Table of Contents

[Moving Average Components 1](#_Toc116019411)

[ARMA Model 2](#_Toc116019412)

[A Problem with Multi-Period Forecasts 5](#_Toc116019413)

[Walk-Forward Models 5](#_Toc116019414)

[Stationarity 7](#_Toc116019415)

[Differencing 8](#_Toc116019416)

[Avoiding Too Much Differencing 9](#_Toc116019417)

[Augmented Dickey-Fuller Test (ADF) 10](#_Toc116019418)

[ARIMA for Differencing 11](#_Toc116019419)

[Grid Searching ARIMA Parameters 12](#_Toc116019420)

[Auto-ARIMA 14](#_Toc116019421)

# Moving Average Components

**Note:** A moving average model (MA Model) \*is not to be confused with the moving average\*.

Moving average components within an autoregressive model refer to unexplained variance at each time step. The MA(q) component is a linear combination of past error terms.

In a moving average model (MA model), depends only on the lagged forecast errors where εt is white noise and q is the number of time steps.

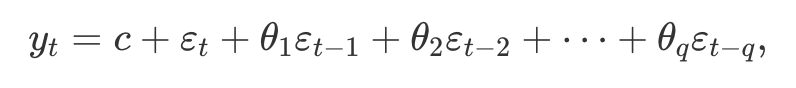
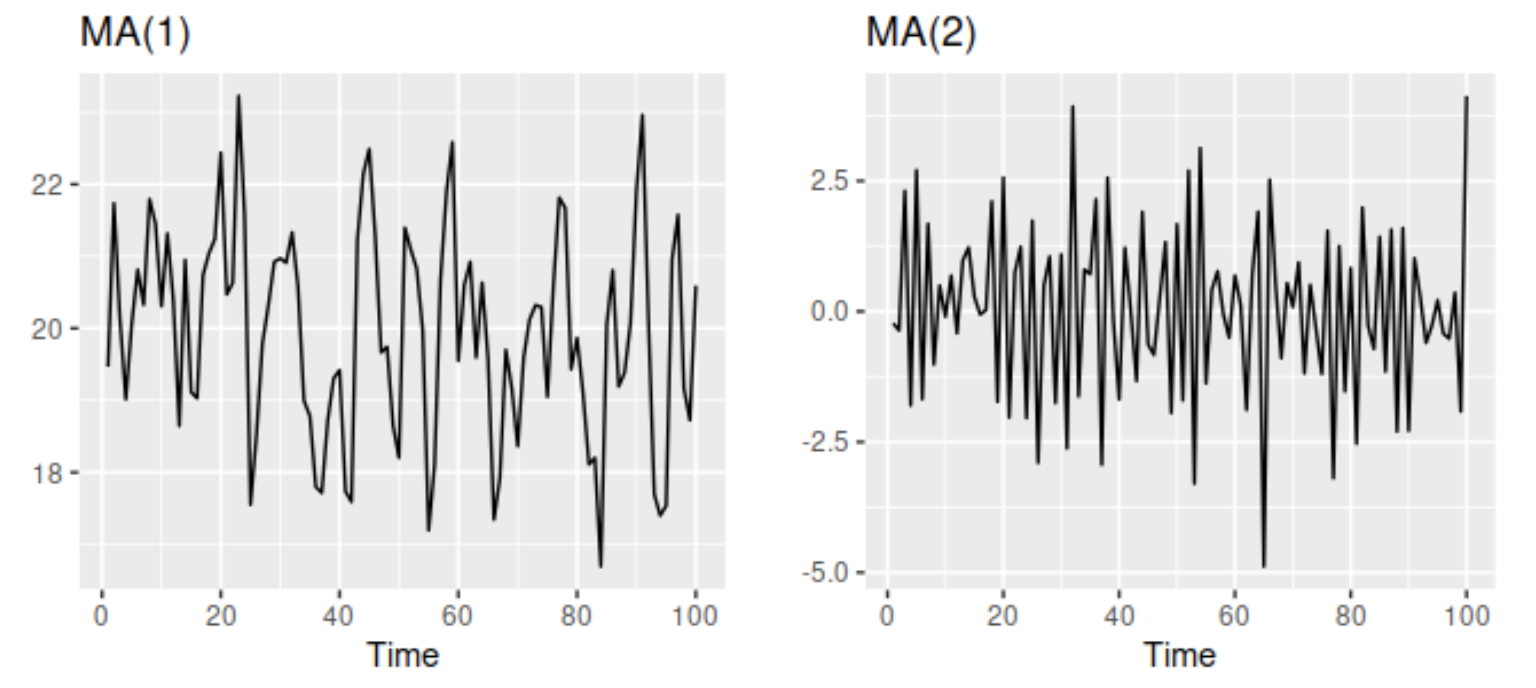
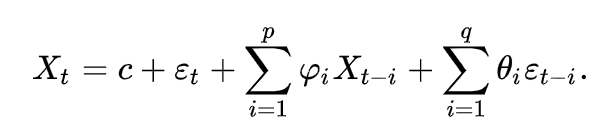


Figure 1: Unexplained shocks at time steps 1 and 2.



## ARMA Model

AR and MA models can sometimes be combined to generate better predictions than are possible with only an AR model. Together, the AR(p) and MA(q) components create an **ARMA** model.

 +

Example 1: Plotting ARMA Predictions

By including a moving average component in addition to an auto-regressive component, we ideally can improve the quality of our predictions. Over a period of 5 days it appears that including the moving average and a in our model improves the predictive result compared to not including it.

|  |
| --- |
| ar ma rmse  6 1 1 33.174186  7 1 2 33.346990  4 0 4 33.748635  3 0 3 33.783283  8 1 3 33.905250  1 0 1 33.986710  2 0 2 34.088609  0 0 0 34.102631  5 1 0 35.686172  …  15 3 0 45.457091  14 2 4 52.583664  19 3 4 59.351976  24 4 4 59.982582 |

Here is the code which grid searches the models based on different lag totals for auto regressive moving average (residual error) components.

