1

Acquire Free Financial Market Data with Cutting-Edge Python Libraries

A May 2017 Economist cover declared data to be the world's most valuable resource. It's none truer than in algorithmic trading. As algorithmic traders, it's our job to acquire and make sense of billions of rows of market data for use in trading algorithms. In this context, it's crucial to gather high-quality, reliable data that can adequately support trading algorithms and market research. Luckily for us, it's possible to acquire high-quality data for free (or nearly free).

This chapter offers recipes for a series of different Python libraries—including the cutting-edge OpenBB Platform—to acquire free financial market data using Python. One of the primary challenges most nonprofessional traders face is getting all the data required for analysis together in one place. The OpenBB Platform addresses this issue. We'll dive into acquiring data for a variety of assets, including stocks, options, futures (both continuous and individual contracts), and Fama-French factors.

One crucial point to remember is that data can vary across different sources. For instance, prices from two sources might differ due to distinct data sourcing methods or different adjustment methods for corporate actions. Some of the libraries we'll cover might download data for the same asset from the same source. However, libraries vary in how they return that data based on options that help you preprocess the data in preparation for research.

Lastly, while we'll focus heavily on mainstream financial data in this chapter, financial data is not limited to prices. The concept of "alternative data," which includes non-traditional data sources such as satellite images, web traffic data, or customer reviews, can be an important source of information for developing trading strategies. The Python tools to acquire and process this type of data are outside the scope of this book. We've intentionally left out the methods of acquiring and processing this type of data since it's covered in other resources dedicated to the topic.

In this chapter, we'll cover the following recipes:

- Working with stock market data with the OpenBB Platform
- Fetching historic futures data with the OpenBB Platform
- Navigating options market data with the OpenBB Platform
- Harnessing factor data using **`pandas_datareader`**

Technical requirements

This book relies on the Anaconda distribution of Python. We'll use Jupyter Notebook and Python script files to write the code. Unless specified otherwise, all the code can be written in Jupyter Notebooks.

Download and install the Anaconda distribution of Python. You can do this by going to **https://www.anaconda.com/download**. Depending on your operating system, the instructions for downloading and installing will vary. Please refer to the Anaconda documentation for detailed instructions.

Anaconda ships with a package manager called **`conda`**. Package managers make it easy to install, remove, and update Python packages. There's a great cheat sheet for the **`conda`** package manager that you can download from https://docs.conda.io/projects/conda/en/4.6.0/_downloads/52a95608c49671267e40c689e0bc00ca/conda-cheatsheet.pdf.

Once you've installed the Anaconda distribution, open your Terminal on Mac or Linux or the Anaconda Prompt on Windows. If you're a Windows user, make sure to use the Command Prompt instead of the Powershell prompt. Then follow these steps:

1. Update the **conda** package manager:

   ```
   conda update -n base conda -y
   ```

2. Create a virtual environment:

   ```
   conda create -n my-quant-stack python=3.10 -y
   ```

3. After the installation process is complete, activate the environment:

   ```
   conda activate my-quant-stack
   ```

4. Install Jupyter Notebook using the package manager that ships with Python, **pip**:

   ```
   pip install notebook matplotlib
   ```

This will set up a virtual environment using Python 3.10 and install Jupyter Notebook.

This chapter will use Two Python libraries to acquire financial market data: the OpenBB Platform and **pandas_datareader**. The good news is that installing the OpenBB Platform installs many of the libraries you will need to acquire financial market data, including **pandas_datareader**. As such, there is no need to install the libraries separately.

Install the OpenBB Platform with all extensions and providers (both officially supported and community-maintained ones) using **pip**:

```
pip install openbb[all]
```

This is the easiest way to set up the OpenBB Platform for this book.

IMPORTANT NOTE

In a macOS zsh Terminal shell, add quotation marks around the library name: **`"openbb[all]"`**

To install a single extension:

```
pip install openbb[charting]
pip install openbb[ta]
```

Or install a single provider:

```
pip install openbb[yfinance]
```

To install the Nightly distribution (this installs all extras by default):

```
pip install openbb-nightly
```

IMPORTANT NOTE

At the time of writing, installing the OpenBB Platform using **`pip`** *isn't compatible with environments such as Google Colab and Kaggle since they come with preinstalled packages that can conflict with the ones used with the OpenBB Platform. If you run into trouble installing the OpenBB Platform, please check the online documentation for the most up-to-date instructions.*

Working with stock market data with the OpenBB Platform

You may remember the **meme stock** hysteria that sent GameStop's stock up 1,744% in January 2021. One of the good things that came from that episode was the GameStonk terminal, now rebranded as OpenBB. OpenBB is the most popular open-source finance projects on GitHub for good reason: it provides a single interface to access hundreds of data feeds from one place in a standard way. OpenBB has a command-line interface that is great for manual investment research. But when it's time to get data into Python, you want the OpenBB Platform. This recipe will guide you through the process of using the OpenBB Platform to fetch stock market data.

Getting ready…

By now, you should have the OpenBB Platform installed in your virtual environment. If not, go back to the beginning of this chapter and get it set up. The OpenBB Platform is free to use and offers a web-based UI to manage your configuration files, store API keys, and get code walkthroughs and examples. Sign up for a free Hub account at https://my.openbb.co/login. The popular course, *Getting Started with Python for Quant Finance*, uses OpenBB exclusively for all the code. Check out **https://www.pyquantnews.com/getting-started-with-python-for-quant-finance** for information on how to join.

How to do it…

Using the OpenBB Platform involves one import:

1. Import the OpenBB Platform:

```
from openbb import obb
obb.user.preferences.output_type = "dataframe"
```

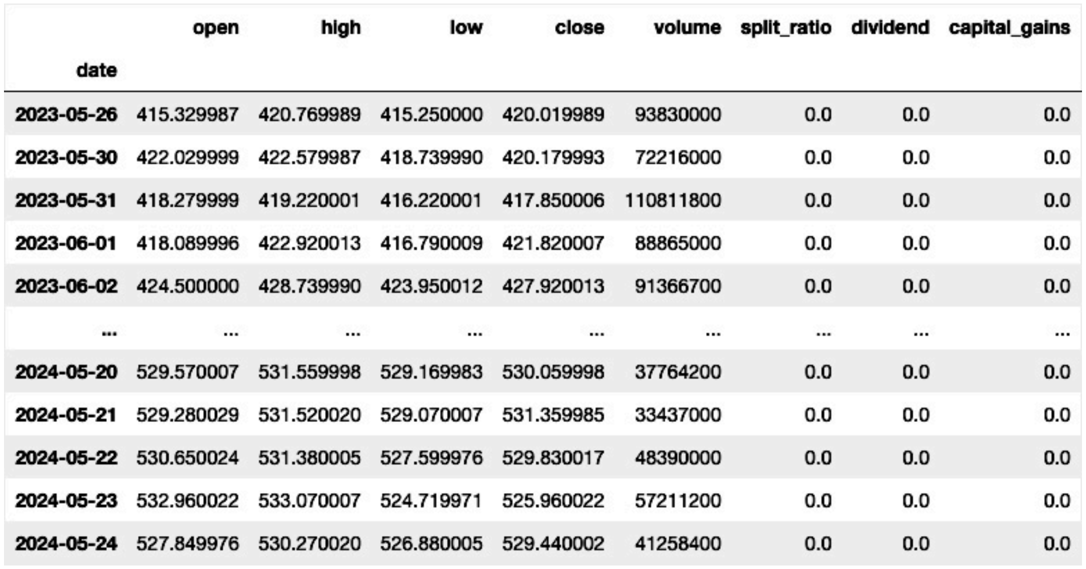
2. Use the **`historical`** method to download price data for the SPY ETF:

   ```
   data = obb.equity.price.historical("SPY", provider="yfinance")
   ```

3. Inspect the resulting DataFrame:

   ```
   print(data)
   ```

   Running the preceding code generates a pandas DataFrame and prints the data to the screen:



	open	high	low	close	volume	split_ratio	dividend	capital_gains
date								
2023-05-26	415.329987	420.769989	415.250000	420.019989	93830000	0.0	0.0	0.0
2023-05-30	422.029999	422.579987	418.739990	420.179993	72216000	0.0	0.0	0.0
2023-05-31	418.279999	419.220001	416.220001	417.850006	110811800	0.0	0.0	0.0
2023-06-01	418.089996	422.920013	416.790009	421.820007	88865000	0.0	0.0	0.0
2023-06-02	424.500000	428.739990	423.950012	427.920013	91366700	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...
2024-05-20	529.570007	531.559998	529.169983	530.059998	37764200	0.0	0.0	0.0
2024-05-21	529.280029	531.520020	529.070007	531.359985	33437000	0.0	0.0	0.0
2024-05-22	530.650024	531.380005	527.599976	529.830017	48390000	0.0	0.0	0.0
2024-05-23	532.960022	533.070007	524.719971	525.960022	57211200	0.0	0.0	0.0
2024-05-24	527.849976	530.270020	526.880005	529.440002	41258400	0.0	0.0	0.0

Figure 1.1: Historic price data for SPY

How it works…

The OpenBB Platform follows an easy-to-understand namespace convention. All the methods for acquiring stock price data are methods on **`openbb.equity`**.

The **`historical`** method accepts a ticker symbol and returns the open, high, low, close, adjusted close, volume, dividend, and split adjustments in a pandas DataFrame. The additional parameters you can specify are as follows:

- **`start_date`**: Start date to get data from with

- **`interval`**: Interval (in minutes) to get data—that is, 1, 5, 15, 30, 60, or 1,440
- **`end_date`**: End date to get data from with
- **`provider`**: Source of data extracted

There's more…

An important benefit of using the OpenBB Platform is choosing your data source. By default, the OpenBB Platform will attempt to download data from free sources such as Yahoo! Finance. In most OpenBB Platform calls, you can indicate a different source. To use a source that requires an API key (either free or paid), you can configure it in the OpenBB Hub.

> *TIP*

> *Check out the OpenBB Platform documentation for the latest functionality:* https://docs.openbb.co.

Let's look at some more of the functions of the OpenBB Platform.

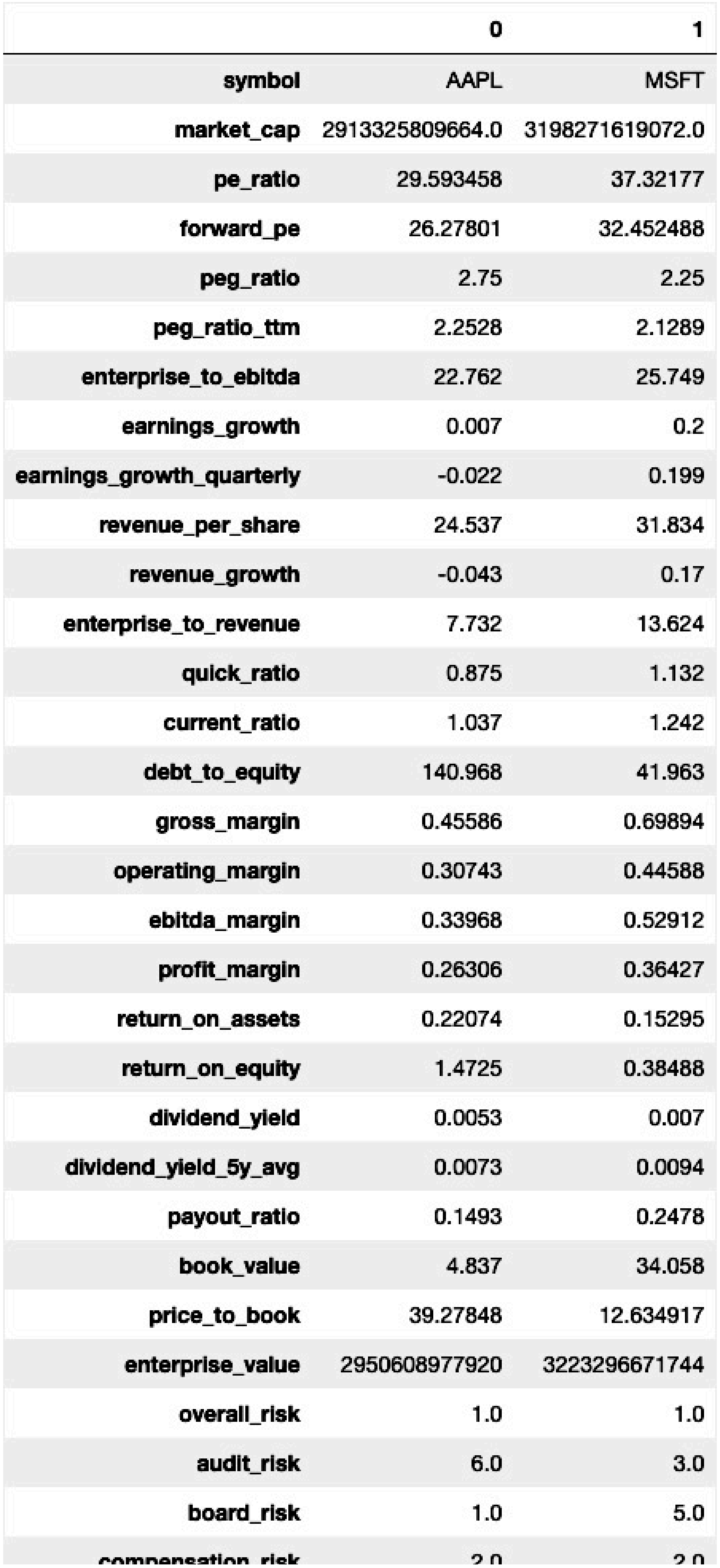
Comparison of fundamental data

Not only can the OpenBB Platform download fundamental data in an organized and usable way, but it can also concatenate it in a single Pandas DataFrame for further analysis.

