INSTITUTE FOR ADVANCED COMPUTING AND SOFTWARE DEVELOPMENT (IACSD), AKURDI,PUNE

Large-Scale Time Series Forecasting For Enhanced Sales Insights

PG-DBDA September 2023

Submitted By:

Group No: 03

Roll No.	Name :
239541	Shriyash Kailas Kamble
239524	Macchindranath R. Dabhade

Mrs. Priyanka Bhor
Project Guide

Mr. Rohit Puranik
Centre Co-ordinator

Abstract

In this study, we have conducted a comprehensive analysis of four prominent time series forecasting models: FB Prophet, ARIMA, ETS (Error-Trend-Seasonality), and the hybrid ETS-ARIMA model. Leveraging historical data from [briefly describe the dataset], our research aimed to assess the performance of these models in predicting future trends.

Our findings reveal distinct advantages among the models: FB Prophet excels in handling irregular data with holidays and outliers; ARIMA demonstrates robustness in capturing linear trends; ETS is effective in modelling data with clear seasonality patterns, and the hybrid ETS-ARIMA approach combines the strengths of both methods for enhanced forecasting accuracy.

These insights offer valuable guidance for selecting the most appropriate forecasting model depending on the specific data characteristics and forecasting requirements, spanning applications from finance to inventory management.

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to everyone who has contributed to the completion of our project.

First and foremost, we would like to thank our project guide **Mrs. Priyanka Bhor** mam for their constant guidance and support throughout the project. We extend our sincere thanks to our respected Centre Co-Ordinator, **Mr. Rohit Puranik**, for allowing us to use the facilities available.

We would also like to express our appreciation to the faculty members of our department for their constructive feedback and encouragement. Their insights and suggestions have helped us to refine our ideas and enhance the quality of our work.

Furthermore, we would like to thank our families and friends for their unwavering support and encouragement throughout our academic journey. Their love and support have been a constant source of motivation and inspiration for us.

Thank you all for your valuable contributions to our project

Shriyash Kailas Kamble (239541)

Macchindranath R. Dabhade (239524)

Table of contents

List of Figures

1. Introduction

1.1 Problem Statement

In today's dynamic business landscape, accurate sales forecasting is essential for effective inventory management, resource allocation, and informed decision-making. This project aims to leverage time series analysis techniques to forecast the quantity of products sold within the particular category based on historical sales data. By analyzing patterns, trends, and seasonality in the data, we seek to develop reliable predictive models that will empower businesses to optimize their operations and anticipate demand fluctuations.

1.2 Product Scope

The product scope of this project entails evaluating four prominent time series forecasting models, namely FB Prophet, ARIMA, ETS (Error-Trend-Seasonality), and the hybrid ETS-ARIMA model, to determine their suitability for predicting future trends in a specific dataset. We will collect and pre-process the relevant data, employ a set of predefined metrics to assess model performance, and partition the data for rigorous testing and validation. The project's primary objective is to identify the most effective forecasting model based on the chosen criteria. Key deliverables will include a comprehensive report summarizing findings, visualizations contrasting forecasted and actual values, and code documentation. While acknowledging potential risks and assumptions in the analysis, we aim to provide valuable insights for stakeholders and future work in the realm of time series forecasting.

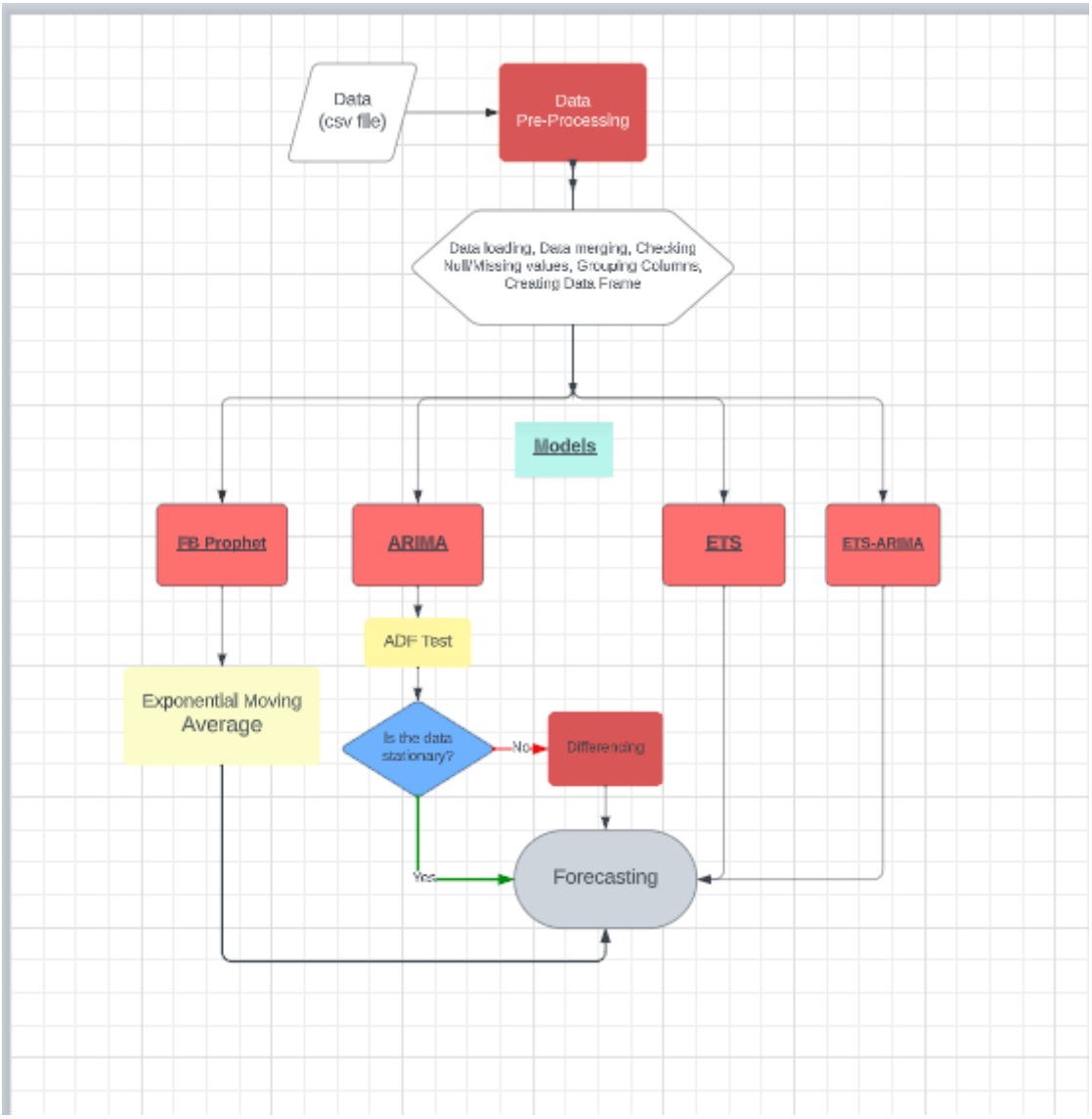
1.3 Aims and Objectives

The aims and objectives of this project are twofold: firstly, to systematically assess and compare the predictive performance of four distinct time series forecasting models—FB Prophet, ARIMA, ETS, and ETS-ARIMA—across a range of temporal datasets, with a focus on determining their strengths and limitations in handling different types of time series data. Secondly, we aim to provide practical guidance for selecting the most appropriate forecasting model depending on specific data characteristics, thus aiding decision-makers in diverse domains, from finance to supply chain management, to make informed choices for improved future predictions. Through rigorous evaluation and transparent reporting, this project seeks to enhance the understanding and application of time series forecasting methodologies.

2. Overall Description

2.1 Workflow of Project

The diagram below shows the workflow of this project



2.2 Data pre-processing and Cleaning

Data pre-processing and cleaning are critical steps in the data analysis and machine learning pipeline. High-quality data is essential for accurate analysis and modelling. In this report, we detail the data pre-processing and cleaning steps performed on two datasets: 'articles.csv' and 'transactions_train.csv'. The goal is to prepare the data for further analysis, including time series forecasting and visualization.

Data Loading

Two datasets were used in this project: 'articles.csv' and 'transactions_train.csv'. These datasets were loaded into two
Pandas Data Frames, 'df_1' and 'df_2', respectively.

```
df_1 = pd.read_csv(r"d:\project\articles.csv")
df_2 = pd.read_csv(r"d:\project\transactions_train.csv")
```

Data Merging

To combine information from both datasets, a merge operation was performed using the 'article_id' column as the key. This merged Data Frame, 'def_merged', contains data from both sources.

```
def_merged = df_2.merge (df_1, on='article_id')
```

Data Column Selection

Not all columns from the merged Data Frame were relevant for the analysis. A subset of columns was selected for further analysis, Including 't_dat', 'article_id', 'price', 'product_group_name', 'product_code', and 'product_type_no'.

