# Chapter 6
# ARIMA Models

## 6.1 Overview

Creating a prediction model for time series data is actually quite simple. After all, one of the most important features is readily available: time. That is the foundation of ARIMA models as these models assume a linear relationship between past and future observations. ARIMA stands for Auto-regressive Integrated Moving Average and comprises of three different types of models. In this chapter we will discuss these components in more detail.

## 6.2 Autoregressive Models

When we fit a linear model between past and future observations of the same variable it is called autoregression. Each past observation is a predictor in this model which can be formulated as follows:

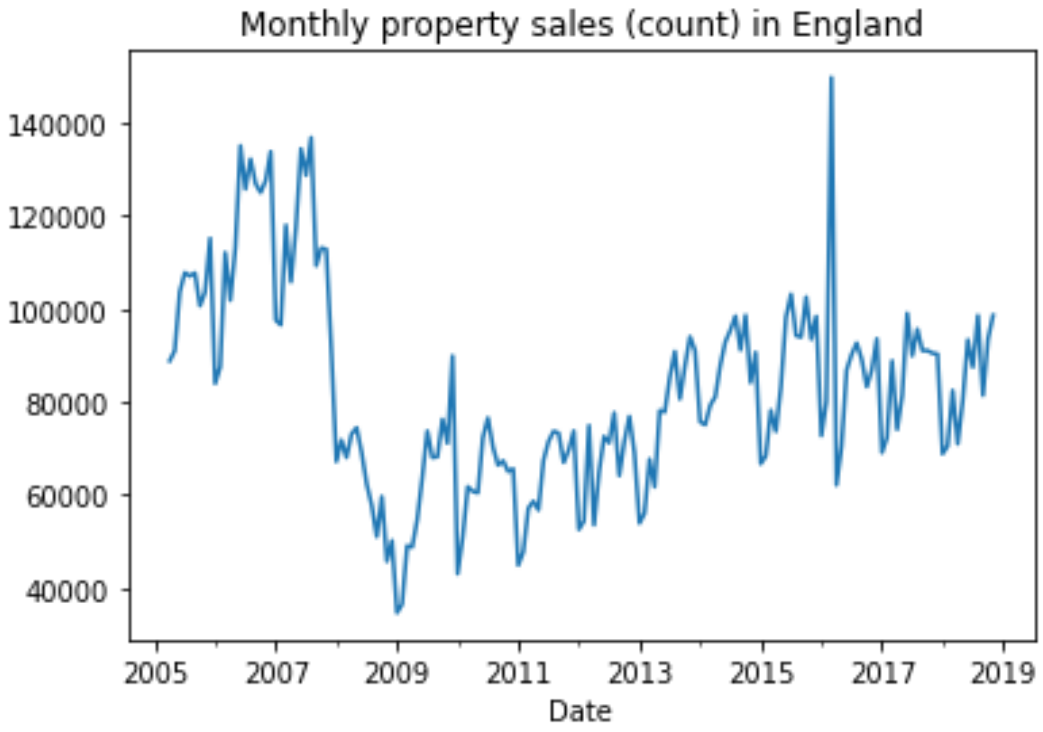
$$y_t = \alpha + \beta_1 y_{t-1} + \beta_2 y_{t-2} + \ldots + \beta_p y_{t-p} + \epsilon$$

Where $y$ are the observations, $\alpha$ is the estimated constant, $\beta$ is the estimated coefficient for each predictor and $\epsilon$ is the error which is assumed to be normally distributed with zero mean and $\sigma$ standard deviation (as is expected with linear regression).

The number of predictors, or the number of past observations, we include in our model is a parameter of the model and is called the *order*. For example, an autoregressive (AR) model of order 2 only considers the two preceding observations as predictors.

