

Time Series Forecasting

ABSTRACT

Time series forecasts are used to predict a future value or a classification at a particular point in time. This is a comprehensive documentation of various Time Series Forecasting methods and their use cases.

Jitender Bhatt

Created by

6/30/2022

Published on

Table of Contents

Time Series Data:.....	3
Components of Time Series Data	4
Time Series Analysis Introduction: What it is all about.....	5
Few questions that need to be asked before approaching to any Time-Series forecasting problem:	6
Inputs vs. Outputs.....	6
Endogenous vs. Exogenous.....	6
Unstructured vs. Structured	6
Regression vs. Classification.....	6
Univariate vs. Multivariate.....	6
Single-step vs. Multi-step	7
Static vs. Dynamic	7
Contiguous vs. Dis-contiguous	7
Stationary vs. Non-Stationary	8
Methods to check Stationarity	9
How to handle Non-stationary Data	11
Exploring & Visualizing Time Series.....	12
Seasonality in Time Series with Python	13
1. Definition.....	13
2. Why we decompose the time series.....	13
3.Approach towards finding seasonality	13
Additive vs Multiplicative Seasonality:.....	15
Additive Seasonality	15
Multiplicative Seasonality	16
Time Series forecasting techniques/models:.....	19
Basic time series forecasting techniques	20
Naïve Approach.....	20
Average method.....	21
Drift method.....	22
Moving Average.....	22
Weighted Moving Average (WMA).....	22
Exponential Smoothing Approaches in Time Series Forecasting.....	24
Single Exponential Smoothing.....	24

Holts/Double Exponential Smoothing	25
Holts-Winter/Triple Exponential Smoothing	26
ARIMA Models:	27
1. Autoregressive models(AR):	27
2. Moving Average models(MA):	27
3. ARIMA and SARIMA:	27
Time series Regression Models:	29
1. Simple Regression Models:	29
2. FBProphet:	29
3. NeuralProphet:	30
ML Methods For Time-Series Forecasting	31
Deep Learning Models for Time Series Forecasting	32
Forecasting Error	39
Handling hierarchical and granular time series data.....	41
Determining what to forecast	42
How can we identify if the data is suitable for forecasting?	42
Hierarchical Time Series	43
1) The bottom-up approach	44
2) The top-down approach	45
3) The middle-out approach.....	45
4) The optimal combination/reconciliation approach.....	46
How to Choose the best approach:	47
Limitations of Time Series Analysis/Forecasting.....	48
INDEX of code snippets	49
Document Version	50

Time Series Data:

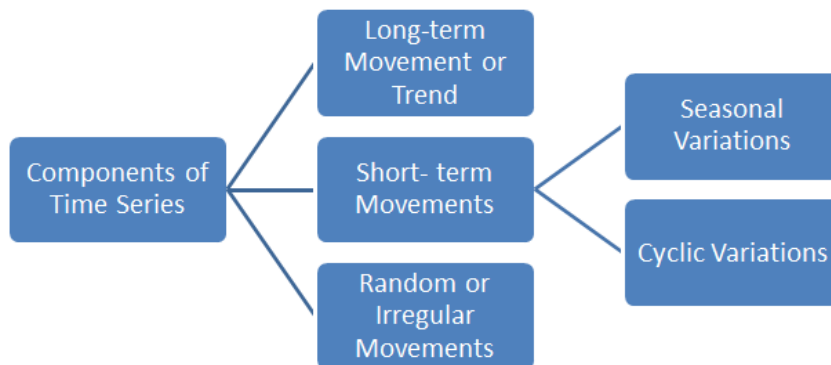
Time series data is a collection of observations obtained through repeated measurements over time.








- Electrical activity in the brain
- Rainfall measurements
- Stock prices
- Number of sunspots
- Annual retail sales
- Monthly subscribers
- Heartbeats per minute

Components of Time Series Data

The main components of Time Series are as Below:

- **Trend:** The trend shows the general tendency of the data to increase or decrease during a long period of time. A trend is a smooth, general, long-term, average tendency. It is not always necessary that the increase or decrease is in the same direction throughout the given period.
- **Seasonality:** A seasonal pattern occurs when a time series is affected by seasonal factors such as the time of the year or the day of the week. Seasonality is always of a fixed and known frequency.
- **Cyclical:** A cycle occurs when the data exhibit rises and falls that are not of a fixed frequency. Usually for businesses it's a four-phase cycle comprising of the phases of prosperity, recession, depression, and recovery. The cyclic variation may be regular or not periodic.
- **Irregularity:** There is another factor which causes the variation in the variable under study. They are not regular variations and are purely random or irregular. These fluctuations are unforeseen, uncontrollable, unpredictable, and are erratic.



	Trend	Seasonality	Cyclical	Irregularity
Time	Fixed Time Interval	Fixed Time Interval	Not Fixed Time Interval	Not Fixed Time Interval
Duration	Long and Short Term	Short Term	Long and Short Term	Regular/Irregular
Visualization				
Nature - I	Gradual	Swings between Up or Down	Repeating Up and Down	Erroneous or High Fluctuation
Nature - II	Upward/Down Trend	Pattern repeatable	No fixed period	Short and Not repeatable
Prediction Capability	Predictable	Predictable	Challenging	Challenging
Market Model				Highly random/Unforeseen Events – along with white noise.

Time Series Analysis Introduction: What it is all about

Time series analysis is a specific way of analyzing a sequence of data points collected over a period. In time series analysis, analysts record data points at regular/irregular intervals over a set.

What sets time series data apart from other data is the relationship of variables with time. Time series analysis typically requires data points ranging over considerable time with respect to the evolving nature of the event for reliable insights. An extensive data set ensures you have a representative sample, and that analysis will be immune to noisy data. It also ensures that any trends or patterns discovered are not outliers/noise and can identify seasonality and cyclicity in the data accurately.

Time series analysis helps us 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, this visualization can go far beyond just line graphs.

Time series forecasting means to forecast or to predict the future value over a period. It entails developing models based on previous data and applying them to make observations and guide future strategic decisions.

The future is forecast or estimated based on what has already happened. Time series adds a time order dependence between observations. This dependence is both a constraint and a structure that provides a source of additional information.

For example, A stock investor analyzed the available data for some stocks and can forecast how a particular stock is performing over in the future. Today's technology allows us to collect massive amounts of data every day and it's easier than ever to gather enough consistent data for comprehensive analysis.

Few questions that need to be asked before approaching to any Time-Series forecasting problem:

Inputs vs. Outputs

What are the inputs and outputs for a forecast?

Inputs: Historical data provided to the model to make a single forecast.

Outputs: Prediction or forecast for a future time step beyond the data provided as input.

Endogenous vs. Exogenous

What are the endogenous and exogenous variables?

Endogenous: Input variables that are influenced by other variables in the system and on which the output variable depends.

Exogenous: Input variables that are not influenced by other variables in the system and on which the output variable depends.

Unstructured vs. Structured

Are the time series variables unstructured or structured?

Unstructured: No obvious systematic time-dependent pattern in a time series variable.

Structured: Systematic time-dependent patterns in a time series variable (e.g. trend and/or seasonality).

It is useful to plot each variable in a time series and inspect the plot looking for possible patterns. A time series for a single variable may not have any obvious pattern. We can think of a series with no pattern as unstructured, as in there is no discernible time-dependent structure.

Alternately, a time series may have obvious patterns, such as a trend or seasonal cycles as structured. We can often simplify the modeling process by identifying and removing the obvious structures from the data, such as an increasing trend or repeating cycle. Some classical methods even allow you to specify parameters to handle these systematic structures directly.

Regression vs. Classification

Are you working on a regression or classification predictive modeling problem?

What are some alternate ways to frame your time series forecasting problem?

Regression: Forecast a numerical quantity.

Classification: Classify as one of two or more labels.

Univariate vs. Multivariate

Are you working on a univariate or multivariate time series problem?

Univariate: One variable measured over time. A single variable measured over time is referred to as a univariate time series. Univariate means one variate or one variable.

Multivariate: Multiple variables measured over time. Multiple variables measured over time is referred to as a multivariate time series: multiple variates or multiple variables.

Single-step vs. Multi-step

Do you require a single-step or a multi-step forecast?

One-Step: Forecast the next time step. A forecast problem that requires a prediction of the next time step is called a one-step forecast model. Whereas a forecast problem that requires a prediction of more than one time step is called a multi-step forecast model.

Multi-Step: Forecast more than one future time steps. The more time steps to be projected into the future, the more challenging the problem given the compounding nature of the uncertainty on each forecasted time step.

Static vs. Dynamic

Do you require a static or a dynamically updated model?

Static: A forecast model is fit once and used to make predictions.

Dynamic: A forecast model is fit on newly available data prior to each prediction.

It is possible to develop a model once and use it repeatedly to make predictions. Given that the model is not updated or changed between forecasts, we can think of this model as being static.

Conversely, we may receive new observations prior to making a subsequent forecast that could be used to create a new model or update the existing model. We can think of developing a new or updated model prior to each forecasts as a dynamic problem.

For example, if the problem requires a forecast at the beginning of the week for the week ahead, we may receive the true observation at the end of the week that we can use to update the model prior to making next weeks forecast. This would be a dynamic model. If we do not get a true observation at the end of the week or we do and choose to not re-fit the model, this would be a static model.

We may prefer a dynamic model, but the constraints of the domain or limitations of a chosen algorithm may impose constraints that make this intractable.

Contiguous vs. Dis-contiguous

Are your observations contiguous or dis-contiguous?

Contiguous: Observations are made uniform over time.

Dis-contiguous: Observations are not uniform over time.

A time series where the observations are uniform over time may be described as contiguous. Many time series problems have contiguous observations, such as one observation each hour, day, month, or year. A time series where the observations are not uniform over time may be described as dis-contiguous.

The lack of uniformity of the observations may be caused by missing or corrupt values. It may also be a feature of the problem where observations are only made available sporadically or at increasingly or decreasingly spaced time intervals.

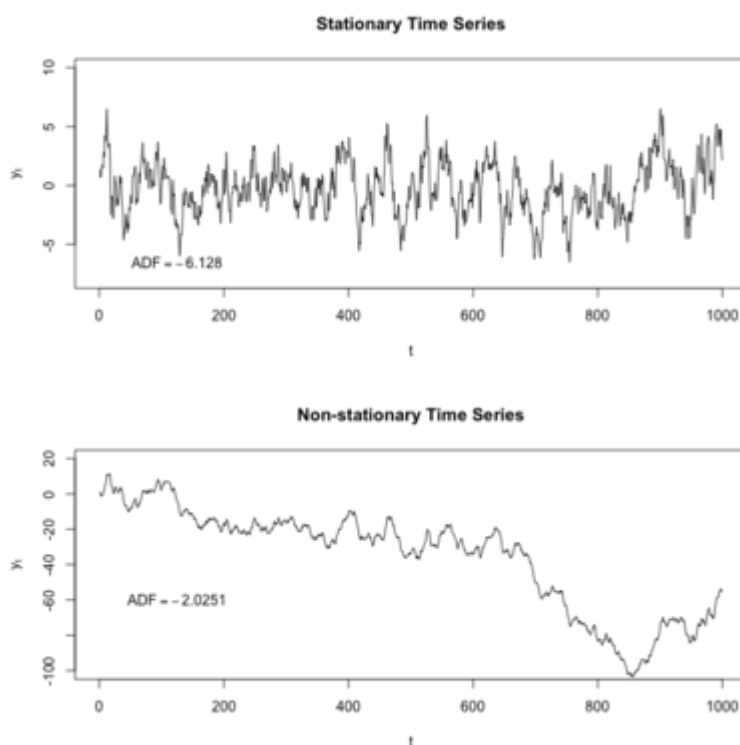
In the case of non-uniform observations, specific data formatting may be required when fitting some models to make the observations uniform over time.

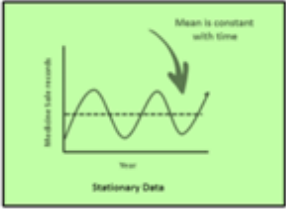
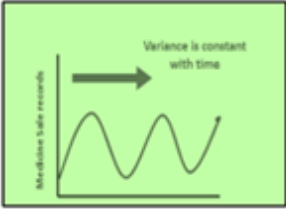
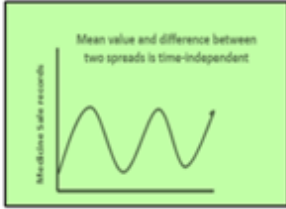
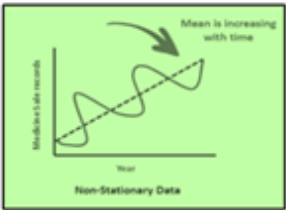

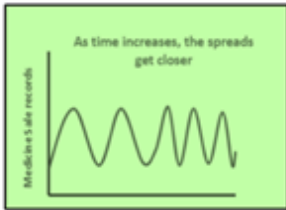
Stationary vs. Non-Stationary

Is your time series data stationary or not?

