TIME SERIES ANALYSIS

What is Time Series Analysis?

Time series analysis is a specific way of analyzing a sequence of data points collected over an interval of time. In time series analysis, analysts record data points at consistent intervals over a set period of time rather than just recording the data points intermittently or randomly. However, this type of analysis is not merely the act of collecting data over time.

What sets time series data apart from other data is that the analysis can show how variables change over time. In other words, time is a crucial variable because it shows how the data adjusts over the course of the data points as well as the final results. It provides an additional source of information and a set order of dependencies between the data.

Time series analysis typically requires a large number of data points to ensure consistency and reliability. An extensive data set ensures you have a representative sample size and that analysis can cut through noisy data. It also ensures that any trends or patterns discovered are not outliers and can account for seasonal variance.
Additionally, time series data can be used for forecasting—predicting future data based on historical data.

What is Time Series Forecasting?

Time series forecasting occurs when you make scientific predictions based on historical time stamped data. It involves building models through historical analysis and using them to make observations and drive future strategic decision-making. An important distinction in forecasting is that at the time of the work, the future outcome is completely unavailable and can only be estimated through careful analysis and evidence-based priors.

***Applications**:*

Forecasting has a range of applications in various industries. It has tons of practical applications including: weather forecasting, climate forecasting, economic forecasting, healthcare forecasting, engineering forecasting, finance forecasting, retail forecasting, business forecasting, environmental studies forecasting, social studies forecasting, and more. Basically anyone who has consistent historical data can analyze that data with time series analysis methods and then model, forecasting, and predict.

Why Time Series Analysis?

Time series analysis helps organizations understand the underlying causes of trends or systemic patterns over time. Using data visualizations, business users can see seasonal trends and dig deeper into why these trends occur. With modern analytics platforms, these visualizations can go far beyond line graphs.

When organizations analyze data over consistent intervals, they can also use time series forecasting to predict the likelihood of future events. Time series forecasting is part of predictive analytics. It can show likely changes in the data, like seasonality or cyclic behavior, which provides a better understanding of data variables and helps forecast better.

Applications:

Time series analysis is used for non-stationary data—things that are constantly fluctuating over time or are affected by time. Industries like finance, retail, and economics frequently use time series analysis because currency and sales are always changing. Stock market analysis is an excellent example of time series analysis in action, especially with automated trading algorithms.Examples of time series analysis in action include:

- Quarterly sales
- Stock prices
- Rainfall measurements
- Temperature readings
- Automated stock trading
- Industry forecasts
- Interest rates

Models of time series analysis include:

- **Classification:** Identifies and assigns categories to the data.
- **Curve fitting:** Plots the data along a curve to study the relationships of variables within the data.
- **Descriptive analysis:** Identifies patterns in time series data, like trends, cycles, or seasonal variation.
- **Explanative analysis:** Attempts to understand the data and the relationships within it, as well as cause and effect.
- **Exploratory analysis:** Highlights the main characteristics of the time series

data, usually in a visual format.

- **Forecasting:** Predicts future data. This type is based on historical trends. It uses the historical data as a model for future data, predicting scenarios that could happen along future plot points.
- **Intervention analysis:** Studies how an event can change the data.
- **Segmentation:** Splits the data into segments to show the underlying properties of the source information.

Time Series Data & Classifications

Time series data consists of observations recorded at sequential time intervals. These data points are influenced by time-related patterns and can be classified into different types based on **nature, structure, and properties**.

Classification of Time Series Data

1. Based on Data Characteristics

- **Univariate Time Series:** Single variable recorded over time (e.g., daily temperature, stock price).
- **Multivariate Time Series:** Multiple related variables recorded together (e.g., temperature, humidity, and wind speed over time).

2. Based on Stationarity

- **Stationary Time Series:** Constant mean and variance over time (e.g., white noise).
- **Non-Stationary Time Series:** Changes in mean, variance, or trends over time (e.g., economic growth trends).

