8
Time Series Models

In the last chapter, we focused on linear models tailored to cross-sectional data where the input data belongs to the same time period as the output they aim to explain or predict. In this chapter, we will focus on time series data where observations differ by period, which also creates a natural ordering. Our goal will be to identify historical patterns in data and leverage these patterns to predict how the time series will behave in the future.

We already encountered panel data with both a cross-sectional and a time series dimension in the last chapter and learned how the Fama-Macbeth regression estimates the value of taking certain factor risks over time and across assets. However, the relationship between returns across time is typically fairly low, so this procedure could largely ignore the time dimension. The models in this chapter focus on time series models where past values contain predictive signals about future developments. Time series models can also predict features that are then used in cross-sectional models.

More specifically, in this chapter, we focus on models that extract signals from previously observed data to predict future values for the same time series. The time dimension of trading makes the application of time series models to market, fundamental, and alternative data very popular. Time series data will become even more prevalent as an ever broader array of connected devices collects regular measurements that may contain predictive signals. Key applications include the prediction of asset returns, correlations or covariances, or volatility.

We focus on linear time series models in this chapter as a baseline for non-linear models like recurrent or convolutional neural networks that we apply to time series data in part 4 of this book. We being by introducing tools to diagnose time series characteristics, including stationarity, and extract features that capture potential patterns. Then we introduce univariate and multivariate time series models and apply them to forecast macro data and volatility patterns. We conclude with the concept of cointegration and how to apply it to develop a pairs trading strategy.

In particular, we will cover the following topics:

- How to use time series analysis to diagnose diagnostic statistics that inform the modeling process
- How to estimate and diagnose autoregressive and moving-average time series models
- How to build Autoregressive Conditional Heteroskedasticity (ARCH) models to predict volatility
- How to build vector autoregressive models
- How to use cointegration for a pairs trading strategy

Analytical tools for diagnostics and feature extraction

Time series data is a sequence of values separated by discrete time intervals that are typically even-spaced (except for missing values). A time series is often modeled as a stochastic process consisting of a collection of random variables, $y(t_1)$, ..., $y(t_T)$, with one variable for each point in time, t_i, i=1, ..., T. A univariate time series consists of a single value, y, at each point in time, whereas a multivariate time series consists of several observations that can be represented by a vector.

The number of periods, $\Delta t = t_i - t_j$, between distinct points in time, t_i, t_j, is called lag, with T-1 lags for each time series. Just as relationships between different variables at a given point in time is key for cross-sectional models, relationships between data points separated by a given lag are fundamental to analyzing and exploiting patterns in time series. For cross-sectional models, we distinguished between input and output variables, or target and predictors, with the labels y and x, respectively. In a time series context, the lagged values of the outcome play the role of the input or x values in the cross-section context.

A time series is called white noise if it is a sequence of independent and identically-distributed random variables, ε_t, with finite mean and variance. In particular, the series is called a Gaussian white noise if the random variables are normally distributed with a mean of zero and a constant variance of σ.

A time series is linear if it can be written as a weighted sum of past disturbances, ε_t, that are also called innovations, and are here assumed to represent white noise, and the mean of the series, μ:

$$y_t = \mu + \sum_{i=0}^{\infty} a_i \epsilon_{t-i}, \quad a_0 = 1, \epsilon \sim \text{i.i.d.}$$

A key goal of time series analysis is to understand the dynamic behavior driven by the coefficients, a_i. The analysis of time series offers methods tailored to this type of data with the goal of extracting useful patterns that, in turn, help us to build predictive models. We will introduce the most important tools for this purpose, including the decomposition into key systematic elements, the analysis of autocorrelation, and rolling window statistics such as moving averages. Linear time series models often make certain assumptions about the data, such as stationarity, and we will also introduce both the concept, diagnostic tools, and typical transformations to achieve stationarity.

For most of the examples in this chapter, we work with data provided by the Federal Reserve that you can access using the `pandas datareader` that we introduced in `Chapter 2`, *Market and Fundamental Data*. The code examples for this section are available in the notebook `tsa_and_arima` notebook.

How to decompose time series patterns

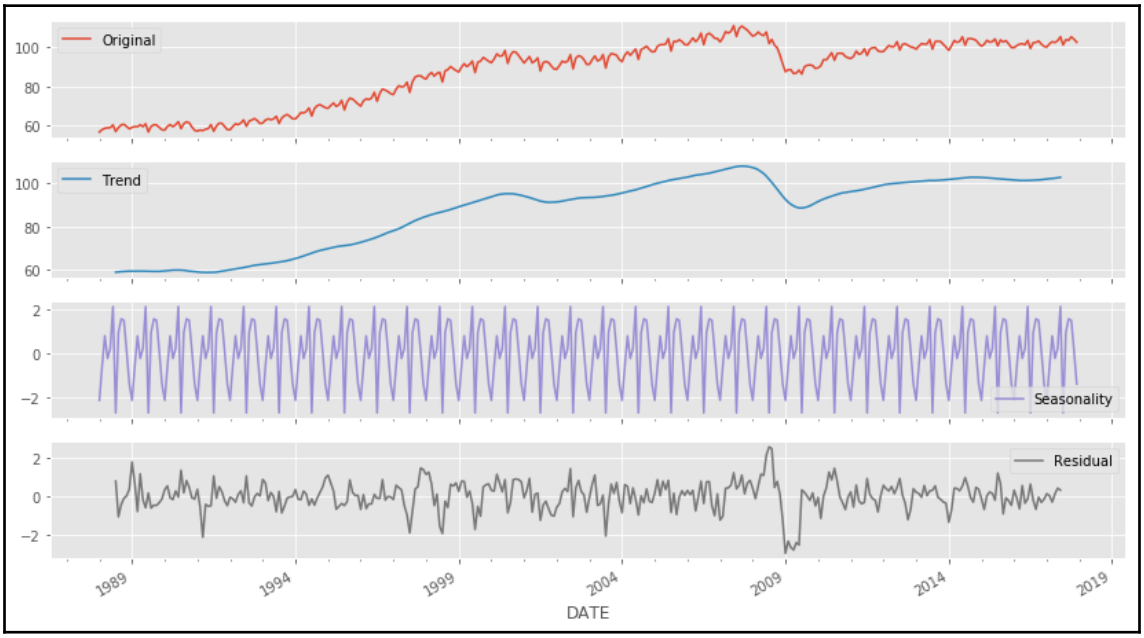
Time series data typically contains a mix of various patterns that can be decomposed into several components, each representing an underlying pattern category. In particular, time series often consist of the systematic components trend, seasonality and cycles, and unsystematic noise. These components can be combined in an additive, linear model, in particular when fluctuations do not depend on the level of the series, or in a non-linear, multiplicative model.

These components can be split up automatically. `statsmodels` includes a simple method to split the time series into a trend, seasonal, and residual component using moving averages. We can apply it to monthly data on industrial manufacturing production with both a strong trend and seasonality component, as follows:

```
import statsmodels.tsa.api as tsa
industrial_production = web.DataReader('IPGMFN', 'fred', '1988',
'2017-12').squeeze()
components = tsa.seasonal_decompose(industrial_production,
model='additive')
```

```
ts = (industrial_production.to_frame('Original')
      .assign(Trend=components.trend)
      .assign(Seasonality=components.seasonal)
      .assign(Residual=components.resid))
ts.plot(subplots=True, figsize=(14, 8));
```

The resulting charts show the additive components. The residual component would be the focus of additional modeling, assuming that the trend and seasonality components are more deterministic and amenable to simple extrapolation:



There are more sophisticated, model-based approaches that are included in the references available on GitHub.

How to compute rolling window statistics

Given the sequential ordering of time series data, it is natural to compute familiar descriptive statistics for periods of a given length to detect stability or changes in behavior and obtain a smoothed representation that captures systematic aspects while filtering out the noise.

Rolling window statistics serve this process: they produce a new time series where each data point represents a summary statistic computed for a certain period of the original data. Moving averages are the most familiar example. The original data points can enter the computation with equal weights, or using weights to, for example, emphasize more recent data points. Exponential moving averages recursively compute weights that shrink or decay, for data points further away from the present. The new data points are typically a summary of all preceding data points, but they can also be computed from a surrounding window.

The `pandas` library includes very flexible functionality to define various window types, including rolling, exponentially weighted and expanding windows. In a second step, you can apply computations to each data captured by a window. These computations include built-in standard computations for individual series, such as the mean or the sum, the correlation or covariance for several series, as well as user-defined functions. The moving average and exponential smoothing examples in the following section make use of these tools.

Moving averages and exponential smoothing

Early forecasting models included moving-average models with exponential weights called exponential smoothing models. We will encounter moving averages again as key building blocks for linear time series.

Forecasts that rely on exponential smoothing methods use weighted averages of past observations, where the weights decay exponentially as the observations get older. Hence, a more recent observation receives a higher associated weight. These methods are popular for time series that do not have very complicated or abrupt patterns.

Exponential smoothing is a popular technique based on weighted averages of past observations, with the weights decaying exponentially as the observations get older. In other words, the more recent the observation, the higher the associated weight. This framework generates reliable forecasts quickly and for a wide range of time series, which is a great advantage and of major importance to applications in industry.

