9

Assess Backtest Risk and Performance Metrics with Pyfolio

No single risk or performance metric tells the entire story of how a strategy might perform in live trading. Metrics such as the Sharpe ratio, for instance, focus mainly on returns relative to volatility but neglect other risks such as drawdown or tail risk. Similarly, using only maximum drawdown as a measure ignores the risk-adjusted returns and might discard strategies that are robust but temporarily underperforming. The composite view obtained through multiple metrics provides a more nuanced understanding of how the strategy is likely to behave under varying market conditions. Taking it a step further, visualizing risk and performance metrics over time can capture strategy dynamics over time. A strategy might exhibit robust metrics during a bull market but underperform in terms of risk-adjusted returns during a bear or sideways market.

In this chapter, we introduce **Pyfolio Reloaded** (**Pyfolio**), which is a risk and performance analysis library. Pyfolio Reloaded is part of the Zipline Reloaded ecosystem and takes the output of a Zipline Reloaded backtest to build a robust suite of risk and performance metrics. We will walk through the process of using Pyfolio Reloaded to generate risk and performance metrics. Throughout the recipes in this chapter, we'll define the most important metrics to consider when assessing the performance of a backtest.

In this chapter, we present the following recipes:

- Preparing Zipline Reloaded backtest results for Pyfolio Reloaded

- Generating strategy performance and return analytics
- Building a drawdown and rolling risk analysis
- Analyzing strategy holdings, leverage, exposure, and sector allocations
- Breaking down strategy performance to trade level

Preparing Zipline backtest results for Pyfolio Reloaded

In ***Chapter 7***, *Event-Based Backtesting Factor Portfolios with Zipline Reloaded*, we learned how to use Zipline Reloaded to backtest a factor strategy. The output of a Zipline Reloaded backtest includes a DataFrame that details various metrics calculated over the backtest period, such as returns, alpha, beta, the Sharpe ratio, and drawdowns. It also provides transaction logs that capture executed orders, including asset, price, and quantity, giving insights into the trading behavior of the strategy. Additionally, Zipline Reloaded outputs an asset-wise breakdown of the portfolio, detailing the holdings and their respective values, which can be vital for risk assessment and position sizing in the portfolio.

Before we can use the DataFrame, there is some required data preprocessing. Helpfully, Pyfolio Reloaded comes with helper functions that do most of the work for us. In this recipe, we'll read in the DataFrame and prepare it to use with Pyfolio. We'll also build a symbol-to-sector mapping and acquire data to represent a benchmark.

Getting ready…

We assume you ran the backtest described in ***Chapter 7***, *Event-Based Backtesting Factor Portfolios with Zipline Reloaded,* and have a file called **`mean_reversion.pickle`** in the current working directory. The pickle file is a serialized version of the DataFrame Zipline Reloaded generated describing the performance results.

How to do it…

We'll use pandas to read the pickle file and the OpenBB Platform to acquire sector data and benchmark prices.

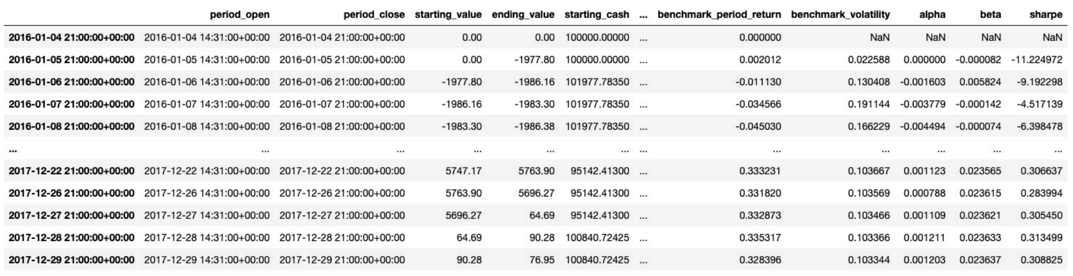
1. Import the libraries:

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

2. Read in the pickle file using the pandas **`read_pickle`** method:

```
perf = pd.read_pickle("mean_reversion.pickle")
```

The result is the performance DataFrame from the Zipline backtest.



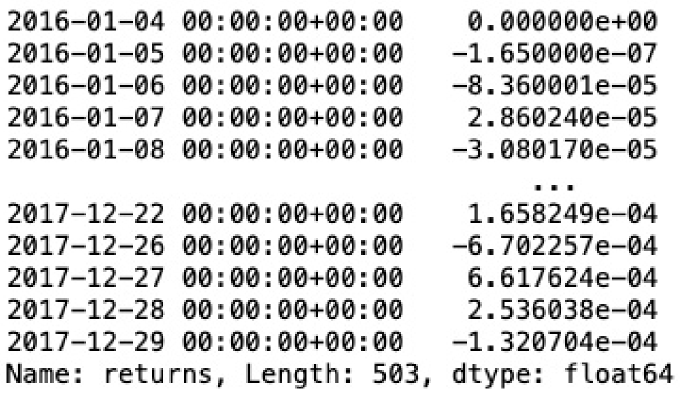
	period_open	period_close	starting_value	ending_value	starting_cash	...	benchmark_period_return	benchmark_volatility	alpha	beta	sharpe
2016-01-04 21:00:00+00:00	2016-01-04 14:31:00+00:00	2016-01-04 21:00:00+00:00	0.00	0.00	100000.00000	...	0.000000	NaN	NaN	NaN	NaN
2016-01-05 21:00:00+00:00	2016-01-05 14:31:00+00:00	2016-01-05 21:00:00+00:00	0.00	-1977.80	100000.00000	...	0.002012	0.022588	0.000000	-0.000082	-11.224972
2016-01-06 21:00:00+00:00	2016-01-06 14:31:00+00:00	2016-01-06 21:00:00+00:00	-1977.80	-1986.16	101977.78350	...	-0.011130	0.130408	-0.001603	0.005824	-9.192298
2016-01-07 21:00:00+00:00	2016-01-07 14:31:00+00:00	2016-01-07 21:00:00+00:00	-1986.16	-1983.30	101977.78350	...	-0.034566	0.191144	-0.003779	-0.000142	-4.517139
2016-01-08 21:00:00+00:00	2016-01-08 14:31:00+00:00	2016-01-08 21:00:00+00:00	-1983.30	-1986.38	101977.78350	...	-0.045030	0.166229	-0.004494	-0.000074	-6.398478
...	...	...	...	...	...	...	...	...	...	...	...
2017-12-22 21:00:00+00:00	2017-12-22 14:31:00+00:00	2017-12-22 21:00:00+00:00	5747.17	5763.90	95142.41300	...	0.333231	0.103667	0.001123	0.023565	0.306637
2017-12-26 21:00:00+00:00	2017-12-26 14:31:00+00:00	2017-12-26 21:00:00+00:00	5763.90	5696.27	95142.41300	...	0.331820	0.103569	0.000788	0.023615	0.283994
2017-12-27 21:00:00+00:00	2017-12-27 14:31:00+00:00	2017-12-27 21:00:00+00:00	5696.27	64.69	95142.41300	...	0.332873	0.103466	0.001109	0.023621	0.305450
2017-12-28 21:00:00+00:00	2017-12-28 14:31:00+00:00	2017-12-28 21:00:00+00:00	64.69	90.28	100840.72425	...	0.335317	0.103366	0.001211	0.023633	0.313499
2017-12-29 21:00:00+00:00	2017-12-29 14:31:00+00:00	2017-12-29 21:00:00+00:00	90.28	76.95	100840.72425	...	0.328396	0.103344	0.001203	0.023637	0.308825

Figure 9.1: Deserialized DataFrame containing the backtest performance metrics

3. Use the Pyfolio helper function to extract returns, positions, and transactions from the DataFrame:

```
returns, positions, transactions = \
    pf.utils.extract_rets_pos_txn_from_zipline(perf)
```

The result is a pandas Series with strategy returns.



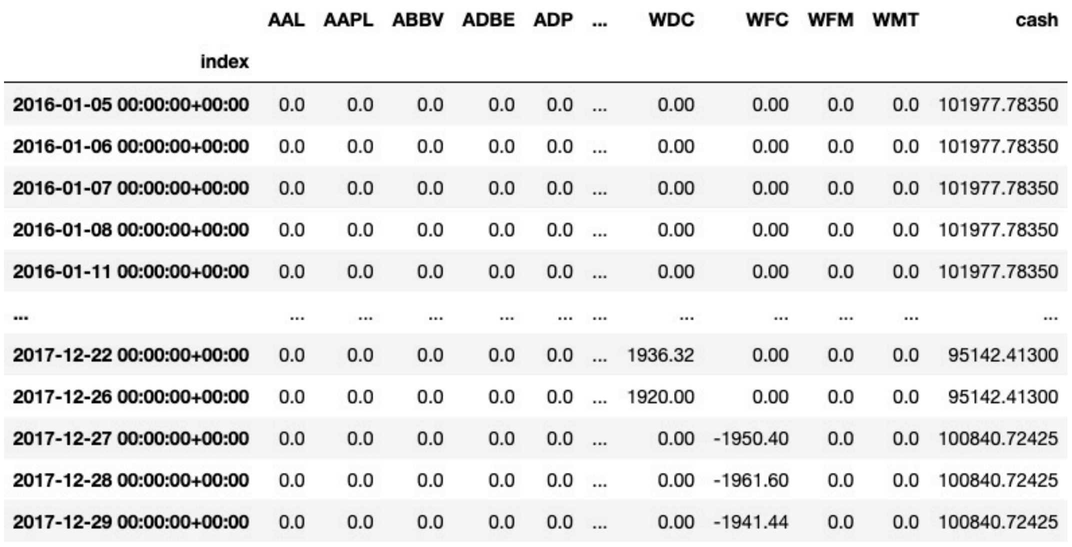
```
2016-01-04 00:00:00+00:00    0.000000e+00
2016-01-05 00:00:00+00:00   -1.650000e-07
2016-01-06 00:00:00+00:00   -8.360001e-05
2016-01-07 00:00:00+00:00    2.860240e-05
2016-01-08 00:00:00+00:00   -3.080170e-05
                                  ...
2017-12-22 00:00:00+00:00    1.658249e-04
2017-12-26 00:00:00+00:00   -6.702257e-04
2017-12-27 00:00:00+00:00    6.617624e-04
2017-12-28 00:00:00+00:00    2.536038e-04
2017-12-29 00:00:00+00:00   -1.320704e-04
Name: returns, Length: 503, dtype: float64
```

Figure 9.2: pandas Series with daily strategy returns

4. The **`positions`** DataFrames contain the Zipline **`Equity`** objects as column labels. Replace the object with the string representations:

```
positions.columns = [
    col.symbol for col in positions.columns[
        :-1]] + ["cash"]
```

The result is DataFrame with the daily positions, including cash.



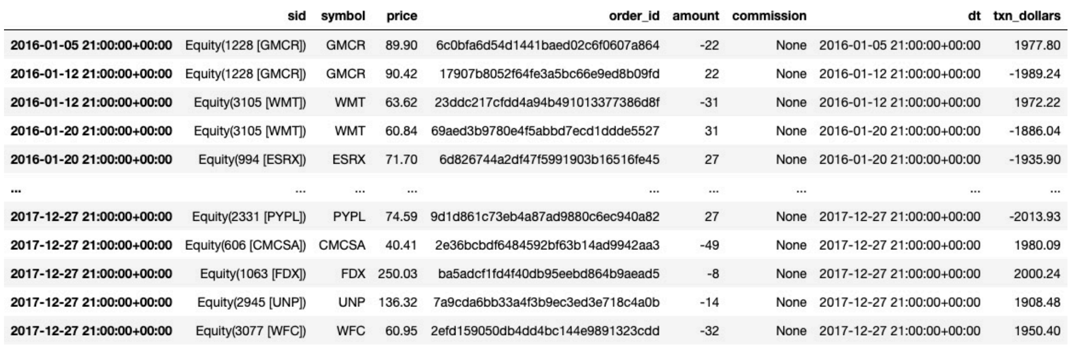
	AAL	AAPL	ABBV	ADBE	ADP	...	WDC	WFC	WFM	WMT	cash
index											
2016-01-05 00:00:00+00:00	0.0	0.0	0.0	0.0	0.0	...	0.00	0.00	0.0	0.0	101977.78350
2016-01-06 00:00:00+00:00	0.0	0.0	0.0	0.0	0.0	...	0.00	0.00	0.0	0.0	101977.78350
2016-01-07 00:00:00+00:00	0.0	0.0	0.0	0.0	0.0	...	0.00	0.00	0.0	0.0	101977.78350
2016-01-08 00:00:00+00:00	0.0	0.0	0.0	0.0	0.0	...	0.00	0.00	0.0	0.0	101977.78350
2016-01-11 00:00:00+00:00	0.0	0.0	0.0	0.0	0.0	...	0.00	0.00	0.0	0.0	101977.78350
...	...	...	...	...	...	...	...	...	...	...	...
2017-12-22 00:00:00+00:00	0.0	0.0	0.0	0.0	0.0	...	1936.32	0.00	0.0	0.0	95142.41300
2017-12-26 00:00:00+00:00	0.0	0.0	0.0	0.0	0.0	...	1920.00	0.00	0.0	0.0	95142.41300
2017-12-27 00:00:00+00:00	0.0	0.0	0.0	0.0	0.0	...	0.00	-1950.40	0.0	0.0	100840.72425
2017-12-28 00:00:00+00:00	0.0	0.0	0.0	0.0	0.0	...	0.00	-1961.60	0.0	0.0	100840.72425
2017-12-29 00:00:00+00:00	0.0	0.0	0.0	0.0	0.0	...	0.00	-1941.44	0.0	0.0	100840.72425

Figure 9.3: DataFrame with the position value each day

5. The **`symbol`** column in the **`transactions`** DataFrame also contains the Zipline **`Equity`** objects. Replace the object with the string representations:

```
transactions.symbol = transactions.symbol.apply(
    lambda s: s.symbol)
```

The result is a DataFrame with the daily transactions:



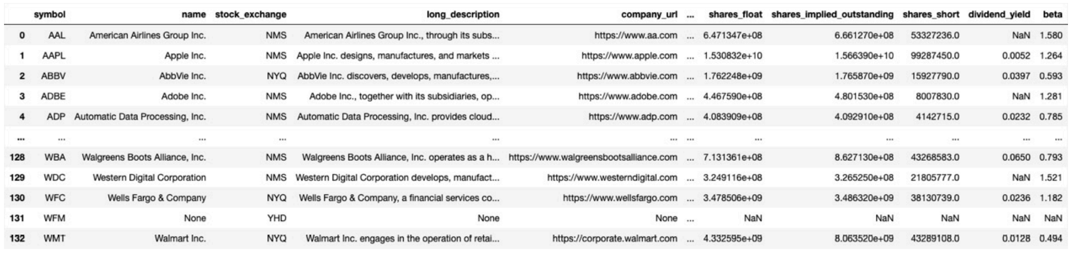
	sid	symbol	price	order_id	amount	commission	dt	txn_dollars
2016-01-05 21:00:00+00:00	Equity(1228 [GMCR])	GMCR	89.90	6c0bfa6d54d1441baed02c6f0607a864	-22	None	2016-01-05 21:00:00+00:00	1977.80
2016-01-12 21:00:00+00:00	Equity(1228 [GMCR])	GMCR	90.42	17907b8052f64fe3a5bc66e9ed8b09fd	22	None	2016-01-12 21:00:00+00:00	-1989.24
2016-01-12 21:00:00+00:00	Equity(3105 [WMT])	WMT	63.62	23ddc217cfdd4a94b491013377386d8f	-31	None	2016-01-12 21:00:00+00:00	1972.22
2016-01-20 21:00:00+00:00	Equity(3105 [WMT])	WMT	60.84	69aed3b9780e4f5abbd7ecd1ddde5527	31	None	2016-01-20 21:00:00+00:00	-1886.04
2016-01-20 21:00:00+00:00	Equity(994 [ESRX])	ESRX	71.70	6d826744a2df47f5991903b16516fe45	27	None	2016-01-20 21:00:00+00:00	-1935.90
...	...	...	...	...	...	...	...	...
2017-12-27 21:00:00+00:00	Equity(2331 [PYPL])	PYPL	74.59	9d1d861c73eb4a87ad9880c6ec940a82	27	None	2017-12-27 21:00:00+00:00	-2013.93
2017-12-27 21:00:00+00:00	Equity(606 [CMCSA])	CMCSA	40.41	2e36bcbdf6484592bf63b14ad9942aa3	-49	None	2017-12-27 21:00:00+00:00	1980.09
2017-12-27 21:00:00+00:00	Equity(1063 [FDX])	FDX	250.03	ba5adcf1fd4f40db95eebd864b9aead5	-8	None	2017-12-27 21:00:00+00:00	2000.24
2017-12-27 21:00:00+00:00	Equity(2945 [UNP])	UNP	136.32	7a9cda6bb33a4f3b9ec3ed3e718c4a0b	-14	None	2017-12-27 21:00:00+00:00	1908.48
2017-12-27 21:00:00+00:00	Equity(3077 [WFC])	WFC	60.95	2efd159050db4dd4bc144e9891323cdd	-32	None	2017-12-27 21:00:00+00:00	1950.40

Figure 9.4: DataFrame with the transactions each day

6. Extract the symbols from the **`positions`** DataFrame and use the OpenBB Platform screener to download an overview for each one. The overview includes the sector that we'll use to build the symbol-to-sector mapping:

```
symbols = positions.columns[:-1].tolist()
screener_data = obb.equity.profile(
    symbols, provider="yfinance")
```

The result is a DataFrame with summary information for each symbol.



	symbol	name	stock_exchange	long_description	company_url	...	shares_float	shares_implied_outstanding	shares_short	dividend_yield	beta
0	AAL	American Airlines Group Inc.	NMS	American Airlines Group Inc., through its subs...	https://www.aa.com	...	6.471347e+08	6.661270e+08	53327236.0	NaN	1.580
1	AAPL	Apple Inc.	NMS	Apple Inc. designs, manufactures, and markets ...	https://www.apple.com	...	1.530832e+10	1.566390e+10	99287450.0	0.0052	1.264
2	ABBV	AbbVie Inc.	NYQ	AbbVie Inc. discovers, develops, manufactures,...	https://www.abbvie.com	...	1.762248e+09	1.765870e+09	15927790.0	0.0397	0.593
3	ADBE	Adobe Inc.	NMS	Adobe Inc., together with its subsidiaries, op...	https://www.adobe.com	...	4.467590e+08	4.801530e+08	8007830.0	NaN	1.281
4	ADP	Automatic Data Processing, Inc.	NMS	Automatic Data Processing, Inc. provides cloud...	https://www.adp.com	...	4.083909e+08	4.092910e+08	4142715.0	0.0232	0.785
...	...	...	...	...	...	...	...	...	...	...	...
128	WBA	Walgreens Boots Alliance, Inc.	NMS	Walgreens Boots Alliance, Inc. operates as a h...	https://www.walgreensbootsalliance.com	...	7.131361e+08	8.627130e+08	43268583.0	0.0650	0.793
129	WDC	Western Digital Corporation	NMS	Western Digital Corporation develops, manufact...	https://www.westerndigital.com	...	3.249116e+08	3.265250e+08	21805777.0	NaN	1.521
130	WFC	Wells Fargo & Company	NYQ	Wells Fargo & Company, a financial services co...	https://www.wellsfargo.com	...	3.478506e+09	3.486320e+09	38130739.0	0.0236	1.182
131	WFM	None	YHD	None	None	...	NaN	NaN	NaN	NaN	NaN
132	WMT	Walmart Inc.	NYQ	Walmart Inc. engages in the operation of retai...	https://corporate.walmart.com	...	4.332595e+09	8.063520e+09	43289108.0	0.0128	0.494

Figure 9.5: DataFrame with the screener results for the tickers used in our strategy

7. Build a mapping between each **`symbol`** and **`sector`**:

```
sector_map = (
    screener_data[["symbol", "sector"]]
    .set_index("symbol")
    .reindex(symbols)
    .fillna("Unknown")
    .to_dict()["sector"]
)
```

The result is a dictionary with symbols as keys and the sector as values. Note for symbols that don't have an associated sector, they're marked as **`Unknown`**.

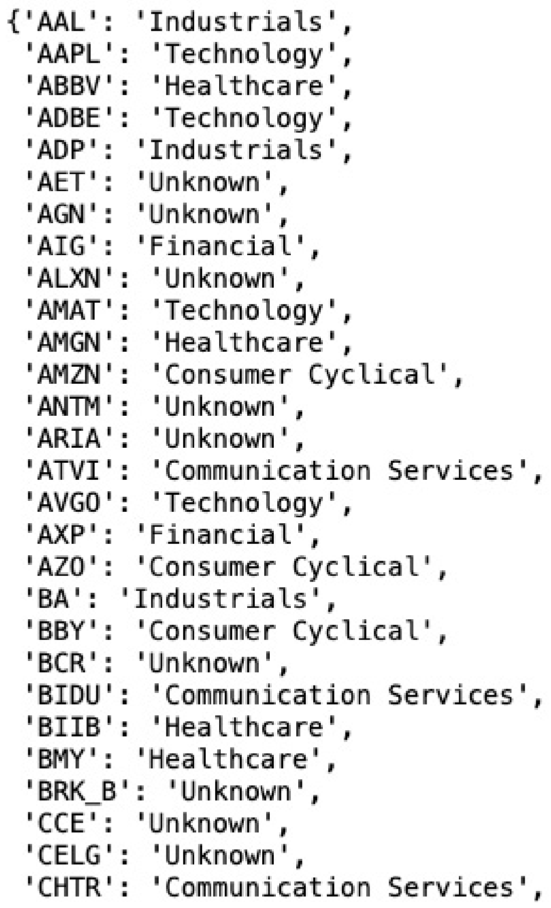
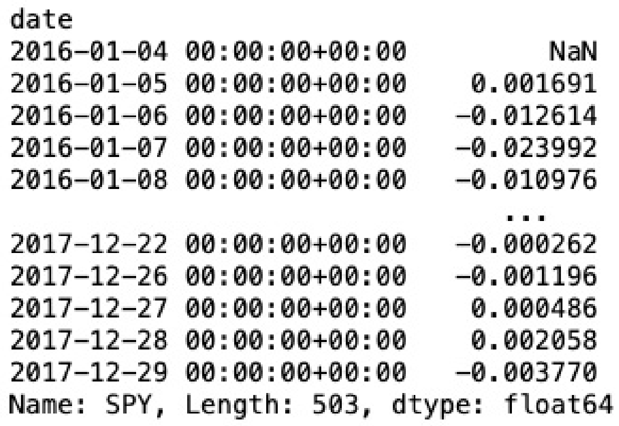

```
{'AAL': 'Industrials',
 'AAPL': 'Technology',
 'ABBV': 'Healthcare',
 'ADBE': 'Technology',
 'ADP': 'Industrials',
 'AET': 'Unknown',
 'AGN': 'Unknown',
 'AIG': 'Financial',
 'ALXN': 'Unknown',
 'AMAT': 'Technology',
 'AMGN': 'Healthcare',
 'AMZN': 'Consumer Cyclical',
 'ANTM': 'Unknown',
 'ARIA': 'Unknown',
 'ATVI': 'Communication Services',
 'AVGO': 'Technology',
 'AXP': 'Financial',
 'AZO': 'Consumer Cyclical',
 'BA': 'Industrials',
 'BBY': 'Consumer Cyclical',
 'BCR': 'Unknown',
 'BIDU': 'Communication Services',
 'BIIB': 'Healthcare',
 'BMY': 'Healthcare',
 'BRK_B': 'Unknown',
 'CCE': 'Unknown',
 'CELG': 'Unknown',
 'CHTR': 'Communication Services',
```

Figure 9.6: Dictionary with a symbol-to-sector mapping for the positions in our backtest

8. Use the OpenBB Platform to download price data for the SPY ETF, which we'll use as the benchmark:

```
spy = obb.equity.price.historical(
    "SPY",
    start_date=returns.index.min(),
    end_date=returns.index.max()
)
spy.index = pd.to_datetime(spy.index)
benchmark_returns = spy.close.pct_change()
benchmark_returns.name = "SPY"
benchmark_returns = benchmark_returns.tz_localize(
    "UTC").filter(returns.index)
```

The result is a pandas Series with daily returns for the SPY ETF.



```
date
2016-01-04 00:00:00+00:00         NaN
2016-01-05 00:00:00+00:00    0.001691
2016-01-06 00:00:00+00:00   -0.012614
2016-01-07 00:00:00+00:00   -0.023992
2016-01-08 00:00:00+00:00   -0.010976
                                ...
2017-12-22 00:00:00+00:00   -0.000262
2017-12-26 00:00:00+00:00   -0.001196
2017-12-27 00:00:00+00:00    0.000486
2017-12-28 00:00:00+00:00    0.002058
2017-12-29 00:00:00+00:00   -0.003770
Name: SPY, Length: 503, dtype: float64
```

Figure 9.7: pandas Series with the daily returns for the SPY ETF

How it works…

The **extract_rets_pos_txn_from_zipline** function takes a DataFrame generated by a Zipline backtest as input and extracts daily returns, net position values, and transaction details. These extracted metrics are returned as a tuple of pandas Series and DataFrames, ready for further analysis or visualization with other Pyfolio functions. The function first normalizes the index of the input DataFrame and sets its time zone to UTC, then extracts the **returns** column for daily strategy returns. It iterates through the **positions** and **transactions** items in the DataFrame to construct daily net position values and transaction details, respectively.

Next, we use a list comprehension to replace each of the column labels in the **positions** DataFrame with strings. We use the pandas **apply** method on the **symbol** column in the **transactions** DataFrame to replace the Zipline Reloaded **Equity** objects with their string representations.

After we extract a list of symbols and download the overview using the OpenBB Platform, we build the symbol-to-sector mapping. We start with the OpenBB Platform output that contains columns **symbol** and **sector**, and sets the index to the **symbol** column. The DataFrame is then reindexed based on the list of **symbols**, fills any missing values with **Unknown**, and finally converts the **sector** column to a dictionary.

Finally, we download and process price data for the SPY ETF which we use as the benchmark in our analysis. The code loads historical data for the dates in our analysis using the **`obb.equity.price.historical`** method. It then calculates the percentage change of the adjusted closing prices to generate benchmark returns, sets the name of the Series to **`SPY`**, localizes the time zone to UTC, and filters the Series to match the index of the **`returns`** DataFrame.

There's more…

You may be wondering why mapping symbols to sectors is important. Incorporating sector information in backtest analysis is crucial for understanding the source of returns and for risk management. Traders often attribute returns to different sectors to identify which segments of the market are driving their portfolio's performance.

This sector-based attribution enables traders to diversify their investments across various sectors, thereby reducing the portfolio's systemic risks associated with any single sector.

Further, by comparing the strategy's returns against a benchmark, like the S&P 500, traders can assess whether the strategy is adding value over and above a passive investment approach. This comparative analysis aids in isolating the strategy's alpha, or risk-adjusted returns, and helps in understanding its behavior relative to the broader market or a specific sector.

See also

Pyfolio offers several helper functions to process Zipline backtest results into a form suitable for further analysis. For more insight into using Pyfolio for risk and performance analysis, see the source code at **https://github.com/stefan-jansen/pyfolio-reloaded/blob/main/src/pyfolio/utils.py**.

In case you want to automate the process of computing risk and performance metrics, **Trade Blotter** is an app that makes performance analytics easy. You can upload your transactions and it does the heavy lifting for you. You can sign up at https://tradeblotter.io/. You can also check out the cohort-based course, *Getting Started With Python for Quant Finance*, which covers Pyfolio in detail. The URL is **https://www.pyquantnews.com/getting-started-with-python-for-quant-finance**.

Generating strategy performance and return analytics

Traders use strategy performance and return analysis to evaluate the effectiveness of their trading algorithms. Return analysis, often visualized through equity curves, or return distributions, offers insights into the strategy's profitability over time. Temporal analyses, such as monthly or annual return breakdowns, help identify seasonality or long-term trends that may impact future performance.

By comparing these metrics and analyses against a benchmark, traders can isolate the strategy's alpha, or the excess return over a passive investment approach. This review enables traders to make data-driven modifications to their strategies, enhancing profitability and risk management. In this recipe, we explore Pyfolio Reloaded strategy performance and return analytics.

Getting ready…

We assume the steps in the *Preparing Zipline Reloaded backtest results for Pyfolio Reloaded* recipe were followed. We'll need **`returns`**, **`positions`**, **`transactions`**, and **`benchmark_returns`** defined for this recipe.

How to do it…

We'll build a series of visualizations using Pyfolio Reloaded that graphically depict strategy performance.

