Advanced Time Series Analysis and Modeling

Rohit R Nath*1
*[1]CH24M561, Group-1
December 29, 2024

ABSTRACT

This report investigates time series modeling techniques applied to two distinct datasets:Airline passenger traffic and Nifty50 stock market index. Employing statistical methods such as ARIMA (Auto-Regressive Integrated Moving Average), SARIMA (Seasonal ARIMA), and the deep-learning based Prophet model, the study evaluates the effectiveness of these approaches in capturing trends, seasonality, and variability. The inclusion of the Box-Cox transformation ensures stabilized variance for robust model performance. Comparative analysis of model accuracy underscores the strengths and limitations of each methodology, providing insights into their applicability across domains.

Keywords: Time series, ARIMA, SARIMA, Prophet, box-cox transformation, stationarity.

I. INTRODUCTION

Time series analysis is a cornerstone of temporal data modeling, allowing researchers to uncover trends, seasonality, and patterns intrinsic to dynamic processes. This report focuses on the modeling of two datasets: the airline passenger traffic dataset, comprising monthly totals of international airline passengers (1949-1960), and the Nifty50 stock market dataset, representing daily price movements of key Indian equities(2005-2024).

We employ statistical and computational methodologies, including ARIMA (Auto-Regressive Integrated Moving Average), SARIMA (Seasonal ARIMA), and Prophet models, to forecast time series data. Furthermore, the application of the Box-Cox transformation addresses variance stabilization, enhancing model robustness.

All the code used for the analysis and modeling are open and available here: https://github.com/rohitrnath/Advanced-Time-Series-Analysis

II. Data Overview

A. Dataset Description

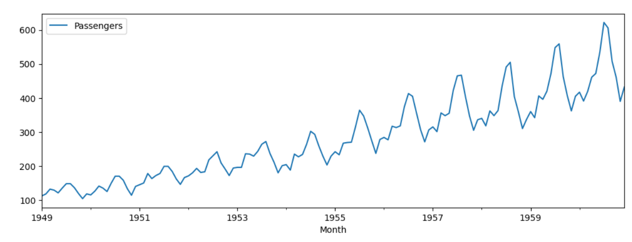
Attributes for Airline Passenger Traffic dataset are as follows:

1. Month: Time stamp for observation
2. Passengers: Number of airline passengers(in thousands)
3. Count: 144 data points

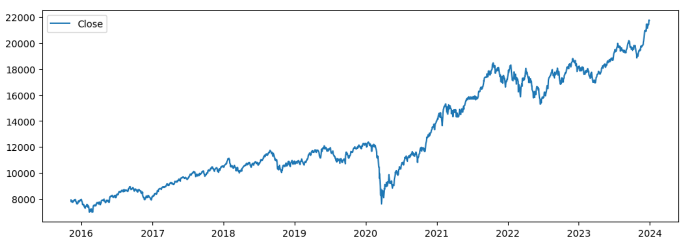
It's monthly data ranging from Jan 1st 1949 to Dec 1st 1960.

Attributes for Nifty50 stock market datasets are as follows:

1. Date: Daily timestamp.
2. Open: Daily candle open price.
3. Close: Daily candle close price.
4. High: Daily candle highest price of the day.
5. Low: Daily candle lowest price of the day.
6. Count:
   a. Train data: 2016 data points.
   b. Test data: 224 data points.



(a)



(b)

Figure 1. Visualization of Airline Passenger Trafffic Dataset (a) and Nifty50 stock market dataset (b).

The dataset contains daily candle data ranging from Jan 1st 2015 to 29th Dec 2023. And one set of test data ranging from Jan 1st 2024 to 25th Nov 2024.

We are only using the Close price from the dataset as it can represent the price of the day.

There is a rare occurane of huge fall in Nifty50 due to the Corona 19 pandemic. Considering this situation we can eliminate all the data before 1st April 2020.

