Time Series Forecasting with Facebook Prophet Modeling Seasonal Patterns in Real-World Data

---

Introduction: Why Time Series Forecasting Is Everywhere

Forecasting is at the heart of intelligent decision-making — in retail, finance, transportation, healthcare, and beyond. Whether you're predicting next month's sales, electricity consumption, or disease outbreaks, you're likely working with **time series data**: observations recorded over regular intervals.

Traditional forecasting models like **ARIMA**, **Exponential Smoothing**, and **Holt-Winters** have been used for decades. However, they often assume a level of stationarity or mathematical rigidity that may not suit modern, complex data with **multiple seasonalities**, **holiday effects**, and **structural changes**.

That's where **Facebook Prophet** steps in.

Prophet is an open-source tool developed by Facebook's Core Data Science team, designed to handle business and operational forecasting problems **at scale**. It automatically detects **yearly, weekly, and daily seasonality**, while allowing human experts to incorporate domain knowledge like holidays or product launches.

Taylor, S. J., & Letham, B. (2018). Forecasting at scale. The American Statistician, 72(1), 37–45. https://doi.org/10.1080/00031305.2017.1380080

Real-World Analogy: Forecasting Airline Demand

Imagine you're the operations manager of a major airline. You need to forecast passenger volume for the next six months to:

- Adjust ticket pricing
- Schedule staff
- Manage airport logistics

Passenger data is recorded **monthly**. You notice:

- Higher demand during summer and holidays
- Lower demand during winter
- Occasional dips (e.g., strikes or major events)

You need a forecasting model that:

- Handles **recurring seasonal patterns**
- Adapts to **long-term growth**
- Allows you to manually flag **known disruptions**

Prophet lets you do exactly that — blending the interpretability of traditional time series models with the flexibility of modern machine learning.

? Why and Where to Use Prophet

Prophet is ideal for:

- Business teams that need interpretable models
- Analysts working with **daily/monthly data with trend + seasonality**
- Scenarios involving **holidays, promotions, or special events**
- Quick prototyping of time series solutions

Domain	Example Use Case
Retail	Forecasting daily product sales
Healthcare	Projecting disease incidence rates
Travel	Predicting monthly airline passengers
Finance	Estimating future cashflows
Energy	Modeling electricity demand cycles

Scikit-Prophet Documentation (https://facebook.github.io/prophet/docs/)

Dataset: International Airline Passengers

For this tutorial, I used the classic **International Airline Passengers dataset**, a well-known monthly time series dataset. It includes:

- 144 data points (Jan 1949 – Dec 1960)
- Target: Number of passengers (in thousands)
- Frequency: Monthly

This dataset is widely used in forecasting literature and is perfect for showing:

- Seasonal trends
- Long-term growth
- Prophet's ability to forecast into the future

Up Next: I'll now move to:

- Load and format the dataset
- Perform initial EDA (plot time series, check seasonality)
- Fit a Prophet model

Step 1: Importing Required Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from prophet import Prophet

sns.set(style='whitegrid')
```

In this cell, I import all the necessary Python libraries to perform time series forecasting and visualization. `pandas` is used for data handling and time manipulation, while `numpy` assists with numerical operations (though not directly used in this tutorial, it's often helpful). I also import `matplotlib.pyplot` and `seaborn` for plotting. `seaborn` provides elegant and informative visualizations, and I apply its `whitegrid` style for a clean, readable plot background. Most importantly, I import `Prophet`, a time series forecasting library developed by Facebook's Core Data Science team. It is designed to produce quick, accurate, and interpretable forecasts of time series data, especially for daily or monthly business time series with strong seasonal effects.

Step 2: Load and Prepare the Dataset