|  |
| --- |
| import statsmodels.api as sm  import statsmodels.tsa.arima.model as sma  import matplotlib.pyplot as plt  import pandas as pd  from sklearn.metrics import mean\_squared\_error  import numpy as np  def getData():  df = sm.datasets.sunspots.load\_pandas().data[['SUNACTIVITY']]  df.index = pd.date\_range(start='1700', end='2009', freq='A')  TEST\_SZ = 5  train = df[0:len(df)-TEST\_SZ]  test = df[len(df)-TEST\_SZ:]  return train, test  def buildModel(df, ar, i, ma):  model = sma.ARIMA(df['SUNACTIVITY'], order=(ar, i, ma)).fit()  return model  def predictAndEvaluate(model, test, title):  print("\n\*\*\*" + title)  print(model.summary())  predictions = model.predict(start='2010', end='2014')  mse = mean\_squared\_error(predictions, test)  rmse = np.sqrt(mse)  print("RMSE: " + str(rmse))  return rmse  train, test = getData()  modelStats = []  for ar in range(0, 5):  for ma in range(0, 5):  model = buildModel(train, ar, 0, ma)  title = str(ar) + "\_0\_" + str(ma)  rmse = predictAndEvaluate(model, test, title)  modelStats.append({"ar":ar, "ma":ma, "rmse":rmse})  dfSolutions = pd.DataFrame(data=modelStats)  dfSolutions = dfSolutions.sort\_values(by=['rmse'])  print(dfSolutions) |

Exercise 1 (10 marks)

Use the following code to find the best ARMA model to find an optimal AR and MA combination for Microsoft stock Open prices. Here is some code to start with:

|  |
| --- |
| import numpy as np  from scipy import stats  import pandas as pd  import matplotlib.pyplot as plt  import statsmodels.api as sm  from statsmodels.tsa.arima.model import ARIMA  dta = sm.datasets.sunspots.load\_pandas().data  import datetime  import pandas\_datareader as pdr  def getStock(stk, ttlDays):      numDays = int(ttlDays)      dt = datetime.date.today()      dtPast = dt + datetime.timedelta(days=-numDays)      df = pdr.get\_data\_yahoo(stk,                              start=datetime.datetime(dtPast.year, dtPast.month,                                                      dtPast.day),                              end=datetime.datetime(dt.year, dt.month, dt.day))      return df  stkName = 'MSFT'  dfStock = getStock(stkName, 400)  # Split the data.  NUM\_TEST\_DAYS = 5  lenData        = len(dfStock)  dfTrain        = dfStock.iloc[0:lenData - NUM\_TEST\_DAYS, :]  dfTest         = dfStock.iloc[lenData-NUM\_TEST\_DAYS:,:]  plt.plot(dfStock.index, dfStock['Open'])  plt.show() |

Hints:

The only possible feature to build the ARMA model with is opening price. No other features are needed.

An easier way to make the prediction is by setting the start to the beginning and end of the test set like so:

start = len(dfTrain)  
 end = start + len(dfTest) -1  
 predictions = model.predict(start=start, end=end, dynamic=True)

Example 1 shows the main body of code that is needed to build the ARMA solution.

State the ARMA model here:

|  |
| --- |
| import statsmodels.api as sm import statsmodels.tsa.arima.model as sma import matplotlib.pyplot as plt import pandas as pd import numpy as np from sklearn.metrics import mean\_squared\_error from scipy import stats from statsmodels.tsa.arima.model import ARIMA dta = sm.datasets.sunspots.load\_pandas().data  import datetime import pandas\_datareader as pdr  def getStock(stk, ttlDays):  numDays = int(ttlDays)  dt = datetime.date.today()  dtPast = dt + datetime.timedelta(days=-numDays)  df = pdr.get\_data\_yahoo(stk,  start=datetime.datetime(dtPast.year, dtPast.month,  dtPast.day),  end=datetime.datetime(dt.year, dt.month, dt.day))  return df   def buildModel(df, ar, i, ma):  model = sma.ARIMA(df['Open'], order=(ar, i, ma)).fit()  return model   prediction\_list = []  def predictAndEvaluate(model, test, title):  print("\n\*\*\*" + title)  print(model.summary())   start = len(dfTrain)  end = start + len(dfTest) - 1  predictions = model.predict(start=start, end=end, dynamic=True)  prediction\_list.append(predictions)  mse = mean\_squared\_error(predictions, test['Open'])  rmse = np.sqrt(mse)  print("RMSE: " + str(rmse))  return rmse  stkName = 'MSFT' dfStock = getStock(stkName, 400)  # Split the data. NUM\_TEST\_DAYS = 5 lenData = len(dfStock) dfTrain = dfStock.iloc[0:lenData - NUM\_TEST\_DAYS, :] dfTest = dfStock.iloc[lenData-NUM\_TEST\_DAYS:,:]  modelStats = [] for ar in range(0, 5):  for ma in range(0, 5):  model = buildModel(dfTrain, ar, 0, ma)  title = str(ar) + "\_0\_" + str(ma)  rmse = predictAndEvaluate(model, dfTest, title)  modelStats.append({"ar":ar, "ma":ma, "rmse":rmse})  dfSolutions = pd.DataFrame(data=modelStats) dfSolutions = dfSolutions.sort\_values(by=['rmse']) print(dfSolutions)  dfStock\_last5 = dfStock.tail(5) plt.plot(dfStock\_last5.index, dfStock\_last5['Open'])  best\_ar = dfSolutions.values[0][0] best\_ma = dfSolutions.values[0][1] best\_index = int(best\_ar) \* 5 + int(best\_ma)  prediction\_list[best\_index].index = pd.date\_range("2022-10-05", "2022-10-11",  periods=len(prediction\_list[best\_index].index))  plt.plot(prediction\_list[best\_index].index, prediction\_list[best\_index]) plt.xticks(rotation=90) plt.tight\_layout() plt.show()  Text  Description automatically generated  Calendar  Description automatically generated  We are choosing index 6 wich is ar 1 ma 1 and RMSE is 16.021945 |

In English, interpret the model coefficients of the ARMA model:

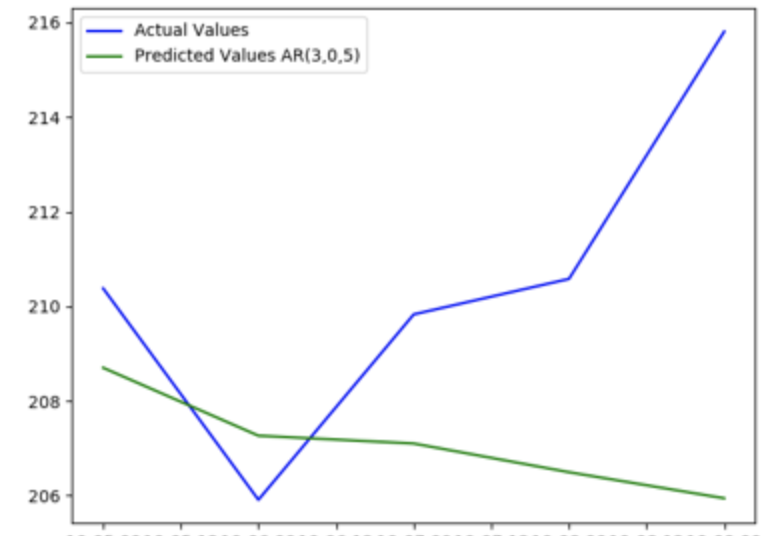
|  |
| --- |
| coefficients, they are weights of past observations of the data. there coefficients are consistenc and follow the normal distribution |

Show a plot of the actual versus predicted values for 5 days here:

|  |
| --- |
| Chart, line chart  Description automatically generated |

## A Problem with Multi-Period Forecasts

A problem you may encounter when making multi-day forecasts is the results are flat.

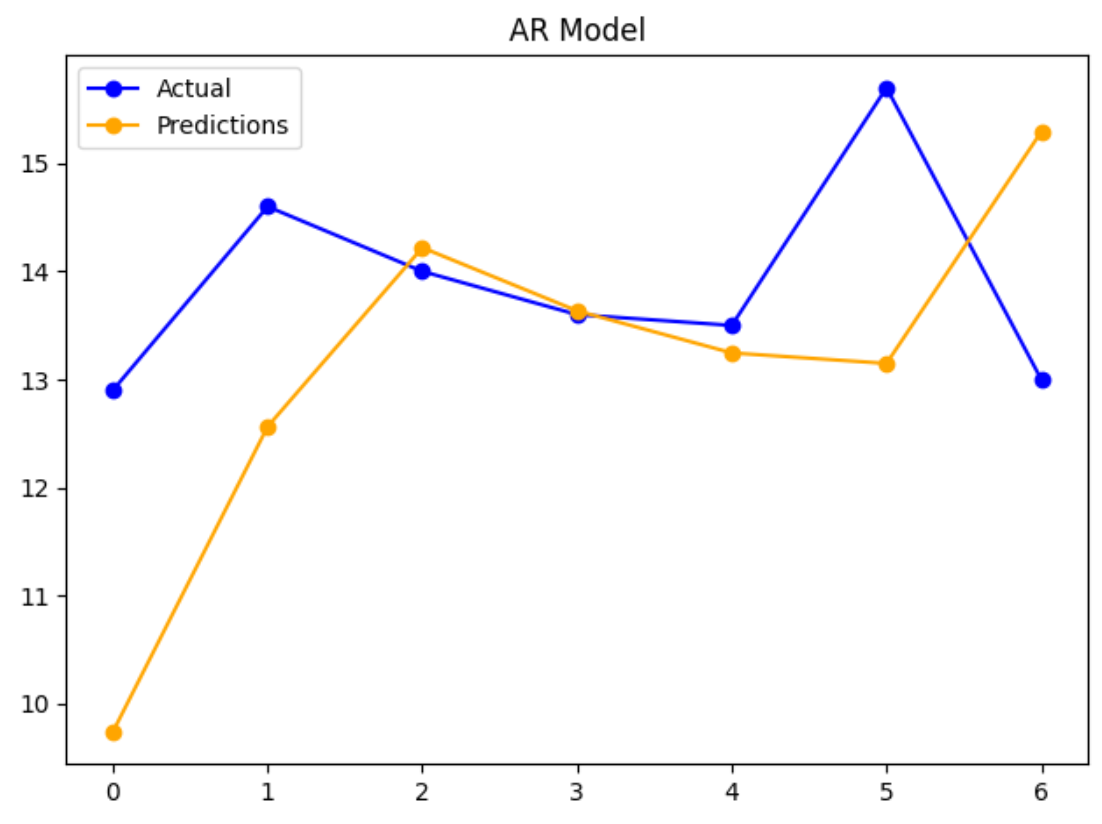


Further-ahead-predictions inherently become increasingly less accurate over time. The results are usually smoothed and they will not represent the fluctuations in volatility over time. Because of this issue, the examples today will focus on making predictions with the latest observations that are one period ahead. For today we will use the number of days as the period.

# Walk-Forward Models

Example : Day Ahead Prediction

This example shows how to build an AR model with a walk-forward prediction that is always one day ahead. At every iteration for each day, the observations are updated with the most recently available data before making a new prediction.



RMSE: 1.928

We are going to re-use these routines in several examples. The documentation for this code is in the comments.