3. Based on Patterns & Trends

- **Trend-Based:** Shows increasing or decreasing behavior over time (e.g., population growth).

- **Seasonal:** Repeats patterns at fixed intervals (e.g., electricity demand in summer vs. winter).
- **Cyclical:** Long-term fluctuations that don't follow a fixed period (e.g., business cycles).
- **Irregular (Random):** No apparent pattern, highly unpredictable (e.g., stock market crashes).

Core Principles of Time Series Analysis Data

1. Decomposition

Time series data is often broken down into four components:

$Y(t)=T(t)+S(t)+C(t)+et$

- **Trend (T(t))** – The overall direction of the data (e.g., increasing sales over years).
- **Seasonality (S(t))** – Regular, repeating cycles (e.g., increased ice cream sales in summer).
- **Cyclic Component (C(t))** – Long-term fluctuations that do not follow a fixed period (e.g., economic cycles).
- **Random Noise (e_t)** – Unexplained variations due to randomness.

2. Stationarity & Differencing

- A **stationary time series** has a constant mean and variance over time.
- If the data is non-stationary, **differencing** (subtracting previous values from current values) is used to make it stationary.

3. Autocorrelation & Lag Analysis

- **Autocorrelation** measures how a time series correlates with its past values (lags).
- **Autocorrelation Function (ACF)** and **Partial Autocorrelation Function (PACF)** help determine the relationship between past observations.

4. Forecasting Models

Time series forecasting predicts future values based on past observations. Popular methods include:

- **Statistical Models**: ARIMA, Exponential Smoothing, Prophet.
- **Machine Learning**: Random Forest, XGBoost, LSTMs, Transformers (e.g., Informer).

Time Series technologies:

1. **Prophet (by Facebook/Meta)**
2. **Neural ODE (Ordinary Differential Equations for time series modeling)**

1. PROPHET (FORECASTING BY META)

INTRODUCTION:

Time series forecasting is a crucial aspect of data science and analytics, enabling organizations and researchers to predict future trends based on historical patterns. One of the most user-friendly and powerful forecasting tools to emerge in recent years is **Prophet**, developed by Meta (formerly Facebook). Prophet is designed to simplify the forecasting process by automatically handling issues such as seasonality, holiday effects, and missing data.

WHAT IS PROPHET?

Prophet is an open-source software package for time series forecasting, created by the data science team at Meta (Facebook). It focuses on making forecasts that are highly interpretable while minimizing the amount of data preprocessing required. Prophet is based on the idea of decomposing a time series into three main components:

1. **Trend** – The overall direction of the series over time, which can be linear or logistic.
2. **Seasonality** – Recurrent patterns within a year, day, or other specific cycles.
3. **Holiday Effects** – Special events (like national holidays or sales promotions) that lead to spikes or dips in the time series.

This decomposition approach helps users understand the reasons behind forecasting outcomes, making Prophet a popular tool for business applications such as sales forecasting, capacity planning, and other strategic decisions.

CORE CONCEPTS OF PROPHET

Prophet's methodology can be summarized by the following equation:

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t,$$

where:

- $y(t)$ is the observed value at time t.
- $g(t)$ represents the **trend** function.
- $s(t)$ indicates **seasonality**.
- $h(t)$ denotes the **holiday** or special event component.
- ϵ_t is the **error** or noise term.

Below are the core concepts that govern how these components are modeled in Prophet.

- **Additive Model**

The additive model approach used by Prophet means the final forecast is obtained by summing the individual contributions from trend, seasonality, holiday effects, and random error. This is in contrast to a multiplicative model, where these components would be multiplied together. The additive structure is intuitive and often easier to interpret:

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t.$$

- **Trend Components**

  - **Piecewise Linear Trend**

    Prophet typically uses a **piecewise linear** function to capture the overall trend. It identifies **changepoints**—moments in the historical data where the trend significantly changes (such as a sudden jump or decline)—and fits separate linear models before and after each changepoint.

