Time Series Analysis (TSA) in Python - Linear Models to GARCH

November 08, 2016

Post Outline

- Motivation
- The Basics
  - Stationarity
  - Serial Correlation (Autocorrelation)
  - Why do we care about Serial Correlation?
- White Noise and Random Walks
- Linear Models
- Log-Linear Models
- Autoregressive Models - AR(p)
- Moving Average Models - MA(q)
- Autoregressive Moving Average Models - ARMA(p, q)
- Autoregressive Integrated Moving Average Models - ARIMA(p, d, q)

Motivation

Early in my quant finance journey, I learned various time series analysis techniques and how to use them but I failed to develop a deeper understanding of how the pieces fit together. I struggled to see the bigger picture of why we use certain models vs others, or how these models build on each other's weaknesses. The underlying purpose for employing these techniques eluded me for too long. That is, until I came to understand this:

> **Time series analysis attempts to understand the past and predict the future** - *Michael Halls Moore [Quantstart.com]*

By developing our time series analysis (TSA) skillset we are better able to understand what has already happened, *and* make better, more profitable, predictions of the future. Example applications include predicting future asset returns, future correlations/covariances, and future volatility.

This post is inspired by the great work Michael Halls Moore has done on his blog, Quantstart, especially his series on TSA. I thought translating some of his work to Python could help others who are less familiar with R. I have also adapted code from other bloggers as well. See References.

THIS ARTICLE IS A LIVING DOCUMENT. I WILL UPDATE IT WITH CORRECTIONS AS NEEDED AND MORE USEFUL INFORMATION AS TIME PASSES.

Before we begin let's import our Python libraries.

```
import os
import sys

import pandas as pd
import pandas_datareader.data as web
import numpy as np

import statsmodels.formula.api as smf
import statsmodels.tsa.api as smt
import statsmodels.api as sm
import scipy.stats as scs
from arch import arch_model

import matplotlib.pyplot as plt
import matplotlib as mpl
%matplotlib inline
p = print

p('Machine: {} {}\n'.format(os.uname().sysname,os.uname().machine))
p(sys.version)

# Machine: Linux x86_64
```

```
# 3.5.2 |Anaconda custom (64-bit)| (default, Jul  2 2016, 17:53:06)
# [GCC 4.4.7 20120313 (Red Hat 4.4.7-1)]
```

Let's use the **pandas_datareader** package to grab some sample data using the Yahoo Finance API.

```
end = '2015-01-01'
start = '2007-01-01'
get_px = lambda x: web.DataReader(x, 'yahoo', start=start, end=end)['Adj Close']

symbols = ['SPY','TLT','MSFT']
# raw adjusted close prices
data = pd.DataFrame({sym:get_px(sym) for sym in symbols})
# log returns
lrets = np.log(data/data.shift(1)).dropna()
```

The Basics

What is a Time Series?

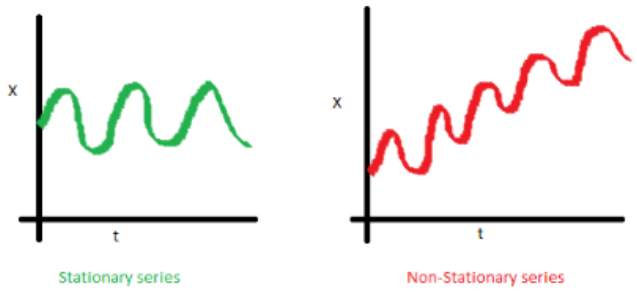
A time series is a series of data points indexed (or listed or graphed) in time order. - Wikipedia

Stationarity

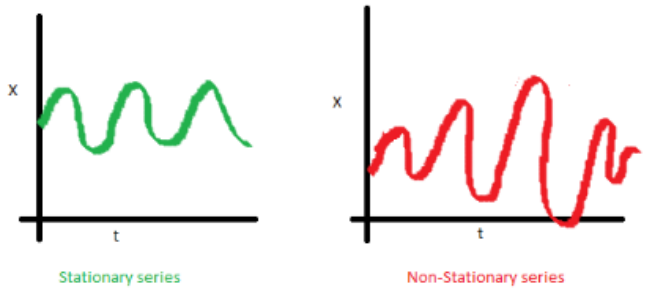
Here I use an infogrpahic found on SeanAbu.com. I find the pictures very intuitive.

What does it mean for data to be stationary?

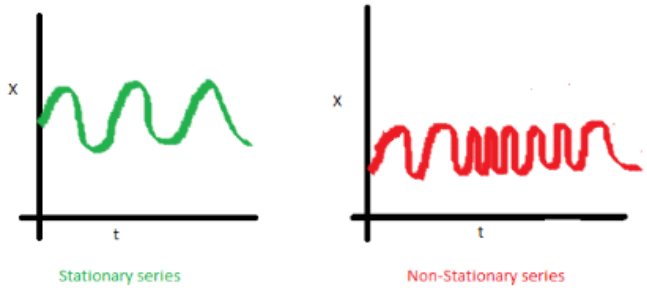
1. The mean of the series should not be a function of time. The red graph below is not stationary because the mean increases over time.



2. The variance of the series should not be a function of time. This property is known as homoscedasticity. Notice in the red graph the varying spread of data over time.



3. Finally, the covariance of the i th term and the (i + m) th term should not be a function of time. In the following graph, you will notice the spread becomes closer as the time increases. Hence, the covariance is not constant with time for the 'red series'.



SEANABU.COM

So what? Why do we care about stationarity?

- A stationary time series (TS) is simple to predict as we can assume that future statistical properties are the same or proportional to current statistical properties.
- Most of the models we use in TSA assume **covariance-stationarity (#3 above)**. This means **the descriptive statistics these models predict e.g. means, variances, and correlations, are only reliable if the TS is stationary and invalid otherwise.**

"For example, if the series is consistently increasing over time, the sample mean and variance will grow with the size of the sample, and they will always underestimate the mean and variance in

future periods. And if the mean and variance of a series are not well-defined, then neither are its correlations with other variables." - http://people.duke.edu/~rnau/411diff.htm

With that said, **most TS we encounter in finance is NOT stationary.** Therefore a large part of TSA involves identifying if the series we want to predict is stationary, and if it is not we must find ways to transform it such that it is stationary. (More on that later)

Serial Correlation (Autocorrelation)

Essentially when we model a time series we decompose the series into three components: trend, seasonal/cyclical, and random. The random component is called the residual or error. It is simply the difference between our predicted value(s) and the observed value(s). Serial correlation is when the residuals (errors) of our TS models are correlated with each other.

Why Do We Care about Serial Correlation?

We care about serial correlation because it is critical for the validity of our model predictions, and is intrinsically related to stationarity. Recall that the residuals (errors) of a *stationary* TS are serially *uncorrelated* by definition! If we fail to account for this in our models the standard errors of our coefficients are underestimated, inflating the size of our T-statistics. The result is too many Type-1 errors, where we reject our null hypothesis even when it is True! **In layman's terms, ignoring autocorrelation means our model predictions will be bunk, and we're likely to draw incorrect conclusions about the impact of the independent variables in our model.**

White Noise and Random Walks

White noise is the first Time Series Model (TSM) we need to understand. By definition a time series that is a white noise process has serially UNcorrelated errors and the expected mean of those errors is equal to zero. Another description for serially uncorrelated errors is, independent and identically distributed (i.i.d.). This is important because, if our TSM is appropriate and successful at capturing the underlying process, the residuals of our model will be i.i.d. and resemble a white noise process. Therefore part of TSA is literally trying to fit a model to the time series such that the residual series is indistinguishable from white noise.

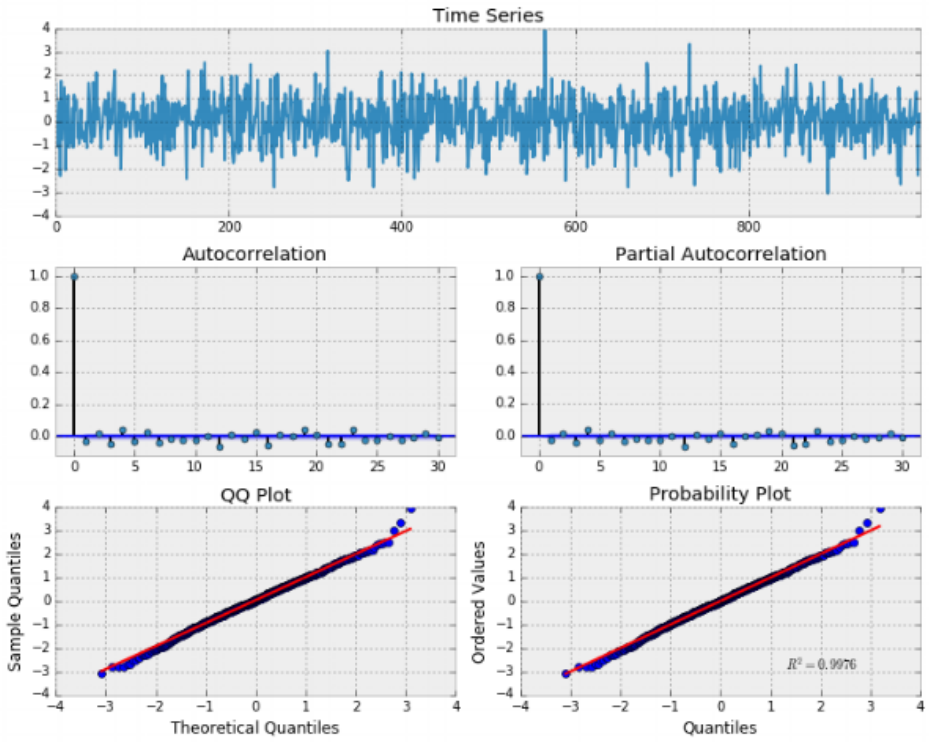
Let's simulate a white noise process and view it. Below I introduce a convenience function for plotting the time series and analyzing the serial correlation visually. This code was adapted from the blog Seanabu.com

```
def tsplot(y, lags=None, figsize=(10, 8), style='bmh'):
    if not isinstance(y, pd.Series):
        y = pd.Series(y)
    with plt.style.context(style):
        fig = plt.figure(figsize=figsize)
        #mpl.rcParams['font.family'] = 'Ubuntu Mono'
        layout = (3, 2)
        ts_ax = plt.subplot2grid(layout, (0, 0), colspan=2)
        acf_ax = plt.subplot2grid(layout, (1, 0))
        pacf_ax = plt.subplot2grid(layout, (1, 1))
        qq_ax = plt.subplot2grid(layout, (2, 0))
        pp_ax = plt.subplot2grid(layout, (2, 1))

        y.plot(ax=ts_ax)
        ts_ax.set_title('Time Series Analysis Plots')
```

```
        smt.graphics.plot_acf(y, lags=lags, ax=acf_ax, alpha=0.5)
        smt.graphics.plot_pacf(y, lags=lags, ax=pacf_ax, alpha=0.5)
        sm.qqplot(y, line='s', ax=qq_ax)
        qq_ax.set_title('QQ Plot')
        scs.probplot(y, sparams=(y.mean(), y.std()), plot=pp_ax)

        plt.tight_layout()
    return
```

We can model a white noise process easily and output the TS plot for visual inspection.

```
np.random.seed(1)

# plot of discrete white noise
randser = np.random.normal(size=1000)
tsplot(randser, lags=30)
```



GUASSIAN WHITE NOISE

We can see that process appears to be random and centered about zero. The autocorrelation (ACF) and partial autocorrelation (PACF) plots also indicate no significant serial correlation. Keep in mind we should see approximately 5% significance in the autocorrelation plots due to pure chance as a result of sampling from the Normal distribution. Below that we can see the QQ and Probability Plots, which compares the distribution of our data with another theoretical distribution. In this case, that theoretical distribution is the standard normal distribution. Clearly our data is distributed randomly, and appears to follow Gaussian (Normal) white noise, as it should.

```
p("Random Series\n -------------\nmean: {:.3f}\nvariance: {:.3f}\nstandard deviation: {:.3f}"
.format(randser.mean(), randser.var(), randser.std()))
```

```
# Random Series
# -------------
# mean: 0.039
# variance: 0.962
# standard deviation: 0.981
```

A Random Walk is defined below:

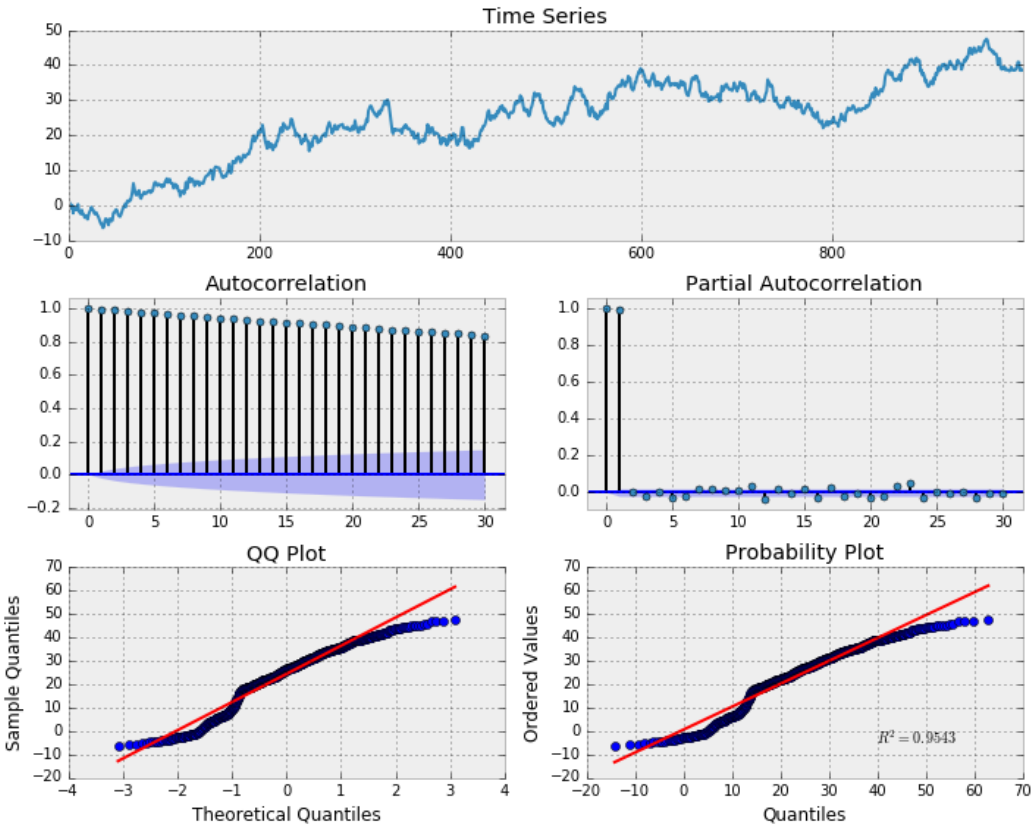
> ### Random Walk
>
> A *random walk* is a time series model x_t such that $x_t = x_{t-1} + w_t$, where w_t is a discrete white noise series.

