985   -2.36e+04    2.4e+04
 day_Saturday    -34.1769   4051.025     -0.008      0.993   -7974.040   7905.687
 day_Sunday      112.5320   4050.202      0.028      0.978   -7825.718   8050.782
 day_Thursday   -158.5640   1.22e+04     -0.013      0.990    -2.4e+04   2.37e+04
 day_Tuesday      78.5259   1.22e+04      0.006      0.995   -2.37e+04   2.39e+04
 day_Wednesday  -104.7068   1.22e+04     -0.009      0.993   -2.39e+04   2.37e+04
 ar.L1             0.1076      0.034      3.212      0.001       0.042       0.173
 ma.L1            -0.9140      0.014    -65.317      0.000      -0.941      -0.887
 sigma2         7734.8534    356.563     21.693      0.000    7036.003    8433.704
 ===================================================================================
 Ljung-Box (L1) (Q):                   0.06   Jarque-Bera (JB):                 2.18
 Prob(Q):                              0.80   Prob(JB):                         0.34
 Heteroskedasticity (H):               0.84   Skew:                             0.01
 Prob(H) (two-sided):                  0.11   Kurtosis:                         2.78
 ===================================================================================
```

Note that among external variables, only ad_spend is significant. Hence model will be redeveloped with only ad_spend as external variable. Seasonality with period of 4 and 7 was examined again.

	m = 7	**m = 4**
Train	121.96	117.75
Test	297.09	164.17

Note that the model with seasonal period=4 is better than period=7. Hence, we will finalize this model and proceed with model diagnostics and performance evaluation.

```
model = SARIMAX(y_train, exog=X_train, order=(2, 1, 1),
seasonal_order=(2,0,1,4))