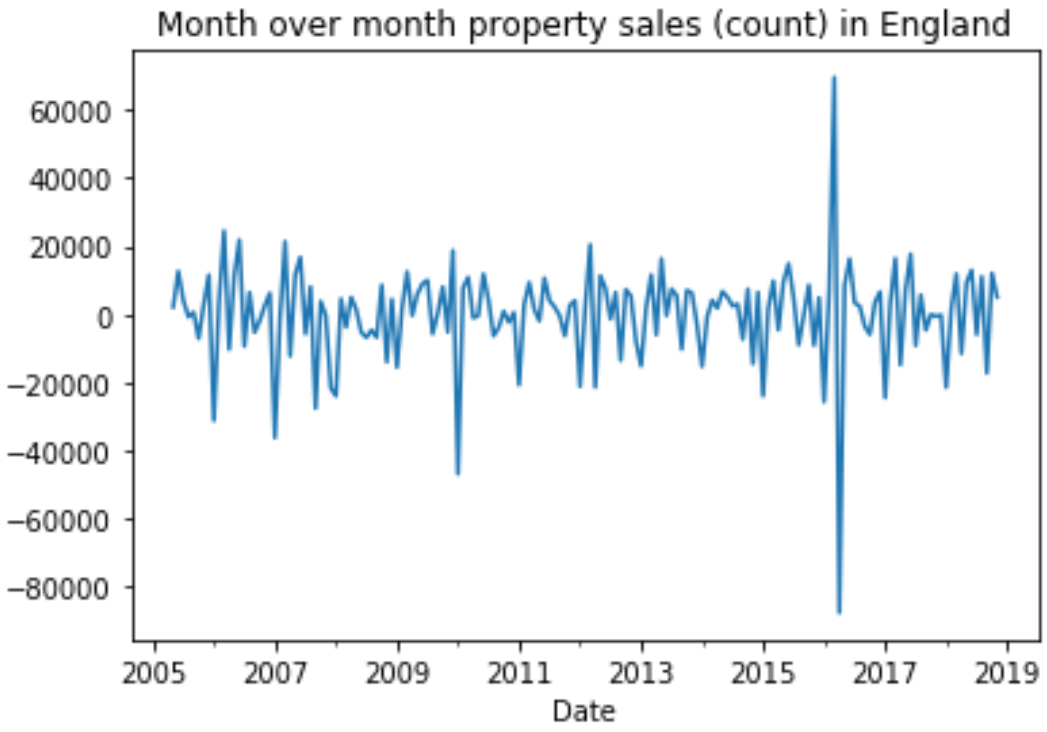
We will be fitting an AR model to the following example data:

```
>>> import pandas as pd
>>> from matplotlib.dates import AutoDateLocator
>>> property_sales = pd.read_csv("property_sales_uk.csv",
      parse_dates=["Date"], index_col=0)
>>> property_sales["England"].plot(title="Monthly property sales
(count) in England");
```



