Data Camp Various Python Courses

# Manipulating Time Series Data in Python

## Working with Time Series in Pandas

### How to use dates and times with pandas

TimeStamp in pandas replaces python datetime.datetime object. It adds properties such as frequency.

data.date = pd.to\_datetime(data.date)

data.set\_index("date", inplace=True)

It converts the date column to type datetime64[ns] and creates the DatetimeIndex column

type(data.index)

> <class 'pandas.core.indexes.datetimes.DatetimeIndex'>

type(data.index[0])

> <class 'pandas.\_libs.tslibs.timestamps.Timestamp'>

DatetimeIndex consists of Timestamps

pd.date\_range(start, end, periods, freq)

creates a sequence of Timestamp objects with frequency info

### Indexing and resampling time series

We can use different ways to filter and index a time series

google[“2021”]

google[“2021-6”: “2023-6”]

google.loc[“2021-6”: “2023-6”, [“open”, “close”]]

Use pandas.dataframe.asfreq() method to add frequency to your Datetime index.

#### Up sampling & down sampling

Up sampling adds NA for missing values. Down sampling aggregates the observations.

### Lags, changes and returns for stock price series

**shift()** shifts days into the future or back in time

**diff()** calculates difference in value for two adjacent periods Xt – Xt-1

**pct\_change()** calculates the return in percentage change on a period Xt / Xt-1. It can also be calculated manually as “return1”

yahoo["shifted"] = yahoo["price"].shift()

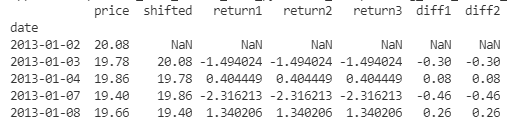
yahoo["return1"] = yahoo["price"].div(yahoo["shifted"]).sub(1).mul(100)

yahoo["return2"] = (yahoo["price"] / yahoo["shifted"] - 1) \* 100

yahoo["return3"] = yahoo["price"].pct\_change().mul(100)

yahoo["diff1"] = yahoo["price"].sub(yahoo["shifted"])

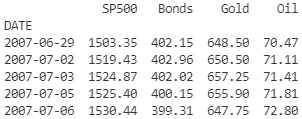
yahoo["diff2"] = yahoo["price"].diff()



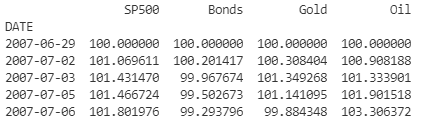
## Basic Time Series Metrics and Resampling

### Compare time series growth rates

In order to compare stock prices from different levels, we normalize the price series to start at 100



normalized = prices.div(prices.iloc[0]).mul(100)



### To compare asset classes to a bench mark index

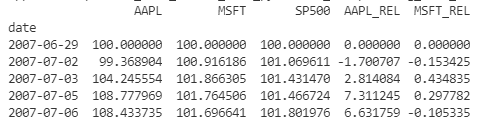
Treat the benchmark index as one of the asset classes, normalized all asset classes’ price series to start at 100

### Compare performance difference vs benchmark index

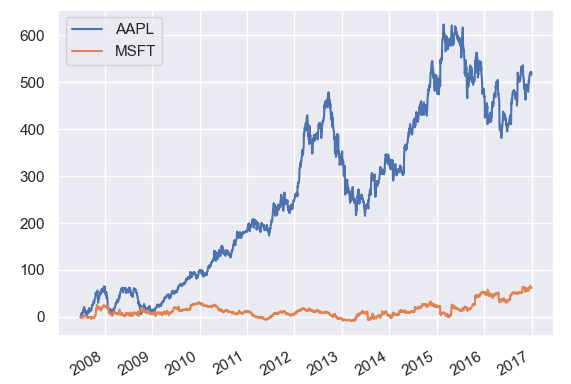
Normalize all asset classes (including index) to start at 100.   
Subtract index from asset classes.

normalized[["MSFT", "AAPL"]].sub(normalized["SP500"], axis=0)

We get MSFT and AAPL’s performances relative to SP500



In a 9 year period, AAPL performances 5 times better than SP500.



### Changing the time series frequency: resampling

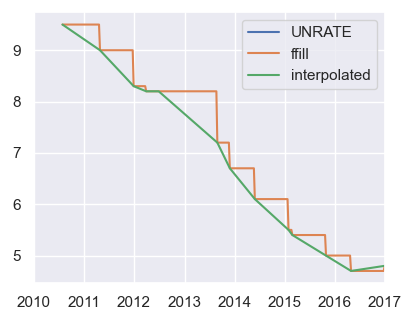
asfreq() resamples the datetime sequence to specified frequency. Depending on the nature of the operations, when up sampling, use fill\_value, method= “ffill” / “bfill” to fill the missing value. When downsampling, specify the aggregation method.

### Up sampling and interpolation

Interpolate interpolates the data points in between the actual ones.

weekly['ffill'] = weekly.UNRATE.ffill()

weekly['interpolated'] = weekly.UNRATE.interpolate()



### Down sampling and aggregation

stocks.resample("M").mean()

daily\_returns.resample("M").agg(["mean", "median", "std"])

## Window Functions: Rolling and Expanding Metrics

What are window functions:

* Identify sub periods of your time series
* Calculate metrics for sub periods inside the window
* Create a new time series of metrics
* Two types of windows:
  + Rolling: same size, sliding
  + Expanding: contain all prior values

### Rolling Window

We can use rolling window to compute the moving average to get a smoother curve.

data["Ozone"].rolling(window="90D").mean()

Or we can plot an upper and lower band using standard deviations

data['q10'] = rolling.quantile(0.1).to\_frame("q10")

data['q90'] = rolling.quantile(0.9).to\_frame("q90")

### Expanding Window

What are expanding windows:

* Calculate metrics for periods up to current date
* Creates new time series that reflect all historical values
* Useful for running rate of return, running min/max
* Two options with pandas
  + .expend() just like .rolling()
  + .cumsum(), cumprod(), cumin() cummax()

How to calculate a running return

**Single period return** Rt: current price over last price minus 1

Rt = (Pt / Pt-1) – 1

**Multi-period return**: product of (1 + Rt) for all periods, minus 1

Rt = (1 + R1) \* (1 + R2) \* … \* (1 + Rt) - 1

In code, we implement it as

# Calculate the daily returns here

returns = data.pct\_change()

# Calculate the cumulative returns here

returns\_plus\_one = returns + 1

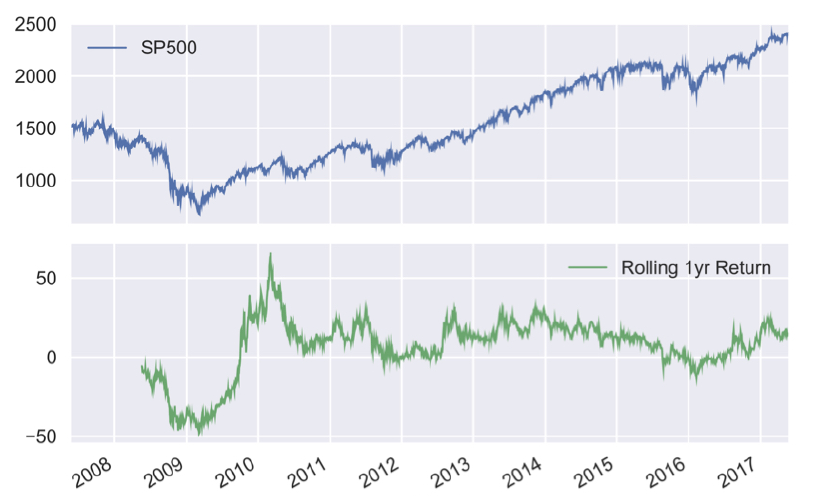
cumulative\_return = returns\_plus\_one.cumprod()

#### Rolling annual rate of return

def multi\_period\_return(period\_returns):

    return np.prod(period\_returns + 1) - 1

rolling\_annual\_returns = daily\_returns.rolling("360D").apply(multi\_period\_return)



### SP500 Price Simulation

Random walk theory & simulations

* Daily stock returns are hard to predict
* Models often assume they are random in nature

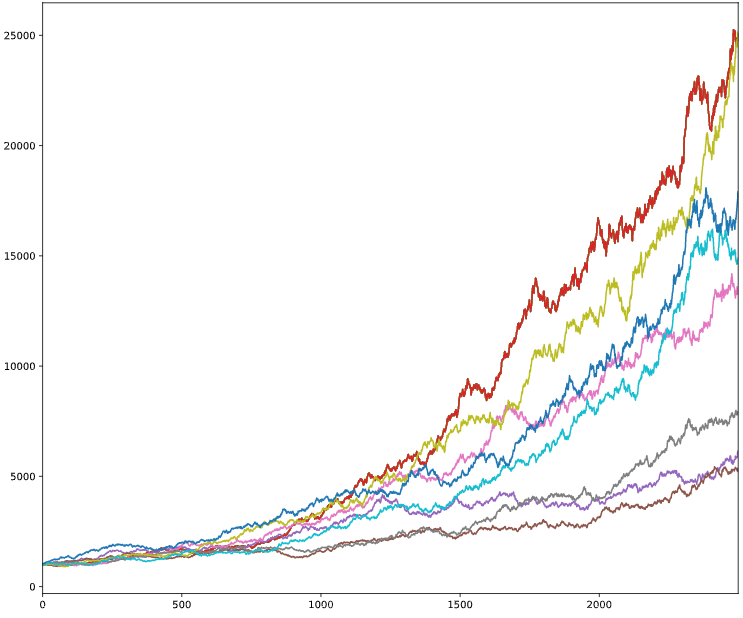
#### Example 1 - Generate random returns

random\_walk = normal(loc=.001, scale=.01, size=2500)

random\_prices = random\_walk.add(1).cumprod()

random\_prices.mul(1000).plot()