results = model.fit(maxiter=1000)
results.summary()
/****/
```

SARIMAX Results

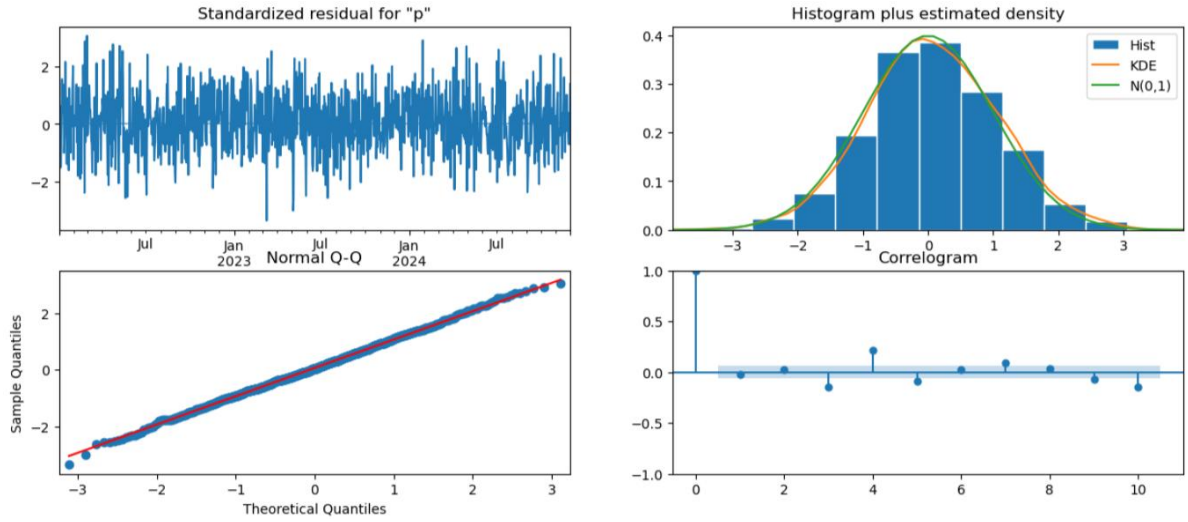
Dep. Variable:	page_views	**No. Observations:**	1066
Model:	SARIMAX(2, 1, 1)x(2, 0, 1, 4)	**Log Likelihood**	-6541.816
Date:	Fri, 31 Oct 2025	**AIC**	13099.632
Time:	15:27:07	**BIC**	13139.398
Sample:	01-01-2022	**HQIC**	13114.700
	- 12-01-2024		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ad_spend	0.3319	0.040	8.317	0.000	0.254	0.410
ar.L1	0.0873	0.041	2.134	0.033	0.007	0.167
ar.L2	-0.1133	0.036	-3.145	0.002	-0.184	-0.043
ma.L1	-0.8560	0.024	-35.129	0.000	-0.904	-0.808
ar.S.L4	-1.5734	0.025	-61.822	0.000	-1.623	-1.524
ar.S.L8	-0.7459	0.024	-31.605	0.000	-0.792	-0.700
ma.S.L4	0.8920	0.021	43.219	0.000	0.852	0.932
sigma2	1.258e+04	553.244	22.736	0.000	1.15e+04	1.37e+04

Ljung-Box (L1) (Q):	0.36	**Jarque-Bera (JB):**	0.11
Prob(Q):	0.55	**Prob(JB):**	0.95
Heteroskedasticity (H):	0.91	**Skew:**	0.02
Prob(H) (two-sided):	0.40	**Kurtosis:**	2.99

The result shows that all coefficients are significant. The residuals got no Autocorrelation (Ljung-Box L1), no Heteroskedasticity and normally distributed (Jarque-Bera).

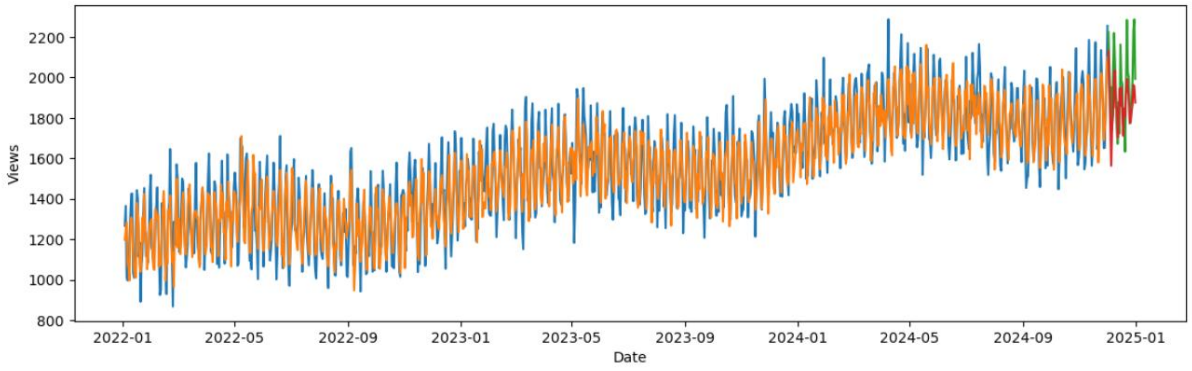
The behaviour of residuals is visualized below and it shows that residuals are evenly distributed around mean and the distribution is quite close to a normal distribution.



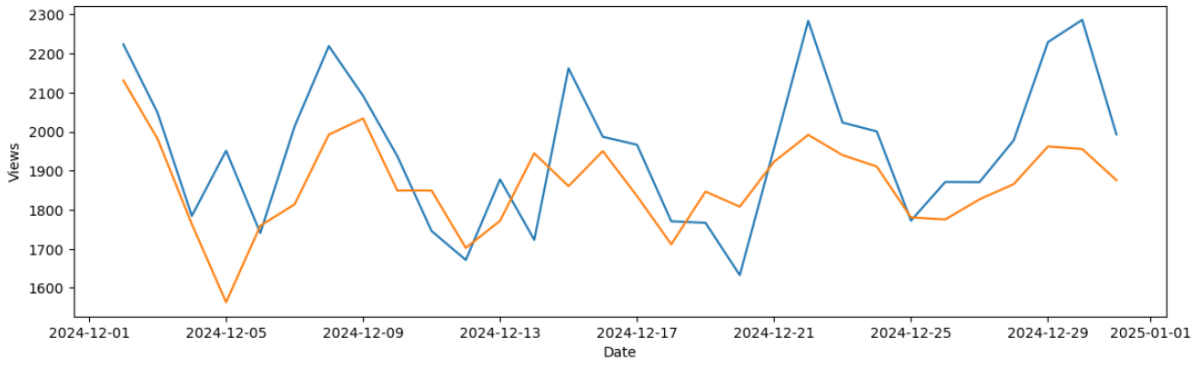
Forecast Evaluation (Predictive Performance Assessment)

The next step in the model building process is to evaluate the model performance on Train and Test data. For this purpose, two appropriate methods (*predict* and *forecast*) of fitted model object (*results*) are available.