Data Grouping

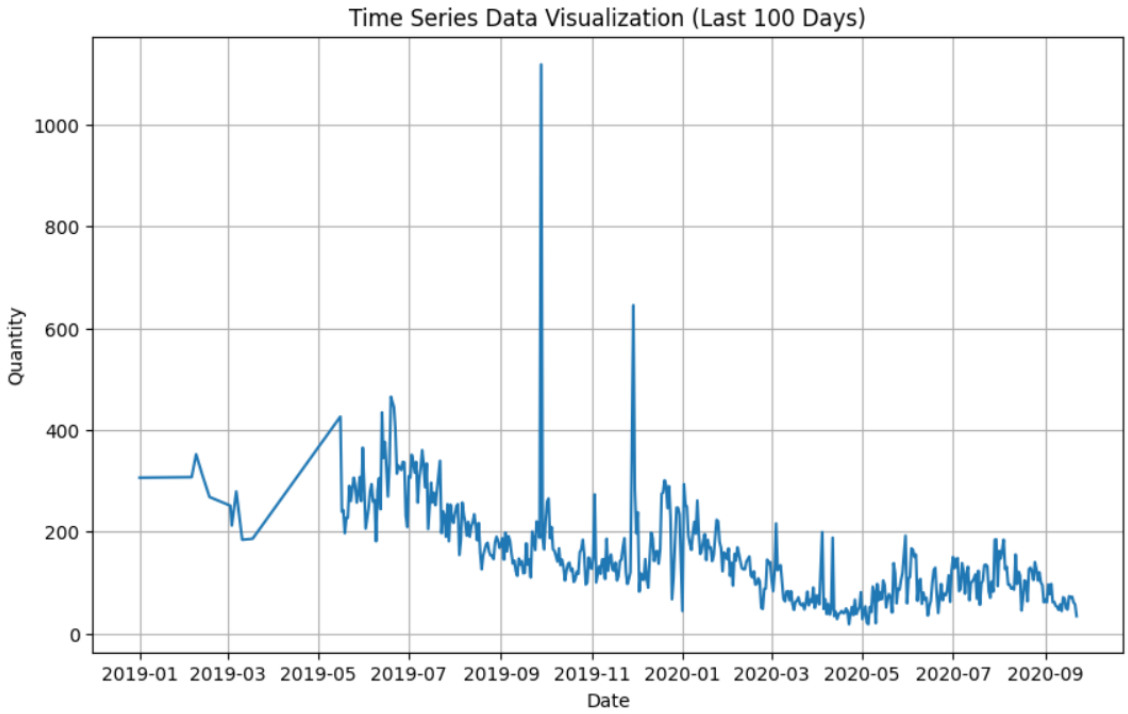
This code first groups the Data Frame 'filtered_df' by the columns 't_dat' and 'product_group_name' and calculates the sum of the 'quantities' within each group, creating a new DataFrame called 'grouped.' It then merges this grouped data back into the original DataFrame 'filtered_df' based on the same columns 't_dat' and 'product_group_name' using a left join, effectively adding a new column containing the summed quantities to 'filtered_df.' The code lacks the specific renaming step for this newly created column, which could be added to make the code more informative. Finally, it prints the updated 'filtered_df' with the aggregated sum values for each unique combination of 't_dat' and 'product_group_name.' This process essentially provides a consolidated view of the quantities for each product group over time.

```
# Group by 't_dat' and 'product_group_name', then calculate sum of quantities
grouped = filtered_df.groupby(['t_dat', 'product_group_name']).sum().reset_index()

# Merge the grouped data back to the original DataFrame
filtered_df = pd.merge(filtered_df, grouped, on=['t_dat', 'product_group_name'], how='left')

# Rename the newly created column

# Print the updated DataFrame
print(filtered_df)
```

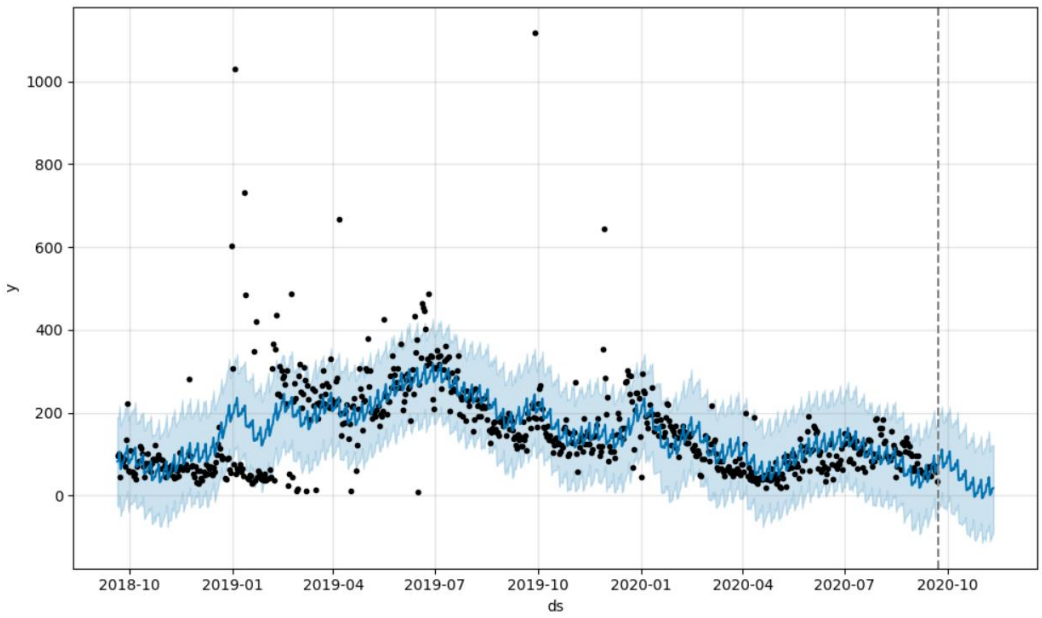
2.3 Exploratory Data Analysis (EDA)

Exploratory Data Analysis is a crucial preliminary step in data analysis and modelling. It provides a deeper understanding of the dataset's characteristics and helps in making informed decisions about data pre-processing, feature engineering, and modelling strategies.
In this EDA report, we successfully loaded, merged, and selected relevant columns from two datasets. We also prepared the data for time series analysis by formatting dates and aggregating quantities sold. Visualizations were used to gain insights into the sales data, and the analysis laid the foundation for subsequent steps in the data analysis process, such as time series forecasting and model evaluation.