In the most intuitive sense, stationarity means that the statistical properties of a process generating a time series do not change over time. It does not mean that the series does not change over time, just that the way it changes does not itself change over time.

Classical time series analysis and forecasting methods are concerned with making non-stationary time series data stationary by identifying and removing trends and removing stationary effects.



	MEAN	Variance	Covariance
Stationary			
Non-Stationary			

Methods to check Stationarity

During the Time Series Analysis model preparation workflow, we must access if the given dataset is Stationary or NOT. Using **Statistical and Plots test**.

1. **Statistical Test:** There are two tests available to test if the dataset is Stationary or NOT.
 - Augmented Dickey-Fuller (ADF) Test
 - Kwiatkowski-Phillips-Schmidt-Shin (KPSS) Test
 1. **Augmented Dickey-Fuller (ADF) Test or Unit Root Test:** The ADF test is the most popular statistical test and with the following assumptions.
 - Null Hypothesis (H0): Series is non-stationary
 - Alternate Hypothesis (HA): Series is stationary
 - p-value > 0.05 Fail to reject (H0)
 - p-value ≤ 0.05 Accept (H1)
 2. **Kwiatkowski-Phillips-Schmidt-Shin (KPSS):** these tests are used for testing a NULL Hypothesis (H0), that will perceive the time-series, as stationary around a deterministic trend against the alternative of a unit root. Since TSA looking for Stationary Data for its further analysis, we must make sure that the dataset should be stationary.

Once we know the patterns, trends, cycles and seasonality, we can check if the series is stationary or not. Dickey – Fuller is one of the popular tests to check it.

A stationary series is one in which the properties – mean, variance and covariance, do not vary with time.

The null (H0) and alternate hypothesis (H1) of this test are:

H0: The series has a unit root (value of $\alpha = 1$), the series is non-stationary.

H1: The series has no unit root, the series is stationary.

If we cannot reject the null hypothesis, we can say that the series is not stationary, and if we do, it is stationary. If the series is not stationary it means it can be linear or constant different.

Let's run the **ADF test**!

You can find the code snippet in the location : [Stationarity_Test.ipynb](#)

```
Dickey-Fuller Test Sonucu :
test Instatistigi      3.414756
p-value                1.000000
#Lags                  12.000000
Gozlem Sayisi          76.000000
Kritik Deger (1%)      -3.519481
Kritik Deger (5%)      -2.900395
Kritik Deger (10%)     -2.587498
dtype: float64
```

The ADF tests gives the results of test statistic, p value and the critical value at 1%, 5%, and 10% confidence intervals.

If test statistic < critical value

then we can reject the null hypothesis.

If test statistic > critical value

then we fail to reject the null hypothesis (which means the series is not stationary).

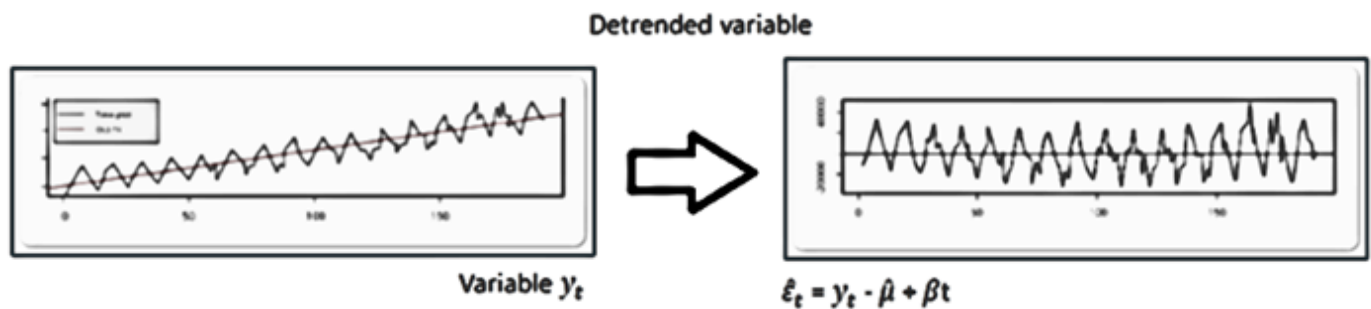
According to the Stationary Test result, since the p-value is greater than 0.05 ($p > 0.05$), H_0 cannot be rejected (i.e. acceptance), the data has a unit root and the series is not stationary (at the same time, the test statistic is greater than the critical value).

How to handle Non-stationary Data

Let's see how to convert non-stationary into stationary data for effective time series modeling. There are 3 major methods available for this conversion.

- Detrending
- Differencing
- Transformation

1. **Detrending:** It involves removing the trend effects from the given dataset and showing only the differences in values from the trend. it always allows the cyclical patterns to be identified.

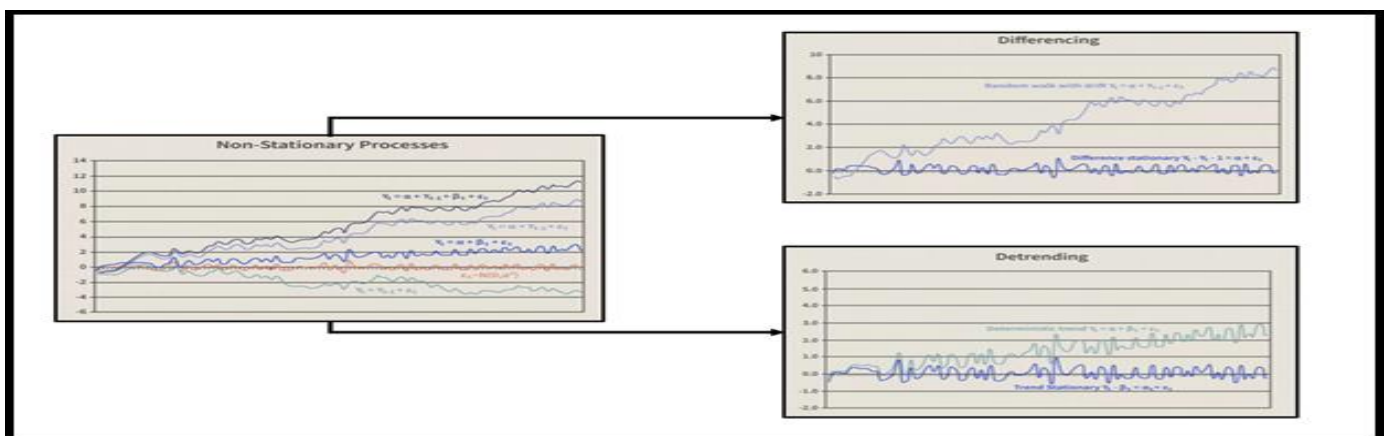


2. **Differencing:** This is a simple transformation of the series into a new time series, which we use to remove the series dependence on time and stabilize the mean of the time series, so trend and seasonality are reduced during this transformation.

$$Y_t = Y_t - Y_{t-1}$$

Y_t = Value with time

Detrending and Differencing extractions



3. **Transformation:** This includes three different methods they are Power Transform, Square Root, and Log Transfer., most used one is Log Transfer.

Exploring & Visualizing Time Series

Time series forecasting is performed in nearly every organization that works with quantifiable data. Retail stores forecast sales. Energy companies forecast reserves, production, demand, and prices. Educational institutions forecast enrollment. Governments forecast tax receipts and spending. International financial organizations forecast inflation and economic activity. The list is long, but the point is short - forecasting is a fundamental analytic process in every organization. The purpose of this tutorial is to get you started doing some fundamental time series exploration and visualization.

Following steps are involved while exploring and visualizing Time Series Data:

1. **Creating time series objects:** Convert your data to a timeseries object for time series analysis.
2. **Time series plots:** Basic visualization of timeseries objects and differentiating trends, seasonality, and cycle variation.
3. **Seasonal plots:** Plotting seasonality trends in time series data.
4. **Autocorrelation of time series:** Computing and visualizing autocorrelation.
5. **White noise:** Differentiating signal from the noise.

Seasonality in Time Series with Python

1. Definition

Seasonality is a characteristic of a time series in which the data experiences regular and predictable changes, such as weekly and monthly. Seasonal behavior is different from cyclic because seasonality is always of a fixed and known period while cyclicity does not have fixed period, e.g., business cycle.

Seasonality can be used to help analyze stocks and economic trends. For instance, companies can use seasonality to help determine certain business decisions such as inventories and staffing.

2. Why we decompose the time series

In time series analysis and forecasting, we usually think that the data is a combination of trend, seasonality and noise and we could form a forecasting model by capturing the best of these components. Typically, there are two decomposition models for time series: additive and multiplicative. It is believed that the additive model is useful when the seasonal variation is relatively constant over time, whereas the multiplicative model is useful when the seasonal variation increases over time.

The real-world problems are messy and noisy, such as the trend is not monotonous and the real model could have both additive and multiplicative components. Nevertheless, these decomposition models provide us a structured and simple way to analyze and forecast the data. Hence, to identify the seasonality in a time series could help you build a better model. This can happen in the following ways:

- Data cleaning: removing the seasonal component will give you a clearer relationship between input and output.
- Interpretability: provide more information of time series.

In `seasonal_decompose()`, we provide two phases of functions:

A. Seasonality test

`seasonal_decompose()` tests whether a time series has a seasonality or not by removing the trend and identifying the seasonality by calculating the autocorrelation (acf). The output includes the number of period, type of model (additive/multiplicative) and acf of the period.

B. Seasonal decomposition

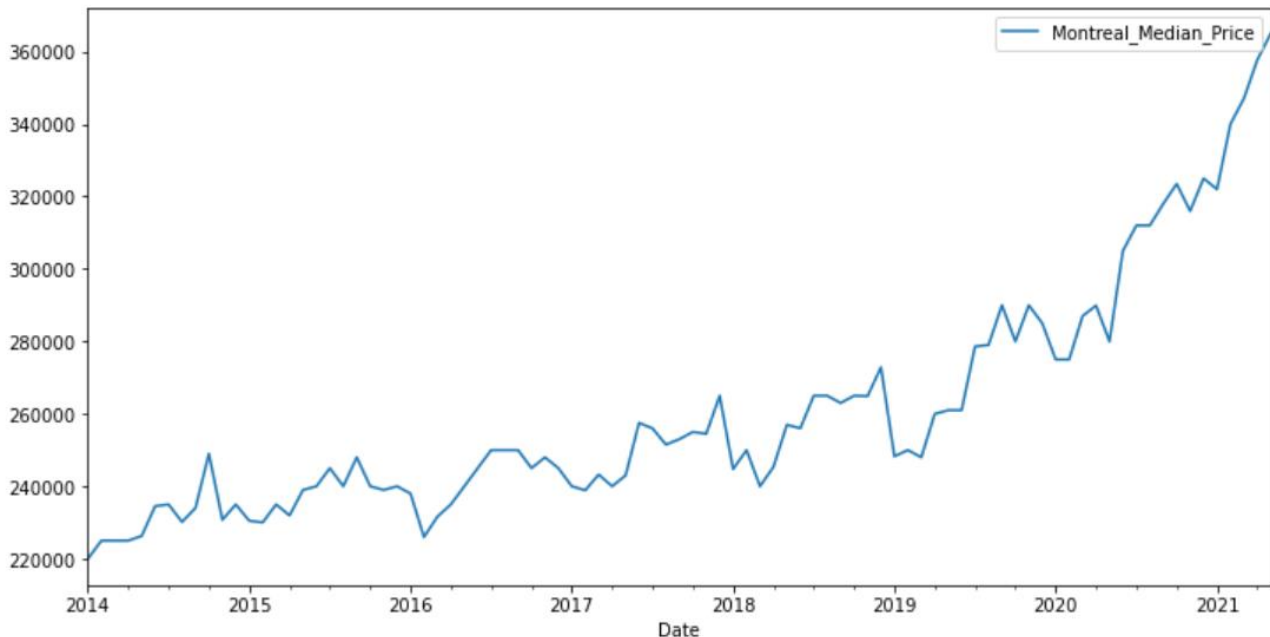
Based on the model structured in the seasonality test phase, the components of trend, seasonality and random noise are determined.

