Table of Contents

[Correlations 1](#_Toc83639081)

[Least Squares Regression 3](#_Toc83639082)

[Autocorrelation 9](#_Toc83639083)

[ACF vs. PACF Plots 9](#_Toc83639084)

[Autoregressive Models 12](#_Toc83639085)

[Optimizing the Model 15](#_Toc83639086)

[Predicting the Future with Auto-regressive Models 18](#_Toc83639087)

## Correlations

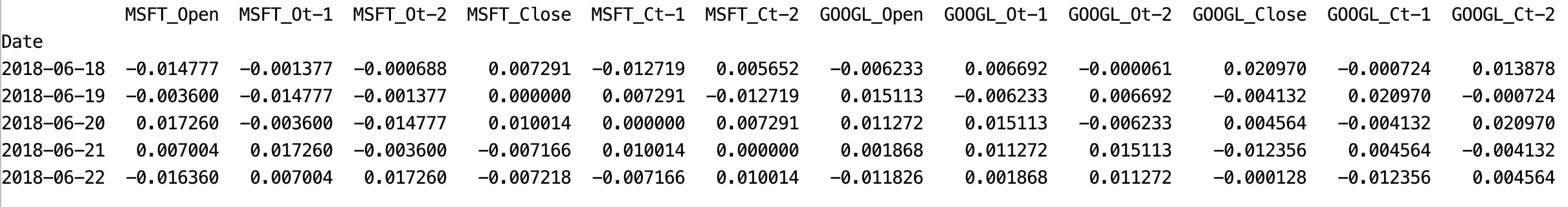
Just like with other data models, time series predictions improve when the predictor variables are highly correlated with the target variable. Figure 1 shows relative correlation between Microsoft Open price percentage changes with lagged Microsoft and Google **percent price change**s.

Example : Plotting Correlations

This example shows how to generate a correlation plot for percentage changes in Microsoft and Google stock prices. Figure 1 shows a decent correlation between Microsoft’s open price percentage change and Microsoft’s closet-1 as well as Google’s closet-1 price percentage changes.

Figure : Price Correlations between Percentage Changes in Microsoft and Google Stock Prices





Here is the code that generates the heatmap shown in Figure 1.

|  |
| --- |
| import matplotlib.pyplot as plt  import pandas as pd  import datetime  import pandas\_datareader as pdr  # Show all columns.  pd.set\_option('display.max\_columns', None)  pd.set\_option('display.width', 1000)  # Do not show warning.  pd.options.mode.chained\_assignment = None # default='warn'  ##################################################################  # CONFIGURATION SECTION  NUM\_DAYS = 1200  NUM\_TIME\_STEPS = 2  TEST\_DAYS = 30  ##################################################################  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  # Creates time shifted columns for as many time steps needed.  def backShiftColumns(df, originalColName, numTimeSteps):  dfNew = df[[originalColName]].pct\_change()  for i in range(1, numTimeSteps + 1):  newColName = originalColName[0] + 't-' + str(i)  dfNew[newColName]= dfNew[originalColName].shift(periods=i)  return dfNew  def prepareStockDf(stockSymbol, columns):  df = getStock(stockSymbol, NUM\_DAYS)  # Create data frame with back shift columns for all features of interest.  mergedDf = pd.DataFrame()  for i in range(0, len(columns)):  backShiftedDf = backShiftColumns(df, columns[i], NUM\_TIME\_STEPS)  if(i==0):  mergedDf = backShiftedDf  else:  mergedDf = mergedDf.merge(backShiftedDf, left\_index=True,  right\_index=True)  newColumns = list(mergedDf.keys())  # Append stock symbol to column names.  for i in range(0, len(newColumns)):  mergedDf.rename(columns={newColumns[i]: stockSymbol +\  "\_" + newColumns[i]}, inplace=True)  return mergedDf  columns = ['Open', 'Close']  msftDf = prepareStockDf('MSFT', columns)  aaplDf = prepareStockDf('GOOGL', columns)  mergedDf = msftDf.merge(aaplDf, left\_index=True, right\_index=True)  mergedDf = mergedDf.dropna()  print(mergedDf)  import seaborn as sns  corr = mergedDf.corr()  plt.figure(figsize = (4,4))  ax = sns.heatmap(corr[['MSFT\_Open']],  linewidth=0.5, vmin=-1,  vmax=1, cmap="YlGnBu")  plt.show() |

Exercise (2 marks)