As we can see, an average daily return of merely 0.1% compounds into a high flying chart



#### Example 2 – Random selected actual SP500 returns

We can randomly select historical SP500 returns and simulate its price action

daily\_returns = fb.price.pct\_change().dropna()

random\_walk = np.random.choice(daily\_returns, n\_obs)

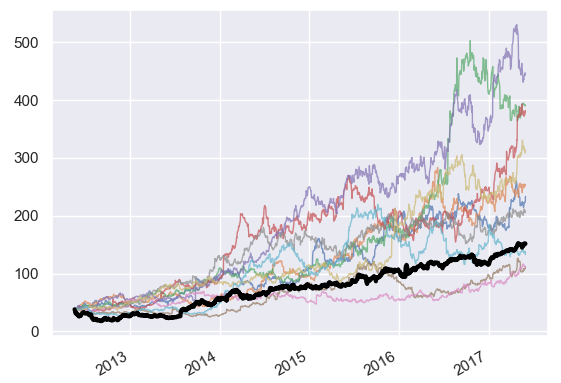
start = fb.price.first('D')

random\_walk = random\_walk.add(1)

random\_price = start.append(random\_walk)

random\_price = random\_price.cumprod()

If the actual price has large deviations, the random walk can produce result that are very different from the actual observation. Because large positives returns or large negative returns can compound in consecutive days.



### Correlations between Time Series

Correaltions ought to be computed on returns, not on the price itself. As two rising stock prices will always have high correlation. Use seaborn heatmap to visualize the correlation.

annual\_returns = annual\_prices.pct\_change()

correlations = annual\_returns.corr()

sns.heatmap(correlations, annot=True)

## Building a value-weighted index

What is a market value-weighted index?

* A stock index is a composition of various stocks
* Components weighted by market capitalization  
   share price \* number of shares = market value

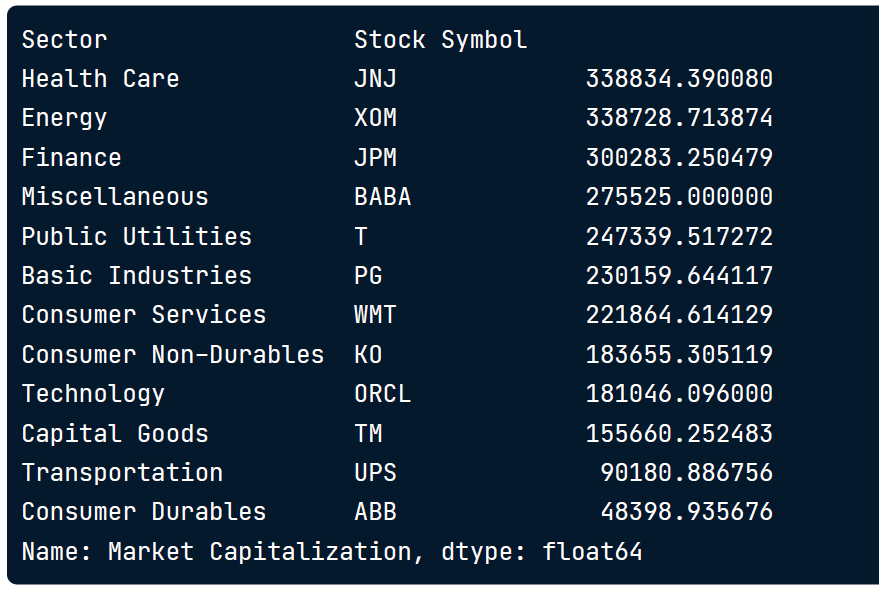
Steps of building a cap-weighted index

* Select components from exchange listing data
* Get component number of shares and stock prices
* Calculate component weights
* Calculate index
* Evaluate performance of components and index

### Selecting Index Components

Selecting stocks with largest market capitalization from each sector

components = listings.groupby("Sector")["Market Capitalization"].nlargest(1)



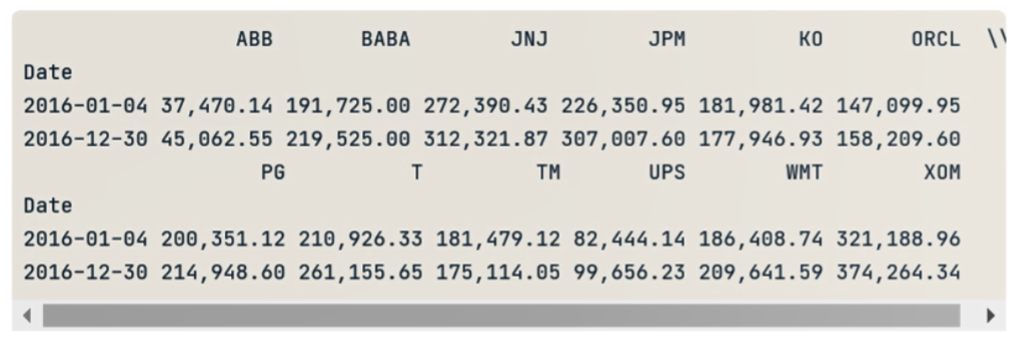
### Build a Market-cap weighted Index

Calculate number of shares by dividing the market capitalization by last sale price

no\_shares = components["Market Capitalization"] / components["Last Sale"]

Calculate aggregated market value per period

market\_cap = stock\_prices.mul(no\_shares)



Sum all the columns to one to get the aggregated index value

raw\_index = market\_cap\_series.sum(axis=1)

Normalize the raw index to start at 100

index = raw\_index.div(raw\_index.iloc[0]).mul(100)

### Evaluate Index Performance

We can evaluate the index in the following metrics

* Index return:
  + Total period return
  + Contribution by components
* Performance vs Benchmark
  + Total period return
  + Rolling returns for sub periods

### Index Correlation

We can analyze the daily return correlations of the index components

# Time Series Analysis in Python

## Correlation and Autocorrelation

### Correlation of Two Time Series

The correlation coefficient is a measure of how much two series vary together.

A common mistake is to measure correlation coefficient of the price of two stocks. If both stocks are trending up, they tend to have high correlations. **Instead**, we should look at the correlation of the “**returns**” of the two stocks.

### Simple Linear Regression

OLS from statsmodel performs linear regression.

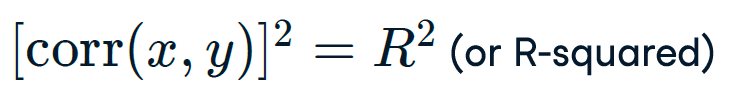
results = sm.OLS(df['R2000\_Ret'], df[['const','SPX\_Ret']]).fit()

df['R2000\_Ret'] is the dependent variable y

df[['const','SPX\_Ret']] is the independent variable X, along with a constant so that OLS can regress with an intercept

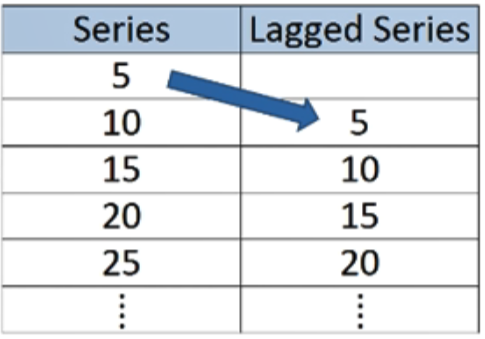
#### Relationship between r-squared and correlation

R-squared is the magnitude of the correlation

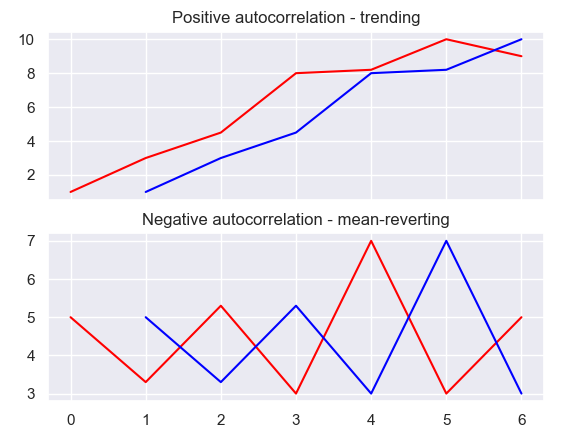


### Autocorrelation

Autocorrelation is the correlation of a time series with a lagged copy of itself

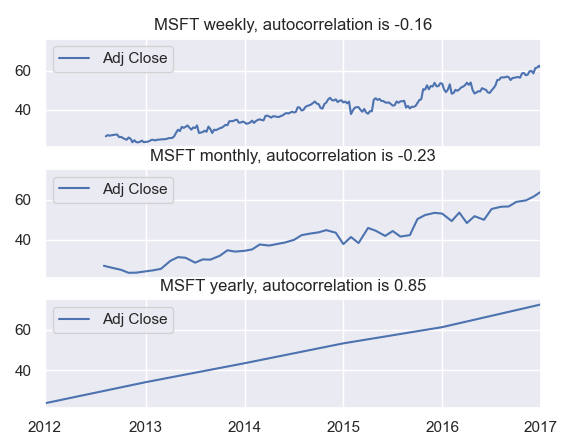


Positive autocorrelation indicates a series is trending. Negative autocorrelation indicates the series is mean-reverting



#### Resampling frequency and n-lagged meta parameter

When looking at MSFT, both weekly and monthly data produce negative autocorrelation. On the yearly basis, we finally get a strong positive autocorrelation, however, we can no longer see that between year 2015 and 2016, the price is consolidating and has a mean reverting nature.