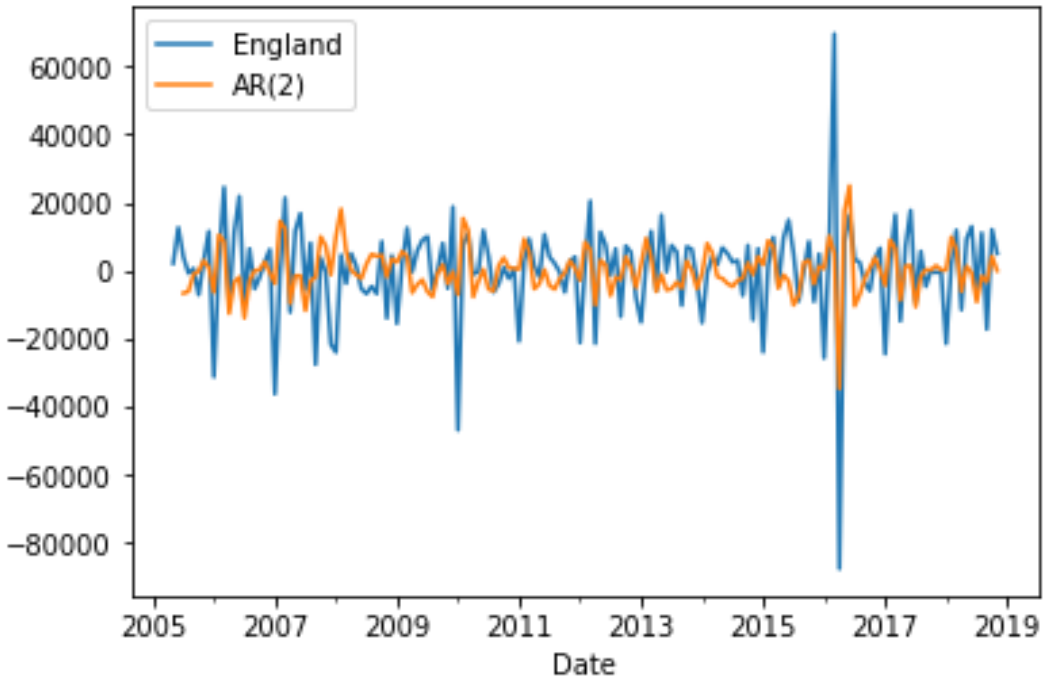

In our specification of an AR model we stated that $\epsilon$ must have constant mean and variance. This is another way of saying that the values must be *stationary*. From the above graph we can deduce that this time series is not stationary, there is both a trend and a seasonal effect. The quickest way of making a time series stationary is to look at differenced values.

```
>>> property_sales_diff = property_sales.diff().dropna()
>>> property_sales_diff["England"].plot(title="Month over month
property sales (count) in England");
```




There is now no clear trend or seasonal effect so we can proceed to fit an AR model. The `AR` class in the **statsmodels** package allows us to fit AR models using maximum likelihood estimation.

```
>>> from statsmodels.tsa.ar_model import AR
>>> import numpy as np
>>> ar = AR(property_sales_diff["England"].values)
>>> ar_model = ar.fit(maxlag=2)
>>> # have to add two NA values at the beginning as these are not
part of the fitted values
>>> fitted_values = np.hstack([[np.NaN, np.NaN],
ar_model.fittedvalues])
>>> property_sales_diff["AR(2)"] = fitted_values
>>> property_sales_diff[["England", "AR(2)"]].plot();
```



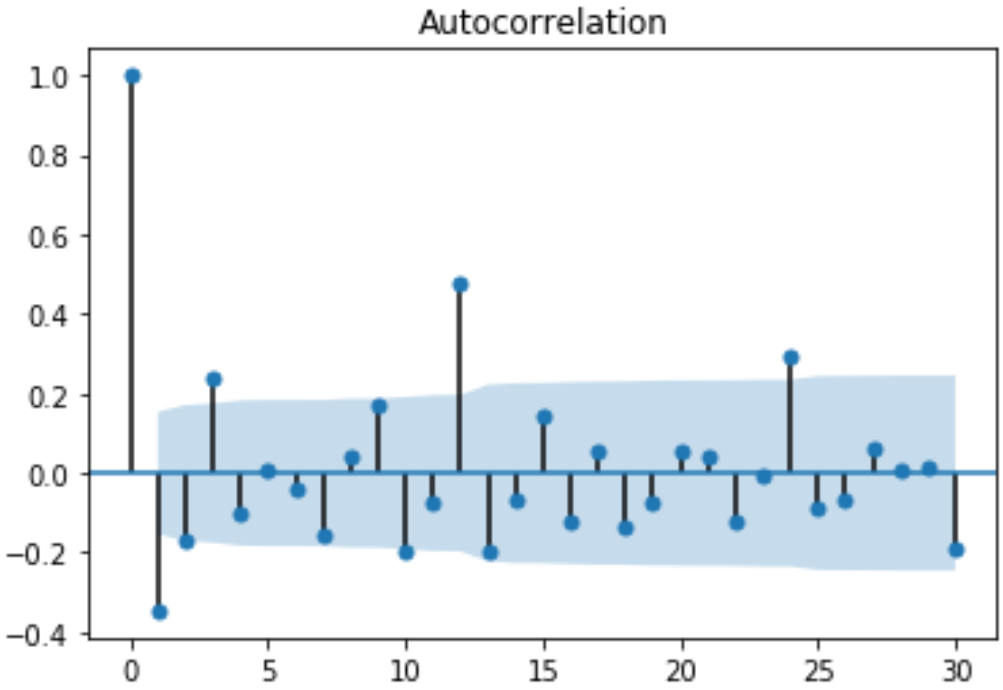

```
>>> ar_model.params

array([-39.68372786,  -0.4616542 ,  -0.32955574])
```

As we have fitted an AR(2) model in this example, the returned parameters are $\alpha$, $\beta_1$ and $\beta_2$. The $\beta_p$ parameters tell us about the dependence of the current value on one time point in the past and two time points in the past respectively. The estimated coefficients in this example suggest that there is an opposite relationship between future and past observations. The graph however shows the model fit isn't a very good one. There could be many reasons why the fit isn't good but the one that has the most effect is the AR order.