3. Approach towards finding seasonality

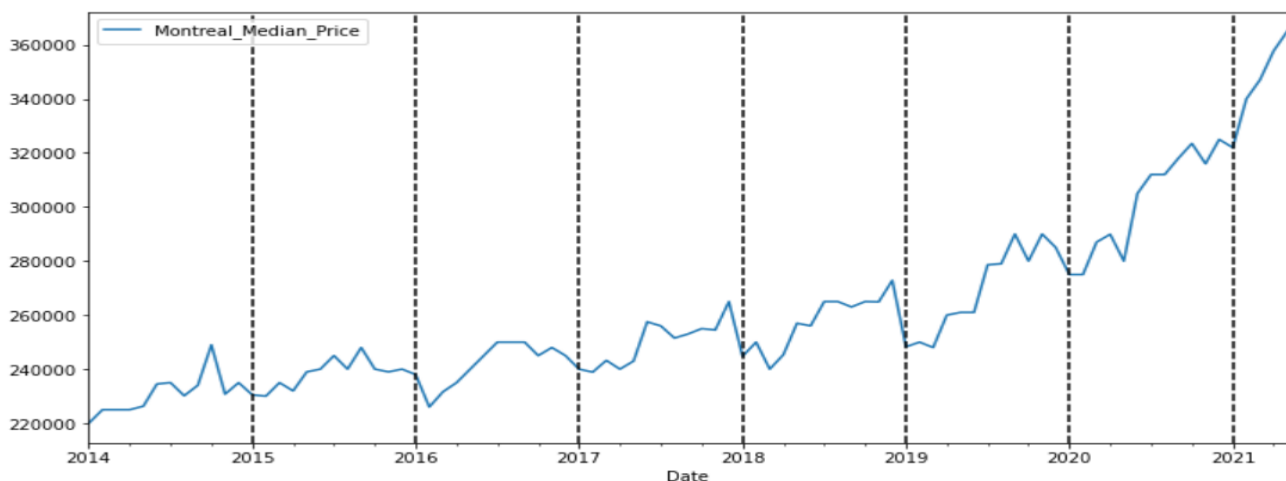
We are using data published by the Quebec Professional Association of Real Estate Brokers. The association publishes monthly real estate stats. For convenience, I've put the monthly median condo prices for the Province of Quebec and the Montreal Metropolitan Area into a CSV file, available here: [quebec_real_estate.csv](#)

The quickest way to get an idea of whether your data has a seasonal trend is by plotting it. Let's see what we get when we plot the median house price in Montreal by month.

You can find the code to generate and how to deal with seasonality in the location: [Timeseries_Seasonality_test.ipynb](#)



We can see from this plot that the prices seem to dip around the new year and peak a few months before, around late summer. Let's dive a little further into this by plotting a vertical line for January of every year.

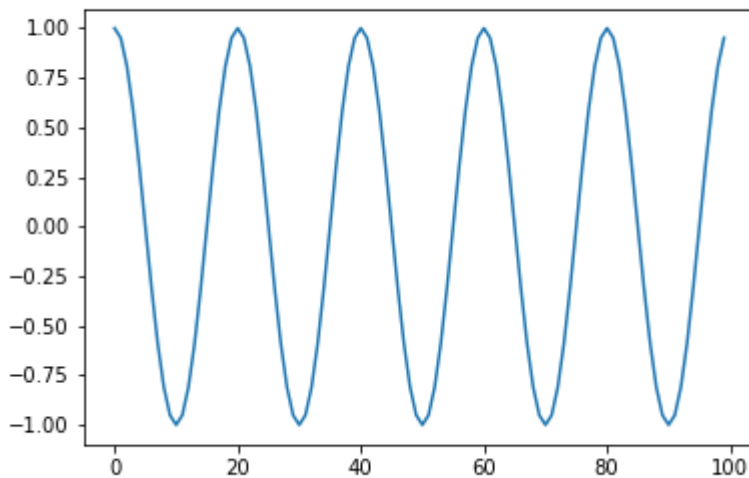


It seems like there's a trend here. In this case, it appears the seasonality has a period of one year. Next, we'll look into a tool we can use to further examine the seasonality and break down our time series into its trend, seasonal, and residual components.

Before we go ahead, let's understand the difference between an additive and a multiplicative seasonality.

Additive vs Multiplicative Seasonality:

There are two types of seasonality that you may come across when analyzing time-series data. To understand the difference between them let's look at a standard time series with perfect seasonality, a cosine wave:



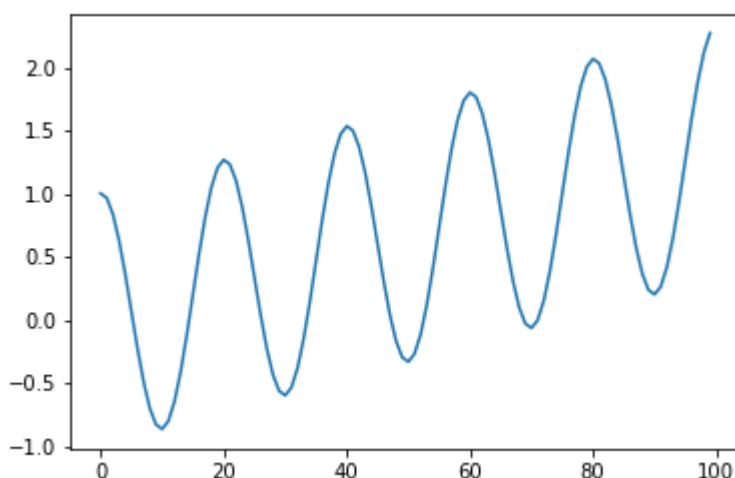
Sine Wave Plot

We can clearly see that the period of the wave is 20 and the amplitude (distance from the center line to the top of a crest or to the bottom of a trough) is 1 and remains constant.

Additive Seasonality

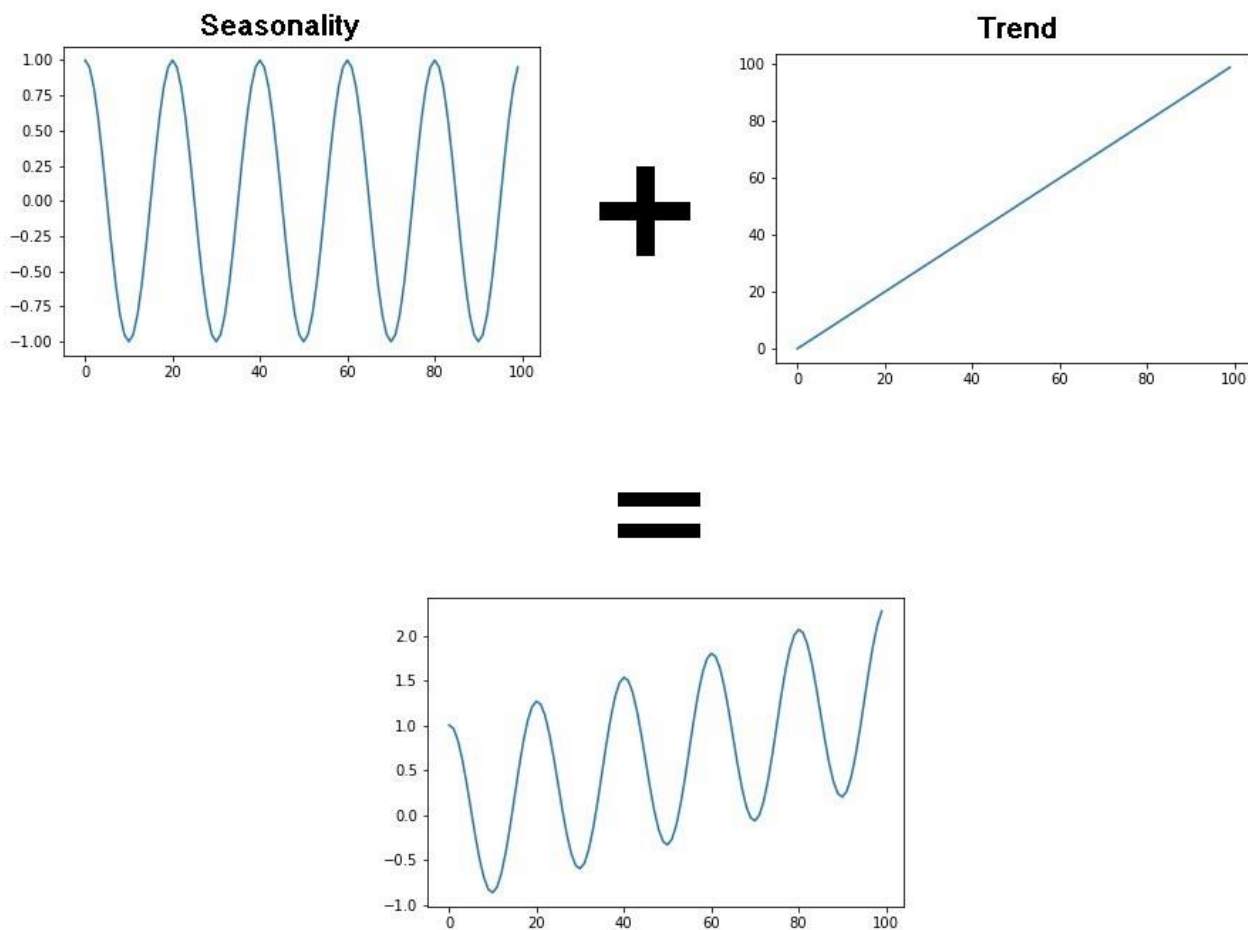
It's rare for actual time series to have constant crest and trough values and instead, we typically see some kind of general trend like an increase or a decrease over time. In our sales price plot, for example, the median price tends to go up over time.

If the amplitude of our seasonality tends to remain the same, then we have what's called an additive seasonality. Below is an example of an additive seasonality.



Additive seasonality

A great way to think about it is by imagining we took our standard cosine wave and simply added a trend to it:



We can even think of our basic cosine model from earlier as an additive model with a constant trend! We can model additive time series using the following simple equation:

$$Y[t] = T[t] + S[t] + e[t]$$

$Y[t]$: Our time-series function

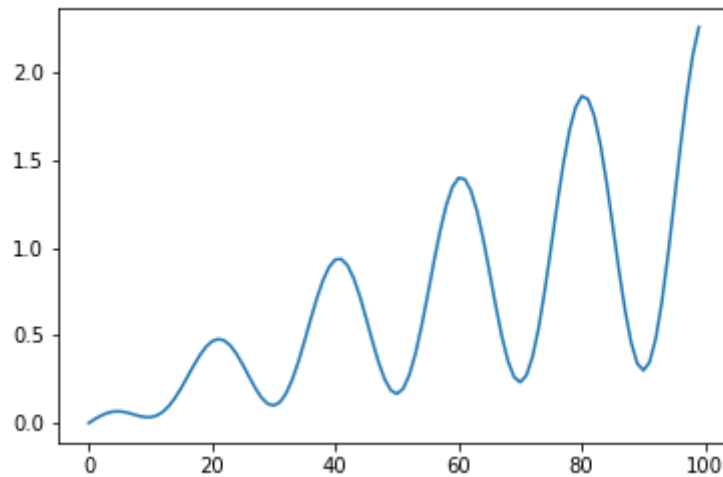
$T[t]$: Trend (general tendency to move up or down)

$S[t]$: Seasonality (cyclic pattern occurring at regular intervals)

$e[t]$: Residual (random noise in the data that isn't accounted for in the trend or seasonality)

