Aggregations with pandas and numpy

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

```
import numpy as np
import pandas as pd
weather = pd.read_csv('/content/weather_by_station.csv', index_col='date', parse_dates=True)
weather.head() # Displaying the first few rows of the dataframe
```

date	datatype	station	value	station_name
2018-01-01	PRCP	GHCND:US1CTFR0039	0.0	STAMFORD 4.2 S, CT US
2018-01-01	PRCP	GHCND:US1NJBG0015	0.0	NORTH ARLINGTON 0.7 WNW, NJ US
2018-01-01	SNOW	GHCND:US1NJBG0015	0.0	NORTH ARLINGTON 0.7 WNW, NJ US
2018-01-01	PRCP	GHCND:US1NJBG0017	0.0	GLEN ROCK 0.7 SSE, NJ US
2018-01-01	SNOW	GHCND:US1NJBG0017	0.0	GLEN ROCK 0.7 SSE, NJ US

Next steps: View recommended plots

```
fb = pd.read_csv('/content/fb_2018.csv', index_col='date', parse_dates=True).assign(
    trading_volume=lambda x: pd.cut(x.volume, bins=3, labels=['low', 'med', 'high']) # Creating trading volume categor
    )
fb.head() # Displaying the first few rows of the dataframe
```

date	open	high	low	close	volume	trading_volume
2018-01-02	177.68	181.58	177.5500	181.42	18151903	low
2018-01-03	181.88	184.78	181.3300	184.67	16886563	low
2018-01-04	184.90	186.21	184.0996	184.33	13880896	low
2018-01-05	185.59	186.90	184.9300	186.85	13574535	low
2018-01-08	187.20	188.90	186.3300	188.28	17994726	low

Next steps: View recommended plots

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:

```
pd.set_option('display.float_format', lambda x: '%.2f' % x) # Set float formatting for display
```

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:

```
fb.agg({  # Aggregate function calls for different columns
    'open': np.mean,  # Calculate mean of 'open' column
    'high': np.max,   # Find maximum value of 'high' column
    'low': np.min,    # Find minimum value of 'low' column
    'close': np.mean, # Calculate mean of 'close' column
    'volume': np.sum  # Sum up values in 'volume' column
})
```

```
open            171.45
high            218.62
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:

```
weather.query(  # Querying the weather data for a specific station
    'station == "GHCND:USW00094728"'  # Filtering for the specified station
    ).pivot(columns='datatype', values='value')[['SNOW', 'PRCP']].sum()  # Pivoting data, selecting specific columns,
```

```
datatype
SNOW   1007.00
PRCP   1665.30
dtype: float64
```

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

```
weather.query(  # Querying the weather data for a specific station
    'station == "GHCND:USW00094728"'  # Filtering for the specified station
    ).pivot(columns='datatype', values='value')[['SNOW', 'PRCP']].agg('sum')  # Pivoting data, selecting specific colu
```

```
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:

```
fb.agg({  # Aggregate function calls for different columns
    'open': 'mean',     # Calculate mean of 'open' column
    'high': ['min', 'max'],  # Find minimum and maximum values of 'high' column
    'low': ['min', 'max'],   # Find minimum and maximum values of 'low' column
    'close': 'mean'     # Calculate mean of 'close' column
    })
```

	open	high	low	close
mean	171.45	NaN	NaN	171.51
min	NaN	129.74	123.02	NaN
max	NaN	218.62	214.27	NaN

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:

```
fb.groupby('trading_volume').mean() # Grouping fb data by trading volume and calculating the mean for each group
```

	open	high	low	close	volume
trading_volume					
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 groupby() , we can still select columns for aggregation:

```
fb.groupby('trading_volume')['close'].agg(['min', 'max', 'mean']) # Grouping fb data by trading volume and calculating t
```