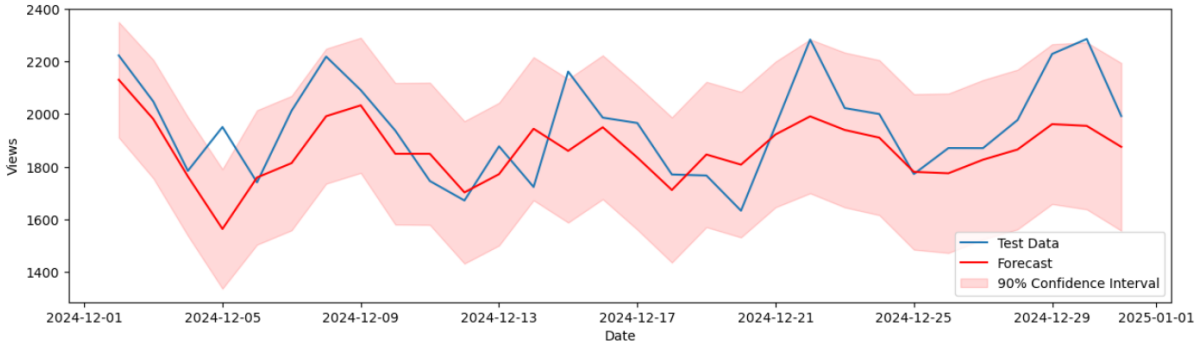
```
train_pred = results.predict(start=0, end=len(y_train)-1, exog=X_train)
test_pred = results.forecast(steps=30, exog=X_test)
```



Note that predicted values are almost coinciding observed values for the Train period indicating excellent fit. Test period shows that forecasted values are capturing the trend very well. Focused view of Test area is critical in this case and shown below.



The figure below shows forecast with 90% confidence interval.



Quantitative Metrics: -

	MAE	MAPE(%)	RMSE
Train	90.66	0.06	117.5
Test	129.33	0.06	164.18

Note that performance of the forecast is quite similar between Train and Test dataset. RMSE can be interpreted as the standard deviation of error distribution and following interpretations can be drawn.

- For the test data, 68% of observations got errors within -164.18 and 164.18.
- 90% of observations got errors within -270.9 and 270.9.

References:-

Yule, G. U. (1927). On a method of investigating periodicities in disturbed series, with special reference to Wolfer's sunspot numbers. Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character, 226(636–646), 267–298. https://doi.org/10.1098/rsta.1927.0007

Wold, H. (1938). A Study in the Analysis of Stationary Time Series. Almqvist & Wiksell, Uppsala.

Slutsky, E. (1937). The summation of random causes as the source of cyclic processes. Econometrica, 5(2), 105–146.

Walker, G. T. (1931). On periodicity in series of related terms. Proceedings of the Royal Society A, 131, 518–532.

Kendall, M. G. & Hill, A. B. (1953). The analysis of economic time-series—Part I: Prices. Journal of the Royal Statistical Society (Series A).

Box, G. E. P. & Jenkins, G. M. (1970). Time Series Analysis: Forecasting and Control. San Francisco: Holden-Day.

Hyndman, R. J., & Khandakar, Y. (2008). Automatic time series forecasting: The forecast package for R. Journal of Statistical Software, 27(3), 1–22. https://doi.org/10.18637/jss.v027.i03

Knight, Micheal (2023). "Personal Ecommerce Website Ad Cost and Viewer Count." Dataset. Kaggle. https://www.kaggle.com/datasets/michealknight/personal-ecommerce-website-ad-cost-and-viewer-count.

