Aggregations with pandas and numpy

Name: Jann Moises Nyll B. De los Reyes

Section: CPE22S3

Date: March 27, 2024

Submitted to: Engr. Roman M. Richard

About the Data

In this notebook, we will be working with 2 data sets:

- Facebook's stock price throughout 2018 (obtained using the stock_analysis package.
- daily weather data for NYC from the <u>National Centers for Environmental Information (NCEI) API.</u>

Note: The NCEI is part of the National Oceanic and Atmospheric Administration (NOAA) and, as you can see from the URL for the API, this resource was created when the NCEI was called the NCDC. Should the URL for this resource change in the future, you can search for the NCEI weather API to find the updated one

Background on the weather data

Data meaning:

AWND: average wind speed

• PRCP: precipitation in millimeters

• SNOW: snowfall in millimeters

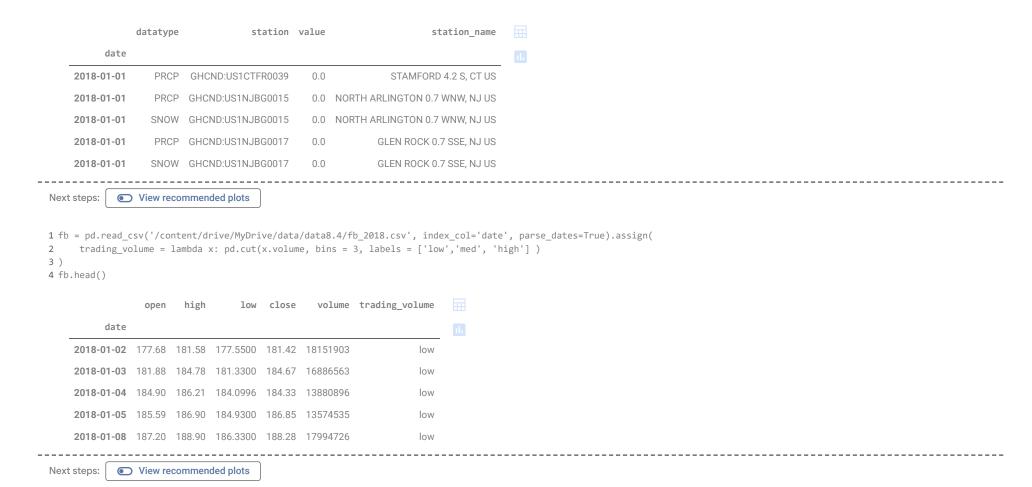
• SNWD: snow depth in millimeters

• TMAX: maximum daily temperature in Celsius

• TMIN: minimum daily temperature in Celsius

Setup

```
1 import numpy as np
2 import pandas as pd
3
4 weather = pd.read_csv('/content/drive/MyDrive/data/data8.4/weather_by_station.csv', index_col = 'date', parse_dates =True)
5 weather.head()
```



Before we dive into any calculations, let's make sure pandas won't put things in scientific notation. We will modify how floats are formatted for displaying. The format we will apply is .2f , which will provide the float with 2 digits after the decimal point:

```
1 pd.set_option('display.float_format', lambda x: '%.2f' % x)
```

Summarizing DataFrames

We learned about agg() in the dataframe operations notebook when we learned about window calculations; however, we can call this on the dataframe directly to aggregate its contents into a single series:

```
1 fb.agg({
     'open': np.mean,
   'high':np.max,
   'low': np.min,
   'close': np.mean,
     'volume': np.sum
7 })
         171.45
   open
                 218.62
   high
   low
                 123.02
   close
                 171.51
   volume 6949682394.00
   dtype: float64
```