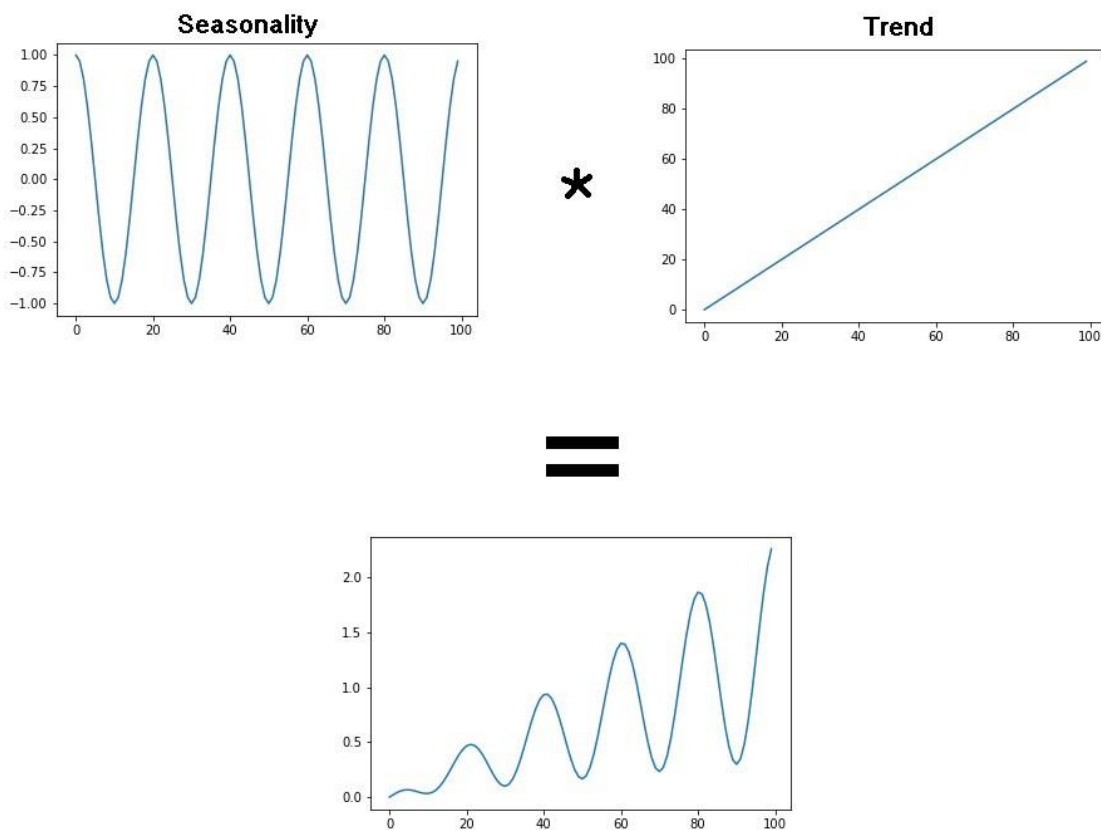
Multiplicative Seasonality

The other type of seasonality that you may encounter in your time-series data is multiplicative. In this type, the amplitude of our seasonality becomes larger or smaller based on the trend. An example of multiplicative seasonality is given below.



Multiplicative seasonality

We can apply a similar train of thought as we used with our additive model and imagine that we took our cosine wave but instead of adding the trend, we multiplied it (hence the name multiplicative seasonality):



Multiplicative seasonality

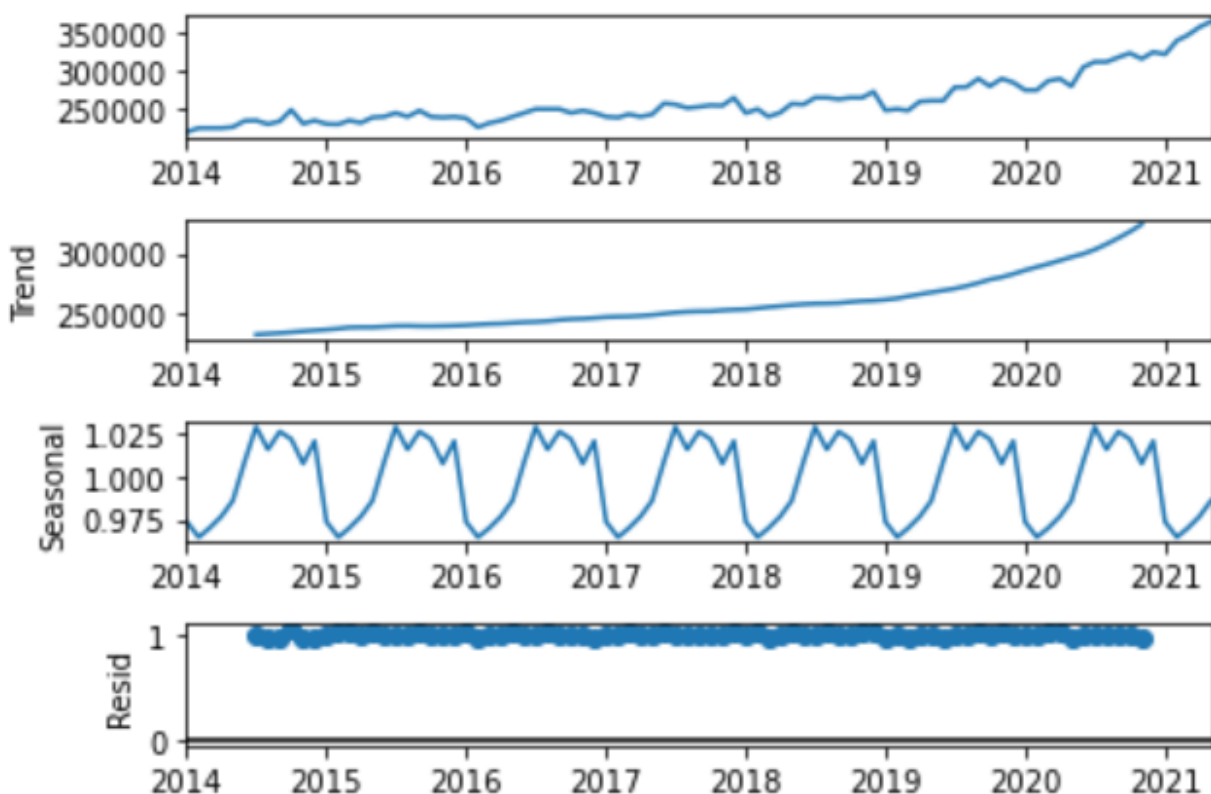
We can model this with a similar equation as our additive model by just swapping the additions for multiplications.

$$Y[t] = T[t] * S[t] * e[t]$$

Now as we are clear about the types of seasonality we can face, let's get back to our data set and try to apply decomposition on it, let's look at how we can break down our real estate time series into its trend, seasonality, and residual components.

We'll be using the `seasonal_decompose` model from the `statsmodels` library.

The `seasonal_decompose` model requires you to **select a model type** for the seasonality (**additive or multiplicative**). We'll select a **multiplicative model** since it would appear the **amplitude of the cycles is increasing with time**. This would make sense since a large factor for housing prices is lending rates which are done as a percentage of the price.



The **trend**, **seasonal**, and **residual** components are returned as pandas series so you can plot them by calling their `plot()` methods or perform further analysis on them. One thing that may be useful is measuring their correlation to outside factors. For example, you could measure the correlation between the trend and mortgage rates, or you could see if there's a strong correlation between the residual and the number of new babies born in the city.

Time Series forecasting techniques/models:

Time series models are used to forecast events based on verified historical data. Common types include moving average, exponential smoothing, ARIMA etc. Not all models will yield the same results for the same dataset, so it's critical to determine which one works best based on the individual time series.

When forecasting, it is important to understand your goals. To narrow down the specifics of your predictive modeling problem, ask questions about:

- a) **Volume of data available** – more data is often more helpful, offering greater opportunity for exploratory data analysis, model testing and tuning, and model fidelity.
- b) **Required time horizon of predictions** – shorter time horizons are often easier to predict – with higher confidence – than longer ones.
- c) **Forecast update frequency** – Forecasts might need to be updated frequently over time or might need to be made once and remain static (updating forecasts as new information becomes available often results in more accurate predictions).
- d) **Forecast temporal frequency** – Often forecasts can be made at lower or higher frequencies, which allows harnessing down sampling and up-sampling of data (this in turn can offer benefits while modeling).

There are few categories in which we can divide our time series models into:

1. Basic time series forecasting techniques
2. Simple Forecasting Model
3. Regression Models
4. Exponential Smoothing Models
5. ARIMA Models

Basic time series forecasting techniques

Naïve Approach

This is one of the simplest methods. It says that the forecast for any period equals the last observed value. If the time series data contain seasonality, it'll be better to take forecasts equal to the value from last season. This is often used for bench-marking purposes.

$$\hat{y}_{T+h|T} = y_T$$

Naïve Approach can be handled in 3 ways:

I. **Stable series:** If the variation in the given series is around the average. In this case the last data point becomes the forecast for the next period.

II. **Seasonal Series:** In a series containing seasonality. The forecast for the “season” is equal to the value of the series last “Season”.

A similar method is useful for highly seasonal data. In this case, we set each forecast to be equal to the last observed value from the same season (e.g., the same month of the previous year). Formally, the forecast for time $T+h$ is written as

$$\hat{y}_{T+h|T} = y_{T+h-m(k+1)},$$

where m = the seasonal period, and k is the integer part of $(h-1)/m$ (i.e., the number of complete years in the forecast period prior to time $T+h$). This looks more complicated than it really is. For example, with monthly data, the forecast for all future February values is equal to the last observed February value. With quarterly data, the forecast of all future Q2 values is equal to the last observed Q2 value (where Q2 means the second quarter). Similar rules apply for other months and quarters, and for other seasonal periods.

III. **Trend:** In a series containing Trend. The forecast is equal to the last value of the series plus or minus the difference between the last two values of the series.

Example: The following data is for the daily sales in dozens for each of the three products of a bakery. Determine the forecast for the 12th day using the Naïve Approach.

Day	Cookies	Muffins	Cakes
1	17	20	12
2	18	19	15
3	22	21	10
4	20	16	25
5	19	18	15
6	16	15	13

7	15	20	12
8	20	19	28
9	21	20	18
10	18	22	16
11	17	23	14

Note: In the Naïve approach we must observe the given dataset carefully.

- Looking at the data for cookies, we can see that the values are relatively close. So, we can assume the series is stable.

As the series is stable, the last day value (11th day) for cookies will be the forecast for the 12th day.

i.e, F_{12} (Forecast for 12th day for cookies) = D_{11} (Demand for 11th day for cookies) = 17 dozens

- Observe the data for muffins. The values are similar with cookies but there is a distinction, the last four values are increasing. So, we assume that the series has trend.

i.e, F_{12} (Forecast for 12th day for Muffins) = D_{11} (Demand for 11th day for Muffins) + [D_{11} (Demand for 11th day for Muffins) - D_{10} (Demand for 10th day for Muffins)]
 = 23 + (23-22) = 24 dozens

- Looking at the data for cakes. The values show “Peak” behavior every four days. We assume that the series is seasonal.
 Day 4 and Day 8 are seasonal.
 F_{12} (Forecast for 12th day for Cakes) = F_8 (Demand for 8th day for Cakes, last seasonality)

Average method

Here, the forecasts of all future values are equal to the average (or “mean”) of the historical data. If we let the historical data be denoted by y_1, \dots, y_t then we can write the forecasts as

$$\hat{y}_{T+h|T} = \bar{y} = (y_1 + \dots + y_T)/T.$$

The notation $\hat{y}_{T+h|T}$ is a short-hand for the estimate of y_{T+h} based on the data y_1, \dots, y_T .

Drift method

A variation on the naïve method is to allow the forecasts to increase or decrease over time, where the amount of change over time (called the **drift**) is set to be the average change seen in the historical data. Thus, the forecast for time $T+h$ is given by:

$$\hat{y}_{T+h|T} = y_T + \frac{h}{T-1} \sum_{t=2}^T (y_t - y_{t-1}) = y_T + h \left(\frac{y_T - y_1}{T-1} \right).$$

This is equivalent to drawing a line between the first and last observations and extrapolating it into the future.

Moving Average

The moving average is a statistical method used for forecasting long-term trends. The technique represents taking an average of a set of numbers in a given range while moving the range. For example, let's say the sales figure of 6 years from 2000 to 2005 is given and it is required to calculate the moving average taking three years at a time. In order to calculate the moving average, one would take an average of 2000-2002, 2001-2003, 2002-2004, 2003-2005, and 2004-2006. Let's understand it with example.

Year	Sales	Moving Average
2000	4	NAN
2001	7	NAN
2002	4	NAN
2003	9	5
2004	7	6.67
2005	10	6.67

Weighted Moving Average (WMA)

- The weighted moving average (WMA) is a technical indicator that assigns a greater weighting to the most recent data points, and less weighting to data points in the distant past.
- The WMA is obtained by multiplying each number in the data set by a predetermined weight and summing up the resulting values.
- Traders use weighting moving average to generate trade signals, to indicate when to buy or sell stocks.

Example: Assume that the number of periods is 10, and we want a weighted moving average of four stock prices of \$70, \$66, \$68, and \$69, with the first price being the most recent.

Using the information given, the most recent weighting will be 4/10, the previous period before that will be 3/10, and the next period before that will be 2/10, and the initial period weighting will be 1/10.

The weighting average for the four different prices will be calculated as follows:

$$\text{WMA} = [70 \times (4/10)] + [66 \times (3/10)] + [68 \times (2/10)] + [69 \times (1/10)]$$

$$\text{WMA} = \$28 + \$19.80 + \$13.60 + \$6.90 = \$68.30$$

Please find the code snippet in the below location:
[BasicTimeSeriesForecasting.ipynb](#)

Exponential Smoothing Approaches in Time Series Forecasting

Exponential smoothers continue to remain attractive in the modern age of deep machine learning. They are simple and easy to implement. They are also surprisingly effective for short-term forecasting. Especially on rapidly-changing time series.

