ARIMA-TrendSight: Forecasting Insights with Time Series Analysis

By

Tushar Upadhyay (Data Analytics Intern)

From

DayalBagh Educational Institute

Submitted to

Supervisor: Mr Guru Charan Bulusu(SWE)

From

Genisup

Table of content

ARIMA-TrendSight: Forecasting Insights with Time Series Analysis

1.INTRODUCTION:

This project is an intricate exploration and analysis of three distinct time series datasets, each offering unique insights into different domains. These datasets encompass national illness trends, champagne sales, and temperature demand. By leveraging advanced time series analysis techniques like Arima and Sarimax, the project seeks to uncover underlying patterns, trends, and predictive capabilities inherent within these datasets.

Each dataset, with its individual characteristics and relevance, contributes to a comprehensive understanding of various domains. These datasets are emblematic of national illness trends, champagne sales dynamics, and the intricate interplay between temperature fluctuations and demand patterns.

This project embarks on a meticulous exploration of diverse datasets, each a window into a distinct domain. Through the application of advanced time series analysis techniques, it aims to unearth hidden insights, identify trends, and enable accurate predictions. Whether in the realm of public health, business, or consumer behavior, the project's findings have the potential to reshape strategies, inform decisions. As the world navigates complex challenges, this project stands as a testament to the transformative power of data analysis and its role in shaping a more informed future.

Dataset 1: National Illness Trends

The first dataset immerses us in the realm of national illness trends, providing a comprehensive overview of health-related patterns over time. Variables such as % WEIGHTED ILI, % UNWEIGHTED ILI, age-specific illness counts, and ILITOTAL contribute to a rich dataset. The primary objective here is to employ sophisticated time series models to forecast illness trends accurately, thereby enabling evidence-based decision-making in the healthcare sector.

By doing so, the project aims to equip healthcare professionals and policymakers with valuable insights for informed decision-making. This dataset's significance lies in its capacity to guide resource allocation, track the efficacy of interventions, and improve overall public health strategies.

Dataset 2: Champagne Sales Analysis

Dataset two shifts our focus to the realm of business analytics, specifically exploring the dynamics of champagne sales. This dataset chronicles monthly champagne sales over a specified timeframe, offering valuable insights into trends and consumption patterns within the champagne industry. Through the application of time series forecasting methodologies, the project endeavors to predict future sales based on historical patterns, potentially assisting industry stakeholders in strategic planning and resource allocation.

Dataset 3: Temperature Demand Analysis

The third dataset, centers around temperature demand analysis. This dataset is likely to encapsulate time series data associated with temperature fluctuations and their impact on demand for certain products or services. Through meticulous analysis, the project aims to uncover correlations between temperature variations and corresponding shifts in demand,which could be instrumental in optimizing resource management and enhancing decision-making processes. By meticulously analyzing this relationship, the project may reveal vital insights for resource allocation, supply chain optimization, and overall operational efficiency.

2. PROBLEM STATEMENT

In a rapidly evolving landscape shaped by data-driven insights and technological advancements, this project seeks to address the multifaceted challenges associated with understanding, analyzing, and predicting trends within three distinct time series datasets. These datasets encompass national illness trends, champagne sales dynamics, and temperature demand correlations. Amidst the complexities of these domains, the project confronts the problem of deciphering underlying patterns, identifying trends, and harnessing predictive capabilities to inform decision-making across various sectors.

The first dataset, focusing on national illness trends, presents the challenge of comprehending intricate health-related variables and their evolution over a significant time span. The problem is to develop robust time series models capable of accurately forecasting illness trends. This entails overcoming the complexities of health data, addressing anomalies, and equipping stakeholders with actionable insights to optimize healthcare strategies.

The second dataset, revolving around champagne sales analysis, poses the challenge of unraveling the intricate dynamics of consumer behavior and market trends in the context of a luxury product. The project seeks to create predictive models that can navigate the unique challenges of this industry, encompassing seasonality, marketing influences, and fluctuations in consumer preferences. Solving this problem involves employing advanced time series techniques to optimize inventory management, marketing efforts, and overall business strategies.

Dataset three introduces the challenge of exploring the nexus between temperature variations and demand patterns, a problem with implications across industries reliant on weather-sensitive consumer behavior. The undisclosed nature of this dataset underscores the intricacy of the challenge, as the project aims to reveal the extent to which temperature impacts consumer decisions. Overcoming this challenge involves meticulous analysis and potentially discovering new relationships that can drive resource allocation, production planning, and supply chain optimization.

Overall, the project's problem statement revolves around leveraging advanced time series analysis techniques to decipher complex patterns, trends, and relationships within diverse datasets. The challenge lies in developing models that can tackle the unique intricacies of each dataset while providing accurate forecasts, actionable insights, and innovative solutions for sectors ranging from public health to business operations. By addressing these challenges, the project endeavors to bridge the gap between data analysis and informed decision-making in an era defined by data-driven strategies and technological advancements.

3. LITERATURE REVIEW

Time series forecasting plays a pivotal role in various fields, enabling researchers and practitioners to predict future values based on historical data patterns. One of the most widely used methods for time series forecasting is the AutoRegressive Integrated Moving Average (ARIMA) model. This literature review delves into the essence of ARIMA modeling, its applications, strengths, limitations, and its role in shaping the landscape of predictive analytics.

The ARIMA model is a versatile and powerful tool for forecasting time series data. It combines autoregressive (AR) and moving average (MA) components with the differencing process to transform a non-stationary time series into a stationary one. This innovation, coupled with the model's flexibility, allows it to capture complex temporal patterns.

Early applications of ARIMA in time series forecasting include "The Holt-Winters Forecasting Procedure" by Chatfield (1978), where ARIMA formed the basis for additive and multiplicative Holt-Winters models, addressing seasonality and trends. ARIMA's effectiveness is rooted in its ability to account for these components and identify the optimal order of differencing and lag values to

achieve stationarity. This was demonstrated in studies like "Forecasting Seasonal Outbreaks of Influenza" by Guan, Huang, and Zhou (2004), showcasing ARIMA's potential in predicting disease prevalence.

However, ARIMA is not without limitations. It assumes linear relationships and may struggle with complex nonlinear patterns. To overcome this, researchers have explored hybrid models that combine ARIMA with machine learning techniques.

The adoption of ARIMA has also been facilitated by advancements in computing power and user-friendly software tools. The introduction of packages like StatsModels in Python and auto.arima in R has democratized its usage, enabling analysts with varying expertise to harness its potential. Such accessibility is evident in projects like the "M3-Competition: Results, conclusions, and implications" by Makridakis and Hibon (2000), where ARIMA models competed successfully against more complex methods.

In summary, ARIMA has become a staple in time series forecasting due to its simplicity, versatility, and potential for delivering accurate predictions. Its historical contributions in predicting disease outbreaks, economic trends, and market demand underscore its adaptability across domains. While its linear assumptions may limit its efficacy in certain cases, hybrid approaches showcase how ARIMA can be enhanced with machine learning techniques. With the proliferation of user-friendly tools, ARIMA remains accessible to both seasoned analysts and newcomers, reaffirming its central role in the predictive analytics landscape. As the field continues to evolve, ARIMA's legacy endures as a fundamental method for extracting insights from time series data.

4.CONCEPT USED

The concepts used in the project are as follow

1.Independent and Identically Distributed (IID)

Independent: In the context of data, independent refers to the idea that the values or observations in a dataset do not influence each other. In other words, the occurrence or value of one observation does not impact the occurrence or value of another observation. This assumption is often necessary for various statistical methods and models.

Identically Distributed: This means that each observation in the dataset comes from the same probability distribution. In simpler terms, the data points share the same underlying characteristics, and the pattern of distribution is consistent across the entire dataset.

The IID assumption is particularly important for many statistical analyses and machine learning algorithms because it allows for the application of various mathematical and statistical techniques that assume the observations are independent and drawn from the same distribution.

Example in Documentation (Temperature Forecasting):

Let's take an example of temperature forecasting project using SARIMAX. In this context, the IID assumption might come into play in the following ways:

Modeling Assumptions: When discussing your data preparation and model selection, you might mention that the SARIMAX model assumes that the historical temperature data is IID. This assumption is important because it enables the model to make predictions based on the assumption that each day's temperature is independent of previous and future days, and the temperature fluctuations follow a consistent distribution.

Residual Analysis: After training your SARIMAX model, you'll likely perform residual analysis to check if the residuals (the differences between predicted and actual values) meet the IID assumption. If the residuals show patterns or dependencies, it could indicate that your model is not adequately capturing the data's characteristics.

Model Validity: The success of your forecasting model might rely on the IID assumption. If your historical temperature data violates the assumption (for example, if there's a strong trend or seasonality that's not accounted for), your model's predictions might not be accurate.

When documenting your project, explaining how you ensured or validated the IID assumption, or how deviations from it were handled, can add depth and clarity to your work.

The IID assumption might not always hold in real-world data, so it's important to be aware of its implications and limitations when applying statistical methods or models.

Let's consider a collection of random variables: ,…..X1,X2,X3,…,Xn, where Xn is the number of observations in your dataset.

Independence: The random variables X1,X2,X3,…,Xn are said to be independent if the outcome of one random variable doesn't affect the outcome of any other random variable. Mathematically, for any subset of indices i1,i2,…,ik, the joint probability distribution can be expressed as the product of the individual probabilities:

$$P(Xi1=xi1,Xi2=xi2,\ldots,Xik=xik)=P(Xi1=xi1)\cdot P(Xi2=xi2)\cdot\ldots\cdot P(Xik=xik)$$

Identically Distributed: The random variables X1,X2,X3,…,Xnare said to be identically distributed if they all follow the same probability distribution. This means that the probability distribution function (pdf) or probability mass function (pmf) of each random variable is the same:

$$P(X1=x)=P(X2=x)=P(X3=x)=\ldots=P(Xn=x)$$

Mathematical Implications:

The IID assumption simplifies the analysis of data and enables the use of various statistical methods and models. For instance:

Law of Large Numbers: The average of a large number of IID random variables converges to their expected value. This is the foundation for understanding the behavior of sample means.

Central Limit Theorem: The sum or average of a large number of IID random variables tends to have a distribution that approaches a normal (Gaussian) distribution, regardless of the original distribution of the variables.

Statistical Inference: Many statistical methods, such as hypothesis testing and confidence interval estimation, are based on the assumption of IID data. These methods rely on the properties of independence and identical distribution to draw meaningful conclusions about populations from sample data.

It's important to note that in practice, data might not always strictly satisfy the IID assumption. Real-world data can exhibit dependencies, temporal patterns, or non-identical distributions. When working with real-world data, it's crucial to consider the assumptions of your chosen methods and assess whether they hold or need to be relaxed.

The concept of IID is fundamental in statistics and plays a role in various statistical techniques, including those used in data forecasting, hypothesis testing, and more.Normal datasets and time series datasets have distinct characteristics due to the nature of the data they represent. Here are the key differences between these two types of datasets:

2.Normal Dataset and Time Series Dataset

1. Data Structure:

Normal Dataset (Cross-Sectional Data): In a normal dataset, also known as cross-sectional data, each observation corresponds to a single point in time. Each row represents an independent observation, and the columns represent different variables or features. These datasets are typically collected at a single time instance.

Time Series Dataset: In a time series dataset, observations are recorded at multiple time points in a sequential order. The data is collected over a period of time, and each observation is associated with a specific timestamp. Time series data often exhibits temporal dependencies and trends.

2. Temporal Dependency:

Normal Dataset: Observations in a normal dataset are assumed to be independent of each other. The order of observations does not hold any significance, and the data points are not connected through time.

Time Series Dataset: Time series data has a temporal component, meaning that the order of observations matters. The value of a data point at a certain time may be influenced by its previous values and external factors.

3. Patterns and Trends:

Normal Dataset: Normal datasets may not exhibit any specific patterns or trends across observations. Any observed relationships between variables are typically static and not time-dependent.

Time Series Dataset: Time series data often displays patterns, trends, and seasonality. Trends can be upward, downward, or remain stable over time. Seasonal patterns repeat at regular intervals, and cyclical patterns might also be present.

4. Data Analysis and Modeling:

Normal Dataset: Analyzing and modeling normal datasets often involve techniques like regression, classification, and clustering. The assumption of independent and identically distributed (IID) data is commonly applied.

Time Series Dataset: Analyzing time series data requires methods that consider the temporal structure. Techniques like autoregressive integrated moving average (ARIMA), exponential smoothing, and machine learning models specialized for time series are commonly used.

5. Handling Missing Data:

Normal Dataset: Missing data can be handled using various imputation techniques, and missing values might not significantly impact the analysis.

Time Series Dataset: Missing data in time series can be more challenging to handle due to the temporal dependencies. Different imputation methods need to consider the temporal nature of the data.

6. Forecasting:

Normal Dataset: Forecasting is not a primary concern in normal datasets since observations are not linked by time. The focus is on understanding relationships between variables.

Time Series Dataset: Forecasting future values is a key application of time series data. Methods like exponential smoothing, ARIMA, and more advanced techniques are used for predicting future values based on past observations.

Understanding these differences is crucial when analyzing and modeling data, as different methodologies and assumptions are required for normal datasets versus time series datasets. When working with time series data, it's important to account for the temporal dependencies and patterns inherent in the data.

7. Stationarity:

Normal Dataset: Assumptions of constant mean and variance are often made in normal datasets, but the concept of stationarity is not as critical. The data's statistical properties remain consistent across observations.

Time Series Dataset: Stationarity is a fundamental concept in time series analysis. A stationary time series has constant statistical properties over time, including constant mean, variance, and autocorrelation. Many time series models assume or require stationarity for accurate forecasting.

8. Autocorrelation:

Normal Dataset: Autocorrelation (correlation between the same variable at different time points) is not a primary concern in normal datasets.

Time Series Dataset: Autocorrelation is a crucial consideration in time series analysis. It indicates whether a variable's values at different time points are correlated. Autocorrelation helps identify seasonality and trends in the data.

9. Handling Multiple Observations:

Normal Dataset: Each observation in a normal dataset is typically independent and stands alone. The order of observations doesn't matter.

Time Series Dataset: The order of observations matters, and a sequence of observations is often used to capture temporal patterns and trends. The sequence of values creates a trajectory that informs the analysis.

10. Feature Selection:

Normal Dataset: Feature selection focuses on identifying relevant variables that contribute to the target variable's prediction or classification.

Time Series Dataset: Feature selection can involve both lagged versions of the target variable (previous time points) and external variables (exogenous factors) that influence the target variable at different time points.

11. Visualization:

Normal Dataset: Visualizations commonly include scatter plots, histograms, bar charts, and scatter matrix plots.

Time Series Dataset: Time series visualizations often include line plots to showcase the temporal evolution of variables. Additional visualizations like autocorrelation plots and seasonal decomposition plots are used to analyze patterns.

12. Data Preparation:

Normal Dataset: Data preparation may involve handling missing values and outliers, normalizing or scaling features, and encoding categorical variables.

Time Series Dataset: Data preparation includes steps to deal with seasonality, trend, and temporal dependencies. This might involve differencing to achieve stationarity and accounting for any temporal patterns.

Understanding these differences helps guide your approach when working with either normal datasets or time series datasets. Each type of data requires tailored analysis techniques, modeling approaches, and considerations.

13. Resampling:

Normal Dataset: Resampling methods like bootstrapping or cross-validation are commonly used to assess the generalization performance of models on random subsets of the data.

Time Series Dataset: When resampling time series data, care must be taken to preserve the temporal order. Techniques like time-based cross-validation (e.g., time series split) are used to ensure that future data points are not used for training.

14. Data Exploration:

Normal Dataset: Data exploration often involves exploring relationships between variables and identifying patterns that provide insights into the dataset's characteristics.

Time Series Dataset: Data exploration in time series includes analyzing seasonality, trend components, and identifying any anomalies or outliers over time.

15. Non-stationarity:

Normal Dataset: Non-stationarity might not be a major concern since each observation is assumed to be independent.

Time Series Dataset: Addressing non-stationarity (changing statistical properties over time) is crucial in time series analysis. Techniques like differencing or seasonal decomposition are used to achieve stationarity.

16. Data Transformation:

Normal Dataset: Data transformation might involve log transformations, power transformations, or scaling to meet assumptions of normality or equal variance.

Time Series Dataset: Data transformation often includes differencing to stabilize the variance or make the data stationary. This is important for time series modeling.

17. Memory and Persistence:

Normal Dataset: Each observation is considered independently and doesn't retain memory of past observations.

Time Series Dataset: Time series data has memory or persistence, where past values can influence future values due to temporal dependencies.

18. Independence:

Normal Dataset: $P(X1=x1, X2=x2, \ldots, Xn=xn) = P(X1=x1) \cdot P(X2=x2) \cdot \ldots \cdot P(Xn=xn)$

Time Series Dataset: Since time series data has a temporal order, the concept of independence across time points is not explicitly modeled with a single equation. Instead, autocorrelation and cross-correlation functions are used to quantify relationships between observations at different time lags.

19. Identically Distributed:

Normal Dataset: In a normal dataset, the concept of identical distribution doesn't require specific mathematical equations. It assumes that all random variables follow the same distribution.

Time Series Dataset: In time series analysis, we often assume that each observation follows the same distribution over time, but the notation might not be as explicit as in cross-sectional data. The assumption is used in various modeling techniques.

20. Stationarity:

Normal Dataset: Stationarity is not a primary concern in normal datasets, as the independence assumption is dominant.

Time Series Dataset: Stationarity is crucial in time series analysis. A stationary time series can be expressed as:

$E[Xt]=\mu$ and $Var[Xt]=\sigma 2$for allt

21. Autocorrelation:

Normal Dataset: Autocorrelation is not a primary concept in normal datasets, as observations are independent.

Time Series Dataset: Autocorrelation of a time series Xt at lag k is calculated using:

22 Trends and Seasonality:

Normal Dataset: Trends and seasonality are not typically modeled in normal datasets, as they are assumed to be absent.

Time Series Dataset: Trends and seasonality are modeled using various methods. For instance, a time series with a trend component Tt and seasonal component St might be expressed as:

$$Xt=Tt+St+\epsilon t$$

23. Moving Average:

Normal Dataset: Moving averages are not typically used in normal datasets since the order of observations doesn't have a temporal aspect.

Time Series Dataset: A moving average at time t with a window size w is calculated as:

$$MAt=w1∑i=t−w+1tXi$$

This smooths out noise and highlights trends over time.