- MICHAEL HALLS MOORE [QUANTSART.COM]

The significance of a random walk is that it is **non-stationary** because the covariance between observations is time-dependent. If the TS we are modeling is a random walk it is unpredictable.

Let's simulate a random walk using the "numpy.random.normal(size=our_sample_size)" function to sample from the standard normal distribution.

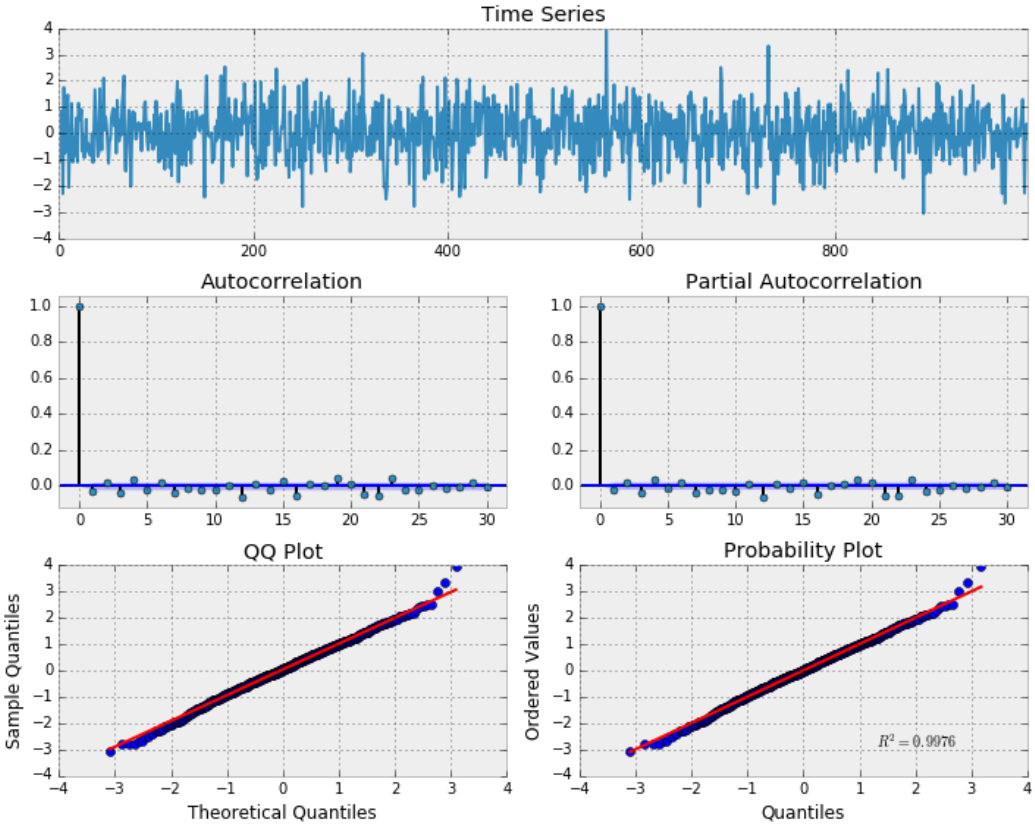
```
# Random Walk without a drift

np.random.seed(1)
n_samples = 1000

x = w = np.random.normal(size=n_samples)
for t in range(n_samples):
    x[t] = x[t-1] + w[t]

_ = tsplot(x, lags=30)
```



RANDOM WALK WITHOUT A DRIFT

Clearly our TS is not stationary. Let's find out if the random walk model is a good fit for our simulated data. Recall that a random walk is ***xt = xt-1 + wt***. Using algebra we can say that ***xt - xt-1 = wt***. Thus the first differences of our random walk series should equal a white noise process! We can use the "**np.diff()**" function on our TS and see if this holds.

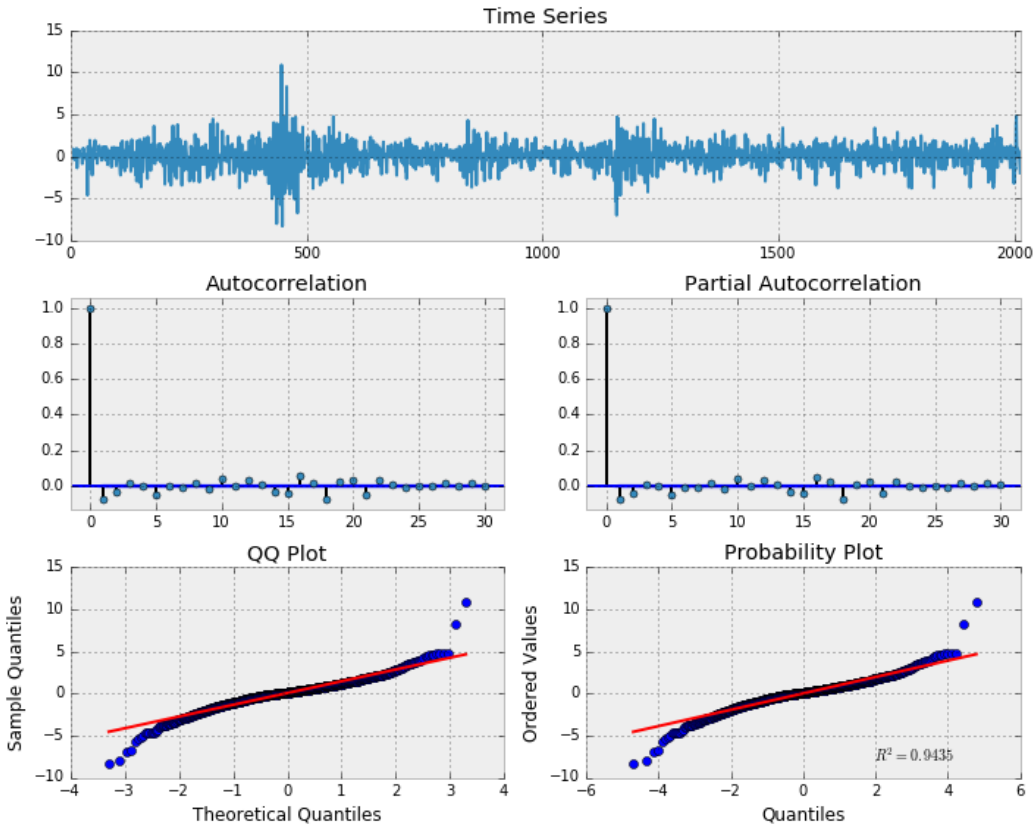
```
# First difference of simulated Random Walk series

_ = tsplot(np.diff(x), lags=30)
```



FIRST DIFFERENCE OF A RANDOM WALK SERIES

Our definition holds as this looks exactly like a white noise process. What if we fit a random walk to the first difference of SPY's prices?

```
# First difference of SPY prices
_ = tsplot(np.diff(data.SPY), lags=30)
```



FITTING A RANDOM WALK MODEL TO SPY ETF PRICES

Wow, it's quite similar to white noise. However, notice the shape of the QQ and Probability plots. This indicates that the process is close to normality but with **'heavy tails'.** There also appears to be some significant serial correlation in the ACF, and PACF plots around lags 1, 5?, 16?, 18 and 21. This means that there should be better models to describe the actual price change process.

Linear Models

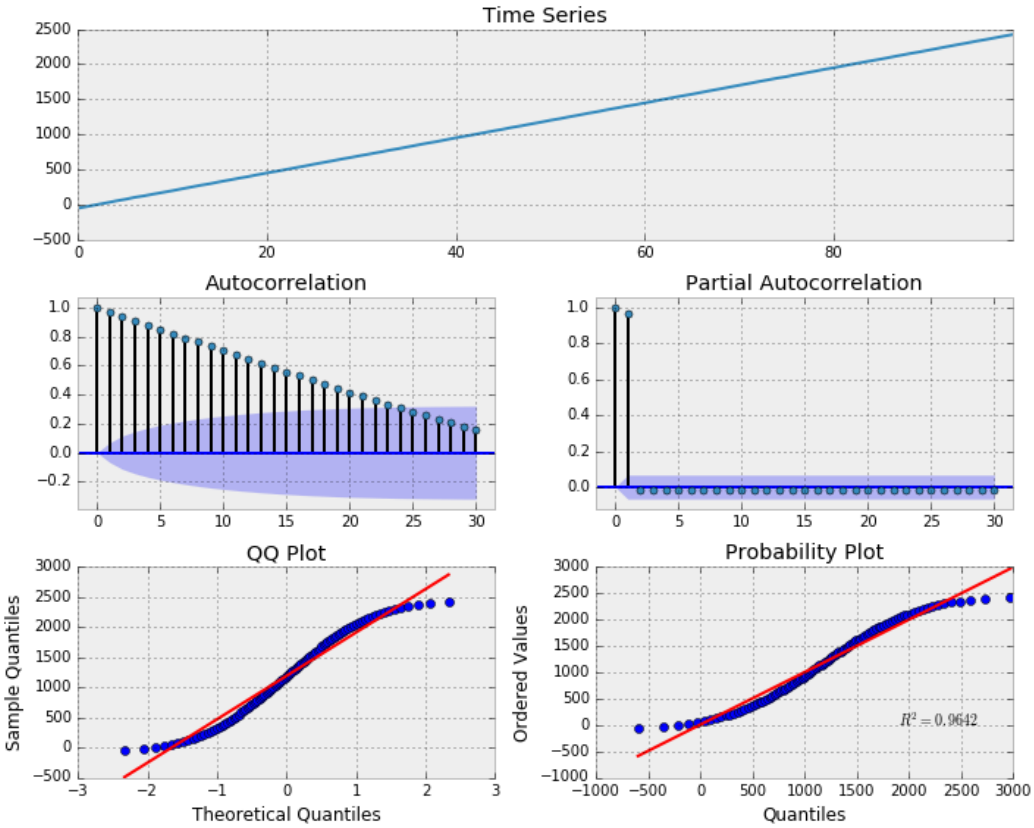
Linear models aka trend models represent a TS that can be graphed using a straight line. The basic equation is:

$$y_t = b_0 + b_1 t + \epsilon_t$$

In this model the value of the dependent variable is determined by the beta coefficients and a singular independent variable, *time.* An example could be a company's sales that increase by the same amount at each time step. Let's look at a contrived example below. In this simulation we assume Firm ABC sales regardless of time are -$50.00 (*beta 0 or the intercept term*) and +$25.00 (*beta 1*) at every time step.

```
# simulate linear trend
# example Firm ABC sales are -$50 by default and +$25 at every time step

w = np.random.randn(100)
y = np.empty_like(w)
```

```
b0 = -50.
b1 = 25.
for t in range(len(w)):
    y[t] = b0 + b1*t + w[t]

_ = tsplot(y, lags=lags)
```



LINEAR TREND MODEL SIMULATION

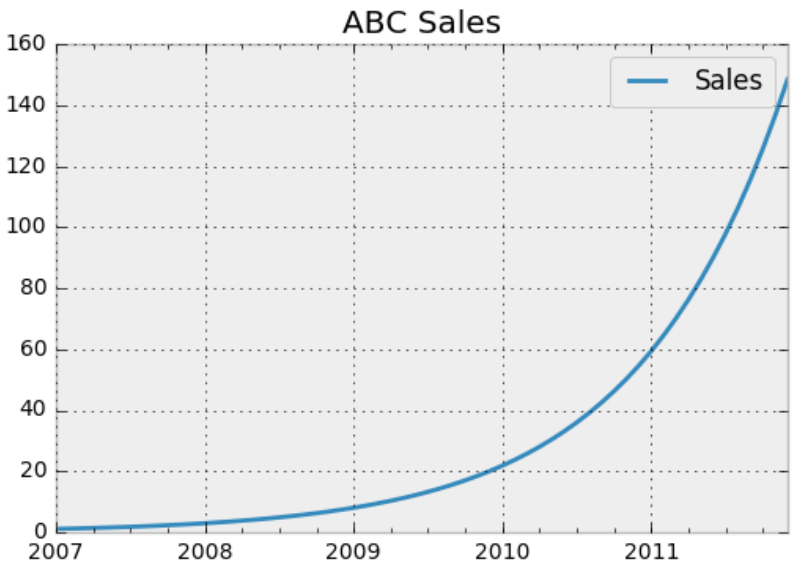
Here we can see that the residuals of the model are correlated and linearly decreasing as a function of the lag. The distribution is approximately normal. Before using this model to make predictions we would have to account for and remove the obvious autocorrelation present in the series. The significance of the PACF at lag 1 indicates that an *autoregressive* model may be appropriate.

Log-Linear Models

These models are similar to linear models except that the data points form an exponential function that represent a constant rate of change with respect to each time step. For example, firm ABC's sales increasing X% at each time step. When plotting the simulated sales data you get a curve that looks like this:

```
# Simulate ABC exponential growth

# fake dates
idx = pd.date_range('2007-01-01', '2012-01-01', freq='M')
```

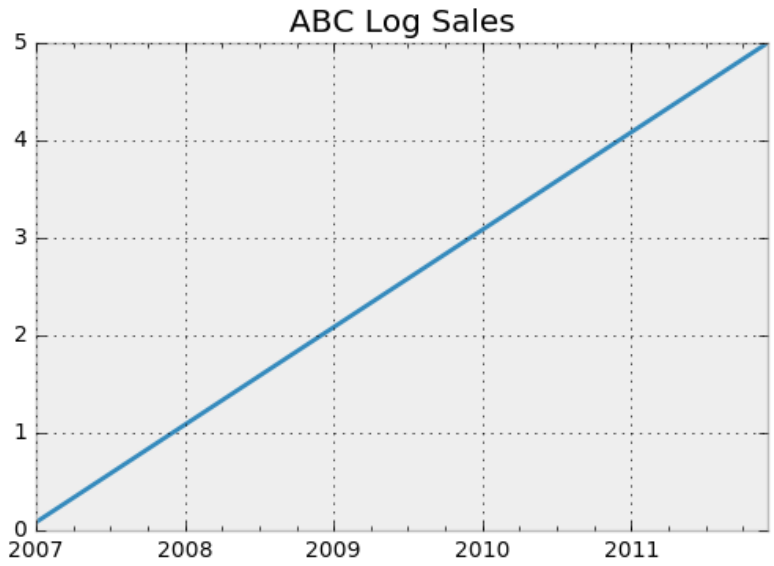
```
# fake sales increasing at exponential rate
sales = [np.exp( x/12 ) for x in range(1, len(idx)+1)]

# create dataframe and plot
df = pd.DataFrame(sales, columns=['Sales'], index=idx)

with plt.style.context('bmh'):
    df.plot()
    plt.title('ABC Sales')
```



SIMULATED EXPONENTIAL FUNCTION

We can then transform the data by taking the natural logarithm of sales. Now a linear regression is a much better fit to the data.

```
# ABC log sales

with plt.style.context('bmh'):
    pd.Series(np.log(sales), index=idx).plot()
    plt.title('ABC Log Sales')
```



NATURAL LOGARITHM OF EXPONENTIAL FUNCTION

These models have a fatal weakness as discussed previously. They assume serially UNcorrelated errors, which as we have seen in the linear model example is not true. In real life, TS data usually violates our stationary assumptions which motivates our progression to autoregressive models.