Exponential smoothing is a time series forecasting method for univariate data.

Exponential smoothing forecasting methods are similar in that a prediction is a weighted sum of past observations, but the model explicitly uses an exponentially decreasing weight for past observations.

When to use Exponential Smoothing:

- When you are forecasting for a large number of items.
- Forecasting horizon is relatively short (Day, Week, Month).
- There is little outside information available about cause and effect.
- Small effort in forecasting is desired. Effort is measured by both a method's ease of application and by computation requirements.
- Updating of the forecast as new data becomes available is easy.
- It is desired that the forecast is adjusted for randomness and tracks, trends and seasonality.

Single Exponential Smoothing

Single Exponential Smoothing, SES for short, also called Simple Exponential Smoothing, is a time series forecasting method for univariate data without a trend or seasonality.

It requires a single parameter, called *alpha* (α), also called the smoothing factor or smoothing coefficient.

This parameter controls the rate at which the influence of the observations at prior time steps decay exponentially. Alpha is often set to a value between 0 and 1. **Large values mean that the model pays attention mainly to the most recent past observations, whereas smaller values mean more of the history is taken into account when making a prediction.**

Hyperparameters:

- **Alpha:** Smoothing factor for the level.

The basic formula is:

$$S_t = \alpha y_{t-1} + (1 - \alpha) S_{t-1}$$

Where:

- α = the smoothing constant, a value from 0 to 1. When α is close to zero, smoothing happens more slowly. Following this, the best value for α is the one that results in the smallest mean squared error (MSE). Various ways exist to do this, but a popular method is the Levenberg-Marquardt algorithm.
- t = time period.

Many alternative formulas exist. For example, Roberts (1959) replaced

y_{t-1} with the current observation, y_t . Another formula uses the forecast for the previous period and current period:

$$\begin{aligned} F_t &= F_{t-1} + a(A_{t-1} - F_{t-1}) \\ &= a \cdot A_{t-1} + (1-a) \cdot F_{t-1} \end{aligned}$$

Where:

- F_{t-1} = forecast for the previous period,
- A_{t-1} = Actual demand for the period,
- a = weight (between 0 and 1). The closer to zero, the smaller the weight.

Holts/Double Exponential Smoothing

Double Exponential Smoothing is an extension to Exponential Smoothing that explicitly adds support for trends in the univariate time series.

In addition to the *alpha* parameter for controlling smoothing factor for the level, an additional smoothing factor is added to control the decay of the influence of the change in trend called *beta* (*b*).

The method supports trends that change in different ways: an additive and a multiplicative, depending on whether the trend is linear or exponential respectively.

Double Exponential Smoothing with an additive trend is classically referred to as Holt's linear trend model, named for the developer of the method Charles Holt.

- **Additive Trend:** Double Exponential Smoothing with a linear trend.
- **Multiplicative Trend:** Double Exponential Smoothing with an exponential trend.

For longer range (multi-step) forecasts, the trend may continue on unrealistically. As such, it can be useful to dampen the trend over time.

Dampening means reducing the size of the trend over future time steps down to a straight line (no trend).

As with modeling the trend itself, we can use the same principles in dampening the trend, specifically additively or multiplicatively for a linear or exponential dampening effect. A damping coefficient *Phi* (*p*) is used to control the rate of dampening.

- **Additive Dampening:** Dampen a trend linearly.
- **Multiplicative Dampening:** Dampen the trend exponentially.

Hyperparameters:

- **Alpha:** Smoothing factor for the level.
- **Beta:** Smoothing factor for the trend.
- **Trend Type:** Additive or multiplicative.
- **Dampen Type:** Additive or multiplicative.

- **Phi:** Damping coefficient.

Holts-Winter/Triple Exponential Smoothing

Triple Exponential Smoothing is an extension of Exponential Smoothing that explicitly adds support for seasonality to the univariate time series.

This method is sometimes called **Holt-Winters Exponential Smoothing**, named for two contributors to the method: Charles Holt and Peter Winters.

In addition to the alpha and beta smoothing factors, a new parameter is added called *gamma* (*g*) that controls the influence on the seasonal component.

As with the trend, the seasonality may be modeled as either an additive or multiplicative process for a linear or exponential change in the seasonality.

- **Additive Seasonality:** Triple Exponential Smoothing with a linear seasonality.
- **Multiplicative Seasonality:** Triple Exponential Smoothing with an exponential seasonality.

Triple exponential smoothing is the most advanced variation of exponential smoothing and through configuration, it can also develop double and single exponential smoothing models.

Additionally, to ensure that the seasonality is modeled correctly, the number of time steps in a seasonal period (*Period*) must be specified. For example, if the series was monthly data and the seasonal period repeated each year, then the *Period*=12.

Hyperparameters:

- **Alpha:** Smoothing factor for the level.
- **Beta:** Smoothing factor for the trend.
- **Gamma:** Smoothing factor for the seasonality.
- **Trend Type:** Additive or multiplicative.
- **Dampen Type:** Additive or multiplicative.
- **Phi:** Damping coefficient.
- **Seasonality Type:** Additive or multiplicative.
- **Period:** Time steps in seasonal period.

Please find the link to the code file to get the more insights on how it actually works: [ExponentialSmoothing.ipynb](#)

ARIMA Models:

1. Autoregressive models(AR):

In an autoregression model, we forecast the variable of interest using a linear combination of past values of the variable. The term autoregression indicates that it is a regression of the variable against itself.

This is like a multiple regression but with lagged values of y as predictors. We refer to this as an $AR(p)$ model, an autoregressive model of order p .

2. Moving Average models(MA):

Rather than using past values of the forecast variable in a regression, a moving average model uses past forecast errors in a regression-like model.

We refer to this as an $MA(q)$ model, a moving average model of order q .

If we combine differencing with autoregression and a moving average model, we obtain a non-seasonal ARIMA model. ARIMA is an acronym for AutoRegressive Integrated Moving Average.

3. ARIMA and SARIMA:

- Autoregressive integrated moving average (ARIMA) models predict future values based on past values.
- ARIMA makes use of lagged moving averages to smooth time series data.
- They are widely used in technical analysis to forecast future security prices.
- Autoregressive models implicitly assume that the future will resemble the past.
- Therefore, they can prove inaccurate under certain market conditions, such as financial crises or periods of rapid technological change.

An autoregressive integrated moving average model is a form of regression analysis that gauges the strength of one dependent variable relative to other changing variables. The model's goal is to predict future securities or financial market moves by examining the differences between values in the series instead of through actual values.

An ARIMA model can be understood by outlining each of its components as follows:

- Autoregression (AR,p): refers to a model that shows a changing variable that regresses on its own lagged, or prior, values.
- Integrated (I,d): represents the differencing of raw observations to allow for the time series to become stationary (i.e., data values are replaced by the difference between the data values and the previous values).
- Moving average (MA,q): incorporates the dependency between an observation and a residual error from a moving average model applied to lagged observations.

Please find the code snippet in the below location : [Arima_Sarima.ipynb](#)

Please refer the code snippet for ARIMA : [TimeSeriesFinanace](#)

ARIMA Formula:

$$y'_t = c + \underbrace{\varphi_1 y'_{t-1} + \dots + \varphi_p y'_{t-p}}_{\substack{\text{lags} \\ \text{(AR)}}} + \underbrace{\theta_1 \varepsilon_{t-1} + \dots + \theta_q \varepsilon_{t-q} + \varepsilon_t}_{\substack{\text{errors} \\ \text{(MA)}}$$

Here p,d,q are hyperparameters:

p: order of Autoregressive term (AR).

d: order of differencing required to make the time series stationary.

q: order of moving average (MA).

Note: to fit an ARIMA or SARIMA model to a time series, the series should be stationary or close to stationary.

Note: The predictors (Variable) should not be collinear or independent of each other.

Few Standard steps that need to be followed while dealing with ARIMA or SARIMA models:

1. Check the series for stationarity: to check the stationarity there are methods like Augmented Dickey Fuller (ADF) test. For graphically checking the stationarity in the Data you can use plot_acf. If the series is stationary, the correlogram dies down fairly quickly (within some lags). If time series is not stationary it will die down very slowly.
2. If the Series is nonstationary, then we need to convert the given series to stationary using difference(diff()) method.
3. Find the value of p,d,q for the ARIMA, SARIMA model :
 - p is the order of the Auto Regressive (AR) term. It refers to the number of lags to be used as predictors. We can find out the required number of AR terms by inspecting the Partial Autocorrelation (PACF) plot. The partial autocorrelation represents the correlation between the series and its lags.
 - q is the order of the Moving Average (MA) term. It refers to the number of lagged forecast errors that should go into the ARIMA Model.
 - We can look at the ACF plot for the number of MA terms.
 - d is the number of differencing required to make the time series stationary.
4. Put the values of p,d,q in ARIMA model.
5. Fit your model.
6. Get the future forecast for the series.

Time series Regression Models:

1. Simple Regression Models:

A simple Linear, polynomial, and exponential regression time series forecasting model can be created by indexing the time variable from 1 to n for n observations.

- The result is a model of trend and doesn't consider seasonality.
- To take seasonality into account, we can add a categorical variable indicating seasons and fit the regressor.
- In addition to categorical seasonal variable, following are some useful predictors for timeseries regression models:
 - Intervention variables
 - Public holidays

2. FBProphet:

FBProphet uses decomposable time series model with 3 main components: trends, seasonal, holidays or events effect and error which are combined into this equation:

$$y(t) = g(t) + s(t) + h(t) + e(t)$$

FBProphet uses time as a regressor and tries to fit several linear and nonlinear function of time as components. By default, FBProphet will fit the data using a linear model but it can be changed to the nonlinear model (logistics growth) from its arguments.

Facebook Prophet is an open-source algorithm for generating time-series models that uses a few old ideas with some new twists. It is particularly good at modeling time series that have multiple seasonalities and doesn't face some of the above drawbacks of other algorithms. At its core is the sum of three functions of time plus an error term:

growth $g(t)$, seasonality $s(t)$, holidays $h(t)$, and error e_t :

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

Growth Function:

The growth function models the overall trend of the data.

The growth function has three main options:

- **Linear Growth:** This is the default setting for Prophet. It uses a set of piecewise linear equations with differing slopes between change points. When linear growth is used, the growth term will look similar to the classic $y = mx + b$ from middle school, except the slope(m) and offset(b) are variable and will change value at each changepoint.
- **Logistic Growth:** This setting is useful when your time series has a cap or a floor in which the values you are modeling becomes saturated and can't surpass a maximum or minimum value (think carrying capacity). When logistic growth is used, the growth term will look similar to a typical equation for a logistic curve

(see below), except it the carrying capacity (C) will vary as a function of time and the growth rate (k) and the offset(m) are variable and will change value at each change point.

$$g(t) = \frac{C(t)}{1 + x^{-k(t-m)}}$$

- **Flat:** Lastly, you can choose a flat trend when there is no growth over time (but there still may be seasonality). If set to flat the growth function will be a constant value.

The Seasonality Function:

The seasonality function is simply a Fourier Series as a function of time.

$$s(t) = \sum_{n=1}^N (a_n \cos(\frac{2\pi nt}{P}) + b_n \sin(\frac{2\pi nt}{P}))$$

Holiday Function:

The holiday function allows Facebook Prophet to adjust forecasting when a holiday or major event may change the forecast. It takes a list of dates (there are built-in dates of US holidays, or you can define your own dates) and when each date is present in the forecast adds or subtracts value from the forecast from the growth and seasonality terms based on historical data on the identified holiday dates.

You can find the code snippets in below location : FbProphet

3. NeuralProphet:

NeuralProphet is a python library for modeling time-series data based on neural networks. It's built on top of PyTorch and is heavily inspired by Facebook Prophet and AR-Net libraries.

Few features of Neural Prophet:

- Using PyTorch's Gradient Descent optimization engine making the modeling process much faster than Prophet
- Using AR-Net for modeling time-series autocorrelation (aka serial correlation)
- Custom losses and metrics
- Having configurable non-linear layers of feed-forward neural networks

You can find the code snippet in the location: Neural-Prophet

ML Methods For Time-Series Forecasting

1. In the Univariate Time-series Forecasting method, forecasting problems contain only two variables in which one is time and the other is the field we are looking to forecast.
 - For example, if you want to predict the mean temperature of a city for the coming week, now one parameter is time(week) and the other is a city.
 - Another example could be when measuring a person's heart rate per minute through using past observations of heart rate only. Now one parameter is time(minute) and the other is a heart rate.
2. On the other hand, in the Multivariate Time-series Forecasting method, forecasting problems contain multiple variables keeping one variable as time fixed and others will be multiple in parameters.