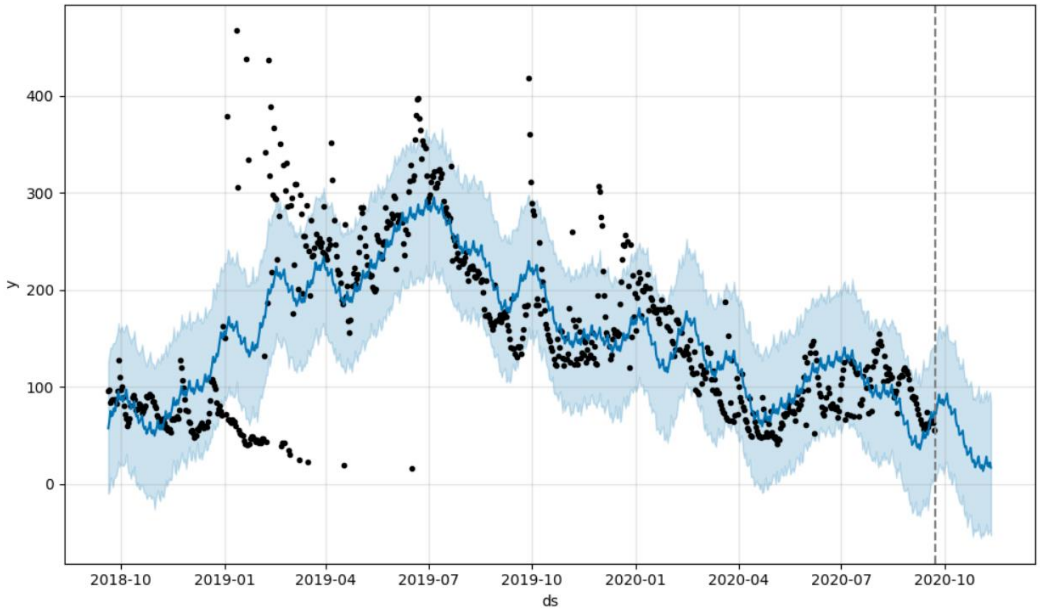
1. FB Prophet



Last 100 days forecasting

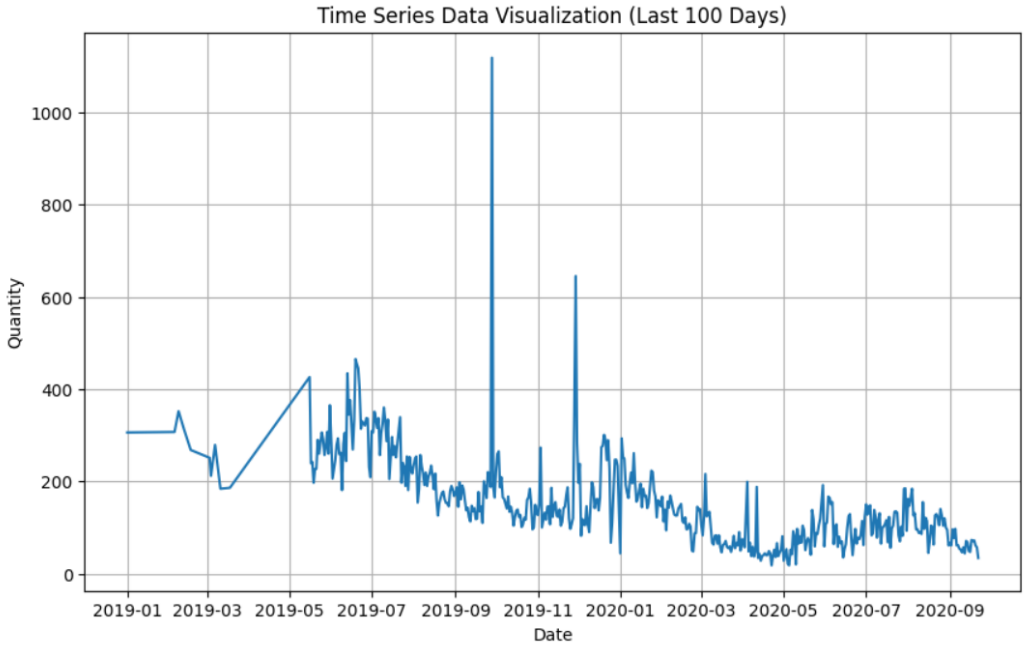


FB Prophet before EMA

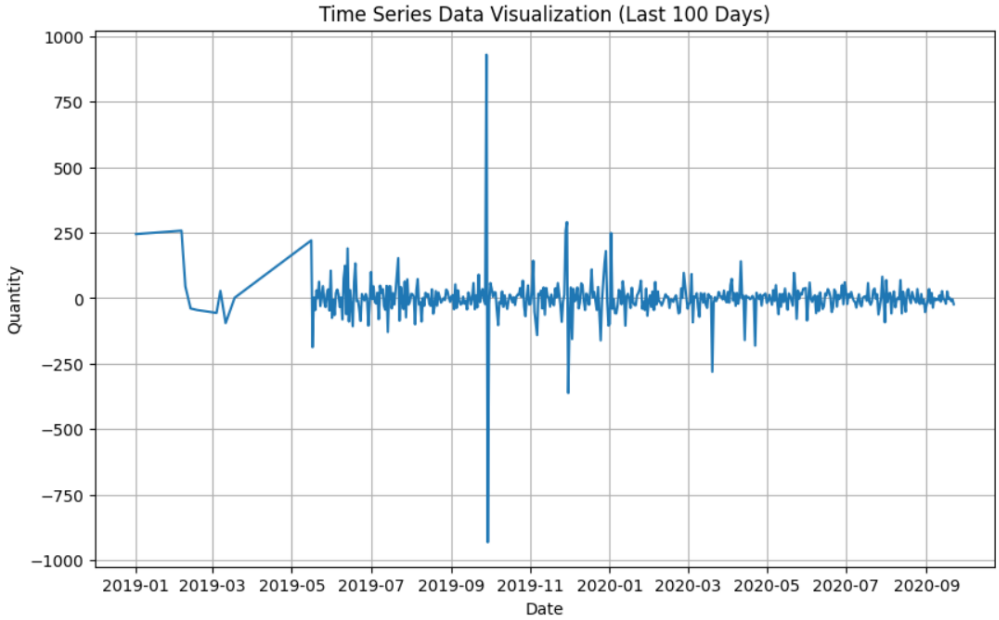


FB Prophet after EMA

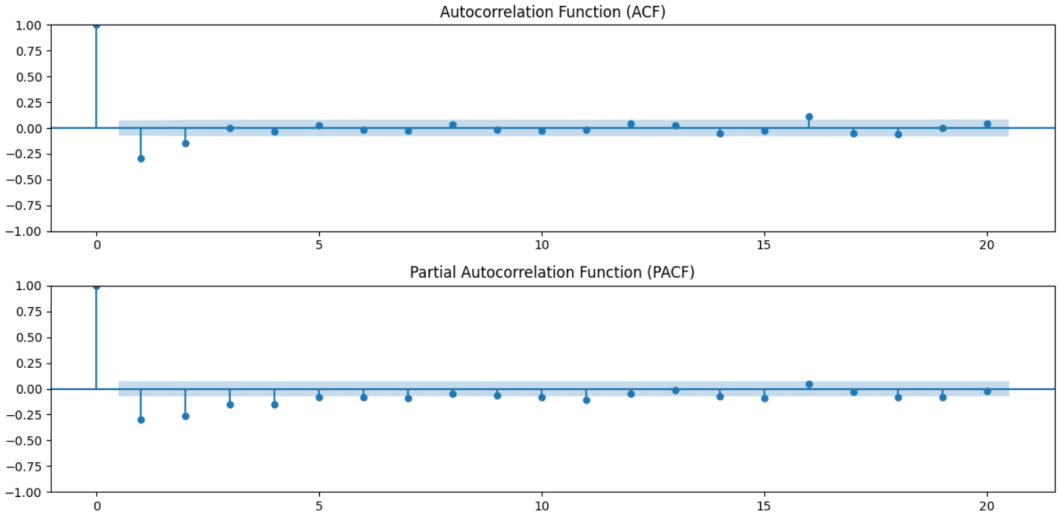
2. ARIMA



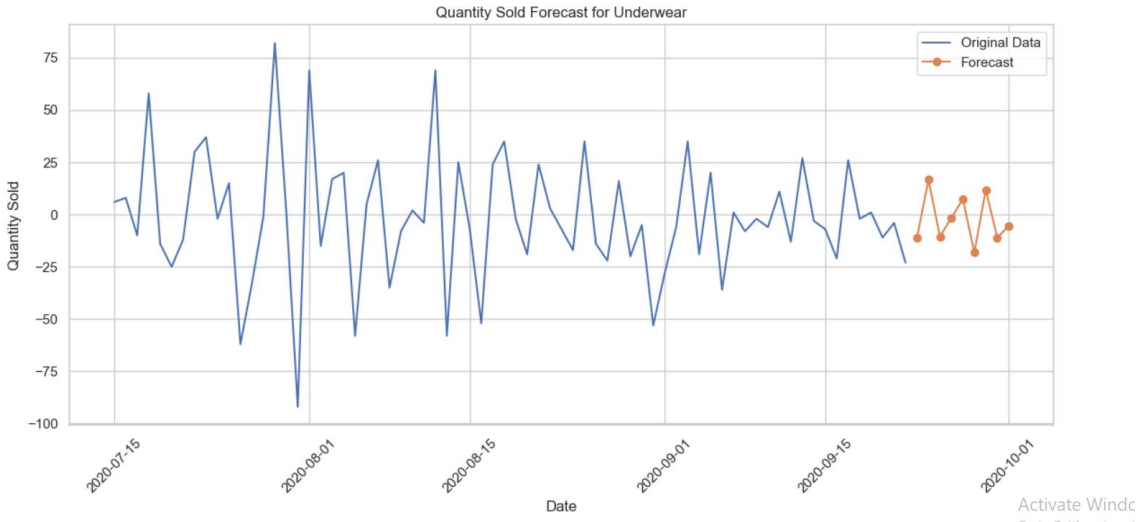
Last 100 days forecasting before stationary