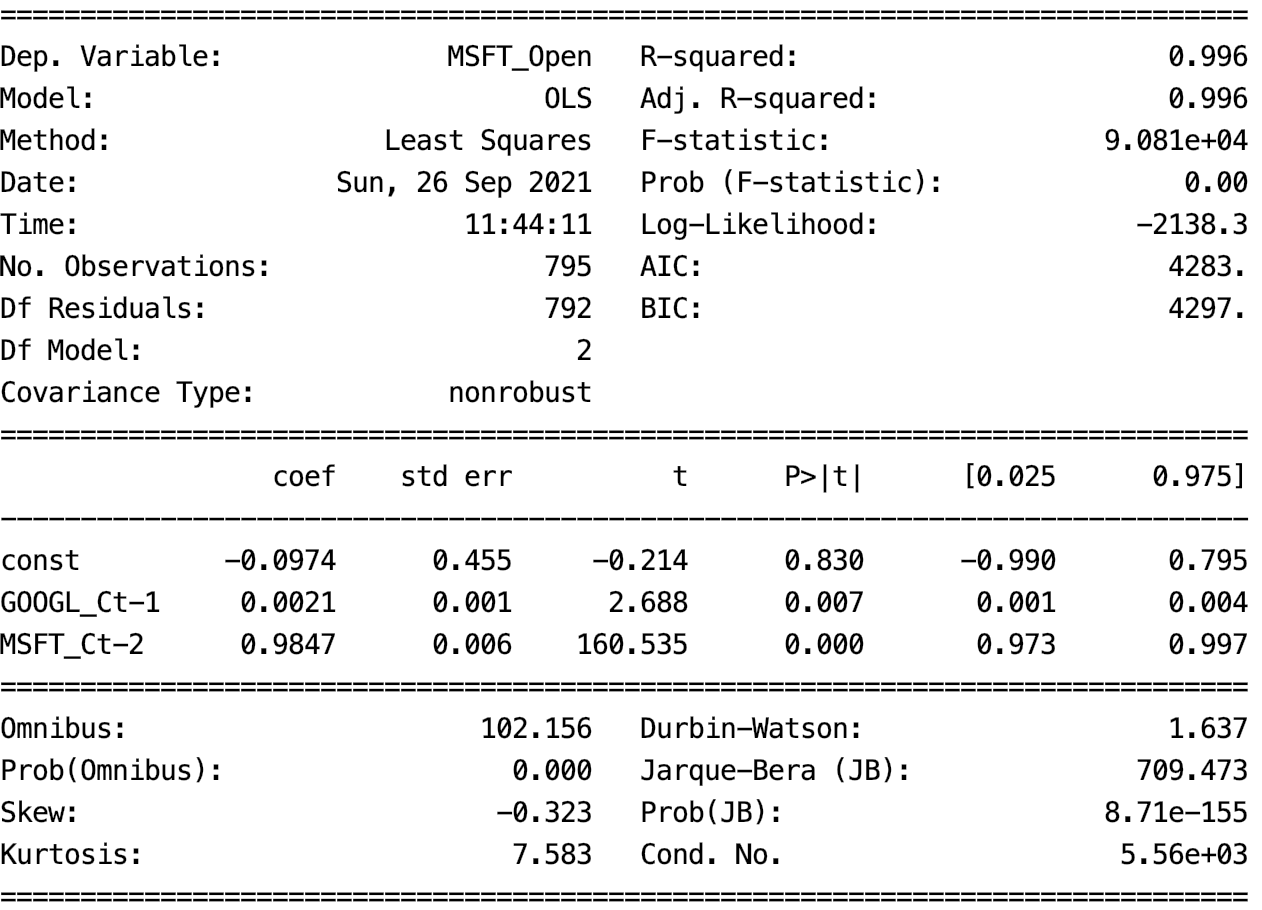
Show the correlation plot Example 1 after switching the alternate stock from Google (GOOGL) to Apple (AAPL). Show your revised plot here:

|  |
| --- |
|  |

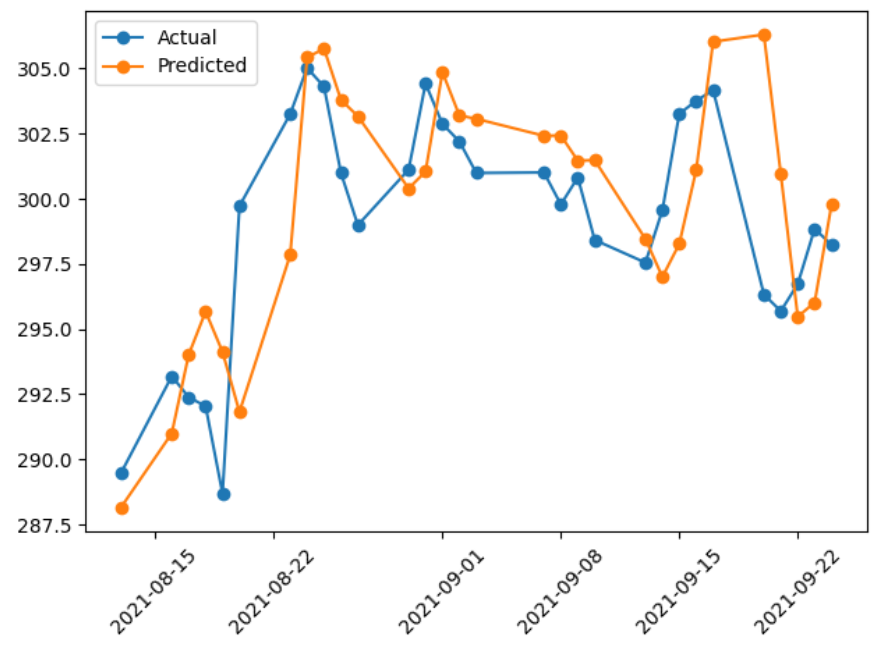
## Least Squares Regression

Example : Least Squares Regression

This example shows how to perform least squares regression to estimate Microsoft ‘Open’ stock prices. The following model suggests that GOOGL\_Ct-1 and MSFT\_Ct-2 may be significant predictor variables.



The RMSE is approximately $3.60. Notice that the model does tend to lag behind the rising and falling of the actual stock price.



To build this example, remove the **pct\_change()** function in Example 1. Then add this code to the end of Example 1.

|  |
| --- |
| xfeatures = ['MSFT\_Ct-2', 'GOOGL\_Ct-1']  X = mergedDf[xfeatures]  y = mergedDf[['MSFT\_Open']]  # Add intercept for OLS regression.  import statsmodels.api as sm  X = sm.add\_constant(X)  # Split into test and train sets. The test data must be  # the latest data range.  lenData = len(X)  X\_train = X[0:lenData-TEST\_DAYS]  y\_train = y[0:lenData-TEST\_DAYS]  X\_test = X[lenData-TEST\_DAYS:]  y\_test = y[lenData-TEST\_DAYS:]  # Model and make predictions.  model = sm.OLS(y\_train, X\_train).fit()  print(model.summary())  predictions = model.predict(X\_test)  # Show RMSE and plot the data.  from sklearn import metrics  import numpy as np  print('Root Mean Squared Error:',  np.sqrt(metrics.mean\_squared\_error(y\_test, predictions)))  plt.plot(y\_test, label='Actual', marker='o')  plt.plot(predictions, label='Predicted', marker='o')  plt.xticks(rotation=45)  plt.legend(loc='best')  plt.show() |

Exercise (Exercise 2 marks)

Change the alternate stock in Example 2 to AAPL. Do the results look more promising than with AAPL? Explain why.

|  |
| --- |
| The results look less promising with AAPL because the ‘Predicted’ line are further apart to the ‘Actual’ line when compared to using GOOGL as the alternate stock. |

Exercise (4 marks)

Change the feature set in Example 2 to only include MSFT\_Ct-1. Show the predicted versus actual plot. Also show the RMSE. How do the results compare with the original model that is built in Example 2?

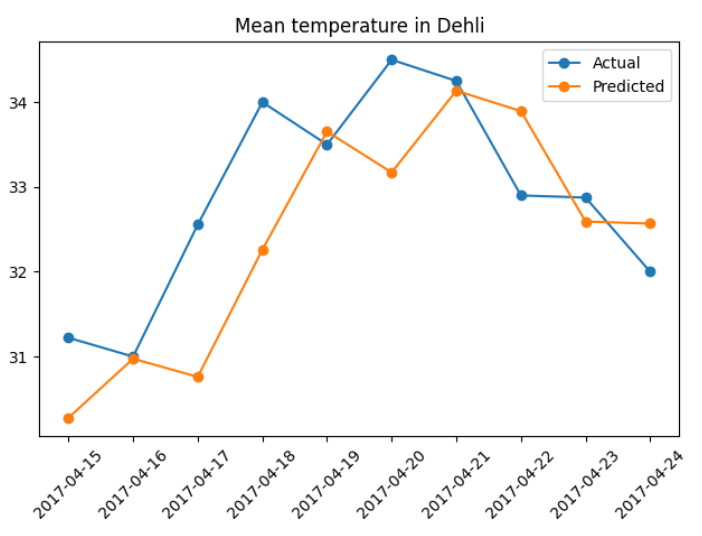
|  |
| --- |
| The ‘Predicted’ line in the plot is now almost a straight line as opposed to moving with the fluctuations from the ‘Actual’ line. |

Example : Linear Regression with Weather Data

Here is an example that uses OLS regression to predict weather in Dehli, India. The data includes variables for temperature, humidity, wind\_speed and pressure.

|  |
| --- |
| meantemp humidity wind\_speed meanpressure  date  2017-01-01 15.913043 85.869565 2.743478 59.000000  2017-01-02 18.500000 77.222222 2.894444 1018.277778  2017-01-03 17.111111 81.888889 4.016667 1018.333333 |

The prediction is decent but it does fall behind the actual trend.



In the end, the best model used the temperature from the day before.

|  |
| --- |
|  |

Here is the code for the full program:

|  |
| --- |
| import matplotlib.pyplot as plt  import pandas as pd  import numpy as np  import statsmodels.api as sm  from sklearn.preprocessing import MinMaxScaler  # Show all columns.  pd.set\_option('display.max\_columns', None)  pd.set\_option('display.width', 1000)  # Do not show warning.  pd.options.mode.chained\_assignment = None  # Load the data.  PATH = '/users/pm/desktop/daydocs/data/'  FILE = 'DailyDelhiClimateTest.csv'  df = pd.read\_csv(PATH + FILE, parse\_dates=['date'], index\_col='date')  print(df)  # Create back-shifted columns for an attribute.  def addBackShiftedColumns(df, colName, timeLags):  for i in range(1, timeLags+1):  newColName = colName + "\_t-" + str(i)  df[newColName] = df[colName].shift(i)  return df  # Build dataframe for modelling.  columns = ['meantemp', 'humidity', 'wind\_speed', 'meanpressure']  modelDf = df.copy()  NUM\_TIME\_STEPS = 3  for i in range(0, len(columns)):  modelDf = addBackShiftedColumns(modelDf, columns[i],  NUM\_TIME\_STEPS)  modelDf = modelDf.dropna()  y = modelDf[['meantemp']]  X = modelDf[[ 'meantemp\_t-1']]  # Add intercept for OLS regression.  X = sm.add\_constant(X)  TEST\_DAYS = 10  # Split into test and train sets. The test data includes  # the latest values in the data.  lenData = len(X)  X\_train = X[0:lenData-TEST\_DAYS]  y\_train = y[0:lenData-TEST\_DAYS]  X\_test = X[lenData-TEST\_DAYS:]  y\_test = y[lenData-TEST\_DAYS:]  # Model and make predictions.  model = sm.OLS(y\_train, X\_train).fit()  print(model.summary())  predictions = model.predict(X\_test)  # Show RMSE.  from sklearn import metrics  print('Root Mean Squared Error:',  np.sqrt(metrics.mean\_squared\_error(y\_test, predictions)))  # Plot the data.  xaxisValues = list(y\_test.index)  plt.plot(xaxisValues, y\_test, label='Actual', marker='o')  plt.plot(xaxisValues, predictions, label='Predicted', marker='o')  plt.xticks(rotation=45)  plt.legend(loc='best')  plt.title("Mean temperature in Dehli")  plt.show() |

## Autocorrelation

Autocorrelation refers to the correlation of a current observation with previous time steps. Partial autocorrelation represents the correlation amount specifically between each lag and the current time step.

### ACF vs. PACF Plots

The partial correlation function (PACF) provides a clearer view of the correlation of each lag with the current step since it removes all other correlations for prior lags. Figure 3 shows the Autocorrelation Function (ACF) and Partial autocorrelation (PACF) plots for sunspot activity.

The correlation of a variable value at the first step is always 1 because this is the correlation of the current variable value with itself. The blue shaded area represents the 95% confidence interval (see Figure 3). The fact that the first two lags of the PACF plot in Figure 3 appear outside the confidence interval means that the correlation between these lags are statistically significant.

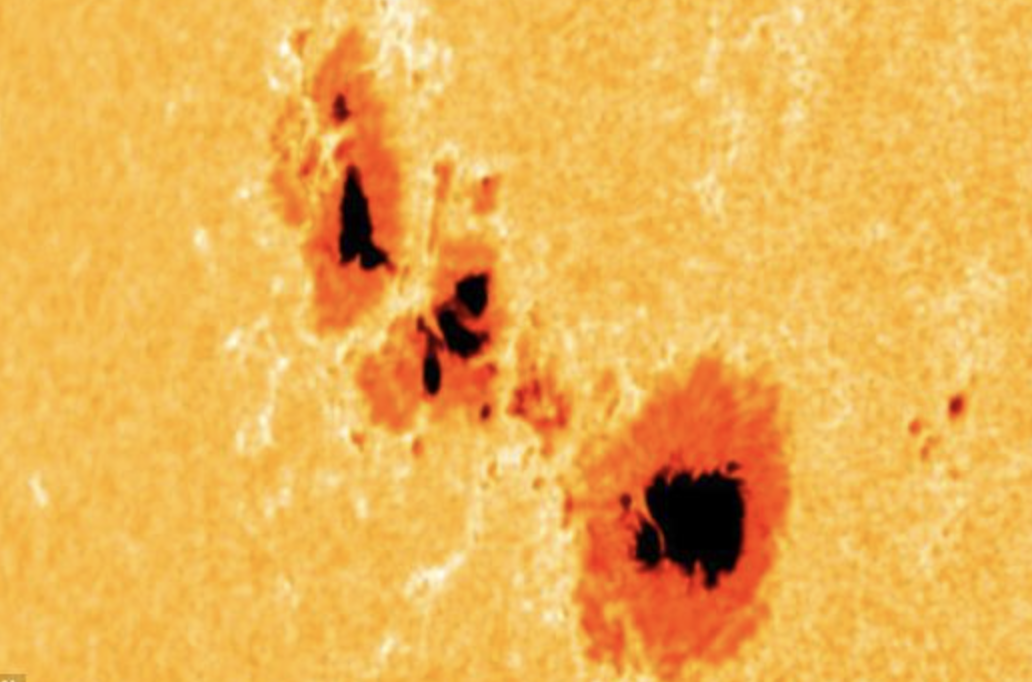
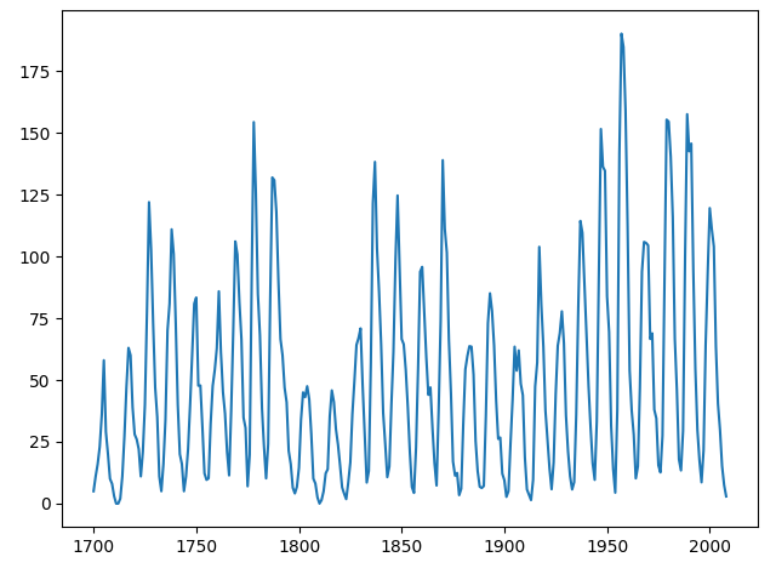
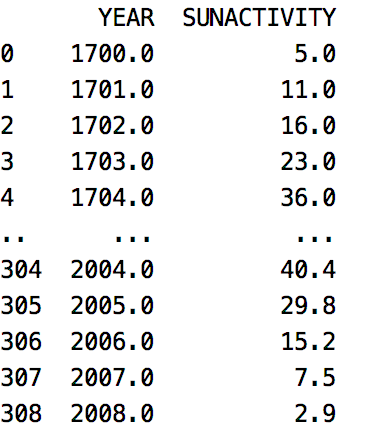
Example : Autocorrelation of Sunspot Activity

This example shows how to model yearly sunspot activity since the 1700’s.

Sunspots are temporary phenomena on the Sun's photosphere that appear as spots darker than the surrounding areas. They are regions of reduced surface temperature caused by concentrations of magnetic field flux. Their number varies according to the approximately 11-year solar cycle. Individual sunspots or groups of sunspots may last anywhere from a few days to a few months, but eventually decay. Sunspots expand and contract as they move across the surface of the Sun, with diameters ranging from 16 km (10 mi) to 160,000 km (100,000 mi). Larger sunspots can be visible from Earth without the aid of a telescope. They may travel at relative speeds of a few hundred meters per second when they first emerge.

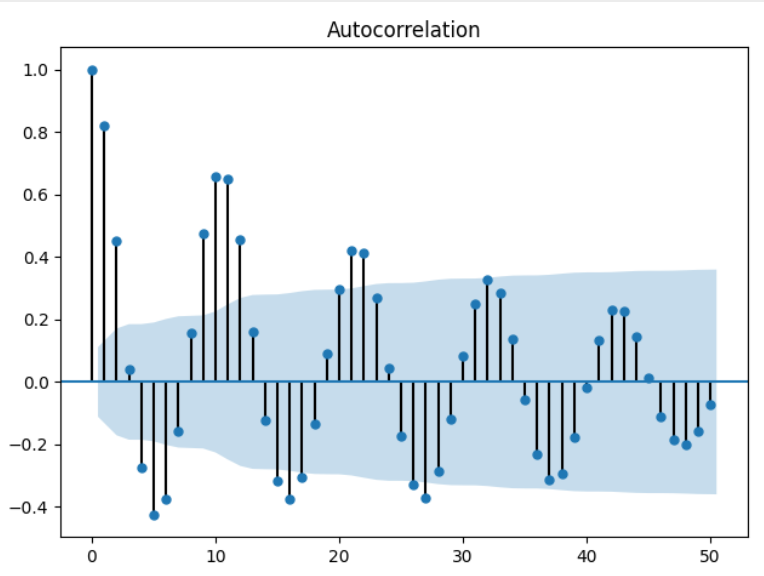
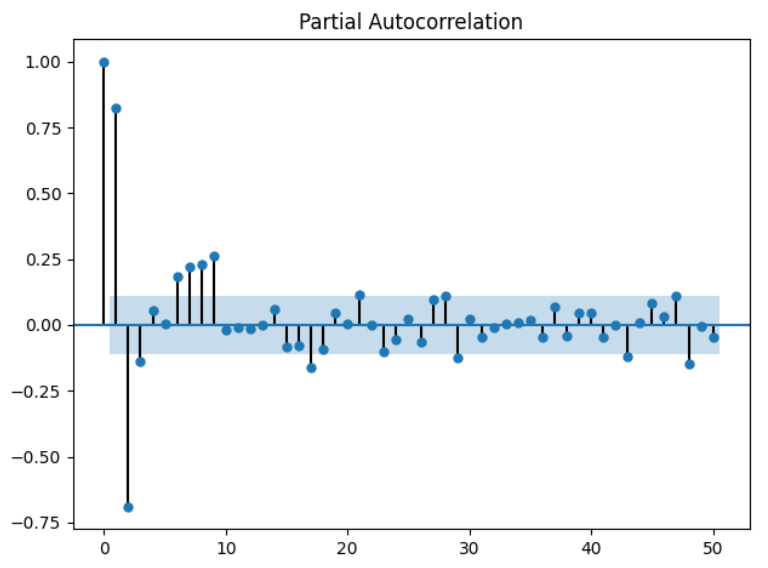
The chart and graph below present the sunspot activity. A picture of sunspots is at the right.

Figure : Sunspot Activity



The auto-correlation plot shows that the current time step is correlated to the previous time steps. The partial auto correlation function shows that the initial lag is correlated with the first 2-time lags and arguably others up until about lag 9.

Figure : Autocorrelation partial autocorrelation for sun spot data

Here is the code that presents the sun spot data:

|  |
| --- |
| import matplotlib.pyplot as plt  import statsmodels.api as sm  dta = sm.datasets.sunspots.load\_pandas().data  print(dta)  plt.plot(dta['YEAR'], dta['SUNACTIVITY'])  plt.show()  # Show autocorrelation function.  # General correlation of lags with past lags.  from statsmodels.graphics.tsaplots import plot\_acf  plot\_acf(dta['SUNACTIVITY'], lags=50)  plt.show()  # Show partial-autocorrelation function.  # Shows correlation of 1st lag with past lags.  from statsmodels.graphics.tsaplots import plot\_pacf  plot\_pacf(dta['SUNACTIVITY'], lags=50)  plt.show()  print(dta) |

Exercise (10 marks)

Using the back-shifting technique from Example 3, generate back-shifted columns and perform OLS regression using the different lags to predict sunspot activity. Use the last 10 samples in the data set for testing. Show your model summary here and ensure that it includes only significant predictors.

|  |
| --- |
|  |

Show the RMSE here:

|  |
| --- |
| Root Mean Squared Error: 20.973711524423326 |

Plot the actual test values versus predicted values here.

|  |
| --- |
|  |

Show your complete program here:

|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  import statsmodels.api as sm  dta = sm.datasets.sunspots.load\_pandas().data  print(dta)  plt.plot(dta['YEAR'], dta['SUNACTIVITY'])  plt.show()  # Show autocorrelation function.  # General correlation of lags with past lags.  from statsmodels.graphics.tsaplots import plot\_acf  plot\_acf(dta['SUNACTIVITY'], lags=50)  plt.show()  # Show partial-autocorrelation function.  # Shows correlation of 1st lag with past lags.  from statsmodels.graphics.tsaplots import plot\_pacf  plot\_pacf(dta['SUNACTIVITY'], lags=50)  plt.show()  print(dta)  # Create back-shifted columns for an attribute.  def addBackShiftedColumns(df, colName, timeLags):  for i in range(1, timeLags+1):  newColName = colName + "\_t-" + str(i)  df[newColName] = df[colName].shift(i)  return df  # Build dataframe for modelling.  columns = ['YEAR', 'SUNACTIVITY']  modelDf = dta.copy()  NUM\_TIME\_STEPS = 3  for i in range(0, len(columns)):  modelDf = addBackShiftedColumns(modelDf, columns[i],  NUM\_TIME\_STEPS)  modelDf = modelDf.dropna()  y = modelDf[['SUNACTIVITY']]  X = modelDf[[ 'SUNACTIVITY\_t-1']]  # Add intercept for OLS regression.  X = sm.add\_constant(X)  TEST\_DAYS = 10  # Split into test and train sets. The test data includes  # the latest values in the data.  lenData = len(X)  X\_train = X[0:lenData-TEST\_DAYS]  y\_train = y[0:lenData-TEST\_DAYS]  X\_test = X[lenData-TEST\_DAYS:]  y\_test = y[lenData-TEST\_DAYS:]  # Model and make predictions.  model = sm.OLS(y\_train, X\_train).fit()  print(model.summary())  predictions = model.predict(X\_test)  # Show RMSE.  from sklearn import metrics  print('Root Mean Squared Error:',  np.sqrt(metrics.mean\_squared\_error(y\_test, predictions)))  # Plot the data.  xaxisValues = list(y\_test.index)  plt.plot(xaxisValues, y\_test.values, label='Actual', marker='o')  plt.plot(xaxisValues, predictions, label='Predicted', marker='o')  plt.xticks(rotation=45)  plt.legend(loc='best')  plt.title("Mean sunspot activity")  plt.tight\_layout()  plt.show() |

How do the ACF and PACF plots suggest that a model to predict sunspot activity based on past sunspot activity is possible?

|  |
| --- |
| The ACF plot shows that the current time step is correlated to the previous time steps.  The fact that the first two lags of the PACF plot appear outside the confidence interval means that the correlation between these lags are statistically significant. |

Exercise (3 marks)

Run the following code to generate ACF and PACF plot for opening prices of Microsoft stock.

|  |
| --- |
| import numpy as np, pandas as pd  from statsmodels.graphics.tsaplots import plot\_acf, plot\_pacf  import matplotlib.pyplot as plt  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  NUM\_DAYS = 70  df = getStock('MSFT', NUM\_DAYS)  print(df)  # Plot ACF for stock.  from statsmodels.graphics.tsaplots import plot\_acf  plot\_acf(df['Open'])  plot\_pacf(df['Open'])  plt.show() |

What do the ACF and PACF plots tell us about the opening prices related to the opening price lags at past time steps? How does the PACF plot relate to the conclusion made in Exercise 3?

|  |
| --- |
| ACF and PACF plots tell us how correlated the opening price at the current time step is compared to previous time step(s).  The PACF plot relates to the conclusion made in Exercise 3 because it removes all other correlations for prior lags. |

Is the most current timestep at the left or right of the ACF and PACF?

|  |
| --- |
| The most current timestep is at the left of the ACF and PACF plots. The further right the plot is, the more time lag there is. |

Exercise (1 mark)