1. Plot the strategy's equity curve against the benchmark:

```
pf.plotting.plot_rolling_returns(
    returns,
    factor_returns=benchmark_returns
)
```

The result is a chart with the strategy's equity curve alongside the cumulative returns of the chosen benchmark.

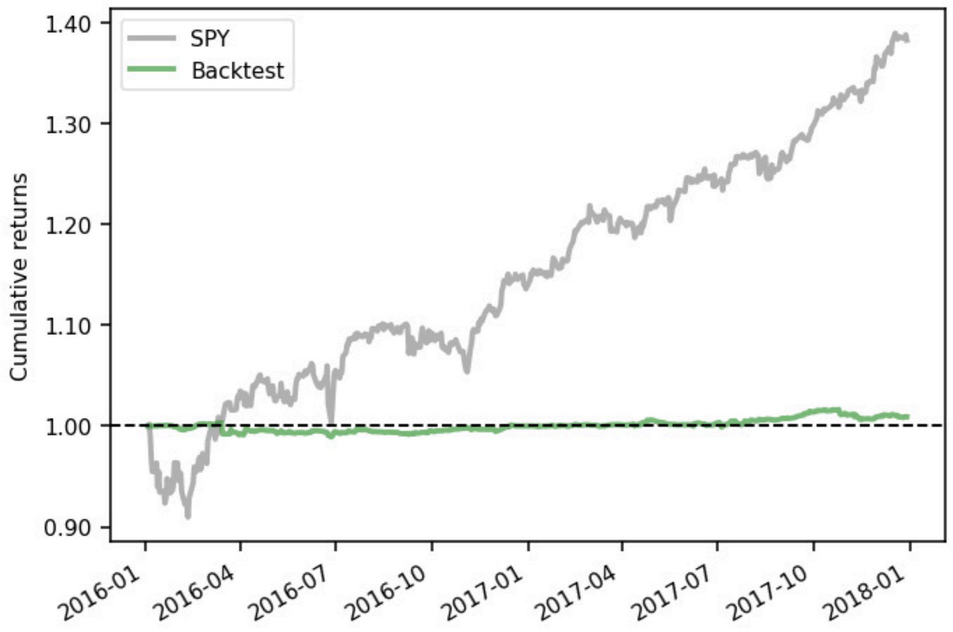


Figure 9.8: Strategy cumulative returns (equity curve) against the benchmark

2. Summarize the distribution of key performance indicators:

```
pf.plotting.plot_perf_stats(
    returns=returns,
    factor_returns=benchmark_returns,
)
```

The result is a horizontal box plot that depicts the distribution of performance indicators.

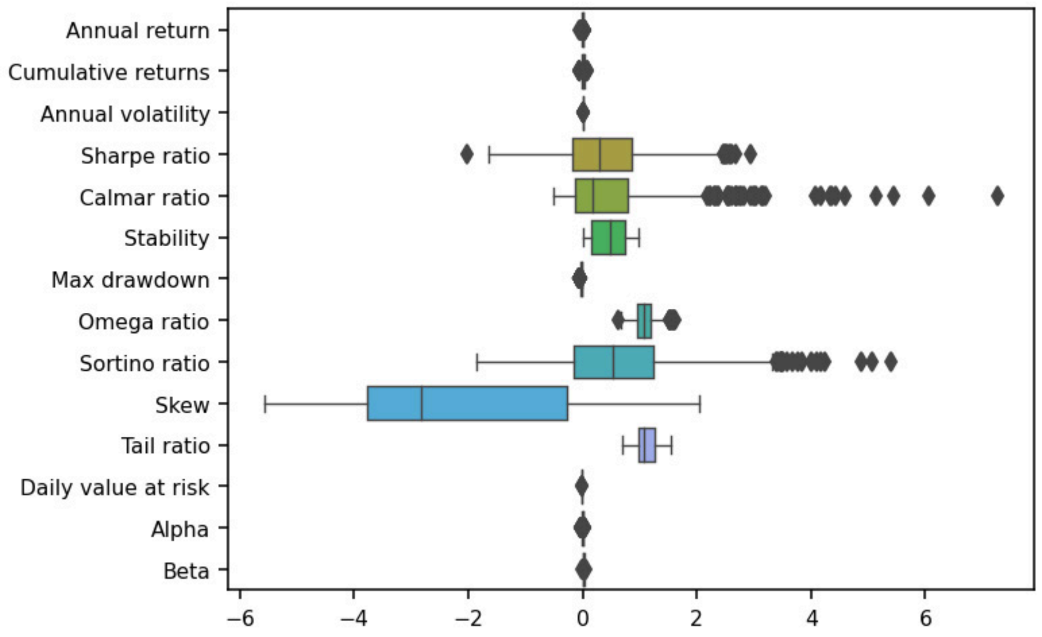


Figure 9.9: Strategy performance metrics

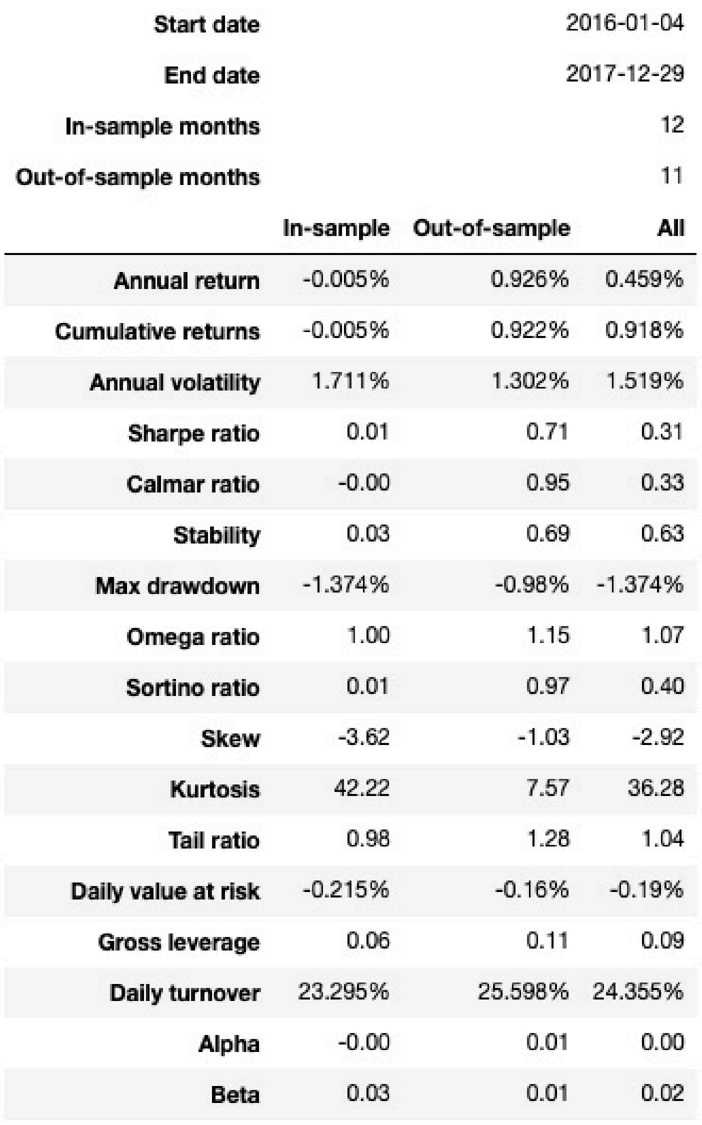
3. Generate a detailed outline of the strategy's performance metrics:

```
pf.plotting.show_perf_stats(
    returns,
    factor_returns=benchmark_returns,
    positions=positions,
    transactions=transactions,
    live_start_date="2017-01-01"
)
```

IMPORTANT NOTE

The **`live_start_date`** *argument in the* **`show_perf_stats`** *function separates the backtest and live trading periods. Metrics are calculated separately for the periods before and after this date, which allows for a more nuanced evaluation of the trading strategy's performance. Specifically, the function will display performance statistics for the backtest period up to* **`live_start_date`** *and for the live trading period starting from* **`live_start_date`**. *Several Pyfolio functions accept this argument.*

The result is a pandas DataFrame containing the key performance metrics.



Start date			2016-01-04
End date			2017-12-29
In-sample months			12
Out-of-sample months			11

	In-sample	**Out-of-sample**	**All**
Annual return	-0.005%	0.926%	0.459%
Cumulative returns	-0.005%	0.922%	0.918%
Annual volatility	1.711%	1.302%	1.519%
Sharpe ratio	0.01	0.71	0.31
Calmar ratio	-0.00	0.95	0.33
Stability	0.03	0.69	0.63
Max drawdown	-1.374%	-0.98%	-1.374%
Omega ratio	1.00	1.15	1.07
Sortino ratio	0.01	0.97	0.40
Skew	-3.62	-1.03	-2.92
Kurtosis	42.22	7.57	36.28
Tail ratio	0.98	1.28	1.04
Daily value at risk	-0.215%	-0.16%	-0.19%
Gross leverage	0.06	0.11	0.09
Daily turnover	23.295%	25.598%	24.355%
Alpha	-0.00	0.01	0.00
Beta	0.03	0.01	0.02

Figure 9.10: Strategy performance metrics

4. Generate a heatmap of the strategy's monthly returns:

```
pf.plotting.plot_monthly_returns_heatmap(returns)
```

The result is a heatmap visualizing the strategy returns during the backtest period.

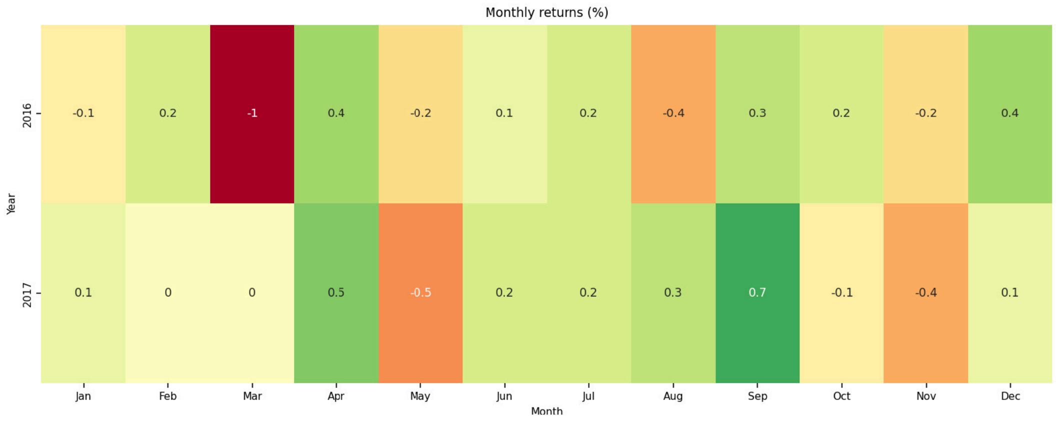


Figure 9.11: Heatmap visualizing the strategy's monthly returns

5. Generate a bar chart of the strategy's annual returns:

```
pf.plotting.plot_annual_returns(returns)
```

The result is a bar chart visualizing the annual returns during the backtest period.

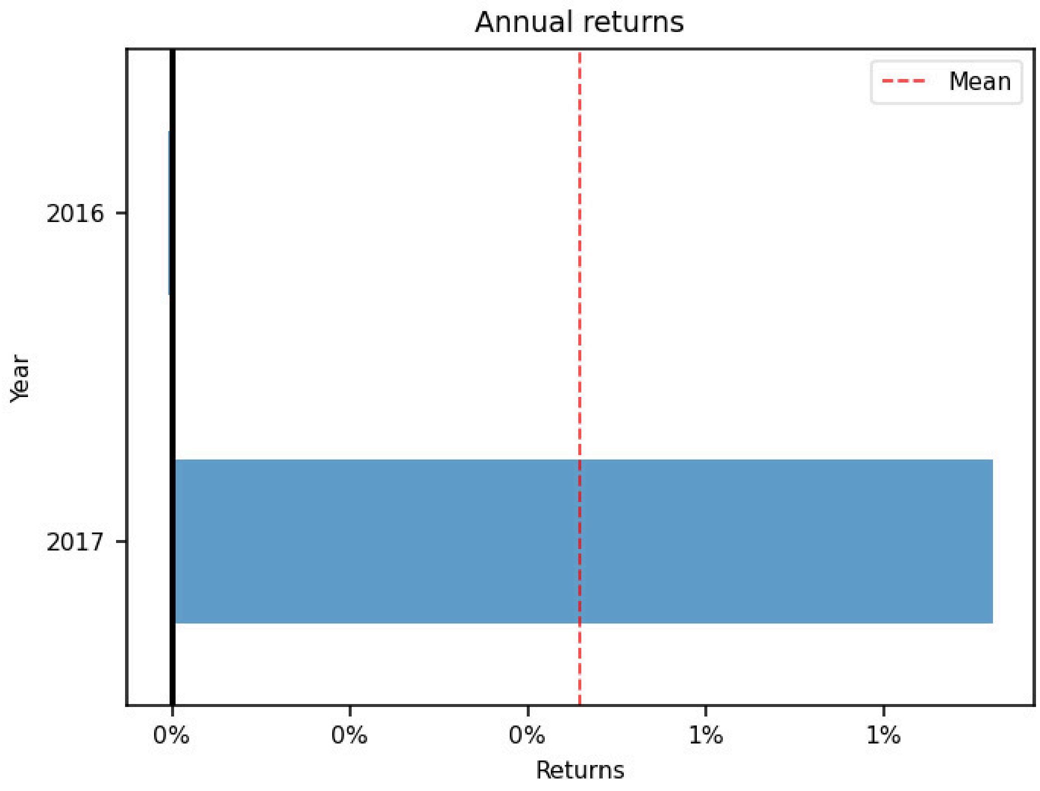


Figure 9.12: Bar chart visualizing the strategy's annual returns

How it works…

Pyfolio does the hard work of parsing the input data and formatting the output charts. Other charts and tables include the following:

- **`plot_rolling_returns`**: Calculates the rolling returns of a portfolio over a specified window and plots them using Matplotlib.
- **`plot_perf_stats`**: Iterates through the returns series, extracting each performance metric, and uses Matplotlib's **`barh`** function to create horizontal bars for each metric.
- **`show_perf_stats`**: Calculates various performance metrics including annual return, annual volatility, and Sharpe ratio, among others, and then displays these metrics in a formatted table.
- **`plot_monthly_returns_heatmap`**: Calculates the mean return for each month across years. It then utilizes Matplotlib to generate a heatmap, where the x-axis represents months, the y-axis represents years, and the color intensity indicates the mean return value.
- **`plot_annual_returns`**: Calculates annual returns by resampling the data to yearly frequency using the mean. It then generates a bar plot of these annual returns using Matplotlib, with the x-axis representing years and the y-axis representing the annual returns.

