Table of Contents

[Time Series Data Manipulation and Visualization Introduction 1](#_Toc51833944)

[DateTime objects 1](#_Toc51833945)

[DateTime Indicies 1](#_Toc51833946)

[Extracting Year, Month and Day 2](#_Toc51833947)

[Frequencies and Ranges 2](#_Toc51833948)

[Graphing with a DateTime Index 4](#_Toc51833949)

[Resampling Summaries 4](#_Toc51833950)

[Locators 5](#_Toc51833951)

[Grid Lines 6](#_Toc51833952)

[Formatters 6](#_Toc51833953)

[Axis Objects 6](#_Toc51833954)

[Comparing Financial Data Over Time 10](#_Toc51833955)

[Time Shifting 12](#_Toc51833956)

# Time Series Data Manipulation and Visualization Introduction

## DateTime objects

When working with dates, it is easy to get distracted with a seemingly endless combination of necessary date formats. Since we are working with time series, I am going to mention some basic structures to raise your awareness around functions that exist to help you 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>

You can create datetime objects with the following constructor.

|  |
| --- |
| from datetime import datetime  import pandas as pd  dt = datetime(year=2015, month=7, day=4)  pd.to\_datetime('7/8/1952')  pd.to\_datetime('7/8/1952, dayfirst=True) |

### DateTime Indicies

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 Series

For example, we can construct a Series object that has time indexed data:

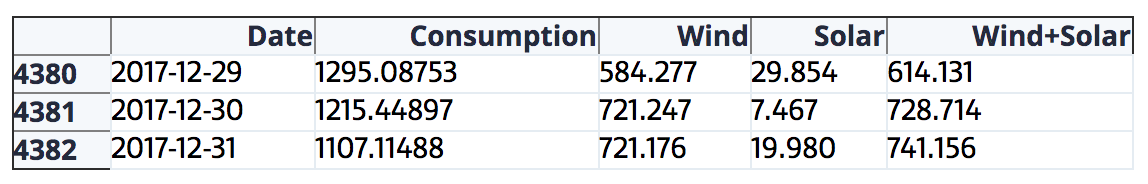
|  |
| --- |
| import pandas as pd  index = pd.DatetimeIndex(['2014-07-04', '2014-08-04', '2015-07-04', '2015-08-04'])  series = pd.Series([0, 111, 222, 333], index=index)  # If we want to add more columns or use data methods  # we need to convert to a dataframe.  df = series.to\_frame()  # Assign default name of 'ID' to default column.  df = df.rename(columns={0:"ID"})  print(df)  print(df.info()) |

This creates the structure:

|  |
| --- |
| ID  2014-07-04 0  2014-08-04 111  2015-07-04 222  2015-08-04 333  <class 'pandas.core.frame.DataFrame'>  DatetimeIndex: 4 entries, 2014-07-04 to 2015-08-04  Data columns (total 1 columns):  # Column Non-Null Count Dtype  --- ------ -------------- -----  0 ID 4 non-null int64  dtypes: int64(1)  memory usage: 64.0 bytes  None |

Example : Setting a DateTime Index

We can also convert an existing column into an index for a pandas data frame or series object. Being able to set the index as a time stamp can greatly simplify the process needed to manipulate and visualize the data in the series.



|  |
| --- |
| opsd\_daily = opsd\_daily.set\_index('Date') |

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

Date time object also make it easy to extract 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, you can specify the frequency and range of the data.

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

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

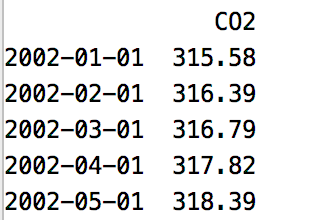
Table 1 refers to common frequency symbols.

