Table of Contents

[Time Series Data Manipulation and Visualization Introduction 1](#_Toc123646403)

[DateTime Objects 1](#_Toc123646404)

[Converting Datetime Objects to Date Objects 2](#_Toc123646405)

[Setting DateTime Indicies for DataFrames 2](#_Toc123646406)

[Extracting Year, Month and Day from DateTime Objects 3](#_Toc123646407)

[Frequencies and Ranges 3](#_Toc123646408)

[Stock Data 5](#_Toc123646409)

[Graphing Time Series Data 7](#_Toc123646410)

[Resampling for Time Series Summaries 7](#_Toc123646411)

[Comparing Financial Data Over Time 9](#_Toc123646412)

[Time Shifting 11](#_Toc123646413)

## Time Series Data Manipulation and Visualization Introduction

When working with dates, it is easy to get distracted with seemingly endless combinations of necessary date formats. Since we are working with time series, I will mention some basic structures to raise awareness around functions that exist to help work this time data.

An excellent reference for DateTime objects and presentation is located at:

<https://jakevdp.github.io/PythonDataScienceHandbook/03.11-working-with-time-series.html>

## DateTime Objects

Example : Creating DateTime Objects

datetime objects can be created with the following initializers:

|  |
| --- |
| import pandas as pd  from datetime import datetime  dt1 = datetime(year=2015, month=12, day=4)  dt2 = pd.to\_datetime('12/8/1952')  dt3 = pd.to\_datetime('12/8/1952', dayfirst=True)  print(dt1)  print(dt2)  print(dt3) |

Here are the contents of each datetime:

|  |
| --- |
| 2015-12-04 00:00:00  1952-12-08 00:00:00  1952-08-12 00:00:00 |

### Converting Datetime Objects to Date Objects

Example : Converting DateTime Objects to Date Objects

To build this example, add this code to Example 1:

|  |
| --- |
| d1 = dt1.date()  print(d1)  d2 = dt2.date()  print(d2)  d3 = dt3.date()  print(d3) |

The output shows the dates from Example 1 without the time stamp.

|  |
| --- |
| 2015-12-04  1952-12-08  1952-08-12 |

### Setting DateTime Indicies for DataFrames

When graphing or working with time series data, it is often helpful to work with a Pandas Series or DataFrame that uses the date as an index.

Example : Initializing a DateTime Index for a DataFrame

Often when initializing the DataFrame data, the date may be given in string format. To convert this string column to a datetime format use:

df['Dates']= pd.to\_datetime(df['Dates'])

This example converts a column of dates in string format to a datetime index with the function:

df = df.set\_index('Dates')

|  |
| --- |
| import pandas as pd  df = pd.DataFrame(  data={  "Dates":['2014-07-04', '2014-08-04', '2015-07-04', '2015-08-04'],  "Temperature":[28,27,29,26]  }  )  print(df)  df['Dates'] = pd.to\_datetime(df['Dates'])  df = df.set\_index('Dates')  print(df)  print(type(df))  print("Index data type: ")  print(type(df.index)) |

The output displays the dataframe with DateTime values for the index.

|  |
| --- |
| Dates Temperature  0 2014-07-04 28  1 2014-08-04 27  2 2015-07-04 29  3 2015-08-04 26  Temperature  Dates  2014-07-04 28  2014-08-04 27  2015-07-04 29  2015-08-04 26  <class 'pandas.core.frame.DataFrame'>  Index data type:  <class 'pandas.core.indexes.datetimes.DatetimeIndex'> |

Example : Converting a String Column to a Date Column

This example shows how to read in string data from a CSV file and convert a column into a date column with the **parse\_dates** attribute. The index column is assigned the date column when the file is loaded.

|  |
| --- |
| import pandas as pd  PATH = "C:\\datasets\\"  FILE = "aritzia.csv"  df = pd.read\_csv(PATH + FILE, parse\_dates=['Date'], index\_col='Date')  print(type(df.index)) # Verify the data type.  print(df) |

### Extracting Year, Month and Day from DateTime Objects

Date time objects enable easy extraction of year, month and day values.