Now that we covered how the plots are generated, let's review some of the key statistics output by Pyfolio:

- **Calmar ratio**: Calmar ratio is calculated by dividing the **compound annual growth rate** (**CAGR**) of a trading strategy by the maximum drawdown experienced over a specified period
- **Omega ratio**: The omega ratio is calculated by dividing the sum of positive excess returns by the absolute sum of negative excess returns over a given threshold
- **Skew**: Skew is calculated by taking the third standardized moment of the return series, essentially measuring the asymmetry of the return distribution
- **Kurtosis**: Kurtosis is calculated by taking the fourth central moment of the returns series and dividing it by the square of the variance, effectively measuring the "tailedness" of the distribution
- **Value at risk**: Daily **value at risk** (**VaR**) is calculated by taking the negative of the quantile of the daily returns at a given confidence

level, typically 5% or 1%

- **Gross leverage**: Gross leverage is calculated as the sum of the absolute values of long and short positions divided by the portfolio's net asset value at each time point

There's more…

Pyfolio offers more details about a strategy's returns, which can provide more insight into how the strategy performed during the backtest.

1. Create a distribution of monthly returns:

```
pf.plotting.plot_monthly_returns_dist(returns)
```

The output is a histogram with the frequency of monthly returns.

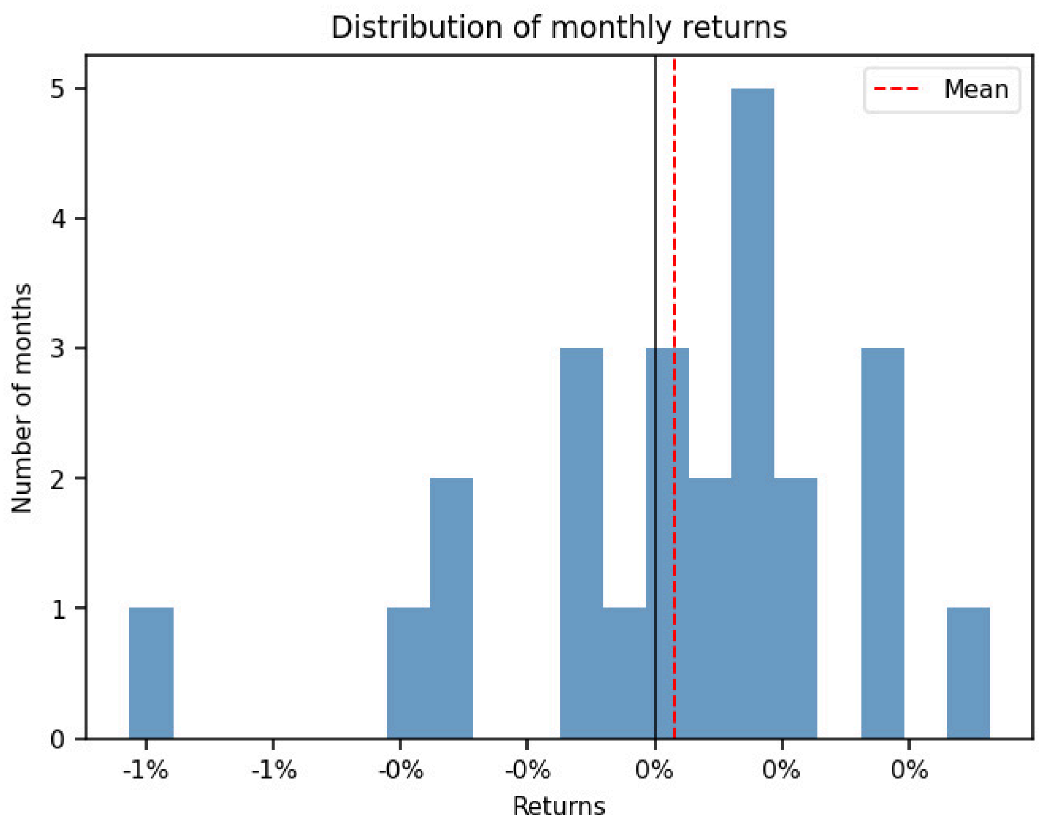


Figure 9.13: Histogram with the frequency of monthly returns

2. Visualize the strategy's daily returns through time:

```
pf.plotting.plot_returns(
    returns,
    live_start_date="2017-01-01"
)
```

The result is a line plot depicting the daily returns.

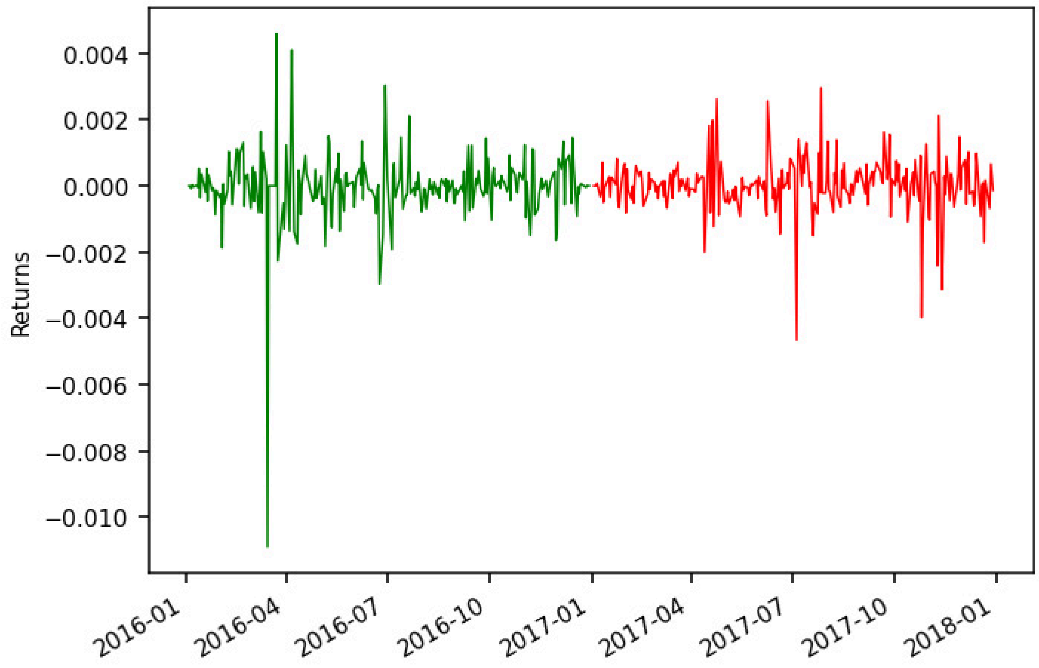


Figure 9.14: Line chart with the daily returns

3. Visualize the return series in quantiles and their cumulative returns for each quantile:

```
pf.plotting.plot_return_quantiles(
    returns,
    live_start_date="2017-01-01"
)
```

The result is a box plot depicting the quantiles of daily, weekly, and monthly returns along with the distribution of cumulative returns.

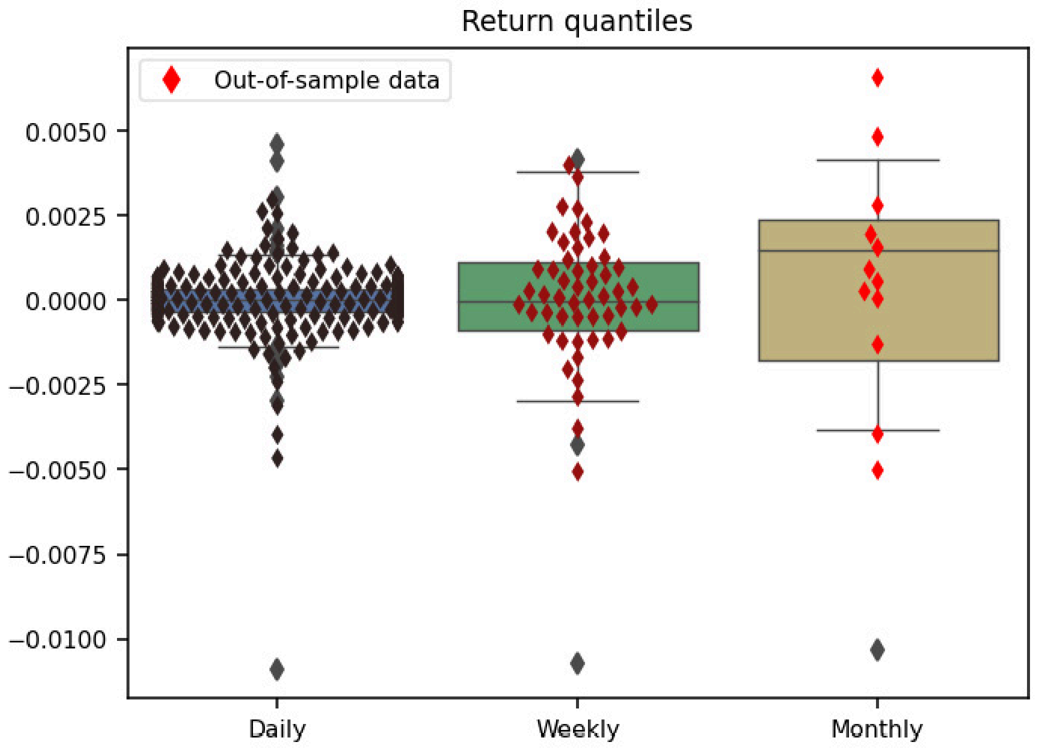


Figure 9.15: Box plot depicting the quantiles of daily, weekly, and monthly returns along with the distribution of cumulative returns

See also

You can dive deeper into Pyfolio's performance and return metrics by reviewing the source code here: **https://github.com/stefan-jansen/pyfolio-reloaded/blob/main/src/pyfolio/plotting.py**

Building a drawdown and rolling risk analysis

A focus only on returns without considering risk is like driving a fast car at high speeds without a seatbelt—it may work for a while, but the consequences can be catastrophic. Risk metrics provide the analytical framework to quantify and manage uncertainty, which lets traders make more informed decisions. These metrics offer insights into the potential volatility, drawdown, and other adverse conditions a strategy might encounter. By incorporating risk analytics into the trading process, traders can better assess the trade-offs between risk and return, optimize their portfolios for maximum risk-adjusted performance, and establish safeguards to mitigate potential losses.

Pyfolio offers several risk metrics to help maintain control of algorithmic trading systems. We'll look at several in this recipe.

Getting ready…

We assume the steps in the *Preparing Zipline Reloaded backtest results for Pyfolio Reloaded* recipe were followed in preparation for this recipe. We'll need **`returns`**, **`positions`**, **`transactions`**, and **`benchmark_returns`** defined for this recipe.

How to do it…

We'll look at Pyfolio Reloaded drawdown analysis and several rolling risk metrics. Rolling risk metrics help traders understand how strategy risk evolves through time.

1. Graphically depict the top 10 drawdowns over the strategy period:

   ```
   pf.plotting.plot_drawdown_periods(returns, top=10)
   ```

   The result is an equity curve depicting the cumulative strategy returns with vertical shading over the periods the strategy was in drawdown.

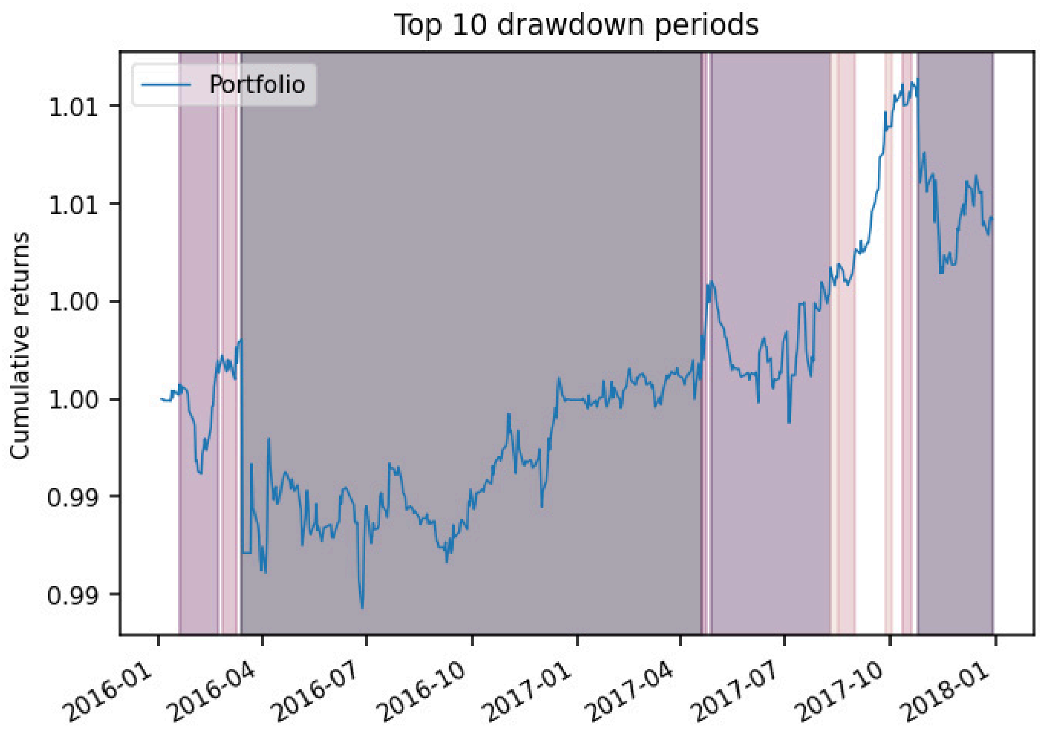


Figure 9.16: Equity curve depicting the cumulative strategy returns with vertical shading over the periods the strategy was in drawdown

2. Visualize the equity drawdown over time:

```
pf.plotting.plot_drawdown_underwater(returns)
```

The result is an **underwater plot** that visualizes the strategy drawdown amounts.

