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

by **Jason Brownlee** on January 9, 2017 in **Time Series**

Last Updated on May 4, 2020

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

ARIMA is an acronym that stands for AutoRegressive Integrated Moving Average. It is a class of model that captures a suite of different standard temporal structures in time series data.

In this tutorial, you will discover how to develop an ARIMA model for time series data with 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.

Discover how to prepare and visualize time series data and develop autoregressive forecasting models in my new book, with 28 step-by-step tutorials, and full python code.

Let's get started.

- **Updated Apr/2019**: Updated the link to dataset.
- **Updated Sept/2019**: Updated examples to use latest API.

Autoregressive Integrated Moving Average Model

An ARIMA model is a class of statistical models 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.

This acronym is descriptive, capturing the key aspects of the model itself. Briefly, they are:

- **AR**: *Autoregression*. A model that uses the dependent relationship between an observation and some number of lagged observations.
- **I**: *Integrated*. The use of differencing of raw observations (e.g. subtracting an observation from an observation at the previous time step) in order to make the time series stationary.
- **MA**: *Moving Average*. A model that uses the depe rror from a moving average model applied to lagged c

Each of these components are explicitly specified in th sed of ARIMA(p,d,q) where the parameters are substitutec c ARIMA model being used.

The parameters of the ARIMA model are defined as fc

- **p**: The number of lag observations included in the
- **d**: The number of times that the raw observations differencing.
- **q**: The size of the moving average window, also called the order of moving average.

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

A value of 0 can be used for a parameter, which indicates to not use that element of the model. This way, the ARIMA model can be configured to perform the function of an ARMA model, and even a simple AR, I, or MA model.

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 in 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.

Shampoo Sales Dataset

This dataset describes the monthly number of sales of shampoo over a 3 year period.

The units are a sales count and there are 36 observations. The original dataset is credited to Makridakis, Wheelwright, and Hyndman (1998).

- Download the dataset.

Download the dataset and place it in your current worl sv“.

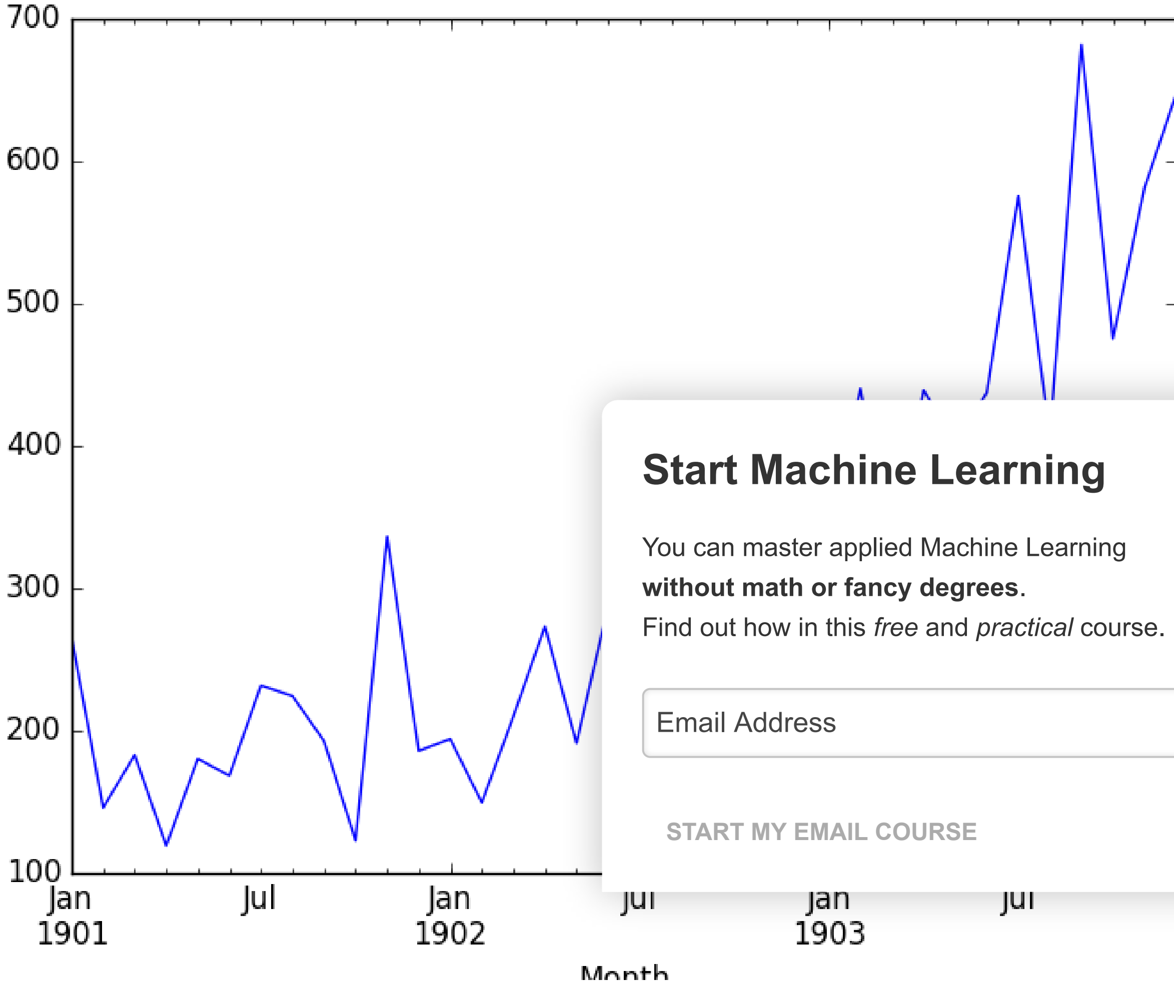
Below is an example of loading the Shampoo Sales da e the date-time field. The dataset is baselined in an arbi

```
from pandas import read_csv
from pandas import datetime
from matplotlib import pyplot

def parser(x):
    return datetime.strptime('190'+x, '%Y-%m')

series = read_csv('shampoo-sales.csv', header=0, parse_dates=[0], index_col=0, squeeze=True, dc
print(series.head())
series.plot()
pyplot.show()
```

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

```
Month
1901-01-01 266.0
1901-02-01 145.9
1901-03-01 183.1
1901-04-01 119.3
1901-05-01 180.3
Name: Sales, dtype: float64
```

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



Shampoo Sales Dataset Plot

We can see that the Shampoo Sales dataset has a clear trend.

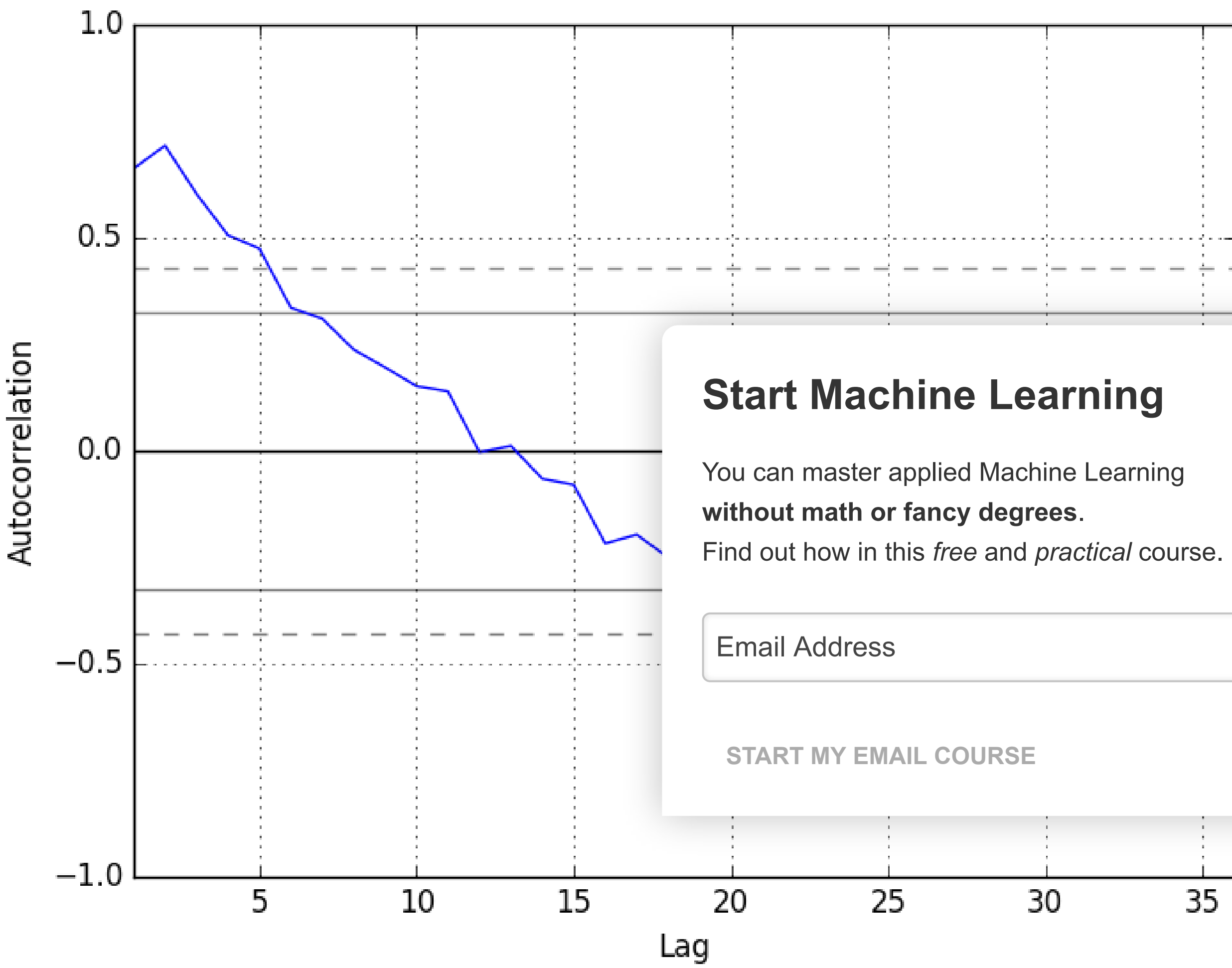
This suggests that the time series is not stationary and will require differencing to make it stationary, at least a difference order of 1.

Let’s also take a quick look at an autocorrelation plot of the time series. This is also built-in to Pandas. The example below plots the autocorrelation for a large number of lags in the time series.

```
from pandas import read_csv
from pandas import datetime
from matplotlib import pyplot
from pandas.plotting import autocorrelation_plot

def parser(x):
    return datetime.strptime('190'+x, '%Y-%m')

series = read_csv('shampoo-sales.csv', header=0, parse_dates=[0], index_col=0, squeeze=True, d
autocorrelation_plot(series)
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.

A good starting point for the AR parameter of the model may be 5.



Autocorrelation Plot of Shampoo Sales Data

ARIMA with Python

The statsmodels library provides the capability to fit an ARIMA model.

An ARIMA model can be created using the statsmodels library as follows:

1. Define the model by calling ARIMA() and passing in the *p*, *d*, and *q* parameters.
2. The model is prepared on the training data by calling the fit() function.
3. Predictions can be made by calling the predict() function and specifying the index of the time or times to be predicted.

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

First, we fit an ARIMA(5,1,0) model. This sets the lag value to 5 for autoregression, uses a difference order of 1 to make the time series stationary, and uses a mc

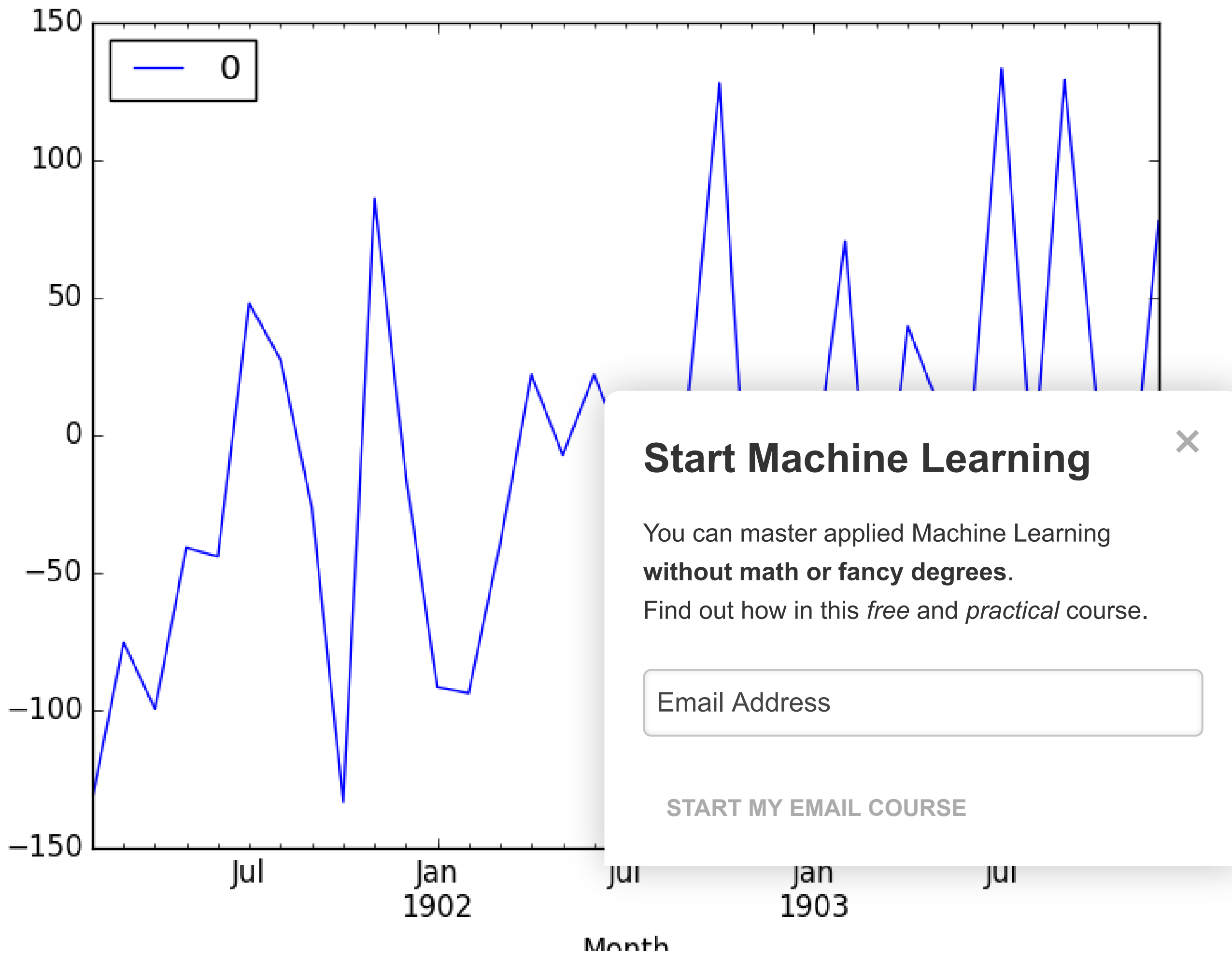
When fitting the model, a lot of debug information is provided about the fit of the linear regression model. We can turn this off by setting the *disp* argument to 0.

```
from pandas import read_csv
from pandas import datetime
from pandas import DataFrame
from statsmodels.tsa.arima_model import ARIMA
from matplotlib import pyplot

def parser(x):
    return datetime.strptime('190'+x, '%Y-%m')

series = read_csv('shampoo-sales.csv', header=0, parse_dates=[0], index_col=0, squeeze=True, dc
# fit model
model = ARIMA(series, order=(5,1,0))
model_fit = model.fit(disp=0)
print(model_fit.summary())
# plot residual errors
residuals = DataFrame(model_fit.resid)
residuals.plot()
pyplot.show()
residuals.plot(kind='kde')
pyplot.show()
print(residuals.describe())
```


Running the example prints a summary of the fit mode
well as the skill of the fit on the on the in-sample obse

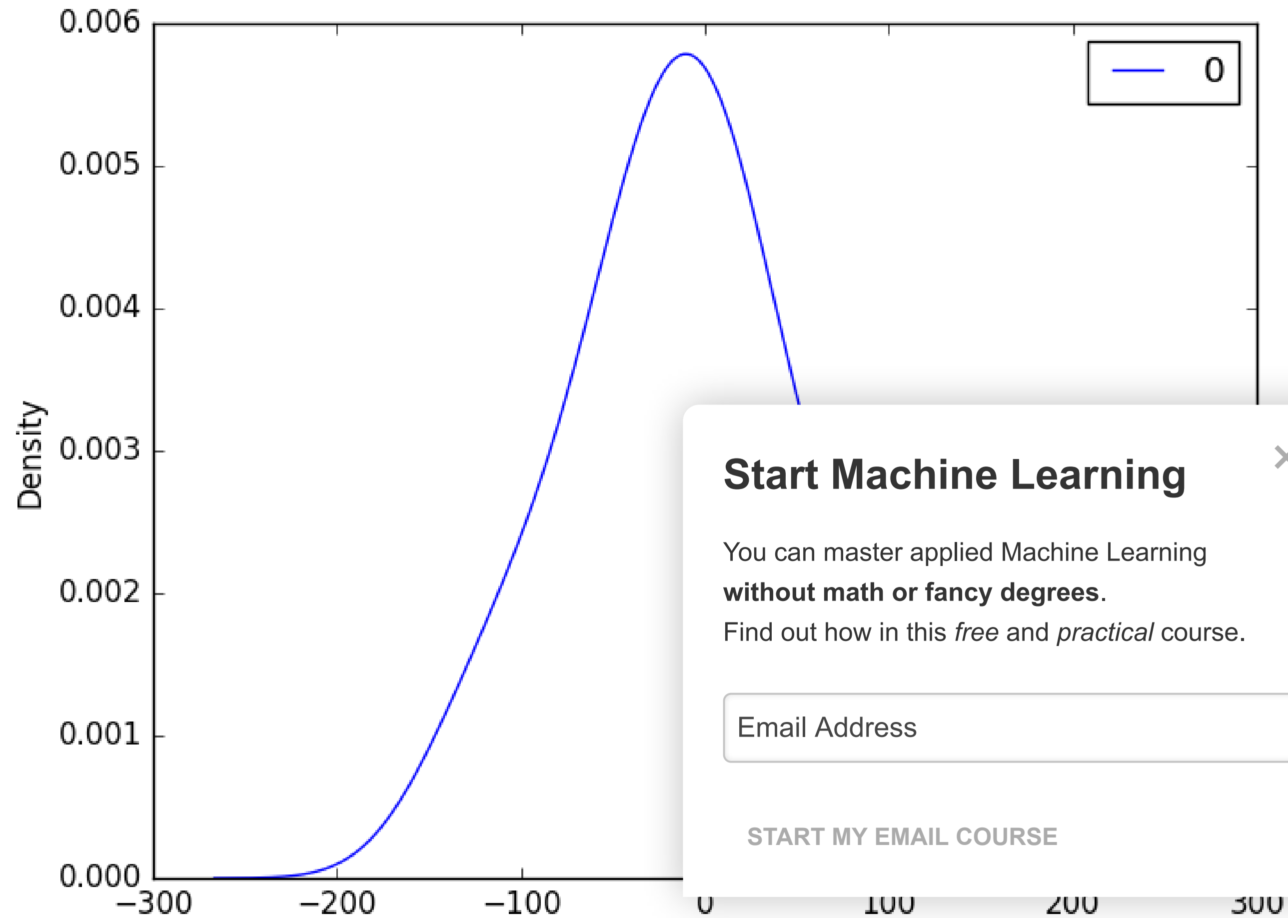
```
                              ARIMA Model Resu
=====================================================
Dep. Variable:                  D.Sales   No. O
Model:                   ARIMA(5, 1, 0)   Log Likelihood                -196.170
Method:                         css-mle   S.D. of innovations             64.241
Date:                  Mon, 12 Dec 2016   AIC                            406.340
Time:                          11:09:13   BIC                            417.227
Sample:                      02-01-1901   HQIC                           410.098
                           - 12-01-1903
==================================================================================
                     coef    std err          z      P>|z|      [95.0% Conf. Int.]
----------------------------------------------------------------------------------
const             12.0649      3.652      3.304      0.003         4.908    19.222
ar.L1.D.Sales     -1.1082      0.183     -6.063      0.000        -1.466    -0.750
ar.L2.D.Sales     -0.6203      0.282     -2.203      0.036        -1.172    -0.068
ar.L3.D.Sales     -0.3606      0.295     -1.222      0.231        -0.939     0.218
ar.L4.D.Sales     -0.1252      0.280     -0.447      0.658        -0.674     0.424
ar.L5.D.Sales      0.1289      0.191      0.673      0.506        -0.246     0.504
                                    Roots
=============================================================================
                 Real           Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1          -1.0617           -0.5064j            1.1763           -0.4292
AR.2          -1.0617           +0.5064j            1.1763            0.4292
AR.3           0.0816           -1.3804j            1.3828           -0.2406
AR.4           0.0816           +1.3804j            1.3828            0.2406
AR.5           2.9315           -0.0000j            2.9315           -0.0000
-----------------------------------------------------------------------------
```

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 centered 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).