Example : Extracting Year, Month and Day

|  |
| --- |
| df['year'] = df.index.year  df['month'] = df.index.month  df['day'] = df.index.day  df['dayName'] = df.index.strftime("%A")  print(df) |

The output is:

|  |
| --- |
| 0 year month day dayName  2014-07-04 0 2014 7 4 Friday  2014-08-04 1 2014 8 4 Monday  2015-07-04 2 2015 7 4 Saturday  2015-08-04 3 2015 8 4 Tuesday |

## Frequencies and Ranges

With time series data, it is possible to generate date units while also specifying frequency and range of the dates.

co2 = pd.Series(data, index=pd.date\_range('1-1-1959', periods=len(data),

freq='M'), name = 'CO2')

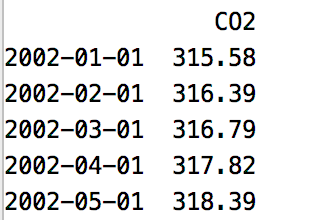
Table 1 refers to common frequency symbols.

Table : Frequency Codes

|  |  |
| --- | --- |
| **Frequency Code** | **Period** |
| Y  MS | Year  Month start |
| M | Month |
| W, W-SUN, W-MON etc. | Week |
| D  B | Day  Business Day |
| h | Hour |
| m | Minute |
| s | Second |
| ms | Millisecond |
| us | Microsecond |
| ns | Nanosecond |
| ps | Picosecond |
| fs | Femtosecond |
| as | Attosecond |

Example : Setting Frequency and Range

This example shows how to create a data frame that is indexed monthly starting from January 1, 2002. This is the data frame that is generated:



Here is the code:

|  |
| --- |
| import pandas as pd  co2 = [  342.76, 343.96, 344.82, 345.82, 347.24, 348.09, 348.66, 347.90, 346.27, 344.21,  342.88, 342.58, 343.99, 345.31, 345.98, 346.72, 347.63, 349.24, 349.83, 349.10,  347.52, 345.43, 344.48, 343.89, 345.29, 346.54, 347.66, 348.07, 349.12, 350.55,  351.34, 350.80, 349.10, 347.54, 346.20, 346.20, 347.44, 348.67]  df = pd.DataFrame({'CO2':co2}, index=pd.date\_range(  start='01-01-2002', periods=len(co2), freq='MS'))  print(df) |

Exercise (2 marks)

Generate a data frame which has a date index that increments weekly starting from the first Monday of the most recent September. Show your revised program here:

|  |
| --- |
| import pandas as pd  co2 = [ 342.76, 343.96, 344.82, 345.82, 347.24, 348.09, 348.66, 347.90, 346.27, 344.21, 342.88, 342.58, 343.99, 345.31, 345.98, 346.72, 347.63, 349.24, 349.83, 349.10, 347.52, 345.43, 344.48, 343.89, 345.29, 346.54, 347.66, 348.07, 349.12, 350.55, 351.34, 350.80, 349.10, 347.54, 346.20, 346.20, 347.44, 348.67]  df = pd.DataFrame({'CO2':co2}, index=pd.date\_range(  start='09-01-2022', periods=len(co2), freq='W-MON')) print(df) |

Show your new data frame here:

|  |
| --- |
|  |

## Stock Data

Applying the models covered in this option directly for predicting stock prices alone is not recommended due to unreliable volatility. The stock market is subjected to countless random and conflicting market actors and variables. However, techniques can still be applied for a better understanding of stock valuations and trends. Also, the abundance of stock data at regular intervals makes stock data an excellent data source for many time series analysis examples in this section.

Example : Obtaining Stock Market Data

This example shows how to obtain stock market data.

