# IMPLEMENTING AND COMPARING FORECASTING ALGORITHMS FOR SALES DATA USING PYTHON: FACEBOOK PROPHET, SARIMA AND HOLT-WINTERS

**Thesis submitted**

**In Partial Fulfillment of the Requirements for the**

**Degree of**

**MASTER OF TECHNOLOGY**

**in**

**Industrial Engineering and Management**

**by**

**Samir Kumar**

**(Roll No. 2K22/IEM/09)**

**Under the Supervision of**

Prof. Suresh Kumar Garg

**Professor, Department of Mechanical Engineering**

**Delhi Technological University**

**To the**

**Department of Mechanical Engineering**

**DELHI TECHNOLOGICAL UNIVERSITY**

**(Formerly Delhi College of Engineering)**
**Shahbad Daulatpur, Main Bawana Road, Delhi-110042, India**

**June, 2024**

**DEPARTMENT OF MECHANICAL ENGINEERING**
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

## ACKNOWLEDGEMENT

I want to convey my heartfelt gratitude to my professor, **Prof. S. K Garg**, for their support and encouragement during the research and writing of this project. His expertise in the subject matter greatly contributed to the depth and quality of the project He showed us the path to achieve our targets by explaining all the tasks to be done and explained to us the importance of this project as well as its industrial relevance. He was always ready to help us and clear our doubts regarding any hurdles in this project. Without his constant support and motivation, this project would not have been successful.

I would also like to extend my heartfelt appreciation to **Prof. B. B. Arora**, Head of the Department of Mechanical Engineering, for his support and encouragement. His vision and leadership have provided a conducive environment for academic and research pursuits.

Lastly, I would like to acknowledge my family and friends for their unwavering support, encouragement, and understanding throughout this research endeavour.

Place: Delhi Samir Kumar

Date: (2K22/IEM/09)

**DELHI TECHNOLOGICAL UNIVERSITY**
(Formerly Delhi College of Engineering)
Shahbad Daulatpur, Main Bawana Road, Delhi-42

## CANDIDATE'S DECLARATION

I Samir Kumar (2K22/IEM/09) hereby certify that the work which is being presented in the thesis entitled "**Implementing and Comparing Forecasting Algorithms for Sales Data Using Python: Facebook Prophet, SARIMA, and Holt-Winters**" in partial fulfillment of the requirements for the award of the Degree of Master of Technology , submitted in the Department of Mechanical Engineering , Delhi Technological University is an authentic record of my own work carried out during the period from January, 2024 to June 2024 under the supervision of Prof. Suresh Kumar Garg.

The matter presented in the thesis has not been submitted by me for the award of any other degree of this or any other Institute.

**Candidate's Signature**

**DELHI TECHNOLOGICAL UNIVERSITY**
(Formerly Delhi College of Engineering)
Shahbad Daulatpur, Main Bawana Road, Delhi-42

## CERTIFICATE BY THE SUPERVISOR

Certified that Samir Kumar (2K22/IEM/09) has carried out their search work presented in this thesis entitled “**Implementing and Comparing Forecasting Algorithms for Sales Data Using Python: Facebook Prophet, SARIMA, and Holt-Winters**” for the award of Master of Technology from Department of Mechanical Engineering, Delhi Technological University, Delhi, under my supervision. The thesis embodies results of original work, and studies are carried out by the student himself and the contents of the thesis do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

Date: Signature

**Prof. Suresh Kumar Garg**
Professor

**Department of Mechanical Engineering**

**Delhi Technological University, Delhi**

# Implementing and Comparing Forecasting Algorithms for Sales Data Using Python: Facebook Prophet, SARIMA, and Holt-Winters

Samir Kumar

## ABSTRACT

The primary objective was to compare the performance of Facebook Prophet, a popular ML algorithm, against established statistical models like SARIMA and Holt-Winters. This study focused on analyzing time-series sales data, acknowledging the significant influence of seasonality on sales patterns. A Python script has been used as a central tool, enabling the evaluation of various forecasting algorithms. This compared Facebook Prophet, SARIMA, and Holt-Winters based on their accuracy, robustness, and applicability to the data. This involved implementing each algorithm in the script and analyzing their performance metrics, such as R-squared scores. The analysis revealed that Facebook Prophet emerged as the most accurate model for the dataset and chosen error metrics. It obtained an R-squared score of 0.94 for fitted data (the best result achieved by using simulated annealing for fine tuning), demonstrating its strong ability to capture the underlying patterns in the sales data. While Holt-Winters performed competitively with an R-squared value of 0.87 (used simulated Annealing in fine tuning) and 0.87 (used simulated annealing in fine tuning) on the training set, it is important to note that these results might vary depending on specific data and evaluation criteria. Interestingly, the remaining algorithms showed relatively minor performance differences, with SARIMA lagging behind at an R-squared score of a 0.61 (the best result by using simulated annealing for fine tuning). This highlights the crucial role of choosing the appropriate algorithm based on individual needs and data characteristics. The second part of the thesis delves into the technical aspects of the Python scripts. We meticulously detail the data analysis and preprocessing steps, unveiling the inner workings of each script with descriptive comments, visuals, and step-by-step execution guides. Witnessing the results unfold, you'll understand how these scripts compare and rank the algorithms based on performance metrics like R-squared. The research underscores the potential of ML in sales forecasting, particularly Facebook Prophet's ability to deliver accurate predictions. Furthermore, the developed Python script offers a versatile tool for running and comparing multiple forecasting models simultaneously. This readily implementable solution worked with businesses across various sectors, from corporations managing inventory to logistics providers anticipating client needs, to leverage historical data and optimize their sales forecasting processes, ultimately contributing to improved efficiency and success.

Keywords: Machine learning, Sales forecasting, SARIMA , Holt Winters, Facebook Prophet

# Table of Contents

# LIST OF TABLES

# LIST OF FIGURES

# List of Abbreviations

| Abbreviations | |
|---|---|
| SARIMA | Seasonal Auto-Regressive Integrated Moving Average |
| FbProphet | Facebook Prophet |
| AR | Auto-Regressive |
| MA | Moving Average |
| ARIMA | Autoregressive Integrated Moving Average |
| Q-Q | Quantile-Quantile |
| AIC | Akaike information criterion |
| MAE | Mean absolute error |
| MAPE | Mean Absolute Percentage Error |
| RMSE | Root Mean Square Error |
| MSE | Mean squared error |
| PACF | Partial autocorrelation function |
| ACF | Autocorrelation function |
| ADF | Augmented Dickey-Fuller |
| SSE | Sum of Squared Errors |
| MLR | Multiple linear regression |
| HW | Holt-Winters |
| ML | Machine Learning |
| LR | Literature review |

# CHAPTER-1

# INTRODUCTION

In the today's dynamic business climate, accurate forecasting is no the longer a luxury, but the necessity. Companies are juggling consumer sales and cost constraints in find themselves, walking a series situation with the overstocking and, understocking. Excess inventory leads to the capital lock-up, obsolete goods, and shrinking margins, while the insufficient stock risks lost sales and the disgruntled customers. This is complex unpredictability between supply and the sales is where the magic of the Machine Learning (ML) shines, offering a tool for the navigating the unpredictable waters of sales forecasting.

Accurate sales forecasting serves as the foundation of effectives supply- chain management, enabling businesses to make informed decisions in regarding the inventory levels, pricing- strategies, and resource- allocation (Alpaydın 2010; Cica et al., 2020; Wenzel et al., 2019). Traditionally, statistical models like ARIMA and SARIMA has held sway in this domain, demonstrating in their usefulness in the specific scenarios (Makridakis et al., 2019; Shadkam, 2020; Vagropoulos et al., 2016). With the ever-evolving business landscape, characterized by increasing complexity and of the dynamism, necessitates the exploration of the alternative approaches that offer greater accuracy and flexibility (Ouedraogo et al., 2019).

This research paper goes into the exciting world of the ML applications in supply - chain management, specifically focusing on sales forecasting. The heart of the paper lies in the understanding of the limitations of the traditional forecasting methods, which is often struggles with the noisy data and unpredictable influences like political, economic, and psychological factors. This is where the ML steps in, offering the more robust and adaptable approach.

Machine learning (ML) algorithms have emerged , as game-changers in the realm of sales- forecasting, offering several advantages over the traditional methods. Their ability is to adapts to diverse data formats and capture non-linear  relationships holds immense promise (Géron et al., 2017; Mohri et al., 2018; Oladipupo Ayodele, 2010). Among these, Facebook Prophet has garnered significant attention due to its user-friendliness, interpretability, and empirical success in various domains (Chakure, 2019; Ouedraogo et al., 2019). However, a thorough understanding of its performance compared to established statistical models remains crucial for making informed implementation decisions (Marjan et al., 2018).

To unlock the potential of ML for sales forecasting, This embark on a comprehensive literature -review. This journey leads us through various ML types, methods, and their specific applications in supply chain management. This tool is to takes the form of two Python scripts, each tailored for the specific purposes. The first script focuses on the single timeseries dataset, comparing the performance of three diverse algorithms: SARIMA ,Holt-Winters, and Facebook Prophet. These algorithms is to  represents the power of autoregression, exponential smoothing, and ensemble machine learning, each having their own strengths and Weakness. Pongdatu and Putra (2023) compared SARIMA and Holt-Winter's Exponential Smoothing for forecasting customer transactions at a retail store. Using sales data from 2013 to 2017, they evaluated accuracy based on Mean Absolute Deviation (MAD). The SARIMA model (1,1,0)(0,1,0)12 had the lowest MAD of 5.592, indicating it as the preferred model for short-term forecasting, while Holt-Winter's method was more effective for seasonal data with trends (Pongdatu & Putra, 2018)

The heart of this investigation lies in the performance comparison of three robust algorithms: SARIMA, the champion of timeseries analysis; Facebook Prophet, a renowned ML forecasting tool; and Holt-Winters, a stalwart in the general timeseries forecasting. Utilizing the  meticulously ,pre-processed historical datasets from the global superstore, these scripts rigorously evaluated each algorithm based on the established performance metrics like Adjusted R-squared and R-squared. While ARIMA , emerged as the leader in The specific context, demonstrating superior accuracy for capturing seasonal- trends and external factors, the analysis underscored the importance of the  individual strengths and the  limitations of each approach.

This research aims to the bridge this gap by conducting the  comparative study of Facebook Prophet and established statistical models for sales forecasting. Leveraging the  comprehensive datasets and rigorous evaluation metrics, the study will assess the relative accuracy, robustness, and generalizability of each approach under varying conditions. The findings will contribute valuables insights for the practitioners and researchers alike, aiding in the selection of the most suitable forecasting technique for their specific needs (Cavalcante et al., 2019; Seyedan & Mafakheri, 2020).

## 1.1 Research Questions

**RQ1:** Considering the dynamics of sales behaviour in supply chains, which machine learning or statistical algorithm most effectively utilizes historical data to generate reliable and actionable sales forecasts?

Motivation: Precise sales forecasting fuels efficient supply chain management. Inaccurate predictions can lead to overstocking, resulting in wasted resource and inventory costs, or understocking, causing missed sales opportunities and customer frustration. Unveiling the optimal algorithm empowers businesses to optimize inventory levels, navigate market fluctuations, and ultimately enhance profitability and customer satisfaction.

**RQ2:** How does the performance of the algorithms vary when fine-tuned using different hyperparameter optimization techniques?

Motivation: The performance of the algorithms with regard to variations of hyperparameter optimization techniques is motivated by the need to improve model accuracy, efficiency, and applicability—all of which will lead to better forecasting outcomes and better business operation.

Rationale for Changes: Sales forecasting is a narrower and more specific term than sales forecasting within the supply chain context. It focuses on predicting the quantity of goods a company expects to sell, rather than the overall sales from all stakeholders in the supply chain. Using dynamics of sales behaviour instead of complexities of logistics sales emphasizes the specific data characteristics and challenges relevant to sales forecasting, making the research question more focused.

The revised motivation section clarifies the potential benefits of identifying the best algorithm for sales forecasting in terms of business performance and customer satisfaction.

## 1.2 Algorithms used for the research

From these diverse techniques, have strategically selected key players based on their suitability for the specific forecasting task and data characteristics. The undisputed sovereign of stable seasonal trends, ARIMA brings its proven statistical might to bear on simple time series forecasting. For complex timeseries with the pronounced seasonality and external influences, SARIMA. Holt-Winters: Representing the Exponential Smoothing power, Holt-Winters brings its expertise in capturing recent trends and seasonality to the data. Facebook Prophet: From the Ensemble method, the Facebook Prophet emerges, its multitude of decision trees promising high accuracy and the versatility in the diverse forecasting scenarios. For selection of hyperparameter for the respective algorithms, use of grid search and simulated annealing optimization techniques is applied to improve the performance of the model.

# CHAPTER 2

# LITERATURE REVIEW

The thesis embarks on a crucial voyage: exploring the vast ocean of Machine Learning (ML) applications in logistics and supply chain management, with a specific focus on its role in conquering the ever-changing tides of sales forecasting. To chart the most effective course, we first embark on a comprehensive literature review (LR), meticulously sifting through the knowledge landscape to identify the most potent ML tools and algorithms navigating this complex domain.

The LR casts a wide net, encompassing articles published from 2016 to 2024, ensuring we capture the latest advancements in this dynamic field. We cast The lines across diverse sources – books, Journals, conferences, and online libraries – seeking the wisdom whispered in both prominent and niche corners of the academic world. However, to ensure focus and clarity, we set some strict criteria to filter The catch:

Inclusion Criteria:

- Articles published between 2016 and 2024 (the sweet spot of recent developments)
- Accessible in English, bridging language barriers
- Available in their entirety, regardless of institutional access
- Aligned with the thematic focus of The research

Exclusion Criteria:

- Lost in translation (non-English articles)
- Incomplete stories (articles lacking full content)

- Stuck in the past (pre-2016 publications)

To cast the net even wider and delve deeper into specific areas, we select with a potent arsenal of keywords. We searched for pearls of wisdom under terms like "machine learning," "timeseries forecasting," and "sales forecasting by using machine learning," meticulously combing through the supply chain itself with keywords like "big data" and "Industry 4.0." But The journey didn't stop there. We ventured into the heart of the forecasting storm, wielding terms like "ARIMA," "SARIMAX," "Holt-Winters," "Facebook Prophet," and "forecasting metrics" to unearth the most effective tools for charting the future of sales.

The beauty of this comprehensive LR lies in its repeatability. By meticulously documenting the search process and criteria, we ensure that anyone following in the footsteps can replicate the journey and arrive at similar findings. This transparency and objectivity are the cornerstones of the research.

The screening process was a carefully orchestrated dance, unfolding in four steps:

1. Casting the Net: We set sail across the online sea, armed with the chosen keywords and academic sources like Scopus, Emerald Insight, and IEEE Xplore.
2. Sifting the Catch: Each article was carefully examined against the inclusion and exclusion criteria, ensuring only the most relevant and valuable pieces landed in the research vessel.
3. Charting the Course: To navigate the ever-growing pile of articles, we meticulously documented them in an Excel spreadsheet, categorizing them by year and content for easier review. Figure 1 showcases this journey, visually depicting the preference for the latest research.
4. Unveiling the Treasures: Finally, we delved deep into each article, analyzing, comparing, and contrasting their findings to weave together the tapestry of the LR.

Through this rigorous process, we identified 550 relevant articles, from which we ultimately selected 53 to serve as the foundation of the LR. These carefully chosen pieces, brimming with insights and expertise, will guide us as we chart the course towards the optimal ML algorithm for sales forecasting in the intricate landscape of logistics and supply chains.

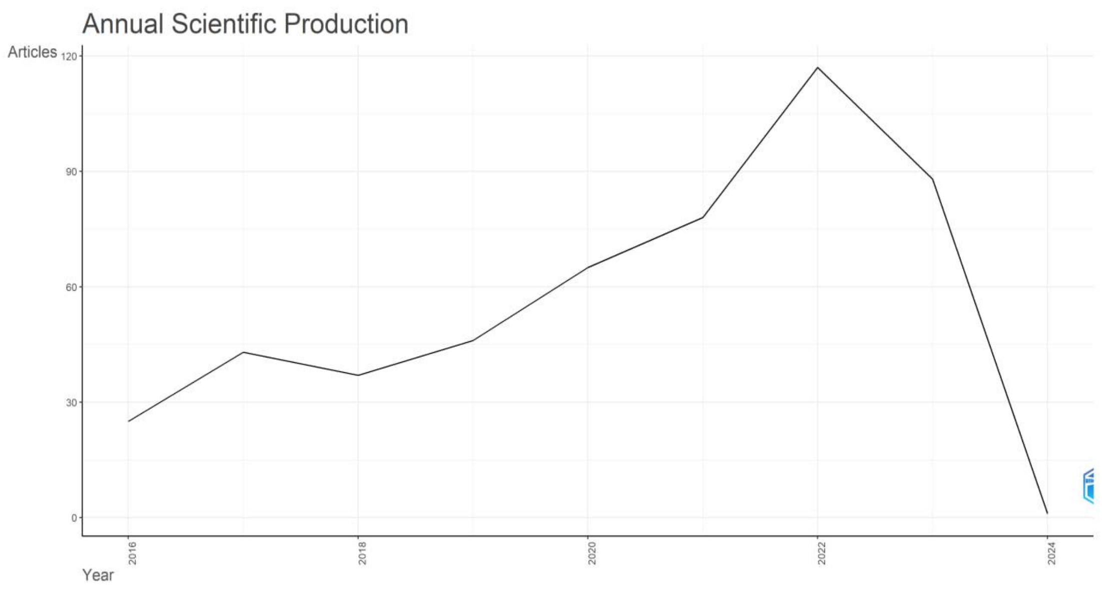


Fig.2.1 Annual Scientific Production

## 2.1 Time Series forecasting

A time series is a sequence of the data points collected, recorded and measured at the successive, evenly spaced time intervals. Each data points represent the observations or measurements taken over time, such as the stock prices, temperature readings, or the sales figures (Chatfield). Time series data is commonly represented graphically with the time on the horizontal axis and the variable of interest on the vertical axis, allowing analysts to identify trends, patterns, and changes over time. In the words of (Ratnadip Adhikari and R. K. Agrawal (2013)), a timeseries is "a consecutive set of the data points, calculated typically over the successive time points.". Mathematically, this act as to translates into a set of vectors, x(t), where t represents the time stamp. Each x(t), is a random variable, a snapshot of the fluctuating phenomenon.

Timeseries also divide into two types i.e univariate and the multivariate. Univariate series focus on a single variable, like tracking the stock market's daily ups and the downs. Multivariate series, on the other hand, juggle multiple variables, like studying how temperature, humidity, and rainfall affect crop yields (Sirisha et al.).

### 2.1.2 Time Series Components

Time series analysis delves into the dynamic tapestry of change, meticulously dissecting its intricate threads. At its core lie four prominent components, each contributing to the unfolding narrative:

1. Trend: This long-term, over-arching movement represents the series' fundamental trajectory.

2. Seasonality: Cyclical fluctuations with the in a year, predictable as the changing seasons, paint - vibrant stripes across the time series.

3. Cyclicity: Beyond the annual waltz, longer-term oscillations, known as cycles, This have their way through the data. Economic cycles, with their phases of prosperity and recession, or the technological innovations with the their periods of rapids advancement .

4. Irregularity: Random fluctuations due to unforeseen events like natural disasters, political turmoil, or any unexpected market crashes contributes these unpredictable deviations from the expected path (Sirisha et al.).

These four components, often intertwined and interacting, contribute to the overall complexity of a time series. Understanding their interplay is crucial for effective analysis and modelling. In this regard, two major model types come into play:

a) Multiplicative Model: This model assumes a multiplicative interdependence between the components, where each factor amplifies or dampens the others' effects. Think of rainfall, impacting crop yields in a way that a dry season (seasonal) contributes a long-term drought (trend), leading to significant fluctuations (irregularity) (Forecasting: Principles and Practice (3rd ed)).

b) Additive Model: This model posits the additive contribution, where each components simply adds or subtracts from the overall value. Imagine, daily sales -figures, being influenced by a steady upward trend, regular Thickened dips (seasonality), and occasional promotions leading to the spikes (irregularity) (Forecasting: Principles and Practice (3rd ed)).

Multiplicative Model: $Y_t = T_t \times S_t \times C_t \times I_t$

Additive Model: $Y_t = T_t + S_t + C_t + I_t$

$Y_t$ is the observation and $T_t, S_t, C_t\ and\ \ I_t$ are respectively the trend, seasonal, cyclical and irregular variation at time (t).

**2.2 Sales prediction with ARIMA forecasting model**

In the realm of time series forecasting, one mathematical maestro stands out: the Box-Jenkins technique (Box & Jenkins, 1970). This elegant method, oftens referred ,to as the ARIMA (Autoregressive Integrated Moving Average) model, gracefully uses the power of autoregressive and, moving average models, forging a powerful tool for the capturing hidden patterns and the predicting future trends. Since its debut, the ARIMA approach , has garnered extensive documentations across the various domains, encompassing aspects like the specification, estimation, and diagnostic validations (Peixeiro).

The notation ARIMA(p,d,q) defines the specific configuration of the ARIMA model for time series forecasting. In this notation:

- p defines the order of the autoregressive process, indicating the number of past data points that influence the current prediction.
- d represents the degree of differencing, denoting the number of times the data needs to be differenced to achieve stationarity.
- q refers to the order of the moving average process, signifying the number of past forecast errors incorporated into the model to improve prediction accuracy.

In the realm of time series analysis, whispers of the past hold the key to unlocking the future. The Autoregressive (AR) model of order p (AR(p)) harnesses this wisdom, mathematically This the echoes of past values, into a tapestry of future predictions (Sirisha et al.).

At its core, AR(p) rests on a deceptively simple yet powerful equation:

- $Y_t = \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \phi_3 Y_{t-3} + \phi_4 Y_{t-4} + \cdots + \phi_n Y_{t-n} + Z_t$

Here, $Y_t$ represents the current value to predict, while the $Y_{t-1}$, $Y_{t-2}$, ..., $Y_{t-n}$ signify the n preceding values, each whispering their influence through the coefficient insights ($\phi_1$, $\phi_2$, ..., $\phi_n$). The final term, $Z_t$, denotes the unpredictable element, a white noise error term with the zero constant variance (Peixeiro).

Using the back-shift operator $B$, $(B)$ $Y_t$ = $Y_{t-1}$, the AR(p) can be shown as:

$$\phi(B)Y_t = Z_t$$

$$\phi(B) = 1 - \phi_1 B - \phi_2 B^2 - \cdots - \phi_P B^P$$

Where $\phi(B)$ is a polynomial in B of order P.

The core of MA(q) lies in a deceptively simple yet insightful equation:

$$Y_t = Z_t + \theta_1 Z_{t-1} + \theta_2 Z_{t-2} + \cdots + \theta_n Z_{t-n}$$

Here, $yt$ represents the predicted value, while $Z_t$, $Z_{t-1}$, ..., $Z_{t-n}$ signify the q most recent white noise error terms, each whispering their influence through coefficient Insights ($\theta_1$, $\theta_2$, ..., $\theta_n$). These error terms, characterized by zero, the and constant variance, embody the inherent stochasticity of the time series (Sirisha et al.).

Using the back-shift operator B, the MA(q) can be written as:

$$\theta(B) = 1 - \theta_1 B - \theta_2 B^2 - \cdots - \theta_q B^q$$

Where:$\theta(B)$ is a polynomial in B of order q (Sirisha et al.).

ARIMA (p, d, q) Model Equation:

An ARIMA (p, d, q) model is a multi-autoregressive moving average model consisting of 'p' autoregressive terms and 'q' moving average terms, differenced 'd' times:

$$Y_t = \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \phi_3 Y_{t-3} + \phi_4 Y_{t-4} + \cdots + \phi_n Y_{t-p} + Z_t + \theta_1 Z_{t-1} + \theta_2 Z_{t-2} + \cdots + \theta_n Z_{t-q}$$

In other words, the forecasted $Y_t$ is a constant linear combination of lagged values and a linear combination of current and past error values (Sirisha et al.).

Using the back-shift operator B, the ARIMA(p, d, q) model can be defined as:

$$\phi(B)(1 - B)^d Y_t = \theta(B)\, Z_t$$

Where: $(1 - B)^d$ is the combined auto regressive operator (Sirisha et al.) .

## 2.3 SARIMA Model

While ARIMA excels in its adaptability and Box-Jenkins methodology for model building, its inherent assumption of linear time series restricts its effectiveness in real-world scenarios. To overcome this limitation, SARIMA introduces seasonality and exogenous variables.

Box and Jenkins proposed the SARIMA model to address seasonality in time series. It employs seasonal differencing to achieve stationarity, a crucial pre- requisite for model fitting. For instance, in monthly data, a first-order seasonal difference , involves subtracting corresponding observations from the previous year. This model is denoted as SARIMA(p, d, q) × (P, D, Q) (Sirisha et al.).

(Shadkam, A. et al., (2020)). emphasizes the merits of SARIMA in handling seasonality and external influences, making it ideal for seasonal sales FORECASTING. In their study, Shadkam utilized the SARIMA methodology to predict electricity sales, considering day of the week, holidays, and temperature as external variables. This approach proved effective in capturing sudden sales surges.

A SARIMA model is defined as SARIMA(p, d, q)(P, D, Q)s, where:

- p, d, and q represent the orders of non-seasonal AR, differencing, and MA terms, respectively.
- P, D, and Q denote the orders of seasonal AR, differencing, and MA terms, respectively.
- s represents the number of periods in a season.

The model equation expresses the relationship between the response variable (yt), exogenous variables ($X_{it}$), and white noise terms ($Z_t$). The equation incorporates various parameters, including regression coefficients (β), Insights for autoregressive and moving average terms (φ and θ), and the backshift operator (Bs) (Sirisha et al.).

Mathematical Formalism:

The SARIMA model can be mathematically represented by the following equation:

$Y_t = \beta_0 + \sum \beta_i X_{it} + (1 - \sum \theta_j\, B_j - \sum \Theta_j B_{js})(1 - \sum \phi_j\, B_j - \sum \Phi_j\, B_{js}) Z_t$ where:

- $Y_t$: Value of the series at time t
- $\beta_0$, $\beta_i$: Parameters of the regression part
- $X_{it}$: Observation of the i-th exogenous variable at time t
- $\theta_j$, $\Theta_j$: Insights of non-seasonal and seasonal moving average terms, respectively
- $\phi_j$, $\Phi_j$: Insights of non-seasonal and seasonal autoregressive terms, respectively
- $B_{js}$: Backshift operator (shifts values back by s periods)
- $Z_t$ : White noise term

**2.3.1 Stationarity**

A stationary time series is one whose statistical properties do not change over time. This implies that it will have a constant mean and variance with autocorrelation independent of time. Many forecasting models, in their assumptions, require stationarity. The moving average model, autoregressive model, and autoregressive moving average model all assume stationarity. We are only allowed to use these models if we verify that data is indeed stationary. Otherwise, models cannot be used, and the forecasts cannot be relied upon. Intuitively, this makes sense, because if the data is non-stationary, its properties are going to change over time, which would mean The model parameters must also change through time. This means we cannot derive a function of future values as a function of past values, since the coefficients change at each point in time, making it unreliable to forecast.

### 2.3.2 Testing for stationarity

The Augmented Dickey-Fuller (ADF) test helps us determine if a time series is stationary by testing for the presence of a unit root. If a unit root is present, the time series is not stationary. The null hypothesis states that a unit root is present, meaning that the time series is not stationary.

Let's consider a very simple time series where the present value $y_t$only depends on its past value $y_{t-1}$ subject to a coefficient α1 , a constant C, and white noise $\epsilon_t$. We can write the following general expression:

$$y_t = C + \alpha_1 y_{t-1} + \epsilon_t$$

In above equation, $\epsilon_t$ represents some error that we cannot predict, and C is a constant. Here, $\alpha_1$ is the root of the time series. This time series will be stationary only if the root lies within the unit circle. Therefore, its value must be between –1 and 1. Otherwise the series is non-stationary (Peixeiro).

### 2.3.3 Autocorrelation function

The autocorrelation function (ACF) measures the linear relationship between lagged values of a time series. In other words, it measures the correlation of the time series with itself. For example, we can calculate the autocorrelation coefficient between $y_t$ and $y_{t-1}$. In this case, the lag is equal to 1, and the coefficient would be denoted as $r_1$ (Peixeiro).

### 2.3.4 Fine-Tuning SARIMA with the AIC

In the realm of time series forecasting, the SARIMA model reigns supreme. But with the various order combinations (p, d, q) and seasonal parameters (P, D, Q, S) to juggle, finding the perfect fit can feel like searching for a needle in a haystack. Enter the Akaike Information Criterion (AIC), a method that guide that illuminates the path to the most fitting SARIMA model (Chakrabarti and Ghosh).

The AIC formula might appear intimidating, but the underlying principle is simple: for each candidate model, it subtracts twice the log-likelihood (a measure of fit) and adds a penalty term proportional to the number of parameters.

Akaike Information Criterion (AIC) is a measure of the quality of a model in relation to other models. It is used of model selection. The AIC is a function of the number of parameters k in a model and the maximum value of the likelihood function $\hat{L}$

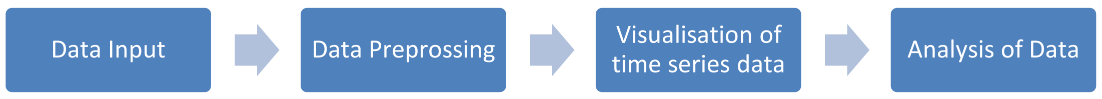
$$\text{AIC} = 2K - 2\ln(\hat{L})$$

This balancing act ensures This avoid two downfalls:

1. Overfitting: A model that meticulously memorizes every data point might appear good on training data, but it falls apart when thrown into the real world. AIC discharges such "overly friendly" models by penalizing their complexity.

2. Underfitting: Conversely, a model overly obsessed with the simplicity might miss vital patterns in the data. AIC nudges us away from such "timid" models by prioritizing goodness of fit.

This is where the auto_arima function shines. It leverages AIC as its compass, automatically exploring different parameter combinations and selecting the one with the AIC, the best one among the candidate models. So, it reduces the manual juggling act and let the AIC guide you toward the perfect SARIMA fit (Chakrabarti and Ghosh).



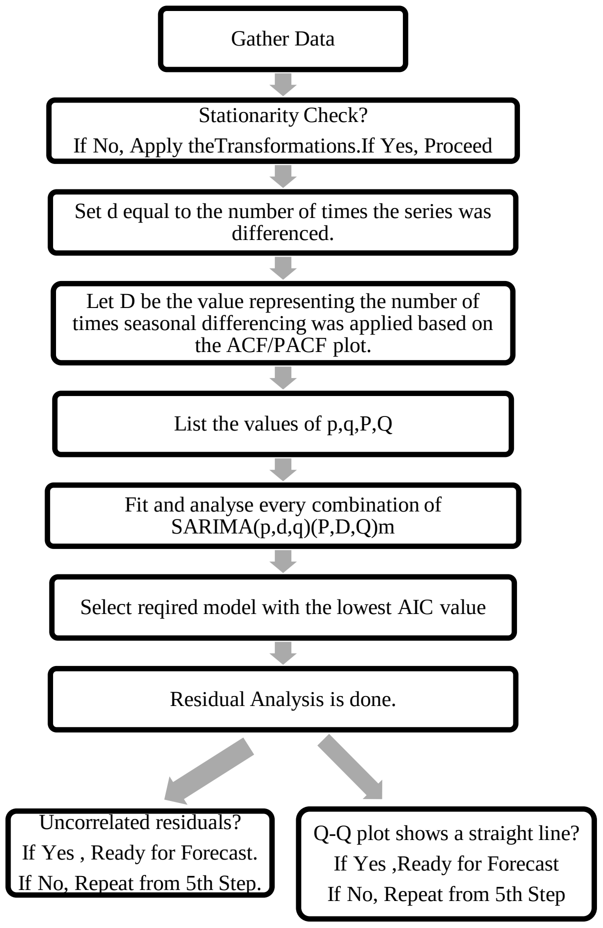

Gather Data
Stationarity Check?
If No, Apply theTransformations.If Yes, Proceed
Set d equal to the number of times the series was differenced.
Let D be the value representing the number of times seasonal differencing was applied based on the ACF/PACF plot.
List the values of p,q,P,Q
Fit and analyse every combination of SARIMA(p,d,q)(P,D,Q)m
Select reqired model with the lowest AIC value
Residual Analysis is done.
Uncorrelated residuals?
If Yes , Ready for Forecast.
If No, Repeat from 5th Step.
Q-Q plot shows a straight line?
If Yes ,Ready for Forecast
If No, Repeat from 5th Step

Fig.2.2 General Modelling procedure for the SARIMA model (Peixeiro).

### 2.4 Holt-Winters Algorithm

While exponential smoothing models excel at capturing general trends in time series data, seasonality throws a curvature in the forecasting of the data. Thankfully, the Holt-Winters algorithm, championed by Holt (1957) and Winters (1960), equipped us with the a powerful tool to tackle this very challenge. This dynamic trio of equations divide the time series into three crucial components. As Holt-Winters forecasting is a powerful method for modelling time series with the trend and seasonality (Winters, 1960). It combines exponential smoothing for level, trend, and seasonal components, providing accurate and interpretable forecasts for diverse applications (Hyndman &

Athanasopoulos et al., 2013). While other models like ARIMA exist, Holt-Winters often proves simpler and more effective for short-term forecasting, particularly when seasonality is prominent (Billah & Rahman et al., 2013):

1. Level ($l_t$): This equation acts as the anchor, capturing the underlying trend and providing a baseline for future forecasts. Smoothed using the parameter α, the level reflects the long-term trajectory of the series, filtering out short-term fluctuations.

2. Trend ($b_t$): This equation quantifies the rate of change, indicating whether the series is steadily increasing, decreasing, or remaining stable. The β parameter governs the smoothing of the trend, determining how much Weight is given to recent changes in the level.

3. Seasonal Factor ($s_t$): This equation breathes life into the model, accounting for recurring patterns with the in a specific period, aptly defined as the seasonality frequency (m) by (Hyndman, Athanasopoulos et al., (2018)) . For monthly data, m = 12, ensuring the seasonal factors capture yearly cycles.

### 2.4.1 Seasonal Trends: A Deep Dive into the Holt-Winters' Additive Method

Forecasting of  the time series data can be a difficult task, especially when recurring patterns and seasonal variations move across the data points. Fortunately, the Holt-Winters' additive method emerges as a powerful tool to capture this complexity, revealing the underlying rhythms and trends with the in seasonality. This robust approach, pioneered by Holt (1957) and Winters (1960), dissects the time series into three crucial components: level, trend, and seasonality, each captured by a dedicated equation and smoothing parameter. At the heart of the model lies the level equation:

$$l_t = \alpha \times (y_t - s_t) + (1 - \alpha) \times (l_{t-1} + b_{t-1})$$

This equation acts as the anchor, capturing the long-term trend and providing a baseline for future forecasts. The parameter α, ranging from 0 to 1, governs the smoothing process. A higher α places, greater Weight on the most recent observation $(y_t)$, adjusting the level more rapidly to capture sudden shifts. Conversely, a α emphasizes past information $(l_{t-1})$, resulting in a smoother trend estimate. The seasonally adjusted observation $(y_t - s_t)$ensures the level reflects the underlying trend with the out distortion from seasonal fluctuations. The trend equation:-

$$b_t = \beta \times (l_t - l_{t-1}) + (1 - \beta) \times b_{t-1}$$

quantifies the rate of change over time. The parameter β, also between 0 and 1, determines how much weight is given to recent changes in the level. A high β reacts swiftly to changes in direction, while a low β favors a more stable trend estimate. This equation captures the gradual increase or decrease in the series, independent of seasonal variations. Finally, the seasonal factor equation:-

$$s_t = \gamma \times (y_t - l_t - b_t) + (1 - \gamma) \times s_{t-m}$$

tackles the recurring patterns of seasonality. The parameter γ, once again with the in the 0 to 1 range, governs the smoothing of the seasonal factors. A higher γ, places more Insight on the current observation $(y_t)$, allowing for quicker adaptation to potential changes in seasonal patterns. Conversely, a lot this γ gives greater Insight ,to past seasonal factors $(s_{t-m})$, ensuring a smoother seasonal adjustment.The term $(y_t - l_t - b_t)$ represents the residual component, the part of the observation not captured by the level and trend. By subtracting it from the current observation, This isolate the seasonal influence and estimate the seasonal factor for the current period.

The Holt-Winters (HW) model reigns supreme among exponential smoothing techniques in the field of forecasting. Its popularity stems from a potent combination of simplicity, low data storage sales, and the ability to automatically adapt to evolving trends and seasonality with the in time series data (Hyndman & Athanasopoulos et al., 2018).