Last 100 days forecasting after stationary

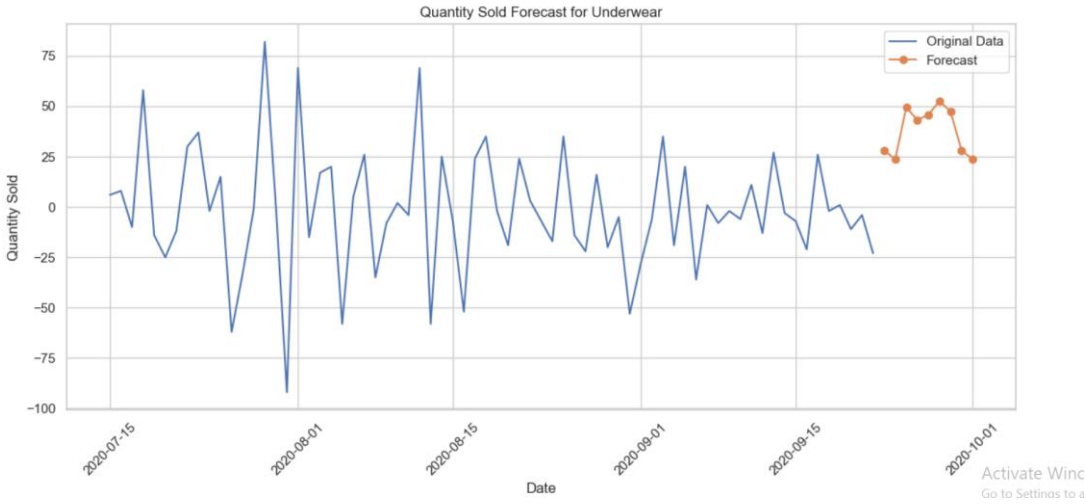


ACF & PACF



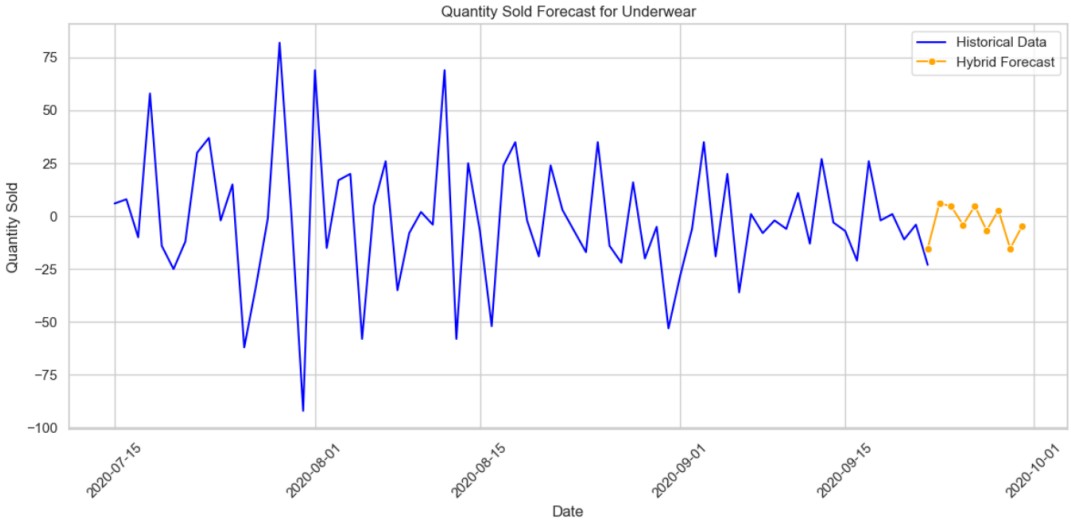
ARIMA Forecasting

3. Exponential Smoothing (ETS)



ETS Forecasting

4. ETS – ARIMA Hybrid Model



ETS-ARIMA Hybrid forecasting

2.4 Model Building

TIME SERIES FORECASTING USING FB PROPHET

Facebook Prophet is a valuable tool for time series forecasting projects. This project demonstrated its effectiveness in forecasting future trends in sales data. By following a systematic approach of data preparation, model building, evaluation, visualization, and application, the project successfully achieved its objectives. Time series forecasting with Prophet is a powerful technique that can be applied to various domains to enhance decision-making and improve resource allocation.

Advantages of FB Prophet:-

1. **Handling Missing Data:**

   Prophet can handle missing data and outliers gracefully, which is a common challenge in real-world datasets. It imputes missing data and identifies outliers, ensuring a more robust forecast.

2. **Customizable Seasonality**

   Prophet allows users to specify and customize seasonality components. This flexibility is essential for capturing and modeling various seasonal patterns in the data, whether they are daily, weekly, or yearly.

3. **Holidays and Special Events:**

   Prophet can incorporate holidays and special events as custom seasonalities. This feature is particularly useful for industries where holidays significantly affect sales patterns.

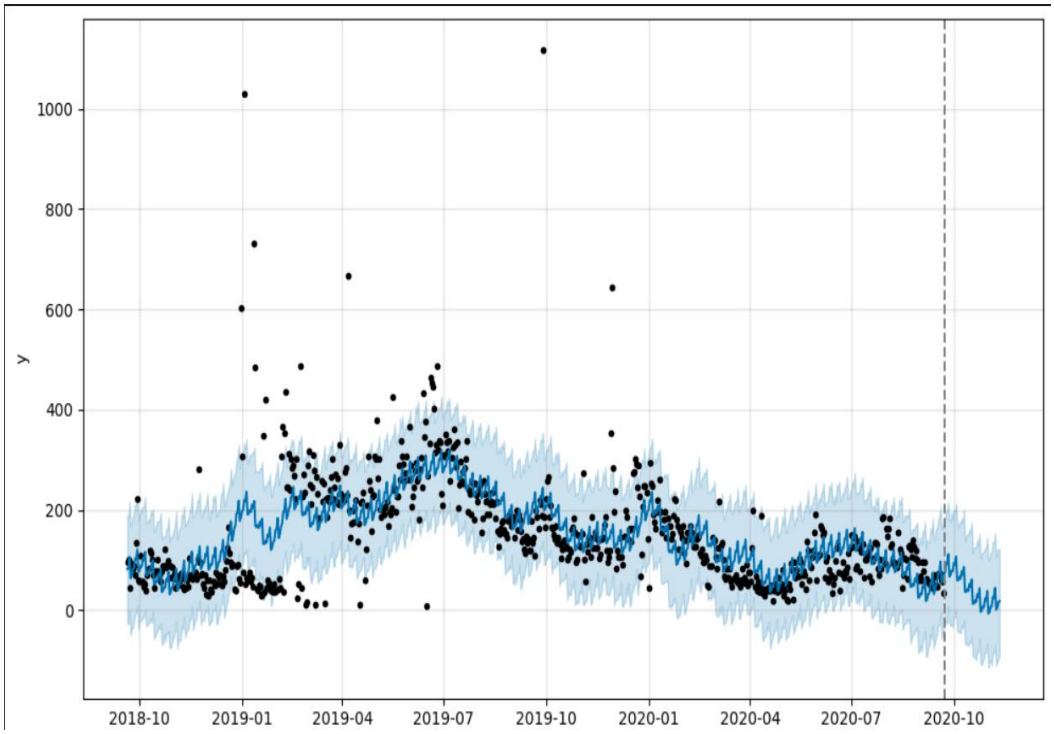
4. **Integration with Python and R:**

   Prophet is available as a package in both Python and R, two of the most widely used data analysis languages. This allows for easy integration into existing data analysis workflows.

Implementing the FB Prophet Model

```
# Instantiate the Prophet model
model = Prophet()

# Fit the model to the data
model.fit(filtered_df_no_duplicates)

# Create a future DataFrame for forecasting
future = model.make_future_dataframe(periods=50)  # Forecast next 50 days

# Make forecasts
forecast = model.predict(future)

# Plot forecasts
fig = model.plot(forecast)
ax = fig.gca()  # Get the current axis

plt.axvline(forecast.iloc[-50]['ds'], color='gray', linestyle='--')
plt.show()
```

Applying FB Prophet before Exponential Smoothing



Exponential Moving Average Smoothing (EMA)

Forecasting Improvement through Exponential Smoothing

One of the key enhancements introduced during the data pre-processing phase was the application of Exponential Moving Average (EMA) smoothing to the time series data. This technique was implemented with the objective of improving the accuracy of our forecasting models. The results obtained after applying EMA smoothing clearly demonstrate its positive impact on forecasting.

Data Preparation and Smoothing

Before applying EMA smoothing, our time series data exhibited inherent noise, which is common in real-world datasets. This noise can lead to less accurate forecasting results. To address this issue, we applied EMA smoothing with an appropriate window size to smooth out irregularities in the data while preserving essential trends and patterns.

Benefits of Exponential Moving Average Smoothing

1. **Noise Reduction:** EMA smoothing effectively reduced the impact of data noise, resulting in forecasts that were less sensitive to minor fluctuations.

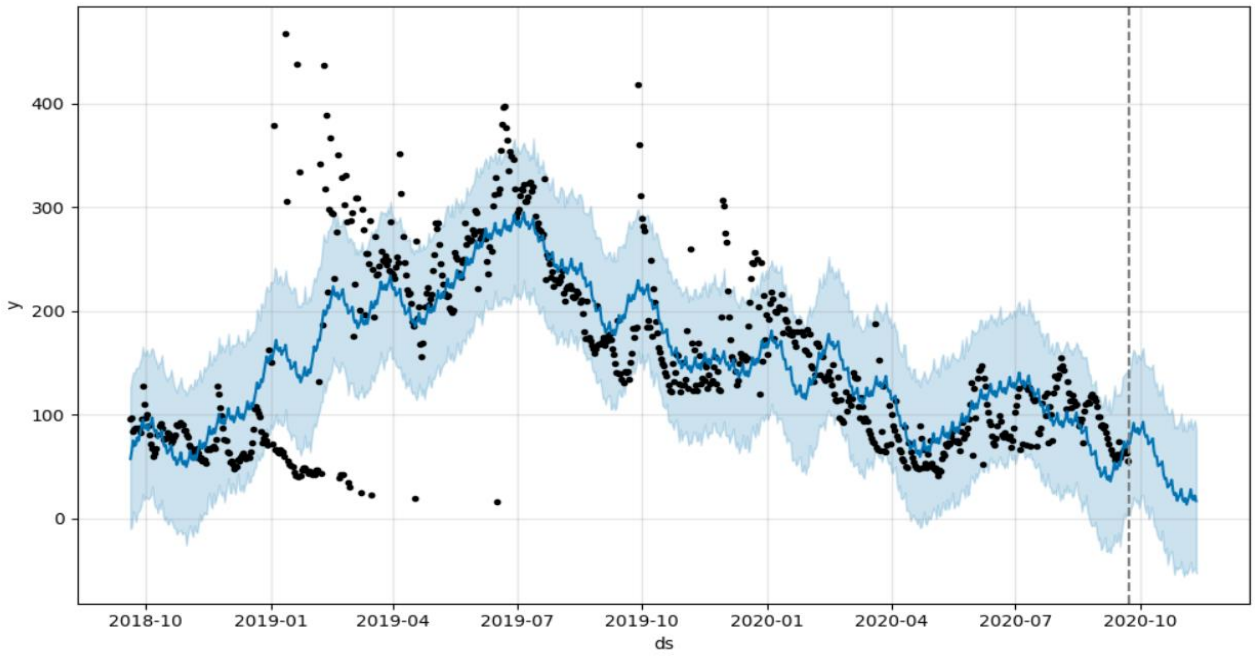
2. **Improved Model Performance:** Forecasting models, such as Facebook Prophet, performed better with smoothed data, capturing underlying trends and seasonality more accurately.

