

Machine-learning for Portfolio Management and Trading: An Introduction with Scikit-Learn

Sylvain Champonnois¹

September 29, 2025

¹sylvain.champonnois@gmail.com, latest version at <https://github.com/schampon/skfin>.

Contents

1	Introduction: Machine Learning Applications in Quantitative Portfolio Management	6
1.1	What hedge funds do	6
1.2	Overview	7
1.3	Quant workflow	8
1.4	MLOps for backtests	8
1.4.1	Pipelines	9
1.4.2	Scikit-learn	9
1.5	Python package	11
2	The Data Deluge: Alternative Data and Its Role in Modern Financial Analysis	12
2.1	Data deluge	12
2.1.1	Data scouting	13
2.1.2	ML research	14
2.2	Datasets	17
2.2.1	data helper functions	17
2.2.2	Ken French data: industry returns	23
2.2.3	Stock returns (2003-2007)	24
2.2.4	13F Berkshire Hathaway	25
2.2.5	FOMC Statements	25
2.2.6	Loughran-McDonalds sentiment dictionary and regulartory filing summaries	25
2.2.7	Goyal long-term market data	26
3	Mean-Variance Portfolio Optimization: Theory and Implementation	27
3.1	Markowitz portfolio optimisation	27
3.2	A shortcut to compute unconstrained mean-variance weights	29
3.3	Mean-variance estimators	30
3.4	Pnl metrics	33
4	Backtesting Industry Momentum Strategies	35
4.1	Industry momentum backtest	35
4.1.1	Industry data	36
4.1.2	Backtesting functions	37
4.1.3	Scikit-Learn TimeSeriesSplit	38
4.2	Empirical results	39
4.2.1	Cumulative pnl	39
4.2.2	Other backtest statistics	40
5	Enhancing Industry Momentum Strategies with Regularized Linear Models	43
5.1	Ridge / Lasso / Elastic net	43

5.2	Revisiting the industry backtest with linear models	44
5.2.1	Scikit-learn Pipeline and Multi-output	45
5.2.2	Linear Regression	46
5.2.3	Ridge	50
5.2.4	Ridge with feature expansion	51
6	Non-Linear Forecasting Models: Boosted Trees and Multi-Layer Perceptrons	53
6.1	Boosted Trees	53
6.2	Multi-layer perceptron	55
6.3	Predicting industry returns with non-linear models	55
6.3.1	Lightgbm	55
6.3.2	MLPRegressor	57
7	Balancing Overfitting and Underfitting: Cross-Validation and Parameter Optimization	63
7.1	Ridge CV	63
7.2	Random parameter search for Lightgbm	64
8	Eigenvalue Decomposition and Risk Bias in Portfolio Covariance Estimation	70
8.1	Risk bias metrics	70
8.2	Return covariance eigenvalues	71
8.3	The Marcenko-Pastur pdf	74
8.4	Understanding covariance shrinkage	75
9	Shrinkage Estimators and Covariance Matrix Regularization in Portfolio Risk Modeling	78
9.1	Risk in the industry momentum backtest	79
9.2	Risk model estimation	81
10	Understanding Portfolio Leverage: Techniques, Risks, and Optimization Constraints	85
10.1	Methods of Implementing Leverage	85
10.2	Leverage-aware optimisation	86
11	Non-Normality in Portfolio Returns: Implications for Risk and Performance Evaluation	91
11.1	Non-normality	91
11.2	The statistics of rolling sharpe ratio	93
12	Style Analysis and Factor Decomposition: Evaluating Industry Momentum and Berkshire Hathaway's Alpha	96
12.1	Style analysis	96
12.2	Industry momentum factor exposure	99
12.3	Residual pnl	100
13	Momentum-Based Factor Timing: Decomposition and Performance Analysis	103
14	Combining Multiple Strategies: Ensemble Methods in Portfolio Construction	107
14.1	Ensemble with scikit-learn	107
14.2	Rolling ensemble backtest	110
14.2.1	StackingBacktester	110
15	Incorporating Transaction Costs and Market Microstructure in Portfolio Optimization	117
15.1	Order book and trades	117
15.2	Impact costs	118
15.3	Mean-variance optimisation with quadratic costs	120

15.4 Positions and turnover	120
15.5 Cost for the Industry Momentum backtest	121
15.5.1 Turnover	122
15.5.2 Liquidity and market cap	122
15.5.3 Mark-to-market and backtesting	123
15.5.4 Backtesting with cost	126
15.5.5 Ridge backtest with cost	129
16 Data Challenges in Calculating Historical Stock Returns	132
16.1 Returns data	132
16.1.1 Scikit-learn returns	132
16.1.2 Yahoo finance returns	133
16.2 Survivorship bias and non-US firms	133
16.2.1 Dirty data and jumps	134
16.3 Matching to other datasets	135
17 Risk Management and Mean-Reversion Strategies in Hedge Funds: Lessons from the 2007 Quant Crisis	137
17.1 Liquidity and autocorrelation of stock returns	138
17.2 Sample	141
17.3 Mean-reversion strategy	141
17.4 Mean-reversion with risk-model shrinkage	144
17.5 Leverage of mean-reversion strategies	145
17.6 Concentrating the predictor instead of the positions	148
18 Trading Around Earnings Announcements: Volatility, Risk, and Regulatory Filings	150
18.1 Earnings announcement premium	150
18.2 Regulatory filings	151
18.3 Filing dates from 10-Ks/10-Qs	152
18.4 Stock returns on filing dates	154
19 Measuring Financial Sentiment in 10-K Filings: Dictionary and Model-Based Approaches	157
19.1 Rule-based sentiment	157
19.2 Learning-based sentiment	159
19.3 10-Ks	161
20 Out-of-Sample Predictability of the S&P 500: A Critical Assessment	166
20.1 Timing the market	166
20.2 Data	168
20.3 Timing backtest	169
20.4 Other timing backtest statistics	172
21 Politics and the stock market	176
21.1 The Presidential puzzle and the division of the government	176
21.2 Politics over century of data	177
22 Market Reactions to Scheduled Macroeconomic Announcements: Evidence from FOMC and Other Key Events	182
22.1 Macroeconomic events	182
22.2 FOMC dates	185
22.3 Returns on statement days	186

23 Text Representation and Sentiment Analysis of FOMC Statements: From Bag-of-Words to Deep Learning Embeddings	190
23.1 Loading the FOMC statements	190
23.2 Bag of words: Sentiment in FOMC statements	191
23.3 Sparse text representation: TFIDF	193
23.3.1 Principal component exploration	194
23.3.2 Clustering	196
23.3.3 Non-negative matrix factorization	197
23.4 Deep-learning embeddings: sentence transformers	199
23.5 UMAP	199
24 Supervised Learning on FOMC Statements: Predicting Rate Decisions and Market Reactions	201
24.1 Supervised learning: vector representation + Elastic net	201
24.1.1 comparison with sentence transformer embeddings	204
24.2 Sentiment in FOMC statements: supervised learning	205
25 Leveraging Large Language Models to Generate Economic Expectations from Historical News and Fed Statements	208
25.1 Generating estimates from news	208
25.2 LLM wrapper	211
25.3 Measuring hawkishness in FOMC statements	214
25.4 Explaining hawkishness	216
25.5 Topics in FOMC statements	218
25.6 Predicting returns with FOMC statements	220
26 Conclusion	225
27 Appendix: Custom Helper Functions for Data Exploration and Visualization	226
27.1 Data visualisation	226
27.2 Dates and mappings	230
27.3 Data utils	235
28 Appendix: Custom Helper Functions for Text Analysis	242
29 Appendix: A Structured Framework for Presenting Empirical Research Findings	246

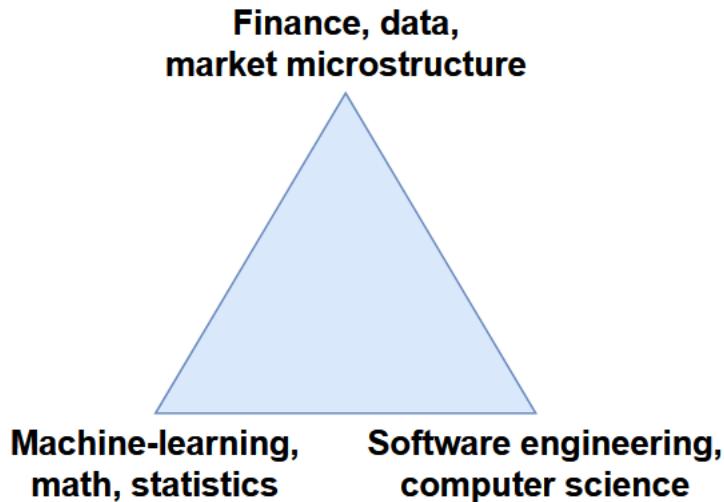
Chapter 1

Introduction: Machine Learning Applications in Quantitative Portfolio Management

1.1 What hedge funds do

This material is an introduction to using machine-learning for portfolio management and trading. Given the centrality of programming in hedge funds today, the concepts are exposed using only jupyter notebooks in python. Moreover, we leverage the `scikit-learn` package (also known as `sklearn`) to illustrate how machine-learning is used in practice in this context.

As shown by the illustration below, we cover here topics that span Finance (market microstructure, portfolio construction, data, etc), Machine-learning (including mathematics and statistics) and Software engineering/Computer science.

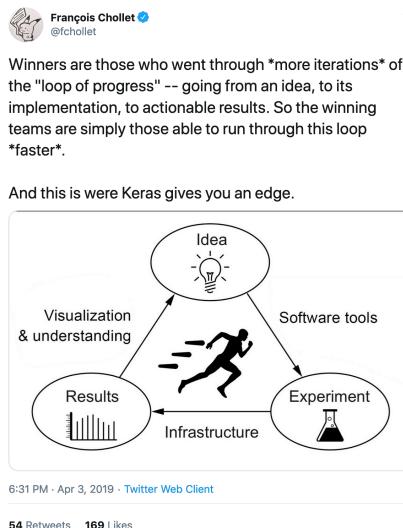


We are interested in how quantitative hedge funds operate in practice. A hedge fund is an asset management company that pools together capital from multiple investors to invest it in a portfolio of securities to maximize returns given a level of risk. Today, quantitative hedge funds are

essentially *consumers* of data – they ingest all sorts of datasets and extract information used to systematically buy or sell securities. Researchers and portfolio managers are deeply involved in the process of data ingestion and information extraction, but they do not directly decide which securities are bought or sold – instead algorithms do.

Because these processes of data ingestion and information extraction are so central to quantitative hedge fund operations, they have become software companies – a lot of the intellectual property (IP) of hedge funds is embedded in the code they write. And in that sense, hedge funds are not so different from other data-science based technology companies. (And in fact the hiring has become very similar, with a lot of interest in profiles out of Computer Science, Machine-Learning, Data engineering, Statistics, etc).

Another point is the Hedge Fund industry is very competitive and given that the frontier of knowledge is today moving very quickly (because of active research in Data, ML, hardware, software, etc) not keeping up with progress runs the risk of lagging by the competition. More precisely, innovation implies coming up with new ideas, testing them, deploying these ideas in trading systems; learning from them – and then deploying newer ideas after that. Of course, this kind of “loop of progress” (see [Francois Chollet \(04/03/2019\)](#)) is common to all AI-impacted industries.

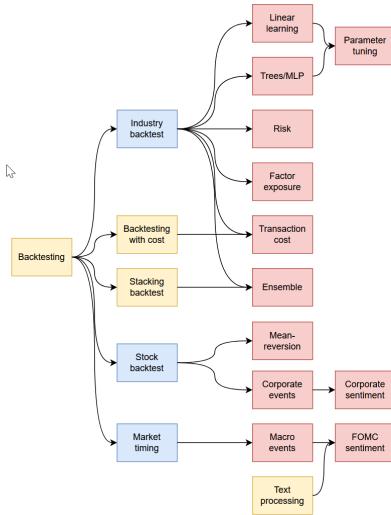


1.2 Overview

Across Jupyter notebooks, we cover three main empirical use-cases: timing industry returns; timing stock returns; and timing the market (ie. timing the Standard and Poor's 500 index). For each use-case, there is a set of toy-datasets (described in the Data section).

Moreover:

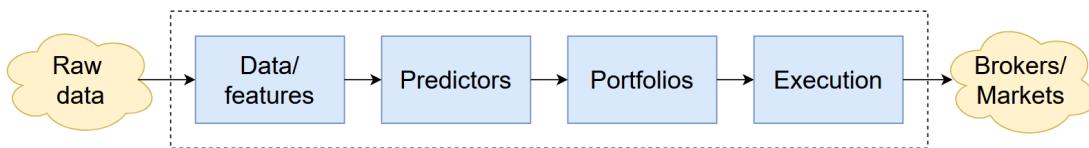
- for the industry return backtest, we illustrate several main concepts: learning (with linear models, boosted trees, and MLP neural nets); risk; factor exposures; transaction costs; ensemble;
- for the stock return backtest, we illustrate mean-reversion, trading around corporate events and sentiment (in earning calls);
- for the market timing backtest, we also illustrate training around macroeconomic events, and extracting sentiment in the statements of the Federal Open Market Committee (FOMC).



1.3 Quant workflow

The graph below shows the typical data workflow of a quantitative fund:

- Raw data is acquired generally by a Data team and possibly transformed into usable features.
- Data quality is verified, specifically ensuring that each observation's timestamp is valid and has not been altered in a manner that introduces information from the future.
- From these features, predictors of asset returns are derived.
- Given a single predictor (or a set of many predictors), portfolios are constructed: these portfolios represent the ideal positions of a fund given the asset forecasts, but also risk forecasts (and possibly, transaction cost forecasts).
- When these ideal positions change from one day to the next (because the underlying data has been updated), the difference in positions initiate trades that are then executed on asset exchanges or with brokers.



1.4 MLOps for backtests

MLOps (machine learning operations) represents a set of practices for the deployment of ML models in production. For quant hedge funds, there are two main concepts that we describe here:

- pipelines
- backtests

1.4.1 Pipelines

Pipeline:

A machine-learning pipeline is an end-to-end description of the automated flow of data from raw inputs to a desired output. Each step represents a transformation of the data, possibly with a fitted model.

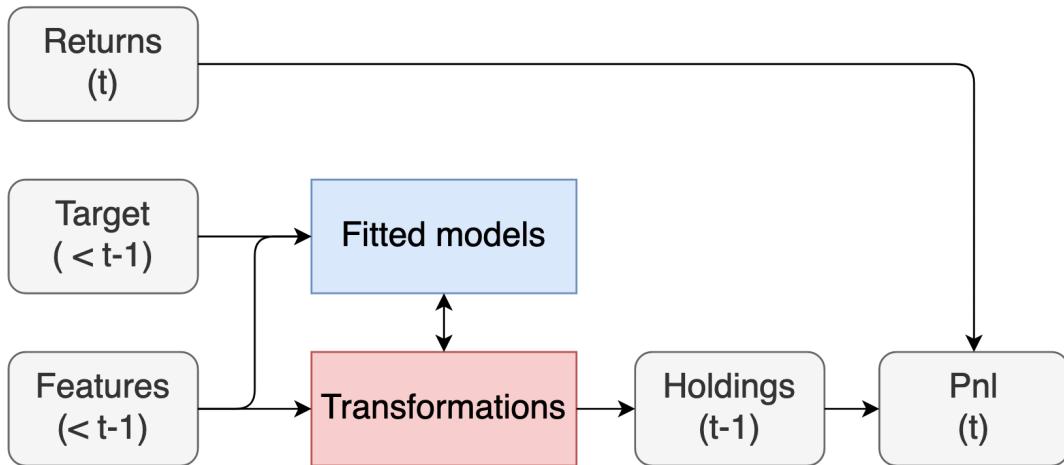
The diagram below illustrates a pipeline for a quant fund. The end point (to the right of the diagram) are the positions or holdings in a set of traded securities – and combined with the returns on these securities, the pnl of a given strategy can be computed. The entry point (to the left of the diagram) are features. A set of transformations (pre-determined in the pipeline) are applied to these features to produce the desired holdings. Some transformations in the pipeline are “fixed” while others depend on fitted models (e.g. a ML predictor of returns or a risk model).

In the diagram, we emphasize the timing of these different objects:

- for a pnl at time t , the features and target include only information up to $t - 1$ so that the holdings are known in $t - 1$ and can accrue returns over the period t .

The following equation summarizes this point:

$$pnl_t = holdings_{t-1} \times returns_t.$$



1.4.2 Scikit-learn

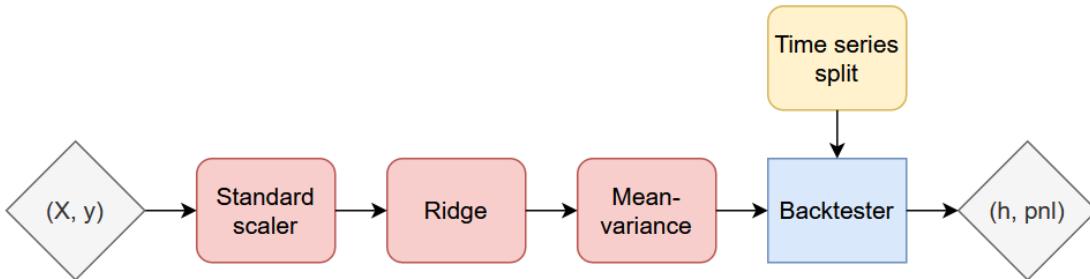
The following notebooks and notes are largely based on scikit-learn. scikit-learn is an extremely powerful (and widely used) package for machine-learning. In particular, it provides a “grammar” for pipelines where each transformation or estimator class has the `fit/transform/predict` methods with arguments as `(X, y)` where `X` represents the features and `y`, the targets.

A look-ahead bias occurs when data dated at t includes information only available after t ; in contrast, point-in-time data ensures that data dated at t is based on only information up to date t . A backtest is a method to simulate a strategy using point-in-time historical data and evaluate its profitability.

In order to illustrate how to use pipelines à la scikit-learn for quantitative portfolio management, we introduce in the next sections two objects:

- a Mean-variance estimator that computes positions from a predictor;
- a Backtester class that fits a scikit-learn estimator over rolling windows (as defined by the TimeSeriesSplit class).

The Backtester class runs the rolling window simulation so that only information up to date $t - 1$ is used to determine the holdings at that date. The following graph shows an example of a learning pipeline (with a StandardScaler and Ridge steps before compute the positions with the Mean-variance estimator) that we will use.



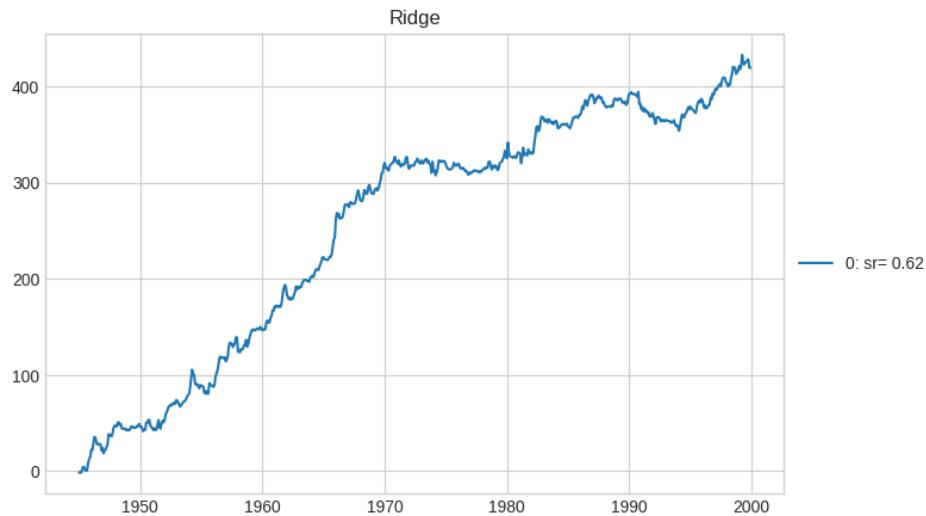
```
[8]: from skfin import Backtester, MeanVariance, Ridge
from skfin.datasets_ import load_kf_returns
from skfin.plot import line
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

estimator = make_pipeline(StandardScaler(with_mean=False), Ridge(), MeanVariance())

returns_data = load_kf_returns(cache_dir="data")
ret = returns_data["Monthly"]["Average_Value_Weighted_Returns"][:1999]

transform_X = lambda x: x.rolling(12).mean().fillna(0)
transform_y = lambda x: x.shift(-1)
features = transform_X(ret)
target = transform_y(ret)
```

```
[9]: pnl_ = Backtester(estimator).train(features, target, ret)
line(pnl_, cumsum=True, title="Ridge")
```



The graph above shows the annualized sharpe ratio (as `sr`): given that the strategy is monthly, this is the montly pnl average divided by the monthly pnl standard deviation and “annualized” by multiplying by the square-root of the number of observation per year ($\sqrt{12}$).

1.5 Python package

All the classes and functions used across notebooks are stored in .py files using the jupyter magic: `%%writefile`. And these files are structured as a repository on github that can be cloned from the command line (once in the correct directory):

```
> git clone https://github.com/schampon/skfin.git && cd skfin
```

The following script helps create an environment with the proper packages:

```
[10]: %%writefile ../create_env.sh
conda create python=3.9 --name skfin -c https://conda.anaconda.org/conda-forge/ -y
conda activate skfin

pip install -r requirements.txt
pip install -e .
python -m ipykernel install --user --name skfin --display-name "Python (skfin)"
```

Overwriting ../create_env.sh

In practice, this is done from the command line with:

```
./create_env.sh
```

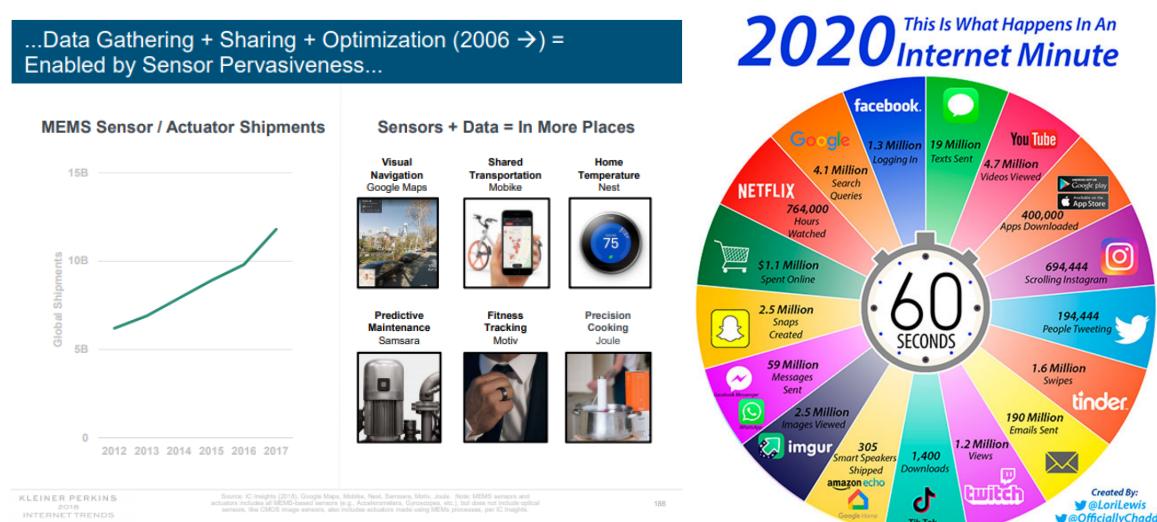
Chapter 2

The Data Deluge: Alternative Data and Its Role in Modern Financial Analysis

In the previous section, we described hedge funds, and more generally the buy-side industry, as consumers of data. In practice hedge funds have been a beneficiary of the exponential growth in the number of available datasets. We first describe the main classes of datasets that have been made available in the industry. We then introduce some of the illustrative toy datasets that will be used across notebooks.

2.1 Data deluge

There are now sensors everywhere in the physical world and most of the online interactions are tracked – leading a data deluge (e.g. see [Mary Meeker \(2018\)](#) on internet trends or [Lori Lewis \(2022\)](#)). On the one hand, the cost of sensors has dropped so much that these sensors can be put practically everywhere and record data from the physical world. On the other hand, the volume and velocity of online data only has also exploded.

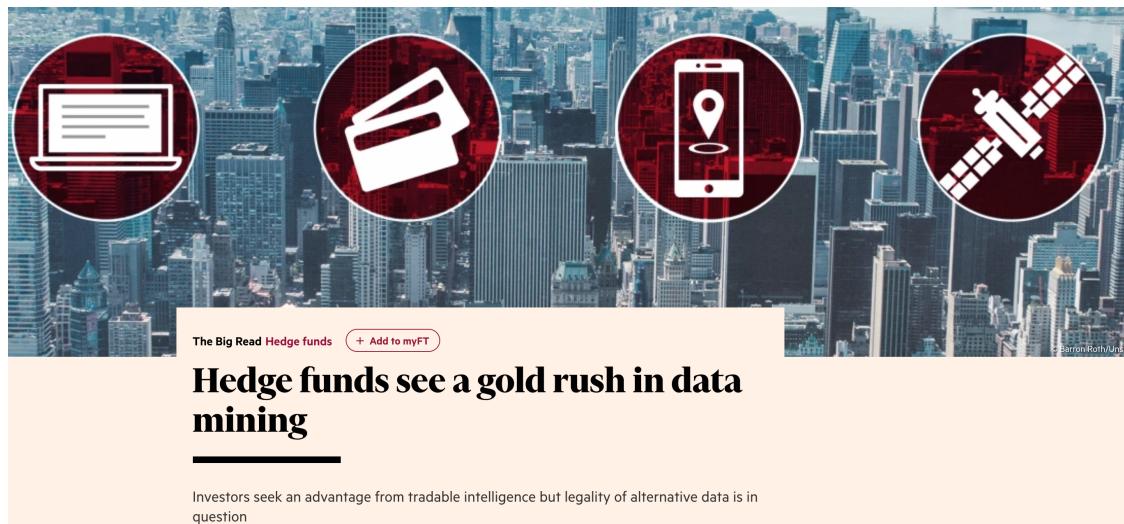


2.1.1 Data scouting

The article “Hedge funds see a gold rush in data mining” [FT \(08/28/2017\)](#) describes use-cases associated with four types of alternative data:

- web traffic
- credit card transaction
- geolocalisation
- satellite imaging

In each of these cases, the main idea is to use alternative data to get a noisy, but real-time forecast of company fundamentals (e.g. quarterly sales).