```
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/airline-passengers.csv"
df = pd.read_csv(url)
df.columns = ['ds', 'y']
df['ds'] = pd.to_datetime(df['ds'])
df.head()
```

This block loads the classic **International Airline Passengers** dataset from a public GitHub repository. The original file contains two columns: `Month` and `Passengers`. Prophet requires specific column names: `ds` for the date column and `y` for the value we want to forecast. So, I rename these columns accordingly using `df.columns = ['ds', 'y']`. Next, the `ds` column, which stores date strings, is converted to `datetime` format using `pd.to_datetime()` so that Prophet can recognize the time structure. Finally, I use `.head()` to preview the first few rows and

confirm that the data is loaded and formatted correctly. This dataset includes monthly totals of international airline passengers (in thousands) from January 1949 to December 1960.

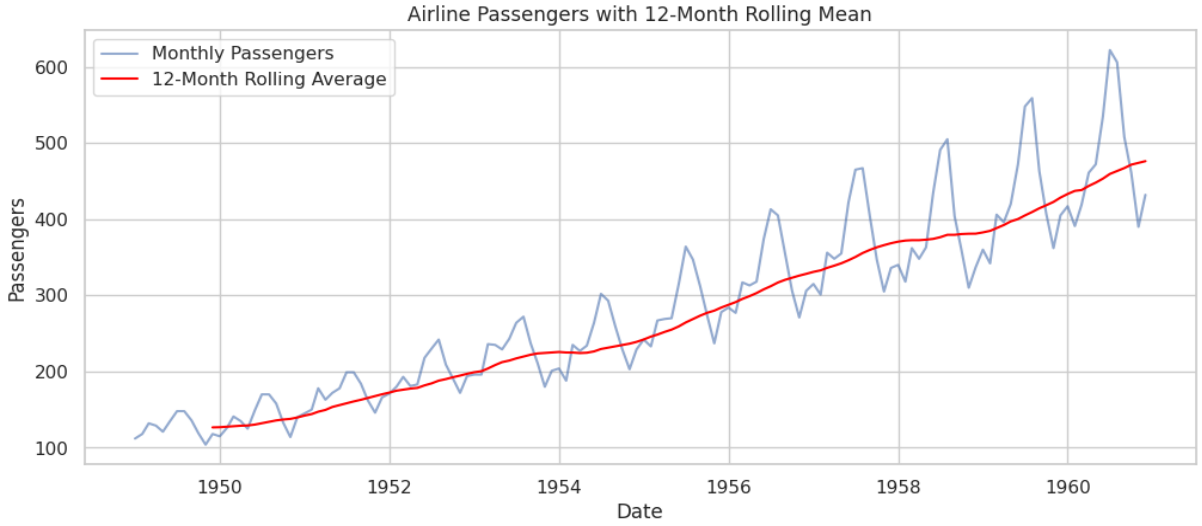
Step 3.1: Plot the Time Series Line Graph

```
# 3.1 Line Plot of Time Series
plt.figure(figsize=(12, 5))
sns.lineplot(x='ds', y='y', data=df, linewidth=2)
plt.title("Monthly International Airline Passengers (1949–1960)")
plt.xlabel("Date")
plt.ylabel("Passengers (in thousands)")
plt.grid(True)
plt.show()
```

This cell visualizes the time series data using a line plot. By mapping the `ds` column (dates) on the x-axis and the `y` column (passenger count) on the y-axis, I create a continuous curve showing how passenger numbers evolved over time. The figure size is expanded for readability. The plot reveals a clear **upward trend** — indicating increasing air travel over time — along with **repeating seasonal patterns**, with peaks typically around mid-year. These patterns make the data ideal for Prophet, which can model both trend and seasonality.

Step 3.2: Rolling Average Plot

```
# 3.2 Rolling Average Plot
df['rolling_mean'] = df['y'].rolling(window=12).mean()
plt.figure(figsize=(12, 5))
plt.plot(df['ds'], df['y'], label='Monthly Passengers', alpha=0.6)
plt.plot(df['ds'], df['rolling_mean'], label='12-Month Rolling Average', color='red')
plt.title("Airline Passengers with 12-Month Rolling Mean")
plt.xlabel("Date")
plt.ylabel("Passengers")
plt.legend()
plt.show()
```

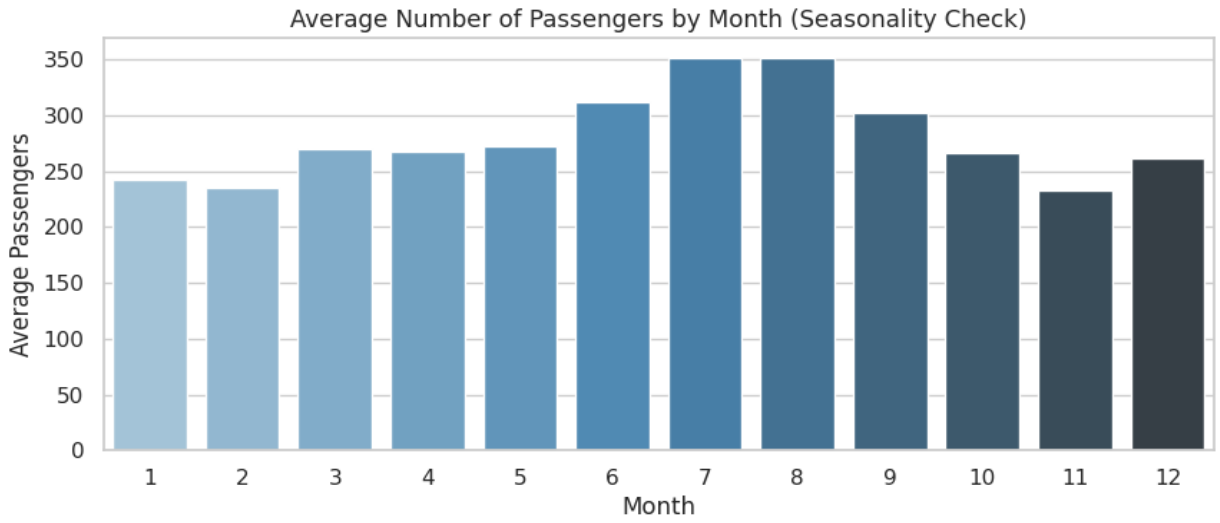


This code adds a **12-month rolling average** to the dataset to smooth out short-term fluctuations and better understand the **long-term trend**. The rolling average is calculated using the `.rolling().mean()` method on the `y` column. I then overlay the rolling average (in red) on top of the original data (in blue), helping students visually distinguish between **short-term noise** and **long-term behavior**. The line plot clearly shows how the passenger count has grown over the years, validating the presence of a trend component that Prophet will later model.

Step 3.3: Monthly Seasonality Bar Chart