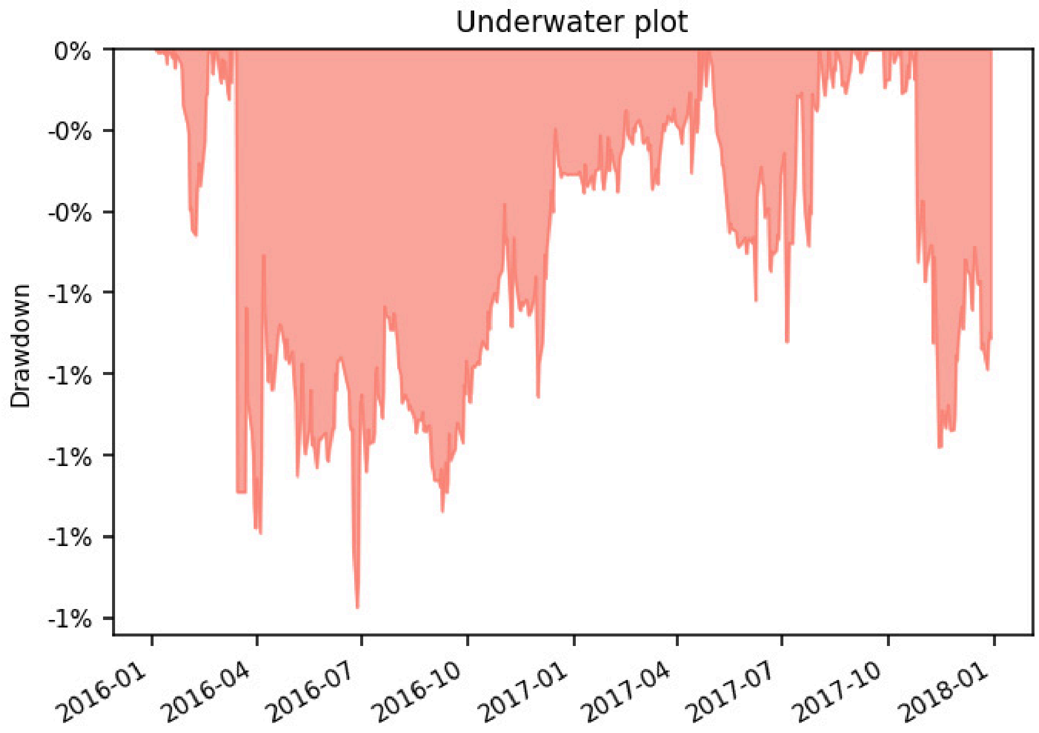
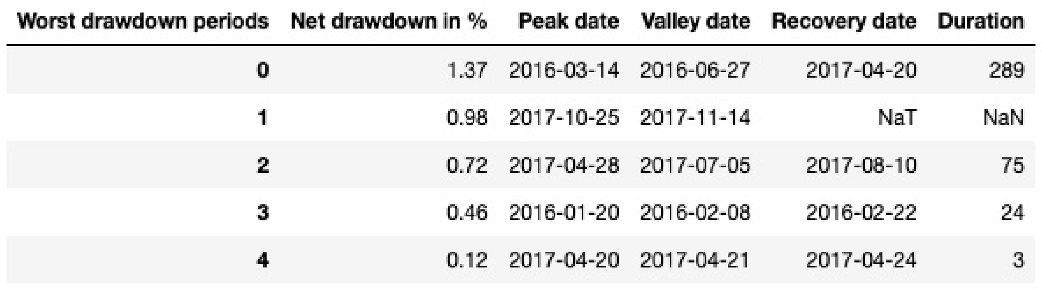


Figure 9.17: Underwater plot visualizing the strategy drawdown amounts

3. Create a table of the worst drawdowns with details of the amount, peak date, valley date, recovery date, and duration:

```
pf.plotting.show_worst_drawdown_periods(returns)
```

The result is a DataFrame detailing the top five drawdown periods.



Worst drawdown periods	Net drawdown in %	Peak date	Valley date	Recovery date	Duration
0	1.37	2016-03-14	2016-06-27	2017-04-20	289
1	0.98	2017-10-25	2017-11-14	NaT	NaN
2	0.72	2017-04-28	2017-07-05	2017-08-10	75
3	0.46	2016-01-20	2016-02-08	2016-02-22	24
4	0.12	2017-04-20	2017-04-21	2017-04-24	3

Figure 9.18: DataFrame detailing the top five drawdown periods

4. Plot the strategy 3-month rolling volatility against the benchmark:

```
pf.plotting.plot_rolling_volatility(
    returns,
    factor_returns=benchmark_returns,
    rolling_window=66
)
```

The result is a chart with the rolling 3-month volatility compared to the benchmark.

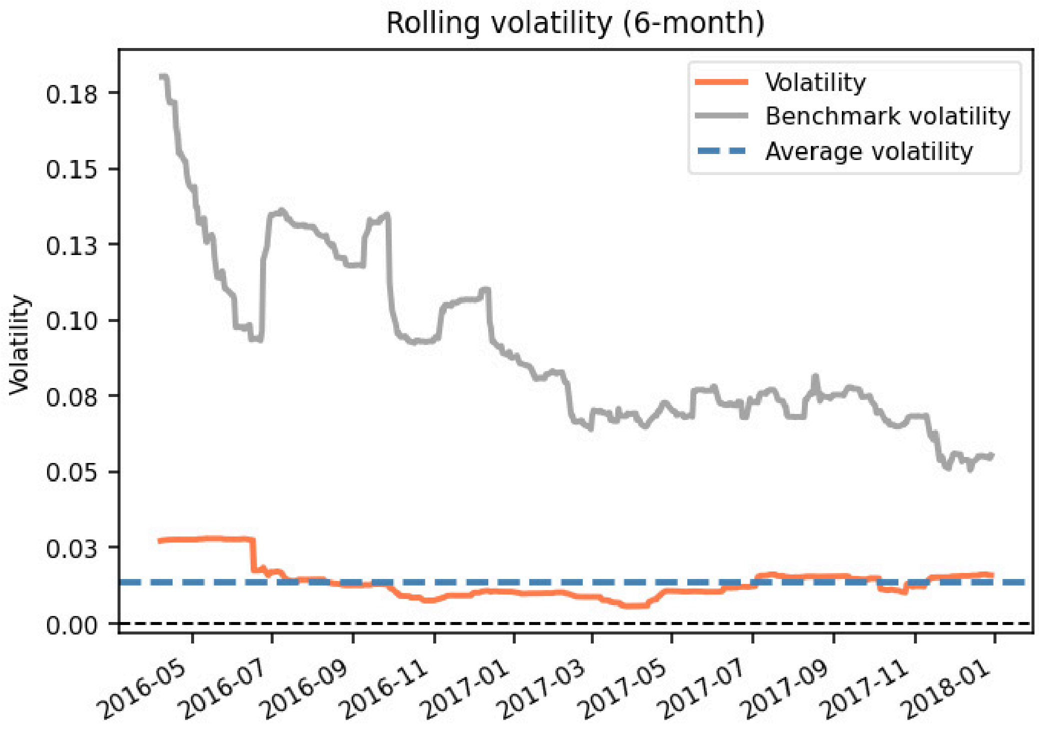


Figure 9.19: Chart with the rolling 3-month volatility compared to the benchmark

5. Plot the strategy 3-month rolling Sharpe ratio, its mean, and the benchmark Sharpe:

```
pf.plotting.plot_rolling_sharpe(
    returns,
    factor_returns=benchmark_returns,
    rolling_window=66
)
```

The result is a chart with the rolling 3-month Sharpe ratio compared to the benchmark.

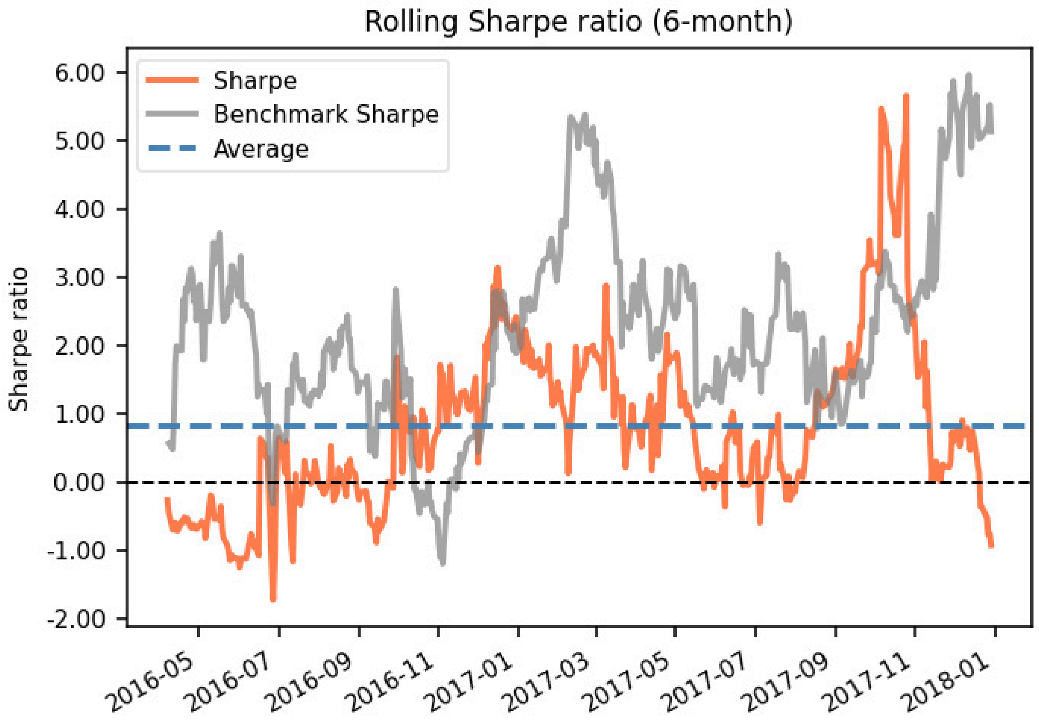


Figure 9.20: Chart with the rolling 3-month Sharpe ratio compared to the benchmark

How it works...

To generate the drawdown and rolling risk metrics, Pyfolio Reloaded uses the same underlying module as the performance and return metrics:

- **`plot_drawdown_periods`**: Computes the drawdown periods by identifying local maxima and subsequent declines. It then uses Matplotlib to visualize these drawdown periods, highlighting the start, valley, and end of each period on the plot.
- **`plot_drawdown_underwater`**: Finds the maximum cumulative return up to each point and subtracts the cumulative return from it. It then utilizes Matplotlib to plot the underwater curve, representing the negative drawdowns over time.
- **`show_worst_drawdown_periods`**: First, this computes the drawdowns, then sorts these drawdown periods and displays the top **`n`** worst drawdown periods in a pandas DataFrame.
- **`plot_rolling_volatility`**: Applies a rolling window and computes the standard deviation of returns within each window. It then uses Matplotlib to plot the computed rolling volatility against time.

- **`plot_rolling_sharpe`**: Applies a rolling window and computes the Sharpe ratio within each window. It then uses Matplotlib to plot the computed rolling Sharpe ratio against time.

Let's cover some of the key risk metrics we covered:

- **Annual volatility**: The annual volatility is calculated by taking the standard deviation of the daily returns and then annualizing it by multiplying it by the square root of the number of trading days (usually 252).
- **Sharpe ratio**: The Sharpe ratio is calculated as the mean of the excess returns divided by the standard deviation of those excess returns.
- **Max drawdown**: Max drawdown is calculated by identifying the maximum difference between a peak and a subsequent trough in a time series of portfolio values.

There's more...

Pyfolio allows for the overlay of specific event timelines, enabling a comparative analysis of portfolio and benchmark performance during these periods. An example would be assessing performance during the market downturn of what is now considered the New Normal. The following steps show you how.

1. Use **`extract_interesting_date_ranges`** to extract the strategy returns from pre-defined stress events:
```
times = pf.timeseries.extract_interesting_date_ranges(returns)
```
2. Then join with the benchmark returns, compute the cumulative returns, and plot:
```
(
    times["New Normal"]
    .to_frame("strategy_returns")
    .join(benchmark_returns)
    .add(1)
    .cumprod()
    .sub(1)
```

```
        .plot()
)
```

The result is a plot that depicts the strategy returns compared to the benchmark during the stress event.

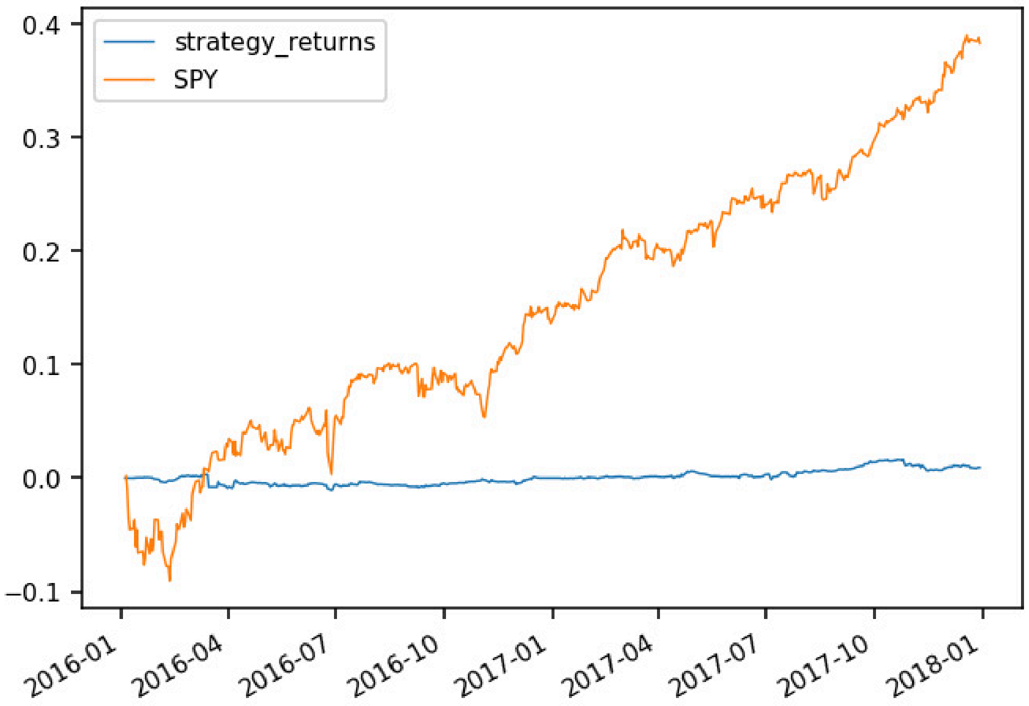


Figure 9.21: Plot that depicts the strategy returns compared to the benchmark during the stress event

See also

To get a list of stress periods and their start and end dates, check into the source code here: **https://github.com/stefan-jansen/pyfolio-reloaded/blob/main/src/pyfolio/interesting_periods.py**

Analyzing strategy holdings, leverage, exposure, and sector allocations

We can extend our strategy analysis with Pyfolio by analyzing holdings, leverage, and sector allocations over time. Analyzing holdings over time helps traders understand the diversification and concentration risks within a portfolio. It helps traders identify overexposure to specific assets, which could be detrimental in adverse market conditions. Leverage analysis is equally important, as excessive borrowing can amplify losses, leading to significant drawdowns or even portfolio liquidation. Monitoring leverage levels over time allows traders to adjust their risk exposure in line with their risk tolerance and market outlook.

