Hands-on Activity 8.3, 8.4 Aggregating Pandas DataFrames

Submitted by: Dela Cruz, Eugene D.G.

Submitted to: Engr. Roman Richard

Section: CPE22S3

Submitted on: 4/1/24

8.3 Dataframe Operations

```
import numpy as np
import pandas as pd
weather = pd.read_csv('/content/nyc_weather_2018.csv', parse_dates=['date'])
weather.head()
```

	attributes	datatype	date	station	value
0	,,N,	PRCP	2018-01-01	GHCND:US1CTFR0039	0.0
1	,,N,	PRCP	2018-01-01	GHCND:US1NJBG0015	0.0
2	,,N,	SNOW	2018-01-01	GHCND:US1NJBG0015	0.0
3	,,N,	PRCP	2018-01-01	GHCND:US1NJBG0017	0.0
4	,,N,	SNOW	2018-01-01	GHCND:US1NJBG0017	0.0

Next steps: View recommended plots

```
fb = pd.read_csv('/content/fb_2018.csv', index_col='date', parse_dates=True)
fb.head()
```

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

Next steps: View recommended plots

```
fb.assign( #assign new colum for zscore of volume
 abs_z_score_volume=lambda x: x.volume.sub(x.volume.mean()).div(x.volume.std()).abs()
).query('abs_z_score_volume > 3') #filter rows where value is greater than 3
```

date	open	high	low	close	volume	abs_z_score_volume
2018-03-19	177.01	177.17	170.06	172.56	88140060	3.145078
2018-03-20	167.47	170.20	161.95	168.15	129851768	5.315169
2018-03-21	164.80	173.40	163.30	169.39	106598834	4.105413
2018-03-26	160.82	161.10	149.02	160.06	126116634	5.120845
2018-07-26	174.89	180.13	173.75	176.26	169803668	7.393705

```
fb.assign( #assign new columns to facebook stock data
 volume_pct_change=fb.volume.pct_change(),
 pct_change_rank=lambda x: x.volume_pct_change.abs().rank(
 ascending=False
 )
).nsmallest(5, 'pct_change_rank') #select the smallest 5 row
```

date	open	high	low	close	volume	volume_pct_change	pct_change_rank
2018-01-12	178.06	181.48	177.40	179.37	77551299	7.087876	1.0
2018-03-19	177.01	177.17	170.06	172.56	88140060	2.611789	2.0
2018-07-26	174.89	180.13	173.75	176.26	169803668	1.628841	3.0
2018-09-21	166.64	167.25	162.81	162.93	45994800	1.428956	4.0
2018-03-26	160.82	161.10	149.02	160.06	126116634	1.352496	5.0

```
fb['2018-01-11':'2018-01-12']
```

date	open	high	low	close	volume
2018-01-11	188.40	188.40	187.38	187.77	9588587
2018-01-12	178.06	181.48	177.40	179.37	77551299

```
(fb > 215).any()
```

```
open       True
high       True
low       False
close      True
volume     True
dtype: bool
```

```
(fb > 215).all()
```

```
open      False
high      False
low       False
close     False
volume     True
dtype: bool
```

```
(fb.volume.value_counts() > 1).sum()
```

```
0
```

```
volume_binned = pd.cut(fb.volume, bins=3, labels=['low', 'med', 'high'])
volume_binned.value_counts()
```

```
low     240
med       8
high      3
Name: volume, dtype: int64
```

```
fb[volume_binned == 'high'].sort_values(
 'volume', ascending=False
)
```

date	open	high	low	close	volume
2018-07-26	174.89	180.13	173.75	176.26	169803668
2018-03-20	167.47	170.20	161.95	168.15	129851768
2018-03-26	160.82	161.10	149.02	160.06	126116634

```
fb['2018-07-25':'2018-07-26']
```

date	open	high	low	close	volume
2018-07-25	215.715	218.62	214.27	217.50	64592585
2018-07-26	174.890	180.13	173.75	176.26	169803668

