Department of Data Science - Data and Visual Analytics Lab

Lab8. Pandas Time Series Analysis

Objectives

After completing this lab, you will be able to

```
- set index with specific column
- resample a specific column or entire dataframe
- shift data forward and backward
- shift time index with day, month, year and so forth
- compute rolling window mean
- Create time series charts
```

In [1]:
```
# Importing required modules
import pandas as pd
from dateutil.parser import parse
```

In [2]:
```
# Settings for pretty plots
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
plt.show()
```

In [3]:
```
# Reading in the data
data = pd.read_csv('amazon_stock.csv')
```

Inspect top 10 rows

In [4]:
```
data.head(10)
```

Out[4]:

	None	ticker	Date	Open	High	Low	Close	Volume	Adj_Close
0	0	AMZN	3/27/2018	1572.40	1575.96	1482.32	1497.05	6793279	1497.05
1	1	AMZN	3/26/2018	1530.00	1556.99	1499.25	1555.86	5547618	1555.86
2	2	AMZN	3/23/2018	1539.01	1549.02	1495.36	1495.56	7843966	1495.56
3	3	AMZN	3/22/2018	1565.47	1573.85	1542.40	1544.10	6177737	1544.10
4	4	AMZN	3/21/2018	1586.45	1590.00	1563.17	1581.86	4667291	1581.86

Remove unwanted columns

Remove first two columns (None and ticker) as they don't add any value to the dataset. Then, print head() to check if removed

```
In [5]: data.drop(['None', 'ticker'], axis=1, inplace=True)
Out[5]: data.head()
```

	Date	Open	High	Low	Close	Volume	Adj_Close
0	3/27/2018	1572.40	1575.96	1482.32	1497.05	6793279	1497.05
1	3/26/2018	1530.00	1556.99	1499.25	1555.86	5547618	1555.86
2	3/23/2018	1539.01	1549.02	1495.36	1495.56	7843966	1495.56
3	3/22/2018	1565.47	1573.85	1542.40	1544.10	6177737	1544.10
4	3/21/2018	1586.45	1590.00	1563.17	1581.86	4667291	1581.86

```
In [6]: #Look at the datatypes of the various columns, call info() data.info()

        <class 'pandas.core.frame.DataFrame'>
        RangeIndex: 1316 entries, 0 to 1315
        Data columns (total 7 columns):
        Date         1316 non-null object
        Open         1316 non-null float64
        High         1316 non-null float64
        Low          1316 non-null float64
        Close        1316 non-null float64
        Volume       1316 non-null int64
        Adj_Close    1316 non-null float64
        dtypes: float64(5), int64(1), object(1)
        memory usage: 72.0+ KB
```

Inspect the datatypes of columns

Looking at the information, it appears that Date column is being treated as a string rather than as dates. To fix this, we'll use the pandas to_datetime() feature which converts the arguments to dates.

Convert "Date" string column into actual Date object

In [7]: data= pd.read_csv('amazon_stock.csv', parse_dates = ['Date'])

