Aggregations with pandas and numpy

About the Data

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

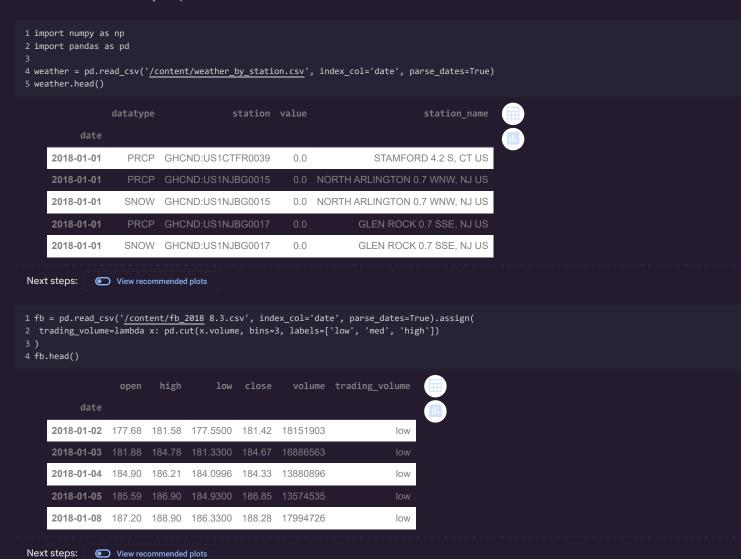
- Facebook's stock price throughout 2018 (obtained using the stock_analysis package).
- · daily weather data for NYC from the National Centers for Environmental Information (NCEI) API.

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 meanings: -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



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

dtype: float64

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({
2 'open': np.mean,
3 'high': np.max,
4 'low': np.min,
5 'close': np.mean,
6 'volume': np.sum
7 })

open 171.45
high 218.62
low 123.02
close 171.51
volume 6949682394.00
```

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() :

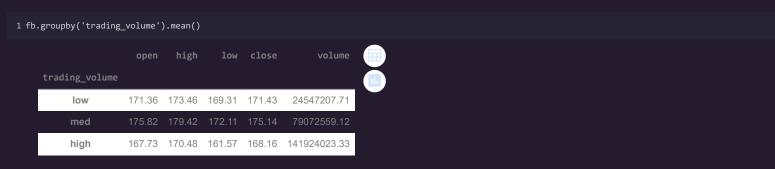
```
1 weather.query(
2 'station == "GHCND:USW00094728"'
3 ).pivot(columns='datatype', values='value')[['SNOW', 'PRCP']].agg('sum')

    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:

Using groupby()

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



After we run the groupby(), we can still select columns for aggregation:

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

| min max mean |
| trading_volume |
| low | 124.06 | 214.67 | 171.43 |
| med | 152.22 | 217.50 | 175.14 |
| high | 160.06 | 176.26 | 168.16 |
```

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:

The hierarchical index in the columns looks like this:

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

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(
  ['station_name', pd.Grouper(freq='Q')]
3 ).sum().unstack().sample(5, random_state=1)
      WANTAGH 1.1 NNE, NY US
                                        279.90
                                                     216.80
                                                                 472.50
                                                                              277.20
     STATEN ISLAND 1.4 SE. NY US
       SYOSSET 2.0 SSW. NY US
                                        323 50
                                                     263.30
                                                                 355 50
                                                                              459 90
        STAMFORD 4.2 S, CT US
      WAYNE TWP 0.8 SSW, NJ US
                                        246.20
                                                     295.30
                                                                 620.90
                                                                              422.00
```

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:

```
1 weather.groupby('station').filter( # station IDs with NY in them
                         in x.nam
   lambda x: 'NY'
3 ).query('datatype == "SNOW"').groupby('station_name').sum().squeeze() \# aggregate and make a series (squeeze() \# aggregate and make a series (squeeze()).
      <ipython-input-22-3ff96a93d3ec>:3: FutureWarning: The default value of numeric_only in DataFrameGroupBy.sum is deprecated. In a future version
         ).query('datatype == "SNOW"').groupby('station_name').sum().squeeze() # aggregate and make a series (squeeze)
      station_name
      ALBERTSON 0.2 SSE, NY US
                                                          1087.00
      AMITYVILLE 0.1 WSW, NY US
                                                           434.00
                                                          1072.00
      AMITYVILLE 0.6 NNE, NY US
      ARMONK 0.3 SE, NY US
BROOKLYN 3.1 NW, NY US
CENTERPORT 0.9 SW, NY US
                                                          1504.00
                                                            305.00
                                                            799.00
      ELMSFORD 0.8 SSW, NY US
FLORAL PARK 0.4 W, NY US
HICKSVILLE 1.3 ENE, NY US
JACKSON HEIGHTS 0.3 WSW, NY US
                                                            863.00
                                                          1015.00
                                                            716.00
                                                            107.00
     JACKSON HELGHIS 0.3 WSW, NY US
LOCUST VALLEY 0.3 E, NY US
LYNBROOK 0.3 NW, NY US
MASSAPEQUA 0.9 SSW, NY US
MIDDLE VILLAGE 0.5 SW, NY US
NEW HYDE PARK 1.6 NE, NY US
NEW YORK 8.8 N, NY US
NORTH WANTAGH 0.4 WSW, NY US
                                                              0.00
                                                           325.00
                                                             41.00
                                                          1249.00
                                                              0.00
                                                               0.00
                                                            471.00
      PLAINEDGE 0.4 WSW, NY US
PLAINVIEW 0.4 ENE, NY US
SADDLE ROCK 3.4 WSW, NY US
                                                           610.00
                                                          1360.00
                                                            707.00
      STATEN ISLAND 1.4 SE, NY US
STATEN ISLAND 4.5 SSE, NY US
SYOSSET 2.0 SSW, NY US
                                                           936.00
                                                             89.00
                                                          1039.00
      VALLEY STREAM 0.6 SE, NY US
WANTAGH 0.3 ESE, NY US
                                                            898.00
                                                          1280.00
      WANTAGH 1.1 NNE, NY US
WEST NYACK 1.3 WSW, NY US
                                                           940.00
                                                          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():

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

<ipython-input-27-e7902658fb9e>:4: FutureWarning: The default value of numeric_only in DataFrameGroupBy.mean is deprecated. In a future versio
.groupby(pd.Grouper(freq='D')).mean()\