We can use this to find the total snowfall and precipitation recorded in Central Park in 2018:

```
1 weather.query(
2     'station == "GHCND:USW00094728"'
3 ).pivot(columns='datatype', values = 'value')[['SNOW','PRCP']].sum()

    datatype
    SNOW    1007.00
    PRCP    1665.30
    dtype: float64

This is equivalent to passing 'sum' to agg():

1 weather.query(
2     'station == "GHCND:USW00094728"'
3 ).pivot(columns='datatype', values = 'value')[['SNOW','PRCP']].agg('sum')

    datatype
    SNOW    1007.00
    PRCP    1665.30
    dtype: float64
```

Note that we aren't limited to providing a single aggregation per column. We can pass a list, and we will get a dataframe back instead of a series. nan values are placed where we don't have a calculation result to display:

```
1 fb.agg({
2    'open':'mean',
3    'high':['min','max'],
4    'low':['min','max'],
5    'close': 'mean'
6 })
```



∨ Using groupby()

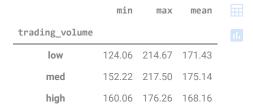
Often we won't want to aggregate on the entire dataframe, but on groups within it. For this purpose, we can run <code>groupby()</code> before the aggregation. If we group by the <code>trading_volume</code> column, we will get a row for each of the values it takes on

1 fb.groupby('trading_volume').mean()

	open	high	low	close	volume	
trading_volume						11.
low	171.36	173.46	169.31	171.43	24547207.71	
med	175.82	179.42	172.11	175.14	79072559.12	
high	167.73	170.48	161.57	168.16	141924023.33	

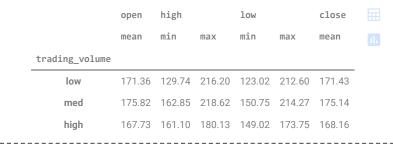
After we run the <code>groupby()</code> , we can still select columns for aggregation:

1 fb.groupby('trading_volume')['close'].agg(['min','max','mean'])



We can still provide a dictionary specifying the aggregations to perform, but passing a list for a column will result in a hierarchical index for the columns:

```
1 fb_agg = fb.groupby('trading_volume').agg({
2     'open':'mean',
3     'high':['min','max'],
4     'low':['min','max'],
5     'close': 'mean'
6 })
7
8 fb_agg
```



Next steps:

View recommended plots

The hierarchical index in the columns looks like this:

```
1 fb_agg.columns
```

Using a list comprehension, we can join the levels (in a tuple) with an _ at each iteration:

```
1 fb_agg.columns = ['_'.join(col_agg) for col_agg in fb_agg.columns]
2 fb_agg.head()
```

	open_mean	high_min	high_max	low_min	low_max	close_mean	
trading_volume							1
low	171.36	129.74	216.20	123.02	212.60	171.43	
med	175.82	162.85	218.62	150.75	214.27	175.14	
high	167.73	161.10	180.13	149.02	173.75	168.16	

Next steps:

View recommended plots

We can group on datetimes despite them being in the index if we use a Grouper:

```
1 weather['2018-10'].query('datatype== "PRCP"').groupby(
2     pd.Grouper(freq = 'D')
3 ).mean().head()
```

This Grouper can be one of many group by values. Here, we find the quarterly total precipitation per station:

1 weather.query('datatype == "PRCP"').groupby(
2 ['station_name', pd.Grouper(freq= 'Q')]
3).sum().unstack().sample(5, random_state= 1)

2018-10-04

2018-10-05 0.97

0.32

<ipython-input-18-3ccbc946d73b>:3: FutureWarning: The default value of numeric_only in DataFrameGroupBy.sum is deprecated. In a future version, numeric_only will default to F
).sum().unstack().sample(5, random_state= 1)

).sum().unstack().sample(5, random_st	ate= 1)			
	value				
date	2018-03-31	2018-06-30	2018-09-30	2018-12-31	11.
station_name					
WANTAGH 1.1 NNE, NY US	279.90	216.80	472.50	277.20	
STATEN ISLAND 1.4 SE, NY US	379.40	295.30	438.80	409.90	

