Table of Contents

[Back-Testing 1](#_Toc124445429)

[Correlations 6](#_Toc124445430)

[Least Squares Regression 8](#_Toc124445431)

## Back-Testing

It is difficult to validate time series data so back-testing is often used with time series data. Back testing involves an automated test which measures the results of a strategy over the long term with existing data.

Effective back tests must satisfy two conditions.

1. Back tests must be fully automated.
2. Back tests must be completely objective.

There is no standard way to write a back test – it does require creativity.

Example : Back-Testing the Buying and Selling of Stock Based on the Cross Over of Moving Averages

This example tests the buy and sell strategy which involves purchasing stock with the short-term moving average crosses over the long-term moving average for AMD stock. When the starting balance is **$10,000** the test indicates that buying and holding AMD stock will become **$33,878.40** if the stock is bought at the start of the trial and sold at the end. However, if moving averages are used to trigger buy and sell stock dates the balance will only become **$26,605.14**. In other words, the moving average strategy is less effective than the buy and hold strategy.

Figure 1 shows the points where the buy and sell activity occurs for the moving average stategy.

Figure : Buy and Sell Moving Average Strategy for AMD Stock

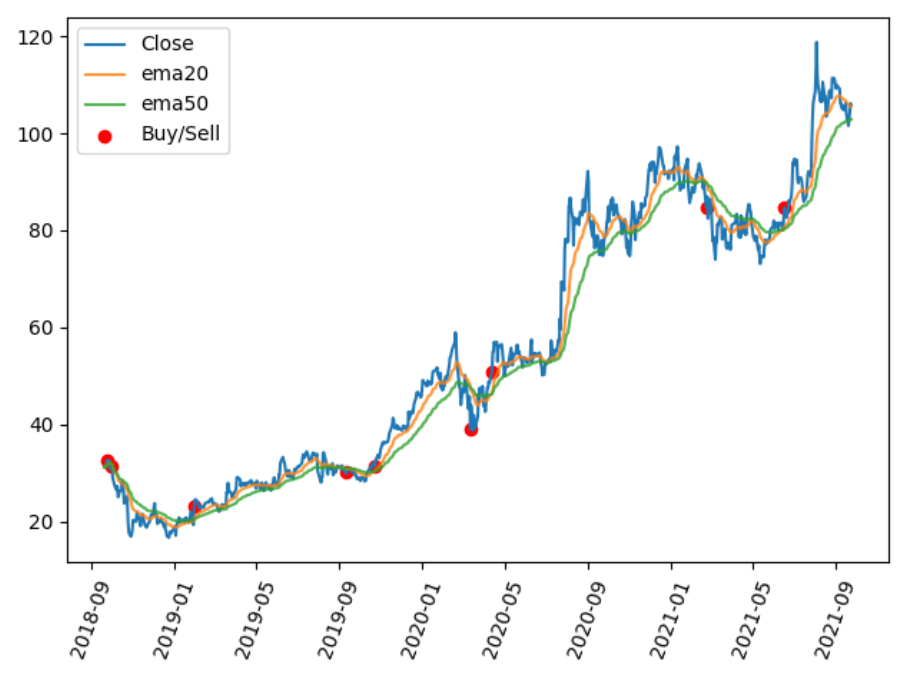


Table : Buy and Hold Earnings Versus Moving Average Triggered Buy and Sell Earnings

