Team Auquan Follow
Dec 8 · 7 min read

Time Series Analysis for Financial Data V— ARIMA Models

Download IPython Notebook here.

In the previous posts in this series, we combined the Autoregressive models and Moving Average models to produce Auto Regressive Moving Average(ARMA) models. We found that we were still unable to fully explain autocorrelation or obtain residuals that are discrete white noise.

Let's further extend this discussion of merging AR and MA models to Auto Regressive Integrated Moving Average(ARIMA) models and see what we get.

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

ARIMA is a natural extension to the class of ARMA models—they can reduce a non-stationary series to a stationary series using a sequence of differences.

We've seen that many of our TS are not stationary, however they can be made stationary by differencing. We saw an example of this in part 1 of the post when we took the first difference of non-stationary Guassian random walk and proved that it equals stationary white noise.

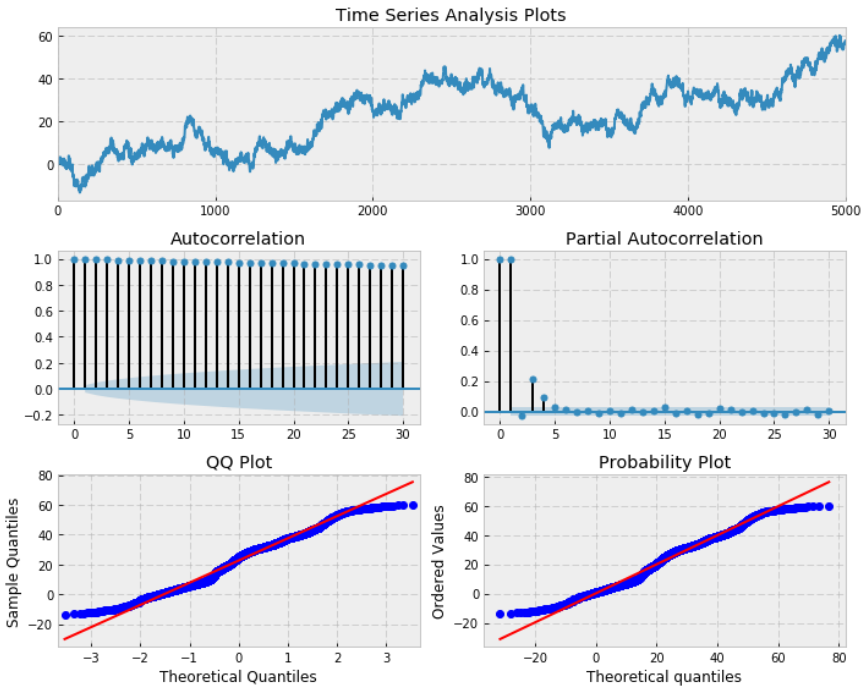
ARIMA essentially performs same function, but does so repeatedly, *d* times, in order to reduce a non-stationary series to a stationary one.

> *A time series x(t), is integrated of order d if differencing the series d times results in a discrete white noise series.*

A time series *x(t)* is *ARIMA(p,d,q)* model if the series is differenced *d* times, and it then follows an *ARMA(p,q)* process.

