How to Make Predictions for Time Series Forecasting with Python

by **Jason Brownlee** on January 27, 2017 in **Time Series**

Selecting a time series forecasting model is just the beginning.

Using the chosen model in practice can pose challenges, including data transformations and storing the model parameters on disk.

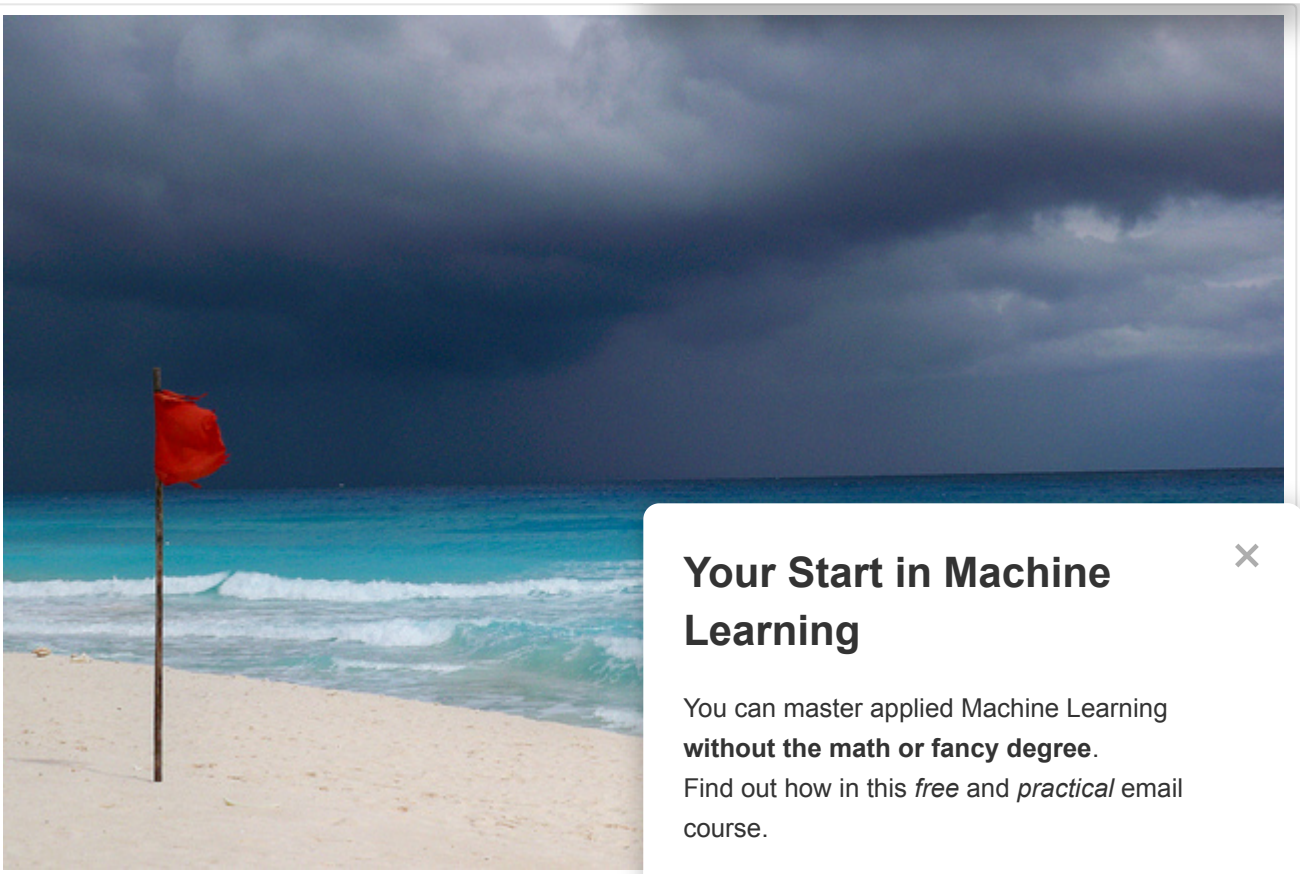
In this tutorial, you will discover how to finalize a time series forecasting model and use it to make predictions in Python.