6.0 ARCH and GARCH Models

ARCH and GARCH models are specialized statistical tools used to analyze and forecast the **volatility** (or risk) of a time series, particularly in financial markets. Unlike traditional time series models like ARIMA, which focus on modeling the conditional **mean** (the expected value), ARCH and GARCH focus on modeling the conditional **variance** (the expected risk) of the errors.

The Core Concept: Conditional Heteroskedasticity

The motivation for these models comes from a phenomenon common in financial data, like stock returns, known as **Volatility Clustering**.

- **Volatility Clustering:** Large price changes (positive or negative, resulting in high volatility) tend to be followed by large price changes, and calm periods (low volatility) tend to be followed by calm periods. Volatility is not constant; it "clusters" over time.
- **Heteroskedasticity:** In statistics, this means the variance of the error term is **not constant** over time.
- **Conditional Heteroskedasticity:** This means the current variance is *conditional* on (i.e., depends on) information from the previous periods. This is what ARCH and GARCH explicitly model

6.1 Autoregressive Conditional Heteroskedasticity (ARCH)

The ARCH model, introduced by Robert Engle in 1982, was the first to formally capture time-varying volatility.

ARCH(q) Model

An ARCH(q) model states that the current conditional variance (σ_t^2) is dependent on the squared values of the previous q error terms (ϵ^2_{t-i}).

Equation (Conditional Variance):

$$\sigma_t^2 = \omega + \sum_{i=1}^{q} \alpha_i \varepsilon_{t-i}^2$$

σ_t^2: The conditional variance at time t (the model's prediction of volatility).

ω: A constant base volatility.

ϵ^2_{t-i}: The squared residuals (shocks/surprises) from the previous q periods. A large past squared error leads to a large current variance.

Analogy: The ARCH model acts like an AR(q) model applied to the squared errors. It says that current high volatility is a result of a direct shock in the recent past.

6.2 Generalized Autoregressive Conditional Heteroskedasticity (GARCH)

The GARCH model, introduced by Tim Bollerslev in 1986, is an extension of ARCH. It improves upon ARCH by allowing the current conditional variance to depend not only on past squared errors but also on past conditional variances themselves.

GARCH(*p*,*q*) Model

A GARCH(*p*,*q*) model includes both the autoregressive (ARCH) terms and a moving average (GARCH) component for the variance.

Equation (Conditional Variance):

$$\sigma_t^2 = \omega + \sum_{i=1}^{q} \alpha_i \varepsilon_{t-i}^2 + \sum_{j=1}^{p} \beta_j \sigma_{t-j}^2$$

$\alpha_i \epsilon^2_{t-i}$: The ARCH term (impact of past shocks/errors).

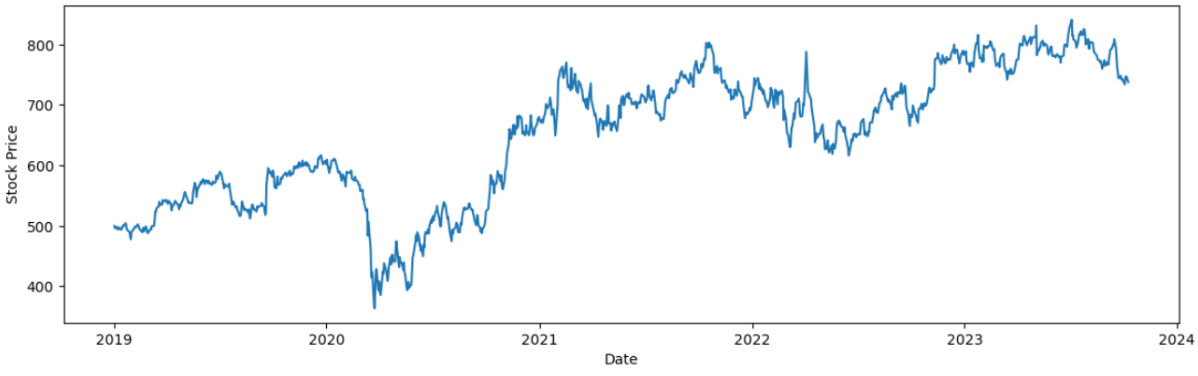
$\beta_j \sigma^2_{t-j}$: The GARCH term (impact of past forecasted variance).