Obviously, predicting company fundamentals is only part of the story of what drives stock prices in financial markets. On the other side, understanding the market participants, in particular retail traders, is also important as shown in China [Bloomberg \(06/15/2019\)](#) or in the US with the Game Stop saga [FT\(01/29/2021\)](#).

[Menu](#) [Search](#)

Bloomberg

Sign In [Subscribe](#)

Economics

Quants Think Like Amateurs in World's Wildest Stock Market

Bloomberg News
15 May 2019, 17:00 BST
Updated on 16 May 2019, 03:39 BST

► Modeling the behavior of individual investors can be tricky
► BlackRock monitors about 100,000 online chat posts a day

Hedge funds rush to get to grips with retail message boards

Professional speculators start efforts to scrape data from Reddit to avoid assaults

The 4.5m-strong Reddit message board /r/WallStreetBets has drawn the attention of hedge funds © FT montage

[X](#) [f](#) [in](#) [Share](#) [Save](#)

Madison Darbyshire and Laurence Fletcher in London, Colby Smith and Michael Mackenzie in New York JANUARY 29 2021

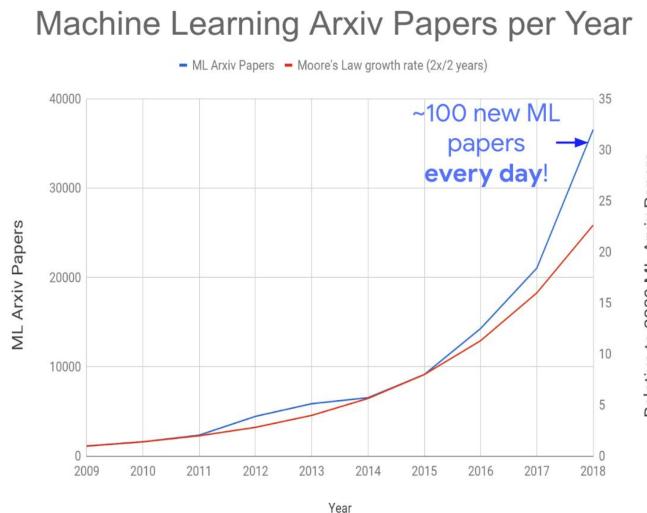
308 [Print](#)

Recent examples show even more sophistication and creativity for new data – e.g. FT (09/13/2024): “How SEC mobile phones can signal an imminent stock price drop.”

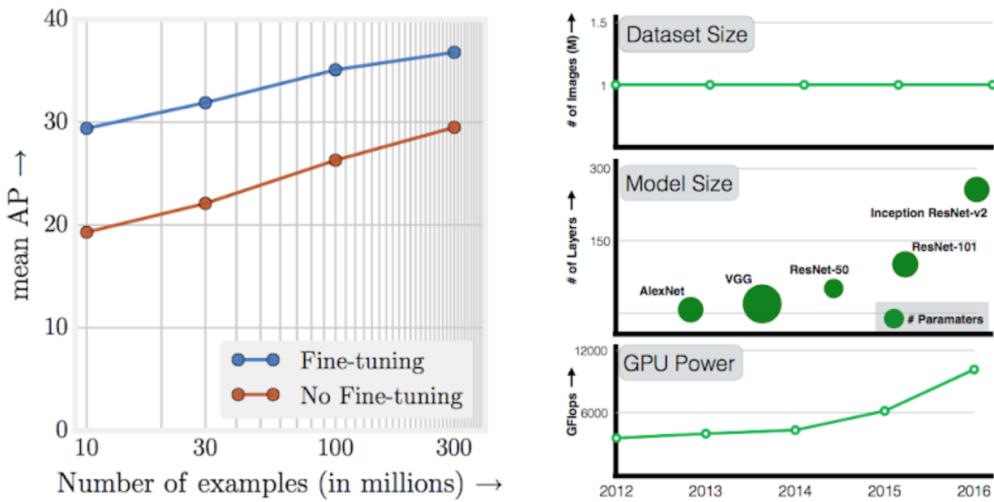


2.1.2 ML research

ML research has become a race with new ideas coming out with an increasing speed – e.g. as illustrated by the number of papers published on the scientific paper repository arxiv.com ([Jeff Dean \(06/02/2019\)](#)).



The success of deep-learning depends on: i) model capacity, ii) computational power, iii) dataset size. [Sun, Shrivastava, Singh, Gupta \(2017\)](#) note that the size of the largest dataset has remained somewhat constant over the last few years.



Here are several trends from the Stanford's AI index report:

- The training costs for cutting-edge AI models have skyrocketed, with OpenAI's GPT-4 and Google's Gemini Ultra costing \\$78m and \\$191m, respectively.
- AI projects on GitHub have surged from 845 in 2011 to about 1.8m in 2023, with a 59.3% increase in 2023 alone.
- As language models have rapidly advanced, surpassing human performance on numerous benchmarks, there has been a growing need for more robust and wide-ranging evaluation methods – with the Holistic Evaluation of Language Models (HELM) providing a comprehensive framework .
- Introduced by OpenAI researchers in 2021, HumanEval is a benchmark for assessing the coding abilities of AI systems. A variant of the GPT-4 model, AgentCoder, is currently leading in HumanEval performance.

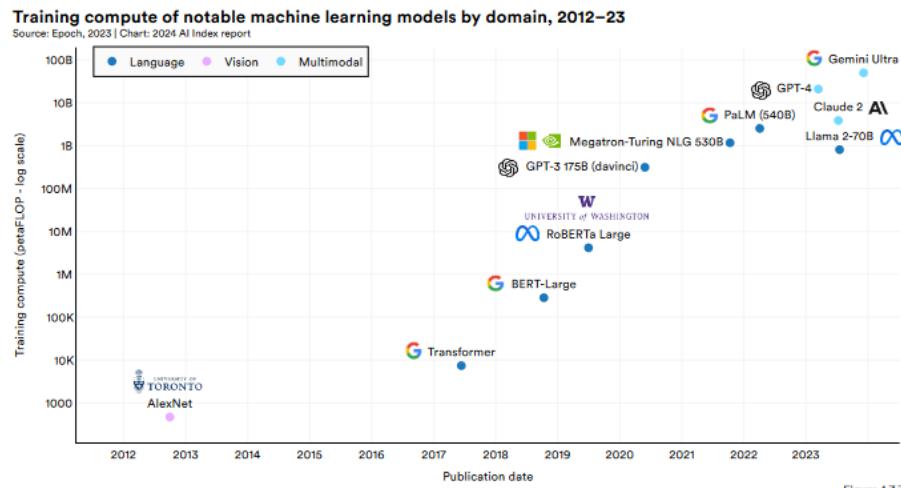


Figure 1.3.7

Estimated training cost of select AI models, 2016–23

Source: Epoch, 2023 | Chart: 2024 AI Index report

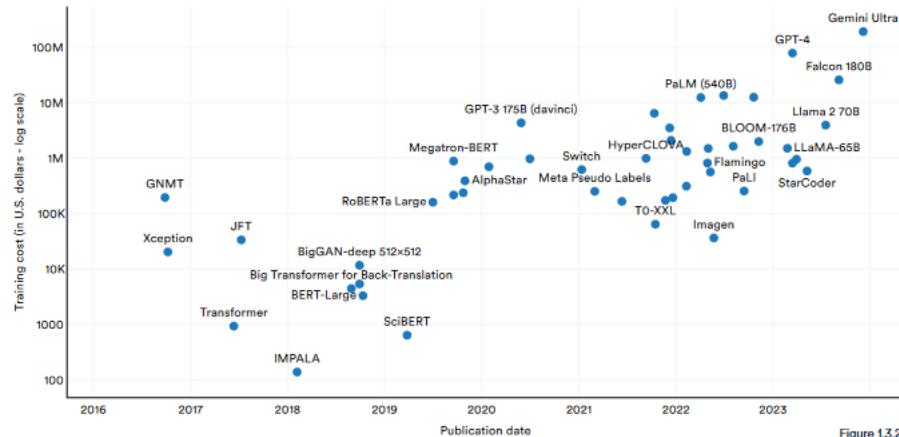


Figure 1.3.22

HELM: mean win rate

Source: CRFM, 2023 | Chart: 2024 AI Index report

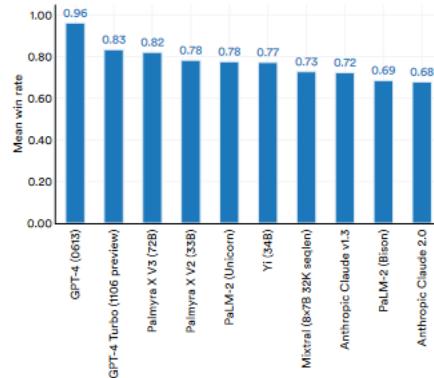


Figure 2.2.3

Leaders on individual HELM sub-benchmarks

Source: CRFM, 2023 | Table: 2024 AI Index report

Task	Leading model	Score
GSM8K - EM	GPT-4 (0613)	0.93
LegalBench - EM	GPT-4 (0613)	0.71
MATH - Equivalent (CoT)	GPT-4 Turbo (1106 preview)	0.86
MMLU - EM	GPT-4 (0613)	0.74
MedQA - EM	GPT-4 Turbo (1106 preview)	0.82
NarrativeQA - F1	Yi (34B)	0.78
NaturalQuestions (closed-book) - F1	Llama 2 (70B)	0.46
NaturalQuestions (open-book) - F1	PaLM-2 (Bison)	0.81
OpenbookQA - EM	GPT-4 (0613)	0.96
WMT 2014 - BLEU-4	Palmyra X V3 (72B)	0.26

Figure 2.2.4

Adoption of AI tools in development tasks, 2023

Source: Stack Overflow Developer Survey, 2023 | Chart: 2024 AI Index report

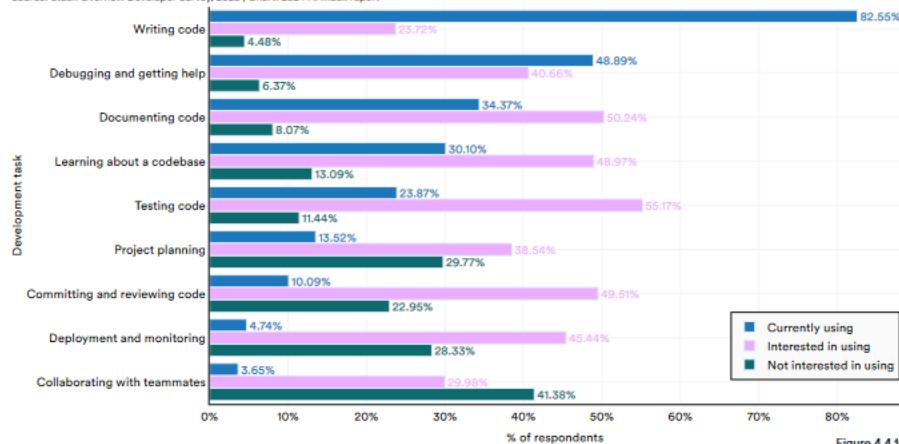


Figure 4.4.15

2.2 Datasets

In particular, we look at several sources of data:

- Ken French's Financial data library
- Berkshire Hathaway financials
- Stock returns
- Federal Open Market Committee (FOMC) statements
- the sentiment dictionary created by Loughran and McDonalds
- US filings to the U.S. Securities and Exchange Commission (SEC) called 10Qs and 10Ks
- Long term stock market data put together by Amit Goyal

2.2.1 data helper functions

In this section, we use the `jupyter` command `%%writefile` to construct a module of data helper function.

```
[12]: %%writefile ../skfin/dataloaders/loaders.py
import logging
import subprocess
import sys
from pathlib import Path
from typing import Dict

import pandas as pd
from tqdm.auto import tqdm

from skfin.dataloaders.cache import CacheManager
from skfin.dataloaders.web_utils import WebUtils
from skfin.dataloaders.cleaners import DataCleaner
from skfin.dataloaders.fomc import FomcUtils
from skfin.dataloaders.constants.mappings import symbol_dict

logging.basicConfig(stream=sys.stdout, level=logging.CRITICAL)
logger = logging.getLogger(__name__)

class DatasetLoader:
    """Class for loading various financial datasets."""

    def __init__(self, cache_dir: str = "data"):
        self.cache_manager = CacheManager(cache_dir)
        self.logger = logging.getLogger(__name__)

    def load_kf_returns(
        self, filename: str = "12_Industry_Portfolios", force_reload: bool = False
    ) -> Dict:
        """
        Load Ken French return data.

        Args:
            filename: Name of the data file to load
            force_reload: If True, ignore cache and reload data

        Returns:
        """


```

```

    Dictionary of return data
"""

if filename == "12_Industry_Portfolios":
    skiprows, multi_df = 11, True
elif filename == "F-F_Research_Data_Factors":
    skiprows, multi_df = 3, False
elif filename == "F-F_Momentum_Factor":
    skiprows, multi_df = 13, False
elif filename == "F-F_Research_Data_Factors_daily":
    skiprows, multi_df = 4, False
else:
    skiprows, multi_df = 11, True

def loader_func():
    path = (
        "http://mba.tuck.dartmouth.edu/pages/faculty/ken.french/ftp/"
        + filename
        + "_CSV.zip"
    )
    files = WebUtils.download_zip_content(path)

    df = pd.read_csv(
        files.open(filename + ".csv"), skiprows=skiprows, index_col=0
    )
    if "daily" in filename:
        return {
            "Daily": df.iloc[:-1].pipe(
                lambda x: x.set_index(pd.to_datetime(x.index))
            )
        }
    else:
        return DataCleaner.clean_kf_dataframes(df, multi_df=multi_df)

    return self.cache_manager.get_cached_dataframe(
        filename=Path(filename), loader_func=loader_func, force_reload=force_reload
    )

def load_buffets_data(self, force_reload: bool = False) -> pd.DataFrame:
    """
Load Buffett's portfolio data.

Args:
    force_reload: If True, ignore cache and reload data

Returns:
    DataFrame containing Buffett's portfolio data
"""

    def loader_func():
        path = "https://github.com/slihn/buffetts_alpha_R/archive/master.zip"
        files = WebUtils.download_zip_content(path)

        df = pd.read_csv(
            files.open("buffetts_alpha_R-master/ffdata_brk13f.csv"), index_col=0
        )
        df.index = pd.to_datetime(df.index, format="%m/%d/%Y")
        return df

    return self.cache_manager.get_cached_dataframe(
        filename=Path("ffdata_brk13f.parquet"),
        loader_func=loader_func,

```

```

        force_reload=force_reload,
    )

def load_sklearn_stock_returns(self, force_reload: bool = False) -> pd.DataFrame:
    """
    Load stock returns data from scikit-learn.

    Args:
        force_reload: If True, ignore cache and reload data

    Returns:
        DataFrame containing stock returns
    """

    def loader_func():
        url = "https://raw.githubusercontent.com/scikit-learn/examples-data/master/" \
              "financial-data"
        df = (
            pd.concat(
                {
                    c: pd.read_csv(f"{url}/{c}.csv", index_col=0, parse_dates=True)[
                        "close"
                    ].diff()
                    for c in symbol_dict.keys()
                },
                axis=1,
            )
            .asfreq("B")
            .iloc[1:]
        )
        return df

    return self.cache_manager.get_cached_dataframe(
        filename=Path("sklearn_returns.parquet"),
        loader_func=loader_func,
        force_reload=force_reload,
    )

def load_fomc_statements(
    self,
    add_url: bool = True,
    force_reload: bool = False,
    progress_bar: bool = False,
    from_year: int = 1999,
) -> pd.DataFrame:
    """
    Load FOMC statements.

    Args:
        add_url: If True, adds URLs to the output
        force_reload: If True, ignore cache and reload data
        progress_bar: If True, displays a progress bar during loading
        from_year: Year from which to load data

    Returns:
        DataFrame containing FOMC statements
    """

    def loader_func():
        urls = FomcUtils.get_fomc_urls(from_year=from_year)
        if progress_bar:

```

```

        urls_ = tqdm(urls)
    else:
        urls_ = urls

    corpus = [
        DataCleaner.bs_cleaner(
            WebUtils.parse_html(WebUtils.get_response(url).text)
        )
        for url in urls_
    ]

    statements = FomcUtils.feature_extraction(corpus).set_index("release_date")
    if add_url:
        statements = statements.assign(url=urls)
    return statements.sort_index()

    return self.cache_manager.get_cached_dataframe(
        filename=Path("fomc_statements.parquet"),
        loader_func=loader_func,
        force_reload=force_reload,
    )

def load_loughran_mcdonald_dictionary(
    self, force_reload: bool = False
) -> pd.DataFrame:
    """
    Load the Loughran-McDonald dictionary.

    Args:
        force_reload: If True, ignore cache and reload data

    Returns:
        DataFrame containing the dictionary data
    """
    filename = Path("Loughran-McDonald_MasterDictionary_1993-2021.csv")

    def loader_func():
        id = "17CmUZM9h8UDGYjCXcjQLyybjTrcjrhik"
        url = f"https://docs.google.com/uc?export=download&confirm=t&id={id}"
        subprocess.run(
            f"wget -O '{self.cache_manager.cache_dir / filename}' '{url}'",
            shell=True,
            capture_output=True,
        )
        return pd.read_csv(self.cache_manager.cache_dir / filename)

    return self.cache_manager.get_cached_dataframe(
        filename=filename, loader_func=loader_func, force_reload=force_reload
    )

def load_10X_summaries(self, force_reload: bool = False) -> pd.DataFrame:
    """
    Load 10-X summaries.

    Args:
        force_reload: If True, ignore cache and reload data

    Returns:
        DataFrame containing 10-X summaries
    """
    filename = Path("Loughran-McDonald_10X_Summaries_1993-2021.csv")

```

```

def loader_func():
    id = "1CUzLrwQSZ4aUTfPB9EkRtZ48gPwbCOHA"
    url = f"https://docs.google.com/uc?export=download&confirm=t&id={id}"
    subprocess.run(
        f"wget -O '{self.cache_manager.cache_dir / filename}' '{url}'",
        shell=True,
        capture_output=True,
    )

    return pd.read_csv(self.cache_manager.cache_dir / filename)

df = self.cache_manager.get_cached_dataframe(
    filename=filename, loader_func=loader_func, force_reload=force_reload
)
return df.assign(
    date=lambda x: pd.to_datetime(x.FILING_DATE, format="%Y%m%d")
).set_index("date")

def load_ag_features(
    self, sheet_name: str = "Monthly", force_reload: bool = False
) -> pd.DataFrame:
    """
    Load Amit Goyal's characteristics data.

    Args:
        sheet_name: Name of the sheet to load
        force_reload: If True, ignore cache and reload data

    Returns:
        DataFrame containing characteristic data
    """
    filename = Path("Data2024.xlsx")

    def loader_func():
        id = "10_nk0kJPvq4eZgNl-1ys63PzhbnM3S2y"
        url = f"https://docs.google.com/spreadsheets/d/{id}/export?format=xlsx"
        subprocess.run(
            f"wget -O '{self.cache_manager.cache_dir / filename}' '{url}'",
            shell=True,
            capture_output=True,
        )

        return pd.read_excel(
            self.cache_manager.cache_dir / filename, sheet_name=sheet_name
        )

    df = self.cache_manager.get_cached_dataframe(
        filename=filename,
        loader_func=loader_func,
        force_reload=force_reload,
        sheet_name=sheet_name,
    )
    return df.assign(
        date=lambda x: pd.to_datetime(x.yyyymm, format="%Y%m")
    ).set_index("date")

```

Overwriting ../skfin/dataloaders/loaders.py

```
[13]: %%writefile ../skfin/datasets_.py
import warnings
```

```

import sys
import logging
from typing import Dict, Optional

import pandas as pd

from skfin.dataloaders import DatasetLoader

warnings.filterwarnings("ignore", category=UserWarning, module="openpyxl")

logging.basicConfig(stream=sys.stdout, level=logging.INFO)
logger = logging.getLogger(__name__)

def load_kf_returns(
    filename: str = "12_Industry_Portfolios",
    force_reload: bool = False,
    cache_dir: Optional[str] = "data",
) -> Dict:
    """Load Ken French return data."""
    loader = DatasetLoader(cache_dir=cache_dir)
    return loader.load_kf_returns(filename, force_reload)

def load_buffets_data(
    force_reload: bool = False, cache_dir: Optional[str] = "data"
) -> pd.DataFrame:
    """Load Buffett's portfolio data."""
    loader = DatasetLoader(cache_dir=cache_dir)
    return loader.load_buffets_data(force_reload)

def load_sklearn_stock_returns(
    force_reload: bool = False, cache_dir: Optional[str] = "data"
) -> pd.DataFrame:
    """Load stock returns data from scikit-learn."""
    loader = DatasetLoader(cache_dir=cache_dir)
    return loader.load_sklearn_stock_returns(force_reload)

def load_fomc_statements(
    add_url: bool = True,
    force_reload: bool = False,
    progress_bar: bool = False,
    from_year: int = 1999,
    cache_dir: Optional[str] = "data",
) -> pd.DataFrame:
    """Load FOMC statements."""
    loader = DatasetLoader(cache_dir=cache_dir)
    return loader.load_fomc_statements(add_url, force_reload, progress_bar, from_year)

def load_loughran_mcdonald_dictionary(
    force_reload: bool = False, cache_dir: Optional[str] = "data"
) -> pd.DataFrame:
    """Load the Loughran-McDonald dictionary."""
    loader = DatasetLoader(cache_dir=cache_dir)
    return loader.load_loughran_mcdonald_dictionary(force_reload)

def load_10X_summaries()

```

```

        force_reload: bool = False, cache_dir: Optional[str] = "data"
    ) -> pd.DataFrame:
        """Load 10-X summaries."""
        loader = DatasetLoader(cache_dir=cache_dir)
        return loader.load_10X_summaries(force_reload)

    def load_ag_features(
        sheet_name: str = "Monthly",
        force_reload: bool = False,
        cache_dir: Optional[str] = "data",
    ) -> pd.DataFrame:
        """Load Amit Goyal's characteristics data."""
        loader = DatasetLoader(cache_dir=cache_dir)
        return loader.load_ag_features(sheet_name, force_reload)

```

Overwriting ../skfin/datasets_.py

2.2.2 Ken French data: industry returns

```
[14]: from skfin.datasets import (
    load_buffets_data,
    load_kf_returns,
    load_sklearn_stock_returns,
)

import logging
logging.getLogger("skfin.dataloaders.cache").setLevel(level=logging.INFO)
```



```
[15]: %%time
returns_data = load_kf_returns(filename="12_Industry_Portfolios", force_reload=True)
```

INFO:skfin.dataloaders.cache:Loading from an external source
 INFO:skfin.dataloaders.cache:Saving to the cache : data/12_Industry_Portfolios
 CPU times: user 74.6 ms, sys: 9.8 ms, total: 84.4 ms
 Wall time: 792 ms

Reloading from a cache directory is faster!


```
[16]: %%time
returns_data = load_kf_returns(filename="12_Industry_Portfolios", force_reload=False)
```

INFO:skfin.dataloaders.cache:Loading from cache : data/12_Industry_Portfolios
 CPU times: user 29 ms, sys: 11.3 ms, total: 40.3 ms
 Wall time: 34.2 ms


```
[17]: returns_data_SMB_HML = load_kf_returns(filename="F-F_Research_Data_Factors")
```

INFO:skfin.dataloaders.cache:Loading from cache : data/F-F_Research_Data_Factors


```
[18]: returns_data_MOM = load_kf_returns(filename="F-F_Momentum_Factor")
```

INFO:skfin.dataloaders.cache:Loading from cache : data/F-F_Momentum_Factor


```
[19]: returns_data_DAILY = load_kf_returns(
    filename="F-F_Research_Data_Factors_daily", force_reload=True
)["Daily"]
```

```
INFO:skfin.dataloaders.cache:Loading from an external source
INFO:skfin.dataloaders.cache:Saving to the cache :
data/F-F_Research_Data_Factors_daily
```