## Some Simple Time Series

### Autocorrelation Function

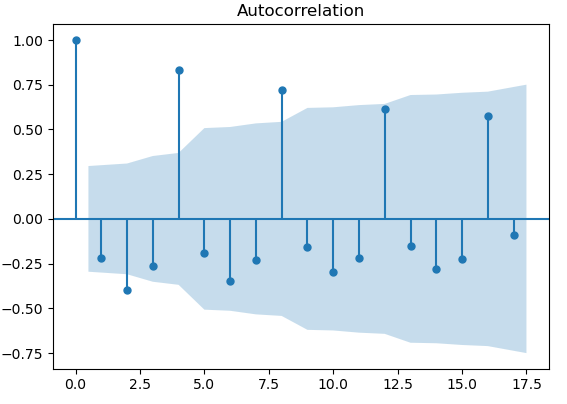
Autocorrelation Function or **ACF**, shows not only the lag-one autocorrelation, but the entire autocorrelation function for different lags. Any significant non-zero autocorrelations implies that the series can be forecast from the past.

from statsmodels.tsa.stattools import acf

plot\_acf(HRB, lags=20, alpha=0.05)

alpha is the confidence interval if the true auto correlation is 0. Alpha = 0.05 means that if the true autocorrelation is zero, there is a less than 95% chance that the observed autocorrelation value will fall out of the blue band.

We can see from the graph that at lag = 4, the autocorrelation is out of the band, it is significant enough that there is a true positive autocorrelation of the series, the observed value is not happening by random chance.



### White Noise

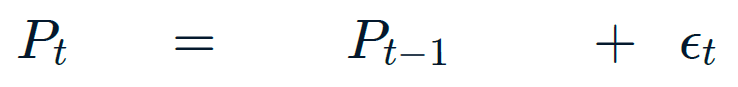
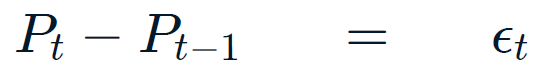
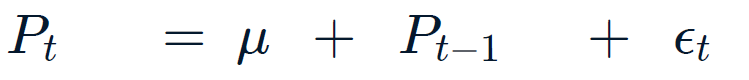
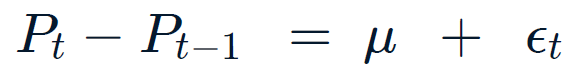
Definition of white noise

* Constant mean with time
* Constant variance with time
* Zero autocorrelation at all lags

The return of the stock market can be modeled by a white noise.

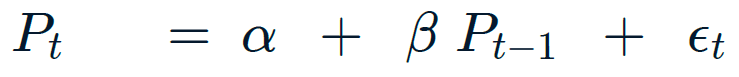
### Random Walk

##### In a random walk

* Today’s price is equal to yesterday’s price plus some noise. εt is the white noise  
  
* Then the change in price is white noise   
    
  If stocks prices follow a random walk, then the returns are white noise
* Random walk with drift, µ is the drift.   
  
* Change in price is white noise with non-zero mean:   
  

#### Statistical Test for Random Walk

We can do a regression test for random walk



Null hypothesis, the price is a random walk / beta equals to 1

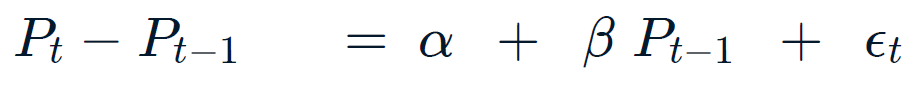
H0: β = 1 (random walk)

If the slope coefficient, beta, is not significantly different from 1, then we cannot reject the null hypothesis.

If the slope coefficient is significantly different from 1, H1: β <> 1, then we can reject the null hypothesis that the series is a random walk.

#### Statistical Test for Random Walk – Augmented Dickey-Fuller test

An equivalent of the above test is to test regression on lagged price and set the null hypothesis that β = 0



Test: H0: β = 0 (random walk), H1: β < 0 (not random walk)

If you add more lagged changes on the right-hand side, it’s the **Augmented Dickey-Fuller** test, ADF

### Stationarity

#### Definition of stationarity

* **Strong stationarity**: entire distribution of data is time-invariant
* **Weak stationarity**: mean, variance and autocorrelation are time-invariant. (i.e., for autocorrelation,   
  corr(Xt, Xt-τ), is only a function of τ

#### Why do we care?

If a process is not stationary, then it becomes difficult to model. Modeling involves estimating a set of parameters, and if a process is not stationary, and the parameters are different at each point in time, then there are too many parameters to estimate.

#### Examples of nonstationary series

A random walk is a common type of non-stationary series. The variance grows with time.

Seasonal series are also non-stationary. The mean varies with the time of the year.

A white noise is a stationary process.

#### Transforming nonstationary series into stationary series

SP500 prices is a non-stationary random walk. If you compute fist difference on the right, it becomes a stationary white noise.

H&R Block’s quarterly earnings are seasonal. If we take the difference with lag of 4, the transformed series looks stationary.

## Autoregressive (AR) Models

The autoregressive model specifies that the output variable depends **linearly** on its **own previous values** and on a stochastic term. Contrary to moving-average model, the autoregressive model is not always stationary and it may contain a unit root.

### AR(1) model



µ is the constant

φ is the model parameter, it specifies a portion of the value from the previous observation

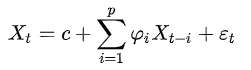
ε is the error term / white noise

When φ is 0, the time series is s white noise process

When φ is 1, it is a random walk

When -1 < φ < 1, the time series becomes **stationary**

### AR(p) model



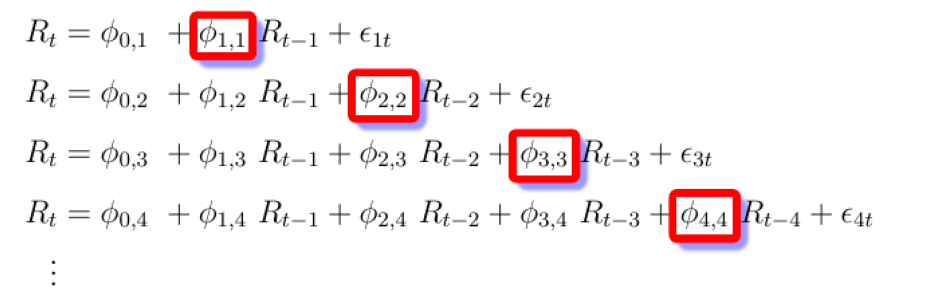
### Identifying the Order of an AR model

Two techniques to determine order

* Partial Autocorrelation Function
* Information criteria

#### PAF

The PAF measures the incremental benefits of adding another lag.



For example, the red box phi 4-4 in the bottom row, is the lag-4 value of the PAF, and it represents how significant adding a fourth lag is when you already have three lags.

#### Information Criteria

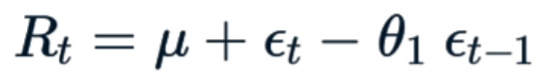
It adjusts goodness-of-fit for n umber of parameters.

Two popular adjusted goodness-of-fit measures are AIC (Akaike Information Criterion) and BIC (Bayesian Information Criterion)

## Moving Average (MA) and ARMA Models

### MA(1) model

Moving average model is a linear regression of the current value of the series against current and previous white noise. THE MA model is always stationary.



µ is the constant

φ is the model parameter, it specifies a portion of the value from the previous observation

εt is the error term of the current period and εt -1 is the error term of the last period

θ is the model parameters, it specific a fraction of the error from the last period to be taken into account

### MA(q) model

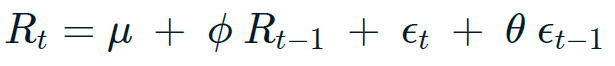


Autocorrelation function can be used to identify the best number of lags of a MA model.

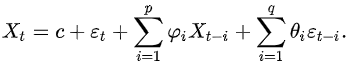
### ARMA models

An AMRA model is a combination of an AR model and a MA model. The AR part regress the variable on its own lagged values. The MA part regresses the pervious error terms.

ARMA(1, 1) model is defined as



ARMA(p, q) model is defined as



## Putting it All Together

### Cointegration

Two series Pt and Qt are random walks. But the linear combination Pt – cQt may not be a random walk.