Analogy: The GARCH model is the volatility-modeling counterpart to an ARMA model for returns. In a GARCH(*p*,*q*) process, yesterday's squared shock (the ARCH component) explains short-run spikes in volatility, while the lagged conditional variance (the GARCH component) allows volatility to remain elevated even when no new large shock occurs. It combines ARCH (short-term shocks via ϵ_{t-i}^2) with persistence (past variances σ_{t-j}^2).

Advantage over ARCH: GARCH(1,1) is often sufficient to model volatility clustering where a high-order ARCH(*q*) model would be required, leading to a much more parsimonious (simpler, fewer parameters) and stable model.

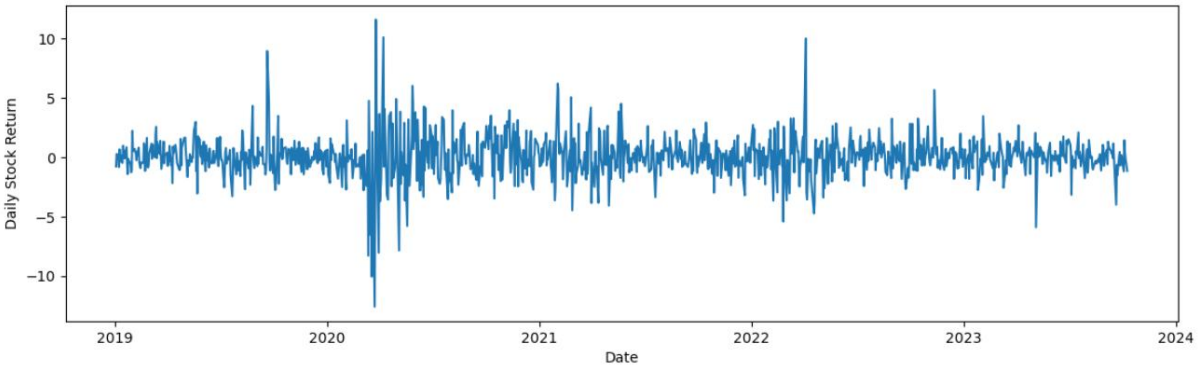
6.2.1 Case:- Estimate and predict return volatility of stock price

The data was read and plotted as a preliminary step. It is a daily stock price data from 2019-01-02 to 2023-10-09 (HDFC Bank).



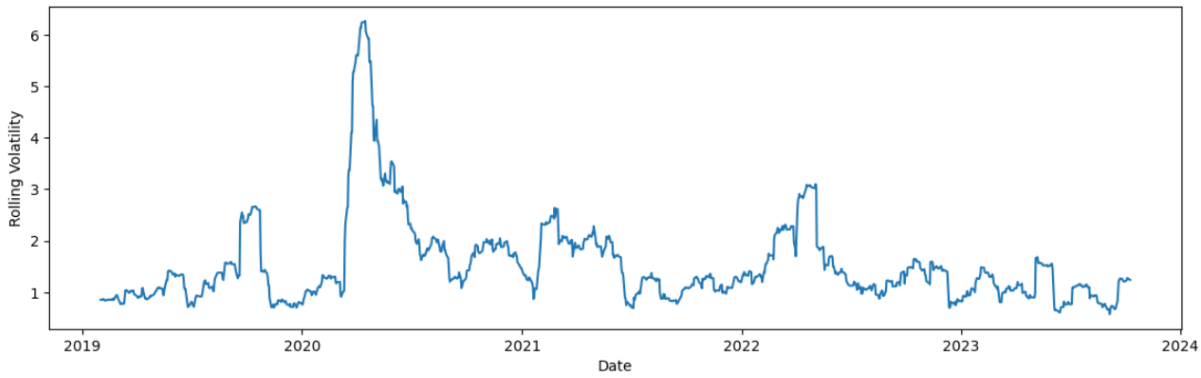
Plot of daily return is provided below.

```
df['return'] = 100 * (df['Close'].pct_change())
```



Volatility of daily stock return is calculated on a rolling basis of 21 days (approximately 1 trading month). Volatility plot is provided below.