This example implements a day ahead prediction

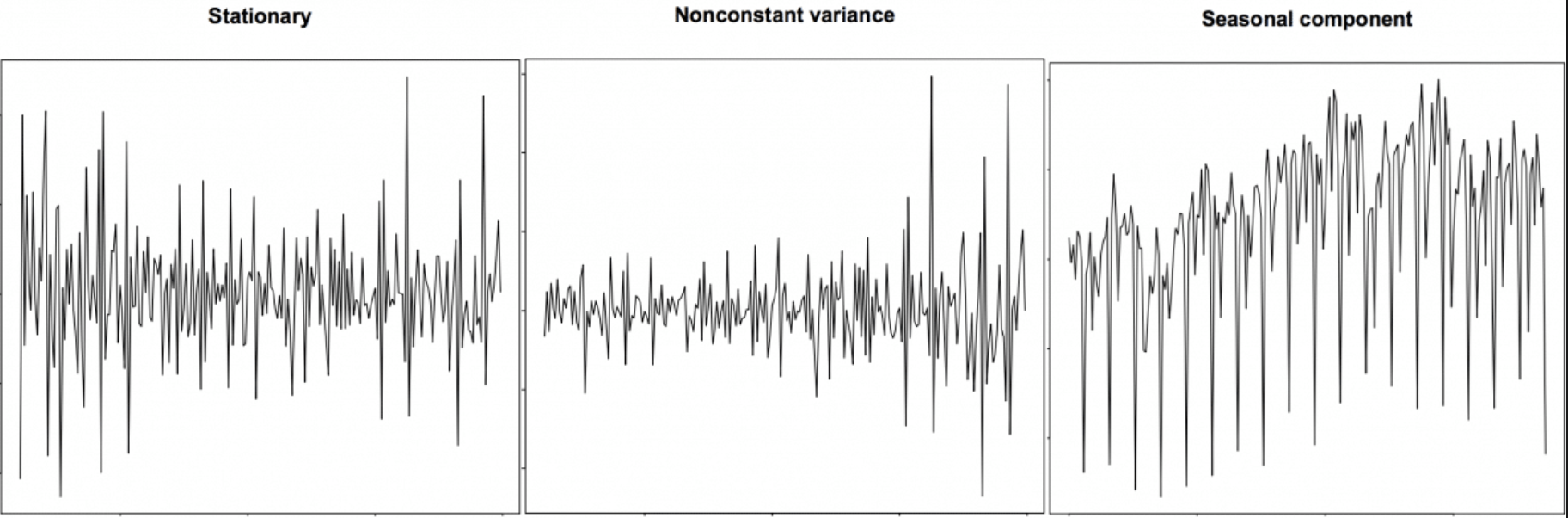
|  |
| --- |
| from pandas import read\_csv  import matplotlib.pyplot as plt  import statsmodels.tsa.arima.model as sma  from sklearn.metrics import mean\_squared\_error  from math import sqrt  PATH = "/Users/pm/Desktop/DayDocs/data/"  series = read\_csv(PATH + 'daily-min-temperatures.csv', header=0, index\_col=0)  # Split the data set so the test set is 7.  NUM\_TEST\_DAYS = 7  X = series.values  size = len(X) - NUM\_TEST\_DAYS  train, test = X[0:size], X[size:]  # Create a list with the training array.  history = [x for x in train]  predictions = []  # predict() receives the model coefficients and all past data (t-1, t-2, t-2) etc.  def predict(coef, history):  yhat = 0.0  for i in range(1, len(coef) + 1):  # Make the prediction (yhat)  # This multiplies L1coeff\*L1  # and L2coeff\*L2 if it exists  # and L3coeff\*L3 if it exists  yhat += coef[i - 1] \* history[-i]  return yhat # Return the prediction.  for t in range(len(test)):  print("History length: " + str(len(history)))  #################################################################  # Model building and prediction section.  model = sma.ARIMA(history, order=(1, 0, 0)).fit()  print("Model parameters: " + str(model.arparams))  # Get the ar\_modle parameters.  ar\_coef = model.arparams  # Make the prediction.  yhat = predict(ar\_coef, history)  #################################################################  predictions.append(yhat) # Store the prediction in a list.  obs = test[t] # Get the actual current value.  history.append(obs) # Append the actual current value to the history list.  # Actual values will be used as t-1, t-2 etc next iteration.  print('>predicted=%.3f, expected=%.3f' % (yhat, obs))  rmse = sqrt(mean\_squared\_error(test, predictions))  print('Test RMSE: %.3f' % rmse)  plt.plot(test, label='Actual', marker='o', color='blue')  plt.plot(predictions, label='Predictions', marker='o', color='orange')  plt.legend()  plt.title("AR Model")  plt.show() |

## Stationarity

Stationarity refers to constant random variance. The left image in exhibits stationarity because the variance is constant and random. The middle image in Figure 2 on the other hand shows increasing variance that appears to be heteroskedastic. The variance on the right of Figure 2 appears to be somewhat cyclical.

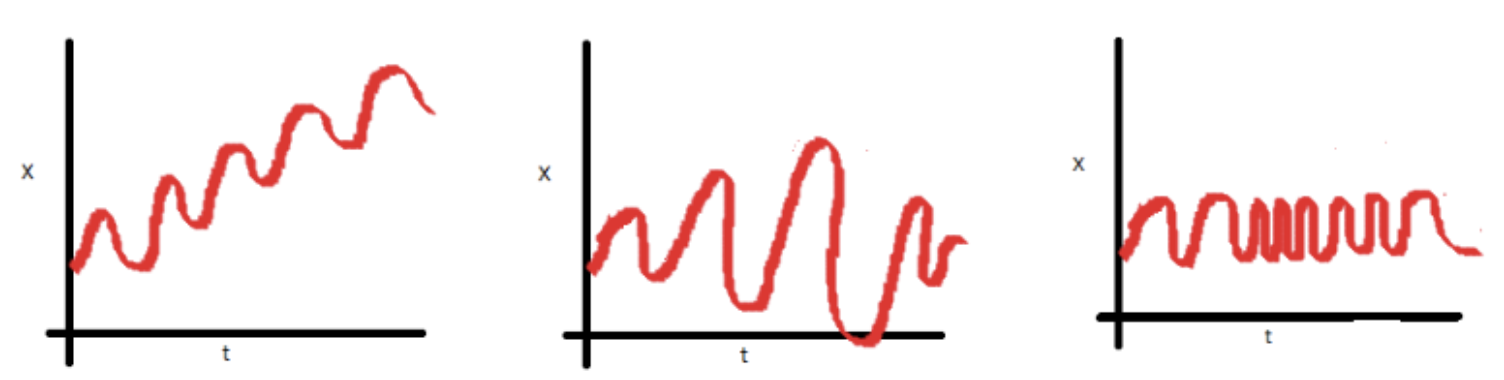
Many time series algorithms require stationary data so it is important to understand how to detect non-stationarity and how to transform it into a stationary set. For example, when trying to understand risk of markets it is important to remove trend, seasonal and cyclical data. There is also a need for stationary data when trying to understand volatility such as random consumption of hydro power which is not explained by trend, cyclical or seasonal factors.

Figure 2: Stationary versus Non-stationary Variance



Non-stationary data can contain trend and seasonal data. A stationary series is one where trend, seasonal and cyclical fluctuations have been removed. The result leads to a mean, variance and covariance that do not vary with time. Figure 3 shows a stationary series on the right and non-stationary series in the middle and on the left.

Figure 3: Trend on the left, uneven variance in the middle, and stationarity on the right.



