# **11 Classical Time Series Forecasting Methods in Python (Cheat Sheet)**

by [**Jason Brownlee**](https://machinelearningmastery.com/author/jasonb/) on August 6, 2018 in [**Time Series**](https://machinelearningmastery.com/category/time-series/)

Tweet Tweet **Share**

**Share**

Last Updated on December 10, 2020

Machine learning methods can be used for classification and forecasting on [time series](https://machinelearningmastery.com/time-series-forecasting/) problems.

Before exploring machine learning methods for time series, it is a good idea to ensure you have exhausted classical linear time series forecasting methods. Classical time series forecasting methods may be focused on linear relationships, nevertheless, they are sophisticated and perform well on a wide range of problems, assuming that your data is suitably prepared and the method is well configured.

In this post, will you will discover a suite of **classical methods for time series forecasting** that you can test on your forecasting problem prior to exploring to machine learning methods.

The post is structured as a cheat sheet to give you just enough information on each method to get started with a working code example and where to look to get more information on the method.

All code examples are in Python and use the Statsmodels library. The APIs for this library can be tricky for beginners (trust me!), so having a working code example as a starting point will greatly accelerate your progress.

This is a large post; you may want to bookmark it.

**Kick-start your project** with my new book [Time Series Forecasting With Python](https://machinelearningmastery.com/introduction-to-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/2020**: Changed AR to AutoReg due to API change.
* **Updated Dec/2020**: Updated ARIMA API to the latest version of statsmodels.



11 Classical Time Series Forecasting Methods in Python (Cheat Sheet)

Photo by [Ron Reiring](https://www.flickr.com/photos/84263554@N00/33076725455/), some rights reserved.

## **Overview**

This cheat sheet demonstrates 11 different classical time series forecasting methods; they are:

1. Autoregression (AR)
2. Moving Average (MA)
3. Autoregressive Moving Average (ARMA)
4. Autoregressive Integrated Moving Average (ARIMA)
5. Seasonal Autoregressive Integrated Moving-Average (SARIMA)
6. Seasonal Autoregressive Integrated Moving-Average with Exogenous Regressors (SARIMAX)
7. Vector Autoregression (VAR)
8. Vector Autoregression Moving-Average (VARMA)
9. Vector Autoregression Moving-Average with Exogenous Regressors (VARMAX)
10. Simple Exponential Smoothing (SES)
11. Holt Winter’s Exponential Smoothing (HWES)

*Did I miss your favorite classical time series forecasting method?*

Let me know in the comments below.

Each method is presented in a consistent manner.

This includes:

* **Description**. A short and precise description of the technique.
* **Python Code**. A short working example of fitting the model and making a prediction in Python.
* **More Information**. References for the API and the algorithm.

Each code example is demonstrated on a simple contrived dataset that may or may not be appropriate for the method. Replace the contrived dataset with your data in order to test the method.

Remember: each method will require tuning to your specific problem. In many cases, I have examples of how to configure and even grid search parameters on the blog already, try the search function.

If you find this cheat sheet useful, please let me know in the comments below.

## **Autoregression (AR)**

The autoregression (AR) method models the next step in the sequence as a linear function of the observations at prior time steps.

The notation for the model involves specifying the order of the model p as a parameter to the AR function, e.g. AR(p). For example, AR(1) is a first-order autoregression model.

The method is suitable for univariate time series without trend and seasonal components.

### **Python Code**

| 1  2  3  4  5  6  7  8  9  10  11 | # AR example  from statsmodels.tsa.ar\_model import AutoReg  from random import random  # contrived dataset  data = [x + random() for x in range(1, 100)]  # fit model  model = AutoReg(data, lags=1)  model\_fit = model.fit()  # make prediction  yhat = model\_fit.predict(len(data), len(data))  print(yhat) |
| --- | --- |

### **More Information**

* [statsmodels.tsa.ar\_model.AutoReg API](https://www.statsmodels.org/stable/generated/statsmodels.tsa.ar_model.AutoReg.html)
* [statsmodels.tsa.ar\_model.AutoRegResults API](https://www.statsmodels.org/stable/generated/statsmodels.tsa.ar_model.AutoRegResults.html)
* [Autoregressive model on Wikipedia](https://en.wikipedia.org/wiki/Autoregressive_model)

## **Moving Average (MA)**

The moving average (MA) method models the next step in the sequence as a linear function of the residual errors from a mean process at prior time steps.