```
rolling_vol = abs(df2['return'].rolling(window=21, min_periods=21).std())
df2['volatility'] = rolling_vol
df2 = df2.dropna()
```



The table below shows summary statistics of volatility. Daily volatility is the overall volatility of daily returns. Monthly and Annual volatility is calculated using the statistical formula.

Monthly_volatility = Daily_volatility*sqrt(21)
Annual_volatility = Daily_volatility*sqrt(252)

```
daily_volatility = df['return'].std()
monthly_volatility = math.sqrt(21) * daily_volatility
annual_volatility = math.sqrt(252) * daily_volatility
```

	Daily Volatility %	Monthly Volatility %	Annual Volatility %
HDFC	1.72	7.89	27.32

Identification of GARCH Model

First step of model identification is to establish the presence of ARCH. The *het_arch* function from *statsmodels.stats.diagnostic* performs the Engle's ARCH-LM (Lagrange Multiplier) test. This test checks for the presence of ARCH effects in the residuals of a time series model.

Null Hypothesis (H_0): There are no ARCH effects present (i.e., the conditional variance is constant, or the residuals are homoskedastic).

Alternative Hypothesis (H_1): ARCH effects are present (i.e., the conditional variance is time-varying, or the residuals are heteroskedastic).

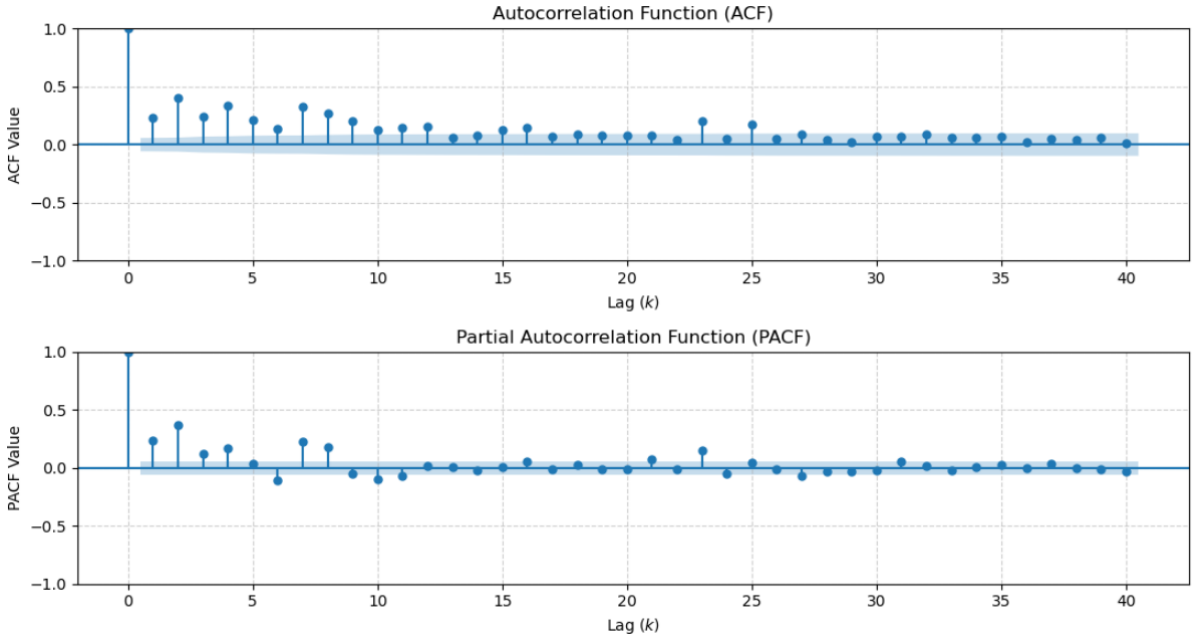
The process start with a model for the *mean* (ARIMA or constant mean) to obtain the residuals (ε_t).

Second step is to plot ACF and PACF functions of squared residuals (similar to ARIMA identification). ACF suggests order of MA(q) and PACF suggests order of AR(p).

Typically, the conclusion of a slowly decaying ACF is to include GARCH terms (q) and significant first few spikes of PACF is to include ARCH terms(q).