### 2.4.2 Holt-Winters Multiplicative Method

The Holt-Winter method with the  multiplicative seasonality employs three smoothing equations to estimate level ($L_t$), trends ($b_t$), and seasonal components ($S_t$) of the series at time t. The forecast for m periods ahead ($F_{t+m}$) is then calculated as the product of the estimated level and trend adjusted by the seasonal component at the corresponding future period.

Equations:

1. Level Analysis :- $L_t = \alpha \times Y_t \times S_{t-s} + (1-\alpha)(L_{t-1} + b_{t-1})$
2. Trends Analysis :- $b_t = \beta \times (L_t - L_{t-1}) + (1-\beta) \times b_{t-1}$
3. Seasonality Analysis :- $S_t = \gamma \times \left(\frac{Y_t}{L_t}\right) + (1-\gamma) \times S_{t-s}$
4. Forecasted Value:- $F_{t+m} = S_{t-s+m} \times (L_t - mb_t)$

where:

1. α, β, γ are the smoothing parameters (0 < α, β, γ < 1)
2. $Y_t$ is  actual value at time t
3. s is a number of seasons in a cycle
4. m is the number of periods ahead for forecasting

Initialization:

1. $L_s = \frac{1}{s}(\sum_1^s Y_i)$ (average of the first cycle)
2. $b_s = {}^1\!/_k\, (\sum_{s+1}^{s+k} \frac{(Y_i - Y_{i-s})}{s}$ (trend based on two cycles, or use k=1 for one cycle)
3. $S_k = \frac{Y_k}{L_s}$ (seasonal indices for the first cycle)

## 2.5. Facebook - Prophet: Tool for the Seasonality Forecasting

Facebook- Prophet has emerged as a popular tool for time series forecasting due to its ease of use and ability to capture complex patterns. However, it is crucial to critically assess its performances against alternative methods and understand its limitations before applying it in real-world scenarios.

Under the hood, Prophet implements a general additive model where each time series y(t) is modelled as the linear combination of a trend g(t), a seasonal component s(t), holiday effects h(t), and an error term $\in_t$ , which is normally distributed. Mathematically, this is expressed as equation stated below:

$$y(t) = g(t) + s(t) + h(t) + \in_t$$

The trend component models the non-periodic long-term changes in the time series. The seasonal component models the periodic change, whether it is yearly, monthly, weekly, or daily. The holiday effect occurs irregularly and potentially on more than one day. Finally, the error term represents any change in value that cannot be explained by the previous three components (Peixeiro).

Several studies have compared Prophet's performance with the Multiple Linear Regression (MLR) across diverse domains, including apartment pricing (Marjan et al., 2018) and groundwater nitrate concentration (Ouedraogo et al., 2019). These comparisons reveal that Prophet often outperforms MLR in terms of accuracy, particularly for short-term forecasts. However, concerns arise regarding its potential for overfitting due to its inherent complexity and limited data scope in some studies (Marjan et al., 2018). Additionally, exploring alternative time series models, such as ARIMA or SARIMA, could provide valuable insights and potentially lead to more accurate predictions (Ouedraogo et al., 2019).

Experts, generally acknowledge the strengths of the Prophet's clear methodology, empirical evaluations, and accessibility (Marjan et al., 2018). However, they also emphasize the need for broader data coverage and further investigation into potential overfitting issues. Additionally, exploring alternative models and carefully evaluating interpretability and uncertainty are crucial for responsible implementation (Ouedraogo et al., 2019).

In conclusion, Facebook Prophet presents a promising tool for time series forecasting, but its application should be informed by a critical understanding of its strengths, limitations, and comparisons with the  other methods. Carefully considering expert ratings, data characteristics, and research results is essential for making informed decisions and achieving accurate predictions. While blog posts offer a starting point, delving into peer-reviewed research and considering alternative approaches to more robust and reliable forecasting outcomes.

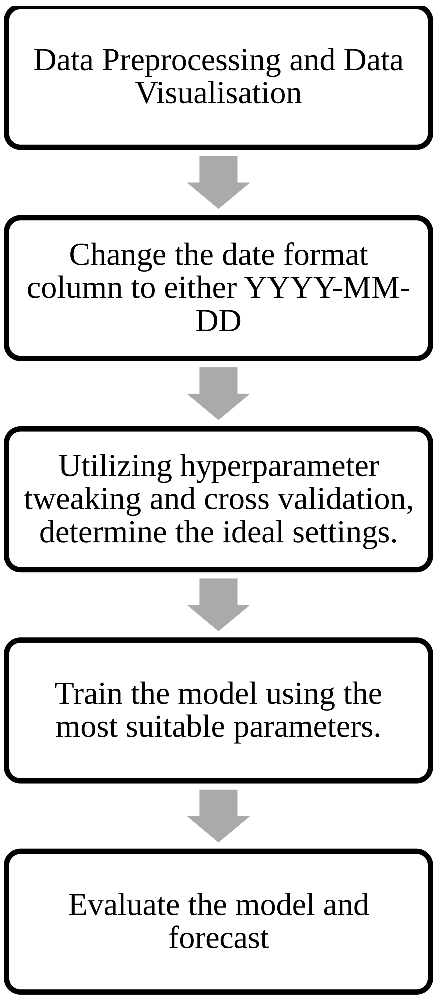

Data Preprocessing and Data Visualisation
Change the date format column to either YYYY-MM-DD
Utilizing hyperparameter tweaking and cross validation, determine the ideal settings.
Train the model using the most suitable parameters.
Evaluate the model and forecast

Fig.2.3  Forecasting process using Prophet (Peixeiro).

As you can see, unlike the ARIMA(p,d,q) model, where future values depend on previous values, this model does not account for the time dependence of the data. Thus, rather than identifying the underlying mechanism, this method is more akin to fitting a curve to the data. While there is a certain amount of predictive information lost when employing this strategy, its flexibility is enhanced by its ability to account for various seasonal periods and evolving trends. It also withstands outliers and missing data well, which is a distinct benefit in a corporate setting. The finding that human conduct resulted in multi-period seasonal time series served as the impetus for the incorporation of numerous seasonal periods. For instance, a pattern may be created by the five-day workweek that occurs every week, or a pattern may be created by the school break that occurs annually. Thus, Prophet models several periodic impacts using the Fourier series in order to account for different seasonal periods.

$$s(t) = \sum_{n=1}^{N} (a_n \cos\left(\frac{2\pi nt}{P}\right) + b_n \sin\left(\frac{2\pi nt}{P}\right))$$

More precisely, the seasonal component s(t) is represented by the equation shown above, where P represents the duration of the seasonal period in days, and N is the number of terms in the Fourier series.In the given equation, if there is a yearly seasonality, the value of P is equal to 365.25, representing the number of days in a year. The value of P for weekly seasonality is 7. N represents the quantity of parameters that we intend to utilize in order to estimate the seasonal components. (Peixeiro).

## 2.6 Fine-Tuning of Hyperparameters with nature inspired Optimization

Forecasting time series data, like monthly sales, often relies on robust models like Holt-Winters. However, the effectiveness of this model hinges on its hyper - parameters, requiring careful tuning for optimal performance. This summary explores three prevalent methods for fine-tuning Holt-Winters hyper - parameters: grid search, gradient descent, and simulated annealing, highlighting their advantages and potential drawbacks (Malki et al.) .

### 2.6.1 Grid Search:

A widely used approach, grid search systematically evaluates various combinations of pre-defined the hyper - parameter values to determine the optimal model (Malki et al.). This exhaustive exploration guarantees finding the "best" configuration with the in the defined search space, but outcome computationally expensive, especially for models with the  numerous hyper - parameters (Hyndman & Athanasopoulos, 2018).

Brownlee (2020) details the use of Random Search and Grid Search for hyperparameter optimization in machine learning models. The guide explains setting up these searches using scikit-learn's RandomizedSearchCV and GridSearchCV, and demonstrates their application with practical examples. The methods efficiently explore hyperparameter spaces to find optimal configurations, improving model performance through cross-validation (Brownlee, 2020).

### 2.6.2 Gradient Descent:

This iterative optimization technique leverages calculated gradients to refine hyper - parameters towards minimizing a chosen error metric. It offers flexibility in defining custom error functions and handle continuous the  hyper -  parameter values. However, gradient descent  works to getting stuck in local minima, potentially missing the optimal solution, and requires careful selection of learning rate and convergence criteria (Bergstra et al., 2012).

### 2.6.3 Simulated Annealing

Inspired by the physical process of annealing, this method introduces randomness to escape local optima. It iteratively evaluates neighbouring the  hyper -  parameter configurations, accepting improvements and occasionally worse solutions with the  a certain probability based on a "temperature" parameter (Henderson et al.). This probabilistic approach enhances the chance of finding the global optimum, particularly for complex models (Geyer, 1991). However, simulated annealing can be slower,  the gradient descent and requires careful tuning of the cooling schedule to ensure convergence. Ervural, Beyca, and Zaim (2016) present a model that combines Genetic Algorithms (GA) with the ARMA method to enhance natural gas consumption forecasting in Istanbul. This approach shows improved accuracy and robustness over traditional ARMA models, as evidenced by lower MAPE and better cost function values (Ervural et al.).

Bhandare and Hajiarbabi (2023) used Simulated Annealing (SA) and Genetic Algorithms (GA) to tune hyperparameters for LightGBM models, achieving higher accuracy and efficiency. SA reached 94.15% accuracy in 83.17 seconds, outperforming GA's 92.39% accuracy in 73.73 seconds and HyperOpt's 87.13% accuracy in 178.03 seconds. These methods optimize the search process for hyperparameters effectively (Bhandare and Hajiarbabi).

Steps for Simulated Annealing:

- Step-1: Choose an initial point $x^{(0)}$ a termination criterion ε . Set temperature T to a sufficiently high value and number of iterations n to be performed at a particular temperature.
- Step-2: Calculate a neighbouring point $x^{(t+1)} = Nx^{(t)}$ randomly.
- Step-3-If ΔE = E($x^{(t+1)}$) - E($x^{(t)}$) < 0 set t = t + 1 ;

Else create a random number r in the range (0,1).

If r <= exp(- $\frac{\Delta \text{E}}{T}$) set t = t + 1 ; Else go to step 2.

- Step-4: If |$x^{(t+1)}$ - $x^{(t)}$| < ε and Tis small, Terminate;

Else if (t mod n) =0, then lower T according to a coding schedule. Go to step 2

## 2.7 Residual diagnostics

In the context of a time series model, the "residuals" are whatever is left over after fitting a model. For many (but not all) time series models, the residuals are equal to the difference between the observations and the corresponding fitted values:

$$e_t = y_t - \hat{y}_t$$

Residual analysis involves two distinct components: a qualitative analysis and a quantitative analysis (Peixeiro). The qualitative analysis primarily examines the Q-Q plot, whereas the quantitative analysis assesses the correlation of the residuals (Fildes).

The Q-Q plot is a graphical display that shows the distribution of two variables, their quantiles, to be specific. For the purpose of time series forecasting, we plot the distribution of residuals on the y-axis against the theoretical normal distribution on the x-axis. This graphical tool enables the assessment of the goodness of fit of the model. If the residuals are normally distributed, a linear trend will appear where we will be able to see the line y = x. This shows that the model is a good fit because the residuals have properties similar to white noise. On the other hand, if the residuals' distribution is not normal, we will have a non-linear trend. From the residuals' distribution analysis, one may conclude that the model is not the right fit since it fails to be close to a normal distribution and therefore fails to have properties of white noise, too (Peixeiro).

The Ljung-Box test is a statistical test used to check if autocorrelation of a dataset differs significantly from zero. In time series forecasting, it can be used to determine whether the residuals of a model have similarities with white noise. The null hypothesis states that the data is independently distributed, which means there is no autocorrelation. Therefore, if the p-value is higher than 0.05, we can't reject the null hypothesis, showing that the residuals are independently distributed. Thus, there is no correlation between them; they have similar characteristics to white noise, and the model is fit for prediction purposes (Peixeiro).

## 2.8 Python Modules

This study has been designed to investigate the prediction of time changes through Seasonal ARIMA (SARIMA) analysis as well as Holt-Winters and Facebook Prophet methods using various important Python libraries available.

### 2.8.1 Pandas

An important library when it comes to handling data in Python programming language since it provides powerful tools for performing data manipulation tasks on time series data such as loading sources of time series in form of CSV files or databases or others. Cleaning those series from unwanted information so that they can be analyzed properly plus establishing an important time-based index which we use in modelling such types (pandas documentation — pandas 2.2.2 documentation).

### 2.8.2 Statsmodels

This is a general-purpose library that provides the statistical base for fitting and evaluating SARIMA models. The package provides functions to select the models based on information criteria such as AIC and BIC, estimation of parameters, and diagnostic tests that can be used to study the model fit and identify potential problems(statsmodels 0.14.1).

### 2.8.3 Visualization Libraries (Plotly, Matplotlib)

Effective communication of the insights found is crucial, and that's where Plotly and Matplotlib come in. With these libraries, the researcher will be able to create informative visualizations: to find out the characteristics of the raw data itself, to visualize the model forecasts, and to compare the performance of different forecasting models. Matplotlib enables the generation of a wide range of plot types for basic to complex visualizations (Matplotlib documentation — Matplotlib 3.9.0 documentation), while Plotly excels in generating interactive visualizations for enhanced understanding (Plotly).

### 2.8.4 NumPy

NumPy is the foundation of nearly all scientific computing libraries in Python; it provides the core numerical computation capabilities. It underlies array manipulations, linear algebra operations, and other mathematical computations used extensively within Pandas, Statsmodels, and other libraries employed in the analysis (NumPy documentation — NumPy v2.1.dev0 Manual).

### 2.8.5 SciPy

The term "Heap" probably refers to SciPy, another powerful scientific computing library. The use statement does not specifically say how SciPy will be used, but it provides functionalities that will come in handy. For example, its optimization algorithms, such as grid search, could be used for hyperparameter tuning in some forecasting models—especially those that do not have built-in hyperparameter optimization routines (SciPy documentation — SciPy v1.13.1 Manual).

### 2.8.6 Prophet

Facebook's Prophet library is specifically designed for time series forecasting. It offers a simple-to-use interface to create and evaluate Facebook Prophet models, which excel at handling time series data with trends, seasonality, and holidays (“Prophet”).

## 2.9. Forecast Performance Metrics

Forecast performance metrics are used to evaluate the accuracy and effectiveness of forecasting models. These metrics help in assessing how well a model predicts future outcomes compared to actual observations. Some common forecast performance metrics include:

**2.9.1 Mean Absolute Error (MAE)**: This metric is the simplest and most intuitive, measures the average absolute difference between predicted and actual values. Its simplicity makes it easy to understand and interpret, but it can be insensitive to outliers, masking large errors with smaller ones (Forecasting: Principles and Practice (3rd ed)).

MAE = $\frac{1}{n}\sum_{1}^{n}\left|Y_i - \widehat{Y_i}\right|$

**2.9.2 Mean Squared Error (MSE):** MSE measures the average squared differences between predicted and actual values. Squaring the errors penalizes larger errors more heavily than smaller ones (Peixeiro).

MSE = $\frac{1}{n}\sum_{1}^{n}\left(Y_i - \widehat{Y_{\iota})}\right)^2$

**2.9.3 Root Mean Squared Error (RMSE):** The square root of MSE brings us back to the original units of The data, making it easier to interpret the magnitude of errors. RMSE is a widely used metric, particularly for continuous data, but it shares the sensitivity to outliers of MSE (Peixeiro).

RMSE = $\sqrt{\frac{1}{n}\sum_{1}^{n}\left(Y_i - \widehat{Y_{\iota})}\right)^2}$

**2.9.4 Mean Absolute Percentage Error (MAPE):** This metric, expressed as a percentage, is particularly useful when dealing with data that varies in scale. MAPE treats all errors equally, regardless of the absolute value, making it suitable for comparing forecasts across different series or time horizons. However, it can be unreliable for zero or very small values, and it can be skewed towards series with large values (Peixeiro).

MAPE = $\frac{1}{n}\sum_{1}^{n}\left|{Y_i - \widehat{Y}_\iota}\Big/{Y_i}\right| \times 100$

**2.9.5 Correlation Coefficient (R):** The coefficient of determination, often denoted as $R^2$, is a metric used to evaluate the goodness of fit of a regression model. It measures the proportion of the variance in the dependent variable that is predictable from the independent variables in the model. In other words, it tells us how well the regression line fits the data (Forecasting: Principles and Practice (3rd ed)).

Mathematically, $R^2$ is calculated as:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

Where:

1. $SS_{res}$ is the sum of squares of residuals (also known as the sum of squared errors or SSE), which represents the difference between the observed and predicted values.
2. $SS_{tot}$ is the total sum of squares, which represents the total variance in the dependent variable.

This metric measures the linear relationship between predicted and actual values, ranging from -1 (perfect negative correlation) to +1 (perfect positive correlation). While R provides insight into the overall strength of the relationship, it doesn't distinguish between underpredictions and overpredictions. In regression analysis, adjusted $R^2$ is a useful measure for assessing precisely a model's explanatory power because it incorporates the predictor-count estimate which helps prevent overfitting while at the same time encouraging robust models for more dependable predictions.

$$\bar{R}^2 = 1 - (1 - R^2)\frac{T - 1}{T - k - 1}$$

where T is the number of observations and is the number of predictors. This is an improvement on $R^2$, as it will no longer increase with each added predictor. Using this measure, the best model will be the one with the largest value of $\bar{R}^2$. Maximising $\bar{R}^2$ is equivalent to minimising the standard error given in Equation given above. (Hyndman & Athanasopoulos, 2018, sec. 5.5)

Ultimately, the choice of metrics is a journey of exploration and discovery, a continuous quest for the perfect lens to illuminate the strengths and weaknesses of The forecasts. By understanding and utilizing the diverse metrics available, we can confidently navigate the forecast jungle, paving the way for more accurate predictions and informed decision-making.

# CHAPTER 3

# RESEARCH METHODLOGY

**Step-by-Step Guide for Analyzing Time Series Data:**

1. **File Selection:** Choose the file, containing the time series data.
2. **Visualization:** Plots the raw data to gains the initial under-standing of its trends and patterns.
3. **Seasonal Decomposition:** Analyze the data's seasonality through the seasonal decomposition and visualize the components.
4. **SARIMA Analysis:** Visualize of the SARIMA model's forecast and simulations alongside with the actual data to assess its fit.Use grid search and simulated annealing for fine tuning of hyperparameter of SARIMA
5. **Comparison Visualization:** Plot the actual, historical, and predicted values from the SARIMA model with the distinct colour for further comparison.
6. **SARIMA Summary:** Print the summary of the fitted SARIMA model.
7. **Holt-Winters Model Selection:** Identify the appropriate Holt-Winters model (additive or multiplicative) based on the characteristics of the time series data. Using different metaheuristics like Grid - search, the Simulated annealing optimization algorithms model in the fine tuning of the hyperparameters and get the result with the best suited model.
8. **Holt-Winters Visualizations:** The Analyse the model's fit by the plotting the forecasts and simulations along-side the actual data.

9. **Comparison Visualization (Holt-Winters):** Plot the actual, historical, and predicted values from the Holt-Winters model with the distinct colours for detailed comparison.
10. **Holt-Winters Summary:** Prints the summary, of the selected Holt-Winters model.
11. **Facebook -Prophet Regression:** Specify ,the numbers of estimators for the Facebook -Prophet model. Use grid search and simulated annealing for hyperparameter fine tuning.
12. **Prediction Comparison:** Print and to compare the expected and predicted values generated by the Facebook- Prophet model, followed by the visualizations of the both for the analysis.
13. **Error Reporting:** Print the $R^2$ metrics, for the all evaluated algorithms (Facebook Prophet, SARIMA, and Holt-Winters).

## 3.1 Data set

This thesis delves, into the realm of product sales - forecasting, with the aim of comparing the performance of different algorithms through the error analysis. To achieved this, This set out to find the suitable datasets representing the real-world sales/sales patterns.

After exploring various data sources, like Kaggle, Google Datasets, and data.gov, This settled on a comprehensive dataset published by ROHIT SAHOO on Kaggle (https://www.kaggle.com/datasets/rohitsahoo/sales-forecasting/). Here ,This are going to take the data-set containing the daily Sales and Profits of the Superstores in 4 year from 2015 to 2018.

The Superstore Sales Dataset, while not specifically focused on EU data, offers a rich playground for the time- series forecasting. It cover four years (2014-2017) and tracks individuals orders through the various stages, from placement (Order Date) to delivery (Ship Date). This detail, allows for the detailed analysis of sales trends, across the  different products - categories (Furniture, Office- Supplies, Technology) and subcategories, revealing the  seasonal patterns and, the potentials drivers of demands. Additionally, customers information like City and Regions provides valuables insights the  into geographical influences on the sales. Ultimately, the "Sales" field capturing the prices, of each sold -items, is the heart of the datasets enabling to the builds and evaluates time -series forecasting models, for predicting of the future sales and to optimise the inventory managements. This data, while not the specifically the EU-centric, offers the robust platforms for the exploring and for predicting sales patterns, making it as  the  valuable resources for any retails business seeking to the  leverage  in  the time- series forecasting for the growth.

Achieving accurate monthly sales forecasts for each product across different central warehouses would offer significant benefits to the company.  However, in this analysis, this focus to the solely on the products level, excluding warehouse and category information's.  The goal is to employ four distinct forecasting algorithms to predict the monthly sales for each product and to compare that their performance against actual values. This comparison will be facilitated by the calculating key error - metrics such as Adjusted R- square, ultimately as enabling us to identify the most effectives algorithms for this specific task. To prepares the dataset for the analysis, This is to utilized python libraries, scripting capabilities. This is to  be allowed us to the tailor the data, to The specific needs before the integrating it into The Python scripts, for the forecasting algorithms. By the systematically, evaluating the performance of different forecasting algorithms and  identifying the most effective approach for this real-world product sales data, this thesis, aims to provides valuable insights and actionable recommendations for the manufacturing company. The findings theoretically, to optimize production scheduling, inventory management, and resource allocation, ultimately as enhancing operational efficiency and the profitability.

# CHAPTER 4

# IMPLEMENTATION OF TIME SERIES FORECASTING

In this project ,we are given few datapoints related to the sales numbers for retail store. We will first do the some exploratory data analysis and after that we will attempt to forecast the store's sales numbers for the next 12 months using 3 different forecasting models. We will use Exponential Smoothing (Holt-Winters additive), SARIMA and Facebook Prophet to fit and forecast.

## 4.1 Importing Essential Tools for Data Analysis

This code block assembles a powerful toolbox for data exploration and visualization. NumPy provides efficient mathematical operations for arrays, while Pandas handles data structures and file I/O like reading CSV files. Seaborn and Matplotlib join forces to create stunning and informative plots, allowing us to uncover patterns and relationships within The data. With these libraries at hand, we're ready to dive deep into the world of data analysis!

## 4.2 Data Imports and Analysis

Using the pd.read_csv function from the Pandas library, the code reads a CSV file named train.csv located in the /kaggle/input/sales-forecasting directory. This likely means the analysis is being done on Kaggle and the data file is stored in a pre-defined location for competition participants.

The read data is stored in a variable named df, which will be used for further analysis throughout the script. The df.head(2) method is used to display the first two rows of the DataFrame stored in df. This gives a quick glimpse of the column names and their corresponding values in the first two data points.The output shows the first two rows of the DataFrame, revealing various columns like order ID, customer name, product category, etc.

We see there is missing data under zipcode. We will not be using this column so we won’t bother with imputation.This means some rows in the DataFrame might have empty or invalid values for this specific column.

Inspecting the dataset with df.info() revealed a wealth of information about its structure and contents. We have 9800 data points spread across 18 columns. Most columns are of type object, indicating categorical data like customer names, regions, and product categories. Thankfully, only 11 entries in the "Postal Code" column are missing, representing a negligible percentage.

Next, we turned The attention to understanding the distribution of key categorical variables. By analyzing the counts returned by value_counts() for each field, we gained valuable insights into the makeup of The data. For instance, we discovered that:

a) Segments: "Consumer" dominates with over 5100 entries which followed by "Corporate" and the  "Home Office". This suggests a focus on the individual customers rather than the large businesses.
b) Regions: The dataset is geographically balanced, with "West", "East", and "Central" regions each contributing roughly 30% of the data. "Office Supplies" ,takes the lead with over 5900 entries, followed by "Furniture" and "Technology". This highlights the focus on office-related products.
c) Sub-Categories: "Binders", "Paper", and "Furnishings" emerge as the most prominent sub-categories within their respective categories, providing further details about the types of products sold.

d) Ship Modes: "Standard Class" is the preferred shipping method, accounting for nearly 60% of orders. Other options , like "Second Class" and "First Class" are also used, with the "Same Day" deliveries being with the least frequent.

These initial explorations offer a crucial first step in understanding the data. By delving into the distribution of categorical variables, we gain a sense of the customer base, product range, and the shipping preferences. This knowledge will be the invaluable as, we move forward with the data cleaning, feature engineering, and building models to extract  the meaningful insights from the dataset.

## 4.3 Exploratory Data Analysis

Plotting sales on US map,  using choropleths and Plotly. This creates a dictionaries of the  codes to achieved this , since the neither state codes, nor longitude/latitude data’s is available. Map is interactive , so hovering over the state will show you details.

### 4.3.1 Visualizing Sales Across the U.S

To understand, the geographical -distribution of sales, This is to utilized Plotly , to create the interactive choropleth map. Since the dataset lacked state codes or longitude/latitude data, this will built a dictionary mapping state names to their abbreviations. Equipped with this map, This added the "Abbreviation" column to the Data-Frame and grouped it  by state to calculate total sales for each.
With the prepared data’s, This generated a choropleth maps using Plotly's go.Choropleth function. This map colours each state based on its total sales, allowing for easy visual comparison. Hovering, over a state displays its name ,and sales figure, adding the interactive layer of information.
This visualization reveals a clear trend: California, New York, and Texas stand out as the top three states in terms of total sales, represented by darker shades on the map. This information can be valuable for strategic decision-making and understanding regional sales patterns.Overall, these visualizations offer a clear picture of how sales vary across the United States, highlighting key trends and allowing for deeper analysis of regional performance

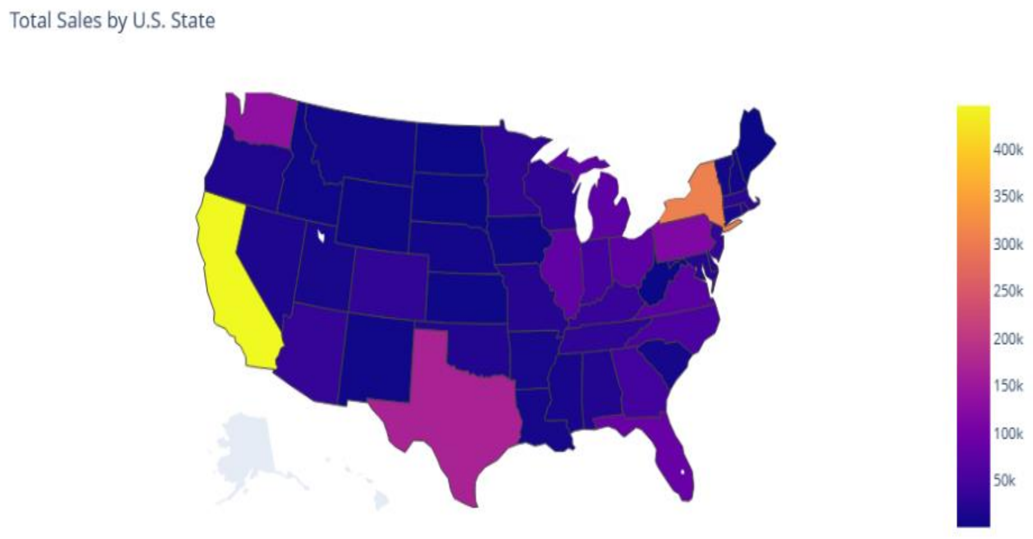


Fig. 4.1 Visualizing Sales Across the U.S

### 4.3.2 Analyzing the Top Sales States with the  a Horizontal Bar Graph:

Building upon the state-levels sales calculations, This delved  deeper with the  a horizontals bar graph. First, This sorted the sum_of_sales Data-Frame by the Sales columns in descending order, and prioritizing states with  the highest total sales data as we get. Then, using Seaborn's barplot function, This created a visually appealing graph with the states on the y-axis and their respective sales values on the x-axis. This is bar graph offers, a straight forward  and to view of the leading states, in terms of the sales such as the Texas, California, and New York and etc. The graph also allows us to easily and compares the  sales figures across the top-performing states and identify any significant gaps or clusters. This visualization complements the previous choropleths map and provides a different perspective on the geographical distribution of the sales.

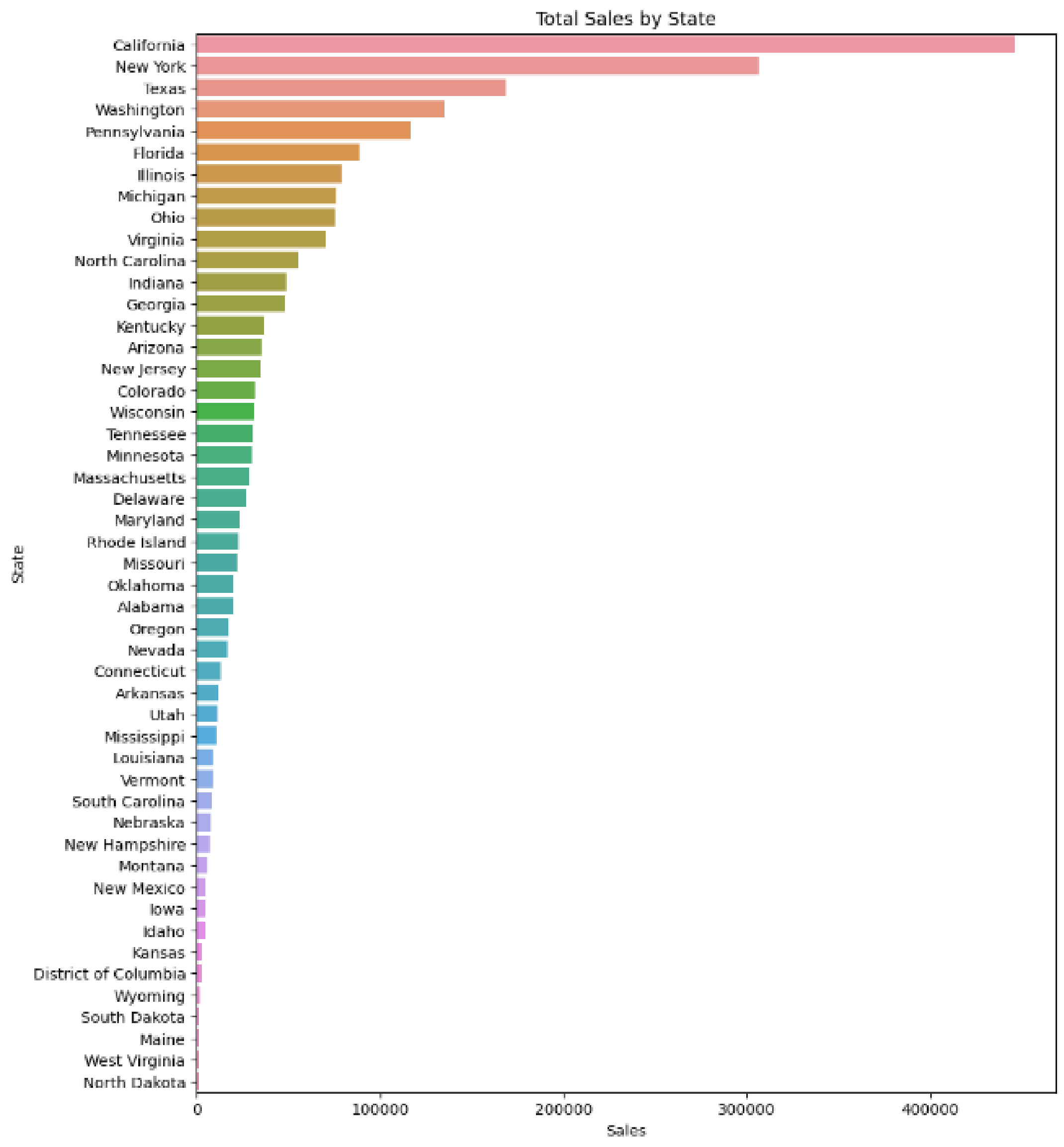


Fig 4.2 Total sales by state in USA

The 'Category' subplot focuses on product classifications (Office Supplies, Furniture, Technology). This visualization will showcase which categories generate the most revenue and potentially highlight areas for expansion or focus. Here Technology product has more sales overall.

Finally, the 'Sub-Category' plot dives deeper with the in each category, revealing the top-performing sub-categories such as phones and chairs. This granularity helps identify specific product types driving sales with the in broader categories.

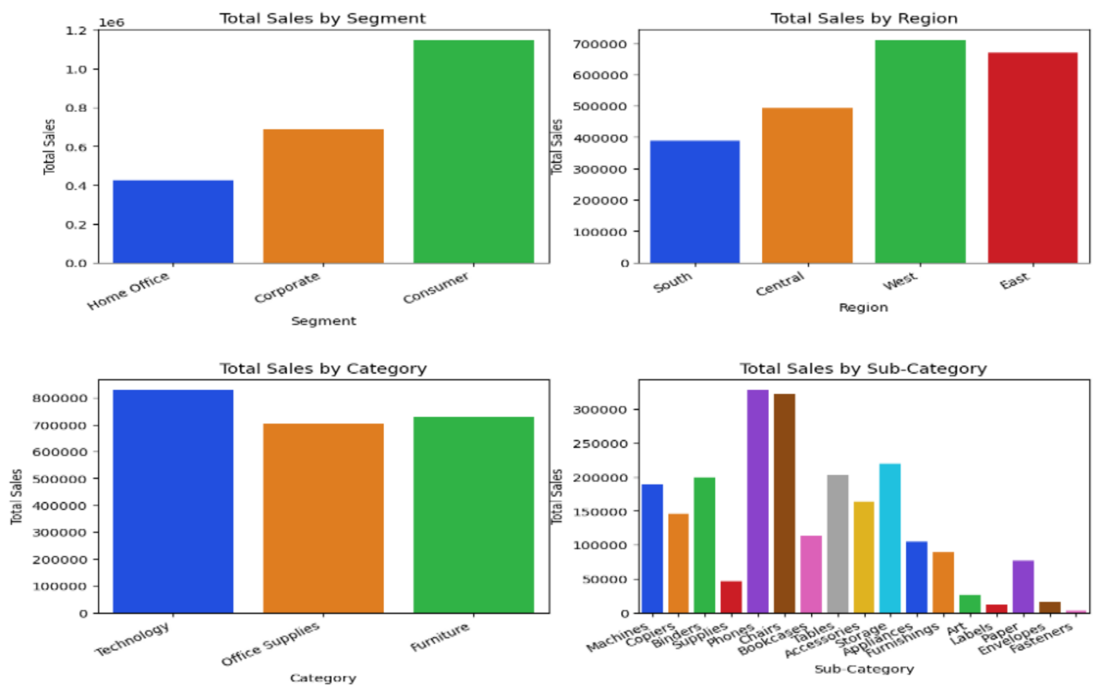


Fig. 4.3 Total sales with the respect to segment,region, category, sub-category

### 4.3.3 Unveiling the Category-Subcategory Relationships with the a Sunburst Chart

To delved deeper into the intricated relationship between product categories and subcategories, this employed the interactive sunburst chart using Plotly Express. This will visualization offers a unique perspective on how the sales are distributed, with in different products hierarchies. First, the Data-Frame was summarized by aggregating sales across both the category and subcategory levels. This is condensed the data into a format suitable for the sunbursts charts. Using px.sunburst, Plotly Express generated the interactive visualizations. The "path" is the parameter to defined the nested hierarchy of categories and ,subcategories, while the "values" specified in the corresponding sales figures. The sunburst chart radiates outwards from the center, with the each ring representing a category and , its subsequent segments representing subcategories. Larger segments signify higher sales with the in that category or subcategory, like Chairs with the furnitures and technology with the phones . Hovering, over any segments reveals details of like the subcategory names and, its sales contribution. Clicking/unclicking on category segments allows, for dynamic exploration, highlighting only the subcategories with the in the chosen category. This is to provides a focused view of subcategory performances , with the in a specific category contexts.