## Differencing

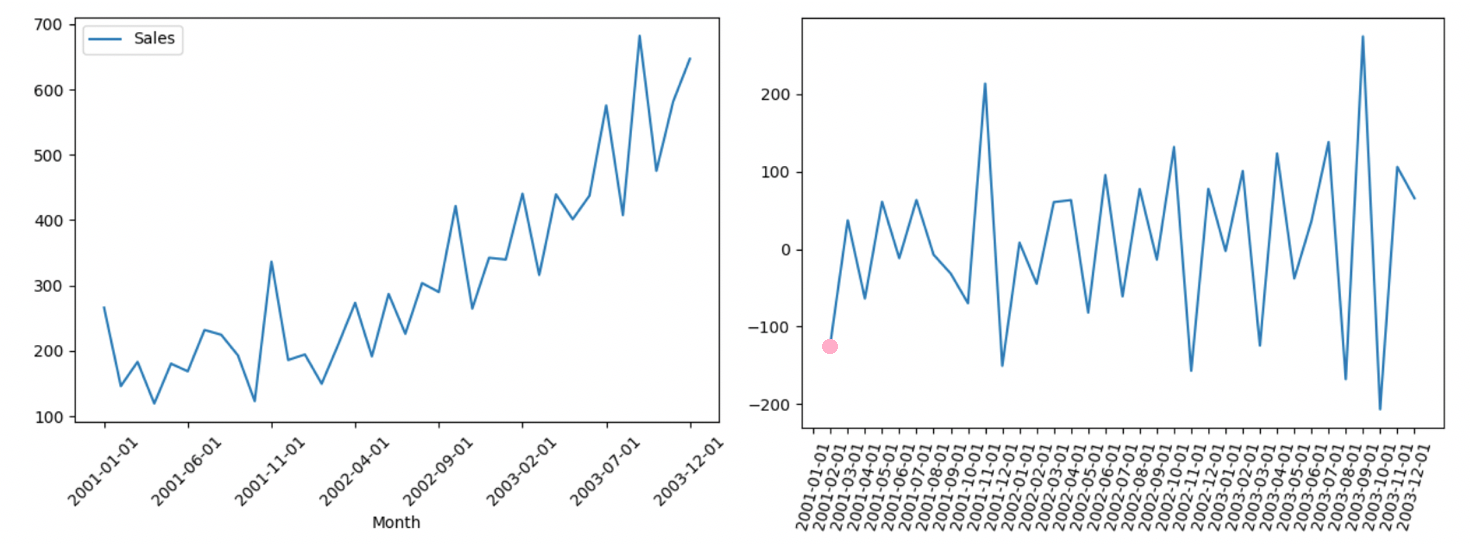
Differencing is a popular transformation that helps to stabilize the mean of a time series. Differencing removes changes in the level of a time series, and therefore eliminates (or reduces) trend and seasonality. Differencing is done to eliminate series dependence over time. This dependence includes trend and seasonality.

|  |
| --- |
| diff = df.diff()  plt.plot(diff)  plt.xticks(rotation=75)  plt.show() |

Example 3: Differencing

This example shows how to transform this undifferenced data set into a differenced data set. The graph on the right of Figure 4 shows non-stationary data. The graph on the right of Figure 4 shows the same data set after differencing.

Figure 4: Transforming Undifferenced Data on the Left to Differenced Data on the Right



Here is the full code sample:

|  |
| --- |
| import pandas as pd  import datetime  import matplotlib.pyplot as plt  PATH = "/Users/pm/Desktop/DayDocs/data/"  FILE = 'shampoo.csv'  df = pd.read\_csv(PATH + FILE, index\_col=0)  df.info()  # Plot data before differencing.  df.plot()  plt.xticks(rotation=45)  plt.show()  # Perform differencing.  dfDifferenced = df.diff()  # Plot data after differencing.  plt.plot(dfDifferenced)  plt.xticks(rotation=75)  plt.show() |

Example 4: Manually Performing Differencing

This example shows how to manually perform the same differencing routine that was implemented in Example 3. In this case:

Differenced sales = - = 145.9 – 266 = -120.1.

Table 1: Sales Differencing Calculations

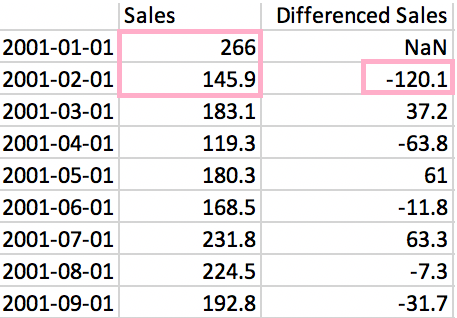


Table 1 shows how the differenced value is calculated for February 1, 2001. This differenced value is also denoted by the pink dot in Figure 4.

Exercise 2 (1 mark)

Starting with Example 4, show the calculations by hand that are needed to calculate the differenced value of $37.20 for March 1st 2001.

|  |
| --- |
| Differenced sales = - = 183.1 – 145.9 = 37.2 |

### Avoiding Too Much Differencing

Differencing can be performed multiple times. However, it is important not to over-difference the data. If you cannot decide which differenced series to choose then choose the series with the least amount of variance.

### Augmented Dickey-Fuller Test (ADF)

The ADF test is often used to determine if a series is stationary. The null hypothesis and alternate hypothesis are shown below.

: The time series is non-stationary.

: The time series is stationary.

If the p-value of the test is less than 0.05 then you reject the null hypothesis and differencing is not needed.