data.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1316 entries, 0 to 1315
Data columns (total 7 columns):
Date         1316 non-null datetime64[ns]
Open         1316 non-null float64
High         1316 non-null float64
Low          1316 non-null float64
Close        1316 non-null float64
Volume       1316 non-null int64
Adj_Close    1316 non-null float64
dtypes: datetime64[ns](1), float64(5), int64(1)
memory usage: 72.0 KB
```

Let us check our data once again, with head()

In [8]: data. head()

Out[8]:

	Date	Open	High	Low	Close	Volume	Adj_Close
0	2018-03-27	1572.40	1575.96	1482.32	1497.05	6793279	1497.05
1	2018-03-26	1530.00	1556.99	1499.25	1555.86	5547618	1555.86
2	2018-03-23	1539.01	1549.02	1495.36	1495.56	7843966	1495.56
3	2018-03-22	1565.47	1573.85	1542.40	1544.10	6177737	1544.10
4	2018-03-21	1586.45	1590.00	1563.17	1581.86	4667291	1581.86

Set Date object to be index

Here Date is one of the columns. But we want date to be the index. So, set Date as index for the data frame. Make inplace=True

In [9]: data.set_index('Date', inplace=True)

In [10]: # Check with head() data.head()

Out[10]:

Date	Open	High	Low	Close	Volume	Adj_Close
2018-03-27	1572.40	1575.96	1482.32	1497.05	6793279	1497.05
2018-03-26	1530.00	1556.99	1499.25	1555.86	5547618	1555.86
2018-03-23	1539.01	1549.02	1495.36	1495.56	7843966	1495.56
2018-03-22	1565.47	1573.85	1542.40	1544.10	6177737	1544.10
2018-03-21	1586.45	1590.00	1563.17	1581.86	4667291	1581.86

Understand Stock Data

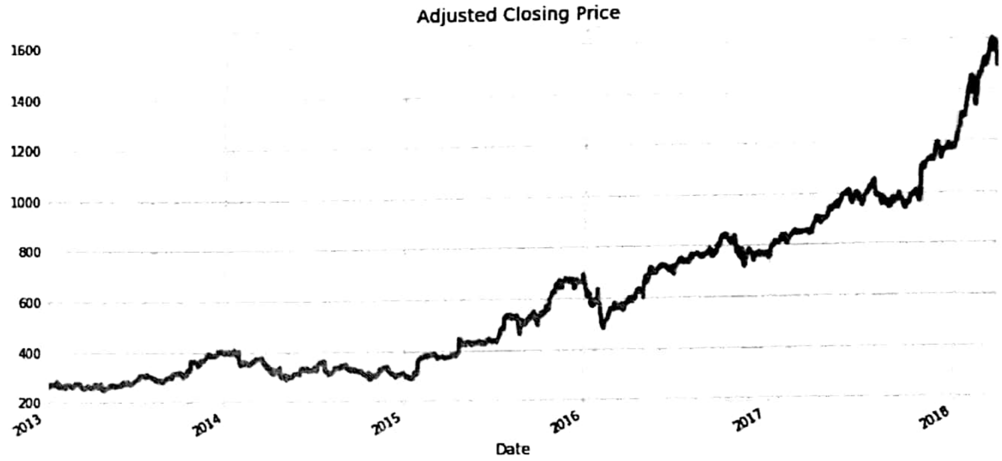
Now our data has been converted into the desired format, let's take a look at its columns for further analysis.

```
- The Open and Close columns indicate the opening and closing price of the stocks o
n a particular day.
- The High and Low columns provide the highest and the lowest price for the stock o
n a particular day, respectively.
- The Volume column tells us the total volume of stocks traded on a particular day.
```

The Adj_Close column represents the adjusted closing price, or the stock's closing price on any given day of trading, amended to include any distributions and/or corporate actions occurring any time before the next day's open. The adjusted closing price is often used when examining or performing a detailed analysis of historical returns.

In [11]:
```
data['Adj_Close'].plot(figsize=(12,6),title='Adjusted Closing Price')
```

Out[11]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x202737f5c50>
```



Interestingly, it appears that Amazon had a more or less steady increase in its stock price over the 2013-2018 window.

Understand DateTimeIndex

Introduction to datetime module

Python's basic tools for working with dates and times reside in the built-in datetime module. In pandas, a single point in time is represented as a pandas.Timestamp and we can use the datetime() function to create datetime objects from strings in a wide variety of date/time formats. datetimes are interchangeable with pandas.Timestamp

In [12]:
```
from datetime import datetime

my_year = 2020
my_month = 5
my_day = 1
my_hour = 13
my_minute = 36
my_second = 45

test_date = datetime(my_year, my_month, my_day)
test_date
```

Out[12]:
```
datetime.datetime(2020, 5, 1, 0, 0)
```

test_date = datetime(my_year, my_month, my_day, my_hour, my_minute, my_second) print("The day is : ", test_date.day) print("The hour is : ", test_date.hour) print("The month is : ", test_date.month)

Find minimum and maximum dates from data frame, call info() method

In [13]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1316 entries, 2018-03-27 to 2013-01-02
Data columns (total 6 columns):
Open         1316 non-null float64
High         1316 non-null float64
Low          1316 non-null float64
Close        1316 non-null float64
Volume       1316 non-null int64
Adj_Close    1316 non-null float64
dtypes: float64(5), int64(1)
memory usage: 72.0 KB
```

For our stock price dataset, the type of the index column is DatetimeIndex. We can use pandas to obtain the minimum and maximum dates in the data.

Print minimum and maximum index value of dataframe

In [14]:
```
print(data.index.max())
print(data.index.min())
```