After completing this tutorial, you will know:

- How to finalize a model and save it and required data to file.
- How to load a finalized model from file and use it to make a prediction.
- How to update data associated with a finalized model in order to make subsequent predictions.

Let's get started.

- **Update Feb/2017**: Updated layout and filenames to separate the AR case from the manual case.



How to Make Predictions for Tim
Photo by joe christianse

Process for Making a Prediction

A lot is written about how to tune specific time series forecasting models, but little help is given to how to use a model to make predictions.

Once you can build and tune forecast models for your data, the process of making a prediction involves the following steps:

1. **Model Selection**. This is where you choose a model and gather evidence and support to defend the decision.
2. **Model Finalization**. The chosen model is trained on all available data and saved to file for later use.
3. **Forecasting**. The saved model is loaded and used to make a forecast.
4. **Model Update**. Elements of the model are updated in the presence of new observations.

We will take a look at each of these elements in this tutorial, with a focus on saving and loading the model to and from file and using a loaded model to make predictions.

Before we dive in, let's first look at a standard univariate dataset that we can use as the context for this tutorial.

Daily Female Births Dataset

This dataset describes the number of daily female birt

The units are a count and there are 365 observations.
(1988).

Learn more and download the dataset from Data Mark

Download the dataset and place it in your current work
births.csv".

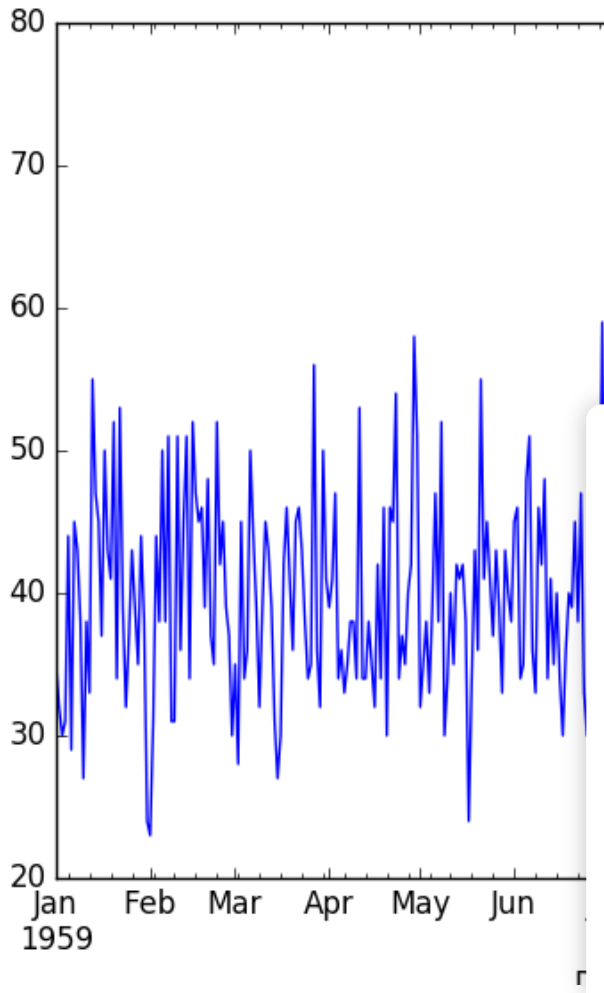
We can load the dataset as a Pandas series. The snip


```
from pandas import Series
from matplotlib import pyplot
series = Series.from_csv('daily-total-female-births.csv', header=0)
print(series.head())
series.plot()
pyplot.show()
```

Running the example prints the first 5 rows of the dataset.

```
Date
1959-01-01    35
1959-01-02    32
1959-01-03    30
1959-01-04    31
1959-01-05    44
```

The series is then graphed as a line plot.



Daily Female Birth Dataset Line Plot

1. Select Time Series Forecast Model

You must select a model.

This is where the bulk of the effort will be in preparing the data, performing analysis, and ultimately selecting a model and model hyperparameters that best capture the relationships in the data.

In this case, we can arbitrarily select an autoregression model (AR) with a lag of 6 on the differenced dataset.

We can demonstrate this model below.