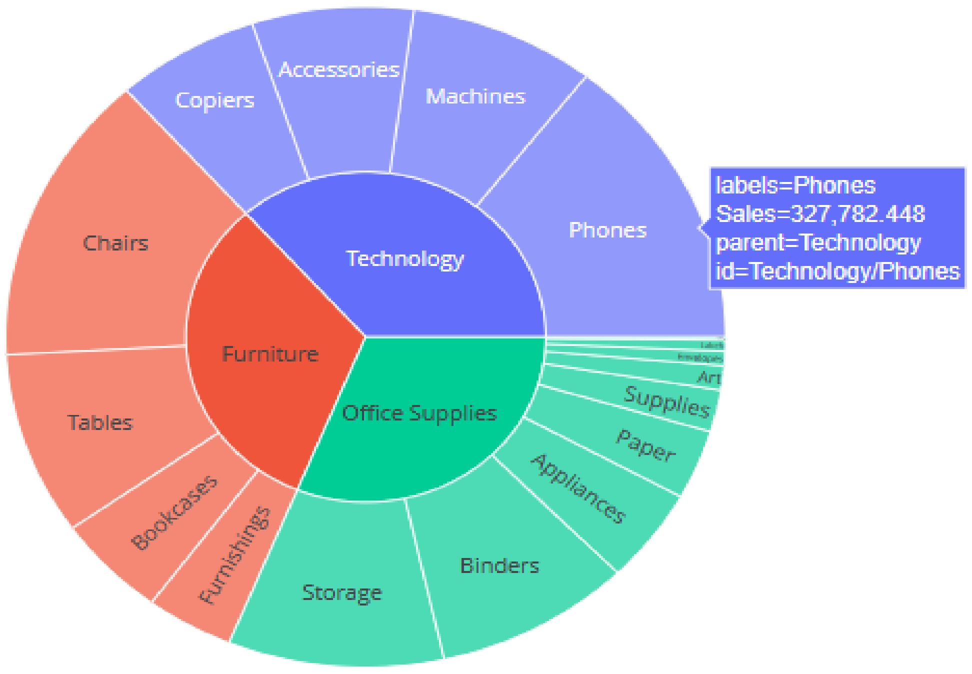


Fig. 4.4 Summarize the Sales data by Category and Sub-Category

### 4.3.4 Unveiling Sales Trends by Shipping Mode: Bar Plot / Interactive Treemap

To understand , how sales vary across the different shipping- methods, the code snippet first groups the data by the "Ship Mode" and calculates the total sales is for each using group-by and sum. This will provides, a foundation for the visualizing sales performance across the "Standard Class", "Second Class", "First Class", and "Same Day" options. Here is Standard -Class has most sales value.

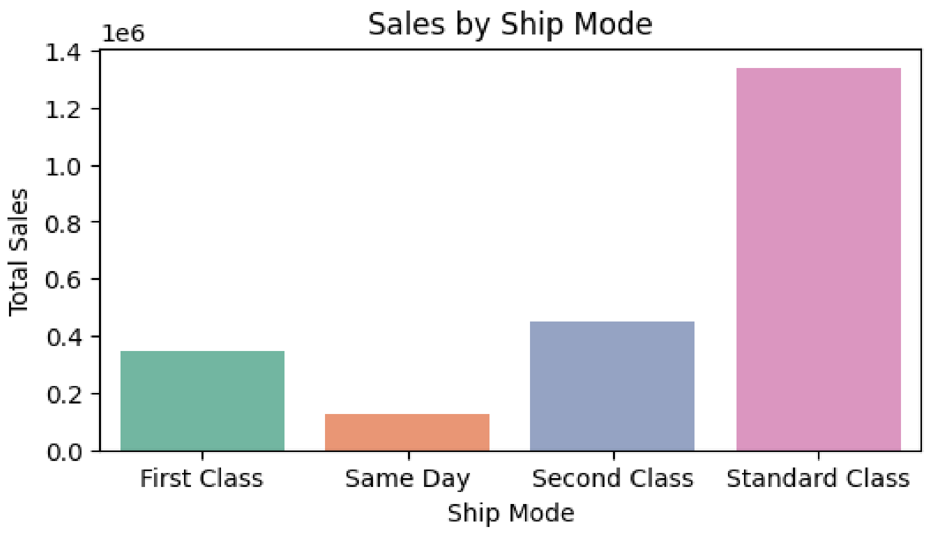


Fig. 4.5   Sales by Ship Mode

### 4.3.5 Drilling Down into the Sales with the  Interactive -Treemap

This interactive visualization, allows to explored these relationships between  products categories, shipping modes, and subcategories by the creating the interactive treemap. It first summarizes sales -data by grouping it across , these three dimensions, and then uses Plotly,  to Express to generate the visualization.

The largest branches, represent categories like the  "Office Supplies" and "Furniture," contribute to the most to total sales. With the in each category, the thickness of sub-category branch, indicates their sales performance for the different shipping modes. For example, "Binders" under "Office Supplies" seem to have the higher sales with the  "Standard Class" shipping. Zooming into the  specific subcategories like "Appliances" under "Technology," , it gives us that, how sales are  distributed across the shipping options like the "Second Class" and ,"First Class."

Enriching the Date Data: The code transforms the "Order Date" column into a datetime format and then extracts day, month, and year as separate columns. This creates valuable new features for deeper analysis like seasonal trends, sales patterns, and year-over-year comparisons.By providing context with the in each order date, this data manipulation unlocks richer insights and potentially reveals hidden patterns with the in the dataset.

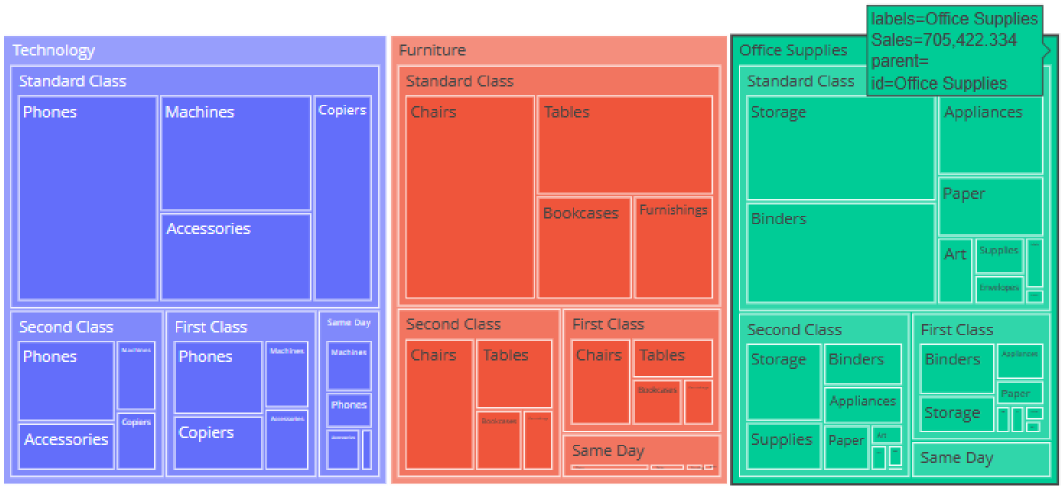


Fig. 4.6 :- Summarize the Sales data by Category, Ship Mode, and Sub-Category

### 4.3.6 Zooming Out for Sales Trends:

To avoid cluttering the view with the daily sales data, the code combines "Month" and "Year" into a single "Date" column and then groups sales figures by month. This creates a concise monthly view. The resulting line plot (in the image you sent) shows a clear trend: sales fluctuate over time, with the some months (like December) consistently exceeding others. This high-level overview helps identify seasonal patterns and potential areas for further investigation. By stepping back from daily data, This gain a broader perspective on sales trends, allowing for more strategic planning and resource allocation based on anticipated monthly sales patterns

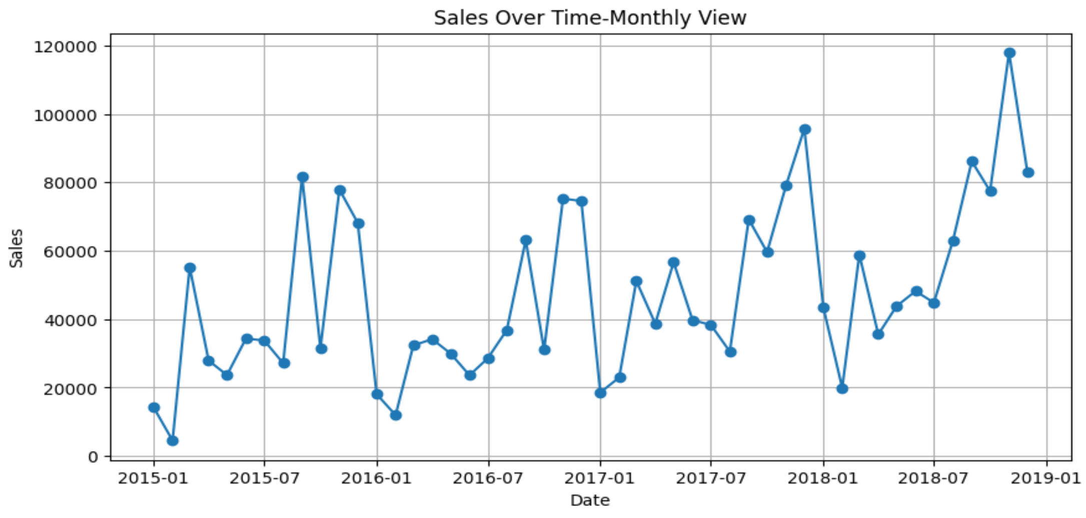


Fig 4.7 Sales over Time- Monthly view

### 4.3.8 Diving into Daily Sales Trends:

Analyzing the sales patterns, by day. It first groups the Data-Frame by the "Order Date" and then calculates the total -sales for each day using group by and, sum. This is create a condensed view of daily -sales performances. Below is the daily -view, This can see some outliers but that should not tinkers, with the The forecasts too much.

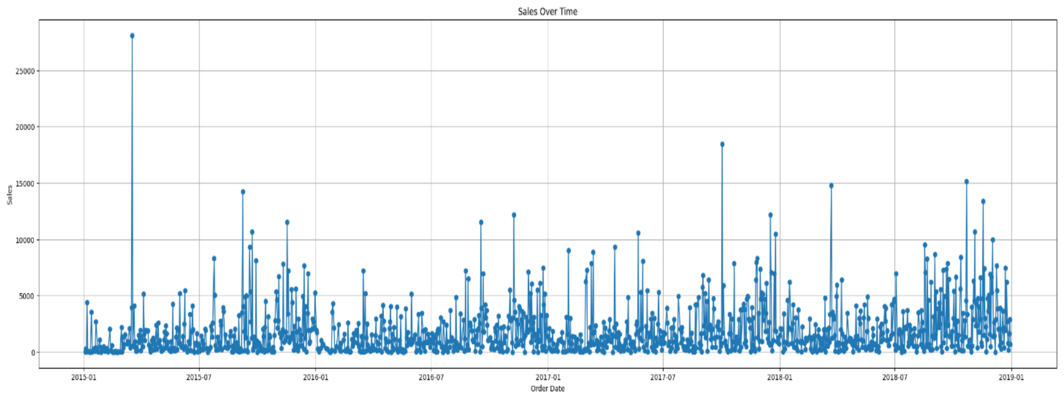


Fig. 4.8 Line Plot of Dataset

This need to check if there are any missing dates on the daily trends and populate those dates with the 0 sales to ensure proper decomposition on the daily view. Also, some models will not work overall if the date trend is not consistent. Total Missing Dates comes out to be 228. Populate missing dates and append to the dataset by adding missing dates to the DataFrame with the Sales number of 0.

## 4.4 Decomposition

Lets look at the daily's and monthy data decompositions for the trends and seasonality. This can immediately see the trend is upwards and heavy in seasonality exists. This is de-composition analysis provides valuable insights, for understanding and predicting future sales patterns by, identifying trends and the seasonal variations. Statsmodels' seasonal_decompose functions, decomposes the monthly- sales data using the additive model. This as separates the time series into four components:

- Trend:- The long-term upward or downward movements in sales.

- Seasonal:- The recurring patterns with the in the year (e.g., higher sales during holidays).
- Residual: The random - fluctuations not captured by trends or seasonality.

Observed: The original monthly sales data.

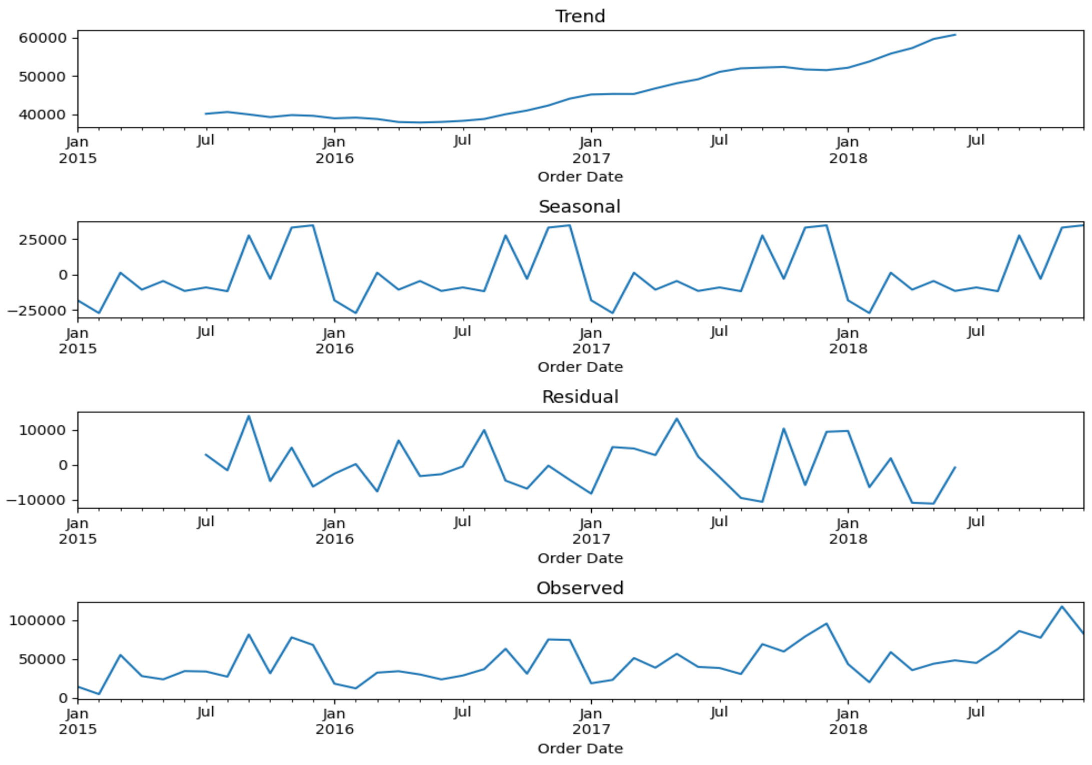


Fig. 4.9  Decompose the time series into monthly components

This image delves , deep into daily sales data by decomposing it into trends and , seasonality, and residual components. It first prepared a copy of the data and ensures the date format is suitabled for time -series  analysis. Then, it is resamples sales figures to daily frequency and the  sums them up to. The core step involves, using Stats-models to de-compose the daily sales data,  into its constituted parts:- the long-terms trend, the recurring yearly -patterns, and the  random fluctuations not captured by the other two. Finally, it is visualizes these components, revealing potentials insights into daily sales-patterns and predicting future-trends. By understanding, how the sales vary on a daily basis, businesses can be optimized inventory management, plan the targeted the promotions, and make the data-driven decisions for overall sales success.

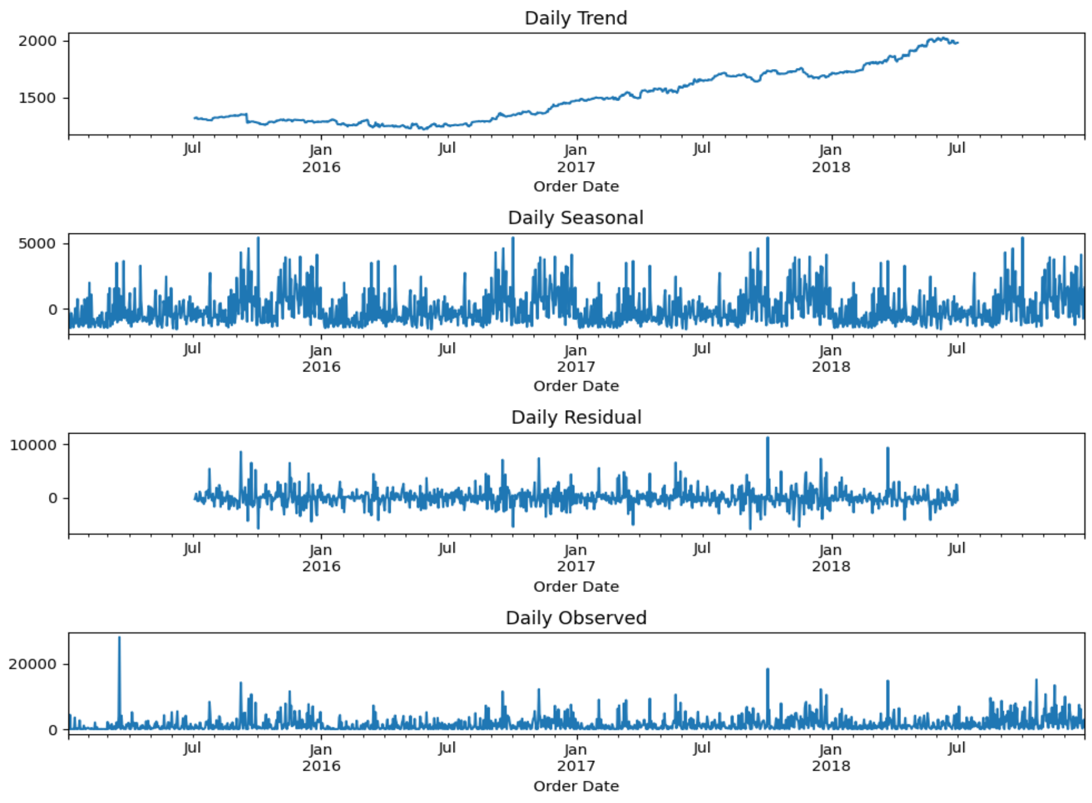


Fig. 4.10 Plot the decomposed component

## 4.5. ACF/PACF Plots

With the in the intricated the land-scape of time series analysis, auto-correlation (ACF) and partial- autocorrelation (PACF) plots, emerge as the powerful instruments, for the unveiling the temporal dependencies hidden with the in the data. These are visualizations served as meticulous guided, illuminates the optimal auto-regressive (A-R) and, moving average (M-A) components necessary to constructs accurate time-series models like ARIMAs and SARIMAs.

Imagine the ACF, as a temporal- cartographer, meticulously mapping of a current observations with the in its past iterations. Each point on the ACF chart, represents the correlation between the present and a designated to the point in the past, revealing that the data's "memory- length." .

The image provided embodied this symphony of whispers and secrets, the ACF and PACF revealing their insights the through their peaks and valleys. Significant correlations at specific lags, like a 12-month echo, unveil the possibility of seasonality.

By deciphering these whispers, you unlock the potentials of ARIMA and selecting the perfect blend of AR with MA components. This gaved a model, faithfully captures the intricate temporal dynamics of the data.
Spikes on lag 0 and 12 point to the seasonality every 12 cycles, which can be also be observed from the decomposition plots.

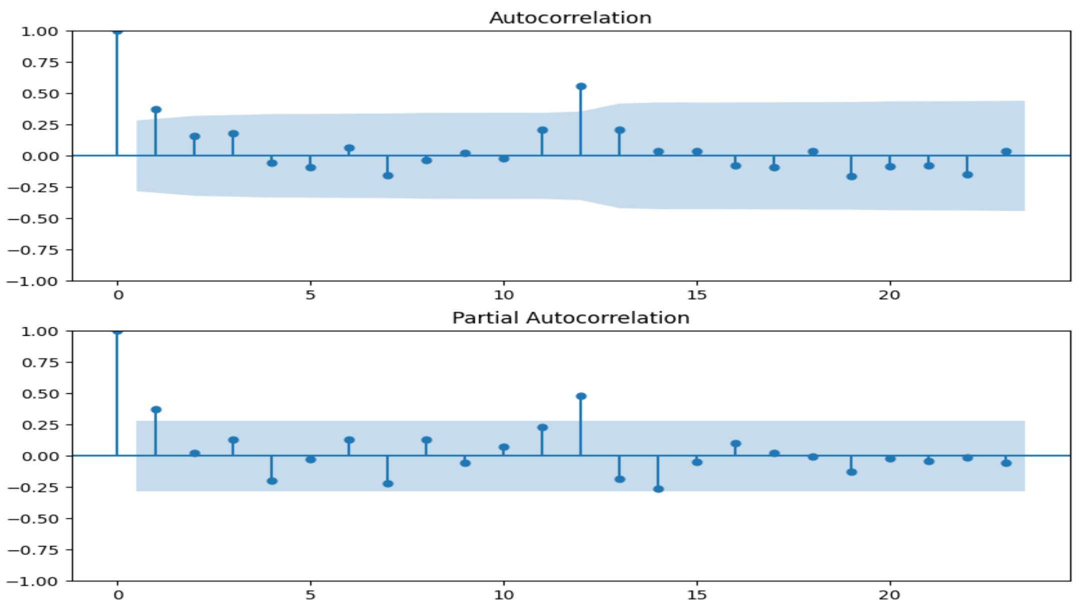
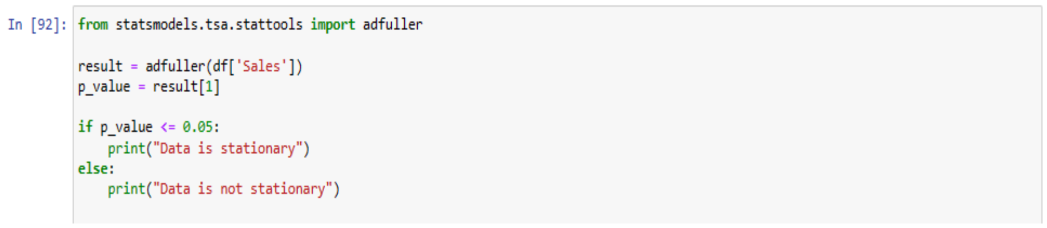


Fig. 4.11 ACF and PACF plots

## 4.6 Checking stationarity on time series

Augmented Dickey-Fuller - (ADF) Test: The ADF test is a statistical tests for stationarity. It tests the null - hypothesis that a unit root is the present in a time - series samples. If the p-value, from the test is less the significance level (e.g., 0.05), rejected the null hypothesis and, consider the data -stationary.



```
In [92]: from statsmodels.tsa.stattools import adfuller

         result = adfuller(df['Sales'])
         p_value = result[1]

         if p_value <= 0.05:
             print("Data is stationary")
         else:
             print("Data is not stationary")
```

### 4.7 Modelling with the  Holt-Winters(Additive Exponential Smoothing)

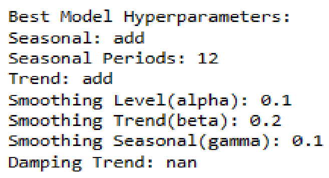
The Holt-Winters model, also known as the Triple Exponential Smoothing model, is the extension of the Holt's Linear Exponential Smoothing model with the added capability of handling seasonality. It is a powerful and widely used time series forecasting method that is particularly useful for data with the both trend and seasonality components.

#### 4.7.1 Model 1 : Fine Tuning Holt Winters using Grid Search

Methods like Grid -Search,  are not compatible with the  statsmodel library of Python , so This establish a typical loop to go through the parameters and, identify what works best. This also do the data split. This will be training and the fitting using the best hyper - parameters, using the test data to forecasts and then This will do throughout of sample fore-cast for the next 12 months. This do the fine tuning on the train data to avoid data leakage. Grid- Search with the  a Twist:-- Since external libraries like GridSearchCV aren't compatible with the  Statsmodels and the code  uses as a manual - loop to explore various combinations of hyper - parameters. The data is then split into training - (80%)  and testing-  (20%)  sets to avoid overfitting and to ensure accurate model evaluation.The loop iterates ,through numerous combinations of seasonal, trend, smoothing, and damping parameters, evaluating each model performance.The model with the  less MSE on the test data is chosen.

##### 4.7.1.1 Decoding the Best Model

The code- prints the hyper - parameters of the winning model, revealing the optimal - configuration for the specific data.In this cases , the best models utilize the  additive-seasonality, with the  a 12-month period, additive -trend, and specific smoothing and the damping- values.



```
Best Model Hyperparameters:
Seasonal: add
Seasonal Periods: 12
Trend: add
Smoothing Level(alpha): 0.1
Smoothing Trend(beta): 0.2
Smoothing Seasonal(gamma): 0.1
Damping Trend: nan
```

### 4.7.1.2 Model Building and  the Evaluation of dataset

This now set up the model, using the best hyper - parameters and scores using r2 square. This also plot actual, fitted, test(forecast) and out of  the samples (extended - forecast) on a graph. It is initializes and fits the Holt-Winters model , with the  chosen hyper - parameters (alpha, beta, gamma, etc.).It calculates R-squared scores for both the test data (0.70) and the training data (0.82), signifying good- fit and predictive accuracy. It is generated forecasts , for the test data (forecasting past performance) and extends the forecast for 12 months beyond the test set (predicting future trends).

A plot , reveals that the actual sales data, the fitted data, the test data forecast, and the extended forecast, showcasing how This will the model captures, historical - patterns and predicts future trends.

This optimized Holt-Winters model, with the  additive seasonality delivers a good fit and accurate forecasts for the monthly sales data.The R-squared scores and the visualization provide insights into the model's performance and the predicted sales trends for the next year. This analysis empathizers , to make informed the business decision,  based on the  anticipated sales fluctuation and seasonality.

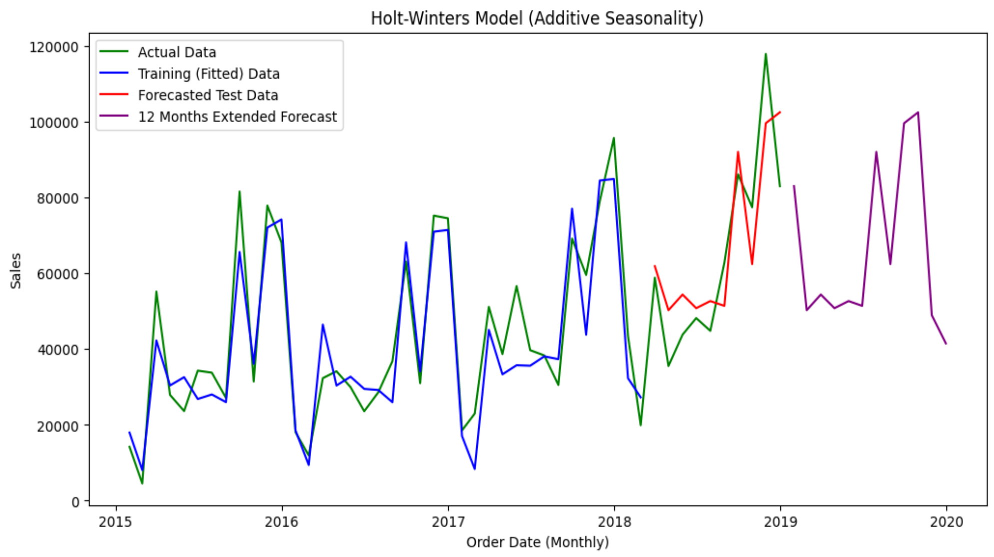


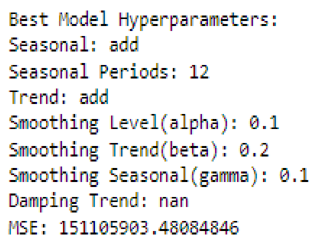
Fig. 4.12  Holt-Winter Model Forecast for 12 months using Grid Search fine tuning

The extended fore-cast, is just the estimate and might not be as accurate as the shorter-termed forecasts. We can be furthered, refine the modelled and improves its performances , by experimenting with the different the hyper - parameter combinations or the other fore-casting methods.

### 4.7.2 Model 2: Fine Tuning Holt Winters using the Simulated Annealing

Simulated Annealing to the identify the optimal hyper - parameters for a Holt-Winters model predicting monthly sales data. The data is first divided into training (80%) and testing (20%) sets. Simulated Annealing, starts with the a random configuration and iteratively explores its neighbours. If a neighbouring configuration offers lower Mean-Squared- Error (MSE) on the testing data, it's accepted as the new best solution. This process mimics the cooling process of the metal, gradually converging towards the configuration with the the lowest MSE, representing the best fit for the model.

The code ultimately identifies the best model hyper - parameters, including seasonality type, seasonal period, trend type, and smoothing/damping factors. It then calculates the final MSE on the testing data. This approach helps fine-tune the Holt-Winters model for the accurate sales - forecasting, considering seasonality and trends with the in the data.



```
Best Model Hyperparameters:
Seasonal: add
Seasonal Periods: 12
Trend: add
Smoothing Level(alpha): 0.1
Smoothing Trend(beta): 0.2
Smoothing Seasonal(gamma): 0.1
Damping Trend: nan
MSE: 151105903.48084846
```

Code applies the Holt-Winters models, with the additive seasonality to forecasted monthly sales data. It uses a grid search approach to identify the optimal the hyper - parameters for the model, including the smoothing parameters for levels, trends, and seasonality (alpha, beta and gamma), and the seasonality period. The chosen one hyper parameters are alpha = 0.1, beta = 0.2, gamma = 0.1, and with a 12-month seasonality period.

The model, is then fitted to the training data and used to generate forecasts for the test data (out-of-sample) and additional 12 months beyond the tests period (extended forecast). The R-squared score for the test data is 0.74, indicating a good fit while the

R-square scores for the training data is 0.87, suggesting that the model, is abled to captured the trends and, seasonality in the historical data's. A plot visualizes the actual sales data, the fitted data for the training period, the fore-casts for the test data, and the extended forecast for the next 12 months. This plot helps to assess the model's performance and identify any potentials issues with the forecasts.

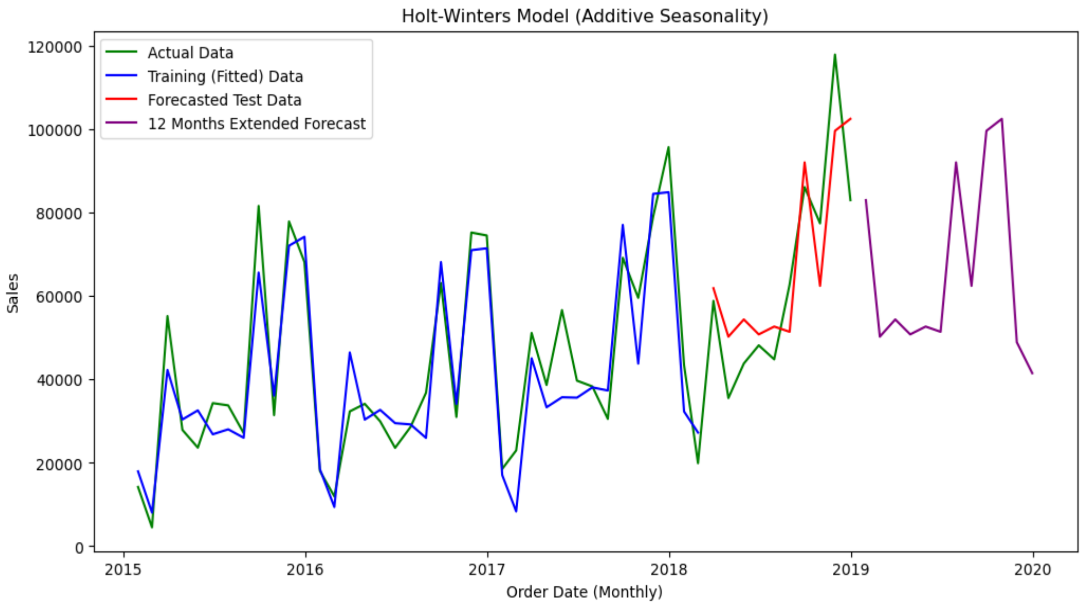


Fig. 4.13 Holt-Winter Model Forecast for 12 months using Simulated Annealing fine tuning

## 4.8 Modelling with the SARIMA

Lets focuses on setting the stage for the SARIMA modelling. It copies the relevant columns ("Order Date" and "Sales") from the DataFrame (df) , to a new one (df_copy) and , then provides summary of its information. The key takeaway is that it is ensures, it have the necessary data structure in place (date-times and numeric sales - values) , before the building the SARIMA model itself. Data Wrangling for the ARIMA that is Pre - processing the data to get it ready, for fine tuning using auto_arima It first - sums daily sales to get monthly values and, focusing on the "Order - Date" and "Sales" columns. Then it resamples the data to the monthly frequency and verifies the aggregation with the a plot. This is pre-processed data with the "Order- Date" as the most suitabled index and "Sales" as the value, is now suitable for auto-arima to automatically identify, the optimal ARIMA parameters for the data.

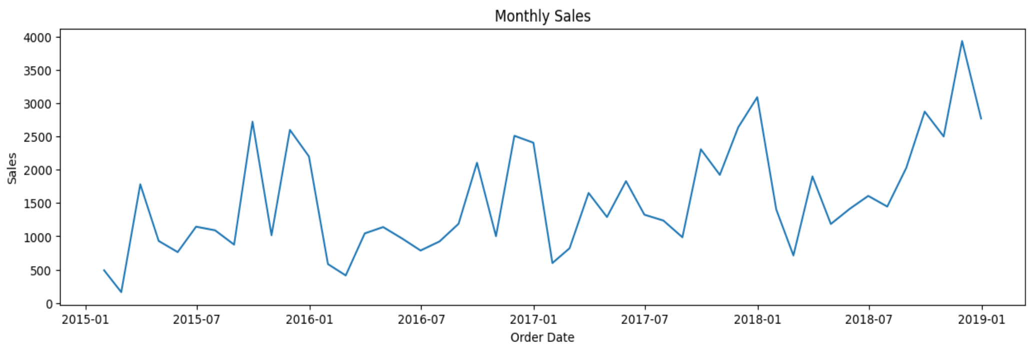


Fig. 4.14 Resample the data on 'Sales' price monthly

### 4.8.1 Model 1: Fine - Tuning of SARIMA using Auto_Arima function

In the context, of the auto_arima for function from the pmdarima library in Python, the AIC (Akaike Information Criterion) , is a statistical- metric used to evaluated the goodness of fit of different ARIMA - models and, to selected the best-fitting model among several candidates.

When using with the auto_arima, the AIC helps to choose the best ARIMA model with the out the need for the manual selection and making it a valuable tool for time-series forecasting. Stepwise search is used to minimize AIC value.

Best model: ARIMA(2,1,4)(2,0,0)[12] intercept

Total fit time: 12.172 seconds

Plugging in best hyper-paramaters into the model. This will fit all the data this time (you can be also try splitting) and , then do the out of samples , forecasts for the next 12- months with the R-squared scores for fitted data is 0.40. This see that model is the not fitting , This will work with the in the first two cycles and does a better jobs fitting after that. Out of sample forecast looks decent and fine.