2.2.3 Stock returns (2003-2007)

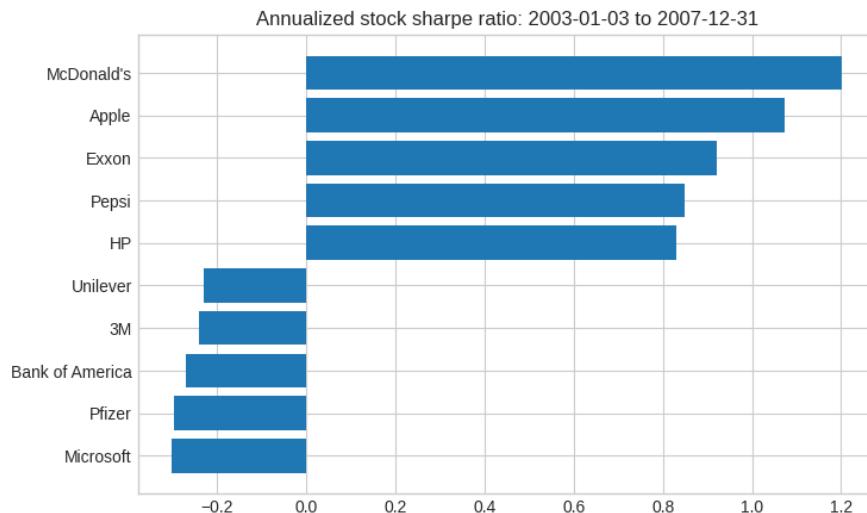
```
[20]: %%time
returns_data = load_sklearn_stock_returns(force_reload=True)

INFO:skfin.dataloaders.cache:Loading from an external source
INFO:skfin.dataloaders.cache:Saving to the cache : data/sklearn_returns.parquet
CPU times: user 715 ms, sys: 35.5 ms, total: 750 ms
Wall time: 10.8 s

[21]: from skfin.dataloaders.constants.mappings import symbol_dict
from skfin.metrics import sharpe_ratio
from skfin.plot import bar

[22]: start_date, end_date = (
    returns_data.index[0].strftime("%Y-%m-%d"),
    returns_data.index[-1].strftime("%Y-%m-%d"),
)

df = (
    returns_data.pipe(sharpe_ratio)
    .rename(index=symbol_dict)
    .sort_values()
    .pipe(lambda x: pd.concat([x.head(), x.tail()]))
)
bar(
    df,
    horizontal=True,
    title=f"Annualized stock sharpe ratio: {start_date} to {end_date}",
)
```



2.2.4 13F Berkshire Hathaway

```
[23]: %%time  
df = load_buffets_data(force_reload=True)
```

```
INFO:skfin.dataloaders.cache:Loading from an external source  
INFO:skfin.dataloaders.cache:Saving to the cache : data/ffdata_brk13f.parquet  
CPU times: user 32.4 ms, sys: 771 µs, total: 33.2 ms  
Wall time: 679 ms
```

2.2.5 FOMC Statements

```
[24]: from skfin.datasets_ import load_fomc_statements
```

```
[25]: %%time  
statements = load_fomc_statements(force_reload=True)
```

```
INFO:skfin.dataloaders.cache:Loading from an external source  
INFO:skfin.dataloaders.cache:Saving to the cache : data/fomc_statements.parquet  
CPU times: user 12.1 s, sys: 194 ms, total: 12.3 s  
Wall time: 1min 21s
```

```
[26]: %%time  
statements = load_fomc_statements(force_reload=False)
```

```
INFO:skfin.dataloaders.cache:Loading from cache : data/fomc_statements.parquet  
CPU times: user 3.03 ms, sys: 6.75 ms, total: 9.78 ms  
Wall time: 10.9 ms
```

2.2.6 Loughran-McDonalds sentiment dictionary and regulartory filing summaries

```
[27]: from skfin.datasets_ import load_loughran_mcdonald_dictionary, load_10X_summaries
```

```
[28]: %%time  
filing_summaries = load_10X_summaries(force_reload=False)
```

```
INFO:skfin.dataloaders.cache:Loading from cache : data/Loughran-  
McDonald_10X_Summaries_1993-2021.csv  
CPU times: user 2.66 s, sys: 931 ms, total: 3.59 s  
Wall time: 4.23 s
```

```
[29]: %%time  
lm = load_loughran_mcdonald_dictionary(force_reload=False)
```

```
INFO:skfin.dataloaders.cache:Loading from cache : data/Loughran-  
McDonald_MasterDictionary_1993-2021.csv  
CPU times: user 119 ms, sys: 9.06 ms, total: 128 ms  
Wall time: 154 ms
```

2.2.7 Goyal long-term market data

```
[30]: from skfin.datasets_ import load_ag_features
```

```
[31]: %%time  
ag = load_ag_features(force_reload=False);
```

```
INFO:skfin.dataloaders.cache:Loading from cache : data/Data2024.xlsx
```

```
ReadMe
```

```
Annual
```

```
Monthly
```

```
Quarterly
```

```
CPU times: user 817 ms, sys: 23.7 ms, total: 841 ms
```

```
Wall time: 1.07 s
```

Chapter 3

Mean-Variance Portfolio Optimization: Theory and Implementation

A portfolio is the ownership of a set of securities that give rights to a stream of payments. The goal of Markowitz portfolio optimisation is to determine the portfolio holdings that maximise the expected return of the portfolio under a risk constraint. Mathematically, it boils down to maximising a “mean-variance objective”. In this section, we review the mathematics of Markowitz portfolio optimisation and how to implement in python “mean-variance optimisers” that follow the scikit-learn “fit/predict” API.

3.1 Markowitz portfolio optimisation

The portfolio universe is made of N assets:

- r is a vector of returns for each asset (with size N);
- α is the asset return forecast: $\alpha = E(r)$;
- V is the return covariance matrix that measures that correlation between each asset: $V = \text{Var}(r)$;
- h is a vector of positions/holdings in each of these assets.

Note: for a vector v , we denote v^T as the transpose of v .

A mean-variance objective is an objective that trades off the portfolio expected return (as $h^T\alpha$) against the portfolio expected risk (as $h^T V h$). In practice, it is written as:

$$U = h^T \alpha - \frac{h^T V h}{2\lambda},$$

where λ is the risk-tolerance.

Lemma [mean-variance]: the allocation h that maximizes the mean-variance objective is

$$h = \lambda V^{-1} \alpha.$$

The ex-ante risk is $h^T V h = \lambda^2 \alpha^T V^{-1} \alpha$ and the ex-ante Sharpe ratio is

$$S = \frac{h^T E(r)}{\sqrt{h^T V h}} = \sqrt{\alpha^T V^{-1} \alpha}.$$

Corollary: The maximisation of the sharpe ratio is equivalent (up to a scaling factor) the mean-variance optimisation.

The mean-variance formula is extended to account for the linear constraints

$$Ah = b.$$

To do so, we introduce the Lagrangian \mathcal{L} (and Lagrange multiplier ξ)

$$\mathcal{L} = h^T \alpha - \frac{h^T V h}{2\lambda} - (h^T A^T - b^T) \xi$$

The Lagrange multiplier ξ is a tuning parameter chosen exactly so that the constraint above holds. At the optimal value of ξ , the constrained problem boils down to an unconstrained problem with the adjusted return forecast $\alpha - A^T \xi$.

Lemma: the allocation that maximizes the objective $h^T \alpha - \frac{h^T V h}{2\lambda}$ under the linear constraint $Ah = b$ is

$$h = V^{-1} A^T \left(A V^{-1} A^T \right)^{-1} b + \lambda V^{-1} \left[\alpha - A^T \left(A V^{-1} A^T \right)^{-1} A V^{-1} \alpha \right]$$

Proof: the first-order condition is

$$\frac{\partial \mathcal{L}}{\partial h} = \alpha - \frac{Vh}{\lambda} - A^T \xi = 0 \Leftrightarrow h = \lambda V^{-1} [\alpha - A^T \xi]$$

The parameter ξ is chosen so that $Ah = b$

$$b = Ah = \lambda A V^{-1} [\alpha - A^T \xi] \Rightarrow \xi = [A V^{-1} A^T]^{-1} \left[A V^{-1} \alpha - \frac{b}{\lambda} \right]$$

The holding vector under constraint is

$$h_\lambda = \underbrace{V^{-1} A^T \left(A V^{-1} A^T \right)^{-1} b}_{\text{minimum variance portfolio}} + \underbrace{\lambda V^{-1} \left[\alpha - A^T \left(A V^{-1} A^T \right)^{-1} A V^{-1} \alpha \right]}_{\text{speculative portfolio}}$$

- The first term is what minimises the risk $h^T V h$ under the constraint $Ah = b$ (in particular, it does not depend on expected returns or risk-tolerance).
- The second term is the speculative portfolio (it is sensitive to both inputs).

The efficient frontier is the relation between expected portfolio return $h^T \alpha$ and portfolio standard deviation $\sqrt{h^T V h}$ for varying level of risk-tolerance

$$(x, y) \mapsto \left(h_\lambda^T \alpha, \sqrt{h_\lambda^T V h_\lambda} \right)$$

When $b = 0$, the efficient frontier between $h_\lambda^T \alpha$ and $\sqrt{h_\lambda^T V h_\lambda}$ is a line through $(0, 0)$; otherwise, it is a parabolic curve.

We focus on pure “alpha views” – that is, long-short “cash-neutral” portfolios where the sum of holdings is zero. In this case $b = 0$ and $A = \mathbf{1}$ where

$$\mathbf{1} = [\ 1 \ \dots \ 1] .$$

3.2 A shortcut to compute unconstrained mean-variance weights

THE JOURNAL OF FINANCE • VOL. LIV, NO. 2 • APRIL 1999

The Sampling Error in Estimates of Mean-Variance Efficient Portfolio Weights

MARK BRITTON-JONES*

ABSTRACT

This paper presents an exact finite-sample statistical procedure for testing hypotheses about the weights of mean-variance efficient portfolios. The estimation and inference procedures on efficient portfolio weights are performed in the same way as for the coefficients in an OLS regression. OLS t - and F -statistics can be used for tests on efficient weights, and when returns are multivariate normal, these statistics have exact t and F distributions in a finite sample. Using 20 years of data on 11 country stock indexes, we find that the sampling error in estimates of the weights of a global efficient portfolio is large.

In this section, we derive a trick to compute unconstrained mean-variance weights just with the return of different assets. More precisely:

- X : daily return of K assets over T days – so that the shape of X is $[T \times K]$.
- y : vector of ones of size T .

Lemma [Mark Britten-Jones]: the markowitz weights of are proportional to the slope coefficient of a regression of the vector of ones y on the pnls X *with no intercept*.

Proof: the coefficient of the regression with no intercept is given by

$$b = (X^T X)^{-1} X^T y.$$

The mean of the pnls is given by $\mu = \frac{1}{T} X^T y$. The variance of the pnls is $V = \frac{1}{T} X^T X - \mu \mu^T$

Using the Woodbury identity (https://en.wikipedia.org/wiki/Woodbury_matrix_identity), we have:

$$b = (V + \mu \mu^T)^{-1} \mu = \left[V^{-1} - \frac{V^{-1} \mu \mu^T V^{-1}}{1 + \mu^T V^{-1} \mu} \right] \mu = \frac{V^{-1} \mu}{1 + \mu^T V^{-1} \mu}.$$

This implies that the OLS slope coefficient b is proportional to the mean-variance holdings $V^{-1} \mu$.

3.3 Mean-variance estimators

In the following python file, we introduce several functions:

- a function that computes mean-variance holdings for batches
- a MeanVariance class that follows the sklearn api
- a Mbj class that computes unconstrained mean-variance weights with the Britten-Jones (1999) trick.

```
[3]: %%writefile ../skfin/mv_estimators.py
import numpy as np
import pandas as pd
from skfin.metrics import sharpe_ratio
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.linear_model import LinearRegression

from dataclasses import dataclass, field
from typing import Callable, Optional


def compute_batch_holdings(pred, V, A=None, risk_target=None):
    """
    compute markowitz holdings with return prediction "mu" and covariance matrix "V"

    Args:
        pred: (numpy.ndarray or pandas.Series or pandas.DataFrame): Expected returns, can be of shape (N,) or (N, K).
        V: (numpy.ndarray): Covariance matrix of shape (N, N).
        A: (numpy.ndarray, optional): Matrix for linear constraints, default is None
        past_h (numpy.ndarray, optional): Not used in the current implementation.
        constant_risk (bool, optional): If True, normalize outputs to maintain constant risk, default is False.

    Returns:
        numpy.ndarray: Computed holdings.
    """

    N, _ = V.shape
    if isinstance(pred, (pd.Series, pd.DataFrame)):
        pred = pred.values
    if pred.ndim == 1:
        pred = pred[:, np.newaxis]
    elif pred.shape[1] == N:
        pred = pred.T

    invV = np.linalg.inv(V)
    if A is None:
        M = invV
    else:
        U = invV.dot(A)
        if A.ndim == 1:
            M = invV - np.outer(U, U.T) / U.dot(A)
        else:
            M = invV - U.dot(np.linalg.inv(U.T.dot(A)).dot(U.T))
    h = M.dot(pred)
    if risk_target is not None:
        h = risk_target * h / np.sqrt(np.diag(h.T.dot(V.dot(h))))
    return h.T
```

```

@dataclass
class MeanVariance(BaseEstimator):
    """
    A mean-variance optimization estimator that computes portfolio holdings
    based on expected returns and the covariance matrix.

    Attributes:
        transform_V (Callable): Function to transform target variable 'y' into a covariance
        ↪matrix.
        A (Optional[np.ndarray]): Constraints matrix for the optimization problem.
        risk_target (float): Risk target for the portfolio.
    """

    transform_V: Callable = field(default=lambda x: np.cov(x.T))
    A: Optional[np.ndarray] = None
    risk_target: float = 1.0

    def __post_init__(self):
        """
        Post-initialization process to set additional attributes or setup.
        """
        self.holdings_kwargs = {'risk_target': self.risk_target}

    @staticmethod
    def compute_batch_holdings(pred, V, A, risk_target, **kwargs):
        """
        Compute portfolio holdings in a batch manner.

        Parameters:
            pred (np.ndarray): Predicted returns.
            V (np.ndarray): Covariance matrix.
            A (np.ndarray): Constraint matrix.
            risk_target (float): Target risk level.
            **kwargs: Additional keyword arguments.

        Returns:
            np.ndarray: Portfolio holdings.
        """
        return compute_batch_holdings(pred=pred, V=V, A=A, **kwargs)

    def fit(self, X, y=None):
        """
        Fit the model by calculating the covariance matrix 'V_' from targets 'y'.

        Parameters:
            X (np.ndarray): Input feature matrix.
            y (np.ndarray): Target variable matrix.
        """
        self.V_ = self.transform_V(y)

    def predict(self, X, **kwargs):
        """
        Predict portfolio holdings based on input features.

        Parameters:
            X (np.ndarray): Input feature matrix.
            **kwargs: Additional keyword arguments.

        Returns:
            np.ndarray: Predicted portfolio holdings.
        """

```

```

    """
A = self.A if self.A is not None else np.ones(X.shape[1])
kwargs = {**kwargs, **self.holdings_kwargs}
h = self.compute_batch_holdings(pred=X, V=self.V_, A=A, **kwargs)
return h

def score(self, X, y):
    """
    Calculate the performance score of the portfolio using Sharpe ratio.

    Parameters:
        X (np.ndarray): Predicted returns.
        y (np.ndarray): Actual returns.

    Returns:
        float: Sharpe ratio of the portfolio.
    """
    return sharpe_ratio(np.sum(X * y, axis=1))

class Mbj(TransformerMixin):
    """
    Computing unconstrained mean-variance weights with the Britten-Jones (1999) trick.
    """

    def __init__(self, positive=False):
        self.positive = positive

    def fit(self, X, y=None):
        m = LinearRegression(fit_intercept=False, positive=self.positive)
        m.fit(X, y=np.ones(len(X)))
        self.coef_ = m.coef_ / np.sqrt(np.sum(m.coef_**2))
        return self

    def transform(self, X):
        return X.dot(self.coef_)

class TimingMeanVariance(BaseEstimator):
    def __init__(self, transform_V=None, a_min=None, a_max=None):
        if transform_V is None:
            self.transform_V = lambda x: np.var(x)
        else:
            self.transform_V = transform_V
        self.a_min = a_min
        self.a_max = a_max

    def fit(self, X, y=None):
        self.V_ = self.transform_V(y)

    def predict(self, X):
        if (self.a_min is None) & (self.a_max is None):
            h = X / self.V_
        else:
            h = np.clip(
                X / np.sqrt(self.V_), a_min=self.a_min, a_max=self.a_max
            ) / np.sqrt(self.V_)
        return h

```

Overwriting ../skfin/mv_estimators.py

```
[4]: from skfin.datasets_ import load_kf_returns
from skfin.mv_estimators import MeanVariance, compute_batch_holdings

returns_data = load_kf_returns(cache_dir="data", force_reload=True)
ret = returns_data["Monthly"]["Average_Value_Weighted_Returns"][:1999]
```

```
[5]: T, N = ret.shape
A = np.ones(N)

h = compute_batch_holdings(ret.mean(), ret.cov(), A)

np.allclose(h.dot(A), [0.0])
```

```
[5]: True
```

```
[6]: A = np.stack([np.ones(N), np.zeros(N)], axis=1)
A[0, 1] = 1

h = compute_batch_holdings(pred=ret.mean(), V=ret.cov(), A=A)
```

3.4 Pnl metrics

The performance of a strategy is evaluated through the profit-and-loss (pnl). The time convention is:

- holdings h_t and returns r_t are known for period t – ie. at the end of period t .
- so to compute pnl with forward-looking information, the holdings must only depend on information up to $t - 1$
- in practice, we will have

$$pnl_t = h_{t-1} \times r_t.$$

First, the measure that captures the gain of a strategy $E[pnl_t]$ for a level of risk $Var[pnl_t]$ is given by the Sharpe ratio:

$$sr = \sqrt{N_{\text{periods per year}}} \times \frac{E[pnl_t]}{\sqrt{Var[pnl_t]}},$$

where $N_{\text{periods per year}}$ is an annualization factor that accounts for the number of observation per year (e.g. 12 for a monthly strategy, 260 for a strategy with pnls only on business days).

```
[7]: %%writefile ../skfin/metrics.py
import numpy as np

def test_monthly(df):
    return int(len(df) / len(df.groupby("M").first())) == 1

def test_bday(df):
    return int(len(df) / len(df.groupby("B").first())) == 1

def test_day(df):
    return int(len(df) / len(df.groupby("D").first())) == 1
```

```
def sharpe_ratio(df, num_period_per_year=None, remove_zeros=True):
    if num_period_per_year is None:
        if test_monthly(df):
            num_period_per_year = 12
        if test_bday(df):
            num_period_per_year = 260
        if test_day(df):
            num_period_per_year = 365
        if num_period_per_year is None:
            return np.nan
    if remove_zeros:
        df = df.replace(0, np.nan)
    return df.mean() / df.std() * np.sqrt(num_period_per_year)

def drawdown(x, return_in_risk_unit=True, window=36, num_period_per_year=12):
    dd = x.cumsum().sub(x.cumsum().cummax())
    if return_in_risk_unit:
        return dd.div(x.rolling(window).std().mul(np.sqrt(num_period_per_year)))
    else:
        return dd
```

Overwriting .../skfin/metrics.py

Chapter 4

Backtesting Industry Momentum Strategies

In this section, we construct a backtest using industry data. More precisely, we use data from Ken French's data library to construct a simple industry momentum return predictor.

The goal of a backtest is to assess the validity of a trading predictor at any point in the past. In particular, it is crucial to avoid any forward-looking bias – in which information available only after time t is mistakenly used at time t . In practice, the predictors are estimated over rolling (or expanding) windows. We implement rolling window estimation with the `sklearn TimeSeriesSplit` object.

To validate a backtest, there are several steps:

1. reasonability of the rationale (why it might/should work);
2. predictability (e.g. statistical significance of the predictor over time and across assets);
3. robustness (does the statistical significance survive perturbations);
4. tradability (does the statistical significance survive cost).

For backtesting, visualisation is very important and we make use of some plotting functions introduced in the Appendix:

```
[1]: from skfin.backtesting import Backtester
      from skfin.plot import bar, heatmap, line
```

4.1 Industry momentum backtest

The setup for predicting industry returns is the following:

- the assets are industries;
- the return forecast α is estimated using rolling-window returns (over L months, $L = 12$) preceding a given date;
- no look-ahead bias: at each date, only information up that date is used;
- such a strategy goes long past “winners” (industries with higher-than-average returns) and goes short “losers” (industries with lower-than-average returns) \Rightarrow Momentum strategy;
- this strategy is often implemented by skipping the most recent month to avoid the 1-month reversal effect.

The article “Do Industries Explain Momentum” (1999) by Moskowitz and Grinblatt in the *Journal of Finance* document that indeed there is momentum in industry returns – past industry returns help predict statistically and economically future industry returns.

THE JOURNAL OF FINANCE • VOL. LIV, NO. 4 • AUGUST 1999

Do Industries Explain Momentum?

TOBIAS J. MOSKOWITZ and MARK GRINBLATT*

ABSTRACT

This paper documents a strong and prevalent momentum effect in industry components of stock returns which accounts for much of the individual stock momentum anomaly. Specifically, momentum investment strategies, which buy past winning stocks and sell past losing stocks, are significantly less profitable once we control for industry momentum. By contrast, industry momentum investment strategies, which buy stocks from past winning industries and sell stocks from past losing industries, appear highly profitable, even after controlling for size, book-to-market equity, individual stock momentum, the cross-sectional dispersion in mean returns, and potential microstructure influences.

Table III
Industry Momentum Trading Profits

L	H =	Panel A: No Gap					Panel B: Month Skipped				
		1	6	12	24	36	1	6	12	24	36
12	Wi	0.0202	0.0164	0.0143	0.0130	0.0138	0.0194	0.0164	0.0141	0.0128	0.0139
	Lo	0.0117	0.0111	0.0116	0.0132	0.0139	0.0128	0.0110	0.0118	0.0131	0.0138
	Wi - Lo	0.0085	0.0053	0.0026	-0.0002	-0.0001	0.0066	0.0054	0.0023	-0.0002	0.0001
		(3.94)	(5.14)	(3.60)	(-0.43)	(-0.10)	(3.11)	(5.27)	(3.18)	(-0.43)	(0.22)
buy	Wi - Mid	0.0052	0.0038	0.0014	-0.0008	-0.0009	0.0044	0.0035	0.0010	-0.0013	-0.0010
		(3.24)	(4.59)	(2.57)	(-1.83)	(-1.94)	(2.72)	(4.18)	(1.84)	(-2.89)	(-2.10)
sell	Mid - Lo	0.0032	0.0015	0.0012	0.0006	0.0008	0.0022	0.0019	0.0013	0.0011	0.0011
		(2.07)	(2.09)	(2.01)	(1.15)	(1.83)	(1.43)	(2.63)	(2.13)	(2.29)	(2.40)
	DGTW	0.0043	0.0030	0.0018	0.0000	0.0001	0.0023	0.0031	0.0015	-0.0000	0.0002
	[Wi - Lo]	(2.30)	(3.89)	(3.11)	(0.01)	(0.33)	(1.26)	(4.11)	(2.64)	(-0.06)	(0.56)

4.1.1 Industry data

To load the data, we use the function `load_kf_returns` introduced in the Data section:

```
[5]: from skfin.datasets import load_kf_returns

returns_data = load_kf_returns(cache_dir="data", force_reload=False)
```

Since the Moskowitz-Grinblatt paper was published in August 1999, we will keep the data after 1999 as out-of-sample and only use the data before 1999.

```
[6]: ret = returns_data["Monthly"]["Average_Value_Weighted_Returns"][:"1999"]
```

4.1.2 Backtesting functions

In the next set of helper file, we introduce the main `Backtester` class and the `fit_and_predict` function to run rolling window estimations.

```
[7]: %%writefile ../skfin/backtesting.py
from dataclasses import dataclass

import numpy as np
import pandas as pd
from joblib import Parallel, delayed
from skfin.mv_estimators import MeanVariance
from sklearn.base import BaseEstimator, clone
from sklearn.model_selection import TimeSeriesSplit
from sklearn.utils.metaestimators import _safe_split


def compute_pnl(h, ret, pred_lag):
    pnl = h.shift(pred_lag).mul(ret)
    if isinstance(h, pd.DataFrame):
        pnl = pnl.sum(axis=1)
    return pnl


def fit_predict(estimator, X, y, train, test, return_estimator=True):
    X_train, y_train = _safe_split(estimator, X, y, train)
    X_test, _ = _safe_split(estimator, X, y, test, train)
    estimator.fit(X_train, y_train)
    if return_estimator:
        return estimator.predict(X_test), estimator
    else:
        return estimator.predict(X_test)


@dataclass
class Backtester:
    estimator: BaseEstimator = MeanVariance()
    max_train_size: int = 36
    test_size: int = 1
    pred_lag: int = 1
    start_date: str = "1945-01-01"
    end_date: str = None
    name: str = None

    def compute_holdings(self, X, y, pre_dispatch="2*n_jobs", n_jobs=1):
        X = X.loc[:self.end_date]
        cv = TimeSeriesSplit(
            max_train_size=self.max_train_size,
            test_size=self.test_size,
            n_splits=1 + len(X.loc[self.start_date : ]) // self.test_size,
        )
        parallel = Parallel(n_jobs=n_jobs, pre_dispatch=pre_dispatch)
        res = parallel(
            delayed(fit_predict)(
                clone(self.estimator), X.values, y.values, train, test, True
            )
        )
        return np.array(res).T
```

```

        )
        for train, test in cv.split(X)
    )
    y_pred, estimators = zip(*res)
    idx = X.index[np.concatenate([test for _, test in cv.split(X)])]
    if isinstance(y, pd.DataFrame):
        cols = y.columns
        h = pd.DataFrame(np.concatenate(y_pred), index=idx, columns=cols)
    elif isinstance(y, pd.Series):
        h = pd.Series(np.concatenate(y_pred), index=idx)
    else:
        h = None
    self.h_ = h
    self.estimators_ = estimators
    self.cv_ = cv
    return self

def compute_pnl(self, ret):
    pnl = compute_pnl(self.h_, ret, self.pred_lag)
    self.pnl_ = pnl.loc[self.start_date : self.end_date]
    if self.name:
        self.pnl_ = self.pnl_.rename(self.name)
    return self

def train(self, X, y, ret):
    self.compute_holdings(X, y)
    self.compute_pnl(ret)
    return self.pnl_