```
2018-03-27 00:00:00
2013-01-02 00:00:00
```

Retrieve index of earliest and latest dates using argmin and argmax

We can also calculate the latest date location and the earliest date index location as follows

In [15]:
```
data.index.argmin()
```

Out[15]: 1315

In [16]:
```
data.index.argmax()
```

Out[16]: 0

1.Resampling Operation

Resample entire data frame

Examining stock price data for every single day isn't of much use to financial institutions, who are more interested in spotting market trends. To make it easier, we use a process called time resampling to aggregate data into a defined time period, such as by month or by quarter. Institutions can then see an overview of stock prices and make decisions according to these trends.

Resample data with year end frequency ("Y") with average stock price

In [17]: data.resample('Y').mean()

Out[17]:

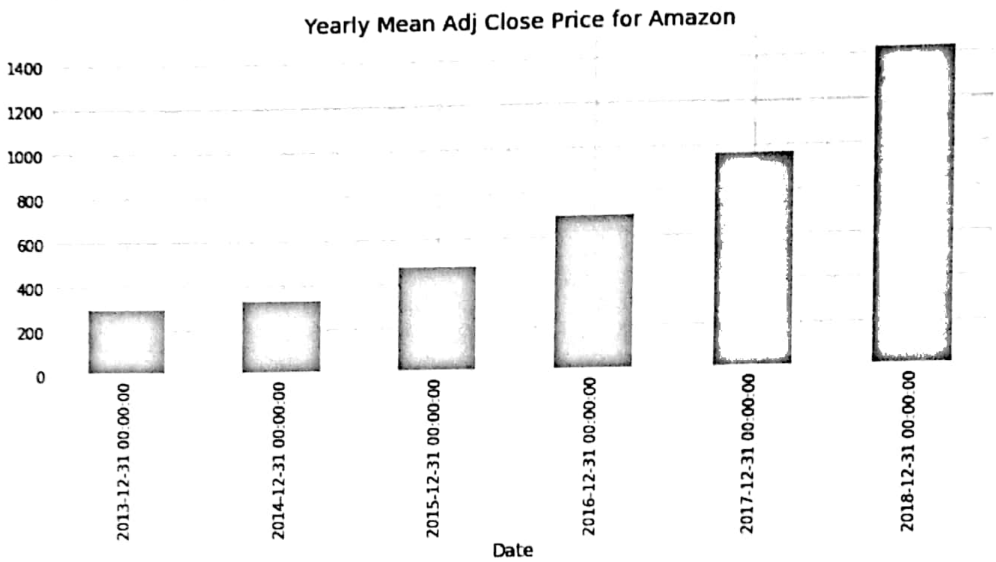
Date	Open	High	Low	Close	Volume	Adj_Close
2013-12-31	297.877223	300.925966	294.656658	298.032235	2.967880e+06	298.032235
2014-12-31	332.798433	336.317462	328.545440	332.550976	4.083223e+06	332.550976
2015-12-31	478.126230	483.248272	472.875443	478.137321	3.797801e+06	478.137321
2016-12-31	699.669762	705.799103	692.646189	699.523135	4.122043e+06	699.523135
2017-12-31	967.565060	973.789752	959.991826	967.403996	3.466207e+06	967.403996
2018-12-31	1429.770000	1446.701017	1409.469661	1429.991186	5.586829e+06	1429.991186

Here, average stock data displayed for December 31st of every year. To find other offset values refer Pandas documentation.

Resample a specific column

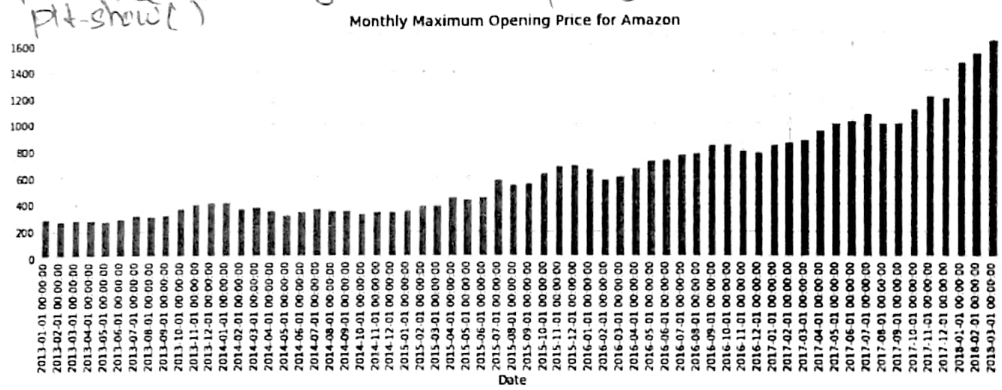
Plot a bar chart to show the yearly (Use "A") mean adjusted close price