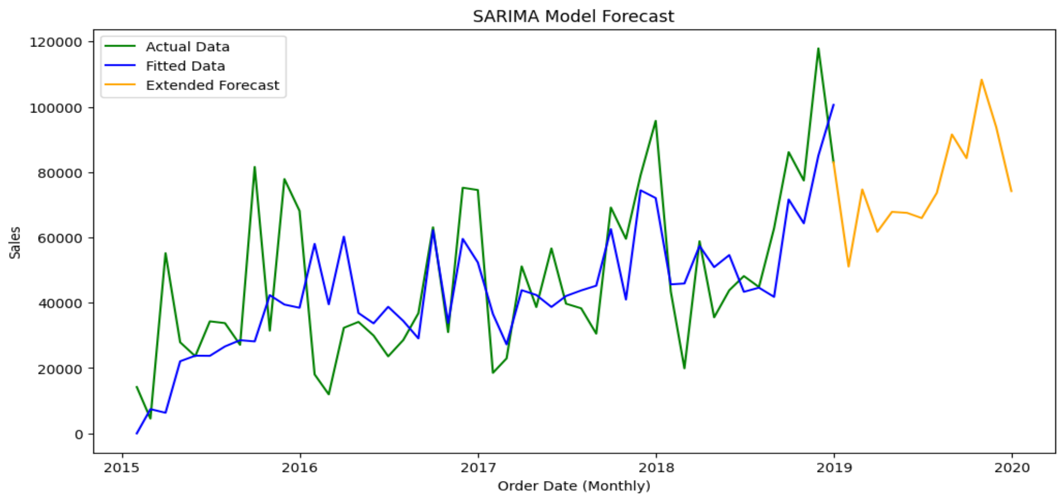


Fig. 4.15 SARIMA Model Forecast for 12 months using auto_arima function

### 4.8.2 Model 2: Fine Tuning of hyperparameter in SARIMA model using the Grid Search

The Python code employs a Seasonal ARIMA model to make a monthly sales forecast. It executes a grid search to find the best hyperparameters that minimize the AIC of the model. The code then fits the SARIMA model using the chosen hyperparameters, generates forecasts for the next twelve months, and plots the actual sales data together with the forecasts.The grid search identified the following hyperparameters as the best configuration: Non-seasonal order: (1, 0, 2) - (p, d, q)

Seasonal order: (1, 0, 0, 12) - (P, D, Q, s)

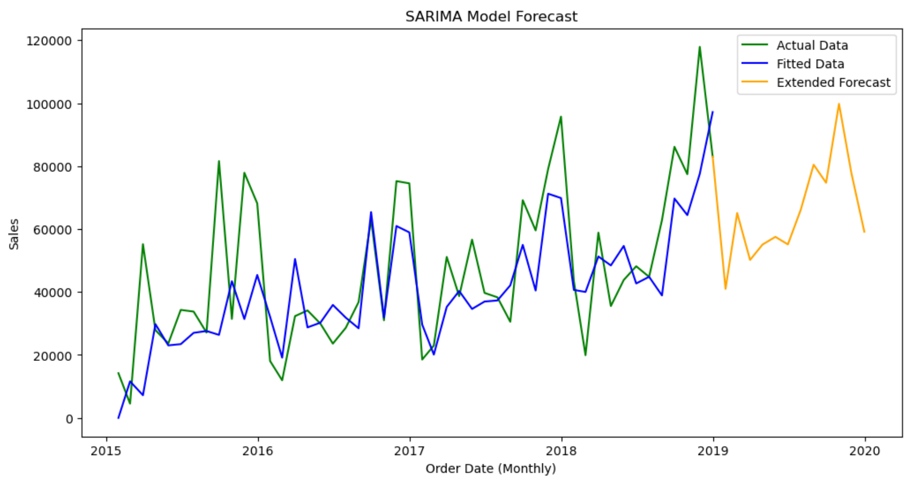


Fig. 4.16 SARIMA Model Forecast for 12 months using grid search fine tuning

## 4.8.3 Model 3: Fine Tuning of hyperparameter in SARIMA model using the Simulated Annealing

The code constructs a hyperparameter tuning procedure for a Seasonal ARIMA model in sales forecasting. It utilizes a Simulated Annealing heuristic to go through a defined search space of hyperparameters and settle on the configuration that minimizes the AIC. After fitting the SARIMA with these optimal hyperparameters, this code generates forecasts over the next twelve months and plots the actual data with forecasts. This approach might improve forecasting accuracy with the proper, data-driven selection of hyperparameters.

Best Model Hyperparameters:
Order: (6, 0, 5)
Seasonal Order: (4, 2, 0, 12)
AIC: 12.0
In the specific code you have, the nonseasonal order is (6, 0, 5) and the seasonal order is (6, 0, 5, 12). That means the model includes six AR terms, no differencing, and five MA terms, with a seasonality period of 12 months—which could represent a yearly seasonality. With the hyperparameters defined, the code then fits the SARIMA model on the prepared data. It uses the fitted model in generating the out-of-sample forecast for the next twelve months. It also calculates the R-squared score, a statistical measure that indicates how well the fitted model explains the variation in the actual data which comes as 0.61. A higher R-squared score indicates a good fit.

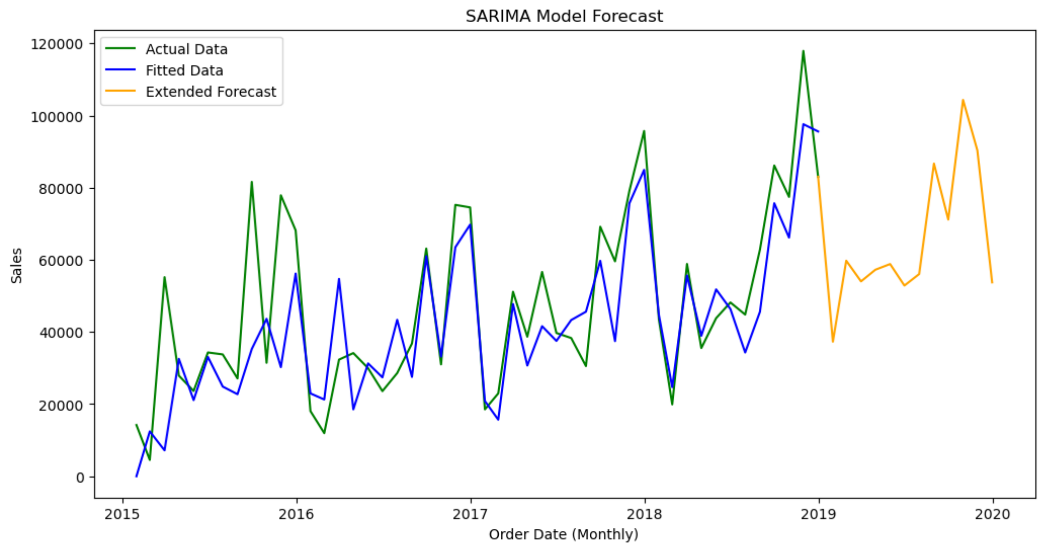


Fig. 4.17: SARIMA Model Forecast for 12 months using simulated annealing

### 4.8.2 Residual Diagnostics of the best model of SARIMA

In general, from the residual analysis of the simulated annealing SARIMA model, it can be observed that the residuals from this model are mostly random and meet the assumptions for normality. Here is the detailed deconstruction of observations from the graphs:

1. Standardized Residual Plot: The residuals are randomly scattered around the zero line with no identifiable pattern. This indicates that the errors are independent and identically distributed, an assumption that needs to be met by a SARIMA model.

2. Histogram: The distribution of the residuals is near a normal distribution, which continues to support the assumption of normality.

3. Normal Q-Q Plot: The points in the Q-Q plot generally follow a straight diagonal line, indicating that the residuals are normally distributed.

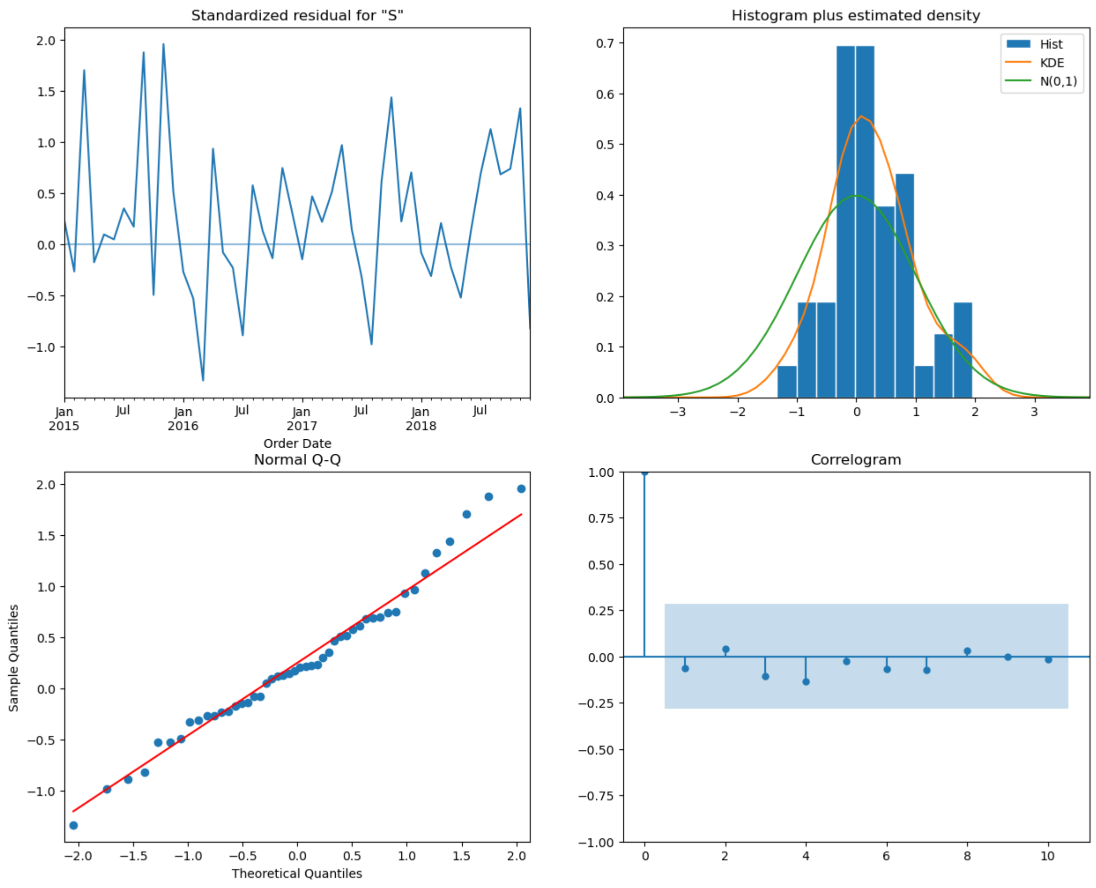


Fig.4.18: Residual Analysis of the best model of the SARIMA(Simulated Annealing)

## 4.9 Modelling using with the Prophet

Prophet is a very powerful algorithm, that does a great job to capture the seasonality in most cases. Employs the Prophet forecasting model to predict the monthly sales for the next year. It ensures the "Order Date" is in date- time format and resamples the data to monthly sums. It is renames columns to match Prophet's requirements ("ds" for date and "y" for sales). The Prophet modelled, is initialized and trained on the prepared data. A future data-frame is created, for the next 12 months.

The model is forecasts future sales for this period. The code extracts actual, fitted (historical), and forecasted- sales data. It is calculates the R-squared scored card for the fitted data of the base model which is 0.91, indicating its accuracy with the base model with no hyper parameter fine tuning. Finally, it will plots the actual, fitted, and forecasted sales, and visualizing the model's performance and future predictions. This can be the adjust in the forecasts to start from where, the trains or actual data ends, for a better looking forecast.

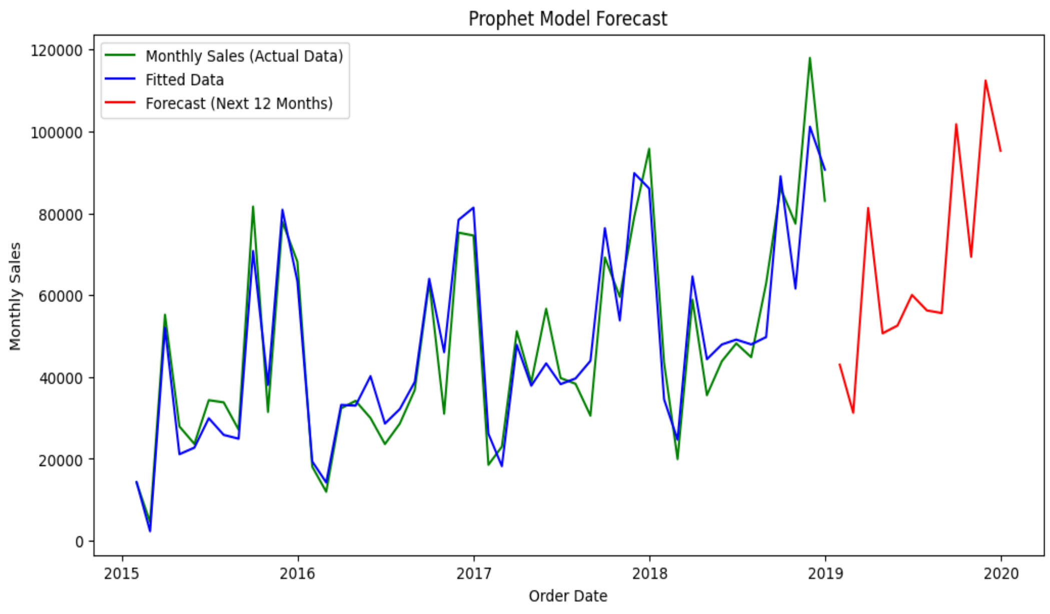


Fig. 4.19 :- Plot the actual data, fitted data, and forecast for the next 12 months by FbProphet

### 4.9.1 Fine tuning of hyperparameter Tuning using Grid Search Optimisation

This code is for monthly sales forecasting using the Prophet model and grid search to find the best hyperparameter configuration. It will first prepare the data by resampling the data to a monthly frequency and summing sales for every month. Then, a grid search is used to try a variety of hyperparameters for the Prophet model, including changepoint_prior_scale, seasonality_prior_scale, holidays_prior_scale, and seasonality_mode. Fitting the model for each set of hyperparameters and then selecting the one that results in the highest R-squared score for the historical data, the forecasts for the next twelve months are generated using the Prophet model with the best hyperparameters. Results: Grid search should yield an R-squared score of 0.92 on the fitted data, indicating the selected hyperparameters offer a model that tends to fit well. Most probably, a chart accompanies the output of this code that usually includes the actual sales data, the fitted values for the historical period, and the forecasted values for the next twelve months.

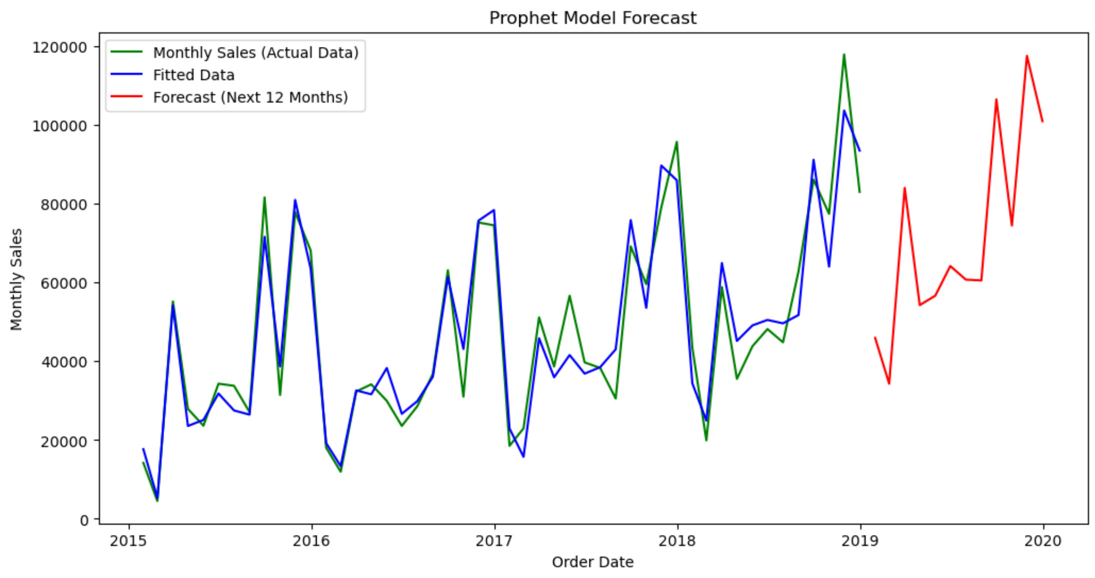


Fig. 4.20 : Plot the actual data, fitted data, and forecast for the next 12 months by FbProphet using grid search hyperparameter fine tuning

### 4.9.2 Fine tuning of hyperparameter using Simulated Annealing Optimisation

This code uses simulated annealing to optimize the hyperparameters for the Prophet forecasting model. Simulated annealing is a probabilistic technique that finds the minimum of a function; in this case, the negative R-squared score is a measure of how well the model fits. This is implemented in a class called ProphetAnnealer, where the function energy() is used to compute the negative R-squared score for a given set of hyperparameters. These parameters are part of the optimal set of hyperparameters for the Prophet model that have been returned by the simulated annealing optimization process: changepoint_prior_scale = 1, seasonality_prior_scale = 2.98, holidays_prior_scale = 1.90, and mode of seasonality = 'additive'. The model achieved a best R-squared score of 0.94 when evaluated on the historical data, which is considered to be a decent fit of the model to the data.. Visualizations are developed to display the actual data, fitted data, and forecasts.

These results show that simulated annealing did locate a set of good hyperparameters—R-squared score of 0.94 on fitted data—therefore, the model has fitted the historical sales data trends.

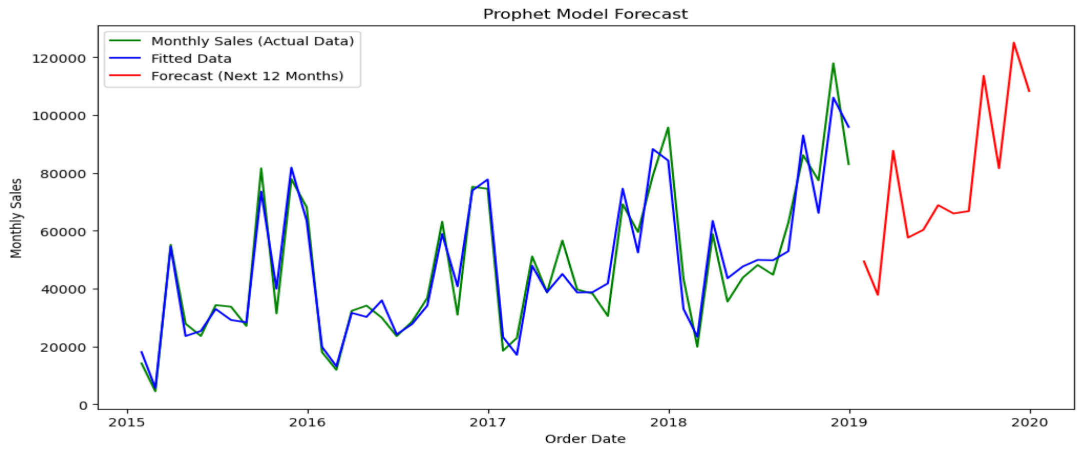


Fig.4.21 Plot the actual data, fitted data, and forecast for the next 12 months by FbProphet using simulated annealing hyperparameter fine tuning

# CHAPTER 5

# RESULT AND CONCLUSION

The exploration shows that machine learning holds immense potential in supply chain management, from supplier classification to stock-out prediction and business partner identification. In this exciting landscape, this thesis delves into sales forecasting, a vital tool for financial and logistics planning in any organization. It harnesses historical sales data to predict future sales, empathizing with informed decision-making.

To tackle this challenge, this is focused on analyzing time series datasets and recognizing the crucial role of seasonality in sales structure. The Python script proved invaluable, offering a platform to compare and evaluate various forecasting algorithms for their suitability to the specific data. Through this comparison, this pitted Facebook Prophet against established statistical models like SARIMA, Holt-Winters, and Facebook Prophet. The results are insightful.

A comparative study of time-series forecasting models reveals significant variations in performance measures. Facebook Prophet has the highest accuracy out of all of them. Facebook Prophet's R-squared value was 0.94, meaning that, in comparison to the other models, it suited the data the best. Additionally, it has the lowest MAE (5036.083), RMSE (62660.693), and MAPE (12.09) values, demonstrating very good prediction accuracy. With an R-squared of 0.87, an MAE of 6657.915, an RMSE of 8238.96, and a MAPE of 19.12, Holt-Winters fared rather well; as such, it can be a very good alternative to achieving accurate forecasting. With an R-squared of 0.61, MAE of 10858.84, RMSE of 15418.17, and MAPE of 28.46, the SARIMA model fared poorly in comparison to Facebook Prophet and Holt-Winters. The selection of the hyperparameter tuning technique and the right models for forecasting are crucial, as the findings demonstrate. Facebook Prophet's top performance in this study can be partly attributed to its optimization through simulated annealing. As a result, it might produce extremely precise sales projections. The investigation confirms that choosing and fine-tuning the right model will always improve forecasting, making it feasible to make wise decisions for business operations.

| Time-series forecasting models are used in sales forecasting. | R-Squared Value (Fitted) | MAE (Fitted) | RMSE (Fitted) | MAPE (Fitted) |
|---|---|---|---|---|
| 1. SARIMA(Best Result using Simulated Annealing in Fine Tuning) | 0.61 | 10858.84 | 15418.17 | 28.46 |
| 2. Holt- Winter(Best Result using Simulated Annealing in fine Tuning) | 0.87 | 6657.91 | 8238.96 | 19.12 |
| 3. Facebook Prophet(Best Result using Simulated Annealing in Fine Tuning) | 0.94 | 5036.08 | 6260.69 | 12.09 |

Table 5.1 Model Performance Matrices

Further, the Python script stands out as a versatile tool, capable of running multiple models and algorithms simultaneously. In conclusion, the script presents a readily implementable solution for real-world sales forecasting. Whether applied by corporations managing product inventory or logistics providers anticipating client needs, this tool leverages historical data to predict future sales, optimize space, transportation, and ultimately, success.

# CHAPTER 6

# FUTURE SCOPE

The future of time series forecasting is brimming with the exciting possibilities, promising even more accurate and versatile approaches to navigating the complexities of future prediction. Here are some key trends that are shaping the landscape:

1. Deep Learning Revolution: Machine learning, especially deep learning, is already transforming the field, with the recurrent neural networks (RNNs) and their variants like long short-term memory (LSTMs) and convolutional neural networks (CNNs) demonstrating remarkable forecasting process. As research delves deeper into advanced architectures and interpretability methods, deep learning models will likely become even more sophisticated and widely adopted.
2. Embracing Hybrid Solutions: While deep learning shines in capturing hidden patterns, its "black box" nature can be a concern. Future trends will likely see hybrid approaches that combine the strengths of deep learning with the traditional statistical models like ARIMA, allowing for explainable forecasts and leveraging interpretability for model refinement.
3. Real-time and Streaming Data: The rise of real-time and streaming data necessitates adaptive forecasting models that can be continuously learn and update based on incoming data. This opens up a vast array of applications in areas like traffic prediction, financial market analysis, and industrial process control.

# REFERENCES

1. Afroz Chakure. (2019) Facebook Prophet Regression Along with its implementation in Python. Medium (The startup). Weblog [Online] 29 Jun. Available from: https://medium.com/swlh/random-forest-and-its-implementation-71824ced454f. [Accessed 27/1/2024].
2. Andreea-Cristina PETRICĂ, Stelian STANCU and Alexandru TINDECHE (2016) Limitation of ARIMA models in financial and monetary economics. Theoretical and Applied Economics Volume XXIII, No. 4(609), Winter, pp. 19-42
3. Aurélien Géron (2017) *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. O'Reilly Media, Inc. ISBN: 9781491962299 .
4. Chakrabarti, Arijit, and Jayanta K. Ghosh. "AIC, BIC and Recent Advances in Model Selection." Elsevier eBooks, 2011, pp. 583–605. https://doi.org/10.1016/b978-0-444-51862-0.50018-6.
5. Chatfield, Chris. "Time-Series Forecasting." Chapman and Hall/CRC eBooks, 2000, https://doi.org/10.1201/9781420036206.
6. Chatfield, Chris. "Time-Series Forecasting." Chapman and Hall/CRC eBooks, 2000, https://doi.org/10.1201/9781420036206.
7. Djordje Cica, Branislav Sredanovic, Sasa Tesic and Davorin Kramar. (2020) Predictive modelling of turning operations under different cooling/lubricating conditions for sustainable manufacturing with machine learning techniques. Published in Applied Computing and Informatics. 2210-8327 DOI 10.1016/j.aci.2020.02.001
8. Ethem Alpaydın (2010) *Introduction to Machine Learning*. 2nd ed. The MIT Press Cambridge, Massachusetts London, England
9. Fattah, Jamal & Ezzine, Latifa & Aman, Zineb & Moussami, Haj & Lachhab, Abdeslam. (2018). Forecasting of sales using ARIMA model. *International Journal of Engineering Business Management*. 10. 184797901880867. 10.1177/1847979018808673.
10. Fildes, R. A. "Time series analysis and forecasting: The Box-Jenkins approach." Long Range Planning, vol. 9, no. 6, Elsevier BV, Dec. 1976, p. 113. Crossref, https://doi.org/10.1016/0024-6301(76)90018-2.
11. Forecasting: Principles and Practice (3rd ed). otexts.com/fpp3.

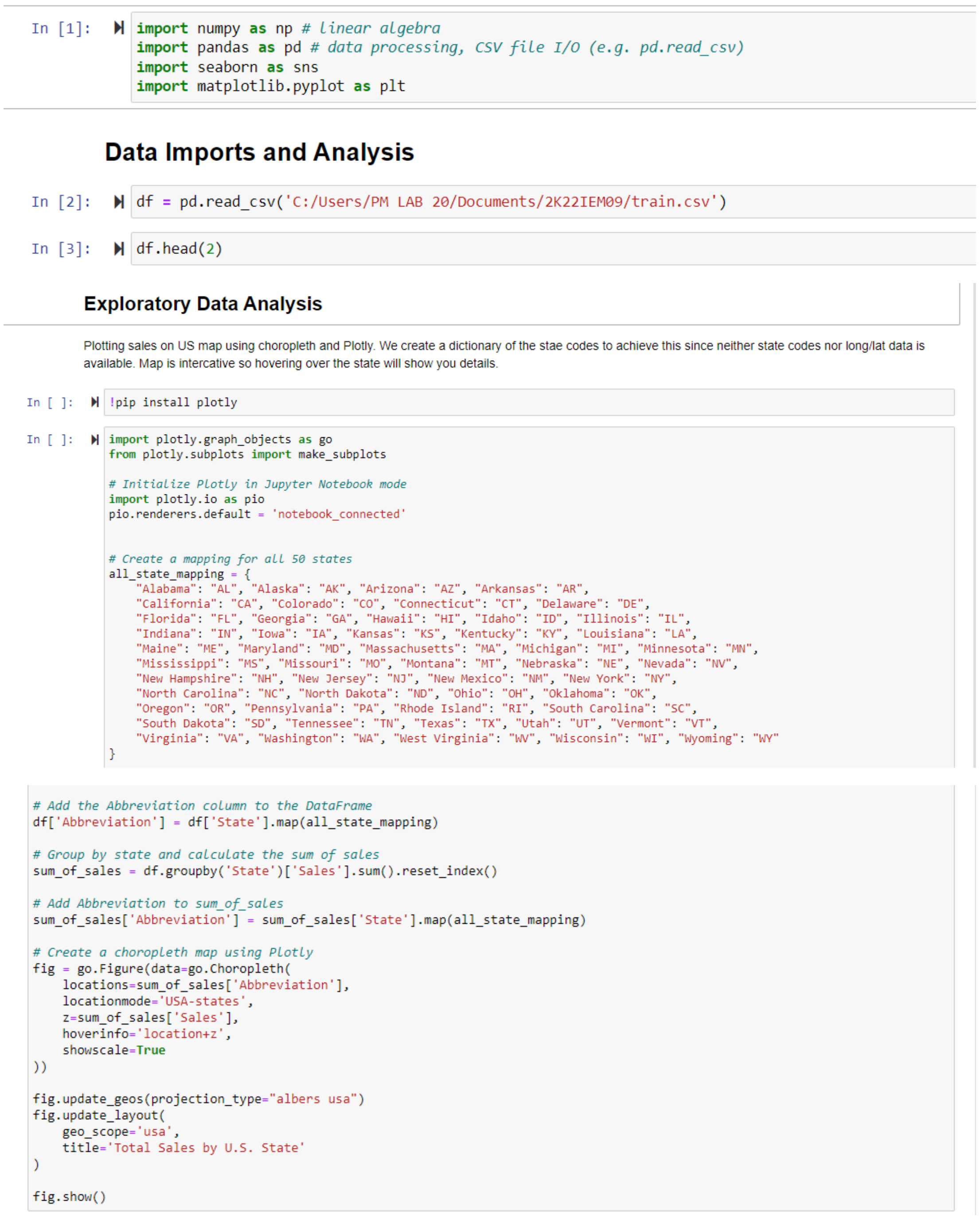
12. Henderson, Darrall, et al. "The Theory and Practice of Simulated Annealing." Kluwer Academic Publishers eBooks, 2006, pp. 287–319. https://doi.org/10.1007/0-306-48056-5_10.
13. Hyndman, R.J., & Athanasopoulos, G. (2018) Forecasting: principles and practice, 2nd edition, OTexts: Melbourne, Australia. OTexts.com/fpp2.
14. Jason Brownlee (2019) A TThe of Machine Learning Algorithms [WWW] Available from: https://machinelearningmastery.com/a-tThe-of-machine-learning-algorithms/ [Accessed 20/11/2023]
15. Mahya Seyedan and Fereshteh Mafakheri (2020) Predictive big data analytics for supply chain sales FORECASTING: methods, applications, and research opportunities. J Big Data 7:53 https://doi.org/10.1186/s40537-020-00329-2
16. Makkar, Sandhya & G.Naga Rama Devi, Dr & Solanki, Vijender. (2020). Applications of Machine Learning Techniques in Supply Chain Optimization. 10.1007/978-981-13-8461-5_98.
17. Makridakis, Spyros & Hyndman, Rob & Petropoulos, Fotios. (2019). Forecasting in social settings: The state of the art. International Journal of Forecasting. 36. 10.1016/j.ijforecast.2019.05.011.
18. Malki, Amer, et al. "SARIMA model-based forecasting required number of COVID-19 vaccines globally and empirical analysis of peoples' view towards the vaccines." Alexandria Engineering Journal /Alexandria Engineering Journal, vol. 61, no. 12, Dec. 2022, pp. 12091–110. https://doi.org/10.1016/j.aej.2022.05.051.
19. Ervural, Beyzanur Cayir, et al. "Model Estimation of ARMA Using Genetic Algorithms: A Case Study of Forecasting Natural Gas Consumption." Procedia: Social & Behavioral Sciences, vol. 235, Nov. 2016, pp. 537–45. https://doi.org/10.1016/j.sbspro.2016.11.066.
20. "Prophet." Prophet, facebook.github.io/prophet.
21. Marjan, Čeh & Kilibarda, Milan & Lisec, Anka & Bajat, Branislav. (2018). Estimating the Performance of Facebook Prophetversus Multiple Regression for Predicting Prices of the Apartments. ISPRS International Journal of Geo-Information. 7. 168. 10.3390/ijgi7050168.
22. Matplotlib documentation — Matplotlib 3.9.0 documentation. matplotlib.org/stable/index.html.
23. Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. (2018) *Foundations of Machine Learning.* 2nd ed. MIT Press
24. NumPy documentation — NumPy v2.1.dev0 Manual. numpy.org/devdocs.

25. Ouedraogo, I., Defourny, P. & Vanclooster, M. (2019) Application of Facebook Prophet regression and comparison of its performance to multiple linear regression in modeling groundwater nitrate concentration at the African continent scale. Hydrogeol J 27, 1081–1098. https://doi.org/10.1007/s10040-018-1900-5
26. pandas documentation — pandas 2.2.2 documentation. pandas.pydata.org/docs.
27. Peixeiro, Marco. "Time Series Forecasting in Python." O'Reilly Online Learning,www.oreilly.com/library/view/time-series forecasting/9781617299889.
28. Plotly. plotly.com/python.
29. Ratnadip Adhikari, R. K. Agrawal (2013) An Introductory Study on Time Series Modeling and Forecasting. LAP Lambert Academic Publishing, Germany
30. SciPy documentation — SciPy v1.13.1 Manual. docs.scipy.org/doc/scipy/index.html.
31. Shadkam, A. (2020). Using SARIMA to forecast electricity sales and consumption in university buildings (T). University of British Columbia. Retrieved from https://open.library.ubc.ca/collections/ubcourses/24/items/1.0391009
32. Sirisha, Uppala Meena, et al. "Profit Prediction Using ARIMA, SARIMA and LSTM Models in Time Series Forecasting: A Comparison." IEEE Access, vol. 10, Jan. 2022, pp. 124715–27. https://doi.org/10.1109/access.2022.3224938.
33. Pongdatu, G. a. N., & Putra, Y. H. (2018). Seasonal Time Series Forecasting using SARIMA and Holt Winter's Exponential Smoothing. IOP Conference Series. Materials Science and Engineering, 407, 012153. https://doi.org/10.1088/1757-899x/407/1/012153
34. Sirisha, Uppala Meena, et al. "Profit Prediction Using ARIMA, SARIMA and LSTM Models in Time Series Forecasting: A Comparison." IEEE Access, vol. 10, Jan. 2022, pp. 124715–27. https://doi.org/10.1109/access.2022.3224938.
35. statsmodels 0.14.1. www.statsmodels.org/stable/index.html.
36. Tinni Chaudhuri, Prashant Verma, Mukti Khetan (2020) Modeling and Forecasting of Rice Production in Some Major States of India using ARIMA. International Journal for Research in Applied Science & Engineering Technology (IJRASET) ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429 Volume 8 Issue XII
37. Wenzel, Hannah; Smit, Daniel; Sardesai, Saskia (2019): A literature review on machine learning in supply chain management, In: Kersten, Wolfgang Blecker, Thorsten Ringle, Christian M. (Ed.): Artificial Intelligence and Digital Transformation in Supply Chain Management: Innovative Approaches for Supply Chains. Proceedings of the Hamburg International Conference of Logistics (HICL), Vol. 27, ISBN 978-3-7502-4947-9, epubli GmbH, Berlin, pp. 413-441, http://dx.doi.org/10.15480/882.2478.

38. Graves, A. (2020). Time Series Forecasting with a SARIMA model. URL https:// https://towardsdatascience.com/time-series-forecasting-with-a-sarima-modeldb051b7ae459
39. Brownlee, J. (2020). Hyperparameter Optimization With Random Search and Grid Search. URL https://machinelearningmastery.com/how-to-grid-search-sarima-model/-hyperparameters-for-time-series-forecasting-in-python/
40. Brownlee, J. (2018). How to Grid Search SARIMA Hyperparameters for Time Series Forecasting. URL https://machinelearningmastery.com/how-to-grid-search-sarima-model/- hyperparameters-for-time-series-forecasting-in-python/
41. Bhandare, Yash Sameer, and Mohammadreza Hajiarbabi. Hyperparameter Tuning with Simulated Annealing and Genetic Algorithm. Jan. 2023, https://doi.org/10.2139/ssrn.4432719.
42. X. Zhu and M. Shen, (2012) ”Based on the ARIMA model with grey theory for short term load forecasting model," *International Conference on Systems and Informatics*, Yantai, pp. 564-567, doi: 10.1109/ICSAI.2012.6223060.
43. Ziegel, E. R., Box, G., Jenkins, G., & Reinsel, G. (1995). Time Series Analysis, Forecasting, and Control. Technometrics, 37(2), 238. https://doi.org/10.2307/1269640.
44. Ziegel, Eric R., et al. “Time Series Analysis, Forecasting, and Control.” Technometrics, vol. 37, no. 2, JSTOR, May 1995, p. 238. Crossref, https://doi.org/10.2307/1269640.

