



How to Create an ARIMA Model for Time Series Forecasting in Python

by **Jason Brownlee** on November 18, 2023 in **Time Series** 851

Share

Post

Share

A popular and widely used statistical method for time series forecasting is the ARIMA model.

ARIMA stands for AutoRegressive Integrated Moving Average and represents a cornerstone in time series forecasting. It is a statistical method that has gained immense popularity due to its efficacy in handling various standard temporal structures present in time series data.

In this tutorial, you will discover how to develop an ARIMA model for time series forecasting in Python.

After completing this tutorial, you will know:

- About the ARIMA model the parameters used and assumptions made by the model.
- How to fit an ARIMA model to data and use it to make forecasts.
- How to configure the ARIMA model on your time series problem.

Kick-start your project with my new book [Time Series Forecasting With Python](#), including *step-by-step tutorials* and the *Python source code* files for all examples.

Let's get started.

- **Updated Apr/2019:** Updated the link to dataset.
- **Updated Sep/2019:** Updated examples to use latest API.
- **Updated Dec/2020:** Updated examples to use latest API.
- **Updated Nov/2023:** #####

Autoregressive Integrated Moving Average Model

The **ARIMA** (AutoRegressive Integrated Moving Average) model stands as a statistical powerhouse for analyzing and forecasting time series data.

It explicitly caters to a suite of standard structures in time series data, and as such provides a simple yet powerful method for making skillful time series forecasts.

ARIMA is an acronym that stands for AutoRegressive Integrated Moving Average. It is a generalization of the simpler AutoRegressive Moving Average and adds the notion of integration.

Let's decode the essence of ARIMA:

- **AR** (Autoregression): This emphasizes the dependent relationship between an observation and its preceding or 'lagged' observations.
- **I** (Integrated): To achieve a stationary time series, one that doesn't exhibit trend or seasonality, differencing is applied. It typically involves subtracting an observation from its preceding observation.
- **MA** (Moving Average): This component zeroes in on the relationship between an observation and the residual error from a moving average model based on lagged observations.

Each of these components is explicitly specified in the model as a parameter. A standard notation is used for ARIMA(p,d,q) where the parameters are substituted with integer values to quickly indicate the specific ARIMA model being used.

The parameters of the ARIMA model are defined as follows:

- **p**: The lag order, representing the number of lag observations incorporated in the model.
- **d**: Degree of differencing, denoting the number of times raw observations undergo differencing.
- **q**: Order of moving average, indicating the size of the moving average window.

A linear regression model is constructed including the specified number and type of terms, and the data is prepared by a degree of differencing to make it stationary, i.e. to remove trend and seasonal structures that negatively affect the regression model.

Interestingly, any of these parameters can be set to 0. Such configurations enable the ARIMA model to mimic the functions of simpler models like ARMA, AR, I, or MA.

Adopting an ARIMA model for a time series assumes that the underlying process that generated the observations is an ARIMA process. This may seem obvious but helps to motivate the need to confirm the assumptions of the model in the raw observations and the residual errors of forecasts from the model.

Next, let's take a look at how we can use the ARIMA model in Python. We will start with loading a simple univariate time series.

Stop learning Time Series Forecasting the *slow way*!

Take my free 7-day email course and discover how to get started (with sample code).

Click to sign-up and also get a free PDF Ebook version of the course.

Start Your FREE Mini-Course Now!

Shampoo Sales Dataset

The Shampoo Sales dataset provides a snapshot of monthly shampoo sales spanning three years, resulting in 36 observations. Each observation is a sales count. The genesis of this dataset is attributed to Makridakis, Wheelwright, and Hyndman (1998).

Getting Started:

- [Download the dataset](#)
- Save it to your current working directory with the filename “shampoo-sales.csv”.

