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.



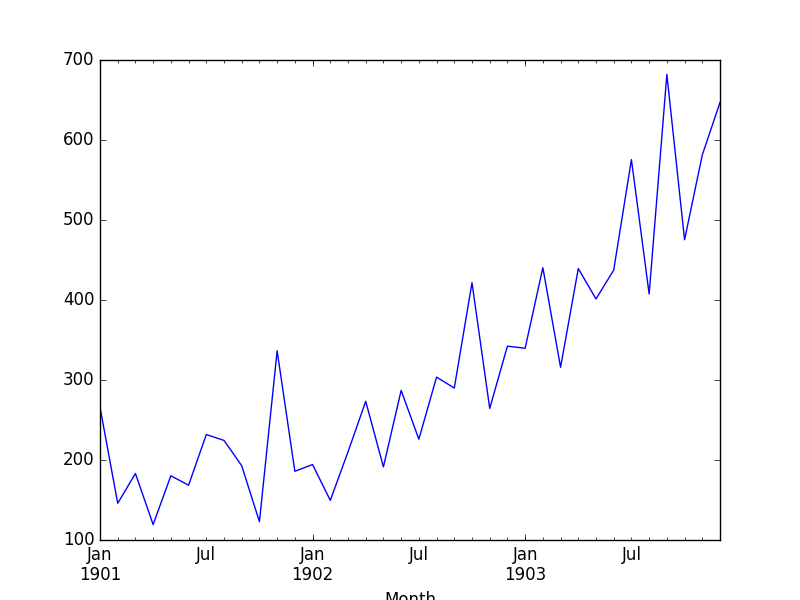
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | from pandas import read\_csv  from pandas import datetime  from matplotlib import pyplot    def parser(x):  return datetime.strptime('190'+x, '%Y-%m')    series = read\_csv('shampoo-sales.csv', header=0, parse\_dates=[0], index\_col=0, squeeze=True, date\_parser=parser)  print(series.head())  series.plot()  pyplot.show() |

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



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | 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.

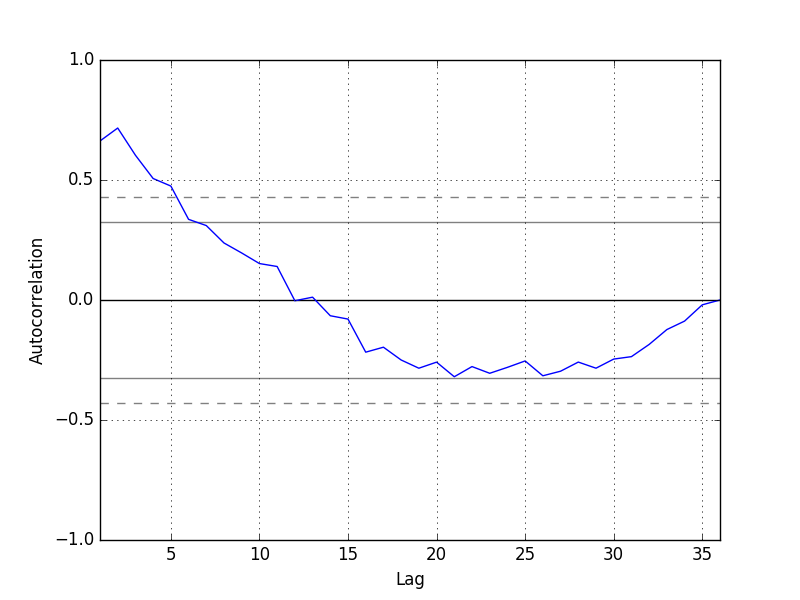
Let’s also take a quick look at an autocorrelation plot of the time series. This is also built-in to Pandas. The example below plots the autocorrelation for a large number of lags in the time series.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | from pandas import read\_csv  from pandas import datetime  from matplotlib import pyplot  from pandas.tools.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, date\_parser=parser)  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()](http://statsmodels.sourceforge.net/devel/generated/statsmodels.tsa.arima_model.ARIMA.html) and passing in the *p*, *d*, and *q* parameters.
2. The model is prepared on the training data by calling the [fit()](http://statsmodels.sourceforge.net/devel/generated/statsmodels.tsa.arima_model.ARIMA.fit.html) function.
3. Predictions can be made by calling the [predict()](http://statsmodels.sourceforge.net/devel/generated/statsmodels.tsa.arima_model.ARIMA.predict.html) 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 moving average model of 0.

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.



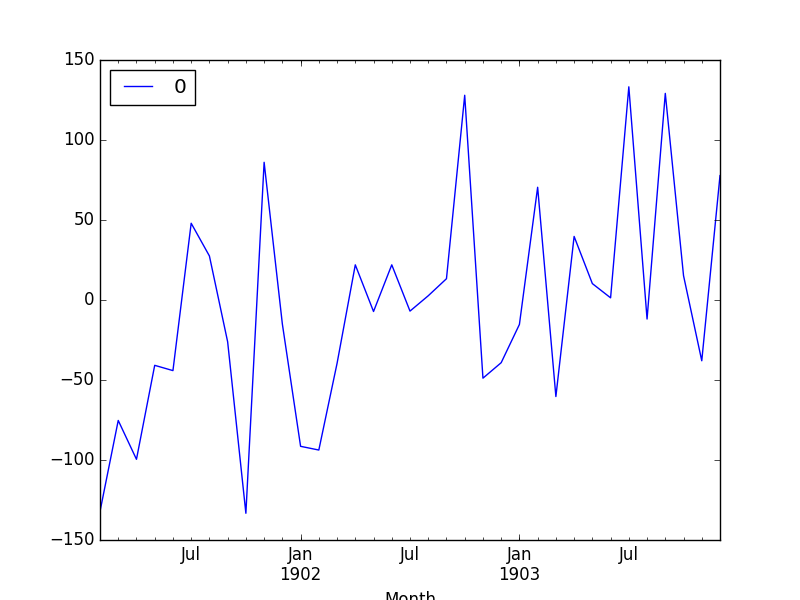
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | from pandas import read\_csv  from pandas import datetime  from pandas import DataFrame  from statsmodels.tsa.arima\_model import ARIMA  from matplotlib import pyplot    def parser(x):  return datetime.strptime('190'+x, '%Y-%m')    series = read\_csv('shampoo-sales.csv', header=0, parse\_dates=[0], index\_col=0, squeeze=True, date\_parser=parser)  # 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 model. This summarizes the coefficient values used as well as the skill of the fit on the on the in-sample observations.



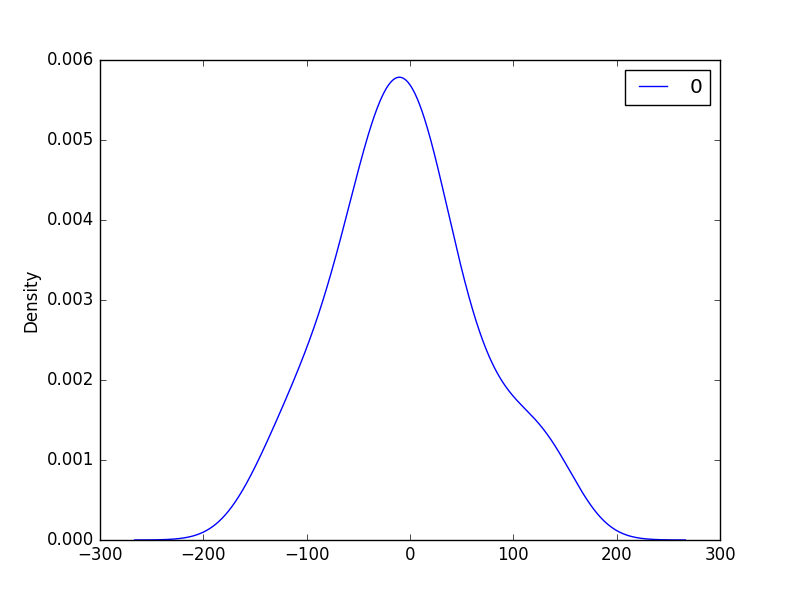
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | ARIMA Model Results  ==============================================================================  Dep. Variable:                D.Sales   No. Observations:                   35  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).



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | 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](http://statsmodels.sourceforge.net/devel/generated/statsmodels.tsa.arima_model.ARIMAResults.html) 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 using the [forecast()](http://statsmodels.sourceforge.net/devel/generated/statsmodels.tsa.arima_model.ARIMAResults.forecast.html) function, which performs a one-step forecast using the model.

We can split the training dataset into train and test sets, use the train set to fit the model, and generate a prediction for each element on the test set.

A rolling forecast is required given the dependence on observations in prior time steps for differencing and the AR model. A crude way to perform this rolling forecast is to re-create the ARIMA model after each new observation is received.

We manually keep track of all observations in a list called history that is seeded with the training data and to which new observations are appended each iteration.

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



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | 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, date\_parser=parser)  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() |

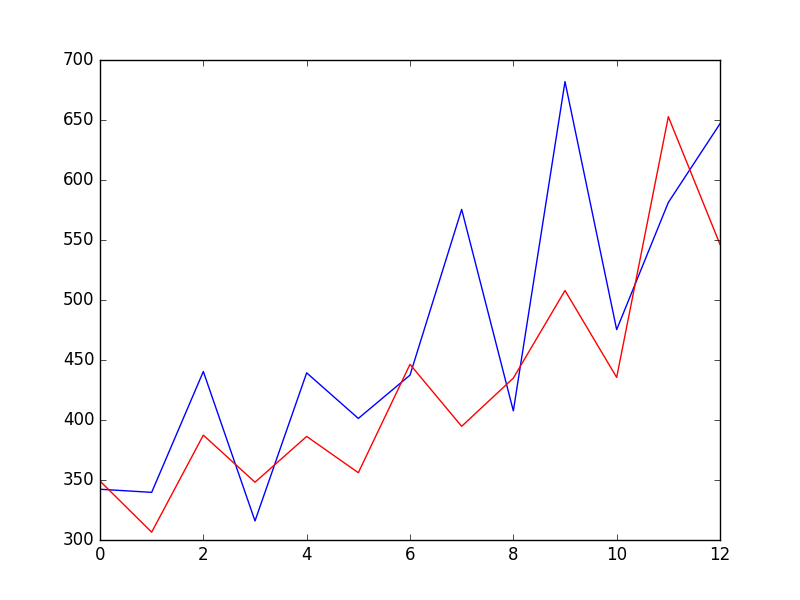
Running the example prints the prediction and expected value each iteration.

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



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | 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 (blue) compared to the rolling forecast predictions (red). We can see the values show some trend and are in the correct scale.



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](https://en.wikipedia.org/wiki/Box%E2%80%93Jenkins_method).

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).