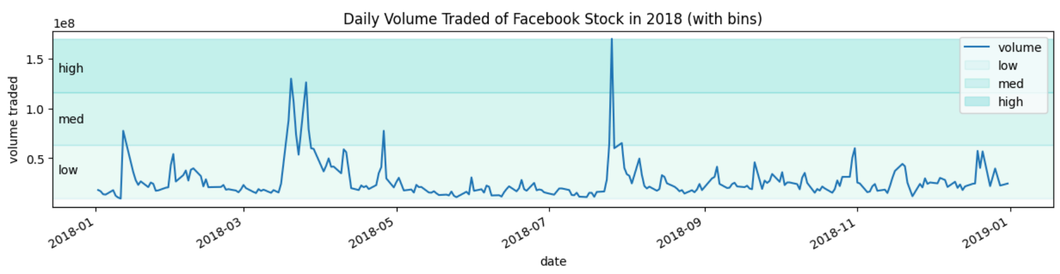
```
fb['2018-03-16':'2018-03-20']
```

date	open	high	low	close	volume
2018-03-16	184.49	185.33	183.41	185.09	24403438
2018-03-19	177.01	177.17	170.06	172.56	88140060
2018-03-20	167.47	170.20	161.95	168.15	129851768

```
import matplotlib.pyplot as plt
```

```
fb.plot(y='volume', figsize=(15, 3), title='Daily Volume Traded of Facebook Stock in 2018 (with bins)')
for bin_name, alpha, bounds in zip( #adding shaded region for volume bins
 ['low', 'med', 'high'], [0.1, 0.2, 0.3], pd.cut(fb.volume, bins=3).unique().categories.values
):
 plt.axhspan(bounds.left, bounds.right, alpha=alpha, label=bin_name, color='mediumturquoise')
 plt.annotate(bin_name, xy=('2017-12-17', (bounds.left + bounds.right)/2.1))

plt.ylabel('volume traded')
plt.legend()
plt.show()
#setting y label and displaying legend
```

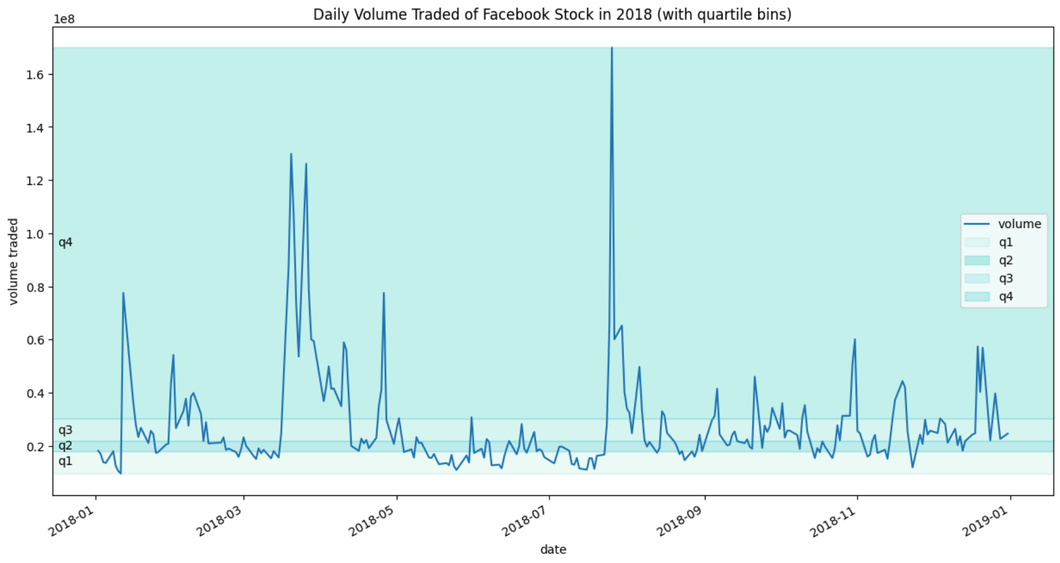



```
volume_qbinned = pd.qcut(fb.volume, q=4, labels=['q1', 'q2', 'q3', 'q4'])
volume_qbinned.value_counts()
```

```
q1    63
q2    63
q4    63
q3    62
Name: volume, dtype: int64
```

```
fb.plot(y='volume', figsize=(15, 8), title='Daily Volume Traded of Facebook Stock in 2018 (with quartile bins)')
for bin_name, alpha, bounds in zip( #adding shaded region for quartile bins
 ['q1', 'q2', 'q3', 'q4'], [0.1, 0.35, 0.2, 0.3], pd.qcut(fb.volume, q=4).unique().categories.values
):
 plt.axhspan(bounds.left, bounds.right, alpha=alpha, label=bin_name, color='mediumturquoise')
 plt.annotate(bin_name, xy=('2017-12-17', (bounds.left + bounds.right)/2.1))

plt.ylabel('volume traded')
plt.legend()
plt.show()
#setting ylabel and displaying legends
```




```
central_park_weather = weather.query(
 'station == "GHCND:USW00094728"'
).pivot(index='date', columns='datatype', values='value')
```

```
central_park_weather.SNOW.clip(0, 1).value_counts()
```

```
0.0    354
1.0     11
Name: SNOW, dtype: int64
```

```
oct_weather_z_scores = central_park_weather.loc[ #selecting october weather data for central park and calculates zscores
    '2018-10', ['TMIN', 'TMAX', 'PRCP']
].apply(lambda x: x.sub(x.mean()).div(x.std()))
oct_weather_z_scores.describe().T #generate descriptive statistics
```