If that’s true

Pt – cQt is forecastable. Pt and Qt are said to be cointegrated

### Two Steps to Test for Cointegration

Regress Pt on Qt and get slope c

Run Augmented Dickey-Fuller test on Pt – CQt to test for random walk.

Alternatively, can use coint() function in statsmodels that combine both steps.

# Visualizing Time Series Data in Python

# ARIMA Models in Python

## ARMA Models

### Unit Root

Unit root is a feature of some stochastic processes that can cause problems in statistical inference involving time series models. A time series with a unit root is non-stationary.

If the roots of the characteristic equation lie in side the unit circle (that is, have a modulus / absolute value) less than one, then the first difference of the process will be stationary. If there are **d** unit roots, the process will have to be differenced **d** times in order to make it stationary.

### Unit root test

In statistics, a unit root test tests whether a time series variable is non-stationary and possesses a unit root.

The null hypothesis is defined as the process of a unit root / non-stationary and

the alternative hypothesis is either stationarity, trend stationarity or explosive root.

### Key Concepts

Trend, seasonality and cyclicality

White noise and stationarity

### Make time series stationary

Use ADFuller test to test for stationarity

Take differences until a time series is stationary

Other transformations are log, square root, percentage change and etc.

### ARMA Models

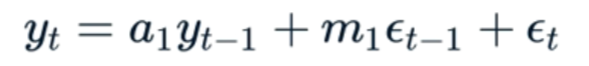
AR model, MA model and ARMA model

## Fitting the Future

### ARMAX Model

ARMAX stands for ARMA model with exogenous inputs. It uses external variables as well as time series.

#### ARMA(1, 1) model

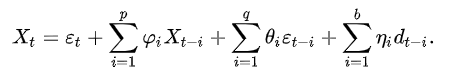


#### ARMAX(1, 1) model



X1 is the coefficient that models that linear dependency between the external variable z and the dependent variable Yt.

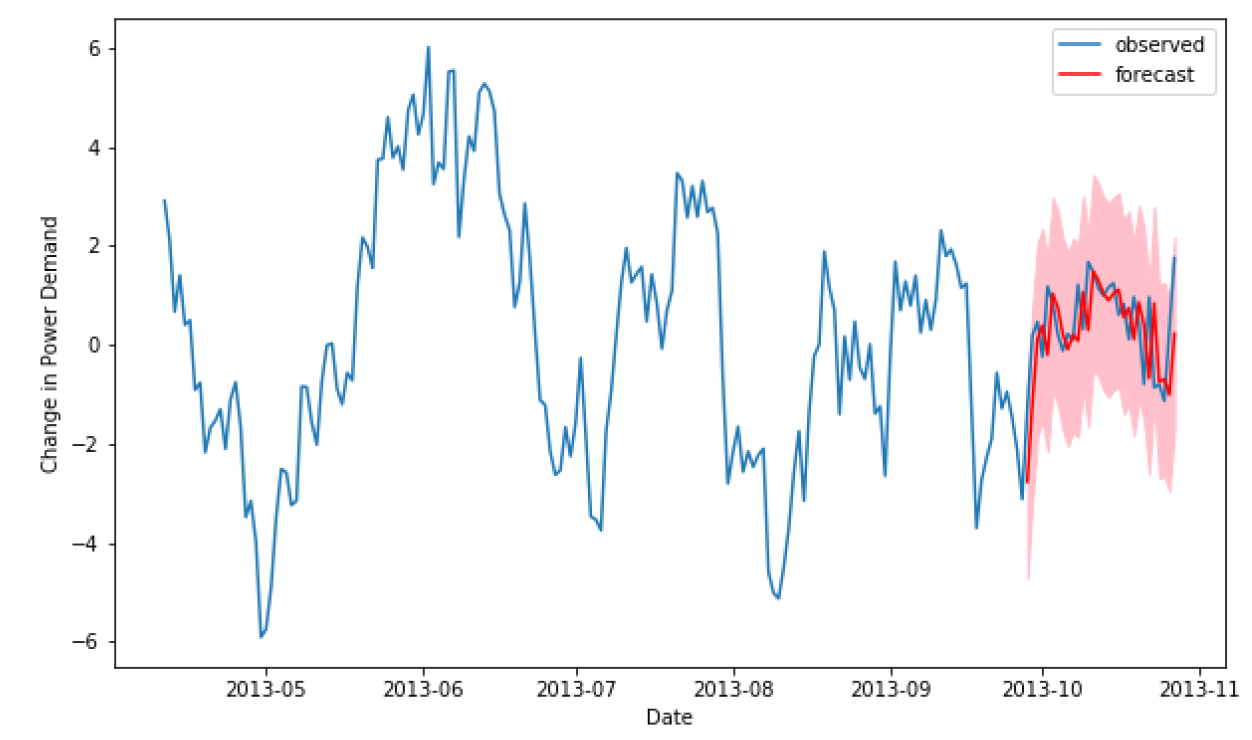
#### ARMAX(p, q, b) model



### Predicting the future

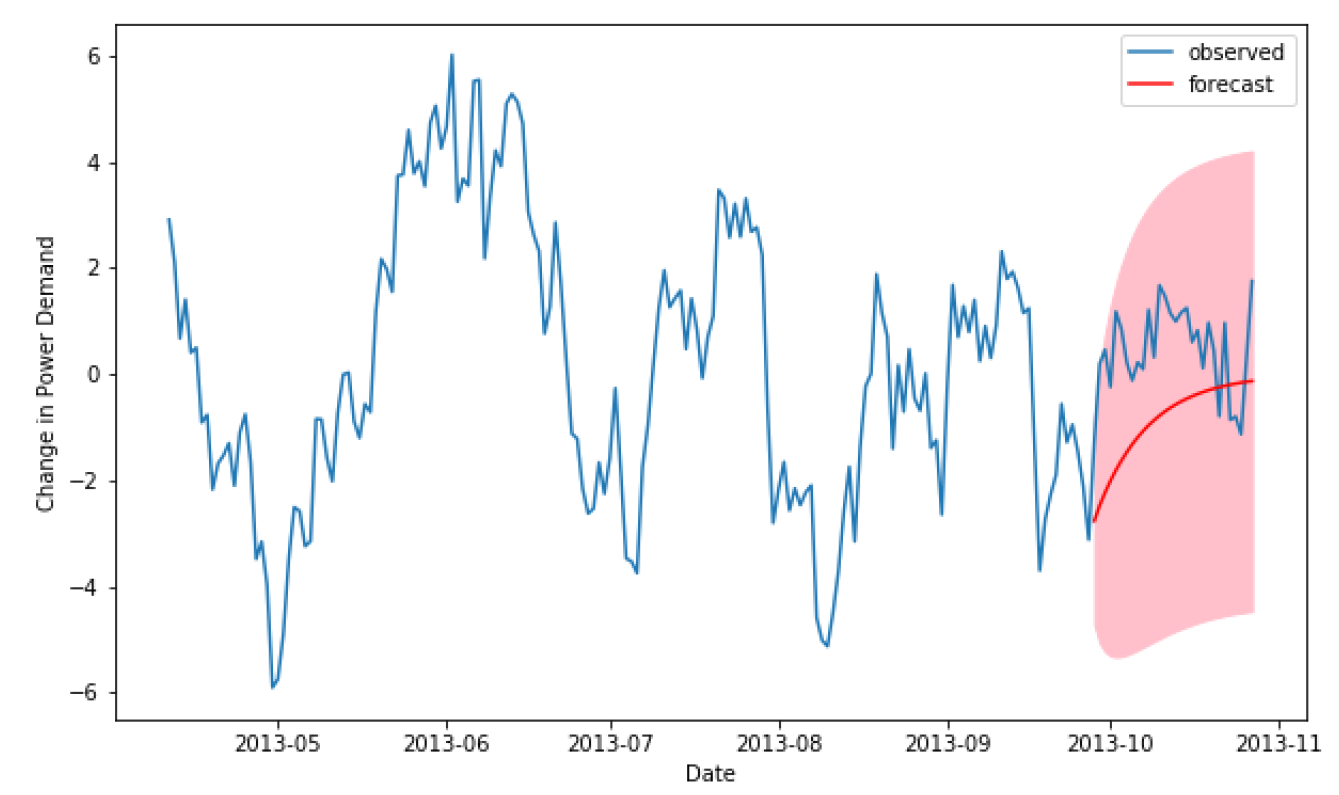
Once we have fitted an SARIMAX model, we can start to make predictions. We make **in-sample predictions** and **out-of-sample forecasts**.

**One-step-ahead** predictions, it takes an observed value Xt, from the sample and make a prediction for the next interval Xt+1. Then it takes the observed value Xt+1 and predicts Xt+2.



The red line is the predicted mean values. The pink shadow is the confidence interval.

Contrary to one-step-ahead prediction, dynamic prediction uses the predicted value to predict the next value. Since we don’t know the shock term at each step, the uncertainty can grow very quickly.