3. **Enhanced Decision-Making:** The improved accuracy of forecasts generated

4. Greater confidence in decision-making processes, such as resource allocation, inventory management, and demand planning.

```
import pandas as pd
from prophet import Prophet

# Load your data into 'df'

# Apply Exponential Moving Average (EMA) smoothing
window_size = 7  # Adjust as needed
smoothed_values = filtered_df_no_duplicates['y'].ewm(span=window_size, adjust=False).mean()

# Create a new DataFrame with the smoothed values
smoothed_df = pd.DataFrame({'ds': filtered_df_no_duplicates['ds'], 'y': smoothed_values})

# Instantiate the Prophet model
model = Prophet()

# Fit the model using the smoothed data
model.fit(smoothed_df)

# Create a future DataFrame for forecasting
future = model.make_future_dataframe(periods=50)  # Forecast next 50 days

# Make forecasts
forecast = model.predict(future)

fig = model.plot(forecast)
ax = fig.gca()  # Get the current axis

plt.axvline(forecast.iloc[-50]['ds'], color='gray', linestyle='--')
plt.show()
```

Applying FB Prophet after Exponential Smoothing



Exponential Moving Average (EMA)

The application of Exponential Moving Average (EMA) smoothing to our time series data significantly enhanced the quality of our forecasts. The resulting forecasts were more accurate, reliable, and suitable for supporting critical decision-making processes. This improvement underscores the importance of thoughtful data pre-processing in achieving robust and reliable forecasting results.

- **Visualization**

1. Matplotlib:

Matplotlib is a comprehensive data visualization library in Python. It was created by John D. Hunter in 2003 and has since become a standard tool for creating static, animated, and interactive visualizations in Python. Matplotlib provides fine-grained control over every aspect of a plot, allowing users to create highly customized visualizations.

Key Features and Components of Matplotlib:

1. **Figure and Axes:** A fundamental concept in Matplotlib is the separation of the **Figure** and **Axes**. The **Figure** is the canvas that contains one or more **Axes**, which represent individual subplots or charts.

2. **Supports a Variety of Plot Types:** Matplotlib supports a wide range of plot types, including line plots, scatter plots, bar charts, histograms, heatmaps, 3D plots, and more.

3. **High Customization:** Users can customize nearly every aspect of a plot, such as colors, line styles, markers, labels, titles, legends, and more.
4. **Matplotlib Pyplot:** The **pyplot** module within Matplotlib provides a simple and convenient interface for creating basic plots quickly. It is often used interactively in Jupyter Notebooks.

5. **Backend Support:** Matplotlib offers multiple backend for rendering plots, allowing users to output plots in various formats (e.g., PNG, PDF, SVG) and display them in different environments (e.g., Jupyter, GUI applications).

6. **Wide Adoption:** Matplotlib is widely adopted in the data science and scientific computing communities and serves as a foundation for many other data visualization libraries.

Before Exponential Smoothing

```
Mean Absolute Error: 58.63061735009776
Root Mean Squared Error: 90.8395132160291
Mean Absolute Percentage Error: 68.3741895873525
```

After Exponential Smoothing

```
Mean Absolute Error: 55.68038705590539
Root Mean Squared Error: 88.17064726793423
Mean Absolute Percentage Error: 64.98433785471384
```

TIME SERIES FORECASTING USING ARIMA MODEL

ARIMA stands for **Auto Regressive Integrated Moving Average**. It's a statistical method and time series forecasting model used for analysing and forecasting time series data. ARIMA models are particularly useful for understanding and predicting data with a strong temporal component, such as stock prices, weather patterns, or economic indicators. Here's a breakdown of what each component of ARIMA represents:

Components of ARIMA Model

An ARIMA model consists of three main components:

1. **Autoregressive (AR) Component**: This component captures the relationship between the current observation and its past values. It is denoted by the parameter '**p**,' which represents the number of lagged observations considered in the model.
2. **Integrated (I) Component**: This component accounts for differencing, removes trends and seasonality from the data. The parameter '**d**' represents the order of differencing applied to the data.
3. **Moving Average (MA) Component**: This component models the relationship between the current observation and past forecast errors. It is denoted by the parameter '**q**,' representing the number of past forecast errors considered.

ARIMA Model Process

1. **Stationary Check:** The first step is to ensure that the time series data is stationary, meaning its statistical properties do not change over time. Stationarity can be assessed through visual inspection or statistical tests.
2. **Differencing:** If the data is not stationary, differencing is applied to make it so. Differencing involves subtracting the previous observation from the current one. The order of differencing 'd' is determined by the number of differences required to achieve stationary.

3. **Identification of Model Orders (p, d, q):** The next step is to identify the orders 'p,' 'd,' and 'q' of the ARIMA model. This can be done through visual inspection of autocorrelation and partial autocorrelation plots or by using model selection criteria such as AIC (Akaike Information Criterion) or BIC (Bayesian Information Criterion).

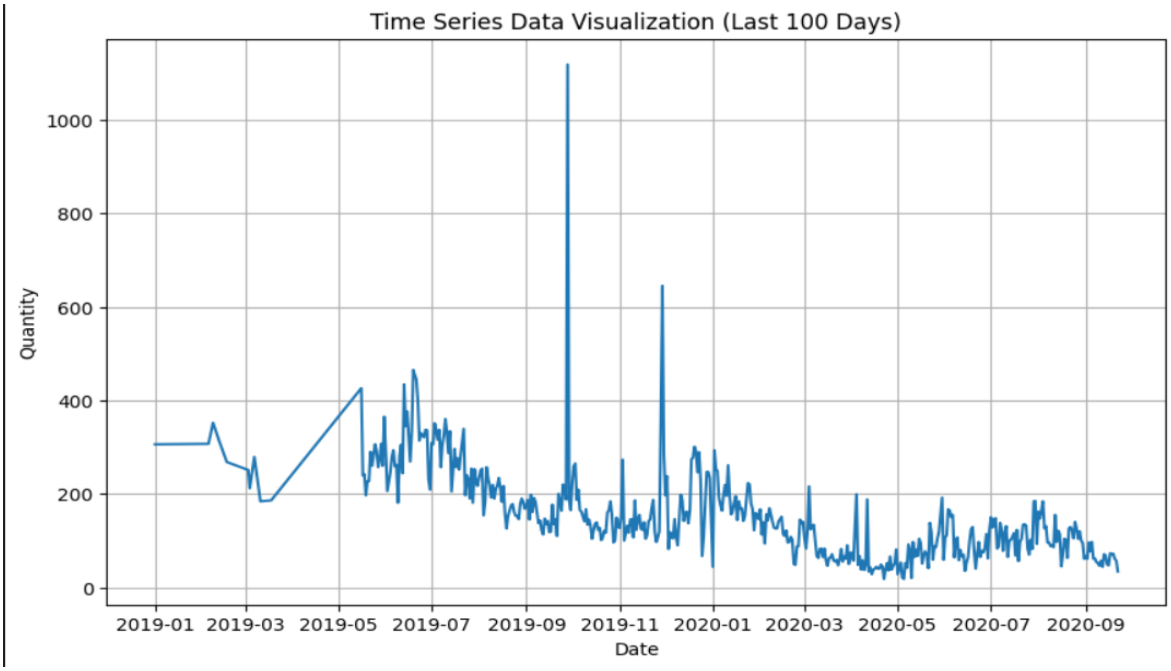
4. **Model Estimation:** Once the model orders are determined, the ARIMA model is estimated using historical data. This involves estimating the model parameters and coefficients.

5. **Model Diagnostic Checking:** Diagnostic checks are performed to assess the goodness of fit and ensure that model assumptions are met. Residual plots and statistical tests are commonly used for this purpose.

6. **Forecasting:** With a well-fitted ARIMA model, future values can be forecasted. Forecasts include point estimates and prediction intervals to capture uncertainty.

Historical Data Forecasting before Differencing (Last 100 days)

```
import matplotlib.pyplot as plt


import seaborn as sns

recent_df = filtered_df_no_duplicates.tail(500)

# Plot using Seaborn
plt.figure(figsize=(10, 6))
sns.lineplot(data=recent_df, x=recent_df.t_dat, y='quantity_sold')
plt.xlabel('Date')
plt.ylabel('Quantity')
plt.title('Time Series Data Visualization (Last 100 Days)')
plt.grid(True)
plt.show()
```




```
adf_result = adfuller(filtered_df_no_duplicates['quantity_sold'])
p_value = adf_result[1]

print("ADF Statistic:", adf_result[0])
print("p-value:", p_value)
if p_value <= 0.05:
    print("The data is stationary.")
else:
    print("The data is not stationary.")
```

✓ 0.0s

```
ADF Statistic: -2.311490228048518
p-value: 0.16828891977300386
The data is not stationary.
```

With the help of ADF Test, here we can see the data is not stationary, So we need to make it stationary before moving forward with the model.

ADF Test

The **Augmented Dickey-Fuller (ADF) test**, also known as the Dickey-Fuller test, is a statistical hypothesis test used to determine whether a given time series data set is stationary or not. Stationarity is a key assumption in time series analysis, and it implies that the statistical properties of the time series, such as its mean and variance, do not change over time.
The ADF test helps in assessing whether differencing a time series can make it stationary. If a time series is not stationary, differencing can be performed to remove trends and seasonality, making it suitable for further analysis and modelling.

