General Forecasting Models

Python for Time Series

* So far we have learned foundational tools for working and analyzing time series data, such as pandas, numpy, and statsmodels.
* We have also worked with methods that can model time series behaviour.

Python for Time Series

- We now move onto forecasting time series data.
- This means we'll explore many different model types for various types of time series.

Python for Time Series

- Forecasting Procedure
  - Choose a Model
  - Split data into train and test sets
  - Fit model on training set
  - Evaluate model on test set
  - Re-fit model on entire data set
  - Forecast for future data

Python for Time Series

- Section Overview
  - Introduction to Forecasting
  - ACF and PACF plots
  - AutoRegression - AR
  - Descriptive Statistics and Tests
  - Choosing ARIMA orders
  - ARIMA based models

Let's get started!

Introduction to Forecasting

PART ONE

* We’ve already seen how the Holt-Winters methods can model an existing time series.
* Let’s now see how we can use that model on future dates, forecasting for dates that haven’t happened yet!

- We'll take a brief aside before Part Two of this lecture to discuss evaluating Forecasting Predictions.

Test Train Split

Python for Time Series

- Test sets will be the most recent end of the data.

Python for Time Series

- How do we decide how large the test data should be?

- The size of the test set is typically about 20% of the total sample, although this value depends on how long the sample is and how far ahead you want to forecast. The test set should ideally be at least as large as the maximum forecast horizon required.

* The test set should ideally be at least as large as the maximum forecast horizon required.
* Keep in mind, the longer the forecast horizon, the more likely your prediction becomes less accurate.

Evaluating Predictions

Evaluating Predictions

- Let's take a quick break to discuss evaluating forecasting results.
- After we fit a model on the training data, we forecast to match up to the test data dates.
- Then we can compare our results for evaluation.

Evaluating Predictions

- You may have heard of some evaluation metrics like accuracy or recall.
- These sort of metrics aren't useful for time series forecasting problems, we need metrics designed for **continuous** values!

Evaluating Predictions

- Let's discuss some of the most common evaluation metrics for regression:
  - Mean Absolute Error
  - Mean Squared Error
  - Root Mean Square Error

Evaluating Predictions

- Whenever we perform a forecast for a continuous value on a test set, we have two values:
  - **y -** the real value of the test data
  - **ŷ -** the predicted value from our forecast

Evaluating Predictions

- Mean Absolute Error (MAE)
  - This is the mean of the absolute value of errors.
  - Easy to understand

$$\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Evaluating Predictions

- An issue with MAE though, is that simply averaging the residuals won’t alert us if the forecast was really off for a few points.
- We want to be aware of any prediction errors that are very large (even if there only a few)

Evaluating Predictions

- Mean Squared Error (MSE)
  - This is the mean of the squared errors.
  - Larger errors are noted more than with MAE, making MSE more popular.

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Evaluating Predictions

- There is an issue with MSE however!
- Because we squared the residual, the units are now also squared.
- For example, if our forecast units was in dollars, the MSE returns back an error in units of **dollars squared**, which is hard to interpret!

Evaluating Predictions

- Root Mean Square Error (RMSE)
  - This is the root of the mean of the squared errors.
  - Most popular (has same units as y)

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Evaluating Predictions

- The most common question from students:
  - **“What is an acceptable RMSE value?”**
- Unfortunately, the answer is complicated and depends on your data!

Evaluating Predictions

- For example, if we have a RMSE of $20.00 USD for a dataset, is that good or bad?
- Depends on the data!
- That is great error range for predicting the future price of a house, but horrible for the future price of a candy bar!

Evaluating Predictions

- You will need to use your own judgement and compare the RMSE to the average values in your data set's test set.
- Then make a decision for the acceptability of the error.
- There are no 100% correct answers here!

Evaluating Predictions

- Another common question:
  - **“How do we evaluate a forecast for future dates?”**
  - Answer:
    - You can’t! Those dates haven’t happened yet so it is impossible to evaluate your predictions!

Evaluating Predictions

- This is why it is so important to perform the train test split on our data!
- Otherwise, we wouldn't have any intuition to how well the model can perform on dates it hasn't seen yet.

Evaluating Predictions

- Let's continue with our Introduction to Forecasting and show you how to grab these error metrics and forecast for future dates we haven't seen before!

Introduction to Forecasting

PART TWO

ACF and PACF

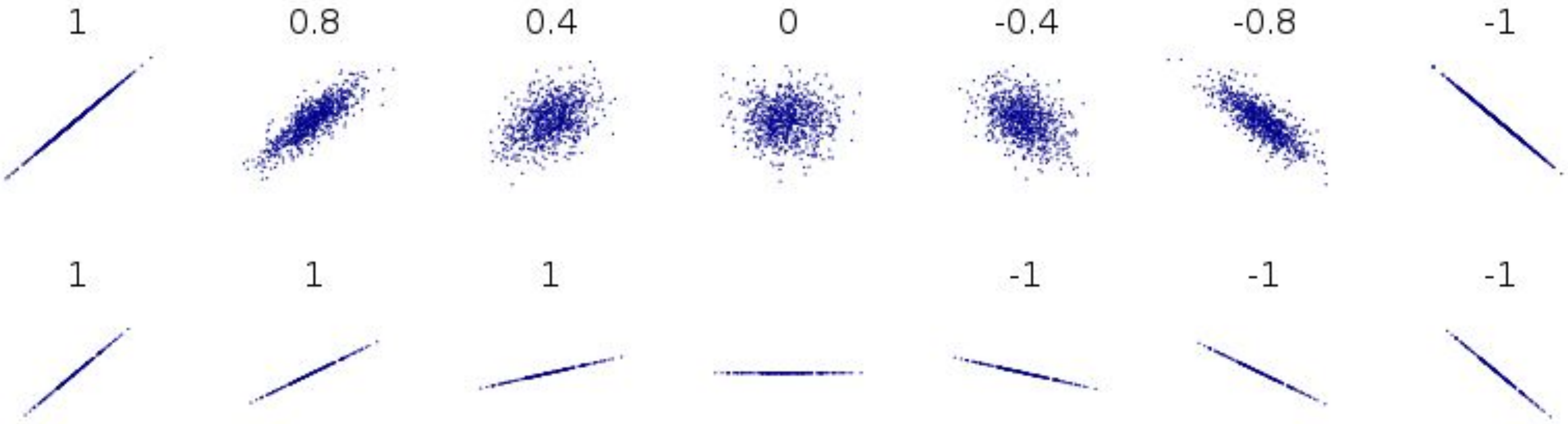
Theory

- Let's learn about 2 very useful plot types
  - ACF - AutoCorrelation Function Plot
  - PACF - Partial AutoCorrelation Function Plot
- To understand these plots, we first need to understand correlation!

- Correlation is a measure of the strength of the linear relationship between two variables.

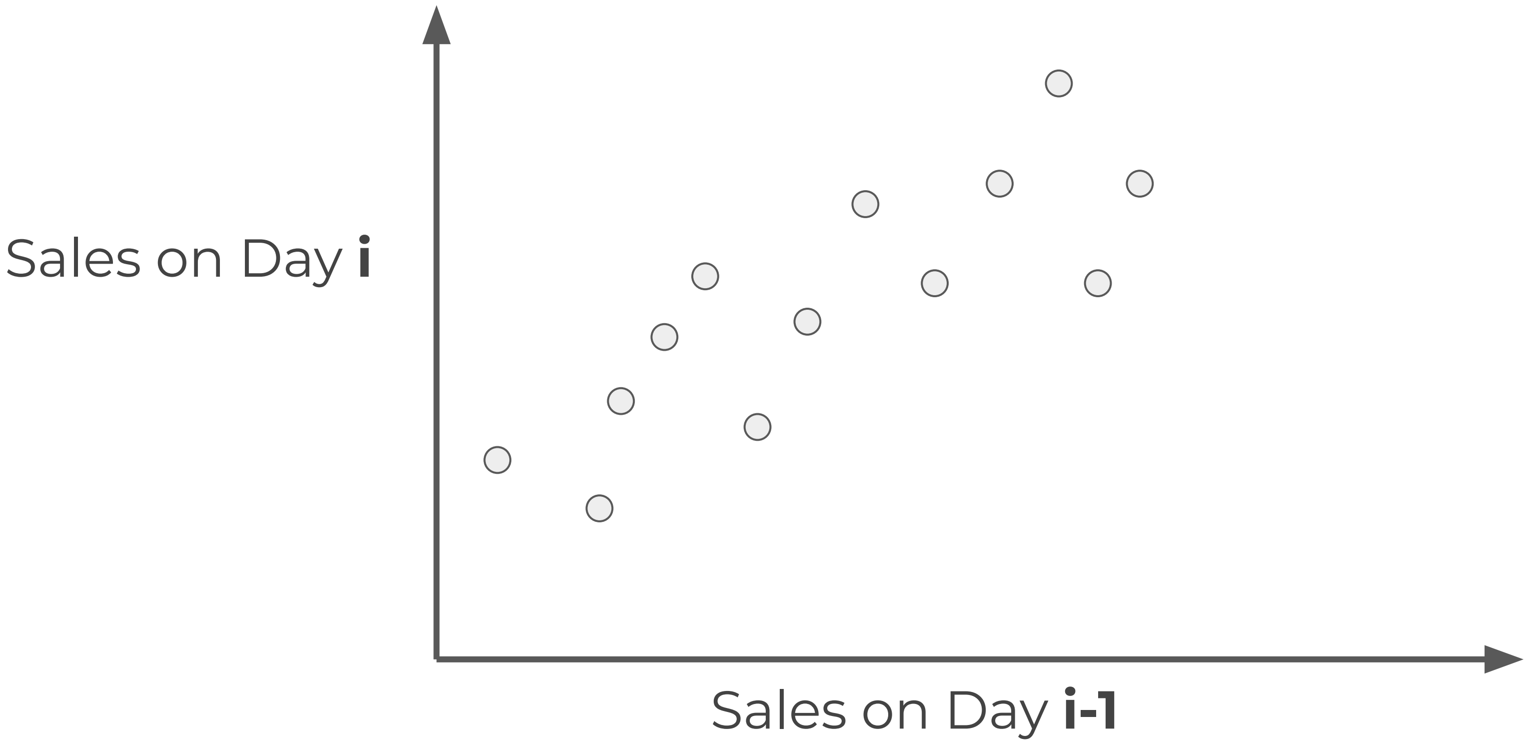
- The closer the correlation is to +1, the stronger the positive linear relationship
- The closer the correlation is to -1, the stronger the negative linear relationship.
- And the closer the correlation is to zero, the weaker the linear relationship, or association.