### Intro to ARIMA models

If a time series is non-stationary, we can take the first difference and it might become stationary. They we fit it to an ARMA model and predict the difference. Use cumsum() to reconstruct the value from the difference.

amazon\_diff = amazon.diff().dropna()

arma = SARIMAX(amazon\_diff, order=(2, 0, 2))

arma\_results = arma.fit()

arma\_diff\_forecast = arma\_results.get\_forecast(steps=10).predicted\_mean

arma\_int\_forecast = np.cumsum(arma\_diff\_forecast)

arma\_value\_forecast = arma\_int\_forecast + amazon.iloc[-1, 0]

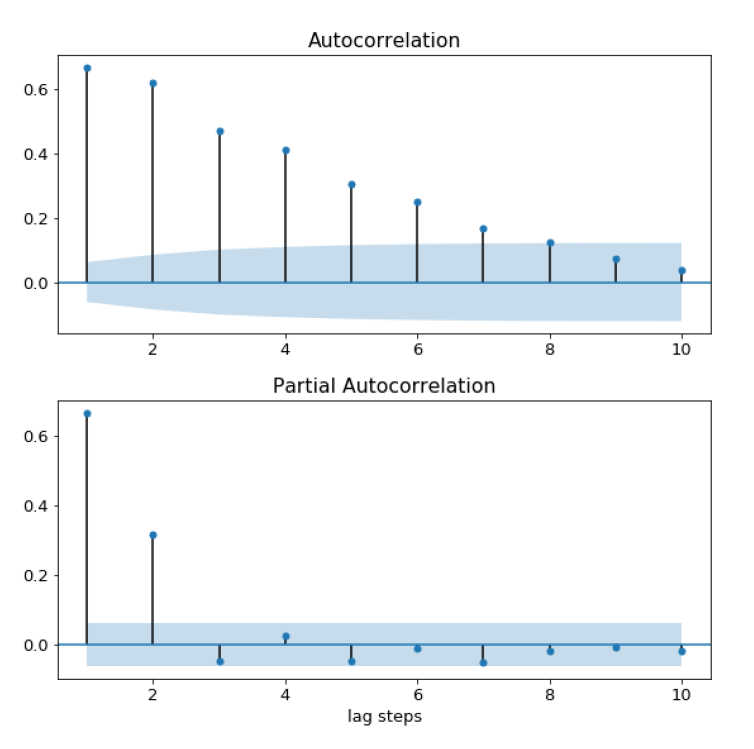
This process of taking the difference and integrating it back can be automated in the **ARIMA** model, auto-regressive **integrated** moving-average model.

## The Best of the Best Models

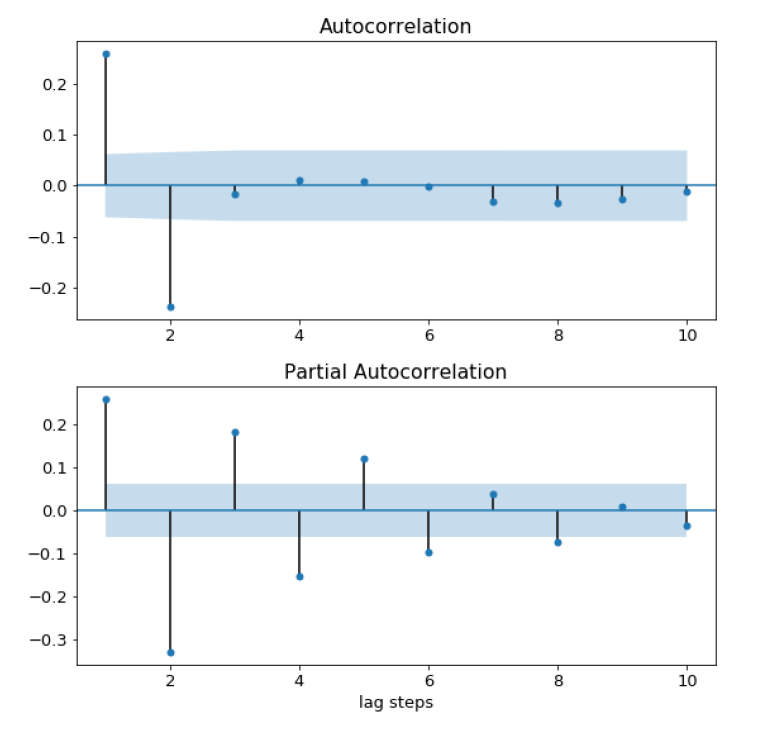
### Intro to ACF and PACF

PACF helps us to find AR(p) model

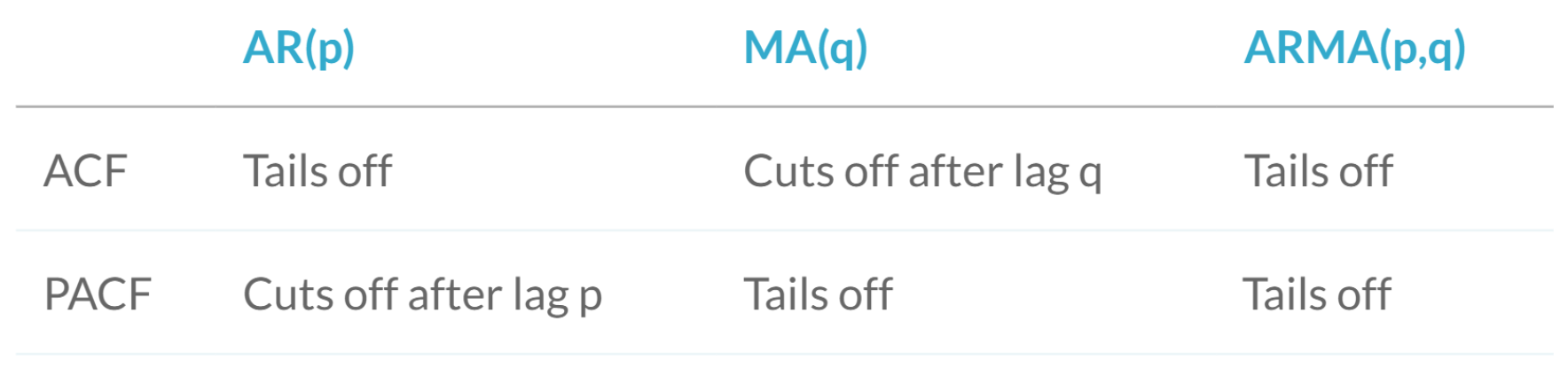
If ACF Tails off and PACF cuts off after lag p, then it is an AR(p) model



If ACF cuts off after lag q and PACF tails off, then it is an MA(q) model

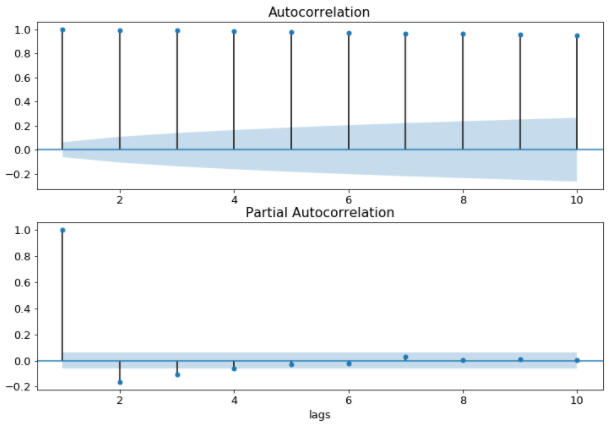


If both ACF and PACF tails off, then it is an ARMA(p, q) model, with unknown value of p and q

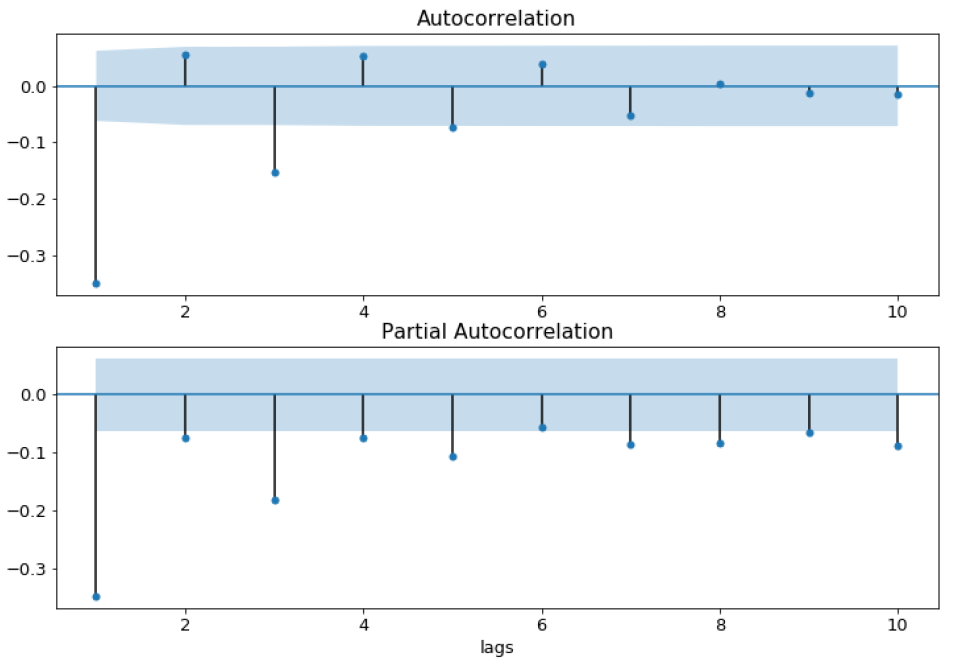


#### Over / under differencing using ACF and PACF

If ACF stays very high and PACF cuts off at lag 1, then the model is non-stationary and we need to take more differences.



