

```
import numpy as np
import pandas as pd
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](#)

```
fb['2018-10-11':'2018-10-15']
```

	open	high	low	close	volume	trading_volume	
date							
2018-10-11	150.13	154.81	149.1600	153.35	35338901		low
2018-10-12	156.73	156.89	151.2998	153.74	25293492		low
2018-10-15	153.32	155.57	152.5500	153.52	15433521		low

```
fb['2018-q1'].equals(fb['2018-01':'2018-03'])
```

```
<ipython-input-3-f01e3c270a70>:1: FutureWarning: Indexing a DataFrame with a datetimelike index using a single string to slice the rows,
fb['2018-q1'].equals(fb['2018-01':'2018-03'])
True
```

```
fb.first('1W')
```

	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

```
fb.last('1W')
```

	open	high	low	close	volume	trading_volume	
date							
2018-12-31	134.45	134.64	129.95	131.09	24625308		low

```
stock_data_per_minute = pd.read_csv('/content/fb_week_of_may_20_per_minute.csv', index_col='date', parse_dates=True,
date_parser=lambda x: pd.to_datetime(x, format='%Y-%m-%d %H-%M')
)
stock_data_per_minute.head()
```

	open	high	low	close	volume	
date						
2019-05-20 09:30:00	181.6200	181.6200	181.6200	181.6200	159049.0	
2019-05-20 09:31:00	182.6100	182.6100	182.6100	182.6100	468017.0	
2019-05-20 09:32:00	182.7458	182.7458	182.7458	182.7458	97258.0	
2019-05-20 09:33:00	182.9500	182.9500	182.9500	182.9500	43961.0	
2019-05-20 09:34:00	183.0600	183.0600	183.0600	183.0600	79562.0	

Next steps: [View recommended plots](#)

```
stock_data_per_minute.groupby(pd.Grouper(freq='1D')).agg({
'open': 'first',
'high': 'max',
'low': 'min',
'close': 'last',
'volume': 'sum'
})
```



	open	high	low	close	volume
date					
2019-05-20	181.62	184.1800	181.6200	182.72	10044838.0
2019-05-21	184.53	185.5800	183.9700	184.82	7198405.0
2019-05-22	184.81	186.5603	184.0120	185.32	8412433.0
2019-05-23	182.50	183.7300	179.7559	180.87	12479171.0
2019-05-24	182.33	183.5227	181.0400	181.06	7686030.0

stock_data_per_minute.at_time('9:30')

	open	high	low	close	volume
date					
2019-05-20 09:30:00	181.62	181.62	181.62	181.62	159049.0
2019-05-21 09:30:00	184.53	184.53	184.53	184.53	58171.0
2019-05-22 09:30:00	184.81	184.81	184.81	184.81	41585.0
2019-05-23 09:30:00	182.50	182.50	182.50	182.50	121930.0
2019-05-24 09:30:00	182.33	182.33	182.33	182.33	52681.0

stock_data_per_minute.between_time('15:59', '16:00')

	open	high	low	close	volume
date					
2019-05-20 15:59:00	182.915	182.915	182.915	182.915	134569.0
2019-05-20 16:00:00	182.720	182.720	182.720	182.720	1113672.0
2019-05-21 15:59:00	184.840	184.840	184.840	184.840	61606.0
2019-05-21 16:00:00	184.820	184.820	184.820	184.820	801080.0
2019-05-22 15:59:00	185.290	185.290	185.290	185.290	96099.0
2019-05-22 16:00:00	185.320	185.320	185.320	185.320	1220993.0
2019-05-23 15:59:00	180.720	180.720	180.720	180.720	109648.0
2019-05-23 16:00:00	180.870	180.870	180.870	180.870	1329217.0
2019-05-24 15:59:00	181.070	181.070	181.070	181.070	52994.0
2019-05-24 16:00:00	181.060	181.060	181.060	181.060	764906.0

```
shares_traded_in_first_30_min = stock_data_per_minute\  
.between_time('9:30', '10:00')\  
.groupby(pd.Grouper(freq='1D'))\  
.filter(lambda x: (x.volume > 0).all())\  
.volume.mean()  
shares_traded_in_last_30_min = stock_data_per_minute\  
.between_time('15:30', '16:00')\  
.groupby(pd.Grouper(freq='1D'))\  
.filter(lambda x: (x.volume > 0).all())\  
.volume.mean()  
shares_traded_in_first_30_min - shares_traded_in_last_30_min
```

18592.967741935485