Consider the same example, predicting the temperature of a city for the coming week, the only difference would come here now temperature will consider impacting factors such as

- Rainfall and time duration of raining,
- Humidity,
- Wind speed,
- Precipitation,
- Atmospheric pressure, etc,

And then the temperature of the city will be predicted accordingly. All these factors are related to temperature and impact it vigorously.

ML Models:

- ARIMA Model: As mentioned in the above section, it is a combination of three different models itself, AR, MA and I, where
 1. "AR" reflects the evolving variable of interest is regressed on its own prior values,
 2. "MA" infers that the regression error is the linear combination of error terms values happened at various stages of time priorly, and
 3. "I" shows the data values are replaced by the difference between their values and the previous values.

Combinedly "ARIMA" tries to fit the data into the model, and ARIMA depends on the accuracy over a broad width of time series.

- ARCH/GARCH Model: Being the extended model of its common version GARCH, Autoregressive Conditional Heteroscedasticity (ARCH) is the most volatile model for time series forecasting and are well trained for catching dynamic variations of volatility from time series.

- Vector Autoregressive Model or VAR model: It gives the independencies between various time-series data which as a generalization of the Univariate Autoregression Model.
- LSTM: Long-short term memory (LSTM) is a deep learning model, it is a kind of Recurrent Neural Network(RNN) to read the sequence dependencies.

It enables us to handle long structures during training the dataset and creates predictions according to previous data.

Deep Learning Models for Time Series Forecasting

Four novel deep learning architectures specialized in time series forecasting. Specifically, these are:

1. N-BEATS (ElementAI)
2. DeepAR (Amazon)
3. Spacetimeformer
4. Temporal Fusion Transformer or TFT (Google)

N-Beats and DeepAR are more battle-tested and have been used in many deployments. Spacetimeformer and TFT are also exceptional models and propose many novelties. They are able to take advantage of new dynamics, beyond the time series context.

1. N-BEATS

This model came straight from the short-lived ElementAI, a company cofounded by Yoshua Bengio. The top level architecture along with its main components is shown in Figure 1:

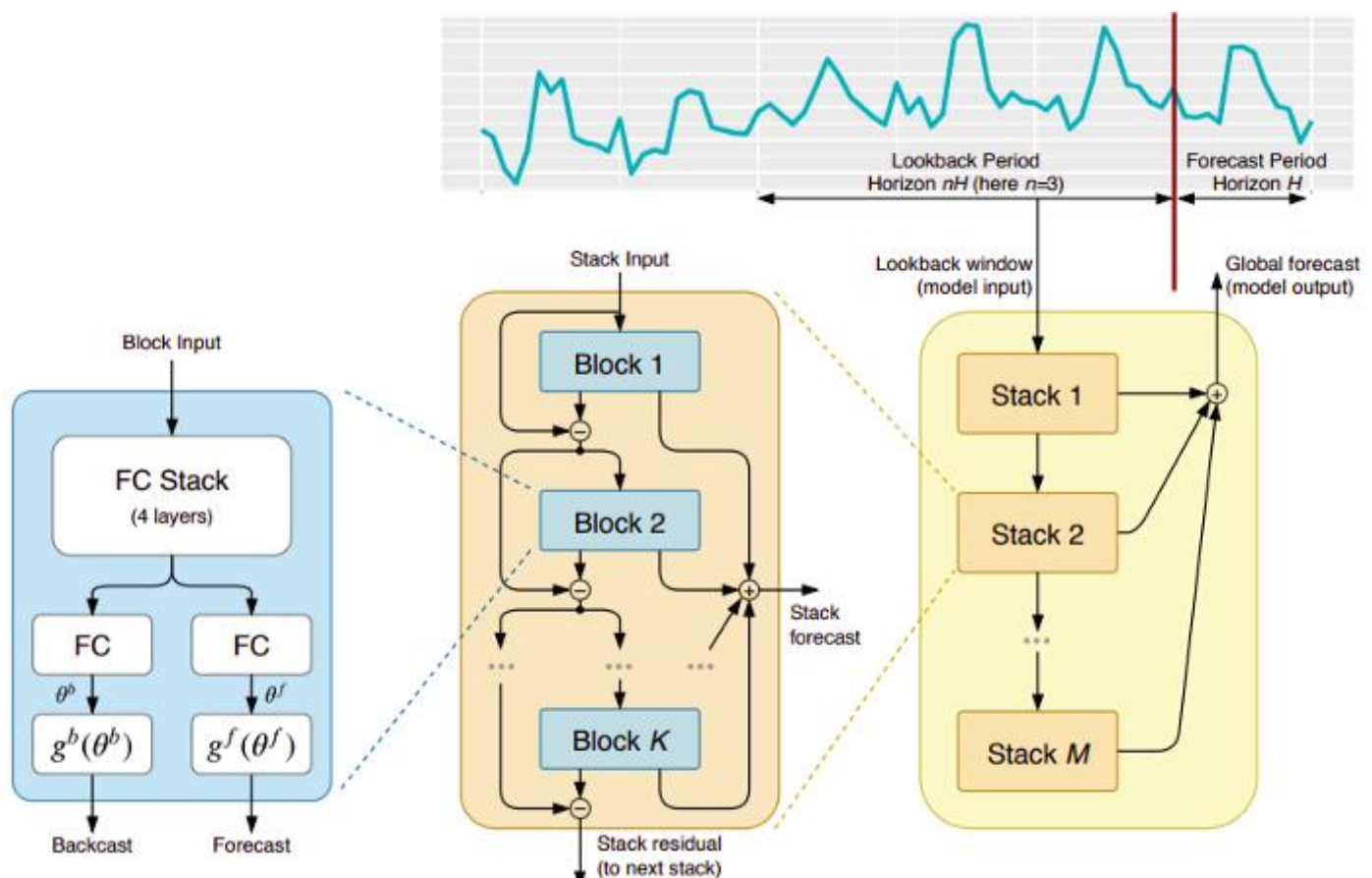


Figure 1: The N-BEATS architecture ([Source](#))

Essentially, N-BEATS is a pure deep learning architecture based on a deep stack of ensembled feed forward networks that are also stacked by interconnecting backcast and forecast links.

Each successive block models only the residual error due to the reconstruction of the backcast from the previous block and then updates the forecast based on that error. This process mimics the Box-Jenkins method when fitting ARIMA models.

These are the model's key advantages:

Expressive and easy to use: The model is simple to understand and has a modular structure (blocks and stacks). Moreover, it is designed to require minimal time-series feature engineering and no input scaling.

Multiple time-series: The model has the ability to generalize on many time-series. In other words, different time series with slightly different distributions can be used as input. In the N-BEATS implementation, this is achieved through meta-learning. Specifically, the meta-learning process consists of two procedures: the inner and the outer learning procedures. The inner learning procedure takes place inside blocks and helps the model capture local temporal characteristics. On the other hand, the outer learning procedure takes place inside stacks and helps the model learn global characteristics across all time-series.

Doubly Residual Stacking: The idea of residual connections and stacking is so brilliant that it is used in almost every type of deep neural networks, such as deep Convnets and Transformers. The same principle is applied in the N-BEATS implementation, but with some extra modifications: Each block has two residual branches, one running through the lookback window (called backcast) and the other through the predicted window (called forecast).

Each successive block models only the residual error due to the reconstruction of the backcast from the previous block and then updates the forecast based on that error. This helps the model better approximate the useful backcast signals, while simultaneously, the final stack forecast prediction is modeled as an hierarchical sum of all partial forecasts. Interestingly, this process mimics the Box-Jenkins method when fitting ARIMA models.

Interpretability: The model has two variants, general and interpretable. In the general variant, the final weights in the fully-connected layers of each block are learned by the network arbitrarily. In the interpretable variant, the last layer of each block is removed. Then, the backcast and forecast branches are multiplied by specific matrices that mimic trend (monotonic function) and seasonality (periodic cyclical function).

Note: The original N-BEATS implementation only works with univariate time series.

2. DeepAR

A novel time series model that combines both deep-learning and autoregressive characteristics. The top-level view of DeepAR is displayed in Figure 2:

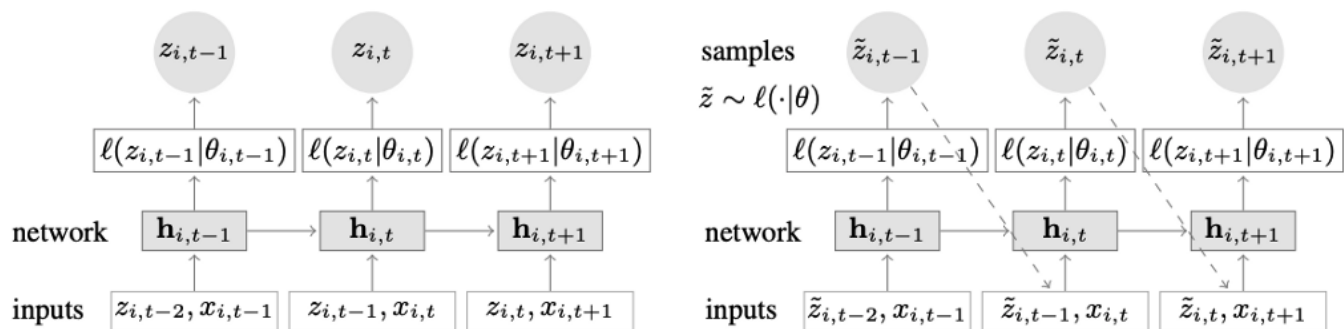


Figure 2: DeepAR model architecture ([Source](#))

These are the model's key advantages:

Multiple time series: DeepAR works really well with multiple time series: A global model is built by using multiple time series with slightly different distributions. Also, this property finds application in many real world scenarios. For example, an electric power company might want to launch a power forecasting service for each of their customers. It is almost certain that each customer would have a different consumption pattern (which means a different distribution).

Rich set of inputs: Apart from historical data, DeepAR also allows the use of known future time sequences (a characteristic of auto-regressive models) and extra static attributes for series. In the aforementioned power demand forecasting scenario, an extra temporal variable could be the month (as an integer, with values between 1-12). Obviously, given that each customer is associated with a single sensor that measures power consumption, the extra static variable would be something like `sensor_id` or `customer_id`.

Automatic scaling: If you are familiar with time series forecasting using neural network architectures like MLPs and RNNs, one crucial preprocessing step is to scale the time sequence using a normalization or standardization technique. In DeepAR, there is no need to do that manually since the model under the hood scales the autoregressive input z of each time series i with a scaling factor v_i , which is simply the average value of that time series. Specifically, the equation of the scaling factor used in the paper's benchmarks is the following:

$$v_i = 1 + \frac{1}{t_0} \sum_{t=1}^{t_0} z_{i,t}$$

In practice however, if the magnitude of the target time series differs significantly, then it may help if you apply your own scaling as well during preprocessing. For instance, in the energy demand forecasting scenario, the dataset could contain medium-voltage

electricity customers (e.g. small factories, consuming power in the order of MW's) and low-voltage customers (e.g. households, consuming power in the order of KW's).

Probabilistic forecasting: DeepAR makes probabilistic forecasts instead of directly outputting the future values. This is done in the form of Monte Carlo samples. Also, these forecasts are used to compute quantile forecasts, by using the quantile loss function. For those who are not familiar with this type of loss, quantile loss is used to calculate not only an estimate, but also a prediction interval around that value.

3. Spacetimeformer

In the context of univariate time sequences, temporal dependencies are all that matters. However, in a multiple time series scenario, things are not so simple. For instance, let's say we have a weather forecasting task, where we want to predict the temperature of five cities. Also, let's assume that these cities belong to a single country. Given what we have seen so far, we could use DeepAR and model each city as an external static covariate.

In other words, the model would consider both temporal and spatial relationships. This is the core idea of Spacetimeformer.

We could also take things one step further and use a model that leverages the spatial relationships between these cities/locations in order to learn extra helpful dependencies. In other words, the model would consider both temporal and spatial relationships. This is the core idea of Spacetimeformer.

Delving into spatiotemporal sequences

As its name implies, this model uses a transformer-based structure under the hood. In time series forecasting with transformer-based models, a popular technique to produce time-aware embeddings is to pass the input through a Time2Vec [6] embedding layer (As a reminder, for NLP tasks, a positional encoding vector is used instead of Time2vec that produces context-aware embeddings). While this technique works really well for univariate time sequences, it doesn't make any sense for multivariate time inputs. In language modeling, each word of a sentence is represented by an embedding, and a word is essentially a concept, part of a vocabulary.