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 |

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('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 September 21. 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('09-21-2002',  periods=len(co2), freq='W')) print(df) |

Show your new data frame here:

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

## Graphing with a DateTime Index

To graph a time series, you can just use the plot() function of the series or DataFrame object to generate a visualization quickly.

|  |
| --- |
| series['Close'].plot() |

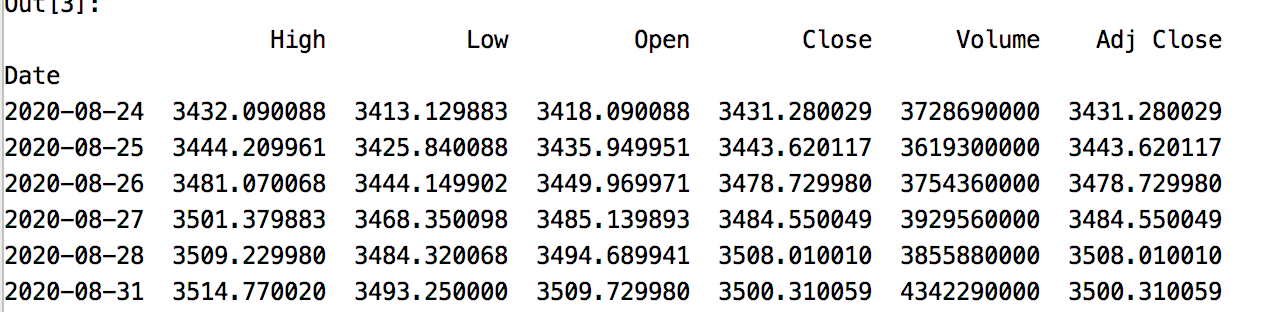
More advanced methods will be discussed below since there will be times when additional customization is needed.

### Resampling Summaries

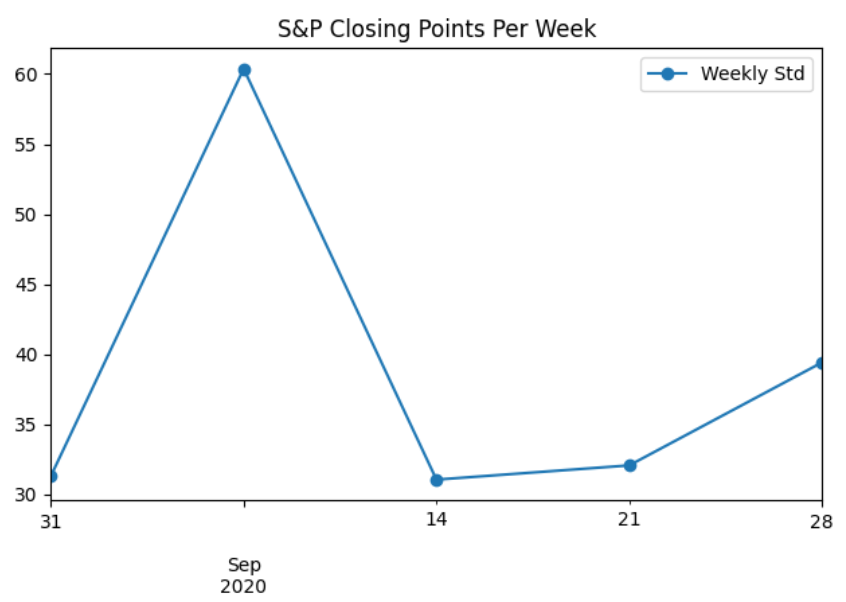
Resampling allows you to quickly perform summary views of time series data for different periods (refer to Table 1) for the appropriate frequency code.

Example : Resampling

This example shows how to quickly transform our daily S&P 500 statistics into a plot which shows the weekly standard deviation for closing points every Monday.



We can quickly generate and plot some basic summaries:



Here is the code. The line of interest that enables this plot is:

df['Close'].resample('W',loffset='1D').std().plot( marker='o', label='Weekly Std')

|  |
| --- |
| import pandas\_datareader as pdr  import datetime  import pandas as pd  import matplotlib.pyplot as plt  # Show all columns.  pd.set\_option('display.max\_columns', None)  pd.set\_option('display.width', 1000)  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 = 30  df = getStock('^GSPC', NUM\_DAYS)  # Show the S&P point standard deviation and show it on a Monday.  df['Close'].resample('W',loffset='1D').std().plot( marker='o', label='Weekly Std')  plt.title("S&P Closing Points Per Week")  plt.legend(loc='best')  plt.show() |

Exercise (2 marks)

Replace the graph that summarizes the S&P500 standard deviation per week with code that shows the min(), max(), mean() closing price each month. Retrieve values for the past 160 days before showing the graph. Show your plot here:

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

Show your code here:

|  |
| --- |
| import pandas\_datareader as pdr import datetime import pandas as pd import matplotlib.pyplot as plt  # Show all columns. pd.set\_option('display.max\_columns', None) pd.set\_option('display.width', 1000)  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 = 160 df = getStock('^GSPC', NUM\_DAYS)  # Show the S&P point standard deviation and show it on a Monday.  df['Close'].resample('W').std().plot( marker='o', label='Weekly Std') plt.title("S&P Closing Points Per Week") plt.legend(loc='best') plt.show()  df['Close'].resample('W').max().plot( marker='o', label='Weekly Max') plt.title("S&P Closing Points Per Week") plt.legend(loc='best') plt.show()  df['Close'].resample('W').min().plot( marker='o', label='Weekly Min') plt.title("S&P Closing Points Per Week") plt.legend(loc='best') plt.show()  df['Close'].resample('W').mean().plot( marker='o', label='Weekly Mean') plt.title("S&P Closing Points Per Week") plt.legend(loc='best') plt.show() |

### Locators

Locators are used in graphing to place gridlines and labels to identify the start of a range for a time series. For example, they can help to identity the beginning of weekly, monthly, quarterly and yearly cycles. For more information see:

<https://www.dataquest.io/blog/tutorial-time-series-analysis-with-pandas/>

Here is a sample which shows how to display a major gridline at the start of each month. A minor gridline is set every Monday.

|  |
| --- |
| # Set vertical major grid.  ax.xaxis.set\_major\_locator(mdates.MonthLocator(bymonthday=1))  ax.xaxis.grid(True, which = 'major', linewidth = 1, color = '#ff0000')  # Set vertical minor grid.  ax.xaxis.set\_minor\_locator(mdates.WeekdayLocator(byweekday=mdates.MONDAY))  ax.xaxis.grid(True, which = 'minor', linewidth = 1, color = '#999999')  # Sets major grid at each quarter.  ax.xaxis.set\_major\_locator(mdates.MonthLocator(bymonth=(1,4,7,10),bymonthday=1))  ax.xaxis.grid(True, which = 'major', linewidth = 1, color = '#ff0000') |

### Grid Lines

Instead of using locators to identify grid lines we can also use the grid function and set the axis to ‘x’, ‘y’ or ‘both.

# Set horizontal grid line.

ax.grid(which='major', axis='y', linestyle='--')

### Formatters

Formatters are used in time series graphing to identify major and minor axis labels in a graph.

This code sample shows how to display monthly day numbers for minor labels and the month name, day and year at the start of each month. The first of the month is assigned since the major locator has already been assigned.

|  |
| --- |
| # Set minor ticks.  ax.xaxis.set\_minor\_formatter(mdates.DateFormatter('%d'))  plt.setp(ax.xaxis.get\_minorticklabels(), rotation=70)  # Set major ticks.  ax.xaxis.set\_major\_formatter(mdates.DateFormatter('%b %d, %Y'))  plt.setp(ax.xaxis.get\_majorticklabels(), rotation=90, color='red') |