  - **Logistic Growth Trend**

    For situations where there is a practical or theoretical limit to growth, Prophet also supports a **logistic growth** model. This type of model plateaus after a certain point, mimicking real-world scenarios (e.g., a market might saturate).

- **Seasonality**

Seasonality refers to recurring patterns that repeat over fixed periods, like weekly, monthly, or yearly. Prophet employs **Fourier series** to model these seasonal patterns. Users can specify the frequency (daily, weekly, annual) and how many Fourier terms to include. This approach helps the model learn complex seasonal behaviors without manually coding them.

- **Holiday Effects**

Many time series experience spikes or dips around specific events—such as Black Friday, public holidays, or marketing campaigns. Prophet allows users to specify these holidays or events. The model then treats them as additional regressors, learning the typical impact of these events on the time series.

- **Error or Residual Terms**

No matter how well a model fits, there will always be some level of randomness that it cannot explain. This portion is captured by , often referred to as the error term or residual.

WORKING OF PROPHET:

Here is the general workflow for how Prophet processes time series data:

1. **Data Input**: The user provides a time series with two essential columns: (date or timestamp) and (the value to forecast).
2. **Data Preprocessing**: Prophet can handle missing values and outliers internally, though some basic cleaning is often beneficial.
3. **Trend Fitting**: The model fits a piecewise linear or logistic growth function to capture long-term trends. Changepoints are automatically detected if not manually specified.
4. **Seasonality Modeling**: Using Fourier series, Prophet models weekly, yearly, or any specified seasonal patterns.
5. **Holiday/Events Modeling**: If provided, significant holidays or events are included as additional regressors.
6. **Forecast Generation**: Prophet projects the trend, seasonality, and holiday effects forward to generate future predictions.
7. **Uncertainty Intervals**: Uncertainty is estimated via a Bayesian approach, giving confidence intervals around forecasts.
8. **Validation & Tuning**: Users can adjust parameters (e.g., number of changepoints, seasonality) to refine the forecast.
9. **Visualization**: Prophet includes built-in plotting functions to visualize the forecast, trend, and seasonal components.

STEP BY STEP PROCESS OF PROPHET'S INTERNAL PROCESS:

- Load Time Series
- Preprocess & Clean (Missing Data, Outliers)
- Identify Trend (Linear/Logistic)
- Seasonal Model (Fourier Series)
- Holiday Effects
- Combine Trend, Seasonality, & Holidays
- Forecast Future
- Visualize & Review

ALGORITHMIC DETAILS

Prophet's internal algorithm can be summarized as follows:

1. **Trend Detection**:
    - The model initially sets a prior for the location of possible changepoints based on a user-defined or automatically detected number of potential changepoints.
    - A separate linear (or logistic) function is fit on each segment of data demarcated by these changepoints.

2. **Seasonality Fitting**:
    - For each seasonal frequency (e.g., annual, weekly), Prophet generates Fourier series terms.
    - These terms are included in a regression to capture repetitive patterns.
3. **Holiday Modeling**:
    - For each holiday or special event, the model adds a parameter that shifts the trend up or down.
    - A prior is imposed on these parameters to regularize them.
4. **Parameter Estimation**:
    - Prophet uses an optimization algorithm (often L-BFGS or similar) to find the best-fit parameters for the trend, seasonality, and holiday components.
    - A Bayesian approach with Markov Chain Monte Carlo (MCMC) can be used for more robust uncertainty estimates.
5. **Forecast Generation**:
    - Once the parameters are estimated, the trend is extrapolated into the future.
    - Seasonal effects are applied for each future date.
    - Holiday/event impacts are also added for relevant dates.
    - The forecast is simply the sum of these components.
6. **Uncertainty Intervals**:
    - Uncertainty is primarily derived from the parameter uncertainties.
    - If MCMC is used, the model can produce distribution-based confidence intervals.

REAL LIFE EXAMPLE:

Imagine a food delivery service looking to forecast the number of daily orders in the upcoming 6 months. The historical dataset includes:

- Date/Time: Each day from January 1, 2020, to December 31, 2022.
- Number of Orders: The daily total orders placed.
- Special Events: Public holidays, major sporting events, and marketing promotions.