```
        prcp
        total_prcp_in_month
        pct_monthly_prcp

        date
        2018-10-12
        34.77
        105.63
        0.33

        2018-01-13
        21.66
        69.31
        0.31

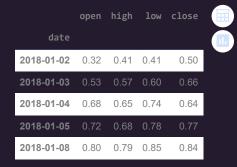
        2018-03-02
        38.77
        137.46
        0.28

        2018-04-16
        39.34
        140.57
        0.28

        2018-04-17
        37.30
        140.57
        0.27
```

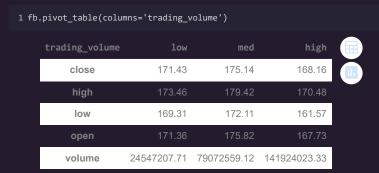
transform() can be used on dataframes as well. We can use it to easily standardize the data:

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

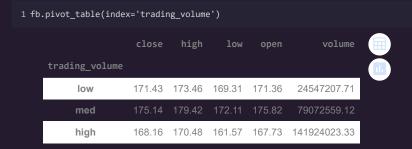


Pivot tables and crosstabs

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:



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



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:

datatype	date	station	station_name	AWND	DAPR	MDPR	PGTM	PRCP	SNOW	SNWD	WSF5	WT01	WT02	WT03	WT04	WT05	WT0
28740	2018- 12-31	GHCND:USW00054787	FARMINGDALE REPUBLIC AIRPORT, NY US	5.00	NaN	NaN	2052.00	28.70	NaN	NaN	 15.70	NaN	NaN	NaN	NaN	NaN	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	NaN	NaN	NaN	Nal
28742	2018- 12-31	GHCND:USW00094741	TETERBORO AIRPORT, NJ US	1.70	NaN	NaN	1954.00	29.20	NaN	NaN	 8.90	NaN	NaN	NaN	NaN	NaN	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	NaN	NaN	NaN	NaN	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	NaN	NaN	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:

```
1 pd.crosstab(
2 index=fb.trading_volume,
3 columns=fb.index.month,
```

colnames=['month'] # name the columns index

5)

```
        month
        1
        2
        3
        4
        5
        6
        7
        8
        9
        10
        11
        12

        trading_volume

        low
        20
        19
        15
        20
        22
        21
        18
        23
        19
        23
        21
        19

        med
        1
        0
        4
        1
        0
        0
        2
        0
        0
        0
        0
        0

        high
        0
        0
        2
        0
        0
        1
        0
        0
        0
        0
```

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

```
2 index=fb.trading_volume,
  {\tt columns=fb.index.month,}
 colnames=['month'],
normalize='columns'
                      1.00 0.71 0.95
                                    1.00
                                         1.00
                                             0.86
                                                   1 00
                                                       1 00
                                                            1 00
                                                                 1 00
                                                                      1 00
         low
                  0.95
        med
                      high
                  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:

```
index=fb.trading_volume,
columns=fb.index.month,
colnames=['month'],
values=fb.close,
aggfunc=np.mean
           month
                   185.24
                          180.27
                                  177.07 163.29 182.93 195.27 201.92 177.49
                                                                               164.38
                                                                                       154.19
                                                                                              141.64
                                                                                                      137.16
        low
        med
                                  164.11
                                                                176.26
                                                                                                        NaN
        high
                     NaN
                            NaN
                                           NaN
                                                   NaN
                                                          NaN
                                                                         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(
3 index=snow_data.station_name,
4 columns=snow_data.index.month,
5 colnames=['month'],
6 values=snow_data.value,
7 aggfunc=lambda x: (x > 0).sum(),
8 margins=True, # show row and column subtotals
9 margins_name='total observations of snow' # name the subtotals
10 )
```

month		2								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