```python
# 3.3 Monthly Average Plot
df['month'] = df['ds'].dt.month
monthly_avg = df.groupby('month')['y'].mean()

plt.figure(figsize=(10, 4))
sns.barplot(x=monthly_avg.index, y=monthly_avg.values, palette='Blues_d')
plt.title("Average Number of Passengers by Month (Seasonality Check)")
plt.xlabel("Month")
plt.ylabel("Average Passengers")
plt.show()
```



Here I investigate **seasonality** by calculating the **average number of passengers for each month** across all years. First, I extract the month from the date using `.dt.month`. Then I compute monthly averages using `.groupby('month')['y'].mean()`. The result is a bar plot that illustrates which months typically experience higher or lower demand. The chart shows a clear seasonal pattern — with mid-year months like July and August having the highest averages, and winter months like February showing lower demand. This confirms the presence of yearly seasonality, which Prophet will handle automatically using internal Fourier terms.

Step 4: Initialize and Fit Prophet Model

```
model = Prophet()
model.fit(df)
```

This cell initializes and fits a **Prophet model** to the historical data. By default, Prophet models:

- A **piecewise linear trend** (with automatic changepoint detection)
- **Yearly seasonality**, inferred from the time series frequency
- **Additive components**, unless explicitly changed

The `.fit()` method trains the model on the dataset, learning the shape of the trend and how seasonal cycles repeat. Prophet is particularly useful because it separates each component — trend, seasonality, and noise — and provides interpretable output.

Step 5: Create Future Dates and Predict

```
future = model.make_future_dataframe(periods=36, freq='M')
forecast = model.predict(future)
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
```

175	1963-07-31	649.339420	620.415055	677.890368
176	1963-08-31	604.903070	578.018469	635.314832
177	1963-09-30	566.670215	538.611194	594.814863
178	1963-10-31	538.990739	508.717607	567.535978
179	1963-11-30	568.380421	538.995873	597.020106

I use Prophet's built-in method `.make_future_dataframe()` to extend the time series **36 months into the future**, which equals 3 full years. The frequency is set to `'M'` for monthly data. Then, I use `.predict()` to generate forecasts on this extended timeline. The `forecast` DataFrame includes:

- `yhat`: the predicted passenger value
- `yhat_lower` and `yhat_upper`: lower and upper bounds of the prediction interval

This data will be visualized in the next step to show both the fitted values and the forecast horizon.

Step 6: Visualize the Forecast

```
fig1 = model.plot(forecast)
plt.title("Forecast of Airline Passengers with Prophet")
plt.xlabel("Date")
plt.ylabel("Predicted Passengers")
```