# APPENDICES

Fig.1 Importing Python libraries on the Jupyter Notebook software. Python libraries are pre-written code snippets that offer tools and reusable functions for common tasks.



```
In [1]:  import numpy as np # linear algebra
         import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
         import seaborn as sns
         import matplotlib.pyplot as plt
```

## Data Imports and Analysis

```
In [2]:  df = pd.read_csv('C:/Users/PM LAB 20/Documents/2K22IEM09/train.csv')
```

```
In [3]:  df.head(2)
```

### Exploratory Data Analysis

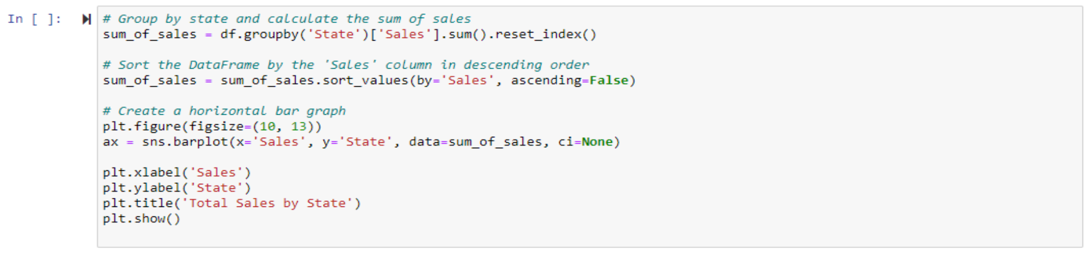
Plotting sales on US map using choropleth and Plotly. We create a dictionary of the stae codes to achieve this since neither state codes nor long/lat data is available. Map is intercative so hovering over the state will show you details.

```
In [ ]:  !pip install plotly
```

```
In [ ]:  import plotly.graph_objects as go
         from plotly.subplots import make_subplots

         # Initialize Plotly in Jupyter Notebook mode
         import plotly.io as pio
         pio.renderers.default = 'notebook_connected'


         # Create a mapping for all 50 states
         all_state_mapping = {
             "Alabama": "AL", "Alaska": "AK", "Arizona": "AZ", "Arkansas": "AR",
             "California": "CA", "Colorado": "CO", "Connecticut": "CT", "Delaware": "DE",
             "Florida": "FL", "Georgia": "GA", "Hawaii": "HI", "Idaho": "ID", "Illinois": "IL",
             "Indiana": "IN", "Iowa": "IA", "Kansas": "KS", "Kentucky": "KY", "Louisiana": "LA",
             "Maine": "ME", "Maryland": "MD", "Massachusetts": "MA", "Michigan": "MI", "Minnesota": "MN",
             "Mississippi": "MS", "Missouri": "MO", "Montana": "MT", "Nebraska": "NE", "Nevada": "NV",
             "New Hampshire": "NH", "New Jersey": "NJ", "New Mexico": "NM", "New York": "NY",
             "North Carolina": "NC", "North Dakota": "ND", "Ohio": "OH", "Oklahoma": "OK",
             "Oregon": "OR", "Pennsylvania": "PA", "Rhode Island": "RI", "South Carolina": "SC",
             "South Dakota": "SD", "Tennessee": "TN", "Texas": "TX", "Utah": "UT", "Vermont": "VT",
             "Virginia": "VA", "Washington": "WA", "West Virginia": "WV", "Wisconsin": "WI", "Wyoming": "WY"
         }
```

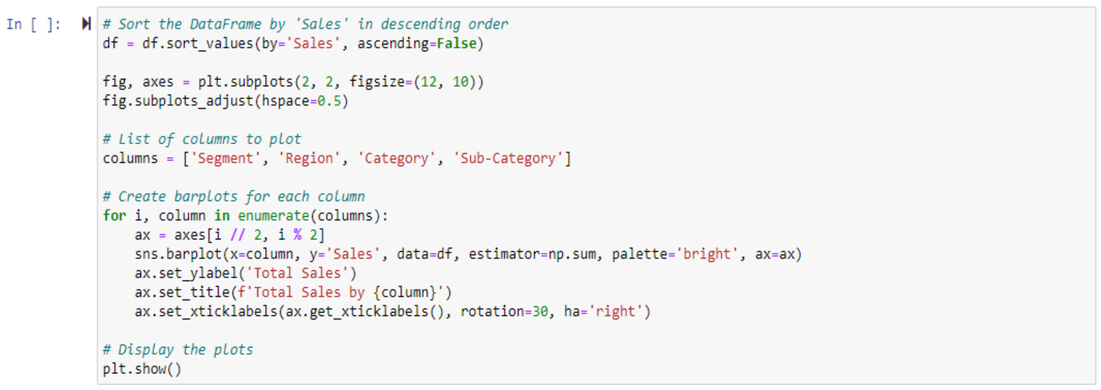
```
# Add the Abbreviation column to the DataFrame
df['Abbreviation'] = df['State'].map(all_state_mapping)

# Group by state and calculate the sum of sales
sum_of_sales = df.groupby('State')['Sales'].sum().reset_index()

# Add Abbreviation to sum_of_sales
sum_of_sales['Abbreviation'] = sum_of_sales['State'].map(all_state_mapping)

# Create a choropleth map using Plotly
fig = go.Figure(data=go.Choropleth(
    locations=sum_of_sales['Abbreviation'],
    locationmode='USA-states',
    z=sum_of_sales['Sales'],
    hoverinfo='location+z',
    showscale=True
))

fig.update_geos(projection_type="albers usa")
fig.update_layout(
    geo_scope='usa',
    title='Total Sales by U.S. State'
)

fig.show()
```

## Fig. 2 Exploratory Data Analysis


In [ ]:

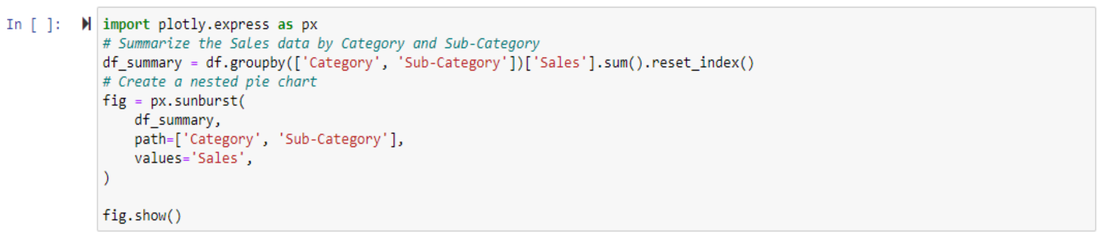
```
# Group by state and calculate the sum of sales
sum_of_sales = df.groupby('State')['Sales'].sum().reset_index()

# Sort the DataFrame by the 'Sales' column in descending order
sum_of_sales = sum_of_sales.sort_values(by='Sales', ascending=False)

# Create a horizontal bar graph
plt.figure(figsize=(10, 13))
ax = sns.barplot(x='Sales', y='State', data=sum_of_sales, ci=None)

plt.xlabel('Sales')
plt.ylabel('State')
plt.title('Total Sales by State')
plt.show()
```

Visualizing some of the other categorical columns

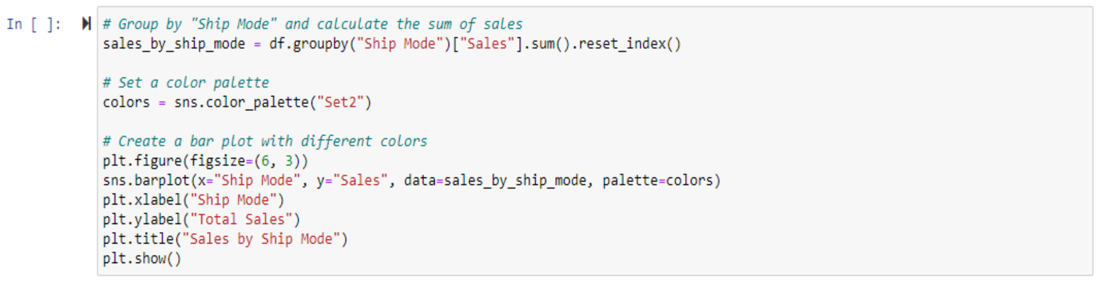

In [ ]:

```
# Sort the DataFrame by 'Sales' in descending order
df = df.sort_values(by='Sales', ascending=False)

fig, axes = plt.subplots(2, 2, figsize=(12, 10))
fig.subplots_adjust(hspace=0.5)

# List of columns to plot
columns = ['Segment', 'Region', 'Category', 'Sub-Category']

# Create barplots for each column
for i, column in enumerate(columns):
    ax = axes[i // 2, i % 2]
    sns.barplot(x=column, y='Sales', data=df, estimator=np.sum, palette='bright', ax=ax)
    ax.set_ylabel('Total Sales')
    ax.set_title(f'Total Sales by {column}')
    ax.set_xticklabels(ax.get_xticklabels(), rotation=30, ha='right')

# Display the plots
plt.show()
```

Here is a nested pie chart(sunburst) showing linkage between Category and Subcategory using plotly. Chart is interactive so hovering over will give you details.Click/unclick on the Categories to see how chart details change.

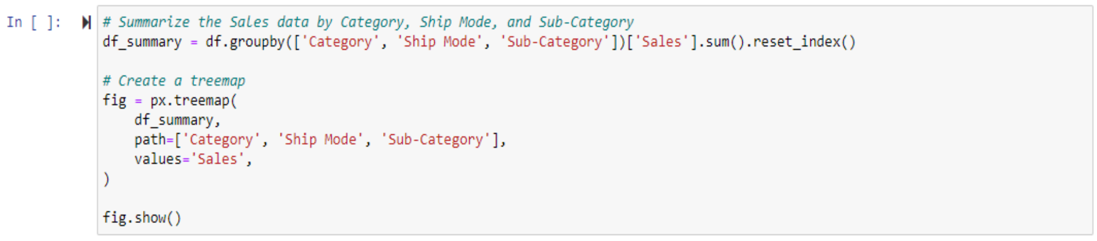

In [ ]:

```
import plotly.express as px
# Summarize the Sales data by Category and Sub-Category
df_summary = df.groupby(['Category', 'Sub-Category'])['Sales'].sum().reset_index()
# Create a nested pie chart
fig = px.sunburst(
    df_summary,
    path=['Category', 'Sub-Category'],
    values='Sales',
)

fig.show()
```


In [ ]:

```
# Group by "Ship Mode" and calculate the sum of sales
sales_by_ship_mode = df.groupby("Ship Mode")["Sales"].sum().reset_index()

# Set a color palette
colors = sns.color_palette("Set2")

# Create a bar plot with different colors
plt.figure(figsize=(6, 3))
sns.barplot(x="Ship Mode", y="Sales", data=sales_by_ship_mode, palette=colors)
plt.xlabel("Ship Mode")
plt.ylabel("Total Sales")
plt.title("Sales by Ship Mode")
plt.show()
```

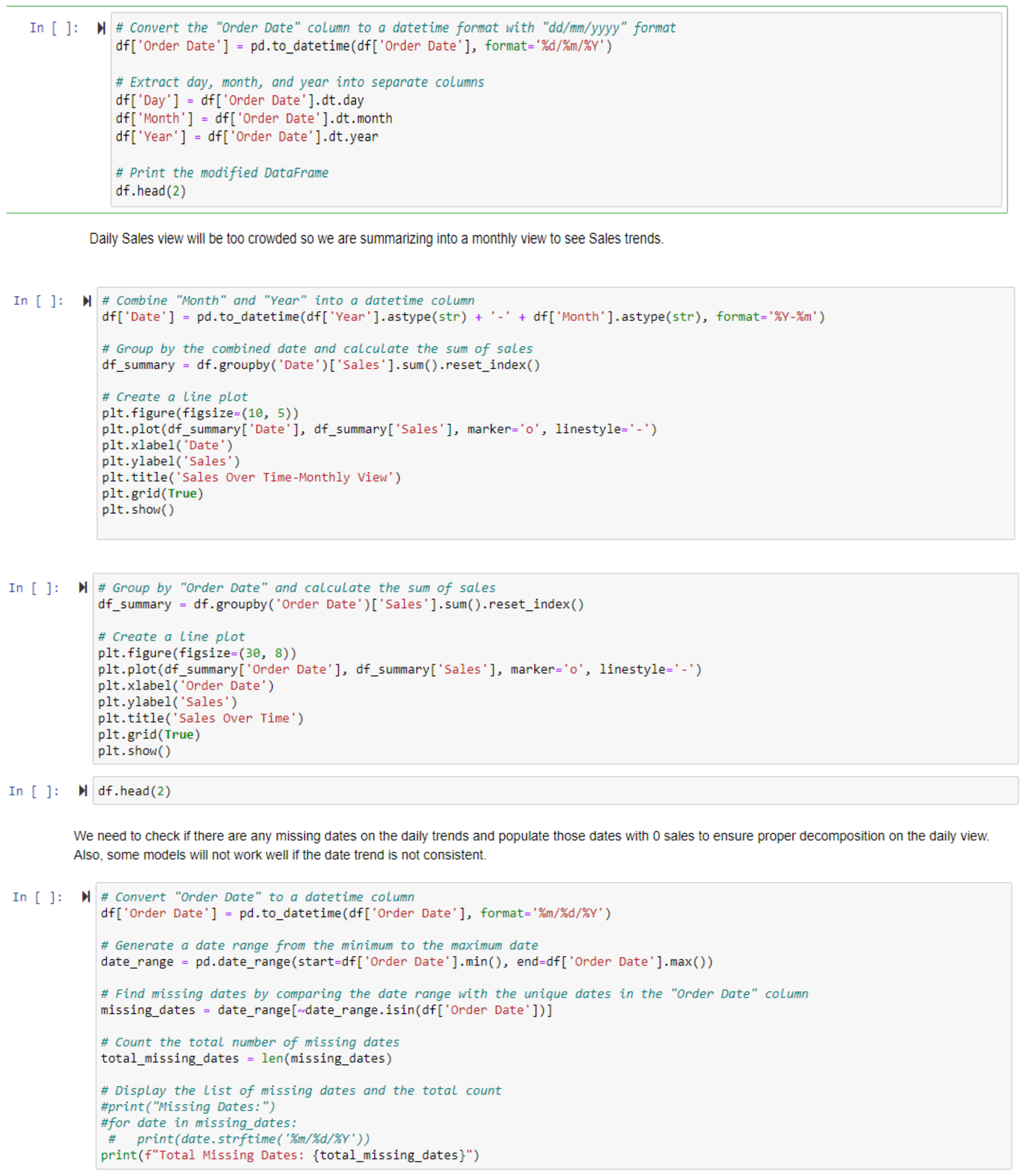
This is a treemap that shows the linkage between Shipping Mode, Category and Sub Category. Interactive.


In [ ]:

```
# Summarize the Sales data by Category, Ship Mode, and Sub-Category
df_summary = df.groupby(['Category', 'Ship Mode', 'Sub-Category'])['Sales'].sum().reset_index()

# Create a treemap
fig = px.treemap(
    df_summary,
    path=['Category', 'Ship Mode', 'Sub-Category'],
    values='Sales',
)

fig.show()
```

We will create few more columns splitting the Order Date column.

Fig. 3 Exploratory Data Analysis Code-2



```
In [ ]:  # Convert the "Order Date" column to a datetime format with "dd/mm/yyyy" format
         df['Order Date'] = pd.to_datetime(df['Order Date'], format='%d/%m/%Y')

         # Extract day, month, and year into separate columns
         df['Day'] = df['Order Date'].dt.day
         df['Month'] = df['Order Date'].dt.month
         df['Year'] = df['Order Date'].dt.year

         # Print the modified DataFrame
         df.head(2)
```

Daily Sales view will be too crowded so we are summarizing into a monthly view to see Sales trends.

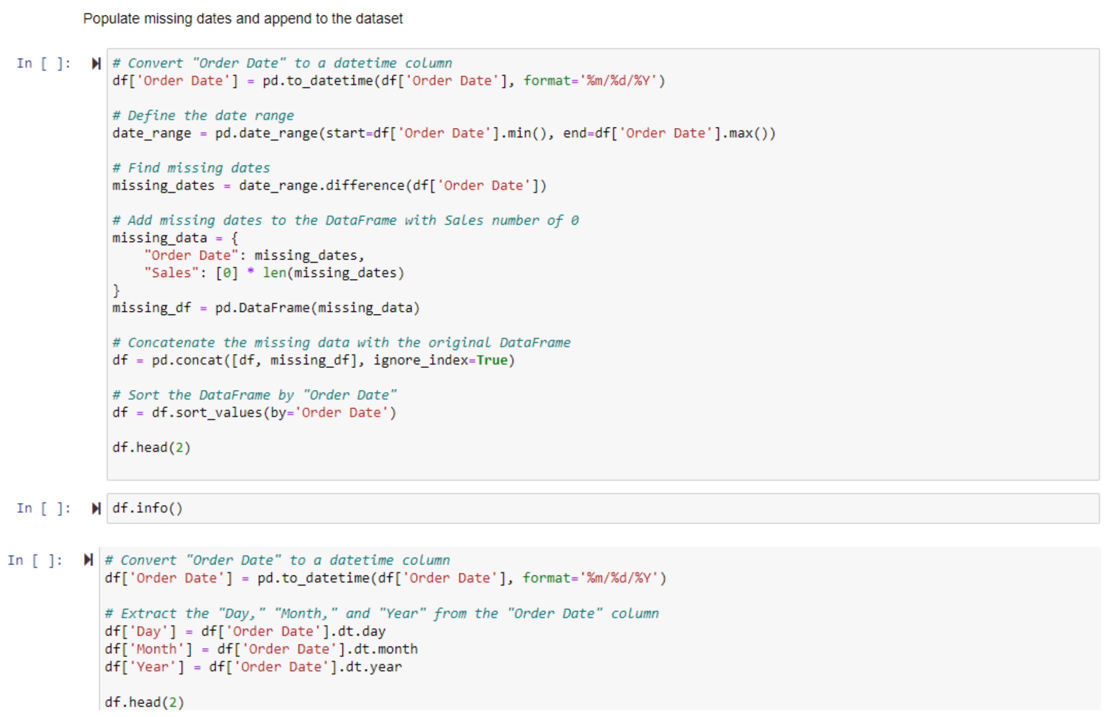
```
In [ ]:  # Combine "Month" and "Year" into a datetime column
         df['Date'] = pd.to_datetime(df['Year'].astype(str) + '-' + df['Month'].astype(str), format='%Y-%m')

         # Group by the combined date and calculate the sum of sales
         df_summary = df.groupby('Date')['Sales'].sum().reset_index()

         # Create a line plot
         plt.figure(figsize=(10, 5))
         plt.plot(df_summary['Date'], df_summary['Sales'], marker='o', linestyle='-')
         plt.xlabel('Date')
         plt.ylabel('Sales')
         plt.title('Sales Over Time-Monthly View')
         plt.grid(True)
         plt.show()
```

```
In [ ]:  # Group by "Order Date" and calculate the sum of sales
         df_summary = df.groupby('Order Date')['Sales'].sum().reset_index()

         # Create a line plot
         plt.figure(figsize=(30, 8))
         plt.plot(df_summary['Order Date'], df_summary['Sales'], marker='o', linestyle='-')
         plt.xlabel('Order Date')
         plt.ylabel('Sales')
         plt.title('Sales Over Time')
         plt.grid(True)
         plt.show()
```

```
In [ ]:  df.head(2)
```

We need to check if there are any missing dates on the daily trends and populate those dates with 0 sales to ensure proper decomposition on the daily view. Also, some models will not work well if the date trend is not consistent.

```
In [ ]:  # Convert "Order Date" to a datetime column
         df['Order Date'] = pd.to_datetime(df['Order Date'], format='%m/%d/%Y')

         # Generate a date range from the minimum to the maximum date
         date_range = pd.date_range(start=df['Order Date'].min(), end=df['Order Date'].max())

         # Find missing dates by comparing the date range with the unique dates in the "Order Date" column
         missing_dates = date_range[~date_range.isin(df['Order Date'])]

         # Count the total number of missing dates
         total_missing_dates = len(missing_dates)

         # Display the list of missing dates and the total count
         #print("Missing Dates:")
         #for date in missing_dates:
          #   print(date.strftime('%m/%d/%Y'))
         print(f"Total Missing Dates: {total_missing_dates}")
```



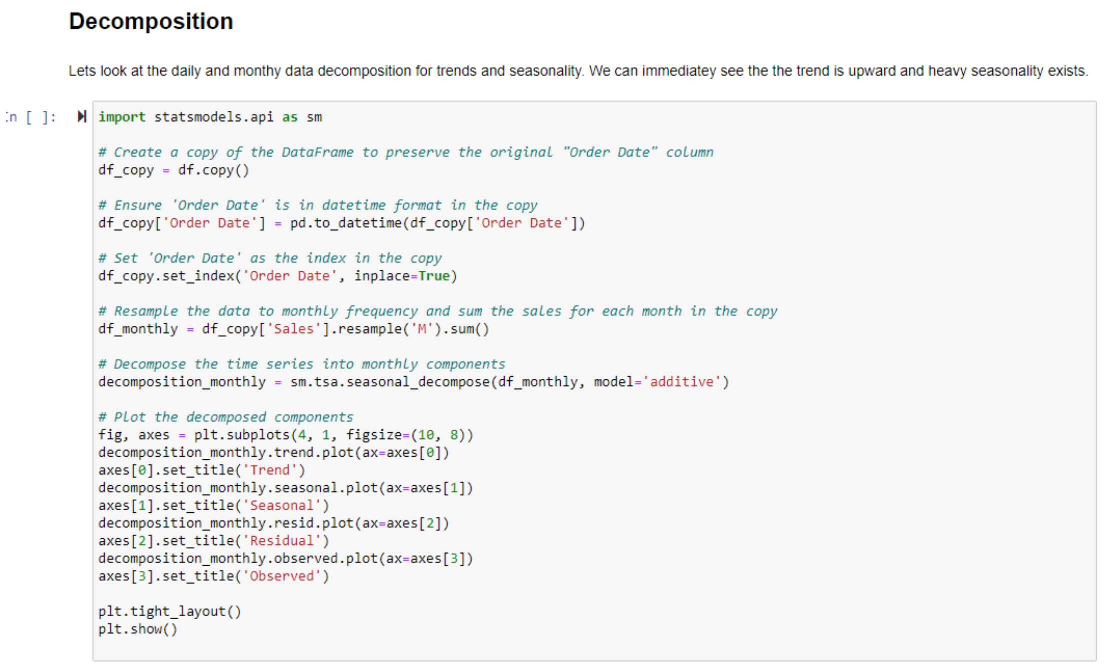
Populate missing dates and append to the dataset

```
In [ ]:  # Convert "Order Date" to a datetime column
         df['Order Date'] = pd.to_datetime(df['Order Date'], format='%m/%d/%Y')

         # Define the date range
         date_range = pd.date_range(start=df['Order Date'].min(), end=df['Order Date'].max())

         # Find missing dates
         missing_dates = date_range.difference(df['Order Date'])

         # Add missing dates to the DataFrame with Sales number of 0
         missing_data = {
             "Order Date": missing_dates,
             "Sales": [0] * len(missing_dates)
         }
         missing_df = pd.DataFrame(missing_data)

         # Concatenate the missing data with the original DataFrame
         df = pd.concat([df, missing_df], ignore_index=True)

         # Sort the DataFrame by "Order Date"
         df = df.sort_values(by='Order Date')

         df.head(2)
```

```
In [ ]:  df.info()
```

```
In [ ]:  # Convert "Order Date" to a datetime column
         df['Order Date'] = pd.to_datetime(df['Order Date'], format='%m/%d/%Y')

         # Extract the "Day," "Month," and "Year" from the "Order Date" column
         df['Day'] = df['Order Date'].dt.day
         df['Month'] = df['Order Date'].dt.month
         df['Year'] = df['Order Date'].dt.year

         df.head(2)
```

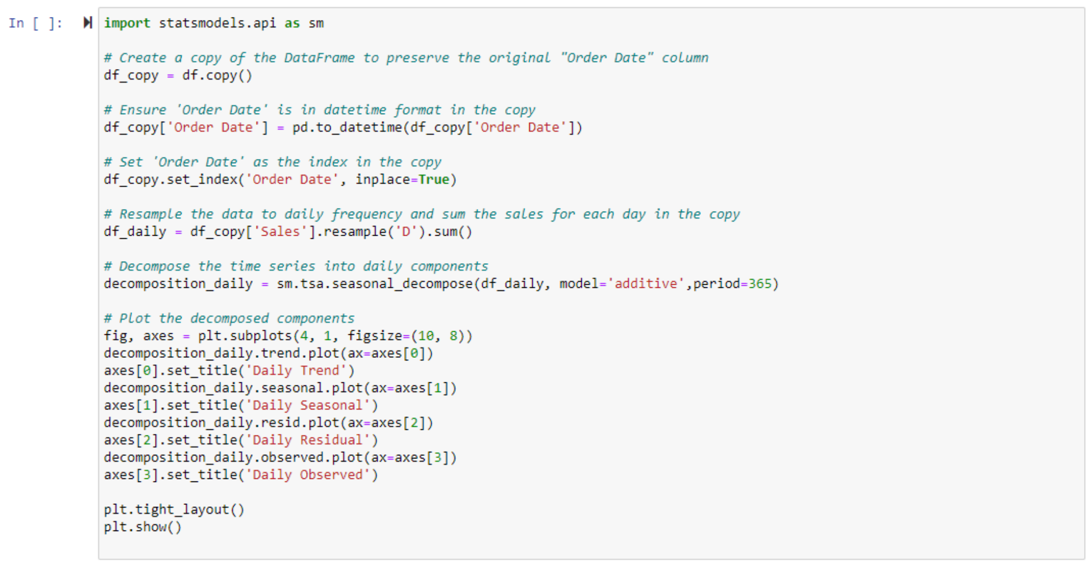
Fig. 4 Decomposition of data for trends and seasonality



## Decomposition

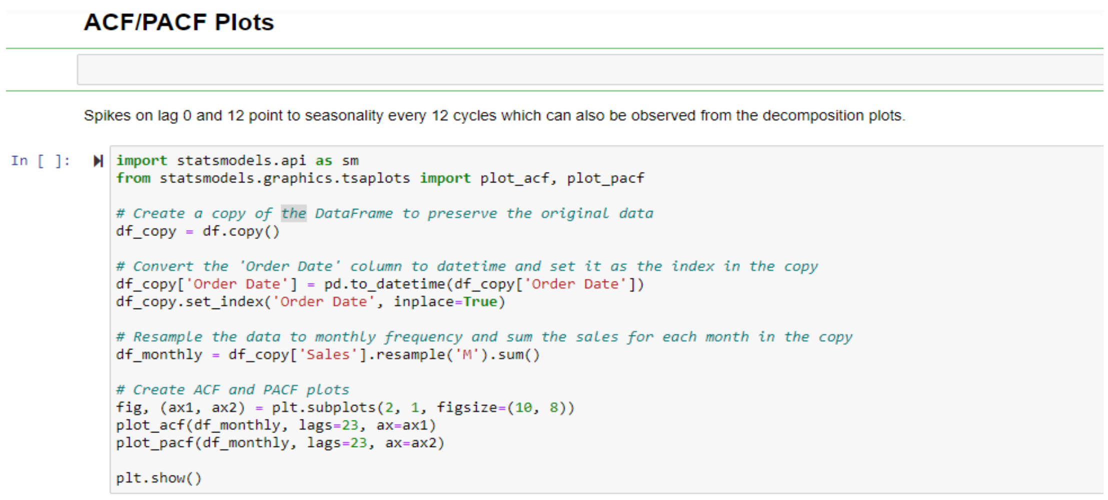
Lets look at the daily and monthy data decomposition for trends and seasonality. We can immediatey see the the trend is upward and heavy seasonality exists.

```
In [ ]:  import statsmodels.api as sm

         # Create a copy of the DataFrame to preserve the original "Order Date" column
         df_copy = df.copy()

         # Ensure 'Order Date' is in datetime format in the copy
         df_copy['Order Date'] = pd.to_datetime(df_copy['Order Date'])

         # Set 'Order Date' as the index in the copy
         df_copy.set_index('Order Date', inplace=True)

         # Resample the data to monthly frequency and sum the sales for each month in the copy
         df_monthly = df_copy['Sales'].resample('M').sum()

         # Decompose the time series into monthly components
         decomposition_monthly = sm.tsa.seasonal_decompose(df_monthly, model='additive')

         # Plot the decomposed components
         fig, axes = plt.subplots(4, 1, figsize=(10, 8))
         decomposition_monthly.trend.plot(ax=axes[0])
         axes[0].set_title('Trend')
         decomposition_monthly.seasonal.plot(ax=axes[1])
         axes[1].set_title('Seasonal')
         decomposition_monthly.resid.plot(ax=axes[2])
         axes[2].set_title('Residual')
         decomposition_monthly.observed.plot(ax=axes[3])
         axes[3].set_title('Observed')

         plt.tight_layout()
         plt.show()
```



```
In [ ]:
import statsmodels.api as sm

# Create a copy of the DataFrame to preserve the original "Order Date" column
df_copy = df.copy()

# Ensure 'Order Date' is in datetime format in the copy
df_copy['Order Date'] = pd.to_datetime(df_copy['Order Date'])

# Set 'Order Date' as the index in the copy
df_copy.set_index('Order Date', inplace=True)

# Resample the data to daily frequency and sum the sales for each day in the copy
df_daily = df_copy['Sales'].resample('D').sum()

# Decompose the time series into daily components
decomposition_daily = sm.tsa.seasonal_decompose(df_daily, model='additive',period=365)

# Plot the decomposed components
fig, axes = plt.subplots(4, 1, figsize=(10, 8))
decomposition_daily.trend.plot(ax=axes[0])
axes[0].set_title('Daily Trend')
decomposition_daily.seasonal.plot(ax=axes[1])
axes[1].set_title('Daily Seasonal')
decomposition_daily.resid.plot(ax=axes[2])
axes[2].set_title('Daily Residual')
decomposition_daily.observed.plot(ax=axes[3])
axes[3].set_title('Daily Observed')

plt.tight_layout()
plt.show()
```

Fig. 5 ACF/PACF plots

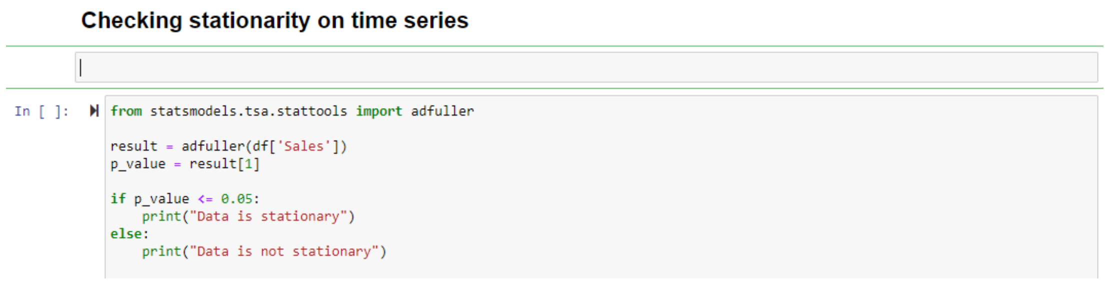


ACF/PACF Plots

Spikes on lag 0 and 12 point to seasonality every 12 cycles which can also be observed from the decomposition plots.

```
In [ ]:
import statsmodels.api as sm
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

# Create a copy of the DataFrame to preserve the original data
df_copy = df.copy()

# Convert the 'Order Date' column to datetime and set it as the index in the copy
df_copy['Order Date'] = pd.to_datetime(df_copy['Order Date'])
df_copy.set_index('Order Date', inplace=True)

# Resample the data to monthly frequency and sum the sales for each month in the copy
df_monthly = df_copy['Sales'].resample('M').sum()

# Create ACF and PACF plots
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 8))
plot_acf(df_monthly, lags=23, ax=ax1)
plot_pacf(df_monthly, lags=23, ax=ax2)

plt.show()
```

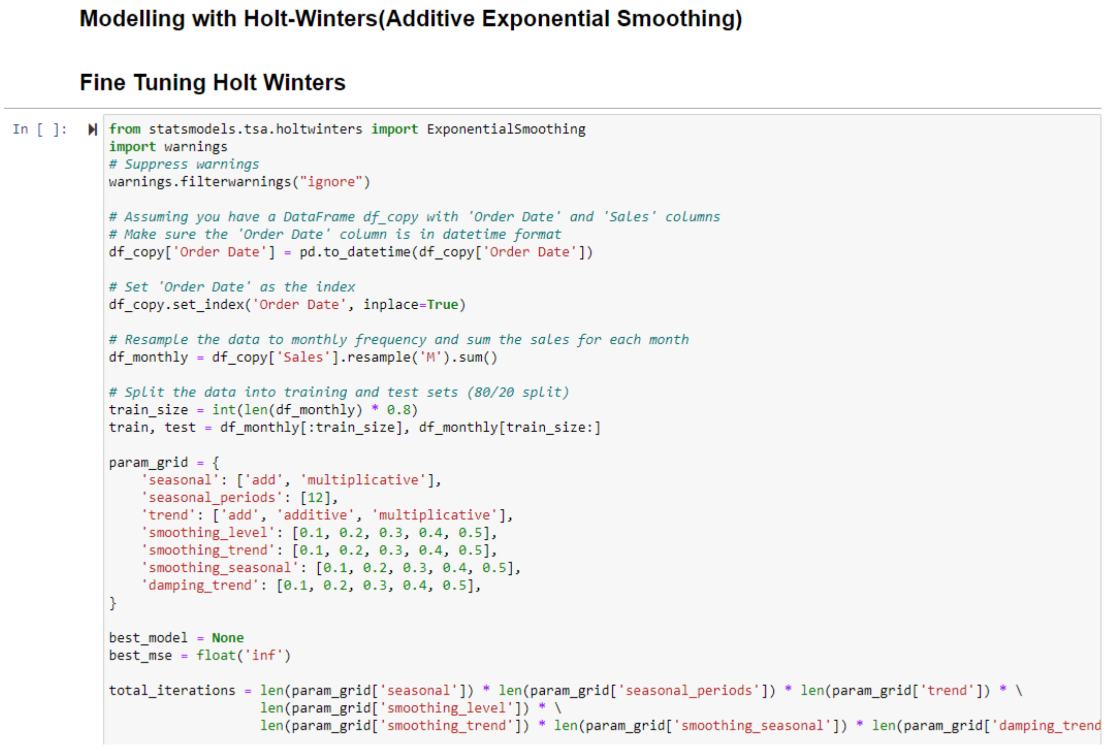
Fig. 6 Stationarity Check of time series data



Checking stationarity on time series

```
In [ ]:
from statsmodels.tsa.stattools import adfuller

result = adfuller(df['Sales'])
p_value = result[1]

if p_value <= 0.05:
    print("Data is stationary")
else:
    print("Data is not stationary")
```

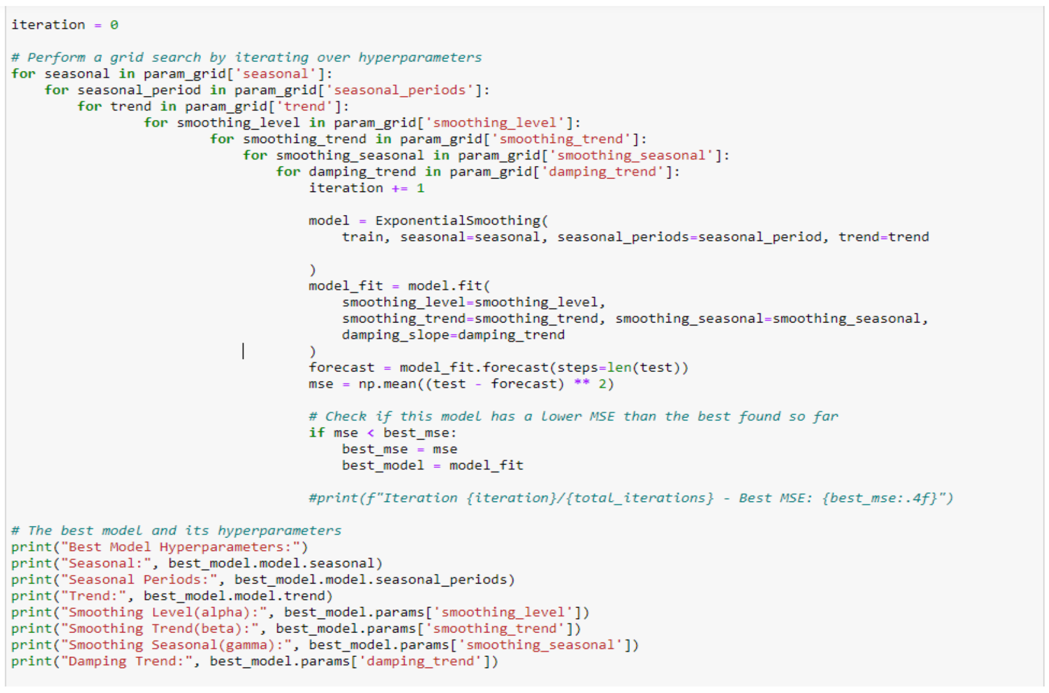
Fig. 7 Modelling of Time Series data with Holt- Winters using Grid Search



