ARIMA Overview

Python for Time Series

- We will now discuss one of the most common time series models, ARIMA.
- Many models are based off the ARIMA model, which stands for AutoRegressive Integrated Moving Average

Python for Time Series

- It is important to understand that ARIMA is not capable of perfectly predicting any time series data.
- Beginner students often want to directly apply ARIMA to time series data that is not directly a function of time, such as stock data.

Python for Time Series

- Stock price data for example has so many outside factors that much of the information informing the price of the stock won't be available with just the time stamped price information.

Python for Time Series

- ARIMA performs very well when working with a time series where the data is directly related to the time stamp, such as the airline passenger data set.
- In that data we saw clear growth and seasonality based on time.

Python for Time Series

- But it is important to keep in mind that an ARIMA model on that data wouldn't be able to understand any outside factors, such as new developments in jet engines, if those effects weren't already present in the current data.

Python for Time Series

- This is all to state that while ARIMA based models are extremely powerful tools, they are not magic, and a large part of using them effectively is understanding your data!

Python for Time Series

- ARIMA models can be complex!
- Make sure to make full use of the various links and extra resources presented throughout this section if you want to later use ARIMA models for other problems.

Python for Time Series

- AutoRegressive Integrated Moving Average (ARIMA) model is a generalization of an autoregressive moving average (ARMA) model.

Python for Time Series

- Both of those models (ARIMA and ARMA) are fitted to time series data either to better understand the data or to predict future points in the series (forecasting).

Python for Time Series

- ARIMA (Autoregressive Integrated Moving Averages)
    - Non-seasonal ARIMA
    - Seasonal ARIMA (SARIMA)
- Also understanding SARIMA with exogenous variables, such as SARIMAX.

Python for Time Series

- We will start by discussing non-seasonal ARIMA models and then move on to seasonal ARIMA models.
- Then we'll learn about more complex models built off of ARIMA.

Python for Time Series

- ARIMA models are applied in some cases where data show evidence of non-stationarity, where an initial differencing step (corresponding to the "integrated" part of the model) can be applied one or more times to eliminate the non-stationarity.

Python for Time Series

- Differencing is actually a very simple idea, but let's put it on hold for now, and talk a bit more about ARIMA!
- We'll touch back on differencing later on.
- Let's talk about the major components of ARIMA.

Python for Time Series

- Non-seasonal ARIMA models are generally denoted ARIMA(p,d,q) where parameters p, d, and q are non-negative integers.
- Let’s discuss what these three components are!

Python for Time Series

- Parts of ARIMA model
- AR (p): Autoregression
    - A regression model that utilizes the dependent relationship between a current observation and observations over a previous period

Python for Time Series

- Parts of ARIMA model
- I (d): Integrated.
    - Differencing of observations (subtracting an observation from an observation at the previous time step) in order to make the time series stationary.

Python for Time Series

- Parts of ARIMA model
- MA (q): Moving Average.
    - A model that uses the dependency between an observation and a residual error from a moving average model applied to lagged observations.

Python for Time Series

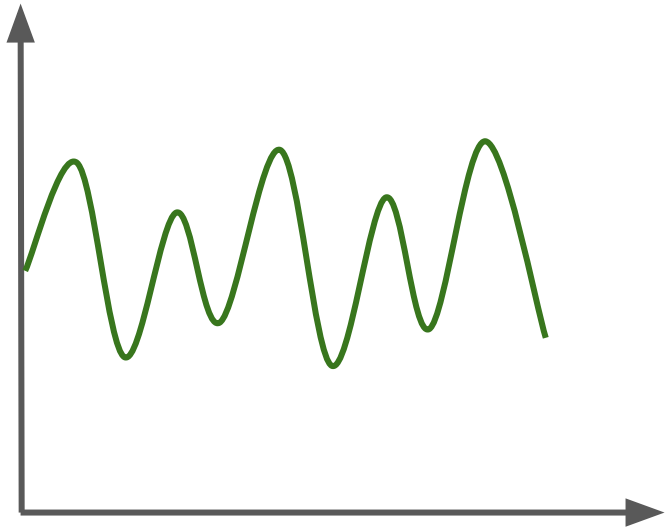
- Stationary vs Non-Stationary Data
    - To effectively use ARIMA, we need to understand Stationarity in our data.
    - So what makes a data set Stationary?
        - A Stationary series has constant mean and variance over time.

Python for Time Series

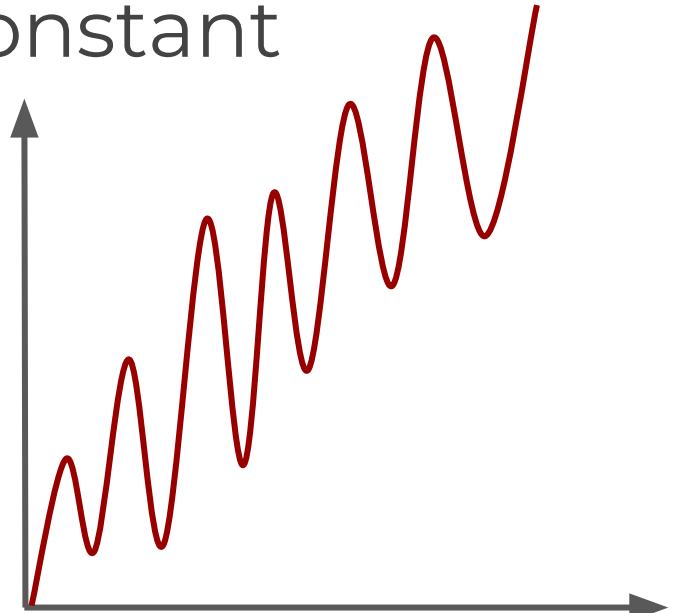
- A Stationary data set will allow our model to predict that the mean and variance will be the same in future periods.
- Let’s take a look at a few examples!

Python for Time Series

- Mean needs to be constant



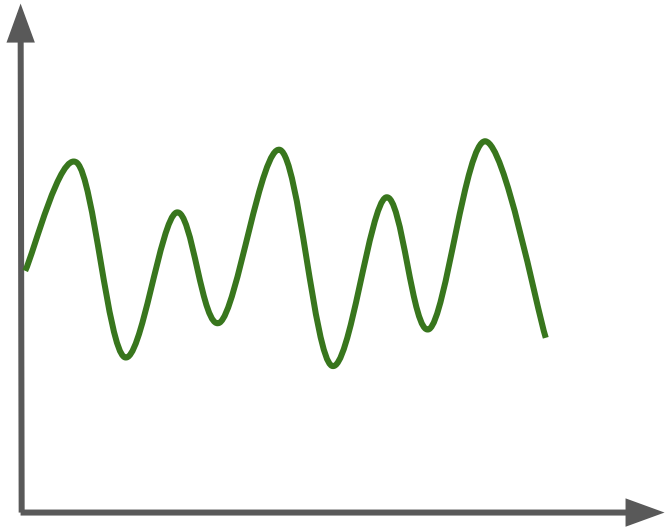
Stationary



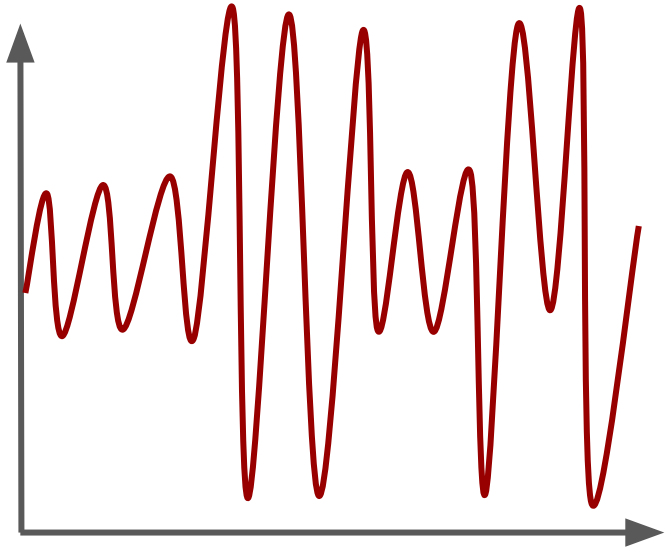
Non-Stationary

Python for Time Series

- Variance should not be a function of time



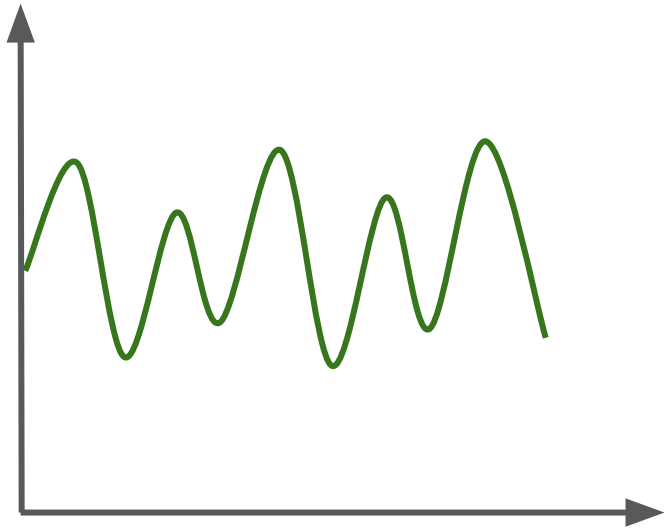
Stationary



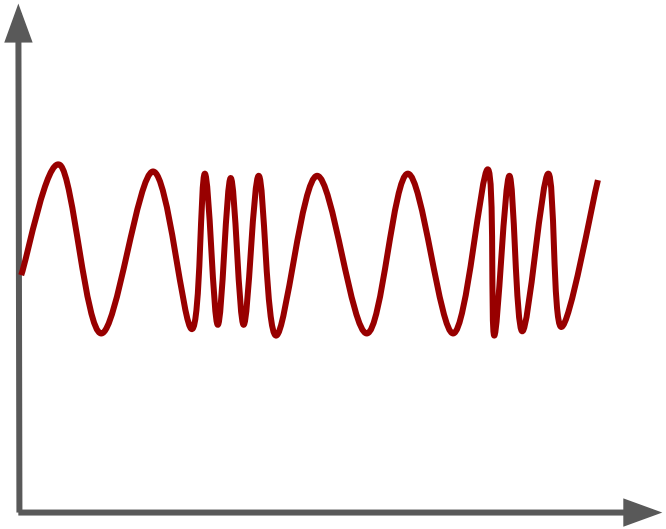
Non-Stationary

Python for Time Series

Covariance should not be a function of time



Stationary



Non-Stationary

Python for Time Series

- There are also mathematical tests you can use to test for stationarity in your data.
- A common one is the Augmented Dickey–Fuller test (we will see how to use this with Python's statsmodels)

Python for Time Series

- If you've determined your data is not stationary (either visually or mathematically), you will then need to transform it to be stationary in order to evaluate it and what type of ARIMA terms you will use.

Python for Time Series

- One simple way to do this is through “differencing”.
- The idea behind differencing is quite simple, let’s see an example…

Python for Time Series

Original Data

Time1	10
Time2	12
Time3	8
Time4	14
Time5	7

First Difference

Time1	NA
Time2	2
Time3	-4
Time4	6
Time5	-7

Second Difference

Time1	NA
Time2	NA
Time3	-6
Time4	10
Time5	-13

Python for Time Series

- You can continue differencing until you reach stationarity (which you can check visually and mathematically)
- Each differencing step comes at the cost of losing a row of data!

Python for Time Series

- For seasonal data, you can also difference by a season.
- For example, if you had monthly data with yearly seasonality, you could difference by a time unit of 12, instead of just 1.

Python for Time Series

- Another common technique with seasonal ARIMA models is to combine both methods, taking the seasonal difference of the first difference.

Python for Time Series

- With your data now stationary it is time to go back and discuss the p,d,q terms and how you choose them.
- There are two main ways to choose these p,d, and q terms.

Python for Time Series

- Method One (Difficult):
    - AutoCorrelation Plots and Partial AutoCorrelation Plots.
    - Using these plots we can choose p,d and q terms based on viewing the decay in the plot.

Python for Time Series

- Method One (Difficult):
    - These plots can be very difficult to read, and often even when reading them correctly, the best performing p,d, or q value may be different than what is read.

Python for Time Series

- Method Two (Easy but takes time):
    - Grid Search
    - Run ARIMA based models on different combinations of p, d, and q and compare the models for on some evaluation metric.

Python for Time Series

- Method Two (Easy but takes time):
    - Due to computational power becoming cheaper and faster, its often a good idea to use the built-in automated tools that search for the correct p, d, and q terms for us!

Python for Time Series

- Later on we will discuss SARIMA models designed to handle seasonal data.
- SARIMA is very similar to ARIMA, but adds another set of parameters (P, D, and Q) for the seasonal component.