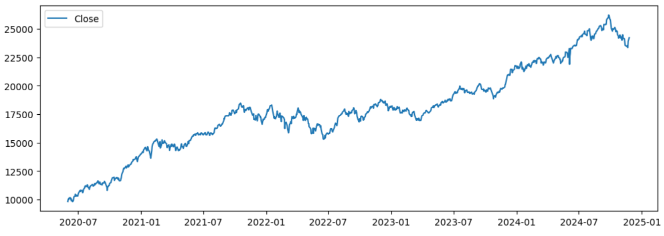


Figure 2: Visualization of Nifty50 after the Corona Pandemic

B. **Dataset Description**

For both datasets, timestamps were converted to datetime objects and appropriately indexed to facilitate time series operations. Missing values, where present, were imputed using interpolation methods.

C. **Exploratory Data Analysis (EDA)**

Visualization of both datasets are present in figure 1. Figure 1.a Airlne traffic line plots reveals the distinct trends and periodic seasonality.

Check for Seasonality

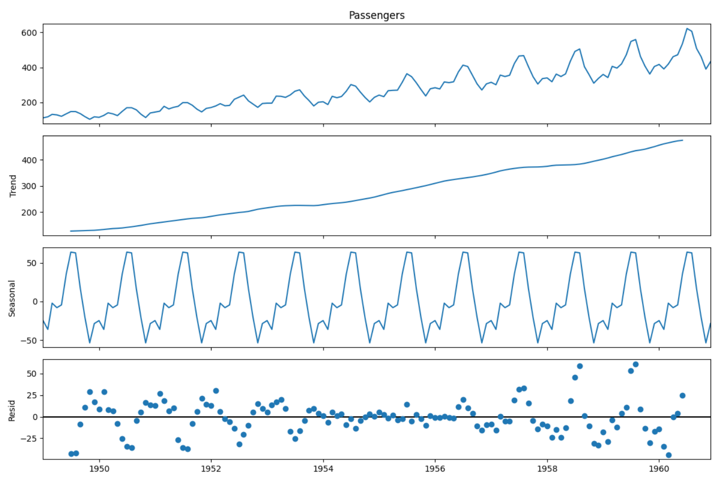
Seasonal decompose is used to decompose a time series into its trend, seasonal and residual (or noise)

components. This is useful in time series analysis to understand and isolate different underlying patterns within the data.

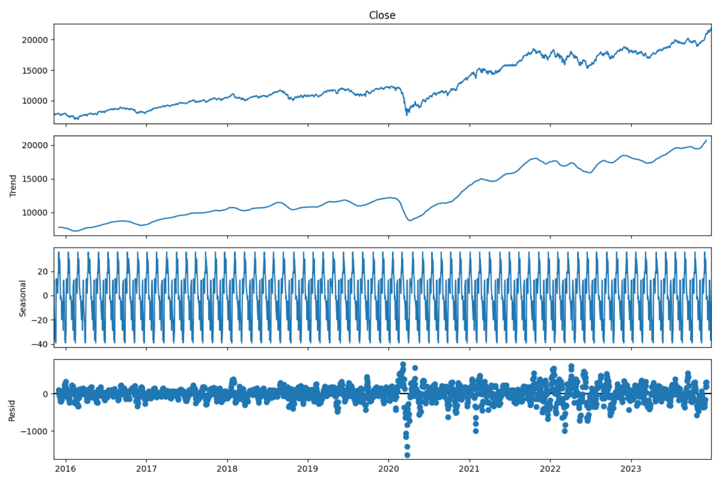
Additive decomposition(figure 3): Assumes the time series is the sum of the trend, seasonal, and residual components.

y(t) = Trend(t) + Seasonal(t) + Residual(t)

Multiplicative decomposition(figure 4): Assumes the time series is the product of the trend, seasonal, and residual component.