```
In [18]:  data['Adj_Close'].resample('A').mean().plot(kind='bar', figsize=(10, 4))
          plt.title('Yearly Mean Adj Close Price for Amazon')
          plt.show()
```



Plot bar chart to show monthly maximum (Use "MS") opening price for all years

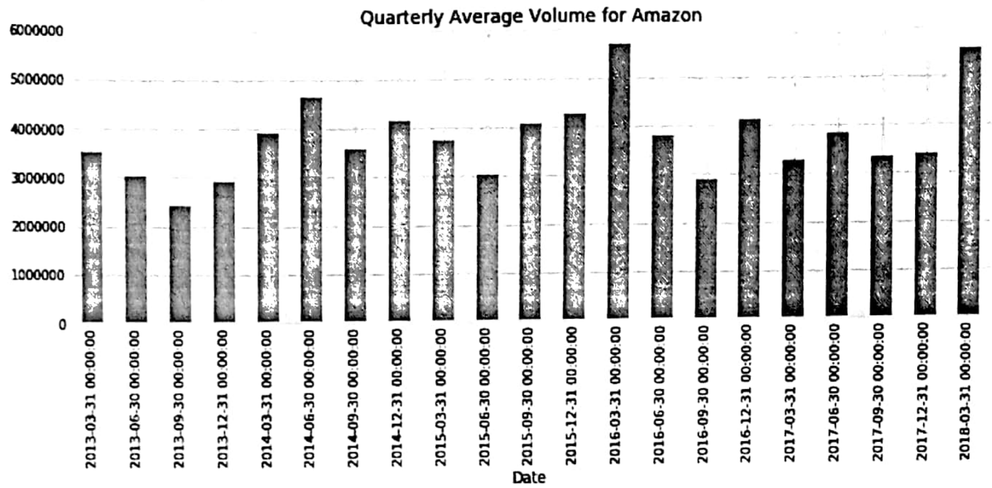
```
In [19]:  data['Adj_close'].resample('MS').mean().plot(kind='bar', figsize=(10,
          plt.title('Monthly maximum opening price for Amazon'
          plt.show()
```



Plot bar chart of Quarterly (Use "Q") Average Volume for all years

In [20]:

```
data['Adj_Close'].resample('Q').mean().plot(kind='bar' figsize=(fig
plt.title('Quaterly Average for Amazon')
plt.show()
```



2.Time Shifting Operations

Shifting data forward and backward

Show head of data

In [21]:

```
data.head()
```

Out[21]:

Date	Open	High	Low	Close	Volume	Adj_Close
2018-03-27	1572.40	1575.96	1482.32	1497.05	6793279	1497.05
2018-03-26	1530.00	1556.99	1499.25	1555.86	5547618	1555.86
2018-03-23	1539.01	1549.02	1495.36	1495.56	7843966	1495.56
2018-03-22	1565.47	1573.85	1542.40	1544.10	6177737	1544.10
2018-03-21	1586.45	1590.00	1563.17	1581.86	4667291	1581.86

Shift data by 1 Day forward

In [22]: data.shift(1, axis=0).head(5)

Out[22]:

Date	Open	High	Low	Close	Volume	Adj_Close
2018-03-27	NaN	NaN	NaN	NaN	NaN	NaN
2018-03-26	1572.40	1575.96	1482.32	1497.05	6793279.0	1497.05
2018-03-23	1530.00	1556.99	1499.25	1555.86	5547618.0	1555.86
2018-03-22	1539.01	1549.02	1495.36	1495.56	7843966.0	1495.56
2018-03-21	1565.47	1573.85	1542.40	1544.10	6177737.0	1544.10

Shift data by 1 Day Backward

In [23]: data.shift(-1, axis=0).head(5)

Out[23]:

Date	Open	High	Low	Close	Volume	Adj_Close
2018-03-27	1530.00	1556.99	1499.25	1555.86	5547618.0	1555.86
2018-03-26	1539.01	1549.02	1495.36	1495.56	7843966.0	1495.56
2018-03-23	1565.47	1573.85	1542.40	1544.10	6177737.0	1544.10
2018-03-22	1586.45	1590.00	1563.17	1581.86	4667291.0	1581.86
2018-03-21	1550.34	1587.00	1545.41	1586.51	4507049.0	1586.51

Shifting Time Index