379.40	295.30	438.80	409.90
323.50	263.30	355.50	459.90
338.00	272.10	424.70	390.00
246.20	295.30	620.90	422.00
	323.50 338.00	323.50 263.30 338.00 272.10	323.50 263.30 355.50 338.00 272.10 424.70

Note that we can use filter() to exclude some groups from aggregation. Here, we only keep groups with 'NY' in the group's name attribute, which is the station ID in this case:

```
AMITYVILLE 0.1 WSW, NY US
                                434.00
                                1072.00
AMITYVILLE 0.6 NNE, NY US
ARMONK 0.3 SE, NY US
                                1504.00
BROOKLYN 3.1 NW, NY US
                                305.00
CENTERPORT 0.9 SW, NY US
                                799.00
ELMSFORD 0.8 SSW, NY US
                                863.00
                                1015.00
FLORAL PARK 0.4 W, NY US
HICKSVILLE 1.3 ENE, NY US
                                716.00
JACKSON HEIGHTS 0.3 WSW, NY US 107.00
LOCUST VALLEY 0.3 E, NY US
                                 0.00
LYNBROOK 0.3 NW, NY US
                                325.00
MASSAPEQUA 0.9 SSW, NY US
                                41.00
MIDDLE VILLAGE 0.5 SW, NY US
                               1249.00
                                 0.00
NEW HYDE PARK 1.6 NE, NY US
NEW YORK 8.8 N, NY US
                                  0.00
NORTH WANTAGH 0.4 WSW, NY US
                                471.00
PLAINEDGE 0.4 WSW, NY US
                                610.00
PLAINVIEW 0.4 ENE, NY US
                               1360.00
                                707.00
SADDLE ROCK 3.4 WSW, NY US
STATEN ISLAND 1.4 SE, NY US
                                936.00
STATEN ISLAND 4.5 SSE, NY US
                               89.00
SYOSSET 2.0 SSW, NY US
                               1039.00
                                898.00
VALLEY STREAM 0.6 SE, NY US
                               1280.00
WANTAGH 0.3 ESE, NY US
WANTAGH 1.1 NNE, NY US
                                940.00
WEST NYACK 1.3 WSW, NY US
                               1371.00
Name: value, dtype: float64
```

Let's see which months have the most precipitation. First, we need to group by day and average the precipitation across the stations. Then we can group by month and sum the resulting precipitation. We use nlargest() to give the 5 months with the most precipitation:

Perhaps the previous result was surprising. The saying goes "April showers bring May flowers"; yet April wasn't in the top 5 (neither was May for that matter). Snow will count towards precipitation, but that doesn't explain why summer months are higher than April. Let's look for days that accounted for a large percentage of the precipitation in a given month.

In order to do so, we need to calculate the average daily precipitation across stations and then find the total per month. This will be the denominator. However, in order to divide the daily values by the total for their month, we will need a Series of equal dimensions. This means we will need to use transform():

```
1 weather.query('datatype == "PRCP"').rename(
     dict(value='prcp'),axis =1
3 ).groupby(pd.Grouper(freq='D')).mean().groupby(
     pd.Grouper(freq='M')
5 ).transform(np.sum)['2018-01-28':'2018-02-03']
    <ipython-input-22-6295a1c12456>:3: FutureWarning: The default value of numeric only in DataFrameGroupBy.mean is deprecated. In a future version, numeric only will default to
     ).groupby(pd.Grouper(freq='D')).mean().groupby(
                 prcp
          date
    2018-01-28
                 69.31
    2018-01-29
                69.31
    2018-01-30 69.31
    2018-01-31 69.31
    2018-02-01 158.11
    2018-02-02 158.11
```