In a multivariate time series context, at a given timestep t , the input has the form $x_{1,t}$, $x_{2,t}$, $x_{m,t}$ where $x_{i,t}$ is the numerical value of feature i and m is the total number of features/sequences. If we pass the input through a Time2Vec layer, a time embedding vector will be produced. In that case, what does this embedding really represent? The answer is it will represent the whole set of input as a single entity(token). Consequently, the model will learn only the temporal dynamics amongst timesteps, but will miss the spatial relationships among features/variables.

Spacetimeformer addresses this issue by flattening the input into a single large vector, called spatiotemporal sequence. If the input consists of N variables, organized into T timesteps, the generated spatiotemporal sequence will have $(N \times T)$ tokens. This is better displayed in Figure 3:

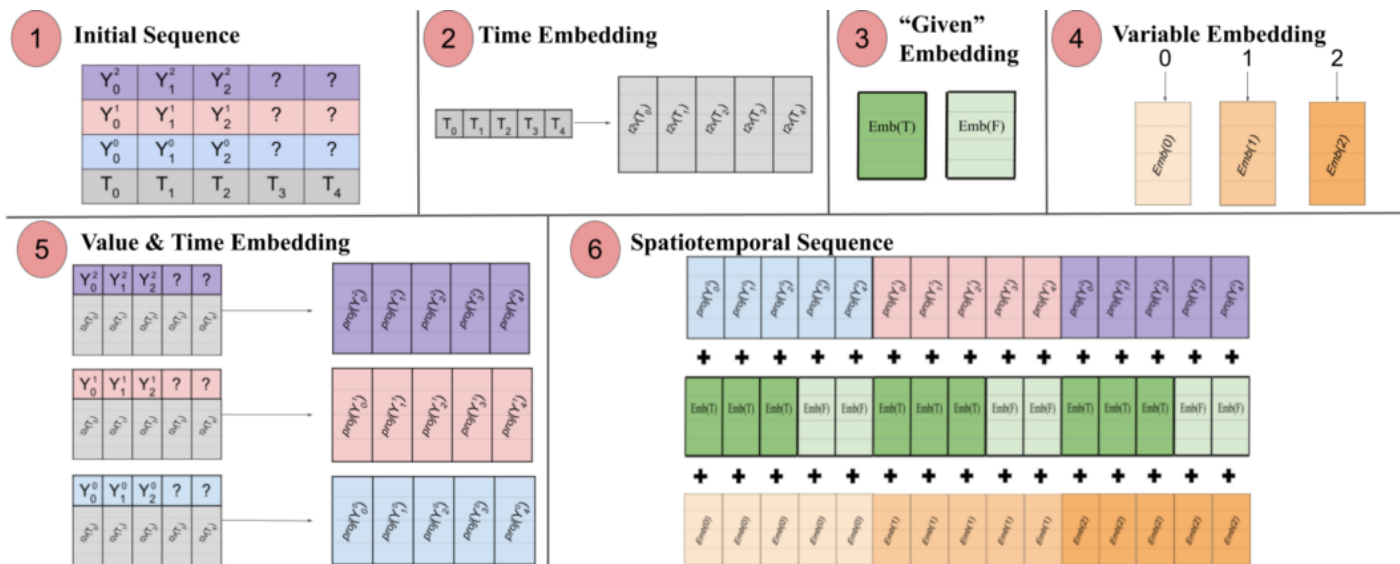


Figure 3: The series of transformations leading to the spatiotemporal sequence: The paper states: “(1) The multivariate input format with time information included. Decoder inputs have missing (“?”) values set to zero where predictions will be made. (2) The time sequence is passed through a Time2Vec layer to generate a frequency embedding that represents periodic input patterns. (3) A binary embedding indicates whether this value is given as context or needs to be predicted. (4) The integer index of each time series is mapped to a “spatial” representation with a lookup-table embedding. (5) The Time2Vec embedding and variable values of each time series are projected with a feed-forward layer. (6) Value&Time, Variable, and Given embeddings are summed and laid out such that MSA attends to relationships across both time and variable space at the cost of a longer input sequence.” Source

In other words, the final sequence has encoded a unified embedding, consisting of temporal, spatial and contextual information.

One drawback of this approach however is that the sequences can become too long, leading to a quadratic increase of resources. This is because according to the attention mechanism, every token/entity is checked against one another. The authors make use of a more efficient architecture suitable for larger sequences, called Performer attention Mechanism.

4. Temporal Fusion Transformer

Temporal Fusion Transformer (TFT) is a transformer-based time series forecasting model published by Google. If you would like a more thorough analysis regarding this awesome model, check this post.

TFT is more versatile than the previous models. For instance, DeepAR does not work with time-dependent features that are known only up to the present.



An example of how all these features are used is shown in **Figure 5**:

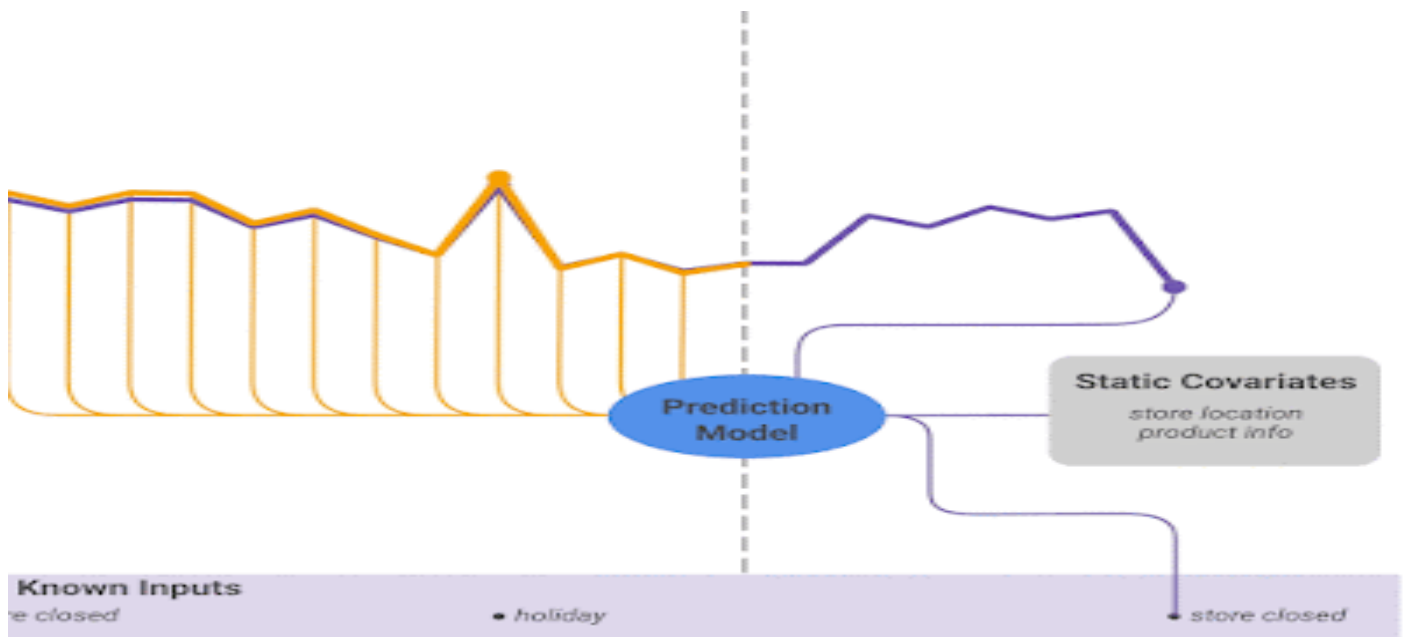


Figure 5: Effect of external static variables on forecasting ([Source](#))

- **Interpretability:** TFT gives much emphasis on interpretability. Specifically, by taking advantage of the **Variable Selection component** (shown in Figure 4), the model can successfully measure the impact of each feature. As a consequence, the model learns the feature importance's.
On the other hand, TFT proposes a novel interpretable Multi-Head Attention mechanism: The attention weights from this layer can reveal which time-steps during the lookback period are the most important. Therefore, visualization of those weights could reveal the most prominent seasonal patterns throughout the whole dataset.
- **Prediction Intervals:** Similar to DeepAR, TFT outputs a prediction interval along with the predicted values, by using quantile regression.

Forecasting Error

Forecasting error help us to find out the accuracy of our forecast. It is defined by the difference between the actual demand for time period t and the forecasted for time period t.

Forecast error = Actual demand for time t - Forecast for time t

$$e_t = A_t - F_t$$

Where , e_t is Forecast error, A_t is Actual demand and F_t is forecast
For time t

Measure of Forecasting Error:

0. Average Error (AE)

$$\text{Average Error (AE)} = \frac{1}{N} \sum_{t=1}^N e_t$$

Note: For a large dataset AE(Average Error) should be zero.

1. Mean Absolute Deviation (MAD)

Mean Absolute Deviation (MAD)

$$= \frac{1}{N} \sum_{t=1}^N |e_t|$$

MAD should be as low as possible.

Note : When the errors that occurs in the forecast are normally distributed, MAD related to the standard deviation as

1 Standard Deviation = 1.25 MAD

Or

1 MAD = 0.8 Standard Deviation

2. Mean Squared Error(MSE)

Mean Squared Error (MSE)

$$= \frac{1}{N} \sum_{t=1}^N e_t^2$$

3. Mean Absolute Percentage Error(MAPE)

Mean absolute percentage Error (MAPE)

$$\frac{1}{N} \sum_{t=1}^N \left| \frac{e_t}{D_t} \times 100 \right|$$

4. Tracking Signal

It is a measurement that indicates whether the forecast average is keeping pace with any genuine upwards or downwards changes in demand.

$$TS = RSFE(\text{Running Sum of Forecasting Error}) / MAD$$

One example for calculating the tracking signal:

D_t (demand)	F_t (Forecast)	e_t (Error)	$ e_t $	RSFE	MDA	TS
900	1000	-100	100	-100	100	-1
950	1000	-50	50	-150	75	-2
1000	1000	0	0	-150	50	-3
1050	1000	50	50	-100	50	-2
900	1000	-100	100	-200	60	-3.33
975	1000	-25	25	-225	54.17	-4.15

Note : TS(Tracking signal) sometimes should be positive and sometimes negative. If it is only positive or only negative that implies that our forecast is biased.

Handling hierarchical and granular time series data

Consider a manufacturing industry (chips) where we have

- category (gluten, gluten-free), sub-category (flavors) and sku (unique product types) in the product hierarchy
- channel (road, sea etc.), sub-channel (truck, railway etc.) in the transporting hierarchy
- customer (Walmart, Costco), store (Walmart#121, Walmart #122 etc.) in the customer hierarchy

We have multiple hierarchies in the data and the data is often sparse at the most granular level.

What can be forecast? And what is suitable for forecast?

Forecasting is required in many situations: deciding whether to build another power generation plant in the next five years requires forecasts of future demand; scheduling staff in a call Centre next week requires forecasts of call volumes; stocking an inventory requires forecasts of stock requirements. Forecasts can be required several years in advance (for the case of capital investments), or only a few minutes beforehand (for telecommunication routing). Whatever the circumstances or time horizons involved, forecasting is an important aid to effective and efficient planning.

Some things are easier to forecast than others. The time of the sunrise tomorrow morning can be forecast precisely. On the other hand, tomorrow's lotto numbers cannot be forecast with any accuracy. The predictability of an event or a quantity depends on several factors including:

1. how well we understand the factors that contribute to it.
2. how much data is available?
3. whether the forecasts can affect the thing we are trying to forecast.

For example, forecasts of electricity demand can be highly accurate because all three conditions are usually satisfied. We have a good idea of the contributing factors: electricity demand is driven largely by temperatures, with smaller effects for calendar variation such as holidays, and economic conditions. Provided there is a sufficient history of data on electricity demand and weather conditions, and we have the skills to develop a good model linking electricity demand and the key driver variables, the forecasts can be remarkably accurate.

On the other hand, when forecasting currency exchange rates, only one of the conditions is satisfied: there is plenty of available data. However, we have a limited understanding of the factors that affect exchange rates, and forecasts of the exchange rate have a direct effect on the rates themselves. If there are well-publicized forecasts that the exchange rate will increase, then people will immediately adjust the price they are willing to pay and so the forecasts are self-fulfilling. In a sense, the exchange rates become their own forecasts. This is an example of the "efficient market hypothesis". Consequently, forecasting whether the exchange rate will rise or fall tomorrow is about as predictable as forecasting whether a tossed coin will come down as a head or a tail. In both situations, you will be correct about 50% of the time, whatever you forecast. In

situations like this, forecasters need to be aware of their own limitations, and not claim more than is possible.