How to measure autocorrelation

Autocorrelation (also called serial correlation) adapts the concept of correlation to the time series context: just as the correlation coefficient measures the strength of a linear relationship between two variables, the autocorrelation coefficient, ρ_k, measures the extent of a linear relationship between time series values separated by a given lag, k:

$$\rho_k = \frac{\sum_{t=k+1}^{T}(y_t - \bar{y})(y_{t-k} - \bar{y})}{\sum_{t=1}^{T}(y_t - \bar{y})^2}$$

Hence, we can calculate one autocorrelation coefficient for each of the T-1 lags in a time series; T is the length of the series. The autocorrelation function (ACF) computes the correlation coefficients as a function of the lag.

The autocorrelation for a lag larger than 1 (that is, between observations more than one time step apart) reflects both the direct correlation between these observations and the indirect influence of the intervening data points. The partial autocorrelation removes this influence and only measures the linear dependence between data points at the given lag distance. The **partial autocorrelation function** (**PACF**) provides all the correlations that result once the effects of a correlation at shorter lags have been removed.

There are algorithms that estimate the partial autocorrelation from the sample autocorrelation based on the exact theoretical relationship between the PACF and the ACF.

A correlogram is simply a plot of the ACF or PACF for sequential lags, k=0,1,...,n. It allows us to inspect the correlation structure across lags at one glance. The main usage of correlograms is to detect any autocorrelation after the removal of the effects of deterministic trend or seasonality. Both the ACF and the PACF are key diagnostic tools for the design of linear time series models and we will review examples of ACF and PACF plots in the following section on time series transformations.

How to diagnose and achieve stationarity

The statistical properties, such as the mean, variance, or autocorrelation, of a stationary time series are independent of the period, that is, they don't change over time. Hence, stationarity implies that a time series does not have a trend or seasonal effects and that descriptive statistics, such as the mean or the standard deviation, when computed for different rolling windows, are constant or do not change much over time. It reverts to its mean, and the deviations have constant amplitude, while short-term movements always look the same in the statistical sense.

More formally, strict stationarity requires the joint distribution of any subset of time series observations to be independent of time with respect to all moments. So, in addition to the mean and variance, higher moments such as skew and kurtosis, also need to be constant, irrespective of the lag between different observations. In most applications, we limit stationarity to first and second moments so that the time series is covariance stationary with constant mean, variance, and autocorrelation.

Note that we specifically allow for dependence between observations at different lags, just like we want the input data for linear regression to be correlated with the outcome. Stationarity implies that these relationships are stable, which facilitates prediction as the model can focus on learning systematic patterns that take place within stable statistical properties. It is important because classical statistical models assume that the time series input data is stationary.

The following sections introduce diagnostics that help detect when data is not stationary, and transformations that help meet these assumptions.

Time series transformations

To satisfy the stationarity assumption of linear time series models, we need to transform the original time series, often in several steps. Common transformations include the application of the (natural) logarithm to convert an exponential growth pattern into a linear trend and stabilize the variance. Deflation implies dividing a time series by another series that causes trending behavior, for example dividing a nominal series by a price index to convert it into a real measure.

A series is trend-stationary if it reverts to a stable long-run linear trend. It can often be made stationary by fitting a trend line using linear regression and using the residuals, or by including the time index as an independent variable in a regression or AR(I)MA model (see the following section on univariate time series models), possibly combined with logging or deflating.

In many cases, de-trending is not sufficient to make the series stationary. Instead, we need to transform the original data into a series of period-to-period and/or season-to-season differences. In other words, we use the result of subtracting neighboring data points or values at seasonal lags from each other. Note that when such differencing is applied to a log-transformed series, the results represent instantaneous growth rates or returns in a financial context.

If a univariate series becomes stationary after differencing d times, it is said to be integrated of the order of d, or simply integrated if d=1. This behavior is due to so-called unit roots.

How to diagnose and address unit roots

Unit roots pose a particular problem for determining the transformation that will render a time series stationary. Time series are often modeled as stochastic processes of the following autoregressive form that we will explore in more detail as a building block for ARIMA models:

$$y_t = a_1 y_{t-1} + a_2 y_{t-2} + \ldots + a_p y_{t-p} + \epsilon_t$$

Where the current value is a weighted sum of past values plus a random disturbance. Such a process has a characteristic equation of the following form:

$$m^p - m^{p-1} a_1 - m^{p-2} a_2 - \ldots - a_p = 0$$

If one of the roots of this equation equals 1, then the process is said to have a unit root. It will be non-stationary but does not necessarily need to have a trend. If the remaining roots of the characteristic equation are less than 1 in absolute terms, the first difference of the process will be stationary, and the process is integrated (of order 1) or I(1). With additional roots larger than 1 in absolute terms, the order of integration is higher and additional differencing will be required.

In practice, time series of interest rates or asset prices are often not stationary, for example, because there does not exist a price level to which the series reverts. The most prominent example of a non-stationary series is the random walk for a time series of price, p_t, for a given starting price, p_0 (for example, a stock's IPO price) and a white-noise disturbance, ε, that satisfies the following:

$$p_t = p_{t-1} + \epsilon_t = \sum_{s=0}^{t} \epsilon_s + p_0$$

Repeated substitution shows that the current value, p_t, is the sum of all prior disturbances or innovations, ε, and the initial price, p_0. If the equation includes a constant term, then the random walk is said to have drift. Hence, the random walk is an autoregressive stochastic process of the following form:

$$y_t = a_1 y_{t-1} + \epsilon_t, \quad a_1 = 1$$

With the characteristic equation, $m - a_1 = 0$, that has a unit root and is both non-stationary and integrated of order 1. On the one hand, given the i.i.d. nature of ε, the variance of the time series equals σ^2, which is not second-order stationary and implies that, in principle, the series could, over time, assume any variable. On the other hand, taking the first difference, $\Delta p_t = p_t - p_{t-1}$, leaves $\Delta p_t = \varepsilon_t$, which is stationary, given the statistical assumption about ε.

The defining characteristic of a unit-root non-stationary series is long memory: since current values are the sum of past disturbances, large innovations persist for much longer than for a mean-reverting, stationary series.

In addition to using the difference between neighboring data points to remove a constant pattern of change, it can be used to apply seasonal differencing to remove patterns of seasonal change. This involves taking the difference of values at a lag distance that represents the length of a seasonal pattern, which is four quarters, or 12 months, apart to remove both seasonality and linear trend.

Identifying the correct transformation, and in particular, the appropriate number and lags for differencing is not always clear-cut. Some rules have been suggested, summarized as follows:

- **Positive autocorrelations up to 10+ lags**: Probably needs higher-order differencing.
- **Lag-1 autocorrelation close to zero or negative, or generally small and patternless**: No need for higher-order differencing.
- **Lag-1 autocorrelation < -0.5**: Series may be over-differenced.
- Slightly over- or under-differencing can be corrected with AR or MA terms.
- Optimal differencing often produces the lowest standard deviation, but not always.
- A model without differencing assumes that the original series is stationary, including mean-reverting. It normally includes a constant term to allow for a non-zero mean.
- A model with one order of differencing assumes that the original series has a constant average trend and should include a constant term.
- A model with two orders of differencing assumes that the original series has a time-varying trend and should not include a constant.

Some authors recommend fractional differencing as a more flexible approach to rendering an integrated series stationary and may be able to keep more information or signal than simple or seasonal differences at discrete intervals (see references on GitHub).

Unit root tests

Statistical unit root tests are a common way to determine objectively whether (additional) differencing is necessary. These are statistical hypothesis tests of stationarity that are designed to determine whether differencing is required.

The augmented Dickey-Fuller (ADF) test evaluates the null hypothesis that a time series sample has unit root against the alternative of stationarity. It regresses the differenced time series on a time trend, the first lag, and all lagged differences, and computes a test statistic from the value of the coefficient on the lagged time series value. `statsmodels` makes it easy to implement (see companion notebook).

Formally, the ADF test for a time series, y_t, runs the linear regression:

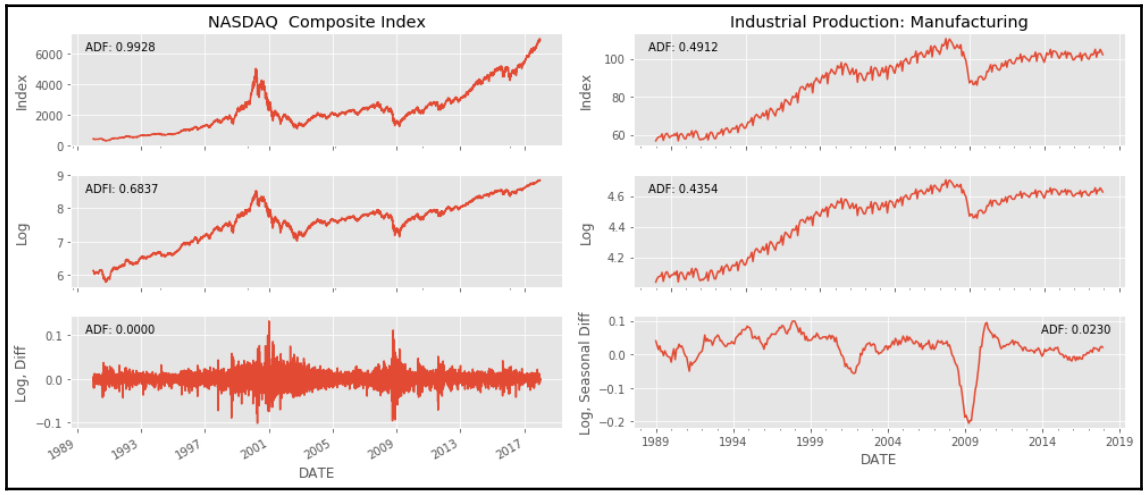
$$\Delta y_t = \alpha + \beta t + \gamma y_{t-1} + \delta_1 \Delta y_{t-1} + \ldots + \delta_{p-1} \Delta y_{t-p+1} + \epsilon_t$$

Where α is a constant, β is a coefficient on a time trend, and p refers to the number of lags used in the model. The α=β =0 constraint implies a random walk, whereas only β=0 implies a random walk with drift. The lag order is usually decided using the AIC and BIC information criteria introduced in `Chapter 7`, *Linear Models*.