First, the data is transformed by differencing, with each observation transformed as:

```
value(t) = obs(t) - obs(t - 1)
```

Next, the AR(6) model is trained on 66% of the historical data. The regression coefficients learned by the model are extracted and used to make predictions in a rolling manner across the test dataset.

As each time step in the test dataset is executed, the prediction is made using the coefficients and stored. The actual observation for the time step is then made available and stored to be used as a lag variable for future predictions.

```
from pandas import Series
from matplotlib import pyplot
from statsmodels.tsa.ar_model import AR
from sklearn.metrics import mean_squared_error
import numpy

# create a difference transform of the dataset
def difference(dataset):
    diff = list()
    for i in range(1, len(dataset)):
        value = dataset[i] - dataset[i - 1]
        diff.append(value)
    return numpy.array(diff)

# Make a prediction give regression coefficie
def predict(coef, history):
    yhat = coef[0]
    for i in range(1, len(coef)):
        yhat += coef[i] * history[-i]
    return yhat

series = Series.from_csv('daily-total-female-
# split dataset
X = difference(series.values)
size = int(len(X) * 0.66)
train, test = X[0:size], X[size:]
# train autoregression
model = AR(train)
model_fit = model.fit(maxlag=6, disp=False)
window = model_fit.k_ar
coef = model_fit.params
# walk forward over time steps in test
history = [train[i] for i in range(len(train))]
predictions = list()
for t in range(len(test)):
    yhat = predict(coef, history)
    obs = test[t]
    predictions.append(yhat)
    history.append(obs)
error = mean_squared_error(test, predictions)
print('Test MSE: %.3f' % error)
# plot
pyplot.plot(test)
pyplot.plot(predictions, color='red')
pyplot.show()
```


Running the example first prints the Mean Squared Error (MSE) of the predictions, which is 52 (or about 7 births on average, if we take the square root to return the error score to the original units).

This is how well we expect the model to perform on average when making forecasts on new data.

```
Test MSE: 52.696
```

Finally, a graph is created showing the actual observations in the test dataset (blue) compared to the predictions (red).

Predictions vs Actual Daily Female Birth Dataset Line Plot

This may not be the very best possible model we could develop on this problem, but it is reasonable and skillful.

2. Finalize and Save Time Series Forecast Model

Once the model is selected, we must finalize it.

This means save the salient information learned by the model so that we do not have to re-create it every time a prediction is needed.

This involves first training the model on all available data and then saving the model to file.

The *statsmodels* implementations of time series models do provide built-in capability to save and load models by calling *save()* and *load()* on the fit ARResults object.

For example, the code below will train an AR(6) model on the entire Female Births dataset and save it using the built-in *save()* function, which will essentially pickle the *ARResults* object.

The differenced training data must also be saved, both the for the lag variables needed to make a prediction, and for knowledge of the number of observations seen, required by the *predict()* function of the *ARResults* object.

Finally, we need to be able to transform the differenced dataset back into the original form. To do this, we must keep track of the last actual observation. This is so that the predicted differenced value can be added to it.

```
# fit an AR model and save the whole model to file
from pandas import Series
from statsmodels.tsa.ar_model import AR
from statsmodels.tsa.ar_model import ARResults
import numpy

# create a difference transform of the datase
def difference(dataset):
    diff = list()
    for i in range(1, len(dataset)):
        value = dataset[i] - dataset[i - 1]
        diff.append(value)
    return numpy.array(diff)

# load dataset
series = Series.from_csv('daily-total-female-
X = difference(series.values)
# fit model
model = AR(X)
model_fit = model.fit(maxlag=6, disp=False)
# save model to file
model_fit.save('ar_model.pkl')
# save the differenced dataset
numpy.save('ar_data.npy', X)
# save the last ob
numpy.save('ar_obs.npy', [series.values[-1]])
```


This code will create a file *ar_model.pkl* that you can load later and use to make predictions.

The entire training dataset is saved as *ar_data.npy* and the last observation is saved in the file *ar_obs.npy* as an array with one item.

The NumPy save() function is used to save the differenced training data and the observation. The load() function can then be used to load these arrays later.