```
count    35.000000
mean     -5.495213
std      68.132882
min    -133.296597
25%     -42.477935
50%      -7.186584
75%      24.748357
max     133.237980
```

Note, that although above 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.

Rolling Forecast ARIMA Model

The ARIMA model can be used to forecast future time steps.

We can use the predict() function on the ARIMAResults object to make predictions. It accepts the index of the time steps to make predictions as arguments. These indexes are relative to the start of the training dataset used to make predictions.

If we used 100 observations in the training dataset to fit the model, then the index of the next time step for making a prediction would be specified to the prediction function as *start=101, end=101*. This would return an array with one element containing the prediction.

We also would prefer the forecasted values to be in the original scale, in case we performed any differencing (*d>0* when configuring the model). This can be specified by setting the *typ* argument to the value *'levels'*: *typ='levels'*.

Alternately, we can avoid all of these specifications by ne-
step forecast using the model.

We can split the training dataset into train and test set a
prediction for each element on the test set.

A rolling forecast is required given the dependence or nd
the AR model. A crude way to perform this rolling fore ew
observation is received.

We manually keep track of all observations in a list ca d
to which new observations are appended each iteratic...

Putting this all together, below is an example of a rolling forecast with the ARIMA model in Python.

```
from pandas import read_csv
from pandas import datetime
from matplotlib import pyplot
from statsmodels.tsa.arima_model import ARIMA
from sklearn.metrics import mean_squared_error

def parser(x):
    return datetime.strptime('190'+x, '%Y-%m')

series = read_csv('shampoo-sales.csv', header=0, parse_dates=[0], index_col=0, squeeze=True, d
X = series.values
size = int(len(X) * 0.66)
train, test = X[0:size], X[size:len(X)]
history = [x for x in train]
predictions = list()
for t in range(len(test)):
    model = ARIMA(history, order=(5,1,0))
    model_fit = model.fit(disp=0)
    output = model_fit.forecast()
    yhat = output[0]
    predictions.append(yhat)
    obs = test[t]
    history.append(obs)
    print('predicted=%f, expected=%f' % (yhat, obs))
error = mean_squared_error(test, predictions)
print('Test MSE: %.3f' % error)
# plot
```

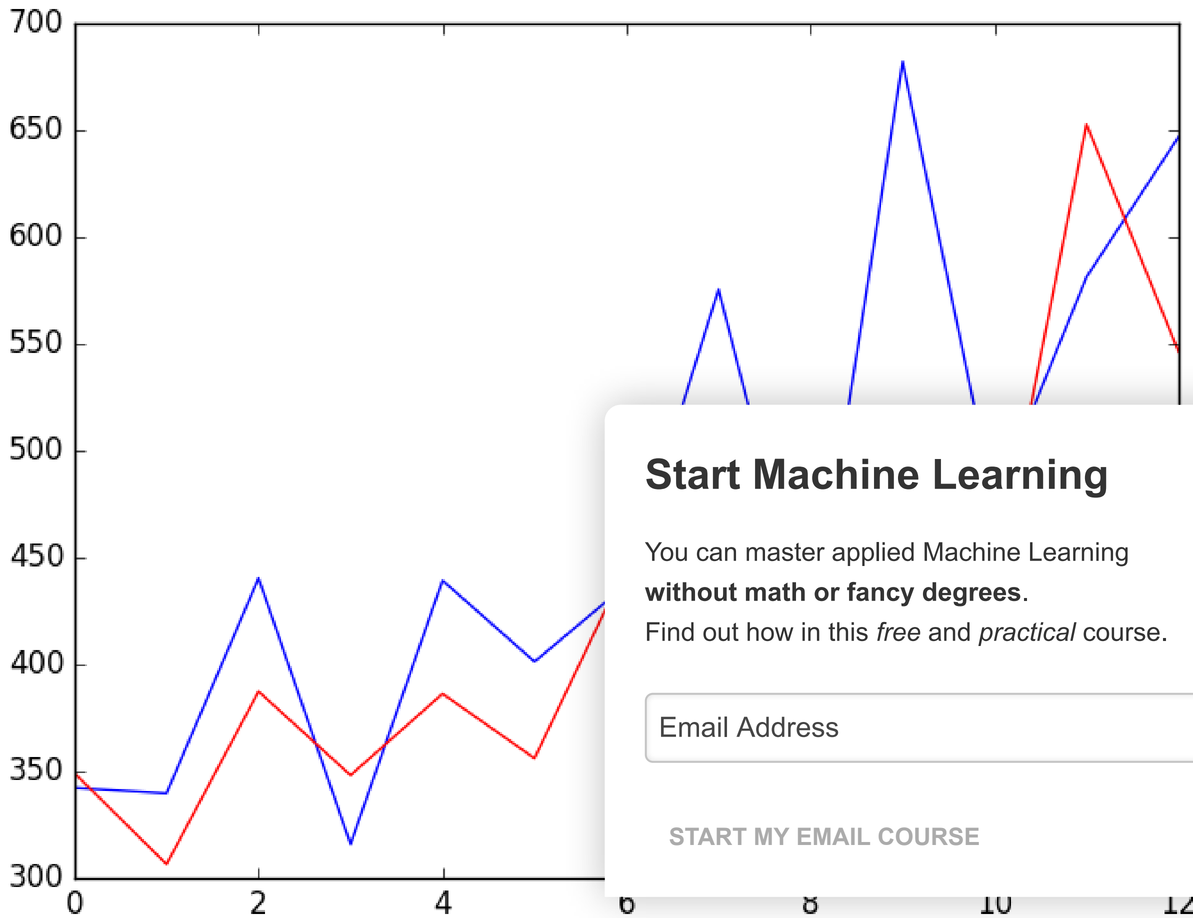
```
pyplot.plot(test)
pyplot.plot(predictions, color='red')
pyplot.show()
```

Running the example prints the prediction and expected value each iteration.

We can also calculate a final mean squared error score (MSE) for the predictions, providing a point of comparison for other ARIMA configurations.