### 6.2.1 Determining the AR order

Rather than trying out different values for the order and assessing the fit of your model the basis of an AR model provides some guidance on determining the order. This basis is the relationship between past and future observations which we can quantify with the autocorrelation. We will utilise the `plot_acf` function to calculate the autocorrelation and plot it for us.

```
>>> from statsmodels.graphics.tsaplots import plot_acf
>>> acf = plot_acf(property_sales_diff['England'].values, lags=30)
```

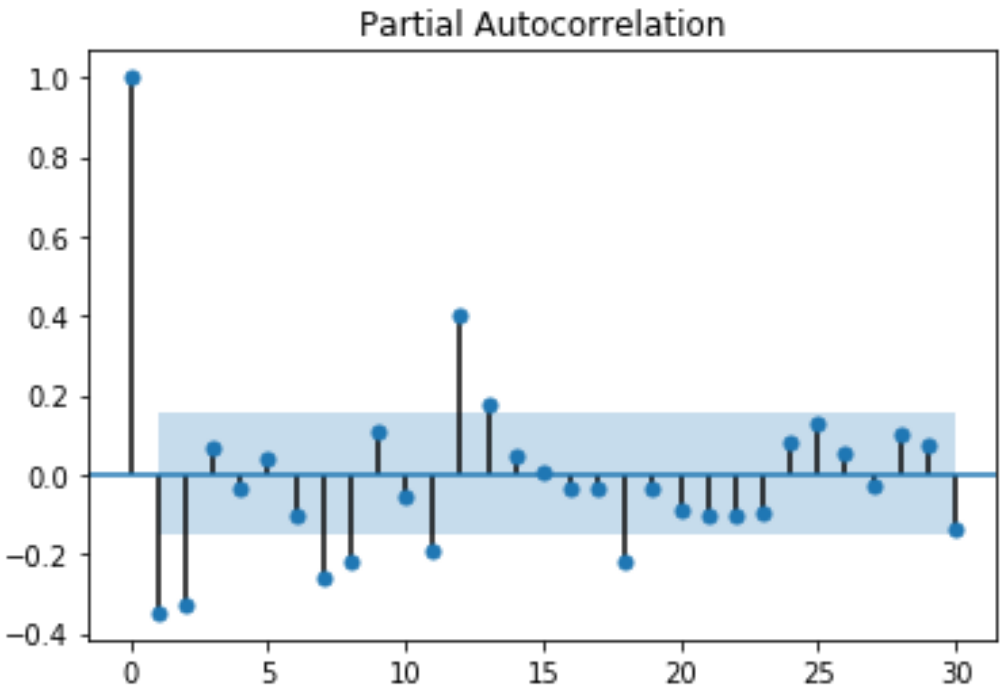



To determine the order of an AR model based on the autocorrelation plot we look at which lags have a significant correlation (value outside of the blue region). In the above plot that would be at lags 1, 2, 3 and 12 but that still doesn't tell us which one it should be.

One of the disadvantages of the autocorrelation is that when there is a correlation between ($y_t$, $y_{t-1}$) and between ($y_{t-1}$, $y_{t-2}$) it becomes hard to tell which correlation is real and which is a side effect. For example, there might be a correlation between $y_{t-2}$ and $y_t$ which is causing $y_{t-2}$ to correlate with $y_{t-1}$. One way to remove this side effect is to look at the partial autocorrelation.

The partial autocorrelation at lag k removes the correlation effect of any lags between $y_t$ and $y_{t-k}$.

```
>>> from statsmodels.graphics.tsaplots import plot_pacf
>>> pacf = plot_pacf(property_sales_diff['England'].values, lags=30)
```




Now we see there are several lags at which the partial autocorrelation is significant. Generally speaking, an AR(p) timeseries will generate a partial autocorrelation plot with a peak at lag p and decreasing correlations after lag p. The partial autocorrelation plot will show a peak only at lag p and insignificant correlations after lag p. Considering this the above plots suggest an AR model might not be the right choice.