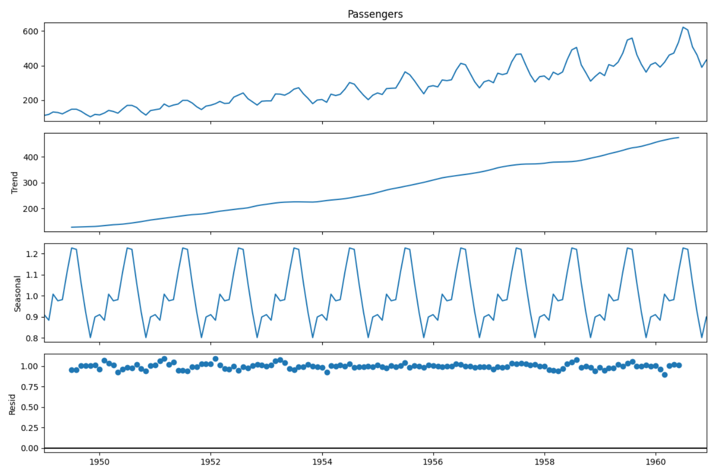


(a) Seasonal decomposition on Airline traffic dataset

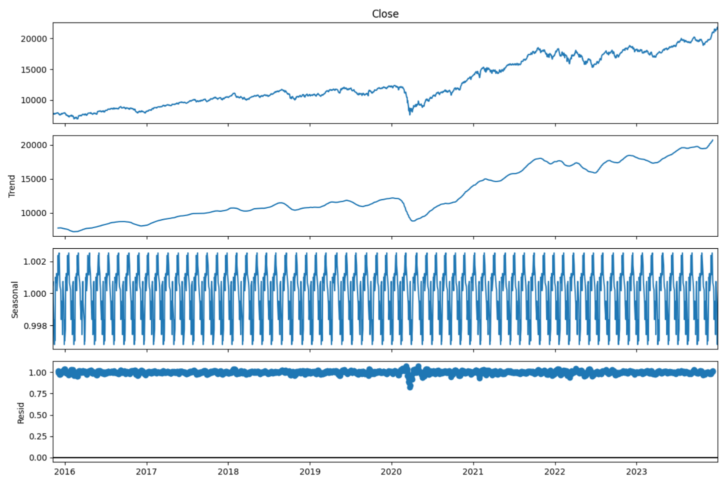


(b) Seasonal decomposition on Nifty50 dataset

Figure 3: Seasonal decomposition using addictive modeling.



(a) Seasonal decomposition on Airline traffic dataset



(b) Seasonal decomposition on Nifty50 dataset

Figure 4: Seasonal decomposition using multiplicative modeling.

y(t) = Trend(t) x Seasonal(t) x Residual(t)

Trend: Captures the long-term movement or direction of the data.
Seasonal: Captures patterns that repeat at fixed intervals (e.g: monthly, quarterly, yearly)
Residual: Captures any randomness or irregularities in the data after removing the trend and seasonality.

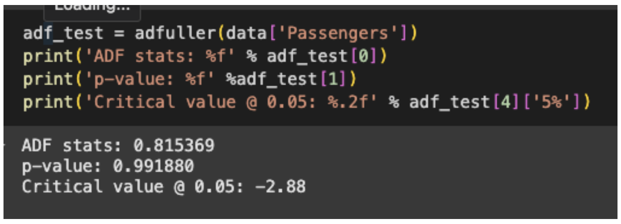
Seasonal decomposition on Airline traffic dataset using addictive model (figure 2.a) highlights stable seasonal patterns and residual with near-constant variance.

Volatility analysis on Nifty50 datasets underscores daily price fluctuations. Rolling averages and detrended series are employed to detect cycles and momentum shifts.

Check for Stationarity

For ARIMA, time series has to be made **stationary** for further analysis**.** For a time-series to be stationary, its statistical properties (mean, variance, etc) will be the same throughout the series, irrespective of the time at which you observe them. A stationary time series will have no long-term predictable patterns such as trends or seasonality. Time plots will show the series to roughly have a horizontal trend with the constant variance.

To check the stationarity of the time series, we will also use the ADF **(Augmented Dickey-Fuller)** test**.** The null hypothesis of the ADF test is that the time series is not stationary.



```
adf_test = adfuller(data['Passengers'])
print('ADF stats: %f' % adf_test[0])
print('p-value: %f' %adf_test[1])
print('Critical value @ 0.05: %.2f' % adf_test[4]['5%'])
```

```
ADF stats: 0.815369
p-value: 0.991880
Critical value @ 0.05: -2.88
```

p-value(0.99) > Critical value(0.05)

So failed to reject null hypothesis(The series is not stationary). Hence, the time series is Non-Stationary.