```
plt.show()
```

This plot overlays the original time series with the Prophet forecast. It shows:

- Historical data up to 1960
- Forecasted values for the next 3 years
- A shaded region representing the **95% confidence interval**

This visual allows users to see where the model believes the trend is heading — and how certain it is about that prediction. The widening of the uncertainty bounds over time is typical and reflects increasing prediction difficulty.

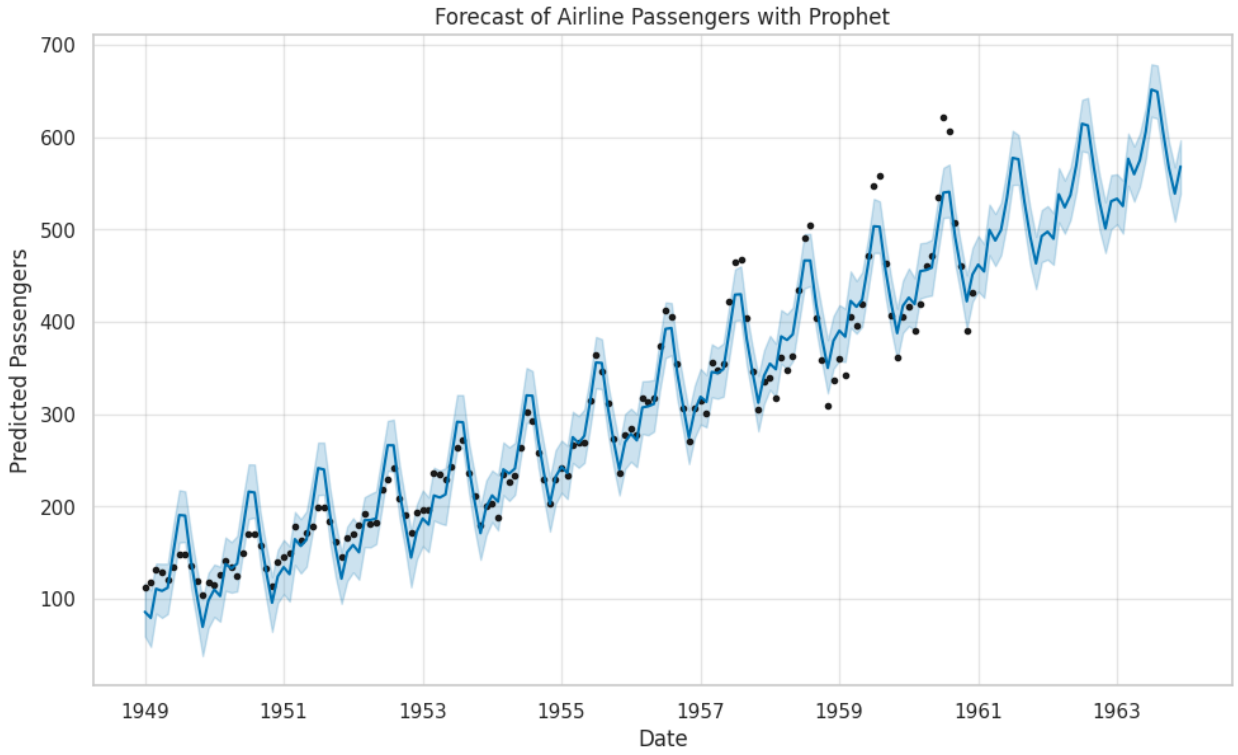


Figure 1 Prophet forecast plot showing projected passenger growth with 95% confidence intervals.

Step 7: Forecast Component Plots