Determining what to forecast

In the early stages of a forecasting project, decisions need to be made about what should be forecast and what can be forecast. For example, if forecasts are required for items in a manufacturing environment, it is necessary to ask whether forecasts are needed for:

1. every product line, or for groups of products?
2. every sales outlet, or for outlets grouped by region, or only for total sales.
3. weekly data, monthly data or annual data?
4. If the data at the required level is suitable statistically to forecast or not?

It is also necessary to consider the forecasting horizon. Will forecasts be required for one month in advance, for 6 months, or for ten years? Different types of models will be necessary, depending on what forecast horizon is most important.

How can we identify if the data is suitable for forecasting?

The data at the required level may not be suitable for statistical forecasting due to the following reasons:

1. High null rate (if the data is sparse at the desired level at the desired frequency)
2. High outlier rate
3. High variation in the data

If the data has the above issues, and is highly granular at this level, it is recommended to create data velocities at higher levels as these patterns would be less erratic and easier to forecast.

We can store the proportions of the data velocities from high granular to low granular levels to scale back the forecasts if necessary.

If the issues still persist, we need to keep in mind the statistical forecasting methods may be unreliable. Below is the script to check if the timeseries data is reliable to forecast using statistical forecasting techniques.

Often real-life data requires has various caveats which we need to understand and deal with appropriately. Consider the above-mentioned chips manufacturing example:

1. There might be marketing campaigns/discounts which affect the sales of the items. The effect can either be removed or we can use models capable of handling them like FBProphet, NeuralProphet etc.
2. When a certain product is newly launched or is on its way to be discontinued, the data will be skewed and needs to be treated appropriately.
3. It is important to account for the macro and micro factors that have a huge influence on the product/concept before we can model the data for forecasting.

Hierarchical Time Series

Consider supply chain operation at an Ice Cream Company.

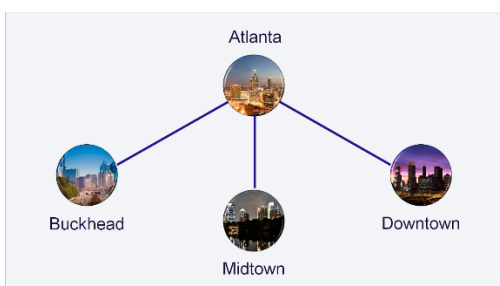
To get familiar with your new company, you start your first week by reviewing several reports with forecasts from different departments. The procurement department's report states that they plan to buy enough raw material to manufacture a million gallons of ice cream next year. However, you notice that the store operations report estimates a demand of only *half* a million gallons. That's not good!

There are three distinct possibilities at play here.

- First, the procurement department's forecasts could be off. The ingredients for a million gallons would create excess inventory, locking up capital that could be used in other areas of the business.
- Second, the sales department's forecasts could be inaccurate. In this case, the distribution network will suffer from inefficiencies, and stores might not be staffed appropriately to handle the customer load that a million gallons would necessitate.
- Third, *both* the forecasts could be wrong. Forecasting is akin to throwing darts, but the two departments should at least throw the same dart.

Bottom line: separate forecasts for the same thing, whether it be ice cream or sales estimates, need to add up to ensure consistency in planning, budgeting, and execution. The solution? Hierarchical time series (HTS) forecasting, which ensures that forecasts at all different levels and parts of the business match up.

A **hierarchical time series (HTS)** is a collection of time series that follows a hierarchical aggregation structure. As an example, assume that there are three stores (Buckhead, Midtown, and Downtown) that sell Scoopex ice cream in Atlanta. A given store's monthly ice cream sales is, itself, a time series. The total gallons of ice cream sold across all three stores is also a time series. Crucially, this collection of four time series has a hierarchical aggregation structure (specifically, a two-level geographical hierarchy).



Other common hierarchies include product hierarchies (e.g., SKU sales aggregate up to product subcategory sales, which further aggregate to product categories, and so on), temporal hierarchies (e.g., forecasts for the next seven days must add up to the week's individual forecast), and more.

Hierarchical time series forecasting is the process of generating coherent forecasts (or reconciling incoherent forecasts), allowing individual time series to be forecast individually, but preserving the relationships within the hierarchy.

In our Ice cream company's example, the reports show that company has inadvertently created *incoherent forecasts*. However, if we ensure that each department's sales estimates are ultimately in agreement, the forecasts will become *coherent*, and consistency will be restored.

HTS forecasting is not a time series forecasting methodology per se, like exponential smoothing or ARIMA. It's actually a collection of techniques that make forecasts coherent across a time series hierarchy.

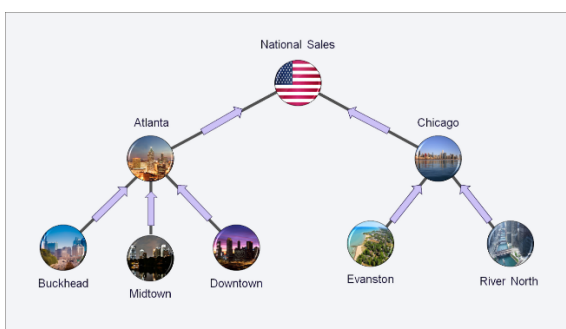
You can use HTS forecasting whenever a time series hierarchy is involved. Such cases are nearly omnipresent in industry. Our Ice cream company's example is a good one for any product company – oftentimes you'll want to match up forecasts for individual stores with the estimates for the greater region or square up individual product forecasts with projections for the whole product category.

There are 4 Common Hierarchical Time Series Forecasting methods

1) The bottom-up approach

As the name suggests, *bottom-up* forecasting involves forecasting the most granular level of the hierarchy, then aggregating up to create estimates for the higher levels.

Applying bottom-up forecasting to our ice cream example requires forecasting the individual store sales for the Buckhead, Midtown, and Downtown locations first. Total Atlanta ice cream sales can then be calculated by simply adding up these store-level forecasts. Rinse and repeat for the Chicagoland locations, and then national sales will follow.



The main advantage of this method is that, because forecasts are obtained at the lowest level of the hierarchy, no information is lost due to aggregation. However, it ignores the relationships between the series (e.g., doesn't take any Atlanta-area forecasts into account when forecasting Chicagoland locations, and vice versa), and usually performs poorly on highly aggregated data. It's also computationally intensive, since you have to forecast the most granular time series in the hierarchy, meaning more data points (and therefore more horsepower and/or runtime is required). Furthermore, information at lower levels of a hierarchy tends to be noisier, potentially resulting in a reduced overall forecast accuracy.

2) The top-down approach

In the *top-down* approach, you first forecast the highest level of the hierarchy, then split up the forecasts to get estimates for the lower levels (typically using historical proportions).

For example, assume the forecast for next month's nationwide ice cream sales is 30,000 gallons. Let's also say that the historical proportion of monthly ice cream sales is about $\frac{1}{3}$ for Atlanta and $\frac{2}{3}$ for Chicago. The top-down approach tells us that Atlanta is projected to sell about 10,000 gallons of ice cream in the next month. Rinse and repeat for the lower levels of the hierarchy.



Due to its simplicity, this is one of the most commonly used methods for HTS forecasting. It provides reliable forecasts for higher levels in the hierarchy, and only a single true forecasting model is required.

However, the top-down approach tends to produce less accurate forecasts at lower levels of the hierarchy due to a loss of information (historical proportions don't always fully capture the true behavior of the lower levels), especially for individual series with difficult distributions.

3) The middle-out approach

The *middle-out* approach is a combination of the bottom-up and top-down approaches, and can be used on hierarchies with at least three levels.

In this approach, you start by forecasting the middle level (neither the most granular nor the most aggregated). After these numbers are calculated, you can forecast the higher levels in the hierarchy using the bottom-up approach and the lower levels with the top-down approach.

For example, in the hierarchy shown below, forecasts will be generated for Atlanta and Chicago. The national sales forecast can be calculated by adding up the individual forecasts for both, and the location-level forecasts are estimated using historical proportions.



The middle-out approach is a healthy compromise between the bottom-up and top-down approach. Resulting forecasts don't lose too much information, yet computational time does not explode.

4) The optimal combination/reconciliation approach

All the approaches mentioned above only really forecast one level of the hierarchy, but the *optimal combination* approach (proposed by Hyndman et al. in 2011) forecasts independently at all levels, using all the information and relationships a hierarchy can offer.



Assuming the base forecasts approximately satisfy the hierarchical aggregation structure (i.e., they're not off by a half a million gallons of ice cream), the individual forecasts are then reconciled using a linear regression model. The newly coherent forecasts are a weighted sum of the forecasts from all levels, with the weights found by solving a system of equations that ensure the natural relationships between the different levels of the hierarchy are satisfied.

The optimal reconciliation approach can give more accurate forecasts than the other methods we've covered so far, providing unbiased forecasts at all levels with minimal loss of information, taking advantage of the relationships between time series to find patterns (e.g., seasonal variations at higher levels can better inform forecasts at lower levels of hierarchy). In addition, each forecast is created independently, meaning different forecasting methods (e.g., ARIMA, ETS, naive methods, etc.) can be used at each level. Though it's potentially the most accurate option, this method is also the most complex and computationally intensive approach, which means that it doesn't scale well for many time series.

How to Choose the best approach:

The best approach typically depends on your goals and constraints. Our general advice would be to start with simpler approaches, then move to more complex and computationally intensive methods if you're unsatisfied with your results. In short, if a faster and simpler approach provides a forecast that suits your needs, you can start with a top-down approach and skip the more complex stuff. To figure out what's good enough for you, figure out if you need certain degrees of accuracy at certain levels of the hierarchy, if you're limited by the available computing or time resources, or if you're constrained in any other way (e.g., your stakeholders must be able to interpret the model instead of just using the results).

The bottom-up, top-down, and middle-out approaches are typically biased toward the level they're forecasting. If getting an accurate forecast for that level is the main goal, then using one of these three might be best. If not, the optimal reconciliation method provides unbiased forecasts that typically work at all levels of the hierarchy but may be computationally restrictive when working with too many time series. If there are no constraints on computational time (we can all dream), the best strategy is to evaluate each method using back-testing, also known as time series cross-validation.

Good Read: [How to Choose the Right Forecasting Technique \(hbr.org\)](https://hbr.org/2017/05/how-to-choose-the-right-forecasting-technique/)

Limitations of Time Series Analysis/Forecasting

Time series methods draw on vastly different areas in statistics, and lately, machine learning. You must know a lot about all these things, in general, to make sense of what you're doing. There is no real unification of the theory, either.

Often there are ways around getting a model that is time-series based where the predictions are almost as good and is faster to implement. Note that this may or may not blow up in your face later. In some cases, however, temporal effects are so weak that it makes more sense to just use the non-temporal ones... which can be difficult to explain (the need to check) to a manager if we've had to spend 2.5 weeks setting up the tests for temporal effects.

Time series has the below-mentioned limitations, we must take care of those during our analysis,

- Like other ML or statistical models, the missing values are not supported by TSA
- The data points must be linear in their relationship.
- Data transformations are mandatory, so a little expensive.
- Most models work on Uni-variate data.
- The various factor that affected the fluctuations of a series cannot be fully adjusted by the time series analysis.
- The various factor that influences the time series may not remain the same for an extended period and so forecasting made on this basis may become unreliable.

Links to some of the problems where Time Series Analysis is challenging:

[\(PDF\) Forecasting Exchange Rates: A Chaos-Based Regression Approach \(researchgate.net\)](#)

INDEX of code snippets

s.no	Topic	Link
1	Time series Seasonality Test	<u>Timeseries_Seasonality_test.ipynb</u>
2	Stationarity Test	<u>Stationarity_Test.ipynb</u>
3	Exponential Smoothing	<u>ExponentialSmoothing.ipynb</u>
4	Basic time series forecasting	<u>BasicTimeSeriesForecasting.ipynb</u>
5	ARIMA / SARIMA	<u>Arima_Sarima.ipynb</u>
6	FbProphet	<u>FbProphet</u>
7	Neural Prophet	<u>Neural-Prophet</u>
8	Time series on Finance Data:ARIMA	<u>TimeSeriesFinanace</u>
9	Forecast Recommendation	<u>ForecastRecommendation .ipynb</u>

Document Version

Revision No.	Document Release Date / Revision on Date	Revision Description	Section No.	Rationale for Change	Change type (add / modify / delete)	Effective Date	Document Revision