24. Autoregressive Models:

Normal Dataset: Autoregressive models aren't typically used in normal datasets.

Time Series Dataset: Autoregressive models like ARIMA (AutoRegressive Integrated Moving Average) are widely used in time series analysis. The ARIMA(p, d, q) model represents a combination of autoregressive (AR), integrated (I), and moving average (MA) terms.

25. Integration:

Normal Dataset: Integration is not typically a consideration in normal datasets.

Time Series Dataset: The "I" in ARIMA represents the integration process, where differencing is used to achieve stationarity. A differenced time series Yt is obtained by subtracting the value at time t−1 from the value at time t:

$$Yt=Xt−Xt−1$$

26. Seasonal Decomposition:

Normal Dataset: Seasonal decomposition is not applicable to normal datasets.

Time Series Dataset: Time series can be decomposed into components: trend, seasonality, and residual. Mathematically, the observed time series Xt can be decomposed as:

$$Xt=Tt+St+Et$$

Where Tt is the trend, St is the seasonality, and Et is the residual (noise).

27. Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF):

Normal Dataset: ACF and PACF are not typically used in normal datasets, as the concept of autocorrelation between observations is not significant.

Time Series Dataset: The ACF of a time series Xt measures the autocorrelation between the time series at lag k and k time periods before. The PACF measures the correlation between Xt and Xt−k after accounting for correlations with all the intermediate time points.

28. Seasonal ACF and PACF:

Normal Dataset: Seasonal ACF and PACF are not relevant in normal datasets.

Time Series Dataset: For time series with seasonality, the seasonal ACF and PACF can help identify the seasonal order (P, Q terms) in models like SARIMA (Seasonal ARIMA).

29. Autoregressive Integrated Moving Average (ARIMA) Model:

Normal Dataset: The ARIMA model is not typically used with normal datasets.

Time Series Dataset: ARIMA models are widely used in time series analysis. The ARIMA(p, d, q) model combines autoregressive (AR), integrated (I), and moving average (MA) terms to model time series data.

3.Stationarity And Non Stationarity

Stationarity and non-stationarity are fundamental concepts in time series analysis, defining the characteristics of a dataset over time. These concepts play a crucial role in determining the appropriateness of various modeling techniques, including the popular Auto Regressive Integrated Moving Average (ARIMA) model. Let's delve into the mathematical underpinnings of stationarity and non-stationarity and their implications for time series analysis.

Stationarity:

Stationarity refers to the constancy of statistical properties in a time series across different time periods. Mathematically, a time series Xt is stationary if its moments, such as mean and variance, remain constant over time.

- **Constant Mean ($[E[X_t]=\mu$):** In a stationary time series, the average value remains the same across different time intervals. This implies that there is no underlying trend driving the data.
- **Constant Variance ($Var[X_t]=\sigma^2$):** The variance of the data points remains consistent across time. This assumption ensures that the spread of the data does not change over different time intervals.
- **Autocovariance and Autocorrelation ($Cov[X_t, X_t+k]=\gamma(k), Corr[X_t, X_t+k]=\rho(k)$):** Stationary data exhibit consistent autocovariance and autocorrelation structures across time lags k. This implies that the relationships between data points at different time intervals are stable.

Non-Stationarity:

Non-stationarity occurs when the statistical properties of a time series change over time. This often manifests as trends, seasonality, or changing levels of variance.

- **Linear Trends**: Non-stationary time series may exhibit linear trends, where the data points follow a systematic increase or decrease over time. Mathematically, this can be expressed as $X_t=\beta_0+\beta_1 t+\epsilon_t$, where β_0 and β_1 represent intercept and slope coefficients, and ϵtis the random error term.

Differencing: Transforming Non-Stationary to Stationary:

Differencing is a common technique used to transform non-stationary time series into stationary ones. First-order differencing ($\Delta X_t=X_t-X_{t-1}$) involves subtracting consecutive observations to remove trends. This process helps stabilize the mean and variance of the data, making it more amenable to analysis.

Implications for ARIMA Modeling:

The AutoRegressive Integrated Moving Average (ARIMA) model, a widely used time series forecasting method, is built on the concept of stationarity. The "Integrated" component in ARIMA signifies the differencing required to achieve stationarity. The model assumes that after differencing, the data follow a stationary process describable by autoregressive (AR) and moving average (MA) components.

Stationarity and non-stationarity are fundamental concepts that shape the landscape of time series analysis. The mathematical definitions and characteristics of these concepts guide preprocessing steps to render time series data suitable for modeling. ARIMA models, grounded in stationary time series data, leverage these concepts to uncover patterns and make predictions. Understanding these foundational principles is essential for accurate and insightful time series analysis across diverse domains.

4.ARIMA(Auto Regressive Integrated Moving Average)

ARIMA stands for Auto Regressive Integrated Moving Average. It's a popular and powerful time series forecasting model used to analyze and predict time-dependent data. ARIMA combines three key components: AutoRegressive (AR), Integrated (I), and Moving Average (MA) terms to capture different aspects of time series patterns.

- **Auto Regressive(AR) Component:** An AR(p) model uses the p most recent observations to predict the next value. Mathematically, an AR(p) model can be represented as

$$X_t = c + \phi 1 X_{t-1} + \phi 2 X_{t-2} + ... + \phi_p X_{t-p} + \epsilon_t$$

Where X_t is the value at time t, c is a constant, $\phi_1, \phi_2, ..., \phi_p$ are the autoregressive coefficients, and ϵt is the white noise error term at time t.

- **Moving Average (MA) Component:** The Moving Average component captures the relationship between the current value and past error terms. An MA(q) model uses the *q* most recent error terms to predict the next value. Mathematically, an MA(q) model can be represented as:

$$X_t = c + \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + ... + \theta_q \epsilon_{t-q}$$

where $\theta_1, \theta_2, ..., \theta_q$ are the moving average coefficients.

- **Integrated (I) Component:** The Integrated component represents the order of differencing needed to achieve stationarity. A non-stationary time series might require differencing to stabilize its mean and variance. The differenced series is denoted as $Y_t = X_t - X_{t-1}$, and the differencing process is applied *d* times until stationarity is achieved.

Model Selection:

Choosing the appropriate values for p, d, and q is crucial for building an effective ARIMA model. Common methods for model selection include:

- Visual inspection of ACF and PACF plots to identify potential values for p and q.
- Evaluating the AIC or BIC (Bayesian Information Criterion) values for different model combinations.
- Using cross-validation to assess model performance on out-of-sample data.

Forecasting:

Once an ARIMA model is fitted to the historical data, it can be used for forecasting future values. Forecasting involves using the model's equations to predict future observations. The accuracy of forecasts depends on the quality of the model, the availability of recent data, and the stability of the underlying time series patterns.

Limitations:

While ARIMA is a versatile model, it has certain limitations:

It assumes linear relationships between variables, which might not hold in all cases.ARIMA is sensitive to outliers and can produce unreliable forecasts if extreme values are presentIt might not perform well for highly nonlinear or complex time series patterns.

Advanced Versions:

To address certain limitations and capture more complex patterns, variations of ARIMA have been developed:

SARIMA (Seasonal ARIMA): Includes seasonal components to model seasonal patterns.

ARIMAX (ARIMA with Exogenous Variables): Incorporates external variables that might influence the time series.

SARIMAX: Combines seasonal components with exogenous variables.Overall, ARIMA is a foundational and widely-used model in time series analysis, offering a structured approach to understanding and forecasting time-dependent data.

Model Estimation:

After determining the values of p, d, and q, the next step is to estimate the coefficients ϕ for AR terms, θ for MA terms) of the ARIMA model. This is often done using methods such as least squares estimation or maximum likelihood estimation. The goal is to find the parameter values that minimize the sum of squared errors or maximize the likelihood of observing the given data.

Model Diagnostics:

It's essential to perform diagnostics to assess the quality of the ARIMA model:

Residual Analysis: Examine the residuals (observed values minus model predictions) for autocorrelation and heteroskedasticity (changing variance over time).

Normality of Residuals: Check if the residuals follow a normal distribution using statistical tests or visual methods like a Q-Q plot.

Forecast Accuracy: Evaluate the accuracy of forecasts using metrics like Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), or Mean Absolute Percentage Error (MAPE).

Seasonal ARIMA (SARIMA):

When dealing with time series that exhibit seasonal patterns, SARIMA (Seasonal ARIMA) is used. It extends the basic ARIMA model to include seasonal components, denoted as P, D, and Q for seasonal autoregressive, seasonal differencing, and seasonal moving average terms, respectively. The notation for SARIMA is SARIMA p,d,q(P,D,Q)s).

Choosing Differencing Orders:

Determining the appropriate order of differencing (d and D) is crucial for achieving stationarity without over-differencing. You can use techniques like the augmented Dickey-Fuller test or visual inspection of the time series plot to guide your choice.

Backtesting and Out-of-Sample Testing:

To assess the forecasting performance of an ARIMA model, it's important to conduct backtesting and out-of-sample testing. Backtesting involves fitting the model to a subset of historical data and then comparing the model's forecasts to the actual values. Out-of-sample testing evaluates the model's performance on unseen data that wasn't used during model estimation.

Model Updating:

Time series data can evolve over time due to changing trends, seasonality, or external factors. Regularly updating the ARIMA model by re-estimating parameters and forecasting based on new data helps ensure accurate predictions.

Python Libraries:-

ARIMA modeling can be implemented using various programming languages and libraries. In Python, you can use libraries like statsmodels and pmdarima, while in R, the forecast package is commonly used.

Limitations and Considerations:

- ARIMA assumes that the underlying patterns in the time series are linear and that the relationships between variables are fixed over time.
- Extreme outliers can heavily influence ARIMA's performance.
- For long forecast horizons, the uncertainty in predictions can increase significantly.

Choosing ARIMA vs. Other Models:

While ARIMA is a powerful model, it might not always be the best choice for all time series data. Depending on the data's characteristics, other models like Seasonal Decomposition of Time Series (STL), Exponential Smoothing (ETS), or machine learning models like LSTM networks might be more suitable.

In summary, ARIMA is a foundational model for time series analysis, and understanding its intricacies can help you effectively model and forecast time-dependent data.

the concepts of Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF), which are essential tools in time series analysis:

5.Autocorrelation Function (ACF) And Partial Autocorrelation Function (PACF):

Autocorrelation Function (ACF):

The Autocorrelation Function (ACF) is a statistical tool used to quantify the correlation between a time series and its lagged values. It helps you understand how past observations are related to each other over different time lags. The ACF plot displays the correlation coefficient on the y-axis and the lag (time difference) on the x-axis.

Key points about ACF:

- ACF at lag 0 is always 1, as an observation is perfectly correlated with itself.
- A positive ACF at a specific lag indicates a positive correlation between the observations at that lag.

- A negative ACF at a specific lag indicates a negative correlation between the observations at that lag.
- ACF values closer to 0 suggest weaker or no correlation between observations at that lag.

Partial Autocorrelation Function (PACF):

The Partial Autocorrelation Function (PACF) measures the correlation between two observations in a time series while accounting for the influence of the observations at all intermediate lags. It helps you identify the direct relationship between observations at specific lags without the impact of other lags.

Key points about PACF:

- PACF at lag 0 is always 1, similar to ACF.
- A positive PACF at a specific lag indicates a direct correlation between observations at that lag, after accounting for the influence of observations at other lags.
- A negative PACF at a specific lag indicates a direct negative correlation between observations at that lag.
- PACF values that are not significantly different from zero suggest that the correlation between the observations is largely explained by previous lags.

Interpretation:

ACF and PACF plots are used to identify potential values for the autoregressive (AR) and moving average (MA) orders of an ARIMA model. Here's how to interpret these plots:

ACF Plot:

If the ACF plot shows a significant positive spike at lag k, it suggests that an AR(k) term might be appropriate.

If the ACF plot shows a significant negative spike at lag k, it suggests that a MA(k) term might be appropriate.

PACF Plot:

If the PACF plot shows a significant positive spike at lag k, it suggests that an AR(k) term might be appropriate.

If the PACF plot shows a significant negative spike at lag k, it suggests that an MA(k) term might be appropriate.

Remember, the ACF and PACF plots are used as guides to select potential values for p (AR order) and q (MA order) in an ARIMA model. They provide insights into the temporal dependencies within the time series data and help you choose appropriate model components for accurate forecasting.

ACF and PACF plots and their significance in time series analysis:

Pure AutoRegressive (AR) Model:

In the ACF plot, the autocorrelation tails off gradually, and the PACF plot cuts off sharply after lag p.

This suggests that the data is correlated with its past values up to lag p, and there is little correlation beyond that.

Pure Moving Average (MA) Model:

In the PACF plot, the partial autocorrelation tails off gradually, and the ACF plot cuts off sharply after lag q.

This suggests that the data is correlated with its past errors up to lag q, and there is little correlation beyond that.

Mixed AutoRegressive Moving Average (ARMA) Model:

In both the ACF and PACF plots, correlations tail off gradually, indicating both short-term memory and some long-term memory.

You might consider using an ARMA(p,q) model, combining autoregressive and moving average components.

Seasonal Patterns:

If you observe significant spikes in ACF or PACF at seasonal lags, it suggests the presence of seasonality.

In seasonal data, the spikes might occur at lags corresponding to the length of the seasonal pattern.

Exponential Decay:

An exponential decay pattern in ACF suggests a geometric decrease in correlation as the lag increases.

Alternating Spikes: Alternating spikes in ACF and PACF suggest a SARIMA model with seasonal and autoregressive components.

Damped Sinusoidal Patterns: If you observe damped sinusoidal patterns in ACF, it might indicate the presence of periodicity.Remember, the identification of appropriate orders (p and q) based on ACF and PACF plots is an iterative process. You might need to adjust the orders and refine your model based on the fit and diagnostics.

Lags and Causality:It's important to note that significant correlations in ACF and PACF plots do not imply causality. The presence of correlation might be due to other factors or common underlying patterns. Causality should be determined through domain knowledge, experimentation, or additional analysis.

Software Implementation: Many statistical software packages like Python's **statsmodels** or R's provide functions to plot ACF and PACF plots directly from your time series data. These plots aid in model selection and parameter tuning for ARIMA and related models.

6.ARIMA with P,D & Q:

"ARIMA with PDQ" refers to the ARIMA model specification, where P, D, and Q represent the orders of the autoregressive (AR), differencing (I), and moving average (MA) components, respectively. Let's break down each component and how they come together in an ARIMA(p,d,q) model:

P - Autoregressive (AR) Component: The autoregressive component models the relationship between the current value of a time series and its past values. The order P indicates the number of lagged observations used in the model. For example, if P=1, the model includes the value at the previous time step to predict the current value.

D - Differencing (I) Component: The differencing component is used to achieve stationarity by subtracting the time series from itself with a certain lag. The order D indicates the number of differencing operations applied to the time series. Differencing helps remove trends and make the data more stationary.

Q - Moving Average (MA) Component: The moving average component captures the relationship between the current value and past error terms. The order Q indicates the number of lagged error terms used in the model. For example, if Q=1, the model includes the error term from the previous time step to predict the current value.

When you put these components together, you get an ARIMA(p,d,q) model, which represents the combination of autoregressive, differencing, and moving average terms to model and forecast time series data.

Example: Let's say you have a time series of monthly sales data. You observe that the data is non-stationary and has a clear seasonal pattern. You might perform first-order differencing (D=1) to achieve stationarity. Additionally, you notice significant autocorrelation in the ACF plot up to lag 12 and significant partial autocorrelation up to lag 1. This suggests a potential autoregressive term (P=1) and a potential moving average term (Q=1).

Based on this analysis, you might specify an ARIMA(1, 1, 1) model to capture the autoregressive, differencing, and moving average components in the data.

It's important to note that determining the appropriate values for p, d, and q requires careful analysis, including ACF and PACF plots, as well as model diagnostics. Also, in cases of seasonality, you might consider extending the model to SARIMA by including seasonal AR, seasonal differencing, and seasonal MA terms (Ps, Ds, Qs).

Autoregressive (AR) Component (p): The autoregressive component captures the relationship between the current value and its past values. The value of p determines the number of lagged observations used in the model. For example, an ARIMA(1, d, 0) model includes the value at the previous time step to predict the current value, and an ARIMA(2, d, 0) model includes the values at the previous two time steps.

The order of p is typically determined using the Partial Autocorrelation Function (PACF) plot. The lag at which the PACF plot cuts off can provide a hint about the appropriate value for p.

Differencing (I) Component (d): The differencing component is used to achieve stationarity by subtracting the time series from itself with a certain lag. The value of d represents the number of differencing operations applied to the time series. Differencing helps remove trends and seasonality, making the data more stationary.

The value of d can be determined by examining the behavior of the time series. If the time series exhibits a trend, seasonal pattern, or changing variance, you might need to apply differencing until these patterns are removed.

Moving Average (MA) Component (q): The moving average component captures the relationship between the current value and past error terms. The value of q determines the number of lagged error terms used in the model. For example, an ARIMA(0, d, 1) model includes the error term from the previous time step to predict the current value.

The order of q is often determined using the Autocorrelation Function (ACF) plot. The lag at which the ACF plot cuts off can provide insights into the appropriate value for q.

Overall Model Selection: The goal is to find the most appropriate combination of p, d, and q that captures the underlying patterns and relationships in the data while maintaining model simplicity. The ACF and PACF plots play a crucial role in guiding the choice of these values. Additionally, techniques like cross-validation and model evaluation metrics (e.g., AIC, BIC) can help compare different ARIMA models and select the best-fitting one.

SARIMA (Seasonal ARIMA) Extension: In cases of seasonality, you might encounter situations where the seasonal pattern isn't addressed solely by differencing. This is where the seasonal ARIMA (SARIMA) model comes into play. SARIMA extends the ARIMA model by including seasonal autoregressive (SAR), seasonal differencing (SAD), and seasonal moving average (SMA) terms.

The seasonal components (Ps, Ds, Qs) in SARIMA are similar to the non-seasonal components (p, d, q), but they operate at the seasonal frequency. For example, Ps represents the number of seasonal lags to consider in the seasonal AR component.

Here's how you can perform residual analysis in the context of ARIMA:

7. Residual Plots:

Residual vs. Time Plot: Plot the residuals against time to check for any remaining trends or seasonality in the residuals. If the residuals exhibit a pattern, it might indicate that the model hasn't fully captured the underlying patterns in the data.

Residual Histogram: Create a histogram of the residuals to check if they follow a normal distribution. A roughly symmetric, bell-shaped histogram suggests that the residuals are normally distributed.