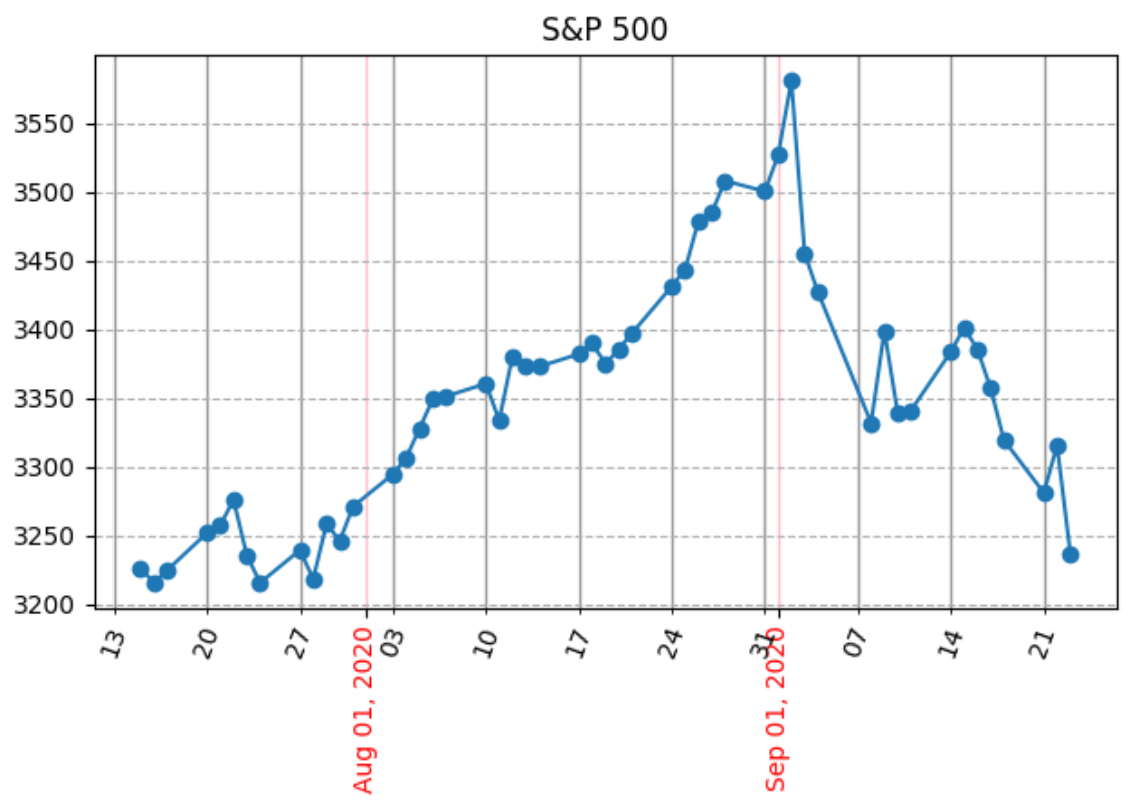
### Axis Objects

To customize the look of the x-axis and y-axis, and to plot multiple time series items together, it helps to store a reference to the axis in an axis object. The subplots() function returns the axis as the second object so two parameters must be specified when creating the reference:

fig, ax = plt.subplots()

Example : Monthly and Weekly locators

This example shows how to draw the following plot with minor gridlines at every Monday and major gridlines at the start of every month.



Here is the code:

|  |
| --- |
| import pandas\_datareader as pdr  import datetime  import pandas as pd  import matplotlib.pyplot as plt  import matplotlib.dates as mdates  # Show all columns.  pd.set\_option('display.max\_columns', None)  pd.set\_option('display.width', 1000)  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('^GSPC', NUM\_DAYS)  # Get a reference to the subplot.  # Two parameters are needed because subplots() returns  # the figure and axis object together.  # Ax lets us set axis properties.  fig, ax = plt.subplots()  # Plot data with markers at each point.  ax.plot(df.index, df['Close'], marker='o', linestyle='-')  # Set horizontal grid line automatically.  ax.grid(which='major', axis='y', linestyle='--')  # Set vertical major grid.  ax.xaxis.set\_major\_locator(mdates.MonthLocator(bymonthday=1))  ax.xaxis.grid(True, which = 'major', linewidth = 1, color = 'pink')  # Set vertical minor grid.  ax.xaxis.set\_minor\_locator(mdates.WeekdayLocator(byweekday=mdates.MONDAY))  ax.xaxis.grid(True, which = 'minor', linewidth = 1, color = '#999999')  # Set minor ticks.  ax.xaxis.set\_minor\_formatter(mdates.DateFormatter('%d'))  plt.setp(ax.xaxis.get\_minorticklabels(), rotation=70)  # Set major ticks.  ax.xaxis.set\_major\_formatter(mdates.DateFormatter('%b %d, %Y'))  plt.setp(ax.xaxis.get\_majorticklabels(), rotation=90, color='red')  plt.title("S&P 500")  plt.xlabel("Month")  plt.show() |

Exercise (4 marks)

Revise Example 7. Change the date range to 600. Change the major marker and gridline to a yearly locator. Show the month and year only for each major x axis label. Use ‘by)’

Change the minor marker and gridline so it is a monthly marker. Show only the month for each minor xaxis marker.