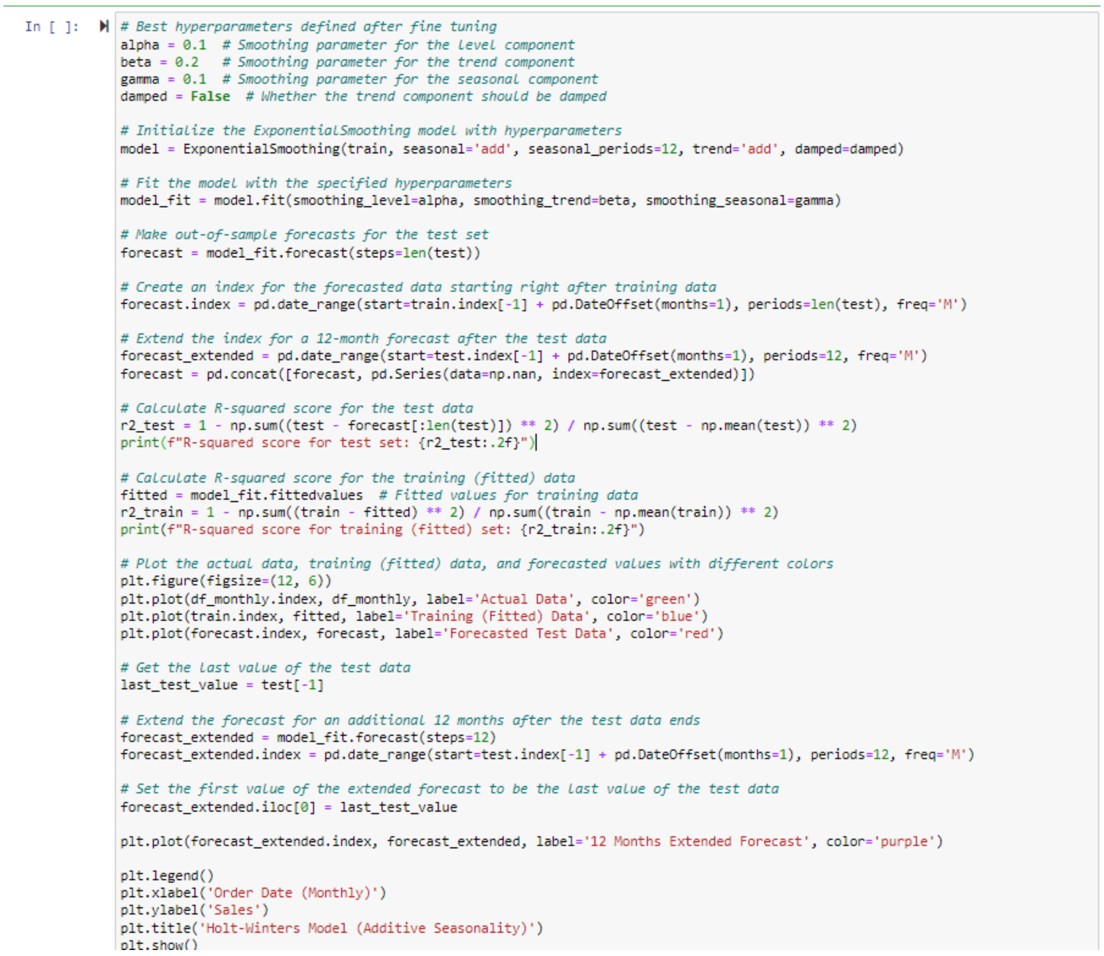
**Modelling with Holt-Winters(Additive Exponential Smoothing)**

**Fine Tuning Holt Winters**

```
In [ ]:   from statsmodels.tsa.holtwinters import ExponentialSmoothing
          import warnings
          # Suppress warnings
          warnings.filterwarnings("ignore")

          # Assuming you have a DataFrame df_copy with 'Order Date' and 'Sales' columns
          # Make sure the 'Order Date' column is in datetime format
          df_copy['Order Date'] = pd.to_datetime(df_copy['Order Date'])

          # Set 'Order Date' as the index
          df_copy.set_index('Order Date', inplace=True)

          # Resample the data to monthly frequency and sum the sales for each month
          df_monthly = df_copy['Sales'].resample('M').sum()

          # Split the data into training and test sets (80/20 split)
          train_size = int(len(df_monthly) * 0.8)
          train, test = df_monthly[:train_size], df_monthly[train_size:]

          param_grid = {
              'seasonal': ['add', 'multiplicative'],
              'seasonal_periods': [12],
              'trend': ['add', 'additive', 'multiplicative'],
              'smoothing_level': [0.1, 0.2, 0.3, 0.4, 0.5],
              'smoothing_trend': [0.1, 0.2, 0.3, 0.4, 0.5],
              'smoothing_seasonal': [0.1, 0.2, 0.3, 0.4, 0.5],
              'damping_trend': [0.1, 0.2, 0.3, 0.4, 0.5],
          }

          best_model = None
          best_mse = float('inf')

          total_iterations = len(param_grid['seasonal']) * len(param_grid['seasonal_periods']) * len(param_grid['trend']) * \
                             len(param_grid['smoothing_level']) * \
                             len(param_grid['smoothing_trend']) * len(param_grid['smoothing_seasonal']) * len(param_grid['damping_trend
```

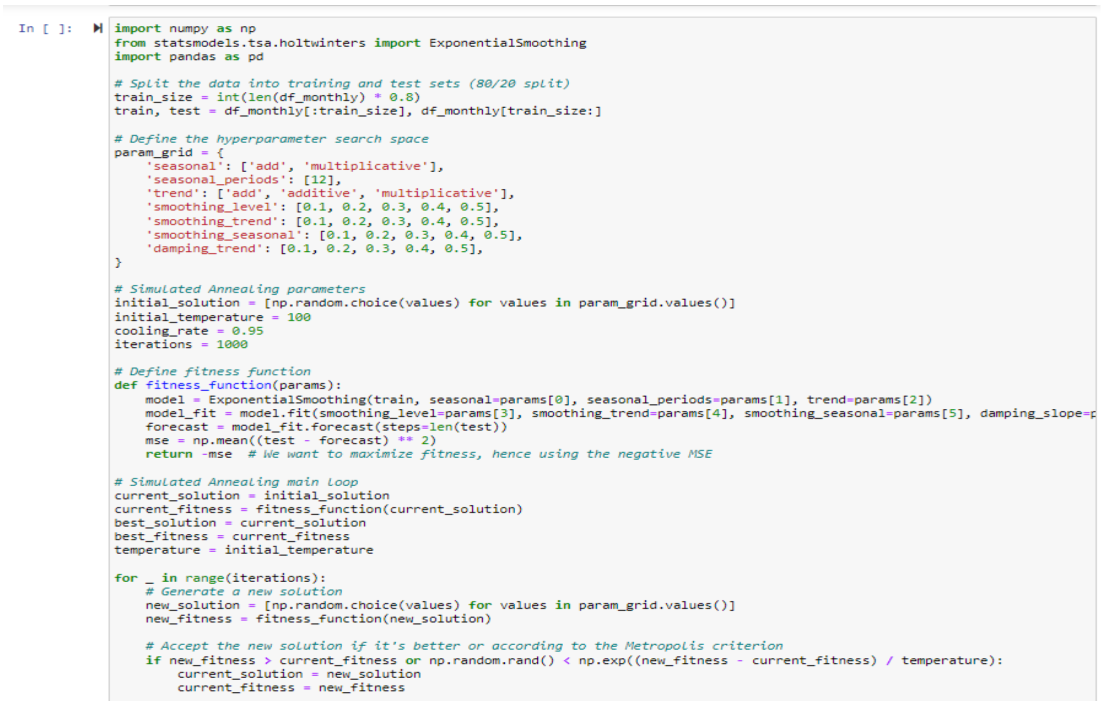


```
iteration = 0

# Perform a grid search by iterating over hyperparameters
for seasonal in param_grid['seasonal']:
    for seasonal_period in param_grid['seasonal_periods']:
        for trend in param_grid['trend']:
            for smoothing_level in param_grid['smoothing_level']:
                for smoothing_trend in param_grid['smoothing_trend']:
                    for smoothing_seasonal in param_grid['smoothing_seasonal']:
                        for damping_trend in param_grid['damping_trend']:
                            iteration += 1

                            model = ExponentialSmoothing(
                                train, seasonal=seasonal, seasonal_periods=seasonal_period, trend=trend

                            )
                            model_fit = model.fit(
                                smoothing_level=smoothing_level,
                                smoothing_trend=smoothing_trend, smoothing_seasonal=smoothing_seasonal,
                                damping_slope=damping_trend
                            )
                            forecast = model_fit.forecast(steps=len(test))
                            mse = np.mean((test - forecast) ** 2)

                            # Check if this model has a lower MSE than the best found so far
                            if mse < best_mse:
                                best_mse = mse
                                best_model = model_fit

                            #print(f"Iteration {iteration}/{total_iterations} - Best MSE: {best_mse:.4f}")

# The best model and its hyperparameters
print("Best Model Hyperparameters:")
print("Seasonal:", best_model.model.seasonal)
print("Seasonal Periods:", best_model.model.seasonal_periods)
print("Trend:", best_model.model.trend)
print("Smoothing Level(alpha):", best_model.params['smoothing_level'])
print("Smoothing Trend(beta):", best_model.params['smoothing_trend'])
print("Smoothing Seasonal(gamma):", best_model.params['smoothing_seasonal'])
print("Damping Trend:", best_model.params['damping_trend'])
```

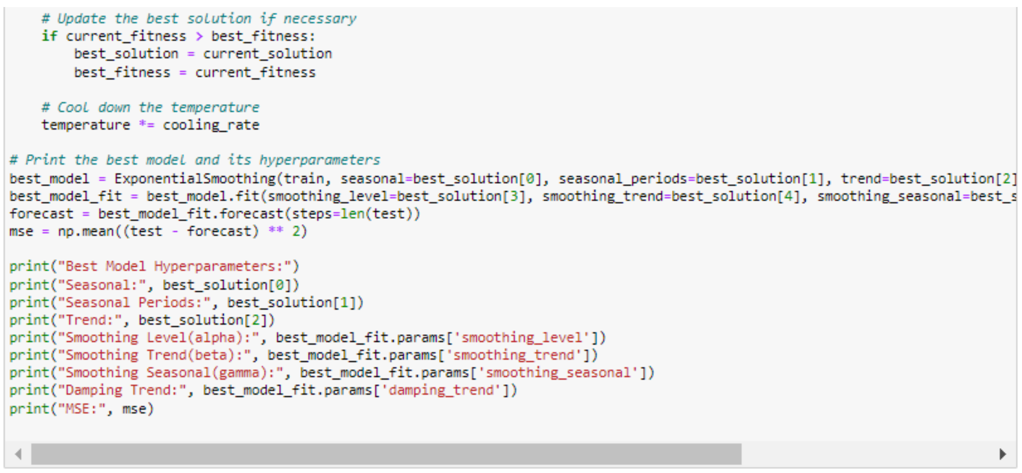


```
In [ ]:  # Best hyperparameters defined after fine tuning
alpha = 0.1  # Smoothing parameter for the level component
beta = 0.2  # Smoothing parameter for the trend component
gamma = 0.1  # Smoothing parameter for the seasonal component
damped = False  # Whether the trend component should be damped

# Initialize the ExponentialSmoothing model with hyperparameters
model = ExponentialSmoothing(train, seasonal='add', seasonal_periods=12, trend='add', damped=damped)

# Fit the model with the specified hyperparameters
model_fit = model.fit(smoothing_level=alpha, smoothing_trend=beta, smoothing_seasonal=gamma)

# Make out-of-sample forecasts for the test set
forecast = model_fit.forecast(steps=len(test))

# Create an index for the forecasted data starting right after training data
forecast.index = pd.date_range(start=train.index[-1] + pd.DateOffset(months=1), periods=len(test), freq='M')

# Extend the index for a 12-month forecast after the test data
forecast_extended = pd.date_range(start=test.index[-1] + pd.DateOffset(months=1), periods=12, freq='M')
forecast = pd.concat([forecast, pd.Series(data=np.nan, index=forecast_extended)])

# Calculate R-squared score for the test data
r2_test = 1 - np.sum((test - forecast[:len(test)]) ** 2) / np.sum((test - np.mean(test)) ** 2)
print(f"R-squared score for test set: {r2_test:.2f}")

# Calculate R-squared score for the training (fitted) data
fitted = model_fit.fittedvalues  # Fitted values for training data
r2_train = 1 - np.sum((train - fitted) ** 2) / np.sum((train - np.mean(train)) ** 2)
print(f"R-squared score for training (fitted) set: {r2_train:.2f}")

# Plot the actual data, training (fitted) data, and forecasted values with different colors
plt.figure(figsize=(12, 6))
plt.plot(df_monthly.index, df_monthly, label='Actual Data', color='green')
plt.plot(train.index, fitted, label='Training (Fitted) Data', color='blue')
plt.plot(forecast.index, forecast, label='Forecasted Test Data', color='red')

# Get the last value of the test data
last_test_value = test[-1]

# Extend the forecast for an additional 12 months after the test data ends
forecast_extended = model_fit.forecast(steps=12)
forecast_extended.index = pd.date_range(start=test.index[-1] + pd.DateOffset(months=1), periods=12, freq='M')

# Set the first value of the extended forecast to be the last value of the test data
forecast_extended.iloc[0] = last_test_value

plt.plot(forecast_extended.index, forecast_extended, label='12 Months Extended Forecast', color='purple')

plt.legend()
plt.xlabel('Order Date (Monthly)')
plt.ylabel('Sales')
plt.title('Holt-Winters Model (Additive Seasonality)')
plt.show()
```

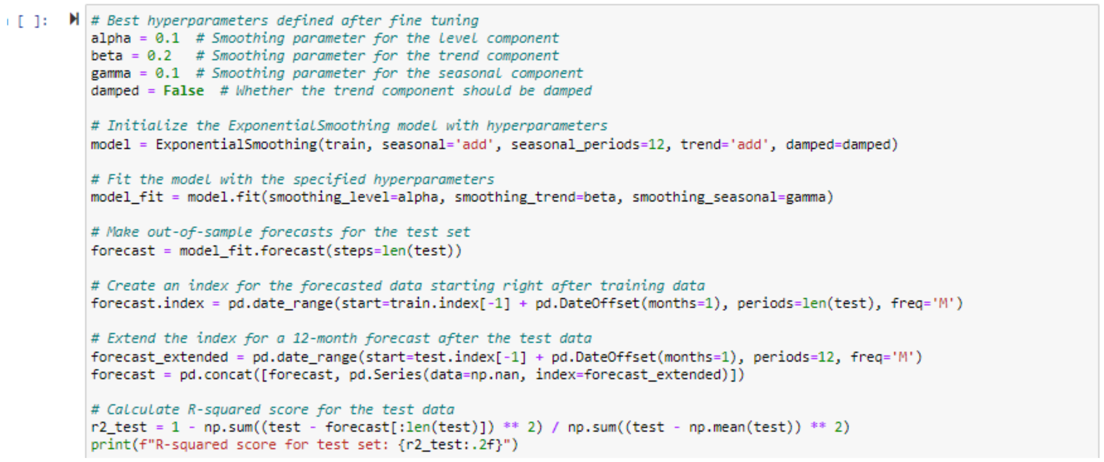
Fig. 8 Modelling with Holt- Winters using Simulated Annealing



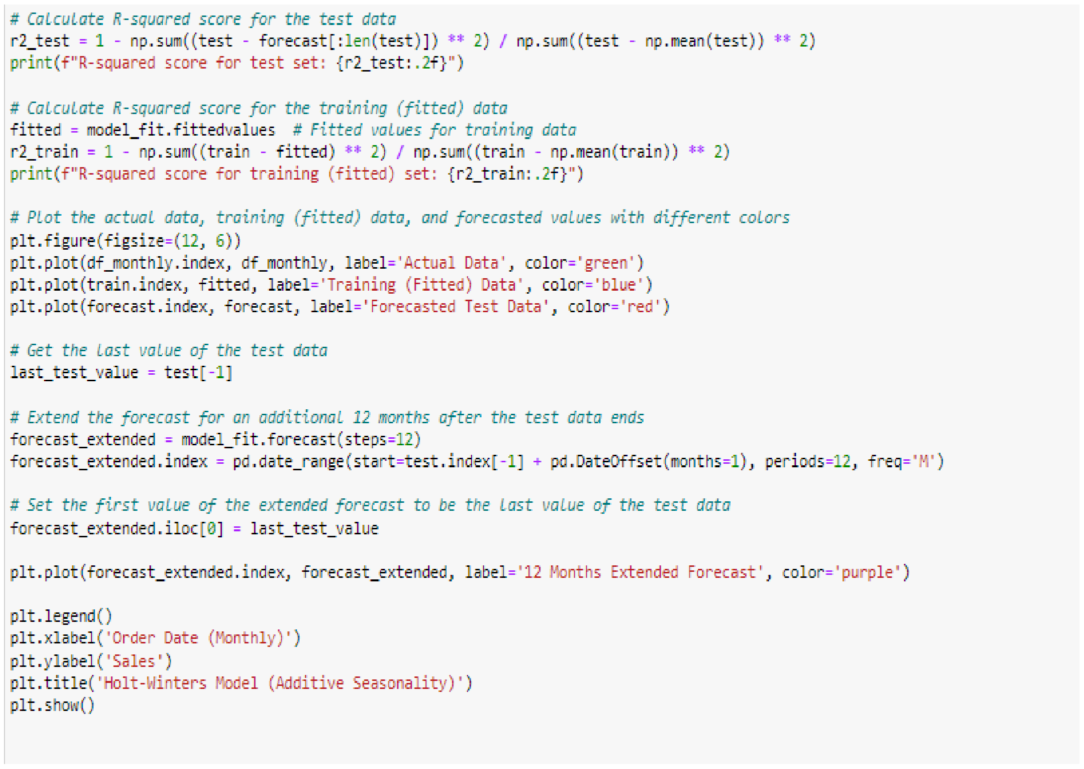
```
In [ ]:  import numpy as np
from statsmodels.tsa.holtwinters import ExponentialSmoothing
import pandas as pd

# Split the data into training and test sets (80/20 split)
train_size = int(len(df_monthly) * 0.8)
train, test = df_monthly[:train_size], df_monthly[train_size:]

# Define the hyperparameter search space
param_grid = {
    'seasonal': ['add', 'multiplicative'],
    'seasonal_periods': [12],
    'trend': ['add', 'additive', 'multiplicative'],
    'smoothing_level': [0.1, 0.2, 0.3, 0.4, 0.5],
    'smoothing_trend': [0.1, 0.2, 0.3, 0.4, 0.5],
    'smoothing_seasonal': [0.1, 0.2, 0.3, 0.4, 0.5],
    'damping_trend': [0.1, 0.2, 0.3, 0.4, 0.5],
}

# Simulated Annealing parameters
initial_solution = [np.random.choice(values) for values in param_grid.values()]
initial_temperature = 100
cooling_rate = 0.95
iterations = 1000

# Define fitness function
def fitness_function(params):
    model = ExponentialSmoothing(train, seasonal=params[0], seasonal_periods=params[1], trend=params[2])
    model_fit = model.fit(smoothing_level=params[3], smoothing_trend=params[4], smoothing_seasonal=params[5], damping_slope=p
    forecast = model_fit.forecast(steps=len(test))
    mse = np.mean((test - forecast) ** 2)
    return -mse  # We want to maximize fitness, hence using the negative MSE

# Simulated Annealing main loop
current_solution = initial_solution
current_fitness = fitness_function(current_solution)
best_solution = current_solution
best_fitness = current_fitness
temperature = initial_temperature

for _ in range(iterations):
    # Generate a new solution
    new_solution = [np.random.choice(values) for values in param_grid.values()]
    new_fitness = fitness_function(new_solution)

    # Accept the new solution if it's better or according to the Metropolis criterion
    if new_fitness > current_fitness or np.random.rand() < np.exp((new_fitness - current_fitness) / temperature):
        current_solution = new_solution
        current_fitness = new_fitness
```



```
    # Update the best solution if necessary
    if current_fitness > best_fitness:
        best_solution = current_solution
        best_fitness = current_fitness

    # Cool down the temperature
    temperature *= cooling_rate

# Print the best model and its hyperparameters
best_model = ExponentialSmoothing(train, seasonal=best_solution[0], seasonal_periods=best_solution[1], trend=best_solution[2]
best_model_fit = best_model.fit(smoothing_level=best_solution[3], smoothing_trend=best_solution[4], smoothing_seasonal=best_s
forecast = best_model_fit.forecast(steps=len(test))
mse = np.mean((test - forecast) ** 2)

print("Best Model Hyperparameters:")
print("Seasonal:", best_solution[0])
print("Seasonal Periods:", best_solution[1])
print("Trend:", best_solution[2])
print("Smoothing Level(alpha):", best_model_fit.params['smoothing_level'])
print("Smoothing Trend(beta):", best_model_fit.params['smoothing_trend'])
print("Smoothing Seasonal(gamma):", best_model_fit.params['smoothing_seasonal'])
print("Damping Trend:", best_model_fit.params['damping_trend'])
print("MSE:", mse)
```

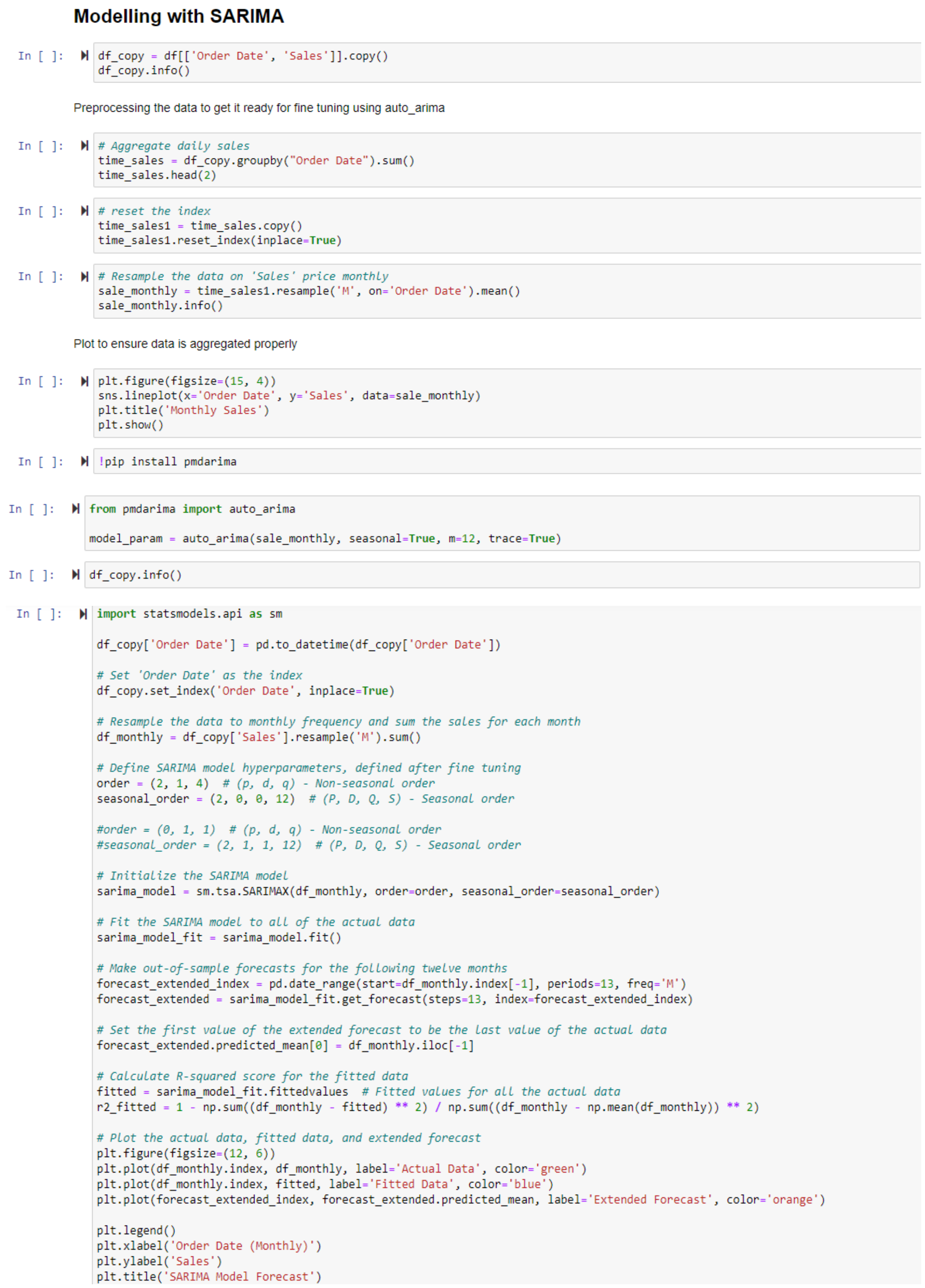


```
[ ]:
# Best hyperparameters defined after fine tuning
alpha = 0.1  # Smoothing parameter for the level component
beta = 0.2   # Smoothing parameter for the trend component
gamma = 0.1  # Smoothing parameter for the seasonal component
damped = False  # Whether the trend component should be damped

# Initialize the ExponentialSmoothing model with hyperparameters
model = ExponentialSmoothing(train, seasonal='add', seasonal_periods=12, trend='add', damped=damped)

# Fit the model with the specified hyperparameters
model_fit = model.fit(smoothing_level=alpha, smoothing_trend=beta, smoothing_seasonal=gamma)

# Make out-of-sample forecasts for the test set
forecast = model_fit.forecast(steps=len(test))

# Create an index for the forecasted data starting right after training data
forecast.index = pd.date_range(start=train.index[-1] + pd.DateOffset(months=1), periods=len(test), freq='M')

# Extend the index for a 12-month forecast after the test data
forecast_extended = pd.date_range(start=test.index[-1] + pd.DateOffset(months=1), periods=12, freq='M')
forecast = pd.concat([forecast, pd.Series(data=np.nan, index=forecast_extended)])

# Calculate R-squared score for the test data
r2_test = 1 - np.sum((test - forecast[:len(test)]) ** 2) / np.sum((test - np.mean(test)) ** 2)
print(f"R-squared score for test set: {r2_test:.2f}")
```



```
# Calculate R-squared score for the test data
r2_test = 1 - np.sum((test - forecast[:len(test)]) ** 2) / np.sum((test - np.mean(test)) ** 2)
print(f"R-squared score for test set: {r2_test:.2f}")

# Calculate R-squared score for the training (fitted) data
fitted = model_fit.fittedvalues  # Fitted values for training data
r2_train = 1 - np.sum((train - fitted) ** 2) / np.sum((train - np.mean(train)) ** 2)
print(f"R-squared score for training (fitted) set: {r2_train:.2f}")

# Plot the actual data, training (fitted) data, and forecasted values with different colors
plt.figure(figsize=(12, 6))
plt.plot(df_monthly.index, df_monthly, label='Actual Data', color='green')
plt.plot(train.index, fitted, label='Training (Fitted) Data', color='blue')
plt.plot(forecast.index, forecast, label='Forecasted Test Data', color='red')

# Get the last value of the test data
last_test_value = test[-1]

# Extend the forecast for an additional 12 months after the test data ends
forecast_extended = model_fit.forecast(steps=12)
forecast_extended.index = pd.date_range(start=test.index[-1] + pd.DateOffset(months=1), periods=12, freq='M')

# Set the first value of the extended forecast to be the last value of the test data
forecast_extended.iloc[0] = last_test_value

plt.plot(forecast_extended.index, forecast_extended, label='12 Months Extended Forecast', color='purple')

plt.legend()
plt.xlabel('Order Date (Monthly)')
plt.ylabel('Sales')
plt.title('Holt-Winters Model (Additive Seasonality)')
plt.show()
```

Fig. 9 Modelling of SARIMA model using pdarima function



**Modelling with SARIMA**

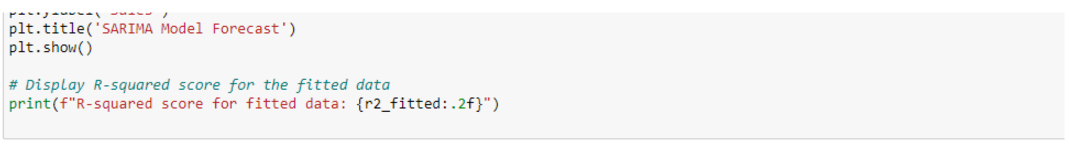
```
In [ ]:  df_copy = df[['Order Date', 'Sales']].copy()
         df_copy.info()
```

Preprocessing the data to get it ready for fine tuning using auto_arima

```
In [ ]:  # Aggregate daily sales
         time_sales = df_copy.groupby("Order Date").sum()
         time_sales.head(2)

In [ ]:  # reset the index
         time_sales1 = time_sales.copy()
         time_sales1.reset_index(inplace=True)

In [ ]:  # Resample the data on 'Sales' price monthly
         sale_monthly = time_sales1.resample('M', on='Order Date').mean()
         sale_monthly.info()
```

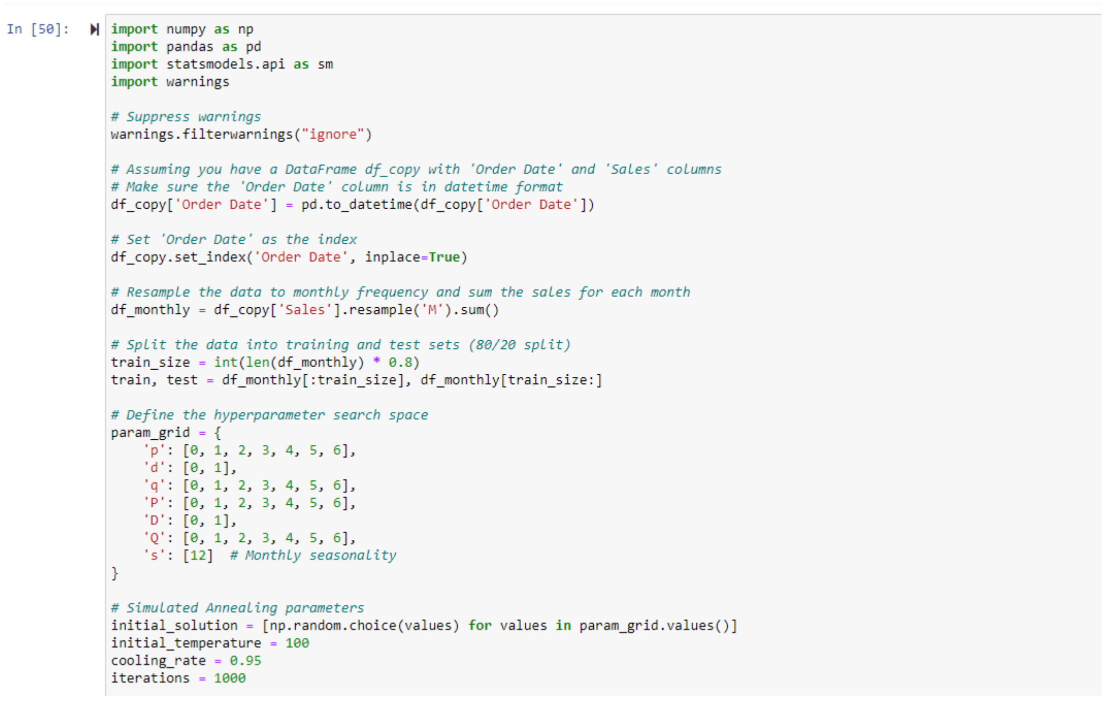
Plot to ensure data is aggregated properly

```
In [ ]:  plt.figure(figsize=(15, 4))
         sns.lineplot(x='Order Date', y='Sales', data=sale_monthly)
         plt.title('Monthly Sales')
         plt.show()

In [ ]:  !pip install pmdarima

In [ ]:  from pmdarima import auto_arima

         model_param = auto_arima(sale_monthly, seasonal=True, m=12, trace=True)

In [ ]:  df_copy.info()

In [ ]:  import statsmodels.api as sm

         df_copy['Order Date'] = pd.to_datetime(df_copy['Order Date'])

         # Set 'Order Date' as the index
         df_copy.set_index('Order Date', inplace=True)

         # Resample the data to monthly frequency and sum the sales for each month
         df_monthly = df_copy['Sales'].resample('M').sum()

         # Define SARIMA model hyperparameters, defined after fine tuning
         order = (2, 1, 4)  # (p, d, q) - Non-seasonal order
         seasonal_order = (2, 0, 0, 12)  # (P, D, Q, S) - Seasonal order

         #order = (0, 1, 1)  # (p, d, q) - Non-seasonal order
         #seasonal_order = (2, 1, 1, 12)  # (P, D, Q, S) - Seasonal order

         # Initialize the SARIMA model
         sarima_model = sm.tsa.SARIMAX(df_monthly, order=order, seasonal_order=seasonal_order)

         # Fit the SARIMA model to all of the actual data
         sarima_model_fit = sarima_model.fit()

         # Make out-of-sample forecasts for the following twelve months
         forecast_extended_index = pd.date_range(start=df_monthly.index[-1], periods=13, freq='M')
         forecast_extended = sarima_model_fit.get_forecast(steps=13, index=forecast_extended_index)

         # Set the first value of the extended forecast to be the last value of the actual data
         forecast_extended.predicted_mean[0] = df_monthly.iloc[-1]

         # Calculate R-squared score for the fitted data
         fitted = sarima_model_fit.fittedvalues  # Fitted values for all the actual data
         r2_fitted = 1 - np.sum((df_monthly - fitted) ** 2) / np.sum((df_monthly - np.mean(df_monthly)) ** 2)

         # Plot the actual data, fitted data, and extended forecast
         plt.figure(figsize=(12, 6))
         plt.plot(df_monthly.index, df_monthly, label='Actual Data', color='green')
         plt.plot(df_monthly.index, fitted, label='Fitted Data', color='blue')
         plt.plot(forecast_extended_index, forecast_extended.predicted_mean, label='Extended Forecast', color='orange')

         plt.legend()
         plt.xlabel('Order Date (Monthly)')
         plt.ylabel('Sales')
         plt.title('SARIMA Model Forecast')
```

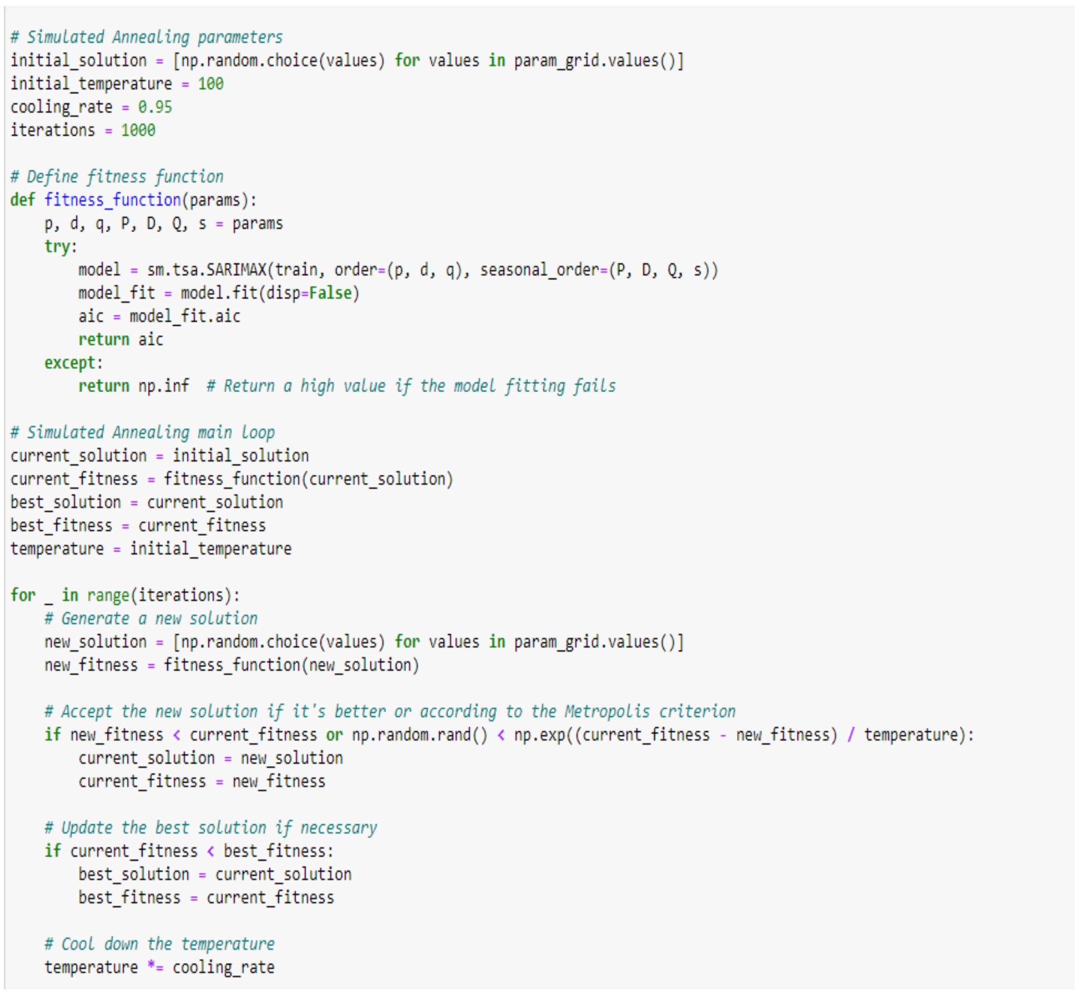


```
plt.title('SARIMA Model Forecast')
plt.show()

# Display R-squared score for the fitted data
print(f"R-squared score for fitted data: {r2_fitted:.2f}")
```