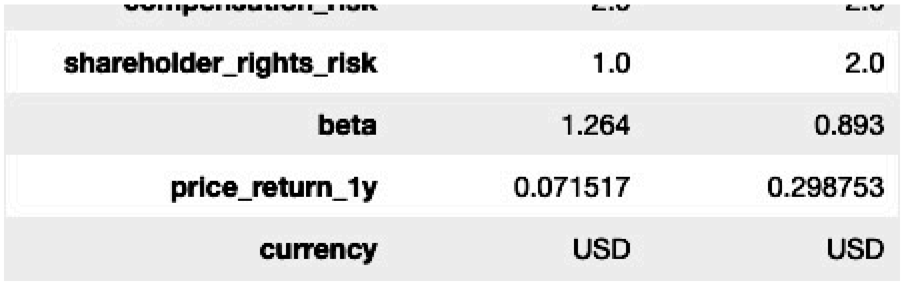
We can use the following code to see the balance sheet metrics from **`AAPL`** and **`MSFT`**:

```
obb.equity.fundamental.metrics(
    "AAPL,MSFT",
    provider="yfinance"
)
```

The output of the preceding snippet is a pandas DataFrame with fundamental data for each ticker that was passed:



	0	1
symbol	AAPL	MSFT
market_cap	2913325809664.0	3198271619072.0
pe_ratio	29.593458	37.32177
forward_pe	26.27801	32.452488
peg_ratio	2.75	2.25
peg_ratio_ttm	2.2528	2.1289
enterprise_to_ebitda	22.762	25.749
earnings_growth	0.007	0.2
earnings_growth_quarterly	-0.022	0.199
revenue_per_share	24.537	31.834
revenue_growth	-0.043	0.17
enterprise_to_revenue	7.732	13.624
quick_ratio	0.875	1.132
current_ratio	1.037	1.242
debt_to_equity	140.968	41.963
gross_margin	0.45586	0.69894
operating_margin	0.30743	0.44588
ebitda_margin	0.33968	0.52912
profit_margin	0.26306	0.36427
return_on_assets	0.22074	0.15295
return_on_equity	1.4725	0.38488
dividend_yield	0.0053	0.007
dividend_yield_5y_avg	0.0073	0.0094
payout_ratio	0.1493	0.2478
book_value	4.837	34.058
price_to_book	39.27848	12.634917
enterprise_value	2950608977920	3223296671744
overall_risk	1.0	1.0
audit_risk	6.0	3.0
board_risk	1.0	5.0
compensation_risk	2.0	2.0



compensation_risk	2.0	2.0
shareholder_rights_risk	1.0	2.0
beta	1.264	0.893
price_return_1y	0.071517	0.298753
currency	USD	USD

Figure 1.2: Balance sheet data for MSFT and AAPL

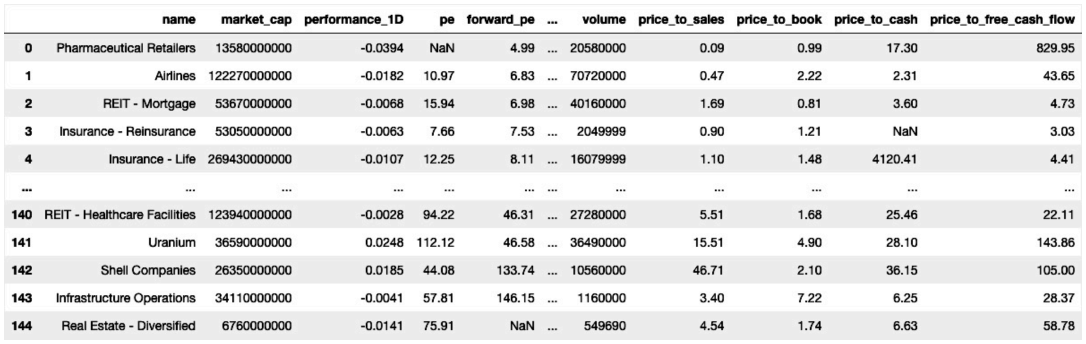
Building stock screeners

One of the most powerful features of the OpenBB Platform is the custom stock screener. It uses the **Finviz** stock screener under the hood and surfaces metrics across a range of stocks based on either pre-built or custom criteria. See the documentation for more on how to use the OpenBB screener functions (https://docs.openbb.co/platform/reference/equity/screener):

1. Create an overview screener based on a list of stocks using the default view:

   ```
   obb.equity.compare.groups(
       group="industry",
       metric="valuation",
       provider="finviz"
   )
   ```

   The output of the preceding snippet is the following pandas DataFrame:



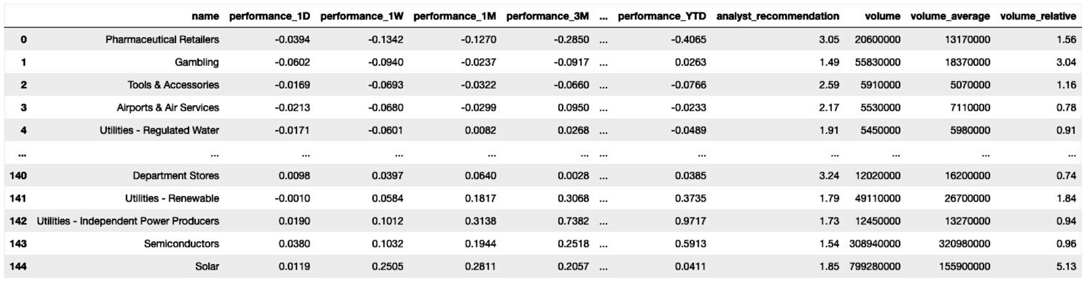
	name	market_cap	performance_1D	pe	forward_pe	...	volume	price_to_sales	price_to_book	price_to_cash	price_to_free_cash_flow
0	Pharmaceutical Retailers	13580000000	-0.0394	NaN	4.99	...	20580000	0.09	0.99	17.30	829.95
1	Airlines	122270000000	-0.0182	10.97	6.83	...	70720000	0.47	2.22	2.31	43.65
2	REIT - Mortgage	53670000000	-0.0068	15.94	6.98	...	40160000	1.69	0.81	3.60	4.73
3	Insurance - Reinsurance	53050000000	-0.0063	7.66	7.53	...	2049999	0.90	1.21	NaN	3.03
4	Insurance - Life	269430000000	-0.0107	12.25	8.11	...	16079999	1.10	1.48	4120.41	4.41
...	...	...	...	...	...	...	...	...	...	...	...
140	REIT - Healthcare Facilities	123940000000	-0.0028	94.22	46.31	...	27280000	5.51	1.68	25.46	22.11
141	Uranium	36590000000	0.0248	112.12	46.58	...	36490000	15.51	4.90	28.10	143.86
142	Shell Companies	26350000000	0.0185	44.08	133.74	...	10560000	46.71	2.10	36.15	105.00
143	Infrastructure Operations	34110000000	-0.0041	57.81	146.15	...	1160000	3.40	7.22	6.25	28.37
144	Real Estate - Diversified	6760000000	-0.0141	75.91	NaN	...	549690	4.54	1.74	6.63	58.78

Figure 1.3: Results of a comparison screener between F, GE, and TSLA

2. Create a screener that returns the top gainers from the technology sector based on a preset:

```
obb.equity.compare.groups(
    group="technology",
    metric="performance",
    provider="finviz"
)
```

The output of the preceding snippet is the following pandas DataFrame:



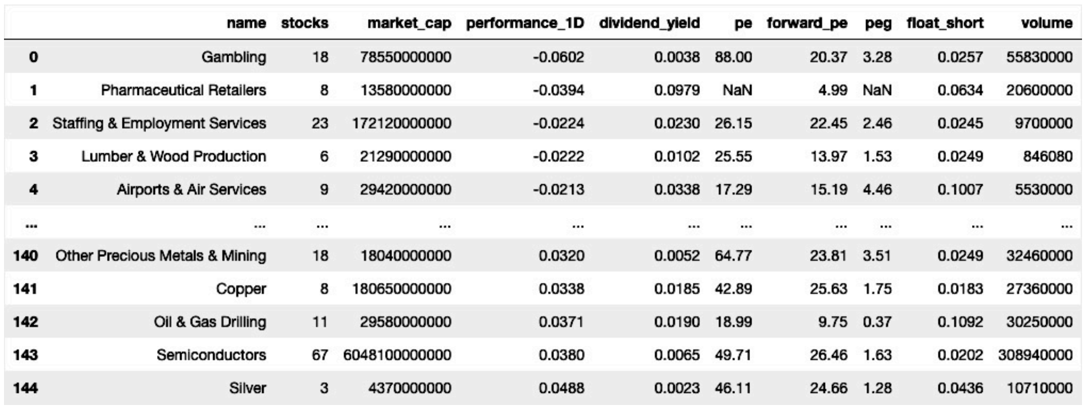
	name	performance_1D	performance_1W	performance_1M	performance_3M	...	performance_YTD	analyst_recommendation	volume	volume_average	volume_relative
0	Pharmaceutical Retailers	-0.0394	-0.1342	-0.1270	-0.2850	...	-0.4065	3.05	20600000	13170000	1.56
1	Gambling	-0.0602	-0.0940	-0.0237	-0.0917	...	0.0263	1.49	55830000	18370000	3.04
2	Tools & Accessories	-0.0169	-0.0693	-0.0322	-0.0660	...	-0.0766	2.59	5910000	5070000	1.16
3	Airports & Air Services	-0.0213	-0.0680	-0.0299	0.0950	...	-0.0233	2.17	5530000	7110000	0.78
4	Utilities - Regulated Water	-0.0171	-0.0601	0.0082	0.0268	...	-0.0489	1.91	5450000	5980000	0.91
...	...	...	...	...	...	...	...	...	...	...	...
140	Department Stores	0.0098	0.0397	0.0640	0.0028	...	0.0385	3.24	12020000	16200000	0.74
141	Utilities - Renewable	-0.0010	0.0584	0.1817	0.3068	...	0.3735	1.79	49110000	26700000	1.84
142	Utilities - Independent Power Producers	0.0190	0.1012	0.3138	0.7382	...	0.9717	1.73	12450000	13270000	0.94
143	Semiconductors	0.0380	0.1032	0.1944	0.2518	...	0.5913	1.54	308940000	320980000	0.96
144	Solar	0.0119	0.2505	0.2811	0.2057	...	0.0411	1.85	799280000	155900000	5.13

Figure 1.4: Results of a screener showing the day's top-gaining stocks

3. Create a screener that presents an overview grouped by sector:

```
obb.equity.compare.groups(
    group="sector",
    metric="overview",
    provider="finviz"
)
```

The output of the preceding snippet is the following pandas DataFrame:



	name	stocks	market_cap	performance_1D	dividend_yield	pe	forward_pe	peg	float_short	volume
0	Gambling	18	78550000000	-0.0602	0.0038	88.00	20.37	3.28	0.0257	55830000
1	Pharmaceutical Retailers	8	13580000000	-0.0394	0.0979	NaN	4.99	NaN	0.0634	20600000
2	Staffing & Employment Services	23	172120000000	-0.0224	0.0230	26.15	22.45	2.46	0.0245	9700000
3	Lumber & Wood Production	6	21290000000	-0.0222	0.0102	25.55	13.97	1.53	0.0249	846080
4	Airports & Air Services	9	29420000000	-0.0213	0.0338	17.29	15.19	4.46	0.1007	5530000
...	...	...	...	...	...	...	...	...	...	...
140	Other Precious Metals & Mining	18	18040000000	0.0320	0.0052	64.77	23.81	3.51	0.0249	32460000
141	Copper	8	180650000000	0.0338	0.0185	42.89	25.63	1.75	0.0183	27360000
142	Oil & Gas Drilling	11	29580000000	0.0371	0.0190	18.99	9.75	0.37	0.1092	30250000
143	Semiconductors	67	6048100000000	0.0380	0.0065	49.71	26.46	1.63	0.0202	308940000
144	Silver	3	4370000000	0.0488	0.0023	46.11	24.66	1.28	0.0436	10710000

Figure 1.5: Results of a screener grouped by sector

See also

For more on OpenBB and the **Finviz** stock screener, check out the following resources:

- The OpenBB Platform documentation, which contains details on dozens of functions to acquire stock market data for free: **https://docs.openbb.co/platform/reference/equity**
- The **Finviz** home page, which includes free access to the web-based screener **https://finviz.com/?a=2548677**
- The top-selling cohort-based course to help beginners get up and running with Python for quant finance, algorithmic trading, and data analysis: **https://www.pyquantnews.com/getting-started-with-python-for-quant-finance**

Fetching historic futures data with the OpenBB Platform

Traders use continuous futures data for backtesting trading strategies. Futures traders use the roll from one contract to another as a potential opportunity for profit. Some traders simply pick a date before expiration to roll to the next contract, while others use sophisticated techniques involving open interest. This **basis trade** is persistently one of the most popular trading strategies for futures traders. These traders want control over the data that's used to compute the basis trade, so acquiring individual contract data is important. This recipe will guide you through the process of using the OpenBB Platform to fetch individual futures contract data.

Getting ready…

By now, you should have the OpenBB Platform installed in your virtual environment. If not, go back to the beginning of this chapter and get it set up.

How to do it…

We'll use the futures functionality in the OpenBB Platform to download individual futures data for free:

1. Import pandas and the OpenBB Platform:

```
import pandas as pd
from openbb import obb
obb.user.preferences.output_type = "dataframe"
```

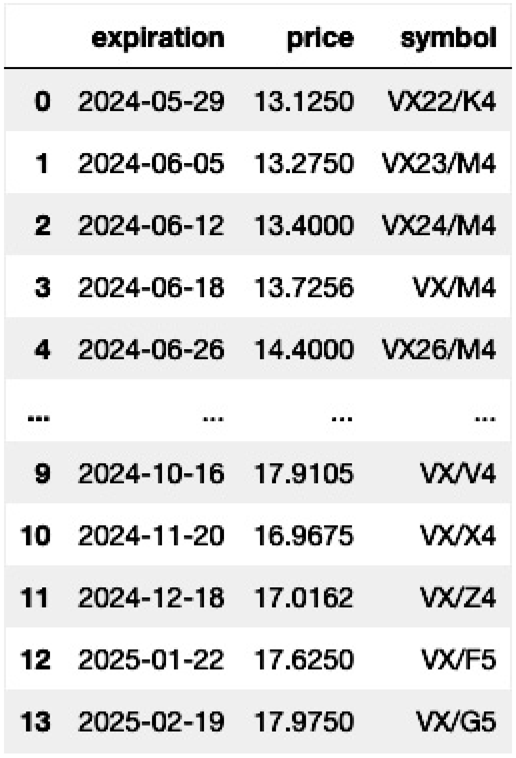
2. Download the current futures curve for the VIX futures contract from the **Chicago Board Options Exchange** (**CBOE**):

```
data = obb.derivatives.futures.curve(symbol="VX")
```