Before we move on there is another way to determine the order, which is to fit models with different orders and then select based on a goodness-of-fit metric. The `AR.fit()` function provides this functionality when the order is not explicitly given.

```
>>> ar_model_opt = ar.fit(ic='aic') # Using Akaike Information
Criterion (AIC) as metric
>>> ar_model_opt.k_ar

13
```

According to the function the best value for the order is 13, although this is not supported by the autocorrelation plots. In the next sections we will see if other types of models are a better fit.

## 6.3 Moving Average Models

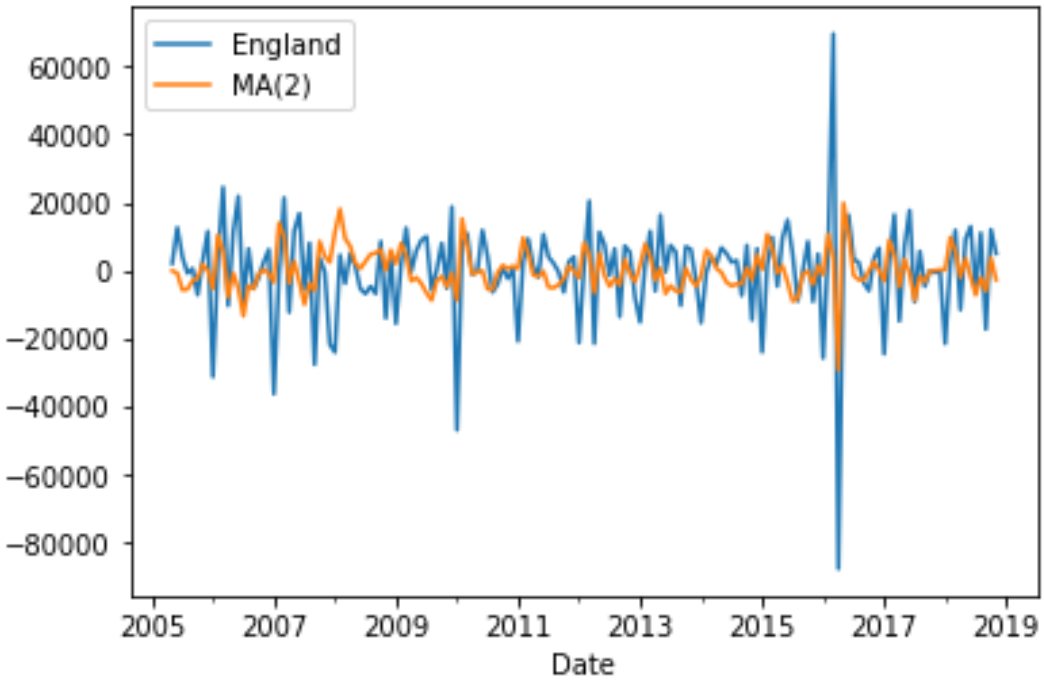
An alternative to autoregression is to not use past observations but instead use past errors as predictors. The errors are assumed to be independent and identically distributed (i.e. they come from the same distribution but are randomly generated, also known as *white noise*). This type of model is called a moving average (MA) model and its formulation is as follows:

$$y_t = \alpha + \beta_1 \epsilon_{t-1} + \beta_2 \epsilon_{t-2} + \ldots + \beta_q \epsilon_{t-q}$$

Where $y$ are again the observations, $\epsilon$ are the errors, $\alpha$ is the estimated constant and $\beta$ is the estimated coeffient for each error. A MA model also has an order which controls how many past errors we consider relevant. The **statsmodels** package does not provide a separate class for fitting MA models. Instead it combines AR models and MA models into the `ARMA` class and allows you to turn one or the other off.