Example 5: ADF Test

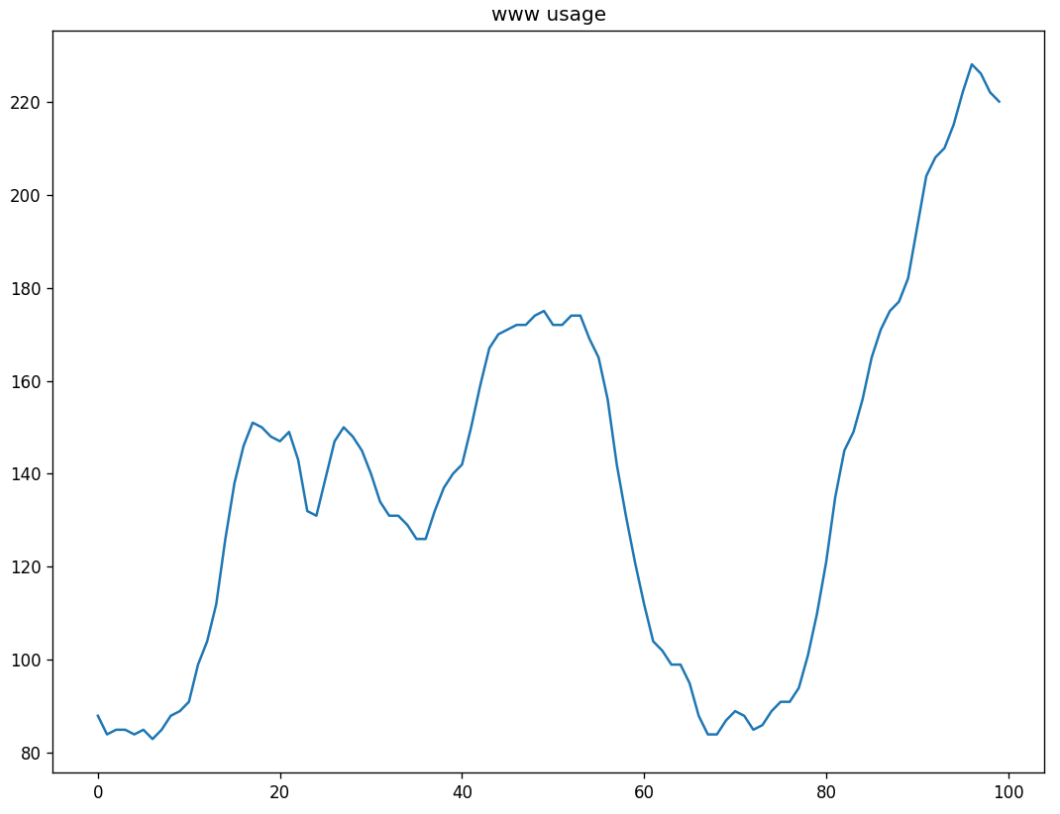
This example shows how to perform the ADF test to determine if the wwwusage data is stationary. Figure 5 shows the www usage data plotted over time. The result of the ADF test is:

ADF Statistic: -2.464240

p-value: 0.124419

The p-value is insignificant so the data is not stationary so more differencing is needed.

Figure 5: www usage data plotted over time



Here is the full example:

|  |
| --- |
| import numpy as np, pandas as pd  from statsmodels.graphics.tsaplots import plot\_acf, plot\_pacf  import matplotlib.pyplot as plt  plt.rcParams.update({'figure.figsize':(9,7), 'figure.dpi':120})  # Import data  df = pd.read\_csv(  "https://raw.githubusercontent.com/selva86/datasets/master/wwwusage.csv", \  names=['value'], header=0)  print(df)  df.value.plot()  plt.title("www usage")  plt.show()  from statsmodels.tsa.stattools import adfuller  result = adfuller(df.value.dropna())  print('ADF Statistic: %f' % result[0])  print('p-value: %f' % result[1]) |

Exercise 3 (8 marks)

Starting with Example 5, difference the data a second time. Example 3 shows how to perform differencing in code. After differencing the data, plot the data. Show your new plot here after performing second-order differencing:

|  |
| --- |
| Chart, line chart, histogram  Description automatically generated |

Then perform an ADF test on your second-order differenced data. Show your code here:

|  |
| --- |
| import numpy as np, pandas as pd from statsmodels.graphics.tsaplots import plot\_acf, plot\_pacf import matplotlib.pyplot as plt plt.rcParams.update({'figure.figsize':(9,7), 'figure.dpi':120})  # Import data df = pd.read\_csv( "/Users/hyerimshin/Desktop/wwwusage.csv", names=['value'], header=0)  df.info()  # Plot data before differencing. df.plot() plt.xticks(rotation=45) plt.show()  # Perform differencing. dfDifferenced = df.diff()  # Plot data after differencing. plt.plot(dfDifferenced) plt.xticks(rotation=75) plt.show()  from statsmodels.tsa.stattools import adfuller result = adfuller(df.value.dropna()) print('ADF Statistic: %f' % result[0]) print('p-value: %f' % result[1]) |

Show the p-value that is obtained by your ADF test above. Indicate if additional differencing is needed.

|  |
| --- |
| I think we need additional differencing since pa value is bigger than 0.05 |

# ARIMA for Differencing

Today we will introduce the full ARIMA model. The I in ARIMA stands for integrated and refers to the order differencing, d, performed on the time series observations before predictions are made in the linear regression model.

Differencing is used to ensure that all patterns are removed from the data. In effect, the differencing will make the data stationary. Stationary data has constant random variance.

When making manual predictions, we must perform this differencing of the dataset prior to calling the predict() function. Below is a function that implements differencing of the entire dataset.

Example 6: Adding Differencing to the ARIMA model

This model implements day ahead predictions with differencing. To begin, add this code at the top of Example 2.