Notice how we have the same value repeated for each day in the month it belongs to. This will allow us to calculate the percentage of the monthly precipitation that occurred each day and then pull out the largest values:

```
1 weather\
       .query('datatype == "PRCP"')\
       .rename(dict(value ='prcp'),axis= 1)\
 3
 4
       .groupby(pd.Grouper(freq='D')).mean()\
 5
       .assign(
           total_prcp_in_month=lambda x: x.groupby(
 6
 7
               pd.Grouper(freq='M')
          ).transform(np.sum),
 8
 9
           pct_monthly_prcp=lambda x: x.prcp.div(
10
              x.total_prcp_in_month
11
12
      ).nlargest(5, 'pct_monthly_prcp')
```

2018-02-03 158.11

<ipython-input-25-6749d50112e7>:4: FutureWarning: The default value of numeric_only in DataFrameGroupBy.mean is deprecated. In a future version, numeric_only will default to
 .groupby(pd.Grouper(freq='D')).mean()\

	prcp	total_prcp_in_month	pct_monthly_prcp	
date				11.
2018-10-12	34.77	105.63	0.33	
2018-01-13	21.66	69.31	0.31	
2018-03-02	38.77	137.46	0.28	
2018-04-16	39.34	140.57	0.28	
2018-04-17	37.30	140.57	0.27	

transform() can be used on dataframes as well. We can use it to easily standardize the data:

- 1 fb[['open','high','low','close']].transform(
- 2 lambda x: (x-x.mean()).div(x.std())
- 3).head()

	open	high	low	close	
date					11
2018-01-02	0.32	0.41	0.41	0.50	
2018-01-03	0.53	0.57	0.60	0.66	
2018-01-04	0.68	0.65	0.74	0.64	
2018-01-05	0.72	0.68	0.78	0.77	
2018-01-08	0.80	0.79	0.85	0.84	

Pivot tables and crosstabs

We saw pivots in before; however, we weren't able to provide any aggregations. With pivot_table() , we get the mean by default as the aggfunc . In its simplest form, we provide a column to place along the columns:

1 fb.pivot_table(columns='trading_volume')

trading_volume	low	med	high	
close	171.43	175.14	168.16	11.
high	173.46	179.42	170.48	
low	169.31	172.11	161.57	
open	171.36	175.82	167.73	
volume	24547207.71	79072559.12	141924023.33	

By placing the trading volume in the index, we get the aggregation from the first example in the group by section above:

1 fb.pivot_table(index='trading_volume')

	close	high	low	open	volume	
trading_volume						11.
low	171.43	173.46	169.31	171.36	24547207.71	
med	175.14	179.42	172.11	175.82	79072559.12	
high	168.16	170.48	161.57	167.73	141924023.33	

With pivot(), we also weren't able to handle multi-level indices or indices with repeated values. For this reason we haven't been able to put the weather data in the wide format. The pivot_table() method solves this issue:

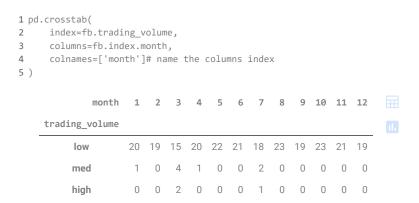
1 weather.reset_index().pivot_table(

- index=['date','station','station_name'],
- 3 columns='datatype',
- 4 values='value',
- 5 aggfunc='median'
- 6).reset_index().tail()

datatype	date	station	station_name	AWND	DAPR	MDPR	PGTM	PRCP	SNOW	SNWD	• • •	WSF5	WT01	WT02	WT03	WT04	WT05	WT06	WT08	WT09	WT11	
28740	2018- 12-31	GHCND:USW00054787	FARMINGDALE REPUBLIC AIRPORT, NY US	5.00	NaN	NaN	2052.00	28.70	NaN	NaN		15.70	NaN									
28741	2018- 12-31	GHCND:USW00094728	NY CITY CENTRAL PARK, NY US	NaN	NaN	NaN	NaN	25.90	0.00	0.00		NaN	1.00	NaN								
28742	2018- 12-31	GHCND:USW00094741	TETERBORO AIRPORT, NJ US	1.70	NaN	NaN	1954.00	29.20	NaN	NaN		8.90	NaN									
28743	2018- 12-31	GHCND:USW00094745	WESTCHESTER CO AIRPORT, NY US	2.70	NaN	NaN	2212.00	24.40	NaN	NaN		11.20	NaN									
28744	2018- 12-31	GHCND:USW00094789	JFK INTERNATIONAL AIRPORT, NY US	4.10	NaN	NaN	NaN	31.20	0.00	0.00		12.50	1.00	1.00	NaN							