If both ACF and PACF are very negative and cuts off at lag 1, then we took too many differences.



### Into to AIC and BIC

We can use for loops to iterate through a range of possible values for p, d and q. Record the AIC and BIC score, find the smallest one for the optimal model parameters.

### Model diagnostics

How good is our model?

We use residual the measure the performance of the model. Residual is the difference between the one-step-head prediction and the observed value.

If the model fits well, the residuals will be white Gaussian noise.

#### SARIMAXResults.plot\_diagnostics()

Produces 4 plots

* Standardized residuals over time
* Histogram plus estimated density of standardized residuals, along with a Normal(0, 1) densitry for reference
* Normal Q-Q plot, with normal reference line.
* Correlogram, ACF plot of the residual

#### SARIMAXResults.summary()

Prob(Q) – p-value for null hypothesis that residuals are uncorrelated

Prob(JB) – p-value for null hypothesis that residuals are normally distributed.

### Box-Jenkins method

Box-Jenkins method defines a set of steps to applying ARIMA models to a time series. The steps are: identification, estimation and validation.

## Seasonal ARIMA Models

### Seasonal time Series

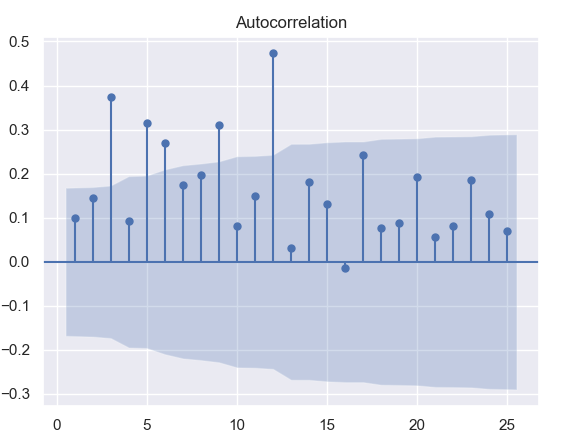
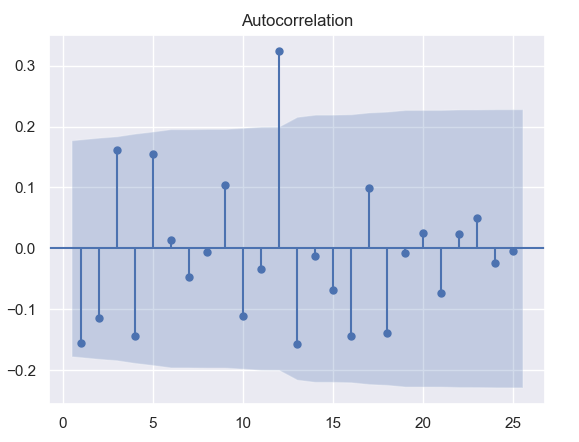
We can identify the interval of the seasonality by subtracting the data from an moving average and then plot the ACF

The left figure shows the ACF of the original time series. Lag 3, 5, 6, 9, and 12 are all significantly different from zero, thus making estimating of the seasonal period difficult.

After taking the difference of an rolling mean (with an arbitrary window)

water\_2 = water - water.rolling(15).mean()

THe ACF clearly shows that the period is 12

### SARIMA models

SARIMA stands for **Seasonal ARIMA** model

It is written as SARIMA (p, d, q) (P, D, Q) s

p, d, q are non-seasonal orders and P, D, Q are seasonal orders. S is the number of time steps per cycle

ARIMA(2, 0, 1) model



SARIMA(0, 0, 0)(2,, 0, 1)7 model



If the time series shows a trend, then we take the normal difference.

If there is a strong seasonal cycle, then we will also take the seasonal difference.

Once we have found the two order of differencing, and made the time series stationary, we need to find the other model orders.

### Automation and Saving

Pyramid-arima is a library that automates the process of finding the best p and q parameters for non-seasonal and seasonal portion of the model.

### SARIMA and Box-Jenkins

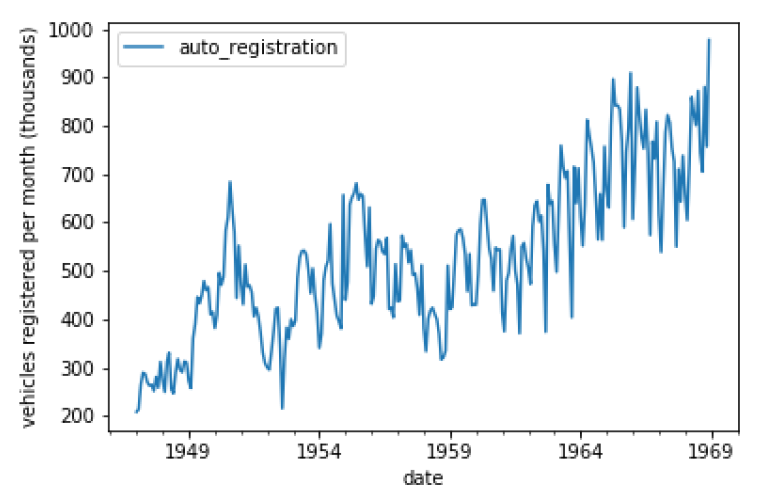
The Box-Jenkins method applies to SARIMA models as well. Then only difference between the SARIMA model and the ARIMA model is in the identification step. We need to

* Find seasonal period
* Find transforms to make data stationary
  + Seasonal and non-seasonal differencing
  + Other transforms

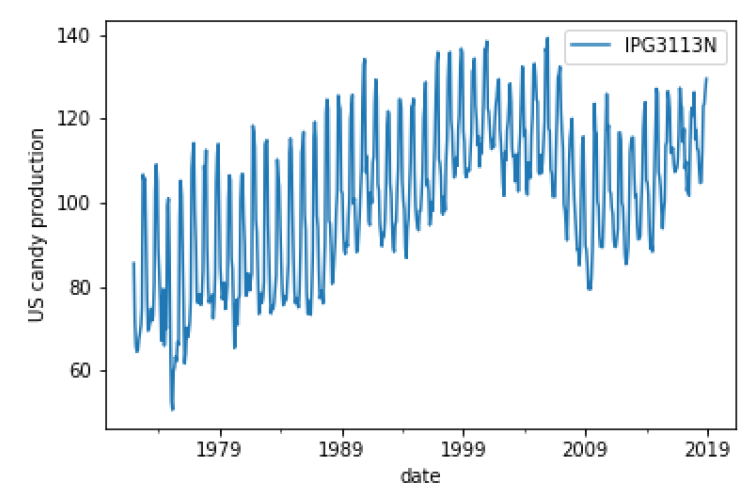
#### Principles on differencing

* Seasonal differencing D should only be 0 or 1
* Non-seasonal and seasonal differencing sums up to 2 at most. d + D should be [0, 2]

For time series with weak seasonal patterns, we only use seasonal differencing **if necessary**



For time series with strong seasonal pattern, **always** use seasonal differencing.



#### Additive vs multiplicative seasonality

Additives series = trend + season. We proceed as usual with differencing

Multiplicative series = trend x season. We Apply log transform first

# Machine Learning for Time Series Data in Python

## Time Series annd Machine Learning Primer

## Timer Series as Inputs to a Model

### The auditory envelope

The auditory envelope captures the general shape of the audio waveform. It relates to the total amount of audio energy present at each moment of time.

audio\_rectified = audio.apply(np.abs)

audio\_rectified\_smooth = audio\_rectified.rolling(50).mean()

### Computing the Tempogram

Tempogram estimates the tempo of a sound over time.

tempos.append(lr.beat.tempo(i\_audio.values, sr=sfreq,

                      hop\_length=2\*\*6, aggregate=None))

# Calculate statistics of each tempo

tempos\_mean = tempos.mean(axis=-1)

tempos\_std = tempos.std(axis=-1)

tempos\_max = tempos.max(axis=-1)

## Predicting Time Series Data

na

## Validating and Inspecting Time Series Models

### Creating features from the past

The biggest difference between time series and non-time series data is the relationship between data points. Because the data has a linear flow (matching the progression of time), patterns will persist over a span of datapoints. As a result, we can use information from the past in order to predict values in the future.

#### Smoothness and autocorrelation

It is important to consider how “smooth” is your data when fitting models with time series. The smoothness of your data reflects how much correlation there is between one time point and those that come before and after it. The extent to which previous timepoints are predictive of subsequent time points is often described as autocorrelation.