```

Overwriting .../skfin/backtesting.py

```
[8]: from skfin.backtesting import Backtester
from skfin.mv_estimators import MeanVariance
```

4.1.3 Scikit-Learn TimeSeriesSplit

```
[9]: from sklearn.model_selection import TimeSeriesSplit
```

Given that the data is monthly, we re-estimate the model every month. This is done by choosing the parameter `n_splits` in the class `TimeSeriesSplit` as the number of months.

```
[10]: start_date = "1945-01-01"
test_size = 1
params = dict(max_train_size=36, test_size=test_size, gap=0)
params["n_splits"] = 1 + len(ret.loc[start_date:]) // test_size

cv = TimeSeriesSplit(**params)
```

More precisely, with `TimeSeriesSplit`:

- the `test` indices are the dates for which the holdings are computed.
- the `train` indices are the date range over which a forecasting model is trained.
- the target will be shifted by -1 and `gap` is set to 0.
- we can estimate batches with `test_size > 1`.
- `n_splits` is determined so that the backtest starts (just) before a certain start date.

```
[11]: for train, test in cv.split(ret):
    break
ret.iloc[train].index[-1].strftime("%Y%m%d"), ret.iloc[test].index[0].strftime("%Y%m%d")
```

```
[11]: ('19441101', '19441201')
```

4.2 Empirical results

4.2.1 Cumulative pnl

We first define two transform functions:

- `transform_X` computes the main feature (or predictor) as the average of the trailing 12-month returns;
- `transform_y` is applied on the returns – here when passed to the mean-variance `MeanVariance` class to compute the covariance matrix; but more generally to serve as a target for predictors with machine-learning.

```
[12]: def transform_X(df, window=12):
        return df.rolling(window=window).mean()

def transform_y(df):
        return df.shift(-1)

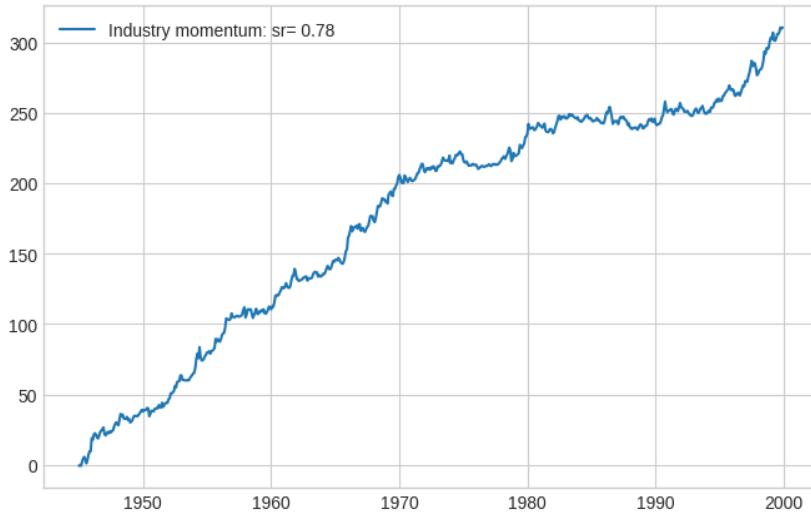
X = transform_X(ret)
y = transform_y(ret)
```

```
[13]: _h = []
for train, test in cv.split(ret):
    m = MeanVariance()
    m.fit(X.values[train], y.values[train])
    _h += [m.predict(X.values[test])]

cols = ret.columns
idx = ret.index[np.concatenate([test for _, test in cv.split(ret)])]
h = pd.DataFrame(np.concatenate(_h), index=idx, columns=cols)
```

Note that the line plotting function shows the sharpe ratio “sr” of the strategy.

```
[14]: pnl = h.shift(1).mul(ret).sum(axis=1)[start_date:].rename("Industry momentum")
line(pnl, cumsum=True, loc="best")
```



We can also use the Backtester class and we test that the two approaches yield the same result.

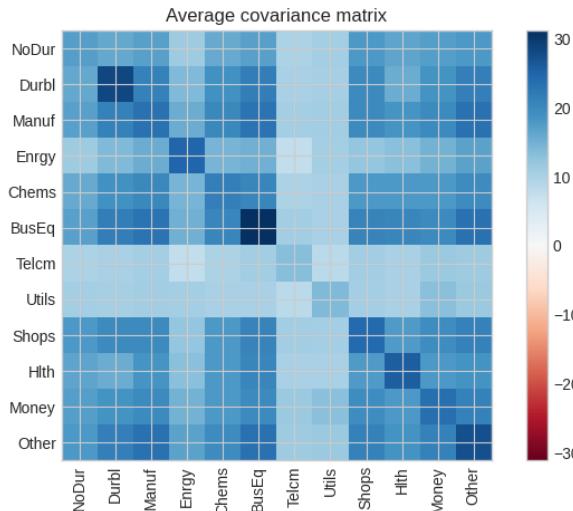
```
[15]: m = Backtester(estimator=MeanVariance(), name="Industry momentum")
pnl_ = m.train(X, y, ret)

[16]: assert np.allclose(h, m.h_)
assert np.allclose(pnl, m.pnl_)
```

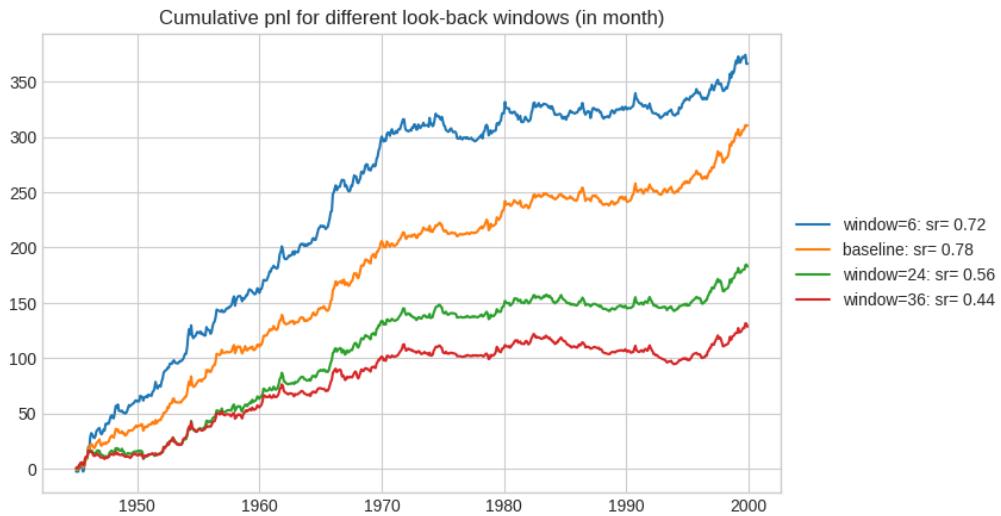
4.2.2 Other backtest statistics

We can also extract information for the estimator – e.g. in this simple case, recover the covariance matrix fitted by the class MeanVariance().

```
[17]: estimators = m.estimators_
V_mean = pd.DataFrame(
    sum([m.V_ for m in estimators]) / len(estimators),
    index=ret.columns,
    columns=ret.columns,
)
heatmap(V_mean, title="Average covariance matrix")
```



```
[18]: pnls = []
for window in [6, 12, 24, 36]:
    X_ = ret.rolling(window).mean()
    name = f"window={window}" if window != 12 else "baseline"
    pnls += [Backtester(estimator=MeanVariance(), name=name).train(X_, y, ret)]
title = "Cumulative pnl for different look-back windows (in month)"
line(pd.concat(pnls, axis=1), cumsum=True, start_date="1945", title=title)
```



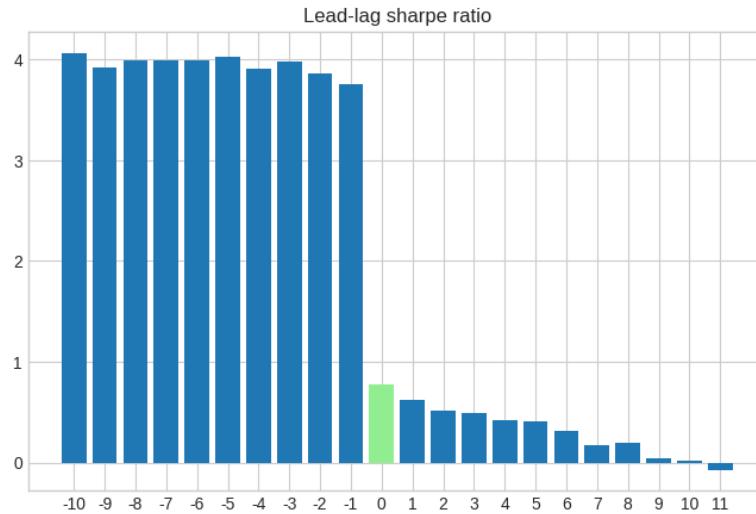
The following graph shows the lead-lag sharpe ratio plot for the industry momentum backtest:

- the horizon “0” (green bar) represents the sharpe ratio of the tradable pnl;
- the lagged horizons (with positive coordinates) show the performance when the positions are lagged by several months and illustrates the alpha decay of the predictor;
- the lead horizons (with negative coordinates) represent the non-tradable counterfactual performance if the information had been available earlier.

For trending predictors (like the industry momentum backtest), the lead sharpe ratios are very high and the alpha decay is quite fast. For contrarian predictors, the lead sharpe ratios are negative.

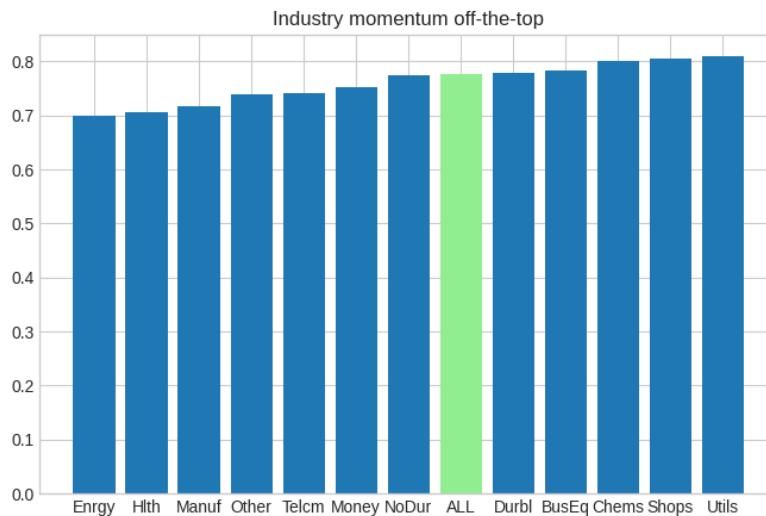
```
[19]: from skfin.metrics import sharpe_ratio

sr = {i: h.shift(1 + i).mul(ret).sum(axis=1).pipe(sharpe_ratio) for i in range(-10, 12)}
bar(sr, baseline=0, sort=False, title="Lead-lag sharpe ratio")
```



The off-the-top approach is to remove an asset from the tradable set and check whether the portfolio sharpe ratio decreases (in which case, this asset is a *contributor*) or increases (in which case, this asset is a *detractor*).

```
[20]: pnls_ott = {}
for c in ret.columns:
    ret_ = ret.drop(c, axis=1)
    X_ = transform_X(ret_)
    y_ = transform_y(ret_)
    pnls_ott[c] = (
        Backtester(estimator=MeanVariance()).train(X_, y_, ret_).pipe(sharpe_ratio)
    )
pnls_ott["ALL"] = pml.pipe(sharpe_ratio)
bar(pnls_ott, baseline="ALL", title="Industry momentum off-the-top")
```



Chapter 5

Enhancing Industry Momentum Strategies with Regularized Linear Models

In this section, we take advantage of some of scikit-learn powerful features such as the pipeline to run backtests with some learning. This is an extension of the industry momentum backtests presented in the previous section.

5.1 Ridge / Lasso / Elastic net

For a Ridge regression, the betas $\langle \beta_1, \dots, \beta_p \rangle$ are chosen to minimize the L2-norm of the vector β :

$$\frac{1}{2} \sum_{i=1}^N (y_i - \beta_0 - \sum_{j=1}^p x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^p \beta_j^2.$$

The Ridge regression generally provides more stable and accurate estimates than a standard linear regression. For the Lasso regression, the betas $\langle \beta_1, \dots, \beta_p \rangle$ are chosen to minimize the L1-norm of β :

$$\frac{1}{2} \sum_{i=1}^N (y_i - \beta_0 - \sum_{j=1}^p x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^p |\beta_j|.$$

The Lasso tends to promote sparse and stable models that can be more easily interpretable. Finally, for the Elastic net: the betas $\langle \beta_1, \dots, \beta_p \rangle$ are chosen to minimize

$$\frac{1}{2} \sum_{i=1}^N (y_i - \beta_0 - \sum_{j=1}^p x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^p [(1-\alpha)\beta_j^2 + \alpha|\beta_j|].$$

The lasso penalty is not very good at selecting individual predictors when they are strong and closely related, while the ridge penalty tends to make the coefficients of related variables similar. The elastic net finds a middle ground by potentially averaging the impact of closely related features while encouraging a simpler model overall.

To give an example, we use a diabetes dataset provided by sklearn.

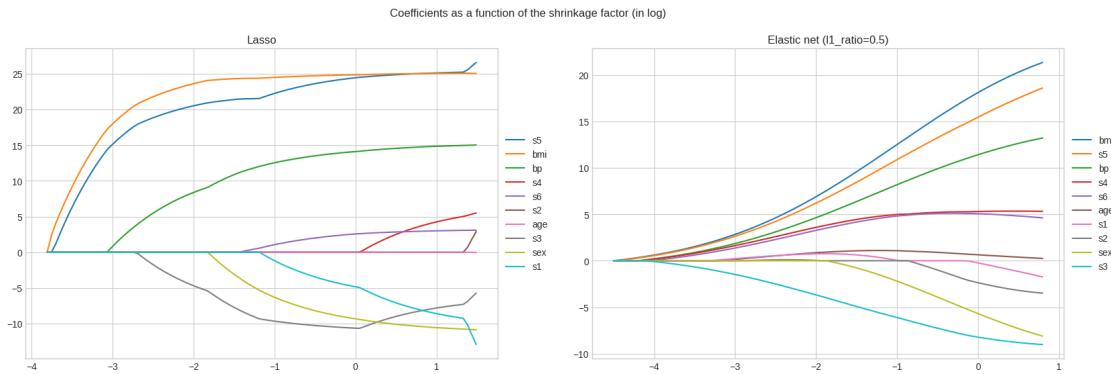
```
[2]: from sklearn.datasets import load_diabetes
      from sklearn.linear_model import enet_path, lasso_path
```

```
[3]: X, y = load_diabetes(return_X_y=True, as_frame=True)

X /= X.std(axis=0)
eps = 5e-3
l1_ratio = 0.5

alphas_lasso, coefs_lasso, _ = lasso_path(X, y, eps=eps)
alphas_enet, coefs_enet, _ = enet_path(X, y, eps=eps, l1_ratio=l1_ratio)
```

```
[4]: fig, ax = plt.subplots(1, 2, figsize=(20, 6))
fig.suptitle("Coefficients as a function of the shrinkage factor (in log)")
line(
    pd.DataFrame(coefs_lasso.T, -1 * np.log(alphas_lasso), columns=X.columns),
    title="Lasso",
    ax=ax[0],
)
line(
    pd.DataFrame(coefs_enet.T, -1 * np.log(alphas_enet), columns=X.columns),
    title=f"Elastic net (l1_ratio={l1_ratio})",
    ax=ax[1],
)
```



See more discussion in

Hastie, Trevor, et al. *The elements of statistical learning: data mining, inference, and prediction*. Vol. 2. New York: Springer, 2009.

5.2 Revisiting the industry backtest with linear models

In the Industry momentum section, we introduced a feature X as 12-month trailing past returns and target y as the one-month forward return. For N industries (e.g. $N = 12$), X and y are vectors of size N . In this section, we are interested in fitting linear models (potentially with regularisation à la Ridge, Lasso or Elastic Net) such that

$$y = BX,$$

where B is a matrix of size $[N \times N]$. (In this equation, we ignore intercept terms which are generally small.) This equation falls into two lines of research:

- in classic Econometrics, this is called a Vector Autoregressive (VAR) model (e.g see the [wikipedia article](#)).

- in Machine-learning, this is often called a `multioutput` model.

The interpretation of this model in the context of industry timing is that it captures cross-industry effect:

- for instance, an increasing trend on oil prices (which is positive for the Energy stocks) might be a negative predictor for sectors that use oil as input (e.g. Manufacturing)

Such VAR/multioutput models will pick up the positive lead-lag correlations across industries and therefore potentially enrich the simple Industry momentum strategy that we introduced in the previous section.

5.2.1 Scikit-learn Pipeline and Multi-output

From the [documentation](#), the definition of a `sklearn` pipeline is:

Sequentially apply a list of transforms and a final estimator. Intermediate steps of the pipeline must be ‘transforms’, that is, they must implement fit and transform methods.
The final estimator only needs to implement fit.

Estimator like linear regressions or Ridge regressions have a `multioutput` argument which allows for a joint fit of multiple targets at the same time. This provides a similar behavior to a `sklearn` transformations. Moreover:

- in the pipelines that we are building, the last step is the `MeanVariance` class that produces the holdings;
- in the `sklearn` pipelines, all the steps except for the last one must be transformations;
- despite formally not having a `transform` function, the `multioutput` linear estimators (such as Ridge and Lasso) are transformations of a vector X of size N into a vector y of size N .

In the following module, we extend the estimators that we will be using to have such `transform` property.

```
[5]: %%writefile ./skfin/estimators.py
from lightgbm.sklearn import LGBMRegressor
from sklearn.base import BaseEstimator
from sklearn.linear_model import LinearRegression, Ridge, RidgeCV
from sklearn.multioutput import MultiOutputRegressor
from sklearn.neural_network import MLPRegressor

def add_transform_method(cls):
    """
    Decorator to add a 'transform' method to a class that uses the 'predict' method.
    """
    def transform(self, X):
        return self.predict(X)

    cls.transform = transform
    return cls

@add_transform_method
class LinearRegression(LinearRegression):
    pass

@add_transform_method
class Ridge(Ridge):
    pass
```

```

@add_transform_method
class RidgeCV(RidgeCV):
    pass

@add_transform_method
class MLPRegressor(MLPRegressor):
    pass

@add_transform_method
class MultiOutputRegressor(MultiOutputRegressor):
    pass

class MultiLGBMRegressor(BaseEstimator):
    """
    Multi-output extension of the lightgbm regressor as a transform class
    get_params and set_params attributes necessary for cloning the class
    """

    def __init__(self, **kwargs):
        if "n_jobs" in kwargs.keys():
            kwargs["n_jobs"] = 1
        else:
            kwargs = {"n_jobs": 1, **kwargs}
        self.m = MultiOutputRegressor(LGBMRegressor(**kwargs))

    def get_params(self, deep=True):
        return self.m.estimator.get_params(deep=deep)

    def set_params(self, **kwargs):
        if "n_jobs" in kwargs.keys():
            kwargs["n_jobs"] = 1
        else:
            kwargs = {"n_jobs": 1, **kwargs}
        return self.m.estimator.set_params(**kwargs)

    def fit(self, X, y):
        return self.m.fit(X, y)

    def transform(self, X):
        return self.m.transform(X)

```

Overwriting ../skfin/estimators.py

5.2.2 Linear Regression

We first load the data and the main custom functions to run a backtest.

```

[6]: from skfin.backtesting import Backtester
from skfin.datasets_ import load_kf_returns
from skfin.mv_estimators import MeanVariance

returns_data = load_kf_returns(cache_dir="data")

ret = returns_data["Monthly"]["Average_Value_Weighted_Returns"][:"1999"]

def transform_X(df, window=12):
    return df.rolling(window=window).mean()

```

```

def transform_y(df):
    return df.shift(-1)

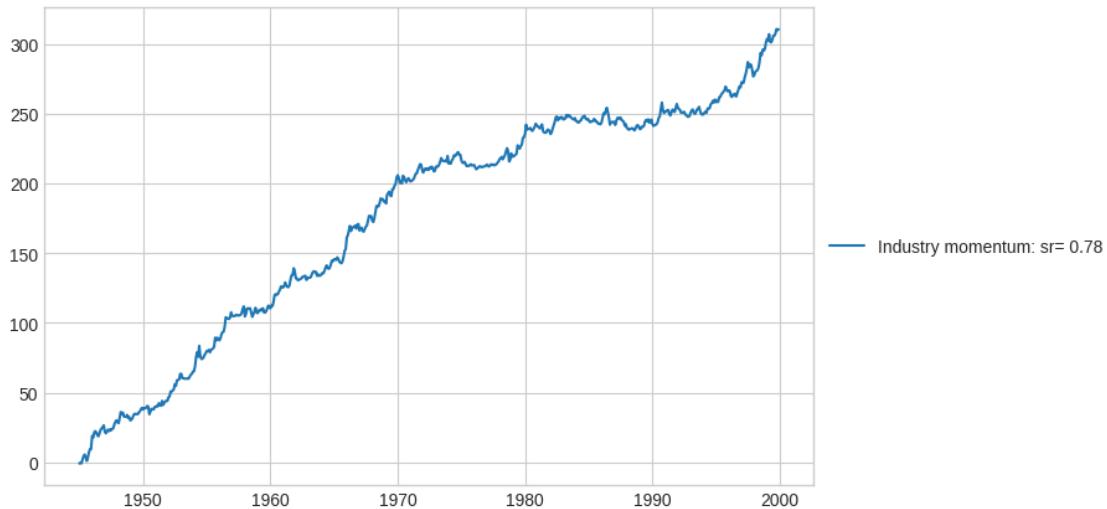
features = transform_X(ret)
target = transform_y(ret)

```

As a reference, we re-compute the pnl of a simple Industry momentum strategy.

```
[7]: pnls = [
    Backtester(MeanVariance(), name="Industry momentum").train(features, target, ret)
]
```

```
[8]: line(pnls, cumsum=True)
```



We now load the functions that are specific to building sklearn pipelines.

```
[9]: from skfin.estimators import LinearRegression, Ridge
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
```

In the following simple pipeline, there are two steps:

- the first step is a simple multioutput linear regression that produces fitted predictions for each industry
- the second step takes these predictors and scales them as holdings with a mean-variance optimisation (with the MeanVariance class introduced earlier).

```
[10]: estimator = make_pipeline(LinearRegression(), MeanVariance())
```

```
[11]: m = (
    Backtester(estimator, name="linear_regression")
    .compute_holdings(features, target)
    .compute_pnl(ret)
)
pnls += [m.pnl_]
line(m.pnl_, cumsum=True)
```



The linear regression fits an intercept and coefficients.