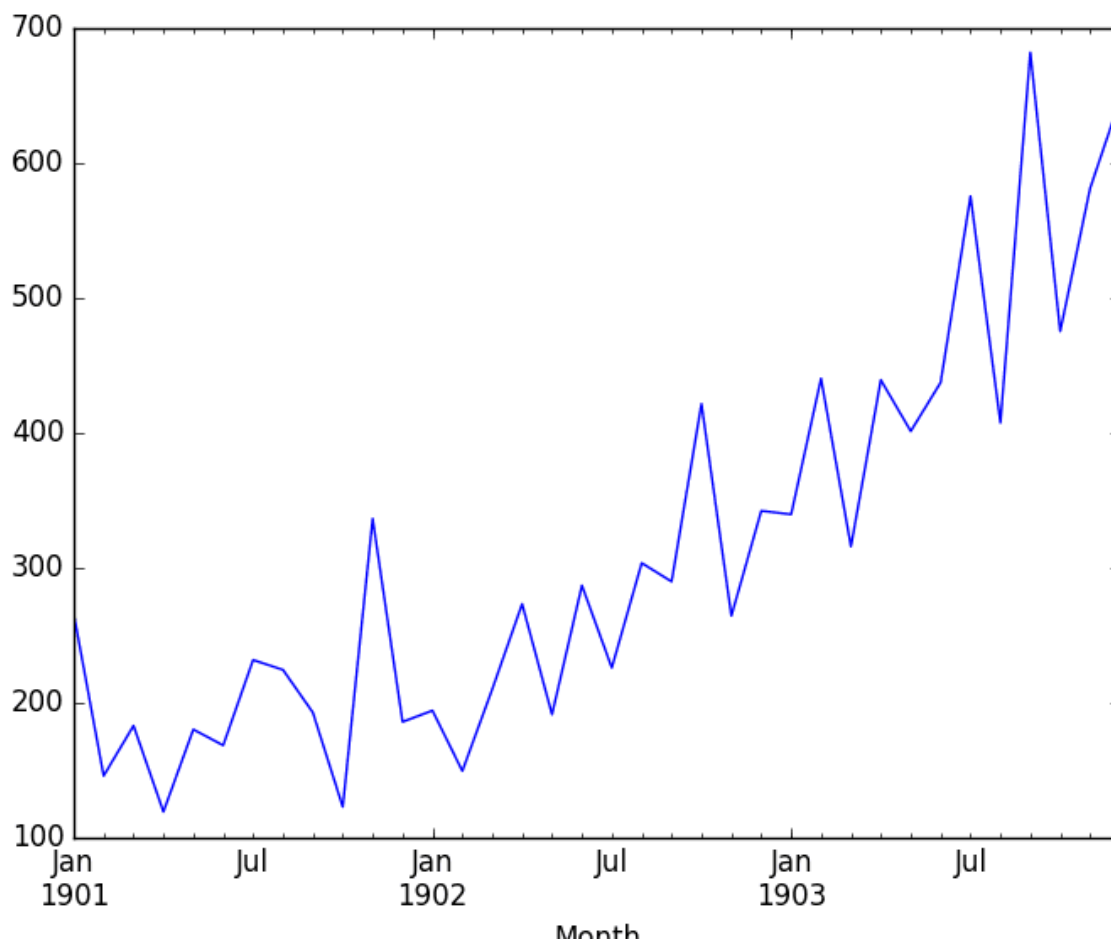
Loading and Visualizing the Dataset:

Below is an example of loading the Shampoo Sales dataset with Pandas with a custom function to parse the date-time field. The dataset is baselined in an arbitrary year, in this case 1900.

```
1 from pandas import read_csv
2 from pandas import datetime
3 from matplotlib import pyplot
4
5 def parser(x):
6     return datetime.strptime('190'+x, '%Y-%m')
7
8 series = read_csv('shampoo-sales.csv', header=0, parse_dates=[0], index_col=0, squeeze=True)
9 print(series.head())
10 series.plot()
11 pyplot.show()
```

When executed, this code snippet will display the initial five dataset entries:

```
1 Month
2 1901-01-01 266.0
3 1901-02-01 145.9
4 1901-03-01 183.1
5 1901-04-01 119.3
6 1901-05-01 180.3
7 Name: Sales, dtype: float64
```



Shampoo Sales Dataset Plot

The data is also plotted as a time series with the month along the x-axis and sales figures on the y-axis.