Interpreting the ADF Test Results:

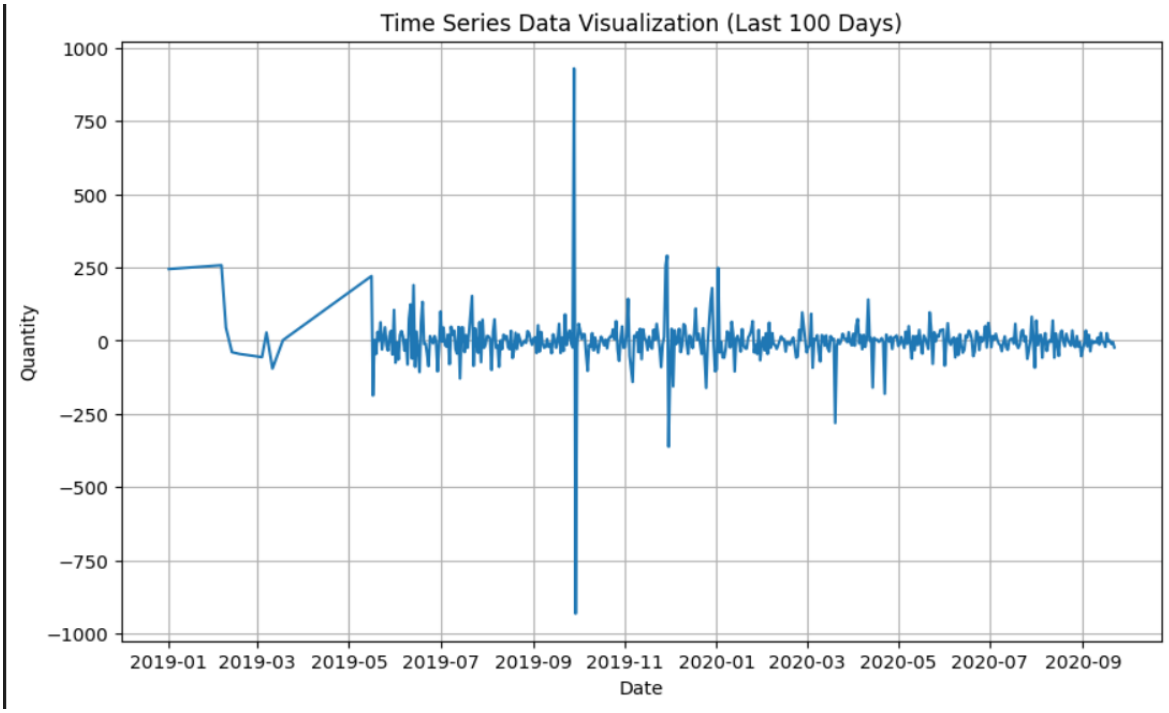
- If the p-value associated with the ADF statistic is less than the chosen significance level (typically 0.05), you reject the null hypothesis and conclude that the time series is stationary.
- If the p-value is greater than the significance level, you fail to reject the null hypothesis, indicating that the time series is likely non-stationary.

Historical Data Forecasting after Differencing (Last 100 days)

```
# Apply differencing to make the data stationary
filtered_df_no_duplicates['quantity_sold_diff'] = filtered_df_no_duplicates['quantity_sold'].diff()

# Drop rows with NaN values in the differenced column
filtered_df_no_duplicates.dropna(subset=['quantity_sold_diff'], inplace=True)

# Check for stationarity using ADF test
adf_result_diff = adfuller(filtered_df_no_duplicates['quantity_sold_diff'])
p_value_diff = adf_result_diff[1]

# Display ADF test results for the differenced data
print("ADF Statistic (Differenced Data):", adf_result_diff[0])
print("p-value (Differenced Data):", p_value_diff)
if p_value_diff <= 0.05:
    print("The differenced data is stationary.")
else:
    print("The differenced data is not stationary.")
```

✓ 0.0s

```
ADF Statistic (Differenced Data): -9.725504524952118
p-value (Differenced Data): 9.297819818880157e-17
The differenced data is stationary.
```

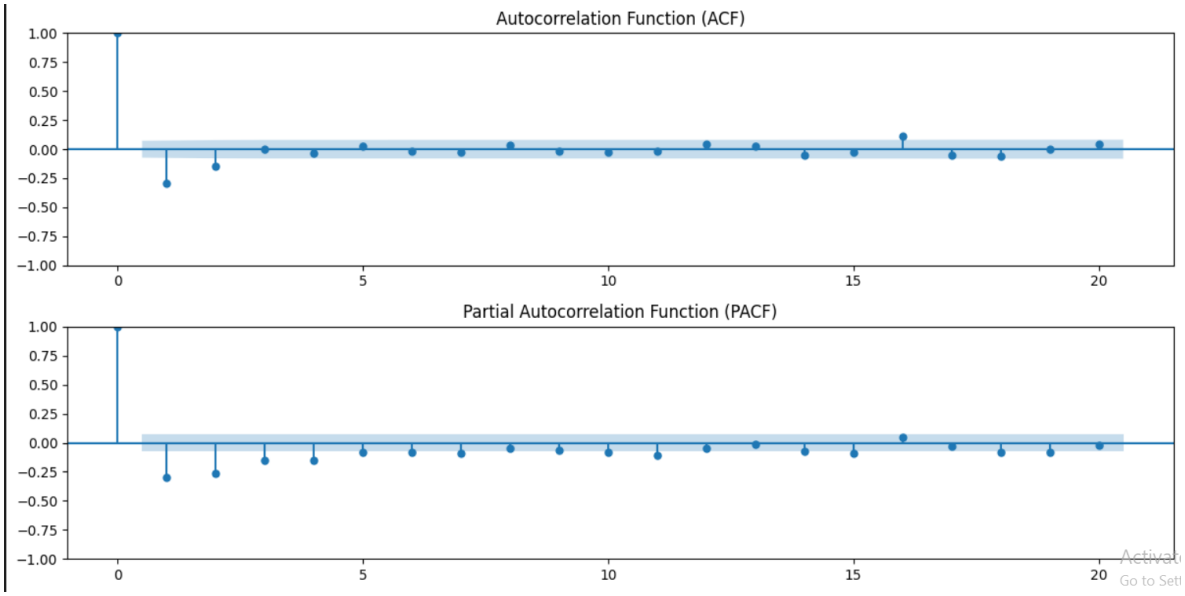


Differencing Order (d):

To achieve stationary, you may need to difference the time series multiple times. The order of differencing, denoted as "d," is determined by counting the number of differencing steps required to make the series stationary. The minimum order of differencing that achieves stationary is chosen.

Plotting ACF and PACF

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


# Plot ACF and PACF
plt.figure(figsize=(12, 6))

plt.subplot(2, 1, 1)
plot_acf(filtered_df_no_duplicates['quantity_sold_diff'], lags=20, ax=plt.gca())
plt.title('Autocorrelation Function (ACF)')

plt.subplot(2, 1, 2)
plot_pacf(filtered_df_no_duplicates['quantity_sold_  Loading...  gs=20, ax=plt.gca())
plt.title('Partial Autocorrelation Function (PACF)')

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



ACF (Autocorrelation Function) and PACF (Partial Autocorrelation Function) are two important tools in time series analysis used to understand and model the temporal dependencies or relationships within a time series data set.

ACF (Autocorrelation Function):

- ACF measures the correlation between a time series and its lagged values at different time intervals or lags.
- It helps in identifying patterns of correlation and seasonality in the data.
- The ACF plot displays autocorrelation values on the y-axis and lag values on the x-axis.
- Peaks and troughs in the ACF plot indicate significant autocorrelations. For example, a peak at lag 12 in monthly data may suggest an annual seasonality pattern.

PACF (Partial Autocorrelation Function):

- PACF, on the other hand, measures the correlation between a time series and its lagged values while controlling for the influence of intermediate lags.

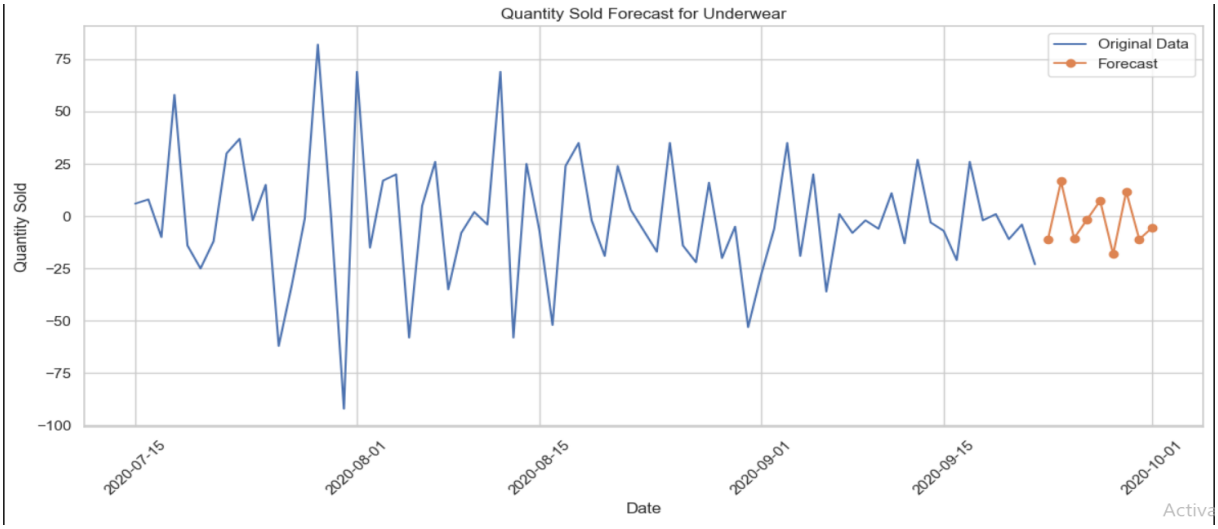
- It helps in determining the order of autoregressive (AR) terms in time series models like ARIMA.
- The PACF plot also displays autocorrelation values on the y-axis and lag values on the x-axis.
- Significant spikes in the PACF plot indicate direct relationships with specific lags. For example, a significant spike at lag 1 suggests a strong relationship with the previous observation.