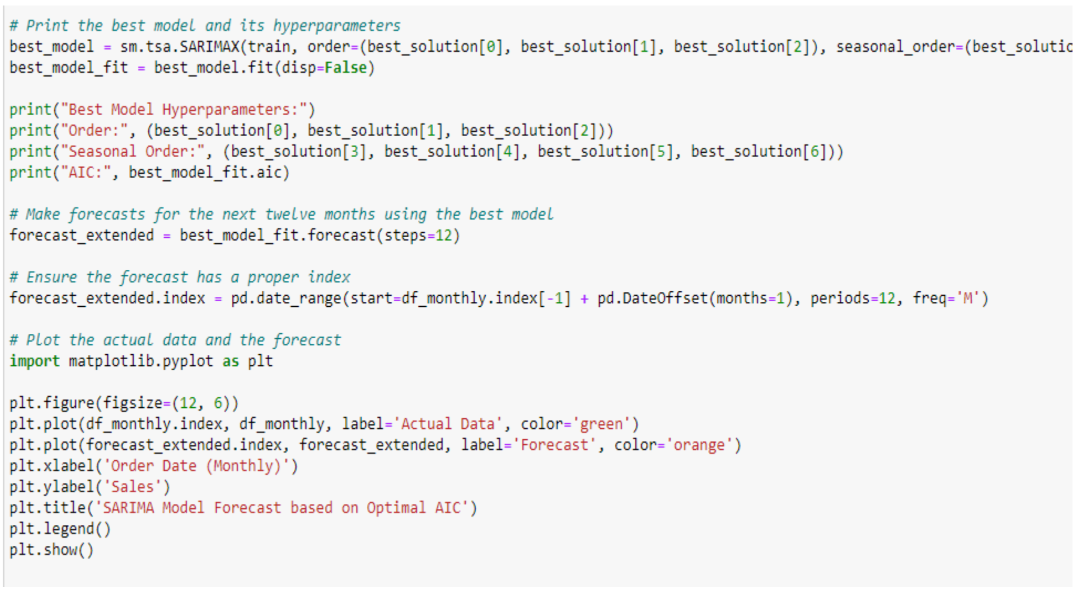
Fig. 10 SARIMA modelling using Simulated Annealing optimisation



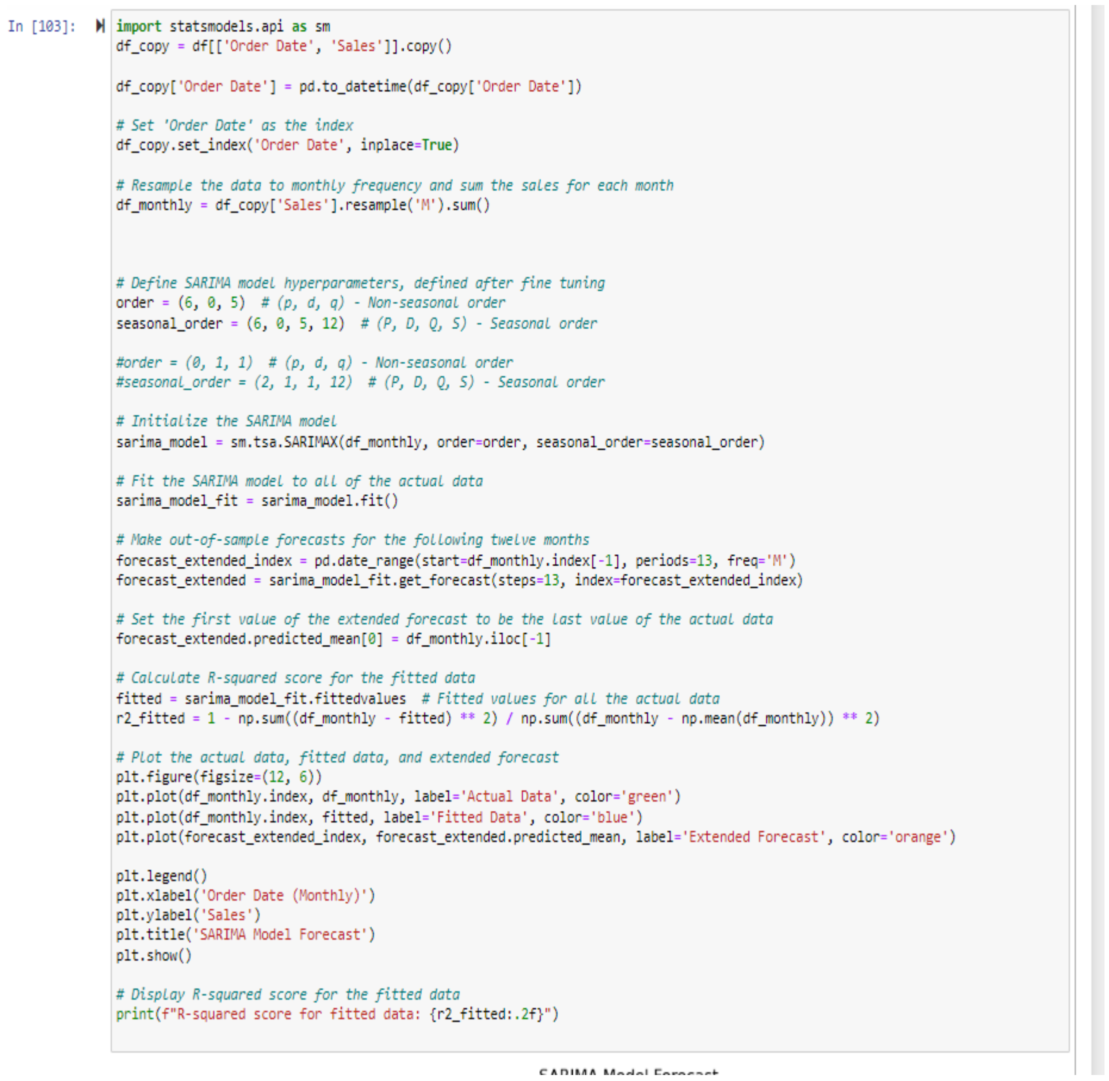
```
In [50]:  import numpy as np
          import pandas as pd
          import statsmodels.api as sm
          import warnings

          # Suppress warnings
          warnings.filterwarnings("ignore")

          # Assuming you have a DataFrame df_copy with 'Order Date' and 'Sales' columns
          # Make sure the 'Order Date' column is in datetime format
          df_copy['Order Date'] = pd.to_datetime(df_copy['Order Date'])

          # Set 'Order Date' as the index
          df_copy.set_index('Order Date', inplace=True)

          # Resample the data to monthly frequency and sum the sales for each month
          df_monthly = df_copy['Sales'].resample('M').sum()

          # Split the data into training and test sets (80/20 split)
          train_size = int(len(df_monthly) * 0.8)
          train, test = df_monthly[:train_size], df_monthly[train_size:]

          # Define the hyperparameter search space
          param_grid = {
              'p': [0, 1, 2, 3, 4, 5, 6],
              'd': [0, 1],
              'q': [0, 1, 2, 3, 4, 5, 6],
              'P': [0, 1, 2, 3, 4, 5, 6],
              'D': [0, 1],
              'Q': [0, 1, 2, 3, 4, 5, 6],
              's': [12]  # Monthly seasonality
          }

          # Simulated Annealing parameters
          initial_solution = [np.random.choice(values) for values in param_grid.values()]
          initial_temperature = 100
          cooling_rate = 0.95
          iterations = 1000
```



```
# Simulated Annealing parameters
initial_solution = [np.random.choice(values) for values in param_grid.values()]
initial_temperature = 100
cooling_rate = 0.95
iterations = 1000

# Define fitness function
def fitness_function(params):
    p, d, q, P, D, Q, s = params
    try:
        model = sm.tsa.SARIMAX(train, order=(p, d, q), seasonal_order=(P, D, Q, s))
        model_fit = model.fit(disp=False)
        aic = model_fit.aic
        return aic
    except:
        return np.inf  # Return a high value if the model fitting fails

# Simulated Annealing main loop
current_solution = initial_solution
current_fitness = fitness_function(current_solution)
best_solution = current_solution
best_fitness = current_fitness
temperature = initial_temperature

for _ in range(iterations):
    # Generate a new solution
    new_solution = [np.random.choice(values) for values in param_grid.values()]
    new_fitness = fitness_function(new_solution)

    # Accept the new solution if it's better or according to the Metropolis criterion
    if new_fitness < current_fitness or np.random.rand() < np.exp((current_fitness - new_fitness) / temperature):
        current_solution = new_solution
        current_fitness = new_fitness

    # Update the best solution if necessary
    if current_fitness < best_fitness:
        best_solution = current_solution
        best_fitness = current_fitness

    # Cool down the temperature
    temperature *= cooling_rate
```



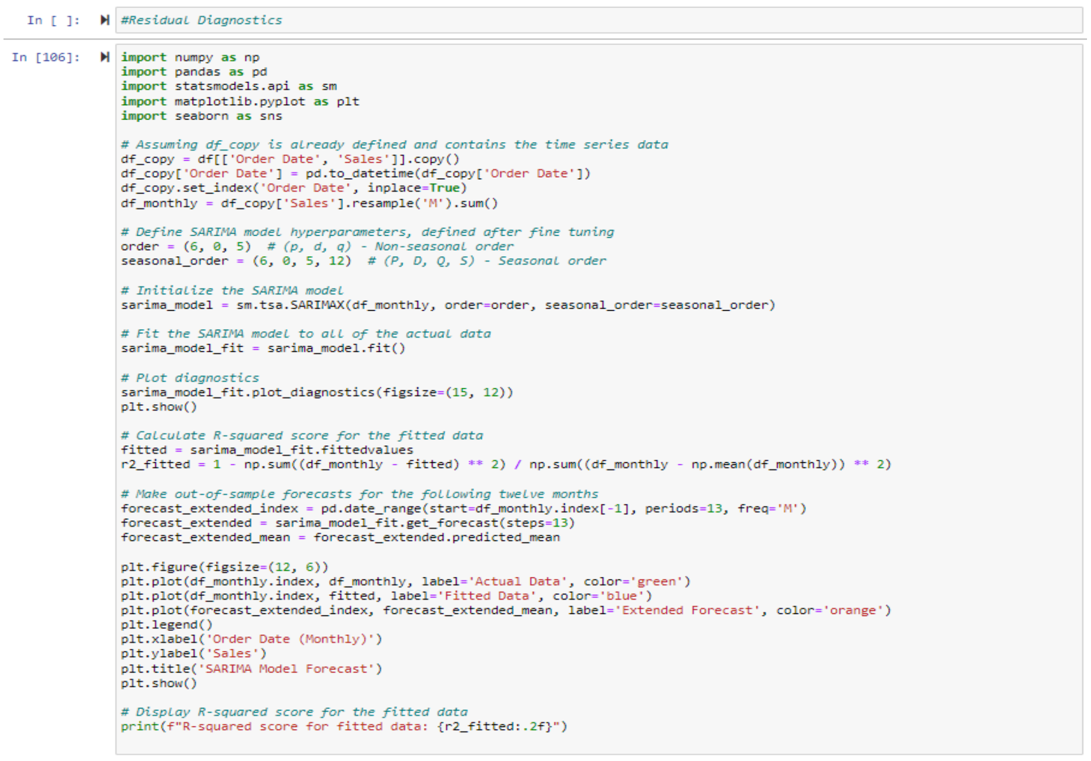
```python
# Print the best model and its hyperparameters
best_model = sm.tsa.SARIMAX(train, order=(best_solution[0], best_solution[1], best_solution[2]), seasonal_order=(best_solutic
best_model_fit = best_model.fit(disp=False)

print("Best Model Hyperparameters:")
print("Order:", (best_solution[0], best_solution[1], best_solution[2]))
print("Seasonal Order:", (best_solution[3], best_solution[4], best_solution[5], best_solution[6]))
print("AIC:", best_model_fit.aic)

# Make forecasts for the next twelve months using the best model
forecast_extended = best_model_fit.forecast(steps=12)

# Ensure the forecast has a proper index
forecast_extended.index = pd.date_range(start=df_monthly.index[-1] + pd.DateOffset(months=1), periods=12, freq='M')

# Plot the actual data and the forecast
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))
plt.plot(df_monthly.index, df_monthly, label='Actual Data', color='green')
plt.plot(forecast_extended.index, forecast_extended, label='Forecast', color='orange')
plt.xlabel('Order Date (Monthly)')
plt.ylabel('Sales')
plt.title('SARIMA Model Forecast based on Optimal AIC')
plt.legend()
plt.show()
```



```python
In [103]:  import statsmodels.api as sm
           df_copy = df[['Order Date', 'Sales']].copy()

           df_copy['Order Date'] = pd.to_datetime(df_copy['Order Date'])

           # Set 'Order Date' as the index
           df_copy.set_index('Order Date', inplace=True)

           # Resample the data to monthly frequency and sum the sales for each month
           df_monthly = df_copy['Sales'].resample('M').sum()



           # Define SARIMA model hyperparameters, defined after fine tuning
           order = (6, 0, 5)  # (p, d, q) - Non-seasonal order
           seasonal_order = (6, 0, 5, 12)  # (P, D, Q, S) - Seasonal order

           #order = (0, 1, 1)  # (p, d, q) - Non-seasonal order
           #seasonal_order = (2, 1, 1, 12)  # (P, D, Q, S) - Seasonal order

           # Initialize the SARIMA model
           sarima_model = sm.tsa.SARIMAX(df_monthly, order=order, seasonal_order=seasonal_order)

           # Fit the SARIMA model to all of the actual data
           sarima_model_fit = sarima_model.fit()

           # Make out-of-sample forecasts for the following twelve months
           forecast_extended_index = pd.date_range(start=df_monthly.index[-1], periods=13, freq='M')
           forecast_extended = sarima_model_fit.get_forecast(steps=13, index=forecast_extended_index)

           # Set the first value of the extended forecast to be the last value of the actual data
           forecast_extended.predicted_mean[0] = df_monthly.iloc[-1]

           # Calculate R-squared score for the fitted data
           fitted = sarima_model_fit.fittedvalues  # Fitted values for all the actual data
           r2_fitted = 1 - np.sum((df_monthly - fitted) ** 2) / np.sum((df_monthly - np.mean(df_monthly)) ** 2)

           # Plot the actual data, fitted data, and extended forecast
           plt.figure(figsize=(12, 6))
           plt.plot(df_monthly.index, df_monthly, label='Actual Data', color='green')
           plt.plot(df_monthly.index, fitted, label='Fitted Data', color='blue')
           plt.plot(forecast_extended_index, forecast_extended.predicted_mean, label='Extended Forecast', color='orange')

           plt.legend()
           plt.xlabel('Order Date (Monthly)')
           plt.ylabel('Sales')
           plt.title('SARIMA Model Forecast')
           plt.show()

           # Display R-squared score for the fitted data
           print(f"R-squared score for fitted data: {r2_fitted:.2f}")
```

SARIMA Model Forecast

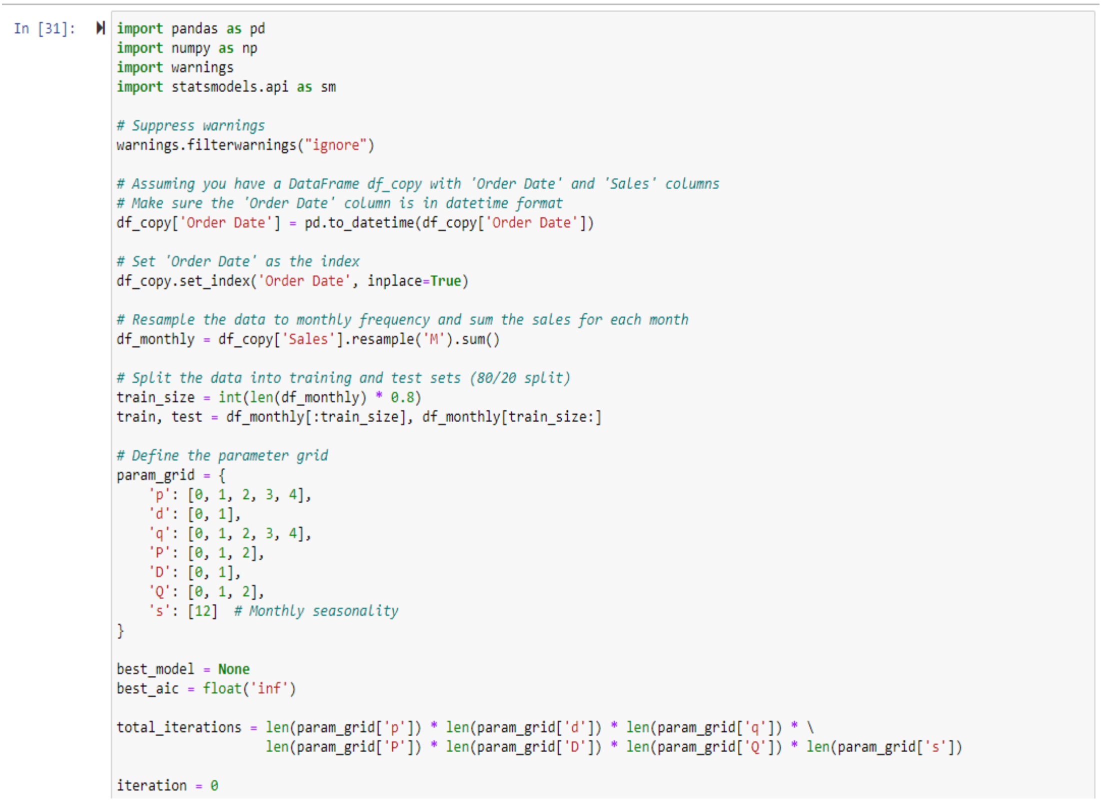
Fig. 11 Residual Diagnostics of the SARIMA model



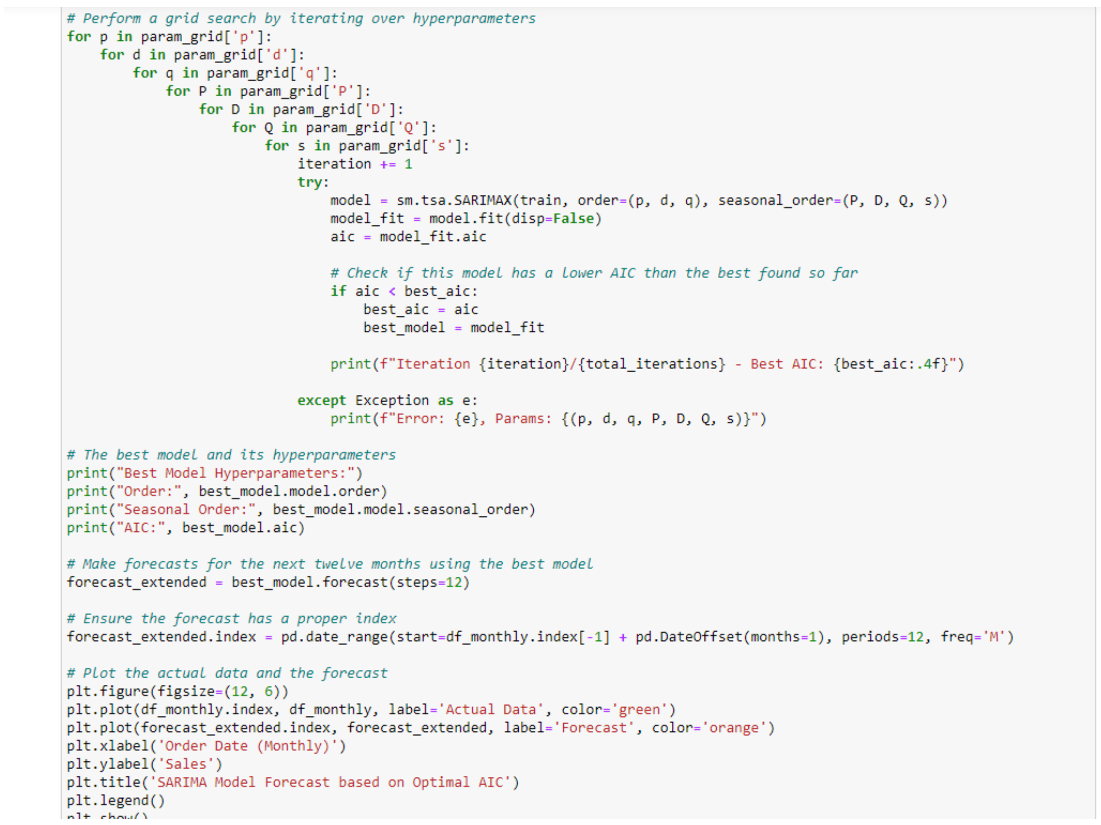
```
In [ ]:   #Residual Diagnostics
```

```
In [106]: import numpy as np
          import pandas as pd
          import statsmodels.api as sm
          import matplotlib.pyplot as plt
          import seaborn as sns

          # Assuming df_copy is already defined and contains the time series data
          df_copy = df[['Order Date', 'Sales']].copy()
          df_copy['Order Date'] = pd.to_datetime(df_copy['Order Date'])
          df_copy.set_index('Order Date', inplace=True)
          df_monthly = df_copy['Sales'].resample('M').sum()

          # Define SARIMA model hyperparameters, defined after fine tuning
          order = (6, 0, 5)  # (p, d, q) - Non-seasonal order
          seasonal_order = (6, 0, 5, 12)  # (P, D, Q, S) - Seasonal order

          # Initialize the SARIMA model
          sarima_model = sm.tsa.SARIMAX(df_monthly, order=order, seasonal_order=seasonal_order)

          # Fit the SARIMA model to all of the actual data
          sarima_model_fit = sarima_model.fit()

          # Plot diagnostics
          sarima_model_fit.plot_diagnostics(figsize=(15, 12))
          plt.show()

          # Calculate R-squared score for the fitted data
          fitted = sarima_model_fit.fittedvalues
          r2_fitted = 1 - np.sum((df_monthly - fitted) ** 2) / np.sum((df_monthly - np.mean(df_monthly)) ** 2)

          # Make out-of-sample forecasts for the following twelve months
          forecast_extended_index = pd.date_range(start=df_monthly.index[-1], periods=13, freq='M')
          forecast_extended = sarima_model_fit.get_forecast(steps=13)
          forecast_extended_mean = forecast_extended.predicted_mean

          plt.figure(figsize=(12, 6))
          plt.plot(df_monthly.index, df_monthly, label='Actual Data', color='green')
          plt.plot(df_monthly.index, fitted, label='Fitted Data', color='blue')
          plt.plot(forecast_extended_index, forecast_extended_mean, label='Extended Forecast', color='orange')
          plt.legend()
          plt.xlabel('Order Date (Monthly)')
          plt.ylabel('Sales')
          plt.title('SARIMA Model Forecast')
          plt.show()

          # Display R-squared score for the fitted data
          print(f"R-squared score for fitted data: {r2_fitted:.2f}")
```

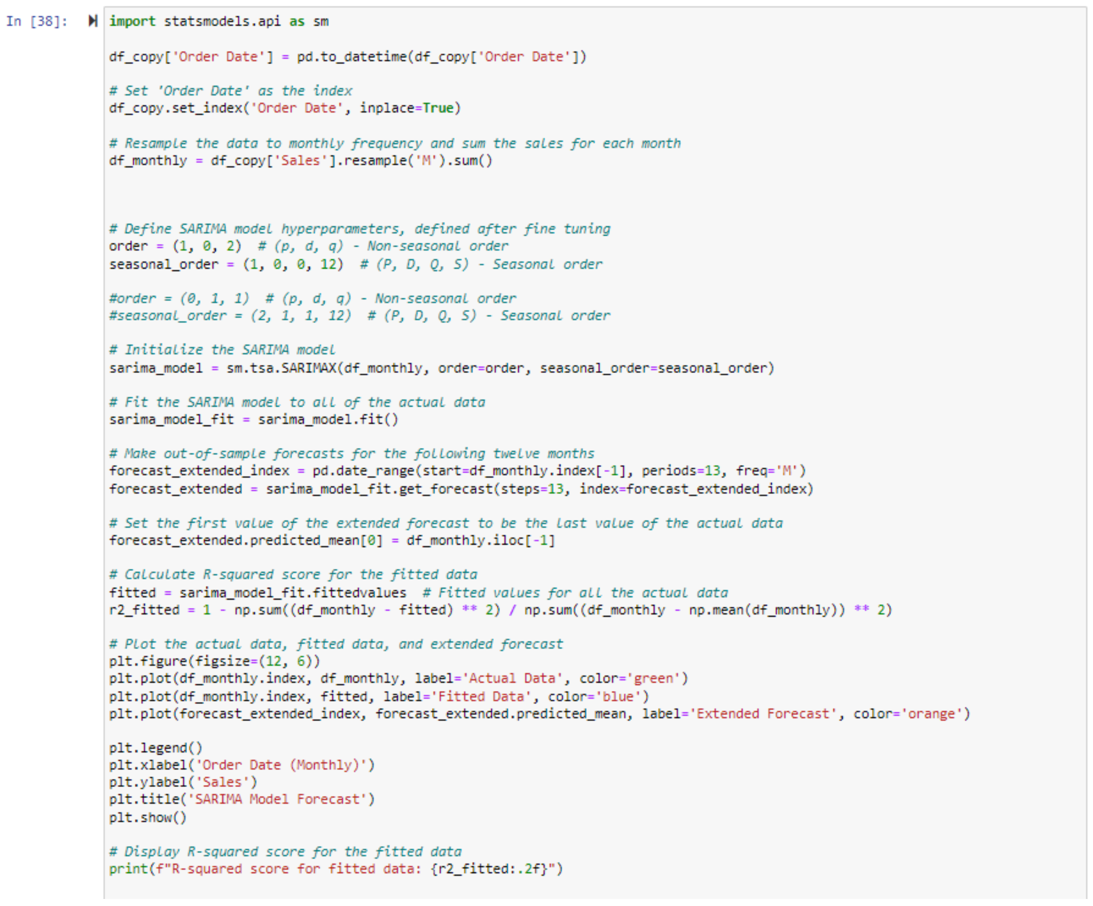
Fig. 12 SARIMA model with Grid Search



```
In [31]:  import pandas as pd
          import numpy as np
          import warnings
          import statsmodels.api as sm

          # Suppress warnings
          warnings.filterwarnings("ignore")

          # Assuming you have a DataFrame df_copy with 'Order Date' and 'Sales' columns
          # Make sure the 'Order Date' column is in datetime format
          df_copy['Order Date'] = pd.to_datetime(df_copy['Order Date'])

          # Set 'Order Date' as the index
          df_copy.set_index('Order Date', inplace=True)

          # Resample the data to monthly frequency and sum the sales for each month
          df_monthly = df_copy['Sales'].resample('M').sum()

          # Split the data into training and test sets (80/20 split)
          train_size = int(len(df_monthly) * 0.8)
          train, test = df_monthly[:train_size], df_monthly[train_size:]

          # Define the parameter grid
          param_grid = {
              'p': [0, 1, 2, 3, 4],
              'd': [0, 1],
              'q': [0, 1, 2, 3, 4],
              'P': [0, 1, 2],
              'D': [0, 1],
              'Q': [0, 1, 2],
              's': [12]  # Monthly seasonality
          }

          best_model = None
          best_aic = float('inf')

          total_iterations = len(param_grid['p']) * len(param_grid['d']) * len(param_grid['q']) * \
                             len(param_grid['P']) * len(param_grid['D']) * len(param_grid['Q']) * len(param_grid['s'])

          iteration = 0
```

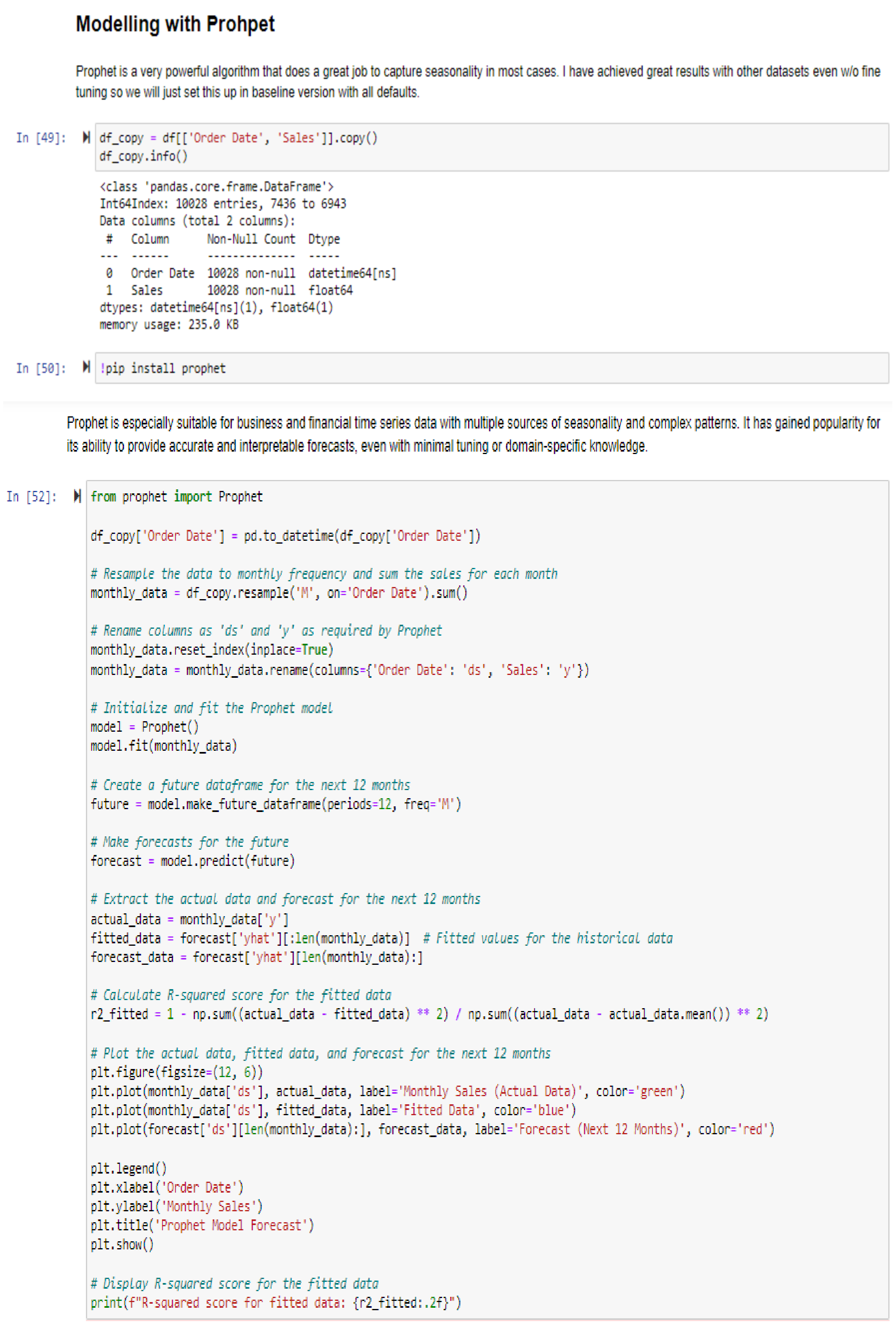


```
# Perform a grid search by iterating over hyperparameters
for p in param_grid['p']:
    for d in param_grid['d']:
        for q in param_grid['q']:
            for P in param_grid['P']:
                for D in param_grid['D']:
                    for Q in param_grid['Q']:
                        for s in param_grid['s']:
                            iteration += 1
                            try:
                                model = sm.tsa.SARIMAX(train, order=(p, d, q), seasonal_order=(P, D, Q, s))
                                model_fit = model.fit(disp=False)
                                aic = model_fit.aic

                                # Check if this model has a lower AIC than the best found so far
                                if aic < best_aic:
                                    best_aic = aic
                                    best_model = model_fit

                                print(f"Iteration {iteration}/{total_iterations} - Best AIC: {best_aic:.4f}")

                            except Exception as e:
                                print(f"Error: {e}, Params: {(p, d, q, P, D, Q, s)}")

# The best model and its hyperparameters
print("Best Model Hyperparameters:")
print("Order:", best_model.model.order)
print("Seasonal Order:", best_model.model.seasonal_order)
print("AIC:", best_model.aic)

# Make forecasts for the next twelve months using the best model
forecast_extended = best_model.forecast(steps=12)

# Ensure the forecast has a proper index
forecast_extended.index = pd.date_range(start=df_monthly.index[-1] + pd.DateOffset(months=1), periods=12, freq='M')

# Plot the actual data and the forecast
plt.figure(figsize=(12, 6))
plt.plot(df_monthly.index, df_monthly, label='Actual Data', color='green')
plt.plot(forecast_extended.index, forecast_extended, label='Forecast', color='orange')
plt.xlabel('Order Date (Monthly)')
plt.ylabel('Sales')
plt.title('SARIMA Model Forecast based on Optimal AIC')
plt.legend()
plt.show()
```



```
In [38]:  import statsmodels.api as sm

          df_copy['Order Date'] = pd.to_datetime(df_copy['Order Date'])

          # Set 'Order Date' as the index
          df_copy.set_index('Order Date', inplace=True)

          # Resample the data to monthly frequency and sum the sales for each month
          df_monthly = df_copy['Sales'].resample('M').sum()


          # Define SARIMA model hyperparameters, defined after fine tuning
          order = (1, 0, 2)  # (p, d, q) - Non-seasonal order
          seasonal_order = (1, 0, 0, 12)  # (P, D, Q, S) - Seasonal order

          #order = (0, 1, 1)  # (p, d, q) - Non-seasonal order
          #seasonal_order = (2, 1, 1, 12)  # (P, D, Q, S) - Seasonal order

          # Initialize the SARIMA model
          sarima_model = sm.tsa.SARIMAX(df_monthly, order=order, seasonal_order=seasonal_order)

          # Fit the SARIMA model to all of the actual data
          sarima_model_fit = sarima_model.fit()

          # Make out-of-sample forecasts for the following twelve months
          forecast_extended_index = pd.date_range(start=df_monthly.index[-1], periods=13, freq='M')
          forecast_extended = sarima_model_fit.get_forecast(steps=13, index=forecast_extended_index)

          # Set the first value of the extended forecast to be the last value of the actual data
          forecast_extended.predicted_mean[0] = df_monthly.iloc[-1]

          # Calculate R-squared score for the fitted data
          fitted = sarima_model_fit.fittedvalues  # Fitted values for all the actual data
          r2_fitted = 1 - np.sum((df_monthly - fitted) ** 2) / np.sum((df_monthly - np.mean(df_monthly)) ** 2)

          # Plot the actual data, fitted data, and extended forecast
          plt.figure(figsize=(12, 6))
          plt.plot(df_monthly.index, df_monthly, label='Actual Data', color='green')
          plt.plot(df_monthly.index, fitted, label='Fitted Data', color='blue')
          plt.plot(forecast_extended_index, forecast_extended.predicted_mean, label='Extended Forecast', color='orange')

          plt.legend()
          plt.xlabel('Order Date (Monthly)')
          plt.ylabel('Sales')
          plt.title('SARIMA Model Forecast')
          plt.show()

          # Display R-squared score for the fitted data
          print(f"R-squared score for fitted data: {r2_fitted:.2f}")
```

Fig. 13 Modelling Prophet algorithms using grid search and simulated annealing



### Modelling with Prohpet

Prophet is a very powerful algorithm that does a great job to capture seasonality in most cases. I have achieved great results with other datasets even w/o fine tuning so we will just set this up in baseline version with all defaults.