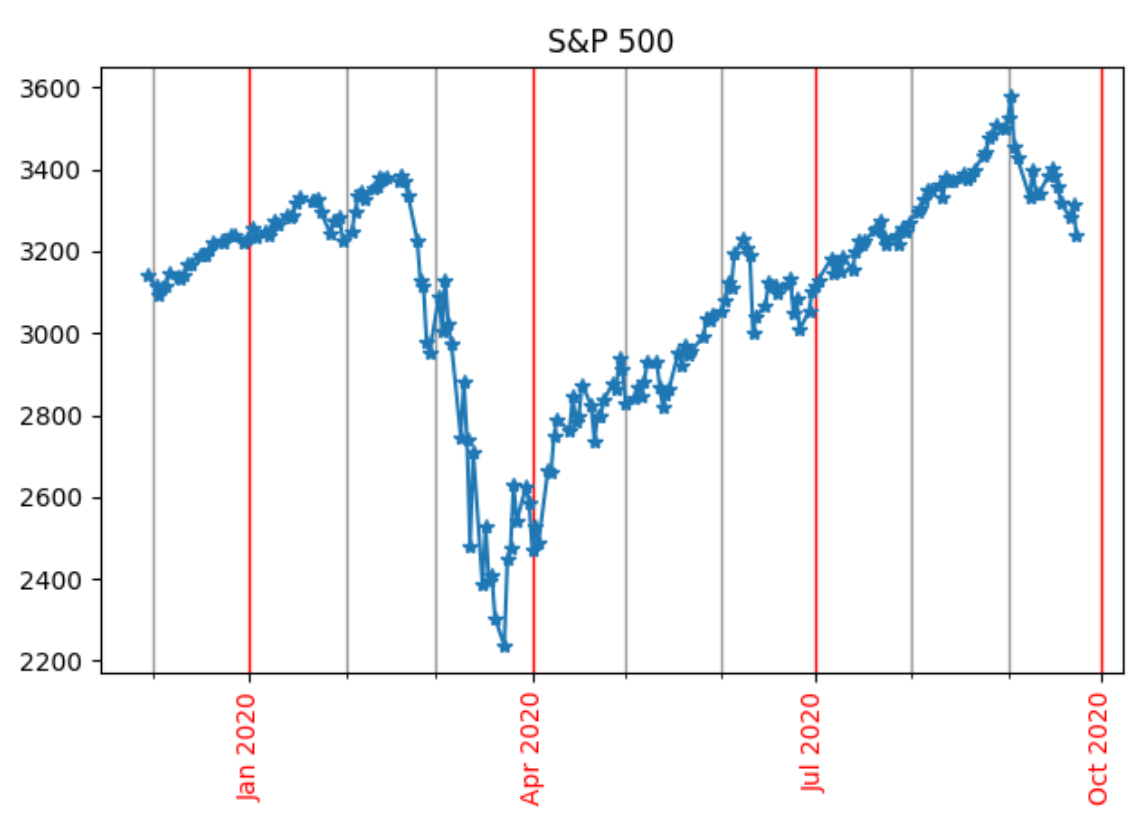
Show your graph here:

|  |
| --- |
|  |

Show your revised program here:

|  |
| --- |
| import pandas\_datareader as pdr import datetime import pandas as pd import matplotlib.pyplot as plt import matplotlib.dates as mdates  # Show all columns. pd.set\_option('display.max\_columns', None) pd.set\_option('display.width', 1000)  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 = 600 df = getStock('^GSPC', NUM\_DAYS)  # Get a reference to the subplot. # Two parameters are needed because subplots() returns # the figure and axis object together.  # Ax lets us set axis properties. fig, ax = plt.subplots()  # Plot data with markers at each point. ax.plot(df.index, df['Close'], marker='o', linestyle='-')  # Set horizontal grid line automatically. ax.grid(which='major', axis='y', linestyle='--')  # Set vertical major grid. ax.xaxis.set\_major\_locator(mdates.YearLocator(day=1)) ax.xaxis.grid(True, which = 'major', linewidth = 1, color = 'pink')  # Set vertical minor grid. ax.xaxis.set\_minor\_locator(mdates.MonthLocator()) ax.xaxis.grid(True, which = 'minor', linewidth = 1, color = '#999999')  # Set minor ticks. ax.xaxis.set\_minor\_formatter(mdates.DateFormatter('%b %d, %Y')) plt.setp(ax.xaxis.get\_minorticklabels(), rotation=70)  # Set major ticks. ax.xaxis.set\_major\_formatter(mdates.DateFormatter('%b %d, %Y')) plt.setp(ax.xaxis.get\_majorticklabels(), rotation=90, color='red')  plt.title("S&P 500") plt.xlabel("Month") plt.show() |