Steps:

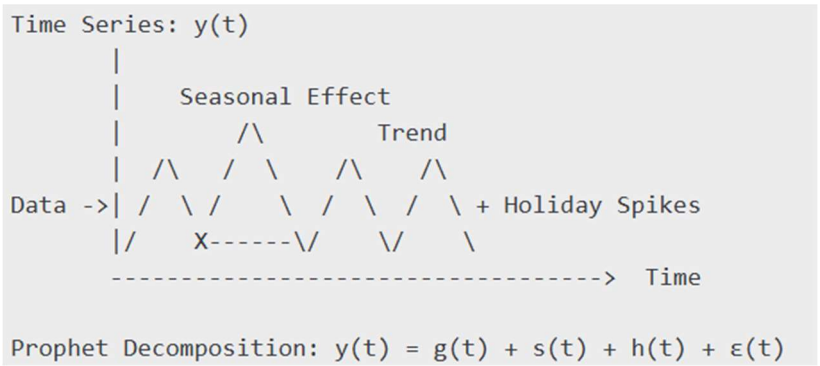
1. **Load the Data:** The team provides the historical orders data to Prophet, with columns named ds (date) and y (daily orders).

2. **Preprocessing:** The team checks for missing dates or improbable outliers, but Prophet can handle small irregularities automatically.
3. **Model Fitting:**
    - Prophet automatically identifies a linear trend in the data.
    - Prophet detects weekly seasonality (weekdays are busier than weekends, for instance) and annual seasonality (holiday spikes in December, for example).
    - Special events like major sporting finals cause distinct spikes in orders.
4. **Forecast Generation**: The model is asked to forecast for 6 months beyond the last known date. It continues the linear trend, includes the weekly pattern, and accounts for any known upcoming holidays or events.
5. **Result:**
    - The company obtains daily order forecasts.
    - Confidence intervals show how uncertain each day's forecast might be.
    - Graphs show the separate contributions of trend, seasonality, and holidays.

PYTHON CODE EXAMPLE:

```python
# 1. Import necessary packages
from prophet import Prophet
import pandas as pd

# 2. Create or load a sample dataset
#    We'll make a small example dataframe with daily data.
#    In a real scenario, you would load this from a CSV file or database.
data = {
    'ds': pd.date_range(start='2021-01-01', periods=10, freq='D'),
    'y': [100, 120, 130, 90, 150, 200, 250, 240, 260, 300]
}
df = pd.DataFrame(data)

# 3. Initialize the Prophet model
model = Prophet(
    weekly_seasonality=True,  # Enable weekly seasonality
    yearly_seasonality=True,  # Enable yearly seasonality
    daily_seasonality=False   # We can turn off daily seasonality if not needed
)

# 4. Fit the model on the historical dataset
model.fit(df)

# 5. Create a dataframe for future predictions
future_dates = model.make_future_dataframe(periods=5)

# 6. Generate the forecast
forecast = model.predict(future_dates)

# 7. View the forecasted data
print(forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']])
```

REPRESENTATION OF HOW PROPHET DECOMPOSES A TIME SERIES:



- Trend (g(t)): Shown as a generally increasing slope.
- Seasonality (s(t)): Represented by the wavy pattern.
- Holiday Effects (h(t)): Sharp spikes or dips.
- Error (ε(t)): Random noise not captured by the main components.

ADVANTAGES AND LIMITATIONS

Advantages

1. **Ease of Use**: Prophet provides a high-level API requiring minimal data preprocessing.
2. **Interpretability**: The decomposition approach makes it clear which factors (trend, seasonality, holidays) influence forecasts.
3. **Robustness**: Automatically handles missing values and outliers.
4. **Built-in Holiday Support**: Users can specify holidays, which often significantly impact real-world time series.
5. **Reasonable Defaults**: Good starting points for changepoint detection and seasonality.