Autoregressive Models - AR(p)

When the dependent variable is regressed against one or more lagged values of itself the model is called autoregressive. The formula looks like this:

$$x_t = \alpha_1 x_{t-1} + \ldots + \alpha_p x_{t-p} + \omega_t$$

$$= \sum_{i=1}^{P} \alpha_i x_{t-i} + \omega_t$$

AR(P) MODEL FORMULA

When you describe the **"order"** of the model, as in, an AR model of order **"p",** the p represents the number of lagged variables used within the model. For example an AR(2) model or *second-order* autoregressive model looks like this:

$$x_t = \alpha_1 x_{t-1} + \alpha_2 x_{t-2} + \omega_t$$

AR(2) MODEL FORMULA

Here, alpha (a) is the coefficient, and omega (w) is a white noise term. Alpha cannot equal zero in an AR

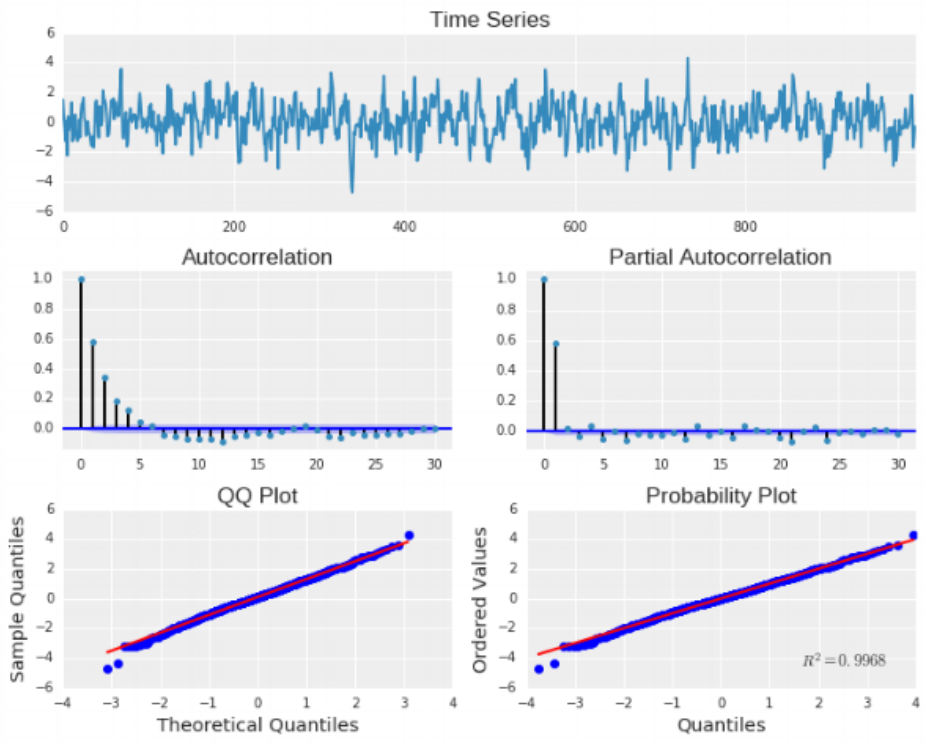
model. Note that an AR(1) model with alpha set equal to 1 is a *random walk* and therefore not stationary.

$$x_t = 1x_{t-1} + \omega_t$$

AR(1) MODEL WITH ALPHA = 1; RANDOM WALK

Let's simulate an AR(1) model with alpha set equal to 0.6

```
# Simulate an AR(1) process with alpha = 0.6

np.random.seed(1)
n_samples = int(1000)
a = 0.6
x = w = np.random.normal(size=n_samples)

for t in range(n_samples):
    x[t] = a*x[t-1] + w[t]

_ = tsplot(x, lags=lags)
```



AR(1) MODEL WITH ALPHA = 0.6

As expected the distribution of our simulated AR(1) model is normal. There is significant serial correlation between lagged values especially at lag 1 as evidenced by the PACF plot.

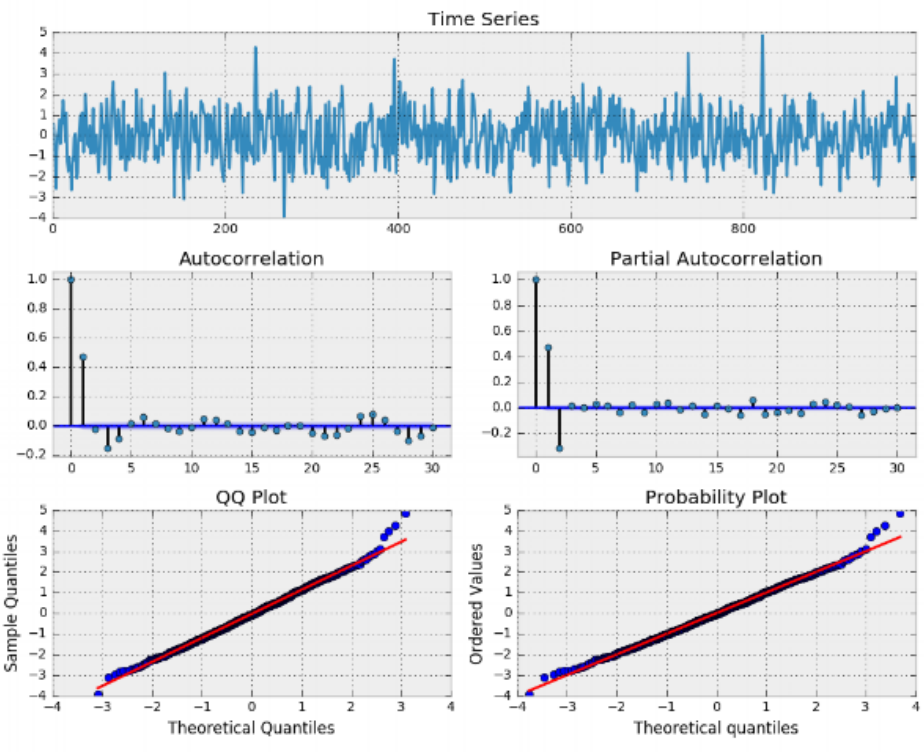
Now we can fit an AR(p) model using Python's statsmodels. First we fit the AR model to our simulated data and return the estimated alpha coefficient. Then we use the statsmodels function **"select_order()"** to see if the fitted model will select the correct lag. If the AR model is correct the estimated alpha coefficient will be close to our true alpha of 0.6 and the selected order will equal 1.

```
# Fit an AR(p) model to simulated AR(1) model with alpha = 0.6

mdl = smt.AR(x).fit(maxlag=30, ic='aic', trend='nc')
%time est_order = smt.AR(x).select_order(
    maxlag=30, ic='aic', trend='nc')

true_order = 1
p('\nalpha estimate: {:3.5f} | best lag order = {}'
  .format(mdl.params[0], est_order))
p('\ntrue alpha = {} | true order = {}'
  .format(a, true_order))
```

```
alpha estimate: 0.58227 | best lag order = 1

true alpha = 0.6 | true order = 1
```

Looks like we were able to recover the underlying parameters of our simulated data. Let's simulate an AR(2) process with alpha_1 = 0.666 and alpha_2 = -0.333. For this we make use of statsmodel's **"arma_generate_samples()"** function. This function allows us to simulate an AR model of arbitrary orders. Note that there are some peculiarities of Python's version which requires us to take some extra steps before using the function.

```
# Simulate an AR(2) process

n = int(1000)
alphas = np.array([.666, -.333])
betas = np.array([0.])

# Python requires us to specify the zero-lag value which is 1
# Also note that the alphas for the AR model must be negated
# We also set the betas for the MA equal to 0 for an AR(p) model
# For more information see the examples at statsmodels.org
ar = np.r_[1, -alphas]
ma = np.r_[1, betas]

ar2 = smt.arma_generate_sample(ar=ar, ma=ma, nsample=n)
_ = tsplot(ar2, lags=lags)
```

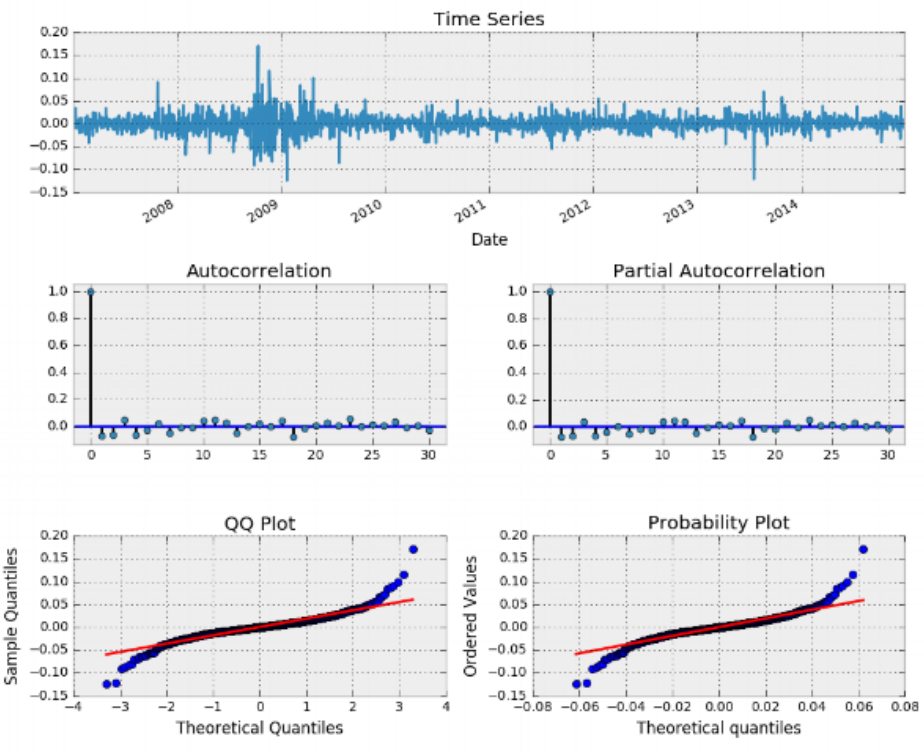


AR(2) SIMULATION WITH ALPHA_1 = 0.666 AND ALPHA_2 = -0.333

Let's see if we can recover the correct parameters.

```
# Fit an AR(p) model to simulated AR(2) process

max_lag = 10
mdl = smt.AR(ar2).fit(maxlag=max_lag, ic='aic', trend='nc')
est_order = smt.AR(ar2).select_order(
    maxlag=max_lag, ic='aic', trend='nc')

true_order = 2
p('\ncoef estimate: {:3.4f} {:3.4f} | best lag order = {}'
  .format(mdl.params[0],mdl.params[1], est_order))
p('\ntrue coefs = {} | true order = {}'
  .format([.666,-.333], true_order))

# coef estimate: 0.6291 -0.3196 | best lag order = 2
# true coefs = [0.666, -0.333] | true order = 2
```

Not bad. Let's see how the AR(p) model will fit MSFT log returns. Here is the return TS.



MSFT LOG RETURNS TIME SERIES

```
# Select best lag order for MSFT returns

max_lag = 30
mdl = smt.AR(lrets.MSFT).fit(maxlag=max_lag, ic='aic', trend='nc')
est_order = smt.AR(lrets.MSFT).select_order(
    maxlag=max_lag, ic='aic', trend='nc')

p('best estimated lag order = {}'.format(est_order))

# best estimated lag order = 23
```

The best order is 23 lags or 23 parameters! Any model with this many parameters is unlikely to be useful in practice. Clearly there is more complexity underlying the returns process than this model can explain.

Moving Average Models - MA(q)

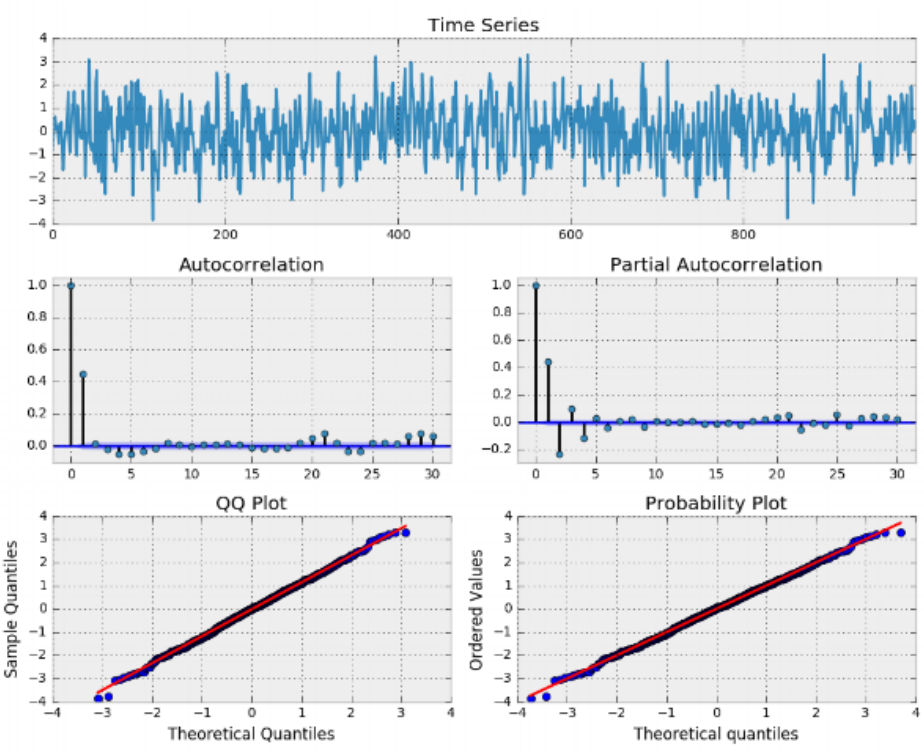
MA(q) models are very similar to AR(p) models. The difference is that the MA(q) model is a linear combination of past white noise error terms as opposed to a linear combo of past observations like the AR(p) model. The motivation for the MA model is that we can observe "shocks" in the error process directly by fitting a model to the error terms. In an AR(p) model these shocks are observed indirectly by using the ACF on the series of past observations. The formula for an MA(q) model is:

$$x_t = \omega_t + \beta_1\omega_{t-1} + \ldots + \beta_p\omega_{t-p}$$

$$= \omega_t + \sum_{i=1}^{P} \beta_i\omega_{t-i}$$

Omega (w) is white noise with E(wt) = 0 and variance of sigma squared. Let's simulate this process using beta=0.6 and specifying the AR(p) alpha equal to 0.

```
# Simulate an MA(1) process

n = int(1000)

# set the AR(p) alphas equal to 0
alphas = np.array([0.])
betas = np.array([0.6])

# add zero-lag and negate alphas
ar = np.r_[1, -alphas]
ma = np.r_[1, betas]

ma1 = smt.arma_generate_sample(ar=ar, ma=ma, nsample=n)
_ = tsplot(ma1, lags=30)
```



SIMULATED MA(1) PROCESS WITH BETA=0.6

The ACF function shows that lag 1 is significant which indicates that a MA(1) model may be appropriate for our simulated series. I'm not sure how to interpret the PACF showing significance at lags 2, 3, and 4 when the ACF only shows significance at lag 1. Regardless we can now attempt to fit a MA(1) model to our simulated data. We can use the same statsmodels **"ARMA()"** function specifying our chosen orders. We call on its **"fit()"** method to return the model output.