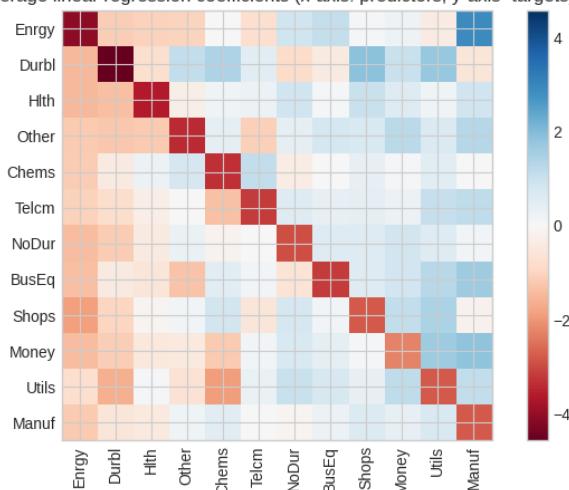
```
[12]: ols_ = m.estimators_[0].named_steps["linearregression"]
coef_ = ols_.coef_
intercept_ = ols_.intercept_
vec = ret.mean().values
np.allclose(ols_.predict(vec[None, :]), coef_.dot(vec) + intercept_)
```

```
[12]: True
```

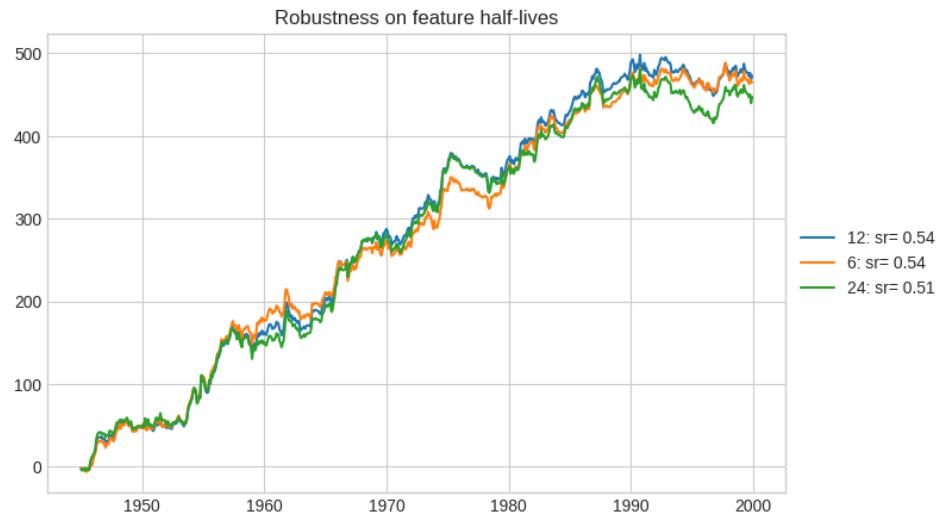
```
[13]: coefs_ = [m.named_steps["linearregression"].coef_ for m in m.estimators_]
coefs_mean = pd.DataFrame(sum(coefs_) / len(coefs_), ret.columns, ret.columns).T
```

```
[14]: heatmap(
    coefs_mean.loc[
        coefs_mean.mean(1).sort_values().index, coefs_mean.mean(1).sort_values().index
    ],
    title="Average linear regression coefficients (x-axis: predictors, y-axis=targets)",
)
```

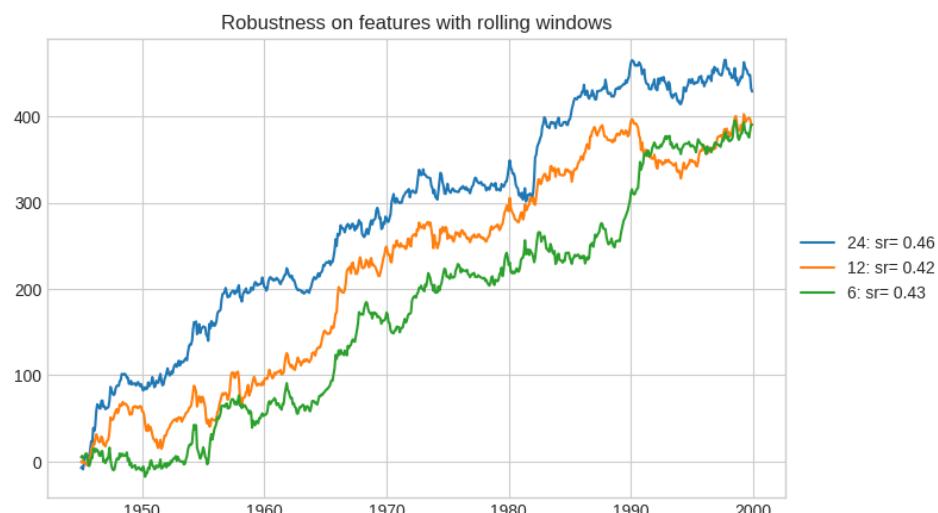
Average linear regression coefficients (x-axis: predictors, y-axis=targets)



```
[15]: pnls_ = {}
for hl in [6, 12, 24]:
    features_ = ret.ewm(halflife=hl).mean().fillna(0)
    pnls_[hl] = Backtester(estimator).train(features_, target, ret)
line(pnls_, cumsum=True, title="Robustness on feature half-lives")
```



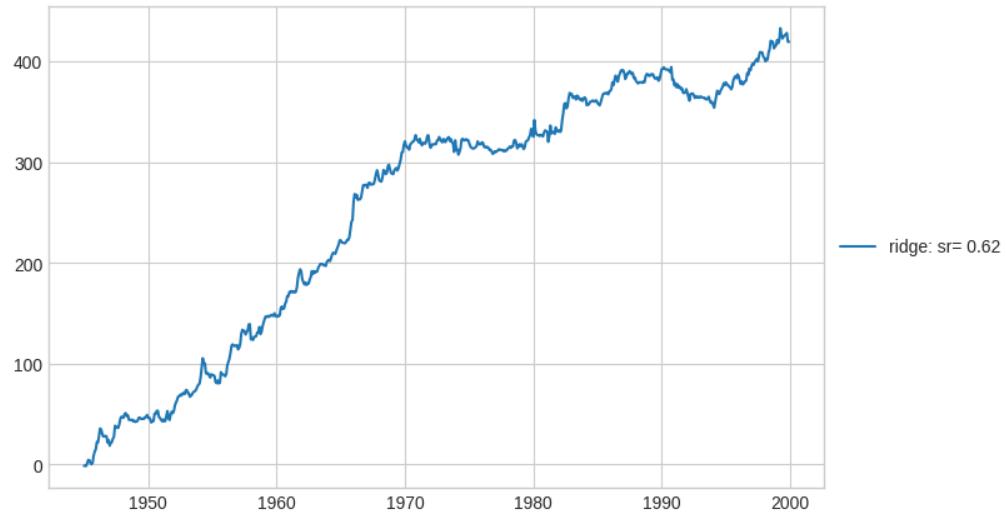
```
[16]: pnls_ = {}
for hl in [6, 12, 24]:
    features_ = ret.rolling(window=hl).mean().fillna(0)
    pnls_[hl] = Backtester(estimator).train(features_, target, ret)
line(pnls_, cumsum=True, title="Robustness on features with rolling windows")
```



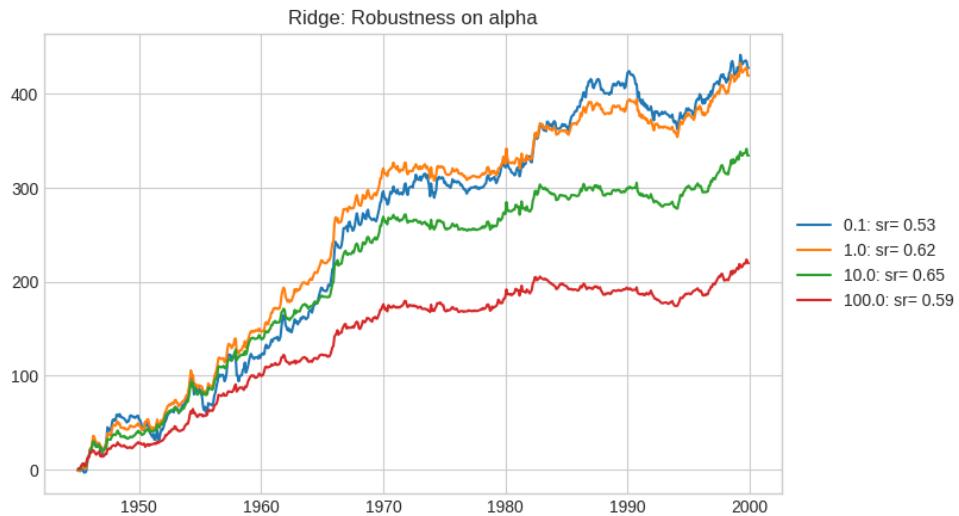
5.2.3 Ridge

Given that the Ridge regression applies a constraint across features, each feature needs to be properly rescaled, which is done here with StandardScalar.

```
[17]: estimator = make_pipeline(StandardScaler(with_mean=False), Ridge(), MeanVariance())
m = (
    Backtester(estimator, name="ridge")
    .compute_holdings(features, target)
    .compute_pnl(ret)
)
pnls_ += [m.pnl_]
line(m.pnl_, cumsum=True)
```



```
[18]: pnls_ = []
for alpha in [0.1, 1, 10, 100]:
    estimator_ = make_pipeline(
        StandardScaler(with_mean=False), Ridge(alpha=alpha), MeanVariance()
    )
    pnls_[alpha] = Backtester(estimator_).train(features, target, ret)
line(pnls_, cumsum=True, title="Ridge: Robustness on alpha")
```



5.2.4 Ridge with feature expansion

We can expand the set of features by using polynomial transforms with `PolynomialFeatures`.

```
[19]: PolynomialFeatures(degree=2).fit_transform(ret.iloc[:10]).shape
```

```
[19]: (10, 91)
```

Number of new features: intercept, initial features (=12), squared features (12), all cross features of degree 1 (=6*11):

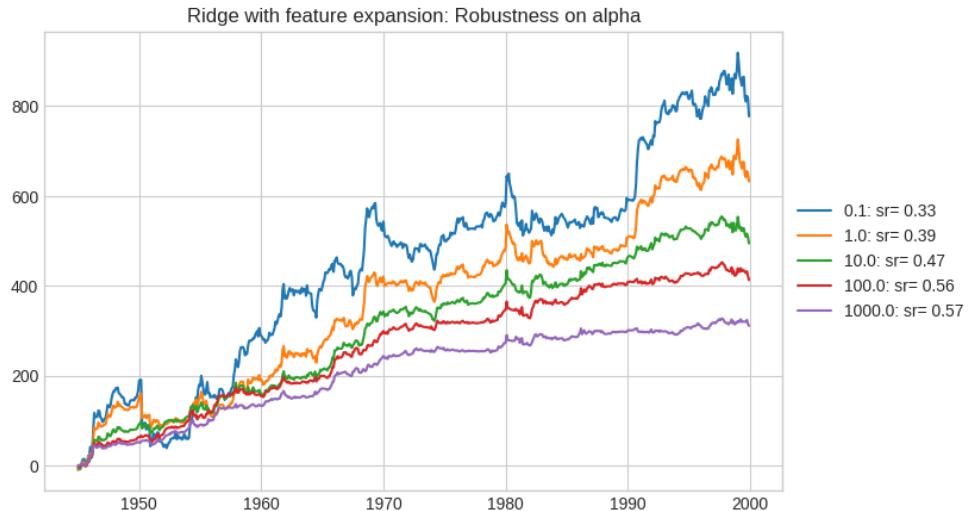
```
[20]: estimator = make_pipeline(
    StandardScaler(with_mean=False),
    PolynomialFeatures(degree=2),
    Ridge(alpha=100),
    MeanVariance(),
)
```

```
[21]: print(f"Number of features generated by degree=2: {1 + 12 + 12 + 6 * 11}")
```

Number of features generated by degree=2: 91

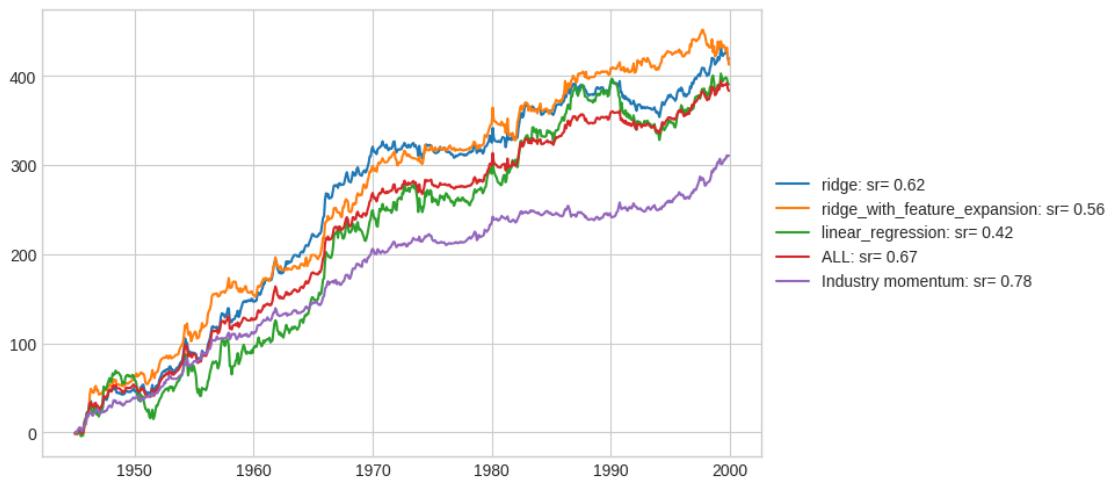
```
[22]: pnls += [
    Backtester(estimator, name="ridge_with_feature_expansion").train(
        features_, target, ret
    )
]
# line(m.pnl_, cumsum=True)
```

```
[23]: pnls_ = {}
for alpha in [0.1, 1, 10, 100, 1000]:
    estimator_ = make_pipeline(
        StandardScaler(with_mean=False),
        PolynomialFeatures(degree=2),
        Ridge(alpha=alpha),
        MeanVariance(),
    )
    pnls_[alpha] = Backtester(estimator_).train(features_, target, ret)
line(pnls_, cumsum=True, title="Ridge with feature expansion: Robustness on alpha")
```



Putting all the types of linear predictors together, we can compare the cumulative pnls in the graph below.

```
[24]: line(pd.concat(pnls, axis=1).assign(ALL=lambda x: x.mean(axis=1)), cumsum=True)
```



Chapter 6

Non-Linear Forecasting Models: Boosted Trees and Multi-Layer Perceptrons

In this section, we review two important classes of non-linear forecasting models: boosted trees and the multi-layer perceptron.

6.1 Boosted Trees

We first discuss boosted trees, in particular as described by the companion paper to the package `xgboost`:

Chen and Guestrin (2016): “XGBoost: A Scalable Tree Boosting System,” *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*.

XGBoost: A Scalable Tree Boosting System

Tianqi Chen
University of Washington
tqchen@cs.washington.edu

Carlos Guestrin
University of Washington
guestrin@cs.washington.edu

For a dataset $\langle x_n, y_n \rangle$ with N samples ($x_n \in \mathbb{R}^M$), a tree ensemble model uses K additive functions:

$$\hat{y}_n = \phi(x_n) = \sum_k f_k(x_n),$$

where f_k is in the space of regression trees $\mathcal{F} = \{f\}$:

- $q: \mathbb{R}^M \rightarrow J$ is a partition;
- $f(x) = w_{q(x)}$ is a constant value on each leaf of the tree.

The objective is to minimize the loss:

$$\mathcal{L}(\phi) = \sum_n l(y_n, \hat{y}_n) + \sum_k \Omega(f_k),$$

where $\Omega(f) = \gamma J + \frac{1}{2}\lambda||w||^2$ is a regularisation term and l is a convex loss function (e.g. mean squared error).

The functions f_k are derived iteratively:

$$\mathcal{L}^k = \sum_n l\left(y_n, \hat{y}_n^{k-1} + f_k(x_n)\right) + \Omega(f_k).$$

With a second-order Taylor expansion, we have

$$\mathcal{L}^k \approx \sum_n \left[l(y_n, \hat{y}_n^{k-1}) + g_n f_k(x_n) + \frac{1}{2} h_n f_k(x_n)^2 \right] + \Omega(f_k),$$

where $g_n = \partial_{\hat{y}} l(y_n, \hat{y}_n^{k-1})$ and $h_n = \partial_{\hat{y}}^2 l(y_n, \hat{y}_n^{k-1})$. For an instance of leaf j , with $I_j = \{n | q(x_n) = j\}$, we can sum by leaf:

$$\mathcal{L}^k = \sum_{j=1}^{J=I} \left(\sum_{n \in I_j} g_n \right) w_j + \frac{1}{2} \left(\sum_{n \in I_j} h_n + \lambda \right) w_j^2 + \gamma J + \text{constant}.$$

For a given $q(x)$, the optimal weight w_j^* of leaf j is

$$w_j^* = -\frac{\sum_{n \in I_j} g_n}{\sum_{n \in I_j} h_n + \lambda}.$$

The corresponding optimal value is then

$$\tilde{\mathcal{L}}^k(q) = -\frac{1}{2} \sum_{j=1}^{J=I} \frac{\left(\sum_{n \in I_j} g_n \right)^2}{\sum_{n \in I_j} h_n + \lambda} + \gamma J + \text{constant}.$$

A greedy algorithm that starts from a single leaf and iteratively adds branches to the tree is used to determine q .

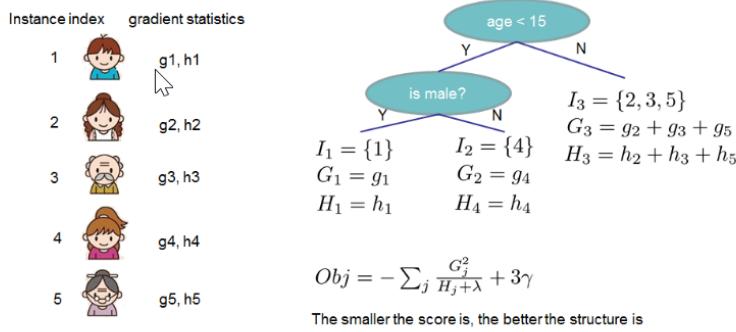


Figure 2: Structure Score Calculation. We only need to sum up the gradient and second order gradient statistics on each leaf, then apply the scoring formula to get the quality score.

In practice, `xgboost` (and `lightgbm`) can be used with custom loss functions – for instance, by providing the gradient and hessian functions.

6.2 Multi-layer perceptron

For the input $x \in \mathbb{M}$, the layer (with hidden size equals to K) of a multi-layer perceptron is given by

$$g(b + Wx)$$

where W is a $[K \times M]$ matrix, b is a scalar and g is an activation function. The output of the last layer has to match the size of the target vector y .

6.3 Predicting industry returns with non-linear models

6.3.1 Lightgbm

```
[4]: from sklearn.estimators import MLPRegressor, MultiLGBMRegressor, Ridge

def transform_X(df, window=12):
    return df.rolling(window=window).mean()

def transform_y(df):
    return df.shift(-1)

features = transform_X(ret)
target = transform_y(ret)
```

As a benchmark based on estimating the cross-industry effects, we first look at the `Ridge` estimator.

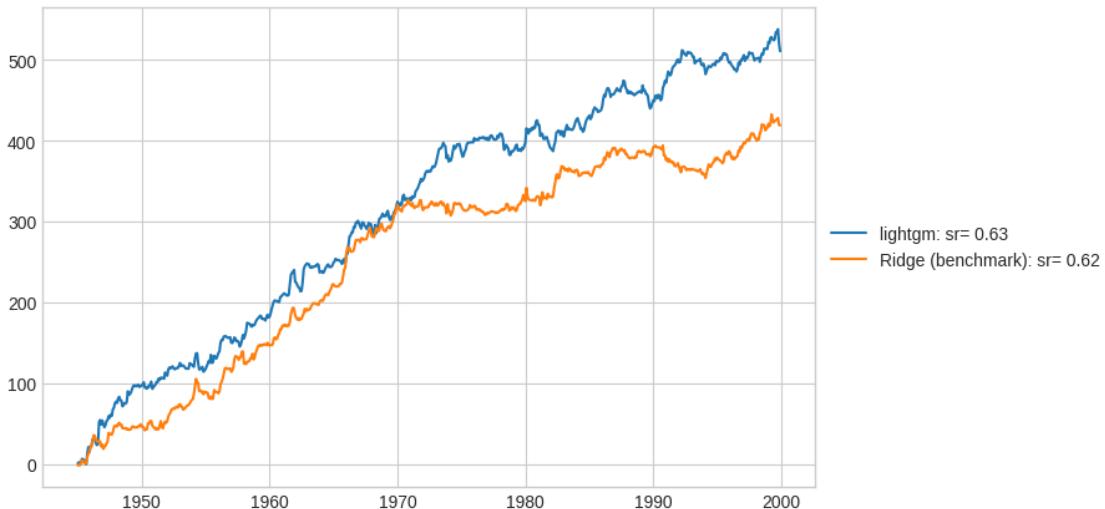
```
[5]: %%time
estimator0 = make_pipeline(StandardScaler(with_mean=False), Ridge(), MeanVariance())
m = (
    Backtester(estimator0, name="Ridge (benchmark)")
    .compute_holdings(features, target)
    .compute_pnl(ret)
)
h0, pnl0, estimators0 = m.h_, m.pnl_, m.estimators_
pnls = [pnl0]
```

CPU times: user 2.54 s, sys: 5.23 s, total: 7.76 s
Wall time: 977 ms

```
[6]: estimator = make_pipeline(
    MultiLGBMRegressor(min_child_samples=5, n_estimators=25), MeanVariance()
)
```

```
[7]: %%time
m = (
    Backtester(estimator, name="lightgm")
    .compute_holdings(features, target)
    .compute_pnl(ret)
)
pnls += [m.pnl_]
line(pnls, cumsum=True)
```

CPU times: user 1min 35s, sys: 3min 59s, total: 5min 34s
Wall time: 42.3 s



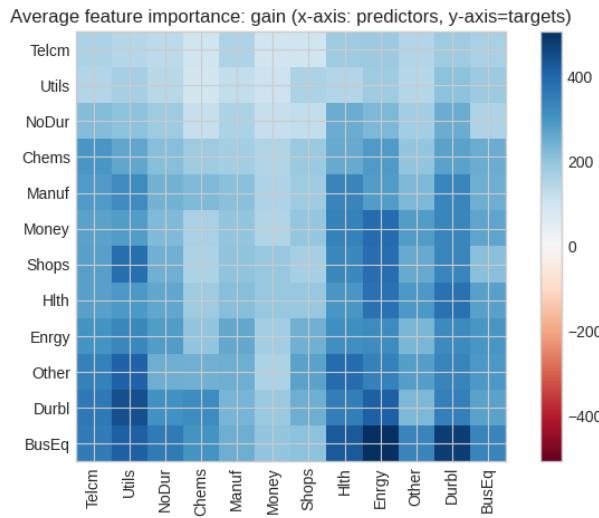
```
[8]: func = lambda x: np.stack(
    [m.booster_.feature_importance(importance_type="gain") for m in x]
)
importance = [func(m_.steps[0][1].m.estimators_) for m_ in m.estimators_]
importance_mean = pd.DataFrame(
    sum(importance) / len(importance), ret.columns, ret.columns
).T

heatmap(
    importance_mean.loc[
```

```

        importance_mean.mean().sort_values().index,
        importance_mean.mean().sort_values().index,
    ],
    title="Average feature importance: gain (x-axis: predictors, y-axis=targets)",
)

```



6.3.2 MLPRegressor

We first focus on a single window to understand how the `MLPRegressor` works in `scikit-learn`.

```

[9]: for train, test in m.cv_.split(ret):
    break

[10]: scaler = StandardScaler(with_mean=True)
X_train = scaler.fit_transform(features.iloc[train])
y_train = target.iloc[train]

[11]: X_test = scaler.transform(features.iloc[test])
y_test = target.iloc[test]

```

We instantiate a simple MLP with 6 neurons. The activation function is a logistic sigmoid.

```

[12]: m = MLPRegressor(
    hidden_layer_sizes=(6,),
    solver="adam",
    learning_rate_init=0.5,
    alpha=100,
    activation="logistic",
    tol=1e-2,
    n_iter_no_change=25,
    early_stopping=False,
)
m.fit(X_train, y_train)
y_pred = m.predict(X_test)

```

When `early_stopping` is `False`, the optimisation stops based on the in-sample score, while when `early_stopping` is `True`, the decision to stop is based on a random sample (e.g. 10% of the training data).

```
[13]: print(f"The number of iterations is {m.n_iter_} (out of a maximum of {m.max_iter}).")
```

The number of iterations is 83 (out of a maximum of 200).

```
[14]: print(
    f"Number of parameter:\n - first layer: {12 * 6 + 6}\n - second layer: {12 * 6 + 12}\n -\n"
    f"total number of parameters: {12 * 12 + 6 + 12}"
)
```

Number of parameter:
- first layer: 78
- second layer: 84
- total number of parameters: 162

```
[15]: m.coefs_[0].shape, m.coefs_[1].shape
```

```
[15]: ((12, 6), (6, 12))
```

```
[16]: m.intercepts_[0].shape, m.intercepts_[1].shape
```

```
[16]: ((6,), (12,))
```

The sigmoid logistic activation function is also known as `expit` and it is provided by the `scipy` package.

The MLP regressor in this case:

- project the vector of size 12 on a vector of size 6
- a bias vector of size 6 is added
- the activation function (here the sigmoid) regularizes the neurons
- the second layer then projects the vector of size 6 on a vector of size 12 (with a bias of size 12).

```
[17]: from scipy.special import expit

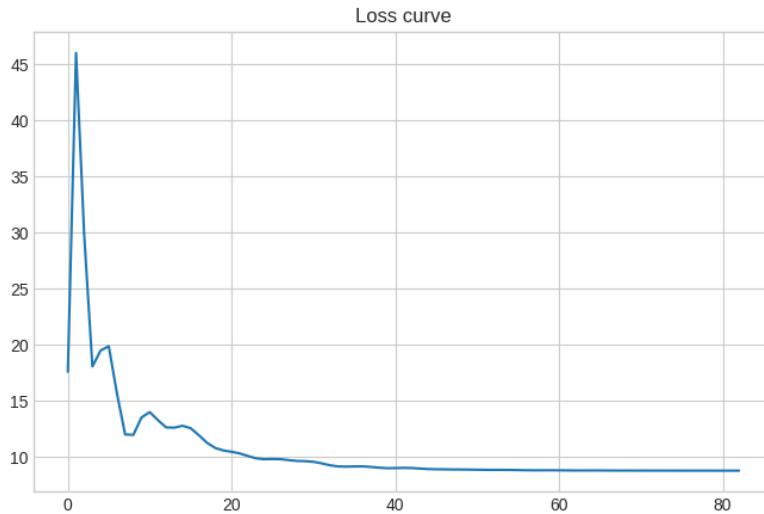
y_pred_ = (
    expit(X_test.dot(m.coefs_[0]) + m.intercepts_[0]).dot(m.coefs_[1])
    + m.intercepts_[1]
)
```

```
[18]: np.allclose(y_pred, y_pred_)
```

```
[18]: True
```

The `sklearn` package provides a loss curve

```
[19]: line(pd.Series(m.loss_curve_), legend=False, title="Loss curve")
```



The quadratic loss of is the squared error:

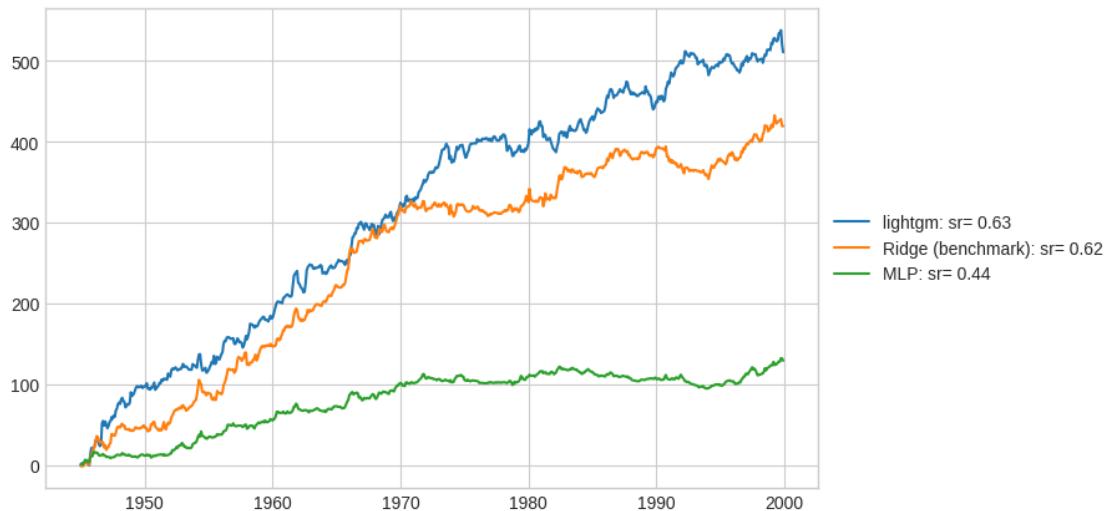
```
[20]: np.allclose(m.loss_curve_[-1], ((y_train - y_pred) ** 2).mean().mean() / 2, atol=1e-2)
```

```
[20]: True
```

We now look at a backtest using MLPRegressor.