Python for Time Series

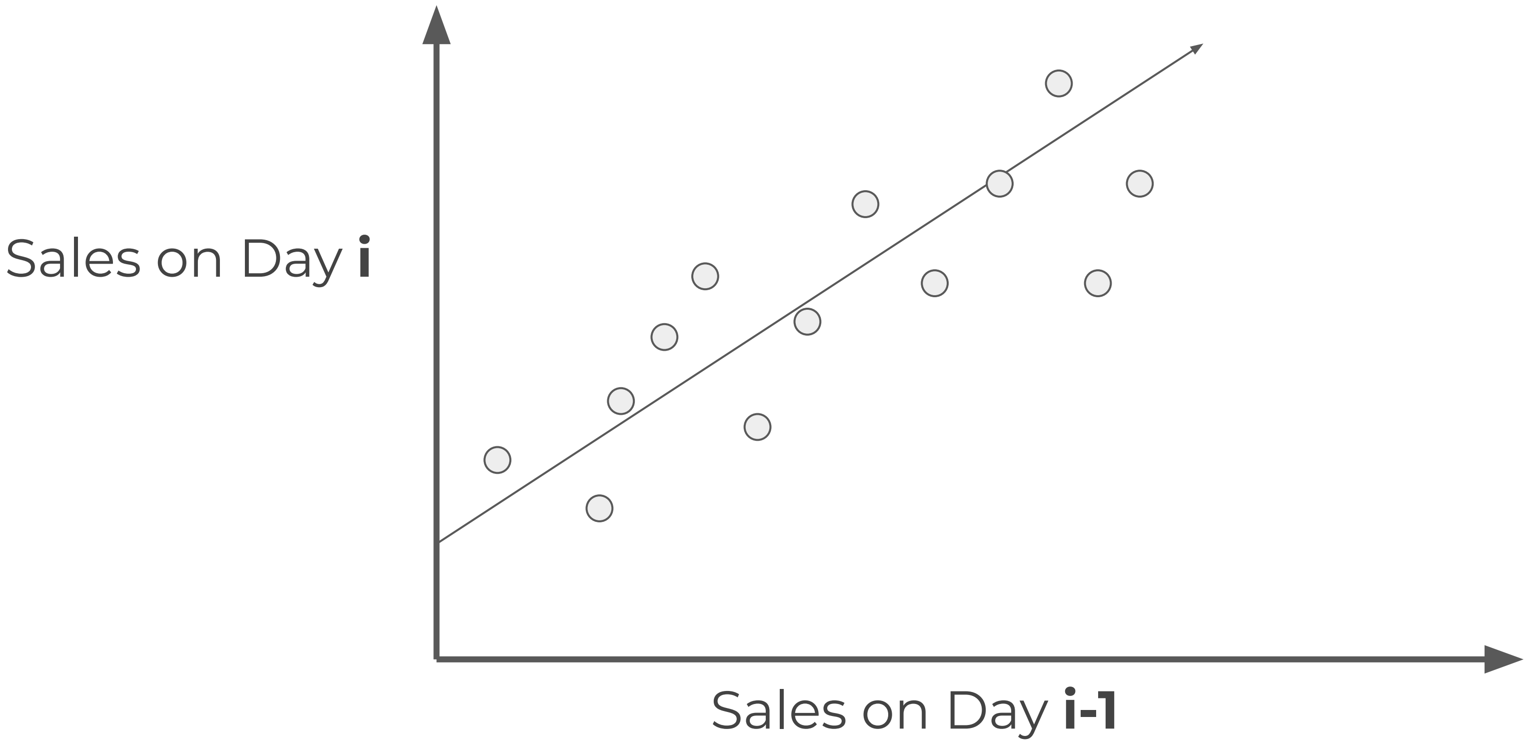


- An autocorrelation plot (also known as a Correlogram) shows the correlation of the series with itself, lagged by x time units.
- So the y axis is the correlation and the x axis is the number of time units of lag.

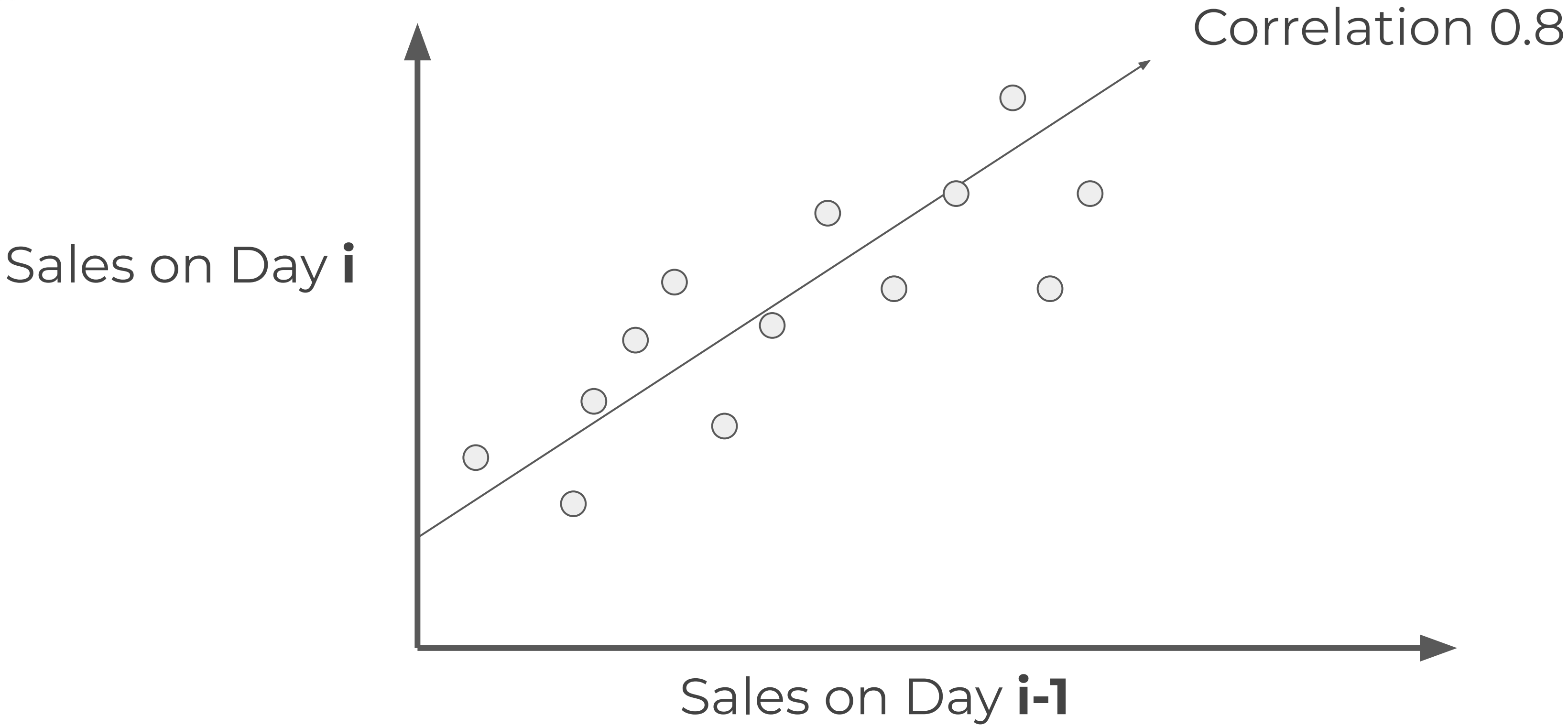
- Imagine we had some sales data.
- We can compare the standard sales data against the sales data shifted by 1 time step.
- This answers the question, “How correlated are today’s sales to yesterday’s sales?”