#### Fitting a model with time-shifted features

We create a time-shifted DataFrame, use it to predict the current value.

shifts = np.arange(1, 11).astype(int)

# Use a dictionary comprehension to create name: value pairs, one pair per shift

shifted\_data = {"lag\_{}\_day".format(day\_shift): prices\_perc.shift(

        day\_shift) for day\_shift in shifts}

# Convert into a DataFrame for subsequent use

prices\_perc\_shifted = pd.DataFrame(shifted\_data)

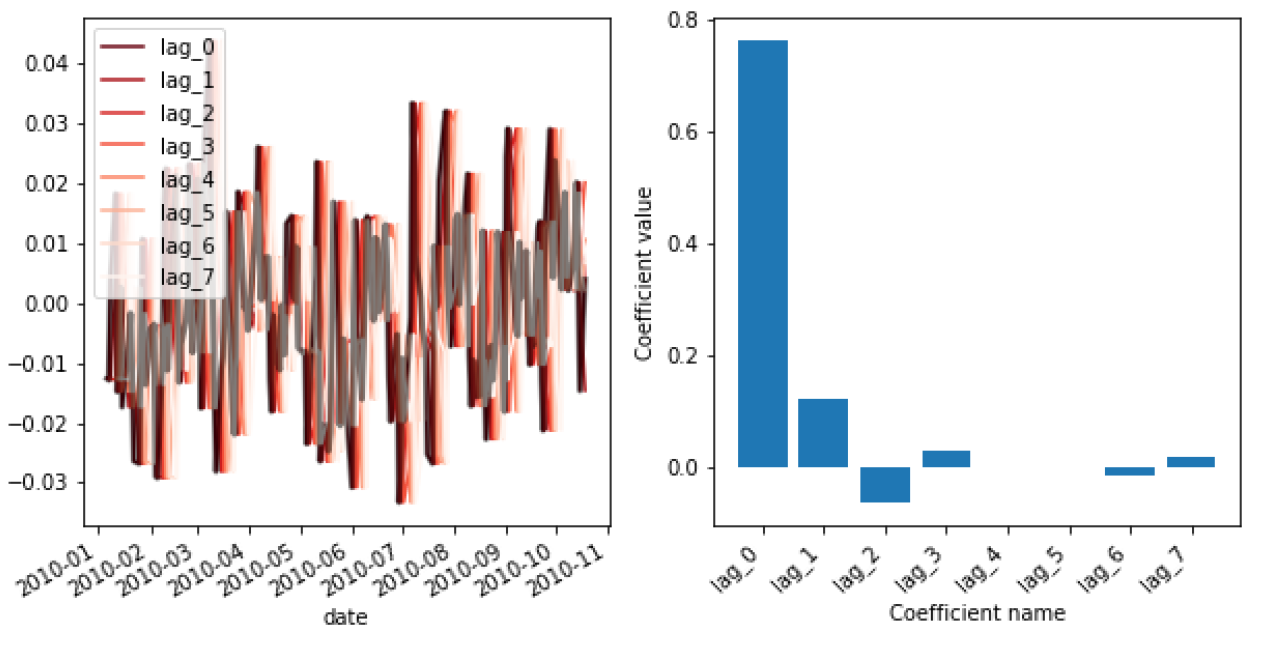
model = Ridge()

model.fit(X, y)

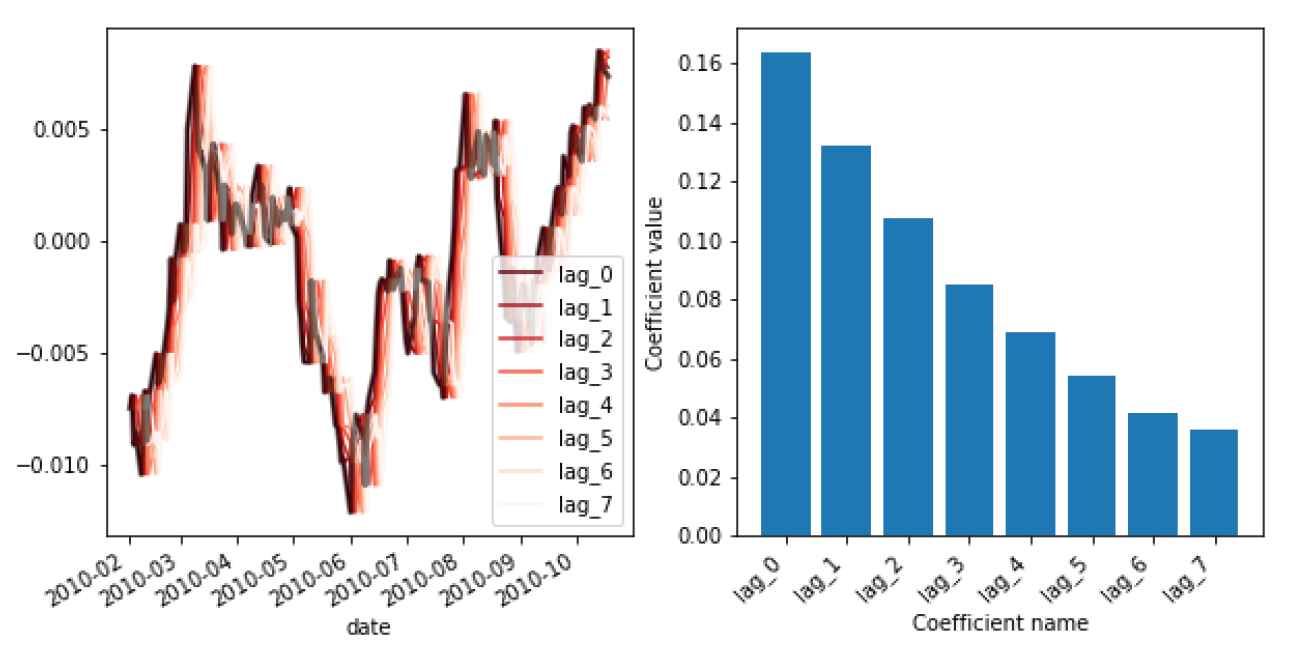
# Run the function to visualize model's coefficients

visualize\_coefficients(model.coef\_, prices\_perc\_shifted.columns, ax=axs[1])

For a time-series that resembles a white noise, the autocorrelation has low values starting from lag 1



For a time-series that is smooth and has a trend, the autocorrelation tails off smoothly.



### Cross-validating time series data

A standard pattern to use cross validation in scikit-learn

for tr, tt in cv.split(X, y):

    # Fit the model on training data

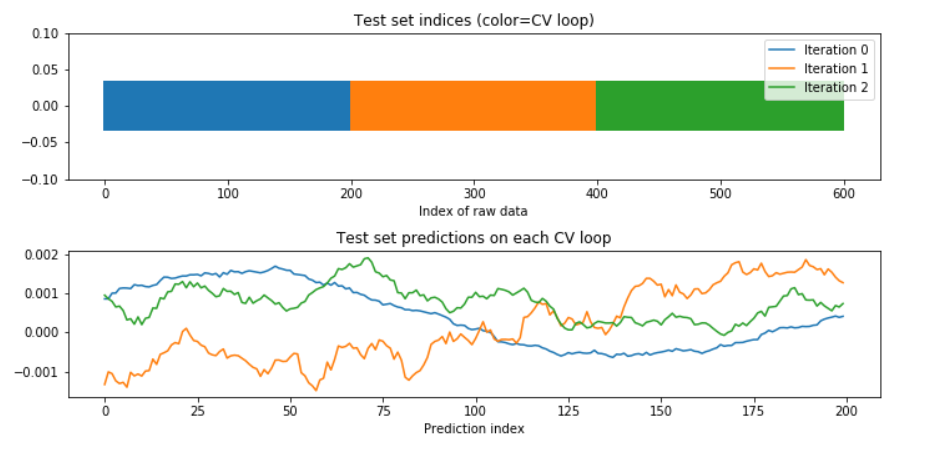
    model.fit(X[tr], y[tr])

model.score(X[tt], y[tt])

#### KFold CV

KFold CV splits the data into K folds, in each iteration, It uses K-1 folds as training data and the 1 fold as testing data.