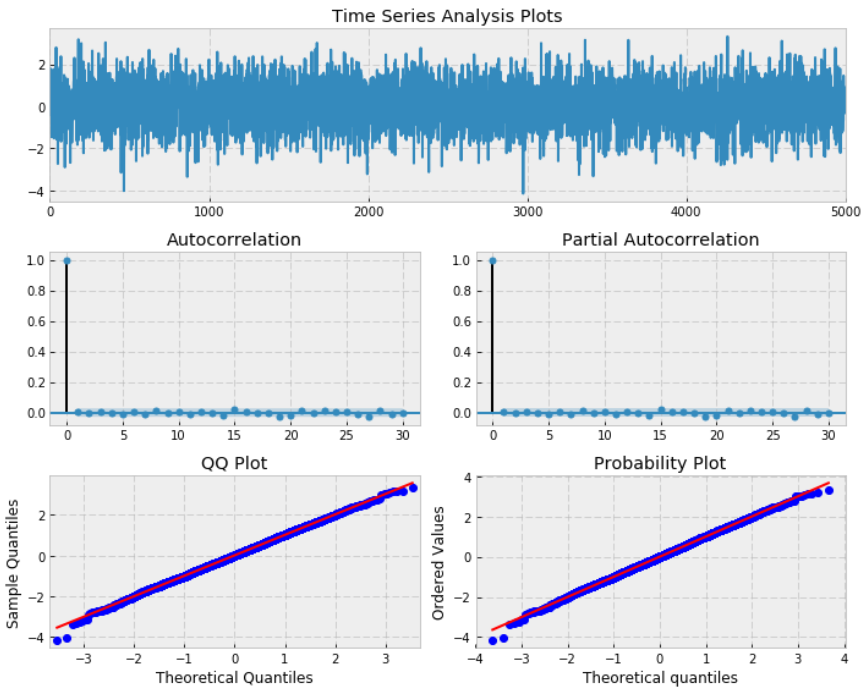
Let's simulate an ARIMA(2,1,1) model, with alphas equal to [0.5,-0.25] and beta = [-0.5] . We will fit an ARIMA model to our simulated data, attempt to recover the parameters.

```
# Simulate an ARIMA(2,1,1) model
# alphas=[0.5,-0.25]
# betas=[-0.5]

max_lag = 30

n = int(5000)
burn = 2000

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

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

arma11 = smt.arma_generate_sample(ar=ar, ma=ma, nsample=n,
burnin=burn)
arima111 = arma11.cumsum()
_ = tsplot(arima111, lags=max_lag)
```



ARIMA(2,1,1) model

```
# Fit ARIMA(p, d, q) model
# 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]
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(arima111,
                                    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


print('aic: %6.2f | order: %s'%(best_aic, best_order))

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



Residuals of ARIMA(2,1,1) fit

aic: 14227.34 | order: (2, 1, 1)

As expected, we predict a ARIMA(2,1,1) model and the residuals looking like a realisation of discrete white noise:

```
sms.diagnostic.acorr_ljungbox(best_mdl.resid, lags=[20],
boxpierce=False)


from statsmodels.stats.stattools import jarque_bera


score, pvalue, _, _ = jarque_bera(mdl.resid)


if pvalue < 0.10:
    print 'The residuals may not be normally distributed.'
else:
    print 'The residuals seem normally distributed.'
```

. . .

```
(array([ 13.88378716]), array([ 0.83633895]))
The residuals seem normally distributed.
```

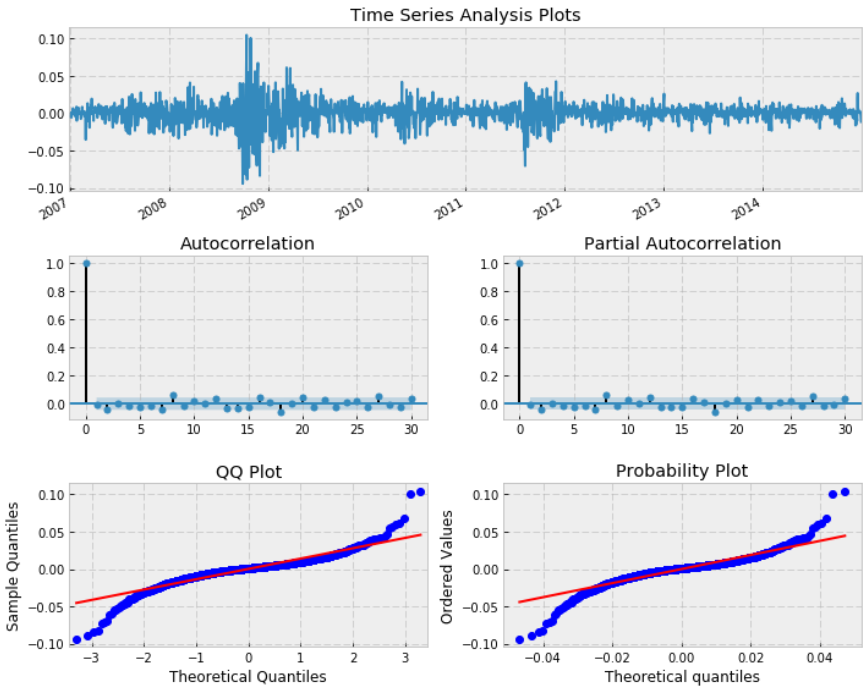
We perform the Ljung-Box test and find the p-value is significantly larger than 0.05 and as such we can state that there is strong evidence for discrete white noise being a good fit to the residuals. Hence, the ARIMA(2,1,1) model is a good fit, as expected.

Modelling SPX returns

Let's now iterate through a non-trivial number of combinations of (p, d, q) orders, to find the best ARIMA model to fit SPX returns. We use the AIC to evaluate each model. The lowest AIC wins.