Sales on Day **i**
Sales on Day **i-1**

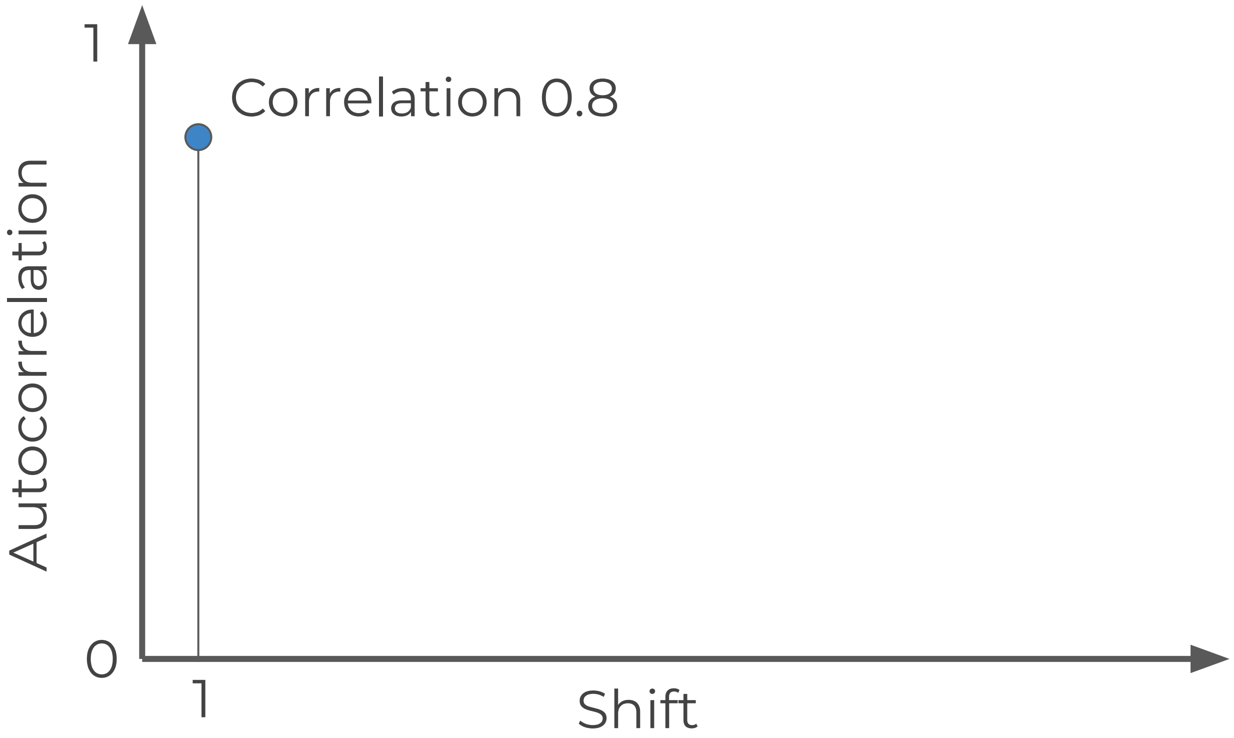
Python for Time Series



Python for Time Series

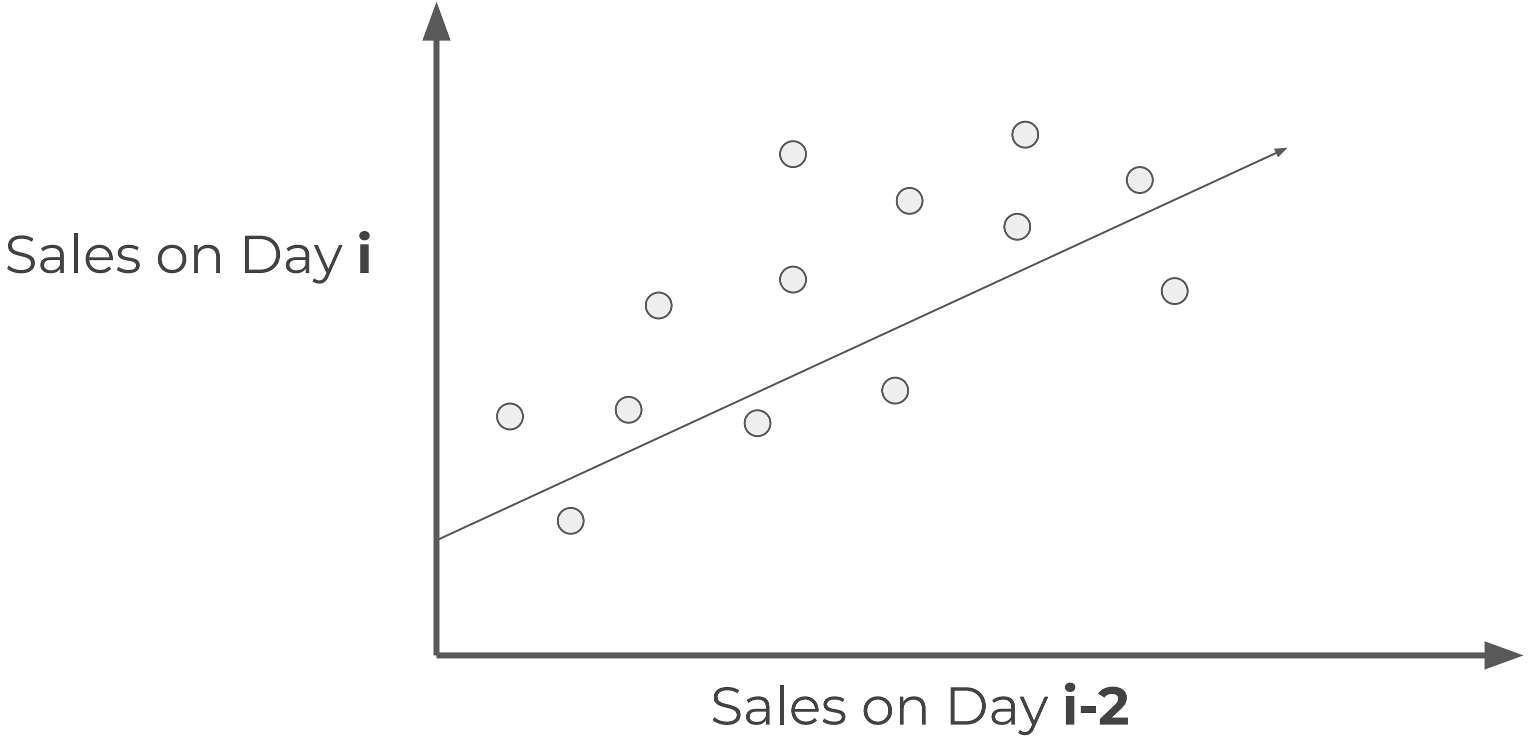


Python for Time Series

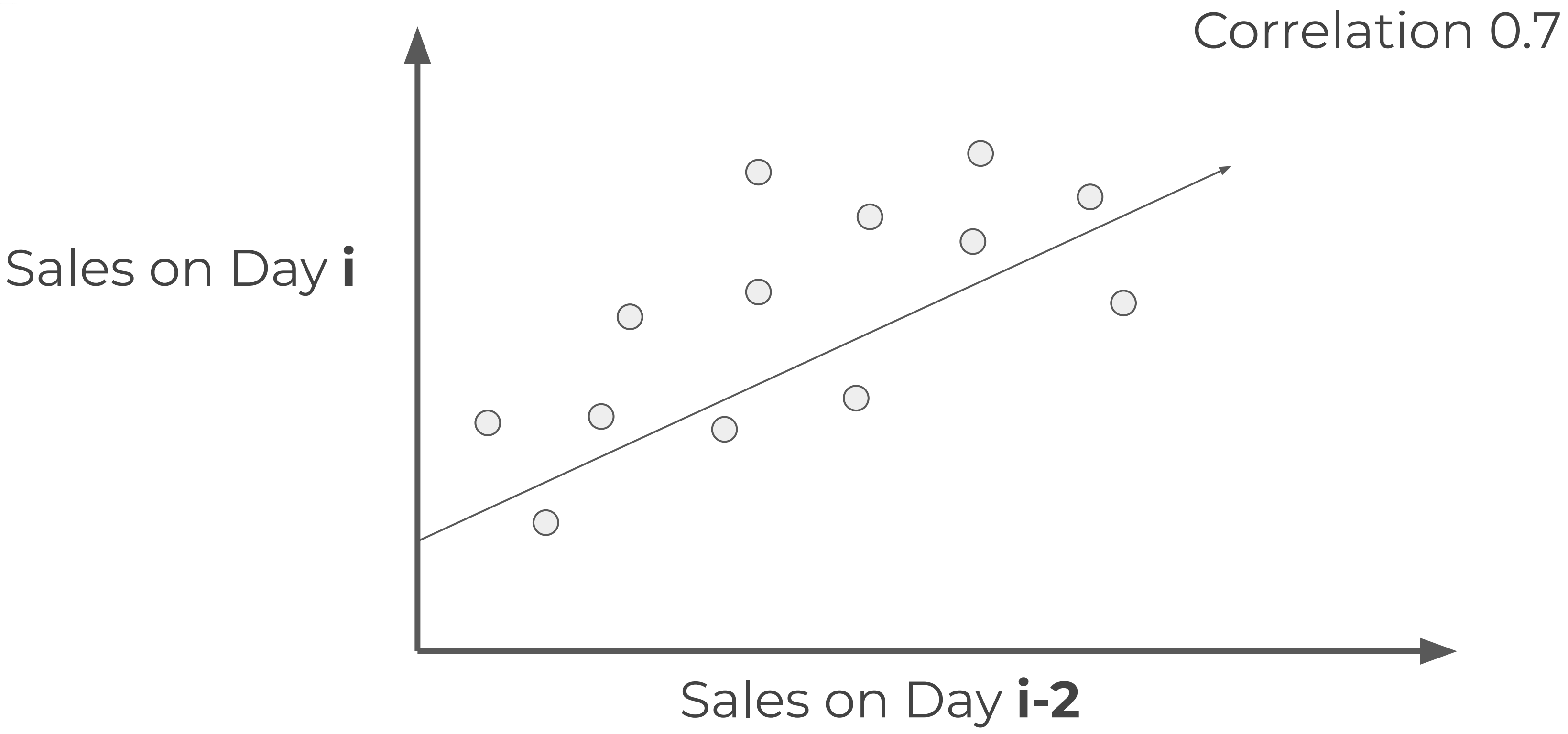


PIERIAN DATA

Python for Time Series

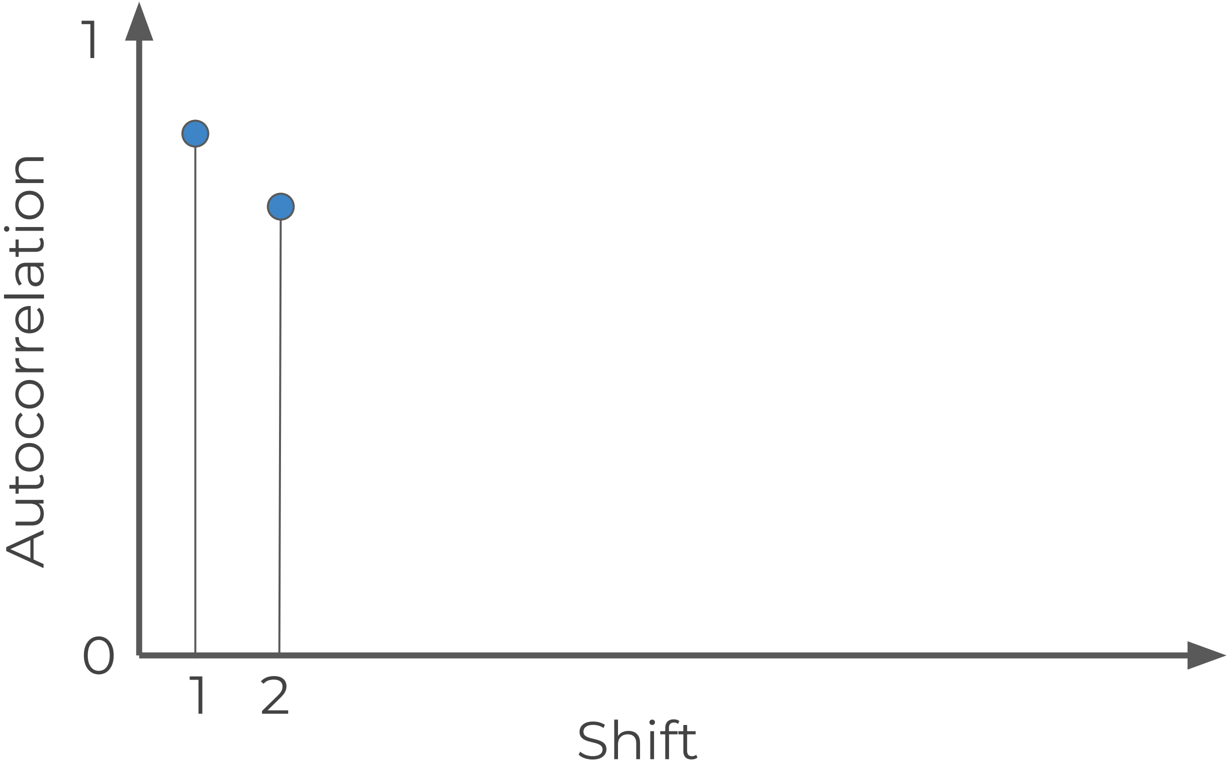


Python for Time Series



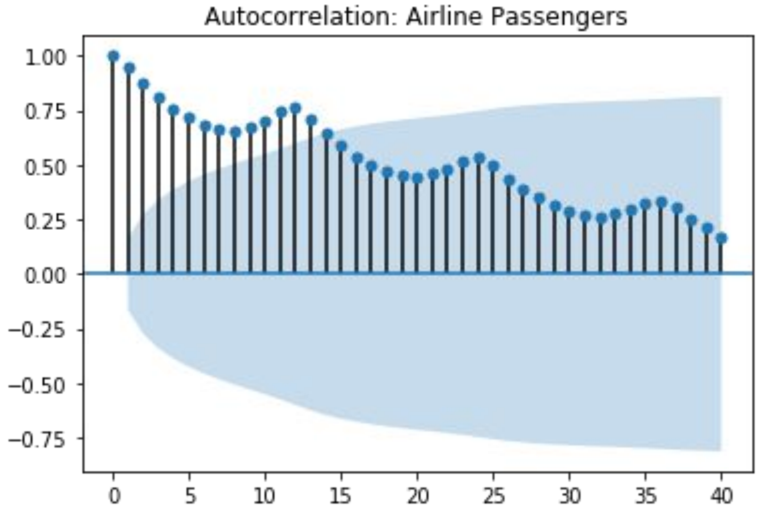
PIERIAN DATA

Python for Time Series



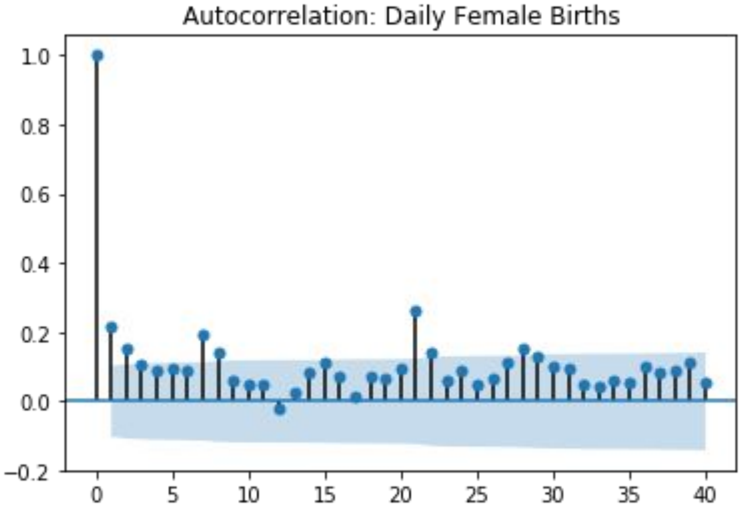
- An autocorrelation plot shows the correlation of the series with itself, lagged by x time units.
- You go on and do this for all possible time lags x and this defines the plot.
- Let’s see some typical examples!