```
pd.DataFrame(  
dict(before=stock_data_per_minute.index, after=stock_data_per_minute.index.normalize())  
)
```

	before	after
0	2019-05-20 09:30:00	2019-05-20
1	2019-05-20 09:31:00	2019-05-20
2	2019-05-20 09:32:00	2019-05-20
3	2019-05-20 09:33:00	2019-05-20
4	2019-05-20 09:34:00	2019-05-20

stock_data_per_minute.index.to_series().dt.normalize().head()

```
date  
2019-05-20 09:30:00    2019-05-20  
2019-05-20 09:31:00    2019-05-20  
2019-05-20 09:32:00    2019-05-20  
2019-05-20 09:33:00    2019-05-20  
2019-05-20 09:34:00    2019-05-20  
Name: date, dtype: datetime64[ns]
```

```
fb.assign(  
    prior_close=lambda x: x.close.shift(),  
    after_hours_change_in_price=lambda x: x.open - x.prior_close,  
    abs_change=lambda x: x.after_hours_change_in_price.abs()  
)
```



	open	high	low	close	volume	trading_volume	prior_close	after_hours
date								
2018-07-26	174.89	180.13	173.75	176.26	169803668		high	217.50
2018-04-26	173.22	176.27	170.80	174.16	77556934		med	159.69
2018-01-12	178.06	181.48	177.40	179.37	77551299		med	187.77

pd.date_range('2018-01-01', freq='D', periods=5) + pd.Timedelta('9 hours 30 minutes')

DatetimeIndex(['2018-01-01 09:30:00', '2018-01-02 09:30:00',
'2018-01-03 09:30:00', '2018-01-04 09:30:00',
'2018-01-05 09:30:00'],
dtype='datetime64[ns]', freq='D')

fb['2018-09'].first_valid_index()

<ipython-input-28-d8ca41528993>:1: FutureWarning: Indexing a DataFrame with a datetimelike index using a single string to slice the rows
fb['2018-09'].first_valid_index()
Timestamp('2018-09-04 00:00:00')

fb['2018-09'].last_valid_index()

<ipython-input-31-ef6e024573c9>:1: FutureWarning: Indexing a DataFrame with a datetimelike index using a single string to slice the rows
fb['2018-09'].last_valid_index()
Timestamp('2018-09-28 00:00:00')

'2018-09-30' in fb.index

False

fb.asof('2018-09-30')

open 168.33
high 168.79
low 162.56
close 164.46
volume 34265638
trading_volume low
Name: 2018-09-30 00:00:00, dtype: object

(
fb.drop(columns='trading_volume')
- fb.drop(columns='trading_volume').shift()
)
.equals(
fb.drop(columns='trading_volume').diff()
)

True

fb.drop(columns='trading_volume').diff().head()

	open	high	low	close	volume
date					
2018-01-02	NaN	NaN	NaN	NaN	NaN
2018-01-03	4.20	3.20	3.7800	3.25	-1265340.0
2018-01-04	3.02	1.43	2.7696	-0.34	-3005667.0
2018-01-05	0.69	0.69	0.8304	2.52	-306361.0
2018-01-08	1.61	2.00	1.4000	1.43	4420191.0

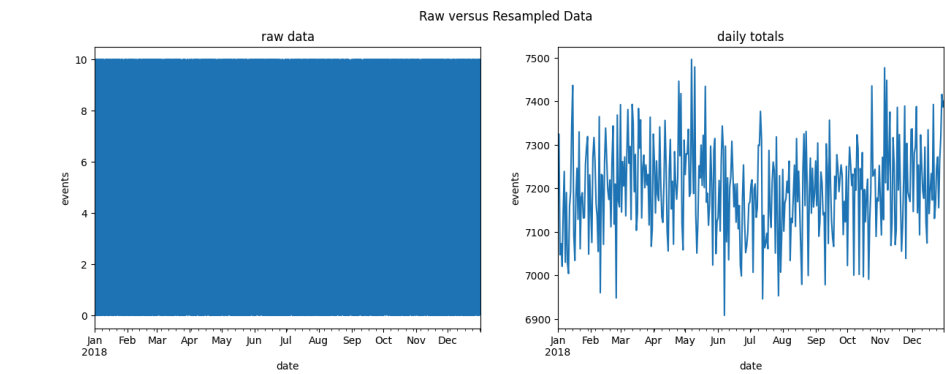
fb.drop(columns='trading_volume').diff(-3).head()

	open	high	low	close	volume
date					
2018-01-02	-7.91	-5.32	-7.3800	-5.43	4577368.0
2018-01-03	-5.32	-4.12	-5.0000	-3.61	-1108163.0
2018-01-04	-3.80	-2.59	-3.0004	-3.54	1487839.0
2018-01-05	-1.35	-0.99	-0.7000	-0.99	3044641.0
2018-01-08	-1.20	0.50	-1.0500	0.51	8406139.0

import matplotlib.pyplot as plt