Sector allocation analysis provides insights into how diversified the portfolio is across different industries. This is important for risk management, as different sectors respond differently to economic cycles and market events. Understanding sector allocations can help traders optimize their portfolios for various market conditions and potentially enhance returns while mitigating risks. In this recipe, we'll look at how Pyfolio is used for portfolio analysis.

Getting ready…

We assume the steps in the *Preparing Zipline Reloaded backtest results for Pyfolio Reloaded* recipe were followed in preparation for this recipe. We'll need **`returns`**, **`positions`**, and **`sector_map`** defined for this recipe.

How to do it…

We'll look at how the strategy holdings, leverage, exposure, and sector allocation evolve throughout the analysis period.

1. Plot the number of daily holdings, average holdings by month, and overall average holdings:

   ```
   pf.plotting.plot_holdings(returns, positions)
   ```

   The result is a plot of the number of daily holdings, average holdings by month, and overall average holdings.

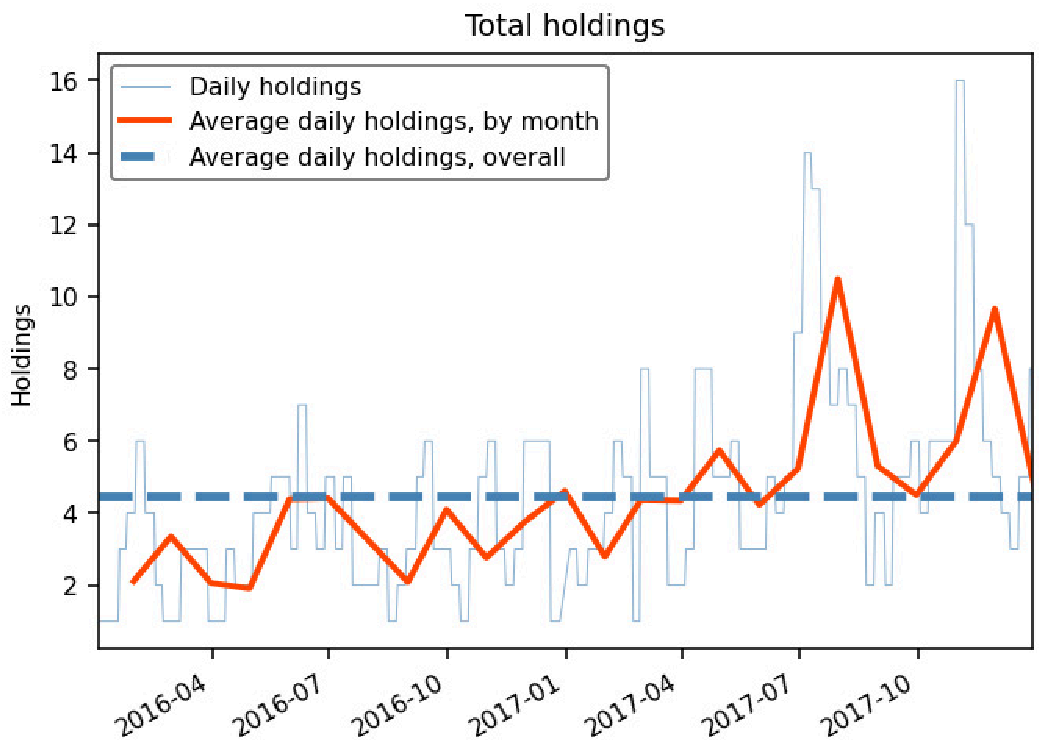


Figure 9.22: Plot of number of daily holdings, average holdings by month, and overall average holdings

2. Plot the number of long and short holdings:

```
pf.plotting.plot_long_short_holdings(
    returns,
    positions
)
```

The result is a plot of the number of long and short holdings.

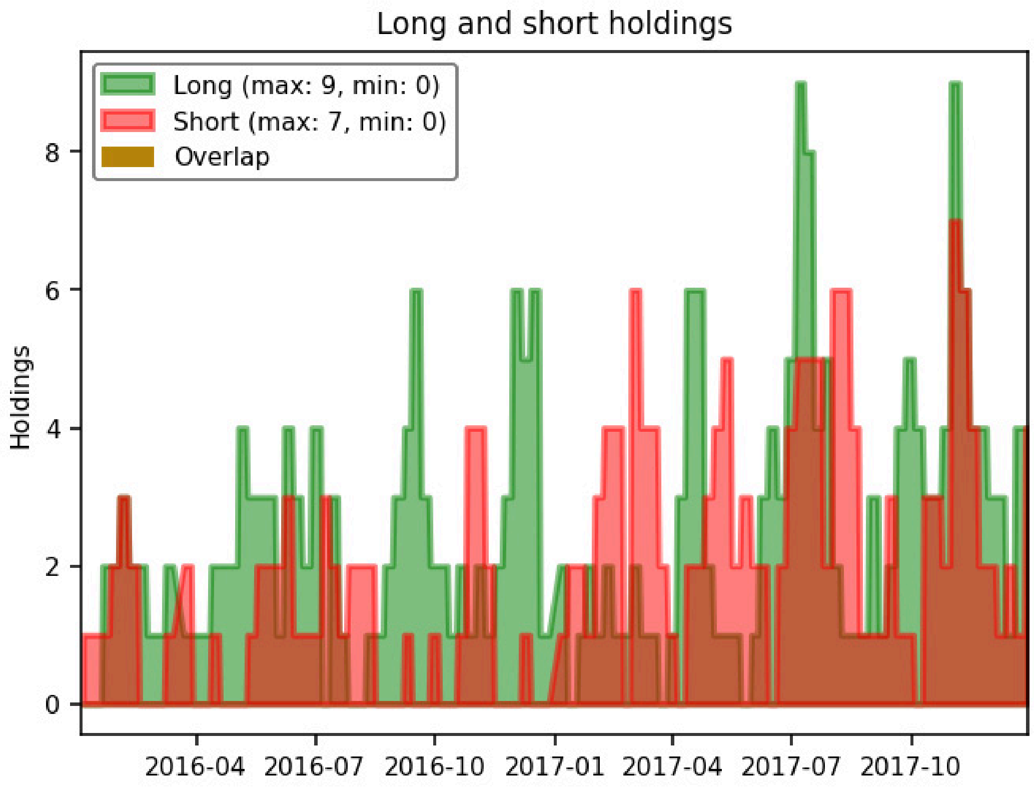


Figure 9.23: Plot of the number of long and short holdings

3. Plot the gross strategy leverage:

```
pf.plotting.plot_gross_leverage(returns, positions)
```

The result is a plot of the gross strategy leverage.

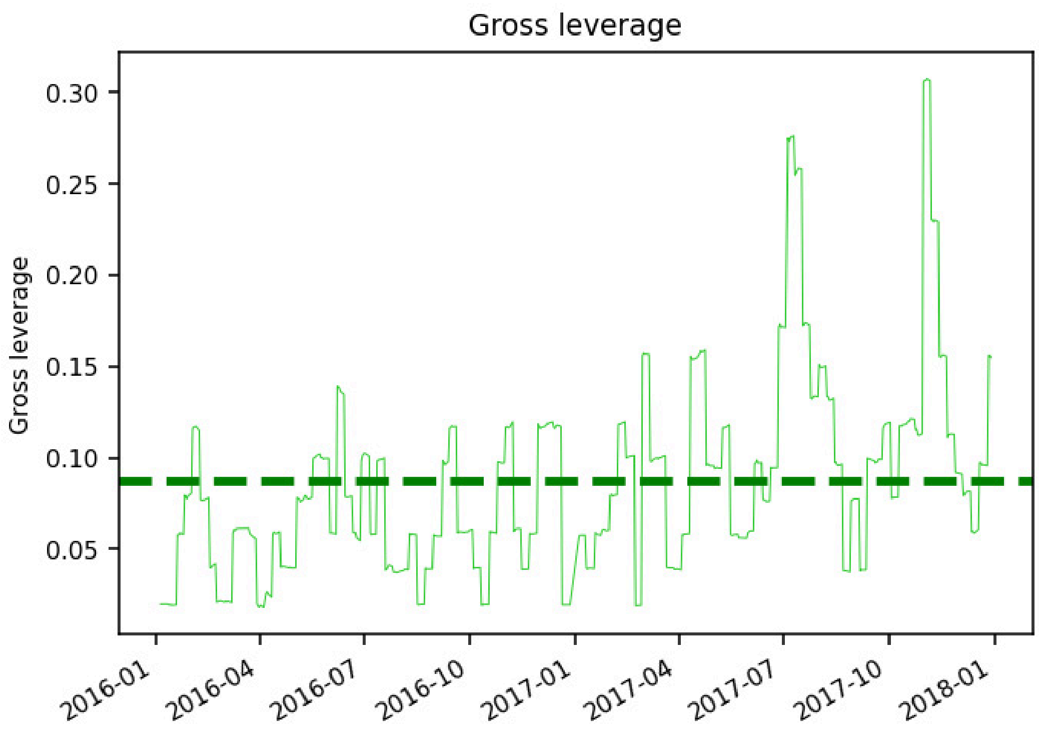


Figure 9.24: Plot of the strategy's gross leverage

4. Plot the long, short, and net exposure:

```
pf.plotting.plot_exposures(returns, positions)
```

The result is a plot of long, short, and net exposure.

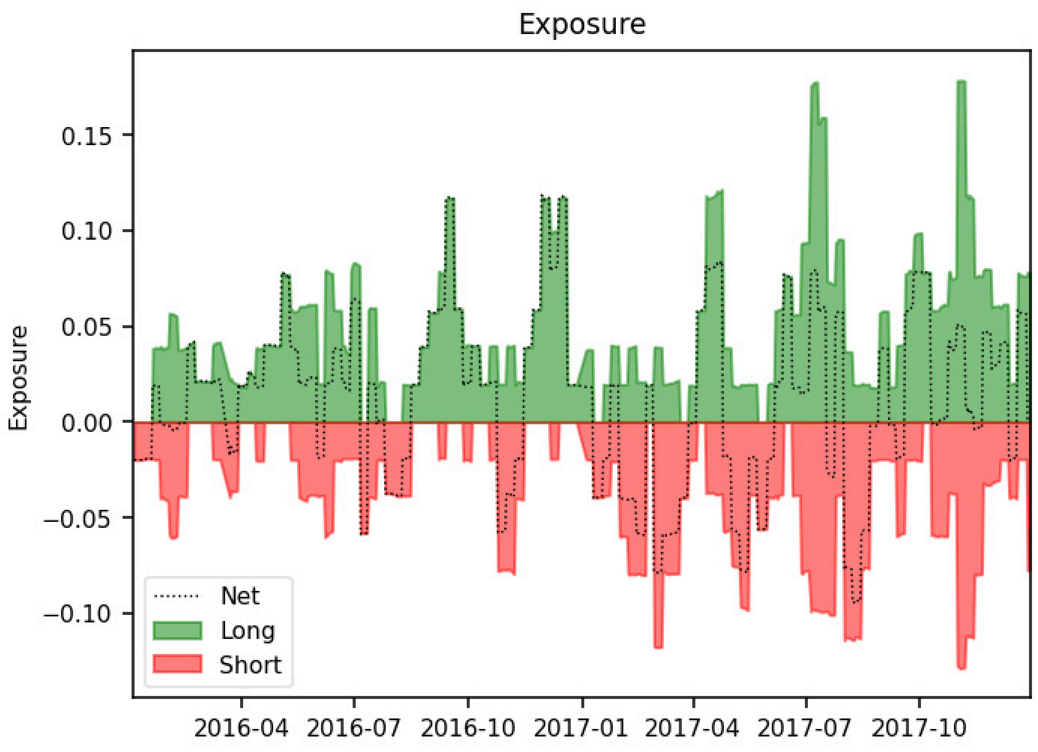


Figure 9.25: Plot of the strategy's long, short, and net exposure

5. Generate a pandas DataFrame with the percentage allocation of each position:

```
positions_alloc = pf.pos.get_percent_alloc(positions)
```

6. Generate a table of the top long, short, and net positions of all time:

```
pf.plotting.show_and_plot_top_positions(
    returns,
    positions_alloc,
    show_and_plot=2
)
```

The result is a pandas DataFrame and chart with the maximum percentage allocation of each of the top 10 strategy holdings.