- Gradual Decline



Python for Time Series

- Sharp Drop-off



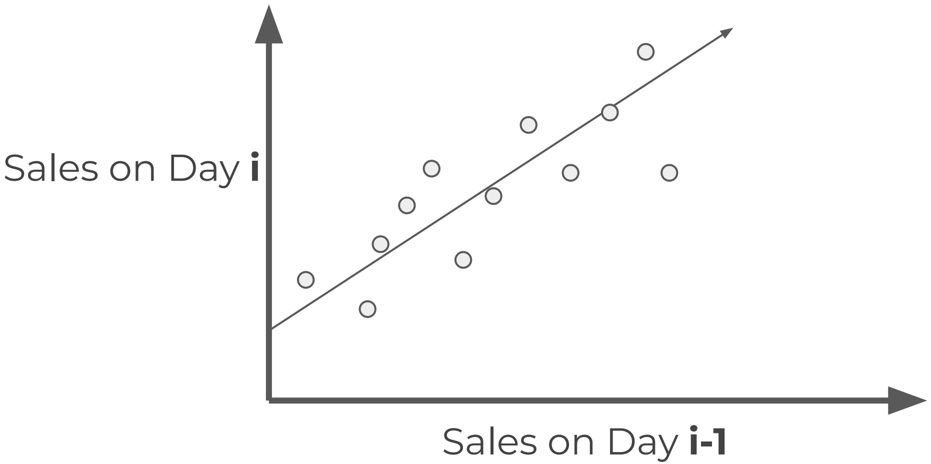
Python for Time Series

* It makes sense that in general there is a decline of some sort, the further away you get with the shift, the less likely the time series would be correlated with itself.

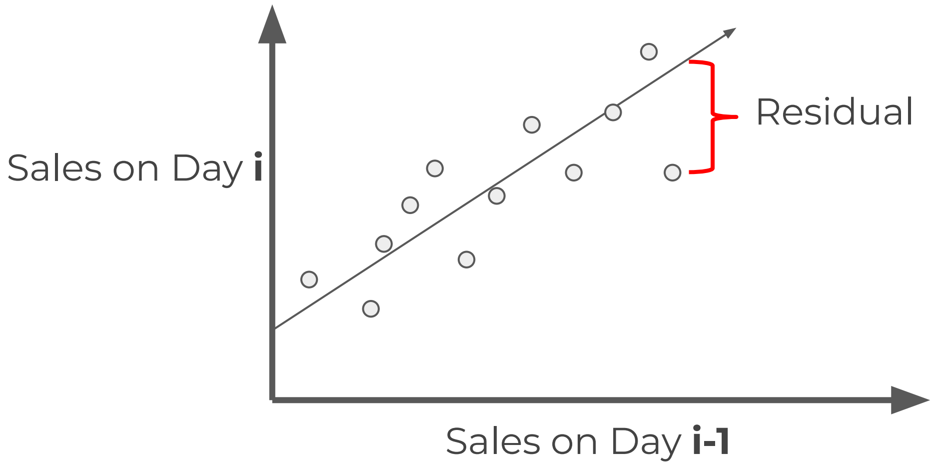
* The actual interpretation and how it relates to ARIMA models can get a bit complicated, but there are some basic common methods we can use for the ARIMA model.

* There are also partial autocorrelation plots!
* These are a little more complicated than autocorrelation plots, but let’s show you the basics.