```
np.random.seed(0)
index = pd.date_range('2018-01-01', freq='T', periods=365*24*60)
raw = pd.DataFrame(
    np.random.uniform(0, 10, size=index.shape[0]), index=index
)
fig, axes = plt.subplots(1, 2, figsize=(15, 5))
raw.plot(legend=False, ax=axes[0], title='raw data')
raw.resample('1D').sum().plot(legend=False, ax=axes[1], title='daily totals')
for ax in axes:
    ax.set_xlabel('date')
    ax.set_ylabel('events')
plt.suptitle('Raw versus Resampled Data')
plt.show()
```



```
stock_data_per_minute.head()
```

	open	high	low	close	volume
date					
2019-05-20 09:30:00	181.6200	181.6200	181.6200	181.6200	159049.0
2019-05-20 09:31:00	182.6100	182.6100	182.6100	182.6100	468017.0
2019-05-20 09:32:00	182.7458	182.7458	182.7458	182.7458	97258.0
2019-05-20 09:33:00	182.9500	182.9500	182.9500	182.9500	43961.0
2019-05-20 09:34:00	183.0600	183.0600	183.0600	183.0600	79562.0

Next steps: [View recommended plots](#)

```
stock_data_per_minute.resample('1D').agg({
'open': 'first',
'high': 'max',
'low': 'min',
'close': 'last',
'volume': 'sum'
})
```

	open	high	low	close	volume
date					
2019-05-20	181.62	184.1800	181.6200	182.72	10044838.0
2019-05-21	184.53	185.5800	183.9700	184.82	7198405.0
2019-05-22	184.81	186.5603	184.0120	185.32	8412433.0
2019-05-23	182.50	183.7300	179.7559	180.87	12479171.0
2019-05-24	182.33	183.5227	181.0400	181.06	7686030.0

```
fb.resample('Q').mean()
```

<ipython-input-40-f6fd3d834d43>:1: FutureWarning: The default value of numeric_only in L
fb.resample('Q').mean()

	open	high	low	close	volume
date					
2018-03-31	179.472295	181.794659	177.040428	179.551148	3.292640e+07
2018-06-30	180.373770	182.277689	178.595964	180.704688	2.405532e+07
2018-09-30	180.812130	182.890886	178.955229	181.028492	2.701982e+07
2018-12-31	145.272460	147.620121	142.718943	144.868730	2.697433e+07



```
fb.drop(columns='trading_volume').resample('Q').apply(
lambda x: x.last('1D').values - x.first('1D').values
)
```

```
date
2018-03-31    [[-22.53, -20.160000000000025, -23.4100000000...
2018-06-30    [[39.509999999999999, 38.399700000000024, 39.84...
2018-09-30    [[-25.039999999999992, -28.659999999999997, -2...
2018-12-31    [[-28.580000000000013, -31.24000000000001, -31...
Freq: Q-DEC, dtype: object
```

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

	price
date	
2019-05-20 09:30:00	181.6200
2019-05-20 09:31:00	182.6100
2019-05-20 09:32:00	182.7458
2019-05-20 09:33:00	182.9500
2019-05-20 09:34:00	183.0600

Next steps: [View recommended plots](#)

```
melted_stock_data.resample('1D').ohlc()['price']
```

	open	high	low	close
date				
2019-05-20	181.62	184.1800	181.6200	182.72
2019-05-21	184.53	185.5800	183.9700	184.82
2019-05-22	184.81	186.5603	184.0120	185.32
2019-05-23	182.50	183.7300	179.7559	180.87
2019-05-24	182.33	183.5227	181.0400	181.06