Until recently, **pandas\_datareader** used to be the main library that I would use to obtain stock market data. pandas\_datareader broke with a recent release but it does work in several online Notebook platforms. Until a more stable work around is available, I will use **yfinance** which performs the same basic tasks as *pandas\_datareader* for collecting market data. The library retrieves stock data for stocks from Yahoo and assembles it in a DataFrame for consumption on the client.

This example shows how to obtain stock data with yfinance to obtain prices of **Microsoft** and **Air Canada** stock. Prices during the last 1000 trading days for each stock are retrieved from Yahoo. The library then stores the results in a data frame.

The output shows the results in data frame format:

|  |
| --- |
| **Microsoft stock**  Open High Low Close Adj Close Volume  Date  2020-04-08 165.669998 166.669998 163.500000 165.130005 160.974976 48318200  2020-04-09 166.360001 167.369995 163.330002 165.139999 160.984741 51385100  2020-04-13 164.350006 165.570007 162.300003 165.509995 161.345398 41905300  2020-04-14 169.000000 173.750000 168.000000 173.699997 169.329330 52874300  2020-04-15 171.199997 173.570007 169.240005 171.880005 167.555145 40940800  ... ... ... ... ... ...  2022-12-27 238.699997 238.929993 235.830002 236.960007 236.960007 16688600  2022-12-28 236.889999 239.720001 234.169998 234.529999 234.529999 17457100  2022-12-29 235.649994 241.919998 235.649994 241.009995 241.009995 19770700  2022-12-30 238.210007 239.960007 236.660004 239.820007 239.820007 21930800  2023-01-03 243.080002 245.750000 237.500000 238.309998 238.309998 12512815  [690 rows x 6 columns]  [\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*100%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*] 1 of 1 completed  **Air Canada stock**  Open High Low Close Adj Close Volume  Date  2020-04-08 17.250000 18.940001 16.850000 18.740000 18.740000 9022700  2020-04-09 20.000000 22.000000 19.459999 20.080000 20.080000 9861500  2020-04-13 20.500000 20.719999 18.790001 19.309999 19.309999 6151800  2020-04-14 19.990000 20.500000 19.280001 19.430000 19.430000 5812600  2020-04-15 19.000000 19.240000 18.299999 18.660000 18.660000 4679900  ... ... ... ... ... ...  2022-12-23 19.320000 19.320000 19.010000 19.170000 19.170000 1225400  2022-12-28 18.990000 19.170000 18.590000 18.610001 18.610001 2855000  2022-12-29 18.719999 19.610001 18.719999 19.490000 19.490000 2918100  2022-12-30 19.299999 19.719999 19.260000 19.389999 19.389999 1628900  2023-01-03 19.680000 19.730000 19.055000 19.080000 19.080000 1115832 |

Here is the code:

|  |
| --- |
| from pandas\_datareader import data as pdr  import yfinance as yfin # Work around until  # pandas\_datareader is fixed.  import pandas as pd  import datetime  # Show all columns.  pd.set\_option('display.max\_columns', None)  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  NUM\_DAYS = 1000  # Search Yahoo for the correct symbols.  df = getStock('MSFT', NUM\_DAYS)  print("Microsoft stock")  print(df)  # Canadian stocks have the suffix  # .TO, .V or .CN for Canadian markets.  NUM\_DAYS = 1000  df = getStock('AC.TO', NUM\_DAYS)  print("Air Canada stock")  print(df) |

Exercise (1 mark)

Find the Yahoo stock symbol for Toronto Dominion bank shares which are sold in Canadian markets. Then, show the data frame content which displays all trade day values during the past ten days.

|  |
| --- |
|  |

## Graphing Time Series Data

Example : Graphing Time Series Data

This example shows how graph data with a reader-friendly display of the date time markers on the X-axis.