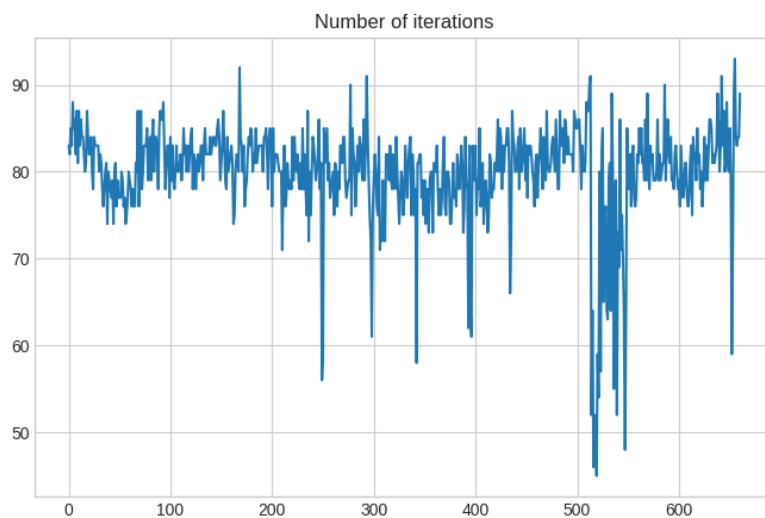
```
[21]: estimator = make_pipeline(
    StandardScaler(with_mean=False),
    MLPRegressor(
        hidden_layer_sizes=(6,),
        learning_rate_init=0.5,
        alpha=100,
        solver="adam",
        activation="logistic",
        tol=1e-2,
        n_iter_no_change=25,
        early_stopping=False,
    ),
    MeanVariance(),
)
m = (
    Backtester(estimator, name="MLP")
    .compute_holdings(features, target)
    .compute_pnl(ret)
)
pnls += [m.pnl_]

line(pnls, cumsum=True)
```



The graph below shows the number of iteration in the Adam stochastic gradient descent of the MLP estimator. While the MLP has no fixed number of iterations, it oscillates around 80.

```
[22]: line(
    pd.Series([m_[1].n_iter_ for m_ in m.estimators_]),
    title="Number of iterations",
    legend=False,
)
```

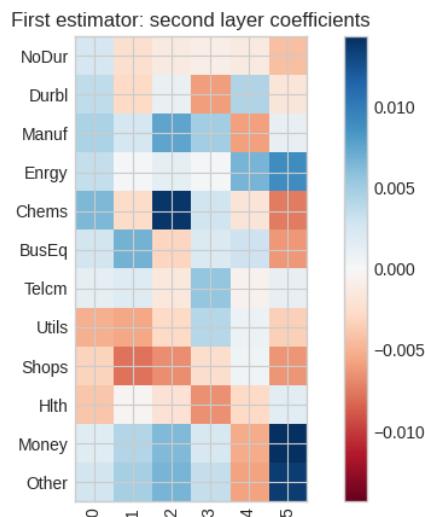
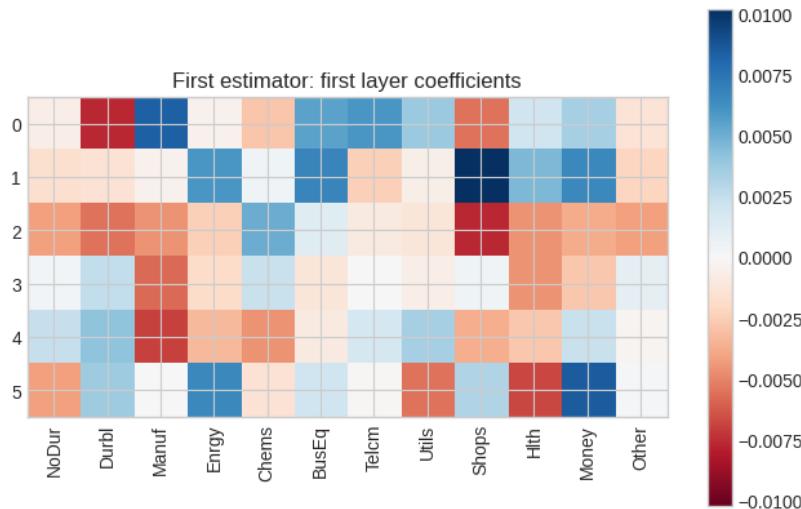


The heatmaps below shows the coefficients W for the first and second layers of the MLP.

```
[23]: for m_ in m.estimators_:
    break

    heatmap(
        pd.DataFrame(m_[1].coefs_[0], index=ret.columns),
        title="First estimator: first layer coefficients",
```

```
)
    heatmap(
        pd.DataFrame(m_[1].coefs_[1], columns=ret.columns),
        title="First estimator: second layer coefficients",
    )
)
```



Given the stochasticity of the estimation, we are interested in evaluating the noise associated to a given run. More precisely, we re-run the backtest with exactly the same estimator (and hence the same parameters). In fact, such stochasticity depends on the amount regularisation, and to make this point, we relax it with `alpha=50` (instead of 100).

```
[24]: %%time
estimator_ = make_pipeline(
    StandardScaler(with_mean=False),
    MLPRegressor(
        hidden_layer_sizes=(6,),
        learning_rate_init=0.5,
```

```

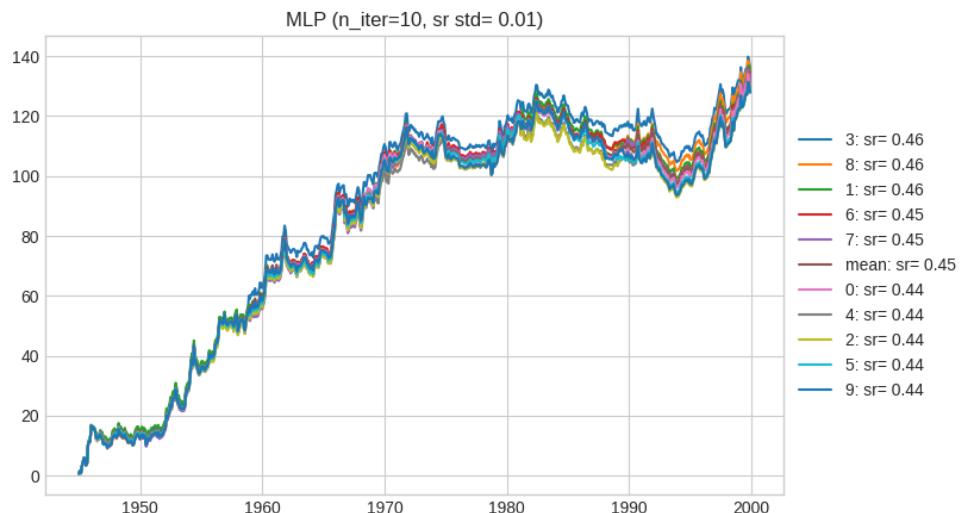
        alpha=50,
        n_iter_no_change=25,
        solver="adam",
        tol=1e-2,
        activation="logistic",
    ),
    MeanVariance(),
)
n_iter = 10
pnls_ = {}
for i in range(n_iter):
    pnls_[i] = Backtester(estimator_).train(features, target, ret)

sr_std = np.std([v.pipe(sharpe_ratio) for k, v in pnls_.items()])

```

CPU times: user 4min 32s, sys: 9min 20s, total: 13min 53s
Wall time: 1min 45s

```
[25]: title = f"MLP (n_iter={n_iter}, sr std= {sr_std:.2f})"
line(
    pd.concat(pnls_, axis=1).assign(mean=lambda x: x.mean(axis=1)),
    cumsum=True,
    title=title,
)
```



Chapter 7

Balancing Overfitting and Underfitting: Cross-Validation and Parameter Optimization

In this section, we cover the topic of overfitting. A folk theorem in asset management is that people are so afraid of overfitting that they tend to (massively) underfit. Or at least, that was the case. Today, better fitting models to extract as much information from a dataset has become a crucial skill.

More precisely, overfitting a particular dataset provides a baseline for how well a system can learn (e.g. see the Recipe for Training Neural Nets by Andrej Karpathy).

7.1 Ridge CV

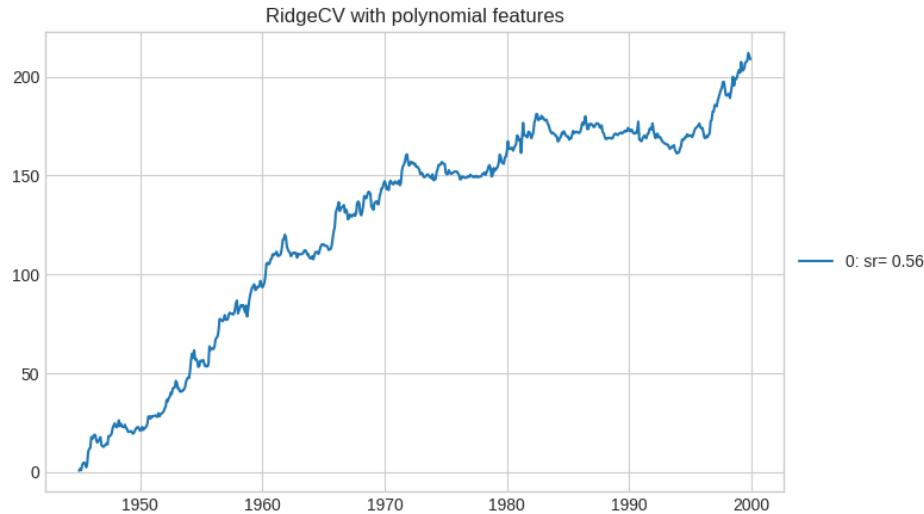
A first strategy is to use estimators that embed some form of cross-validation like RidgeCV. K-fold cross validation is described as follows:

Take a model with parameter s (e.g. the Ridge with tuning parameter `alpha`):

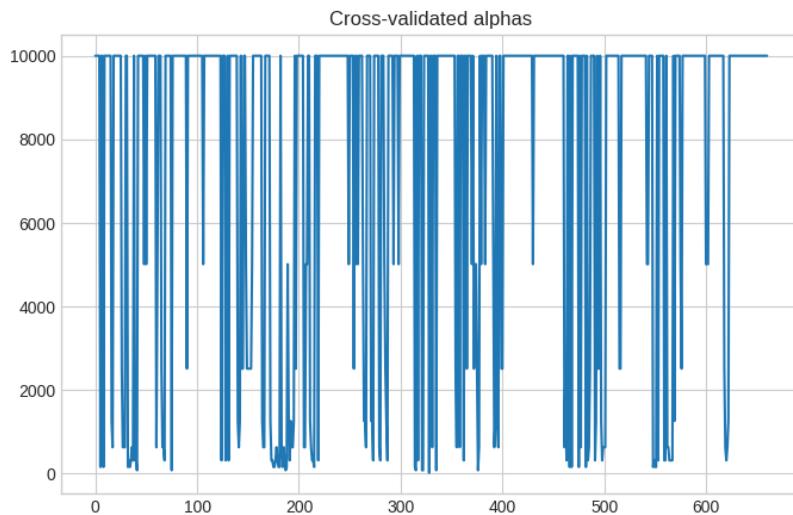
1. divide the data into K roughly equal parts ($K = 5$ or $K = 10$)
2. for each $k \in \{1, 2, \dots, K\}$ fit the model with parameter s to the other $K - 1$ parts and compute its error $E_k(s)$ in predicting the k -th part.
3. the overall cross-validation error is then $CV(s) = \frac{1}{K} \sum_{k=1}^K E_k(s)$.
4. do this for many values of s and choose the value of s that minimize $CV(s)$

```
[2]: alphas = np.exp(np.arange(np.log(10), np.log(10001), (np.log(10000) - np.log(10)) / 10))
```

```
[3]: estimator = make_pipeline(  
    StandardScaler(with_mean=False),  
    PolynomialFeatures(degree=2),  
    RidgeCV(alphas=alphas, cv=5),  
    MeanVariance(),  
)  
  
m = Backtester(estimator).compute_holdings(features, target).compute_pnl(ret)  
line(m.pnl_, cumsum=True, title="RidgeCV with polynomial features")
```



```
[4]: line(
    pd.Series([m[2].alpha_ for m in m.estimators_]),
    title="Cross-validated alphas",
    legend=False,
)
```



In this example, the fitted alphas over rolling windows are not very stable (probably given the small rolling windows used here).

7.2 Random parameter search for Lightgbm

We first compute a Lightgbm benchmark with the fixed baseline parameters used in a previous section.

Some resources on the parameters of Lightgbm can be found in its documentation:

- <https://lightgbm.readthedocs.io/en/latest/Parameters.html#learning-control-parameters>

```
[5]: estimator = make_pipeline(
    MultiLGBMRegressor(min_child_samples=5, n_estimators=25), MeanVariance()
)

pnl_lgb = {"fixed_params": Backtester(estimator).train(features, target, ret)}
```

We now do a search with random parameters drawn from predetermined distributions:

- the random parameter generators come from the `scipy.stats` module – in particular `randint`, `uniform` and `loguniform`.
- we use the `scikit-learn` function `ParameterSampler` as wrapper.

Setup:

- the objective is to maximize the sharpe ratio over the early period 1945 to 1972 (as the train period).
- the evaluation is the performance of the backtest over the 1972-to-2000 period (as the test period).

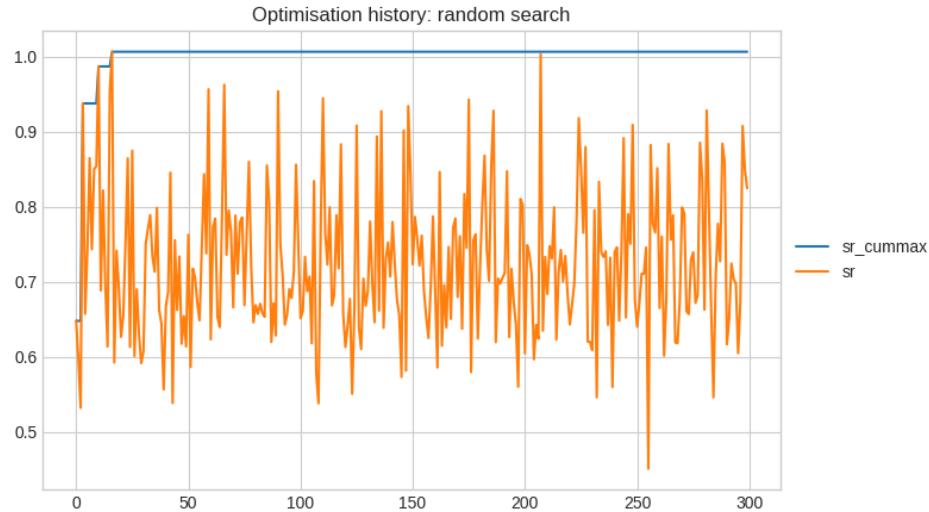
```
[6]: from scipy.stats import loguniform, randint, uniform
from sklearn.model_selection import ParameterSampler
```

```
[7]: n_iter = 300
start_date = "1945-01-01"
end_date = "1972-04-01"
param_distributions = {
    "max_depth": randint(3, 10),
    "num_leaves": randint(2, 2**8),
    "n_estimators": randint(5, 50),
    "min_split_gain": uniform(0, 1.0),
    "min_child_samples": randint(1, 5),
    "reg_lambda": loguniform(1e-8, 1.0),
    "reg_alpha": loguniform(1e-8, 1.0),
}
```

```
[8]: force_recompute = False
cache_dir = Path(os.getcwd()) / "cache"
if not cache_dir.is_dir():
    os.makedirs(cache_dir)

fname = cache_dir / "hpo_lgb.parquet"
if (force_recompute) | (not fname.exists()):
    results_ = []
    for i, prm in enumerate(
        ParameterSampler(param_distributions=param_distributions, n_iter=n_iter)
    ):
        estimator = make_pipeline(MultiLGBMRegressor(**prm), MeanVariance())
        pnl_ = Backtester(estimator, end_date=end_date).train(features, target, ret)
        prm.update({"sr": pnl_.pipe(sharpe_ratio)})
        results_[i] = pd.Series(prm)
    results = pd.DataFrame.from_dict(results_, orient="index").sort_values("sr")
    results.to_parquet(fname)
else:
    results = pd.read_parquet(fname)
```

```
[9]: df = results.sort_index()[[["sr"]]].assign(sr_cummax=lambda x: x.sr.cummax())
line(df, title="Optimisation history: random search")
```



Using the sharpe ratio statistics presented in a previous section, we can compute a standard error around the maximum sharpe ratio:

```
[10]: sr_max = results.iloc[-1]["sr"] / np.sqrt(12)
sr_std = np.sqrt(12) * np.sqrt((1 + 0.5 * sr_max**2) / len(ret[start_date:end_date]))
sr_range = results["sr"].pipe(lambda x: x.max() - x.min())
print(
    f"The sharpe ratio standard deviation at the maximum sharpe ratio (of {sr_max * np.sqrt(12):.2f}) is {sr_std:.2f}"
)
print(f"The range of the sharpe ratios in the random search is {sr_range:.2f}")
```

The sharpe ratio standard deviation at the maximum sharpe ratio (of 1.01) is 0.20

The range of the sharpe ratios in the random search is 0.56

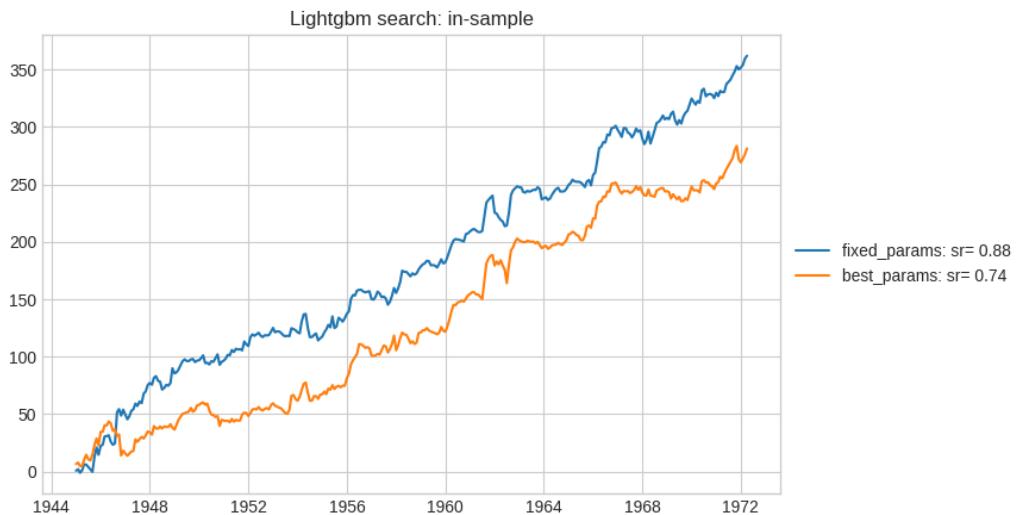
```
[11]: best_params = results.drop("sr", axis=1).iloc[-1].to_dict()
best_params["num_leaves"] = int(best_params["num_leaves"])
best_params["max_depth"] = int(best_params["max_depth"])
best_params["min_child_samples"] = int(best_params["min_child_samples"])
best_params["n_estimators"] = int(best_params["n_estimators"])
best_params
```

```
[11]: {'max_depth': 3,
'min_child_samples': 4,
'min_split_gain': 0.4620819529118264,
'n_estimators': 13,
'num_leaves': 232,
'reg_alpha': 0.008802935155703386,
'reg_lambda': 0.14660332145137617}
```

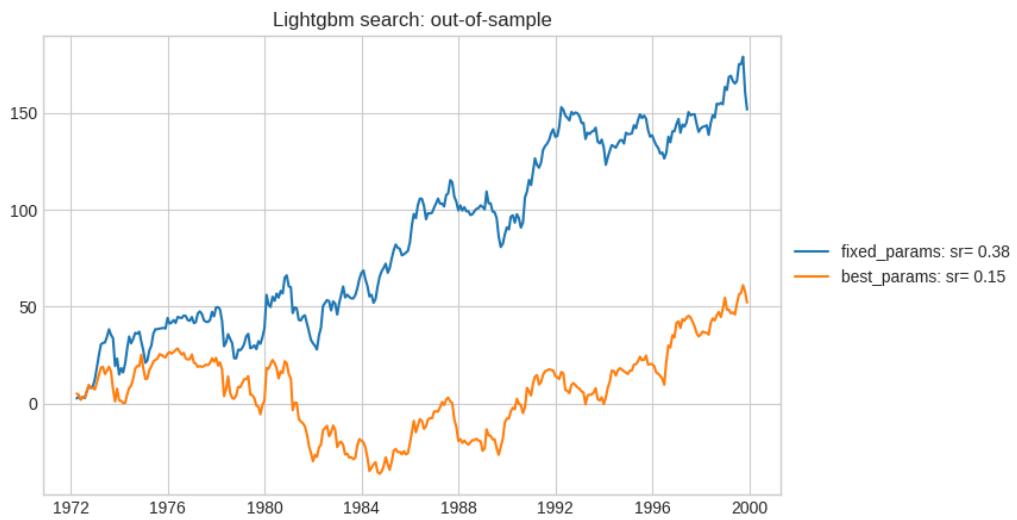
```
[12]: estimator = make_pipeline(MultiLGBMRegressor(**best_params), MeanVariance())
pnl_lgb["best_params"] = Backtester(estimator).train(features, target, ret)
```

Chapter 7. Balancing Overfitting and Underfitting: Cross-Validation and Parameter Optimization

```
[13]: line(
    {k: v.loc[start_date:end_date] for k, v in pnl_lgb.items()},
    cumsum=True,
    title="Lightgbm search: in-sample",
)
```



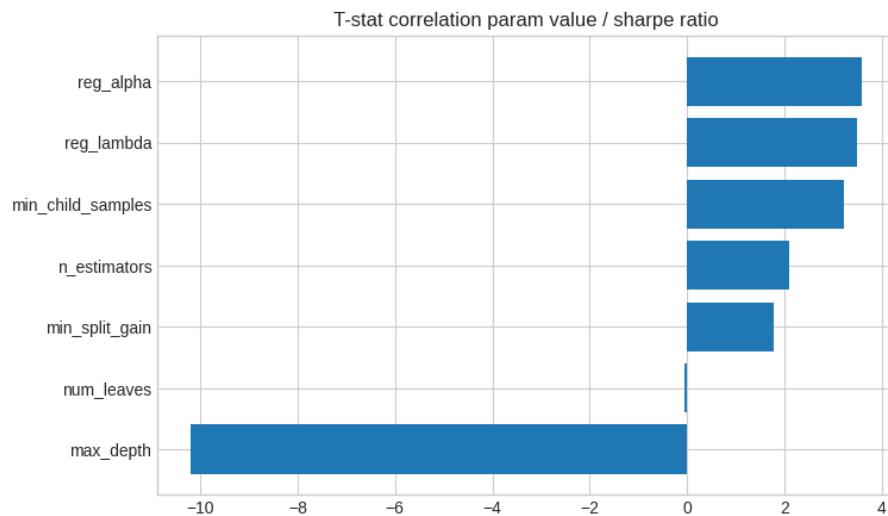
```
[14]: line(
    {k: v.loc[end_date:] for k, v in pnl_lgb.items()},
    cumsum=True,
    title="Lightgbm search: out-of-sample",
)
```



What are the parameters that are correlated with the sharpe ratio?

```
[15]: bar(
    results.corr()["sr"].mul(np.sqrt(n_iter)).drop("sr"),
    title="T-stat correlation param value / sharpe ratio",
)
```

```
    horizontal=True,
)
```



To assess more precisely the impact of parameters on the sharpe ratio, we run a regression.

```
[16]: from statsmodels import api
m = api.OLS(results["sr"], api.add_constant(results.drop("sr", axis=1))).fit()
```

```
[17]: m.summary()
```

```
[17]: <class 'statsmodels.iolib.summary.Summary'>
"""
                OLS Regression Results
=====
Dep. Variable:                  sr   R-squared:                 0.484
Model:                          OLS   Adj. R-squared:            0.472
Method: Least Squares   F-statistic:                 39.11
Date: Mon, 29 Sep 2025   Prob (F-statistic):        1.52e-38
Time: 17:44:23             Log-Likelihood:           368.57
No. Observations:              300   AIC:                   -721.1
Df Residuals:                  292   BIC:                   -691.5
Df Model:                      7
Covariance Type:               nonrobust
=====
=====
```

	coef	std err	t	P> t	[0.025
0.975]					
-----	-----	-----	-----	-----	-----
const	0.7920	0.021	37.522	0.000	0.750
0.834					
max_depth	-0.0279	0.002	-13.739	0.000	-0.032
-0.024					
min_child_samples	0.0155	0.004	4.023	0.000	0.008
0.023					
min_split_gain	0.0480	0.015	3.209	0.001	0.019
0.078					
n_estimators	0.0009	0.000	2.763	0.006	0.000

Chapter 7. Balancing Overfitting and Underfitting: Cross-Validation and Parameter Optimization

```
0.002
num_leaves      2.155e-05   5.57e-05    0.387    0.699   -8.81e-05
0.000
reg_alpha       0.1476     0.027     5.457    0.000    0.094
0.201
reg_lambda      0.0920     0.022     4.189    0.000    0.049
0.135
=====
Omnibus:           1.872   Durbin-Watson:          0.891
Prob(Omnibus):    0.392   Jarque-Bera (JB):        1.713
Skew:              -0.049  Prob(JB):                  0.425
Kurtosis:          3.357  Cond. No.            1.03e+03
=====
```

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
 - [2] The condition number is large, 1.03e+03. This might indicate that there are strong multicollinearity or other numerical problems.
- """

Chapter 8

Eigenvalue Decomposition and Risk Bias in Portfolio Covariance Estimation

This chapter explores quantitative approaches to measuring financial risk, with a focus on techniques for estimating and evaluating the risk in return distributions. Through analysis of covariance matrices, eigenvalue decomposition, and out-of-sample testing, it highlights both the capabilities and limitations of common risk measurement methods.