A moving average model is different from calculating the moving average of the time series.

The notation for the model involves specifying the order of the model q as a parameter to the MA function, e.g. MA(q). For example, MA(1) is a first-order moving average model.

The method is suitable for univariate time series without trend and seasonal components.

### **Python Code**

We can use the ARIMA class to create an MA model and setting a zeroth-order AR model. We must specify the order of the MA model in the order argument.

| 1  2  3  4  5  6  7  8  9  10  11 | # MA example  from statsmodels.tsa.arima.model import ARIMA  from random import random  # contrived dataset  data = [x + random() for x in range(1, 100)]  # fit model  model = ARIMA(data, order=(0, 0, 1))  model\_fit = model.fit()  # make prediction  yhat = model\_fit.predict(len(data), len(data))  print(yhat) |
| --- | --- |

### **More Information**

* [Moving-average model on Wikipedia](https://en.wikipedia.org/wiki/Moving-average_model)

## **Autoregressive Moving Average (ARMA)**

The Autoregressive Moving Average (ARMA) method models the next step in the sequence as a linear function of the observations and residual errors at prior time steps.

It combines both Autoregression (AR) and Moving Average (MA) models.

The notation for the model involves specifying the order for the AR(p) and MA(q) models as parameters to an ARMA function, e.g. ARMA(p, q). An ARIMA model can be used to develop AR or MA models.

The method is suitable for univariate time series without trend and seasonal components.

### **Python Code**

| 1  2  3  4  5  6  7  8  9  10  11 | # ARMA example  from statsmodels.tsa.arima.model import ARIMA  from random import random  # contrived dataset  data = [random() for x in range(1, 100)]  # fit model  model = ARIMA(data, order=(2, 0, 1))  model\_fit = model.fit()  # make prediction  yhat = model\_fit.predict(len(data), len(data))  print(yhat) |
| --- | --- |

### **More Information**

* [Autoregressive–moving-average model on Wikipedia](https://en.wikipedia.org/wiki/Autoregressive%E2%80%93moving-average_model)

## **Autoregressive Integrated Moving Average (ARIMA)**

The Autoregressive Integrated Moving Average (ARIMA) method models the next step in the sequence as a linear function of the differenced observations and residual errors at prior time steps.

It combines both Autoregression (AR) and Moving Average (MA) models as well as a differencing pre-processing step of the sequence to make the sequence stationary, called integration (I).

The notation for the model involves specifying the order for the AR(p), I(d), and MA(q) models as parameters to an ARIMA function, e.g. ARIMA(p, d, q). An ARIMA model can also be used to develop AR, MA, and ARMA models.

The method is suitable for univariate time series with trend and without seasonal components.

### **Python Code**

| 1  2  3  4  5  6  7  8  9  10  11 | # ARIMA example  from statsmodels.tsa.arima.model import ARIMA  from random import random  # contrived dataset  data = [x + random() for x in range(1, 100)]  # fit model  model = ARIMA(data, order=(1, 1, 1))  model\_fit = model.fit()  # make prediction  yhat = model\_fit.predict(len(data), len(data), typ='levels')  print(yhat) |
| --- | --- |

### **More Information**

* [Autoregressive integrated moving average on Wikipedia](https://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average)

## **Seasonal Autoregressive Integrated Moving-Average (SARIMA)**

The [Seasonal Autoregressive Integrated Moving Average (SARIMA)](https://machinelearningmastery.com/sarima-for-time-series-forecasting-in-python/) method models the next step in the sequence as a linear function of the differenced observations, errors, differenced seasonal observations, and seasonal errors at prior time steps.