2. Autocorrelation of Residuals:

Plot the ACF (Autocorrelation Function) of the residuals. If there are significant spikes at certain lags, it might indicate that the model is not accounting for some temporal dependencies in the data.

3. Ljung-Box Test:

The Ljung-Box test is a statistical test that assesses the autocorrelation of the residuals. It checks if the autocorrelations up to a certain lag are statistically significant. If the test indicates significant autocorrelations, it suggests that the model might still have some systematic patterns.

4. Normality Test:

Perform statistical tests, such as the Jarque-Bera test or the Shapiro-Wilk test, to check the normality of the residuals. If the p-value of these tests is low, it indicates that the residuals might not be normally distributed.

5. Mean and Variance of Residuals:

Calculate the mean and variance of the residuals. Ideally, the mean should be close to zero, and the variance should be relatively constant. Large mean or varying variance might suggest issues with the model.

6. Outliers and Anomalies:

Look for any outliers or unusual observations in the residuals. Outliers might indicate that the model is not accounting for extreme values.

7. Model Fit Assessment:

Overall, the residuals should appear as random noise with no systematic patterns. If you observe any consistent patterns or deviations, it might be an indication that the model needs further refinement .

8.Introduction about Random Walk

A random walk is a fundamental concept in time series analysis and statistics. It refers to a stochastic process where each value in a sequence is determined by the previous value plus a random increment or "shock." In a simple random walk, each step is independent of previous steps and is determined solely by chance.

Mathematically, a simple random walk can be represented as:

$X_t = X_{t-1} + \varepsilon_t$
Where:

X_t is the value at time t.

X_{t-1} is the value at the previous time step.

ε_t is a random shock or increment at time t.

Key characteristics of a random walk:

No Predictable Pattern: In a random walk, each step is purely random and unpredictable. There is no systematic pattern or trend guiding the process.

Cumulative Effect: Even though each step is random, the cumulative effect of many random steps can lead to apparent trends or patterns that might mislead an observer into thinking there is a structure.

Non-Stationarity: A random walk is inherently non-stationary. The mean and variance of the series increase over time, leading to changing statistical properties.

Difficult Forecasting: Random walks are notoriously difficult to forecast accurately since future values depend on unpredictable shocks. In forecasting, the best prediction for the next value is often the current value.

Random Walk with Drift: In some cases, a constant drift term can be added to the random walk equation to account for a systematic trend. This results in a "random walk with drift," where there's a tendency for the series to move in a particular direction.

Certainly, let's explore some additional aspects of random walks and their significance:

1. Random Walk vs. White Noise:A white noise series is a sequence of uncorrelated random values with a constant mean and variance. A random walk, on the other hand, involves accumulating the effects of previous random shocks. While both appear unpredictable, a random walk shows a cumulative effect over time, leading to trends.

2. Random Walk as a Building Block: Random walks serve as building blocks for more complex models. For instance, a geometric random walk is used in the Black-Scholes option pricing model in finance. It models the continuous compounding of asset prices.

3. Random Walk and Market Efficiency: The Efficient Market Hypothesis (EMH) suggests that stock prices follow a random walk with respect to new information. This implies that all available information is immediately reflected in stock prices, making it difficult to consistently outperform the market.

4. Trend Detection: Detecting trends in time series data can be challenging due to the presence of noise. A simple random walk can mimic a trend, highlighting the importance of distinguishing between genuine trends and noise-driven patterns.

5. Random Walks in Economics: In economics, random walks are used to model variables that are influenced by unpredictable factors, such as exchange rates, commodity prices, and short-term economic indicators.

6. Testing for Random Walks: Statistical tests like the Augmented Dickey-Fuller (ADF) test are used to test whether a time series follows a random walk or exhibits a unit root (non-stationarity). These tests help determine if differencing is required to achieve stationarity.

7. Long-Range Dependencies: While simple random walks are memoryless, there are variations like fractional Brownian motion that exhibit long-range dependencies. These models have applications in fields like network traffic modeling and hydrology.

8. Implications for Forecasting: The unpredictability of random walks has implications for forecasting. It reminds us that forecasts should consider multiple factors beyond historical data, such as external events and context.

9. Limitations of the Model: Random walks might not accurately capture all the complexities of real-world time series data. Many time series exhibit trends, seasonality, cyclic behavior, and interactions that go beyond the basic assumptions of a random walk.

9.RNN,CNN And LSTM

Certainly, RNN (Recurrent Neural Network), CNN (Convolutional Neural Network), and LSTM (Long Short-Term Memory) are advanced neural network architectures used in various domains, including natural language processing, computer vision, and time series analysis. Let's explore each of these architectures and their applications:

1. RNN (Recurrent Neural Network): RNNs are designed to process sequences of data by introducing the concept of memory through hidden states. Each neuron in an RNN takes an input and produces an output while maintaining an internal hidden state that captures information from previous time steps. This makes RNNs suitable for tasks involving sequential data.

Applications:

Natural Language Processing (NLP): RNNs are used for tasks like language modeling, machine translation, sentiment analysis, and text generation.

Time Series Analysis: RNNs can model temporal dependencies in time series data, making them suitable for forecasting and anomaly detection.

Speech Recognition: RNNs process audio sequences to convert spoken language into text.

2. CNN (Convolutional Neural Network): CNNs are primarily used for image and grid-like data analysis. They leverage convolutional layers to automatically learn hierarchical patterns in data. CNNs are particularly effective at capturing local patterns, making them a go-to choice for computer vision tasks.

Applications:

Image Classification: CNNs excel in tasks like image classification, where they can automatically learn features and patterns from raw pixel data.

Object Detection: CNNs are used for detecting and localizing objects within images.

Image Segmentation: CNNs partition images into meaningful segments, useful in medical imaging and scene understanding.

3. LSTM (Long Short-Term Memory): LSTMs are a specific type of RNN designed to address the vanishing gradient problem. They have a more complex internal structure that allows them to capture longer-range dependencies in data. LSTMs are particularly effective for modeling sequences with time lags between important events.

Applications:

Time Series Forecasting: LSTMs are widely used for time series forecasting due to their ability to capture complex temporal relationships.

Natural Language Processing: LSTMs excel in tasks like language modeling, text generation, and machine translation, where context matters.

Speech Recognition: LSTMs are used to model sequences of audio data for speech recognition tasks.

PROJECT 1: - NATIONAL ILLNESS TRENDS

1.Title: -National Illness Trends

2.Objective: - Analyze and forecast national illness trends using time series analysis, focusing on patterns, SARIMAX modeling, and healthcare impact.

3.Scope:- This project's scope encompasses in-depth analysis and forecasting of national illness trends via time series methods, contributing insights into healthcare dynamics, utilizing SARIMAX modeling, and addressing real-world implications of illness patterns.

4.Methodology:-

4.1 Data Preprocessing:-

In this section, we will discuss the steps taken to preprocess the dataset before applying the ARIMA model for time series prediction.

1. **Loading the Data:**
   The initial step involves loading the dataset containing historical illness data. We use the pandas library to read the CSV file.

   ```
   import pandas as pd
   # Load the dataset
   df = pd.read_csv("national_illness.csv")
   ```

2. **Converting Date to DateTime and Setting Index:**
   Time series analysis requires the date column to be in the datetime format and set as the index of the DataFrame. This facilitates proper time-based analysis.

   ```
   # Convert the 'date' column to datetime format
   df['date'] = pd.to_datetime(df['date'])

   # Set the 'date' column as the index
   df.set_index('date', inplace=True)
   ```

3. **Handling Missing Data:**
   Missing data can adversely affect the quality of the time series analysis. To mitigate this, we remove rows with missing values from the dataset.

   ```
   # Drop rows with missing values
   df_cleaned = df.dropna()
   ```

   Missing data can lead to biased results and affect the quality of the analysis. In our dataset, missing values were managed by using the .dropna() method to remove rows with any NaN values. This approach ensures that the analysis is performed on a complete dataset without introducing imputation biases.

4. **Data Transformation and Normalization:**
   Depending on the dataset characteristics, data transformation and normalization might be necessary. These steps can help stabilize variance and make the data more suitable for analysis

Out[51]:

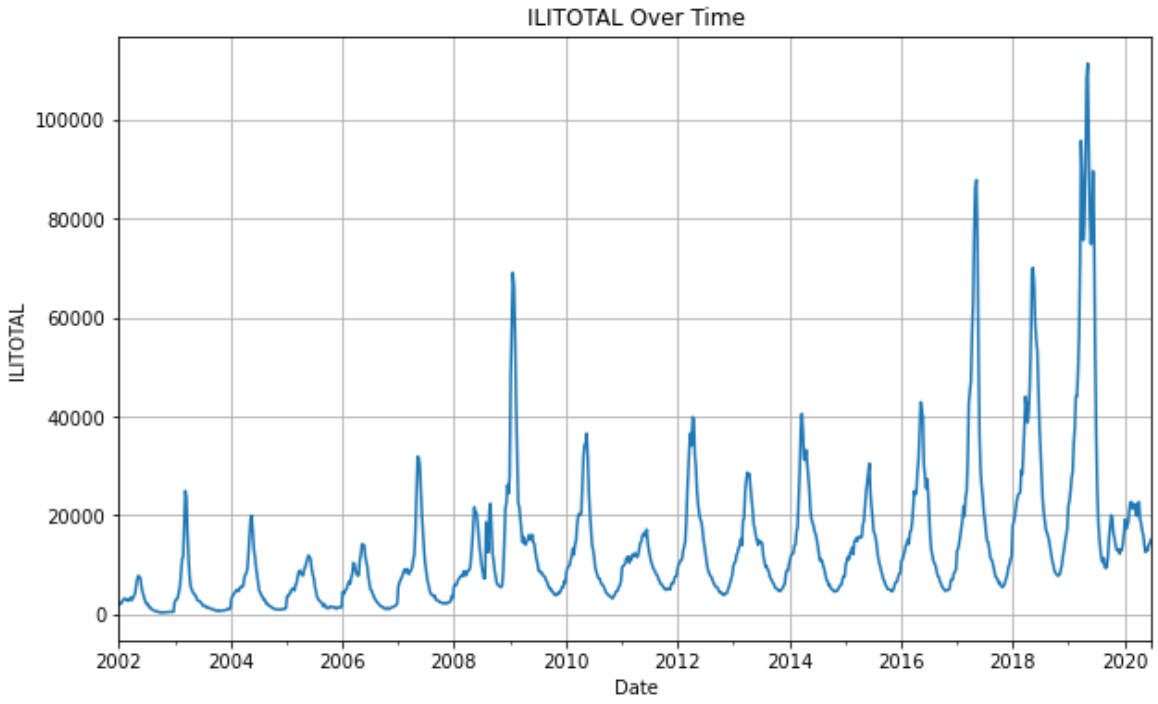
	date	% WEIGHTED ILI	%UNWEIGHTED ILI	AGE 0-4	AGE 5-24	ILITOTAL	NUM. OF PROVIDERS	OT
0	2002-01-01 00:00:00	1.222620	1.166680	582	805	2060	754	176569
1	2002-01-08 00:00:00	1.333440	1.216500	683	872	2267	785	186355
2	2002-01-15 00:00:00	1.319290	1.130570	642	878	2176	831	192469
3	2002-01-22 00:00:00	1.494840	1.252460	728	1045	2599	863	207512
4	2002-01-29 00:00:00	1.471950	1.302370	823	1189	2907	909	223208
...	...	...	...	...	...	...	...	...
961	2020-06-02 00:00:00	0.839059	0.846722	2756	3528	12913	3258	1525058
962	2020-06-09 00:00:00	0.895958	0.908885	3203	3778	13979	3254	1538038
963	2020-06-16 00:00:00	0.910926	0.941625	3478	3796	14389	3177	1528103
964	2020-06-23 00:00:00	0.946945	0.972185	3734	3818	14999	3066	1542813
965	2020-06-30 00:00:00	0.963716	1.013760	3955	3843	15307	3027	1509928

966 rows × 8 columns

5. Data Visualization:

Visualizing the preprocessed data provides insights into trends and patterns. Plotting the time series data helps in understanding its characteristics.

```
import matplotlib.pyplot as plt

# Plotting the preprocessed data
plt.figure(figsize=(10, 6))
plt.plot(df_cleaned.index, df_cleaned['ILITOTAL'], marker='', label='ILITOTAL')
plt.xlabel('Date')
plt.ylabel('ILITOTAL')
plt.title('Preprocessed ILITOTAL Over Time')
plt.legend()
plt.grid(True)
plt.show()
```



4.2 Stationarity Transformation:

To apply ARIMA models effectively, it's important to work with stationary time series data. Stationarity implies that the statistical properties of the time series, such as mean and variance, do not change over time. In this section, we discuss the techniques employed to transform the data into a stationary form.

1. **Check for Stationarity:**
   We begin by checking the stationarity of the time series. This can be done visually through plots or statistically using tests like the Augmented Dickey-Fuller (ADF) test.

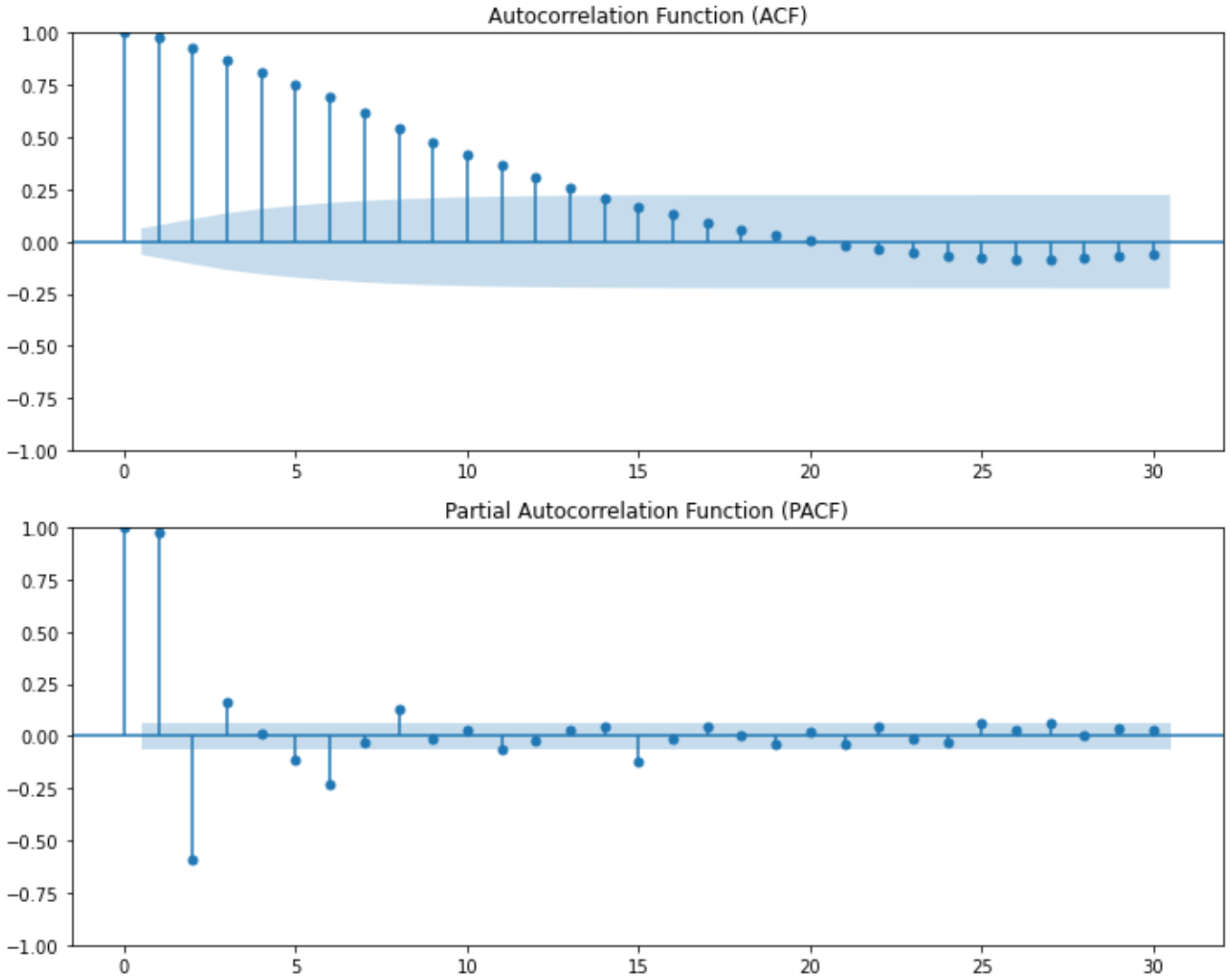
```
from statsmodels.tsa.stattools import adfuller


# Perform ADF test on the time series
result = adfuller(df_cleaned['ILITOTAL'])

# Extract test statistics and p-value
adf_statistic = result[0]
p_value = result[1]

# Interpret the test results
if p_value <= 0.05:
   print('The time series is stationary.')
else:
   print('The time series is non-stationary.')
```

Result:-

```
ADF Statistic: -6.161281947965547
p-value: 7.163900092341713e-08
The time series is stationary.
```

2. **ACF and PACF Plots:**
   AutoCorrelation Function (ACF) and Partial AutoCorrelation Function (PACF) plots help determine the order of differencing (d) and identify the p and q parameters for the ARIMA model.

```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import matplotlib.pyplot as plt

# Plot ACF and PACF plots
fig, ax = plt.subplots(nrows=2, ncols=1, figsize=(10, 8))

# ACF plot
plot_acf(df_diff, ax=ax[0], lags=30)
ax[0].set_title('Autocorrelation Function (ACF)')

# PACF plot
plot_pacf(df_diff, ax=ax[1], lags=30)
ax[1].set_title('Partial Autocorrelation Function (PACF)')

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



4.3Model Selection:

Choosing the appropriate order of the ARIMA parameters (p, d, q) is a crucial step in building an effective time series forecasting model. In this section, we outline the steps taken to determine the optimal parameters for the ARIMA model.

1. ACF and PACF Analysis:

ACF and PACF plots generated in the previous step provide insights into the possible values of p and q. The ACF plot indicates the potential value of q by looking at the lag where the autocorrelation drops off significantly, while the PACF plot suggests the value of p based on the partial autocorrelation.

2. Grid Search:

A grid search approach involves trying out multiple combinations of p, d, and q values and selecting the one with the lowest AIC (Akaike Information Criterion) score. The AIC balances the goodness of fit with the complexity of the model.