Limitations

1. **Primarily Univariate**: Prophet was mainly designed for forecasting a single time series, though it can handle additional regressors to an extent.
2. **Less Flexible for Complex Dependencies**: It may underperform in cases where deep learning models excel at capturing highly complex relationships.
3. **Limited Handling of Abrupt Regime Changes**: While Prophet includes changepoints, extremely sudden or chaotic changes might not be well modeled.
4. **Computational Overhead**: For very large datasets or extensive parameter tuning, Prophet can still be computationally intensive.

Therefore, Prophet revolutionizes time series forecasting by offering a framework that is both powerful and easy to use for non-experts. Its additive modeling of trend, seasonality, and holidays aligns well with many real-world applications, especially in business and marketing. By automating changepoint detection and offering straightforward parameter adjustments, Prophet helps data analysts generate interpretable, high-quality forecasts with relatively little manual effort.

When forecasting tasks require more advanced modeling of complex, high-dimensional data with intricate dependencies, other machine learning or deep learning approaches might be necessary. However, for many business applications—like sales, website traffic, or resource demand—Prophet strikes an excellent balance between usability, interpretability, and accuracy.

In summary, the key points to remember about Prophet include:

- It decomposes time series into trend, seasonality, and holiday effects.
- It is based on an additive model with piecewise linear or logistic trends.
- It effectively handles missing data and outliers.
- It provides intuitive confidence intervals via a Bayesian framework.

2. Neural ODE (Neural Ordinary Differential Equations)

Introduction

Neural Ordinary Differential Equations (Neural ODEs) are an innovative approach in machine learning that extend traditional deep learning models by treating data transformations as a continuous process over time rather than through a fixed number of layers. This method allows for greater adaptability, efficiency, and robustness in modeling dynamic systems such as financial markets, healthcare, physics-based simulations, and industrial processes.

What are Neural ODEs?

Traditional deep learning models rely on a fixed number of layers to process and transform input data. Neural ODEs differ by modeling transformations continuously over time using an ODE solver, allowing them to dynamically adjust their computations based on the problem being solved.

Unlike conventional neural networks that require manually defining the depth of the model, Neural ODEs determine the depth dynamically using an ODE solver, reducing memory consumption and increasing flexibility.

How Neural ODEs Work in Comparison to Traditional Neural Networks

Neural ODEs extend traditional models by:

- Handling irregularly sampled time-series data effectively.
- Using continuous transformations, which improve generalization and efficiency.
- Reducing memory footprint, as fewer parameters are needed compared to deep networks.
- Allowing adaptive computation, adjusting the number of computations required based on the complexity of the data.

Step-by-Step Working of Neural ODEs

1. **Define the Neural ODE Function:** A neural network is used to approximate a function governing the evolution of the system.
2. **Initialize the System:** The system starts with an initial condition based on observed data.
3. **ODE Solver Integration:** The solver continuously updates the function over time.
4. **Optimization Process:** The model is trained by minimizing prediction errors.
5. **Prediction and Forecasting:** The trained model predicts future values, dynamically adjusting based on new inputs

Algorithmic Details of Neural ODEs

1. Data Collection & Preprocessing

2. Defining the Model & ODE Function
3. Training with Historical Data
4. Testing & Fine-Tuning Model Parameters
5. Making Predictions in Real-time
6. Deployment for Continuous Monitoring

Real Life Example: Heart Rate Monitoring with Neural ODEs

Step 1: Data Collection & Preprocessing

A healthcare provider collects heart rate (HR) data using wearable devices such as smartwatches or ECG monitors. Unlike static datasets, the data arrives continuously over time, with values recorded at irregular intervals:

Time (t)	Heart Rate (HR)	Activity Status
08:00:00	72 BPM	Resting
08:07:00	74 BPM	Light Walking
08:15:00	75 BPM	Sitting
08:29:00	78 BPM	Jogging

Step 2: Model Formulation & Initial Training

A Neural ODE function is trained to predict heart rate variations based on past data and external influences (e.g., stress, activity levels). The function adapts continuously, refining itself with new observations.