It combines the ARIMA model with the ability to perform the same autoregression, differencing, and moving average modeling at the seasonal level.

The notation for the model involves specifying the order for the AR(p), I(d), and MA(q) models as parameters to an ARIMA function and AR(P), I(D), MA(Q) and m parameters at the seasonal level, e.g. SARIMA(p, d, q)(P, D, Q)m where “m” is the number of time steps in each season (the seasonal period). A SARIMA model can be used to develop AR, MA, ARMA and ARIMA models.

The method is suitable for univariate time series with trend and/or seasonal components.

### **Python Code**

| 1  2  3  4  5  6  7  8  9  10  11 | # SARIMA example  from statsmodels.tsa.statespace.sarimax import SARIMAX  from random import random  # contrived dataset  data = [x + random() for x in range(1, 100)]  # fit model  model = SARIMAX(data, order=(1, 1, 1), seasonal\_order=(0, 0, 0, 0))  model\_fit = model.fit(disp=False)  # make prediction  yhat = model\_fit.predict(len(data), len(data))  print(yhat) |
| --- | --- |

### **More Information**

* [statsmodels.tsa.statespace.sarimax.SARIMAX API](http://www.statsmodels.org/dev/generated/statsmodels.tsa.statespace.sarimax.SARIMAX.html)
* [statsmodels.tsa.statespace.sarimax.SARIMAXResults API](http://www.statsmodels.org/dev/generated/statsmodels.tsa.statespace.sarimax.SARIMAXResults.html)
* [Autoregressive integrated moving average on Wikipedia](https://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average)

## **Seasonal Autoregressive Integrated Moving-Average with Exogenous Regressors (SARIMAX)**

The Seasonal Autoregressive Integrated Moving-Average with Exogenous Regressors ([SARIMAX](https://machinelearningmastery.com/sarima-for-time-series-forecasting-in-python/)) is an extension of the SARIMA model that also includes the modeling of exogenous variables.

Exogenous variables are also called covariates and can be thought of as parallel input sequences that have observations at the same time steps as the original series. The primary series may be referred to as endogenous data to contrast it from the exogenous sequence(s). The observations for exogenous variables are included in the model directly at each time step and are not modeled in the same way as the primary endogenous sequence (e.g. as an AR, MA, etc. process).

The SARIMAX method can also be used to model the subsumed models with exogenous variables, such as ARX, MAX, ARMAX, and ARIMAX.

The method is suitable for univariate time series with trend and/or seasonal components and exogenous variables.

### **Python Code**

| 1  2  3  4  5  6  7  8  9  10  11  12  13 | # SARIMAX example  from statsmodels.tsa.statespace.sarimax import SARIMAX  from random import random  # contrived dataset  data1 = [x + random() for x in range(1, 100)]  data2 = [x + random() for x in range(101, 200)]  # fit model  model = SARIMAX(data1, exog=data2, order=(1, 1, 1), seasonal\_order=(0, 0, 0, 0))  model\_fit = model.fit(disp=False)  # make prediction  exog2 = [200 + random()]  yhat = model\_fit.predict(len(data1), len(data1), exog=[exog2])  print(yhat) |
| --- | --- |

### **More Information**

* [statsmodels.tsa.statespace.sarimax.SARIMAX API](http://www.statsmodels.org/dev/generated/statsmodels.tsa.statespace.sarimax.SARIMAX.html)
* [statsmodels.tsa.statespace.sarimax.SARIMAXResults API](http://www.statsmodels.org/dev/generated/statsmodels.tsa.statespace.sarimax.SARIMAXResults.html)
* [Autoregressive integrated moving average on Wikipedia](https://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average)

## **Vector Autoregression (VAR)**

The Vector Autoregression (VAR) method models the next step in each time series using an AR model. It is the generalization of AR to multiple parallel time series, e.g. multivariate time series.