datatype	count	mean	std	min	25%	50%	75%	max
TMIN	31.0	-1.790682e-16	1.0	-1.339112	-0.751019	-0.474269	1.065152	1.843511
TMAX	31.0	1.951844e-16	1.0	-1.305582	-0.870013	-0.138258	1.011643	1.604016
PRCP	31.0	4.655774e-17	1.0	-0.394438	-0.394438	-0.394438	-0.240253	3.936167

```
oct_weather_z_scores.query('PRCP > 3')
```

datatype	TMIN	TMAX	PRCP
date			
2018-10-27	-0.751019	-1.201045	3.936167

```
central_park_weather.loc['2018-10', 'PRCP'].describe()
```

```
count    31.000000
mean      2.941935
std       7.458542
min       0.000000
25%       0.000000
50%       0.000000
75%       1.150000
max      32.300000
Name: PRCP, dtype: float64
```

```
import numpy as np
fb.apply(
 lambda x: np.vectorize(lambda y: len(str(np.ceil(y))))(x)
).astype('int64').equals(
 fb.applymap(lambda x: len(str(np.ceil(x))))
)
```

```
True
```

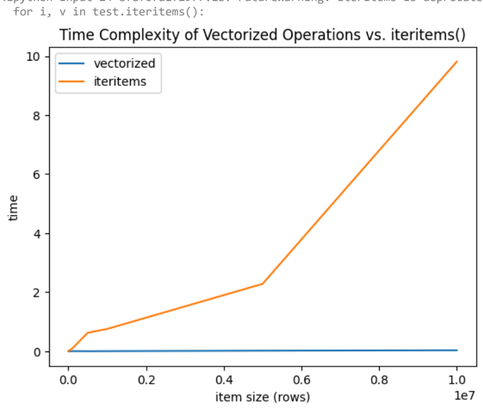
```
import time
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
np.random.seed(0) #setting random seed
vectorized_results = {} #initializing dicts to store results
iteritems_results = {}
for size in [10, 100, 1000, 10000, 100000, 500000, 1000000, 5000000, 10000000]: #iterate different size
  test = pd.Series(np.random.uniform(size=size)) #create test data

  start = time.time() #timing
  x = test + 10
  end = time.time()
  vectorized_results[size] = end - start

  start = time.time() #timing
  x = []
  for i, v in test.iteritems():
   x.append(v + 10)
  x = pd.Series(x)
  end = time.time()
  iteritems_results[size] = end - start

pd.DataFrame( #create dataframe to plot
  [pd.Series(vectorized_results, name='vectorized'), pd.Series(iteritems_results, name='iteritems')]
).T.plot(title='Time Complexity of Vectorized Operations vs. iteritems()')
plt.xlabel('item size (rows)') #add labels to axes
plt.ylabel('time')
plt.show()
```

```
<ipython-input-24-8f8f07a1f297>:18: FutureWarning: iteritems is deprecated and will be removed in a future version. Use .items instead.
  for i, v in test.iteritems():
```




```
central_park_weather['2018-10'].assign(
 rolling_PRCP=lambda x: x.PRCP.rolling('3D').sum()
)[['PRCP', 'rolling_PRCP']].head(7).T
```

```
<ipython-input-25-ae7842e6fdff>:1: FutureWarning: Indexing a DataFrame with a datetimelike index using a single string to slice the rows, like `frame[string]`, is deprecated and will be removed in a future version. Use `frame.loc[string]` instead.
  central_park_weather['2018-10'].assign(
```