Exercise (3 marks)f



Starting with the following code and add monthly minor labels to indicate the month by name. Change the major interval so it every four months.

|  |
| --- |
| import pandas\_datareader as pdr  import datetime  import pandas as pd  import matplotlib.pyplot as plt  import matplotlib.dates as mdates  # Show all columns.  pd.set\_option('display.max\_columns', None)  pd.set\_option('display.width', 1000)  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 = 300  df = getStock('^GSPC', NUM\_DAYS)  #df = getStock('MSFT, NUM\_DAYS) #Microsoft stock  # Get a reference to the subplot.  # Two parameters are needed because subplots() returns  # the figure and axis object together.  # Ax lets us set axis properties.  fig, ax = plt.subplots()  # Plot data with markers at each point.  ax.plot(df.index, df['Close'], marker='\*', linestyle='-')  # Set vertical major grid.  ax.xaxis.set\_major\_locator(mdates.MonthLocator(bymonth=(1,4,7,10),bymonthday=1))  ax.xaxis.grid(True, which = 'major', linewidth = 1, color = '#ff0000')  # Set vertical minor grid.  ax.xaxis.set\_minor\_locator(mdates.MonthLocator(bymonthday=1))  ax.xaxis.grid(True, which = 'minor', linewidth = 1, color = '#999999')  # Rotate x-ticks.  ax.xaxis.set\_major\_formatter(mdates.DateFormatter('%b %Y'))  plt.setp(ax.xaxis.get\_majorticklabels(), rotation=90, color='red')  plt.title("S&P 500")  plt.xlabel("Date")  plt.show() |

Show your revised code here:

|  |
| --- |
| import pandas\_datareader as pdr import datetime import pandas as pd import matplotlib.pyplot as plt import matplotlib.dates as mdates  # Show all columns. pd.set\_option('display.max\_columns', None) pd.set\_option('display.width', 1000)  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 = 300 df = getStock('^GSPC', NUM\_DAYS) #df = getStock('MSFT, NUM\_DAYS) #Microsoft stock   # Get a reference to the subplot. # Two parameters are needed because subplots() returns # the figure and axis object together.  # Ax lets us set axis properties. fig, ax = plt.subplots()  # Plot data with markers at each point. ax.plot(df.index, df['Close'], marker='\*', linestyle='-')  # Set vertical major grid. ax.xaxis.set\_major\_locator(mdates.MonthLocator(bymonth=(12, 4, 8),bymonthday=1)) ax.xaxis.grid(True, which = 'major', linewidth = 1, color = '#ff0000')  # Set vertical minor grid. ax.xaxis.set\_minor\_locator(mdates.MonthLocator(bymonthday=1)) ax.xaxis.grid(True, which = 'minor', linewidth = 1, color = '#999999')  # Rotate x-ticks. ax.xaxis.set\_major\_formatter(mdates.DateFormatter('%b %Y')) ax.xaxis.set\_minor\_formatter(mdates.DateFormatter('%b %Y')) plt.setp(ax.xaxis.get\_majorticklabels(), rotation=90, color='red') plt.title("S&P 500") plt.xlabel("Date") plt.show() |