```
from statsmodels.tsa.arima.model import ARIMA
model = ARIMA(df['return'], order=(1,1,0))
results = model.fit()
residuals = results.resid

from statsmodels.stats.diagnostic import het_arch
arch_test = het_arch(residuals)
print('LM-pvalue:  ', arch_test[1])
/****/

LM-pvalue:   9.706412141362732e-70
```

Since p<=0.05, H_0 is rejected to accept alternative hypothesis to conclude that ARCH effects present.



The plots show a slowly decaying ACF indicating GARCH terms (p) and first few spikes of PACF indicating ARCH terms(q).

GARCH Estimation

It is widely reported that GARCH(1,1) is more than adequate for most prediction situations (Bollerslev et.al., 1992). Hence, no tuning is attempted to identify optimal p and q. *arch* library enables GARCH estimation as below.

```
garch_model = arch_model(df['return'], p = 1, q = 1, mean = 'constant', vol =
'GARCH', dist = 'normal')

gm_result = garch_model.fit(disp='off')
gm_result.summary()
/****/
```

Constant Mean - GARCH Model Results

Dep. Variable:	return	**R-squared:**	0.000
Mean Model:	Constant Mean	**Adj. R-squared:**	0.000
Vol Model:	GARCH	**Log-Likelihood:**	-2136.93
Distribution:	Normal	**AIC:**	4281.85
Method:	Maximum Likelihood	**BIC:**	4302.14
		No. Observations:	1178
Date:	Sat, Nov 01 2025	**Df Residuals:**	1177
Time:	15:08:18	**Df Model:**	1

Mean Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
mu	0.0517	4.250e-02	1.217	0.223	[-3.156e-02, 0.135]

Volatility Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	0.0946	5.794e-02	1.633	0.102	[-1.892e-02, 0.208]
alpha[1]	0.0989	4.971e-02	1.990	4.655e-02	[1.510e-03, 0.196]
beta[1]	0.8680	6.505e-02	13.342	1.317e-40	[0.740, 0.995]

Note that the coefficients alpha[1] and beta[1] are significant which is critical. The insignificance of mu doesn't necessarily harm the model's ability to capture volatility, as the GARCH model primarily focuses on the variance (σ_t^2). The mean model is secondary here. An insignificant omega is not uncommon in financial data, especially if $\alpha_1+\beta_1$ is close to 1 (high persistence), as most of the variance is driven by lagged terms. The model can still perform well for volatility forecasting.

μ(Mean): The μ parameter represents the mean or average value of the time series. It captures the baseline or expected level of the series, independent of volatility. In a GARCH model, the mean can be included to account for any non-zero mean or trend in the data.

ω(Constant Variance): The ω parameter represents the constant or unconditional variance component in the GARCH model. It reflects the long-term volatility level or the baseline volatility that is assumed to persist regardless of past observations. It is also referred to as the "archiving effect" or the "persistence of volatility."

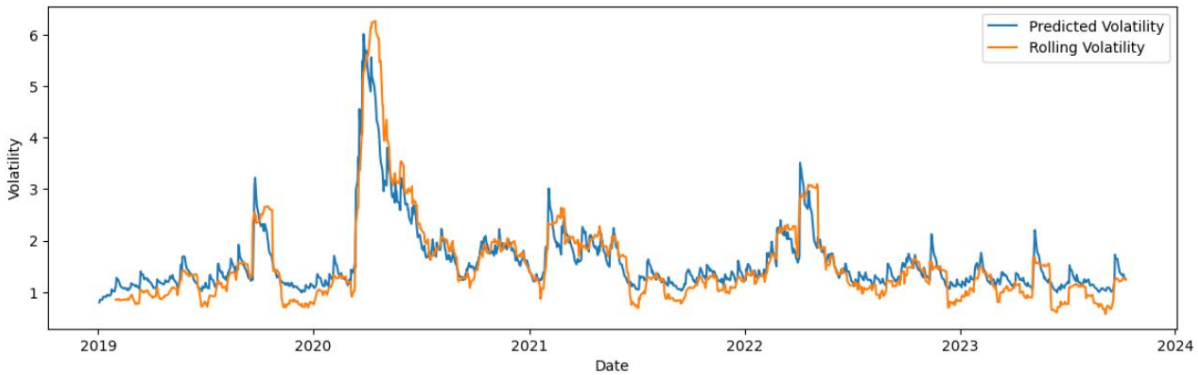
α(ARCH Coefficients): The α parameters (α1, α2, α3, etc.) are the coefficients associated with the autoregressive terms for the squared residuals in the GARCH model. They represent the impact or influence of the lagged squared residuals on the current volatility. Higher α values indicate a stronger impact of past squared residuals on the current volatility, implying that large past shocks have a more pronounced effect on volatility.