The notation for the model involves specifying the order for the AR(p) model as parameters to a VAR function, e.g. VAR(p).

The method is suitable for multivariate time series without trend and seasonal components.

### **Python Code**

| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | # VAR example  from statsmodels.tsa.vector\_ar.var\_model import VAR  from random import random  # contrived dataset with dependency  data = list()  for i in range(100):  v1 = i + random()  v2 = v1 + random()  row = [v1, v2]  data.append(row)  # fit model  model = VAR(data)  model\_fit = model.fit()  # make prediction  yhat = model\_fit.forecast(model\_fit.y, steps=1)  print(yhat) |
| --- | --- |

### **More Information**

* [statsmodels.tsa.vector\_ar.var\_model.VAR API](http://www.statsmodels.org/dev/generated/statsmodels.tsa.vector_ar.var_model.VAR.html)
* [statsmodels.tsa.vector\_ar.var\_model.VARResults API](http://www.statsmodels.org/dev/generated/statsmodels.tsa.vector_ar.var_model.VARResults.html)
* [Vector autoregression on Wikipedia](https://en.wikipedia.org/wiki/Vector_autoregression)

## **Vector Autoregression Moving-Average (VARMA)**

The Vector Autoregression Moving-Average (VARMA) method models the next step in each time series using an ARMA model. It is the generalization of ARMA to multiple parallel time series, e.g. multivariate time series.

The notation for the model involves specifying the order for the AR(p) and MA(q) models as parameters to a VARMA function, e.g. VARMA(p, q). A VARMA model can also be used to develop VAR or VMA models.

The method is suitable for multivariate time series without trend and seasonal components.

### **Python Code**

| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | # VARMA example  from statsmodels.tsa.statespace.varmax import VARMAX  from random import random  # contrived dataset with dependency  data = list()  for i in range(100):  v1 = random()  v2 = v1 + random()  row = [v1, v2]  data.append(row)  # fit model  model = VARMAX(data, order=(1, 1))  model\_fit = model.fit(disp=False)  # make prediction  yhat = model\_fit.forecast()  print(yhat) |
| --- | --- |

### **More Information**

* [statsmodels.tsa.statespace.varmax.VARMAX API](http://www.statsmodels.org/dev/generated/statsmodels.tsa.statespace.varmax.VARMAX.html)
* [statsmodels.tsa.statespace.varmax.VARMAXResults](http://www.statsmodels.org/dev/generated/statsmodels.tsa.statespace.varmax.VARMAXResults.html)
* [Vector autoregression on Wikipedia](https://en.wikipedia.org/wiki/Vector_autoregression)

## **Vector Autoregression Moving-Average with Exogenous Regressors (VARMAX)**

The Vector Autoregression Moving-Average with Exogenous Regressors (VARMAX) is an extension of the VARMA model that also includes the modeling of exogenous variables. It is a multivariate version of the ARMAX method.

Exogenous variables are also called covariates and can be thought of as parallel input sequences that have observations at the same time steps as the original series. The primary series(es) are referred to as endogenous data to contrast it from the exogenous sequence(s). The observations for exogenous variables are included in the model directly at each time step and are not modeled in the same way as the primary endogenous sequence (e.g. as an AR, MA, etc. process).

The VARMAX method can also be used to model the subsumed models with exogenous variables, such as VARX and VMAX.

The method is suitable for multivariate time series without trend and seasonal components with exogenous variables.