3. Inspect the resulting DataFrame:

```
print(data)
```

Running the preceding code generates the futures curve for the VIX futures contract:



```
     expiration     price     symbol
0    2024-05-29   13.1250    VX22/K4
1    2024-06-05   13.2750    VX23/M4
2    2024-06-12   13.4000    VX24/M4
3    2024-06-18   13.7256      VX/M4
4    2024-06-26   14.4000    VX26/M4
...         ...       ...        ...
9    2024-10-16   17.9105      VX/V4
10   2024-11-20   16.9675      VX/X4
11   2024-12-18   17.0162      VX/Z4
12   2025-01-22   17.6250      VX/F5
13   2025-02-19   17.9750      VX/G5
```

Figure 1.6: Settlement prices for the forward Eurodollar futures contracts

4. Update the DataFrame index to the expiration dates and plot the settlement prices:

```
data.index = pd.to_datetime(data.expiration)
data.plot()
```

By running the proceeding code, we plot the VIX futures curve:

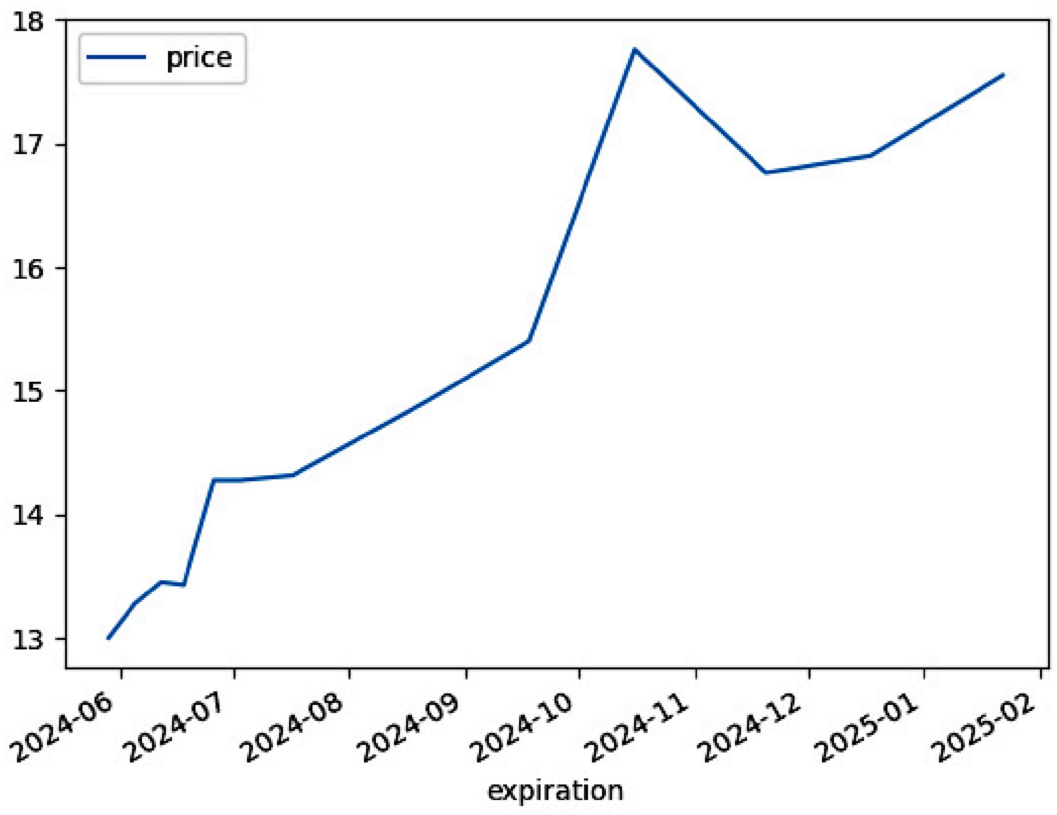


Figure 1.7: VIX futures curve

There's more…

You can use the **`obb.derivatives.futures.historical`** method to get historical data for an individual expiration. Stitching together data across a range of years can provide insight into the market's expectation of supply and demand of the underlying commodity:

1. First, create a list containing the year and month expirations you're interested in:

```
expirations = [
    "2024-12",
    "2025-12",
    "2026-12",
    "2027-12",
    "2028-12",
    "2029-12",
    "2030-12",
]
```

2. The preceding code creates a Python list of expiration years and dates in string format. Now, loop through each of the expirations to download the data:

```
contracts = []
for expiration in expirations:
    df = (
        obb
        .derivatives
        .futures
        .historical(
            symbol="CL",
            expiration=expiration,
            start_date="2020-01-01",
            end_date="2022-12-31"
        )
    ).rename(columns={
        "close": expiration
    })
    contracts.append(df[expiration])
```

3. For each of the contracts, use the OpenBB Platform to download historical futures data for the CL contract between January 1, 2020, and 31 December 31, 2022. Using the pandas **`rename`** method, change the column name from **`"close"`** to the expiration date. Finally, append the newly created pandas DataFrame to a list of DataFrames:

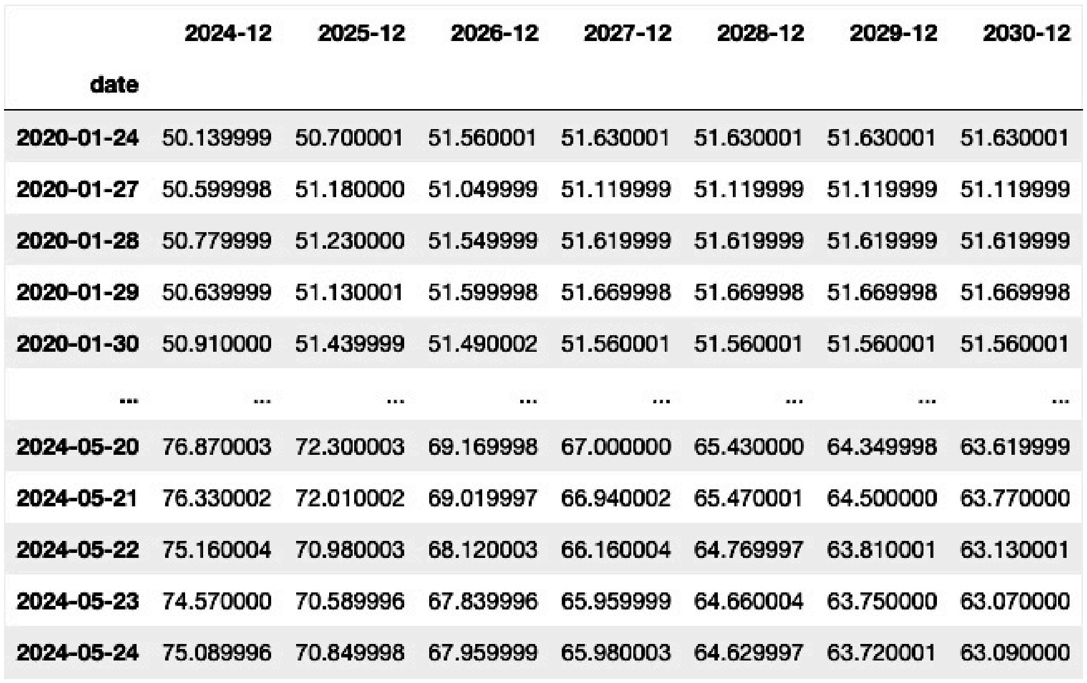
```
historical = (
    pd
    .DataFrame(contracts)
    .transpose()
```

```
      .dropna()
  )
```