Methods to De-trend the Time Series

Transformation: Transforming the values using power, square root, log, etc can help to linearize the data. For example, taking a log of the values can help in obtaining a linear trend to the series with an exponential trend. It's called box-cox transformation.

log(exp(x))=x

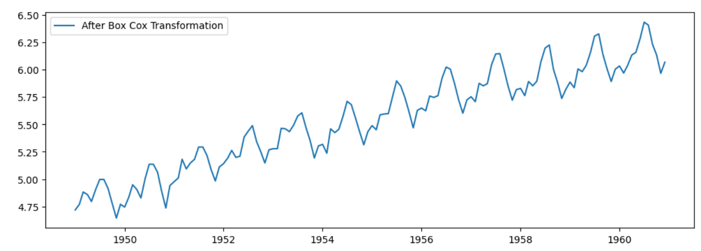


Figure 5: Airline Traffic data after box-cox transformation

The variance become almost constant after the **box-cox transformation**. Both the earlier and later year variance is almost similar. But we can see that still the series has an upward trend. So, the mean is not constant yet. Hence, we need to do Differencing for making the mean constant.

Differencing: A new series is constructed by calculating the value at the current time by differencing the value of actual observation of current time and its previous time.

Value(t) = actual_observation(t) – actual_observation(t-1)

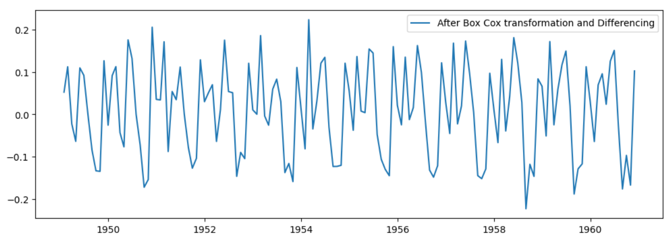
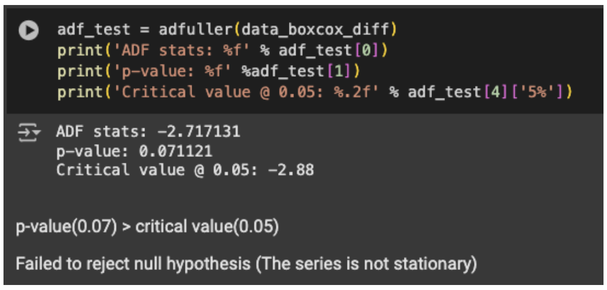


Figure 5: Airline Traffic data after differencing (1)

We can see there is no trend(upward or downward) after differencing on the Box-Cox Transformation. It is a horizontal trend. The mean became constant and the variance is almost constant. Let's test the stationarity again.



```
adf_test = adfuller(data_boxcox_diff)
print('ADF stats: %f' % adf_test[0])
print('p-value: %f' %adf_test[1])
print('Critical value @ 0.05: %.2f' % adf_test[4]['5%'])
```

```
ADF stats: -2.717131
p-value: 0.071121
Critical value @ 0.05: -2.88
```

p-value(0.07) > critical value(0.05)

Failed to reject null hypothesis (The series is not stationary)

Seasonal Differencing: The values of the time series are calculated by differencing between one observation and its previous Nth observation. This can help in removing the trend.