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Lt: 50

St: 20

-------------------------------------------------------

**Buy and hold closing balance: $33878.40**

-------------------------------------------------------

buyDt buy$ sellDt sell$ balance

0 2018-09-24 32.610001 2018-10-01 31.420000 9635.859837

1 2019-01-30 23.090000 2019-09-10 30.230000 12613.239582

2 2019-10-23 31.360001 2020-03-12 39.009998 15688.538662

3 2020-04-13 50.939999 2021-02-23 84.739998 26065.138428

**Moving average strategy closing balance: $26065.14**

|  |
| --- |
| from pandas\_datareader import data as pdr  import yfinance as yfin # Work around until  # pandas\_datareader is fixed.  import datetime  import matplotlib.pyplot as plt  import pandas as pd  import numpy as np  import warnings  warnings.filterwarnings("ignore")  # Show all columns.  pd.set\_option('display.max\_columns', None)  # Increase number of columns that display on one line.  pd.set\_option('display.width', 1000)  def getStock(stk, ttlDays):  numDays = int(ttlDays)  # Only gets up until day before during  # trading hours  dt = datetime.date.today()  # For some reason, must add 1 day to get current stock prices  # during trade hours. (Prices are about 15 min behind actual prices.)  dtNow = dt + datetime.timedelta(days=1)  dtNowStr = dtNow.strftime("%Y-%m-%d")  dtPast = dt + datetime.timedelta(days=-numDays)  dtPastStr = dtPast.strftime("%Y-%m-%d")  yfin.pdr\_override()  df = pdr.get\_data\_yahoo(stk, start=dtPastStr, end=dtNowStr)  return df  def getNewBalance(startBalance, startPrice, endPrice):  qty = int(startBalance / startPrice)  cashLeftOver = startBalance - qty \* startPrice  endValue = qty \* endPrice  balance = cashLeftOver + endValue  return balance  def showBuyAndHoldEarnings(df, balance):  startClosePrice = df.iloc[0]['Close']  endClosePrice = df.iloc[len(df) - 1]['Close']  newBalance = getNewBalance(balance, startClosePrice, endClosePrice)  print("Buy and hold closing balance: $" + str(round(newBalance, 2)))  def showStrategyEarnings(df, balance, lt, st):  buyPrice = 0  buyDate = None  sellDate = None  bought = False  buySellDates = []  prices = []  dfStrategy = pd.DataFrame(columns=['buyDt', 'buy$', 'sellDt',  'sell$', 'balance'])  dates = list(df.index)  for i in range(0, len(df)):  if (df.iloc[i]['Buy'] and not bought):  buyPrice = df.iloc[i]['Close']  buyDate = dates[i]  bought = True  buySellDates.append(buyDate)  prices.append(buyPrice)  elif (df.iloc[i]['Sell'] and bought):  sellPrice = df.iloc[i]['Close']  balance = getNewBalance(balance, buyPrice, sellPrice)  sellDate = dates[i]  buySellInfo = {'buyDt': buyDate, 'buy$': buyPrice,  'sellDt': sellDate, 'sell$': sellPrice,  'balance': balance, }  dfStrategy = dfStrategy.append(buySellInfo, ignore\_index=True)  bought = False  buySellDates.append(sellDate)  prices.append(sellPrice)  print(dfStrategy)  print("\nMoving average strategy closing balance: $" + str(round(balance, 2)))  return buySellDates, prices  def showBuyAndSellDates(df, startBalance):  strategyDates, strategyPrices = showStrategyEarnings(df, startBalance, lt, st)  plt.plot(df.index, df['Close'], label='Close')  plt.plot(df.index, df['ema20'], label='ema20', alpha=0.4)  plt.plot(df.index, df['ema50'], label='ema50', alpha=0.4)  plt.scatter(strategyDates, strategyPrices, label='Buy/Sell', color='red')  plt.xticks(rotation=70)  plt.legend()  plt.show()  def showInvestmentDifferences(dfStock, lt, st):  df = dfStock.copy()  df['ema50'] = df['Close'].ewm(span=lt).mean()  df['ema20'] = df['Close'].ewm(span=st).mean()  # Remove nulls.  df.dropna(inplace=True)  df.round(3)  own\_positions = np.where(df['ema20'] > df['ema50'], 1, 0)  df['Position'] = own\_positions  df.round(3)  df['Buy'] = (df['Position'] == 1) & (df['Position'].shift(1) == 0)  df['Sell'] = (df['Position'] == 0) & (df['Position'].shift(1) == 1)  START\_BALANCE = 10000  print("-------------------------------------------------------")  showBuyAndHoldEarnings(df, START\_BALANCE)  print("-------------------------------------------------------")  showBuyAndSellDates(df, START\_BALANCE)  longterms = [50]  shortterms = [30]  dfStock = getStock('AMD', 1100)  for lt in longterms:  for st in shortterms:  print("\b\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  print("Lt: " + str(lt))  print("St: " + str(st))  showInvestmentDifferences(dfStock, lt, st) |