The ADF test statistics uses the sample coefficient, γ, that, under the null hypothesis of unit-root non-stationarity equals zero, and is negative otherwise. It intends to demonstrate that, for an integrated series, the lagged series value should not provide useful information in predicting the first difference above and beyond lagged differences.

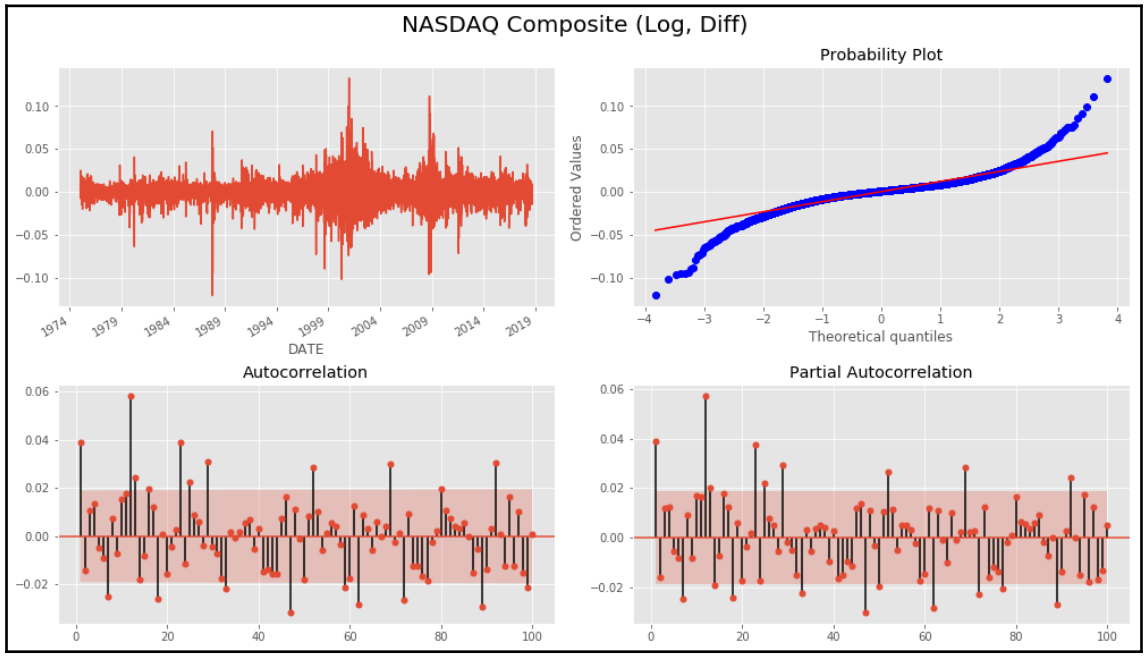
How to apply time series transformations

The following chart shows time series for the NASDAQ stock index and industrial production for the 30 years through 2017 in original form, as well as the transformed versions after applying the logarithm and subsequently applying first and seasonal differences (at lag 12), respectively. The charts also display the ADF p-value, which allows us to reject the hypothesis of unit-root non-stationarity after all transformations in both cases:

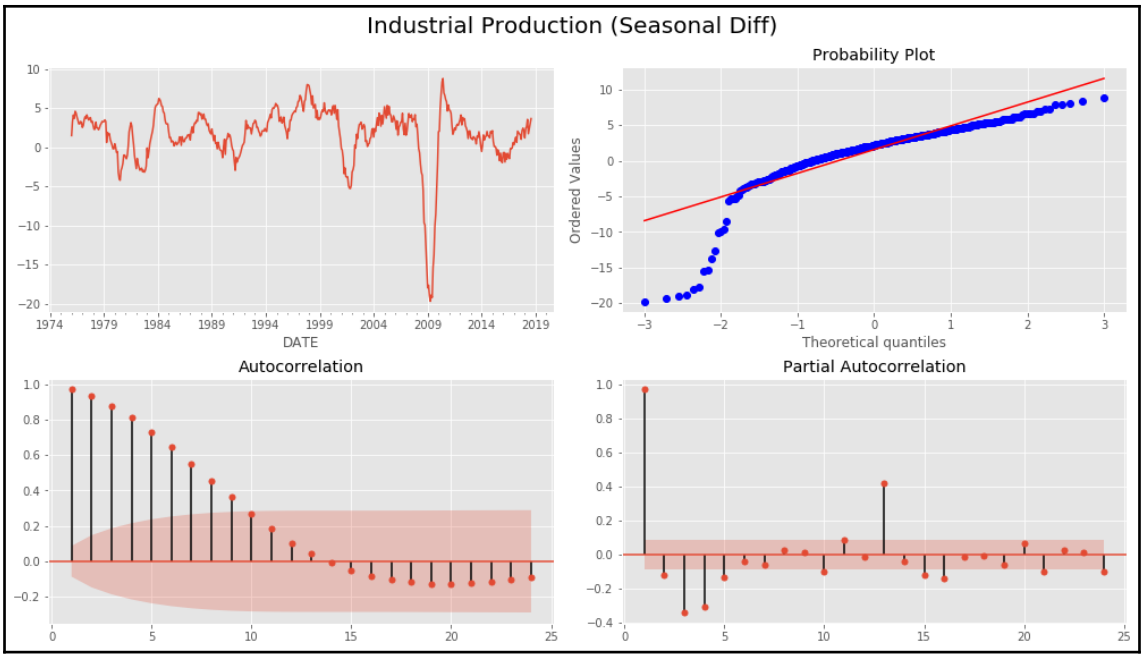


We can further analyze the relevant time series characteristics for the transformed series using a Q-Q plot that compares the quantiles of the distribution of the time series observation to the quantiles of the normal distribution and the correlograms based on the ACF and PACF.

For the NASDAQ plot, we notice that while there is no trend, the variance is not constant but rather shows clustered spikes around periods of market turmoil in the late 1980s, 2001, and 2008. The Q-Q plot highlights the fat tails of the distribution with extreme values more frequent than the normal distribution would suggest. The ACF and the PACF show similar patterns with autocorrelation at several lags appearing significant:



For the monthly time series on industrial manufacturing production, we notice a large negative outlier following the 2008 crisis as well as the corresponding skew in the Q-Q plot. The autocorrelation is much higher than for the NASDAQ returns and declines smoothly. The PACF shows distinct positive autocorrelation patterns at lag 1 and 13, and significant negative coefficients at lags 3 and 4:



Univariate time series models

Multiple linear-regression models expressed the variable of interest as a linear combination of predictors or input variables. Univariate time series models relate the value of the time series at the point in time of interest to a linear combination of lagged values of the series and possibly past disturbance terms.

While exponential smoothing models are based on a description of the trend and seasonality in the data, ARIMA models aim to describe the autocorrelations in the data. ARIMA(p, d, q) models require stationarity and leverage two building blocks:

- **Autoregressive** (**AR**) terms consisting of p-lagged values of the time series
- **Moving average** (**MA**) terms that contain q-lagged disturbances

The I stands for integrated because the model can account for unit-root non-stationarity by differentiating the series d times. The term autoregression underlines that ARIMA models imply a regression of the time series on its own values.

We will introduce the ARIMA building blocks, simple autoregressive (AR) and moving average (MA) models, and explain how to combine them in autoregressive moving-average (ARMA) models that may account for series integration as ARIMA models or include exogenous variables as AR(I)MAX models. Furthermore, we will illustrate how to include seasonal AR and MA terms to extend the toolbox to also include SARMAX models.

How to build autoregressive models

An AR model of order p aims to capture the linear dependence between time series values at different lags and can be written as follows:

$$\mathrm{AR}(p): \quad y_t = \phi_0 + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \ldots + \phi_p y_{t-p} + \epsilon_t, \quad \epsilon \sim \text{i.i.d.}$$

This closely resembles a multiple linear regression on lagged values of y_t. This model has the following characteristic equation:

$$1 - \phi_1 x - \phi_2 x^2 - \ldots - \phi_p x^p = 0$$

The inverses of the solution to this equation in x are the characteristic roots, and the AR(p) process is stationary if these roots are all less than 1 in absolute terms, and unstable otherwise. For a stationary series, multi-step forecasts will converge to the mean of the series.