The snippet below will load the model, differenced data, and last observation.

```
# load the AR model from file
from statsmodels.tsa.ar_model import ARResults
import numpy
loaded = ARResults.load('ar_model.pkl')
print(loaded.params)
data = numpy.load('ar_data.npy')
last_ob = numpy.load('ar_obs.npy')
print(last_ob)
```

Running the example prints the coefficients and the last observation.

```
[ 0.12129822 -0.75275857 -0.612367   -0.510971
```

```
 -0.23412997]
[50]
```

I think this is good for most cases, but is also pretty heavy. You are subject to changes to the statsmodels API.

My preference is to work with the coefficients of the model directly, as in the case above, evaluating the model using a rolling forecast.

In this case, you could simply store the model coefficients and later load them and make predictions.

The example below saves just the coefficients from the model, as well as the minimum differenced lag values required to make the next prediction and the last observation needed to transform the next prediction made.

```
# fit an AR model and manually save coefficie
from pandas import Series
from statsmodels.tsa.ar_model import AR
import numpy

# create a difference transform of the datase
def difference(dataset):
    diff = list()
    for i in range(1, len(dataset)):
        value = dataset[i] - dataset[i - 1]
        diff.append(value)
    return numpy.array(diff)

# load dataset
series = Series.from_csv('daily-total-female-
X = difference(series.values)
# fit model
window_size = 6
model = AR(X)
model_fit = model.fit(maxlag=window_size, disp=False)
# save coefficients
coef = model_fit.params
numpy.save('man_model.npy', coef)
# save lag
lag = X[-window_size:]
numpy.save('man_data.npy', lag)
# save the last ob
numpy.save('man_obs.npy', [series.values[-1]])
```


The coefficients are saved in the local file *man_model.npy*, the lag history is saved in the file *man_data.npy*, and the last observation is saved in the file *man_obs.npy*.

These values can then be loaded again as follows:

```
# load the manually saved model from file
import numpy
coef = numpy.load('man_model.npy')
print(coef)
lag = numpy.load('man_data.npy')
print(lag)
last_ob = numpy.load('man_obs.npy')
print(last_ob)
```

Running this example prints the loaded data for review. We can see the coefficients and last observation match the output from the previous example.

```
[ 0.12129822 -0.75275857 -0.612367   -0.51097172 -0.4176669  -0.32116469
 -0.23412997]
[-10   3  15  -4   7  -5]
[50]
```

Now that we know how to save a finalized model, we can use it to make forecasts.

3. Make a Time Series Forecast

Making a forecast involves loading the saved model and estimating the observation at the next time step.

If the ARResults object was serialized, we can use the

The example below shows how the next time period c

The model, training data, and last observation are loa

The period is specified to the *predict()* function as the et.
This index may be stored directly in a file instead of st
efficiency.

The prediction is made, which is in the context of the d o
the original units, it must be added to the last known o


```
# load AR model from file and make a one-step
from statsmodels.tsa.ar_model import ARResults
import numpy
# load model
model = ARResults.load('ar_model.pkl')
data = numpy.load('ar_data.npy')
last_ob = numpy.load('ar_obs.npy')
# make prediction
predictions = model.predict(start=len(data), end=len(data))
# transform prediction
yhat = predictions[0] + last_ob[0]
print('Prediction: %f' % yhat)
```

Running the example prints the prediction.

```
Prediction: 46.755211
```

We can also use a similar trick to load the raw coefficients and make a manual prediction.
The complete example is listed below.

```
# load a coefficients and from file and make a manual prediction
import numpy

def predict(coef, history):
    yhat = coef[0]
    for i in range(1, len(coef)):
        yhat += coef[i] * history[-i]
    return yhat
```

```
# load model
coef = numpy.load('man_model.npy')
lag = numpy.load('man_data.npy')
last_ob = numpy.load('man_obs.npy')
# make prediction
prediction = predict(coef, lag)
# transform prediction
yhat = prediction + last_ob[0]
print('Prediction: %f' % yhat)
```

Running the example, we achieve the same prediction as we would expect, given the underlying model and method for making the prediction are the same.

```
Prediction: 46.755211
```

4. Update Forecast Model

Our work is not done.