```
In [24]: data.head(10)
```

Out[24]:

Date	Open	High	Low	Close	Volume	Adj_Close
2018-03-27	1572.40	1575.96	1482.32	1497.05	6793279	1497.05
2018-03-26	1530.00	1556.99	1499.25	1555.86	5547618	1555.86
2018-03-23	1539.01	1549.02	1495.36	1495.56	7843966	1495.56
2018-03-22	1565.47	1573.85	1542.40	1544.10	6177737	1544.10
2018-03-21	1586.45	1590.00	1563.17	1581.86	4667291	1581.86
2018-03-20	1550.34	1587.00	1545.41	1586.51	4507049	1586.51
2018-03-19	1554.53	1561.66	1525.35	1544.93	6376619	1544.93
2018-03-16	1583.45	1589.44	1567.50	1571.68	5145054	1571.68
2018-03-15	1595.00	1596.91	1578.11	1582.32	4026744	1582.32
2018-03-14	1597.00	1606.44	1590.89	1591.00	4164395	1591.00

Shift Time Index by 3 Months

```
In [25]: data.shift(periods=3, freq='M').head()
```

Out[25]:

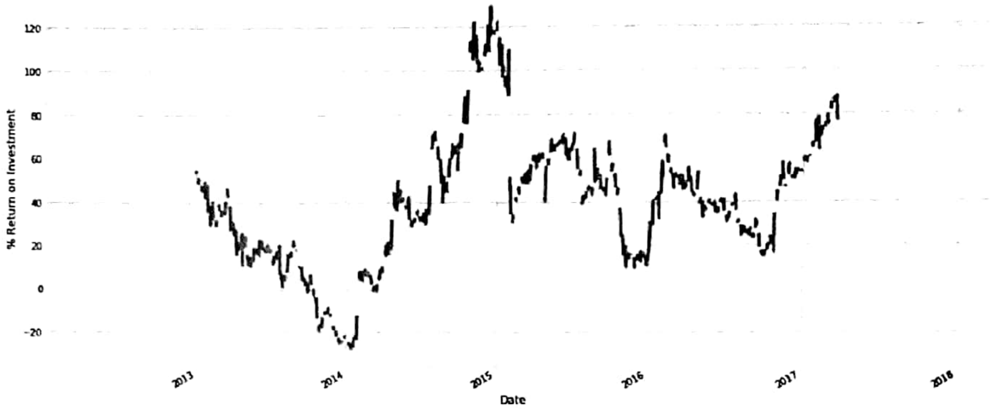
Date	Open	High	Low	Close	Volume	Adj_Close
2018-03-28	1572.40	1575.96	1482.32	1497.05	6793279	1497.05
2018-03-27	1530.00	1556.99	1499.25	1555.86	5547618	1555.86
2018-03-24	1539.01	1549.02	1495.36	1495.56	7843966	1495.56
2018-03-23	1565.47	1573.85	1542.40	1544.10	6177737	1544.10
2018-03-22	1586.45	1590.00	1563.17	1581.86	4667291	1581.86

Application: Computing Return on investment

A common context for this type of shift is computing differences over time. For example, we use shifted values to compute the one-year return on investment for Amazon stock over the course of the dataset

```
In [26]: ROI = 100 * (data['Adj_Close'].tshift(periods=-365, freq = 'D') / data['Adj_Cl
ose'] - 1)
ROI.plot(figsize=(16,8))
plt.ylabel('% Return on Investment')
```