To build this example, add this code to the end of Example 7.

|  |
| --- |
| NUM\_DAYS = 1000  df = getStock('LUV', NUM\_DAYS)  print("South West Airlines")  print(df)  import matplotlib.pyplot as plt  def showStock(df, title):  plt.plot(df.index, df['Close'])  plt.title(title)  plt.xticks(rotation=70)  plt.show()  showStock(df, "South West Airlines Close Prices") |

Exercise (3 marks)

Draw a plot of Amazon stock closing prices from January 1st 2022 to present. Adjust the title as needed. Show your code here:

|  |
| --- |
| from pandas\_datareader import data as pdr import yfinance as yfin # Work around until  # pandas\_datareader is fixed. import pandas as pd import datetime  # Show all columns. pd.set\_option('display.max\_columns', None) 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  #days since jan 2022: 365 + 11 (jan 11) NUM\_DAYS = 376 df = getStock('AMZN', NUM\_DAYS) print("Amazon") print(df)  import matplotlib.pyplot as plt def showStock(df, title):  plt.plot(df.index, df['Close'])  plt.title(title)  plt.xticks(rotation=70)  plt.show() showStock(df, "Amazon Close Prices") |

Show your plot screenshot here:

|  |
| --- |
|  |

### Resampling for Time Series Summaries

Resampling allows you to quickly perform summary views of time series data for different periods (refer to Table 1) for the appropriate frequency code.

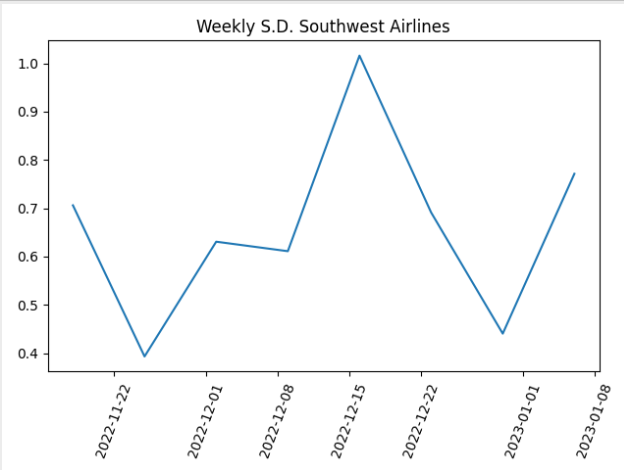
Example : Resampling

This example shows how to use resampling generate a summary of weekly standard deviations ending each Friday for the closing price of Southwestern airlines during the past 60 days.

After resampling, the data frame shows the weekly standard deviations for closing price at each Friday.

|  |
| --- |
| Close  2022-11-11 NaN  2022-11-18 0.706031  2022-11-25 0.393166  2022-12-02 0.630735  2022-12-09 0.611082  2022-12-16 1.015957  2022-12-23 0.691252  2022-12-30 0.440420  2023-01-06 0.771341  2023-01-13 NaN |

The plot shows a reader-friendly visual weekly summary of closing stock price standard deviation:



This code generates the weekly summary of closing price standard deviation dated each Friday.

series = df['Close'].resample('W').std()

series.index = series.index + to\_offset("5D")

|  |
| --- |
| # Get Southwestern stock for last 60 days  NUM\_DAYS = 60  df = getStock('LUV', NUM\_DAYS)  print("South West Airlines")  print(df)  # Create weekly summary of closing price standard deviations  from pandas.tseries.frequencies import to\_offset  series = df['Close'].resample('W').std()  series.index = series.index + to\_offset("5D")  summaryDf = series.to\_frame()  # Convert datetime index to date and then graph it.  summaryDf.index = summaryDf.index.date  print(summaryDf)  showStock(summaryDf, "Weekly S.D. Southwest Airlines") |

Exercise (2 marks)