8.1 Risk bias metrics

For a given portfolio h_Φ using the covariance V_Φ , the metric that we use is the **risk-bias** given by

$$\text{RiskBias}_\Phi = \text{Std} \left[\frac{h_\Phi^T r}{\sqrt{h^T V_\Phi h}} \right] - 1,$$

where the variance is evaluated over empirical returns.

The returns r_t are over N assets (so that r_t is a vector of size N). The covariance of returns is:

$$V = \text{Var}(r_t).$$

Performing a singular value decomposition of V yields:

$$V = USU^T,$$

where U contains the eigenvectors and S is a diagonal matrix with the eigenvalues. Each eigenvector can be considered as the weight of factor and we construct the factor returns as:

$$f_t = U^T r_t.$$

Lemma: the factors f_t are uncorrelated and their volatility is by the eigenvalue.

Proof.

$$\text{Var}(f_t) = U^T \text{Var}(r_t) U = U^T U S U^T U = S.$$

Lemma: the eigenvector associated to the largest eigenvalue maximizes $u^T V u$ such that $u^T u = 1$.

Proof. Introducing the Lagrange multiplier ξ on the constraint, the first-order condition is

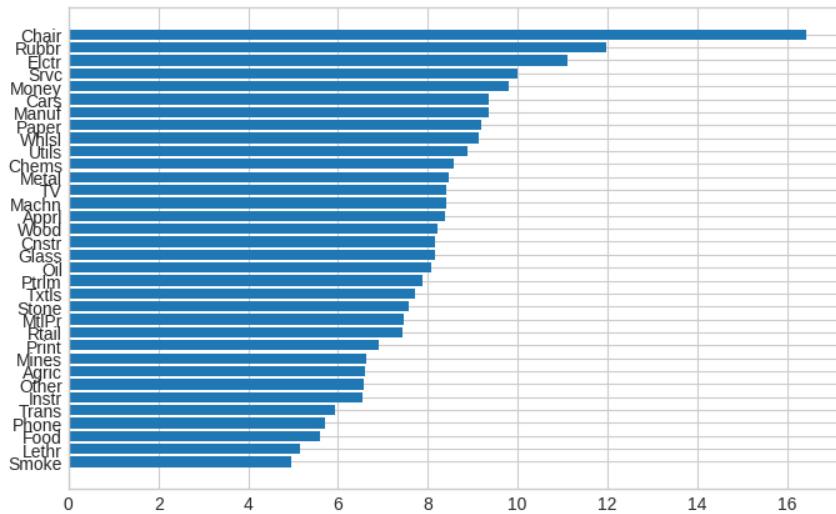
$$Vu = \xi u,$$

so that u is an eigenvector and the value of the objective is the eigenvalue associated to u . So the objective is maximized for the largest eigenvalue.

Corollary: the eigenvector associated to the smallest eigenvalue minimizes $u^T Vu$ such that $u^T u = 1$.

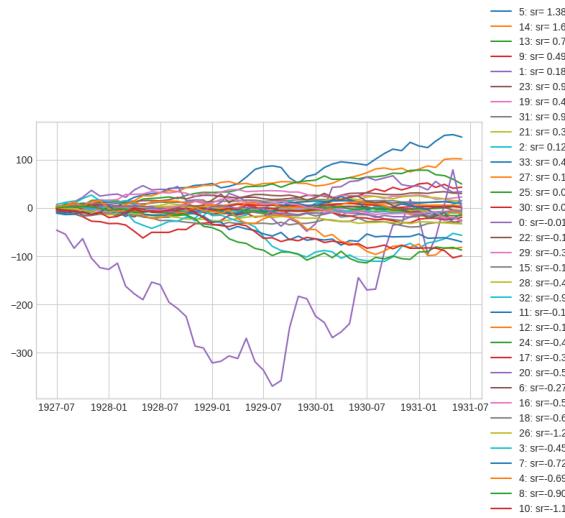
8.2 Return covariance eigenvalues

```
[2]: T = 60
X = ret.iloc[:T].dropna(how="all", axis=1)
bar(X.std().sort_values(), horizontal=True)
```



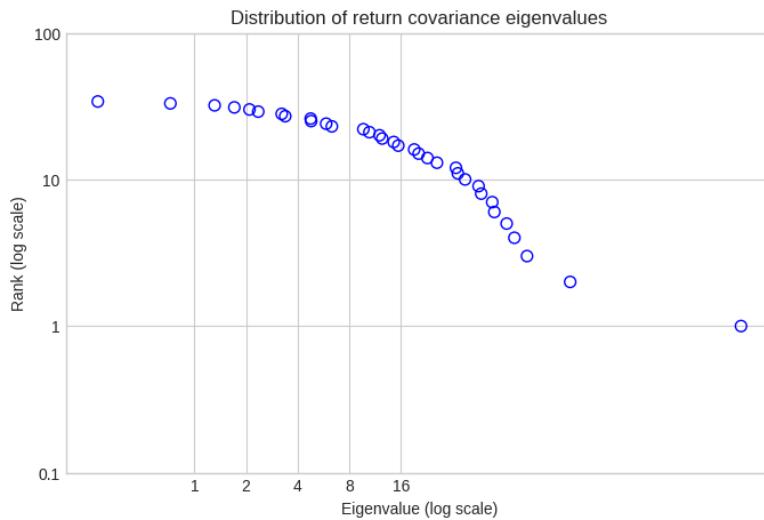
```
[3]: u, s, _ = np.linalg.svd(X.cov())
mode_returns = X.dot(u)
```

```
[4]: line(mode_returns, cumsum=True, legend=True)
```



The graph below shows that the largest eigenvalue is multiple-order of magnitude larger than the smallest one.

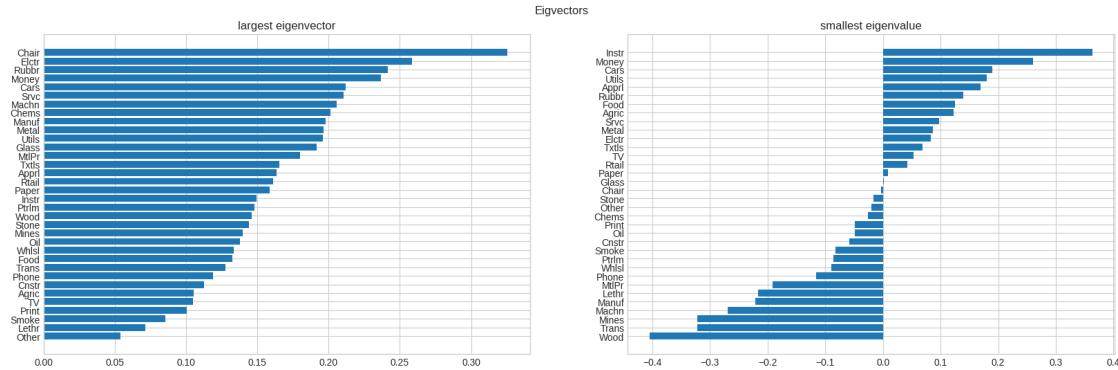
```
[5]: df = pd.Series(s, np.arange(1, X.shape[1] + 1))
scatter(
    df,
    xscale="log",
    yscale="log",
    xlabel="Eigenvalue (log scale)",
    ylabel="Rank (log scale)",
    xticks=[1, 2, 4, 8, 16],
    yticks=[0.1, 1, 10, 100],
    title="Distribution of return covariance eigenvalues",
)
```



```
[6]: print(f"The ratio of the largest to the smallest eigenvalue is {s[0] / s[-1]:.1f}")
```

The ratio of the largest to the smallest eigenvalue is 5671.4

```
[7]: d = {
    "largest eigenvector": pd.Series(u[:, 0] / np.sign(np.mean(u[:, 0])), X.columns),
    "smallest eigenvalue": pd.Series(u[:, -1] / np.sign(np.mean(u[:, 1])), X.columns),
}
fig, ax = plt.subplots(1, 2, figsize=(20, 6))
fig.suptitle("Eigenvectors", y=0.95)
for i, (k, v) in enumerate(d.items()):
    bar(v, title=k, ax=ax[i], horizontal=True)
```



The Lemma and Corollary above show that the eigenvalues measure the *in-sample* variance of a mode. But how well does the in-sample variance predicts the out-of-sample variance?

To test assess this point, we construct the pnls of modes (defined as the portfolio with the eigenvectors as positions), normalized by the ex-ante standard deviation (as the square-root of the eigenvalue) and signed so that the in-sample pnl is positive.

```
[8]: from sklearn.model_selection import TimeSeriesSplit

start_date = "1945-01-01"
test_size = 1
params = dict(max_train_size=T, test_size=test_size, gap=0)
params["n_splits"] = 1 + len(ret.loc[start_date:]) // test_size

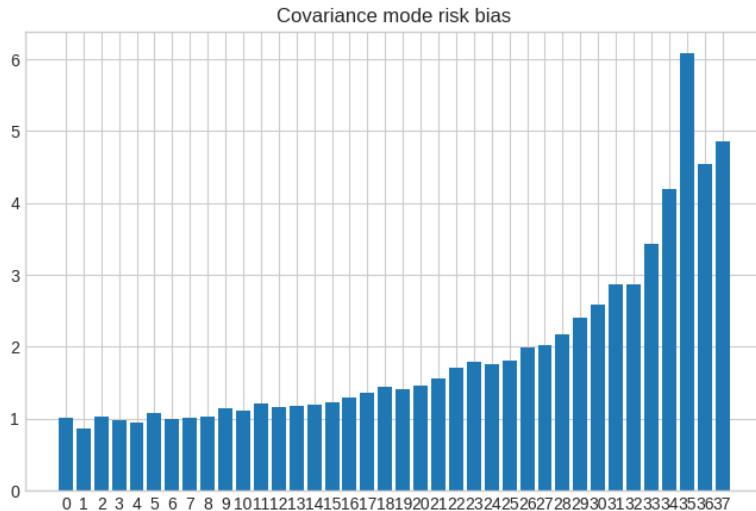
cv = TimeSeriesSplit(**params)
```

```
[9]: mode_pnl = []
for train, test in cv.split(ret):
    V_ = ret.fillna(0).iloc[train].cov()
    u, s, _ = np.linalg.svd(V_)
    retain_max = np.sum([v > 1e-5 for v in s])
    s[retain_max:] = np.nan * s[retain_max:]
    mu = ret.iloc[train].fillna(0).dot(u).mean()
    mode_pnl += [ret.fillna(0).iloc[test].dot(u).mul(np.sign(mu)).div(np.sqrt(s))]

mode_pnl = pd.concat(mode_pnl)
```

The graph below shows the out-of-sample risk of each mode pnl which has been rescaled to unit ex-ante risk (so that the natural baseline is 1). This metric is called a **risk bias** and will be defined formally in the next section. We see that for the first largest modes, the risk bias is close to 1, so that the ex-ante risk measures well the out-of-sample risk. However, for the smallest modes, this ex-ante measurse is completely off. For the smallest modes, the positions “overfit” information from the covariance matrix (in particular the correlation) and it is intuitive that the small in-sample risk estimates mean-revert to larger out-of-sample volatility.

```
[10]: bar(mode_pnl.std(), sort=False, title="Covariance mode risk bias")
```



8.3 The Marcenko-Pastur pdf

The Marcenko-Pastur probability density function (pdf) describes the theoretical distribution of eigenvalues of large random covariance matrices, which is particularly useful in the context of financial risk analysis. It defines the range and shape of the eigenvalue spectrum under certain assumptions, allowing one to distinguish between noise and meaningful signals in empirical covariance matrices.

```
[11]: import matplotlib.pyplot as plt

def marcenko_pastur_pdf(x, q, sigma=1):
    """Marcenko-Pastur Probability Density Function (PDF)."""
    b = sigma**2 * (1 + np.sqrt(1 / q)) ** 2
    a = sigma**2 * (1 - np.sqrt(1 / q)) ** 2
    if a < 0:
        a = 0
    pdf = np.zeros_like(x, dtype=float)

    mask = (x >= a) & (x <= b)
    pdf[mask] = (1 / (2 * np.pi * sigma**2 * x[mask] * q)) * np.sqrt(
        (b - x[mask]) * (x[mask] - a)
    )

    return pdf
```

```
[12]: T, N = X.shape

V_ = X.iloc[:T].cov()
u, s, _ = np.linalg.svd(V_)

plt.hist(s, bins=30, density=True, alpha=0.6, color="g", label="Empirical histogram")

# Compute the Marcenko-Pastur distribution
q = N / T
x = np.linspace(s.min(), s.max(), 100)
```

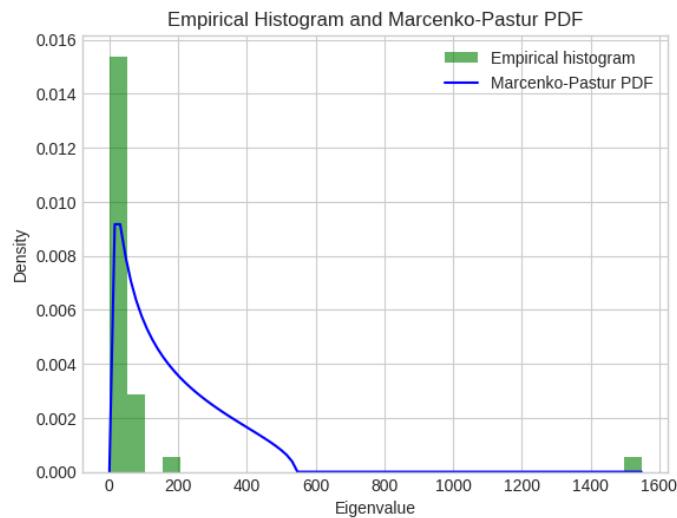
```

mp_pdf = marcenko_pastur_pdf(x, q, sigma=10)

# Plot the Marcenko-Pastur distribution
plt.plot(x, mp_pdf, color="blue", label="Marcenko-Pastur PDF")

# Add labels and legend
plt.xlabel("Eigenvalue")
plt.ylabel("Density")
plt.title("Empirical Histogram and Marcenko-Pastur PDF")
plt.legend()
plt.grid(True)

```



8.4 Understanding covariance shrinkage

There are several methods to shrink a covariance matrix. A first method shrinks the sample covariance matrix V toward its diagonal, reducing the influence of off-diagonal (correlation) terms, thus emphasizing variances over covariances:

$$V_\omega = \omega \times \text{Diag}(V) + (1 - \omega) \times V.$$

A second method used by the `ShrunkCovariance` method in `sklearn` shrinks V toward a scaled identity matrix, where all variables are assumed to have equal variance and zero covariance, effectively imposing a model of equal risk and no correlation.

$$V_\omega = \omega \frac{\text{Tr}(V)}{N} \times I_N + (1 - \omega) \times V.$$

The primary difference is that the first approach retains individual variances while discarding correlations, whereas the second approach replaces both the variances and covariances with a uniform average variance across all assets.

```
[13]: from sklearn.covariance import ShrunkCovariance

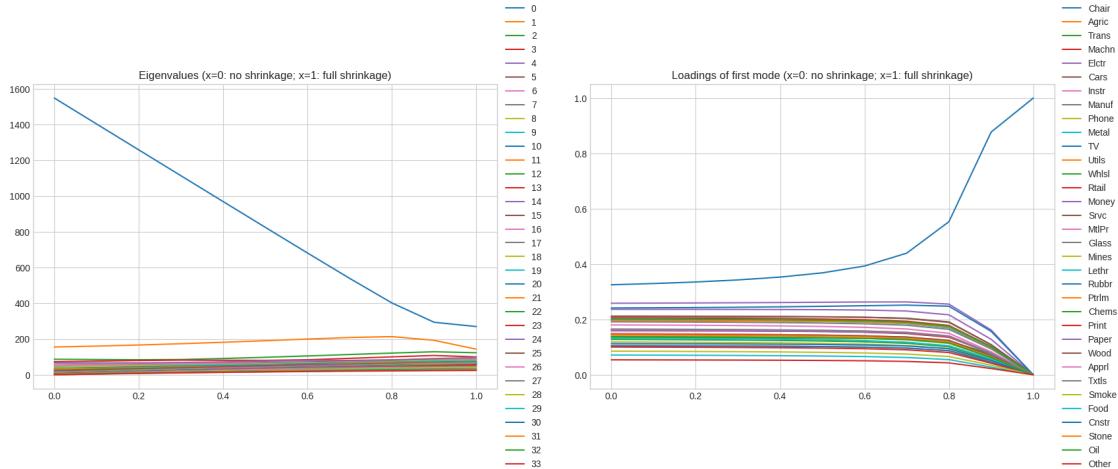
def simple_shrunk_covariance(x, shrinkage):
```

```
v = np.cov(x.T)
return shrinkage * np.diag(np.diag(v)) + (1 - shrinkage) * v
```

```
[14]: S = {}
U0 = {}
for shrinkage in np.arange(0, 1.1, 0.1):
    V_ = shrinkage * np.diag(np.diag(X.cov())) + (1 - shrinkage) * X.cov()
    u, s, _ = np.linalg.svd(V_)
    S[shrinkage] = s
    U0[shrinkage] = u[:, 0] * np.sign(np.mean(u[:, 0]))
S = pd.DataFrame.from_dict(S, orient="index")
U0 = pd.DataFrame.from_dict(U0, orient="index").rename(
    columns={i: c for i, c in enumerate(X.columns)})
)

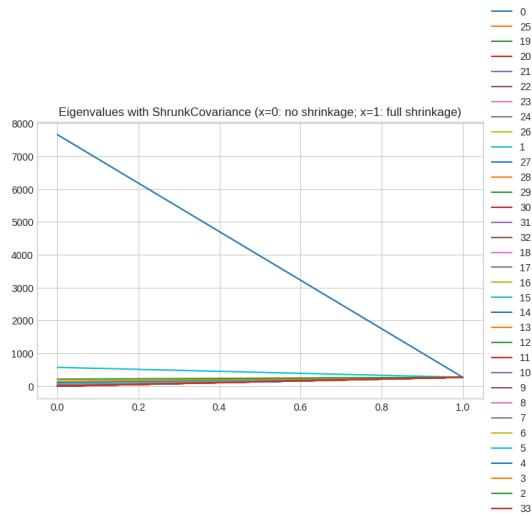
fig, ax = plt.subplots(1, 2, figsize=(20, 6))
line(S, title="Eigenvalues (x=0: no shrinkage; x=1: full shrinkage)", ax=ax[0])

line(
    U0,
    title="Loadings of first mode (x=0: no shrinkage; x=1: full shrinkage)",
    ax=ax[1],
)
```



```
[15]: S = {}
for shrinkage in np.arange(0, 1.01, 0.1):
    V_ = ShrunkenCovariance(shrinkage=shrinkage).fit(X.cov()).covariance_
    _, s, _ = np.linalg.svd(V_)
    S[shrinkage] = s
S = pd.DataFrame.from_dict(S, orient="index")

line(
    S,
    title="Eigenvalues with ShrunkenCovariance (x=0: no shrinkage; x=1: full shrinkage)",
)
```



Chapter 9

Shrinkage Estimators and Covariance Matrix Regularization in Portfolio Risk Modeling

A key ingredient of portfolio construction is the ability to predict portfolio risk (in particular, with a risk-model) to be able to properly size the positions.

In this section, we discuss different ways to estimate risk. More precisely, for the empirical covariance matrix V , there might be transformation $\Phi : V \mapsto V_\Phi$ that improve the forward-looking estimates (and the portfolio construction).

The risk-model is defined here as the covariance of returns V . To understand its impact on the backtest, it is important to remember that in the mean-variance optimisation, it is the inverse of the covariance matrix V^{-1} that is used.

Viewed from the point of view of a singular value decomposition, the smallest eigenvalues of V are not only estimated with noise, but their impact is magnified in V^{-1} , leading to potentially significant noise in the estimate of positions.



Available at
WWW.MATHEMATICSWEB.ORG
POWERED BY SCIENCE @ DIRECT*

Journal of Multivariate Analysis 88 (2004) 365–411

Journal of
**Multivariate
Analysis**

<http://www.elsevier.com/locate/jmva>

A well-conditioned estimator for
large-dimensional covariance matrices

Olivier Ledoit^{a,b} and Michael Wolf^{c,*}¹

^aAnderson Graduate School of Management, UCLA, USA

^bEquities Division, Credit Suisse First Boston, USA

^cDepartment of Economics and Business, Universitat Pompeu Fabra, Ramon Trias Fargas 25–27, 08005
Barcelona, Spain

Received 26 February 2001

The insight of Ledoit and Wolf (2004) is to use a weighted average of two matrices to reduce estimation error

- the empirical covariance matrix V is asymptotically an unbiased estimated – but with a slow convergence for small samples
- there are biased estimators but with a faster rate of convergence — for instance the diagonal $\text{Diag}(V)$ of V – and on smaller samples, such biased estimators can be more efficient than the unbiased ones
- The covariance matrix used in the portfolio optimisation is

$$V_\omega = \omega \times \text{Diag}(V) + (1 - \omega) \times V.$$

How to determine ω ? Ledoit and Wolf (2004) minimize a norm that applies to matrices (Frobenius norm). In what follows, we test different shrinkage values.