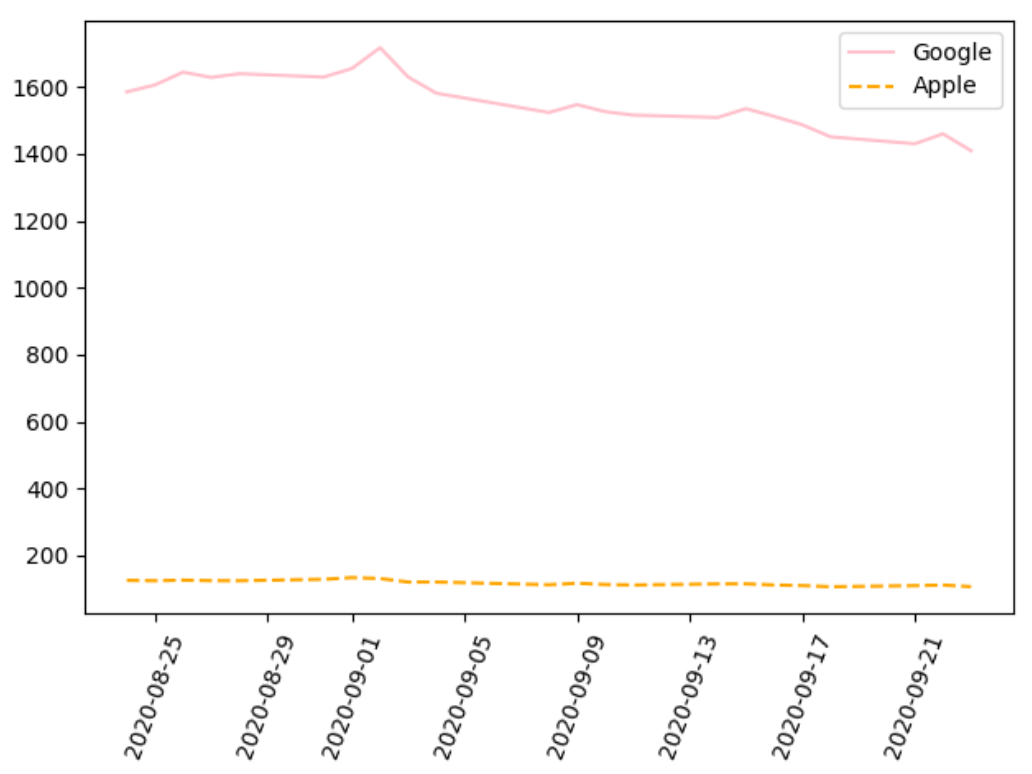
Show a drawing of your graph here:

|  |
| --- |
| Chart  Description automatically generated |

## Comparing Financial Data Over Time

When comparing stock prices, it is difficult to visualize rates of change with raw prices because prices vary so much. Google stock costs $1400. Apple stock costs approximately $111. Plotting the actual closing prices of both stocks together does not help to identify any potential relationships between the two stocks. At least we can say that the relative changes are not apparent in Figure 1.

Figure : Comparing Actual Closing Prices for Different Stock



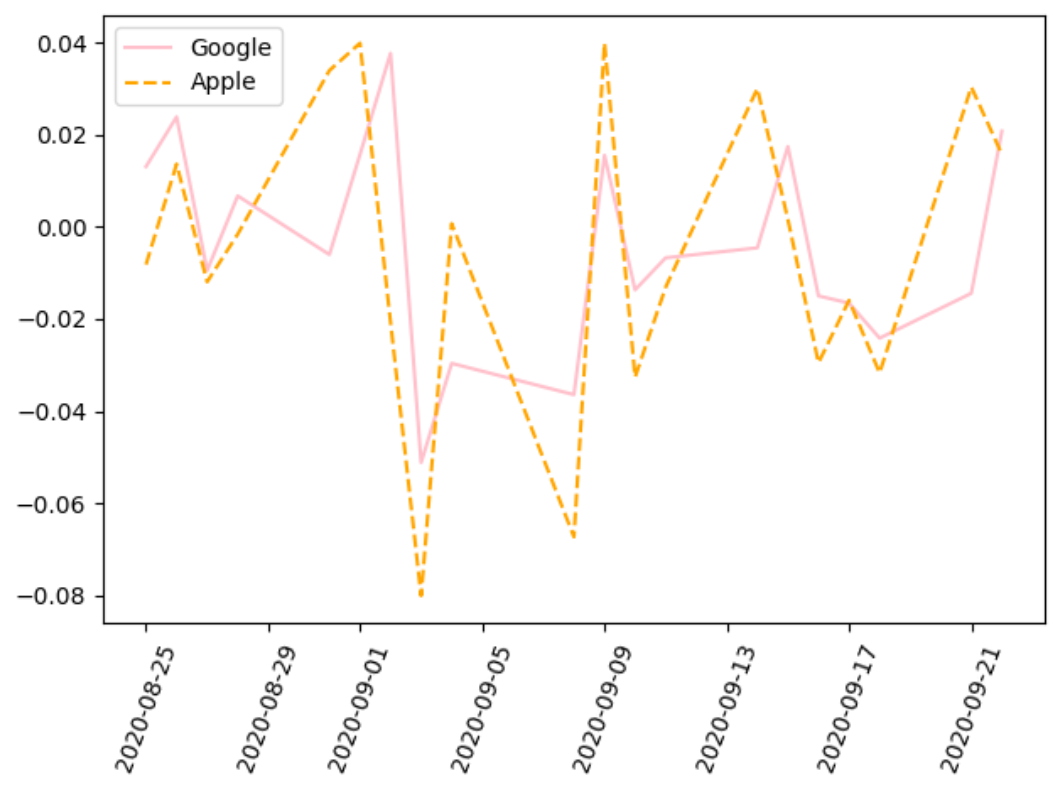
To enable better stock comparisons, financial movements over time are often rescaled using a log returns function.