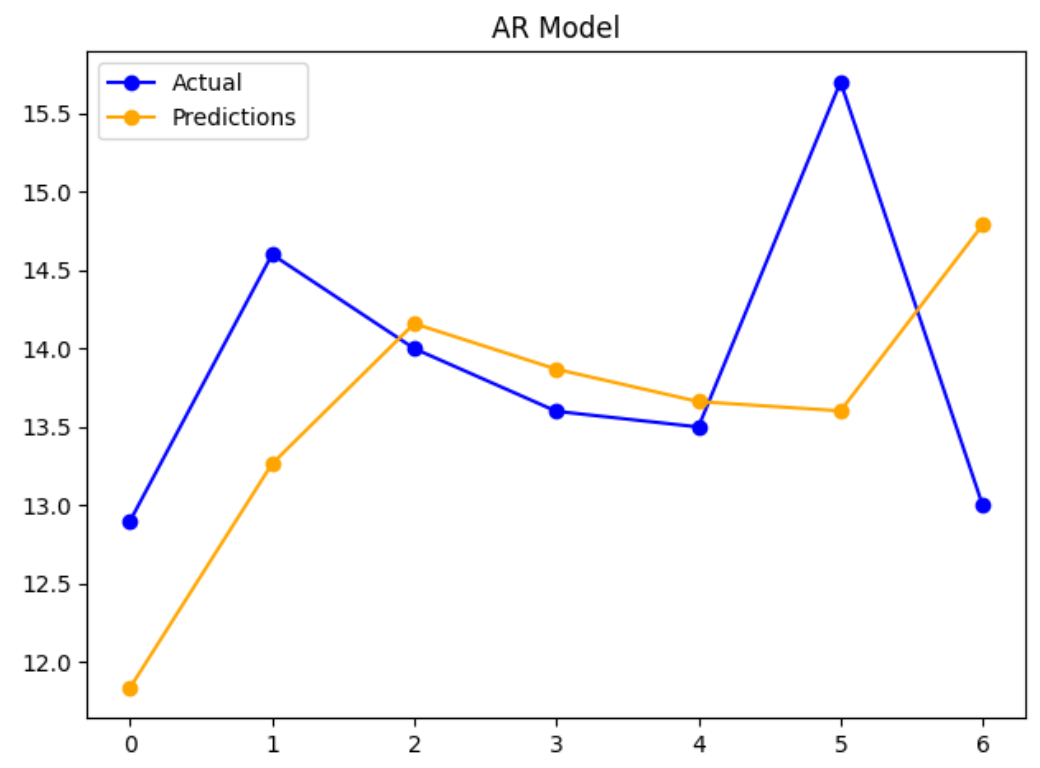
|  |
| --- |
| import numpy as np  def difference(dataset):  diff = list()  for i in range(1, len(dataset)):  value = dataset[i] - dataset[i - 1]  diff.append(value)  return np.array(diff) |

Then, replace the model building and prediction section of Example 2 with this version which uses the p,d,q terms.

|  |
| --- |
| #################################################################  # Model building and prediction section.  model = sma.ARIMA(history, order=(1, 1, 1)).fit()  ar\_coef, ma\_coef = model.arparams, model.maparams  resid = model.resid  diff = difference(history)  yhat = history[-1] + predict(ar\_coef, diff) + predict(ma\_coef, resid)  ################################################################# |

The RMSE of 1.232 is the best fitting yet compared to the ARMA model (RMSE=1.4050) and the other models which perform more poorly.

RMSE: 1.232



## Grid Searching ARIMA Parameters

Example 7: Grid searching ARIMA p,d,q Parameters

It is possible to optimize ARIMA parameters through a grid search. This article discusses a novel way to iterate through all possible ARIMA combinations with different values for p, d and q. For more information see:

<https://machinelearningmastery.com/grid-search-arima-hyperparameters-with-python/>

You may want to try this routine but you are not expected to run it. This grid search is painfully slow. If you do run it though you will notice that it prints out an MSE rating for each of the combinations.

The first leading model candidate is ARIMA(0,1,2). The parameters for the first model candidate are p=0, d=1 and q=2. In other words, p=0 implies there is no (AR) autoregressive component. The parameter d=1 implies first order differencing is required. q=2 implies that two lags of the (MA) moving average which are used. Remember that the moving average component is referring to residual error in this case and is not to be confused with weighted moving average.

ARIMA(0, 0, 0) MSE=15.818

ARIMA(0, 0, 1) MSE=8.882

ARIMA(0, 0, 2) MSE=7.471

ARIMA(0, 1, 0) MSE=6.829

ARIMA(0, 1, 1) MSE=6.325

ARIMA(0, 1, 2) MSE=5.461

ARIMA(0, 2, 0) MSE=16.157

ARIMA(0, 2, 1) MSE=6.835

ARIMA(0, 2, 2) MSE=6.331

ARIMA(1, 0, 0) MSE=6.091

ARIMA(1, 0, 1) MSE=6.016

ARIMA(1, 0, 2) MSE=5.434

ARIMA(1, 1, 0) MSE=6.604

You are not expected to run this code but in case you want or need to here is the code:

|  |
| --- |
| from pandas import read\_csv  import matplotlib.pyplot as plt  from statsmodels.tsa.arima\_model import ARIMA  from sklearn.metrics import mean\_squared\_error  from math import sqrt  import warnings  PATH = "/Users/pm/Desktop/DayDocs/data/"  series = read\_csv(PATH + 'daily-min-temperatures.csv', header=0, index\_col=0)  # Evaluate an ARIMA model for a given order (p,d,q).  def evaluate\_arima\_model(X, arima\_order):  # Prepare training dataset.  train\_size = int(len(X) \* 0.66)  train, test = X[0:train\_size], X[train\_size:]  history = [x for x in train]    # Make predictions.  predictions = list()    for t in range(len(test)):  model = ARIMA(history, order=arima\_order)  model\_fit = model.fit(disp=0)  yhat = model\_fit.forecast()[0]  predictions.append(yhat)  history.append(test[t])    # Calculate out of sample error,  error = mean\_squared\_error(test, predictions)  return error  # Evaluate combinations of p, d and q values for an ARIMA model.  def evaluate\_models(dataset, p\_values, d\_values, q\_values):  dataset = dataset.astype('float32')  best\_score, best\_cfg = float("inf"), None  for p in p\_values:  for d in d\_values:  for q in q\_values:  order = (p, d, q)  try:  mse = evaluate\_arima\_model(dataset, order)  if mse < best\_score:  best\_score, best\_cfg = mse, order  print('ARIMA%s MSE=%.3f' % (order, mse))  except:  continue  print('Best ARIMA%s MSE=%.3f' % (best\_cfg, best\_score))  # Set parameter ranges.  p\_values = [0, 1, 2, 4, 6, 8, 10]  d\_values = range(0, 3)  q\_values = range(0, 3)  warnings.filterwarnings("ignore")  # Evaluate performance.  evaluate\_models(series.values, p\_values, d\_values, q\_values) |

## Auto-ARIMA

An auto\_arima() function can find an optimal p,d,q combination more quickly than the grid search in Example 7. We will look more at auto\_arima() during the lesson that follows this lab. For now, here is an example to get started with the function.