```
predicted=349.117688, expected=342.300000
predicted=306.512968, expected=339.700000
predicted=387.376422, expected=440.400000
predicted=348.154111, expected=315.900000
predicted=386.308808, expected=439.300000
predicted=356.081996, expected=401.300000
predicted=446.379501, expected=437.400000
predicted=394.737286, expected=575.500000
predicted=434.915566, expected=407.600000
predicted=507.923407, expected=682.000000
predicted=435.483082, expected=475.300000
predicted=652.743772, expected=581.300000
predicted=546.343485, expected=646.900000
Test MSE: 6958.325
```

A line plot is created showing the expected values (blu ... d).
We can see the values show some trend and are in th



ARIMA Rolling Forecast Line Plot

The model could use further tuning of the p, d, and maybe even the q parameters.

Configuring an ARIMA Model

The classical approach for fitting an ARIMA model is to follow the Box-Jenkins Methodology.

This is a process that uses time series analysis and diagnostics to discover good parameters for the ARIMA model.

In summary, the steps of this process are as follows:

1. **Model Identification.** Use plots and summary statistics to identify trends, seasonality, and autoregression elements to get an idea of the amount of differencing and the size of the lag that will be required.
2. **Parameter Estimation.** Use a fitting procedure to find the coefficients of the regression model.
3. **Model Checking.** Use plots and statistical tests of the residual errors to determine the amount and type of temporal structure not captured by the model.

The process is repeated until either a desirable level of fit is achieved on the in-sample or out-of-sample observations (e.g. training or test datasets).

The process was described in the classic 1970 textbook on the topic titled Time Series Analysis: Forecasting and Control by George Box and Gwilym Jenkins. An updated 5th edition is now available if you are interested in going deeper into this type of model and methodology.

Given that the model can be fit efficiently on modest-sized time series datasets, grid searching parameters of the model can be a valuable approach.

For an example of how to grid search the hyperparameters of the ARIMA model, see the tutorial:

- How to Grid Search ARIMA Model Hyperparamet

Summary

In this tutorial, you discovered how to develop an ARI

Specifically, you learned:

- About the ARIMA model, how it can be configured
- How to perform a quick time series analysis using
- How to use an ARIMA model to forecast out of sa

Do you have any questions about ARIMA, or about this tutorial?
Ask your questions in the comments below and I will do my best to answer.