```
# Fit ARIMA(p, d, q) model to SPX log 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]
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.SPX,
                          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


print('aic: {:6.2f} | order: {}'.format(best_aic,
best_order))

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



Residuals of Modelling SPX returns from 2007–2015 as ARIMA(3,0,2) model

```
aic: -11515.95 | order: (3, 0, 2)
```

Note that the best model has a differencing of 0. This is expected because we already took the first difference of log prices to calculate the stock returns. The result is essentially identical to the ARMA(3, 2) model we fit in the previous post. Clearly this ARIMA model has not explained the conditional volatility in the series either! The ljung box test below also shows a pvalue of less than 0.05

```
sms.diagnostic.acorr_ljungbox(best_mdl.resid, lags=[20],
boxpierce=False
```

(array([39.20689685]), array([0.00628326]))

Excluding periods of Conditional Volatility

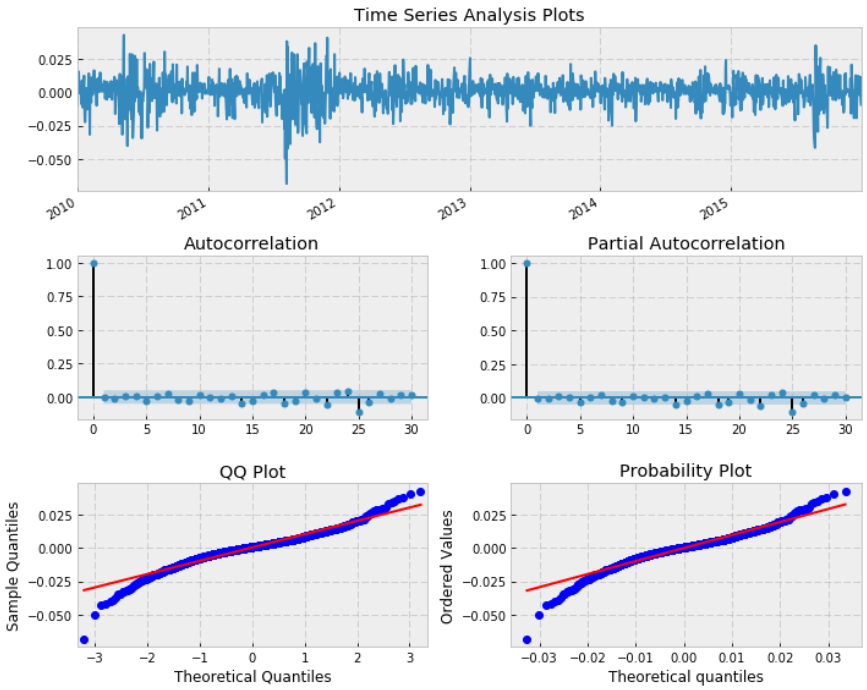
Let's now try the same model on SPX data from 2010–2016

```
end = '2016-01-01'
start = '2010-01-01'
symbols = ['SPX']
data = dl.load_data_nologs('nasdaq', symbols , start, end)
['ADJ CLOSE']
# log returns
lrets = np.log(data/data.shift(1)).dropna()
```

. . .

```
# Fit ARIMA(p, d, q) model to SPX log 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]
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.SPX,
                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


print('aic: {:6.2f} | order: {}'.format(best_aic,
best_order))

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



Residuals of Modelling SPX returns from 2010–2016 as ARIMA(3,0,3) model

```
aic: -9622.34 | order: (3, 0, 3)
```

Our residuals look much closer to white noise! How did our model suddenly improve?

```
sms.diagnostic.acorr_ljungbox(best_mdl.resid, lags=[20],
boxpierce=False)
```

```
from statsmodels.stats.stattools import jarque_bera

score, pvalue, _, _ = jarque_bera(mdl.resid)

if pvalue < 0.10:
    print 'The residuals may not be normally distributed.'
else:
    print 'The residuals seem normally distributed.'
```

. . .