Show a summary dataframe which displays the average monthly stock closing price at month end for Southwestern Airlines over the past 1200 days. Show the dataframe here:

|  |
| --- |
|  |

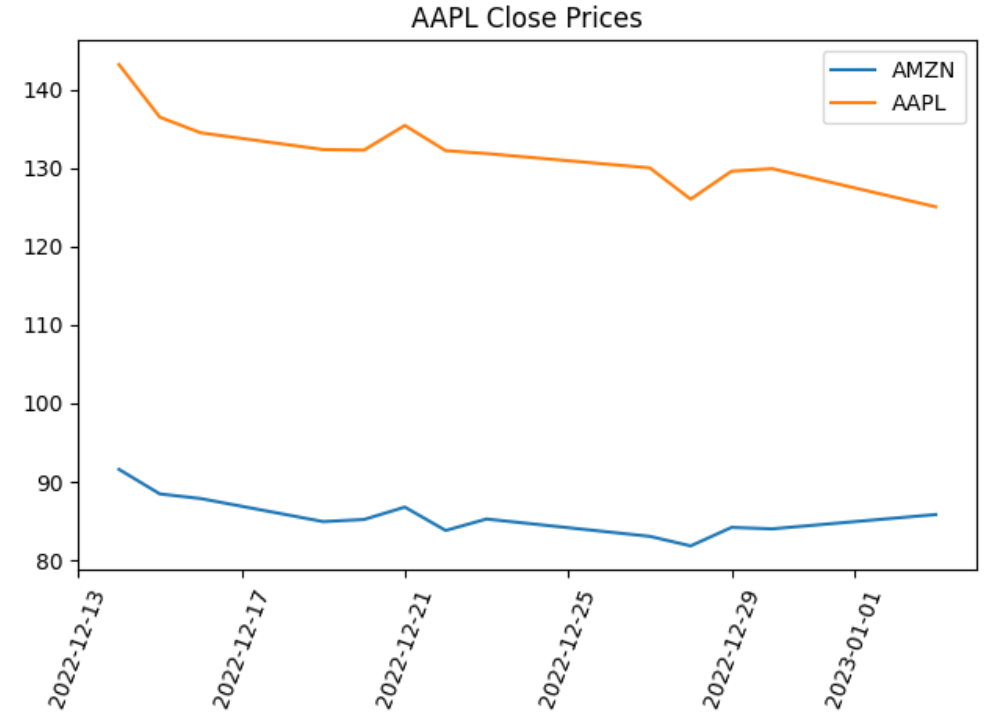
Show the plot here.

|  |
| --- |
|  |

## Comparing Financial Data Over Time

When comparing stock prices, it is sometimes difficult to compare rates of change with raw prices because prices vary so much.

Figure : Comparing Actual Closing Prices for Different Stock



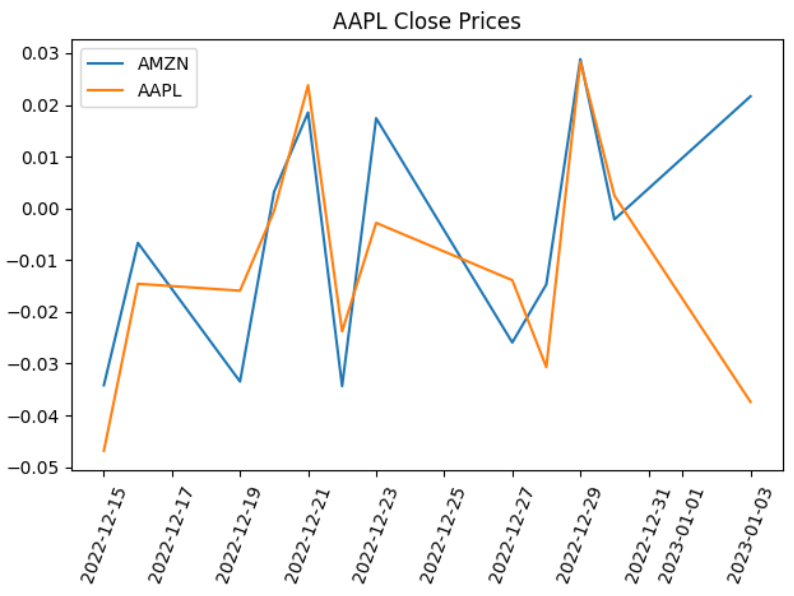
To enable better stock comparisons, financial movements over time are often rescaled using a log returns function.