date	2018-10-01	2018-10-02	2018-10-03	2018-10-04	2018-10-05	2018-10-06	2018-10-07
datatype							
PRCP	0.0	17.5	0.0	1.0	0.0	0.0	0.0
rolling_PRCP	0.0	17.5	17.5	18.5	1.0	1.0	0.0

```
central_park_weather['2018-10'].rolling('3D').mean().head(7).iloc[:,:6]
```

```
<ipython-input-26-2abb37634d3b>:1: FutureWarning: Indexing a DataFrame with a datetimelike index using a single string to slice the rows, like `frame[string]`, is deprecated and will be removed in a future version. Use `frame.loc[string]` instead.
  central_park_weather['2018-10'].rolling('3D').mean().head(7).iloc[:,:6]
```

datatype	AWND	PRCP	SNOW	SNWD	TMAX	TMIN
date						
2018-10-01	0.900000	0.000000	0.0	0.0	24.400000	17.200000
2018-10-02	0.900000	8.750000	0.0	0.0	24.700000	17.750000
2018-10-03	0.966667	5.833333	0.0	0.0	24.233333	17.566667
2018-10-04	0.800000	6.166667	0.0	0.0	24.233333	17.200000
2018-10-05	1.033333	0.333333	0.0	0.0	23.133333	16.300000
2018-10-06	0.833333	0.333333	0.0	0.0	22.033333	16.300000
2018-10-07	1.066667	0.000000	0.0	0.0	22.600000	17.400000

```
central_park_weather['2018-10-01':'2018-10-07'].rolling('3D').agg(
 {'TMAX': 'max', 'TMIN': 'min', 'AWND': 'mean', 'PRCP': 'sum'}
).join( # join with original data for comparison
 central_park_weather[['TMAX', 'TMIN', 'AWND', 'PRCP']],
 lsuffix='_rolling'
).sort_index(axis=1) # sort columns so rolling calcs are next to originals
```

datatype	AWND	AWND_rolling	PRCP	PRCP_rolling	TMAX	TMAX_rolling	TMIN	TMIN_rolling
date								
2018-10-01	0.9	0.900000	0.0	0.0	24.4	24.4	17.2	17.2
2018-10-02	0.9	0.900000	17.5	17.5	25.0	25.0	18.3	17.2
2018-10-03	1.1	0.966667	0.0	17.5	23.3	25.0	17.2	17.2
2018-10-04	0.4	0.800000	1.0	18.5	24.4	25.0	16.1	16.1
2018-10-05	1.6	1.033333	0.0	1.0	21.7	24.4	15.6	15.6
2018-10-06	0.5	0.833333	0.0	1.0	20.0	24.4	17.2	15.6
2018-10-07	1.1	1.066667	0.0	0.0	26.1	26.1	19.4	15.6

```
central_park_weather.PRCP.expanding().sum().equals(central_park_weather.PRCP.cumsum())
```

```
False
```

```
central_park_weather['2018-10-01':'2018-10-07'].expanding().agg(
 {'TMAX': np.max, 'TMIN': np.min, 'AWND': np.mean, 'PRCP': np.sum}
).join(
 central_park_weather[['TMAX', 'TMIN', 'AWND', 'PRCP']],
 lsuffix='_expanding'
).sort_index(axis=1)
```