```
(array([ 18.93350068]), array([ 0.52615227]))


The residuals seem normally distributed.
```

The p-value of our test is now greater than 0.05!

We deliberately truncated the S&P500 data to start from 2010 onwards, which conveniently excludes the volatile periods around 2007–2008. Hence we have excluded a large portion of the S&P500 where we had excessive volatility clustering. This impacts the serial correlation of the series and hence has the effect of making the series seem “more stationary” than it has been in the past.

> *This is a very important point. When analysing time series we need to be extremely careful of conditionally heteroscedastic series, such as stock market indexes. In quantitative finance, trying to determine periods of differing volatility is often known as ”regime detection”. It is one of the harder tasks to achieve!*

Time Series Forecasting

Finally, we are able to do what we actually set out to do! We have at least accumulated enough knowledge to make a simple forecast of future returns. We use statmodels forecast() method—we need to

provide 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%

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_95', 'upper_95'])
fc_99 = pd.DataFrame(np.column_stack([ci99]), index=idx,
columns=
                     ['lower_99', 'upper_99'])
fc_all = fc_95.combine_first(fc_99)
fc_all.head()
```

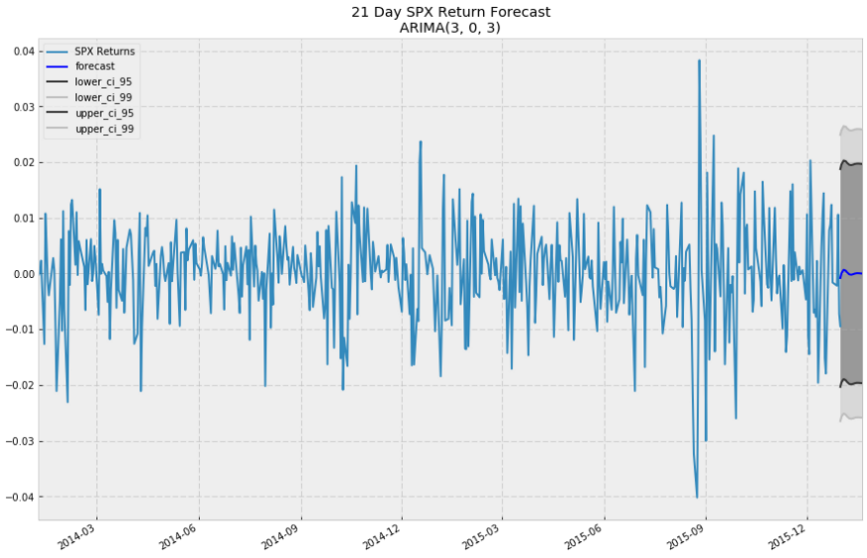
. . .

```
           |forecast | lower_95  |  lower_99 | upper_95 |
upper_99 |
-------------------------------------------------------------
--------
2015-12-31 |-0.00079 | -0.020347 | -0.026490 | 0.018754 |
0.024897 |
2016-01-01 |0.000004 | -0.019563 | -0.025712 | 0.019572 |
0.025721 |
2016-01-02 |0.000358 | -0.019215 | -0.025365 | 0.019931 |
0.026081 |
2016-01-03 |0.000667 | -0.018968 | -0.025138 | 0.020302 |
0.026472 |
2016-01-04 |0.000586 | -0.019051 | -0.025222 | 0.020223 |
0.026394 |
```

. . .

```
# Plot 21 day forecast for SPX returns

plt.style.use('bmh')
```

```
fig = plt.figure(figsize=(15,10))
ax = plt.gca()

ts = lrets.SPX.iloc[-500:].copy()
ts.plot(ax=ax, label='SPX 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 SPX Return
Forecast\nARIMA{}'.format(n_steps, best_order))
plt.legend(loc='best', fontsize=10)
```



Now that we have the ability to fit and forecast models such as ARIMA, we're very close to being able to create strategy indicators for trading.

You can already start analysing different time series, like difference between prices of two correlated stocks, and try the above models to check for stationarity. Once you find a model that fits the series well enough to leave white noise like residuals, you can start using that model for forecasting future values. And that's really all there is to forecasting!