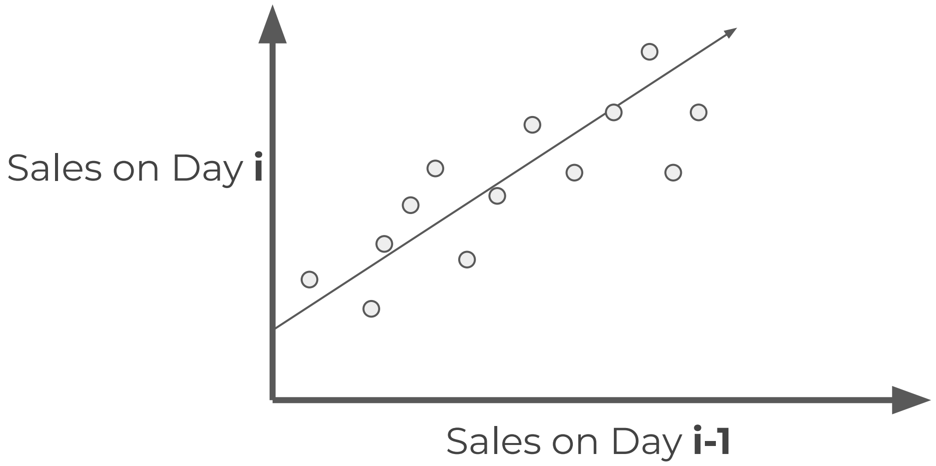
Python for Time Series

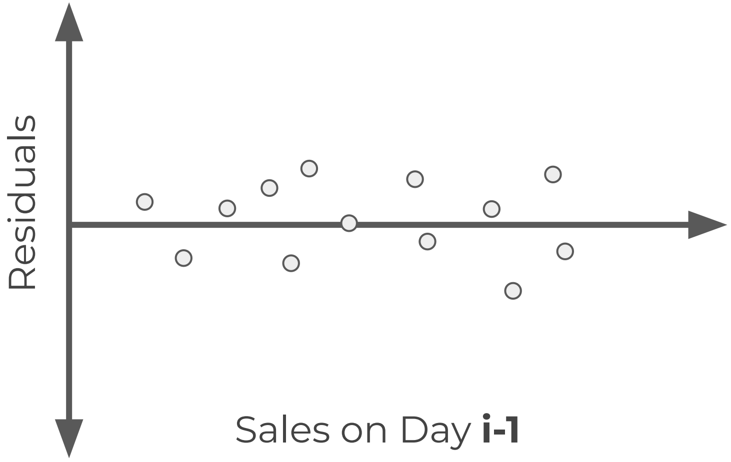


Python for Time Series



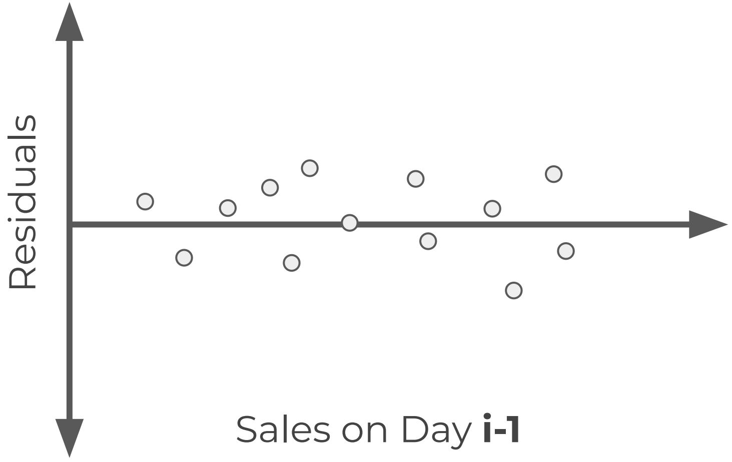
Python for Time Series



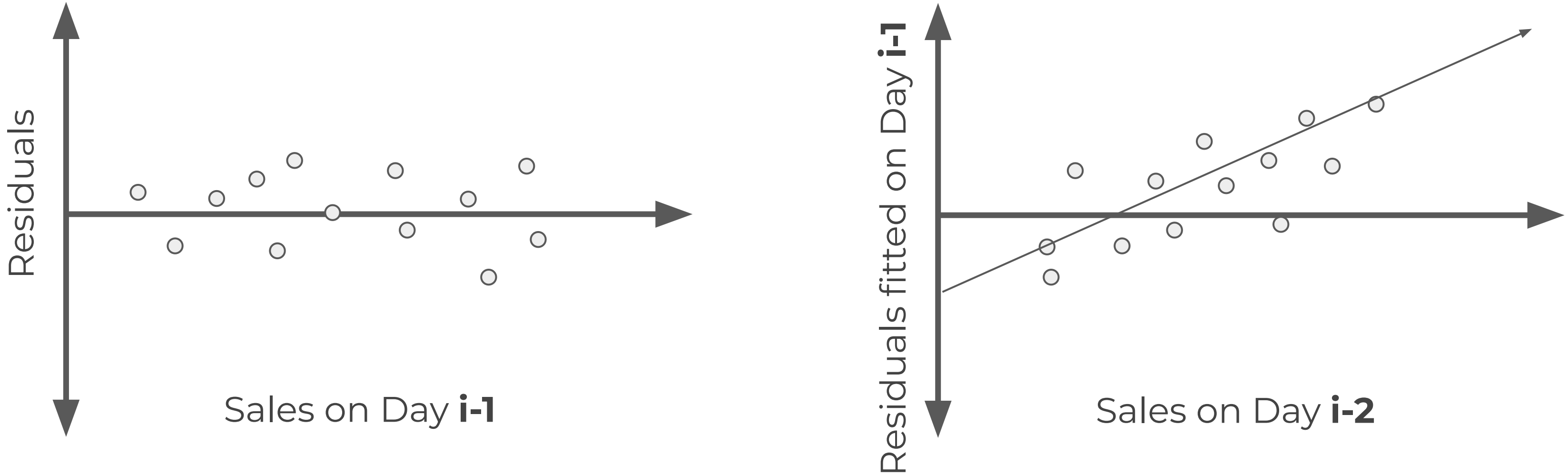


Python for Time Series

Python for Time Series



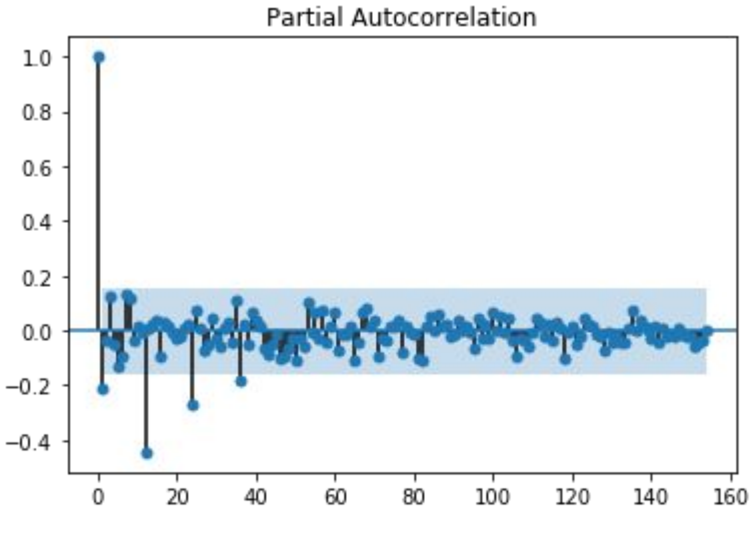
Python for Time Series



PIERIAN DATA

Python for Time Series

* Let's see an example of what the plot can look like:



- We essentially plot out the relationship between the previous day's residuals versus the real values of the current day.
- In general we expect the partial autocorrelation to drop off quite quickly.

- The ACF describes the autocorrelation between an observation and another observation at a prior time step that includes direct and indirect dependence information.

- The PACF only describes the direct relationship between an observation and its lag.

- These two plots can help choose order parameters for ARIMA based models.
- Later on, we will see that it is usually much easier to perform a grid search of the parameter values, rather than attempt to read these plots directly.

- Let’s explore how to create these plots with statsmodels!

ARIMA Overview

- 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

* 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.

- 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.

* 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.

- 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!

- 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.

- 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.

- 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.

- 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.

* 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!

- 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.

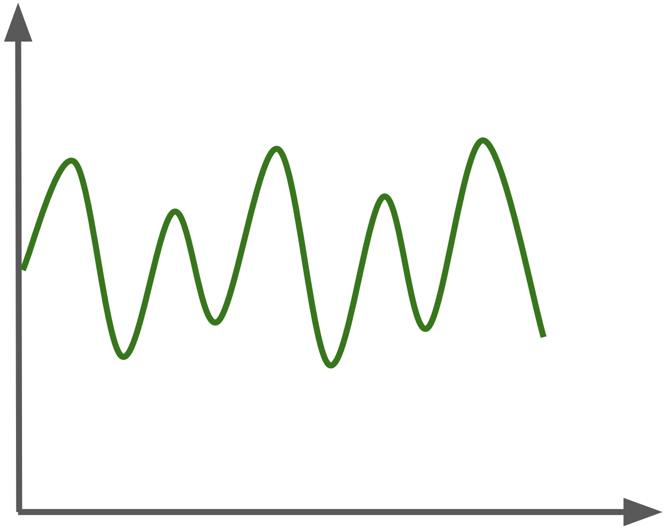
- Parts of ARIMA model
- MA (q): Moving Average.
    - A model that uses the dependency between an observation and a residual error from a moving average model applied to lagged observations.

- 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.

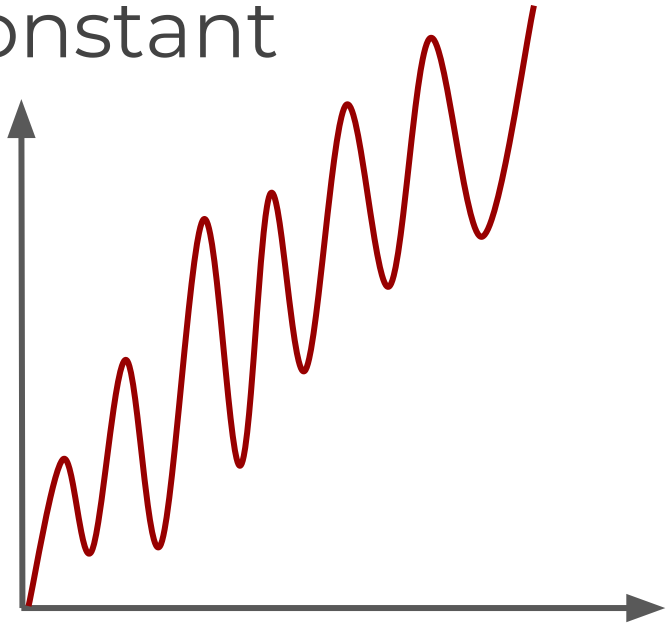
- A Stationary data set will allow our model to predict that the mean and variance will be the same in future periods.
- Let’s take a look at a few examples!

Python for Time Series

- Mean needs to be constant



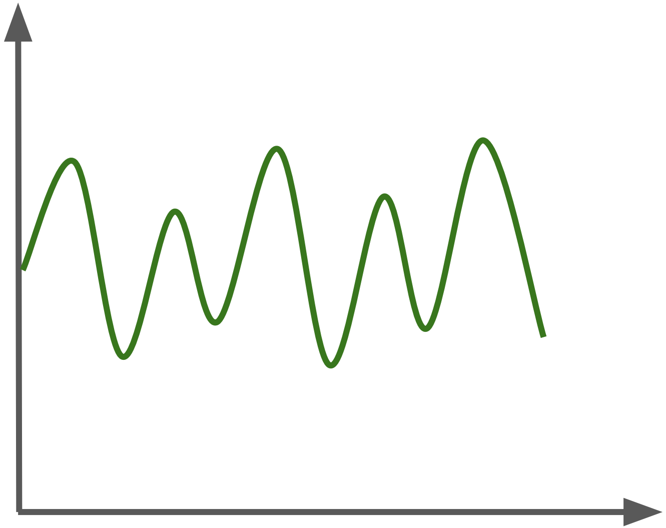
Stationary



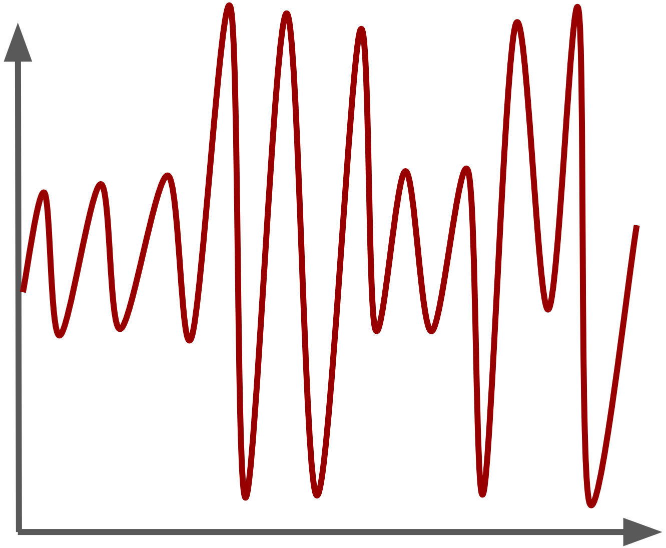
Non-Stationary

PIERIAN DATA

- Variance should not be a function of time



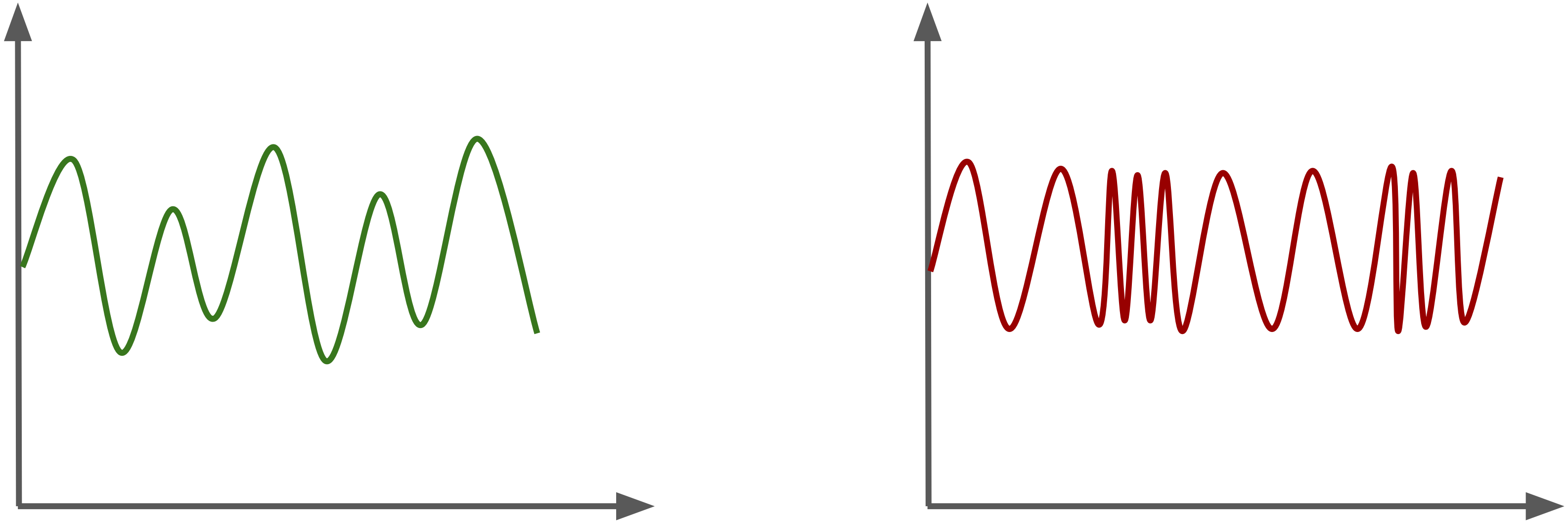
Stationary



Non-Stationary

Python for Time Series

Covariance should not be a function of time



Stationary

Non-Stationary

- 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)

- 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.

- One simple way to do this is through "differencing".
- The idea behind differencing is quite simple, let's see an example...

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

- 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!

- 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.

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

- 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.

- 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.

- 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!

- 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.

- Let's begin by focusing on a special case of ARIMA, where the I and MA components are zero, leaving us with a simplified AR model.