Value(t) = actual_observation(t) – actual_observation(t-N)

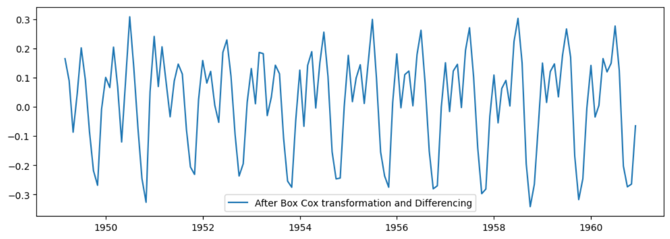
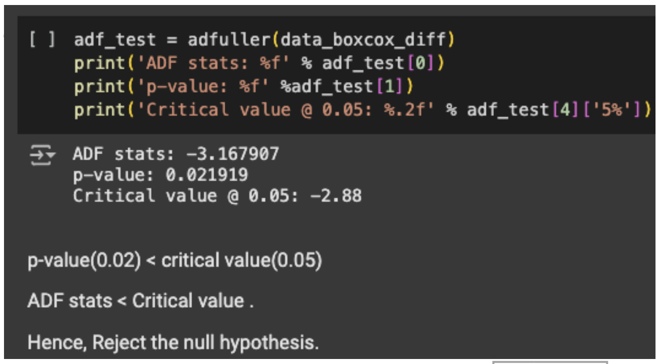


Figure 6: Airline Traffic data after seasonal differencing (2)

Checking the stationarity again:



```
adf_test = adfuller(data_boxcox_diff)
print('ADF stats: %f' % adf_test[0])
print('p-value: %f' %adf_test[1])
print('Critical value @ 0.05: %.2f' % adf_test[4]['5%'])
```

```
ADF stats: -3.167907
p-value: 0.021919
Critical value @ 0.05: -2.88
```

p-value(0.02) < critical value(0.05)

ADF stats < Critical value .

Hence, Reject the null hypothesis.

The series is stationary.
Hence the difference value of ARIMA or SARIMA, d= 2.

Autocrrelation

Autocorrelation Function(ACF): captures both direct and indirect relationship with its lagged values.

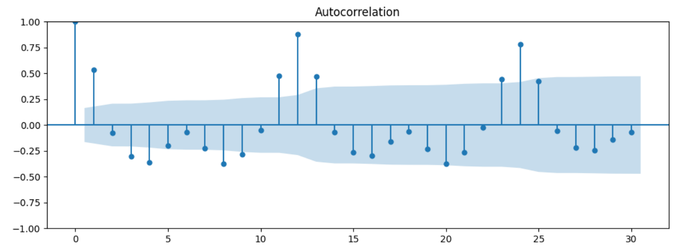


Figure 6: shows the ACF of the Airline traffic data after box-cox transformation and seasonal differencing(2).

The shaded portion is less significant. With lag 0, the series is just correlated with itself. Hence, correlation=1 at lag=0. ACF captures both direct and indirect correlation with the lag time.

Hence the AR lag value for Airline traffic data, p=1.

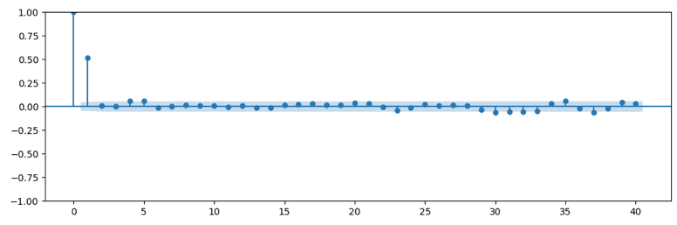


Figure 7: shows the ACF of the Nifty50 dataset after differencing(1).

Hence the AR lag value for Nifty50 dataset, p=1.

Partial Autocorrelation Function(PACF): captures only direct relation by ignoring intermediate relations.

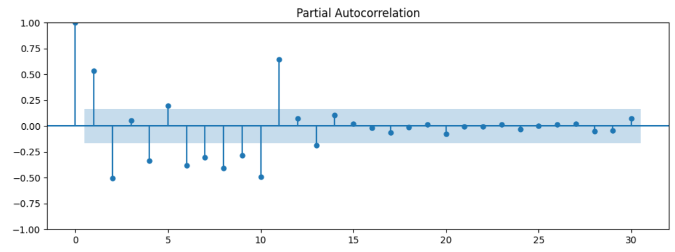


Figure 8: shows the PACF of the Airline traffic data after box-cox transformation and seasonal differencing(2).

We can observe a little change in the correlation plot as because the PACF only captures the direct correlation with the time lag. It bypasses the tile lags in between.