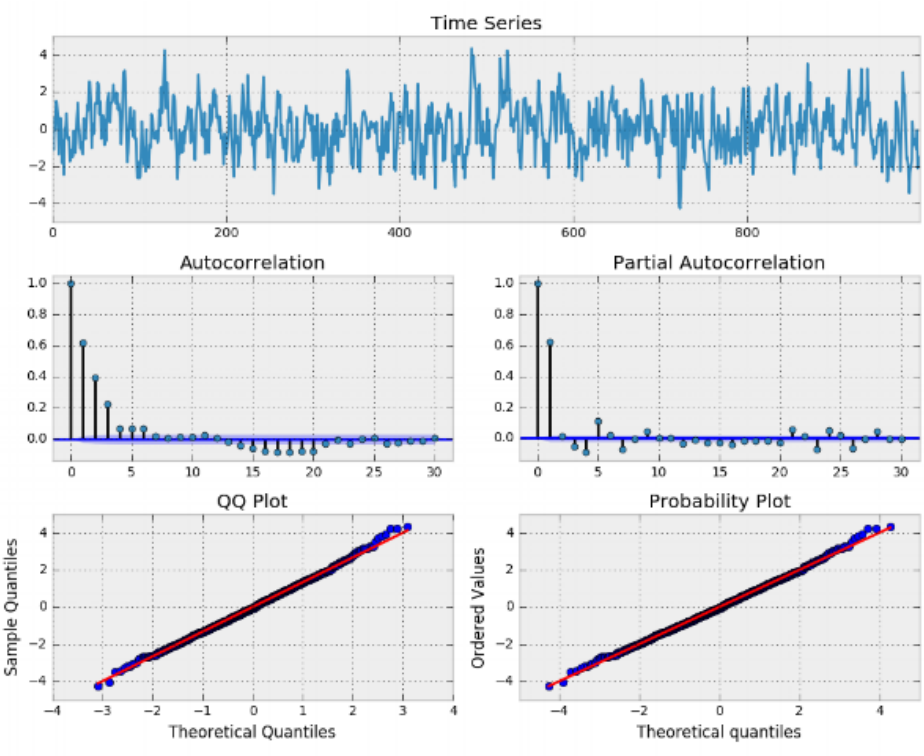
```
# Fit the MA(1) model to our simulated time series
# Specify ARMA model with order (p, q)

max_lag = 30
mdl = smt.ARMA(ma1, order=(0, 1)).fit(
    maxlag=max_lag, method='mle', trend='nc')
p(mdl.summary())
```

```
                              ARMA Model Results
==============================================================================
Dep. Variable:                      y   No. Observations:                 1000
Model:                     ARMA(0, 1)   Log Likelihood               -1418.321
Method:                           mle   S.D. of innovations              0.999
Date:                Thu, 03 Nov 2016   AIC                           2840.642
Time:                        09:24:44   BIC                           2850.458
Sample:                             0   HQIC                          2844.373

==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ma.L1.y        0.5824      0.025     23.208      0.000       0.533       0.632
                                    Roots
=============================================================================
                 Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
MA.1           -1.7171           +0.0000j            1.7171            0.5000
-----------------------------------------------------------------------------
```

MA(1) MODEL SUMMARY

The model was able to correctly estimate the lag coefficent as 0.58 is close to our true value of 0.6. Also notice that our 95% confidence interval does contain the true value. Let's try simulating an MA(3) process, then use our ARMA function to fit a third order MA model to the series and see if we can recover the correct lag coefficients (betas). Betas 1-3 are equal to 0.6, 0.4, and 0.2 respectively.

```
# Simulate MA(3) process with betas 0.6, 0.4, 0.2

n = int(1000)
alphas = np.array([0.])
betas = np.array([0.6, 0.4, 0.2])
ar = np.r_[1, -alphas]
ma = np.r_[1, betas]

ma3 = smt.arma_generate_sample(ar=ar, ma=ma, nsample=n)
_ = tsplot(ma3, lags=30)
```



SIMULATED MA(3) PROCESS WITH BETAS = [0.6, 0.4, 0.2]

```
# Fit MA(3) model to simulated time series

max_lag = 30
mdl = smt.ARMA(ma3, order=(0, 3)).fit(
    maxlag=max_lag, method='mle', trend='nc')
p(mdl.summary())
```

```
                              ARMA Model Results
==============================================================================
Dep. Variable:                      y   No. Observations:                 1000
Model:                     ARMA(0, 3)   Log Likelihood               -1447.741
Method:                           mle   S.D. of innovations              1.029
Date:                Thu, 03 Nov 2016   AIC                           2903.482
Time:                        09:39:42   BIC                           2923.113
Sample:                             0   HQIC                          2910.943

==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ma.L1.y        0.6193      0.030     20.528      0.000       0.560       0.678
ma.L2.y        0.4471      0.034     13.106      0.000       0.380       0.514
ma.L3.y        0.2883      0.030      9.582      0.000       0.229       0.347
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
MA.1            0.0148           -1.4814j            1.4815           -0.2484
MA.2            0.0148           +1.4814j            1.4815            0.2484
MA.3           -1.5805           -0.0000j            1.5805           -0.5000
-----------------------------------------------------------------------------
```

MA(3) MODEL SUMMARY

The model was able to estimate the real coefficients effectively. Our 95% confidence intervals also contain the true parameter values of 0.6, 0.4, and 0.3. Now let's try fitting an MA(3) model to the SPY's log returns. Keep in mind we do not know the *true* parameter values.

```
# Fit MA(3) to SPY returns

max_lag = 30
Y = lrets.SPY
mdl = smt.ARMA(Y, order=(0, 3)).fit(
    maxlag=max_lag, method='mle', trend='nc')
p(mdl.summary())
_ = tsplot(mdl.resid, lags=max_lag)
```

ARMA Model Results

Dep. Variable:	SPY	No. Observations:	2013
Model:	ARMA(0, 3)	Log Likelihood	5756.951
Method:	mle	S.D. of innovations	0.014
Date:	Thu, 03 Nov 2016	AIC	-11505.903
Time:	09:48:41	BIC	-11483.473
Sample:	01-04-2007 - 12-31-2014	HQIC	-11497.670

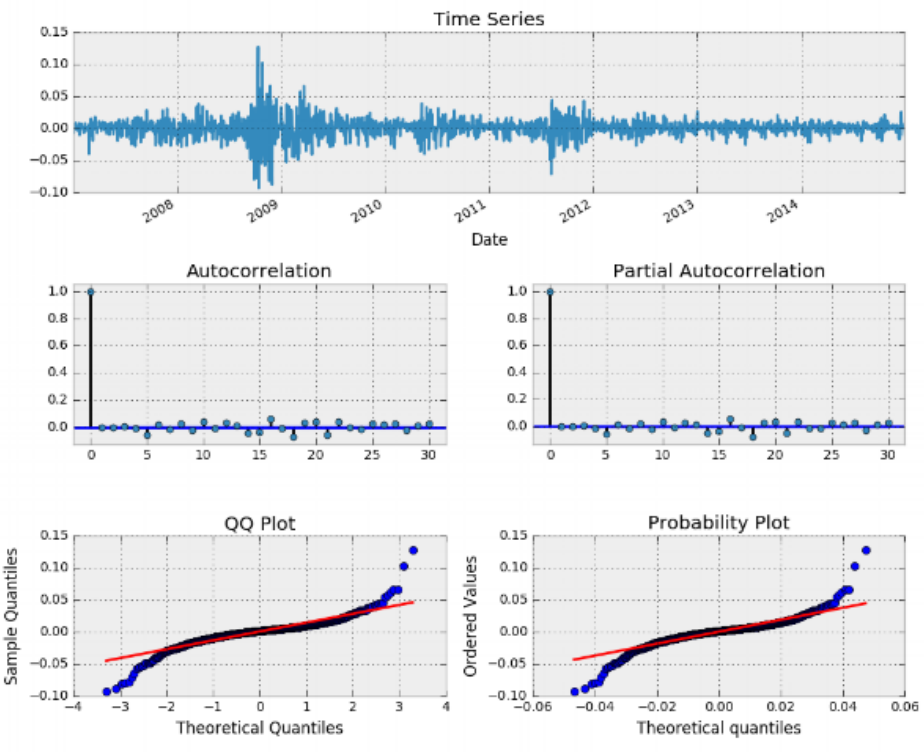
	coef	std err	z	P>|z|	[0.025	0.975]
ma.L1.SPY	-0.0959	0.022	-4.314	0.000	-0.139	-0.052
ma.L2.SPY	-0.0737	0.023	-3.256	0.001	-0.118	-0.029
ma.L3.SPY	0.0274	0.022	1.260	0.208	-0.015	0.070

Roots

	Real	Imaginary	Modulus	Frequency
MA.1	-2.8909	-0.0000j	2.8909	-0.5000
MA.2	2.7906	-2.2012j	3.5542	-0.1063
MA.3	2.7906	+2.2012j	3.5542	0.1063

SPY MA(3) MODEL SUMMARY

Let's look at the model residuals.



SPY MA(3) MODEL RESIDUALS

Not bad. Some of the ACF lags concern me especially at 5, 16, and 18. It could be sampling error but that combined with the heaviness of the tails makes me think this isn't the best model to predict future SPY returns.

Autoregressive Moving Average Models - ARMA(p, q)

As you may have guessed, the ARMA model is simply the merger between AR(p) and MA(q) models. Let's recap what these models represent to us from a quant finance perspective:

- AR(p) models try to capture *(explain)* the momentum and mean reversion effects often observed in trading markets.
- MA(q) models try to capture *(explain)* the shock effects observed in the white noise terms. These shock effects could be thought of as unexpected events affecting the observation process e.g. Surprise earnings, A terrorist attack, etc.

"For a set of products in a grocery store, the number of active coupon campaigns introduced at different times would constitute multiple 'shocks' that affect the prices of the products in question."

- AM207: Pavlos Protopapas, Harvard University

ARMA's weakness is that it ignores the *volatility clustering* effects found in most financial time series.

The model formula is:

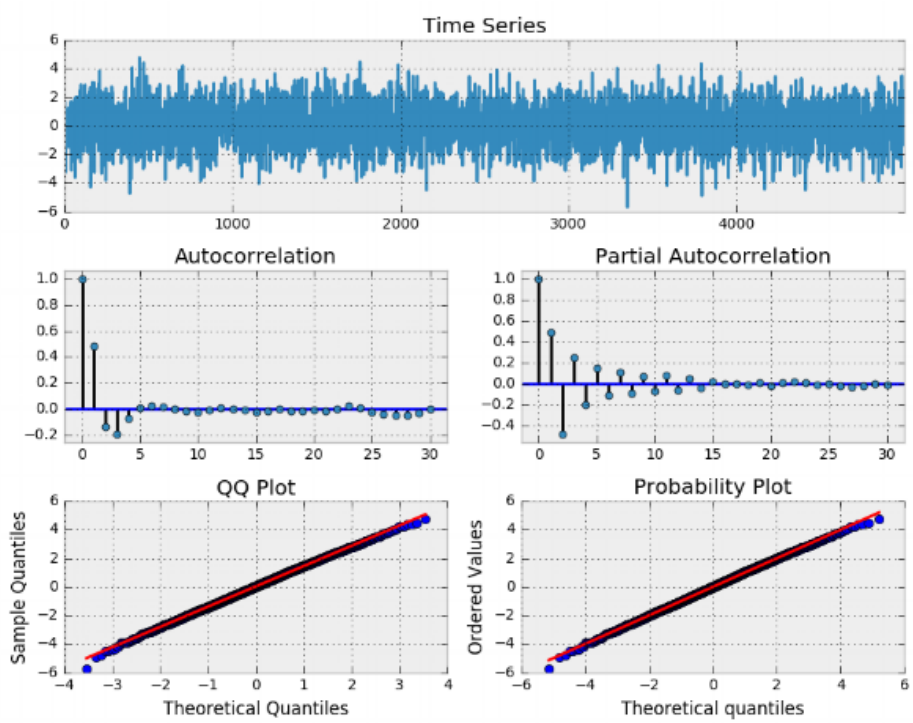
$$x_t = \alpha_1 x_{t-1} + \alpha_2 x_{t-2} + \ldots + \omega_t + \beta_1 \omega_{t-1} + \beta_2 \omega_{t-2} \ldots + \beta_q \omega_{t-q}$$

$$= \sum_{i=1}^{P} \alpha_i x_{t-i} + \omega_t + \sum_{i=1}^{Q} \beta_i \omega_{t-i}$$

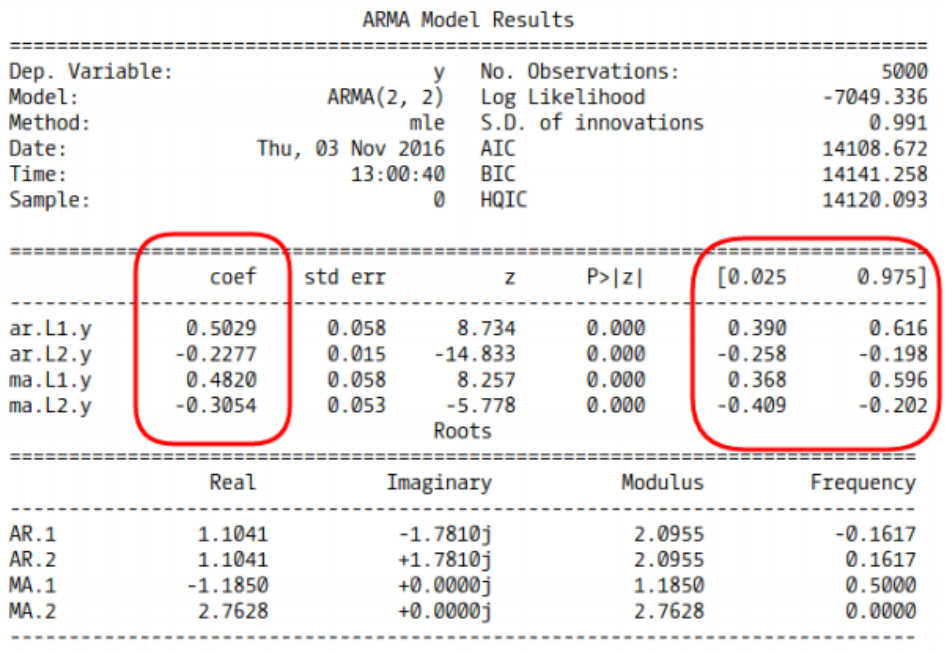
ARMA(P, Q) EQUATION

Let's simulate an ARMA(2, 2) process with given parameters, then fit an ARMA(2, 2) model and see if it can correctly estimate those parameters. Set alphas equal to [0.5,-0.25] and betas equal to [0.5,-0.3].

```
# Simulate an ARMA(2, 2) model with alphas=[0.5,-0.25] and betas=[0.5,-0.3]
max_lag = 30

n = int(5000) # lots of samples to help estimates
burn = int(n/10) # number of samples to discard before fit

alphas = np.array([0.5, -0.25])
betas = np.array([0.5, -0.3])
ar = np.r_[1, -alphas]
ma = np.r_[1, betas]

arma22 = smt.arma_generate_sample(ar=ar, ma=ma, nsample=n, burnin=burn)
_ = tsplot(arma22, lags=max_lag)

mdl = smt.ARMA(arma22, order=(2, 2)).fit(
    maxlag=max_lag, method='mle', trend='nc', burnin=burn)
p(mdl.summary())
```