We can estimate the model parameters with the familiar least squares method using the p+1, ..., T observations to ensure there is data for each lagged term and the outcome.

How to identify the number of lags

In practice, the challenge consists in deciding on the appropriate order p of lagged terms. The time series analysis tools for serial correlation play a key role. The ACF estimates the autocorrelation between observations at different lags, which in turn results from both direct and indirect linear dependence.

Hence, for an AR model of order k, the ACF will show a significant serial correlation up to lag k and, due to the inertia caused by the indirect effects of the linear relationship, will extend to subsequent lags and eventually trail off as the effect was weakened. On the other hand, the PACF only measures the direct linear relationship between observations a given lag apart so that it will not reflect correlation for lags beyond *k*.

How to diagnose model fit

If the model captures the linear dependence across lags, then the residuals should resemble white noise.

In addition to inspecting the ACF to verify the absence of significant autocorrelation coefficients, the Ljung-Box Q statistic allows us to test the hypothesis that the residual series follows white noise. The null hypothesis is that all m serial correlation coefficients are zero against the alternative that some coefficients are not. The test statistic is computed from the sample autocorrelation coefficients, ρ_k, for different lags, k, and follows an X^2 distribution:

$$Q(m) = T(T+2)\sum_{l=1}^{m}\frac{\rho_l^2}{T-l}$$

As we will see, `statsmodels` provides information about the significance of coefficients for different lags, and insignificant coefficients should be removed. If the Q statistic rejects the null hypothesis of no autocorrelation, you should consider additional AR terms.

How to build moving average models

An MA model of order q uses q past disturbances rather than lagged values of the time series in a regression-like model, as follows:

$$\mathrm{MA}(q): \quad y_t = c + \epsilon_t + \theta_1\epsilon_{t-1} + \theta_2\epsilon_{t-2} + \ldots + \theta_q\epsilon_{t-q}, \quad \epsilon \sim \text{i.i.d.}$$

Since we do not observe the white-noise disturbance values, ε_t, MA(q) is not a regression model like the ones we have seen so far. Rather than using least squares, MA(q) models are estimated using **maximum likelihood** (**MLE**), alternatively initializing or estimating the disturbances at the beginning of the series and then recursively and iteratively computing the remainder.

The MA(q) model gets its name from representing each value of y_t as a weighted moving average of the past q innovations. In other words, current estimates represent a correction relative to past errors made by the model. The use of moving averages in MA(q) models differs from that of exponential smoothing or the estimation of seasonal time series components because an MA(q) model aims to forecast future values as opposed to de-noising or estimating the trend cycle of past values.

MA(q) processes are always stationary because they are the weighted sum of white noise variables that are themselves stationary.

How to identify the number of lags

A time series generated by an MA(q) process is driven by the residuals from the q prior-model predictions. Hence, the ACF for the MA(q) process will show significant coefficients for values up to the lag, q, and then decline sharply because this is how the series values are assumed to have been generated.

The relationship between AR and MA models

An AR(p) model can be expressed as an MA(∞) process using repeated substitution. When imposing constraints on the size of its coefficients, an MA(q) process, it becomes invertible and can be expressed as an AR(∞) process.

How to build ARIMA models and extensions

Autoregressive integrated moving-average ARIMA(p, d, q) models combine AR(p) and MA(q) processes to leverage the complementarity of these building blocks and simplify model development by using a more compact form and reducing the number of parameters, in turn reducing the risk of overfitting.

The models also take care of eliminating unit-root nonstationarity by using the d^{th} difference of the time series values. An ARIMA(p, 1, q) model is the same as using an ARMA(p, q) model with the first differences of the series. Using y' to denote the original series after non-seasonal differencing d times, the ARIMA(p, d, q) model is simply:

$$\text{ARIMA}(p,d,q): \quad y_t = \text{AR}(p) + \text{MA}(q) = \phi_0 + \phi_1 y_{t-1} + \ldots + \phi_p y_{t-p} + \epsilon_t + \theta_1 \epsilon_{t-1} + \ldots + \theta_q \epsilon_{t-q}, \quad \epsilon \sim \text{i.i.d.}$$

ARIMA models are also estimated using Maximum Likelihood. Depending on the implementation, higher-order models may generally subsume lower-order models. For example, `statsmodels` includes all lower-order p and q terms and does not permit removing coefficients for lags below the highest value. In this case, higher-order models will always fit better. Be careful not to overfit your model to the data by using too many terms.

How to identify the number of AR and MA terms

Since AR(p) and MA(q) terms interact, the information provided by the ACF and PACF is no longer reliable and can only be used as a starting point.

Traditionally, the AIC and BIC information criteria have been used to rely on in-sample fit when selecting the model design. Alternatively, we can rely on out-of-sample tests to cross-validate multiple parameter choices.

The following summary provides some generic guidance to choose the model order in the case of considering AR and MA models in isolation:

- The lag beyond which the PACF cuts off is the indicated number of AR terms. If the PACF of the differenced series cuts off sharply and/or the lag-1 autocorrelation is positive, add one or more AR terms.
- The lag beyond which the ACF cuts off is the indicated number of MA terms. If the ACF of the differenced series displays a sharp cutoff and/or the lag-1 autocorrelation is negative, consider adding an MA term to the model.
- AR and MA terms may cancel out each other's effects, so always try to reduce the number of AR and MA terms by 1 if your model contains both to avoid overfitting, especially if the more complex model requires more than 10 iterations to converge.
- If the AR coefficients sum to nearly 1 and suggest a unit root in the AR part of the model, eliminate 1 AR term and difference the model once (more).
- If the MA coefficients sum to nearly 1 and suggest a unit root in the MA part of the model, eliminate 1 MA term and reduce the order of differencing by 1.
- Unstable long-term forecasts suggest there may be a unit root in the AR or MA part of the model.

Adding features – ARMAX

An ARMAX model adds input variables or covariate on the right-hand side of the ARMA time series model (assuming the series is stationary so we can skip differencing):

$$\text{ARIMA}(p,d,q): \quad y_t = \beta x_t + \text{AR}(p) + \text{MA}(q) = \beta x_t + \phi_0 + \phi_1 y_{t-1} + \ldots + \phi_p y_{t-p} + \epsilon_t + \theta_1 \epsilon_{t-1} + \ldots + \theta_q \epsilon_{t-q}, \quad \epsilon \sim \text{i.i.d.}$$

This resembles a linear regression model but is quite difficult to interpret because the effect of β on y_t is not the effect of an increase in x_t by one unit as in linear regression. Instead, the presence of lagged values of y_t on the right-hand side of the equation implies that the coefficient can only be interpreted given the lagged values of the response variable, which is hardly intuitive.

Adding seasonal differencing – SARIMAX

For time series with seasonal effects, we can include AR and MA terms that capture the seasonality's periodicity. For instance, when using monthly data and the seasonal effect length is one year, the seasonal AR and MA terms would reflect this particular lag length.

The ARIMAX(p, d, q) model then becomes a SARIMAX(p, d, q) x $(P, D, Q)_s$ model, which is a bit more complicated to write out, but the references on GitHub, including the statsmodels documentation, provide this information in detail.

We will now build a seasonal ARMA model using macro-data to illustrate the implementation.

How to forecast macro fundamentals

We will build a SARIMAX model for monthly data on an industrial production time series for the 1988-2017 period. As illustrated in the first section on analytical tools, the data has been log-transformed, and we are using seasonal (lag-12) differences. We estimate the model for a range of both ordinary and conventional AR and MA parameters using a rolling window of 10 years of training data, and evaluate the RMSE of the 1-step-ahead forecast, as shown in the following simplified code (see GitHub for details):

```
for p1 in range(4):                 # AR order
    for q1 in range(4):             # MA order
        for p2 in range(3):         # seasonal AR order
            for q2 in range(3):     # seasonal MA order
                y_pred = []
                for i, T in enumerate(range(train_size, len(data))):
                    train_set = data.iloc[T - train_size:T]
                    model = tsa.SARIMAX(endog=train_set,            # model
specification
                                        order=(p1, 0, q1),
                                        seasonal_order=(p2, 0, q2,
12)).fit()

                    preds.iloc[i, 1] = model.forecast(steps=1)[0]    # 1-
step ahead forecast

                mse = mean_squared_error(preds.y_true, preds.y_pred)
                test_results[(p1, q1, p2, q2)] = [np.sqrt(mse),
preds.y_true.sub(preds.y_pred).std(),
                                                  np.mean(aic)]
```

We also collect the AIC and BIC criteria that show a very high rank correlation coefficient of 0.94, with BIC favoring models with slightly fewer parameters than AIC. The best five models by RMSE are:

```
                  RMSE          AIC          BIC
p1 q1 p2 q2
2  3  1  0    0.009323 -772.247023 -752.734581
3  2  1  0    0.009467 -768.844028 -749.331586
2  2  1  0    0.009540 -770.904835 -754.179884
   3  0  0    0.009773 -760.248885 -743.523935
   2  0  0    0.009986 -758.775827 -744.838368
```

We re-estimate a SARIMA(2, 0 ,3) x (1, 0, 0) model, as follows:

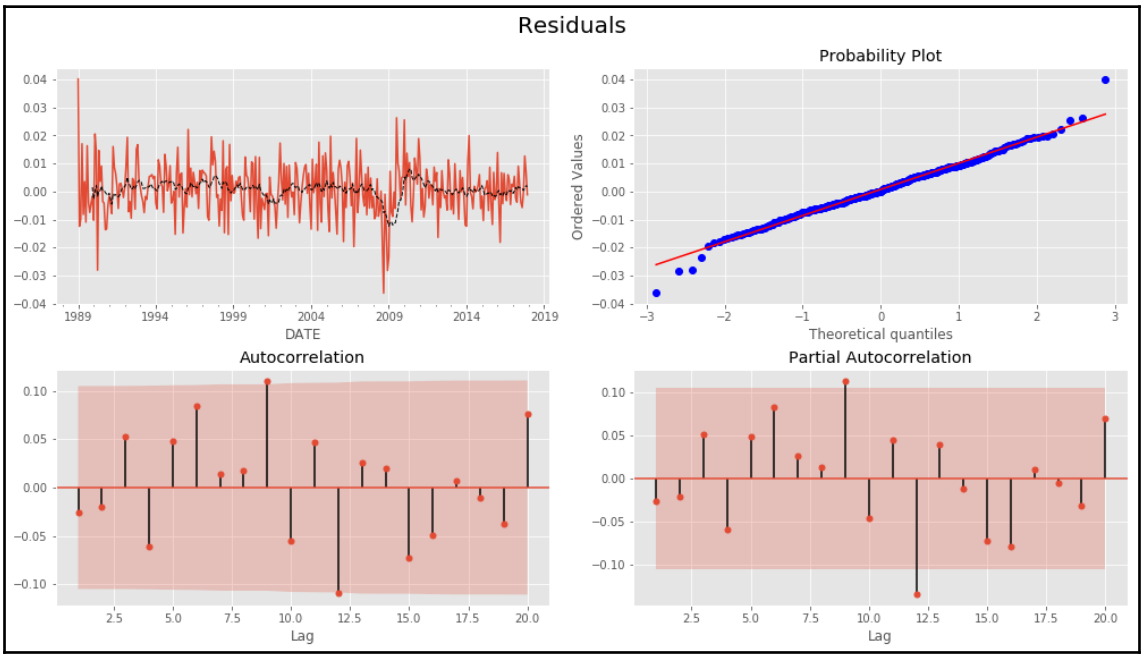
```
best_model = tsa.SARIMAX(endog=industrial_production_log_diff, order=(2, 0,
3),
                         seasonal_order=(1, 0, 0, 12)).fit()
print(best_model.summary())
```

We obtain the following summary:

```
                                Statespace Model Results
==========================================================================================
Dep. Variable:                                IPGMFN   No. Observations:                  348
Model:             SARIMAX(2, 0, 3)x(1, 0, 0, 12)   Log Likelihood                1139.719
Date:                            Sat, 22 Sep 2018   AIC                           -2265.438
Time:                                    17:48:17   BIC                           -2238.472
Sample:                                01-01-1989   HQIC                          -2254.702
                                     - 12-01-2017
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          1.4934      0.104     14.351      0.000       1.289       1.697
ar.L2         -0.5159      0.102     -5.083      0.000      -0.715      -0.317
ma.L1         -0.5499      0.114     -4.813      0.000      -0.774      -0.326
ma.L2          0.2872      0.062      4.662      0.000       0.166       0.408
ma.L3          0.1815      0.070      2.589      0.010       0.044       0.319
ar.S.L12      -0.4486      0.047     -9.533      0.000      -0.541      -0.356
sigma2      8.141e-05   5.65e-06     14.399      0.000    7.03e-05    9.25e-05
===================================================================================
Ljung-Box (Q):                       61.58   Jarque-Bera (JB):                 9.97
Prob(Q):                              0.02   Prob(JB):                         0.01
Heteroskedasticity (H):               1.07   Skew:                            -0.20
Prob(H) (two-sided):                  0.71   Kurtosis:                         3.73
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

The coefficients are significant, and the Q statistic rejects the hypothesis of further autocorrelation. The correlogram similarly indicates that we have successfully eliminated the series' autocorrelation:



How to use time series models to forecast volatility

A particularly important area of application for univariate time series models is the prediction of volatility. The volatility of financial time series is usually not constant over time but changes, with bouts of volatility clustering together. Changes in variance create challenges for time series forecasting using the classical ARIMA models. To address this challenge, we will now model volatility so that we can predict changes in variance.

Heteroskedasticity is the technical term for changes in a variable's variance. The **autoregressive conditional heteroskedasticity** (**ARCH**) model expresses the variance of the error term as a function of the errors in previous periods. More specifically, it assumes that the error variance follows an AR(p) model.

The **generalized autoregressive conditional heteroskedasticity** (**GARCH**) model broadens the scope to ARMA models. Time series forecasting often combines ARIMA models for the expected mean and ARCH/GARCH models for the expected variance of a time series. The 2003 Nobel Prize in Economics was awarded to Robert Engle and Clive Granger for developing this class of models. The former also runs the Volatility Lab at New York University's Stern School (see GitHub references) with numerous online examples and tools concerning the models we will discuss and their numerous extensions.

The autoregressive conditional heteroskedasticity (ARCH) model

The ARCH(p) model is simply an AR(p) model applied to the variance of the residuals of a time series model that makes this variance at time t conditional on lagged observations of the variance. More specifically, the error terms, ε_t, are residuals of a linear model, such as ARIMA, on the original time series and are split into a time-dependent standard deviation, σ_t, and a disturbance, z_t, as follows:

$$\text{ARCH}(p): \quad \text{var}(x_t) = \sigma_t^2 = \omega + \alpha_1 \epsilon_{t-1}^2 + \ldots + \alpha_p \epsilon_{t-p}^2, \quad \epsilon_t = \sigma_t z_t, \quad z_t \sim \text{i.i.d.}$$

An ARCH(p) model can be estimated using OLS. Engle proposed a method to identify the appropriate ARCH order using the Lagrange multiplier test that corresponds to the F-test of the hypothesis that all coefficients in linear regression are zero (see `Chapter 7`, *Linear Models*).

One strength of the model is that it produces volatility, estimates positive excess kurtosis—that is, fat tails relative to the normal distribution—which in turn is in line with empirical observations about returns. Weaknesses include that the model assumes the same effect for positive and negative volatility shocks because it depends on the square of the previous shocks, whereas asset prices are known to respond differently to positive and negative shocks. The ARCH model also does not offer new insight into the source of variations of a financial time series because it just mechanically describes the conditional variance. Finally, ARCH models are likely to overpredict the volatility because they respond slowly to large, isolated shocks to the return series.

For a properly-specified ARCH model, the standardized residuals (divided by the model estimate for the period of standard deviation) should resemble white noise and can be subjected to a Ljung-Box Q test.

Generalizing ARCH – the GARCH model

The ARCH model is relatively simple but often requires many parameters to capture the volatility patterns of an asset-return series. The **generalized ARCH** (**GARCH**) model applies to a log-return series, r_t, with disturbances, $\varepsilon_t = r_t - \mu$, that follow a GARCH(p, q) model if:

$$\epsilon_t = \sigma_t z_t, \quad \sigma_t^2 = \omega + \sum_{i=1}^{p} \alpha_i \epsilon_{t-i}^2 + \sum_{j=1}^{q} \beta_i \sigma_{t-j}^2, \quad z_t \sim \text{i.i.d.}$$

The GARCH(p, q) model assumes an ARMA(p, q) model for the variance of the error term, ε_t.

Similar to ARCH models, the tail distribution of a GARCH(1,1) process is heavier than that of a normal distribution. The model encounters the same weaknesses as the ARCH model. For instance, it responds equally to positive and negative shocks.

Selecting the lag order

To configure the lag order for ARCH and GARCH models, use the squared residuals of the time series trained to predict the mean of the original series. The residuals are zero-centered so that their squares are also the variance. Then inspect the ACF and PACF plots of the squared residuals to identify autocorrelation patterns in the variance of the time series.

How to build a volatility-forecasting model

The development of a volatility model for an asset-return series consists of four steps:

1. Build an ARMA time series model for the financial time series based on the serial dependence revealed by the ACF and PACF.
2. Test the residuals of the model for ARCH/GARCH effects, again relying on the ACF and PACF for the series of the squared residual.
3. Specify a volatility model if serial correlation effects are significant, and jointly estimate the mean and volatility equations.
4. Check the fitted model carefully and refine it if necessary.

When applying volatility forecasting to return series, the serial dependence may be limited so that a constant mean may be used instead of an ARMA model.

The `arch` library provides several options to estimate volatility-forecasting models. It offers several options to model the expected mean, including a constant mean, the AR(p) model discussed in the section on univariate time series models above as well as more recent heterogeneous autoregressive processes (HAR) that use daily (1 day), weekly (5 days), and monthly (22 days) lags to capture the trading frequencies of short-, medium-, and long-term investors.

The mean models can be jointly defined and estimated with several conditional heteroskedasticity models that include, in addition to ARCH and GARCH, the **exponential GARCH** (**EGARCH**) model, which allows for asymmetric effects between positive and negative returns and the **heterogeneous ARCH** (**HARCH**) model, which complements the HAR mean model.

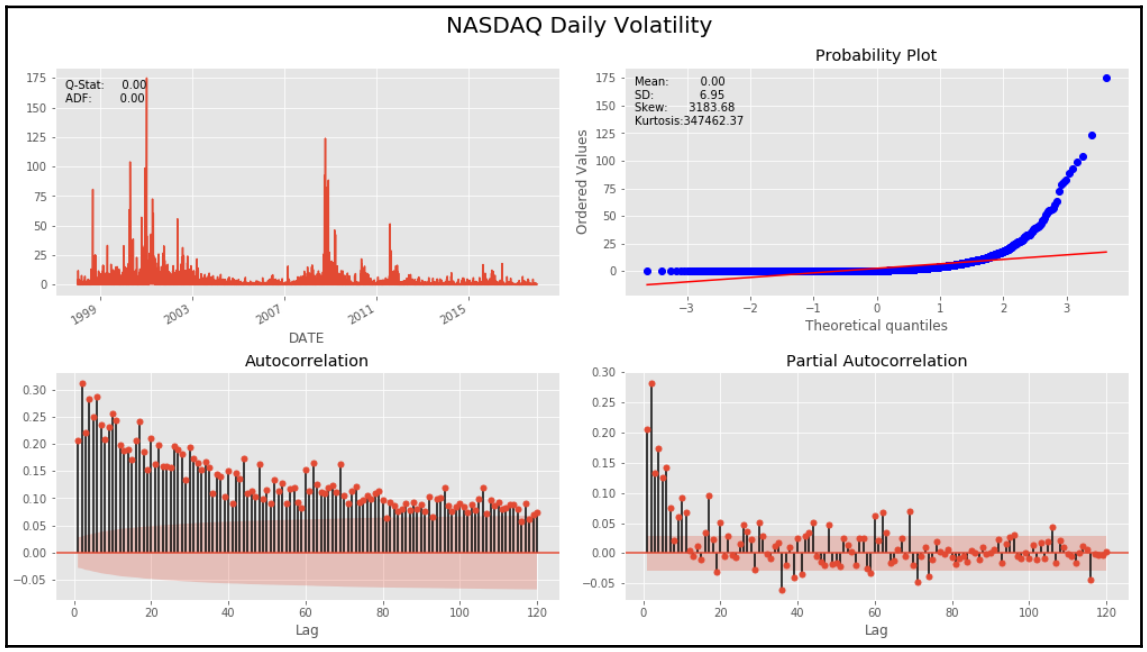
We will use daily NASDAQ returns from 1998-2017 to demonstrate the usage of a GARCH model (see the notebook `arch_garch_models` for details):