Hence the MA lag value for Airline traffic data, q=2.

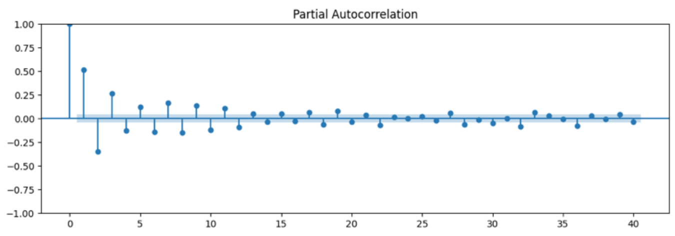


Figure 9: shows the PACF of the Nifty50 dataset after differencing(1).

Hence the MA lag value for Nifty50 dataset, q=22.

III. METHODS AND RESULTS

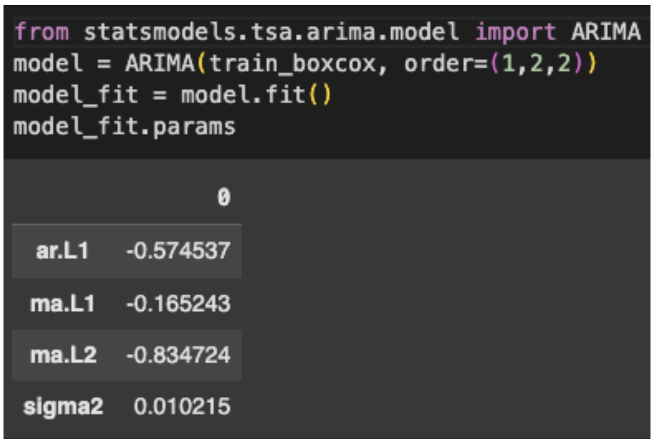
A. ARIMA Modeling

Model Identification

Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) plots guided the selection of ARIMA parameters (p, d, q). As per the data analysis, the parameters for Airline Traffic dataset are p=1, d=2 and q=2. Parameters for the Nifty50 dataset are p=1, d=2 and q=22

Model Fitting

Here is the sample code for model fitting done for Airline Traffic dataset.



```
from statsmodels.tsa.arima.model import ARIMA
model = ARIMA(train_boxcox, order=(1,2,2))
model_fit = model.fit()
model_fit.params
```

```
                0
ar.L1   -0.574537
ma.L1   -0.165243
ma.L2   -0.834724
sigma2   0.010215
```

Performance Evaluation

Root Mean Square Error (RMSE) and Mean Absolute Percentage Error (MAPE) were computed for both datasets to evaluate forecast accuracy.

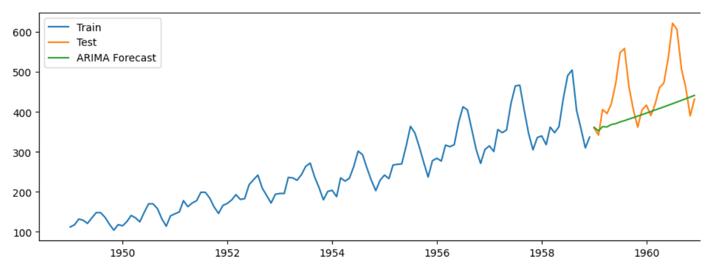


Figure 10: The visualization of the time series for Airline Traffic dataset with both prediction(green) with ARIMA and ground truth(orange)