In [49]:
```
df_copy = df[['Order Date', 'Sales']].copy()
df_copy.info()
```

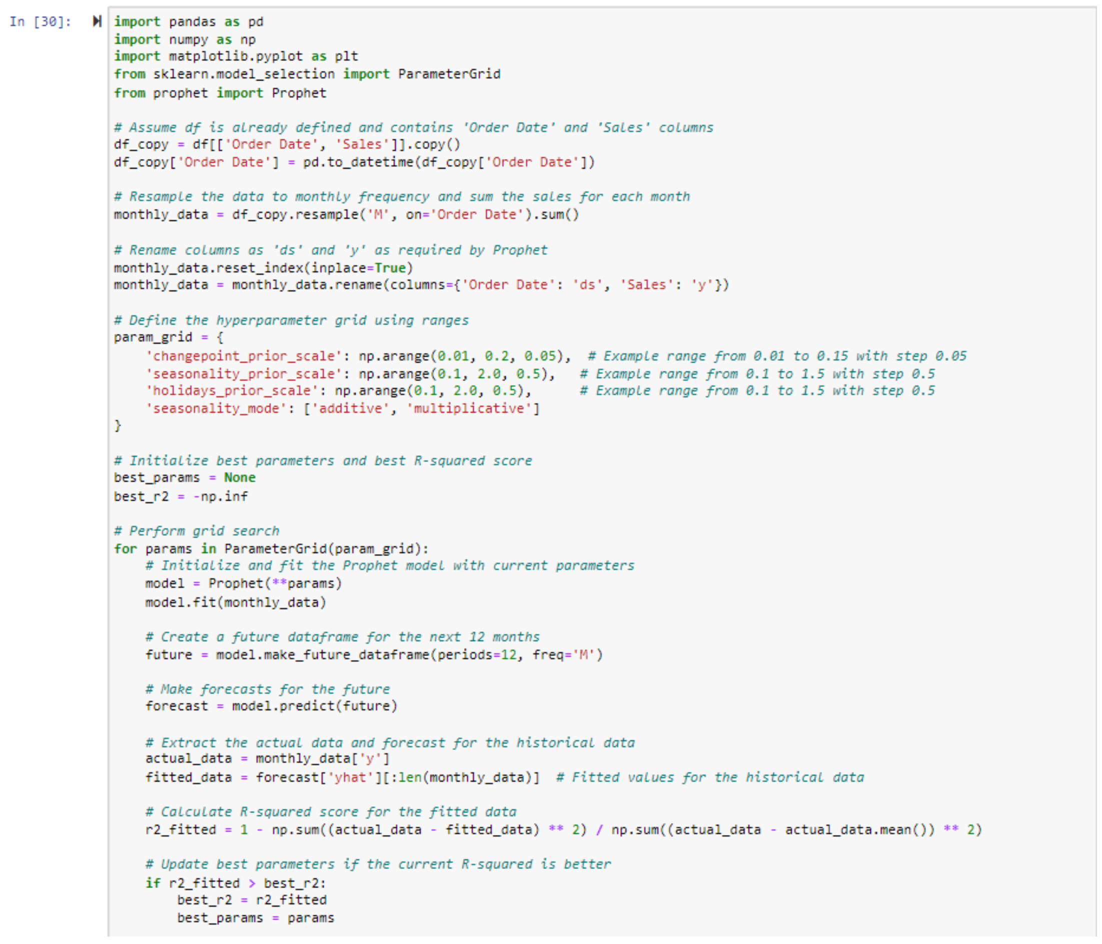
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10028 entries, 7436 to 6943
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Order Date  10028 non-null  datetime64[ns]
 1   Sales       10028 non-null  float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 235.0 KB
```

In [50]:
```
!pip install prophet
```

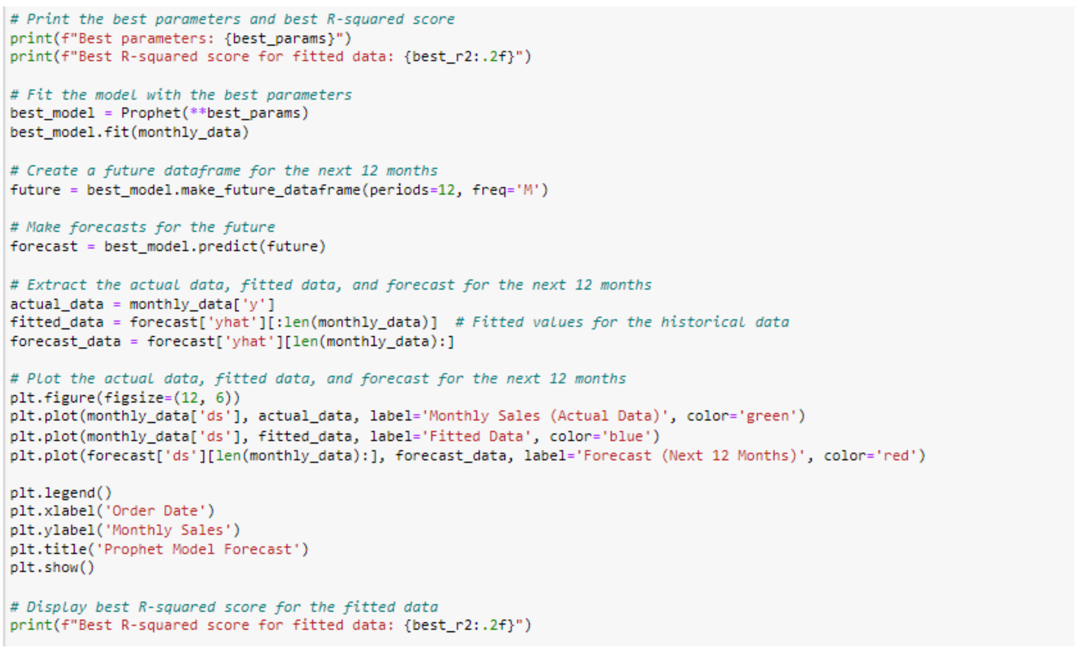
Prophet is especially suitable for business and financial time series data with multiple sources of seasonality and complex patterns. It has gained popularity for its ability to provide accurate and interpretable forecasts, even with minimal tuning or domain-specific knowledge.

In [52]:
```
from prophet import Prophet

df_copy['Order Date'] = pd.to_datetime(df_copy['Order Date'])

# Resample the data to monthly frequency and sum the sales for each month
monthly_data = df_copy.resample('M', on='Order Date').sum()

# Rename columns as 'ds' and 'y' as required by Prophet
monthly_data.reset_index(inplace=True)
monthly_data = monthly_data.rename(columns={'Order Date': 'ds', 'Sales': 'y'})

# Initialize and fit the Prophet model
model = Prophet()
model.fit(monthly_data)

# Create a future dataframe for the next 12 months
future = model.make_future_dataframe(periods=12, freq='M')

# Make forecasts for the future
forecast = model.predict(future)

# Extract the actual data and forecast for the next 12 months
actual_data = monthly_data['y']
fitted_data = forecast['yhat'][:len(monthly_data)]  # Fitted values for the historical data
forecast_data = forecast['yhat'][len(monthly_data):]

# Calculate R-squared score for the fitted data
r2_fitted = 1 - np.sum((actual_data - fitted_data) ** 2) / np.sum((actual_data - actual_data.mean()) ** 2)

# Plot the actual data, fitted data, and forecast for the next 12 months
plt.figure(figsize=(12, 6))
plt.plot(monthly_data['ds'], actual_data, label='Monthly Sales (Actual Data)', color='green')
plt.plot(monthly_data['ds'], fitted_data, label='Fitted Data', color='blue')
plt.plot(forecast['ds'][len(monthly_data):], forecast_data, label='Forecast (Next 12 Months)', color='red')

plt.legend()
plt.xlabel('Order Date')
plt.ylabel('Monthly Sales')
plt.title('Prophet Model Forecast')
plt.show()

# Display R-squared score for the fitted data
print(f"R-squared score for fitted data: {r2_fitted:.2f}")
```

Fig. 14 Modelling of prophet using grid search for fine tuning.

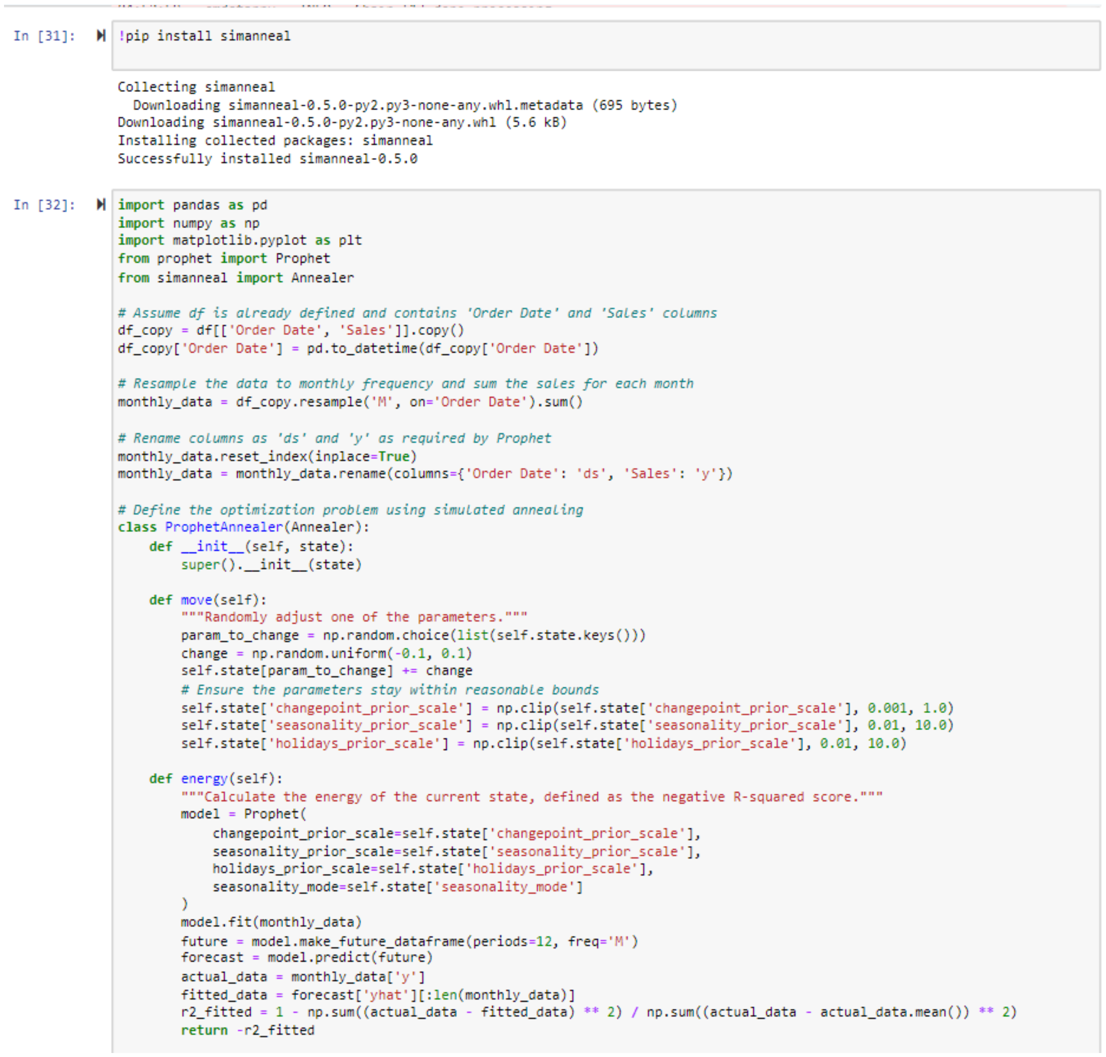


```
In [30]:  import pandas as pd
          import numpy as np
          import matplotlib.pyplot as plt
          from sklearn.model_selection import ParameterGrid
          from prophet import Prophet

          # Assume df is already defined and contains 'Order Date' and 'Sales' columns
          df_copy = df[['Order Date', 'Sales']].copy()
          df_copy['Order Date'] = pd.to_datetime(df_copy['Order Date'])

          # Resample the data to monthly frequency and sum the sales for each month
          monthly_data = df_copy.resample('M', on='Order Date').sum()

          # Rename columns as 'ds' and 'y' as required by Prophet
          monthly_data.reset_index(inplace=True)
          monthly_data = monthly_data.rename(columns={'Order Date': 'ds', 'Sales': 'y'})

          # Define the hyperparameter grid using ranges
          param_grid = {
              'changepoint_prior_scale': np.arange(0.01, 0.2, 0.05),  # Example range from 0.01 to 0.15 with step 0.05
              'seasonality_prior_scale': np.arange(0.1, 2.0, 0.5),   # Example range from 0.1 to 1.5 with step 0.5
              'holidays_prior_scale': np.arange(0.1, 2.0, 0.5),      # Example range from 0.1 to 1.5 with step 0.5
              'seasonality_mode': ['additive', 'multiplicative']
          }

          # Initialize best parameters and best R-squared score
          best_params = None
          best_r2 = -np.inf

          # Perform grid search
          for params in ParameterGrid(param_grid):
              # Initialize and fit the Prophet model with current parameters
              model = Prophet(**params)
              model.fit(monthly_data)

              # Create a future dataframe for the next 12 months
              future = model.make_future_dataframe(periods=12, freq='M')

              # Make forecasts for the future
              forecast = model.predict(future)

              # Extract the actual data and forecast for the historical data
              actual_data = monthly_data['y']
              fitted_data = forecast['yhat'][:len(monthly_data)]  # Fitted values for the historical data

              # Calculate R-squared score for the fitted data
              r2_fitted = 1 - np.sum((actual_data - fitted_data) ** 2) / np.sum((actual_data - actual_data.mean()) ** 2)

              # Update best parameters if the current R-squared is better
              if r2_fitted > best_r2:
                  best_r2 = r2_fitted
                  best_params = params
```



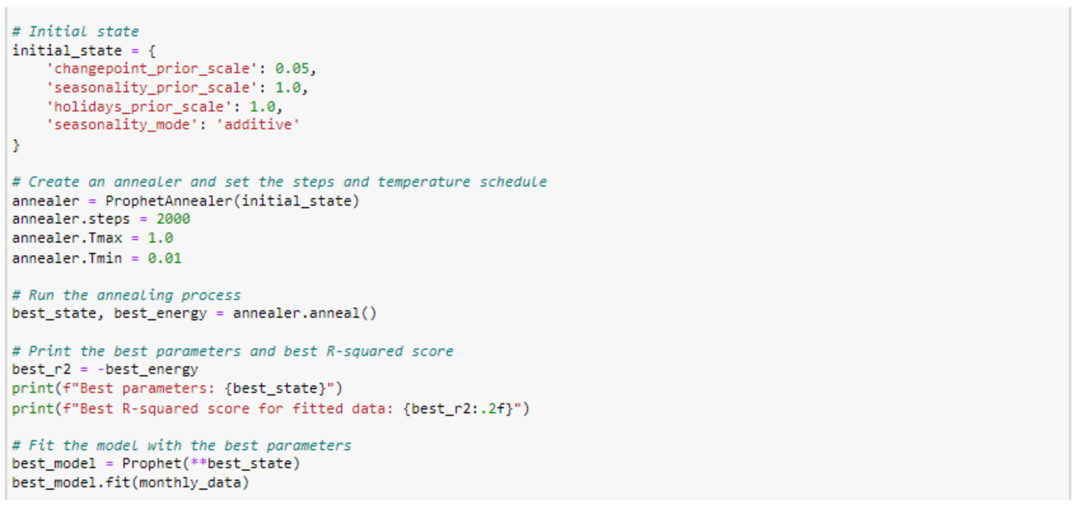
```
# Print the best parameters and best R-squared score
print(f"Best parameters: {best_params}")
print(f"Best R-squared score for fitted data: {best_r2:.2f}")

# Fit the model with the best parameters
best_model = Prophet(**best_params)
best_model.fit(monthly_data)

# Create a future dataframe for the next 12 months
future = best_model.make_future_dataframe(periods=12, freq='M')

# Make forecasts for the future
forecast = best_model.predict(future)

# Extract the actual data, fitted data, and forecast for the next 12 months
actual_data = monthly_data['y']
fitted_data = forecast['yhat'][:len(monthly_data)]  # Fitted values for the historical data
forecast_data = forecast['yhat'][len(monthly_data):]

# Plot the actual data, fitted data, and forecast for the next 12 months
plt.figure(figsize=(12, 6))
plt.plot(monthly_data['ds'], actual_data, label='Monthly Sales (Actual Data)', color='green')
plt.plot(monthly_data['ds'], fitted_data, label='Fitted Data', color='blue')
plt.plot(forecast['ds'][len(monthly_data):], forecast_data, label='Forecast (Next 12 Months)', color='red')

plt.legend()
plt.xlabel('Order Date')
plt.ylabel('Monthly Sales')
plt.title('Prophet Model Forecast')
plt.show()

# Display best R-squared score for the fitted data
print(f"Best R-squared score for fitted data: {best_r2:.2f}")
```

## Fig. 15 Modelling of Prophet model using simulated annealing

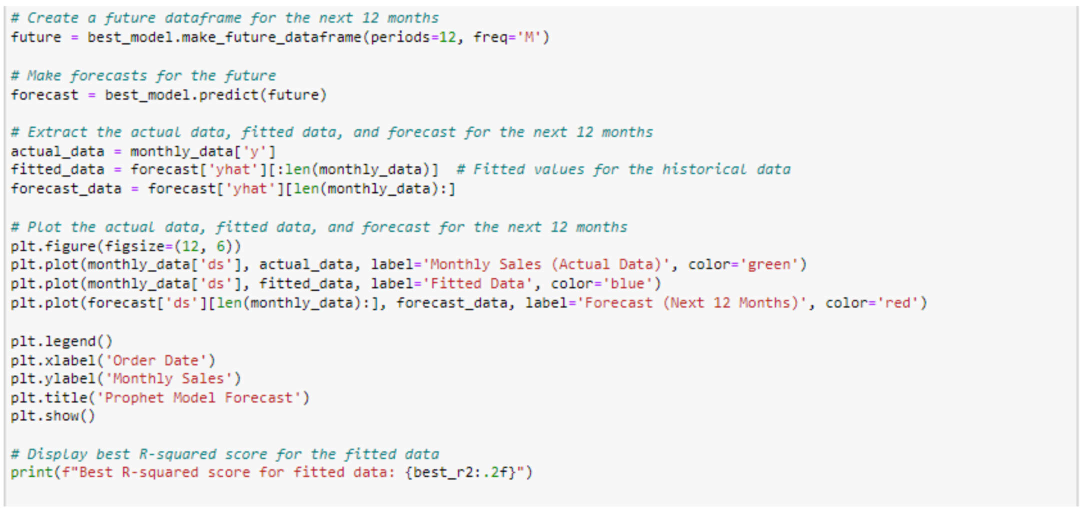


```
In [31]:  !pip install simanneal

          Collecting simanneal
            Downloading simanneal-0.5.0-py2.py3-none-any.whl.metadata (695 bytes)
          Downloading simanneal-0.5.0-py2.py3-none-any.whl (5.6 kB)
          Installing collected packages: simanneal
          Successfully installed simanneal-0.5.0
```

```
In [32]:  import pandas as pd
          import numpy as np
          import matplotlib.pyplot as plt
          from prophet import Prophet
          from simanneal import Annealer

          # Assume df is already defined and contains 'Order Date' and 'Sales' columns
          df_copy = df[['Order Date', 'Sales']].copy()
          df_copy['Order Date'] = pd.to_datetime(df_copy['Order Date'])

          # Resample the data to monthly frequency and sum the sales for each month
          monthly_data = df_copy.resample('M', on='Order Date').sum()

          # Rename columns as 'ds' and 'y' as required by Prophet
          monthly_data.reset_index(inplace=True)
          monthly_data = monthly_data.rename(columns={'Order Date': 'ds', 'Sales': 'y'})

          # Define the optimization problem using simulated annealing
          class ProphetAnnealer(Annealer):
              def __init__(self, state):
                  super().__init__(state)

              def move(self):
                  """Randomly adjust one of the parameters."""
                  param_to_change = np.random.choice(list(self.state.keys()))
                  change = np.random.uniform(-0.1, 0.1)
                  self.state[param_to_change] += change
                  # Ensure the parameters stay within reasonable bounds
                  self.state['changepoint_prior_scale'] = np.clip(self.state['changepoint_prior_scale'], 0.001, 1.0)
                  self.state['seasonality_prior_scale'] = np.clip(self.state['seasonality_prior_scale'], 0.01, 10.0)
                  self.state['holidays_prior_scale'] = np.clip(self.state['holidays_prior_scale'], 0.01, 10.0)

              def energy(self):
                  """Calculate the energy of the current state, defined as the negative R-squared score."""
                  model = Prophet(
                      changepoint_prior_scale=self.state['changepoint_prior_scale'],
                      seasonality_prior_scale=self.state['seasonality_prior_scale'],
                      holidays_prior_scale=self.state['holidays_prior_scale'],
                      seasonality_mode=self.state['seasonality_mode']
                  )
                  model.fit(monthly_data)
                  future = model.make_future_dataframe(periods=12, freq='M')
                  forecast = model.predict(future)
                  actual_data = monthly_data['y']
                  fitted_data = forecast['yhat'][:len(monthly_data)]
                  r2_fitted = 1 - np.sum((actual_data - fitted_data) ** 2) / np.sum((actual_data - actual_data.mean()) ** 2)
                  return -r2_fitted
```



```
# Initial state
initial_state = {
    'changepoint_prior_scale': 0.05,
    'seasonality_prior_scale': 1.0,
    'holidays_prior_scale': 1.0,
    'seasonality_mode': 'additive'
}

# Create an annealer and set the steps and temperature schedule
annealer = ProphetAnnealer(initial_state)
annealer.steps = 2000
annealer.Tmax = 1.0
annealer.Tmin = 0.01

# Run the annealing process
best_state, best_energy = annealer.anneal()

# Print the best parameters and best R-squared score
best_r2 = -best_energy
print(f"Best parameters: {best_state}")
print(f"Best R-squared score for fitted data: {best_r2:.2f}")

# Fit the model with the best parameters
best_model = Prophet(**best_state)
best_model.fit(monthly_data)
```



```python
# Create a future dataframe for the next 12 months
future = best_model.make_future_dataframe(periods=12, freq='M')

# Make forecasts for the future
forecast = best_model.predict(future)

# Extract the actual data, fitted data, and forecast for the next 12 months
actual_data = monthly_data['y']
fitted_data = forecast['yhat'][:len(monthly_data)]  # Fitted values for the historical data
forecast_data = forecast['yhat'][len(monthly_data):]

# Plot the actual data, fitted data, and forecast for the next 12 months
plt.figure(figsize=(12, 6))
plt.plot(monthly_data['ds'], actual_data, label='Monthly Sales (Actual Data)', color='green')
plt.plot(monthly_data['ds'], fitted_data, label='Fitted Data', color='blue')
plt.plot(forecast['ds'][len(monthly_data):], forecast_data, label='Forecast (Next 12 Months)', color='red')

plt.legend()
plt.xlabel('Order Date')
plt.ylabel('Monthly Sales')
plt.title('Prophet Model Forecast')
plt.show()

# Display best R-squared score for the fitted data
print(f"Best R-squared score for fitted data: {best_r2:.2f}")
```

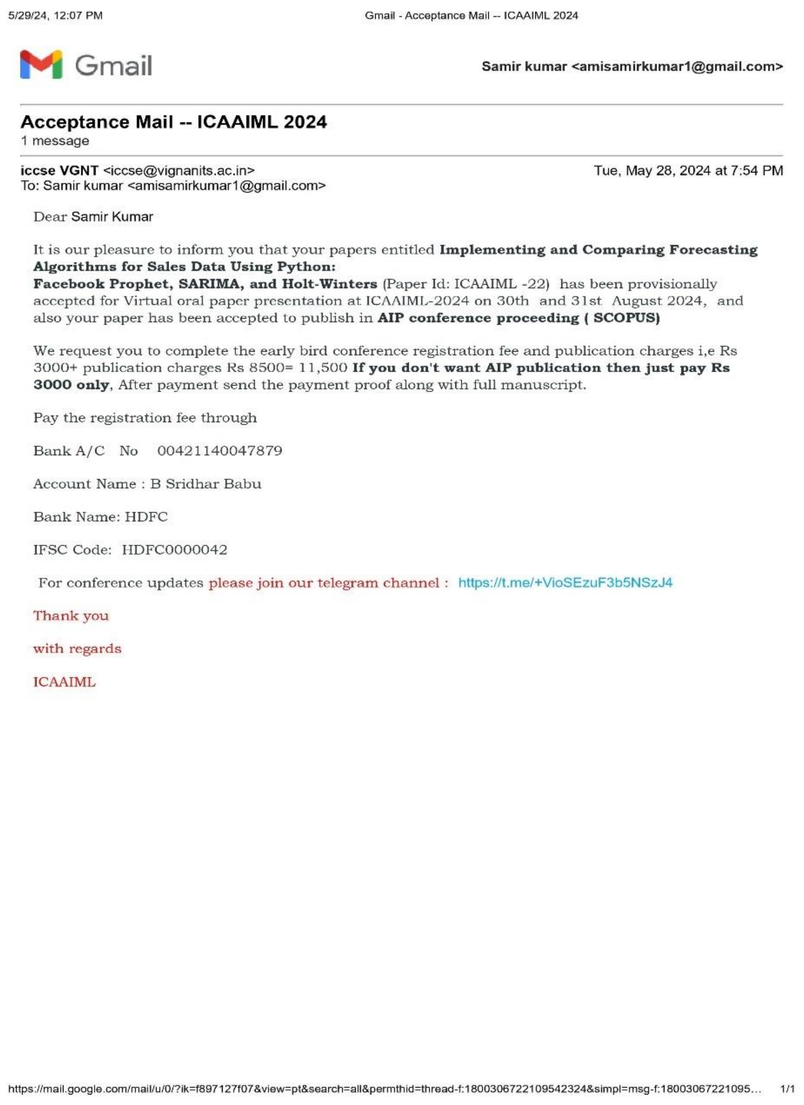
**DELHI TECHNOLOGICAL UNIVERSITY**

(Formerly Delhi College of Engineering)

Shahbad Daulatpur, Main Bawana Road, Delhi-42

## LIST OF PUBLICATIONS AND THEIR PROOF

| Title | Conference |
|---|---|
| Implementing and Comparing Forecasting Algorithms for Sales Data Using Python: Facebook Prophet, SARIMA, and Holt-Winters. | 1st International Conference on "Applied Artificial Intelligence, machine learning (ICAAIML 2024)" |



5/29/24, 12:07 PM Gmail - Acceptance Mail -- ICAAIML 2024

Gmail

Samir kumar <amisamirkumar1@gmail.com>

**Acceptance Mail -- ICAAIML 2024**

1 message

**iccse VGNT** <iccse@vignanits.ac.in> Tue, May 28, 2024 at 7:54 PM

To: Samir kumar <amisamirkumar1@gmail.com>

Dear Samir Kumar

It is our pleasure to inform you that your papers entitled **Implementing and Comparing Forecasting Algorithms for Sales Data Using Python: Facebook Prophet, SARIMA, and Holt-Winters** (Paper Id: ICAAIML -22) has been provisionally accepted for Virtual oral paper presentation at ICAAIML-2024 on 30th and 31st August 2024, and also your paper has been accepted to publish in **AIP conference proceeding ( SCOPUS)**

We request you to complete the early bird conference registration fee and publication charges i,e Rs 3000+ publication charges Rs 8500= 11,500 **If you don't want AIP publication then just pay Rs 3000 only**, After payment send the payment proof along with full manuscript.

Pay the registration fee through

Bank A/C No 00421140047879

Account Name : B Sridhar Babu

Bank Name: HDFC

IFSC Code: HDFC0000042

For conference updates please join our telegram channel : https://t.me/+VioSEzuF3b5NSzJ4

Thank you

with regards

ICAAIML

https://mail.google.com/mail/u/0/?ik=f897127f07&view=pt&search=all&permthid=thread-f:1800306722109542324&simpl=msg-f:18003067221095... 1/1

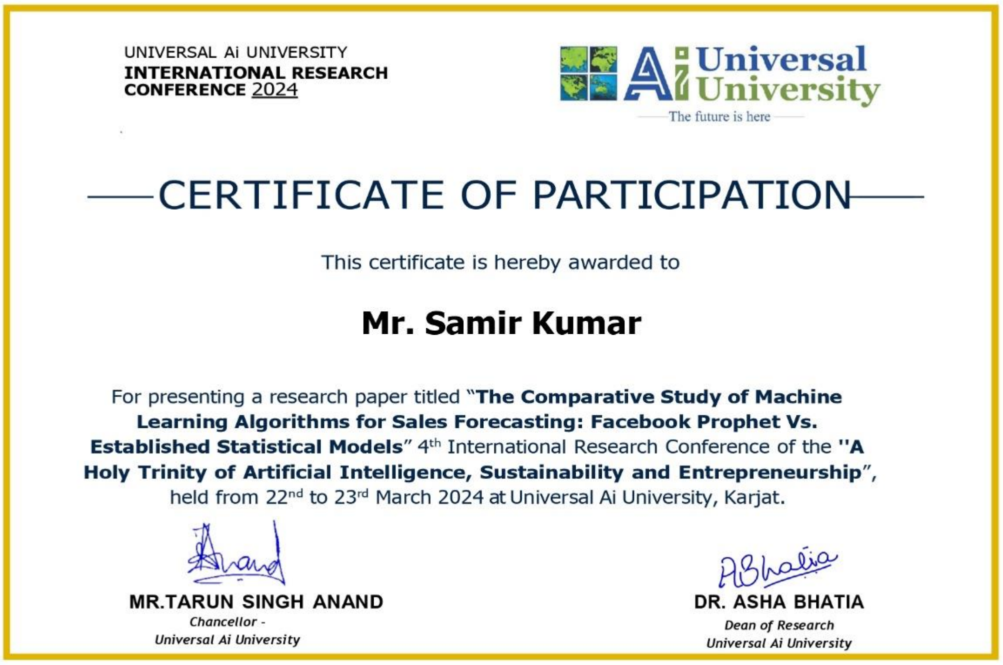
# DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Shahbad Daulatpur, Main Bawana Road, Delhi-42

**LIST OF PUBLICATIONS AND THEIR PROOF**

| Title | Conference |
|---|---|
| The Comparative Study of Machine Learning Algorithms for Sales Forecasting: Facebook Prophet Vs. Established Statistical Models | 4th International Research Conference of the "A Holy Trinity of Artificial Intelligence, Sustainability and Entrepreneurship" |



UNIVERSAL Ai UNIVERSITY
**INTERNATIONAL RESEARCH CONFERENCE** 2024

Ai Universal University
The future is here

CERTIFICATE OF PARTICIPATION

This certificate is hereby awarded to

**Mr. Samir Kumar**

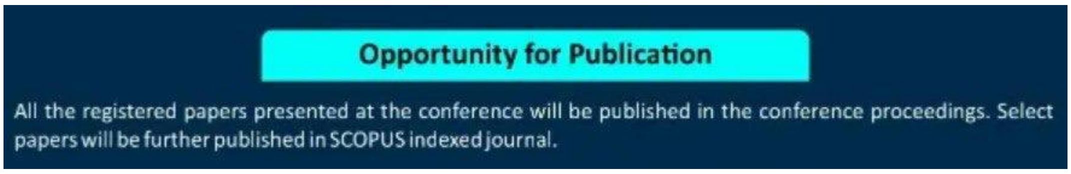
For presenting a research paper titled "**The Comparative Study of Machine Learning Algorithms for Sales Forecasting: Facebook Prophet Vs. Established Statistical Models**" 4th International Research Conference of the "**A Holy Trinity of Artificial Intelligence, Sustainability and Entrepreneurship**", held from 22nd to 23rd March 2024 at Universal Ai University, Karjat.

[illegible]

**MR.TARUN SINGH ANAND**
*Chancellor - Universal Ai University*

ABhatia

**DR. ASHA BHATIA**
*Dean of Research Universal Ai University*



**Opportunity for Publication**

All the registered papers presented at the conference will be published in the conference proceedings. Select papers will be further published in SCOPUS indexed journal.

**DELHI TECHNOLOGICAL UNIVERSITY**
(Formerly Delhi College of Engineering)
Shahbad Daulatpur, Main Bawana Road, Delhi-42

## PLAGIARISM VERIFICATION

Title of the Thesis Implementing and Comparing Forecasting Algorithms for Sales Data Using Python: Facebook Prophet, SARIMA, and Holt-Winters

Total Pages: Name of the scholar: Samir Kumar

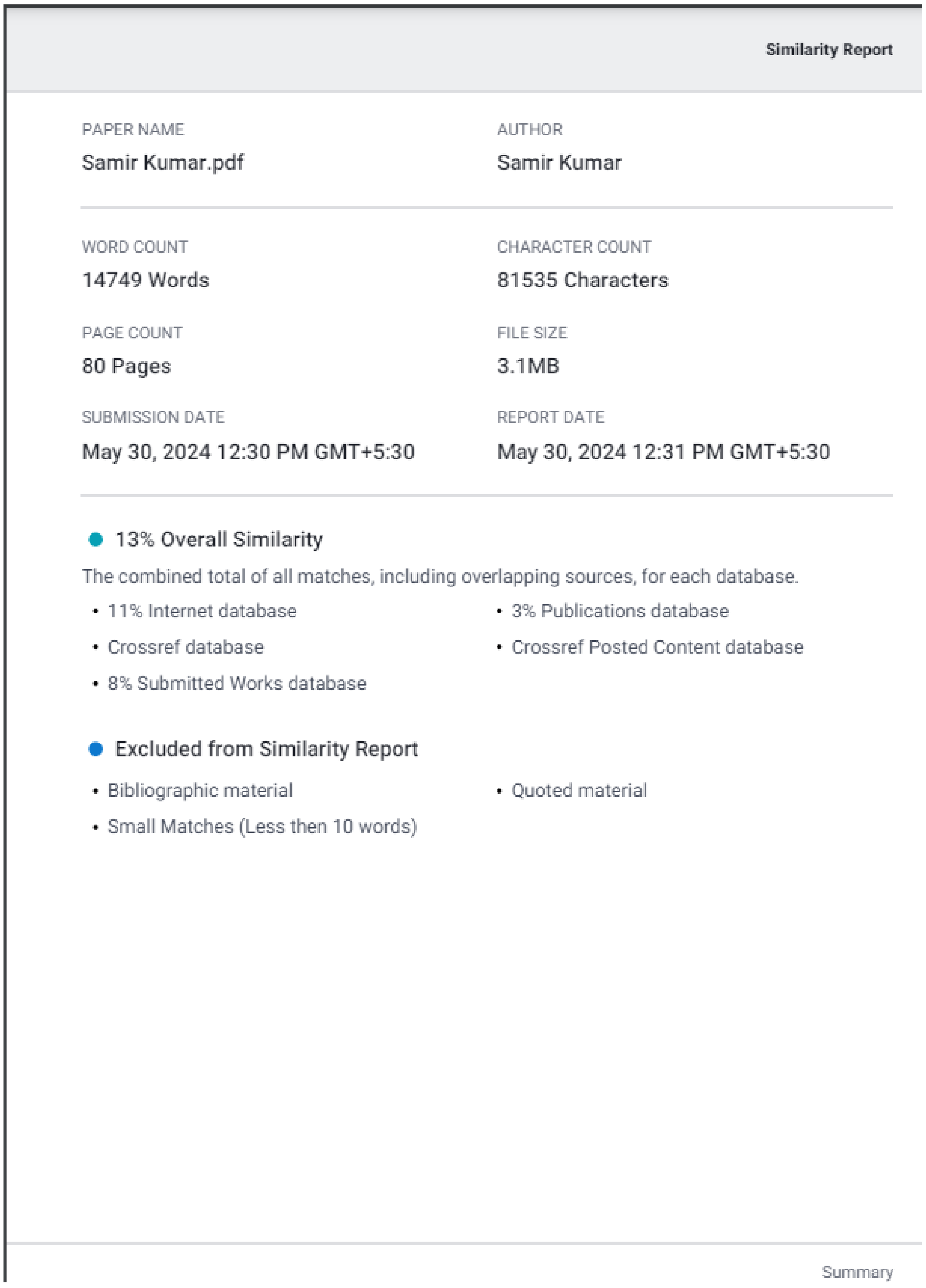
Supervisor(s): Dr. S. K. Garg
Department: Department of Mechanical Engineering
This is the report that the above thesis was scanned for similarity detection. The process and outcome are given below:

Software used: Turnitin Similarity Index: 13% Total Word Count: 14749

Date:

**Candidate's Signature** **Signature of Supervisor(s)**

## PLAGIARISM REPORT



Similarity Report

| | |
|---|---|
| PAPER NAME | AUTHOR |
| Samir Kumar.pdf | Samir Kumar |
| WORD COUNT | CHARACTER COUNT |
| 14749 Words | 81535 Characters |
| PAGE COUNT | FILE SIZE |
| 80 Pages | 3.1MB |
| SUBMISSION DATE | REPORT DATE |
| May 30, 2024 12:30 PM GMT+5:30 | May 30, 2024 12:31 PM GMT+5:30 |

● 13% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

- 11% Internet database
- 3% Publications database
- Crossref database
- Crossref Posted Content database
- 8% Submitted Works database

● Excluded from Similarity Report

- Bibliographic material
- Quoted material
- Small Matches (Less then 10 words)

Summary

# BRIEF PROFILE

My name is Samir Kumar, I am from Lakhisarai, Bihar. I have completed my Bachelor of Technology in Civil Engineering from Ch. Braham Prakash Government Engineering College, Jaffarpur, Delhi with 85.5 %. At present, I am pursuing Master of Technology in Industrial Engineering and Management from Delhi Technological University, New Delhi. I am currently working on a project topic "Implementing and Comparing Forecasting Algorithms for Sales Data Using Python: Facebook Prophet, SARIMA, and Holt-Winters".

**Software Skills**

Power Bi, Tableau, Python, SQL, MS Excel, QM

**Position of Responsibility**

Data Analyst Intern at STATXO Analytics PVT. LTD (April 2024 - Present)

Teaching Assistant at Delhi Technological University, Delhi (2022- 2024)