```
nasdaq = web.DataReader('NASDAQCOM', 'fred', '1998',
'2017-12-31').squeeze()
nasdaq_returns = np.log(nasdaq).diff().dropna().mul(100) # rescale to
facilitate optimization
```

The rescaled daily return series exhibits only limited autocorrelation, but the squared deviations from the mean do have substantial memory reflected in the slowly-decaying ACF and the PACF high for the first two and cutting off only after the first six lags:

```
plot_correlogram(nasdaq_returns.sub(nasdaq_returns.mean()).pow(2),
lags=120, title='NASDAQ Daily Volatility')
```

The function `plot_correlogram` produces the following output:



Hence, we can estimate a GARCH model to capture the linear relationship of past volatilities. We will use rolling 10-year windows to estimate a GARCH(p, q) model with p and q ranging from 1-4 to generate 1-step out-of-sample forecasts. We then compare the RMSE of the predicted volatility relative to the actual squared deviation of the return from its mean to identify the most predictive model. We are using winsorized data to limit the impact of extreme return values reflected in the very high positive skew of the volatility:

```
trainsize = 10 * 252  # 10 years
data = nasdaq_returns.clip(lower=nasdaq_returns.quantile(.05),
                           upper=nasdaq_returns.quantile(.95))
T = len(nasdaq_returns)
test_results = {}
for p in range(1, 5):
    for q in range(1, 5):
        print(f'{p} | {q}')
        result = []
        for s, t in enumerate(range(trainsize, T-1)):
            train_set = data.iloc[s: t]
            test_set = data.iloc[t+1]  # 1-step ahead forecast
            model = arch_model(y=train_set, p=p, q=q).fit(disp='off')
            forecast = model.forecast(horizon=1)
            mu = forecast.mean.iloc[-1, 0]
            var = forecast.variance.iloc[-1, 0]
            result.append([(test_set-mu)**2, var])
        df = pd.DataFrame(result, columns=['y_true', 'y_pred'])
        test_results[(p, q)] = np.sqrt(mean_squared_error(df.y_true,
df.y_pred))
```

The GARCH(2, 2) model achieves the lowest RMSE (same value as GARCH(4, 2) but with fewer parameters), so we go ahead and estimate this model to inspect the summary:

```
am = ConstantMean(nasdaq_returns.clip(lower=nasdaq_returns.quantile(.05),
                                      upper=nasdaq_returns.quantile(.95)))
am.volatility = GARCH(2, 0, 2)
am.distribution = Normal()
model = am.fit(update_freq=5)
print(model.summary())
```

The output shows the maximized log-likelihood as well as the AIC and BIC criteria that are commonly minimized when selecting models based on in-sample performance (see `Chapter 7`, *Linear Models*). It also displays the result for the mean model, which in this case is just a constant estimate, as well as the GARCH parameters for the constant omega, the AR parameters, α, and the MA parameters, β, all of which are statistically significant:

```
                     Constant Mean - GARCH Model Results
==============================================================================
Dep. Variable:              NASDAQCOM   R-squared:                      -0.001
Mean Model:             Constant Mean   Adj. R-squared:                 -0.001
Vol Model:                      GARCH   Log-Likelihood:               -7484.02
Distribution:                  Normal   AIC:                           14980.0
Method:            Maximum Likelihood   BIC:                           15019.0
                                        No. Observations:                 4852
Date:                Sun, Sep 23 2018   Df Residuals:                     4846
Time:                        15:43:41   Df Model:                            6
                            Mean Model
==============================================================================
                 coef    std err          t      P>|t|      95.0% Conf. Int.
------------------------------------------------------------------------------
mu             0.0521  1.491e-02      3.491  4.804e-04 [2.284e-02,8.130e-02]
                          Volatility Model
==============================================================================
                 coef    std err          t      P>|t|       95.0% Conf. Int.
------------------------------------------------------------------------------
omega          0.0196  8.287e-03      2.365  1.804e-02  [3.354e-03,3.584e-02]
alpha[1]       0.0247  1.470e-02      1.678  9.340e-02 [-4.148e-03,5.346e-02]
alpha[2]       0.0627  2.196e-02      2.853  4.324e-03    [1.962e-02,  0.106]
beta[1]        0.5648      0.181      3.120  1.806e-03      [  0.210,  0.920]
beta[2]        0.3337      0.180      1.853  6.393e-02   [-1.932e-02,  0.687]
==============================================================================