SIMULATED ARMA(2, 2) PROCESS



```
                            ARMA Model Results
==============================================================================
Dep. Variable:                      y   No. Observations:                 5000
Model:                     ARMA(2, 2)   Log Likelihood               -7049.336
Method:                           mle   S.D. of innovations              0.991
Date:                Thu, 03 Nov 2016   AIC                          14108.672
Time:                        13:00:40   BIC                          14141.258
Sample:                             0   HQIC                         14120.093
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1.y        0.5029      0.058      8.734      0.000       0.390       0.616
ar.L2.y       -0.2277      0.015    -14.833      0.000      -0.258      -0.198
ma.L1.y        0.4820      0.058      8.257      0.000       0.368       0.596
ma.L2.y       -0.3054      0.053     -5.778      0.000      -0.409      -0.202
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.1041           -1.7810j            2.0955           -0.1617
AR.2            1.1041           +1.7810j            2.0955            0.1617
MA.1           -1.1850           +0.0000j            1.1850            0.5000
MA.2            2.7628           +0.0000j            2.7628            0.0000
-----------------------------------------------------------------------------
```

ARMA(2, 2) MODEL SUMMARY

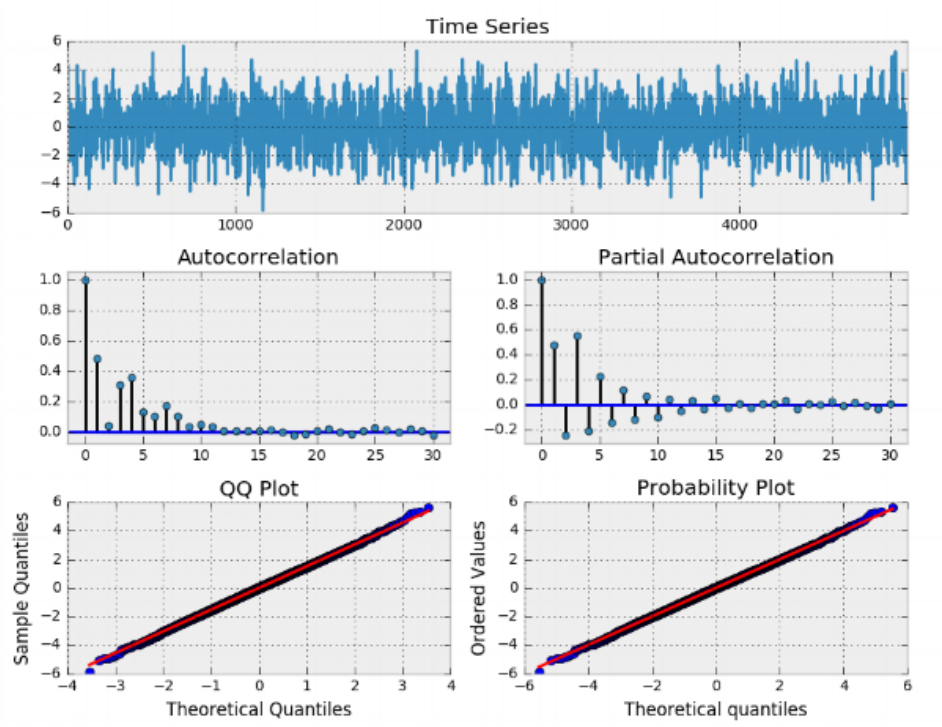
The model has correctly recovered our parameters, and our true parameters are contained within the 95% confidence interval.

Next we simulate a ARMA(3, 2) model. After, we cycle through a non trivial number of combinations of p, q to fit an ARMA model to our simulated series. We choose the best combination based on which model produces the lowest Akaike Information Criterion (AIC).

```
# Simulate an ARMA(3, 2) model with alphas=[0.5,-0.25,0.4] and betas=[0.5,-0.3]

max_lag = 30

n = int(5000)
burn = 2000

alphas = np.array([0.5, -0.25, 0.4])
betas = np.array([0.5, -0.3])

ar = np.r_[1, -alphas]
ma = np.r_[1, betas]

arma32 = smt.arma_generate_sample(ar=ar, ma=ma, nsample=n, burnin=burn)
_ = tsplot(arma32, lags=max_lag)

# pick best order by aic
# smallest aic value wins
best_aic = np.inf
best_order = None
best_mdl = None

rng = range(5)
for i in rng:
    for j in rng:
        try:
            tmp_mdl = smt.ARMA(arma32, order=(i, j)).fit(method='mle', trend='nc')
```

```
            tmp_aic = tmp_mdl.aic
            if tmp_aic < best_aic:
                best_aic = tmp_aic
                best_order = (i, j)
                best_mdl = tmp_mdl
        except: continue


p('aic: {:6.5f} | order: {}'.format(best_aic, best_order))

# aic: 14108.27213 | order: (3, 2)
```

The correct order was recovered above. Below we see the output of our simulated time series before any model fitting.



SIMULATED ARMA(3, 2) SERIES WITH ALPHAS = [0.5,-0.25,0.4] AND BETAS = [0.5,-0.3]

ARMA Model Results

Dep. Variable:	y	No. Observations:	5000
Model:	ARMA(3, 2)	Log Likelihood	-7048.136
Method:	mle	S.D. of innovations	0.990
Date:	Thu, 03 Nov 2016	AIC	14108.272
Time:	14:15:41	BIC	14147.375
Sample:	0	HQIC	14121.977

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1.y	0.5549	0.028	19.502	0.000	0.499	0.611
ar.L2.y	-0.2790	0.016	-16.977	0.000	-0.311	-0.247
ar.L3.y	0.4166	0.014	30.250	0.000	0.390	0.444
ma.L1.y	0.4399	0.030	14.452	0.000	0.380	0.500
ma.L2.y	-0.3420	0.029	-11.749	0.000	-0.399	-0.285

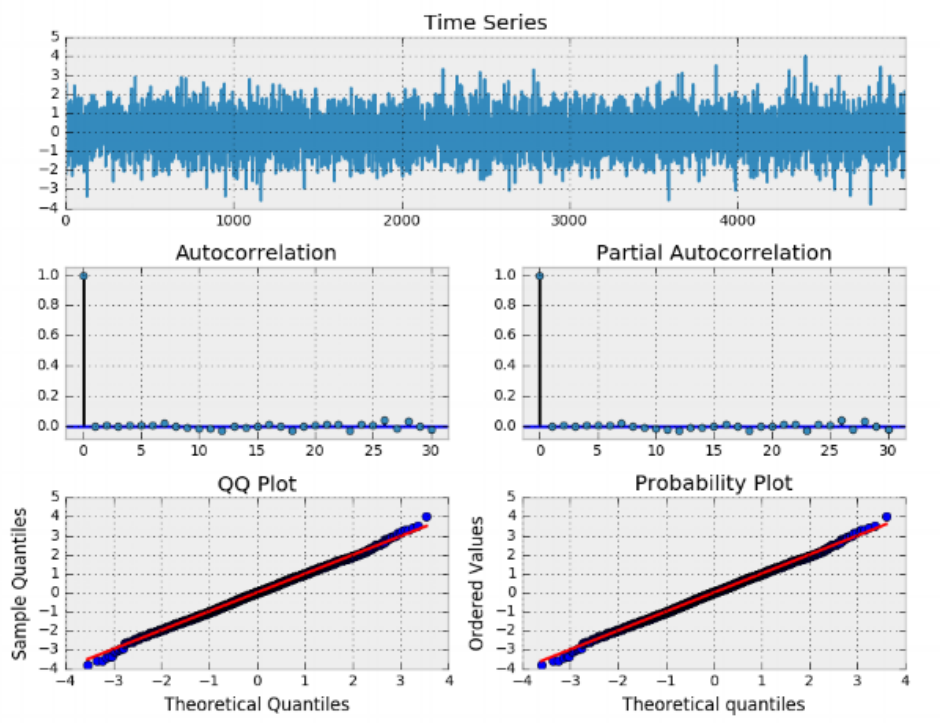
Roots

	Real	Imaginary	Modulus	Frequency
AR.1	1.2094	-0.0000j	1.2094	-0.0000
AR.2	-0.2699	-1.3827j	1.4088	-0.2807
AR.3	-0.2699	+1.3827j	1.4088	0.2807
MA.1	-1.1838	+0.0000j	1.1838	0.5000
MA.2	2.4698	+0.0000j	2.4698	0.0000

ARMA(3, 2) BEST MODEL SUMMARY

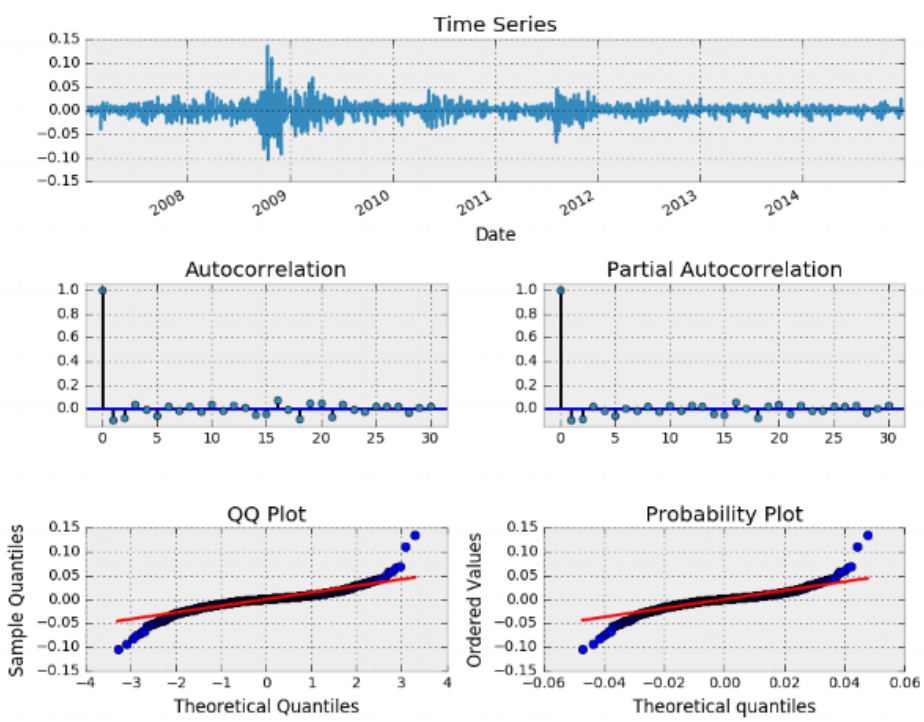
We see that the correct order was selected and the model correctly estimated our parameters. However notice the MA.L1.y coefficent; the true value of 0.5 is almost outside of the 95% confidence interval!

Below we observe the model's residuals. Clearly it is a white noise process, thus the best model has been fit to *explain* the data.



ARMA(3, 2) BEST MODEL RESIDUAL WHITE NOISE

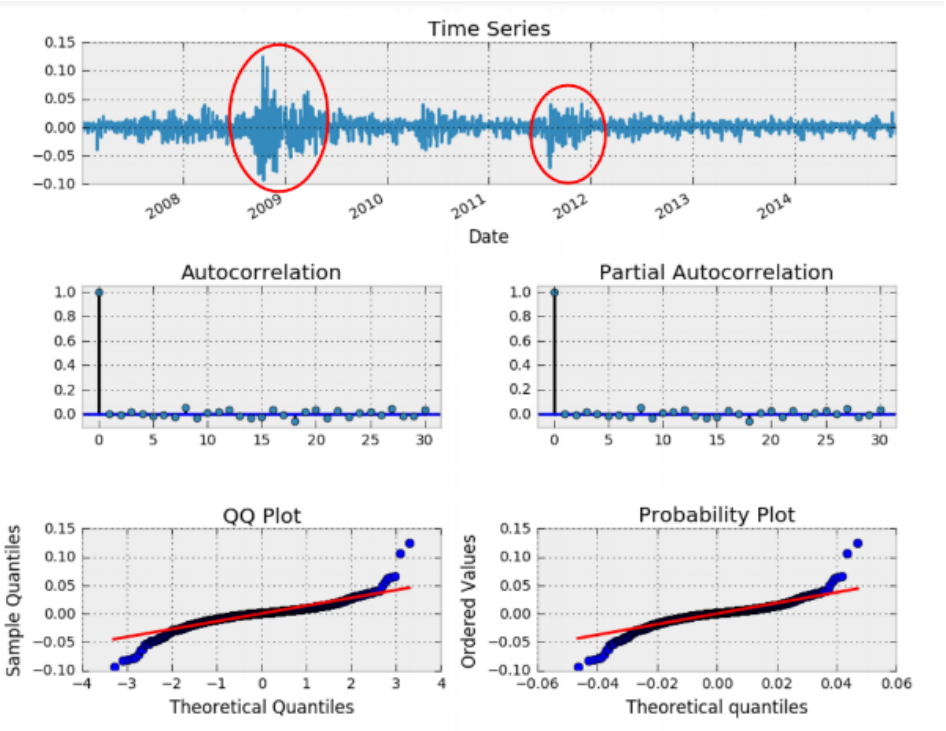
Next we fit an ARMA model to SPY returns. The plot below is the time series before model fitting.



SPY RETURNS

```
# Fit ARMA model to SPY returns

best_aic = np.inf
best_order = None
best_mdl = None

rng = range(5) # [0,1,2,3,4,5]
for i in rng:
    for j in rng:
        try:
            tmp_mdl = smt.ARMA(lrets['SPY'], order=(i, j)).fit(
                method='mle', trend='nc'
            )
            tmp_aic = tmp_mdl.aic
            if tmp_aic < best_aic:
                best_aic = tmp_aic
                best_order = (i, j)
                best_mdl = tmp_mdl
        except: continue


p('aic: {:6.5f} | order: {}'.format(best_aic, best_order))

# aic: -11518.22902 | order: (4, 4)
```

We plot the model residuals.