4. Concatenate the DataFrames together, swap the columns and rows using the **`transpose`** method, and drop any records with no data using the **`dropna`** method. Inspect the resulting DataFrame:

   ```
   print(historical)
   ```

   By printing the DataFrame, we will see the historical settlement prices:



	2024-12	2025-12	2026-12	2027-12	2028-12	2029-12	2030-12
date							
2020-01-24	50.139999	50.700001	51.560001	51.630001	51.630001	51.630001	51.630001
2020-01-27	50.599998	51.180000	51.049999	51.119999	51.119999	51.119999	51.119999
2020-01-28	50.779999	51.230000	51.549999	51.619999	51.619999	51.619999	51.619999
2020-01-29	50.639999	51.130001	51.599998	51.669998	51.669998	51.669998	51.669998
2020-01-30	50.910000	51.439999	51.490002	51.560001	51.560001	51.560001	51.560001
...	...	...	...	...	...	...	...
2024-05-20	76.870003	72.300003	69.169998	67.000000	65.430000	64.349998	63.619999
2024-05-21	76.330002	72.010002	69.019997	66.940002	65.470001	64.500000	63.770000
2024-05-22	75.160004	70.980003	68.120003	66.160004	64.769997	63.810001	63.130001
2024-05-23	74.570000	70.589996	67.839996	65.959999	64.660004	63.750000	63.070000
2024-05-24	75.089996	70.849998	67.959999	65.980003	64.629997	63.720001	63.090000

Figure 1.8: Historic settlement prices for the December CL futures contract

The result is the historical data between January 2020 and December 2022 for each of the December expirations between 2023 and 2030:

5. To visualize the market's expectation of the future supply and demand of the December contract, you can plot the last price:

   ```
   historical.iloc[-1].plot()
   ```

   Here's the output:

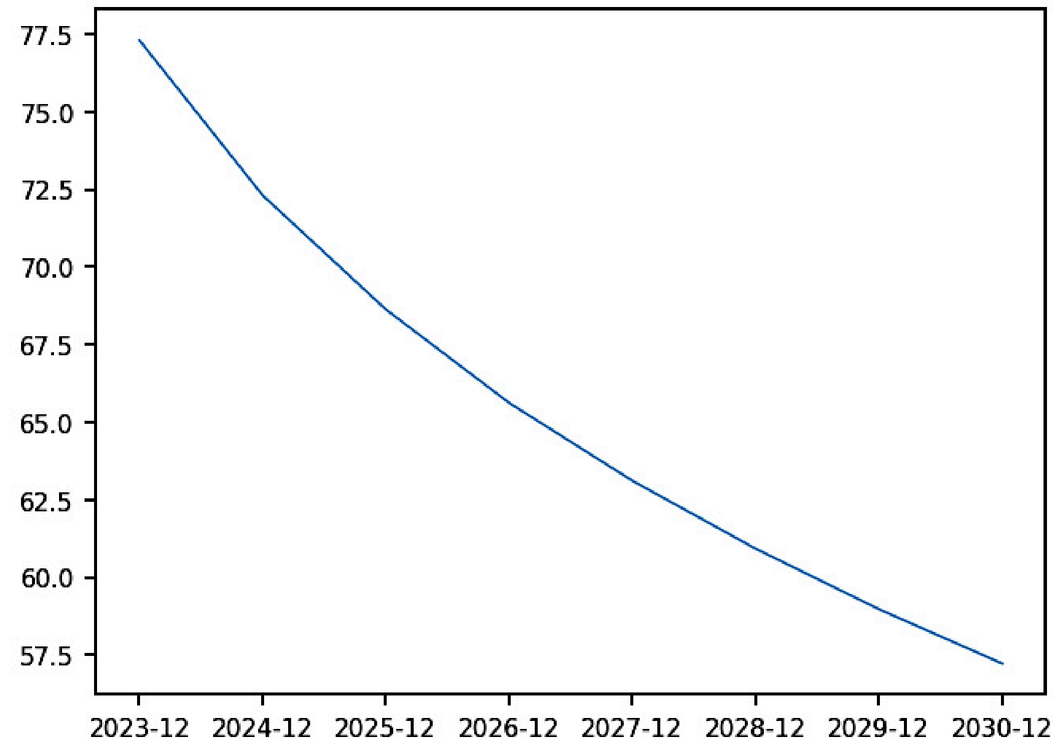


Figure 1.9: Futures curve for the December CL contract

See also

For more on the OpenBB Platform futures functionality, you can browse the following documentation:

- Documentation on the OpenBB Platform's futures **curve** method: https://docs.openbb.co/platform/reference/derivatives/futures/curve
- Documentation on the OpenBB Platform's historical **curve** method: https://docs.openbb.co/platform/reference/derivatives/futures/historical

Navigating options market data with the OpenBB Platform

Options are exchange-listed derivative contracts that convey the right (but not the obligation) to buy or sell the underlying stock at a certain price on or before a certain expiration date. Options are among the most versatile financial instruments in the market. They allow traders to define their risk profiles before entering trades and express market views not only on the direction of the underlying but the volatility. While options offer a high degree of flexibility for trading, this feature complicates data collection for research and backtesting.

A single underlying stock can have an array of options contracts with different combinations of strike prices and expiration dates. Collecting and manipulating this data is a challenge. The combination of options contracts for all strikes and expiration dates is commonly referred to as an options chain. There can be thousands of individual options contracts at a given time for a single underlying stock. Not only does the number of individual contracts pose a challenge, but getting price data has historically been expensive. With the introduction of the OpenBB Platform, it is now only a few lines of Python code to download options chains into a pandas DataFrame. This recipe will walk you through acquiring options data using the OpenBB Platform.

Getting ready…

By now, you should have the OpenBB Platform installed in your virtual environment. If not, go back to the beginning of this chapter and get it set up.

How to do it…

Similar to how we used the OpenBB Platform for futures data, we can use it for options data too:

1. Import the OpenBB Platform and Matplotlib for visualization:

```
from openbb import obb
obb.user.preferences.output_type = "dataframe"
```