datatype	AWND	AWND_expanding	PRCP	PRCP_expanding	TMAX	TMAX_expanding	TMIN	TMIN_expanding
date								
2018-10-01	0.9	0.900000	0.0	0.0	24.4	24.4	17.2	17.2
2018-10-02	0.9	0.900000	17.5	17.5	25.0	25.0	18.3	17.2
2018-10-03	1.1	0.966667	0.0	17.5	23.3	25.0	17.2	17.2
2018-10-04	0.4	0.825000	1.0	18.5	24.4	25.0	16.1	16.1
2018-10-05	1.6	0.980000	0.0	18.5	21.7	25.0	15.6	15.6
2018-10-06	0.5	0.900000	0.0	18.5	20.0	25.0	17.2	15.6
2018-10-07	1.1	0.928571	0.0	18.5	26.1	26.1	19.4	15.6

```
fb.assign(
 close_ewma=lambda x: x.close.ewm(span=5).mean()
).tail(10)[['close', 'close_ewma']]
```

	close	close_ewma
date		
2018-12-17	140.19	142.235433
2018-12-18	143.66	142.710289
2018-12-19	133.24	139.553526
2018-12-20	133.40	137.502350
2018-12-21	124.95	133.318234
2018-12-24	124.06	130.232156
2018-12-26	134.18	131.548104
2018-12-27	134.52	132.538736
2018-12-28	133.20	132.759157
2018-12-31	131.09	132.202772

```
def get_info(df):
 return '%d rows and %d columns and max closing z-score was %d' % (*df.shape, df.close.max())
fb['2018-Q1'].apply(lambda x: (x - x.mean())/x.std()).pipe(get_info)\
 == get_info(fb['2018-Q1'].apply(lambda x: (x - x.mean())/x.std()))
```

```
<ipython-input-31-df4ec8f2b7d9>:3: FutureWarning: Indexing a DataFrame with a datetimelike index using a single string to slice the rows, like `frame[string]`, is deprecated and will be removed in a future version. Use `frame.loc[string]` instead.
  fb['2018-Q1'].apply(lambda x: (x - x.mean())/x.std()).pipe(get_info)\
<ipython-input-31-df4ec8f2b7d9>:4: FutureWarning: Indexing a DataFrame with a datetimelike index using a single string to slice the rows, like `frame[string]`, is deprecated and will be removed in a future version. Use `frame.loc[string]` instead.
  == get_info(fb['2018-Q1'].apply(lambda x: (x - x.mean())/x.std()))
True
```

```
fb.pipe(pd.DataFrame.rolling, '20D').mean().equals(fb.rolling('20D').mean())
```

```
True
```

```
pd.DataFrame.rolling(fb, '20D').mean().equals(fb.rolling('20D').mean())
```

```
True
```

```
def window_calc(df, func, agg_dict, *args, **kwargs):
    """
    Run a window calculation of your choice on a DataFrame.
    Parameters:
    - df: The DataFrame to run the calculation on.
    - func: The window calculation method that takes df
    as the first argument.
    - agg_dict: Information to pass to `agg()`, could be a
    dictionary mapping the columns to the aggregation
    function to use, a string name for the function,
    or the function itself.
    - args: Positional arguments to pass to `func`.
    - kwargs: Keyword arguments to pass to `func`.
    Returns:
    - A new DataFrame object.
    """
    return df.pipe(func, *args, **kwargs).agg(agg_dict)
```

```
window_calc(fb, pd.DataFrame.expanding, np.median).head()
```

	open	high	low	close	volume
date					
2018-01-02	177.68	181.580	177.5500	181.420	18151903.0
2018-01-03	179.78	183.180	179.4400	183.045	17519233.0
2018-01-04	181.88	184.780	181.3300	184.330	16886563.0
2018-01-05	183.39	185.495	182.7148	184.500	15383729.5
2018-01-08	184.90	186.210	184.0996	184.670	16886563.0

```
window_calc(fb, pd.DataFrame.ewm, 'mean', span=3).head()
```