	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:

```
fb_agg = fb.groupby('trading_volume').agg({
    'open': 'mean',     # Calculate mean of 'open' column
    'high': ['min', 'max'],  # Find minimum and maximum values of 'high' column
    'low': ['min', 'max'],   # Find minimum and maximum values of 'low' column
    'close': 'mean'     # Calculate mean of 'close' column
    })
fb_agg
```

	open	high		low		close
	mean	min	max	min	max	mean
trading_volume						
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

The hierarchical index in the columns looks like this:

```
fb_agg.columns
```

```
MultiIndex([( 'open', 'mean'),
            ( 'high',  'min'),
            ( 'high',  'max'),
            (  'low',  'min'),
            (  'low',  'max'),
            ('close', 'mean')],
           )
```

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

```
fb_agg.columns = ['_'.join(col_agg) for col_agg in fb_agg.columns]
fb_agg.head()
# Joining columns with underscore
```

trading_volume	open_mean	high_min	high_max	low_min	low_max	close_mean
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

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

```
weather['2018-10'].query('datatype == "PRCP"').groupby(
    pd.Grouper(freq='D') # Grouping by day
    ).mean().head()
```

```
<ipython-input-13-97c0ad1e4d17>:1: FutureWarning: Indexing a DataFrame with a datetimel
  weather['2018-10'].query('datatype == "PRCP"').groupby(
<ipython-input-13-97c0ad1e4d17>:3: FutureWarning: The default value of numeric_only in
  ).mean().head()
```

date	value
2018-10-01	0.01
2018-10-02	2.23
2018-10-03	19.69
2018-10-04	0.32
2018-10-05	0.97

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

```
weather.query('datatype == "PRCP"').groupby(
    ['station_name', pd.Grouper(freq='Q')] # Grouping by station_name and quarterly frequency
    ).sum().unstack().sample(5, random_state=1)
```

```
<ipython-input-14-bd52d75866ff>:3: FutureWarning: The default value of numeric_only in
  ).sum().unstack().sample(5, random_state=1)
```

	value			
date	2018-03-31	2018-06-30	2018-09-30	2018-12-31
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
SYOSSET 2.0 SSW, NY US	323.50	263.30	355.50	459.90
STAMFORD 4.2 S, CT US	338.00	272.10	424.70	390.00
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:

```
weather.groupby('station').filter( # station IDs with NY in them
                                  lambda x: 'NY' in x.name
                                   ).query('datatype == "SNOW"').groupby('station_name').sum().squeeze() # aggregate and
```

```
<ipython-input-15-fb590386f467>:3: FutureWarning: The default value of numeric_only in DataFrameGroupBy.sum is dep
  ).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
AMITYVILLE 0.6 NNE, NY US        1072.00
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
FLORAL PARK 0.4 W, NY US         1015.00
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
NEW HYDE PARK 1.6 NE, NY US         0.00
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
SADDLE ROCK 3.4 WSW, NY US        707.00
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
VALLEY STREAM 0.6 SE, NY US       898.00
WANTAGH 0.3 ESE, NY US           1280.00
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:

```
weather.query('datatype == "PRCP"').groupby(
    pd.Grouper(freq='D') # Grouping by day
    ).mean().groupby(pd.Grouper(freq='M')).sum().value.nlargest()
```

```
<ipython-input-16-23ece830d392>:3: FutureWarning: The default value of numeric_only in DataFrameGroupBy.mean is de
  ).mean().groupby(pd.Grouper(freq='M')).sum().value.nlargest()
date
2018-11-30   210.59
2018-09-30   193.09
2018-08-31   192.45
2018-07-31   160.98
2018-02-28   158.11
Name: value, dtype: float64
```

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() :

```
weather.query('datatype == "PRCP"').rename(
    dict(value='prcp'), axis=1 # Renaming 'value' column to 'prcp'
    ).groupby(pd.Grouper(freq='D')).mean().groupby(
        pd.Grouper(freq='M')
        ).transform(np.sum)['2018-01-28':'2018-02-03']
```