5 rows × 30 columns

We can use the pd.crosstab() function to create a frequency table. For example, if we want to see how many low-, medium-, and high-volume trading days Facebook stock had each month, we can use crosstab



We can normalize with the row or column totals with the normalize parameter. This shows percentage of the total:

```
1 pd.crosstab(
      index=fb.trading_volume,
2
3
       columns=fb.index.month,
      colnames=['month'],
5
      normalize='columns'
6)
                 month
                                                                           8 9 10 11 12
     trading_volume
            low
                         0.95 1.00 0.71 0.95 1.00 1.00 0.86 1.00 1.00 1.00 1.00 1.00
            med
                         0.05 \quad 0.00 \quad 0.19 \quad 0.05 \quad 0.00 \quad 0.00 \quad 0.10 \quad 0.00 \quad 0.00 \quad 0.00 \quad 0.00
            high
                         0.00 \quad 0.00 \quad 0.10 \quad 0.00 \quad 0.00 \quad 0.05 \quad 0.00 \quad 0.00 \quad 0.00 \quad 0.00 \quad 0.00
```

If we want to perform a calculation other than counting the frequency, we can pass the column to run the calculation on to values and the function to use to aggfunc:

```
1 pd.crosstab(
2     index = fb.trading_volume,
3     columns=fb.index.month,
4     colnames=['month'],
5     values=fb.close,
6     aggfunc=np.mean
7 )
```

month 10 11 12 trading_volume low 185.24 180.27 177.07 163.29 182.93 195.27 201.92 177.49 164.38 154.19 141.64 137.16 med NaN 164 76 174 16 NaN NaN 194 28 NaN NaN NaN NaN

We can also get row and column subtotals with the margins parameter. Let's count the number of times each station recorded snow per month and include the subtotals:

1 snow_data = weather.query('datatype =="SNOW"') 2 pd.crosstab(index =snow_data.station_name, columns=snow data.index.month, colnames=['month'], values=snow_data.value, 6 aggfunc=lambda x: (x>0).sum(), 8 margins =True, #show row and columns subtotal margins_name = 'total observations of snow' # name the subtotals 10)

}	month	1	2	3	4	5	6	7	8	9	10	11	12	total observations of snow
	station_name													
	ALBERTSON 0.2 SSE, NY US	3.00	1.00	3.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	9
	AMITYVILLE 0.1 WSW, NY US	1.00	0.00	1.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	3
	AMITYVILLE 0.6 NNE, NY US	3.00	1.00	3.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	8
	ARMONK 0.3 SE, NY US	6.00	4.00	6.00	3.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	3.00	23
	BLOOMINGDALE 0.7 SSE, NJ US	2.00	1.00	3.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	8
	WESTFIELD 0.6 NE, NJ US	3.00	0.00	4.00	1.00	0.00	NaN	0.00	0.00	0.00	NaN	1.00	NaN	9
	WOODBRIDGE TWP 1.1 ESE, NJ US	4.00	1.00	3.00	2.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	11
	WOODBRIDGE TWP 1.1 NNE, NJ US	2.00	1.00	3.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	7
	WOODBRIDGE TWP 3.0 NNW, NJ US	NaN	0.00	0.00	NaN	NaN	0.00	NaN	NaN	NaN	0.00	0.00	NaN	0
	total observations of snow	190.00	97.00	237.00	81.00	0.00	0.00	0.00	0.00	0.00	0.00	49.00	13.00	667

99 rows × 13 columns

1

 \Rightarrow