```
fig2 = model.plot_components(forecast)
plt.show()
```

This final plot breaks the forecast into its individual components:

- **Trend**: the long-term pattern of growth
- **Seasonality**: the repeated yearly pattern of peaks and dips

- (Optionally: holidays, if modeled)

These components are easy to explain and interpret, especially for business and operational use. This decomposition aligns well with stakeholder expectations: you can say *"this month is strong due to the seasonal trend"* or *"growth is flattening due to trend saturation."*

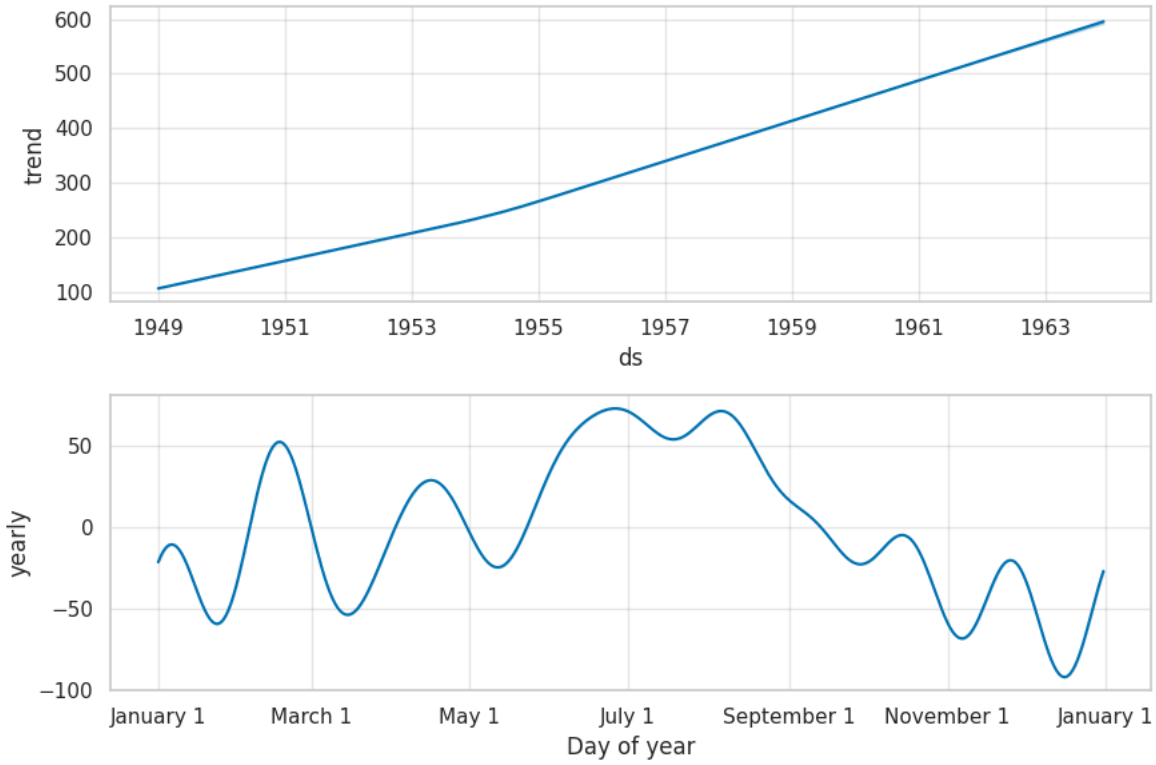


Figure 2 Trend and seasonal decomposition plot, showing individual model components as understood by Prophet.

Summary: Forecasting Trends with Interpretability

In this tutorial, I explored how **Facebook Prophet** can be used for interpretable and flexible time series forecasting using the classic **Airline Passengers dataset**. We began by performing **exploratory data analysis (EDA)** to identify key patterns such as upward trends, seasonal spikes, and non-stationarity — all of which make this dataset a suitable candidate for additive modeling.

Using Prophet, I trained a forecasting model that captured both the **long-term growth** and **repeating seasonal cycles** in passenger demand. I generated future forecasts, visualized them with uncertainty bounds, and decomposed the forecast into its **trend** and **seasonality** components. The model's design — grounded in statistical principles and enriched with machine learning flexibility — allows business and research users to produce accurate forecasts without sacrificing **transparency or interpretability**.

This tutorial also demonstrated best practices in time series formatting, seasonal validation through monthly averages, and clear communication of forecasting results. By integrating EDA,

forecasting, and explainable components, students gain not just a tool, but a **framework for real-world time series thinking**.

"Forecasting is not about predicting the future — it's about understanding the patterns that shape it."

References

Taylor, S. J., & Letham, B. (2018). Forecasting at scale. *The American Statistician, 72*(1), 37–45. https://doi.org/10.1080/00031305.2017.1380080

Brownlee, J. (2021). *Monthly International Airline Passengers Dataset*. Machine Learning Mastery. Retrieved from https://raw.githubusercontent.com/jbrownlee/Datasets/master/airline-passengers.csv

Facebook Core Data Science Team. (2023). *Prophet Official Documentation*. Retrieved from k https://facebook.github.io/prophet/

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research, 12*, 2825–2830. http://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf

Tukey, J. W. (1977). *Exploratory Data Analysis*. Reading, MA: Addison-Wesley.

GitHub Repository:
https://github.com/dheerajn1503/23098026-prophet-airline-forecasting

Notebook (.ipynb):
https://github.com/dheerajn1503/23098026-prophet-airline-forecasting/blob/main/prophet_airline_forecast_tutorial.ipynb

README.md:
https://github.com/dheerajn1503/23098026-prophet-airline-forecasting/blob/main/README.md

requirements.txt:
https://github.com/dheerajn1503/23098026-prophet-airline-forecasting/blob/main/Requirements.txt

Report (.pdf):
https://github.com/dheerajn1503/23098026-prophet-airline-forecasting/blob/main/23098026_Machine_learning_Tutorial.pdf