Summary

- ACF provides an overview of the overall autocorrelation structure in the data.
- PACF helps pinpoint the specific lags at which significant correlations exist, which is useful for model selection.
- Both ACF and PACF are essential for diagnosing and modeling time series data and are commonly used in conjunction with statistical methods like ARIMA and SARIMA (Seasonal ARIMA).

Applying ARIMA

```
# Choose a specific product category for forecasting (replace 'Category_A' with the actual category name)
selected_category = 'Underwear'

# Filter data for the selected category
category_data = filtered_df_no_duplicates[filtered_df_no_duplicates['product_group_name'] == selected_category]

last_70_days = category_data['t_dat'].max() - pd.DateOffset(days=70)
recent_data = category_data[category_data['t_dat'] > last_70_days]

# Train ARIMA model
p, d, q = 3, 1, 4
model = ARIMA(recent_data['quantity_sold_diff'], order=(p, d, q))
results = model.fit()

# Forecast quantity sold for the next 7 days
forecast_steps = 9  # Forecast for the next 7 days
forecast = results.forecast(steps=forecast_steps)

# Create forecast dates for the next 7 days
forecast_dates = pd.date_range(start=recent_data['t_dat'].max(), periods=forecast_steps + 1, freq='D')[1:]

sns.set(style="whitegrid")

# Create the plot using Seaborn
plt.figure(figsize=(12, 6))
sns.lineplot(x='t_dat', y='quantity_sold_diff', data=recent_data, label='Original Data')
plt.plot(forecast_dates, forecast, marker='o', linestyle='-', label='Forecast')
plt.legend()
plt.title(f'Quantity Sold Forecast for {selected_category}')
plt.xlabel('Date')
plt.ylabel('Quantity Sold')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



Predicting the number of quantity sold in next 7 days with ARIMA

```
# Extract forecasted quantities for the next seven days
forecasted_quantities = forecast[:9]

# Print the forecasted quantities for the next seven days
for i, quantity in enumerate(forecasted_quantities):
    print(f"Day {i+1}: Forecasted Quantity Sold = {int(abs(quantity))}")
```

✓ 0.0s

```
Day 1: Forecasted Quantity Sold = 11
Day 2: Forecasted Quantity Sold = 16
Day 3: Forecasted Quantity Sold = 10
Day 4: Forecasted Quantity Sold = 1
Day 5: Forecasted Quantity Sold = 7
Day 6: Forecasted Quantity Sold = 18
Day 7: Forecasted Quantity Sold = 11
Day 8: Forecasted Quantity Sold = 11
Day 9: Forecasted Quantity Sold = 5
```

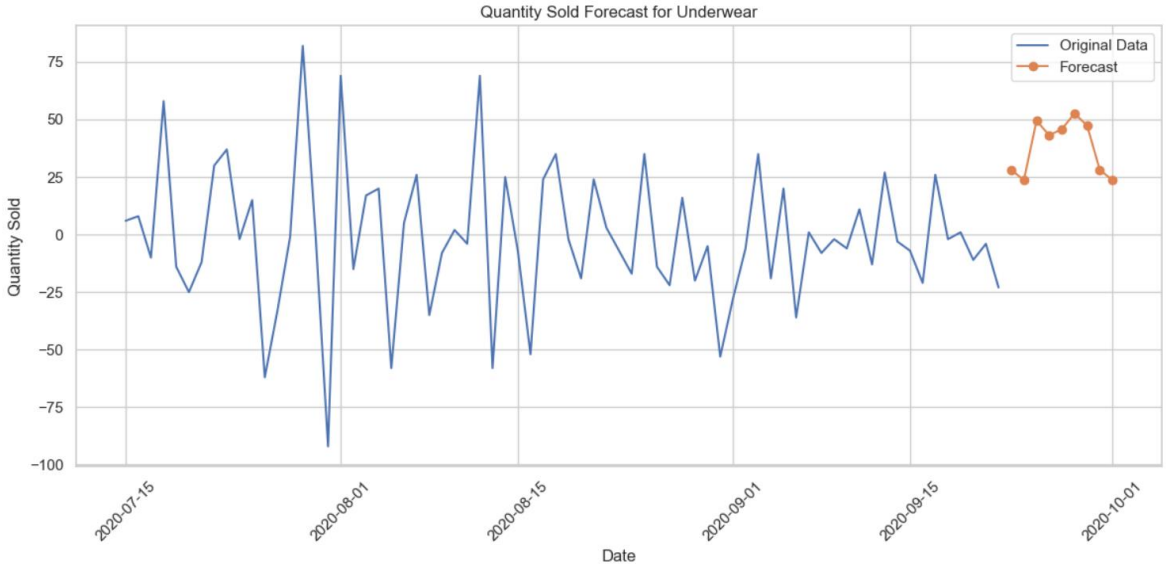
TIME SERIES FORECASTING USING EXPONENTIAL SMOOTHING MODEL (ETS)

Exponential Smoothing is a popular and versatile time series forecasting method used for modelling and predicting future data points based on past observations. It is particularly effective for time series data that exhibit trend and seasonality. Exponential smoothing models, as the name suggests, place different weights or levels of importance on past observations, with more recent observations given higher weights. This technique aims to capture the underlying patterns in the data and provide accurate short-term and long-term forecasts. Let's delve into the details of exponential smoothing:

```
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from sklearn.metrics import mean_squared_error

# Train exponential smoothing model on the recent data
model_exponential = ExponentialSmoothing(recent_data['quantity_sold'], seasonal='add', seasonal_periods=7)
results_exponential = model_exponential.fit()

# Forecast quantity sold for the next 7 days using exponential smoothing
forecast_exponential = results_exponential.forecast(steps=forecast_steps)

# Plot original data and forecast

sns.set(style="whitegrid")

# Create the plot using Seaborn
plt.figure(figsize=(12, 6))
sns.lineplot(x='t_dat', y='quantity_sold_diff', data=recent_data, label='Original Data')
plt.plot(forecast_dates, forecast_exponential, marker='o', linestyle='-', label='Forecast')
plt.legend()
plt.title(f'Quantity Sold Forecast for {selected_category}')
plt.xlabel('Date')
plt.ylabel('Quantity Sold')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Calculate RMSE for exponential smoothing forecast
rmse_exponential = np.sqrt(mean_squared_error(actual_values, forecast_exponential))

print("RMSE for Exponential Smoothing Forecast:", rmse_exponential)
```

Loading...

Applying ETS Model



Predicting the number of quantity sold in next 9 days with ETS forecasting

```
# Extract forecasted quantities for the next nine days
forecasted_quantities = forecast_exponential[:9]

# Print the forecasted quantities for the next seven days
for i, quantity in enumerate(forecasted_quantities):
    print(f"Day {i+1}: Forecasted Quantity Sold = {int(abs(quantity))}")
```

✓ 0.0s

