Aggregations with pandas and numpy

John Rome A. Belocora

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

Setup

```
import numpy as np
import pandas as pd
weather = pd.read_csv('weather_by_station.csv', index_col='date', parse_dates=True)
weather.head()
```

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

```
# Using the agg() method to perform aggregation operations on columns of the DataFrame `fb`
# The agg() method takes a dictionary where keys represent column names, and values represent aggregation functions

fb = pd.read_csv('fb_2018.csv', index_col='date', parse_dates=True).assign(
trading_volume=lambda x: pd.cut(x.volume, bins=3, labels=['low', 'med', 'high'])
)
fb.head()
```

date	open	high	low	close	volume	trading_volume
2018-01-02	177.68	181.58	177.55	181.42	18151903	low
2018-01-03	181.88	184.78	181.33	184.67	16886563	low
2018-01-04	184.90	186.21	184.10	184.33	13880896	low
2018-01-05	185.59	186.90	184.93	186.85	13574535	low
2018-01-08	187.20	188.90	186.33	188.28	17994726	low

Next steps: View recommended plots

```
# The display.float_format option allows custom formatting of floating-point numbers when displaying DataFrames
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:

```
# Using the agg() method to perform aggregation operations on columns of the DataFrame `fb`
# The agg() method takes a dictionary where keys represent column names, and values represent aggregation functions

fb.agg({
'open': np.mean,
'high': np.max,
'low': np.min,
'close': np.mean,
'volume': np.sum
})
```

```
open            171.45
high            218.62
low             123.02
close           171.51
volume   6949682394.00
dtype: float64
```

```
# The query() method is used with a condition to filter rows
# It returns a DataFrame containing data only for the specified weather station

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

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

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

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

```
# Some columns have multiple aggregation functions specified as a list

fb.agg({
    'open': 'mean',
    'high': ['min', 'max'],
    'low': ['min', 'max'],
    'close': 'mean'
})
```

	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()
```

trading_volume	open	high	low	close	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

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

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

```
fb_agg = fb.groupby('trading_volume').agg({
'open': 'mean',
'high': ['min', 'max'],
'low': ['min', 'max'],
'close': 'mean'
})
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

```
fb_agg.columns
```

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

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

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

Next steps: View recommended plots

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

```
<ipython-input-18-9aedd3242e78>:1: FutureWarning: Indexing a DataFrame with a datetimelike index using a single string to slice the row
  weather['2018-10'].query('datatype == "PRCP"').groupby(
<ipython-input-18-9aedd3242e78>:3: FutureWarning: The default value of numeric_only in DataFrameGroupBy.mean is deprecated. In a future
  ).mean().head()
```

	value
date	
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

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

```
<ipython-input-19-6ce2f6186f6b>:3: FutureWarning: The default value of numeric_only in DataFrameGroupBy.sum is deprecated. In a future
  ).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

```
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 make a series (squeeze)
```

```
<ipython-input-20-799de504673b>:3: FutureWarning: The default value of numeric_only in DataFrameGroupBy.sum is deprecated. In a future
  ).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
```

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

```
<ipython-input-21-610904b0030a>:3: FutureWarning: The default value of numeric_only in DataFrameGroupBy.mean is deprecated. In a future
  ).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
```

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

```
<ipython-input-22-c33a54b21001>:3: FutureWarning: The default value of numeric_only in DataFrameGroupBy.mean is deprecated. In a future
  ).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

```
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),
pct_monthly_prcp=lambda x: x.prcp.div(
x.total_prcp_in_month
)
).nlargest(5, 'pct_monthly_prcp')
```

```
<ipython-input-24-e5da73b4d9c0>:4: FutureWarning: The default value of numeric_only in DataFrameGroupBy.mean is deprecated. In a future
  .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

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

	open	high	low	close
date				
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

```
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

```
weather.reset_index().pivot_table(
    index=['date', 'station', 'station_name'],
    columns='datatype',
    values='value',
    aggfunc='median'
).reset_index().tail()
```

datatype	date	station	station_name	AWND	DAPR	MDPR	PGTM	PRCP	SNOW	SNWD	...	WSF5	WT01	WT02	WT03	WT04	WT
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	Na
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	Na
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	Na
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	Na

```
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

```
pd.crosstab(
    index=fb.trading_volume,
    columns=fb.index.month,
    colnames=['month'],
    normalize='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

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

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