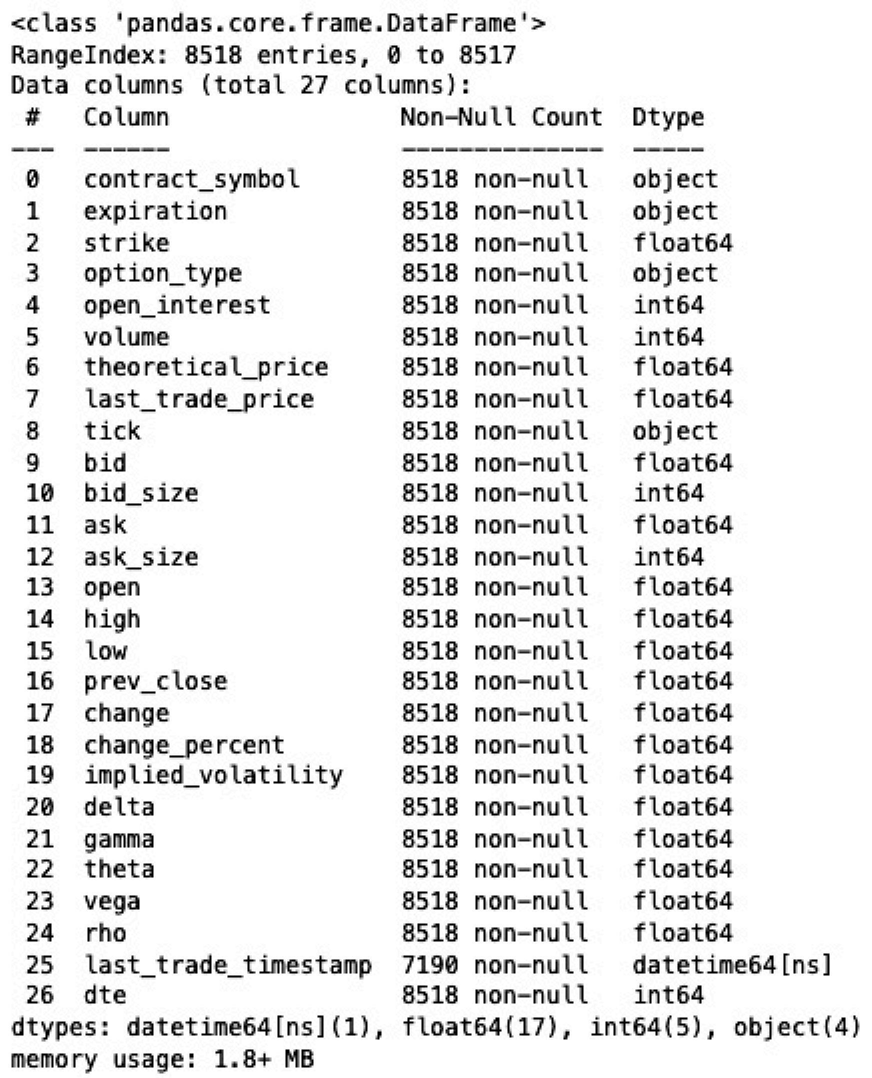
2. Use the **`chains`** method to download the entire options chain:

```
chains = obb.derivatives.options.chains(symbol="SPY")
```

3. Inspect the resulting DataFrame:

```
chains.info()
```

By running the preceding code, we'll see the details of the options chains data:



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8518 entries, 0 to 8517
Data columns (total 27 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   contract_symbol      8518 non-null   object
 1   expiration           8518 non-null   object
 2   strike               8518 non-null   float64
 3   option_type          8518 non-null   object
 4   open_interest        8518 non-null   int64
 5   volume               8518 non-null   int64
 6   theoretical_price    8518 non-null   float64
 7   last_trade_price     8518 non-null   float64
 8   tick                 8518 non-null   object
 9   bid                  8518 non-null   float64
 10  bid_size             8518 non-null   int64
 11  ask                  8518 non-null   float64
 12  ask_size             8518 non-null   int64
 13  open                 8518 non-null   float64
 14  high                 8518 non-null   float64
 15  low                  8518 non-null   float64
 16  prev_close           8518 non-null   float64
 17  change               8518 non-null   float64
 18  change_percent       8518 non-null   float64
 19  implied_volatility   8518 non-null   float64
 20  delta                8518 non-null   float64
 21  gamma                8518 non-null   float64
 22  theta                8518 non-null   float64
 23  vega                 8518 non-null   float64
 24  rho                  8518 non-null   float64
 25  last_trade_timestamp 7190 non-null   datetime64[ns]
 26  dte                  8518 non-null   int64
dtypes: datetime64[ns](1), float64(17), int64(5), object(4)
memory usage: 1.8+ MB
```

Figure 1.10: Preview of the data downloaded for the SPY options chains

Note that there are 8,518 options contracts for the **SPY Exchange Traded Fund** (**ETF**) that can be downloaded from CBOE (for free).

How it works...

The **`obb.derivatives.options.chains`** method downloads the entire options chain and stores it in a pandas DataFrame. The **`obb.derivatives.options.chains`** has an additional optional parameter:

- **`provider`**: The source from which the data should be downloaded. The default is CBOE. You can also select Tradier, Intrinio, or TMX. Note that for Tradier, Intrinio, and TMX, you need to provide your API key, which can be configured in the OpenBB Hub.

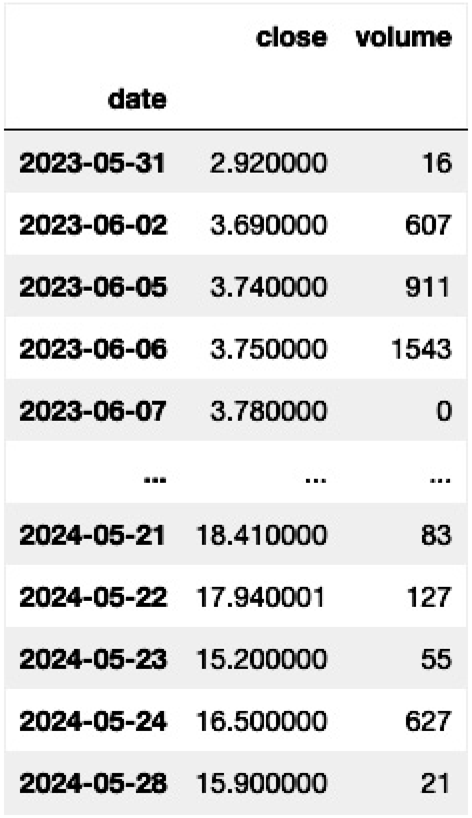
There's more...

You can use the OpenBB Platform to download historical options data for a single contract. To do this, you need the option symbol.

We'll use the **`obb.equity.price.historical`** method to get the historical options data for an SPY call option with a strike price of $550 expiring on December 20, 2024:

```
data = obb.equity.price.historical(
    symbol="SPY241220C00550000",
    provider="yfinance"
)[["close", "volume"]]
```

The result is a pandas DataFrame with the closing price and volume of the options contract.



	close	volume
date		
2023-05-31	2.920000	16
2023-06-02	3.690000	607
2023-06-05	3.740000	911
2023-06-06	3.750000	1543
2023-06-07	3.780000	0
...	...	...
2024-05-21	18.410000	83
2024-05-22	17.940001	127
2024-05-23	15.200000	55
2024-05-24	16.500000	627
2024-05-28	15.900000	21

Figure 1.11: Closing prices and volume of the SPY options contract

Options Greeks

Options Greeks measure how options prices change given a change in one of the inputs to an options pricing model. For example, **delta** measures how an options price changes given a change in the underlying stock price.

Using `obb.derivatives.options.chains`, the OpenBB Platform returns the most used Greeks including Delta, Gamma, Theta, Vega, and Rho.

See also

Options are a fascinating and deep topic that is rich with opportunities for trading. You can learn more about options, volatility, and how to analyze both via the OpenBB Platform:

- Free articles, code, and other resources for options trading: **https://pyquantnews.com/free-python-resources/options-trading-with-python/**
- Learn more about financial options if you're unfamiliar with them: **https://en.wikipedia.org/wiki/Option_(finance)**
- Documentation on the OpenBB Platform's options methods: **https://docs.openbb.co/platform/reference/derivatives/options**

Harnessing factor data using pandas_datareader

Diversification is great until the entire market declines in value. That's because the overall market influences all assets. Factors can offset some of these risks by targeting drivers of return not influenced by the market. Common factors are size (large-cap versus small-cap) and style (value versus growth). If you think small-cap stocks will outperform large-cap stocks, then you might want exposure to small-cap stocks. If you think value stocks will outperform growth stocks, then you might want exposure to value stocks. In either case, you want to measure the risk contribution of the factor. Eugene Fama and Kenneth French built the Fama-French three-factor model in 1992. The three Fama-French factors are constructed using six value-weight portfolios formed on capitalization and book-to-market.

The three factors are as follows:

- **Small Minus Big**, which represents the differential between the average returns of three small-cap portfolios and three large-cap portfolios.
- **High Minus Low**, which quantifies the difference in average returns between two value-oriented portfolios and two growth-oriented portfolios.
- **Rm-Rf**, which denotes the market's excess return over the risk-free rate.

We'll explore how to measure and isolate alpha in **Chapter 5**, *Build Alpha Factors for Stock Portfolios*. This recipe will guide you through the process of using **pandas_datareader** to fetch historic factor data for use in your analysis.

Getting ready…

By now, you should have the OpenBB Platform installed in your virtual environment. If not, go back to the beginning of this chapter and get it set up. By installing the OpenBB Platform, **pandas_datareader** will be installed and ready to use.

How to do it…

Using the **pandas_datareader** library, we have access to dozens of investment research factors:

1. Import **pandas_datareader**:

```
import pandas_datareader as pdr
```

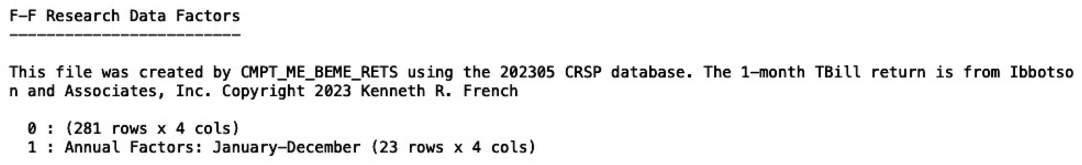
2. Download the monthly factor data starting in January 2000:

```
factors = pdr.get_data_famafrench("F-F_Research_Data_Factors")
```

3. Get a description of the research data factors:

```
print(factors[“DESCR”])
```

The result is an explanation of the data included in the DataFrame:



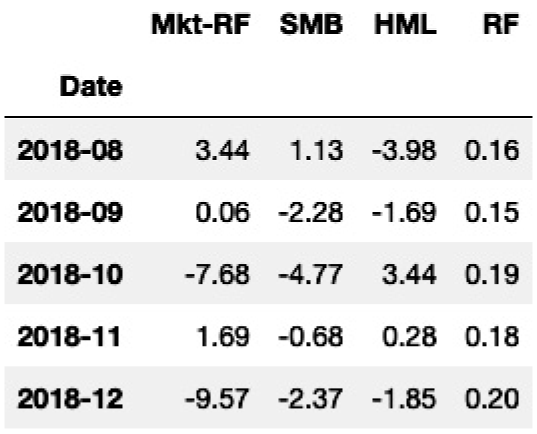
```
F-F Research Data Factors
-------------------------

This file was created by CMPT_ME_BEME_RETS using the 202305 CRSP database. The 1-month TBill return is from Ibbotso
n and Associates, Inc. Copyright 2023 Kenneth R. French

  0 : (281 rows x 4 cols)
  1 : Annual Factors: January-December (23 rows x 4 cols)
```

Figure 1.12: Preview of the description that is downloaded with factor data

4. Inspect the monthly factor data:

```
print(factors[0].head())
```

By running the preceding code, we get a DataFrame containing monthly factor data:



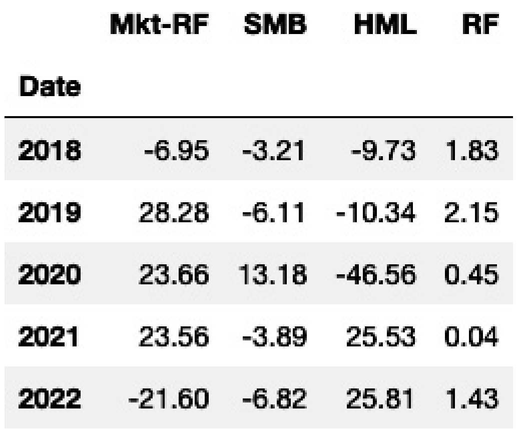
	Mkt-RF	SMB	HML	RF
Date				
2018-08	3.44	1.13	-3.98	0.16
2018-09	0.06	-2.28	-1.69	0.15
2018-10	-7.68	-4.77	3.44	0.19
2018-11	1.69	-0.68	0.28	0.18
2018-12	-9.57	-2.37	-1.85	0.20

Figure 1.13: Preview of the monthly data downloaded from the Fama-French Data Library

5. Inspect the annual factor data:

```
print(factors[1].head())
```

By running the preceding code, we get a DataFrame containing annual factor data:



	Mkt-RF	SMB	HML	RF
Date				
2018	-6.95	-3.21	-9.73	1.83
2019	28.28	-6.11	-10.34	2.15
2020	23.66	13.18	-46.56	0.45
2021	23.56	-3.89	25.53	0.04
2022	-21.60	-6.82	25.81	1.43

Figure 1.14: Preview of the annual data downloaded from the Fama-French Data Library

How it works…

Under the hood, **`pandas_datareader`** fetches data from the Fama-French Data Library by downloading a compressed CSV file, uncompressing it, and creating a pandas DataFrame.

There are 297 different datasets with different factor data available from the Fama-French Data Library. Here are some popular versions of the Fama-French 3-factor model for different regions:

- **`Developed_3_Factors`**
- **`Developed_ex_US_3_Factors`**
- **`Europe_3_Factors`**
- **`Japan_3_Factors`**
- **`Asia_Pacific_ex_Japan_3_Factors`**

You can use these in the **`get_data_famafrench`** method, just like **`F-F_Research_Data_Factors`**.

Some datasets return a dictionary with more than one DataFrame representing data for different time frames, portfolio weighting methodologies, and aggregate statistics. Data for these portfolios can be accessed using numerical keys. For example, the **`5_Industry_Portfolios`** dataset returns eight DataFrames in the dictionary. The first can be accessed using the **`0`** key, the second using the **`1`** key, and so on. Each dictionary includes a description of the dataset, which can be accessed using the **`DESCR`** key.

There's more…

`pandas_datareader` can be used to access data from many remote online sources. These include Tiingo, IEX, Alpha Vantage, FRED, Eurostat, and many more. Review the full list of data sources on the documentation page: **https://pandas-datareader.readthedocs.io/en/latest/remote_data.html**.

See also

For more details on the factors available in the investment factor research library, take a look at the following resources. For another example of using the Fama-French 3-factor model, see the resources on the PyQuant News website:

- Documentation for all the Fama-French factor data: **https://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html**
- Details on the Fama-French 3-factor model: **https://mba.tuck.dartmouth.edu/pages/faculty/ken.french/Data_Library/f-f_factors.html**
- Code walkthrough for using the Fama-French 3-factor model: **https://www.pyquantnews.com/past-pyquant-newsletter-issues**