Once the next real observation is made available, we

Specifically, we must update:

1. The differenced training dataset used as inputs to
2. The last observation, providing a context for the p

Let's assume the next actual observation in the series
The new observation must first be differenced with the
differenced observations. Finally, the value can be sto

In the case of the stored AR model, we can update the *ar_data.npy* and *ar_obs.npy* files. The complete example is listed below:

```
# update the data for the AR model with a new obs
import numpy
# get real observation
observation = 48
# load the saved data
data = numpy.load('ar_data.npy')
last_ob = numpy.load('ar_obs.npy')
# update and save differenced observation
diffed = observation - last_ob[0]
data = numpy.append(data, [diffed], axis=0)
numpy.save('ar_data.npy', data)
# update and save real observation
last_ob[0] = observation
numpy.save('ar_obs.npy', last_ob)
```

We can make the same changes for the data files for the manual case. Specifically, we can update the *man_data.npy* and *man_obs.npy* respectively.

The complete example is listed below.

```
# update the data for the manual model with a
```

```
import numpy
# get real observation
observation = 48
# update and save differenced observation
lag = numpy.load('man_data.npy')
last_ob = numpy.load('man_obs.npy')
diffed = observation - last_ob[0]
lag = numpy.append(lag[1:], [diffed], axis=0)
numpy.save('man_data.npy', lag)
# update and save real observation
last_ob[0] = observation
numpy.save('man_obs.npy', last_ob)
```

We have focused on one-step forecasts.

These methods would work just as easily for multi-step forecasts, by using the model repetitively and using forecasts of previous time steps as input lag values to

Consider Storing All Observatio

Generally, it is a good idea to keep track of all the obs

This will allow you to:

- Provide a context for further time series analysis
- Train a new model in the future on the most recer
- Back-test new and different models to see if perfo

For small applications, perhaps you could store the ra

It may also be desirable to store the model coefficients and required lag data and last observation in plain text for easy review.

For larger applications, perhaps a database system could be used to store the observations.

Summary

In this tutorial, you discovered how to finalize a time series model and use it to make predictions with Python.

Specifically, you learned:

- How to save a time series forecast model to file.
- How to load a saved time series forecast from file and make a prediction.
- How to update a time series forecast model with new observations.

Do you have any questions?
Ask your questions in the comments below and I will do my best to answer.

About Jason Brownlee

Jason Brownlee, Ph.D. is a machine learni
with modern machine learning methods via
View all posts by Jason Brownlee →

‹ How to Use Power Transforms for Time Series Forecast Data with Python

How to Decompose Time Series Data into Trend and Seasonality ›

20 Responses to *How to Make Predictions for Time Series Forecasting with Python*

shamsul February 24, 2017 at 7:35 am #

REPLY

Great Work sir.

is it ok to apply time series analysis to predict patients status in ICU using ICU Database such as MIMIC II database? if it is then would be kind enough to guide me on this.

Thanking you

Jason Brownlee February 24, 2017 at 10:17 am #

REPLY

It is not my area of expertise. It might be a fun learning exercise.

This process will guide you through your predictive modeling problem:
http://machinelearningmastery.com/start-here/#process

Hoang March 16, 2017 at 5:50 am #

REPLY

Thanks for the tutorial.

I noticed that when I ran the following code:

series = pd.read_csv('daily-total-female-births.csv', hea

print(series.head())
series.plot()
plt.show()

I am not getting the same graph. The line graph appea
there additional codes not shown in the tutorial? Also,
and the code ran fine.

Thanks for your help.

Jason Brownlee March 16, 2017 at 8:04 am #

REPLY

Hi Hoang,

There's no additional code.

Confirm you have the latest version of Pandas and matplotlib installed. Also confirm that your data file contains date-times and observations with no additional data or footer information.

Hoang March 17, 2017 at 4:23 am #

REPLY

Thanks Jason, for your quick response.

I am using Pandas v0.19.2 and matplotlib v2.0.0.

The additional data was deleted in the data file (thanks to your suggestion) and now the graph looks very similar, however the dates are missing. I'm assuming it's because the indices are displayed on the x-axis. It was corrected once I set 'Date' as the index.

series=series.set_index('Date')

The graph doesn't look exactly like your graph but it's close enough.