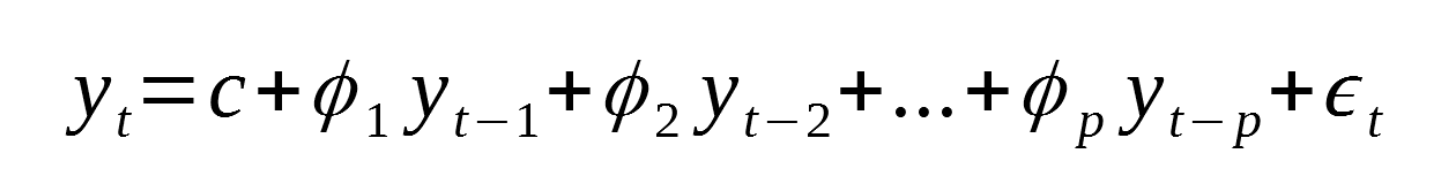
Why are the y-xis in Figure 3 ranges between -1 and 1?

|  |
| --- |
| This range indicates the level of correlation. A value of 1 is directly correlated while a value of -1 is directly inversely correlated. |

## Autoregressive Models

Autoregression models are time series models that use observations from previous time steps as input to a regression equation to predict the value at a future time step.

Where phi is a coefficient found by the model and y is an observation at a point in time the model becomes:



Where:

* p: is the order
* c: is a constant
* epsilon: noise

The statsmodel library helps to automate the generation of an autoregressive model for a single variable series.

|  |
| --- |
| model = AutoReg(train, lags=NUM\_TIME\_STEPS)  model\_fit = model.fit() |

When the model is fit it is possible to obtain the coefficients:

|  |
| --- |
| print('Coefficients: %s' % **model\_fit.params**) |

We can then use the model to make a forecast for the same number of time steps:

|  |
| --- |
| predictions = model\_fit.predict(start=len(train), end=NUM\_TEST\_DAYS, dynamic=False) |

Example : Building Autoregression Models

This example shows how to build an autoregressive model to predict sunspot activity for the most recent 10 years in the data set. We will use the ARIMA class to build our AR model. The ARIMA class offers several features that we will use very soon. For this case though, we are building an autoregression model which performs regression on the last two time steps. The number of time steps is assigned with the first argument of three in the order attribute. Will discuss the other arguments later but now set the second and third parameters of the order attribute to zero so we can have a pure AR model.

|  |
| --- |
| arma\_mod20 = ARIMA(dta['SUNACTIVITY'], order=(2, 0, 0)).fit() |

There are several ways to evaluate the model fit. We can examine the RMSE, AIC and BIC.

|  |
| --- |
| rmse = np.sqrt(mean\_squared\_error(dfTest['SUNACTIVITY'].values,  np.array(predictions)))  print('Test RMSE: %.3f' % rmse)  print('Model AIC %.3f' % model.aic)  print('Model BIC %.3f' % model.bic) |

We can also view a plot of the actual versus the predicted values (see ).

Table : Evaluating the AR Model

|  |  |
| --- | --- |
| \*\*\* Evaluating ARMA(2,0,0)  Coefficients: const 49.728505  ar.L1 1.386171  ar.L2 -0.687217  sigma2 278.758455  dtype: float64  Test RMSE: 25.145  Model AIC 2542.401  Model BIC 2557.203 |  |

Here is the code which builds the model and presents the results for evaluation of the sunspot activity predictions.

|  |
| --- |
| import pandas as pd  import matplotlib.pyplot as plt  import statsmodels.api as sm  from statsmodels.tsa.arima.model import ARIMA  from sklearn.metrics import mean\_squared\_error  import numpy as np  dta = sm.datasets.sunspots.load\_pandas().data  dta.index = pd.Index(sm.tsa.datetools.dates\_from\_range('1700', '2008'), freq='Y')  # Show autocorrelation function.  # General correlation of lags with past lags.  from statsmodels.graphics.tsaplots import plot\_acf  plot\_acf(dta['SUNACTIVITY'], lags=50)  plt.show()  # Split the data.  NUM\_TEST\_YEARS = 10  lenData = len(dta)  dfTrain = dta.iloc[0:lenData - NUM\_TEST\_YEARS, :]  dfTest = dta.iloc[lenData-NUM\_TEST\_YEARS:,:]  def buildModelAndMakePredictions(AR\_time\_steps, dfTrain, dfTest):  # This week we will use the ARIMA model.  model = ARIMA(dfTrain['SUNACTIVITY'], order=(AR\_time\_steps, 0, 0),  freq='Y').fit()  print("\n\*\*\* Evaluating ARMA(" + str(AR\_time\_steps) + ",0,0)")  print('Coefficients: %s' %model.params)  # Strings which can be converted to time stamps are passed in.  # For this case the entire time range for the test set is represented.  predictions = model.predict('1999-12-31', '2008-12-31', dynamic=True)  rmse = np.sqrt(mean\_squared\_error(dfTest['SUNACTIVITY'].values,  np.array(predictions)))  print('Test RMSE: %.3f' % rmse)  print('Model AIC %.3f' % model.aic)  print('Model BIC %.3f' % model.bic)  return model, predictions  print(dfTest)  arma\_mod20, predictionsARMA\_20 = buildModelAndMakePredictions(2, dfTrain, dfTest)  plt.plot(dfTest.index, dfTest['SUNACTIVITY'],  label='Actual Values', color='blue')  plt.plot(dfTest.index, predictionsARMA\_20,  label='Predicted Values AR(20)', color='orange')  plt.legend(loc='best')  plt.show() |

Example : Comparing Predictions

In this example we will compare an AR(3) model with the AR(2) model. The AR(3) model does slightly better.

|  |  |  |
| --- | --- | --- |
| \*\*\* Evaluating ARMA(2,0,0)  Coefficients: const 49.728505  ar.L1 1.386171  ar.L2 -0.687217  sigma2 278.758455  dtype: float64  Test RMSE: 25.145  Model AIC 2542.401  Model BIC 2557.203 | \*\*\* Evaluating ARMA(3,0,0)  Coefficients: const 49.767980  ar.L1 1.303286  ar.L2 -0.520229  ar.L3 -0.120298  sigma2 274.716644  dtype: float64  Test RMSE: 22.480  Model AIC 2540.073  Model BIC 2558.575 |  |

To build this example, add this code just after where the AR(2) model is declared.

|  |
| --- |
| arma\_mod30, predictionsARMA\_30 = buildModelAndMakePredictions(3, dfTrain, dfTest) |

Then place this code where the other lines are plotted.

|  |
| --- |
| plt.plot(dfTest.index, predictionsARMA\_30,  label=**'Predicted Values AR(30)'**, color=**'brown'**) |

## Optimizing the Model

As you can see in , choosing different time step values for the autoregression model can lead to different results. We can actually automate the process of selecting the best time step.