```
fb.resample('6H').asfreq().head()
```

	open	high	low	close	volume	trading_volume
date						
2018-01-02 00:00:00	177.68	181.58	177.55	181.42	18151903.0	low
2018-01-02 06:00:00	NaN	NaN	NaN	NaN	NaN	NaN
2018-01-02 12:00:00	NaN	NaN	NaN	NaN	NaN	NaN
2018-01-02 18:00:00	NaN	NaN	NaN	NaN	NaN	NaN
2018-01-03 00:00:00	181.88	184.78	181.33	184.67	16886563.0	low

```
fb.resample('6H').pad().head()
```

```
fb.resample('6H').fillna('nearest').head()
```

```
fb.resample('6H').asfreq().assign(
    volume=lambda x: x.volume.fillna(0),
    close=lambda x: x.close.fillna(method='ffill'),
    open=lambda x: np.where(x.open.isnull(), x.close, x.open),
    high=lambda x: np.where(x.high.isnull(), x.close, x.high),
    low=lambda x: np.where(x.low.isnull(), x.close, x.low)
).head()
```

	open	high	low	close	volume	trading_volume
date						
2018-01-02 00:00:00	177.68	181.58	177.55	181.42	18151903.0	low
2018-01-02 06:00:00	181.42	181.42	181.42	181.42	0.0	NaN
2018-01-02 12:00:00	181.42	181.42	181.42	181.42	0.0	NaN
2018-01-02 18:00:00	181.42	181.42	181.42	181.42	0.0	NaN
2018-01-03 00:00:00	181.88	184.78	181.33	184.67	16886563.0	low



```
import sqlite3

fb_prices.index.second.unique()

Int64Index([0], dtype='int64', name='date')

aapl_prices.index.second.unique()

Int64Index([ 0, 52, 36, 34, 55, 35,  7, 12, 59, 17,  5, 20, 26, 23, 54, 49, 19,
            53, 11, 22, 13, 21, 10, 46, 42, 38, 33, 18, 16,  9, 56, 39,  2, 50,
            31, 58, 48, 24, 29,  6, 47, 51, 40,  3, 15, 14, 25,  4, 43,  8, 32,
            27, 30, 45,  1, 44, 57, 41, 37, 28],
           dtype='int64', name='date')
```

```
pd.merge_asof(
fb_prices, aapl_prices,
left_index=True, right_index=True, # datetimes are in the index
# merge with nearest minute
direction='nearest', tolerance=pd.Timedelta(30, unit='s')
).head()
```

	FB	AAPL
date		
2019-05-20 09:30:00	181.6200	183.5200
2019-05-20 09:31:00	182.6100	NaN
2019-05-20 09:32:00	182.7458	182.8710
2019-05-20 09:33:00	182.9500	182.5000
2019-05-20 09:34:00	183.0600	182.1067

```
pd.merge_ordered(
fb_prices.reset_index(), aapl_prices.reset_index()
).set_index('date').head()
```

	FB	AAPL
date		
2019-05-20 09:30:00	181.6200	183.520
2019-05-20 09:31:00	182.6100	NaN
2019-05-20 09:31:52	NaN	182.871
2019-05-20 09:32:00	182.7458	NaN
2019-05-20 09:32:36	NaN	182.500

```
pd.merge_ordered(
fb_prices.reset_index(), aapl_prices.reset_index(),
fill_method='ffill'
).set_index('date').head()
```

	FB	AAPL
date		
2019-05-20 09:30:00	181.6200	183.520
2019-05-20 09:31:00	182.6100	183.520
2019-05-20 09:31:52	182.6100	182.871
2019-05-20 09:32:00	182.7458	182.871
2019-05-20 09:32:36	182.7458	182.500

Comments

Delving into time series analysis, this document addresses the specific challenges and techniques associated with time-dependent data. Covering manipulation, forecasting, and visualization of time series data, it provides a comprehensive overview of working with datasets where timing is a critical dimension. The examples of financial and weather data highlight the relevance of time series analysis in diverse fields.

Conclusion

Time series analysis is a complex yet rewarding area of data science, with applications ranging from stock market analysis to weather forecasting. The document showcases how pandas can be used to handle date and time data, perform rolling window calculations, and visualize time series data. Understanding these techniques is essential for anyone looking to analyze temporal data patterns, forecast future trends, and derive insights from historical data sequences.