SPY BEST MODEL RESIDUALS ARMA(4, 4)

The ACF and PACF are showing no significant autocorrelation. The QQ and Probability Plots show the residuals are approximately normal with heavy tails. However, this model's residuals do NOT look like white noise! Look at the highlighted areas of obvious conditional heteroskedasticity (*conditional volatility*) that the model has not captured.

Autoregressive Integrated Moving Average Models - ARIMA(p, d, q)

ARIMA is a natural extension to the class of ARMA models. As previously mentioned many of our TS are not stationary, however they can be made stationary by differencing. We saw an example of this when we took the first difference of a Guassian random walk and proved that it equals white noise. Said another way, we took the nonstationary random walk and transformed it to stationary white noise by first-differencing.

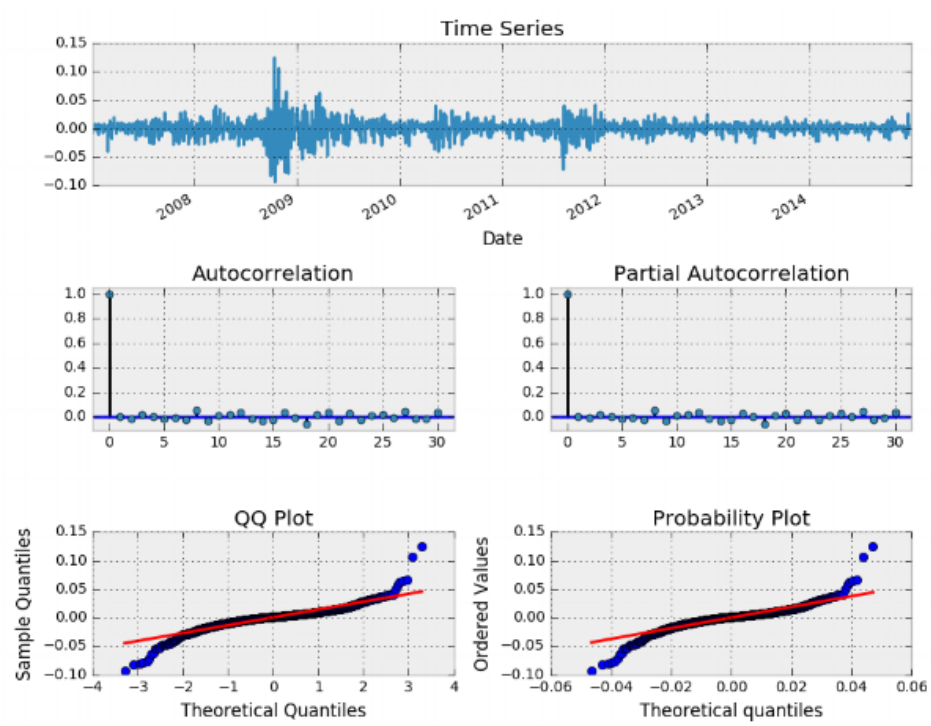
Without diving too deeply into the equation, just know the **"d"** references the number of times we are differencing the series. A side note, in Python we must use **np.diff()** function if we need to difference a series more than once. The **pandas** functions **DataFrame.diff()/Series.diff() only takes the first difference of a dataframe/series and does not implement the recursive differencing needed in TSA.**

In the following example, we iterate through a non-trivial number of combinations of (p, d, q) orders, to find the best ARIMA model to fit SPY returns. We use the AIC to evaluate each model. The lowest AIC wins.

```
# Fit ARIMA(p, d, q) model to SPY Returns
# pick best order and final model based on aic
```

```
best_aic = np.inf
best_order = None
best_mdl = None

pq_rng = range(5) # [0,1,2,3,4]
d_rng = range(2) # [0,1]
for i in pq_rng:
    for d in d_rng:
        for j in pq_rng:
            try:
                tmp_mdl = smt.ARIMA(lrets.SPY, order=(i,d,j)).fit(method='mle', trend='nc')
                tmp_aic = tmp_mdl.aic
                if tmp_aic < best_aic:
                    best_aic = tmp_aic
                    best_order = (i, d, j)
                    best_mdl = tmp_mdl
            except: continue


p('aic: {:6.5f} | order: {}'.format(best_aic, best_order))
# aic: -11518.22902 | order: (4, 0, 4)

# ARIMA model resid plot
_ = tsplot(best_mdl.resid, lags=30)
```

It should be no surprise that the best model has a differencing of 0. Recall that we already took the first difference of log prices to calculate the stock returns. Below, I plot the model residuals. The result is essentially identical to the ARMA(4, 4) model we fit above. Clearly this ARIMA model has not explained the conditional volatility in the series either!



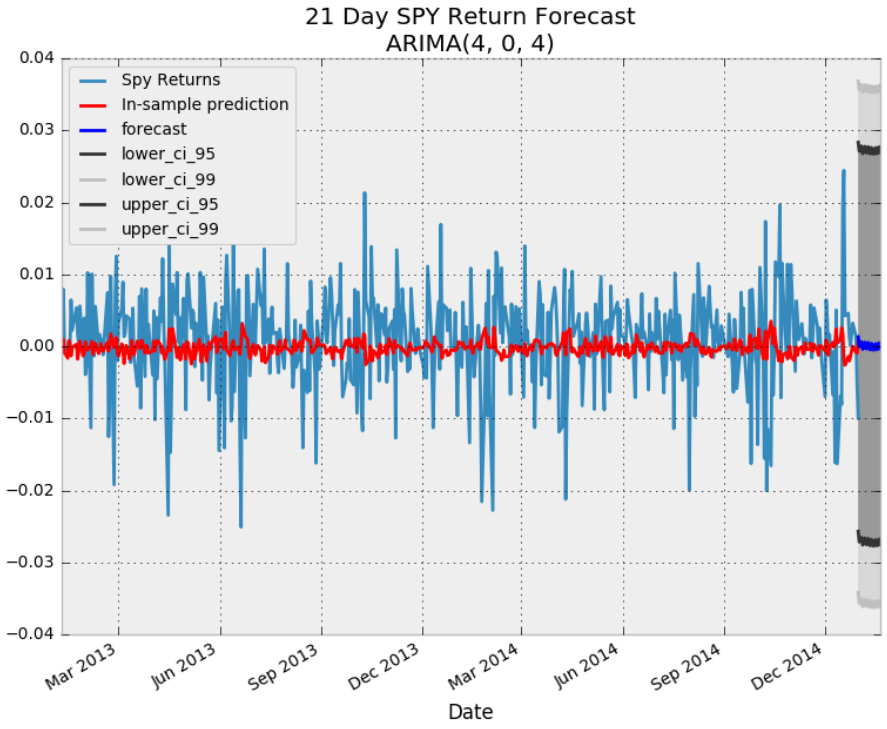
ARIMA MODEL FIT TO SPY RETURNS

Now we have at least accumulated enough knowledge to make a simple forecast of future returns. Here we make use of our model's **forecast()** method. As arguments, it takes an integer for the number of time steps to predict, and a decimal for the alpha argument to specify the confidence intervals. The default setting is 95% confidence. For 99% set alpha equal to 0.01.

```
# Create a 21 day forecast of SPY returns with 95%, 99% CI
n_steps = 21

f, err95, ci95 = best_mdl.forecast(steps=n_steps) # 95% CI
_, err99, ci99 = best_mdl.forecast(steps=n_steps, alpha=0.01) # 99% CI

idx = pd.date_range(data.index[-1], periods=n_steps, freq='D')
fc_95 = pd.DataFrame(np.column_stack([f, ci95]),
                     index=idx, columns=['forecast', 'lower_ci_95', 'upper_ci_95'])
fc_99 = pd.DataFrame(np.column_stack([ci99]),
                     index=idx, columns=['lower_ci_99', 'upper_ci_99'])
fc_all = fc_95.combine_first(fc_99)
fc_all.head()
```

	forecast	lower_ci_95	lower_ci_99	upper_ci_95	upper_ci_99
2014-12-31	0.001318	-0.025693	-0.034180	0.028329	0.036816
2015-01-01	-0.000036	-0.027166	-0.035691	0.027094	0.035619
2015-01-02	0.000426	-0.026748	-0.035287	0.027600	0.036139
2015-01-03	0.000642	-0.026532	-0.035071	0.027816	0.036355
2015-01-04	-0.000408	-0.027584	-0.036123	0.026769	0.035308

```
# Plot 21 day forecast for SPY returns

plt.style.use('bmh')
fig = plt.figure(figsize=(9,7))
ax = plt.gca()

ts = lrets.SPY.iloc[-500:].copy()
ts.plot(ax=ax, label='Spy Returns')
# in sample prediction
pred = best_mdl.predict(ts.index[0], ts.index[-1])
pred.plot(ax=ax, style='r-', label='In-sample prediction')

styles = ['b-', '0.2', '0.75', '0.2', '0.75']
fc_all.plot(ax=ax, style=styles)
plt.fill_between(fc_all.index, fc_all.lower_ci_95, fc_all.upper_ci_95, color='gray', alpha=0.7)
plt.fill_between(fc_all.index, fc_all.lower_ci_99, fc_all.upper_ci_99, color='gray', alpha=0.2)
plt.title('{} Day SPY Return Forecast\nARIMA{}'.format(n_steps, best_order))
plt.legend(loc='best', fontsize=10)
```



21-DAY SPY RETURN FORECAST - ARIMA(4,0,4)

Autoregressive Conditionally Heteroskedastic Models - ARCH(p)

ARCH(p) models can be thought of as simply an AR(p) model applied to the variance of a time series. Another way to think about it, is that the variance of our time series NOW *at time t*, is conditional on past observations of the variance in previous periods.

$$Var(y_t|y_{t-1}) = \sigma_t^2 = \alpha_0 + \alpha_1 y_{t-1}^2$$

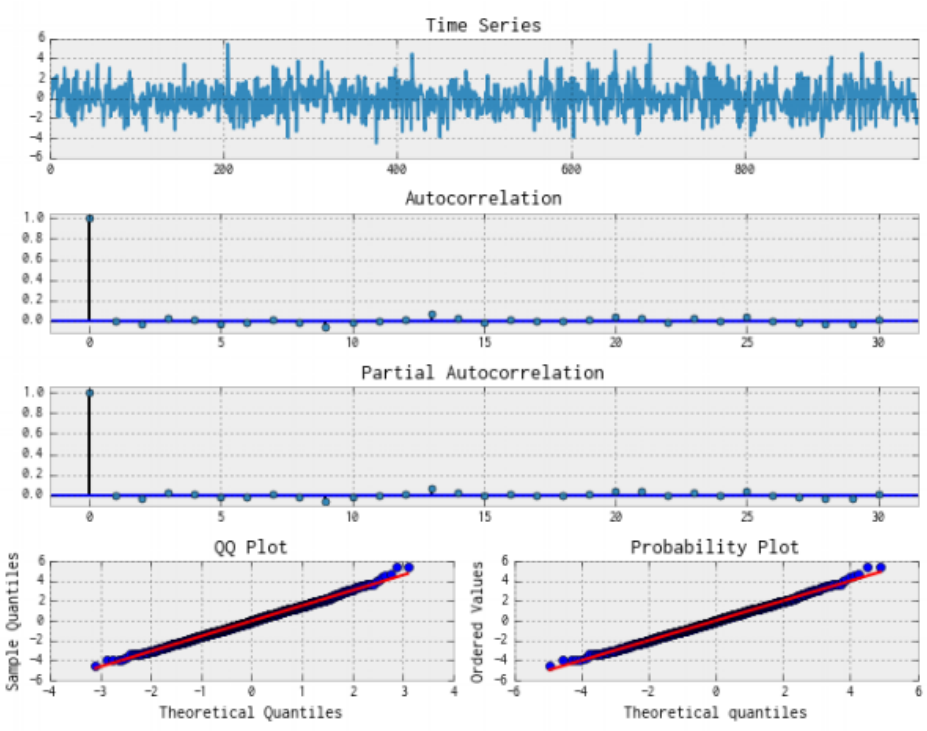
ARCH(1) MODEL FORMULA - PENN STATE

Assuming the series has zero mean we can express the model as:

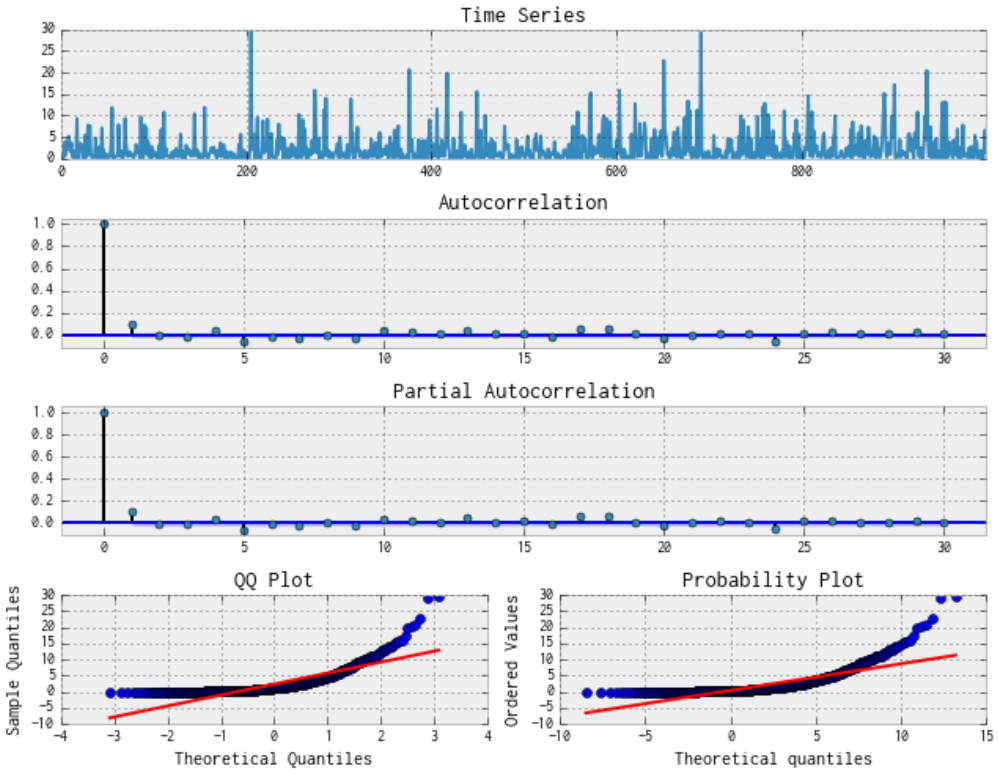
$$y_t = \sigma_t \epsilon_t, \text{ with } \sigma_t = \sqrt{\alpha_0 + \alpha_1 y_{t-1}^2}, \text{ and } \epsilon_t \sim iid(0,1)$$

ARCH(1) MODEL IF ZERO MEAN

```
# Simulate ARCH(1) series
# Var(yt) = a_0 + a_1*y{t-1}**2
# if a_1 is between 0 and 1 then yt is white noise

np.random.seed(13)

a0 = 2
a1 = .5

y = w = np.random.normal(size=1000)
Y = np.empty_like(y)

for t in range(len(y)):
    Y[t] = w[t] * np.sqrt((a0 + a1*y[t-1]**2))

# simulated ARCH(1) series, looks like white noise
tsplot(Y, lags=30)
```



SIMULATED ARCH(1) PROCESS



SIMULATED ARCH(1)**2 PROCESS

Notice the ACF, and PACF seem to show significance at lag 1 indicating an AR(1) model for the variance may be appropriate.