Step 3: Advanced Training with Real-time Data

1. **Initial Model Training**: Using historical data.
2. **Integration with Real-time Data Streams**: Wearable devices continuously provide new observations.
3. **Adaptive Learning**: The model updates dynamically, making better predictions over time.

Step 4: Making Predictions & Real-world Impact

The trained model predicts heart rate values based on activity levels and prior trends:

Time (t)	Predicted HR
09:00:00	80 BPM
09:10:00	82 BPM
09:20:00	85 BPM

Step 5: Neural ODEs can detect unusual heart rate patterns and alert doctors or users if irregularities occur, such as:

- Tachycardia (High HR spikes) due to stress or underlying health issues.
- Bradycardia (Unusually low HR) indicating potential health concerns.
- Abnormal oscillations, suggesting irregular heartbeat patterns.

Step 6: Integration with Wearable Health Technology

Neural ODEs can be embedded in smartwatches and medical wearables to:

- Provide real-time health monitoring.
- Detect early signs of cardiovascular diseases.
- Generate automated health reports for medical practitioners.

Industrial and Environmental Applications of Neural ODEs

1. **Industrial Predictive Maintenance**: Predicting machinery failures before they occur.
2. **Environmental Science**: Modeling climate change and pollution patterns.
3. **Autonomous Systems**: Helping self-driving cars adapt in real-time to road conditions.

Neural ODEs vs. Traditional Deep Learning Models

Feature	Neural ODEs	Traditional Deep Learning
Adaptability	High	Fixed-layered approach
Computational Efficiency	Efficient	Expensive
Handling Irregular Data	Excellent	Limited
Interpretability	High	Low

Future of Neural ODEs in AI

Neural ODEs are expected to revolutionize AI, particularly in areas requiring real-time adaptability, scientific simulations, and continuous learning. They have the potential to replace many traditional models where rigid architectures fail to generalize well.

Neural ODEs are a powerful tool that bridges deep learning and dynamic systems modeling. Their ability to process continuous-time data efficiently makes them highly effective for real-world applications. As AI advances, Neural ODEs will likely become a cornerstone of next-generation predictive modeling across industries.

Advantages:

● Memory Efficiency: They can have lower memory footprints compared to deep, discrete-layer networks.

● Adaptive Computation: ODE solvers can adapt their computational effort to the complexity of the input, potentially saving computation time.

● Continuous-Time Modeling: They naturally handle continuous-time dynamics.

● Integration with Physical Models: They can be more easily integrated with existing physical models.

Disadvantages:

● Computational Cost: Solving ODEs can be computationally expensive, especially for complex systems.

● Training Complexity: Training Neural ODEs can be more challenging than training traditional neural networks.

● Solver Dependence: The performance of a Neural ODE can depend on the choice of ODE solver.

● Potential for Instability: Depending on the ODE that is being used, instability can occur.

Prophet vs. Neural ODEs Comparison

Feature	Prophet	Neural ODEs
Purpose	Time-series forecasting with emphasis on seasonality and holidays.	Modeling continuous-time dynamics, learning from continuous data, and representing systems governed by differential equations.
Model Type	Additive regression model.	Neural network architecture based on ordinary differential equations.
Data Type	Primarily univariate time-series data with strong seasonal patterns.	Continuous-time data, irregular time series, data representing dynamic systems.
Approach	Decomposes time series into trend, seasonality, and holiday components.	Models the rate of change of data using a neural network as the derivative function.
Interpretability	Highly interpretable, provides clear components of the forecast.	Can be less interpretable, requires analyzing the learned ODE.
Computational Cost	Relatively low, efficient for many time-series applications.	Can be computationally expensive due to ODE solving.
Handling Irregular Data	Handles missing data, but is best suited for regular time series.	Naturally handles irregular time series.
use case	Business time series forecasting, for things such as sales prediction.	modelling physical systems, or systems with irregular time based data.
Seasonality and Trends	Excels at modeling those features.	can model them, but they are not the primary goal.