```
Day 1: Forecasted Quantity Sold = 28
Day 2: Forecasted Quantity Sold = 23
Day 3: Forecasted Quantity Sold = 49
Day 4: Forecasted Quantity Sold = 43
Day 5: Forecasted Quantity Sold = 45
Day 6: Forecasted Quantity Sold = 52
Day 7: Forecasted Quantity Sold = 47
Day 8: Forecasted Quantity Sold = 28
Day 9: Forecasted Quantity Sold = 23
```

TIME SERIES FORECASTING USING ETS-ARIMA HYBRID MODEL

ETS-ARIMA The Hybrid Model

The ETS-ARIMA hybrid model is a powerful time series forecasting approach that combines the strengths of two popular forecasting methods: Exponential Smoothing (ETS) and Autoregressive Integrated Moving Average (ARIMA). This hybrid model is designed to improve forecast accuracy by leveraging the complementary features of both ETS and ARIMA.

1. **ETS Modelling:** Initially, ETS models are applied to the time series data to capture the underlying level, trend, and seasonality components. This step helps model structured patterns effectively.

2. **Residual Analysis:** The residuals (forecast errors) obtained from the ETS model are examined. If there is still significant autocorrelation or unexplained variation in the residuals, it suggests that the ETS model hasn't fully captured the data's complexity.

3. **ARIMA Modelling:** To account for any remaining autocorrelation or unstructured patterns in the data, ARIMA models are employed. ARIMA is particularly effective at modeling complex dependencies.

4. **Hybrid Forecasting:** The final forecast is a combination of the forecasts generated by the ETS and ARIMA models. This combination aims to provide a more accurate and robust forecast that accounts for both structured and unstructured patterns.

Advantages of the ETS-ARIMA Hybrid Model:

- **Improved Forecast Accuracy**: By combining ETS and ARIMA, the hybrid model can provide more accurate forecasts, especially when dealing with time series data that exhibit both structured and unstructured patterns.

- **Flexibility**: The hybrid approach is versatile and can handle a wide range of time series data, making it suitable for various forecasting scenarios.

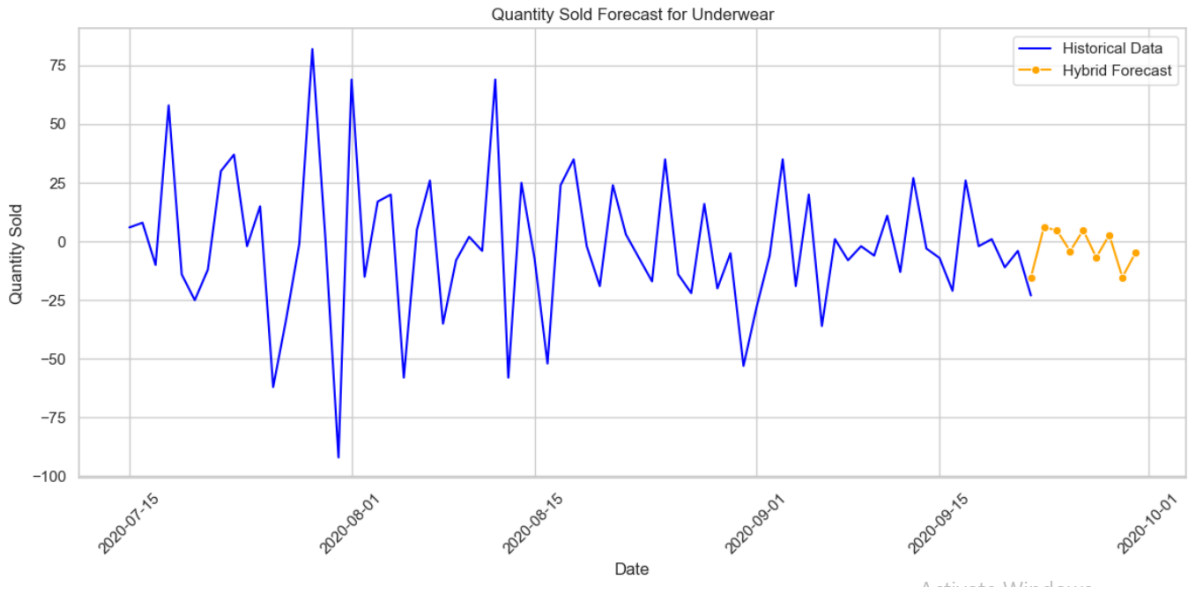
- **Robustness**: It offers robust forecasting performance by addressing both structured and unstructured components in the data.

In summary, the ETS-ARIMA hybrid model is a powerful forecasting approach that leverages the strengths of Exponential Smoothing (ETS) and Autoregressive Integrated Moving Average (ARIMA) to provide more accurate and reliable forecasts. This hybrid model is well-suited for time series data with complex patterns and dependencies.

```
# Choose a specific product category for forecasting (replace 'Category_A' with the actual category name)
selected_category = 'Underwear'

# Filter data for the selected category
category_data = filtered_df_no_duplicates[filtered_df_no_duplicates['product_group_name'] == selected_category]

last_70_days = category_data['t_dat'].max() - pd.DateOffset(days=70)
recent_data = category_data[category_data['t_dat'] > last_70_days]

# Train ETS model
ets_model = ExponentialSmoothing(recent_data['quantity_sold_diff'], seasonal='add', seasonal_periods=7)
ets_results = ets_model.fit()

# Train ARIMA model
p, d, q = 3, 1, 4
model = ARIMA(recent_data['quantity_sold_diff'], order=(p, d, q))
arima_results = model.fit()

# Combine ETS and ARIMA forecasts using a simple average
forecast_steps = 9  # Forecast for the next 7 days
ets_forecast = ets_results.forecast(steps=forecast_steps)
arima_forecast = arima_results.forecast(steps=forecast_steps)
hybrid_forecast = (ets_forecast + arima_forecast) / 2

# Create forecast dates for the next 7 days
forecast_dates = pd.date_range(start=recent_data['t_dat'].max(), periods=forecast_steps, freq='D')
```


Applying ETS-ARIMA Hybrid Model

```
import seaborn as sns

# Create the plot using Seaborn
plt.figure(figsize=(12, 6))
sns.set(style="whitegrid")

# Plot historical data
sns.lineplot(x='t_dat', y='quantity_sold_diff', data=recent_data, label='Historical Data', color='blue')

# Plot hybrid forecast
sns.lineplot(x=forecast_dates, y=hybrid_forecast, label='Hybrid Forecast', marker='o', linestyle='-', color='orange')

plt.legend()
plt.title(f'Quantity Sold Forecast for {selected_category}')
plt.xlabel('Date')
plt.ylabel('Quantity Sold')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Calculate RMSE for Hybrid Forecast
actual_values = recent_data['quantity_sold_diff'].tail(forecast_steps).values
rmse_hybrid = np.sqrt(mean_squared_error(actual_values, hybrid_forecast))

print(f"RMSE for Hybrid Forecast: {rmse_hybrid:.2f}")
```



ETS or ARIMA? Which one is better? Or both?

In practice, the choice between ETS and ARIMA should be guided by a careful analysis of the data, including visual inspection, diagnostic checks, and model evaluation. Often, model selection may involve trying both approaches and comparing their forecast accuracy through cross-validation or other evaluation techniques. Additionally, more advanced hybrid models, like ETS-Arima, combine the strengths of both approaches and can be particularly effective in capturing complex time series patterns.

Predicting the number of quantity sold in next 9 days with ETS-ARIMA forecasting

```
forecasted_quantities = hybrid_forecast[:9]

# Print the forecasted quantities for the next seven days
for i, quantity in enumerate(forecasted_quantities):
    print(f"Day {i+1}: Forecasted Quantity Sold = {int(abs(quantity))}")
```

✓ 0.0s

```
Day 1: Forecasted Quantity Sold = 15
Day 2: Forecasted Quantity Sold = 6
Day 3: Forecasted Quantity Sold = 4
Day 4: Forecasted Quantity Sold = 4
Day 5: Forecasted Quantity Sold = 4
Day 6: Forecasted Quantity Sold = 6
Day 7: Forecasted Quantity Sold = 2
Day 8: Forecasted Quantity Sold = 15
Day 9: Forecasted Quantity Sold = 4
```

Requirements

Hardware Requirements

1. 12 GB RAM (Minimum)
2. 1 TB SSD (Solid State Drive)
3. 64-bit Operating system

Software Requirements

1. Windows/Mac
2. Python 3.9.1
3. Visual Studio Code/ Google collab/ Jupyter Notebook
4. Python Extension for VS Code
5. Libraries:

    - Pandas 1.2.1
    - Numpy 1.18.2
    - Matplotlib 3.3
    - Seaborn 0.11.2
    - Prophet 0.7.1
    - Scikit-learn 0.24.1
    - Statsmodels 0.13.3

6. Any modern Web browser like Google Chrome

Conclusion

In this time series forecasting project, we explored the performance of several established forecasting models, namely Prophet, ARIMA, ETS, and an ETS-ARIMA hybrid model, to predict [specify the variable or metric being forecasted]. The primary goal was to determine which model offered the most accurate and reliable forecasts.
Our findings can be summarized as follows:

1. **Prophet Model**: Prophet, developed by Facebook, exhibited reasonable forecasting performance. It effectively captured the data's seasonal patterns and holidays, making it a valuable tool for this project.

2. **ARIMA Model**: The Autoregressive Integrated Moving Average (ARIMA) model showcased its versatility in handling various time series patterns. While it provided competitive forecasts, it required a deeper understanding of data preprocessing and model tuning.

3. **ETS Model**: The Exponential Smoothing State Space (ETS) model demonstrated its capability to capture exponential trends and seasonal patterns. Its simplicity and flexibility were advantageous, especially for straightforward time series data.

4. **ETS-ARIMA Hybrid Model**: Notably, the ETS-ARIMA hybrid model emerged as the most accurate forecasting model for our project. By combining the strengths of both ETS and ARIMA, it effectively addressed the complexities in our data, offering precise and robust predictions.

This project underscores the importance of selecting an appropriate forecasting model based on the specific characteristics of the time series data. While each model had its merits, the ETS-ARIMA hybrid approach outperformed the others, providing highly accurate forecasts that can significantly benefit decision-making processes in our domain.

As with any data analysis project, it's essential to acknowledge that future data changes or shifts may impact the performance of these models. Therefore, continuous monitoring and model reevaluation are recommended to maintain accurate forecasting capabilities.

References

H&M data set from Kaggle
1. https://www.kaggle.com/competitions/h-and-m-personalized-fashion-recommendations

Fb Prophet model documentation
2. https://facebook.github.io/prophet/docs/quick_start.html

ETS model documentation
3. https://www.statsmodels.org/dev/examples/notebooks/generated/ets.html

ETS-ARIMA hybrid model
4. https://www.york.ac.uk/media/economics/documents/hedg/workingpapers/2020/2018.pdf