We can see that the Shampoo Sales dataset has a clear trend. This suggests that the time series is not stationary and will require differencing to make it stationary, at least a difference order of 1.

Pandas offers a built-in capability to plot autocorrelations. The following example showcases the autocorrelation for an extensive set of time series lags:

```
1 from pandas import read_csv
2 from pandas import datetime
```

```
3 from matplotlib import pyplot
4 from pandas.plotting import autocorrelation_plot
5
6 def parser(x):
7     return datetime.strptime('190'+x, '%Y-%m')
8
9 series = read_csv('shampoo-sales.csv', header=0, parse_dates=[0], index_col=0, squeeze=True)
10 autocorrelation_plot(series)
11 pyplot.show()
```

Running the example, we can see that there is a positive correlation with the first 10-to-12 lags that is perhaps significant for the first 5 lags.

This provides a hint: initiating the AR parameter of our model with a value of 5 could be a beneficial starting point.

Autocorrelation Plot of Shampoo Sales Data

ARIMA with Python

The statsmodels library stands as a vital tool for those looking to harness the power of ARIMA for time series forecasting in Python.

Building an ARIMA Model: A Step-by-Step Guide:

1. **Model Definition:** Initialize the ARIMA model by invoking `ARIMA()` and specifying the `p`, `d`, and `q` parameters.
2. **Model Training:** Train the model on your dataset using the `fit()` method.
3. **Making Predictions:** Generate forecasts by utilizing the `predict()` function and designating the desired time index or indices.

Let's start with something simple. We will fit an ARIMA model to the entire Shampoo Sales dataset and review the residual errors.

We'll employ the ARIMA(5,1,0) configuration:

- 5 lags for autoregression (AR)
- 1st order differencing (I)
- No moving average term (MA)

```
1 # fit an ARIMA model and plot residual errors
2 from pandas import datetime
3 from pandas import read_csv
4 from pandas import DataFrame
5 from statsmodels.tsa.arima.model import ARIMA
6 from matplotlib import pyplot
7 # load dataset
8 def parser(x):
9     return datetime.strptime('190'+x, '%Y-%m')
10 series = read_csv('shampoo-sales.csv', header=0, index_col=0, parse_dates=True, squeeze=True)
11 series.index = series.index.to_period('M')
12 # fit model
13 model = ARIMA(series, order=(5,1,0))
14 model_fit = model.fit()
15 # summary of fit model
16 print(model_fit.summary())
17 # line plot of residuals
18 residuals = DataFrame(model_fit.resid)
19 residuals.plot()
20 pyplot.show()
21 # density plot of residuals
22 residuals.plot(kind='kde')
23 pyplot.show()
24 # summary stats of residuals
25 print(residuals.describe())
```

Running the example prints a summary of the fit model. This summarizes the coefficient values used as well as the skill of the fit on the on the in-sample observations.