Example : Selecting the time step order

This example shows how to find the best time step for our AR model. For this example, we are displaying the BIC (Bayesian Information Criterion) scores for AR models which include lag counts that range between 0 and 10.

|  |
| --- |
| dict\_values([ 0  0 3174.049905  1 2830.369176  2 2637.569703  3 2638.070335  4 2642.878700  5 2648.610908  6 2645.594430  7 2635.308015  8 2625.737548  9 2611.689366  10 2617.421383, (9, 0)]) |

Adding this code to will generate and display the information criterion or **ic** statistic for the past 10-time steps. For this model, AR(9) appears to be the best fit since the BIC score is the lowest. However, AR(2) offers a simpler and competitive option.

|  |
| --- |
| print("\n\*\*\* Comparing AR Model Time Steps")  import warnings  warnings.filterwarnings("ignore")  res = sm.tsa.arma\_order\_select\_ic(dta['SUNACTIVITY'], max\_ar=10, max\_ma=0, ic='bic')  print(res.values()) |

Exercise : (3 marks)

Plot the results for your AR(9) model with the AR(3), AR(2) predictions and the actual values. Show the graph here:

|  |
| --- |
|  |

Exercise : (3 marks)

Compare the AIC, BIC and RMSE values for each of the three AR models in a table. Show your table here. Please make it easy to read.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | **\*\*\* Evaluating ARMA(2,0,0)** | **\*\*\* Evaluating ARMA(3,0,0)** | **\*\*\* Evaluating ARMA(9,0,0)** | | Coefficients: const 49.728506 | Coefficients: const 49.767980 | Coefficients: const 49.450261 | | ar.L1 1.386171 | ar.L1 1.303286 | ar.L1 1.162713 | | ar.L2 -0.687217 | ar.L2 -0.520229 | ar.L2 -0.402611 | |  | ar.L3 -0.120298 | ar.L3 -0.155085 | |  |  | ar.L4 0.141164 | |  |  | ar.L5 -0.092369 | |  |  | ar.L6 0.017273 | |  |  | ar.L7 0.037691 | |  |  | ar.L8 -0.066205 | |  |  | ar.L9 0.245096 | | sigma2 278.758454 | sigma2 274.716642 | sigma2 223.760774 | | dtype: float64 | dtype: float64 | dtype: float64 | | Test RMSE: 25.145 | Test RMSE: 22.480 | Test RMSE: 14.885 | | Model AIC 2542.401 | Model AIC 2540.073 | Model AIC 2492.359 | | Model BIC 2557.203 | Model BIC 2558.575 | Model BIC 2533.064 | |

Exercise : (2 marks)

Explain in your own words why the AR(9) model is better than the AR(3) and AR(2) models.

|  |
| --- |
| AR(9) has a lower AIC and BIC compared to the AR(3) and AR(2) models which indicates a better fit. This shows, for this dataset, the more time steps there are, the better the model will perform. |

Exercise : (3 marks)

Using the equation definition for autoregressive models. Show the equation for the AR(3) model using the model coefficients here. (Refer to to obtain the model coefficients)

|  |
| --- |
| yt = 49.767980 + 1.303286 + -0.520229 + -0.120298 |

Example : Australian Temperatures Autocorrelation

We will start with the minimum daily temperatures in Melbourne Australia. The ACF and PACF plots for daily temperature are displayed in Figure 4.

Figure : ACF and PACF plots for Melbourne Daily Temperatures

|  |
| --- |
|  |

Here is the code that loads and plots the data.

|  |
| --- |
| from pandas import read\_csv  import matplotlib.pyplot as plt  from statsmodels.tsa.ar\_model import AutoReg  from sklearn.metrics import mean\_squared\_error  from math import sqrt  # Load the data.  PATH = "/Users/pm/Desktop/DayDocs/data/"  series = read\_csv(PATH + 'daily-min-temperatures.csv',  header=0, index\_col=0, parse\_dates=True, squeeze=True)  # Plot ACF.  from statsmodels.graphics.tsaplots import plot\_acf  plot\_acf(series, lags=20)  plt.show()  # Plot PACF.  from statsmodels.graphics.tsaplots import plot\_pacf  plot\_pacf(series, lags=20)  plt.show() |

Example : Building a 3-Day Model

This example shows how to build a 3-day model to predict minimum daily temperatures for Melbourne Australia. This technique is really useful because it helps us to focus our efforts to optimize our 3-day forecast. To build this example, add this code to the end of the code in Example 8.

|  |
| --- |
| NUM\_TEST\_DAYS = 3  # Split dataset into test and train.  X = series.values  lenData = len(X)  train = X[0:lenData-NUM\_TEST\_DAYS]  test = X[lenData-NUM\_TEST\_DAYS:]  # Train.  model = AutoReg(train, lags=3)  model\_fit = model.fit()  print('Coefficients: %s' % model\_fit.params)  print(model\_fit.summary())  # Make predictions.  predictions = model\_fit.predict(start=len(train),  end=len(train)+len(test)-1,  dynamic=False)  for i in range(len(predictions)):  print('predicted=%f, expected=%f' % (predictions[i], test[i]))  rmse = sqrt(mean\_squared\_error(test, predictions))  print('Test RMSE: %.3f' % rmse)  # Plot results.  plt.plot(test, marker='o', label='actual')  plt.plot(predictions, color='brown', linewidth=4,  marker='o', label='predicted')  plt.legend()  plt.show() |

Now we have a 3-day model with some degree of confidence that it will predict within a specific range. Three days may not be that impressive. However, the AR model gives us a chance to build and evaluate a 5-day or a 7-day model while helping to ensure the predictions within that range are reasonably accurate.