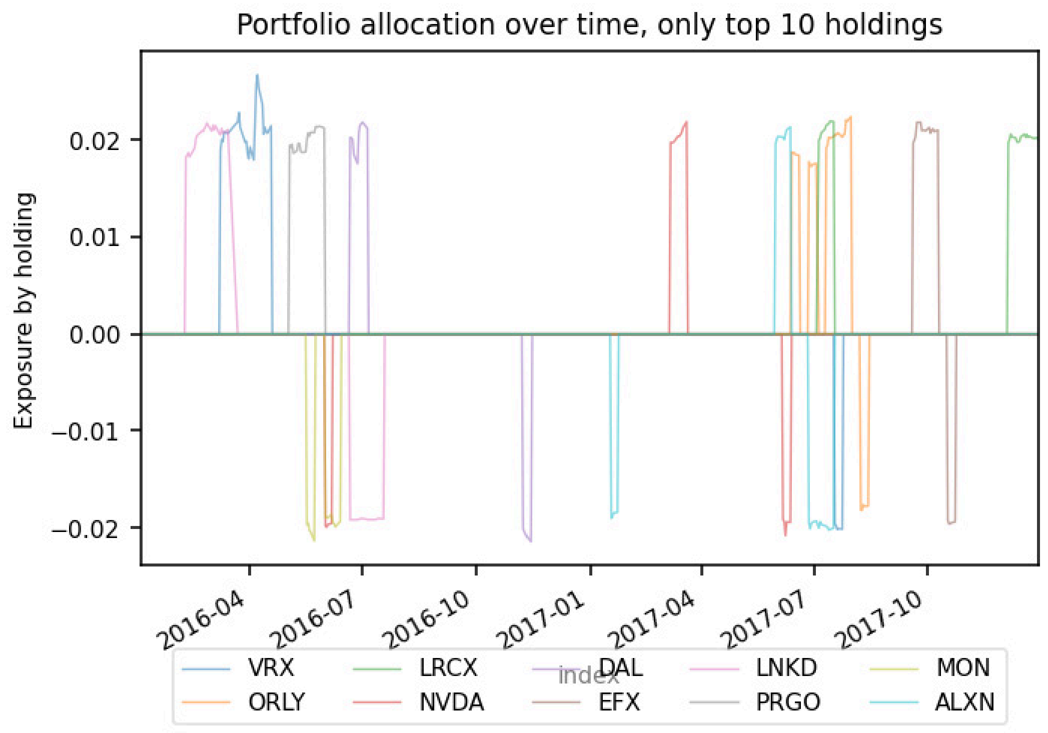


Figure 9.26: pandas DataFrame and chart with the maximum percentage allocation of each of the top 10 strategy holdings

7. Generate the sector allocations based on the positions and sector mapping:

```
sector_alloc = pf.pos.get_sector_exposures(
    positions,
    symbol_sector_map=sector_map
)
```

8. Plot the sector allocation using Pyfolio's **`plot_sector_allocations`** method:

```
pf.plotting.plot_sector_allocations(
    returns,
    sector_alloc=sector_alloc
)
```

   The result is a chart that graphically depicts the strategy's sector allocation, including cash.

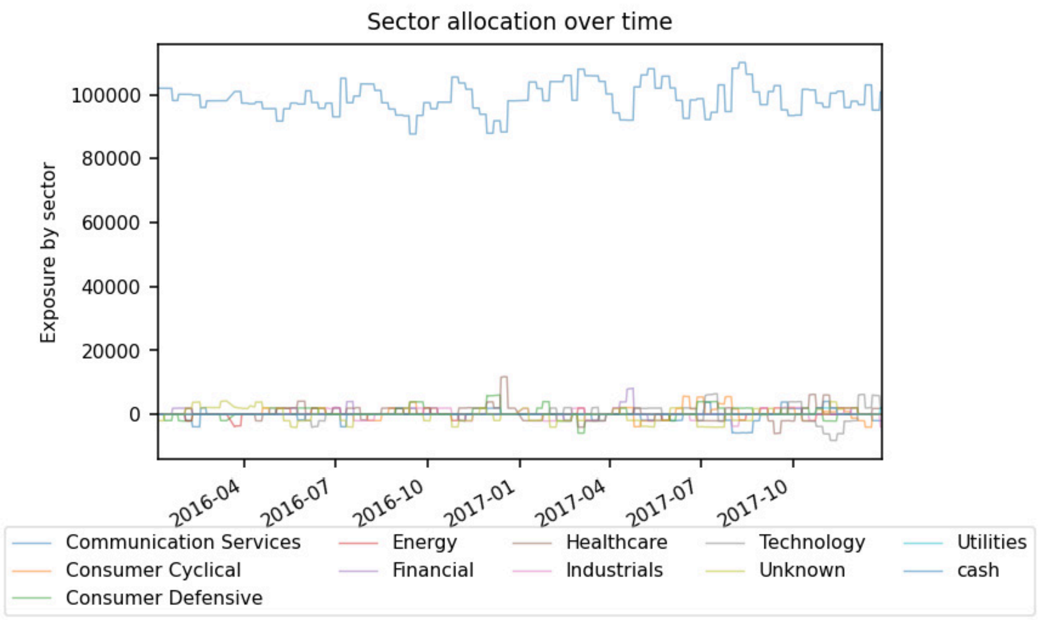


Figure 9.27: Chart of the strategy's sector allocation over time

How it works…

The **`plot_holdings`** function is designed to visualize the total number of stocks with an active position (either short or long) over time. It takes in a pandas Series of daily returns (**`returns`**) and a pandas DataFrame of daily net position values (**`positions`**). The plot gives traders a time-series representation of the strategy's exposure to the market. The daily holdings line gives you an immediate sense of how many positions are held each day, while the monthly and overall averages provide a more smoothed-out perspective, useful for understanding the strategy's typical market exposure.

The **`plot_long_short_holdings`** takes the same arguments as **`plot_holdings`**. It filters out cash positions and counts the number of long and short positions for each day. Using Matplotlib, it plots these counts as shaded regions over time, with green indicating long positions and red indicating short positions. Conceptually, the plot serves as a visual representation of the strategy's exposure to long and short positions over time.

`plot_gross_leverage` takes **`returns`** and **`positions`** and plots the gross leverage over time using Matplotlib. The output provides a time-series view of the strategy's gross leverage which is the ratio of the absolute sum of the long and short positions to the strategy's net asset value. For traders using margin, monitoring gross leverage is critical to quantify the level of risk exposure relative to the portfolio's net asset value. Excessive leverage can amplify both gains and losses, potentially leading to rapid depletion of capital or margin calls. Understanding and managing leverage is essential for risk control and ensuring compliance with trading limits or regulatory requirements.

`plot_exposures` helps visualize the long, short, and net exposure across the strategy. Exposure is calculated by summing the values of either long, short, or all positions on each day and dividing by the sum of all positions on the same day. Long exposure represents the proportion of the portfolio invested in long positions, short exposure indicates the proportion in short positions. The net exposure gives an overall exposure level, which can be interpreted as the strategy's directional bias. A positive net exposure would suggest a bullish stance, while a negative value would indicate a bearish outlook.

The **`show_and_plot_top_positions`** is a snapshot of a strategy's most significant long and short positions. The function takes a pandas DataFrame of positions and a specified number **`N`** to identify the top N long and short positions by net market value. It then calculates the mean position for each asset over the analysis period. The function outputs a table displaying these top positions and also generates a bar plot to visualize them over time.

Finally, **`plot_sector_allocations`** calculates the daily sector allocations by aggregating the positions based on the sector mappings for each stock. The plot is a snapshot of the strategy's average exposure to sectors. It is useful for understanding the portfolio's diversification and risk profile across sectors for hedging purposes.

There's more…

If you're trading intraday strategies, transaction costs play an important part in the profitability of a strategy. Transaction costs can be direct, which are the commissions and fees we pay to our broker and exchanges, and indirect, which include market impact and slippage.

Pyfolio has several tools for measuring these costs:

- **`plot_turnover`**: Plots turnover, which is the number of shares traded for a period as a fraction of total shares. The output displays the daily total, daily average per month, and all-time daily.
- **`plot_slippage_sweep`**: Plots equity curves at different per-dollar slippage assumptions.
- **`plot_slippage_sensitivity`**: Plots curve relating per-dollar slippage to average annual returns.
- **`plot_capacity_sweep`**: Performs a sweep over different starting portfolio values to assess the impact on the Sharpe ratio, considering transaction costs and market slippage, and then plots the resulting Sharpe ratios against the portfolio values.
- **`plot_daily_turnover_hist:`** Plots a histogram of daily turnover rates.
- **`plot_daily_volume`**: Plots the trading volume per day.

Active traders that are concerned about their transaction costs can use these additional tools to analyze costs on their strategy performance.

See also

If you're new to asset allocation and why it matters, you can get a good primer at Investopedia here: **https://www.investopedia.com/terms/a/assetallocation.asp**. A concept close to leverage is margin, which is discussed here: **https://www.investopedia.com/terms/m/margin.asp**. Finally, tools like Trade Blotter can help automate a lot of the risk and performance analytics for you: **https://tradeblotter.io**.

Breaking Down Strategy Performance to Trade Level

So far in this chapter, we've considered risk and performance metrics at the strategy level. This is an important perspective, but not the only one. In this recipe, we'll use Pyfolio Reloaded to look at the strategy at the trade level. Examining strategy risk and performance at the trade level provides more granular insights into how the returns of the strategy are composed. It lets identify specific trades, or classes of trades, that may be contributing disproportionately to risk or returns. This level of examination helps traders optimize strategy features like trade execution, entry and exit criteria, or even asset class. In this recipe, we'll look at trade-level statistics for our strategy.

Getting ready…

We assume the steps in the *Preparing Zipline Reloaded backtest results for Pyfolio Reloaded* recipe were followed in preparation for this recipe. We'll need **`transactions`** defined for this recipe.

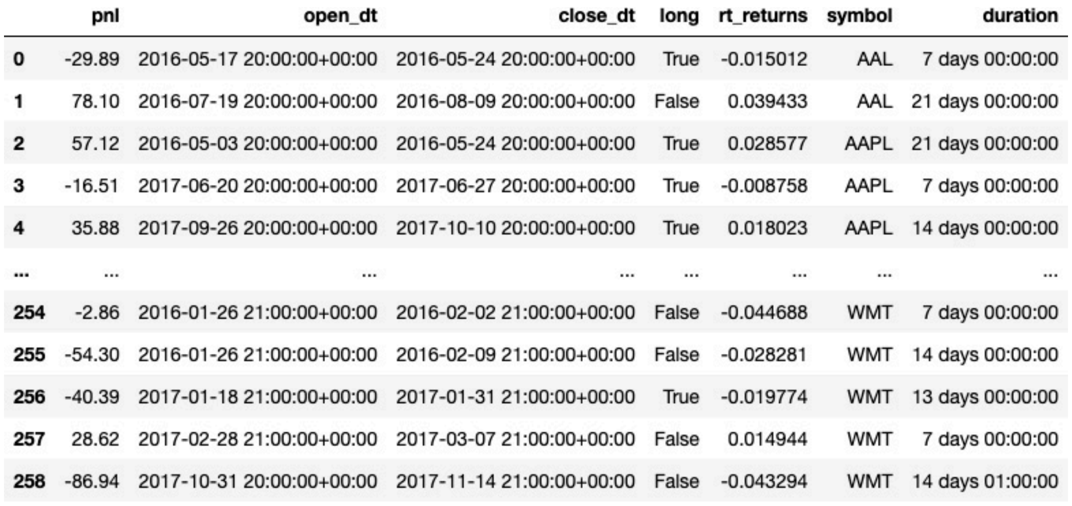
How to do it…

Pyfolio has a utility function to identify a trade from the transaction history. A trade is considered an opening and closing transaction of the same quantity for the same asset.

1. Extract the round trips from the strategy transaction history:

```
round_trips = pf.round_trips.extract_round_trips(
    transactions[["amount", "price", "symbol"]]
)
```

   The result is a pandas DataFrame with the profit or loss, return, and duration of all round trips.



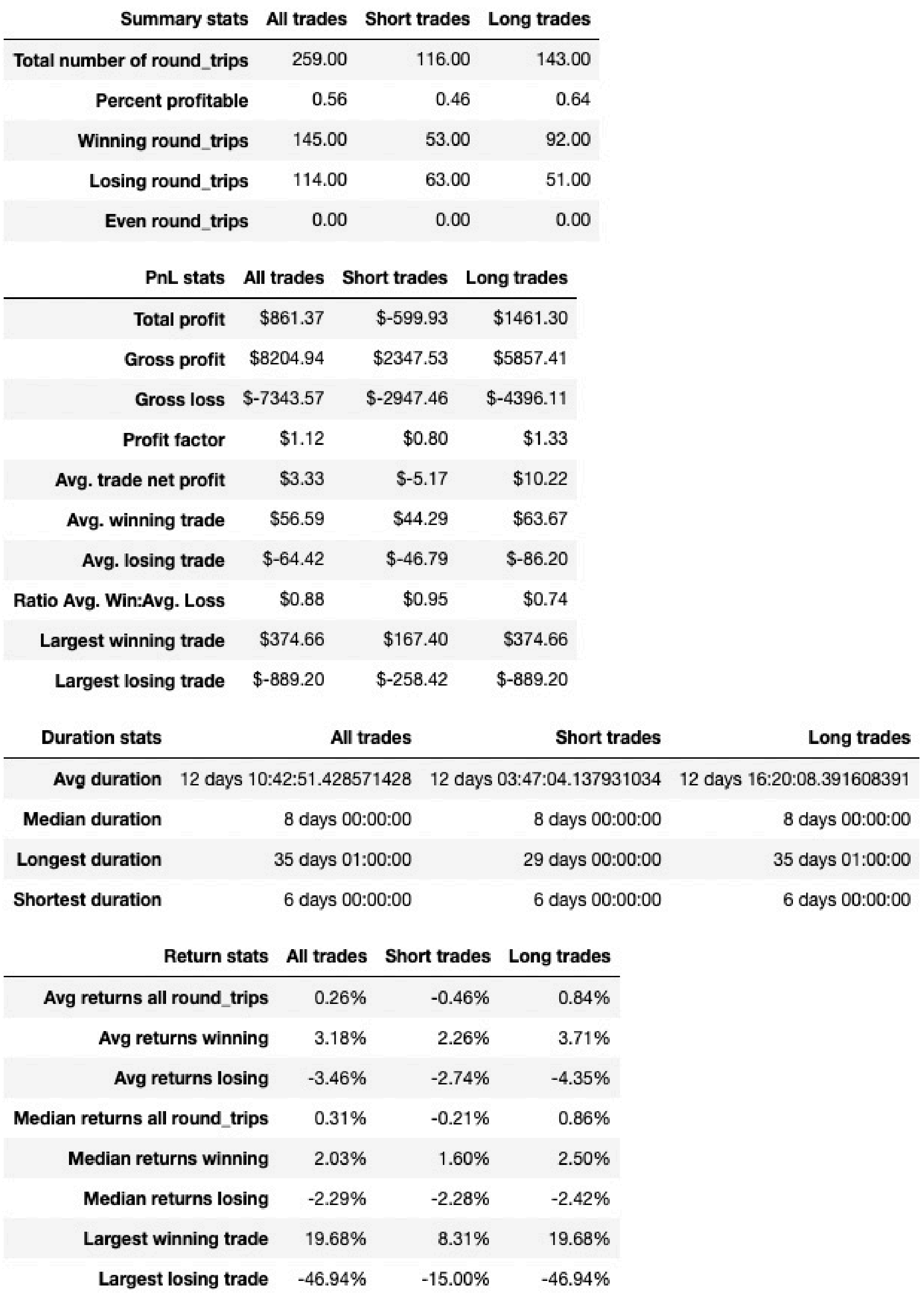
	pnl	open_dt	close_dt	long	rt_returns	symbol	duration
0	-29.89	2016-05-17 20:00:00+00:00	2016-05-24 20:00:00+00:00	True	-0.015012	AAL	7 days 00:00:00
1	78.10	2016-07-19 20:00:00+00:00	2016-08-09 20:00:00+00:00	False	0.039433	AAL	21 days 00:00:00
2	57.12	2016-05-03 20:00:00+00:00	2016-05-24 20:00:00+00:00	True	0.028577	AAPL	21 days 00:00:00
3	-16.51	2017-06-20 20:00:00+00:00	2017-06-27 20:00:00+00:00	True	-0.008758	AAPL	7 days 00:00:00
4	35.88	2017-09-26 20:00:00+00:00	2017-10-10 20:00:00+00:00	True	0.018023	AAPL	14 days 00:00:00
...	...	...	...	...	...	...	...
254	-2.86	2016-01-26 21:00:00+00:00	2016-02-02 21:00:00+00:00	False	-0.044688	WMT	7 days 00:00:00
255	-54.30	2016-01-26 21:00:00+00:00	2016-02-09 21:00:00+00:00	False	-0.028281	WMT	14 days 00:00:00
256	-40.39	2017-01-18 21:00:00+00:00	2017-01-31 21:00:00+00:00	True	-0.019774	WMT	13 days 00:00:00
257	28.62	2017-02-28 21:00:00+00:00	2017-03-07 21:00:00+00:00	False	0.014944	WMT	7 days 00:00:00
258	-86.94	2017-10-31 20:00:00+00:00	2017-11-14 21:00:00+00:00	False	-0.043294	WMT	14 days 01:00:00

Figure 9.28: DataFrame containing the round trips extracted from the transaction history

2. Use the **`print_round_trip_stats`** function to generate the summary statistics of the strategy trades:

```
pf.round_trips.print_round_trip_stats(
    round_trips.rename(
        columns={"rt_returns": "returns"}
    )
)
```

The result is a series of pandas DataFrames with aggregate and asset-specific statistics.