Generalized Autoregressive Conditionally Heteroskedastic Models - GARCH(p,q)

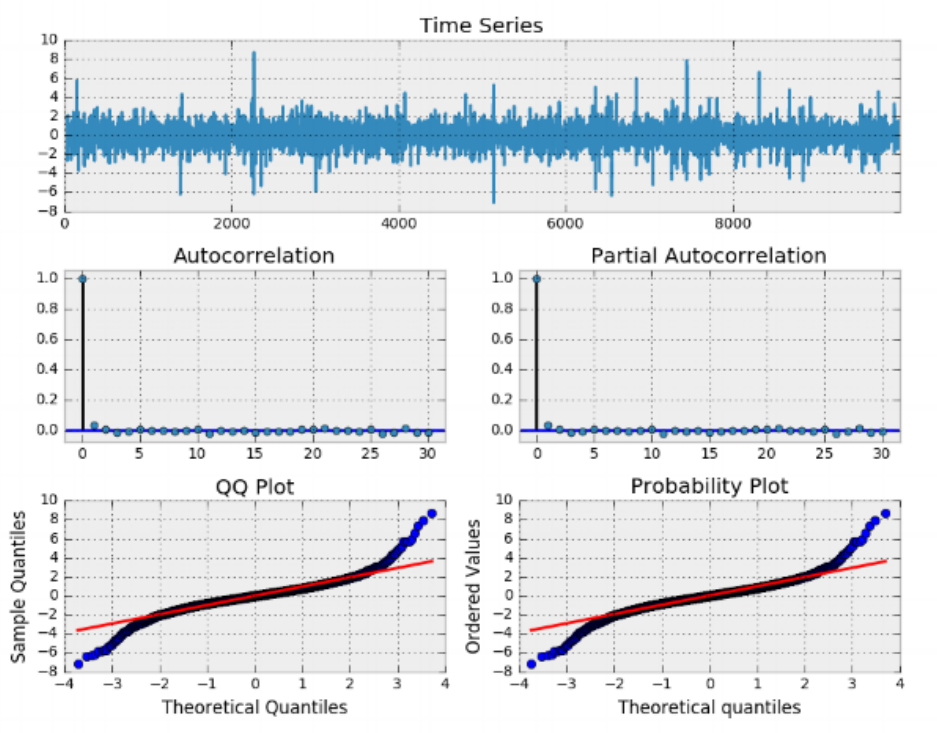
Simply put GARCH(p, q) is an ARMA model applied to the variance of a time series i.e., it has an autoregressive term and a moving average term. The AR(p) models the variance of the residuals (squared errors) or simply our time series squared. The MA(q) portion models the variance of the process. The basic GARCH(1, 1) formula is:

$$\epsilon_t = \sigma_t w_t$$
$$\sigma_t^2 = \alpha_0 + \alpha_1 \epsilon_{t-1}^2 + \beta_1 \sigma_{t-1}^2$$

GARCH(1, 1) FORMULA FROM QUANTSTART.COM

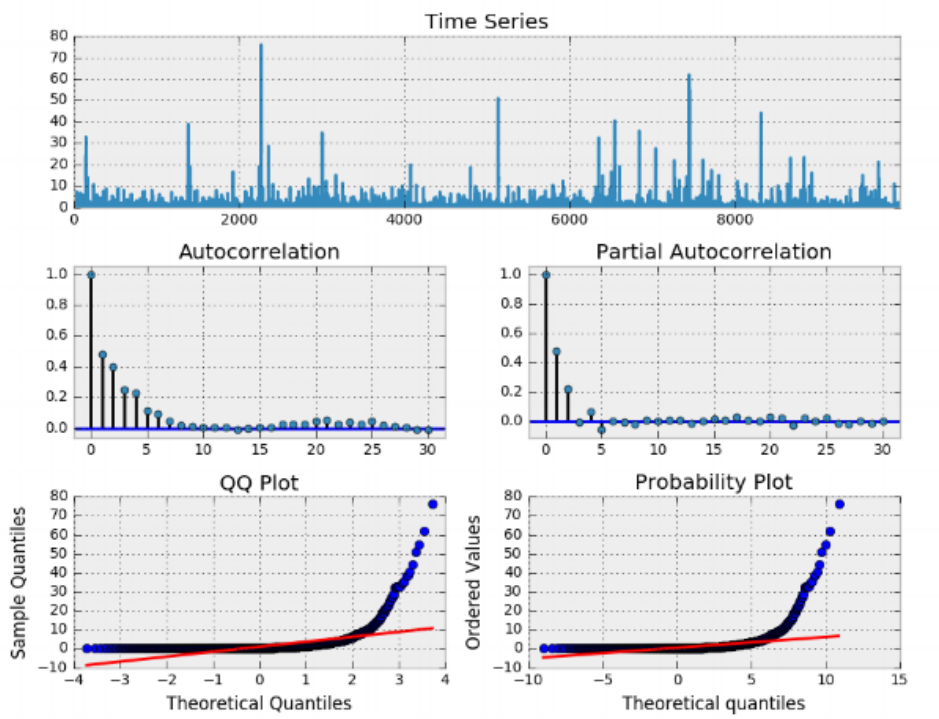
Omega (w) is white noise, and alpha and beta are parameters of the model. Also alpha_1 + beta_1 must be less than 1 or the model is unstable. We can simulate a GARCH(1, 1) process below.

```
# Simulating a GARCH(1, 1) process

np.random.seed(2)

a0 = 0.2
a1 = 0.5
b1 = 0.3

n = 10000
w = np.random.normal(size=n)
eps = np.zeros_like(w)
sigsq = np.zeros_like(w)

for i in range(1, n):
    sigsq[i] = a0 + a1*(eps[i-1]**2) + b1*sigsq[i-1]
    eps[i] = w[i] * np.sqrt(sigsq[i])

_ = tsplot(eps, lags=30)
```



SIMULATED GARCH(1, 1) PROCESS

Again, notice that overall this process closely resembles white noise, however take a look when we view the squared **eps** series.



SIMULATED GARCH(1, 1) PROCESS SQUARED

There is clearly autocorrelation present and the significance of the lags in both the ACF and PACF indicate we need both AR and MA components for our model. Let's see if we can recover our process parameters using a GARCH(1, 1) model. Here we make use of the **arch_model** function from the **ARCH** package.

```
# Fit a GARCH(1, 1) model to our simulated EPS series
# We use the arch_model function from the ARCH package

am = arch_model(eps)
res = am.fit(update_freq=5)
p(res.summary())
```

```
Iteration:      5,   Func. Count:     38,   Neg. LLF: 12311.793822714439
Iteration:     10,   Func. Count:     71,   Neg. LLF: 12238.59267719981
Optimization terminated successfully.    (Exit mode 0)
            Current function value: 12237.3032673
            Iterations: 13
            Function evaluations: 89
            Gradient evaluations: 13
                    Constant Mean - GARCH Model Results
==============================================================================
Dep. Variable:                      y   R-squared:                      -0.000
Mean Model:             Constant Mean   Adj. R-squared:                 -0.000
Vol Model:                      GARCH   Log-Likelihood:               -12237.3
Distribution:                  Normal   AIC:                           24482.6
Method:            Maximum Likelihood   BIC:                           24511.4
                                        No. Observations:                10000
Date:                Thu, Nov 03 2016   Df Residuals:                    9996
Time:                        22:18:30   Df Model:                           4
                                Mean Model
==============================================================================
                 coef    std err          t      P>|t|       95.0% Conf. Int.
------------------------------------------------------------------------------
mu         -6.7225e-03  6.735e-03     -0.998      0.318 [-1.992e-02,6.478e-03]
                            Volatility Model
============================================================================
                 coef    std err          t      P>|t|      95.0% Conf. Int.
----------------------------------------------------------------------------
omega          0.2021  1.043e-02     19.383  1.084e-83 [  0.182,  0.223]
alpha[1]       0.5162  2.016e-02     25.611 1.144e-144 [  0.477,  0.556]
beta[1]        0.2879  1.870e-02     15.395  1.781e-53 [  0.251,  0.325]
============================================================================

Covariance estimator: robust
```

GARCH MODEL FIT SUMMARY

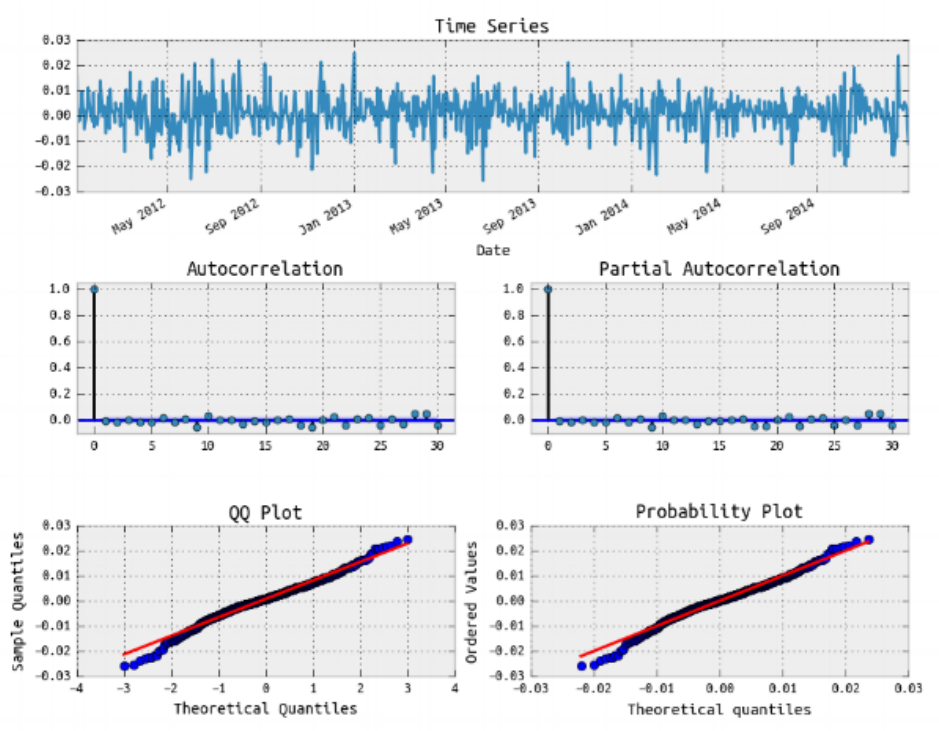
Now let's run through an example using SPY returns. The process is as follows:

- Iterate through combinations of ARIMA(p, d, q) models to best fit our time series.
- Pick the GARCH model orders according to the ARIMA model with lowest AIC.
- Fit the GARCH(p, q) model to our time series.
- Examine the model residuals and squared residuals for autocorrelation

Also note that I've chosen a specific time period to better highlight key points. However the results will be different depending on the time period under study.

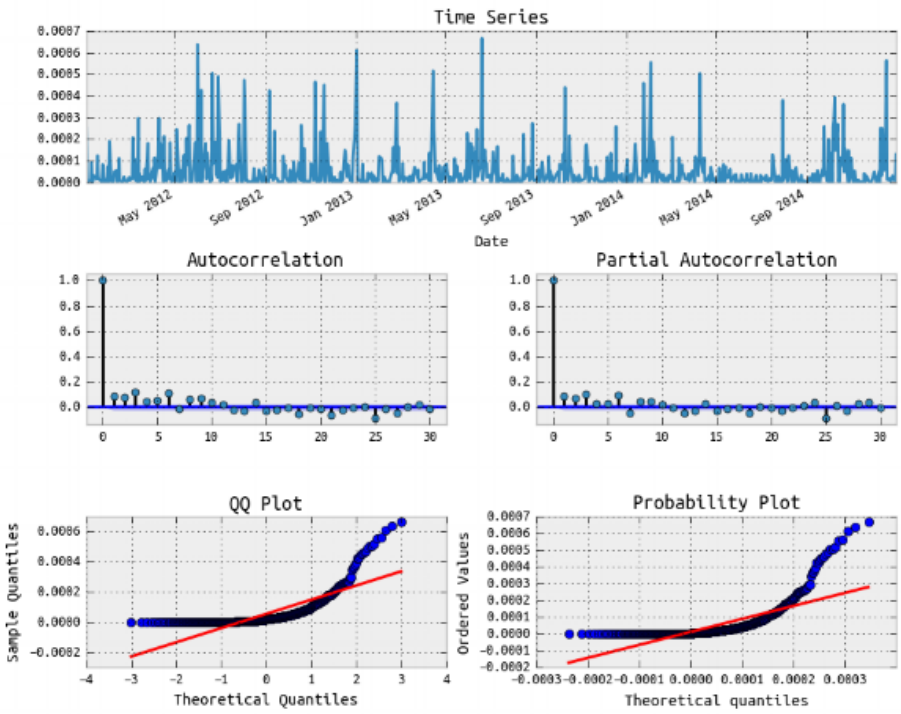
```
def _get_best_model(TS):
    best_aic = np.inf
    best_order = None
    best_mdl = None

    pq_rng = range(5) # [0,1,2,3,4]
    d_rng = range(2) # [0,1]
    for i in pq_rng:
        for d in d_rng:
            for j in pq_rng:
                try:
                    tmp_mdl = smt.ARIMA(TS, order=(i,d,j)).fit(
                        method='mle', trend='nc'
                    )
                    tmp_aic = tmp_mdl.aic
                    if tmp_aic < best_aic:
                        best_aic = tmp_aic
```

```
                    best_order = (i, d, j)
                    best_mdl = tmp_mdl
                except: continue
    p('aic: {:6.5f} | order: {}'.format(best_aic, best_order))
    return best_aic, best_order, best_mdl

# Notice I've selected a specific time period to run this analysis
TS = lrets.SPY.ix['2012':'2015']
res_tup = _get_best_model(TS)

# aic: -5255.56673 | order: (3, 0, 2)
```



RESIDUALS OF ARIMA(3,0,2) MODEL FIT TO SPY RETURNS

Looks like white noise.