Example 8: Optimizing p,d,q parameters with auto\_arima()

This example shows how to automate the process to find a reasonably optimal solution for making predictions with the minimum daily temperatures data set. This example searches for the optimal p, d, q combination at each iteration through the loop with the most recent observations from the data set. Starting with Example 6, add a reference to the required package;

|  |
| --- |
| import pmdarima as pm |

Note: If you have trouble with pmdarima uninstall and reinstall with pip.

Note:

If you receive this error;

ERROR: Could not install packages due to an OSError: [WinError 5] Access is denied:

On windows try a pip install when running PyCharm as administrator.

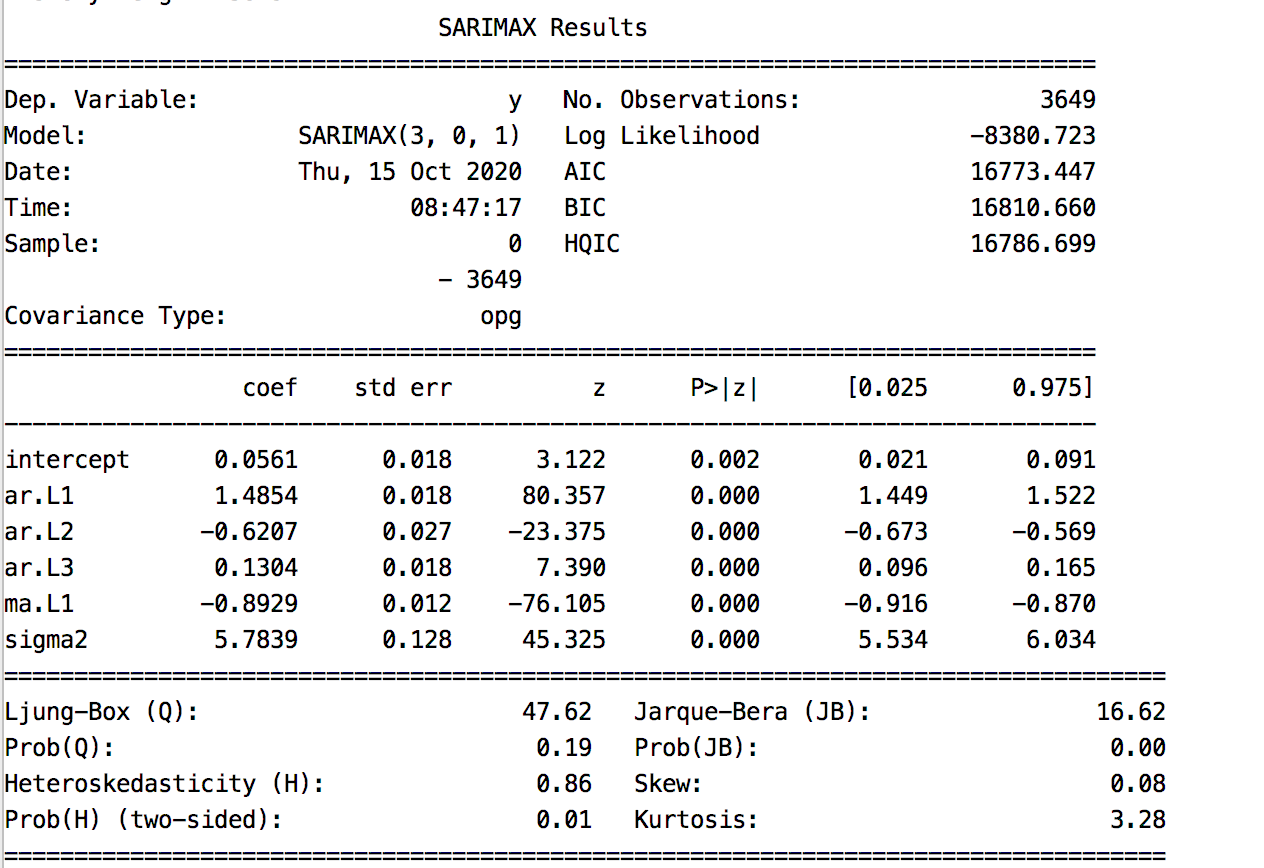
OR try running the install and example 8 in Spyder.

Next, swap out the model building and prediction section with this version.

|  |
| --- |
| #################################################################  # Model building and prediction section.  # Build day ahead model.  model = pm.auto\_arima(history, start\_p=1, start\_q=1,  test='adf',  max\_p=3, max\_q=3, m=0,  start\_P=0, seasonal=False,  error\_action='ignore',  suppress\_warnings=True,  stepwise=True)  print(model.summary())  fc, confint = model.predict(n\_periods=1,  return\_conf\_int=True)  yhat = fc[0]  ################################################################# |

The results are decent. The RMSE is 1.246. This is very good performance when compared to all other models but ARIMA(p=1, d=1,q=1) model was able to achieve an RMSE of 1.232.

Model summaries are displayed during each iteration. The routine happened to choose an ARIMA(p=3, d=0, q=1) model at each iteration. All coefficients shown in these models are statistically significant.



To practice applying the material discussed in this lab, this next set of exercises will use the **daily-total-female-births.csv** file as a data source.

Exercise 4 (1 mark)

After loading the **daily-total-female-births.csv** data source into a data frame, print the data frame with the head() instruction to see what is in it. Show your contents here:

|  |
| --- |
| Table  Description automatically generated with medium confidence |

Exercise 5 (1 mark)

Print the data frame with the describe() function to observe the mean, standard deviation and percentiles. Show the output here:

|  |
| --- |
| Table  Description automatically generated |

Exercise 6 (4 marks)

Using the auto\_arima() function while looping through the test data, find the best fitting combination of p, d and q lag values. Show a screenshot of the model summary for the last combination here:

|  |
| --- |
| Text  Description automatically generated  Graphical user interface, text  Description automatically generated |

Show the RMSE here:

|  |
| --- |
| Test RMSE: 7.467 |

Exercise 7 (1 mark)

For model summary in the last iteration, what is the optimal p, d, q combination for the ARIMA model given 7 test days? Are the coefficients statistically significant?

|  |
| --- |
| 2, 0 , 1 |

Exercise 8 (3 marks)

Using the model summary that you obtained during the last iteration while using the auto\_arima() function, write out the equation that is generated for your ARMA model. You may ignore the constant. Show your equation here:

|  |
| --- |
| Text  Description automatically generated |