AutoRegression - AR

- In a moving average model as we saw with Holt-Winters, we forecast the variable of interest using a linear combination of predictors.

- In our example we forecasted numbers of airline passengers in thousands based on a set of level, trend and seasonal predictors.

* ARIMA stands for AutoRegression Integrated Moving Average.
* If we drop the Integrated and Moving Average components, then we're only left with AR.

* Later on we will revisit the idea of a full ARIMA model, but for now, let's explore the simplified AR model.

- In an autoregression model, we forecast using a linear combination of past values of the variable. The term autoregression describes a regression of the variable against itself. An autoregression is run against a set of lagged values of order **p**.

* The autoregressive model specifies that the output variable depends linearly on its own previous values and on a stochastic term (an imperfectly predictable term).

- Together with the moving-average (MA) model, it is a special case and key component of the more general ARMA and ARIMA models of time series, which have a more complicated stochastic structure; it is also a special case of the vector autoregressive model (VAR).

- Let's check out the formula for AR.
- Where **c** is a constant, ***ϕ*_1** and ***ϕ*_2** are lag coefficients up to order **p**, and **ε_t** is white noise

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \ldots + \phi_p y_{t-p} + \varepsilon_t$$

- Let's check out the formula for AR.
- Where **c** is a constant, ***ϕ*_1** and ***ϕ*_2** are lag coefficients up to order **p**, and **ε_t** is white noise

$$\boxed{y_t} = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \dots + \phi_p y_{t-p} + \varepsilon_t$$

- Let's check out the formula for AR.
- Where **c** is a constant, ***ϕ*_1** and ***ϕ*_2** are lag coefficients up to order **p**, and **ε_t** is white noise

$$y_t = c + \boxed{\phi_1 y_{t-1}} + \boxed{\phi_2 y_{t-2}} + \dots + \boxed{\phi_p y_{t-p}} + \varepsilon_t$$

- Let's check out the formula for AR.
- Where **c** is a constant, ***ϕ*_1** and ***ϕ*_2** are lag coefficients up to order **p**, and **ε_t** is white noise

$$y_t = \boxed{c} + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \ldots + \phi_p y_{t-p} + \boxed{\varepsilon_t}$$

* For example, an AR(1) model would follow the formula:

$$y_t = c + \phi_1 y_{t-1} + \varepsilon_t$$

- An AR(2) model would follow the formula

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \varepsilon_t$$

- Higher order AR models become mathematically very complex.
- Fortunately for us, we can let statsmodels library choose the best order for the model.

AutoRegression - AR

STATSMODELS

Descriptive Statistics and Tests

PART ONE

- In upcoming lectures we'll talk about different forecasting models like ARMA, ARIMA, Seasonal ARIMA and others. Each model addresses a different type of time series.

- For this reason, in order to select an appropriate model we need to know something about the data.

- Statsmodels provides a variety of built in tests to explore the underlying attributes of a time series.
- We'll learn how to determine if a time series is stationary, if it's independent, and if two series demonstrate causality.

- Tests for Stationarity
  - To determine whether a series is stationary we can use the augmented Dickey-Fuller Test.
  - This performs a test in the form of a classic null hypothesis test and returns a p value.

- Dickey-Fuller Test
  - In this test the null hypothesis states that $\Phi = 1$ (this is also called a unit test).
  - If p value is low (<0.05) we reject the null hypothesis, so we assume the dataset is stationary.

- Dickey-Fuller Test
  - In this test the null hypothesis states that Φ = 1 (this is also called a unit test).
  - If p value is high (>0.05) we **fail to reject** the null hypothesis.

* Dickey-Fuller Test
    * It can be tricky to remember the null hypothesis, so later on we will develop a nice function that returns an easy to read report!

Python for Time Series

- Granger Causality Tests
  - The Granger causality test is a hypothesis test to determine if one time series is useful in forecasting another.

- Granger Causality Tests
  - While it is fairly easy to measure correlations between series it's another thing to observe changes in one series correlated to changes in another after a consistent amount of time.

Python for Time Series

* Granger Causality Tests
    * This test is used to see if there is an indication of causality, but keep in mind, it could always be some outside factor unaccounted for!

- Evaluating Forecasts
- We’re already familiar with:
    - MAE
    - MSE
    - RMSE
    - But we still haven’t touched on AIC and BIC

- AIC - Akaike Information Criterion
  - Developed by Hirotugu Akaike in 1971.
  - His publication on it is one of the top 100 most cited publications of all time!
  - AIC is now such a common metric, many writers no longer cite the original paper.

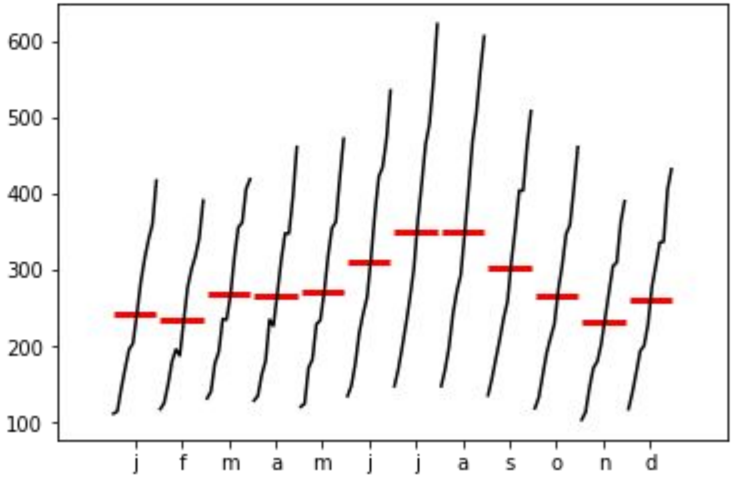
- AIC - Akaike Information Criterion
    - The AIC evaluates a collection of models and estimates the quality of each model **relative** to the others.
    - **Penalties** are provided for the n**umber of parameters** used in an effort to thwart overfitting.

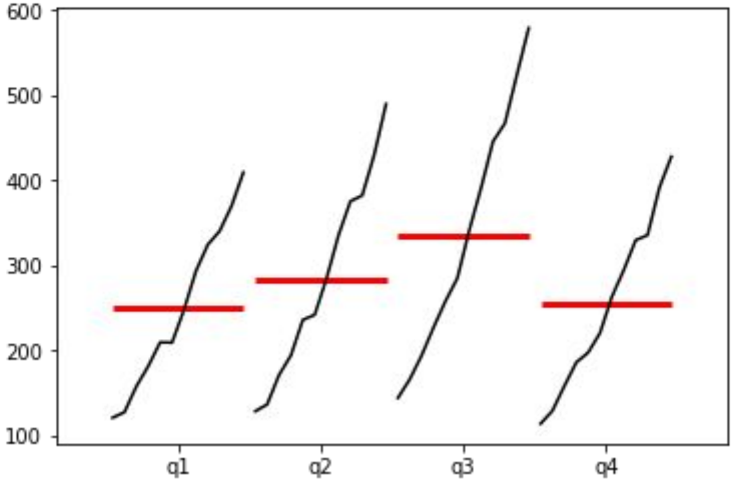
- AIC - Akaike Information Criterion
  - Overfitting results in performing very well on training data, but poorly on new unseen data.

- BIC - Bayesian Information Criterion
  - Very similar to AIC, just the mathematics behind the model comparisons utilize a Bayesian approach.
  - Developed in 1978 by Gideon Schwarz

Python for Time Series

- We will also explore Seasonality Plots.





Let's get started!

Descriptive Statistics and Tests

PART TWO

ARIMA Theory Overview

- Before we dive into how to choose orders for full ARIMA models, let's do a quick review of the actual formulas for ARIMA.

Python for Time Series

- Recall the 3 components
    - AR - AutoRegression
    - I - Integrated
    - MA - Moving Average

- AR - AutoRegression
  - The AR part of ARIMA indicates that the evolving variable of interest is regressed on its own lagged (i.e., prior) values.

- AR - AutoRegression
  - Building the regression model off of previous y values:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \ldots + \phi_p y_{t-p} + \varepsilon_t$$

* MA - Moving Average
    * Indicates the regression error is actually a **linear** combination of **error terms** whose values occurred contemporaneously and at various times in the past.

Python for Time Series

* MA - Moving Average
    * A model that uses the dependency between an observation and a residual error from a moving average model applied to lagged observations.

- MA - Moving Average
  - Recall that when we plotted out a moving average with pandas, it would “smooth” out the noise from the time series.

- MA - Moving Average
    - We essentially set up another regression model, that focuses on this residual term between a moving average and the real values.

$$\varepsilon_t + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q}$$

- MA - Moving Average
  - We could then choose another order for this MA component.

$$\varepsilon_t + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q}$$

* I - Integrated
  * Indicates that the data values have been replaced with the difference between their values and the previous values

- I - Integrated
  - This basically just means how many times did we have to difference the data to get it stationary so the AR and MA components could work.

- Non-seasonal ARIMA models are generally denoted ARIMA(p,d,q) where parameters p, d, and q are non-negative integers.

- **p** is the order (number of time lags) of the autoregressive model, **d** is the degree of differencing (the number of times the data have had past values subtracted), and **q** is the order of the moving-average model.

* So what does this equation actually look like?
* Let’s first consider just ARMA (no differencing term).

- **ARMA(p′,q)** is defined by:

$$X_t - \alpha_1 X_{t-1} - \cdots - \alpha_{p'} X_{t-p'} = \varepsilon_t + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q}$$

- $\mathbf{X_t}$ is the time series data (t is the index)
- **α** are the parameters of the AR model

Python for Time Series

- $\boxed{\textbf{AR}}$**MA(**$\boxed{\textbf{p'}}$**,q)** is defined by:

$$\boxed{X_t - \alpha_1 X_{t-1} - \cdots - \alpha_{p'} X_{t-p'}} = \varepsilon_t + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q}$$

- $\mathbf{X_t}$ is the time series data (t is the index)
- **α** are the parameters of the AR model

* **ARMA(p',q)** is defined by:

$$X_t - \alpha_1 X_{t-1} - \cdots - \alpha_{p'} X_{t-p'} = \boxed{\varepsilon_t + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q}}$$

* $\boldsymbol{\varepsilon_t}$ are the error terms
* $\boldsymbol{\theta}$ are the parameters of the MA model

Python for Time Series

- **ARMA(p′,q)** is defined by:

$$\left(1-\sum_{i=1}^{p'}\alpha_i L^i\right)X_t=\left(1+\sum_{i=1}^{q}\theta_i L^i\right)\varepsilon_t$$

- L is the lag operator

- **ARIMA(p′,q,d)** is defined by:

$$\left(1-\sum_{i=1}^{p}\phi_i L^i\right)(1-L)^d X_t=\left(1+\sum_{i=1}^{q}\theta_i L^i\right)\varepsilon_t$$

* **ARIMA(p′,q,d)** is defined by:

$$\left(1-\sum_{i=1}^{p}\phi_i L^i\right)(1-L)^d X_t = \left(1+\sum_{i=1}^{q}\theta_i L^i\right)\varepsilon_t$$

- We see here we have 3 main parameters to choose: p, d, and q.
- Let's explore how we can choose these (or let statsmodels choose them for us).

Choosing ARIMA Orders

PART ONE

- In this lecture we will discuss the best way to figure out what p,d,q, and P,D,Q values to use for ARIMA based models.
- We'll first discuss the "classical" method of reading ACF and PACF plots, then move on to discuss grid searches.

* Our main priority here is to try to figure out the orders for the AR and MA components, and if we need to difference our data (the I component).

- Depending on the dataset, it is quite common to only require AR or MA components, you may not need both!

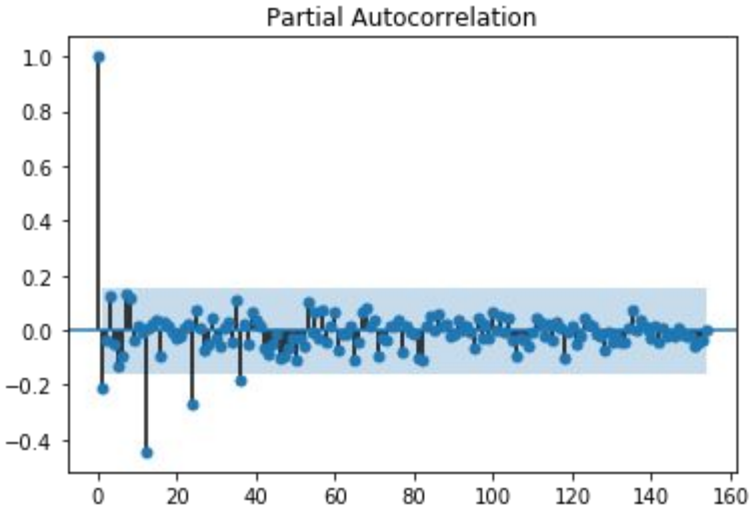
- If the autocorrelation plot shows positive autocorrelation at the first lag (lag-1), then it suggests to use the AR terms in relation to the lag

* If the autocorrelation plot shows negative autocorrelation at the first lag, then it suggests using MA terms.
* This will allow you to decide what actual values of p,d, and q to provide your ARIMA model.

- p: The number of lag observations included in the model.
- d: The number of times that the raw observations are differenced
- q: The size of the moving average window, also called the order of moving average.

Python for Time Series

* Let's see an example of what the plot can look like:



- Typically a sharp drop after lag "k" suggests an AR-k model should be used.
- If there is a gradual decline, it suggests an MA model.

- Identification of an AR model is often best done with the PACF.
- Identification of an MA model is often best done with the ACF rather than the PACF.
- View the notebook and resource links for more details.

- Finally once you've analyzed your data using ACF and PACF you are ready to begin to apply ARIMA or Seasonal ARIMA, depending on your original data.
- You will provide the p,d, and q terms for the model.

- An ARIMA will then take three terms p,d, and q. (We'll see this in the coding example)
- For seasonal ARIMA there will be an additional set of P,D,Q terms that we will see.

* As previously mentioned, it can be very difficult to read these plots, so it is often more effective to perform a grid search across various combinations of p,d,q values.

- The pmdarima (Pyramid ARIMA) is a separate library designed to perform grid searches across multiple combinations of p,d,q, and P,D,Q.
- This is by far the most effective way to get good fitting models!

- The pmdarima (Pyramid ARIMA) is a separate library designed to perform grid searches across multiple combinations of p,d,q, and P,D,Q.
- This is by far the most effective way to get good fitting models!

- The pmdarima library utilizes the Akaike information criterion (AIC) as a metric to compare the performance of various ARIMA based models.

- AIC was developed by Hirotugu Akaike in the 1970s.
- When comparing models we want to minimize the AIC value.

$$\mathrm{AIC} = 2k - 2\ln(\hat{L})$$

- Suppose that we have a statistical model of some data. Let **k** be the number of estimated parameters in the model. Let **L** be the maximum value of the likelihood function for the model.

$$\mathrm{AIC} = 2k - 2\ln(\hat{L})$$

Choosing ARIMA Orders

PART TWO

ARMA and ARIMA

Python for Time Series

SARIMA

* Where ARIMA accepts the parameters (p,d,q), SARIMA accepts an additional set of parameters (P,D,Q)m that specifically describe the seasonal components of the model.

* Here P, D and Q represent the seasonal regression, differencing and moving average coefficients, and m represents the number of data points (rows) in each seasonal cycle.

- The statsmodels implementation of SARIMA is called SARIMAX. The “X” added to the name means that the function also supports exogenous regressor variables. We'll cover these in a future lecture.

SARIMAX Models

PART ONE

- The statsmodels implementation of SARIMA is called SARIMAX. The “X” added to the name means that the function also supports exogenous regressor variables.

- Quick Note: Label is the term we'll be using for the column we're trying to predict.
- Examples:
  - Label was the CO2 Level in Mauna Loa
  - Label was the Number of Passengers

- For example, let’s imagine we were trying to forecast the number of visitors to a restaurant and we had historical data on previous visitor numbers.

* With just this previous historical data, we could attempt to use a SARIMA based model to use historical lagged values to predict future visit numbers.
* But what if we had some other features we wanted to include, like holidays?

- With just this previous historical data, we could attempt to use a SARIMA based model to use historical lagged values to predict future visit numbers.
- But what if we had some other features we wanted to include, like holidays?

- Let's walk through an example data set, where our goal is to predict the number of total visitors across 4 restaurants.
- Using our previous approaches, the only data we can use is **previous historical label data.**

Python for Time Series

- Typical data sets without exogenous variables:

	rest1	rest2	rest3	rest4	total
date					
2016-01-01	65.0	25.0	67.0	139.0	296.0
2016-01-02	24.0	39.0	43.0	85.0	191.0
2016-01-03	24.0	31.0	66.0	81.0	202.0
2016-01-04	23.0	18.0	32.0	32.0	105.0
2016-01-05	2.0	15.0	38.0	43.0	98.0

Python for Time Series

* Total daily visitors across 4 restaurants.

	rest1	rest2	rest3	rest4	total
date					
2016-01-01	65.0	25.0	67.0	139.0	296.0
2016-01-02	24.0	39.0	43.0	85.0	191.0
2016-01-03	24.0	31.0	66.0	81.0	202.0
2016-01-04	23.0	18.0	32.0	32.0	105.0
2016-01-05	2.0	15.0	38.0	43.0	98.0

Python for Time Series

- Total daily visitors across 4 restaurants.

	rest1	rest2	rest3	rest4	total
date					
2016-01-01	65.0	25.0	67.0	139.0	296.0
2016-01-02	24.0	39.0	43.0	85.0	191.0
2016-01-03	24.0	31.0	66.0	81.0	202.0
2016-01-04	23.0	18.0	32.0	32.0	105.0
2016-01-05	2.0	15.0	38.0	43.0	98.0

- Total daily visitors across 4 restaurants.

	rest1	rest2	rest3	rest4	total
date					
2016-01-01	65.0	25.0	67.0	139.0	296.0
2016-01-02	24.0	39.0	43.0	85.0	191.0
2016-01-03	24.0	31.0	66.0	81.0	202.0
2016-01-04	23.0	18.0	32.0	32.0	105.0
2016-01-05	2.0	15.0	38.0	43.0	98.0

Python for Time Series

* Notice that even though we have multiple columns, these are all still just the label!

	rest1	rest2	rest3	rest4	total
date					
2016-01-01	65.0	25.0	67.0	139.0	296.0
2016-01-02	24.0	39.0	43.0	85.0	191.0
2016-01-03	24.0	31.0	66.0	81.0	202.0
2016-01-04	23.0	18.0	32.0	32.0	105.0
2016-01-05	2.0	15.0	38.0	43.0	98.0

Python for Time Series

* Exogenous variables are outside information, not historical label data

	weekday	holiday	holiday_name	rest1	rest2	rest3	rest4	total
date								
2016-01-01	Friday	1	New Year's Day	65.0	25.0	67.0	139.0	296.0
2016-01-02	Saturday	0	na	24.0	39.0	43.0	85.0	191.0
2016-01-03	Sunday	0	na	24.0	31.0	66.0	81.0	202.0
2016-01-04	Monday	0	na	23.0	18.0	32.0	32.0	105.0
2016-01-05	Tuesday	0	na	2.0	15.0	38.0	43.0	98.0

Python for Time Series

- We can add in exogenous variables such as holidays.

	weekday	holiday	holiday_name	rest1	rest2	rest3	rest4	total
date								
2016-01-01	Friday	1	New Year's Day	65.0	25.0	67.0	139.0	296.0
2016-01-02	Saturday	0	na	24.0	39.0	43.0	85.0	191.0
2016-01-03	Sunday	0	na	24.0	31.0	66.0	81.0	202.0
2016-01-04	Monday	0	na	23.0	18.0	32.0	32.0	105.0
2016-01-05	Tuesday	0	na	2.0	15.0	38.0	43.0	98.0

* We could also attempt to feature engineer off of the weekday column.

	weekday	holiday	holiday_name	rest1	rest2	rest3	rest4	total
date								
2016-01-01	Friday	1	New Year's Day	65.0	25.0	67.0	139.0	296.0
2016-01-02	Saturday	0	na	24.0	39.0	43.0	85.0	191.0
2016-01-03	Sunday	0	na	24.0	31.0	66.0	81.0	202.0
2016-01-04	Monday	0	na	23.0	18.0	32.0	32.0	105.0
2016-01-05	Tuesday	0	na	2.0	15.0	38.0	43.0	98.0

- For example, create a 0/1 column for True or False if its a weekend or not.

	weekday	holiday	holiday_name	rest1	rest2	rest3	rest4	total
date								
2016-01-01	Friday	1	New Year's Day	65.0	25.0	67.0	139.0	296.0
2016-01-02	Saturday	0	na	24.0	39.0	43.0	85.0	191.0
2016-01-03	Sunday	0	na	24.0	31.0	66.0	81.0	202.0
2016-01-04	Monday	0	na	23.0	18.0	32.0	32.0	105.0
2016-01-05	Tuesday	0	na	2.0	15.0	38.0	43.0	98.0

- You should usually have some intuition about what relates to the column you are trying to forecast.
- For statsmodels, exogenous variables should be converted to numerical values.

- There are a variety of ways to do this (e.g. one-hot encoding, dummy variables, etc...)
- This usually involves just mapping values to some 0 or 1 True or False scale.
- This can be done with pandas with the **pd.get_dummies()** command.

- For example:

	Price	Qty	City
0	6.225481	5.716618	New York
1	1.131167	6.297597	Chicago
2	2.538992	5.016772	Boston
3	5.622141	3.294433	Boston
4	7.739604	0.554239	Chicago
5	4.991203	4.917935	Chicago
6	0.898304	7.721118	Chicago
7	8.644638	4.647118	Chicago
8	4.737761	8.780549	Chicago
9	9.453818	2.781897	New York

	Price	Qty	Boston	Chicago	New York
0	6.225481	5.716618	0	0	1
1	1.131167	6.297597	0	1	0
2	2.538992	5.016772	1	0	0
3	5.622141	3.294433	1	0	0
4	7.739604	0.554239	0	1	0
5	4.991203	4.917935	0	1	0
6	0.898304	7.721118	0	1	0
7	8.644638	4.647118	0	1	0
8	4.737761	8.780549	0	1	0
9	9.453818	2.781897	0	0	1

* You should usually have some intuition about what relates to the column you are trying to forecast.
* SARIMAX makes it easy to add in additional columns as exogenous variables, let’s take a look!

SARIMAX Models

PART TWO

SARIMAX Models

PART THREE

- So far we've only run a SARIMA based model on our data, now let's add in the exogenous variable!
- Statsmodels makes this easy, its simply an additional parameter call.

Python for Time Series

* However, there is something important to note here!
* We need to know the future of this exogenous variable.
* What does that mean exactly?

* Let’s review the actual forecasting process for basic SARIMA:
  * We first re-train on all our data
  * We set the future date span
  * We forecast values

Date	Y Label
D1	Y1
D2	Y2
D3	Y3
D4	Y4

TRAIN

Date	Y Label
D1	Y1
D2	Y2
D3	Y3
D4	Y4

TRAIN

Date	Y Label
D5	?
D6	?
D7	?
D8	?

FUTURE DATES

Python for Time Series

Date	Y Label
D1	Y1
D2	Y2
D3	Y3
D4	Y4

TRAIN

Date	Y Label
D5	?
D6	?
D7	?
D8	?

FUTURE DATES

Date	Y Label
D5	Fore_5
D6	Fore_6
D7	Fore_7
D8	Fore_8

FORECASTED VALUES

Python for Time Series

- For SARIMAX, we need to provide more information for the future dates.
- We need to provide the known exogenous variable into the future.
- We **can not** also predict this exogenous variables, because then we are attempting to predict 2 things at once!

Date	EXO	Y Label
D1	X1	Y1
D2	X2	Y2
D3	X3	Y3
D4	X4	Y4

TRAIN

Python for Time Series

Date	EXO	Y Label
D1	X1	Y1
D2	X2	Y2
D3	X3	Y3
D4	X4	Y4

TRAIN

Date	EXO	Y Label
D5	X5	?
D6	X6	?
D7	X7	?
D8	X8	?

FUTURE DATES

Python for Time Series

Date	EXO	Y Label
D1	X1	Y1
D2	X2	Y2
D3	X3	Y3
D4	X4	Y4

TRAIN

Date	EXO	Y Label
D5	X5	?
D6	X6	?
D7	X7	?
D8	X8	?

FUTURE DATES

Date	EXO	Y Label
D5	X5	Y5
D6	X6	Y6
D7	X7	Y7
D8	X8	Y8

FORECASTED VALUES

- This means that we need to already know this future exogenous information for certain, or at least have very confident estimations for it based on some other data.

- It wouldn’t make sense to be predicting both exogenous and the y label into the future, since we just trained our model to predict y label based on the existing exogenous variable at that same timestamp.

- Let’s explore this process with statsmodels!

VAR Models

Theory

- In our previous SARIMAX example, the forecast variable **y_t** was influenced by the exogenous predictor variable, but not vice versa.
- That is, the occurrence of a holiday affected restaurant patronage but not the other way around.

- However, there are some cases where variables affect each other!
- What kind of model can we use in these situations?
    - We can attempt to use the Vector AutoRegression model!

* All variables in a VAR enter the model in the same way: each variable has an equation explaining its evolution based on its own lagged values, the lagged values of the other model variables, and an error term.

- VAR modeling does not require as much knowledge about the forces influencing a variable.
- The only prior knowledge required is a list of variables which can be hypothesized to affect each other intertemporally.

- ***Forecasting: Principles and Practice*** describes a case where changes in personal consumption expenditures **C_t** were forecast based on changes in personal disposable income **I_t**.

* We've seen that an autoregression AR(p) model is described by the following:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \ldots + \phi_p y_{t-p} + \varepsilon_t$$

- A K-dimensional VAR model of order p, denoted VAR(p), considers each variable y_k in the system.

- For example, The system of equations for a 2-dimensional VAR(1) model is:

$$y_{1,t} = c_1 + \phi_{11,1} y_{1,t-1} + \phi_{12,1} y_{2,t-1} + \varepsilon_{1,t}$$
$$y_{2,t} = c_2 + \phi_{21,1} y_{1,t-1} + \phi_{22,1} y_{2,t-1} + \varepsilon_{2,t}$$

- For example, The system of equations for a 2-dimensional VAR(1) model is:

$$\boxed{y_{1,\,t}} = c_1 + \boxed{\phi_{11,\,1}y_{1,\,t-1}} + \phi_{12,\,1}y_{2,\,t-1} + \varepsilon_{1,\,t}$$
$$y_{2,\,t} = c_2 + \boxed{\phi_{21,\,1}y_{1,\,t-1}} + \phi_{22,\,1}y_{2,\,t-1} + \varepsilon_{2,\,t}$$

- For example, The system of equations for a 2-dimensional VAR(1) model is:

$$y_{1,\,t} = c_1 + \phi_{11,\,1} y_{1,\,t-1} + \boxed{\phi_{12,\,1} y_{2,\,t-1}} + \varepsilon_{1,\,t}$$
$$\boxed{y_{2,\,t}} = c_2 + \phi_{21,\,1} y_{1,\,t-1} + \boxed{\phi_{22,\,1} y_{2,\,t-1}} + \varepsilon_{2,\,t}$$

- Carrying this further, the system of equations for a 2-dimensional VAR(3) model is:

$$y_{1,t} = c_1 + \phi_{11,1} y_{1,t-1} + \phi_{12,1} y_{2,t-1} + \phi_{11,2} y_{1,t-2} + \phi_{12,2} y_{2,t-2} + \phi_{11,3} y_{1,t-3} + \phi_{12,3} y_{2,t-3} + \varepsilon_{1,t}$$
$$y_{2,t} = c_2 + \phi_{21,1} y_{1,t-1} + \phi_{22,1} y_{2,t-1} + \phi_{21,2} y_{1,t-2} + \phi_{22,2} y_{2,t-2} + \phi_{21,3} y_{1,t-3} + \phi_{22,3} y_{2,t-3} + \varepsilon_{2,t}$$

- The general steps involved in building a VAR model are:
  - Examine the data
  - Visualize the data
  - Test for stationarity

Python for Time Series

- The general steps involved in building a VAR model are:
    - Select the appropriate order p
    - Instantiate the model and fit it to a training set

- The general steps involved in building a VAR model are:
    - If necessary, invert the earlier transformation
    - Evaluate model predictions against a known test set
    - Forecast the future

Let’s get started!

VAR Models

CODE ALONG

Python for Time Series

- Let's explore how we would forecast into the future using VAR for two time series that we believe have effects on **eachother**.
- We'll use M2 Money Stock and Personal Consumption from FRED.

- Personal Consumption Expenditures
- M2 Money Stock
    - savings deposits
    - small-denomination time deposits
    - balances in retail money market mutual funds

- Recall the system of equations for a 2-dimensional VAR(1) model is:

$$y_{1,t} = c_1 + \phi_{11,1} y_{1,t-1} + \phi_{12,1} y_{2,t-1} + \varepsilon_{1,t}$$
$$y_{2,t} = c_2 + \phi_{21,1} y_{1,t-1} + \phi_{22,1} y_{2,t-1} + \varepsilon_{2,t}$$

- Y1 = Personal Consumption Expenditures
- Y2 = M2 Money Stock

$$y_{1,t} = c_1 + \phi_{11,1} y_{1,t-1} + \phi_{12,1} y_{2,t-1} + \varepsilon_{1,t}$$
$$y_{2,t} = c_2 + \phi_{21,1} y_{1,t-1} + \phi_{22,1} y_{2,t-1} + \varepsilon_{2,t}$$

* We'll need to see what is the best value of p through code!

$$y_{1,t} = c_1 + \phi_{11,1} y_{1,t-1} + \phi_{12,1} y_{2,t-1} + \varepsilon_{1,t}$$
$$y_{2,t} = c_2 + \phi_{21,1} y_{1,t-1} + \phi_{22,1} y_{2,t-1} + \varepsilon_{2,t}$$

- We will need to figure our optimal order (p) for our VAR model.
- Pyramid Auto Arima won’t do the grid search for us, but we can easily run various p values through a loop and then check which model has the best AIC.

- Recall AIC will also punish model for being too complex, even if they perform slightly better on some other metric.
- So we expect to see a drop in AIC as p gets larger and then at a certain point (lag order p value) an increasing AIC.

* We'll also need to manually check for stationarity and difference the time series if they are not stationary.
* In the case of this lecture, we'll notice the time series require different differencing amounts.

- We will difference them the same amount however, in order to make sure they have the same number of rows.
- Let’s get started!

VARMA Models

THEORY

- Recall the system of equations for a 2-dimensional VAR(1) model is:

$$y_{1,t} = c_1 + \phi_{11,1} y_{1,t-1} + \phi_{12,1} y_{2,t-1} + \varepsilon_{1,t}$$
$$y_{2,t} = c_2 + \phi_{21,1} y_{1,t-1} + \phi_{22,1} y_{2,t-1} + \varepsilon_{2,t}$$

- We're also already familiar with the ARMA model (note, here the relation is solved for y_t):

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \ldots + \phi_p y_{t-p} + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \ldots + \theta_q \varepsilon_{t-q} + \varepsilon_t$$

- We can create an analogous function using VARMA to consider 2 related time series:

$$y_{1,t} = c_1 + \phi_{11,1}y_{1,t-1} + \phi_{12,1}y_{2,t-1} + \theta_{11,1}\varepsilon_{1,t-1} + \theta_{12,1}\varepsilon_{2,t-1} + \varepsilon_{1,t}$$
$$y_{2,t} = c_2 + \phi_{21,1}y_{1,t-1} + \phi_{22,1}y_{2,t-1} + \theta_{21,1}\varepsilon_{1,t-1} + \theta_{22,1}\varepsilon_{2,t-1} + \varepsilon_{2,t}$$

- Let’s explore how we can expand a VAR model to a full VARMA model.

VARMA Models

CODE ALONG

Python for Time Series

* Let’s explore how we would perform VARMA on the same data sets.
* This process will be very similar to the previous VAR lecture series, so we’ll guide you through the existing notebook and point out the main differences.

* One thing we will notice at the end is that VARMA actually performs poorly on the data sets, which is a good indication that there is probably not enough interaction between these two time series to warrant the Vector component.

Forecasting Exercises

OVERVIEW

Forecasting Exercises

SOLUTIONS