Dataframe Operations

Setup

```
import numpy as np
import pandas as pd

weather = pd.read_csv('/content/drive/MyDrive/Module_8_files/nyc_weather_2018.csv', parse_dates=['date'])
weather.head()
```

	date	datatype	station	attributes	value
0	2018-01-01	PRCP	GHCND:US1CTFR0039	,,N,0800	0.0
1	2018-01-01	PRCP	GHCND:US1NJBG0015	,,N,1050	0.0
2	2018-01-01	SNOW	GHCND:US1NJBG0015	,,N,1050	0.0
3	2018-01-01	PRCP	GHCND:US1NJBG0017	,,N,0920	0.0
4	2018-01-01	SNOW	GHCND:US1NJBG0017	,,N,0920	0.0

```
fb = pd.read_csv('/content/drive/MyDrive/Module_8_files/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

Arithmetic and statistics

```
# added a new 'abs_z_score_volume' column to the fb dataframe
# where it was filled by the absolute z scored of the 'volume' column
# fb is filted to only show entries that exceed 3 in the said column
fb.assign(
 abs_z_score_volume=lambda x: x.volume.sub(x.volume.mean()).div(x.volume.std()).abs()
).query('abs_z_score_volume > 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

```
# creates an additional column named 'pct_change_rank' where it was filled
# by the absolute percentage change of the values in the 'volume' column

fb.assign(
 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')
```

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-							

```
# shows the fb drop in stocks from Jan 11 to Jan 12
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

```
# shows that fb's OHLC never have a low above 215
(fb > 215).any()
```

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

```
# Facebook's OHLC (open, high, low, and close) prices all had at least one day they were at $215 or less:
(fb > 215).all()
```

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

Binning and thresholds

```
(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

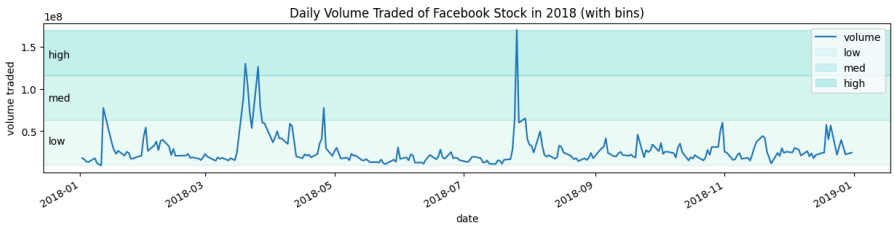
```
fb['2018-03-16':'2018-03-20']
```

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(
 ['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()
```

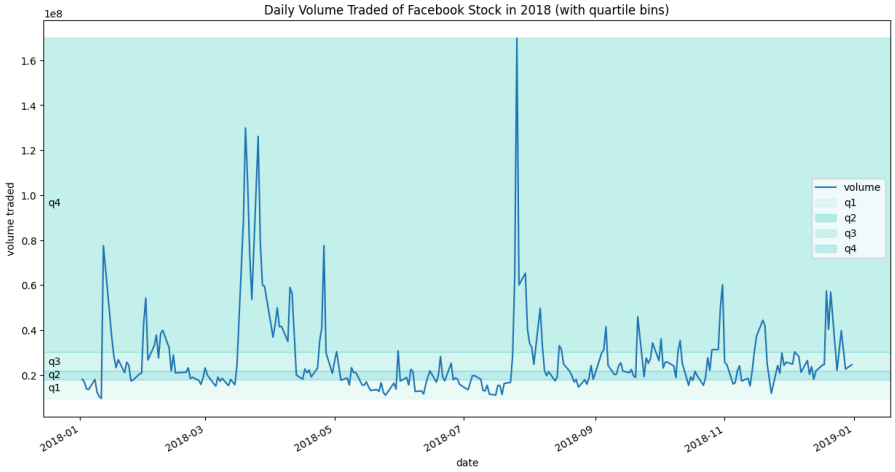



```
volume_qbinned = pd.qcut(fb.volume, q=4, labels=['q1', 'q2', 'q3', 'q4'])
volume_qbinned.value_counts()
```

```
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(
 ['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()
```




```
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    351
1.0     10
Name: SNOW, dtype: int64
```

Applying Functions