```
Out[26]: Text(0, 0.5, '% Return on Investment')
```

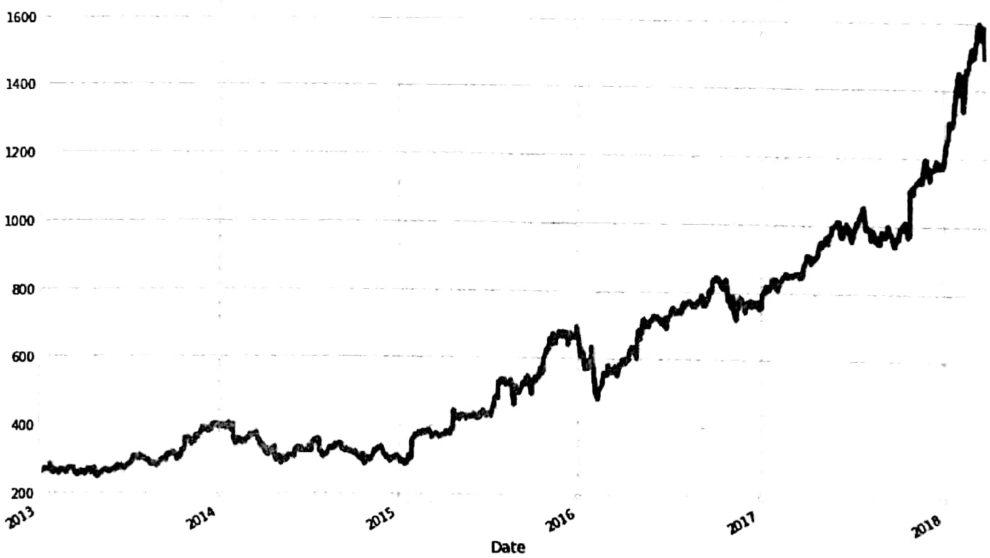


3. Rolling Window or Moving Window Operations

Time series data can be noisy due to high fluctuations in the market. As a result, it becomes difficult to gauge a trend or pattern in the data. Here is a visualization of the Amazon's adjusted close price over the years where we can see such noise (ie, line is not smooth).

```
In [27]: data['Adj_Close'].plot(figsize = (12,8), color='red')
```

```
Out[27]: <matplotlib.axes._subplots.AxesSubplot at 0x202759270b8>
```



It would be nice if we could average this out by a week, which is where a rolling mean comes in. A rolling mean, or moving average, is a transformation method which helps average out noise from data. It works by simply splitting and aggregating the data into windows according to function, such as mean(), median(), count(), etc.

Find rolling mean for 7 days and show top-10 rows

In [28]:
```
data.rolling(7).mean().head(10)
```

Out[28]:

Date	Open	High	Low	Close	Volume	Adj_Close
2018-03-27	NaN	NaN	NaN	NaN	NaN	NaN
2018-03-26	NaN	NaN	NaN	NaN	NaN	NaN
2018-03-23	NaN	NaN	NaN	NaN	NaN	NaN
2018-03-22	NaN	NaN	NaN	NaN	NaN	NaN
2018-03-21	NaN	NaN	NaN	NaN	NaN	NaN
2018-03-20	NaN	NaN	NaN	NaN	NaN	NaN
2018-03-19	1556.885714	1570.640000	1521.894286	1543.695714	5.987651e+06	1543.695714
2018-03-16	1558.464286	1572.565714	1534.062857	1554.357143	5.752191e+06	1554.357143
2018-03-15	1567.750000	1578.268571	1545.328571	1558.137143	5.534923e+06	1558.137143
2018-03-14	1576.034286	1586.471429	1558.975714	1571.771429	5.009270e+06	1571.771429

The first six values have all become blank as there wasn't enough data to actually fill them when using a window of seven days

Plot a line char for "Open" column.

Followed by, average rolling window of 30 days on the same "Open" column

```
data['Adj-close'].plot()
```

In [29]:
```
data.rolling(window=30).mean()['Adj-close'].plot(figsize=(16,6))
```

Out[29]: <matplotlib.axes._subplots.AxesSubplot at 0x20275c220b8>

Remember, first 29 days aren't going to have the blue line because there wasn't enough data to actually calculate that rolling mean.

In [ ]: