Time series is different because:

1. Data is correlated over the time and this correlation is not used in other ML techniques
2. Need a lot of data
3. If the forecast for one period is wrong, error can propagate over time.
4. Time series data is non stationary
5. In multi variables sometimes independent variables are also required to be predicted for the future to determine future dependent variables.

# Time series:

Sequence of data captured at equally spaced intervals.

# Standard measures:

1. Forecast miss % or error %
2. Error rates by Horizon (time span)

# Types of Forecasting:

These are the two types of forecasting problems. Consider that the vast majority of applications employ univariate models, harder to combine variables when using time series data.

1. **Univariate**

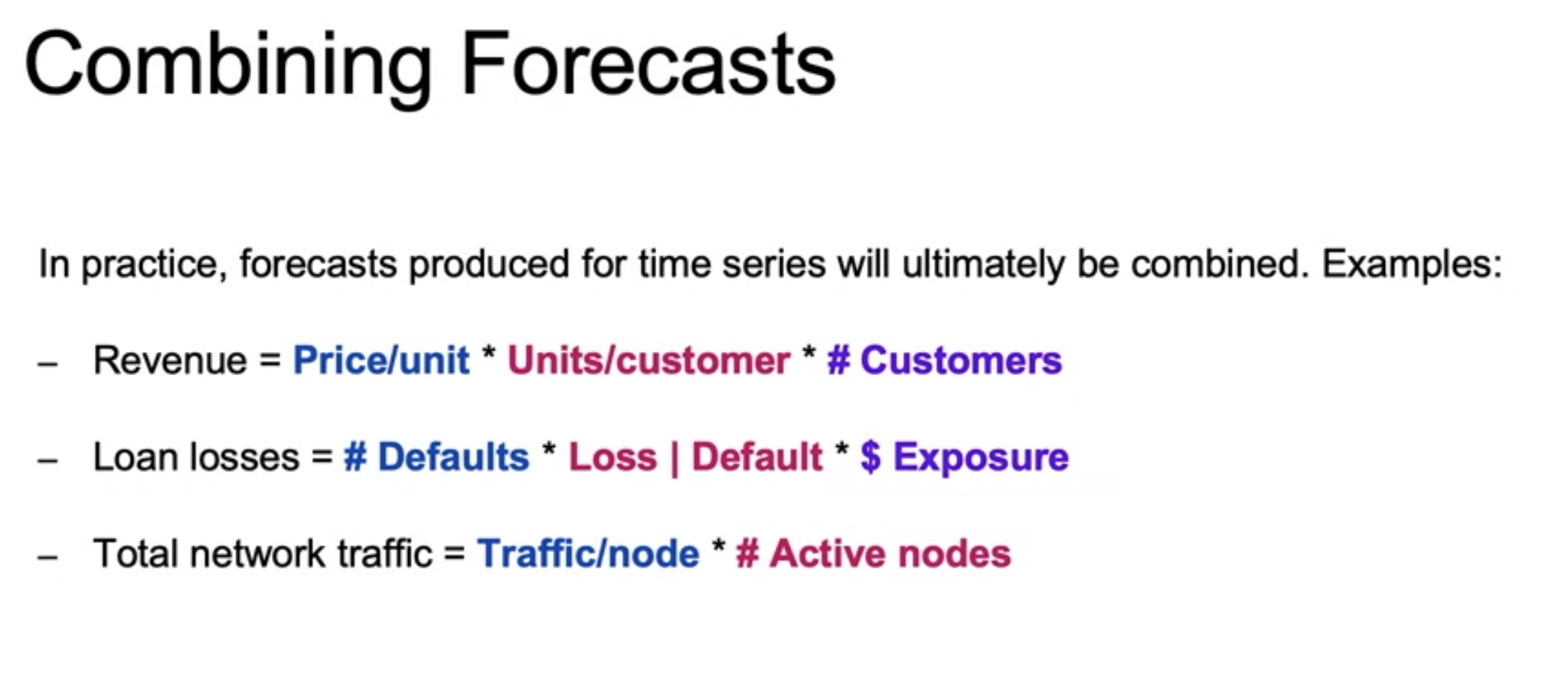
Think of single data series containing of:

* Continuous data, binary data, or categorical data
* Multiple unrelated series
* Conditional series

2. **Panel or Multivariate:**

Think of multiple related series identifying groups such as customer types, department or channel, or geographic joint estimation across series

**Combining forecast:**



So should we consider forecasting individually or combine these and then forecast?

# Important functions / Libraries and data types:

**Numpy** -

array,

datetime64 (datetime64[D], datetime64[M]),

Timedelta64

**Pandas** - Series

DataFrame

Index (Datetime Index, Timedelta index, Period Index)

Df.index

Df.dtypes

df.set\_index(‘column name’, inplace = True)

unstack(create separate rows for each category)

reindex(new\_index, fillna = 0)

.diff()

.pct\_change

np.arange()

np.random.normal, np.empty\_like

np.split

# Frequency of data

Some time series applications require data from all periods with a frequency assigned. For frequency, there should not be a missing or duplicate index (time period). Frequency can be week days, business days, daily etc.andas automatically detects the frequency if data is standardized.

Thus, frequency helps in ensuring that data is standardized & is req for functionality like resampling

# Resampling of data:

Data can be resampled to agg or granular freq with ***.resample(‘W’/’M’/’Q’).aggregate\_fn()***

aggregate\_fn can be sum, mean.

Aggregate\_fn is required by up sampling or aggregating

While downsampling, interpolation is required - **but aggregated value being directly interpolated on month is not right. For e.g. if annual sum of sales is 120 and monthly we aggregate it to 120, 125 etc it doesnt make sense.**

***.interpolate(method = ‘spline’, order = 3) can be used.***

# Plotting of data:

# Time series decomposition:

Components of time series data

**Trend**: Long term direction : (General / overall direction of time series)

**Seasonality** : Periodic behavior / Seasons : captures effects that occurs with specific frequency

**Residual**: Irregular behavior that cannot be predicted by trend or seasonality.

Residuals can be random fluctuations or short term fluctuations.

Models perform better if known source of variation are removed (for e.g Trend and seasonality)

## Decomposition models:

Time series models can have these components in different ways and so different models to decompose them

1. Additive models :
   1. Decomposing models as Trend + Seasonality + Residual
   2. Used when magnitudes of seasonality and residual are independent of trend (Does not change with trend)
2. Multiplicative decomposition:
   1. Trend\*Seasonality\*Residual
   2. Multiplicative can be transformed to additive using log
   3. Used when magnitudes of seasonal and residual values fluctuate with trend (imagine sin(x) becomes 5\*sin(x)
3. Pseudo additive decomposition:
   1. Combination of above two
   2. Used when time series have values near 0
   3. Features are like multiplicative model but we cannot divide by 0

Decomposition allows us to remove these components & simplify things.

Common ways to decompose a time series are:

1. Exponential smoothing (single, double or triple)
2. Locally Estimated scatter plot smoothing - Powerful method as compared to classical approaches which does better with seasonal components that change over time and allows user to control that rate of change & robust to outliers. *(But can only handle only additive components)*
3. Frequency based methods: Uses spectral analysis. Defines underlying recurring patterns without specifying any frequency

# Stationarity:

1. Stationarity impacts our ability of forecasting
2. Stationary time series have mean and variance constant over time which makes it easier for us to model.
3. Non-stationary series are harder to model.
4. We handle non stationary data by Identifying sources of non stationarity & Transform series to make it stationary
5. For a time series to be stationary:
   1. Have constant mean (no trend, rolling means can be checked) and variance (rolling variance can be checked) ***Which period is to be considered for rolling mean, what if one period has constant mean & larger or shorter period does not.***
   2. Constant autocorrelation structure (i.e. lag in time to which a particular data point is correlated should be constant). ***Autocorrelation above 1 is not suitable, we will need to apply transformation. Constant autocorrelation results in constant mean and changing autocorrelation results in changing mean.***
   3. Have no periodic / seasonality component

# Identifying stationary and non stationary behavior:

## Plots:

Create time series plots, trend, seasonality can be easily looked.

## Summary statistics & histograms:

Convert data into chunks and check means and variance : if constant, then stationary or else non stationary.

Plot histograms: Normal distribution signifies constant mean and variance, uniform distribution means trend

## Augmented Dickey-fuller test:

Null hypothesis: Not a stationary series

Alternative hypothesis: Stationary series

Used to determine if a series is stationary

Adf is the test statistic, more negative it is we are more certain that our series is stationary

ADF has trouble in identifying varying variance

# Non stationary to stationary transformations:

1. Remove trend and seasonality - Residual is stationary:
2. Changing variance can be handled with log transformations
3. Autocorrelation can be removed using difference with lag i.e. new series should

T[:-1] - T[1:]

# Data Smoothing:

Smoothing improves the ability to forecast by reducing impact of noise.

Some common methods of smoothing are:

1. Simple average smoothing
2. Equally weighted moving average
3. Exponentially weighted moving average

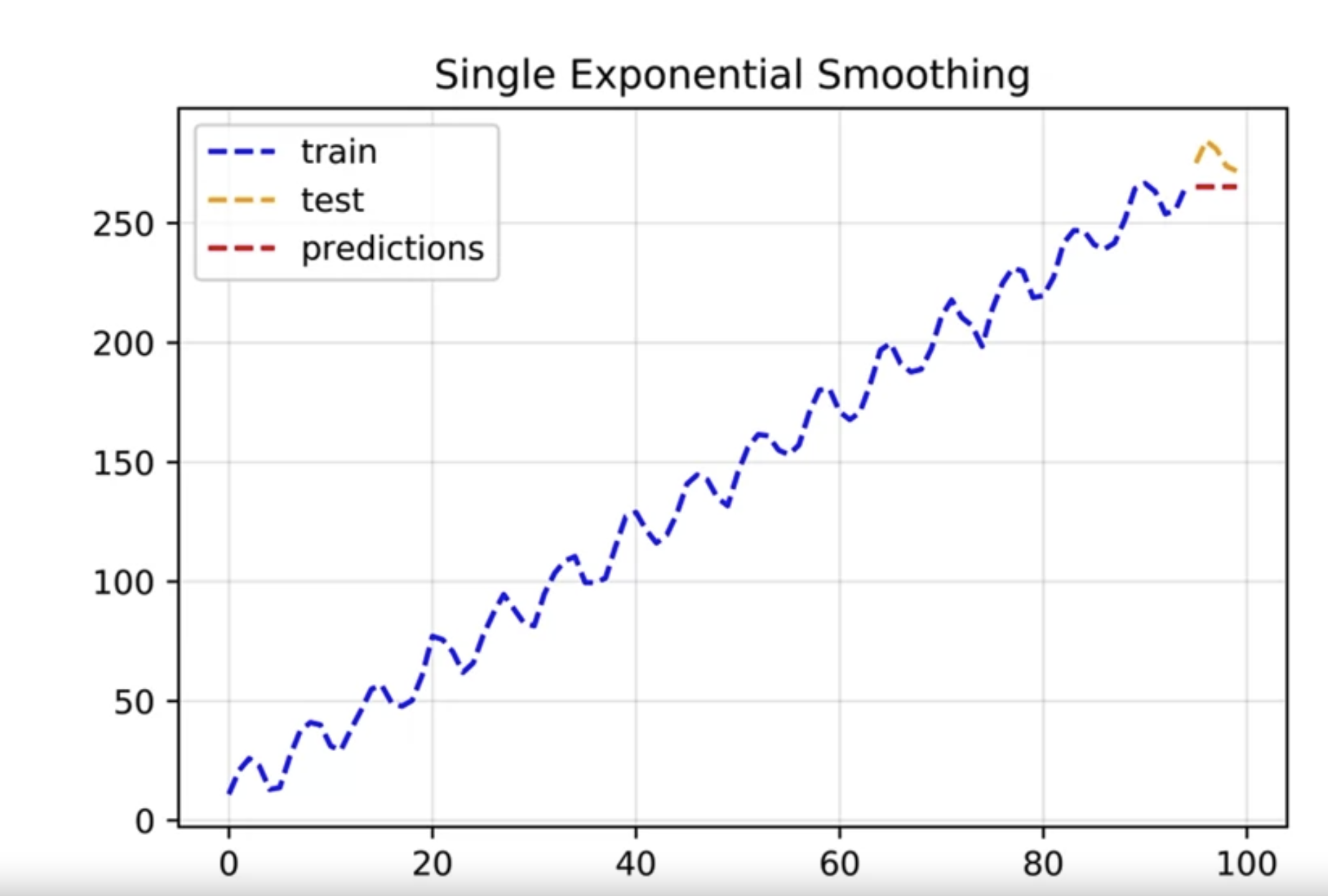
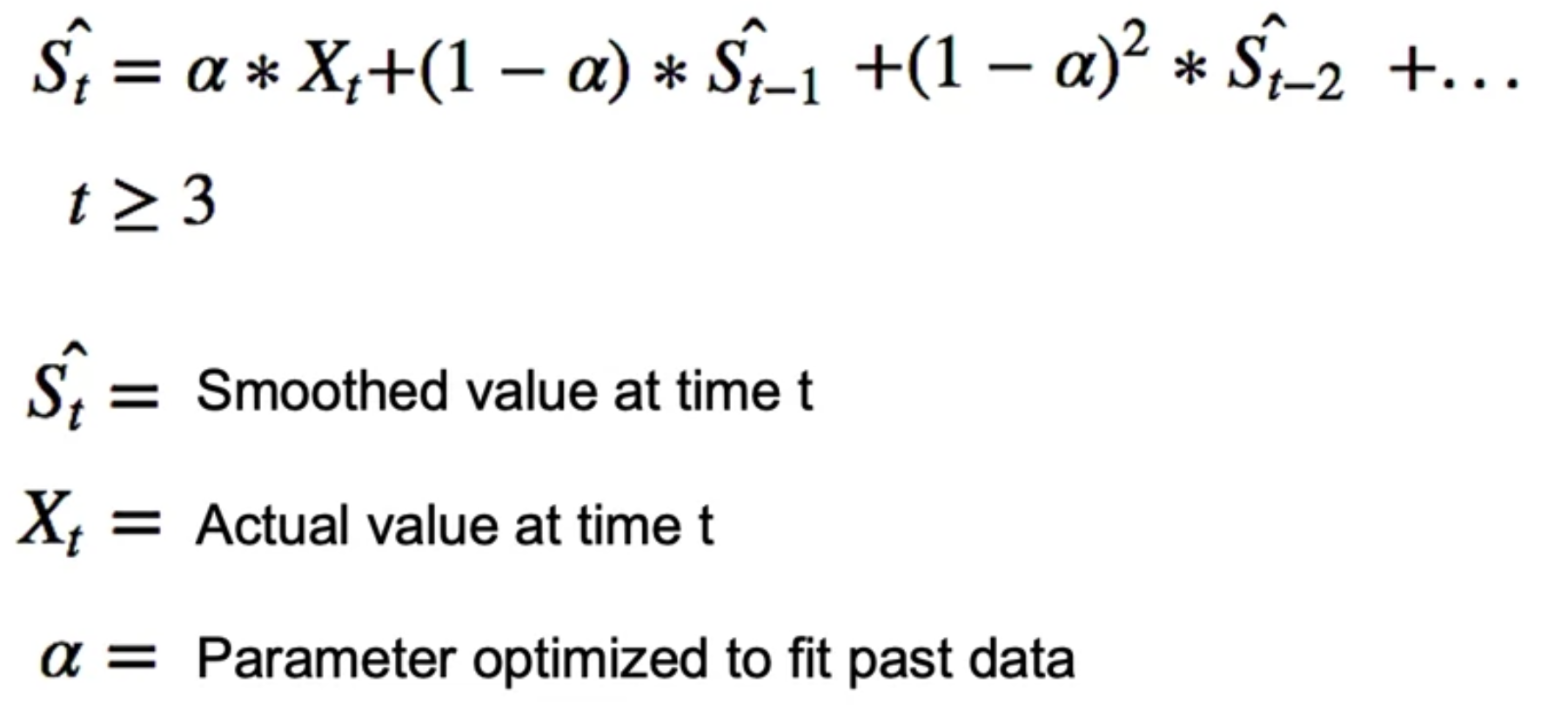
Smoothing allows us to improve forward looking forecast

1. **Forecasting with simple average**: One way to forecast a stationary data is : as the mean is constant, the mean of a window including the last value is known.

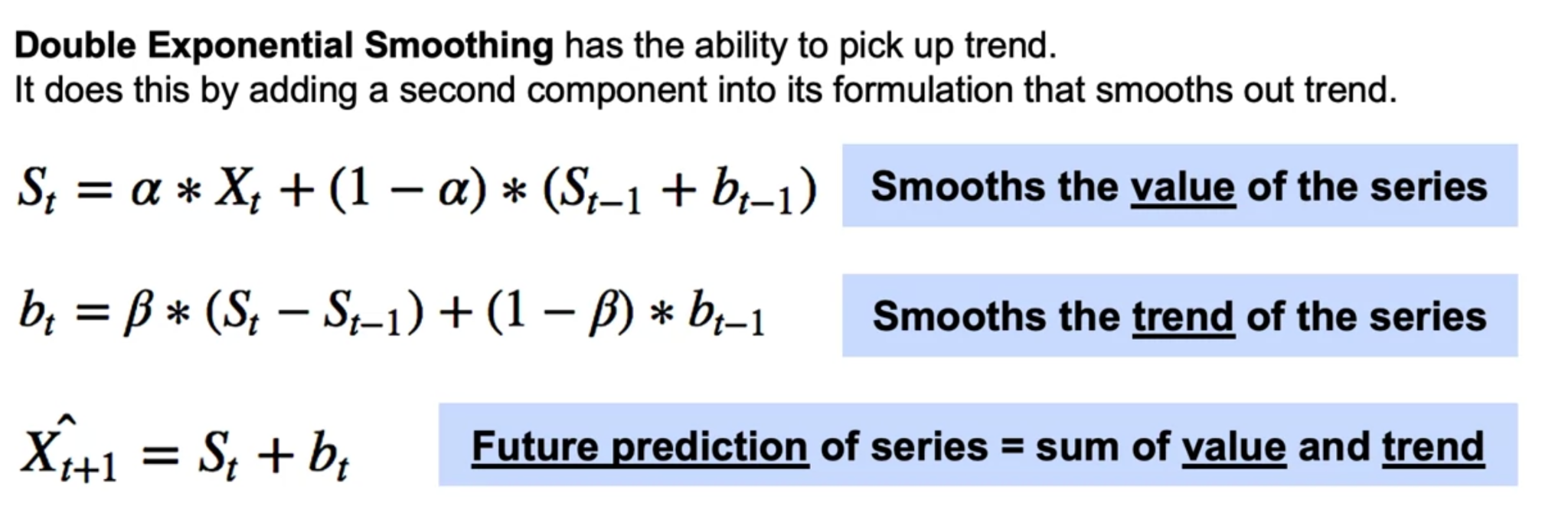
What if there is a trend?

1. **Moving average:** Smoothing technique, allows us to avoid local fluctuations
   1. **Equally weighted**: Becomes a problem when trend is aggressive
   2. **Exponentially weighted**: (ax(t-2) + bx(t-1) + cx(t)) / (a+b+c) where c>b>a but it does handle aggressive trends like exponential trend, although it is better than equal weighted.
2. **Advance smoothing:**

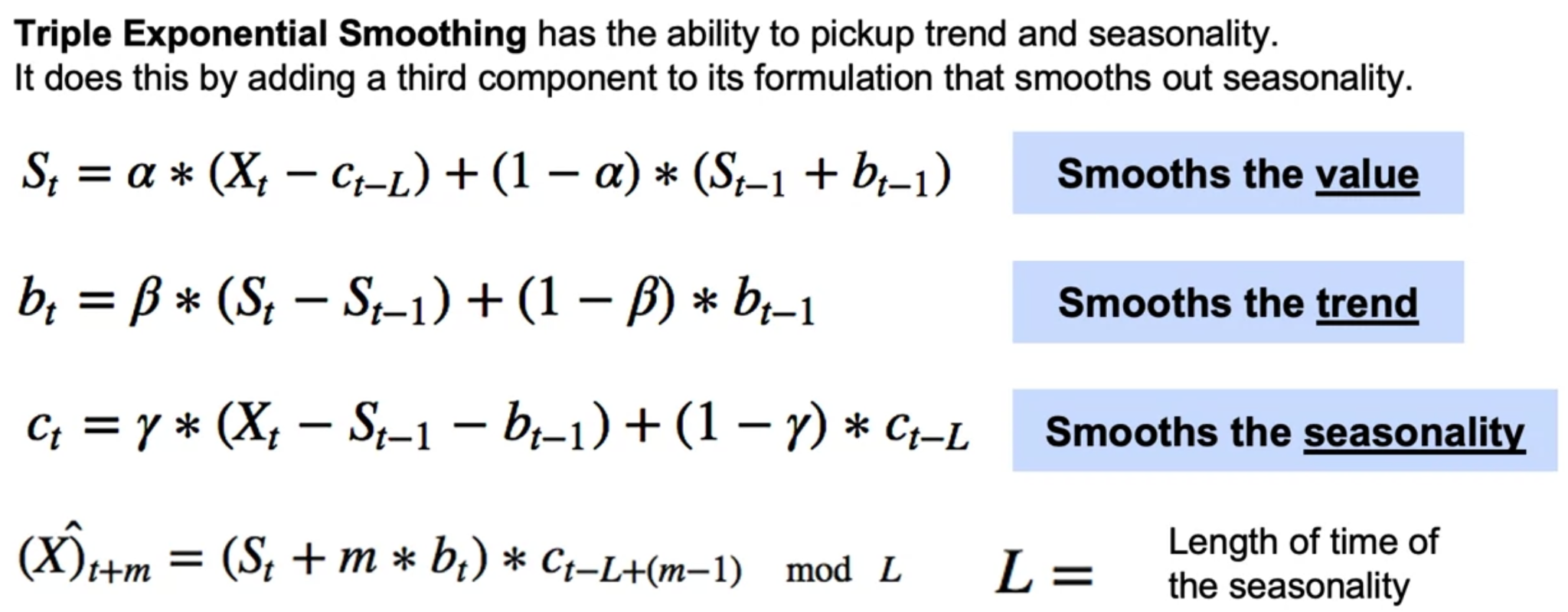
**Single exponential smoothing**

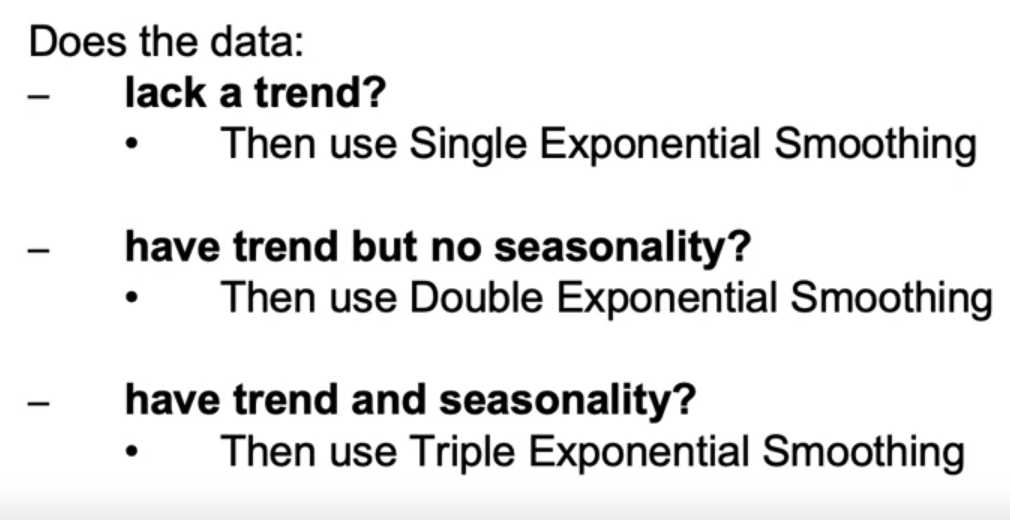
****

It produces value almost equal to latest value and does not capture trend or seasonality



However it does not pick up seasonality



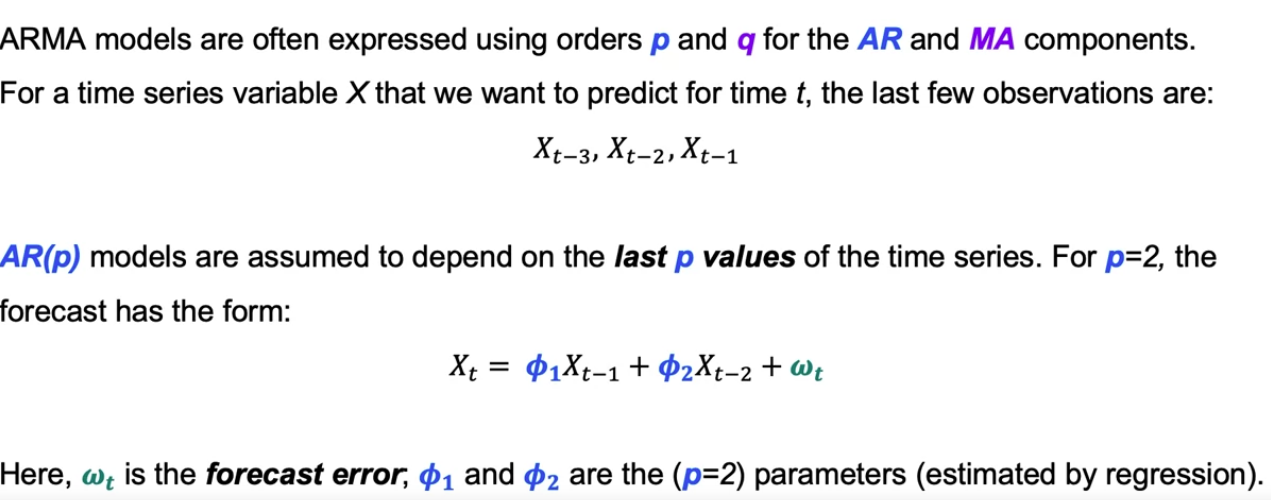


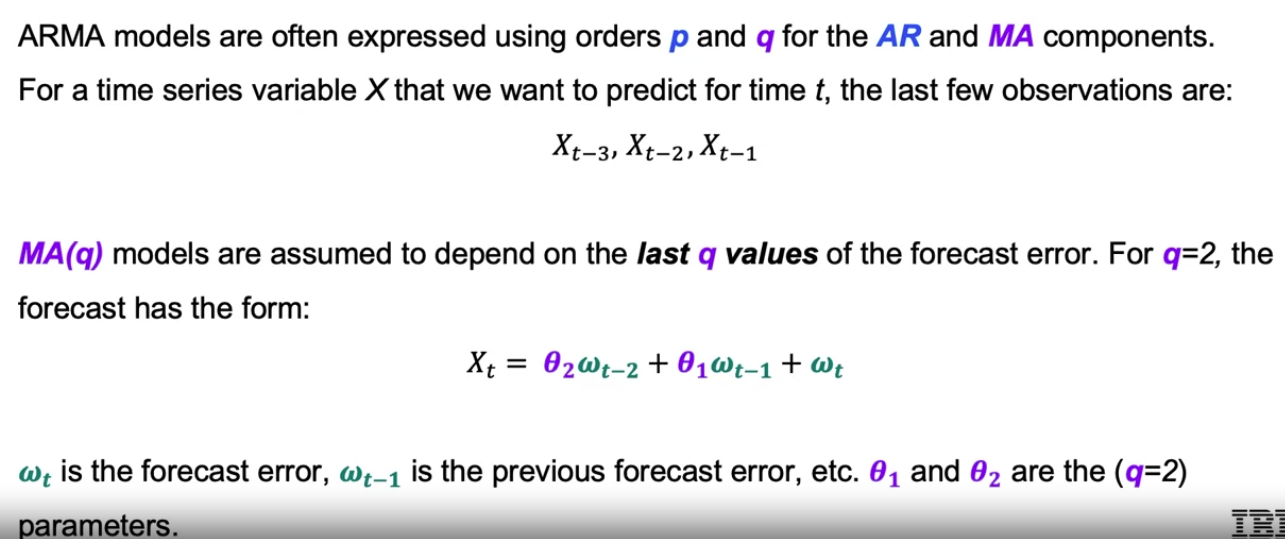
ARMA Models (Box Jenkins approach):

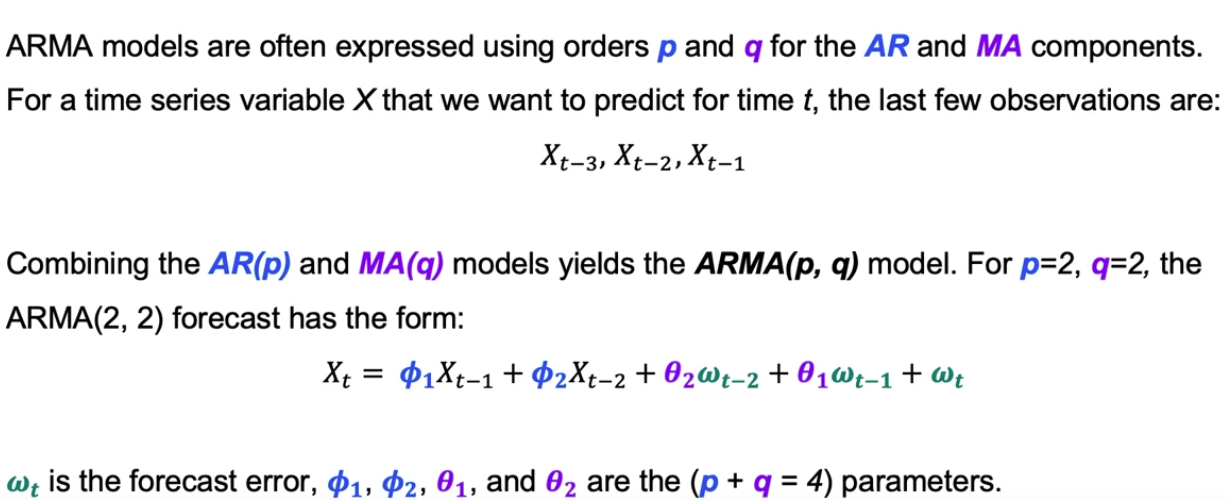
ARMA combines 2 models

AR - Autoregressive model which anticipates series dependence on its own past values so if value is higher than mean then the following value will also be higher than mean

MA - Moving average models which anticipate on past forecast errors - if value is off from mean then forecast will also be off







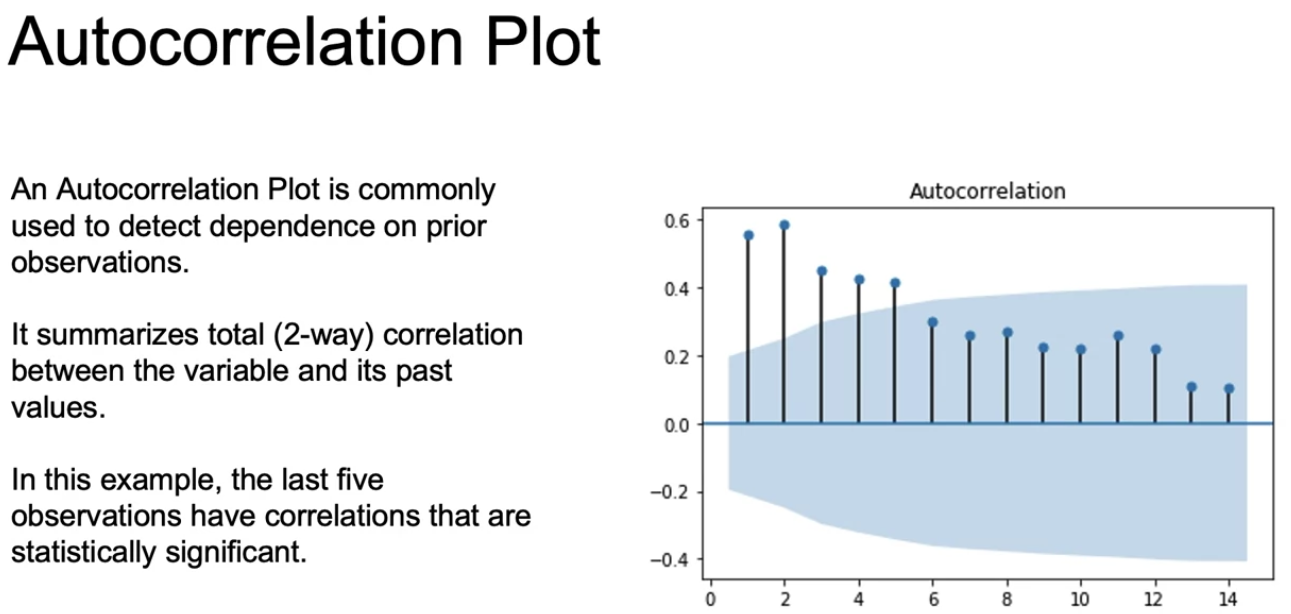
Assumptions of ARMA models:

Time series is assumed to be stationary.

At Least 100 observations should be fitting ARMA models.

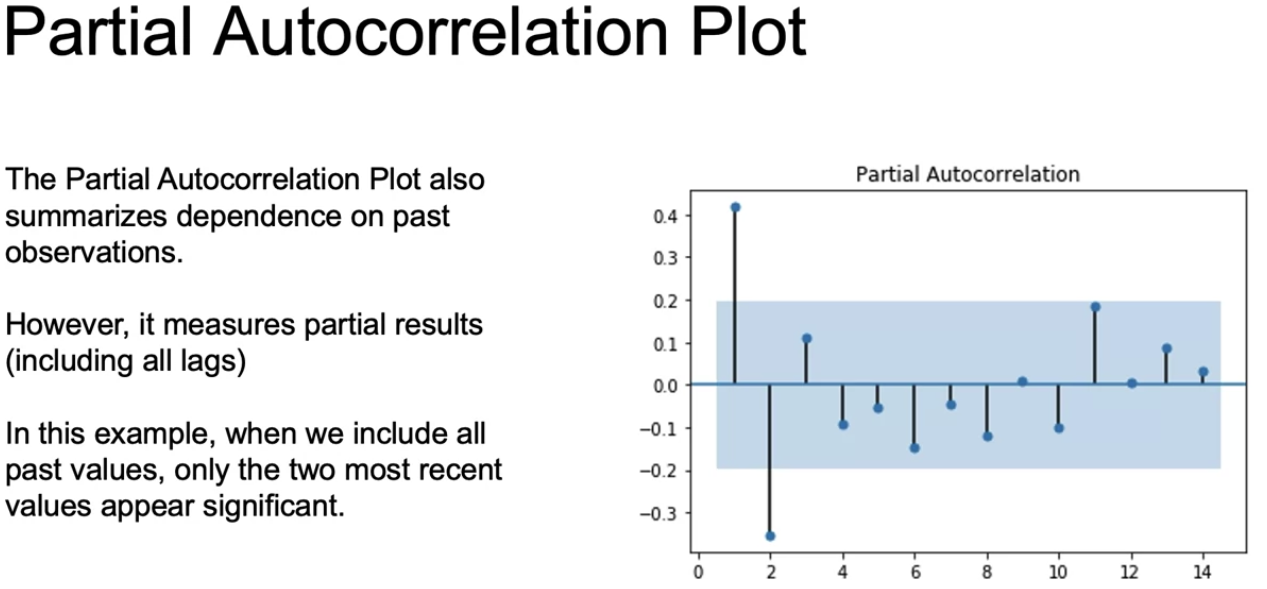
Determining Seasonality in series:

1. Autocorrelation and partial autocorrelation plots
2. Seasonal subseries plot
3. Intuition - Depending on seasonal sales, holidays etc

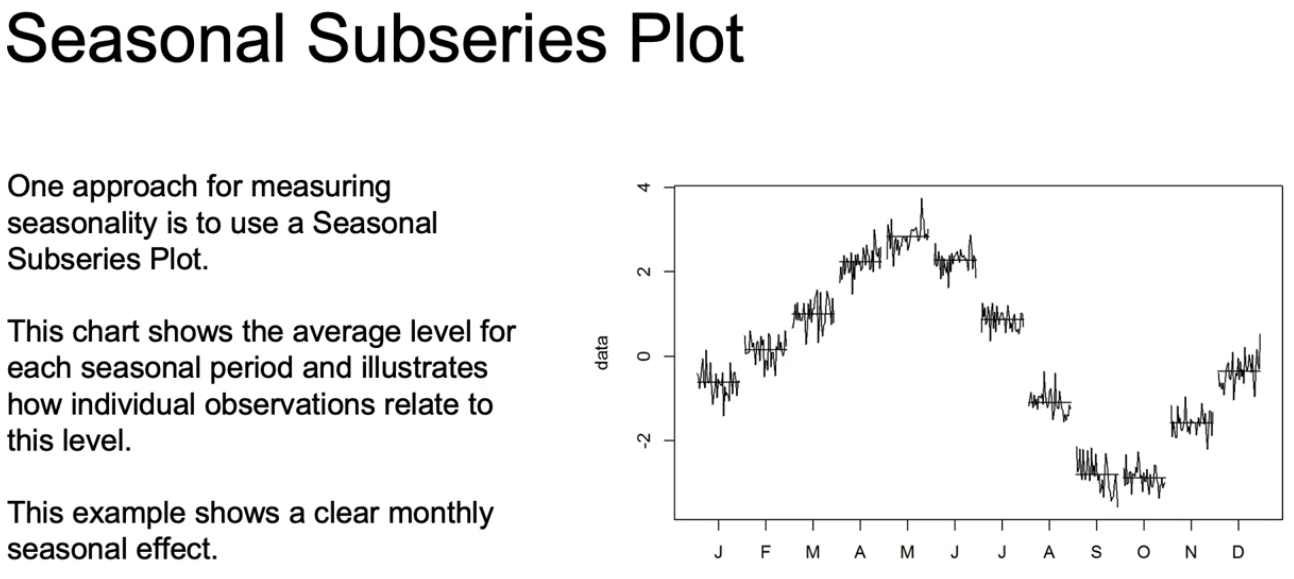


But if there is a correlation with lag1, there is automatically correlation with lag2 and Autocorrelation plot cannot be used to understand that.

Blue shaded region is the region of statistical significance



Partial autocorrelation independently measures the correlation by considering correlations due to precious lag.

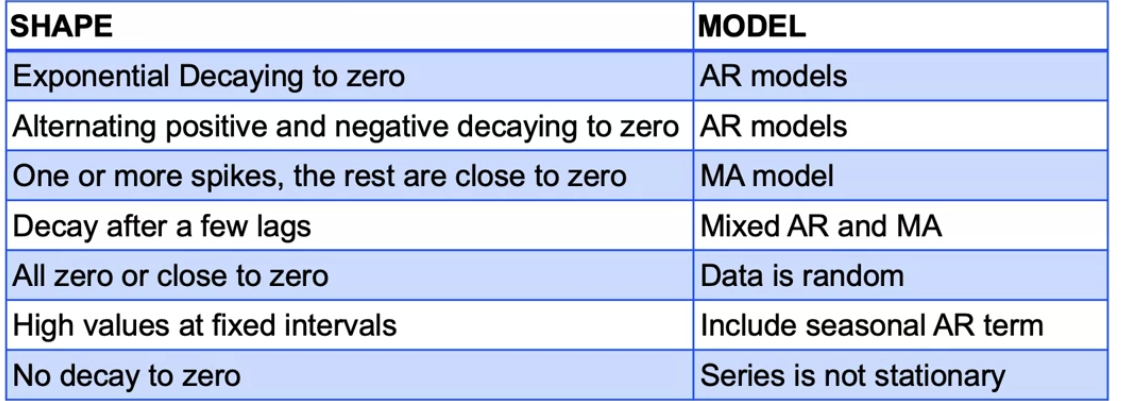


For identifying the order of AR and MA model i.e. p & q, we can treat them as hyper parameters and use grid search or cross validation etc or use Autocorrelation or Partial autocorrelation plots.

For AR models, (p) is determined using a partial autocorrelation plot, such that partial autocorrelation becomes insignificant of p + 1 and beyond.

For MA models, lag q is determined by autocorrelation plots , such that partial autocorrelation becomes insignificant of q + 1 and beyond

We check ACF to determine which model to use and based on shape we choose:



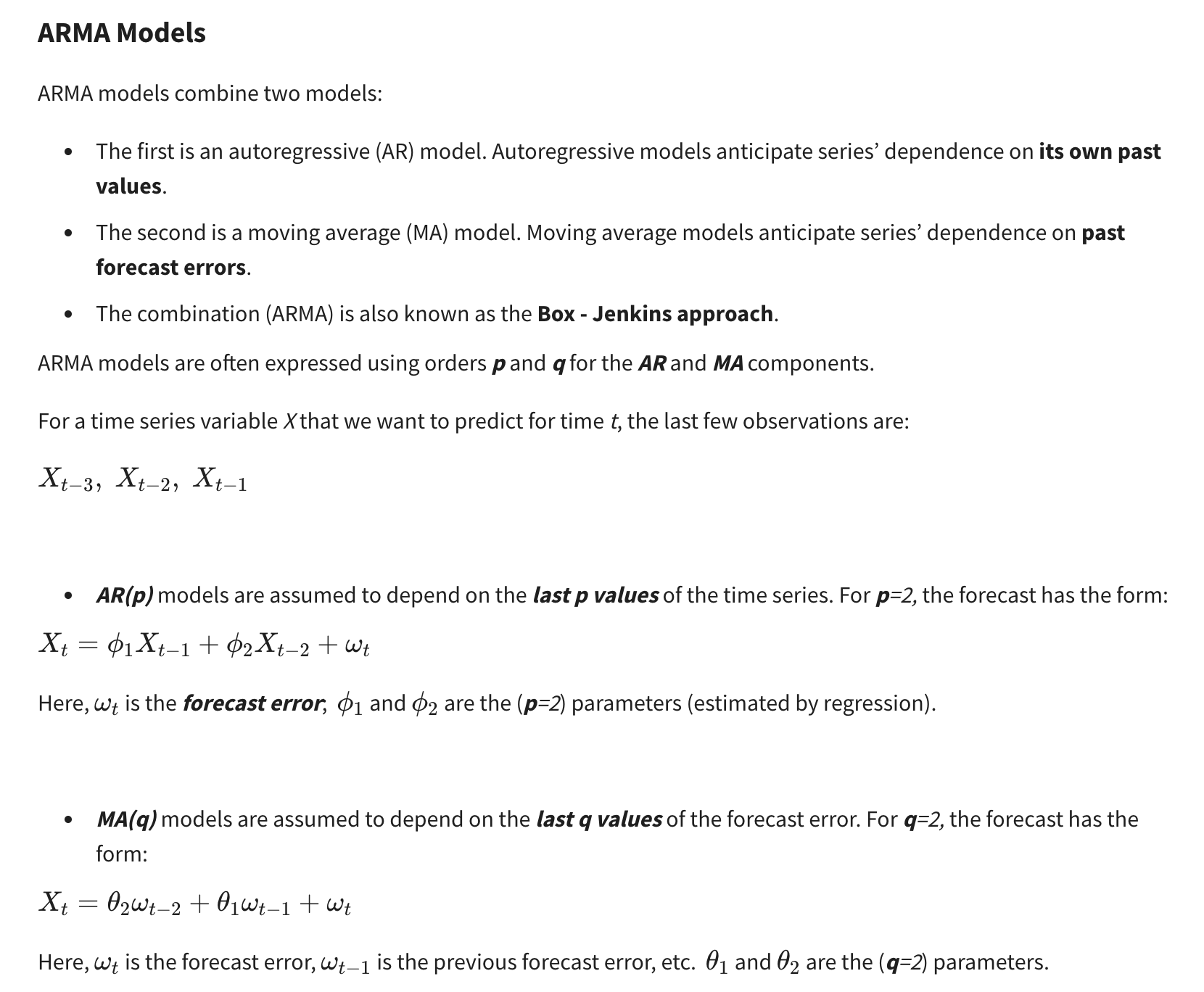
For #3 if we have one more spikes, rest are close to 0 it means errors are correlated and not the values

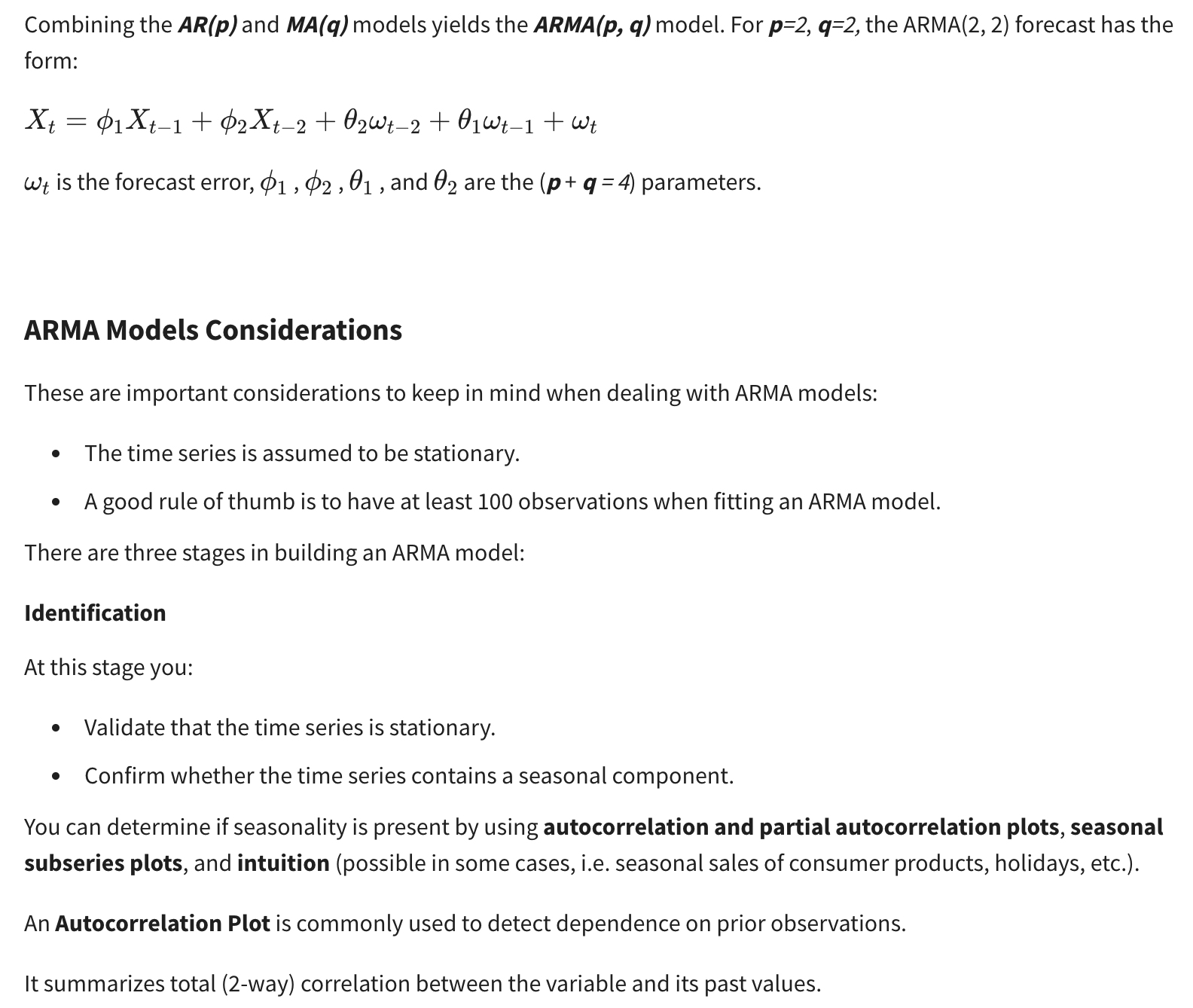
For #7 if in ACF plot we are not decaying it means the mean is not constant & the series is stationary.

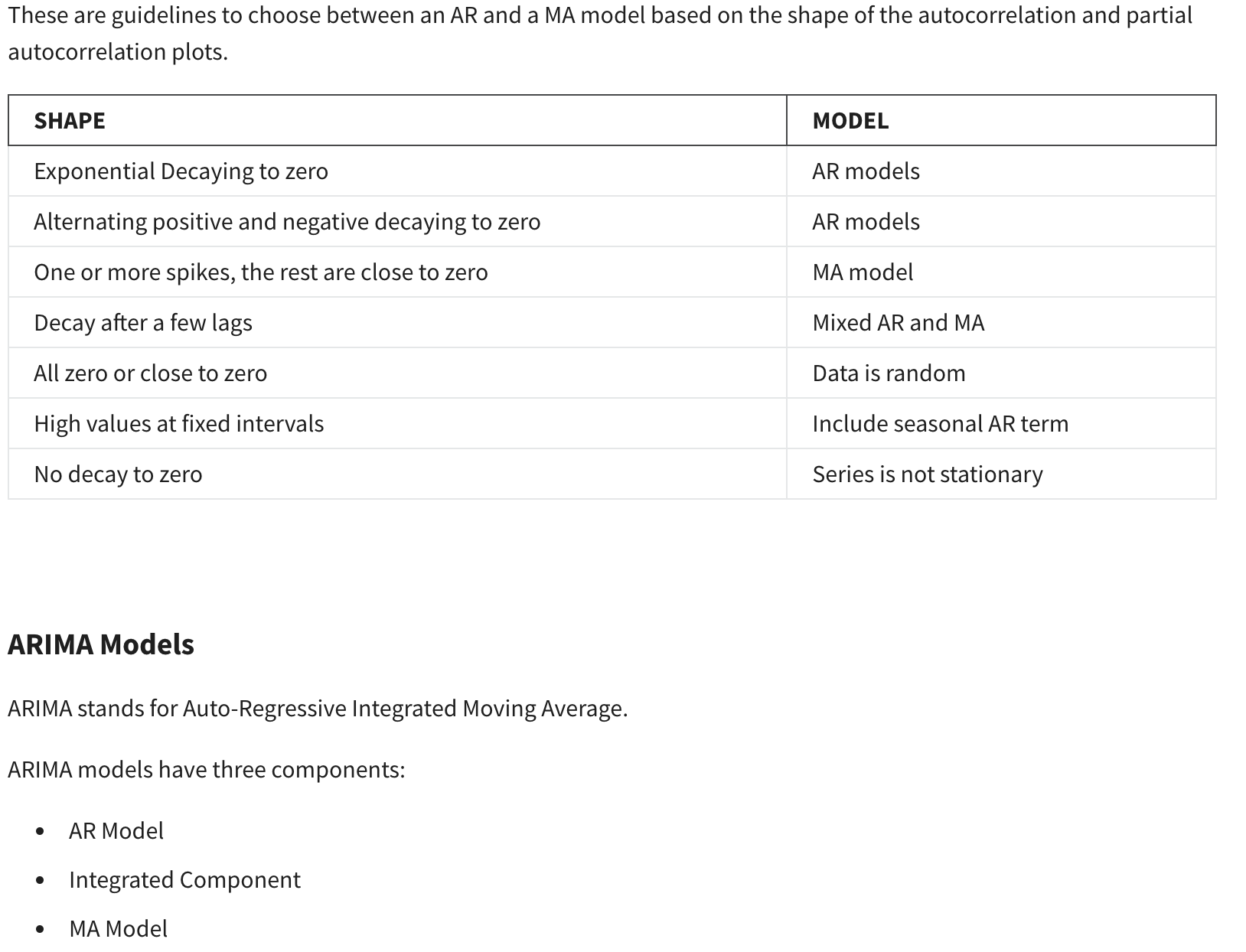
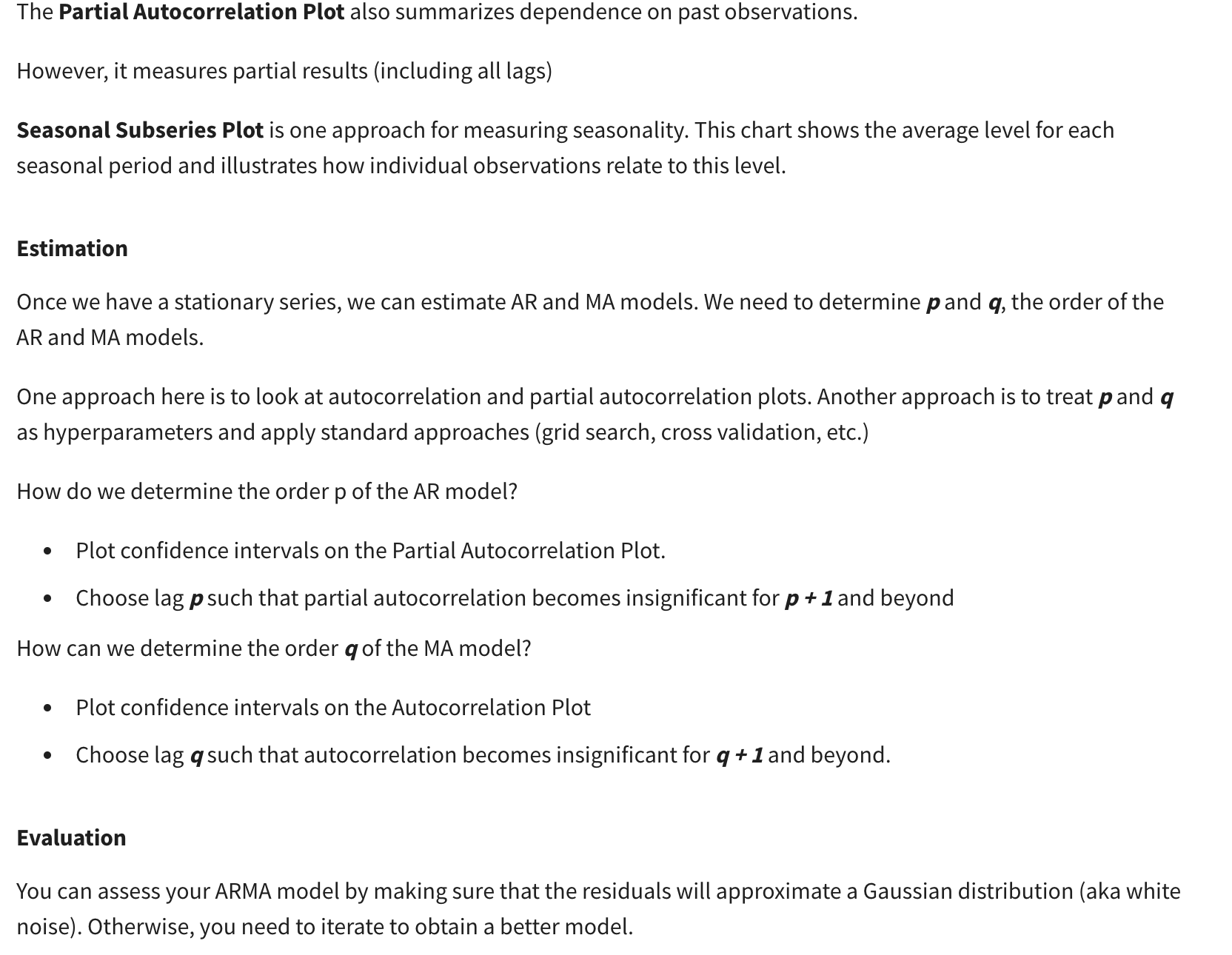
Estimating parameters of ARMA model requires complex methods Non-linear least square and Maximum likelihood estimation are common approaches. Most software fits it for us.

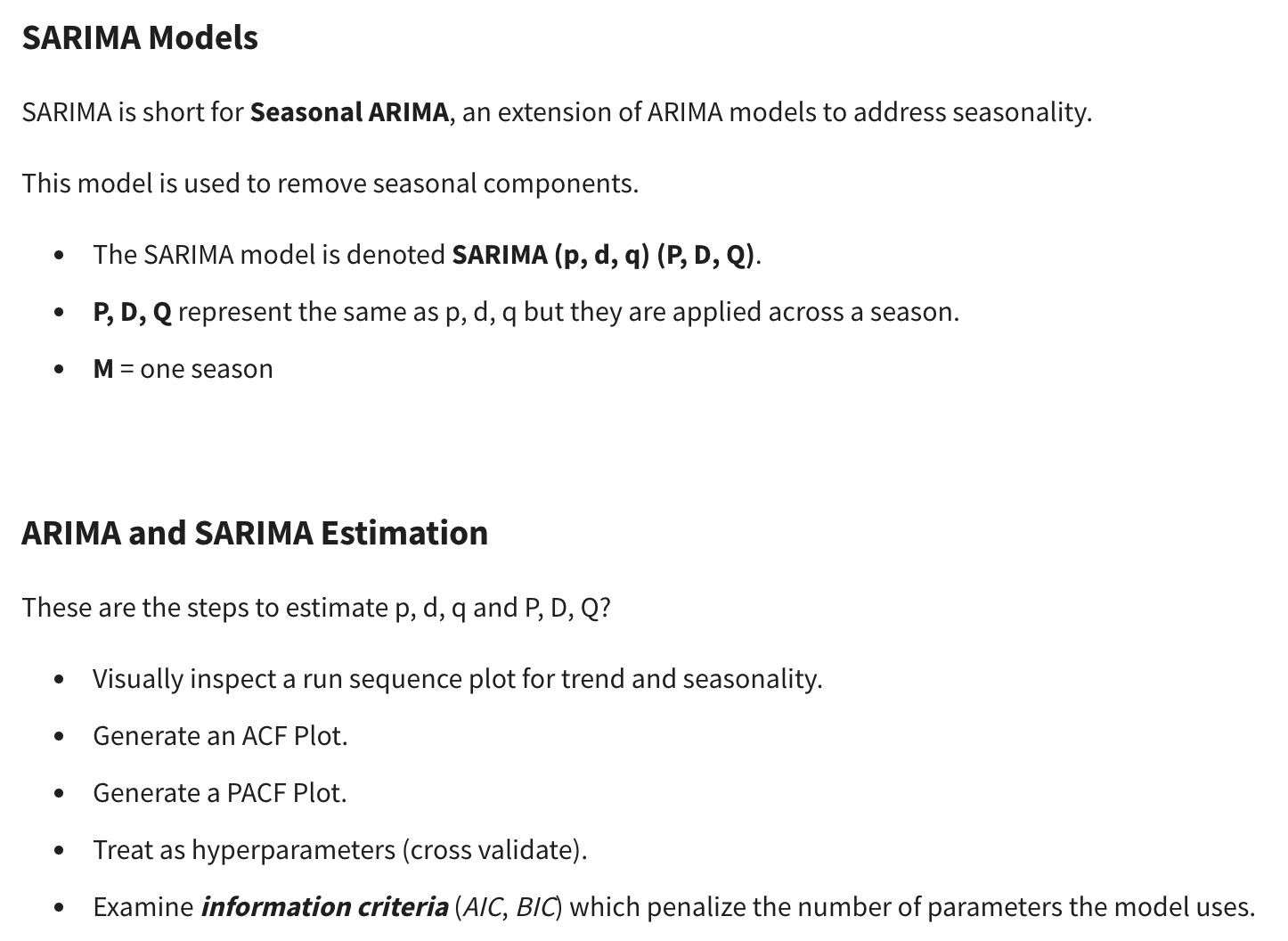
If ARMA model is good, residuals will be a random white noise with Gaussian distribution (Random noise is normally distributed)

# ARMA, ARIMA and SARIMA models









# Deep Learning for Time Series Forecasting

Neural networks offer several benefits over traditional time series forecasting models, including:

* Automatically learn how to incorporate series characteristics like trend, seasonality, and autocorrelation into predictions.
* Able to capture very complex patterns.
* Can simultaneously model many related series instead of treating each separately.

Some disadvantages of using Deep Learning for Time Series Forecasting are:

* Models can be complex and computationally expensive to build (GPUs can help).
* Deep Learning models often overfit.
* It is challenging to explain / interpret predictions made by the model (“black box”).
* Tend to perform best with large training datasets.

**Recurrent neural networks (RNNs)** map a sequence of inputs to predicted output(s).

* Most common format is “**many-to-one**”, that maps an input sequence to one output value.
* Input at each time step sequentially updates the RNN cell’s “**hidden state**” (“**memory**”).
* After processing the input sequence, the hidden state information is used to predict the output.

RNNs often struggle to process long input sequences. It is mathematically difficult for RNNs to capture long-term dependencies over many time steps, which is a problem for Time Series, [as sequences are often hundreds of steps](https://www.coursera.org/learn/time-series-survival-analysis/supplement/T37OF/summary-review). Another type of Neural Networks, Long short-term memory networks (LSTMs) can mitigate these issues with a better memory system

**Long short-term memory networks** share RNNs’ conceptual structure.

* LSTM cells have the same role as RNN cells in sequential processing of the input sequence.
* LSTM cells are internally more complex, with gating mechanisms and two states: a hidden state and a cell state.

Long short-term memory networks regulate information flow and memory storage.

* LSTM cells share forget, input, and output gates that control how memory states are updated and information is passed forward.
* At each time step, the input and current states determine the gate computations.

**LSTMs vs RNNs**

LSTMs are better suited for handling long-term dependencies than RNNs. However, they are much more complex, requiring many more trainable weights. As a result, LSTMs tend to take longer to train (slower backpropagation) and can be more prone to overfitting.

These are some guidelines on how to choose LSTMs or RNNs in a Forecasting task:

Always consider the problem at hand:

* If sequences are many time steps long, an RNN may perform poorly.
* If training time is an issue, using a LSTM may be too cumbersome.
* Graphics processing units (GPUs) speed up all neural network training, but are especially recommended when training LSTMs on large datasets.

# Survival Analysis

**Survival Analysis** focuses on estimating the length of time until an event occurs. It is called ‘survival analysis’ because it was largely developed by medical researchers interested in estimating the expected lifetime of different cohorts. Today, these methods are applied to many types of events in the business domain.

Examples:

* How long will a customer remains on books before churning
* How long until equipment needs repairs

**Survival Analysis** is useful when we want to measure the risk of events occurring and our data are Censored.

* This can be referred to as failure time, event time, or survival time.
* If our data are complete and unbiased, standard regression methods may work.
* Survival Analysis allows us to consider cases with incomplete or censored data.

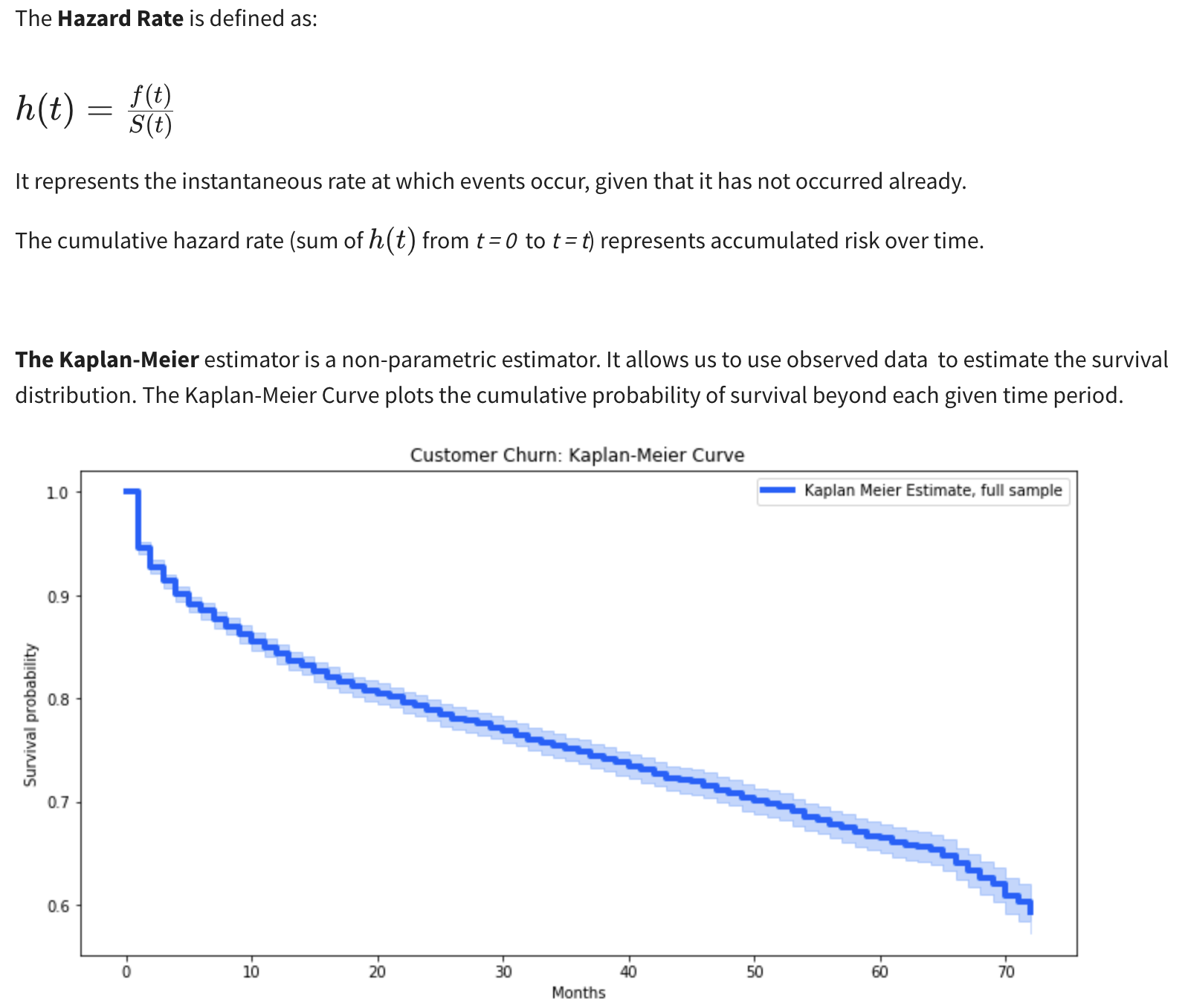
The **Survival Function** is defined as

*S*(*t*)=*P*(*T*>*t*) . Itmeasures the probability that a subject will survive past time *t*.

This function:

* Is decreasing (non-increasing) over time.
* Starts at 1 for all observations when *t=0*
* Ends at 0 for a high-enough *t*

The **Hazard Rate** is defined as:



Using the Kaplan-Meier Curve allows us to visually inspect differences in survival rates by category. We can use Kaplan-Meier Curves to examine whether there appear to be differences based on this feature.

To see whether survival rates differ based on number of services, we estimate **Kaplan-Meier** curves for different groups.

## **Survival Analysis Approaches**

The **Kaplan-Meier approach** provides sample averages. However, we may want to make use of individual-level data to predict survival rates.

Some well-known Survival models for estimating Hazard Rates include these **Survival Regression** approaches. These methods:

* Allow us to generate estimates of total risk as a function of time
* Make use of censored and uncensored observations to predict hazard rates
* Allow us to estimate feature effects

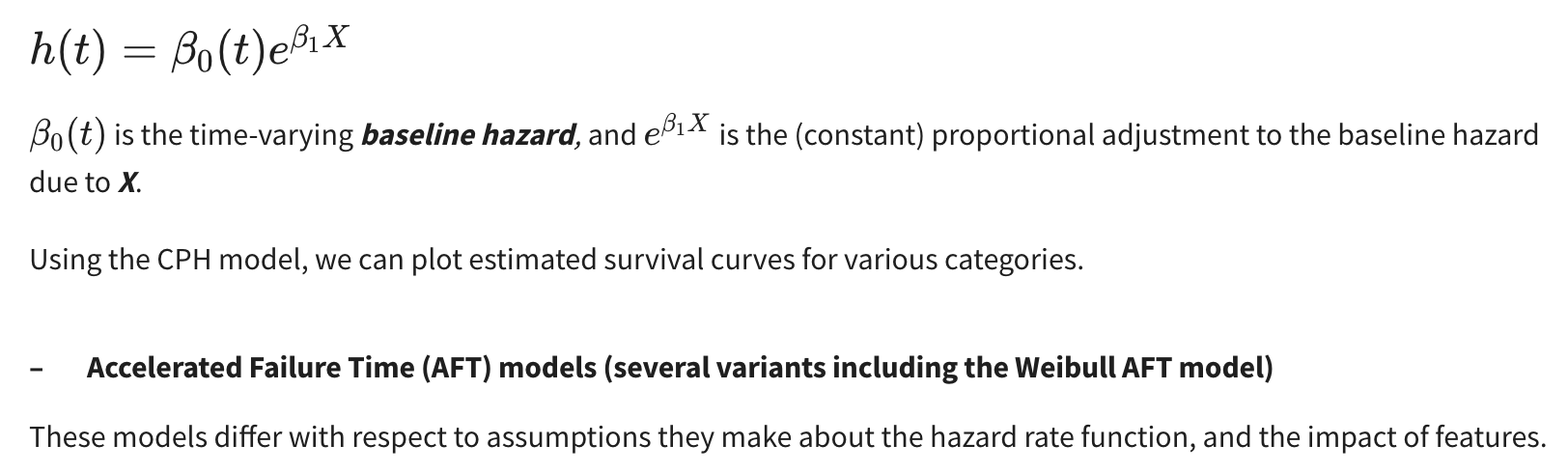
Although these methods use time, these methods are not generally predicting a time to an event, rather predicting survival risk (or hazard risk) as a function of time.

### **– The Cox Proportional Hazard (CPH) model**

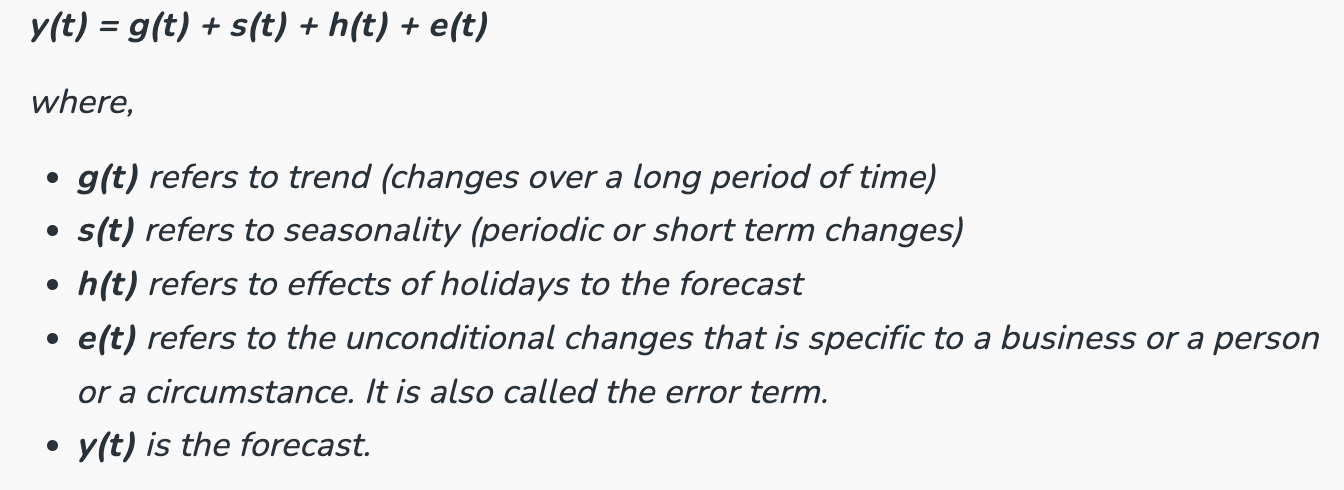
This is one of the most common survival models. It assumes features have a **constant proportional impact** on the hazard rate.

For a single non-time-varying feature *X*, the hazard rate

*h*(*t*) is modeled as:



# FB Prophet:



1. **Model:** Decomposible additive model where nonlinear trends fit seasonality. It also considers holidays
2. Prophet is automated, it by default uses a **linear model** to forecast, but when we require saturation for e.g. when a growth has saturation i.e. max/min value that can be achieved ***(carrying capacity)***, we can use **logistic growth** model where we can specify cap/floor. This cap can be specified for each data point and need not be a constant.
3. **Changepoints:** Prophet automatically detects the change points (abrupt changes in trend), it uses L1 regularization to make sure only significant change points are considered.
   1. We can also control when we see that it is overfitting or not detecting a change point. By default only the first 80% of time series have trend points and the next 20% do not - to avoid overfit, we can change that by changing **changepoint\_range**.
   2. **Trend flexibility:** If trend changes are overfit or underfit, we can adjust that using **changepoint\_prior\_scale**, (By default it is 0.05) Increasing it will make trend more flexible and decreasing will make it less flexible (range can be 0.001 to 5)
4. **Cross validation:** Cutoff points are set and the model is trained up to each and then error is measured for future points: We can specify the forecast horizon, size of initial training period and spacing between cutoff dates. Cutoff dates can also be specified as lists. Plot cross validation can also be used with a rolling window. Cross validation can be parallelised using parallel="processes". Cross validation can be used for hyperparameter tuning, see example
5. **Holidays:** A data frame for holidays can be created and passed in to holiday parameter. In Forecast components, we can observe this. Built in country holidays can also be used by specifying the country name
6. **Seasonality:** Order of curve for seasonality can also be set. Higher the order, seasonality will overfit and try to determine some part of noise also. By default it is 10
7. Requires time series column to be named as ds and variable to be y
8. **Conditional Seasonality:** For e.g. if weekly seasonality behaves differently during different seasons, we can add a column to create a flag that seasonality is to be fixed and removed & thus fit different types of seasonalities.
9. **Prior scale (Seasonal & Holiday):** Prior scale changes the impact of seasonality and holidays
10. **Regressors:** Regressors can also be added but should be in both train and predict. Coeff of regressors can be determined from (from prophet.utilities import regressor\_coefficients)
11. Ds to be in date time format
12. **Uncertainty levels:** There are 3 sources of uncertainty : trend, seasonality & additional observation noise.
13. Automatically detects the seasonality if 2 cycles of data are present.
14. In model.plot, dots are the real data, blue line is the predicted data & light blue is confidence interval
15. Model.components shows components
16. Cross validation:

# Hyperparamter tuning for Prophet:

| **Parameter** | **Possible range** | **Comments** |
| --- | --- | --- |
| Change point prior scale | [0.001,0.5] | 0.005 is default and works mostly, increasing it will make trend more flwxible (increase overfitting)  If we make it more flexible, can accomodate abrupt changes |
| Change point range | [0.1] | % of time series in which trend is allowed to change  By default is it 0.8 and not preferred to be tuned but can be changed in range [0.8, 0.95]  Recent abrupt changes can be captured if we change this |
| Seasonilty prior Scale | [0.01, 10] | Decides flexibility in capturing seasonality  Be deafault is 10 as there is no overfit usually in seasonality  As we decrease it, magnitude of seasonilty will be less |
| Seasonilty mode | Additive', 'Multiplicative' | Based on additive or multiplicative decomposition |
| Holiday prior scale | [0.01, 10] | By default is 10  Flexinility to fit holidays  Similar to seasonilty prior scale |
| Yearly, Weekly, Daily seasonality | auto, True, False | Should not be tuned  Turns on respective seasonality if this granularity of data is present  Seasonility prior scale should tuned rather than this |