Another alternative which is more intuitive involves plotting the percentage change for each item (see Example 8). The pandas data reader library includes a p*ct\_change()* function to automate this calculation.

Percent change = .

Example : Comparing Percentage Change

When visualizing percentage changes of Google and Apple now, notice how the correlation between these two tech stocks is much more obvious.



Here is the code:

|  |
| --- |
| import pandas\_datareader as pdr  import datetime  import pandas as pd  import matplotlib.pyplot as plt  import matplotlib.dates as mdates  # Show all columns.  pd.set\_option('display.max\_columns', None)  pd.set\_option('display.width', 1000)  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 = 30  dfGoogle = getStock('GOOGL', NUM\_DAYS)  dfApple = getStock('AAPL', NUM\_DAYS)  # Get a reference to the subplot.  # Two parameters are needed because subplots() returns  # the figure and axis object together.  # Ax lets us set axis properties.  fig, ax = plt.subplots()  # Plot data with markers at each point.  ax.plot(dfGoogle.index, dfGoogle['Close'].pct\_change(), color='pink', label='Google')  ax.plot(dfApple.index, dfApple['Close'].pct\_change(), '--', color='orange', label='Apple')  plt.setp( ax.xaxis.get\_majorticklabels(), rotation=70 )  plt.legend(loc="best")  plt.show() |

Exercise (2 marks)

It is not a coincidence that Google and Apple are moving together like many tech stocks. Add a third plot line to show the Microsoft (MSFT) stock percent change as well. Show your revised code here. This might offer some ideas on how you can improve your predictions for homework 1 in COMP4948.

|  |
| --- |
| import pandas\_datareader as pdr import datetime import pandas as pd import matplotlib.pyplot as plt import matplotlib.dates as mdates  # Show all columns. pd.set\_option('display.max\_columns', None) pd.set\_option('display.width', 1000)  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 #df = getStock('MSFT, NUM\_DAYS) #Microsoft stock NUM\_DAYS = 30 dfGoogle = getStock('GOOGL', NUM\_DAYS) dfApple = getStock('AAPL', NUM\_DAYS) dfMS = getStock('MSFT', NUM\_DAYS)  # Get a reference to the subplot. # Two parameters are needed because subplots() returns # the figure and axis object together.  # Ax lets us set axis properties. fig, ax = plt.subplots()  # Plot data with markers at each point. ax.plot(dfGoogle.index, dfGoogle['Close'].pct\_change(), color='pink', label='Google') ax.plot(dfApple.index, dfApple['Close'].pct\_change(), '--', color='orange', label='Apple') ax.plot(dfApple.index, dfMS['Close'].pct\_change(), '--', color='blue', label='MicroSoft') plt.setp( ax.xaxis.get\_majorticklabels(), rotation=70 ) plt.legend(loc="best") plt.show() |

Show your plot here:

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

## Time Shifting

The sample for the homework1 in COMP4948 showed a fancy and maybe confusing way to generate 60 columns which show the open price for stock at time steps in the past. There is an arguably easier way to perform this modification with the *shift()* function.

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('09-01-2020',  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 9. 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() -> dropout the Non values

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]  df = pd.DataFrame({'CO2':co2}, index=pd.date\_range('09-01-2020',  periods=len(co2), freq='B')) df['CO2\_t-1'] = df['CO2'].shift(periods=1) df['CO2\_t-2'] = df['CO2'].shift(periods=2) df = df.dropna()  print(df) |

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

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