A moving average model should not be confused with the moving average used for smoothing a time series. The latter is an aggregation over time while the former is a statistical model for the time series.

```
>>> from statsmodels.tsa.arima_model import ARMA
>>> arma = ARMA(property_sales_diff["England"].values, order=(0, 2))
>>> arma_model = arma.fit()
>>> property_sales_diff["MA(2)"] = arma_model.fittedvalues
>>> property_sales_diff[["England", "MA(2)"]].plot();
```




### *6.3.1 Determining the MA Order*

To determine the order of the AR model we looked at peaks in the partial autocorrelation plots. To determine the order of the MA model we do the same but now we're looking for the opposite. We look for peaks in the autocorrelation at lag q and insignificant values after lag q.

The correlation plots above don't exhibit this pattern either, suggesting that a MA model is also not a good fit. The autocorrelation patterns we described are great for identifying strictly AR models or strictly MA models but when it comes to ARMA models they are no longer useful.

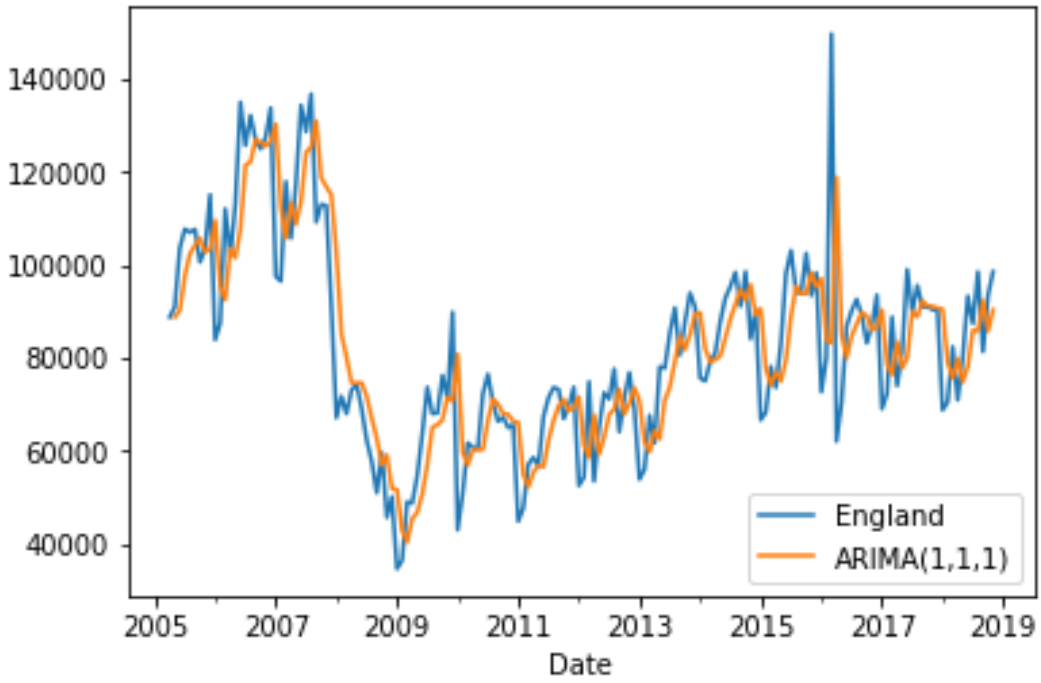
## 6.4 ARIMA models

When neither an AR model nor a MA model provides a good fit, it means the underlying process is more complicated. To allow a model to capture this complexity we can combine the AR and MA component, giving us an ARMA model. At the same time we mustn't forget that the time series we put into the model must be stationary.

Previously we solved this problem by differencing the time series, or "integrating". This is what the I in ARIMA stands for, it is the number of times we need to difference, or integrate, our time series to make it stationary. The formulation of an ARIMA model is simply a concatenation of the AR and MA models with the addition that the observations ($y$) are now differenced one or more times.