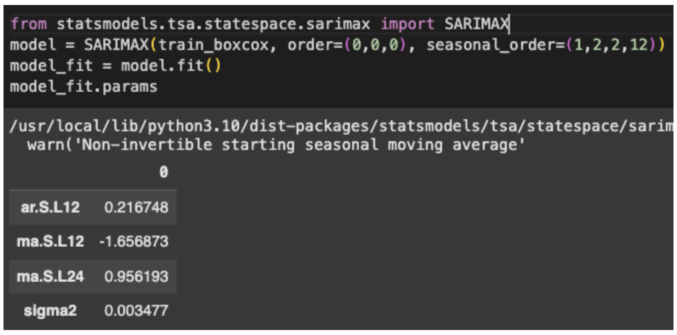
B. SARIMA Modeling

Model Identification

SARIMA is ARIMA with seasonal component added. Seasonal ARIMA was implemented to account for periodicity observed in the datasets. Seasonal parameters (P, D, Q, s) were selected using ACF and seasonal decomposition analysis. As per the data analysis, the parameters for Airline Traffic dataset are P=1, D=2, Q=2 and m=12, because the seasonality is monthly. Parameters for the Nifty50 dataset are P=2, D=1, Q=2 and m=7.

Model Fitting

Here is the sample code for model fitting done for Airline Traffic dataset.



```
from statsmodels.tsa.statespace.sarimax import SARIMAX
model = SARIMAX(train_boxcox, order=(0,0,0), seasonal_order=(1,2,2,12))
model_fit = model.fit()
model_fit.params
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarim
  warn('Non-invertible starting seasonal moving average'
                  0
  ar.S.L12   0.216748
  ma.S.L12  -1.656873
  ma.S.L24   0.956193
  sigma2     0.003477
```

Performance Evaluation

Metrics for SARIMA were compared with ARIMA, with SARIMA demonstrating superior accuracy in capturing seasonality.

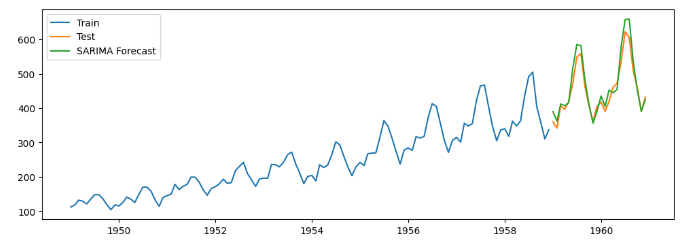


Figure 11: The visualization of the time series for Airline Traffic dataset with both prediction(green) with SARIMA and ground truth(orange)

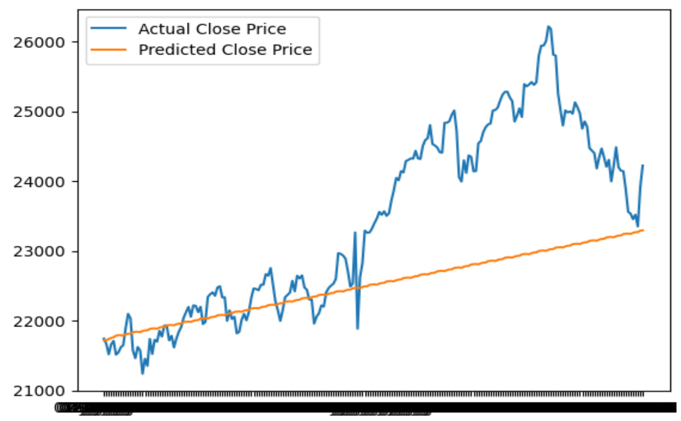


Figure 12: The visualization of the time series for Nifty50 dataset with both prediction(orange) with SARIMA and ground truth(blue)

C. PROPHET Modeling

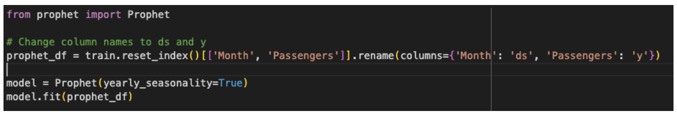
Prophet is a Bayesian forecasting model designed to accommodate trend and seasonality changes. Its intuitive parameterization makes it ideal for business and financial time series.

Prophet having following advantages:

1) Accurate and Fast: It is accurate and can generate results a lot faster as compared to other time series libraries.
2) Reliable: It can accommodate strong seasonal effects in the data, Prophet is robust to missing data and shifts in the trend, and typically handles outlier as well.
3) Domain Knowledge Integration: Forecasting can be made better by adding knowledge expertise like holidays and patterns.