β(GARCH Coefficients): The β parameters (β1, β2, β3, etc.) are the coefficients associated with the moving average terms for the conditional variance in the GARCH model. They represent the impact or influence of the lagged conditional variances on the current volatility. Higher β values indicate a stronger influence of past conditional variances on the current volatility, suggesting that the persistence of volatility is driven by previous volatility levels.

GARCH Estimation of Volatility

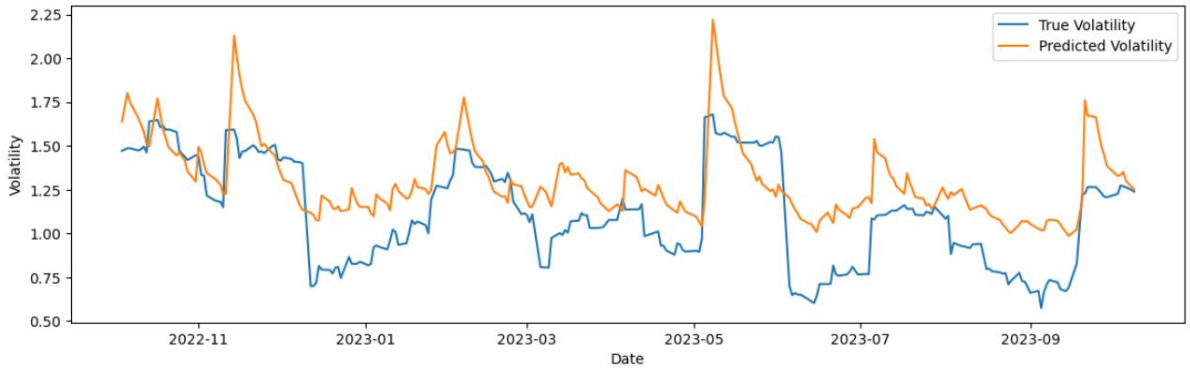
The estimated GARCH model can be used to predict volatility and this can be compared with actual values for make an evaluation of model quality. The plot below shows the actual and prediction of volatility.

```
vol_est = pd.DataFrame(gm_result.conditional_volatility)
rolling_vol = abs(df['return'].rolling(window=21,
min_periods=21).std().dropna())
garch_and_rolling_std = pd.concat([vol_est,rolling_vol], axis=1).dropna()
```



The figure shows that predicted volatility almost coincides with actual rolling volatility. Note that model estimation used entire data and hence it is an in-sample prediction.

The plot below shows rolling one day ahead forecast. Here a GARCH(1,1) fitted on the training sample up to time t-1 and volatility for time t is forecasted. This mimics a real-time forecast process, only information up to that date is available.



References:-

Engle, Robert F. (1982). "Autoregressive Conditional Heteroskedasticity with Estimates of the Variance of United Kingdom Inflation." Econometrica, 50(4), 987–1007.

Bollerslev, Tim (1986). "Generalized Autoregressive Conditional Heteroskedasticity." Journal of Econometrics, 31(3), 307–327.

7.0 Miscellaneous

7.1 Resources

There is a pdf version of this booklet available at https://github.com/regivm/ALittleBookofTimeSerieswithPython/blob/main/A_Little_Book_of_Time_Series_with_Python.pdf

Associated Python codes can be accessed at the below location.
https://github.com/regivm/ALittleBookofTimeSerieswithPython

7.2 Contact

I will be grateful if you can send me (Regi Mathew) corrections or suggestions for improvements to my email address regivm@yahoo.com