In the **statsmodels** package the ARIMA model can be fitted with the `ARIMA` class. The `order` argument accepts a value for the order of the AR model (p) followed by the number of times the series must be differenced (d) and finally the order of the MA model (q).

```
>>> from statsmodels.tsa.arima_model import ARIMA
>>> arima = ARIMA(property_sales["England"].values, order=(1,1,1))
>>> arima_model = arima.fit()
>>> # calling predict with typ='levels' will return predictions in
the same unit as the original series
>>> fitted_values = arima_model.predict(typ='levels')
>>>
>>> # we again end up with a shorter time series as some values are
used to initialize the model
>>> fill_values = [np.NaN]*(len(property_sales)-len(fitted_values))
>>> property_sales["ARIMA(1,1,1)"] = np.hstack([fill_values,
      fitted_values])
>>> property_sales[["England", "ARIMA(1,1,1)"]].plot();
```




### *6.4.1 Determining the ARIMA Order*

With ARIMA we now have three parameters we need to determine. The most important parameter is the integration order (d) as that produces the stationary time series. The simplest procedure to determine this parameter is to repeatedly difference the time series until it is stationary.

Stationarity can be assessed visually (check for trend/seasonality) or through statistical tests. The **statsmodels** package provides two of these *unit root tests*: `adfuller` (Augmented Dickey-Fuller) and `kpss` (Kwiatkowski-Phillips-Schmidt-Shin).

```
>>> from statsmodels.tsa.stattools import adfuller
>>> result = adfuller(property_sales["England"].values)
>>> result[1] # p-value

0.09280048697399007
```

In the Augmented Dickey-Fuller test the null hypothesis is that the data is non-stationary. With a p-value of 0.1 we cannot reject the null hypothesis (this doesn't mean the alternative is true). We will apply differencing again until we have a stationary time series.

```
>>> result = adfuller(property_sales_diff["England"].values)
>>> result[1]

0.050459481117940366

>>> property_sales_diff_two = property_sales_diff.diff().dropna()
>>> result =
adfuller(property_sales_diff_two["England"].diff().dropna().values)
>>> result[1]

5.789707896583707e-15
```

Now we can reject the null hypothesis so we stop differencing the time series. This means the integration order in our ARIMA model will be 2.

As a next step we could take the differenced series, look at its autocorrelation and partial autocorrelation plot and determine if an AR model or a MA model is more appropriate. Instead **statsmodels** provides the function `arma_order_select_ic` that finds the best model based on a given metric.

```
>>> from statsmodels.tsa.stattools import arma_order_select_ic
>>> result =
arma_order_select_ic(property_sales_diff_two["England"].values,
ic='aic')
>>> result.aic_min_order

(2, 1)
```

The **statsmodels** package also provides an interface to the popular X-13ARIMA modeling library, maintained by the US Census Bureau (https://www.census.gov/srd/www/x13as/). This includes the function `x13_arima_select_order` which automatically determines all three parameters of an ARIMA model.

## 6.5 Box-Jenkins approach

In *Time series analysis: Forecasting and control* by George Box and Gwilym Jenkins (1970), the authors proposed a step by step process to create ARIMA models. We have covered some aspects of this approach in previous sections but we will bring them together here.

### *6.5.1 Data analysis and wrangling*

The first step is analyse the time series to uncover any outliers or obvious patterns. In this part we also ensure the time series is stationary using a combination of differencing and unit root tests.

### *6.5.2 Model creation*

The next step is to create an ARIMA model with a specified or estimated order. We can find appropriate values for the order by analysing the autocorrelation and partial autocorrelation plots as we have done before. Alternatively, we can use **statsmodels** to automatically find the order for us.

### *6.5.3 Estimation of coefficients*

After we have decided on the order of the ARIMA model we must estimate its coefficients. This is the job of the `fit` function for each of the classes in **statsmodels**. If we let the package find the order this will automatically give us the estimates as well.

### *6.5.4 Diagnostics*

Similarly to a linear regression, we must check if our model does not violate any of the underlying assumptions. Most notably we must ensure there is no obvious pattern left in the residuals and they are in fact white noise.

We can do this either visually or through statistical methods (such as the Ljung-Box test). If any of the assumptions are not met then it's necessary to refine our model in step 2.

A possible final step is to apply the model to unseen data and generate new forecasts. The out-of-sample error would then provide an estimate of accuracy of the model.

1. Load the `air_passengers.csv` dataset and plot the time series.
2. Assign the last year from the dataset to a `test` variable and assign the remaining data to a `training` variable
3. Apply the Box-Jenkins approach to find the best model using the `training` data
4. Generate a forecast for the last year using the best model and calculate the out-of-sample error