```
<ipython-input-17-bee011e04987>:3: FutureWarning: The default value of numeric_only in
  ).groupby(pd.Grouper(freq='D')).mean().groupby(
```

date	prcp
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
2018-02-03	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:

```
weather\
.query('datatype == "PRCP"')\
.rename(dict(value='prcp'), axis=1)\
.groupby(pd.Grouper(freq='D')).mean()\
.assign(
    total_prcp_in_month=lambda x: x.groupby(
        pd.Grouper(freq='M')
        ).transform(np.sum), # Calculating total precipitation in month
    pct_monthly_prcp=lambda x: x.prcp.div(
        x.total_prcp_in_month # Calculating percentage of monthly precipitation
        )
    ).nlargest(5, 'pct_monthly_prcp') # Selecting top 5 days with highest percentage of monthly precipitation
```

```
<ipython-input-18-e67cdadbb7f7>:4: FutureWarning: The default value of numeric_only in
  .groupby(pd.Grouper(freq='D')).mean()\
```

date	prcp	total_prcp_in_month	pct_monthly_prcp
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:

```
fb[['open', 'high', 'low', 'close']].transform(
    lambda x: (x - x.mean()).div(x.std()) # Standardizing the columns 'open', 'high', 'low', 'close'
    ).head()
```

date	open	high	low	close
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:

```
fb.pivot_table(columns='trading_volume')
```

trading_volume	low	med	high
close	171.43	175.14	168.16
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:

```
fb.pivot_table(index='trading_volume')
```

	close	high	low	open	volume
trading_volume					
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:

```
weather.reset_index().pivot_table(
    index=['date', 'station', 'station_name'],  # Setting 'date', 'station', and 'station_name' as index
    columns='datatype',  # Using 'datatype' as columns
    values='value',  # Values to fill the table
    aggfunc='median'  # Using median as aggregation function
    ).reset_index().tail()  # Resetting index and displaying the last rows
```

datatype	date	station	station_name	AWND	DAPR	MDPR	PGTM	PRCP	SNOW	SNWD	...	WSF5	WT01
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
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

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:

```
pd.crosstab(
    index=fb.trading_volume,
    columns=fb.index.month,
    colnames=['month'] # name the columns index
    )
```

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	0	1	0	0	0	0	0

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

```
pd.crosstab(
    index=fb.trading_volume,  # Rows grouped by 'trading_volume'
    columns=fb.index.month,  # Columns grouped by month of index
    colnames=['month'],  # Naming the columns as 'month'
    normalize='columns'  # Normalizing by columns
    )
```

month	1	2	3	4	5	6	7	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	0.00	0.19	0.05	0.00	0.00	0.10	0.00	0.00	0.00	0.00	0.00
high	0.00	0.00	0.10	0.00	0.00	0.00	0.05	0.00	0.00	0.00	0.00	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:

```
pd.crosstab(
    index=fb.trading_volume,  # Rows grouped by 'trading_volume'
    columns=fb.index.month,  # Columns grouped by month of index
    colnames=['month'],  # Naming the columns as 'month'
    values=fb.close,  # Values to aggregate
    aggfunc=np.mean  # Aggregating values by taking the mean
    )
```

month	1	2	3	4	5	6	7	8	9	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	179.37	NaN	164.76	174.16	NaN	NaN	194.28	NaN	NaN	NaN	NaN	NaN
high	NaN	NaN	164.11	NaN	NaN	NaN	176.26	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:

```
snow_data = weather.query('datatype == "SNOW"')  # Filtering snow data
pd.crosstab(
    index=snow_data.station_name,  # Rows grouped by 'station_name'
```

```
columns=snow_data.index.month,  # Columns grouped by month of index
colnames=['month'],  # Naming the columns as 'month'
values=snow_data.value,  # Values to aggregate
aggfunc=lambda x: (x > 0).sum(),  # Aggregating values by counting positive occurrences
margins=True,  # Showing row and column subtotals
margins_name='total observations of snow'  # Naming the subtotal
)
```

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	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