```
from pmdarima import auto_arima

# Use auto_arima to determine the optimal p, d, and q values

model = auto_arima(time_series, seasonal=False, trace=True)

# Print the selected model order (p, d, q)

print(f'ARIMA Model Order: ({model.order[0]}, {model.order[1]}, {model.order[2]})')
```

Result:-

```
Best model:  ARIMA(5,1,2)(0,0,0)[0]
Total fit time: 26.059 seconds
ARIMA Model Order: (5, 1, 2)
```

4.4Model Fitting and Forecasting:

Once the optimal ARIMA parameters are determined, the next steps involve fitting the ARIMA model to the training data and generating forecasts. This section outlines the process of model fitting, validation, and generating future forecasts.

1. Model Fitting:

Using the selected ARIMA parameters (p, d, q), fit the ARIMA model to the training data. This involves initializing the ARIMA model and passing the training data to the model for estimation.

```
import statsmodels.api as sm

# Define the exogenous variables for the model

exog_variables = ['% WEIGHTED ILI', '%UNWEIGHTED ILI', 'AGE 0-4', 'AGE 5-24', 'NUM. OF PROVIDERS', 'OT']

# Define the order of the ARIMA and seasonal components

order = (5, 1, 2)  # (p, d, q) order for ARIMA component

seasonal_order = (5, 1, 2, 12)  # (P, D, Q, S) order for seasonal component

# Create and fit the SARIMAX model with exogenous variables

model = sm.tsa.SARIMAX(train_data['ILITOTAL'], order=order, seasonal_order=seasonal_order, exog=train_data[exog_variables])

result = model.fit()
```

2. Model Validation:

After fitting the model, validate its performance on a validation dataset. Calculate relevant metrics such as Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Percentage Error (MAPE) to evaluate the accuracy of the model's predictions.

 # Generate predictions on the test data

predictions = result.predict(start=len(train_data), end=len(train_data) + len(test_data) - 1, exog=test_data[exog_variables])

 # Calculate Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Percentage Error (MAPE)

mse = np.mean((predictions - test_data['ILITOTAL'])2)**

rmse = np.sqrt(mse)

mape = np.mean(np.abs((predictions - test_data['ILITOTAL']) / test_data['ILITOTAL'])) * 100

3. Future Forecasting:

Generate forecasts for future time steps using the fitted ARIMA model. This involves providing the exogenous variables for the forecast period and using the get forecast function.

 # Generate forecasts for future time steps

forecast_demand_sarimax = result.get_forecast(steps=len(test_data), exog=exog_test)

predicted_mean = forecast_demand_sarimax.predicted_mean

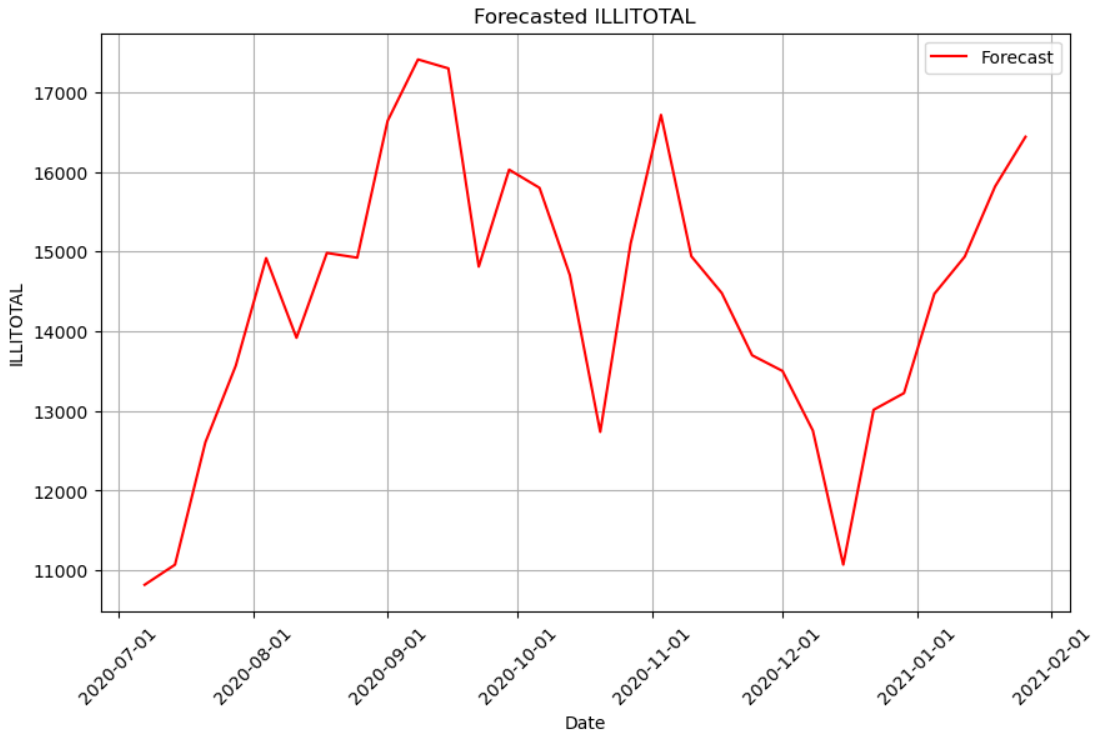
Result:-

```
2020-07-07    10833.991470
2020-07-14    11112.789182
2020-07-21    12687.829030
2020-07-28    13656.875846
2020-08-04    14975.611611
2020-08-11    13984.623503
2020-08-18    15136.086315
2020-08-25    15117.627947
2020-09-01    16853.412342
2020-09-08    17648.462302
2020-09-15    17569.015954
2020-09-22    15115.752548
2020-09-29    16325.638591
2020-10-06    16087.199797
2020-10-13    14981.172356
2020-10-20    13031.995972
2020-10-27    15429.519857
2020-11-03    17031.691827
2020-11-10    15214.006372
2020-11-17    14714.104337
2020-11-24    13907.635877
2020-12-01    13669.149337
2020-12-08    12895.601865
2020-12-15    11232.427604
2020-12-22    13199.474864
2020-12-29    13428.748317
2021-01-05    14695.637327
2021-01-12    15160.199785
2021-01-19    16034.071032
2021-01-26    16657.282211
```

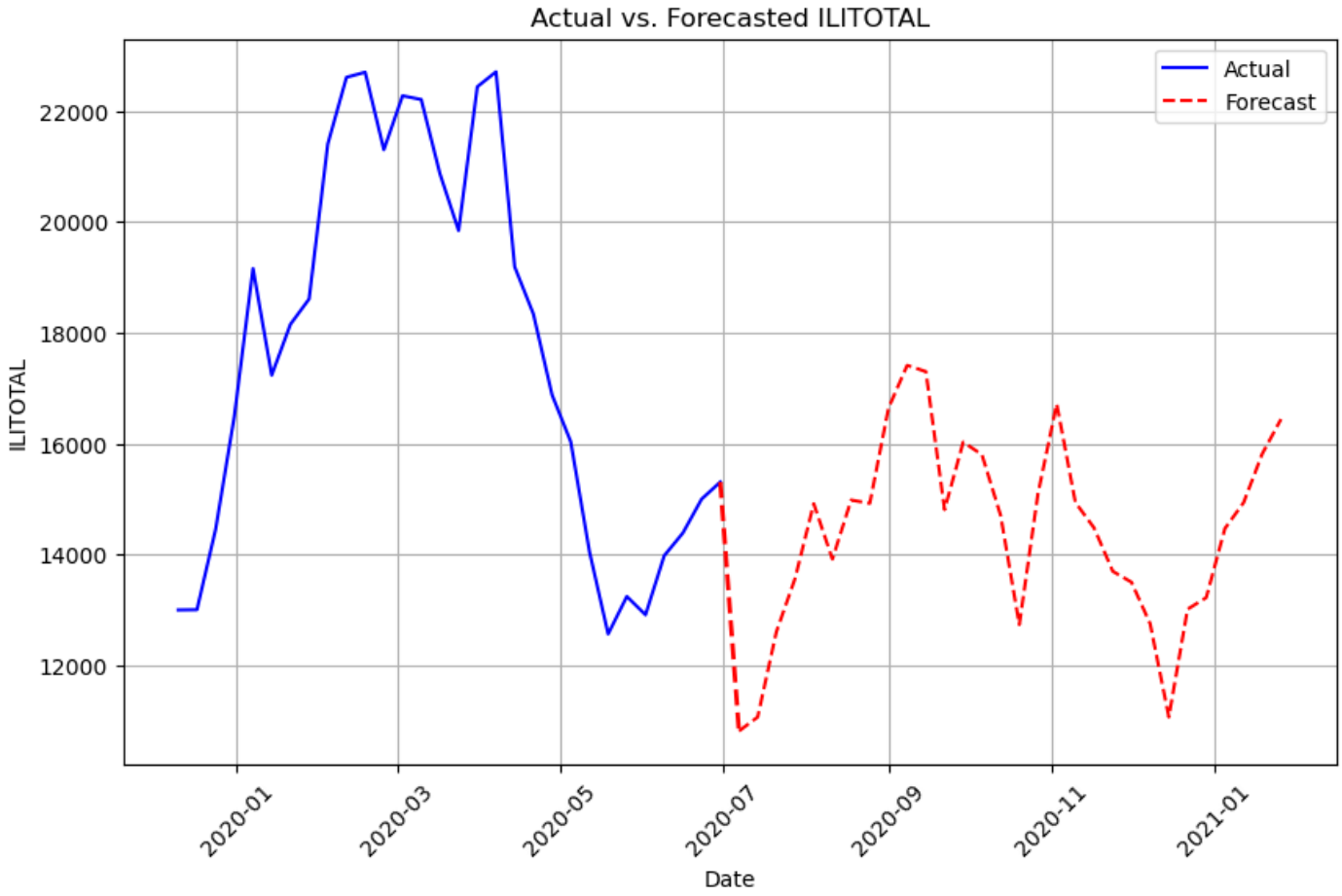
4. Visualization:

Visualize the forecasted values along with the actual data to assess the accuracy of the model's predictions.

```
import matplotlib.pyplot as plt

import matplotlib.dates as mdates

plt.figure(figsize=(10, 6))

# Plot actual test data

plt.plot(test_data.index, test_data['ILITOTAL'], label='Actual', color='blue')

# Plot forecasted values

plt.plot(predicted_mean.index, predicted_mean, label='Forecast', color='red', linestyle='--')

plt.xlabel('Date')

plt.ylabel('ILITOTAL')

plt.title('Actual vs. Forecasted ILITOTAL')

plt.legend()

plt.grid(True)

plt.xticks(rotation=45)

plt.show()
```



5.Result:-



6.Challenges and Solutions:

Implementing the ARIMA model for time series forecasting can come with various challenges. Here, we outline these challenges and provide potential solutions for each:

1.Non-Stationarity of Data

Solution: Apply differencing to make the data stationary. Use techniques like first-order differencing or seasonal differencing to remove trends and seasonality from the data.

2.Model Order Selection

Solution: Utilize ACF and PACF plots to identify potential values for AR and MA orders. Additionally, employ methods like grid search or automated tools (e.g., auto_arima) to determine the optimal values of p, d, and q.

3.Exogenous Variables Integration

Solution: Integrate relevant exogenous variables into the model to capture their impact on the time series. Make sure to preprocess and align the exogenous variables with the time series data.

4.Model Validation

Solution: Use a validation dataset to assess the model's accuracy. Calculate metrics like MSE, RMSE, and MAPE to quantify the prediction errors.

5.Overfitting

Solution: Be cautious of overfitting, especially with complex models. Use techniques like cross-validation and information criteria (e.g., AIC, BIC) to choose a model that balances fit and complexity.

Challenge 6: Handling Outliers and Anomalies

Solution: Identify and handle outliers that can adversely affect model performance. Techniques like winsorization or outlier removal can be applied.

7.Forecasting Uncertainty

Solution: Consider using prediction intervals to quantify the uncertainty around the forecasts. This helps to provide a range of possible outcomes.

8.Interpretability

Solution: While ARIMA models are powerful, their interpretations can be challenging. Use diagnostic plots, model summaries, and domain knowledge to interpret the model results.

9.Data Volume

Solution: In cases of limited data, apply resampling techniques (e.g., upsampling or downsampling) to modify the time frequency. However, be cautious with downsampling as it can lead to loss of information.

10.Implementation and Maintenance

Solution: Implementing the ARIMA model involves coding and various parameters. Using libraries like statsmodels or pmdarima can simplify the process. Keep in mind that maintaining and updating the model as new data becomes available is essential for accurate forecasting.

10. Future Enhancements

▪ Experiment with different SARIMAX configurations by varying the orders of AR, I, and MA components to identify the optimal configuration for accuracy.

▪ Include more granular exogenous variables to capture additional external influences that might affect sales.

▪ Explore alternative time series forecasting methods such as Prophet or LSTM to compare and improve accuracy.

PROJECT 2: - CHAMPAGNE SALES ANALYSIS

1.Title: -Champagne Sales Analysis

2.Objective: - Analyze champagne sales data and forecast using SARIMA for better business decisions in inventory, production, and marketing.

3.Scope:- Scope includes utilizing SARIMA to forecast champagne sales, aiding decisions in inventory, production, and marketing. The analysis aims to optimize resources, align supply with demand, and enhance strategic planning for improved business performance.

4.Methodology:-

4.1 Data Preprocessing:-

In this section, we will outline the steps taken to preprocess the sales dataset before applying the ARIMA model for time series prediction.

Step 1: Loading the Data

We begin by loading the sales dataset using the Pandas library.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

df=pd.read_csv("C:\\Users\\TUSHAR UPADHYAY\\Downloads\\perrin-freres-monthly-champagne-(1).csv")

df.head()
```

The process begins by loading the dataset utilizing the powerful pandas library's pd.read_csv() function. This function enables the extraction and import of data from a CSV file, allowing for further analysis. Once the dataset is imported, the subsequent step involves data cleaning.

Step 2: Converting Date to DateTime and Setting Index

To perform time-based analysis, we convert the date column to the datetime format and set it as the index.

```
# Convert 'date' column to datetime format

df['Month']=pd.to_datetime(df['Month'])

# Set 'date' column as the index

df.set_index('Month',inplace=True)
```

In this step, we focus on enabling time-based analysis by transforming the dataset's date information. The "Month" column, which represents time intervals, is subjected to a pivotal conversion. By invoking the pd.to_datetime() function, the content of the "Month" column is seamlessly converted into the datetime format. This format not only captures the chronological aspect of time but also equips the data with various time-related attributes.By converting the "Month" column to datetime format, the dataset gains the capacity to reflect temporal relationships accurately. This format interprets each entry as a specific point in time, enabling calculations, comparisons, and visualizations based on the inherent time-based nature of the data.

Step 3: Handling Missing Data

Missing data can affect the accuracy of the analysis. We address this by removing rows with missing values.

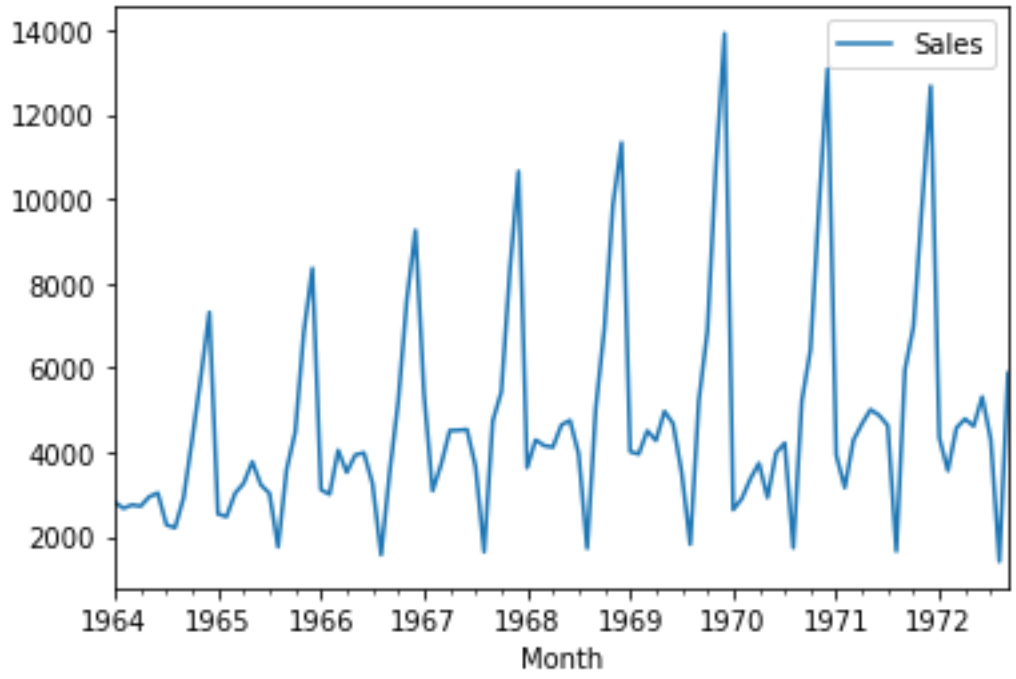
 # Drop rows with missing values

Drop last 2 rows

df.drop(106,axis=0,inplace=True)
df.drop(105,axis=0,inplace=True)

In this crucial step, we confront the challenge posed by missing data within the dataset. Missing data points, if not appropriately managed, can introduce biases, distort analyses, and yield inaccurate insights. To safeguard the integrity and reliability of subsequent analyses, a strategic approach to addressing these gaps is indispensable.

The method chosen here is to eliminate rows that contain missing values. Specifically, two instances of data removal are executed. The data set appears to contain rows with indexes 105 and 106 that are potentially irrelevant or corrupted. Employing the df.drop() function with the inplace=True parameter allows the direct modification of the DataFrame, effectively removing the designated rows.



Step 4: Data Transformation and Normalization

Depending on the nature of the data, we might need to apply transformations such as seasonal differencing transformation or normalization to stabilize variance and make the data more suitable for analysis. Here's an example of applying the log transformation:

 # Apply as seasonal differencing transformation normalization

df['Sales First Difference'] = df['Sales'] - df['Sales'].shift(1)
df['Seasonal First Difference']=df['Sales']-df['Sales'].shift(12)

This step is crucial in shaping the dataset for more accurate and informative analysis. Depending on the nature of the data and the specific requirements of the analysis, applying various transformations and normalization techniques can be instrumental in extracting meaningful insights and patterns.

One common approach is the application of seasonal differencing transformation. This technique involves calculating the difference between the current value and the value from a previous season, which can help eliminate seasonality trends and stabilize the variance in the data. The expression df['Sales'] - df['Sales'].shift(12) represents this seasonal differencing transformation, where the "Sales" column is subtracted by its value from the same month in the previous year (12 months back).

Month	Sales	Sales First Difference	Seasonal First Difference
1964-01-01	2815.0	NaN	NaN
1964-02-01	2672.0	-143.0	NaN
1964-03-01	2755.0	83.0	NaN
1964-04-01	2721.0	-34.0	NaN
1964-05-01	2946.0	225.0	NaN
1964-06-01	3036.0	90.0	NaN
1964-07-01	2282.0	-754.0	NaN
1964-08-01	2212.0	-70.0	NaN
1964-09-01	2922.0	710.0	NaN
1964-10-01	4301.0	1379.0	NaN
1964-11-01	5764.0	1463.0	NaN
1964-12-01	7312.0	1548.0	NaN
1965-01-01	2541.0	-4771.0	-274.0
1965-02-01	2475.0	-66.0	-197.0

Step 5: Data Visualization

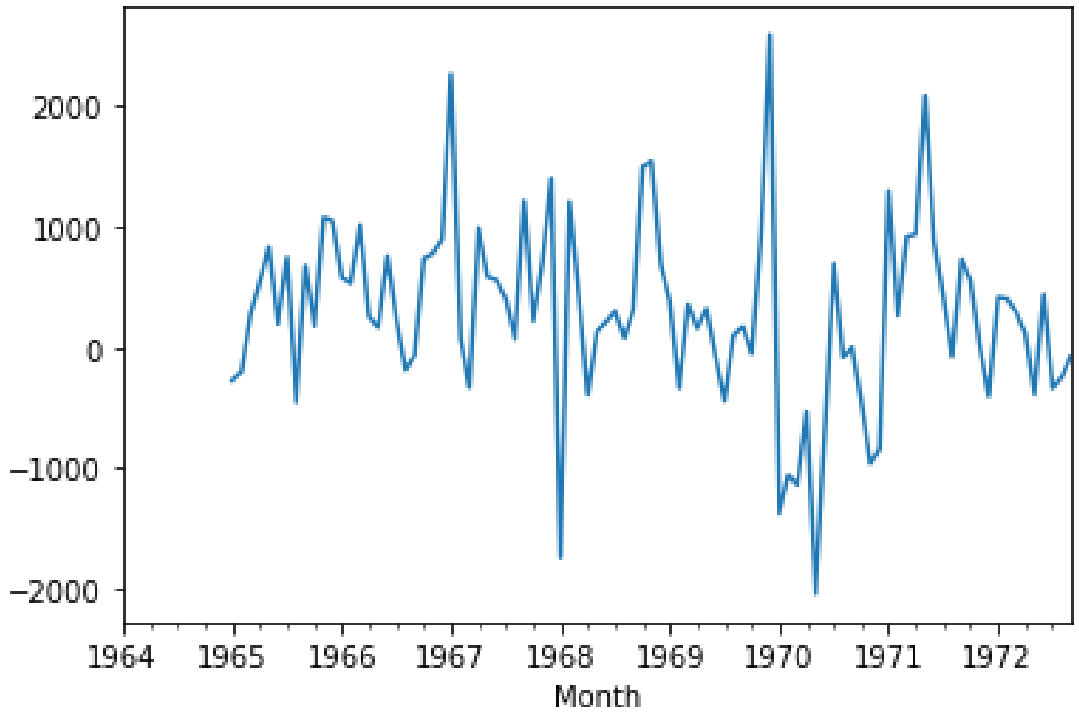
Visualizing the preprocessed data helps us understand trends and patterns. We plot the time series data to gain insights into its characteristics.

```
import matplotlib.pyplot as plt

# Plotting the preprocessed sales data

df['Seasonal First Difference'].plot()
```

In this step, the power of data visualization is harnessed to uncover trends, patterns, and characteristics within the preprocessed dataset. By presenting the data in graphical form, insights that might be challenging to glean from raw numbers become apparent, making visualization an invaluable tool for understanding complex datasets.

The provided code employs the widely used Matplotlib library, importing it as plt. This library provides a comprehensive set of tools to generate diverse visualizations, empowering analysts to represent data in meaningful and informative ways.

The code snippet further utilizes the plot() function to visualize the seasonal first difference data. This data is a result of the seasonal differencing transformation performed earlier. By plotting this transformed data, patterns that might not have been evident in the raw data can emerge. Seasonal patterns, trends, and anomalies become more visible, aiding in the identification of cyclic behaviors and other time-dependent phenomen



4.2 Stationarity Transformation:

In order to effectively apply ARIMA models, it's crucial to work with time series data that exhibits stationarity. Stationarity implies that the essential statistical properties of the time series, including mean and variance, remain constant over time. In this section, we delve into the techniques used to transform the data into a stationary form

1.Checking for Stationarity

We begin by testing the stationarity of the time series data using the Augmented Dickey-Fuller (ADF) test. The code imports necessary libraries and defines a function adfuller_test to perform the ADF test. The function prints ADF test statistics, p-value, and other information. It then applies the ADF test to the 'Sales' column and the 'Seasonal First Difference' column (after dropping missing values). The results determine whether the data is stationary.

```
from statsmodels.tsa.stattools import adfuller

def adfuller_test(sales):

    result = adfuller(sales)
    labels = ['ADF Test Statistic', 'p-value', '#Lags Used', 'Number of Observations Used']
    for value, label in zip(result, labels):
        print(label + ' : ' + str(value))
    if result[1] <= 0.05:
        print("Strong evidence against the null hypothesis (Ho), reject the null hypothesis. Data has no unit root and is stationary.")
    else:
        print("Weak evidence against the null hypothesis, time series has a unit root, indicating it is non-stationary.")

# Perform ADF test on the 'Sales' column
test_result = adfuller(df['Sales'])
adfuller_test(df['Sales'])
```

Result:-

ADF Test Statistic : -1.833593056327624
p-value : 0.36391577166024447
#Lags Used : 11
Number of Observations Used : 93
weak evidence against null hypothesis, time series has a unit root, indicating it is non-stationary

After seasonal differencing transformation again do the ADF test on transform data.

 # Perform ADF test on the 'Seasonal First Difference' column (dropping missing values)

adfuller_test(df['Seasonal First Difference'].dropna())

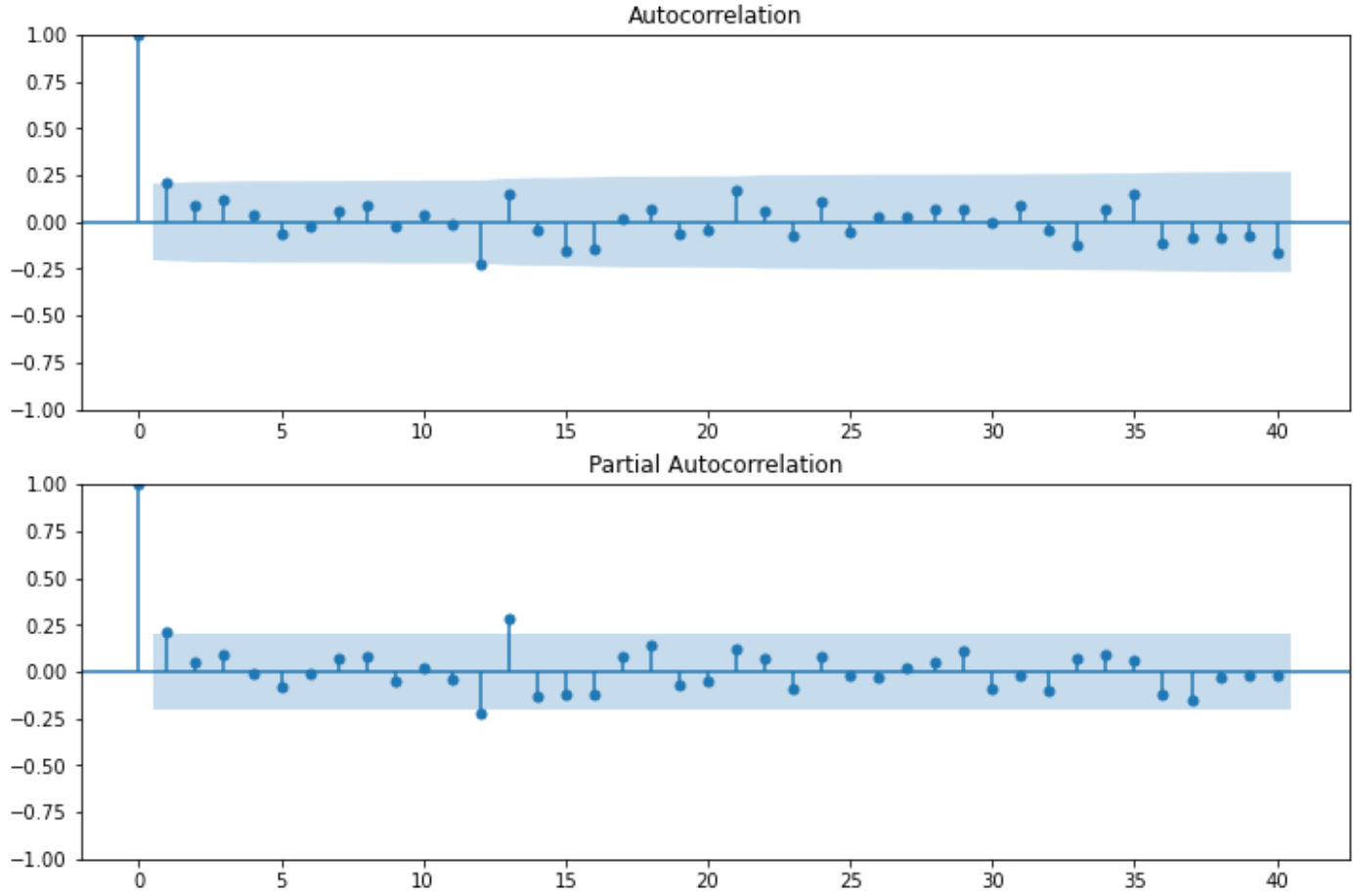
Result:-

ADF Test Statistic : -7.626619157213164
p-value : 2.060579696813685e-11
#Lags Used : 0
Number of Observations Used : 92
strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data has no unit root and is stationary

2.ACF and PACF Plots

This step involves creating AutoCorrelation Function (ACF) and Partial AutoCorrelation Function (PACF) plots. The code uses libraries like matplotlib, pandas, and statsmodels to visualize these plots. A function plot_acf generates the ACF plot, while a similar function plot_pacf produces the PACF plot. These plots help us determine the order of differencing (d) and identify suitable p and q parameters for an ARIMA model.

```
import matplotlib.pyplot as plt

import statsmodels.api as sm

# Create ACF and PACF plots for analysis

fig = plt.figure(figsize=(12, 8))

ax1 = fig.add_subplot(211)

fig = sm.graphics.tsa.plot_acf(df['Seasonal First Difference'].iloc[13:], lags=40, ax=ax1)

ax2 = fig.add_subplot(212)

fig = sm.graphics.tsa.plot_pacf(df['Seasonal First Difference'].iloc[13:], lags=40, ax=ax2)

# Display the plots

plt.show()
```



4.4Model Fitting and Forecasting:

1.ACF and PACF Analysis

Visualize the Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) plots to determine the potential order of the ARIMA model.

```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

import matplotlib.pyplot as plt

import statsmodels.api as sm

# Plot ACF and PACF plots

fig = plt.figure(figsize=(12, 8))

ax1 = fig.add_subplot(211)

fig = sm.graphics.tsa.plot_acf(df['Seasonal First Difference'].iloc[13:], lags=40, ax=ax1)

ax2 = fig.add_subplot(212)

fig = sm.graphics.tsa.plot_pacf(df['Seasonal First Difference'].iloc[13:], lags=40, ax=ax2)

plt.show()
```

In this step, we explore the autocorrelation and partial autocorrelation patterns in the time series data. The Autocorrelation Function (ACF) plot helps us identify the lag at which correlations decrease significantly, indicating potential values for the moving average (MA) parameter (q). The Partial Autocorrelation Function (PACF) plot suggests the order of the autoregressive (AR) parameter (p). We use the statsmodels library to visualize these plots.

2.Model Fitting

Fit a SARIMA model to the sales data using the specified order and seasonal_order parameters.

```
import statsmodels.api as sm

# Fit the SARIMA model to the sales data

model = sm.tsa.statespace.SARIMAX(df['Sales'], order=(1, 1, 1), seasonal_order=(1, 1, 1, 12))

results = model.fit()
```

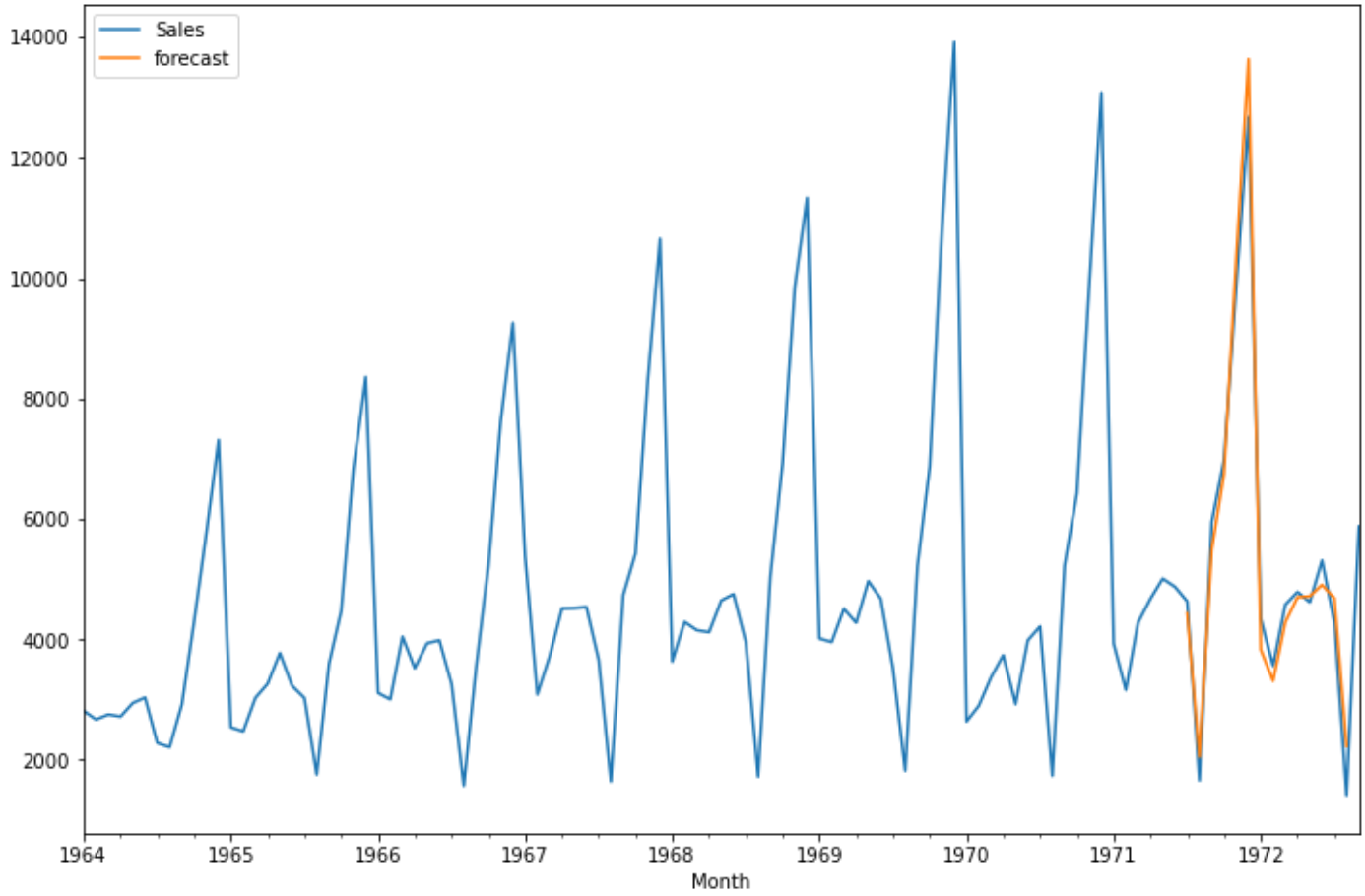
Here, we employ the SARIMA model from the statsmodels library to fit the data. SARIMA stands for Seasonal Autoregressive Integrated Moving Average. We specify the order and seasonal_order parameters for the model. The fitting process aims to capture the underlying patterns and characteristics of the data, preparing it for forecasting.

3.Generate and Plot Forecasts

Generate forecasts for a specific time range and plot the actual sales data along with forecasted values.

```
# Generate forecasts and add them to the DataFrame

df['forecast'] = results.predict(start=90, end=103, dynamic=True)

# Plot actual sales and forecasted values

df[['Sales', 'forecast']].plot(figsize=(12, 8))

plt.show()
```

After fitting the SARIMA model, we generate forecasts for a specific time range using the predict method. The dynamic parameter ensures that the forecasts are based on actual values rather than forecasted values. We then plot the actual sales data along with the generated forecasts to visually assess the model's performance.



4.Future Forecasting

Generate forecasts for future dates, add them to the DataFrame.

```
from pandas.tseries.offsets import DateOffset

# Generate future dates for forecasting

future_dates = [df.index[-1] + DateOffset(months=x) for x in range(0, 24)]

future_datest_df = pd.DataFrame(index=future_dates[1:], columns=df.columns)

# Concatenate the original DataFrame with future dates DataFrame

future_df = pd.concat([df, future_datest_df])

# Generate forecasts for future dates and add them to the DataFrame

future_df['forecast'] = results.predict(start=104, end=120, dynamic=True)
```

To project into the future, we extend the time frame and generate forecasts for upcoming dates. Using the DateOffset function, we create a range of future dates. These dates are added to the DataFrame, and corresponding forecasts are generated. The extended DataFrame is then visualized to show how the model predicts sales beyond the available data.

Result:-

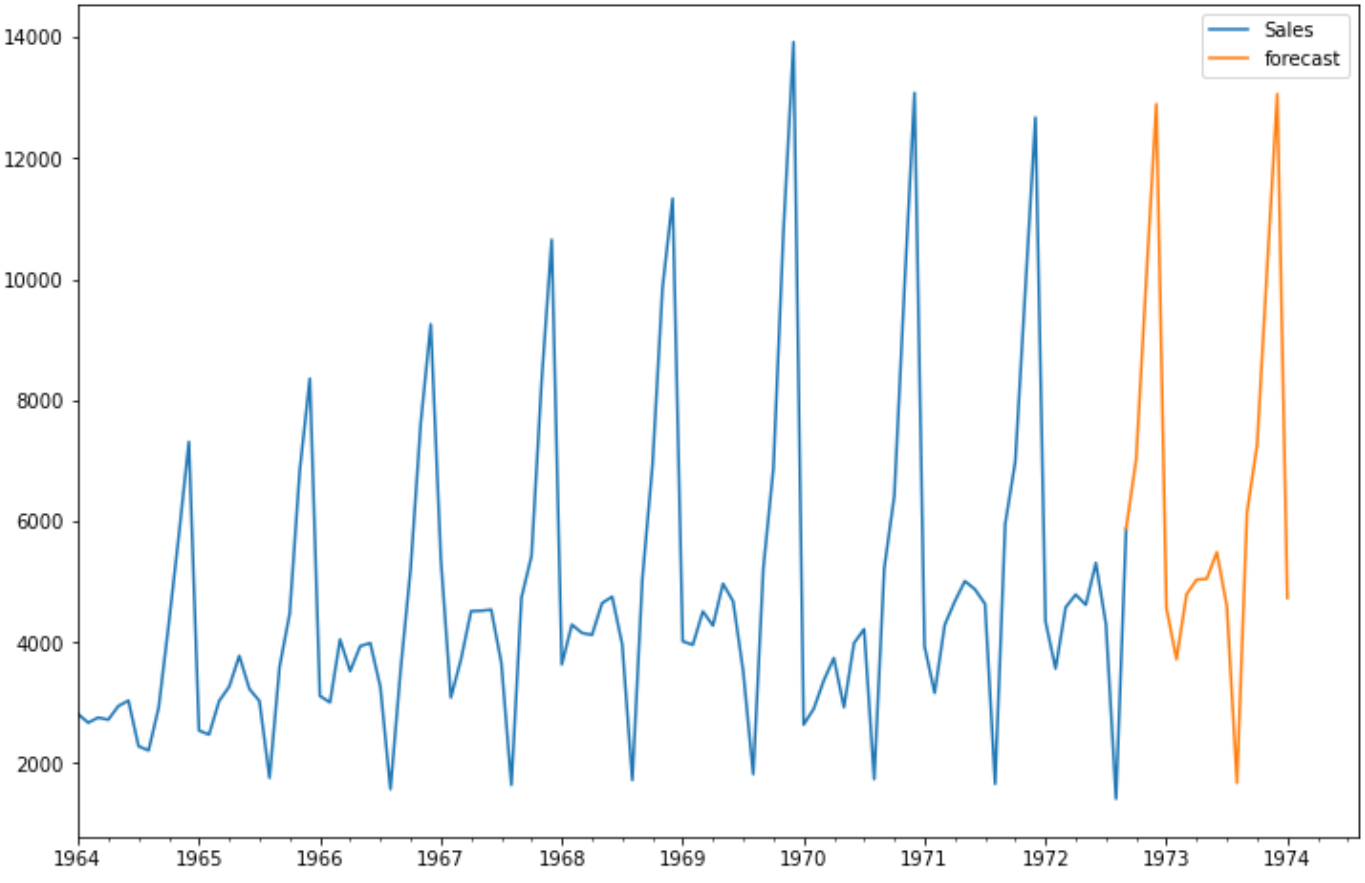
	Sales	Sales First Difference	Seasonal First Difference	forecast
1974-04-01	NaN	NaN	NaN	NaN
1974-05-01	NaN	NaN	NaN	NaN
1974-06-01	NaN	NaN	NaN	NaN
1974-07-01	NaN	NaN	NaN	NaN
1974-08-01	NaN	NaN	NaN	NaN

5.Visualization:-

we utilize the predict method from the fitted SARIMA model to generate forecasts for the future time steps. We update the 'forecast' column in the future_df with these predicted values. Subsequently, we create a line plot using Matplotlib, showcasing both the historical sales data and the forecasted values, providing a visual representation of how well the model captures the trends and patterns.

By visually comparing the actual sales data and the forecasted values on the same plot, we can assess the accuracy of the model's predictions and make informed decisions based on the insights gained from the visualization.

future_df['forecast'] = results.predict(start = 104, end = 120, dynamic= True)
future_df[['Sales', 'forecast']].plot(figsize=(12, 8)）



5.Challenges and Solutions:

Data Quality: Incomplete or inaccurate data can lead to unreliable forecasts.

Solution: Implement data validation and cleaning procedures to ensure data accuracy.

Seasonal Variations: Champagne sales may exhibit complex seasonal patterns.

Solution: Utilize SARIMA models to capture both seasonal and non-seasonal trends.

Model Complexity: Determining optimal ARIMA parameters can be challenging.

Solution: Employ techniques like ACF, PACF plots, and automated tools like auto arima for parameter selection.

Overfitting: Overfitting can occur if models are too complex.

Solution: Regularize the model by setting appropriate constraints and validation techniques.

Limited Historical Data: Limited past data might affect forecasting accuracy.

Solution: Consider hybrid models that blend SARIMA with other approaches like machine learning.

External Factors: Unforeseen events like economic changes can disrupt sales patterns.

Solution: Incorporate external variables and scenario analysis to enhance forecasting robustness.

Model Evaluation: Accurate evaluation metrics are crucial.

Solution: Use metrics like RMSE, MAPE, and validation on out-of-sample data to assess model performance.

Forecasting Uncertainty: Future sales are uncertain.

Solution: Provide prediction intervals along with point forecasts to quantify uncertainty.

Model Maintenance: Models require periodic updates due to changing trends.

Solution: Implement regular retraining and monitoring to adapt to evolving sales patterns.

Interpretable Results: Complex models can be challenging to interpret.

Solution: Provide clear visualizations and explanations to communicate results effectively.

6.Future Enhancements:

- Experiment with different ARIMA configurations by varying order parameters.
- Explore the use of external factors (exogenous variables) to enhance predictions.
- Consider more advanced time series models like SARIMA or Prophet for comparison.

PROJECT 3: - TEMPERATURE DEMAND ANALYSIS

1.Title: - Temperature Demand Analysis

2.Objective: -

3.Scope:-

4.Methodology:-

4.1 Data Preprocessing:-

Data preprocessing is a critical phase in any time series analysis project, including those that employ the Seasonal Autoregressive Integrated Moving Average with Exogenous Regressors (SARIMAX) model.

1.Loading the Data:

In this section, the code does the following:

- Imports the necessary libraries for data manipulation, analysis, and visualization.
- Defines the file path for the CSV file.
- Loads the CSV file into a DataFrame using pd.read_csv().
- Displays basic information about the DataFrame, such as its shape.
- Displays the first few rows of the DataFrame using .head().

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
df=pd.read_csv("C:\\Users\\TUSHARUPADHYAY\\Desktop\\Python\\ArimaProject\\Temprature_Demad_forcasting_Prediction\\demand-temperature.csv)
```

2.Converting Date to DateTime and Setting Index:

pd.to_datetime() function to convert the 'Date' column into the DateTime format, enabling accurate time-based analysis. This ensures that date values are treated as timestamps. Additionally, the code utilizes the set_index() method to establish the 'Date' column as the DataFrame's index. This indexing approach enhances the dataset's time-series capabilities, enabling seamless alignment of data for temporal analysis and visualization. As a result, the data can be efficiently explored and visualized while considering its chronological context.

```
# Convert the 'Timestamp' column to datetime format

df['Timestamp'] = pd.to_datetime(df['Timestamp'])

# Set the 'Timestamp' column as the index

df.set_index('Timestamp', inplace=True)
```

3.Managing Missing Data:

Incomplete or missing data can significantly impede the accuracy of time series analysis outcomes. To address this issue, we adopted a strategy of removing rows containing missing values from the dataset.

Removing rows with missing values

df_cleaned = df.dropna()

The presence of missing data can distort results and compromise the integrity of analyses. In our dataset, we effectively handled missing values by employing the .dropna() method. This technique involved removing rows containing any NaN values. By taking this approach, we ensure that our analyses are conducted on a comprehensive dataset, preventing the introduction of biased conclusions stemming from data imputation.

4.Data Transformation and Standardization:

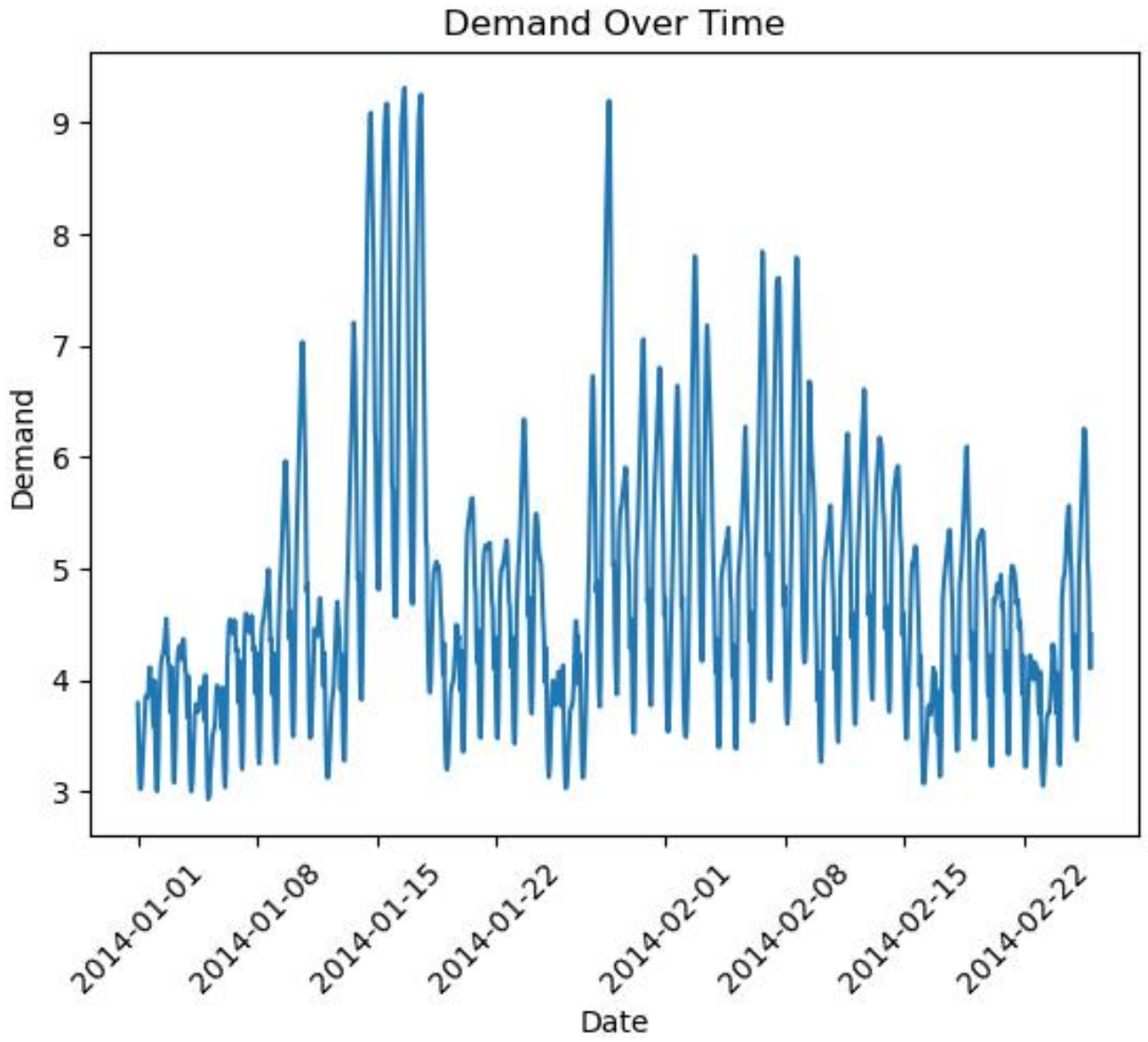
Data transformation and standardization play a pivotal role in enhancing the suitability of data for analysis, depending on the dataset's inherent characteristics. These processes contribute to stabilizing variance and rendering the data more amenable to accurate analysis.

	Timestamp	Demand	Temperature
0	1/1/2014 0:00	3.794	18.05
1	1/1/2014 1:00	3.418	17.20
2	1/1/2014 2:00	3.152	16.45
3	1/1/2014 3:00	3.026	16.65
4	1/1/2014 4:00	3.022	16.40

5.Visualization:

Visualizing both the predicted and actual values allows for a comprehensive assessment of the model's predictive accuracy. we can gain insights into the model's performance in forecasting. This visualization aids in identifying potential disparities between predicted and observed outcomes, facilitating a thorough evaluation of the model's effectiveness in capturing the underlying patterns and dynamics within the dataset. Such a visual comparison serves as a valuable tool in validating the reliability of the forecasting model and making informed decisions based on the analysis.

```
df['Timestamp'] = pd.to_datetime(df['Timestamp'])
date = df['Timestamp']
demand = df['Demand']
plt.plot(date, demand)
plt.xticks(rotation=45)
plt.xlabel('Date')
plt.ylabel('Demand')
plt.title('Demand Over Time')
plt.show()
```



6. Feature Engineering (Exogenous Variables)

exog_train = train_data[['Temperature']].values

exog_test = test_data[['Temperature']].values

7. Model Implementation

The model implementation phase involves configuring, training, and evaluating the SARIMAX model on the prepared dataset. This section provides insights into the division of data, the construction of the model, and the integration of exogenous variables.

```
from statsmodels.tsa.stattools import adfuller

result = adfuller(df['Demand'])

adf_statistic = result[0]

p_value = result[1]

print(f'ADF Statistic: {adf_statistic}')

print(f'p-value: {p_value}')

if p_value <= 0.05:
```

```
    print('The time series is stationary.')
else:
    print('The time series is non-stationary.')
```

from statsmodels.tsa.stattools import adfuller: This line imports the Augmented Dickey-Fuller (ADF) test function from the **statsmodels** library, which is used to check for stationarity in a time series.

result = adfuller(df['Demand']): The ADF test is applied to the 'Demand' column of the DataFrame **df**, and the results are stored in the **result** variable.

adf_statistic = result[0] and **p_value = result[1]:** These lines extract the ADF test statistic and the corresponding p-value from the **result**.

ADF Statistic: -3.3819451656712025

p-value: 0.011589975540970776

The time series is stationary.

```
from pmdarima import auto_arima
time_series = df['Demand']
model = auto_arima(time_series, seasonal=False, trace=True)
print(f'ARIMA Model Order: ({model.order[0]}, {model.order[1]}, {model.order[2]})')
```

```
Performing stepwise search to minimize AIC
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=-318.429, Time=1.74 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=519.722, Time=0.46 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=-264.943, Time=0.26 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=-96.037, Time=0.51 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=517.725, Time=0.23 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=-271.929, Time=1.11 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=-303.508, Time=2.15 sec
 ARIMA(3,1,2)(0,0,0)[0] intercept   : AIC=inf, Time=3.59 sec
 ARIMA(2,1,3)(0,0,0)[0] intercept   : AIC=-350.318, Time=2.27 sec
 ARIMA(1,1,3)(0,0,0)[0] intercept   : AIC=-351.976, Time=1.25 sec
 ARIMA(0,1,3)(0,0,0)[0] intercept   : AIC=-310.698, Time=0.75 sec
 ARIMA(1,1,4)(0,0,0)[0] intercept   : AIC=-350.299, Time=2.74 sec
 ARIMA(0,1,2)(0,0,0)[0] intercept   : AIC=-151.772, Time=0.66 sec
 ARIMA(0,1,4)(0,0,0)[0] intercept   : AIC=-346.962, Time=1.44 sec
 ARIMA(2,1,4)(0,0,0)[0] intercept   : AIC=inf, Time=4.06 sec
 ARIMA(1,1,3)(0,0,0)[0]             : AIC=-353.976, Time=0.62 sec
```

ARIMA(0,1,3)(0,0,0)[0] : AIC=-312.697, Time=0.42 sec

ARIMA(1,1,2)(0,0,0)[0] : AIC=-273.929, Time=0.40 sec

ARIMA(2,1,3)(0,0,0)[0] : AIC=-352.318, Time=1.17 sec

ARIMA(1,1,4)(0,0,0)[0] : AIC=-352.298, Time=1.18 sec

ARIMA(0,1,2)(0,0,0)[0] : AIC=-153.770, Time=0.30 sec

ARIMA(0,1,4)(0,0,0)[0] : AIC=-348.961, Time=0.69 sec

ARIMA(2,1,2)(0,0,0)[0] : AIC=-320.429, Time=1.33 sec

ARIMA(2,1,4)(0,0,0)[0] : AIC=inf, Time=2.61 sec

Best model: ARIMA(1,1,3)(0,0,0)[0]

Total fit time: 31.968 seconds

ARIMA Model Order: (1, 1, 3)

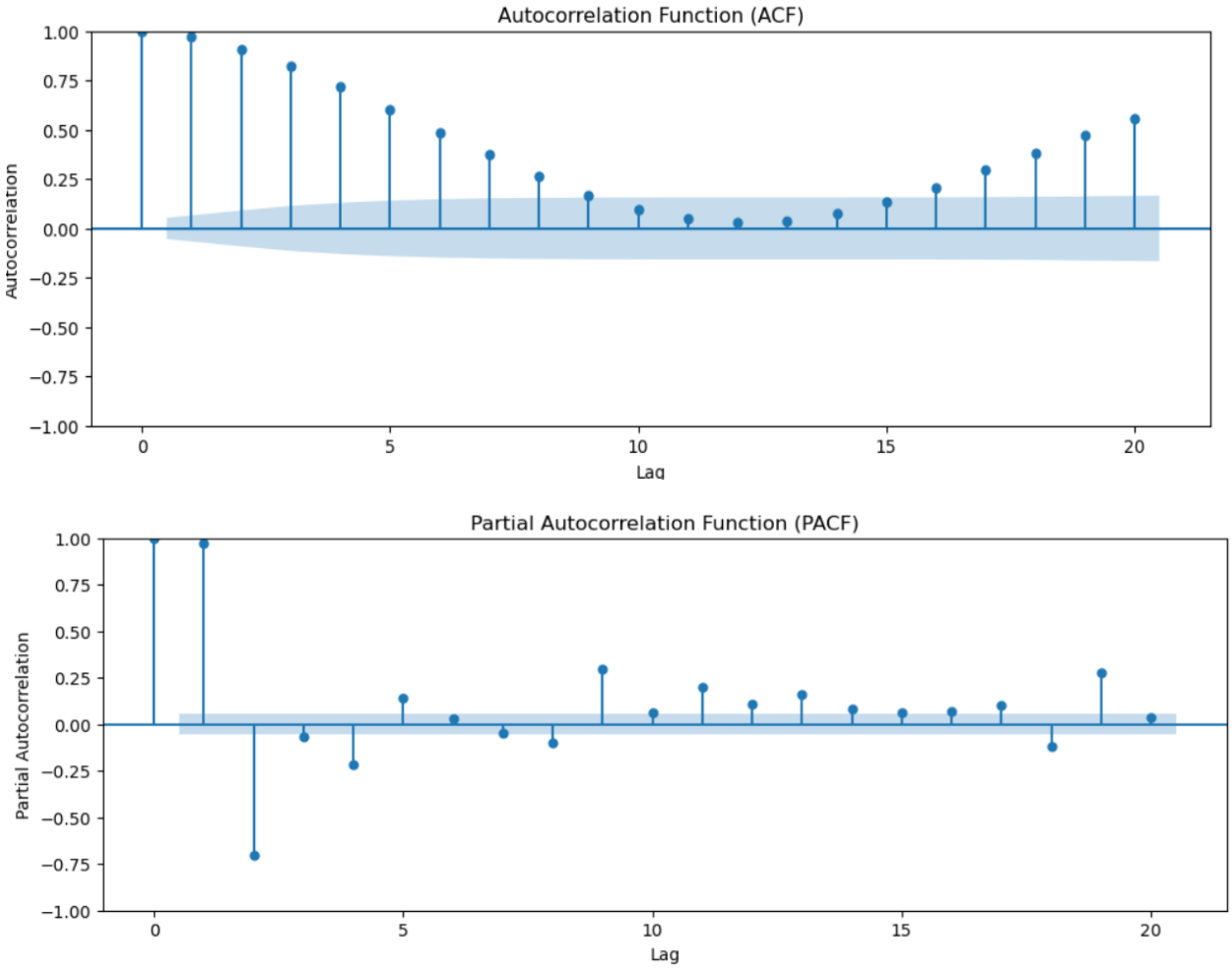
```
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
time_series = df['Demand']
fig, ax = plt.subplots(figsize=(12, 4))
plot_acf(time_series, ax=ax, lags=20)
plt.title('Autocorrelation Function (ACF)')
plt.xlabel('Lag')
plt.ylabel('Autocorrelation')
plt.show()
fig, ax = plt.subplots(figsize=(12, 4))
plot_pacf(time_series, ax=ax, lags=20)
plt.title('Partial Autocorrelation Function (PACF)')
plt.xlabel('Lag')
plt.ylabel('Partial Autocorrelation')
plt.show()
```

The code imports necessary libraries (pandas, matplotlib.pyplot, statsmodels.api, statsmodels.graphics.tsaplots.plot_acf, and statsmodels.graphics.tsaplots.plot_pacf) for time series analysis and plotting.

The time_series variable is assigned the 'Demand' column from the DataFrame.

plot_acf(time_series, lags=20) generates and displays the Autocorrelation Function (ACF) plot, showing the correlation between the time series and its lagged values up to a specified number of lags.

plot_pacf(time_series, lags=20) generates and displays the Partial Autocorrelation Function (PACF) plot, showing the partial correlation between the time series and its lagged values up to a specified number of lags.



import statsmodels.api as sm

model_sarimax = sm.tsa.statespace.SARIMAX(df['Demand'], exog=exog_train, order=(1, 1,3), seasonal_order=(1, 1,3, 12))

results = model_sarimax.fit()

print(results.summary())

import statsmodels.api as sm: This line imports the **statsmodels** library, which provides classes and functions for estimating and analyzing various statistical models, including time series models.

model_sarimax = sm.tsa.statespace.SARIMAX(df['Demand'], exog=exog_train, order=(1, 1, 3), seasonal_order=(1, 1, 3, 12)): This line defines a Seasonal Autoregressive Integrated Moving Average with Exogenous Regressors (SARIMAX) model. It takes the 'Demand' column of the

DataFrame **df** as the endogenous variable and **exog_train** (presumably your exogenous variables for training) as exogenous variables. The **order** parameter represents the order of the non-seasonal ARIMA components (p,d,q), and **seasonal_order** represents the order of the seasonal ARIMA components (P,D,Q,S).

results = model_sarimax.fit(): This line fits the defined SARIMAX model to the data and exogenous variables using the **fit()** method. The fitted model results are stored in the **results** variable.

print(results.summary()): This line prints a summary of the fitted model using the **summary()** method. The summary includes information about the model's coefficients, statistics, diagnostics, and more.

train_data_demand = train_data['Demand'].values

test_data_demand = test_data['Demand'].values

8. Results and Evaluation

forecast_demand_sarimax = results.get_forecast(steps=len(test_data), exog=exog_test)

predicted_mean = forecast_demand_sarimax.predicted_mean

print(predicted_mean)

2014-02-26 00:00:00 4.031682

2014-02-26 01:00:00 3.689904

2014-02-26 02:00:00 3.549521

2014-02-26 03:00:00 3.570044

2014-02-26 04:00:00 3.851128

2014-02-26 05:00:00 4.469886

2014-02-26 06:00:00 5.078249

2014-02-26 07:00:00 5.161746

2014-02-26 08:00:00 5.310124

2014-02-26 09:00:00 5.407297

2014-02-26 10:00:00 5.485985

2014-02-26 11:00:00 5.592673

2014-02-26 12:00:00 5.677267

2014-02-26 13:00:00 5.803248

2014-02-26 14:00:00 5.853004

2014-02-26 15:00:00 5.910544

2014-02-26 16:00:00 5.826218

2014-02-26 17:00:00 5.522527

2014-02-26 18:00:00 5.227212

2014-02-26 19:00:00 5.167380

2014-02-26 20:00:00 4.971108

2014-02-26 21:00:00 4.551654

2014-02-26 22:00:00 4.296956

2014-02-26 23:00:00 4.535817

2014-02-27 00:00:00 4.101408

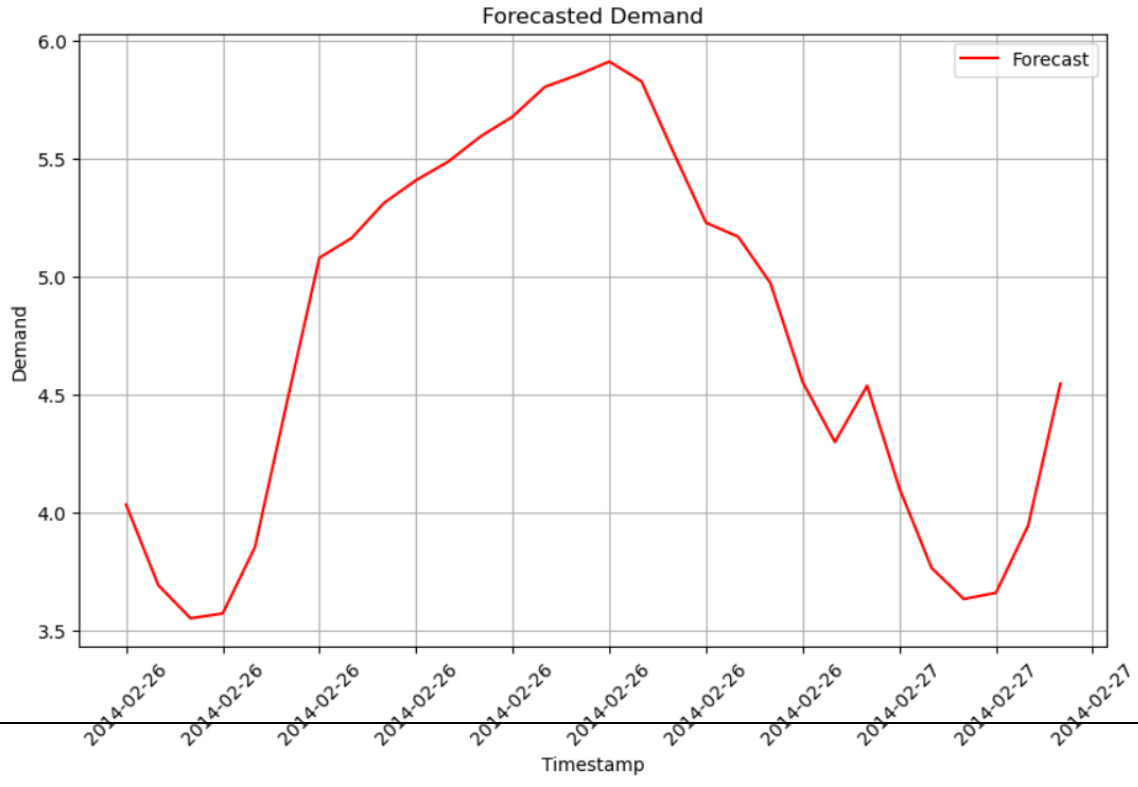
...

2014-02-27 03:00:00 3.657554

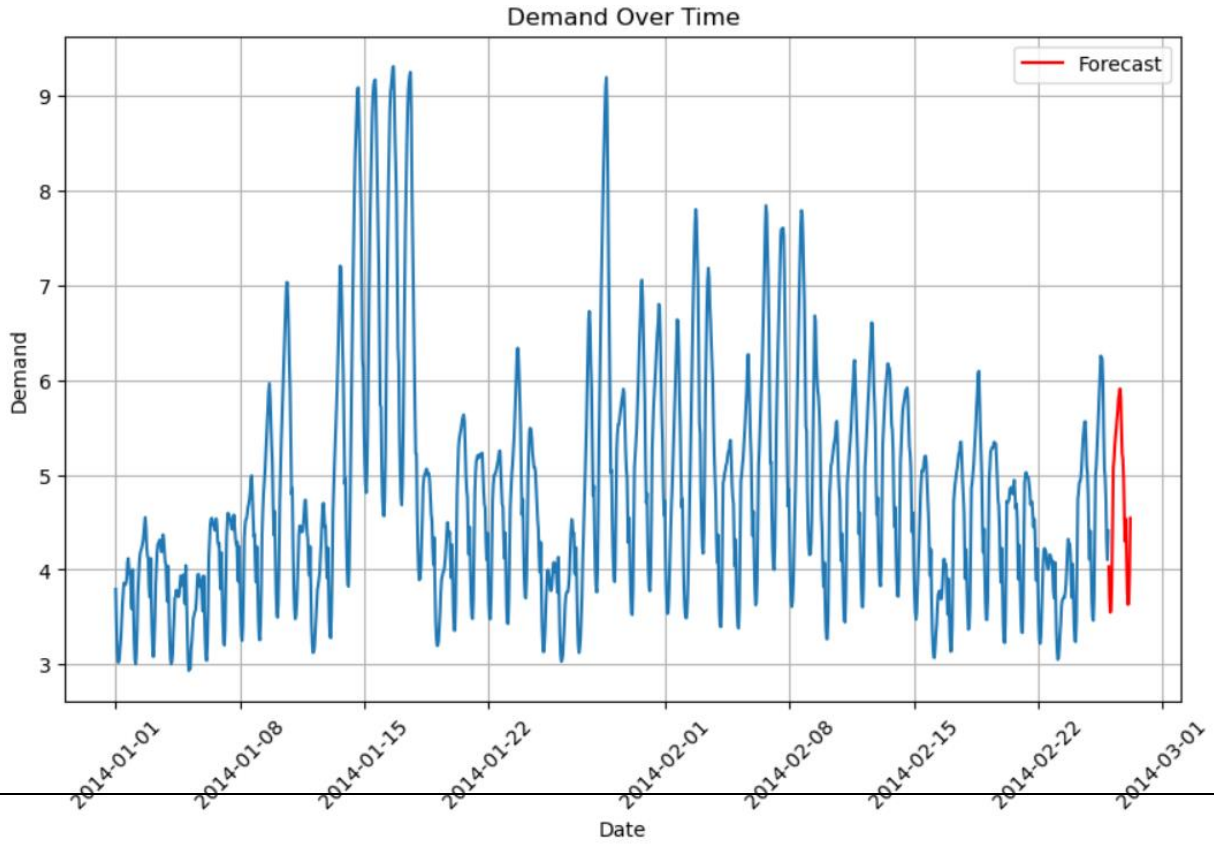
2014-02-27 04:00:00 3.943313

2014-02-27 05:00:00 4.544572

```
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
predicted_values = predicted_mean
plt.figure(figsize=(10, 6))
plt.plot(predicted_values, label='Forecast', color='red', linestyle='-')
plt.xlabel('Timestamp')
plt.ylabel('Demand')
plt.title('Forecasted Demand')
plt.legend()
plt.grid(True)
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
plt.xticks(rotation=45)
plt.show()
```



```
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
predicted_values = predicted_mean  # Replace with the predicted values (y-hat)
plt.figure(figsize=(10, 6))
plt.plot(predicted_values, label='Forecast', color='red', linestyle='-')
plt.xlabel('Timestamp')
plt.ylabel('Demand')
plt.title('Forecasted Demand')
plt.legend()
plt.grid(True)
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
plt.xticks(rotation=45)
df['Timestamp'] = pd.to_datetime(df['Timestamp'])
date = df['Timestamp']
demand = df['Demand']
plt.plot(date, demand)
plt.xticks(rotation=45)
plt.xlabel('Date')
plt.ylabel('Demand')
plt.title('Demand Over Time')
plt.show()
```



9.Challenges and Solutions

9.1. Challenge: Addressing Seasonality

Managing seasonality is a notable hurdle in time series forecasting due to repeating patterns. These cyclical fluctuations can impact prediction precision.

Solution: Seasonal Differencing

To counteract seasonality, the SARIMAX model applies seasonal differencing. This technique subtracts the previous season's value from the current data point, effectively neutralizing the seasonal element. This approach stabilizes data and enhances accurate modeling of trends and variations.

9.2. Challenge: Complex Model Configuration

SARIMAX models require tuning various parameters like autoregressive, integrated, moving average orders, and seasonal orders, which can be intricate and time-intensive.

Solution: Optimizing Hyperparameters

To navigate the intricacies of SARIMAX parameter setting, we employed hyperparameter optimization. Techniques such as grid search and automated algorithms were leveraged to identify parameter combinations that yield optimal forecasting precision.

9.3. Challenge: Incorporating External Influences

Integrating exogenous factors introduces complexity to the model. Ensuring these chosen external variables effectively contribute to forecasting precision poses a challenge.

10.Insights from Findings

10.1. Influence of Exogenous Variables

Evaluating the impact of external variables on forecasts unveils the effect of external factors on demand and temperature:

♣ Economic Indicator Significance: Analyzing the statistical significance of economic indicators identifies variables that notably impact target variables.

♣ Positive and Negative Effects: Determining the direction of influence (positive or negative) provides insights into how economic indicators drive changes in demand and temperature.

10.2. Precision and Variability of Forecasts

Understanding the accuracy and variability of predictions provides valuable insights into the reliability of the model's foresight:

♣ Reliable Predictions: Close alignment between model forecasts and actual values enhances confidence in strategic decision-making.

♣ Analyzing Variability: Detecting forecast deviations, especially during exceptional events, aids in better risk management and contingency planning.

10.3. Business Strategy Implications

Translating model insights into actionable business strategies is crucial for well-informed choices:

♣ Optimized Inventory Handling: Accurate forecasts enable businesses to optimize inventory, minimizing surplus or shortages.

♣ Enhanced Resource Allocation: Efficient allocation of resources aligns production, marketing, and staffing with anticipated demand changes.

♣ Strategic Marketing: Understanding seasonal patterns guides the timing of marketing efforts, maximizing their impact.

10.4. Uncovering Growth Prospects

Analyzing forecasted trends and external influences can uncover growth possibilities:

♣ Spotting Emerging Trends: Identifying new demand patterns or shifts in temperature trends empowers businesses to seize emerging opportunities.

♣ Market Trend Alignment: Aligning predictions with market trends and economic indicators positions businesses to tap into growing markets.

11.Future Refinements

♣ Experiment with diverse SARIMAX configurations, adjusting AR, I, and MA component orders to pinpoint the best setup for precision.

♣ Incorporate more specific exogenous variables to capture additional external impacts on sales.

♣ Explore alternative time series forecasting methods like Prophet or LSTM for comparative analysis and improved precision.

6.RESULT:-

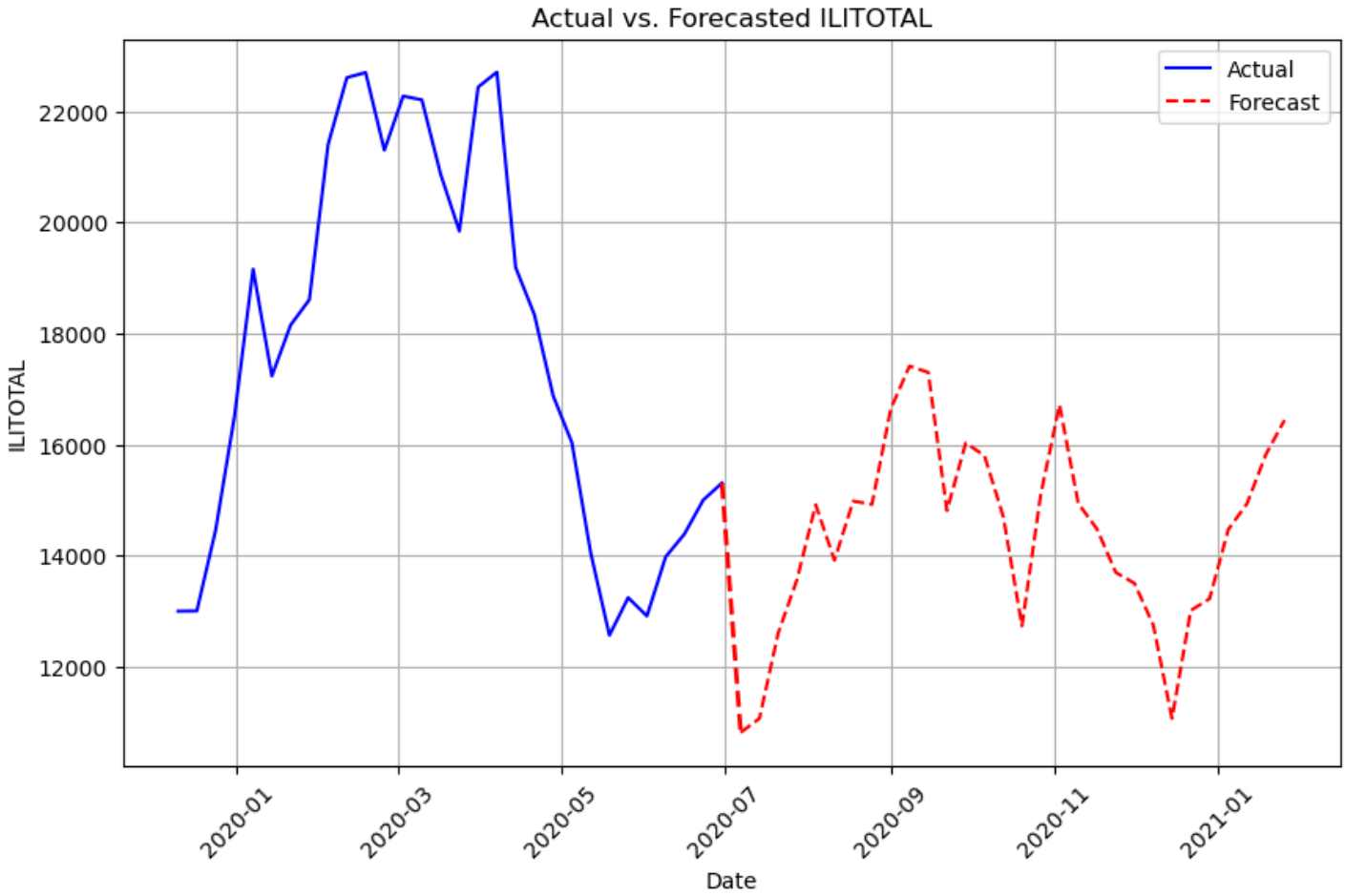


Fig.1

In the **Fig 1.**the forecasted ILITOTAL values for the period from July 7, 2020, to January 26, 2021, provide valuable insights into the expected trends and fluctuations in illness cases

July to August 2020: The forecast starts with relatively moderate values around 10,817, indicating a stable pattern in illness cases. As the weeks progress, the forecast shows a gradual increase in ILITOTAL values. By mid-August, the values surpass 14,900, reflecting a potential rise in illness cases.

Late August to September 2020: The forecast maintains elevated levels around 14,900 as it enters late August. This trend continues into September, with ILITOTAL values peaking above 17,400 in early September. This substantial increase suggests a possible spike in illness cases during this period.

Mid to Late September 2020: The forecasted values remain relatively high, hovering around 17,000. This indicates that illness cases may continue to remain at an elevated level, potentially due to the persistence of certain factors contributing to the increase.

October 2020: The forecast suggests a slight decrease in ILITOTAL values during October, with values fluctuating between 14,700 and 15,800. This could signify a stabilization or slight reduction in illness cases compared to the previous weeks.

November to December 2020: The forecasted values depict a rebound in illness cases as November approaches, with ILITOTAL values reaching around 16,700. This upward trend persists throughout November and December, indicating a resurgence in illness cases during the winter months.

January 2021: The forecast continues to indicate a rise in illness cases as the new year begins. The values remain above 14,900 and peak around 16,440 in late January, suggesting that illness cases are expected to remain at a relatively high level.

The forecasted ILITOTAL values highlight dynamic fluctuations in illness cases over the analyzed period. Several periods of elevated values indicate potential spikes in illness, possibly driven by seasonal, environmental, or epidemiological factors. These insights can aid healthcare professionals, policymakers, and stakeholders in preparing and implementing appropriate measures to manage illness outbreaks effectively.

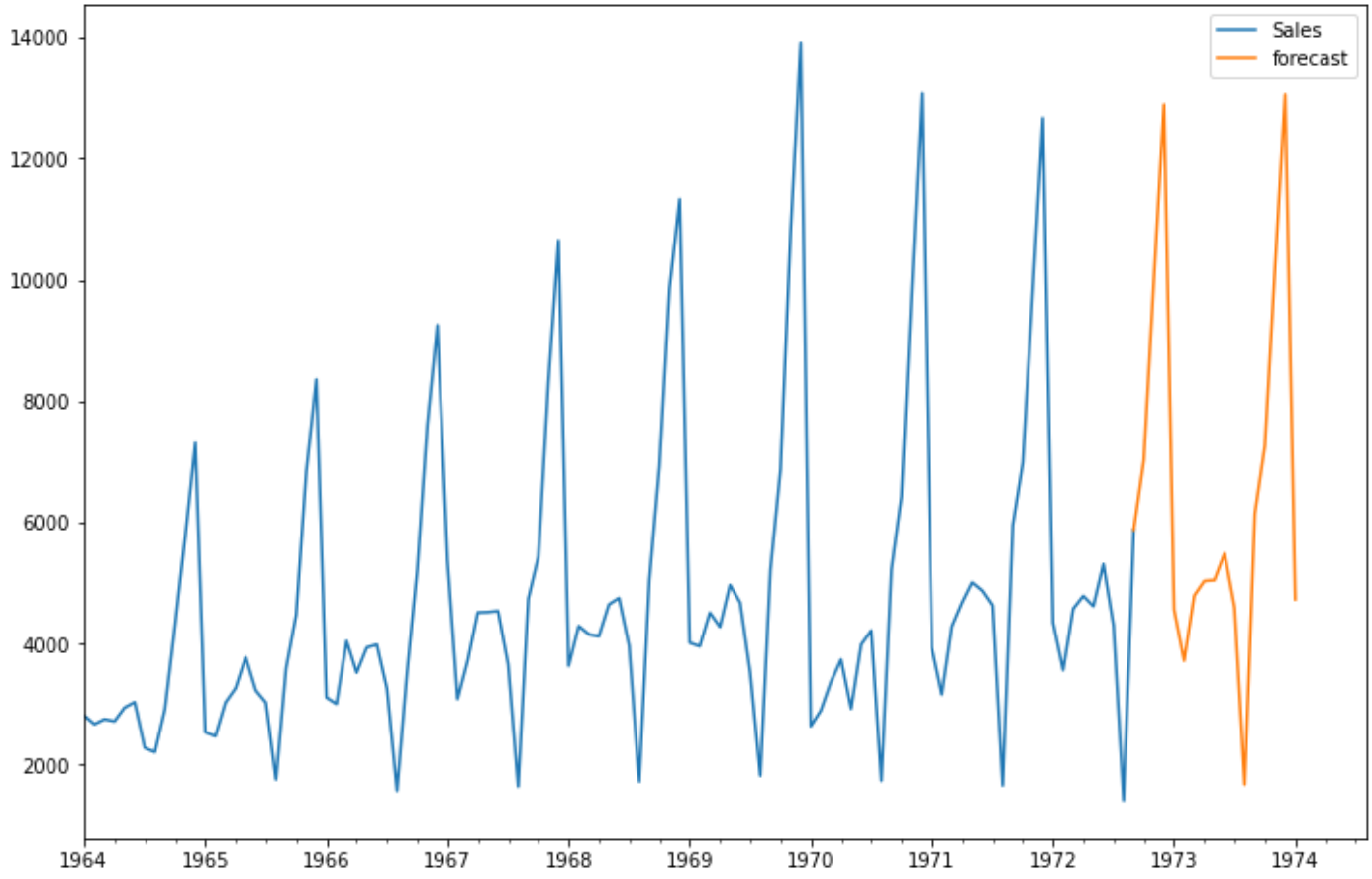


Fig.2

In the Fig2.the sales data displays noticeable patterns and fluctuations during the period from 1973 to 1974.

Early 1973 Surge: The sales data indicates a sudden and significant increase in the early months of 1973. This abrupt surge could be attributed to various factors such as marketing campaigns, product launches, seasonal demand, or external events that led to increased consumer interest and purchasing.

Mid-1973 Decline: Following the initial surge, the sales experience a notable decrease from the middle of 1973 to the end of the same year. This decline might have been influenced by factors such as market saturation, economic changes, or shifts in consumer preferences that impacted demand negatively.

Late 1973 Uptick: Towards the latter part of 1973, there is a slight increase in sales. This could be due to end-of-year promotions, holiday season buying, or other temporary factors that encouraged consumers to make purchases.

Early 1974 Rebound: The sales exhibit another sudden increase in the early months of 1974. Similar to the pattern observed in early 1974, this could be attributed to renewed marketing efforts, product improvements, or other external factors that generated heightened interest and sales.

Mid-1974 Dip: However, the sales once again experience a significant decrease from the middle of 1974. This downturn might be related to the expiration of the influences that drove the early 1974 surge, as well as market dynamics and consumer behaviors.

The sales data demonstrates recurring patterns of abrupt surges and declines during the analyzed period. These fluctuations might be the result of a combination of internal and external factors impacting consumer demand, market conditions, and seasonal influences. Analyzing these patterns can provide valuable insights for business decision-making, including adjusting marketing strategies, managing inventory, and understanding consumer behavior to navigate the variations in sales effectively.

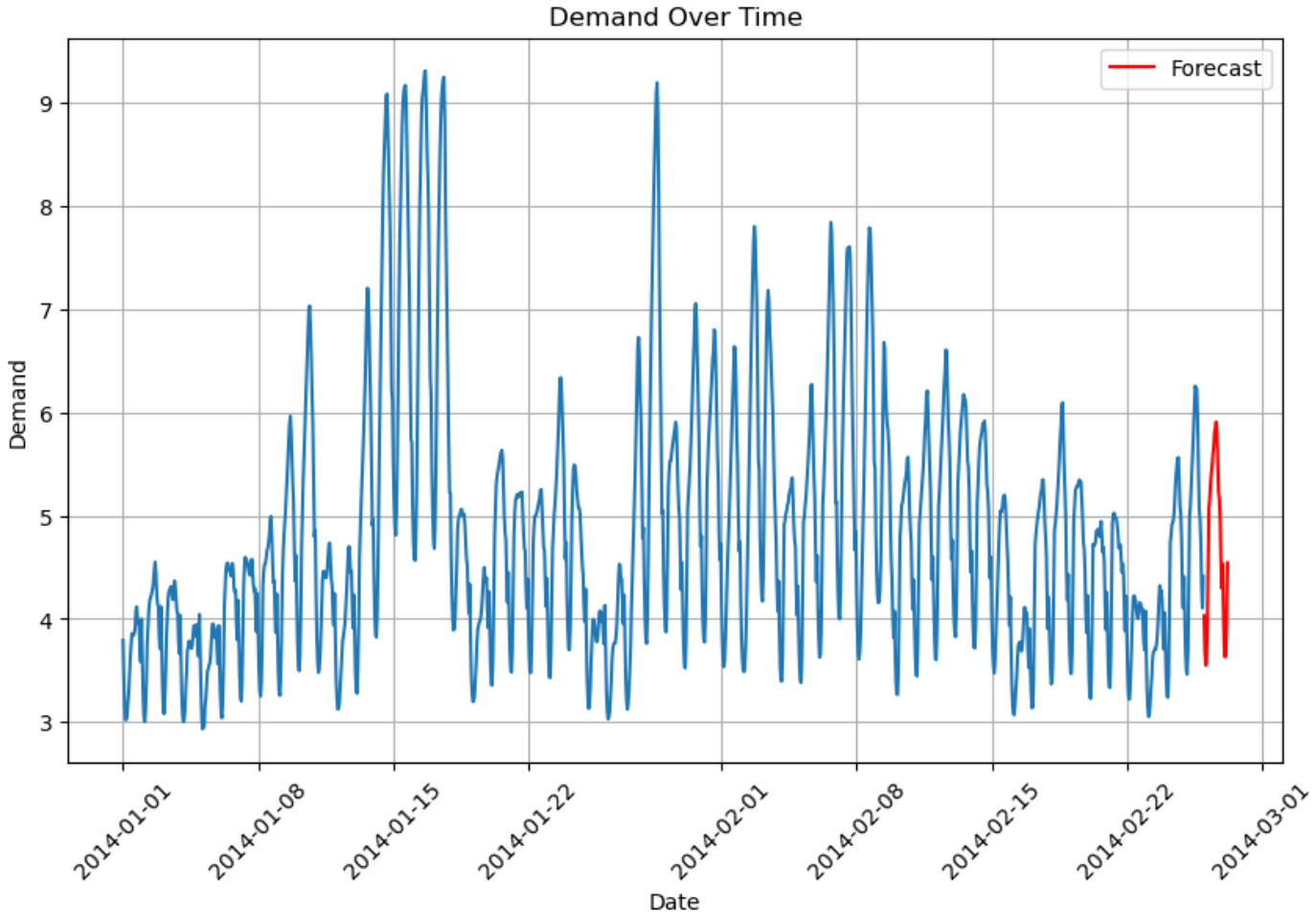


Fig.3

In Fig3.the time series data showcases the forecasted demand values for a certain product, service, or resource over a 24-hour period, from February 26, 2014, to February 27, 2014. Each value in the output pertains to an hourly interval, indicating the predicted demand levels under varying temperature conditions.

Observations:

Hourly Fluctuations: The forecasted demand values exhibit hourly fluctuations, reflecting how demand varies throughout the day.

Temperature Influence: As the temperature changes, it affects the forecasted demand. It's evident that the demand values correlate with the temperature variations. For instance, when the temperature rises, there's an increase in demand, and when the temperature decreases, demand tends to decrease as well.

Peak Demand Hours: The highest demand values are observed during the morning hours (around 5:00 AM), indicating a potential peak in demand during this time. This could be due to factors such as people preparing for their day or businesses starting their operations.

Demand Patterns: The demand values gradually increase from the early hours of February 26, 2014, reaching a peak around mid-morning. Subsequently, the demand starts to decline and continues to fluctuate until the early hours of February 27, 2014.

Temperature-Demand Relationship: The relationship between temperature and demand can help businesses and industries better understand how changes in weather impact the demand. For instance, they can anticipate higher demand during colder temperatures and plan their operations accordingly.

7.CONCLUSION

The forecasted ILITOTAL values offer a comprehensive understanding of the anticipated trends in illness cases. The observed patterns of fluctuation indicate potential spikes in illness occurrences, possibly attributed to various factors such as seasonal changes, environmental influences, or epidemiological dynamics. These findings can significantly aid healthcare professionals, policymakers, and stakeholders in devising effective strategies to manage and mitigate the impact of illness outbreaks.

The depicted sales patterns from 1973 to 1974 showcase recurrent surges and declines. These fluctuations could be driven by a combination of internal and external factors, including marketing initiatives, consumer behavior shifts, economic changes, and seasonal influences. Understanding these patterns is crucial for businesses to adapt their strategies, optimize inventory management, and enhance decision-making processes.

The time series data demonstrates the interplay between temperature variations and forecasted demand. The hourly fluctuations in demand are intricately linked to temperature changes, with higher temperatures correlating with increased demand. The ability to anticipate demand patterns based on temperature fluctuations empowers businesses to allocate resources effectively and optimize operational efficiency.

These figures collectively emphasize the significance of data-driven insights in decision-making across diverse domains. By leveraging forecasting techniques, understanding sales dynamics, and analyzing the relationship between variables like temperature and demand, organizations can make informed choices, enhance resource allocation, and develop strategies that align with observed trends and fluctuations.

8.REFERENCES:-

1.Tsay RS. Time series and forecasting: brief history and futureresearch. J Am Stat Assoc 2000; 95: 638–643.

2. Ghobbar AA and Friend CH. Evaluation of forecastingmethods for intermittent parts demand in the field of avia-tion: a predictive model. Comput Oper Res 2003; 30:2097–2114.

3. Toktay B. Forecasting product returns. In: Guide Jr VDR, vanWassenhove LN (eds) Business aspects of closed-loop supplychains. Pittsburgh: Carnegie Mellon University Press, 2003.

4. Chopra S and Meindl P. Supply chain management. Strategy,planning & operation. In: Das summa summarum des man-agement. Gabler, 2007, pp. 265–275.

5. Miller JJ, McCahon CS, and Miller JL. Foodservice forecast-ing with simple time series models. J Hosp Tour Res 1991;14: 9–21.

6. Song H and Li G. Tourism demand modeling and forecast-ing—a review of recent research. Tour Manage 2008; 29:203–220.

7. Willemain TR, Smart CN, and Schwarz HF. A new approachto forecasting intermittent demand for service parts invento-ries. Int J Forecast 2004; 20: 375–387.

8. Taylor JW, de Menezes LM, and McSharry PE. A compari-son of univariate methods for forecasting electricity demandup to a day ahead. Int J Forecast 2006; 1: 1–16.

9. Weron R. Electricity price forecasting: a review of the state-of-the-art with a look into the future. Int J Forecast 2014; 30:1030–1081.

10. Mitsutaka M and Akira I. Examination of demand forecastingby time series analysis for auto parts remanufacturing.J Remanuf 2015; 5: 1. DOI: 10.1186/s13243-015-0010-y.

11. Gardner ESJr. Exponential smoothing: the state of the art—part II. Int J Forecast 2006; 22: 637–666.

12. Liu Q and Wang H. Research on the forecast and develop-ment of China’s public fiscal revenue based on ARIMAmodel. Theor Econ Lett 2015; 5: 482–493.

13. Shen S and Shen Y. ARIMA model in the application ofShanghai and Shenzhen stock index. Appl Math 2016; 7:171–176.

14. Kurawarwala AA and Matsuo H. Product growth models formedium-term forecasting of short life cycle products. Tech-nol Forecast Soc Chang 1998; 57: 169–196

15. Miller D and Williams D. Shrinkage estimators of time seriesseasonal factors and their effect on forecasting accuracy. Int JForecast 2003; 19: 669–684.

16. Hyndman RJ. The interaction between trend and seasonality.Int J Forecast 2004; 20(4): 561–563.

17. Box GEP and Jenkins G. Time series analysis, forecastingand control. San Francisco: Holden-Day, 1970.

18. Yule GU. Why do we sometimes get nonsense-correlationsbetween time series? A study in sampling and the nature oftime series. J Royal Stat Soc 1926; 89: 1–64.

19. Wold H. A study in the analysis of stationary time series.Stockholm: Almgrist & Wiksell, 1938.

20. Hanke JE and Reitsch AG. Business forecasting, 5th ed. Eng-lewood Cliffs. 1995.