### **Python Code**

| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | # VARMAX example  from statsmodels.tsa.statespace.varmax import VARMAX  from random import random  # contrived dataset with dependency  data = list()  for i in range(100):  v1 = random()  v2 = v1 + random()  row = [v1, v2]  data.append(row)  data\_exog = [x + random() for x in range(100)]  # fit model  model = VARMAX(data, exog=data\_exog, order=(1, 1))  model\_fit = model.fit(disp=False)  # make prediction  data\_exog2 = [[100]]  yhat = model\_fit.forecast(exog=data\_exog2)  print(yhat) |
| --- | --- |

### **More Information**

* [statsmodels.tsa.statespace.varmax.VARMAX API](http://www.statsmodels.org/dev/generated/statsmodels.tsa.statespace.varmax.VARMAX.html)
* [statsmodels.tsa.statespace.varmax.VARMAXResults](http://www.statsmodels.org/dev/generated/statsmodels.tsa.statespace.varmax.VARMAXResults.html)
* [Vector autoregression on Wikipedia](https://en.wikipedia.org/wiki/Vector_autoregression)

## **Simple Exponential Smoothing (SES)**

The Simple Exponential Smoothing (SES) method models the next time step as an exponentially weighted linear function of observations at prior time steps.

The method is suitable for univariate time series without trend and seasonal components.

### **Python Code**

| 1  2  3  4  5  6  7  8  9  10  11 | # SES example  from statsmodels.tsa.holtwinters import SimpleExpSmoothing  from random import random  # contrived dataset  data = [x + random() for x in range(1, 100)]  # fit model  model = SimpleExpSmoothing(data)  model\_fit = model.fit()  # make prediction  yhat = model\_fit.predict(len(data), len(data))  print(yhat) |
| --- | --- |

### **More Information**

* [statsmodels.tsa.holtwinters.SimpleExpSmoothing API](http://www.statsmodels.org/dev/generated/statsmodels.tsa.holtwinters.SimpleExpSmoothing.html)
* [statsmodels.tsa.holtwinters.HoltWintersResults API](http://www.statsmodels.org/dev/generated/statsmodels.tsa.holtwinters.HoltWintersResults.html)
* [Exponential smoothing on Wikipedia](https://en.wikipedia.org/wiki/Exponential_smoothing)

## **Holt Winter’s Exponential Smoothing (HWES)**

The [Holt Winter’s Exponential Smoothing](https://machinelearningmastery.com/how-to-grid-search-triple-exponential-smoothing-for-time-series-forecasting-in-python/) (HWES) also called the Triple Exponential Smoothing method models the next time step as an exponentially weighted linear function of observations at prior time steps, taking trends and seasonality into account.

The method is suitable for univariate time series with trend and/or seasonal components.

### **Python Code**

| 1  2  3  4  5  6  7  8  9  10  11 | # HWES example  from statsmodels.tsa.holtwinters import ExponentialSmoothing  from random import random  # contrived dataset  data = [x + random() for x in range(1, 100)]  # fit model  model = ExponentialSmoothing(data)  model\_fit = model.fit()  # make prediction  yhat = model\_fit.predict(len(data), len(data))  print(yhat) |
| --- | --- |

### **More Information**

* [statsmodels.tsa.holtwinters.ExponentialSmoothing API](http://www.statsmodels.org/dev/generated/statsmodels.tsa.holtwinters.ExponentialSmoothing.html)
* [statsmodels.tsa.holtwinters.HoltWintersResults API](http://www.statsmodels.org/dev/generated/statsmodels.tsa.holtwinters.HoltWintersResults.html)
* [Exponential smoothing on Wikipedia](https://en.wikipedia.org/wiki/Exponential_smoothing)

## **Further Reading**

This section provides more resources on the topic if you are looking to go deeper.

* [Statsmodels: Time Series analysis API](http://www.statsmodels.org/dev/tsa.html)
* [Statsmodels: Time Series Analysis by State Space Methods](http://www.statsmodels.org/dev/statespace.html)

## **Summary**

In this post, you discovered a suite of classical time series forecasting methods that you can test and tune on your time series dataset.

Did I miss your favorite classical time series forecasting method?

Let me know in the comments below.

Did you try any of these methods on your dataset?

Let me know about your findings in the comments.

**Do you have any questions?**

Ask your questions in the comments below and I will do my best to answer.