Exercise (2 marks)

Using the back test, try to find a different combination of long and short-term durations which yield a better return for AMD stock when using the moving average strategy. What combination of long and short-term periods offers better results?

|  |
| --- |
| Lt: 50  St: 35 |

Show the text output here:

|  |
| --- |
|  |

Show the graph here:

|  |
| --- |
|  |

Exercise (4 marks)

Implement the back test for XOM (Exxon).

|  |
| --- |
| from pandas\_datareader import data as pdr import yfinance as yfin # Work around until  # pandas\_datareader is fixed. import datetime import matplotlib.pyplot as plt import pandas as pd import numpy as np import warnings warnings.filterwarnings("ignore")  # Show all columns. pd.set\_option('display.max\_columns', None)  # Increase number of columns that display on one line. pd.set\_option('display.width', 1000)  def getStock(stk, ttlDays):  numDays = int(ttlDays)  # Only gets up until day before during  # trading hours  dt = datetime.date.today()  # For some reason, must add 1 day to get current stock prices  # during trade hours. (Prices are about 15 min behind actual prices.)  dtNow = dt + datetime.timedelta(days=1)  dtNowStr = dtNow.strftime("%Y-%m-%d")  dtPast = dt + datetime.timedelta(days=-numDays)  dtPastStr = dtPast.strftime("%Y-%m-%d")  yfin.pdr\_override()  df = pdr.get\_data\_yahoo(stk, start=dtPastStr, end=dtNowStr)  return df  def getNewBalance(startBalance, startPrice, endPrice):  qty = int(startBalance / startPrice)  cashLeftOver = startBalance - qty \* startPrice  endValue = qty \* endPrice  balance = cashLeftOver + endValue  return balance  def showBuyAndHoldEarnings(df, balance):  startClosePrice = df.iloc[0]['Close']  endClosePrice = df.iloc[len(df) - 1]['Close']  newBalance = getNewBalance(balance, startClosePrice, endClosePrice)  print("Buy and hold closing balance: $" + str(round(newBalance, 2)))  def showStrategyEarnings(df, balance, lt, st):  buyPrice = 0  buyDate = None  sellDate = None  bought = False   buySellDates = []  prices = []   dfStrategy = pd.DataFrame(columns=['buyDt', 'buy$', 'sellDt',  'sell$', 'balance'])  dates = list(df.index)  for i in range(0, len(df)):  if (df.iloc[i]['Buy'] and not bought):  buyPrice = df.iloc[i]['Close']  buyDate = dates[i]  bought = True  buySellDates.append(buyDate)  prices.append(buyPrice)   elif (df.iloc[i]['Sell'] and bought):  sellPrice = df.iloc[i]['Close']  balance = getNewBalance(balance, buyPrice, sellPrice)  sellDate = dates[i]  buySellInfo = {'buyDt': buyDate, 'buy$': buyPrice,  'sellDt': sellDate, 'sell$': sellPrice,  'balance': balance, }  dfStrategy = dfStrategy.append(buySellInfo, ignore\_index=True)  bought = False  buySellDates.append(sellDate)  prices.append(sellPrice)   print(dfStrategy)  print("\nMoving average strategy closing balance: $" + str(round(balance, 2)))  return buySellDates, prices  def showBuyAndSellDates(df, startBalance):  strategyDates, strategyPrices = showStrategyEarnings(df, startBalance, lt, st)  plt.plot(df.index, df['Close'], label='Close')  plt.plot(df.index, df['ema20'], label='ema20', alpha=0.4)  plt.plot(df.index, df['ema50'], label='ema50', alpha=0.4)  plt.scatter(strategyDates, strategyPrices, label='Buy/Sell', color='red')  plt.xticks(rotation=70)  plt.legend()  plt.show()  def showInvestmentDifferences(dfStock, lt, st):  df = dfStock.copy()  df['ema50'] = df['Close'].ewm(span=lt).mean()  df['ema20'] = df['Close'].ewm(span=st).mean()   # Remove nulls.  df.dropna(inplace=True)  df.round(3)  own\_positions = np.where(df['ema20'] > df['ema50'], 1, 0)  df['Position'] = own\_positions  df.round(3)   df['Buy'] = (df['Position'] == 1) & (df['Position'].shift(1) == 0)  df['Sell'] = (df['Position'] == 0) & (df['Position'].shift(1) == 1)   START\_BALANCE = 10000   print("-------------------------------------------------------")  showBuyAndHoldEarnings(df, START\_BALANCE)  print("-------------------------------------------------------")  showBuyAndSellDates(df, START\_BALANCE)  longterms = [50] shortterms = [35] dfStock = getStock('XOM', 1100)  for lt in longterms:  for st in shortterms:  print("\b\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  print("Lt: " + str(lt))  print("St: " + str(st))  showInvestmentDifferences(dfStock, lt, st) |

What conclusion can be made about buying and selling with the moving average strategy for Exxon? Does the moving average perform better than buy and hold for the period under examination?

|  |
| --- |
| Buy using long term 50 and short term 35, there is an increase of about $400 for performance. |

Show the text output here:

|  |
| --- |
|  |

Show the graph here:

|  |
| --- |
|  |

Exercise (2 marks)

This is a subjective answer. What kind of changes do you think could be made to improve the results of the buy and sell strategy with moving averages?

|  |
| --- |
| If we further increase the short term value, the average strategy closing balance goes up by another $500 or so. Focusing on the short term will increase gains. |

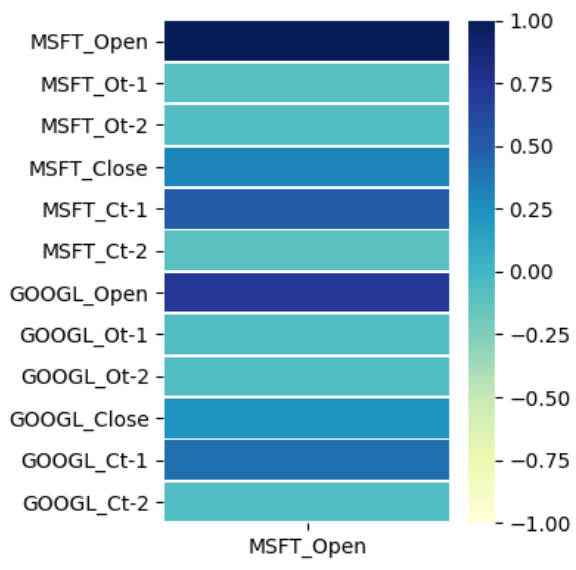
## Correlations

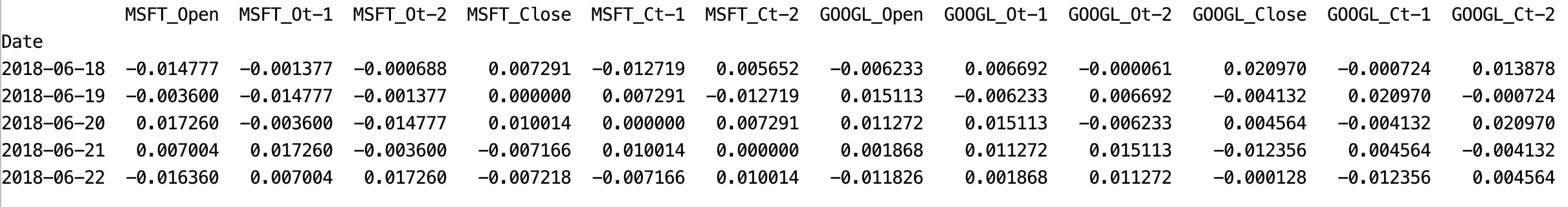
Just like with other data models, time series predictions improve when the predictor variables are highly correlated with the target variable. Figure 2 shows relative correlation between Microsoft Open price percentage changes with lagged Microsoft and Google **percent price change**s.