Another alternative which is more intuitive involves plotting the percentage change for each item (see Example 10). The pandas data reader library includes a *pct\_change()* function to automate this calculation.

Percent change = .

Example : Comparing Percentage Change

When visualizing percentage changes of Amazon and Apple now, notice how the relative daily correlation between these two tech stocks is much more detailed.



Here is the code:

|  |
| --- |
| import matplotlib.pyplot as plt  def showStocks(df, stock, title):  plt.plot(df.index, df['Close'], label=stock)  plt.xticks(rotation=70)  NUM\_DAYS = 20  df = getStock('AMZN', NUM\_DAYS)  df['Close'] = df['Close'].pct\_change()  showStocks(df,'AMZN',"AMZN Close Prices")  df = getStock('AAPL', NUM\_DAYS)  df['Close'] = df['Close'].pct\_change()  showStocks(df, 'AAPL', "AAPL Close Prices")  # Make graphs appear.  plt.legend()  plt.show() |

Exercise (2 marks)

It is not a coincidence that Amazon and Apple are moving together like many tech stocks within the same sector. Add a third plot line to show the Microsoft (MSFT) stock percent change as well.

|  |
| --- |
| from pandas\_datareader import data as pdr import yfinance as yfin # Work around until  # pandas\_datareader is fixed. import pandas as pd import datetime  # Show all columns. pd.set\_option('display.max\_columns', None) 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   import matplotlib.pyplot as plt def showStocks(df, stock, title):  plt.plot(df.index, df['Close'], label=stock)  plt.xticks(rotation=70)  NUM\_DAYS = 20 df = getStock('AMZN', NUM\_DAYS) df['Close'] = df['Close'].pct\_change() showStocks(df,'AMZN',"AMZN Close Prices")  df = getStock('AAPL', NUM\_DAYS) df['Close'] = df['Close'].pct\_change() showStocks(df, 'AAPL', "AAPL Close Prices")  df = getStock('MSFT', NUM\_DAYS) df['Close'] = df['Close'].pct\_change() showStocks(df, 'MSFT', "MSFT Close Prices")  # Make graphs appear.  plt.legend() plt.show() |

Show your plot here:

|  |
| --- |
|  |

## Time Shifting

Once we start building data models with time series data, we are going to want to use past target values at earlier time steps. This past time step values can then be used as predictor variables for future target values. Sometimes multiple past steps can be used as predictor variables.

Example : Time Shifting

This example shows how to generate a time step column for t-1. The new data frame after running the code shows the new column:

|  |
| --- |
| CO2 CO2\_t-1  2020-09-01 342.76 NaN  2020-09-02 343.96 342.76  2020-09-03 344.82 343.96  2020-09-04 345.82 344.82  2020-09-07 347.24 345.82  2020-09-08 348.09 347.24  2020-09-09 348.66 348.09  2020-09-10 347.90 348.66  2020-09-11 346.27 347.90 |

Here is the code.

|  |
| --- |
| import pandas as pd  co2 = [342.76, 343.96, 344.82, 345.82, 347.24, 348.09, 348.66, 347.90, 346.27]  df = pd.DataFrame({'CO2':co2}, index=pd.date\_range('09-01-2022',  periods=len(co2), freq='B'))  df['CO2\_t-1'] = df['CO2'].shift(periods=1)  print(df) |

Exercise (2 marks)

Create a third column for your data frame in Example 11. This time though, shift the original data back two time steps (T-2). Also, before displaying the data frame, remove all rows with NaN values in it by calling:

df = df.dropna()

Show your revised program here:

|  |
| --- |
|  |

Show your updated data frame with the t, t-1 and t-2 columns here:

|  |
| --- |
|  |