Jason Brownlee March 17, 2017 at 8:31 am #

REPLY

Glad to hear you made some progress, thanks for providing the details. It may help others.

Nelson Silva July 23, 2017 at 7:59 am #

REPLY

Hi Jason, Thank you so much for your nice examples.
I've been wasting my time playing with game results da
handy. Usually, in these games there is no tendency at
How would you approach such predictions, for exampl
Where can I find some more advanced time-series pre
the outcome of 3 variables that linked to each other. Si

Thank you

Jason Brownlee July 24, 2017 at 6:46 am #

Do you mean lotto? I believe that is randc

Once you frame your time series as a supervised l
learning methods.

For predicting 3 variables, I would recommend a neural network model.

Jaskaran August 9, 2017 at 8:36 am #

REPLY

Hi Jason,
Great content as always.

I had a question.

1) Why did you not use ARIMA model here and why AR? Is there a difference?

Jason Brownlee August 10, 2017 at 6:38 am #

REPLY

There is a difference, AR does not include the differencing and moving average. It's a simpler method.

It is just an example for making predictions, you co

Martha August 22, 2017 at 3:33 am #

REPLY

Hi Jason,

Really good stuff!

# make prediction
predictions = model.predict(start=len(data), end=len(data))

here the minimum value for start should be window (6 in this case) and less than the end time.

Jason Brownlee August 22, 2017 at 6:48 am

Thanks Martha.

Garrett Dabbs November 19, 2017 at 12:36 pm #

"These methods would work just as easily for
and using forecasts of previous time steps as input lag
steps."

Hello, Jason! I'm a little new to this forecasting modelir
your tutorial and examples no problem. However, as I'm still poking around to see how this works, is there a good example you could point to that explains your above quote? I get that the predict function looks at coef[0] to ultimately perform its calculation, and our forecasting approach provides the next time step's prediction, but how do I tell the function "look at the NEXT time step after that one…" etc? Is it coef[1]… coef[n]?

Apologies if these are really dumb questions, I find this fascinating at how accurate it can be on even real world data, and want to do statistical analyses to find out when the "long term forecast" falls off a cliff when it comes to accuracy, I just haven't been able to figure out how to ask for time steps farther away than 1.

Jason Brownlee November 20, 2017 at 10:10 am #

REPLY

It is a good question.

You can use the model recursively but taking the forecast and using it as an observation in order to make the next forecast.

This post might have more details on the topic:
https://machinelearningmastery.com/multi-step-time-series-forecasting/

Sash March 27, 2018 at 8:57 am #

REPLY

Jason – awesome example.

Would you use the same method if you say, wanted to predict occupancy on a given floor, of a given building – provided you have time-series data of all of the login's (and logoffs) of all occupants that come onto the floor – dating back say 1 or 2 years?

Also, would you use the same method if you wanted to "predict" this occupancy from NOW to say, 2 – 3 days in advance – for a specific day? how would the time-step look like for the next day or in 2 days?

Jason Brownlee March 27, 2018 at 4:17 pm #

Perhaps. Try it and see how it performs o

Sarthak May 12, 2018 at 3:35 am #

This is a great tutorial. I have a question why
And why using difference ?
Is it necessary to transform the data ?

Jason Brownlee May 12, 2018 at 6:50 am #

REPLY

The data had a trend, I removed it via differencing.

Learn more about differencing here:
https://machinelearningmastery.com/remove-trends-seasonality-difference-transform-python/

Amresh August 27, 2018 at 8:26 pm #

REPLY

How can I check the prediction performance

Jason Brownlee August 28, 2018 at 5:59 am #

REPLY

You can calculate the error between predictions and actual values.

Leave a Reply

Name (required)

Email (will not be published) (requ

Website

SUBMIT COMMENT

Welcome to Machine Learning Mastery!

Hi, I'm Jason Brownlee, Ph.D.
My goal is to make developers like *YOU* awesome at applied machine learning.

Read More

Time Series Forecasting with Python
Explore time series data and predict the future.

You might also like…

- How to Install Python for Machine Learning

Your Start in Machine Learning

- Your First Machine Learning Project in Python
- Your First Neural Network in Python
- Your First Classifier in Weka
- Your First Time Series Forecasting Project

Your Start in Machine Learning