Remember to proceed with caution when using the AR model. The AR model does not account for random variance and unforeseen shocks to the environment. It can also be adversely affected by cyclical, seasonal or long-term trends. Other variables may also affect the AR model performance.

Exercise (2 marks)

Approximately how many lags does the PACF plot from Example 8 suggest should be used in building the model?

|  |
| --- |
| Approximately 5 lags should be used in building the model. |

Exercise (4 marks)

Try running your code for Example 9 with the number of time steps that you listed in Exercise 11. Next, view the model summary output with your new step count. Show your model summary here:

|  |
| --- |
|  |

Which coefficients are statistically significant?

|  |
| --- |
| All coefficients are p < 0.05 so they are all significant. |

Is the RMSE higher with the new set of coefficients when compared to using data from only three time steps?

|  |
| --- |
| The RMSE is slightly lower (1.508 vs 1.512) when compared to using data from only three time steps. |

### Predicting the Future with Auto-regressive Models

Updating these types of forecasts can be challenging. You might consider updating the model coefficients on a regular basis.

To make predictions with an AR model we can extract the weights from the auto-regressive process. Then we can feed in the latest observation, make a prediction and then update the time steps. This process can be repeated iteratively.

Example : Forecasting into the Future

This example shows how to make predictions in the future with the weights that are returned by the auto-regressive process. Since the model is designed with a 3-day range I am only using the model for a 3-day range. To build this example, add this code onto the code from Example 9.

|  |
| --- |
| # Use model coefficients from autoregression to make a prediction.  def makePrediction(t\_1, t\_2, t\_3):  intercept = 1.88820768  t1Coeff = 0.70018223  t2Coeff = - 0.05949822  t3Coeff = 0.19010829  prediction = intercept + t1Coeff\*t\_1\  + t2Coeff\*t\_2\  + t3Coeff\*t\_3  return prediction  testLen = len(test)  t\_1 = test[testLen-1]  t\_2 = test[testLen-2]  t\_3 = test[testLen-3]  futurePredictions = []  for i in range(0, NUM\_TEST\_DAYS):  prediction = makePrediction(t\_1, t\_2, t\_3)  futurePredictions.append(prediction)  t\_3 = t\_2  t\_2 = t\_1  t\_1 = prediction  print("Here is a one week temperature forecast: ")  print(futurePredictions) |

Exercise (5 marks)

Change the model so it becomes a 7-day model. Update the makePredictions() function. Show your updated makePredictions() function here:

|  |
| --- |
| from pandas import read\_csv import matplotlib.pyplot as plt from statsmodels.tsa.ar\_model import AutoReg from sklearn.metrics import mean\_squared\_error from math import sqrt  # Load the data. PATH = "C:\\Users\\Austin\\Desktop\\Class\\bcit\\CST\\Term 4\\COMP4949\\dataset\\" series = read\_csv(PATH + 'daily-min-temperatures.csv',  header=0, index\_col=0, parse\_dates=True, squeeze=True)  # Plot ACF. from statsmodels.graphics.tsaplots import plot\_acf  plot\_acf(series, lags=20) plt.show()  # Plot PACF. from statsmodels.graphics.tsaplots import plot\_pacf  plot\_pacf(series, lags=20) plt.show()  NUM\_TEST\_DAYS = 7  # Split dataset into test and train. X = series.values lenData = len(X) train = X[0:lenData - NUM\_TEST\_DAYS] test = X[lenData - NUM\_TEST\_DAYS:]  # Train. model = AutoReg(train, lags=7) model\_fit = model.fit() print('Coefficients: %s' % model\_fit.params)  print(model\_fit.summary())  # Make predictions. predictions = model\_fit.predict(start=len(train),  end=len(train) + len(test) - 1,  dynamic=False)  for i in range(len(predictions)):  print('predicted=%f, expected=%f' % (predictions[i], test[i])) rmse = sqrt(mean\_squared\_error(test, predictions)) print('Test RMSE: %.3f' % rmse)  # Plot results. plt.plot(test, marker='o', label='actual') plt.plot(predictions, color='brown', linewidth=4,  marker='o', label='predicted')  plt.legend() plt.show()   # Use model coefficients from autoregression to make a prediction. def makePrediction(t\_1, t\_2, t\_3, t\_4, t\_5, t\_6, t\_7):  intercept = 1.1153  t1Coeff = 0.6264  t2Coeff = -0.0751  t3Coeff = 0.0739  t4Coeff = 0.0619  t5Coeff = 0.0659  t6Coeff = 0.0442  t7Coeff = 0.1027   prediction = intercept + t1Coeff \* t\_1 \  + t2Coeff \* t\_2 \  + t3Coeff \* t\_3 + \  t4Coeff \* t\_4 + \  t5Coeff \* t\_5 + \  t6Coeff \* t\_6 + \  t7Coeff \* t\_7  return prediction   testLen = len(test)  t\_1 = test[testLen - 1] t\_2 = test[testLen - 2] t\_3 = test[testLen - 3] t\_4 = test[testLen - 4] t\_5 = test[testLen - 5] t\_6 = test[testLen - 6] t\_7 = test[testLen - 7]  futurePredictions = [] for i in range(0, NUM\_TEST\_DAYS):  prediction = makePrediction(t\_1, t\_2, t\_3, t\_4, t\_5, t\_6, t\_7)  futurePredictions.append(prediction)  t\_7 = t\_6  t\_6 = t\_5  t\_5 = t\_4  t\_4 = t\_3  t\_3 = t\_2  t\_2 = t\_1  t\_1 = prediction  print("Here is a one week temperature forecast: ") print(futurePredictions) |

Using the final day in the data set as your first input, show your future prediction output here:

|  |
| --- |
| predicted=11.552024, expected=12.900000  predicted=12.497649, expected=14.600000  predicted=12.705489, expected=14.000000  predicted=12.452495, expected=13.600000  predicted=12.228582, expected=13.500000  predicted=12.183076, expected=15.700000  predicted=11.895867, expected=13.000000  Test RMSE: 1.868  Here is a one week temperature forecast:  [12.81167, 13.174570088, 13.266794286123199, 13.215742140218772, 13.101037728248386, 13.156488837436063, 12.940074567650889] |