Covariance estimator: robust
```

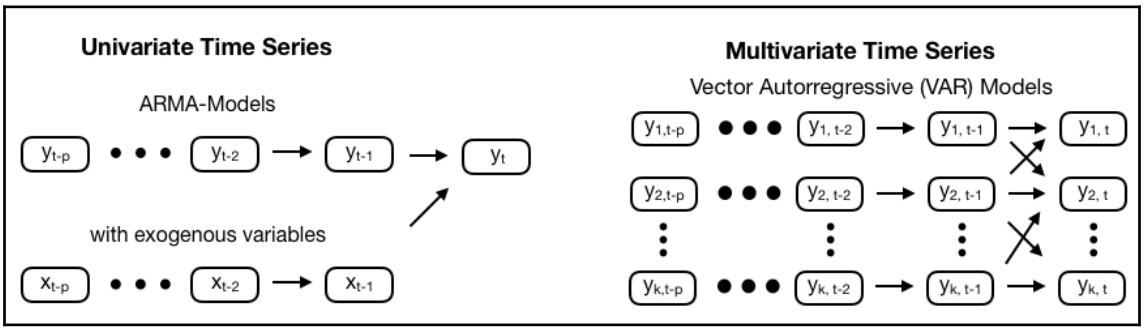
Let's now explore models for multiple time series and the concept of cointegration, which will enable a new trading strategy.

Multivariate time series models

Multivariate time series models are designed to capture the dynamic of multiple time series simultaneously and leverage dependencies across these series for more reliable predictions.

Systems of equations

Univariate time series models like the ARMA approach, we just discussed are limited to statistical relationships between a target variable and its lagged values or lagged disturbances and exogenous series in the ARMAX case. In contrast, multivariate time series models also allow for lagged values of other time series to affect the target. This effect applies to all series, resulting in complex interactions, as illustrated in the following diagram:



In addition to potentially better forecasting, multivariate time series are also used to gain insights into cross-series dependencies. For example, in economics, multivariate time series are used to understand how policy changes to one variable, for example, an interest rate, may affect other variables over different horizons. The impulse-response function produced by the multivariate model we will look at serves this purpose and allows us to simulate how one variable responds to a sudden change in other variables. The concept of Granger causality analyzes whether one variable is useful in forecasting another (in the least squares sense). Furthermore, multivariate time series models allow for a decomposition of the prediction error variance to analyze how other series contribute.

The vector autoregressive (VAR) model

We will see how the vector autoregressive VAR(p) model extends the AR(p) model to k series by creating a system of k equations where each contains p lagged values of all k series. In the simplest case, a VAR(1) model for *k=2* takes the following form:

$$y_{1,t} = c_1 + a_{1,1}y_{1,t-1} + a_{1,2}y_{2,t-1} + \epsilon_{1,t}$$
$$y_{2,t} = c_2 + a_{2,1}y_{1,t-1} + a_{2,2}y_{2,t-1} + \epsilon_{1,t}$$

This model can be expressed somewhat more concisely in matrix form:

$$\begin{bmatrix} y_{1,t} \\ y_{2,t} \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} + \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix} \begin{bmatrix} y_{1,t-1} \\ y_{2,t-1} \end{bmatrix} + \begin{bmatrix} \epsilon_{1,t} \\ \epsilon_{2,t} \end{bmatrix}$$

The coefficients on the own lags provide information about the dynamics of the series itself, whereas the cross-variable coefficients offer some insight into the interactions across the series. This notation extends to the k series and order p, as follows:

$$\underset{k\times 1}{\boldsymbol{y}_t} = \underset{k\times 1}{\boldsymbol{c}} + \underset{k\times k}{\boldsymbol{A}_1}\underset{k\times 1}{\boldsymbol{y}_{t-1}} + \ldots + \underset{k\times k}{\boldsymbol{A}_p}\underset{k\times 1}{\boldsymbol{y}_{t-p}} + \underset{k\times 1}{\boldsymbol{\epsilon}_t}$$

VAR(p) models also require stationarity, so that the initial steps from univariate time series modeling carry over. First, explore the series and determine the necessary transformations, then apply the Augmented Dickey-Fuller test to verify that the stationarity criterion is met for each series and apply further transformations otherwise. It can be estimated with OLS conditional on initial information or with maximum likelihood, which is equivalent for normally-distributed errors but not otherwise.

If some or all of the k series are unit-root non-stationary, they may be co-integrated. This extension of the unit root concept to multiple time series means that a linear combination of two or more series is stationary and, hence, mean-reverting. The VAR model is not equipped to handle this case without differencing, instead use the Vector Error Correction model (VECM, see references on GitHub). We will further explore cointegration because, if present and assumed to persist, it can be leveraged for a pairs-trading strategy.

The determination of the lag order also takes its cues from the ACF and PACF for each series but is constrained by the fact that the same lag order applies to all series. After model estimation, residual diagnostics also call for a result resembling white noise, and model selection can use in-sample information criteria or, preferably, out-of-sample predictive performance to cross-validate alternative model designs if the ultimate goal is to use the model for prediction.

As mentioned in the univariate case, predictions of the original time series require us to reverse the transformations applied to make a series stationary before training the model.

How to use the VAR model for macro fundamentals forecasts

We will extend the univariate example of a single time series of monthly data on industrial production and add a monthly time series on consumer sentiment, both provided by the Federal Reserve's data service. We will use the familiar `pandas-datareader` library to retrieve data from 1970 through 2017:

```
df = web.DataReader(['UMCSENT', 'IPGMFN'], 'fred', '1970',
'2017-12').dropna()
df.columns = ['sentiment', 'ip']
```

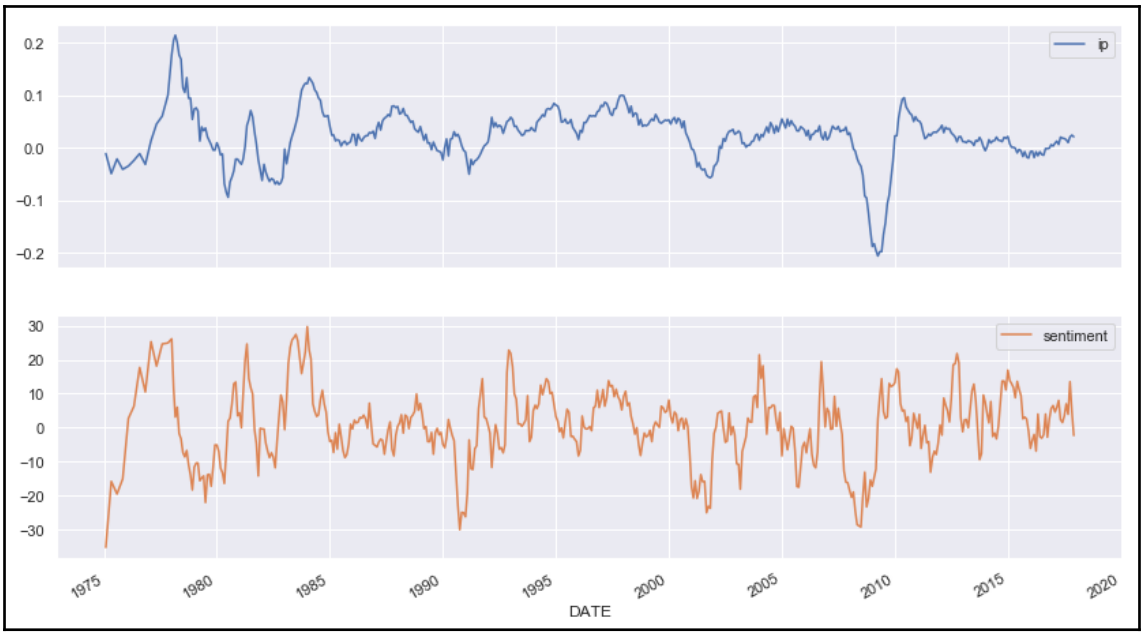
Log-transforming the industrial production series and seasonal differencing using lag 12 of both series yields stationary results:

```
df_transformed = pd.DataFrame({'ip': np.log(df.ip).diff(12),
                               'sentiment': df.sentiment.diff(12)}).dropna()

test_unit_root(df_transformed) # see notebook for details and additional
plots

           p-value
ip          0.0003
sentiment   0.0000
```

This leaves us with the following series:



To limit the size of the output, we will just estimate a VAR(1) model using the `statsmodels` `VARMAX` implementation (which allows for optional exogenous variables) with a constant trend using the first 480 observations:

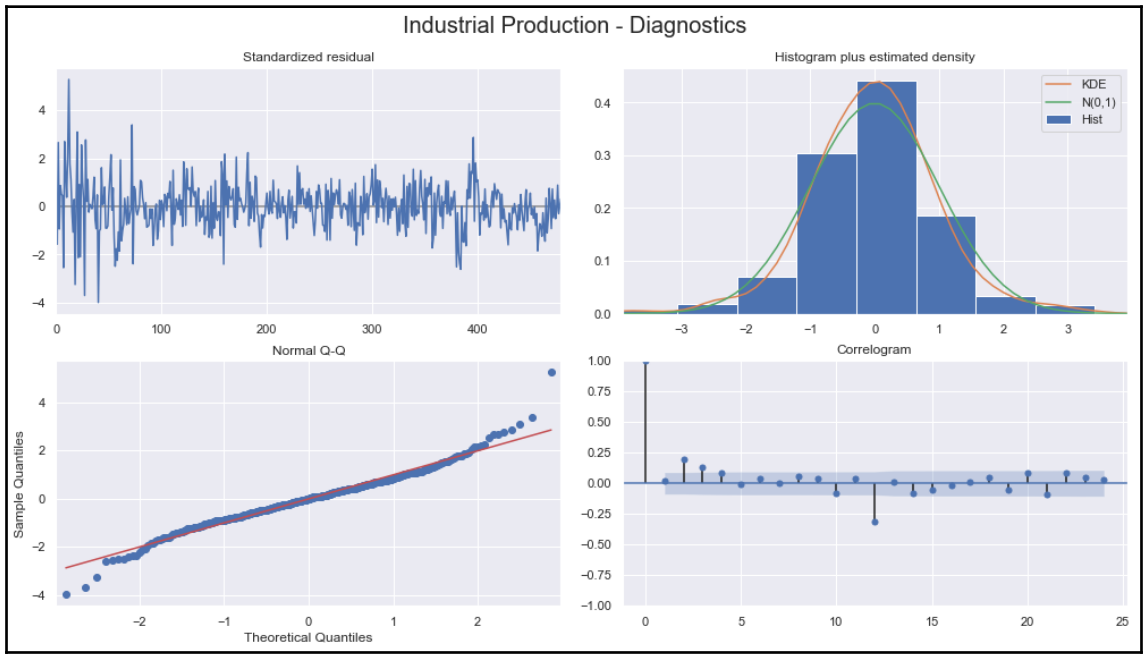
```
model = VARMAX(df_transformed.iloc[:480], order=(1,1),
trend='c').fit(maxiter=1000)
```

This results in the following summary:

```
                           Statespace Model Results
==============================================================================
Dep. Variable:     ['ip', 'sentiment']   No. Observations:                  480
Model:                      VARMA(1,1)   Log Likelihood                -68.938
                          + intercept   AIC                            163.875
Date:                 Sun, 23 Sep 2018   BIC                            218.134
Time:                         17:53:02   HQIC                           185.203
Sample:                              0
                                 - 480
Covariance Type:                   opg
===================================================================================
Ljung-Box (Q):              129.82, 165.15   Jarque-Bera (JB):        140.59, 16.05
Prob(Q):                      0.00, 0.00   Prob(JB):                   0.00, 0.00
Heteroskedasticity (H):       0.47, 1.10   Skew:                       0.19, 0.21
Prob(H) (two-sided):          0.00, 0.55   Kurtosis:                   5.62, 3.79
                            Results for equation ip
==============================================================================
                     coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0016      0.001      2.531      0.011       0.000       0.003
L1.ip          0.9276      0.010     95.539      0.000       0.909       0.947
L1.sentiment   0.0006   5.92e-05     10.283      0.000       0.000       0.001
L1.e(ip)       0.0095      0.037      0.259      0.796      -0.062       0.081
L1.e(sentiment)  -0.0001      0.000     -0.836      0.403      -0.000       0.000
                          Results for equation sentiment
==============================================================================
                     coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.3773      0.272      1.388      0.165      -0.155       0.910
L1.ip        -14.5753      5.375     -2.712      0.007     -25.109      -4.041
L1.sentiment   0.8795      0.023     37.840      0.000       0.834       0.925
L1.e(ip)      40.2063     18.695      2.151      0.032       3.565      76.847
L1.e(sentiment)   0.0411      0.051      0.800      0.424      -0.060       0.142
                            Error covariance matrix
==============================================================================
                             coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
sqrt.var.ip               0.0128      0.000     41.131      0.000       0.012       0.013
sqrt.cov.ip.sentiment     0.0309      0.229      0.135      0.893      -0.418       0.480
sqrt.var.sentiment        5.2713      0.147     35.759      0.000       4.982       5.560
==============================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

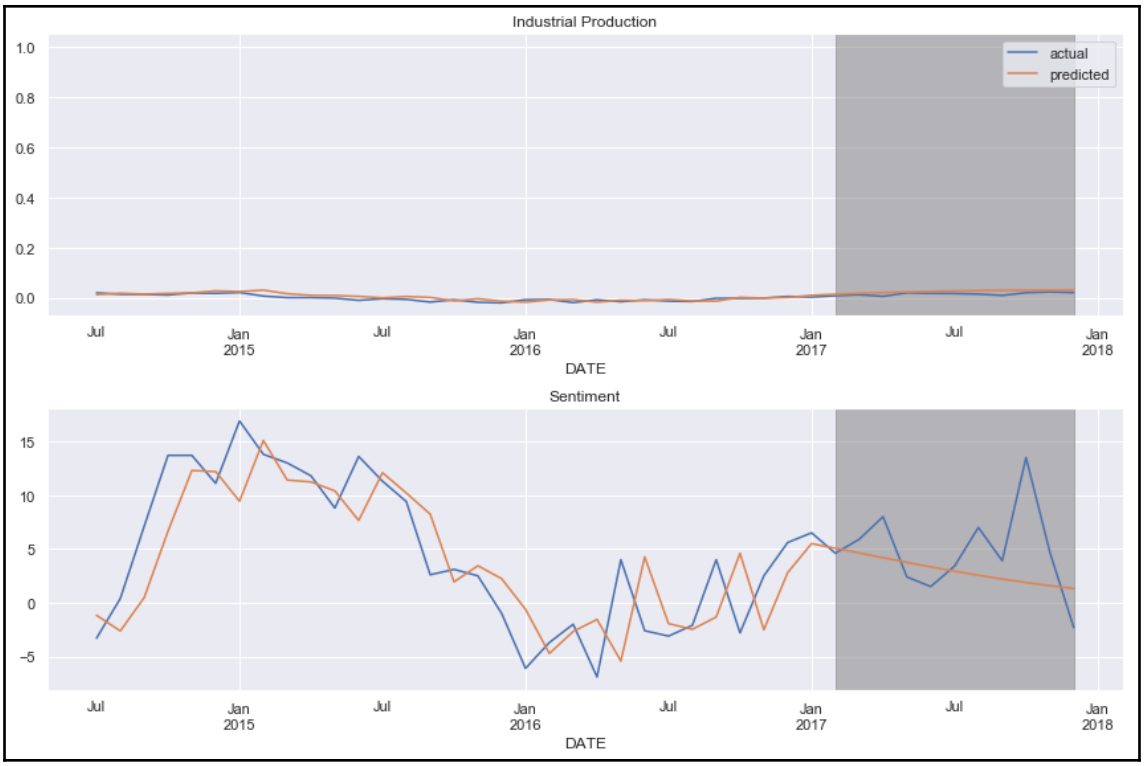
The output contains the coefficients for both time series equations, as outlined in the preceding VAR(1) illustration. statsmodels provides diagnostic plots to check whether the residuals meet the white noise assumptions, which are not exactly met in this simple case:



Out-of-sample predictions can be generated as follows:

```
preds = model.predict(start=480, end=len(df_transformed)-1)
```

A visualization of actual and predicted values shows how the prediction lags the actual values and does not capture non-linear out-of-sample patterns well:



Cointegration – time series with a common trend

The concept of an integrated multivariate series is complicated by the fact that all the component series of the process may be individually integrated but the process is not jointly integrated in the sense that one or more linear combinations of the series exist that produce a new stationary series.

In other words, a combination of two co-integrated series has a stable mean to which this linear combination reverts. A multivariate series with this characteristic is said to be co-integrated. This also applies when the individual series are integrated of a higher order and the linear combination reduces the overall order of integration.

cointegration is different from correlation: two series can be highly correlated but need not be co-integrated. For example, if two growing series are constant multiples of each other, their correlation will be high but any linear combination will also grow rather than revert to the mean.

The VAR analysis can still be applied to integrated processes using the error-correction form of a VAR model that uses the first differences of the individual series plus an error correction term in levels.

Testing for cointegration

There are two major approaches to testing for cointegration:

- The Engle–Granger two-step method
- The Johansen procedure

The Engle–Granger method involves regressing one series on another, and then applying an ADF unit-root test to the regression residual. If the null hypothesis can be rejected so that we assume the residuals are stationary, then the series are co-integrated. A key benefit of this approach is that the regression coefficient represents the multiplier that renders the combination stationary, that is, mean-reverting. We will return to this aspect when leveraging cointegration for a pairs-trading strategy. On the other hand, this approach is limited to identifying cointegration for pairs of series as opposed to larger groups of series.

The Johansen procedure, in contrast, tests the restrictions imposed by cointegration on a **vector autoregression** (**VAR**) model as discussed in the previous section. More specifically, after subtracting the target vector from both sides of the generic VAR(p) preceding equation, we obtain the **error correction model** (**ECM**) formulation:

$$\boldsymbol{\Delta y_t} = \boldsymbol{c} + \boldsymbol{\Pi y_{t-1}} + \boldsymbol{\Gamma_1 \Delta y_{t-1}} + \ldots + \boldsymbol{\Gamma_p \Delta y_{t-p}} + \boldsymbol{\epsilon}_t$$

The resulting modified VAR(p) equation has only one vector term in levels, that is, not expressed as difference using the operator, Δ. The nature of cointegration depends on the properties of the coefficient matrix, Π, of this term, in particular on its rank. While this equation appears structurally similar to the ADF test setup, there are now several potential constellations of common trends and orders of integration because there are multiple series involved. For details, see the references listed on GitHub, including with respect to practical challenges regarding the scaling of individual series.

How to use cointegration for a pairs-trading strategy

Pairs-trading relies on a stationary, mean-reverting relationship between two asset prices. In other words, the ratio or difference between the two prices, also called the spread, may over time diverge but should ultimately return to the same level. Given such a pair, the strategy consists of going long (that is, purchasing) the under-performing asset because it would require a period of outperformance to close the gap. At the same time, one would short the asset that has moved away from the price anchor in the positive direction to fund the purchase.

cointegration represents precisely this type of stable relationship between two price series anchored by a common mean. Assuming cointegration persists, convergence must ultimately ensue, either by the underperforming stock rising or the outperforming stock coming down. The strategy would be profitable regardless, which has the added advantage of being hedged against general market movements either way.

However, the spread will constantly change, sometimes widening and sometimes narrowing, or remain unchanged as both assets move in unison. The challenge of pairs-trading consists of maintaining a hedged position by adjusting the relative holdings as the spread changes.

In practice, given a universe of assets, a pairs-trading strategy will search for co-integrated pairs by running a statistical test on each pair. The key challenge here is to account for multiple testing biases, as outlined in `Chapter 6`, *Machine Learning Workflow*. The `statsmodels` library implements both the Engle-Granger cointegration test and the Johansen test.

In order to estimate the spread, run a linear regression to get the coefficient for the linear combination of two integrated asset price series that produce a stationary combined series. As mentioned, using linear regression to estimate the coefficient is known as the Engle-Granger test of cointegration.

Summary

In this chapter, we explored linear time series models for the univariate case of individual series as well as multivariate models for several interacting series. We encountered applications that predict macro fundamentals, models that forecast asset or portfolio volatility with widespread use in risk management, as well as multivariate VAR models that capture the dynamics of multiple macro series, as well as the concept of cointegration, which underpins the popular pair-trading strategy.

Similar to the previous chapter, we saw how linear models add a lot of structure to the model, that is, they make strong assumptions that potentially require transformations and extensive testing to verify that these assumptions are met. If they are, model-training and -interpretation is straightforward, and the models provide a good baseline case that more complex models may be able to improve on, as we will see in the following chapters.