Example : Plotting Correlations

This example shows how to generate a correlation plot for percentage changes in Microsoft and Google stock prices. Figure 2 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 2.

|  |
| --- |
| from pandas\_datareader import data as pdr  import yfinance as yfin # Work around until  # pandas\_datareader is fixed.  import datetime  import matplotlib.pyplot as plt  import pandas as pd  def getStock(stk, ttlDays):  numDays = int(ttlDays)  # Only gets up until day before during  # trading hours  dt = datetime.date.today()  # For some reason, must add 1 day to get current stock prices  # during trade hours. (Prices are about 15 min behind actual prices.)  dtNow = dt + datetime.timedelta(days=1)  dtNowStr = dtNow.strftime("%Y-%m-%d")  dtPast = dt + datetime.timedelta(days=-numDays)  dtPastStr = dtPast.strftime("%Y-%m-%d")  yfin.pdr\_override()  df = pdr.get\_data\_yahoo(stk, start=dtPastStr, end=dtNowStr)  return df  # 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  ##################################################################  # 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)

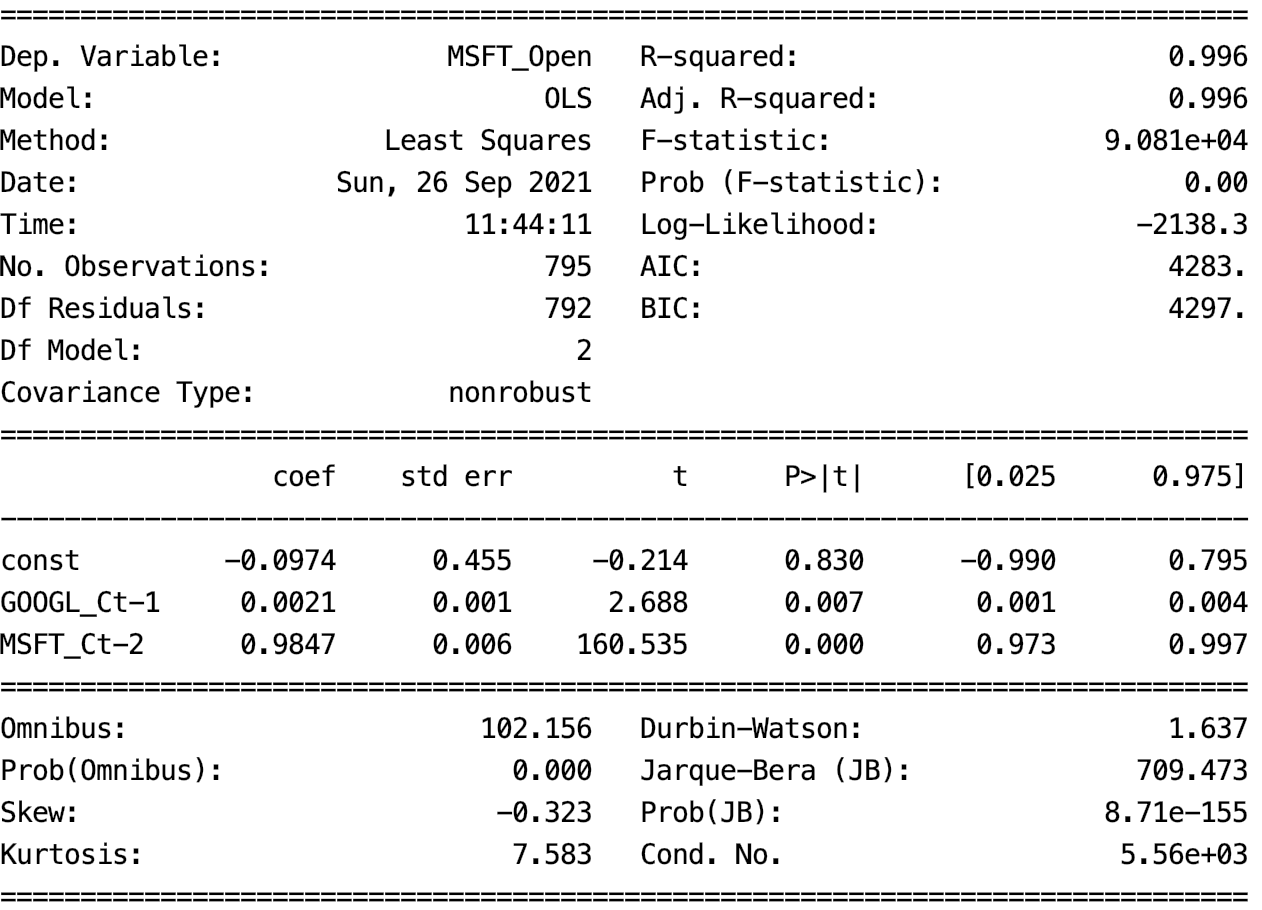
Show the correlation plot Example 2 after switching the alternate stock from Google (GOOGL) to Apple (AAPL). Show your revised plot here:

|  |
| --- |
|  |

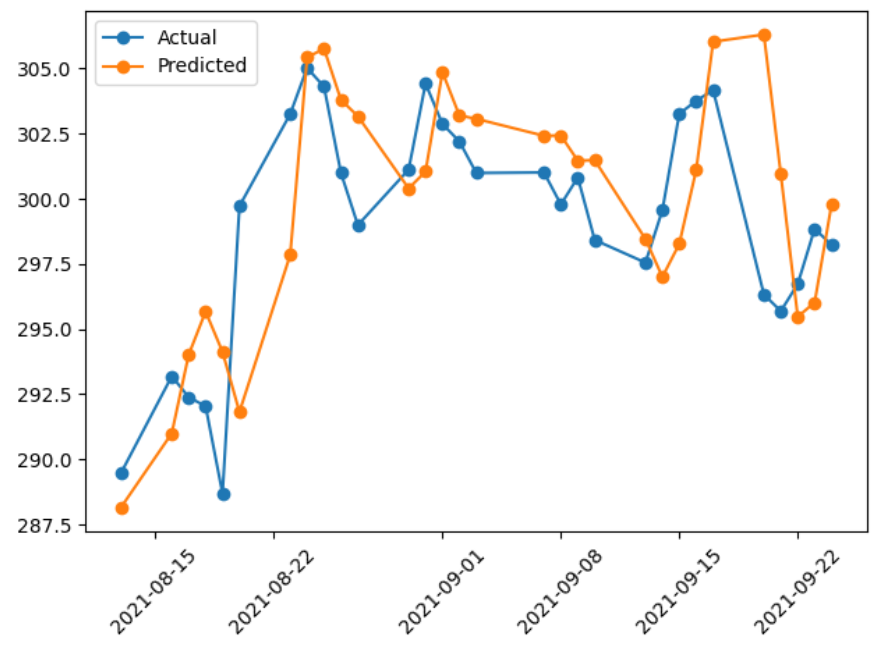
## Least Squares Regression

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 2. Then add this code to the end of Example 2.

|  |
| --- |
| 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 3 to AAPL. Do the results look more promising than with AAPL? Explain why.

|  |
| --- |
| No, RMSE and R-squared are worse now. |

Exercise (4 marks)

Change the feature set in Example 3 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 3?

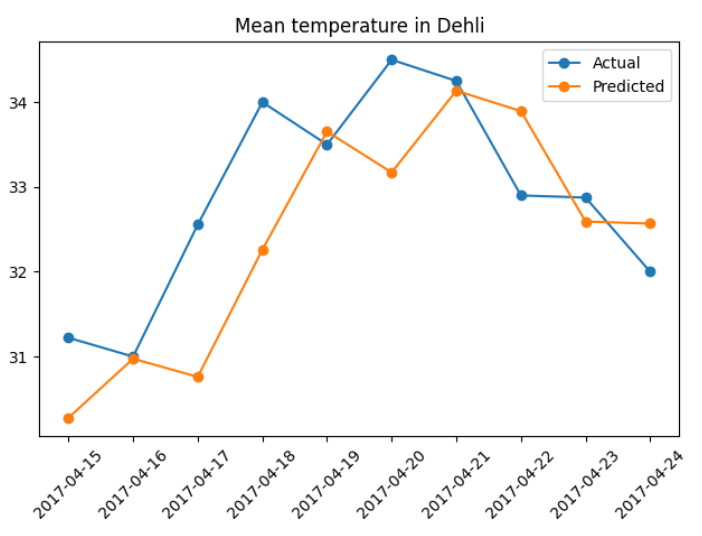
|  |
| --- |
| RMSE is lower so there is an improvement. |

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() |