Summary stats	All trades	Short trades	Long trades
Total number of round_trips	259.00	116.00	143.00
Percent profitable	0.56	0.46	0.64
Winning round_trips	145.00	53.00	92.00
Losing round_trips	114.00	63.00	51.00
Even round_trips	0.00	0.00	0.00

PnL stats	All trades	Short trades	Long trades
Total profit	$861.37	$-599.93	$1461.30
Gross profit	$8204.94	$2347.53	$5857.41
Gross loss	$-7343.57	$-2947.46	$-4396.11
Profit factor	$1.12	$0.80	$1.33
Avg. trade net profit	$3.33	$-5.17	$10.22
Avg. winning trade	$56.59	$44.29	$63.67
Avg. losing trade	$-64.42	$-46.79	$-86.20
Ratio Avg. Win:Avg. Loss	$0.88	$0.95	$0.74
Largest winning trade	$374.66	$167.40	$374.66
Largest losing trade	$-889.20	$-258.42	$-889.20

Duration stats	All trades	Short trades	Long trades
Avg duration	12 days 10:42:51.428571428	12 days 03:47:04.137931034	12 days 16:20:08.391608391
Median duration	8 days 00:00:00	8 days 00:00:00	8 days 00:00:00
Longest duration	35 days 01:00:00	29 days 00:00:00	35 days 01:00:00
Shortest duration	6 days 00:00:00	6 days 00:00:00	6 days 00:00:00

Return stats	All trades	Short trades	Long trades
Avg returns all round_trips	0.26%	-0.46%	0.84%
Avg returns winning	3.18%	2.26%	3.71%
Avg returns losing	-3.46%	-2.74%	-4.35%
Median returns all round_trips	0.31%	-0.21%	0.86%
Median returns winning	2.03%	1.60%	2.50%
Median returns losing	-2.29%	-2.28%	-2.42%
Largest winning trade	19.68%	8.31%	19.68%
Largest losing trade	-46.94%	-15.00%	-46.94%

Figure 9.29: Trade-level summary statistics

3. Plot the duration of each round trip, per asset, over time:

```
pf.plotting.plot_round_trip_lifetimes(round_trips)
```

The result is a plot that visualizes the holding period for each asset traded.

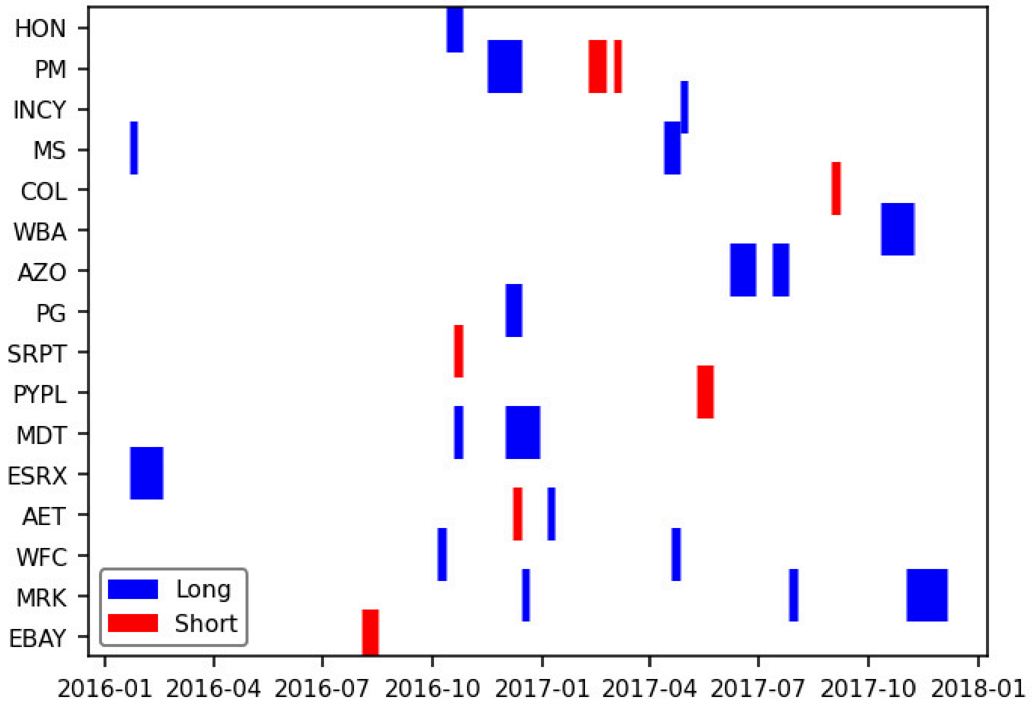


Figure 9.30: Plot visualizing the holding period for each asset traded

How it works…

Round trips are computed by identifying the opening and closing transactions for each asset in a strategy. We start by extracting executed orders from the transactions DataFrame and categorizes them as either "buy" or "sell" based on the sign of the quantity. It then groups these transactions by asset and sorts them by date. For each asset, the algorithm iterates through the sorted transactions to match an opening transaction (buy for long or sell for short) with its corresponding closing transaction (sell for long or buy for short). The time, price, and quantity of both the opening and closing transactions are recorded to compute the round-trip characteristics such as duration, profit and loss, and other relevant metrics.

Conceptually, a round trip is a complete cycle of opening and closing a position in a specific asset, and its analysis is crucial for understanding the effectiveness of our strategy. Metrics derived from round-trip analysis can provide valuable insights into transaction costs, holding periods, and the risk/return profile of individual trades, thereby aiding in strategy optimization and risk management.

Now that we covered how the plots are generated, let's review some of the key statistics output by Pyfolio:

- **Percent profitable**: Calculated by dividing the number of profitable round trips by the total number of round trips
- **Winning round_trips**: Calculated by counting the number of round-trip trades that resulted in a positive profit and loss
- **Losing round_trips**: Calculated by counting the number of round-trip trades that resulted in a negative profit and loss
- **Profit factor**: Calculated by dividing the sum of all profitable trades by the absolute sum of all losing trades
- **Avg. winning trade**: Computed by taking the mean of all profitable trades

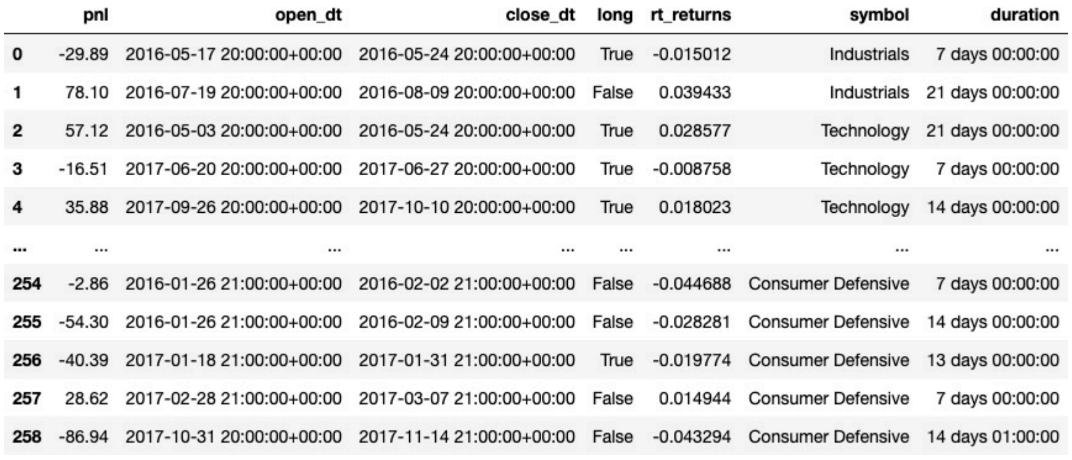
There's more…

Depending on our strategy, it's sometimes useful to aggregate the round trip performance statistics by section, instead of by asset. By passing in the **`sector_map`** dictionary we built in the *Preparing Zipline Reloaded backtest results for Pyfolio Reloaded* recipe, we can aggregate by sector.

1. Apply the sector mapping to the extracted round trips:

```
round_trips_by_sector = pf.round_trips.apply_sector_mappings_to_round_trips
    round_trips,
    sector_map
)
```

   The result is a pandas DataFrame similar to *Figure 9.28* except aggregated by sector, instead of asset symbol.



	pnl	open_dt	close_dt	long	rt_returns	symbol	duration
0	-29.89	2016-05-17 20:00:00+00:00	2016-05-24 20:00:00+00:00	True	-0.015012	Industrials	7 days 00:00:00
1	78.10	2016-07-19 20:00:00+00:00	2016-08-09 20:00:00+00:00	False	0.039433	Industrials	21 days 00:00:00
2	57.12	2016-05-03 20:00:00+00:00	2016-05-24 20:00:00+00:00	True	0.028577	Technology	21 days 00:00:00
3	-16.51	2017-06-20 20:00:00+00:00	2017-06-27 20:00:00+00:00	True	-0.008758	Technology	7 days 00:00:00
4	35.88	2017-09-26 20:00:00+00:00	2017-10-10 20:00:00+00:00	True	0.018023	Technology	14 days 00:00:00
...	...	...	...	...	...	...	...
254	-2.86	2016-01-26 21:00:00+00:00	2016-02-02 21:00:00+00:00	False	-0.044688	Consumer Defensive	7 days 00:00:00
255	-54.30	2016-01-26 21:00:00+00:00	2016-02-09 21:00:00+00:00	False	-0.028281	Consumer Defensive	14 days 00:00:00
256	-40.39	2017-01-18 21:00:00+00:00	2017-01-31 21:00:00+00:00	True	-0.019774	Consumer Defensive	13 days 00:00:00
257	28.62	2017-02-28 21:00:00+00:00	2017-03-07 21:00:00+00:00	False	0.014944	Consumer Defensive	7 days 00:00:00
258	-86.94	2017-10-31 20:00:00+00:00	2017-11-14 21:00:00+00:00	False	-0.043294	Consumer Defensive	14 days 01:00:00

Figure 9.31: DataFrame with round trips mapped to sector

2. Plot the duration of each round trip, per sector, over time:

```
pf.plotting.plot_round_trip_lifetimes(round_trips)
```

The result is a plot that visualizes the holding period for each sector traded.

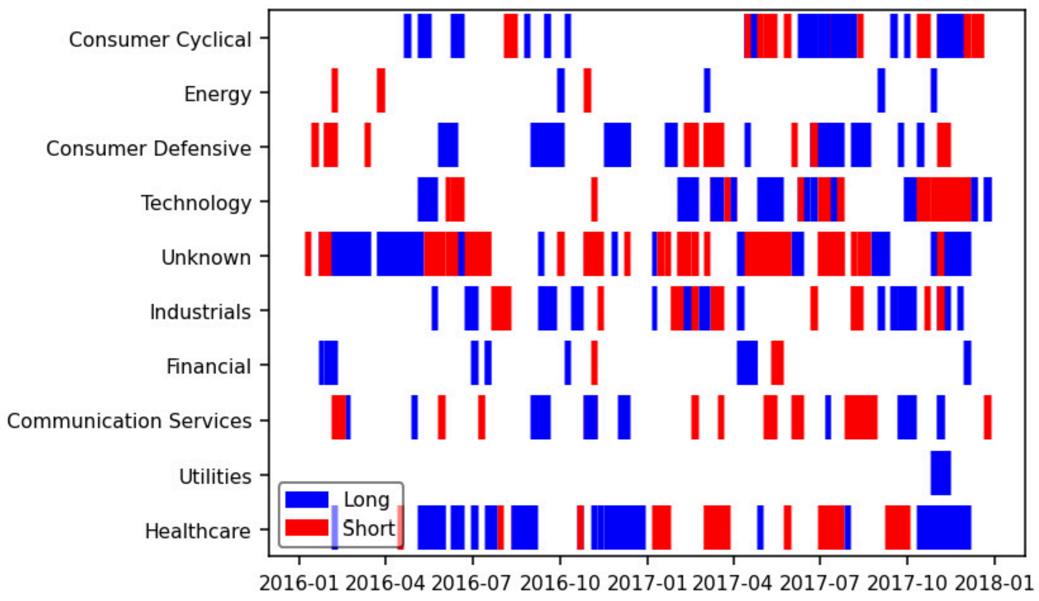


Figure 9.32: Plot visualizing the holding period for each sector traded

See also

As we've said, no single risk or performance metric is enough for a complete picture of your strategy. Pyfolio provides many important risk and performance statistics, but there are more. For a good walkthrough of other important risk and performance metrics, you can refer to blogs online.