Model Fitting

Prophet takes a dataframe as input with only 2 columns - ds and y. The ds (datestamp) column should be in a specific format, ideally YYYY-MM-DD for a date or YYYY-MM-DD HH:MM:SS for a timestamp to be parsed by Prophet. The y column must be numeric, and represents the variable we wish to forecast. Therefore we must rename the columns before proceeding further



```
from prophet import Prophet

# Change column names to ds and y
prophet_df = train.reset_index()[['Month', 'Passengers']].rename(columns={'Month': 'ds', 'Passengers': 'y'})

model = Prophet(yearly_seasonality=True)
model.fit(prophet_df)
```

Performance Evaluation

Forecast accuracy was evaluated using cross-validation and performance metrics, such as RMSE and MAPE.

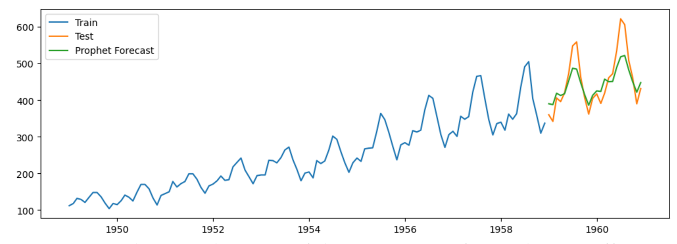


Figure 13: The visualization of the time series for Airline Traffic dataset with both prediction(green) with Prophet and ground truth(orange)

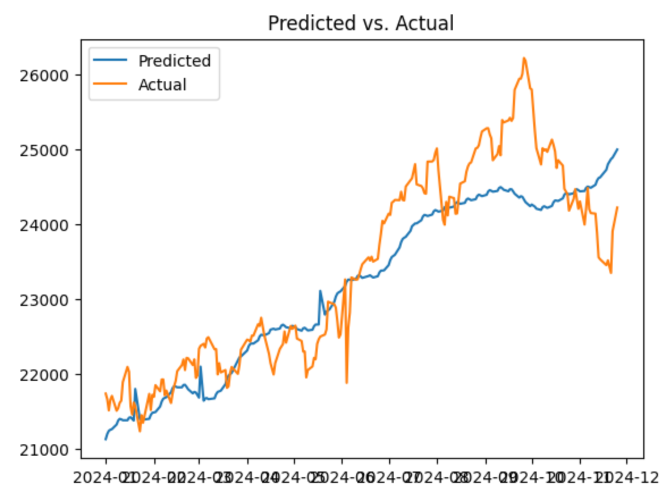


Figure 14: The visualization of the time series for Nifty50 dataset with both prediction(orange) with Prophet and ground truth(blue)

D. **Evaluation Results**

TABLE I
RMSE AND MAPE RESULTS

Dataset	**Model**	**RMSE**	**MAPE**
Airline Traffic	ARIMA	89.02	12.85
Airline Traffic	SARIMA	25.04	4.36
Airline Traffic	PROPHET	40.39	6.54
Nifty50	SARIMA	967.48	53,92
Nifty 50	PROPHET	587.97	44.07

IV.CONCLUSION

This study highlights the efficacy of ARIMA, SARIMA, and Prophet models for time series forecasting across aviation and financial domains. The Box-Cox transformation contributed to stabilized variance, while Prophet's adaptability to trend shifts and seasonality was evident in its accuracy.

Even though deep-learning based approaches such as Prophet can give higher accuracy for non-linear and non-stationary datasets, the statistical approaches such as SARIMA can surpass those models with proper modeling.

V. REFERENCES

[1] https://neptune.ai/blog/arima-sarima-real-world-time-series-forecasting-guide
[2] https://github.com/rohitrnath/Advanced-Time-Series-Analysis
[3] https://towardsdatascience.com/forecasting-time-series-data-stock-price-analysis-324bcc520af5
[4] https://github.com/sahidul-shaikh/time-series-forecasting-airline-passenger-traffic.git
[5] https://towardsdatascience.com/box-cox-transform-for-time-series-cc45f26082c6
[6] https://www.kaggle.com/code/nikhil1e9/stock-price-forecasting-using-prophet