```
oct_weather_z_scores = central_park_weather.loc[
 '2018-10', ['TMIN', 'TMAX', 'PRCP']
 ].apply(lambda x: x.sub(x.mean()).div(x.std()))
oct_weather_z_scores.describe().T
```

datatype	count	mean	std	min	25%	50%	75%	max
TMIN	30.0	8.511710e-17	1.0	-1.368497	-0.781258	-0.409916	1.032274	1.809502
TMAX	30.0	2.463307e-16	1.0	-1.325795	-0.918525	-0.121315	0.979182	1.568424
PRCP	30.0	1.295260e-17	1.0	-0.401814	-0.401814	-0.401814	-0.239899	3.867458

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

datatype	TMIN	TMAX	PRCP
date			
2018-10-27	-0.781258	-1.221811	3.867458

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

```
count    30.000000
mean      3.040000
```

```
std       7.565694
min       0.000000
25%       0.000000
50%       0.000000
75%       1.225000
max      32.300000
Name: PRCP, dtype: float64
```

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

```
count    30.000000
mean      3.040000
std       7.565694
min       0.000000
25%       0.000000
50%       0.000000
75%       1.225000
max      32.300000
Name: PRCP, dtype: float64
```

```
import time
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
np.random.seed(0)
vectorized_results = {}
iteritems_results = {}
for size in [10, 100, 1000, 10000, 100000, 500000, 1000000, 5000000, 10000000]:
 test = pd.Series(np.random.uniform(size=size))

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

 start = time.time()
 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(
 [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)')
plt.ylabel('time')
plt.show()
```

Window Calculations

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

```
<ipython-input-24-853d3f44ac2c>:1: FutureWarning: Indexing a DataFrame with a datetimeli
  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-25-2abb37634d3b>:1: FutureWarning: Indexing a DataFrame with a datetimeli
  central_park_weather['2018-10'].rolling('3D').mean().head(7).iloc[:,:6]
```

datatype	ADPT	ASLP	ASTP	AWBT	AWND	PRCP
date						
2018-10-01	172.000000	10247.000000	10200.000000	189.000000	0.900000	0.000000
2018-10-02	180.500000	10221.500000	10176.000000	194.500000	0.900000	8.750000
2018-10-03	172.333333	10205.333333	10159.000000	187.000000	0.966667	5.833333
2018-10-04	176.000000	10175.000000	10128.333333	187.000000	0.800000	6.166667
2018-10-05	155.666667	10177.333333	10128.333333	170.333333	1.033333	0.333333
2018-10-06	157.333333	10194.333333	10145.333333	170.333333	0.833333	0.333333
2018-10-07	163.000000	10217.000000	10165.666667	177.666667	1.066667	0.000000

```
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

```
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_e
date								
2018-10-01	0.9	0.900000	0.0	0.0	24.4	24.4	17.2	
2018-10-02	0.9	0.900000	17.5	17.5	25.0	25.0	18.3	
2018-10-03	1.1	0.966667	0.0	17.5	23.3	25.0	17.2	
2018-10-04	0.4	0.825000	1.0	18.5	24.4	25.0	16.1	

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

date	close	close_ewma
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-29-df4ec8f2b7d9>:3: FutureWarning: Indexing a DataFrame with a datetimelike index using a single string to slice the rows
  fb['2018-Q1'].apply(lambda x: (x - x.mean())/x.std()).pipe(get_info)\
<ipython-input-29-df4ec8f2b7d9>:4: FutureWarning: Indexing a DataFrame with a datetimelike index using a single string to slice the rows
  == 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
```

```
!pip install window_calc
```

```
ERROR: Could not find a version that satisfies the requirement window_calc (from versions: none)
ERROR: No matching distribution found for window_calc
```

```
from window_calc import window_calc
window_calc??
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-32-d7a126469004> in <cell line: 1>()
----> 1 from window_calc import window_calc
      2 get_ipython().run_line_magic('pinfo2', 'window_calc')

ModuleNotFoundError: No module named 'window_calc'

---------------------------------------------------------------------------
NOTE: If your import is failing due to a missing package, you can
manually install dependencies using either !pip or !apt.

To view examples of installing some common dependencies, click the
"Open Examples" button below.
---------------------------------------------------------------------------
```

OPEN EXAMPLES

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

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

```
window_calc(
 central_park_weather['2018-10'],
```

```
    pd.DataFrame.rolling,
    {'TMAX': 'max', 'TMIN': 'min', 'AWND': 'mean', 'PRCP': 'sum'},
    '3D'
  ).head()
```