SQUARED RESIDUALS OF ARIMA(3,0,2) MODEL FIT TO SPY RETURNS

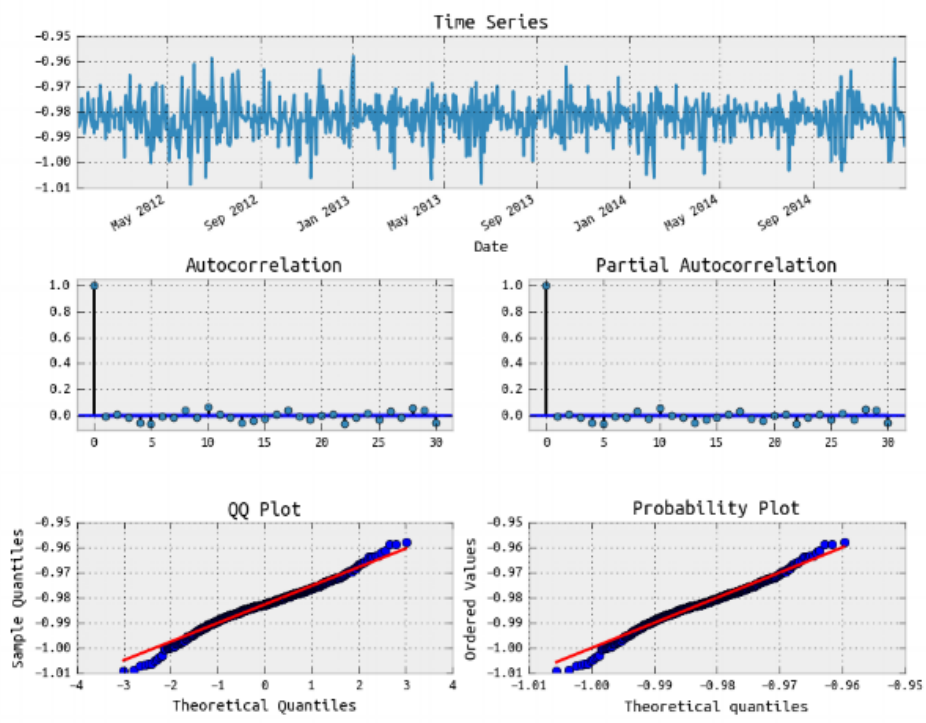
Squared residuals show autocorrelation. Let's fit a GARCH model and see how it does.

```
# Now we can fit the arch model using the best fit arima model parameters

p_ = best_order[0]
o_ = best_order[1]
q_ = best_order[2]

# Using student T distribution usually provides better fit
am = arch_model(TS, p=p_, o=o_, q=q_, dist='StudentsT')
res = am.fit(update_freq=5, disp='off')
p(res.summary())
```

```
                     Constant Mean - GARCH Model Results
==============================================================================
Dep. Variable:                     SPY   R-squared:                -17801.354
Mean Model:                Constant Mean   Adj. R-squared:           -17801.354
Vol Model:                         GARCH   Log-Likelihood:             -1137.75
Distribution:      Standardized Student's t   AIC:                       2291.50
Method:                  Maximum Likelihood   BIC:                       2328.50
                                              No. Observations:              754
Date:                      Tue, Nov 08 2016   Df Residuals:                  746
Time:                             12:13:55   Df Model:                        8
                                 Mean Model
============================================================================
                 coef    std err          t      P>|t|      95.0% Conf. Int.
----------------------------------------------------------------------------
mu             0.9834  6.408e-03    153.464      0.000 [  0.971,  0.996]
                              Volatility Model
============================================================================
                 coef    std err          t      P>|t|       95.0% Conf. Int.
----------------------------------------------------------------------------
omega      1.8447e-05  1.562e-05      1.181      0.238 [-1.216e-05,4.906e-05]
alpha[1]       0.5357      0.435      1.231      0.218      [ -0.317,  1.389]
alpha[2]       0.0843      1.816  4.643e-02      0.963      [ -3.475,  3.643]
alpha[3]       0.3627      1.132      0.320      0.749      [ -1.856,  2.581]
beta[1]    9.3192e-03      2.546  3.661e-03      0.997      [ -4.980,  4.999]
beta[2]    9.3202e-03      2.204  4.229e-03      0.997      [ -4.310,  4.329]
                              Distribution
============================================================================
                 coef    std err          t      P>|t|  95.0% Conf. Int.
----------------------------------------------------------------------------
nu            15.4669     10.544      1.467      0.142 [ -5.199, 36.132]
============================================================================

Covariance estimator: robust

WARNING: The optimizer did not indicate sucessful convergence. The message was
Positive directional derivative for linesearch. See convergence_flag.
```

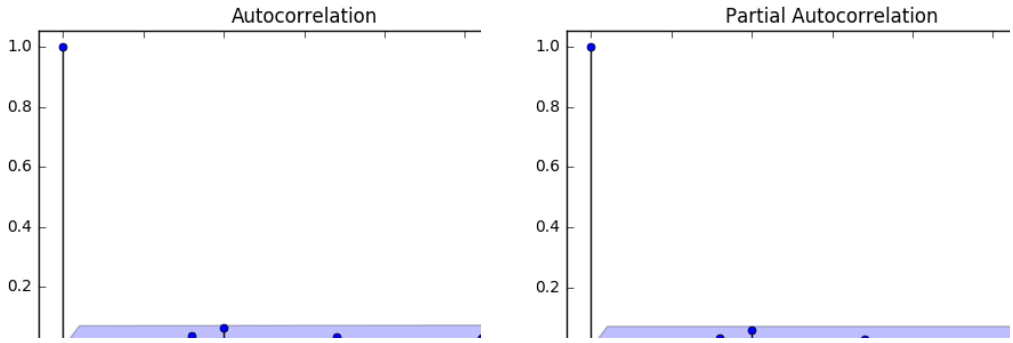
GARCH(3, 2) MODEL FIT TO SPY RETURNS

Convergence warnings can occur when dealing with very small numbers. Multiplying the numbers by factors of 10 to scale the magnitude can help when necessary, however for this demonstration it isn't necessary. Below are the model residuals.



RESIDUALS OF GARCH(3, 2) MODEL FIT TO SPY RETURNS

Looks like white noise above. Now let's view the ACF and PACF of the squared residuals.



Looks like we have achieved a good model fit as there is no obvious autocorrelation in the squared residuals.

References

- Quantstart.com - https://www.quantstart.com/articles#time-series-analysis
- Harvard Lectures in Python - http://iacs-courses.seas.harvard.edu/courses/am207/blog/lecture-17.html
- Penn State Stats - https://onlinecourses.science.psu.edu/stat510/node/78
- stationary pic + tsplot - http://www.seanabu.com/2016/03/22/time-series-seasonal-ARIMA-model-in-python/
- stationary quote, etc - http://people.duke.edu/~rnau/411diff.htm
- interpreting qq plots - http://stats.stackexchange.com/questions/101274/how-to-interpret-a-qq-plot
- Kaplan SchweserNotes (Level 2) - Quantitative Methods

Preview Post Comment…

luigi 3 years ago · 0 Likes

Hi,
very nice post.
However, when you use the function tsplot the confidence interval of the acf and pacf are not appearing. I tried using you function as well but also in my case they do not come out, can you please suggest me how to make them? if I remove the line smt.graphics.plot_acf(y, lags=lags, ax=acf_ax, alpha=0.5) out of the function they are shown but inside your function they are not

Thanks
Luigi

Måns 4 years ago · 0 Likes

Thanks for an awesome blog post! I have a question; suppose I would want to plot the predicted values in the original scale. How do I transform back from np.log(data/data.shift(1)) you make in the beginning? Thanks!

Brian Christopher 4 years ago · 0 Likes

To get simple returns from log returns use: np.exp(data) - 1

4 years ago · 0 Likes

Hello All,

Can I ask a question in White Noise and also appear in other sectors. In White Noise part, there is a sentence "By definition a time series that is a white noise process has serially UNcorrelated errors and the expected mean of those errors is equal to zero. ". My question is for this case, there is no model made, it is just a time series, so where is the errors coming from? Thank you

Eric 4 years ago · 0 Likes

The following code seems wrong to me to simulate the ARCH(1):
"
y = w = np.random.normal(size=1000)
Y = np.empty_like(y)
for t in range(len(y)):
Y[t] = w[t] * np.sqrt((a0 + a1*y[t-1]**2))
"

Brian Christopher 4 years ago · 0 Likes

Hello Eric,

You'll have to be more specific. It looks exactly like the formula I referenced above. Let me know what your alternative simulation is.

4 years ago · 0 Likes

Hi

I believe the last chunk of code does not work as displayed, with
p_ = order[0]
o_ = order[1]
q_ = order[2]

throwing an error, name 'order' is not defined

correct to:
p_ = best_order[0]
o_ = best_order[1]
q_ = best_order[2]

did work

Brian Christopher 4 years ago · 0 Likes

Thanks Gary. Updated.

nathan 4 years ago · 0 Likes

Hello, is there a way to carry out the rabdom sampling process on the VARMA also?
Just like you have done on the ARMA/ARIMA

KC 4 years ago · 0 Likes

according to the ARCH module API doc, the p and q are reversal of the statsmodels ARIMA's.

from https://arch.readthedocs.io/en/latest/univariate/volatility.html#garch
p (int) – Order of the symmetric innovation
o (int) – Order of the asymmetric innovation
q (int) – Order of the lagged (transformed) conditional variance

---

Pier-Olivier 4 years ago · 0 Likes

What should you do if the ARIMA parameters are 0, 0, 1? GARCH requires one of P or O to be strictly positive. Does that mean a good fit model cannot be attained?

Victor 4 years ago · 0 Likes

I guess you should perform GARCH on the residuals of ARIMA???

June 4 years ago · 0 Likes

Thanks for such a good post.
My question is why arch use the same order as arma, as you comments with code # Now we can fit the arch model using the best fit arima model parameters?

Milos 4 years ago · 0 Likes

I believe you actually should have performed ARIMA procedure on a squared Time Series not the regular one. It is known that GARCH can be represented as an ARMA on squared time series.
Cheers,
Milos

Helmut Simon 5 years ago · 0 Likes

In your ARM(3) example you say that "Our 95% confidence intervals also contain the true parameter values of 0.6, 0.4, and 0.3." But in the code above the parameters given are [0.6, 0.4, 0.2]..

Helmut Simon 5 years ago · 0 Likes

Thanks for a very interesting post. I notice that generally in the summaries your log-likelihood is negative and the AIC positive, but in at least one case (fitting an MA(3) model to the SPY's log returns) it's the other way around. What is the significance of this?

Helmut Simon 5 years ago · 0 Likes

More research has helped me understand that the AIC may be positive or negative and this is not significant. The lowest signed value is the best, not the closest to zero.

5 years ago · 0 Likes

the GARCH(1,1) simulation looks so simple it is tempting to just dive in and make it directly .. but, I'm noticing that I can't identify where p and q are in your version. I want to see the same thing now, but with p and q explicit in the implementation!

Jun 5 years ago · 0 Likes

Dude, in the end, when you check the ACF of your "res.resid*res.resid", it's still correlated!

AyoeLay 5 years ago · 0 Likes

Great post...
And I want to know about bilinear model...

Andreas 5 years ago · 0 Likes

Great post! I have actually been doing this myself, converting R code to Python, as I am more comfortable working in Python and it inter-operates better with my toolset. My question is, after the last step in the article, how do you use this model to forecast the next day's returns in SPY?

june 4 years ago · 0 Likes

I have the same question.
How to integrate GARCH with ARIMA for prediction?

Christina 5 years ago · 0 Likes

Has ARIMA(4,0,4) failed to calibrate for SPY return? The 21day in-sample prediction looks so far away from the actual. Also when i plotted the cumulative sum, it shows that the prediction and the actual are going towards different directions...

Also i see convergence error when I run it:
ConvergenceWarning: Maximum Likelihood optimization failed to converge. Check mle_retvals

TT 5 years ago · 0 Likes

This is no doubt one of the best introduction online. Very well explained. Thank you!

Akane 6 years ago · 0 Likes

You sir are a life saver, this was incredibly useful. I'm migrating from Matlab/Stata with not much programming experience so thank you. Would you be able to cover Hawkes process simulation in python by any chance?

Enjo 6 years ago · 0 Likes

Hello Brian,

I like your post very much - thanks for that.
There are however compilation errors in your Python script and I'd gladly run through it with you. Are you interested? If so, what's the best way to contact you in this regard?

Rajesh 7 years ago · 0 Likes

Hi,

Interesting post, thanks for sharing this with the wider data science community. TS is a kind of analysis that ought to be done more often!

In the model assumptions and evaluation criteria, a good practice is to check for residual normality. The QQ-plots in several models indicate normality.

Also, it would be nice if we can evaluate whether the residuals are themselves autocorrelated - because such behaviour indicates that our model assumption is being violated.

I have a question on best practices you might use - do you know of cross validation methods similar to ML cross validation (such as holding or k-fold CV) for TSA? IIRC only Rob Hyndman mentions cross validation but I haven't found a mathematical treatment or function for this yet. Please comment.

Once again, thanks for the post.

Brian Christopher 7 years ago · 0 Likes

Hi Rajesh,

Thanks for commenting.

You can use cross validation with time series analysis quite simply using K-Fold and leave-one-out alternatives. Essentially you're doing the same thing you would in a classical machine learning problem; using subsets of your data to make multiple predictions.

Variations of this theme include simply estimating your model for relevant but slightly different time periods, shifting your start and end dates.

In my experience developing equity trading models, TSA alone is too unstable to create valid predictions because we know that stock returns are not stationary and therefore violate the central tenet of these techniques.

In the article above we do evaluate whether the model residuals are autocorrelated. Using the tsplot() function we output the ACF, and PACF for several different models and discuss their interpretation.

Nyan 7 years ago · 0 Likes

Hi,

Thanks for the good work and sharing it . However, I have a question.

The model residuals for fitting SPX log returns with GARCH model is around 0.98 (one to last graph) while those using ARIMA model is around zero. Meaning the fit with GARCH is not good at all. What good is the normal distribution of the residuals if the residuals are this high ? There has to be something missing here.

Best,

Brian Christopher 7 years ago · 0 Likes

Hi Nyan,

Thanks for taking a critical look at the analysis.

I've tried to find the ARIMA output that shows the residuals were equal to zero but can't locate it. If you're talking about the Garch(1, 1) model simulation, the model residuals are much smaller because we purposefully simulated a Garch(1, 1) process then fit a Garch(1, 1) model to it.

Regarding the Garch(3,2) model fit to SPY returns, I intentionally overfit the model by selecting a low volatility regime for purposes of demonstration. The model in its current form would not be reliable for serious use cases.

Erk 7 years ago · 1 Like

Hi Brian,

I was thinking exact same thing as Nyan. You haven't showed the ARIMA model summary in the post but if you just give a look with;

```
TS = lrets.SPY.ix['2012':'2015']
res_tup = _get_best_model(TS)
p(res_tup[2].summary())
```

You would see pretty good P values for the ARIMA model. However the amplitude of the predictions/fitted values for that model is an order of magnitude smaller compared to the residual term. Thus there is limited usability in practice.