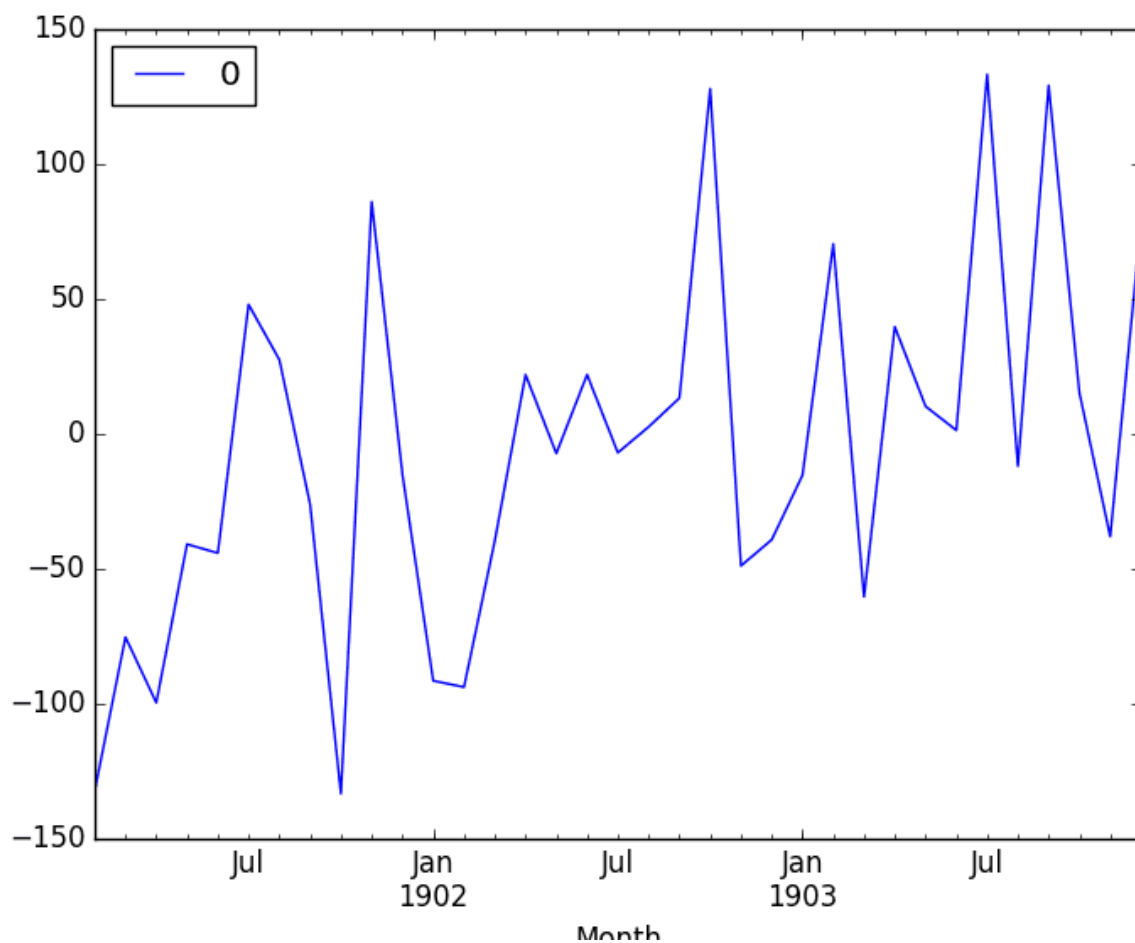
SARIMAX Results			
=====			
Dep. Variable:	Sales	No. Observations:	36
Model:	ARIMA(5, 1, 0)	Log Likelihood	-198.485
Date:	Thu, 10 Dec 2020	AIC	408.969

```

6 Time: 09:15:01 BIC 418.301
7 Sample: 01-31-1901 HQIC 412.191
8 - 12-31-1903
9 Covariance Type: opg
10 =====
11 coef std err z P>|z| [0.025 0.975]
12 -----
13 ar.L1 -0.9014 0.247 -3.647 0.000 -1.386 -0.417
14 ar.L2 -0.2284 0.268 -0.851 0.395 -0.754 0.298
15 ar.L3 0.0747 0.291 0.256 0.798 -0.497 0.646
16 ar.L4 0.2519 0.340 0.742 0.458 -0.414 0.918
17 ar.L5 0.3344 0.210 1.593 0.111 -0.077 0.746
18 sigma2 4728.9608 1316.021 3.593 0.000 2149.607 7308.314
19 =====
20 Ljung-Box (L1) (Q): 0.61 Jarque-Bera (JB): 0.96
21 Prob(Q): 0.44 Prob(JB): 0.62
22 Heteroskedasticity (H): 1.07 Skew: 0.28
23 Prob(H) (two-sided): 0.90 Kurtosis: 2.41
24 =====

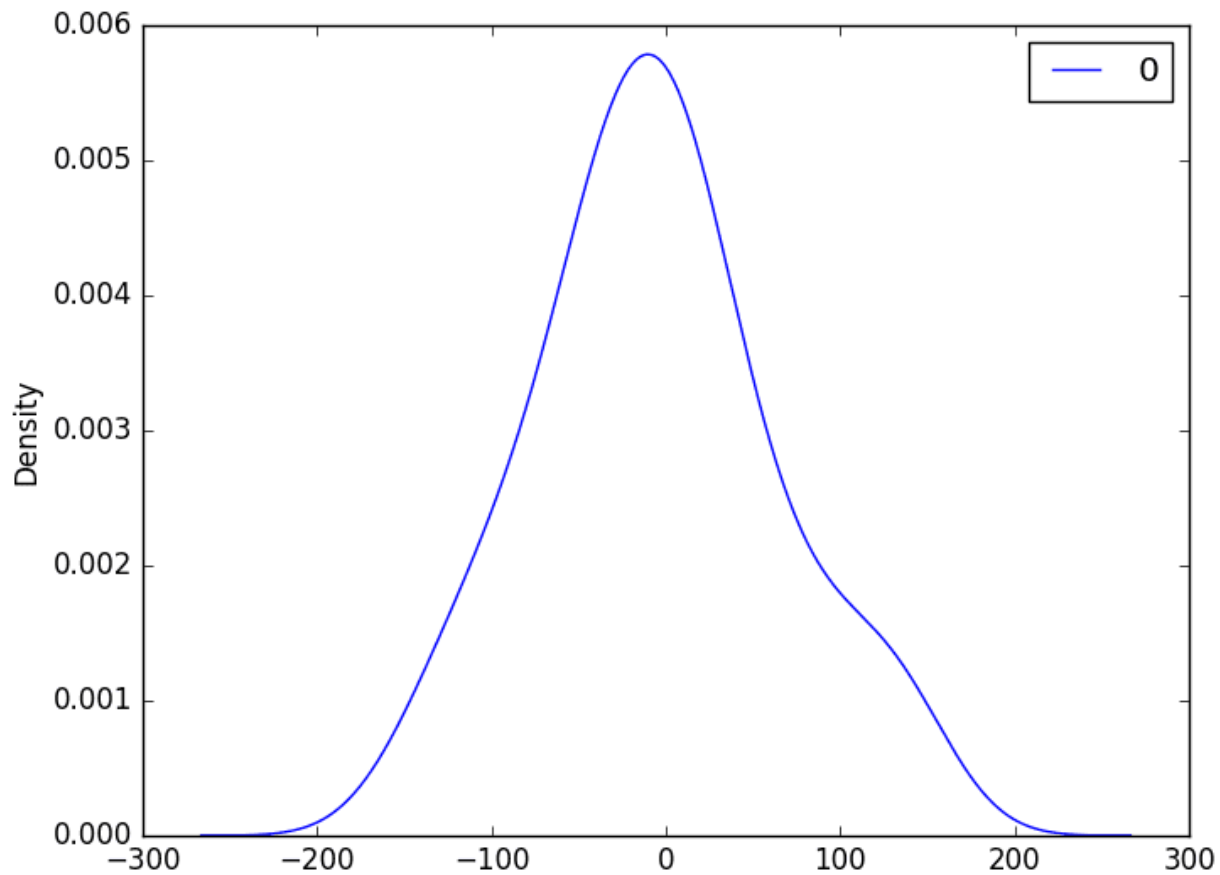
```

First, we get a line plot of the residual errors, suggesting that there may still be some trend information not captured by the model.



ARMA Fit Residual Error Line Plot

Next, we get a density plot of the residual error values, suggesting the errors are Gaussian, but may not be centred on zero.



ARMA Fit Residual Error Density Plot

The distribution of the residual errors is displayed. The results show that indeed there is a bias in the prediction (a non-zero mean in the residuals).

1	count	36.000000
2	mean	21.936144
3	std	80.774430
4	min	-122.292030
5	25%	-35.040859
6	50%	13.147219
7	75%	68.848286
8	max	266.000000

Note, that although we used the entire dataset for time series analysis, ideally we would perform this analysis on just the training dataset when developing a predictive model.

Next, let's look at how we can use the ARIMA model to make forecasts.