	open	high	low	close	volume
date					
2018-01-02	177.680000	181.580000	177.550000	181.420000	1.815190e+07
2018-01-03	180.480000	183.713333	180.070000	183.586667	1.730834e+07
2018-01-04	183.005714	185.140000	182.372629	184.011429	1.534980e+07
2018-01-05	184.384000	186.078667	183.736560	185.525333	1.440299e+07
2018-01-08	185.837419	187.534839	185.075110	186.947097	1.625679e+07

```
window_calc(
 central_park_weather['2018-10'],
 pd.DataFrame.rolling,
 {'TMAX': 'max', 'TMIN': 'min', 'AWND': 'mean', 'PRCP': 'sum'},
 '3D'
).head()
```

```
<ipython-input-43-9c6c3549bb1a>:2: FutureWarning: Indexing a DataFrame with a datetimelike index using a single string to slice the rows, like `frame[string]`, is deprecated and will be removed in a future version. Use `frame.loc[string]` instead.
  central_park_weather['2018-10'],
```

datatype	TMAX	TMIN	AWND	PRCP
date				
2018-10-01	24.4	17.2	0.900000	0.0
2018-10-02	25.0	17.2	0.900000	17.5
2018-10-03	25.0	17.2	0.966667	17.5
2018-10-04	25.0	16.1	0.800000	18.5
2018-10-05	24.4	15.6	1.033333	1.0

```
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()
```

	datatype	station	value	station_name
date				
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'])
)
fb.head()
```

	open	high	low	close	volume	trading_volume
date						
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

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

```
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
```

```
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
```

```
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

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

	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

```
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

```
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()
```

	open_mean	high_min	high_max	low_min	low_max	close_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

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

```
<ipython-input-57-708005da144d>:1: FutureWarning: Indexing a DataFrame with a datetimelike index using a single string to slice the rows, like `frame[string]`, is deprecated and will be removed in a future version. Use `frame.loc[string]` instead.
  weather['2018-10'].query('datatype == "PRCP"').groupby(
<ipython-input-57-708005da144d>:3: FutureWarning: The default value of numeric_only in DataFrameGroupBy.mean is deprecated. In a future version, numeric_only will default to False. Either specify numeric_only or select only columns which should be valid for the function.
  ).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-58-b1fc85a2870b>:3: FutureWarning: The default value of numeric_only in DataFrameGroupBy.sum is deprecated. In a future version, numeric_only will default to False. Either specify numeric_only or select only columns which should be valid for the function.
  ).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-59-81d267dea16f>:3: FutureWarning: The default value of numeric_only in DataFrameGroupBy.sum is deprecated. In a future version, numeric_only will default to False. Either specify numeric_only or select only columns which should be valid for the function.
  ).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-60-978bd538e3b2>:3: FutureWarning: The default value of numeric_only in DataFrameGroupBy.mean is deprecated. In a future version, numeric_only will default to False. Either specify numeric_only or select only columns which should be valid for the function.
  ).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-61-951bb02e735d>:3: FutureWarning: The default value of numeric_only in DataFrameGroupBy.mean is deprecated. In a future version, numeric_only will default to False. Either specify numeric_only or select only columns which should be valid for the function.
  ).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

```
weather\
.query('datatype == "PRCP"')\
.rename(dict(value='prcp'), axis=1)\ #rename the value column to prcp
.groupby(pd.Grouper(freq='D')).mean()\ #grouping by daily freauency and calculate the mean precipitation
.assign( #calculate the total precipitation monthly
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-62-e5da73b4d9c0>:4: FutureWarning: The default value of numeric_only in DataFrameGroupBy.mean is deprecated. In a future version, numeric_only will default to False. Either specify numeric_only or select only columns which should be valid for the function.
  .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()
```

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

```
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')
```

trading_volume	close	high	low	open	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	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	NaN	NaN	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	NaN	NaN	NaN	NaN
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	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	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	NaN	NaN	NaN

5 rows × 30 columns

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

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

```
pd.crosstab( #create cross tabulation of facebook trading volume by month
index=fb.trading_volume,
columns=fb.index.month,
colnames=['month'],
normalize='columns' #normalizing the values by columns
)
```

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

```
pd.crosstab( #create cross tabulation of facebook trading volume by month
index=fb.trading_volume,
columns=fb.index.month,
colnames=['month'],
values=fb.close,
aggfunc=np.mean #calculate the mean closing price
)
```

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

```
snow_data = weather.query('datatype == "SNOW"')
pd.crosstab( #create cross tabulation by station and month
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
)
```

station_name \ month	1	2	3	4	5	6	7	8	9	10	11	12	total observations of snow
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