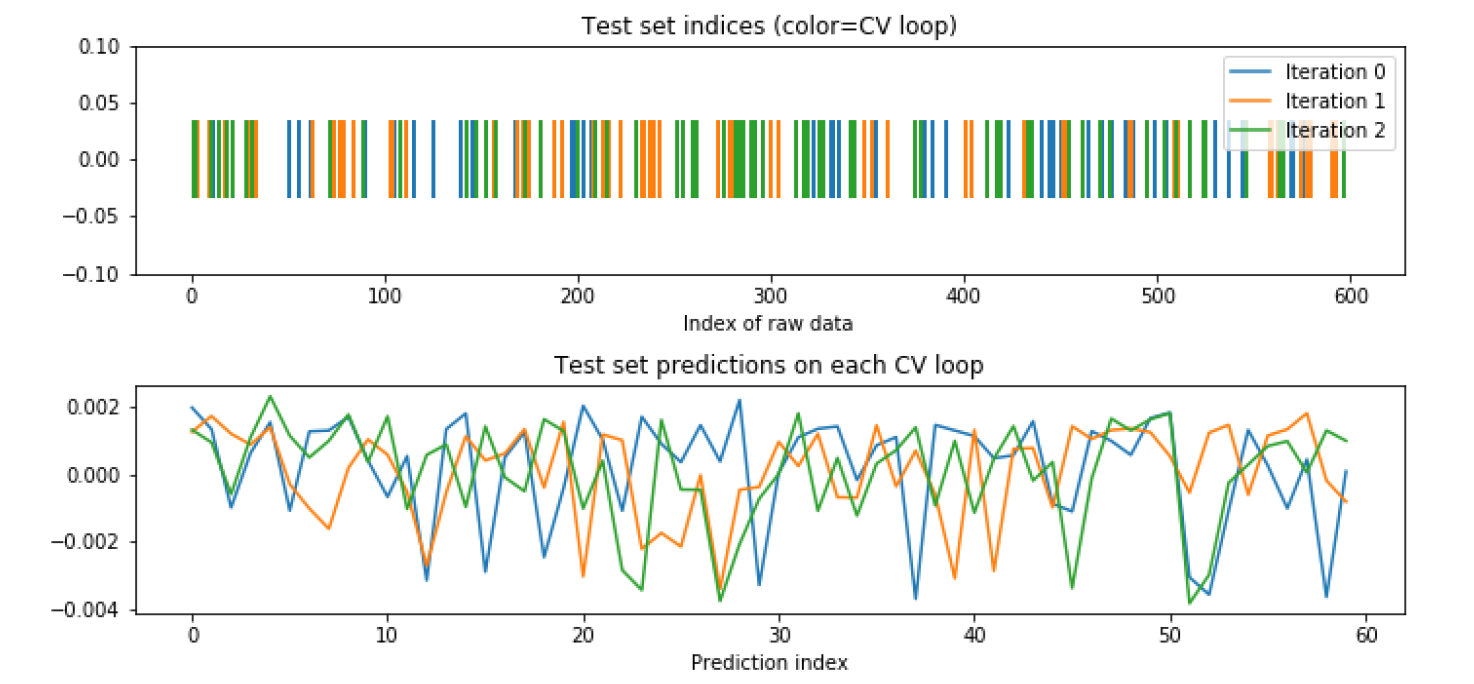
Visualize your CV behavior as a sanity check of the process.



#### ShuffleSplit CV

ShuffleSplit shuffles data.

* This only works if the data is independent and identically distributed.
* You should not shuffle your data when making predictions with time series.

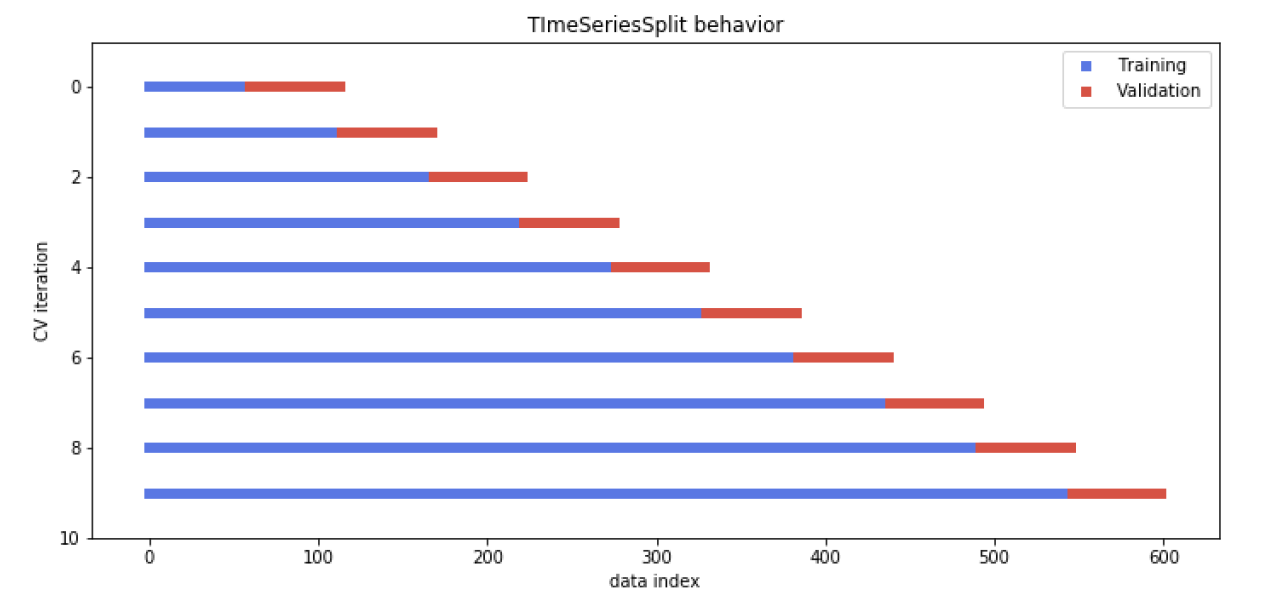


#### TimeSeriesSplit

When using KFold CV without shuffling, we still broke the linear passage of time.

* In general, we should not use datapoints in the future to predict data in the past
* We should always use training data from the past to predict the future.

The TimeSeriesSplit class helps us to achieve this.



### Stationarity and stability

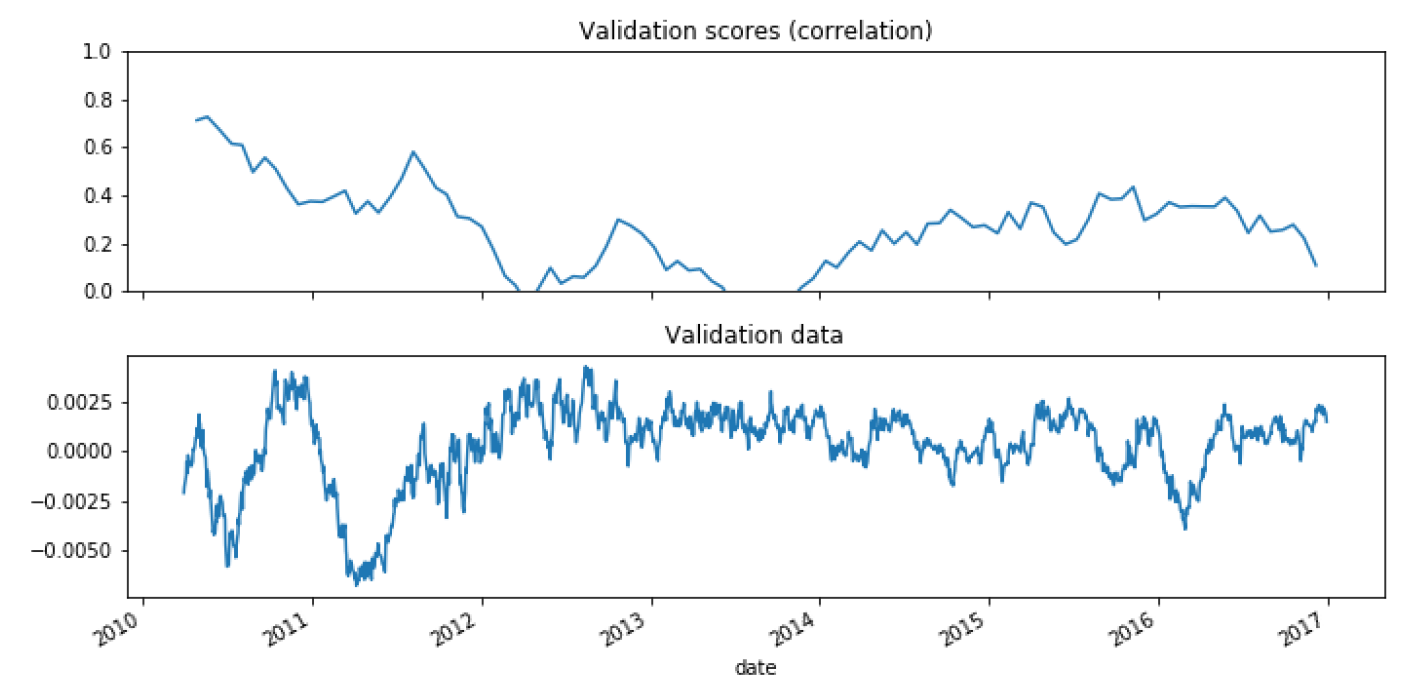
A stationary signal is one that does not change its statistical properties over time. It has the same mean, standard deviation, and general trends.

Most model assumes that the relationship between inputs and outputs is static. If this relationship change over time (as the data is not stationary), then the model will generate predictions using outdated relationships. How can we quantify annd corret for this?

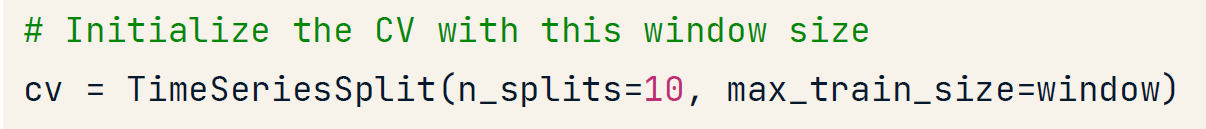
#### Cross Validation to Quantify Parameter Stability

We can use TimeSeriesSplit to cross validate the data. We calculate model parameters on each interaton. And we assess parameter stability across all CV splits.

In this example, the validation scores dips below zero during the period 2013 and 2014. It indicates that the statistical characterstics of the time series changed during the period.



We can correct this by fix th e window size of the CV function.



The validation score somehow improves