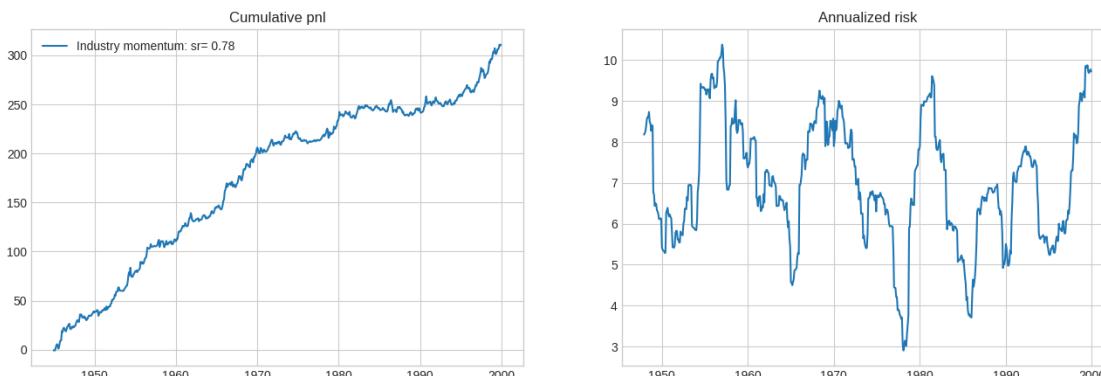
9.1 Risk in the industry momentum backtest

```
[3]: from skfin.metrics import drawdown, sharpe_ratio
```

We first compute the Industry momentum benchmark.

```
[4]: m = Backtester(MeanVariance()).compute_holdings(features, target).compute_pnl(ret)
h0, pnl0, estimators0 = m.h_, m.pnl_, m.estimators_
```

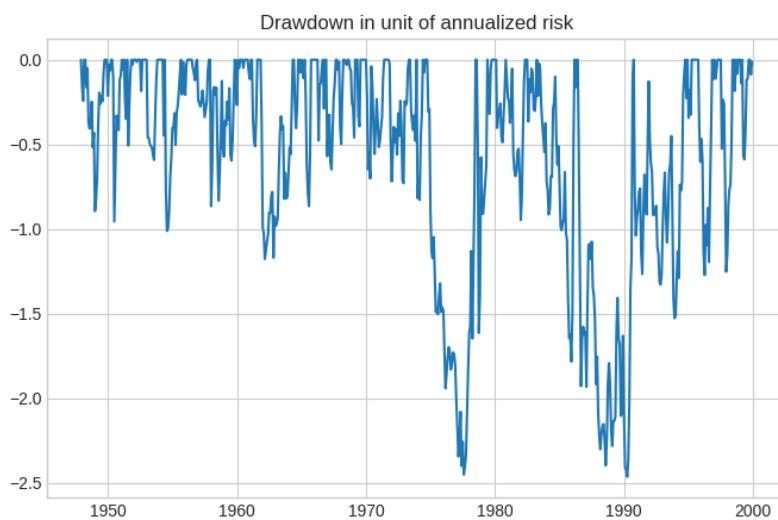
```
[5]: fig, ax = plt.subplots(1, 2, figsize=(16, 5))
line(
    pnl0.rename("Industry momentum"),
    cumsum=True,
    loc="best",
    title="Cumulative pnl",
    ax=ax[0],
)
line(
    pnl0.rolling(36).std().mul(np.sqrt(12)),
    title="Annualized risk",
    legend=False,
    ax=ax[1],
)
```



The definition of a drawdown (in unit of annualized risk) is:

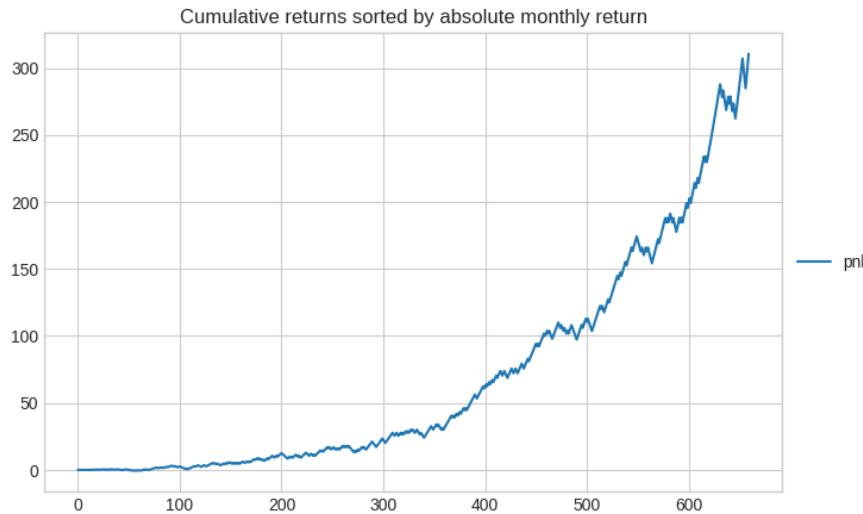
$$dd_t = \frac{\sum_{s=0}^{s=t} pnl_s - \text{Max}_{\tau} (\sum_{s=0}^{s=\tau} pnl_s)}{\text{annualized_factor} \times \sqrt{\text{Var}[pnl_s | s \leq t]}}.$$

```
[6]: line(
    pnl0.pipe(drawdown),
    title="Drawdown in unit of annualized risk",
    legend=False,
    figsize=(8, 5),
)
```



The following graph shows that on the period up to 2000, large absolute returns tend to be positive. It turns out that in the following period, the pnl of Momentum becomes left-skewed with large negative returns. For instance, February/March 2009 is a famous example of a Momentum drawdown.

```
[7]: line(
    pnl0.rename("pnl")
    .to_frame()
    .assign(pnl_abs=lambda x: x.pnl.abs())
    .sort_values("pnl_abs")
    .reset_index(drop=True)[["pnl"]],
    cumsum=True,
    title="Cumulative returns sorted by absolute monthly return",
    legend_sharpe_ratio=False,
)
```

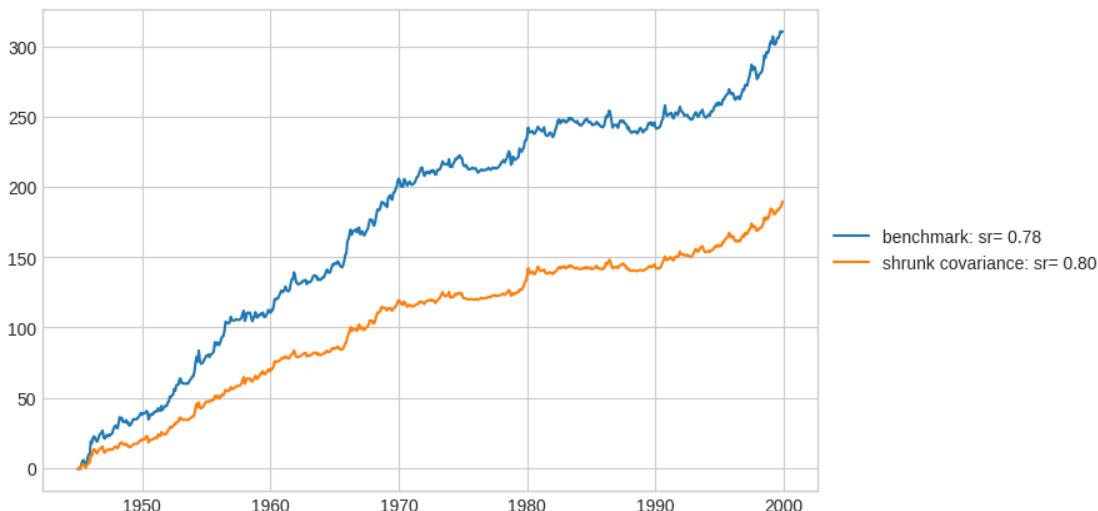


9.2 Risk model estimation

```
[8]: from sklearn.covariance import LedoitWolf, ShrunkCovariance
```

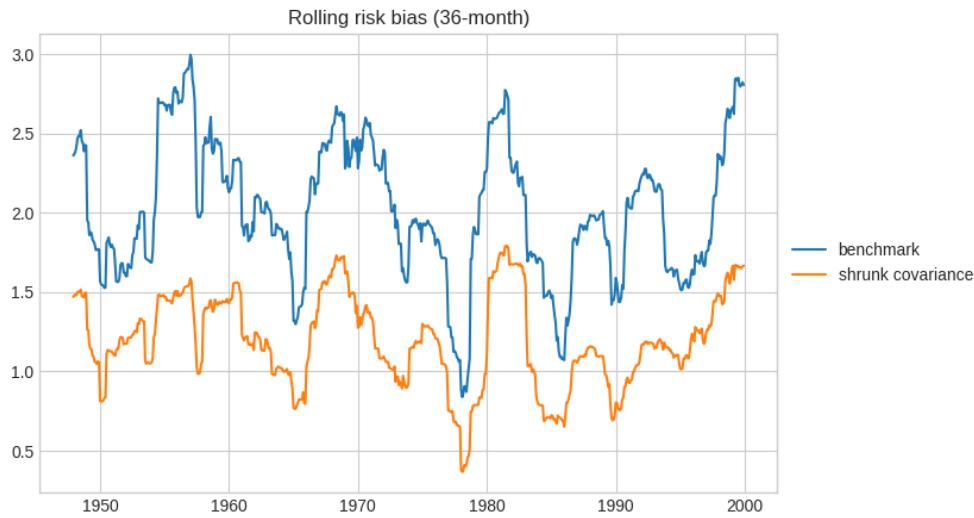
The default value of the shrinkage parameter for `ShrunkCovariance` is 0.1. When `shrinkage=0`, there is no shrinkage and when `shrinkage=1`, all the off-diagonal terms are set to zero and the covariance matrix is diagonal.

```
[9]: transform_V_ = lambda x: ShrunkCovariance(shrinkage=0.1).fit(x).covariance_
m = (
    Backtester(MeanVariance(transform_V=transform_V_))
    .compute_holdings(features, target)
    .compute_pnl(ret)
)
h, pnl, estimators = m.h_, m.pnl_, m.estimators_
line({"benchmark": pnl0, "shrunk covariance": pnl}, cumsum=True)
```



The estimation of risk with the shrunk covariance is much closer to the ex-ante risk (of 1).

```
[10]: line(
    {"benchmark": pnl0.rolling(36).std(), "shrunk covariance": pnl.rolling(36).std()},
    title="Rolling risk bias (36-month)",
)
```



The ratio of the largest to the smallest eigenvalue is an order of magnitude smaller for the backtest with the shrunk covariance relative to the benchmark.

```
[11]: for m in estimators0:
    break

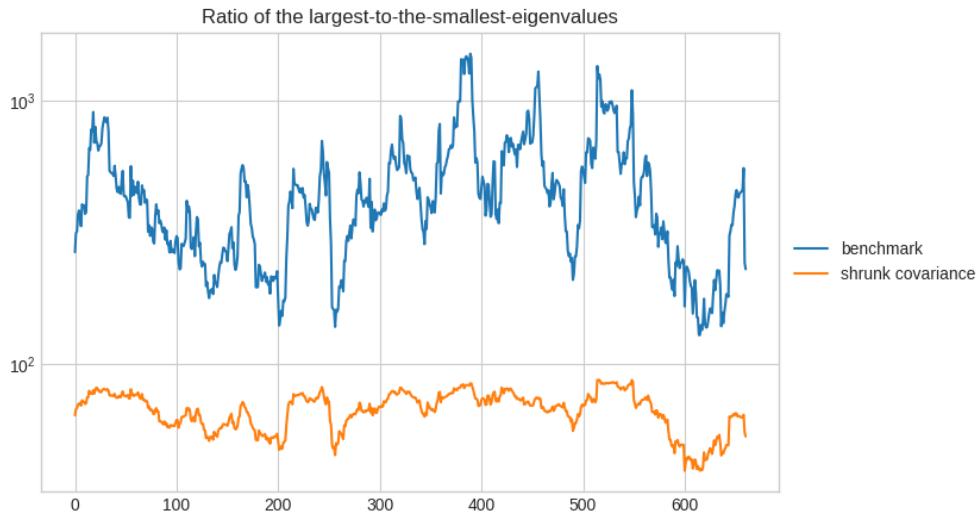
[12]: s = np.linalg.svd(np.diag(np.diag(m.V_)), compute_uv=False)

[13]: get_eigenvalues = lambda estimators: pd.DataFrame(
    [np.linalg.svd(m.V_, compute_uv=False) for m in estimators]
)

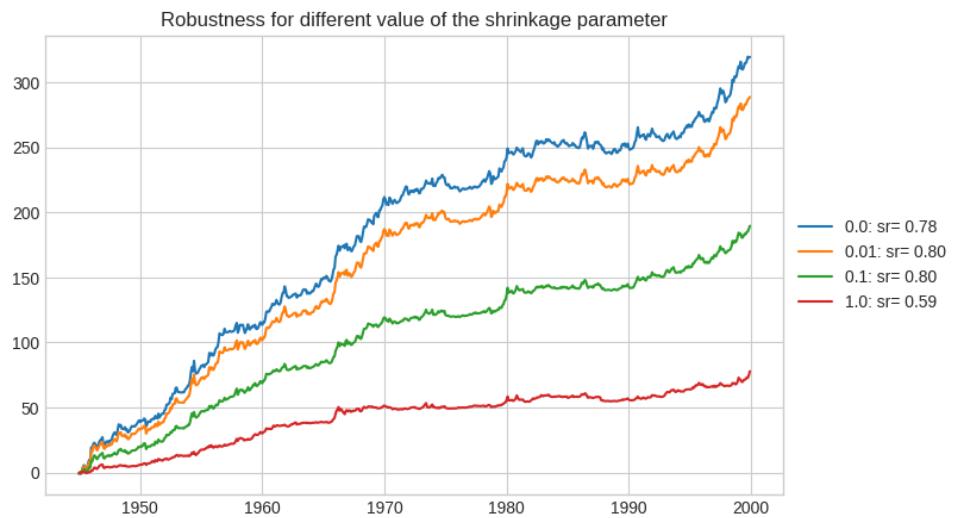
ratio_largest_smallest_eigenvalue = lambda x: x.pipe(
    lambda x: x.iloc[:, 0] / x.iloc[:, -1]
)

eigenvalues0 = get_eigenvalues(estimators0)
eigenvalues = get_eigenvalues(estimators)

line(
{
    "benchmark": eigenvalues0.pipe(ratio_largest_smallest_eigenvalue),
    "shrunk covariance": eigenvalues.pipe(ratio_largest_smallest_eigenvalue),
},
yscale="log",
title="Ratio of the largest-to-the-smallest-eigenvalues",
)
```



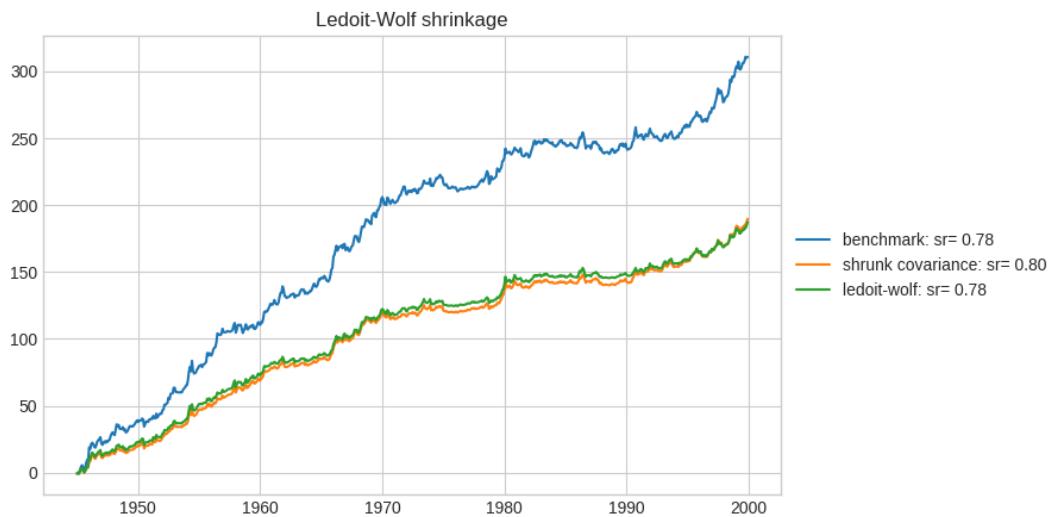
```
[14]: pnls = {}
for shrinkage in [0, 0.01, 0.1, 1]:
    transform_V_ = lambda x: ShrunkCovariance(shrinkage=shrinkage).fit(x).covariance_
    estimator = MeanVariance(transform_V=transform_V_)
    pnls[shrinkage] = Backtester(estimator).train(features, target, ret)
line(
    pnls, cumsum=True, title="Robustness for different value of the shrinkage parameter"
)
```



A related shrinkage is to use the LedoitWolf method to determine the shrinkage and it yield similar performance.

```
[15]: transform_V_ = lambda x: LedoitWolf().fit(x).covariance_
estimator = MeanVariance(transform_V=transform_V_)
pnl_ = Backtester(estimator).train(features, target, ret)
line(
    {"benchmark": pnl0, "shrunk covariance": pnl, "ledoit-wolf": pnl_},
```

```
cumsum=True,  
title="Ledoit-Wolf shrinkage",  
)
```



The key empirical point is that the sharpe ratio is maximized for a covariance that involves a small amount of shrinkage.

Chapter 10

Understanding Portfolio Leverage: Techniques, Risks, and Optimization Constraints

10.1 Methods of Implementing Leverage

Portfolio leverage refers to the use of various financial techniques and instruments to amplify the potential returns of an investment portfolio. Essentially, it involves borrowing funds or using derivative instruments to increase the size of a position beyond the actual capital invested. This approach allows investors to potentially enhance returns, but it also increases risk, as the potential for losses is also magnified.

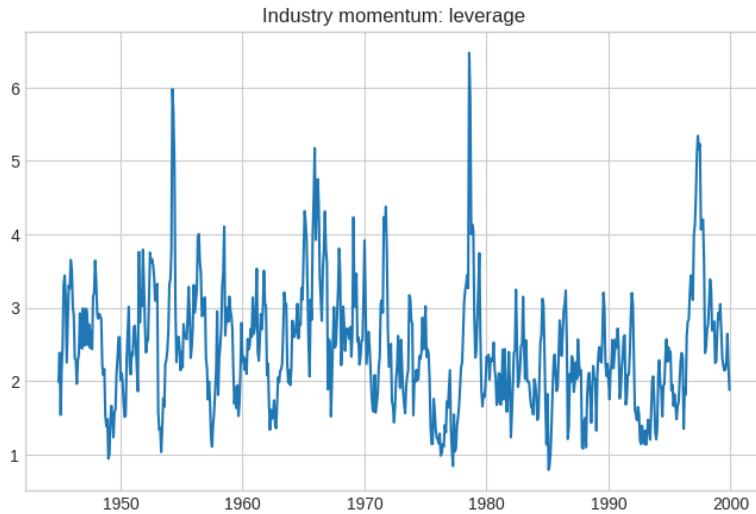
Leverage can be achieved through several means, including:

1. Margin Trading: Borrowing money from a broker to buy more securities than one could with available cash alone.
2. Derivatives: Using instruments like options, futures, and swaps that provide exposure to an underlying asset at a fraction of the asset's cost.
3. Borrowing: Taking out loans to invest more capital into the portfolio.
4. Short Selling: Borrowing securities to sell them with the aim of buying them back at a lower price.

While leverage can increase the profitability of winning investments, it can lead to significant losses if the market moves against the leveraged position. Risk management is all the more important when leverage is high.

For example, we go back to the industry momentum backtest and measure the leverage of the strategy.

```
[2]: m0 = (
    Backtester(MeanVariance(risk_target=1), name="baseline")
    .compute_holdings(features, target)
    .compute_pnl(ret)
)
line(
    m0.h_.abs().sum(axis=1).rename("leverage"),
    title="Industry momentum: leverage",
    legend=False,
)
```



10.2 Leverage-aware optimisation

With the same previous portfolio setup (universe is made of N assets, r is a vector of returns for each asset, α is the asset return forecast: $\alpha = E(r)$ and V is the return covariance matrix that measures that correlation between each asset: $V = \text{Var}(r)$), we add a leverage constraint:

$$U = h^T \alpha - \frac{h^T V h}{2\lambda} - \eta \sum_n |h_n|,$$

where λ is the risk-tolerance and η is a measure of the cost of leverage. The leverage metric is an L1-norm $\|h\|_1 = \sum_n |h_n|$.

```
[3]: %%writefile ./skfin/mv_estimators_with_leverage.py
import cvxpy as cp
import numpy as np
from dataclasses import dataclass
from typing import Optional
from skfin.mv_estimators import MeanVariance

def compute_holdings_with_leverage(
    pred, V, A=None, risk_target=None, leverage_target=None
):
    """
    Optimize the array h to maximize np.sum(h * pred) under specified constraints.

    Parameters:
    - pred (numpy.array): A 1D array representing the predicted returns.
    - V (numpy.array): A 2D covariance matrix of returns.
    - leverage_target (float): The desired leverage, defined as the sum of absolute values of h.

    Returns:
    - numpy.array: The optimal array h that maximizes the objective under the constraints.
    """
    n = len(pred)

    # Define the optimization variable
```

```

h = cp.Variable(n)

# Objective function: maximize the sum of element-wise multiplication of h and pred
objective = cp.Maximize(cp.sum(h @ pred))

# Constraints
constraints = []
if A is not None:
    constraints += [h.T @ A == 0]
if risk_target is not None:
    constraints += [
        cp.quad_form(h, V) <= risk_target
    ] # Ensures the quadratic form ( $h^T V h$ ) is 1
if leverage_target is not None:
    constraints += [
        cp.norm(h, 1) <= leverage_target
    ] # Ensures the sum of absolute values of h equals leverage_target

# Create and solve the problem
problem = cp.Problem(objective, constraints)
problem.solve(solver=cp.SCS, eps=1e-9, max_iters=10000, verbose=False)
problem.solve()

# Retrieve the optimal value for h
h_value = h.value

return h_value

@dataclass
class MeanVarianceWithLeverage(MeanVariance):
    """
    Mean-variance optimization estimator with leverage constraints.

    Attributes:
        leverage_target (Optional[float]): Leverage target for the portfolio.
    """

    leverage_target: Optional[float] = None

    def __post_init__(self):
        """
        Post-initialization to update holdings keyword arguments.
        """
        self.holdings_kwargs = {
            "risk_target": self.risk_target,
            "leverage_target": self.leverage_target,
        }

    @staticmethod
    def compute_batch_holdings(pred, V, A, **kwargs):
        """
        Compute portfolio holdings considering leverage.

        Parameters:
            pred (np.ndarray): Predicted returns (squeezed for computation).
            V (np.ndarray): Covariance matrix.
            A (np.ndarray): Constraint matrix.
            **kwargs: Additional keyword arguments.

        Returns:
        """

```

```

    np.ndarray: Portfolio holdings considering leverage.
"""
pred = pred.squeeze() # Squeeze prediction array for computation
h = compute_holdings_with_leverage(pred, V, A, **kwargs)
return h[:, np.newaxis].T

```

Overwriting ../skfin/mv_estimators_with_leverage.py

```
[4]: from skfin.mv_estimators_with_leverage import MeanVarianceWithLeverage

for train, test in m0.cv_.split(features):
    m = MeanVariance()
    m.fit(features.values[train], target.values[train])
    h0 = m.predict(features.values[test])

    m_ = MeanVarianceWithLeverage(leverage_target=1)
    m_.fit(features.values[train], target.values[train])
    h = m_.predict(features.values[test], leverage_target=1)
    break

df = pd.DataFrame(
    {"baseline": h0.squeeze(), "leverage=1": h.squeeze()}, index=features.columns
)

print(
    f"Correlation between unconstrained and unit-leverage holdings : {df.corr().iloc[0, 1]:.2f}"
)
```

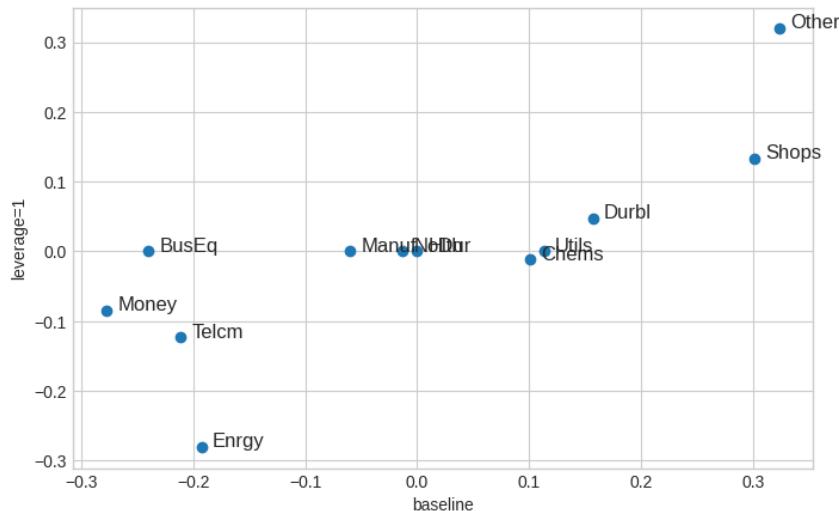
Correlation between unconstrained and unit-leverage holdings : 0.79

The following graph shows that with a leverage constraint, some positions are exactly equal to zero (while these positions would be non-zero with no leverage constraint). This is similar to zero-coefficients in a Lasso regression.

```
[5]: fig, ax = plt.subplots(1, 1, figsize=(8, 5))

cols = list(df.columns)
idx = list(df.index)

ax.scatter(df.iloc[:, 0].values, df.iloc[:, 1].values)
ax.set_xlabel(cols[0])
ax.set_ylabel(cols[1])
xshift, yshift, rotation = 0.01, 0, 0
for i, txt in enumerate(idx):
    ax.text(
        df.iloc[i, 0] + xshift,
        df.iloc[i, 1] + yshift,
        txt,
        fontsize=12,
        rotation=rotation,
    )
```



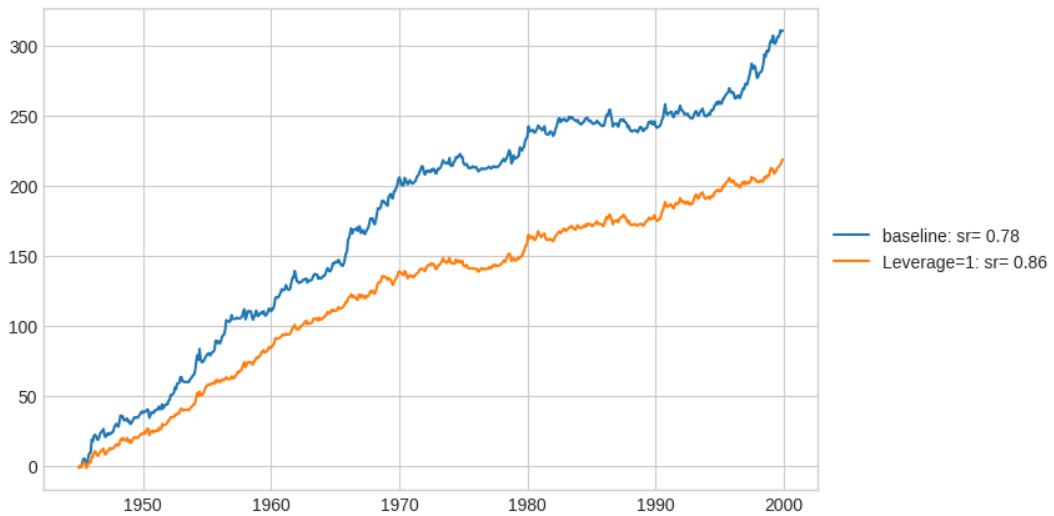
```
[6]: %%time
m = Backtester(estimator=MeanVarianceWithLeverage(leverage_target=1), name="Leverage=1")
pnl_ = m.train(features, target, ret)
```

CPU times: user 16.6 s, sys: 15.7 s, total: 32.3 s

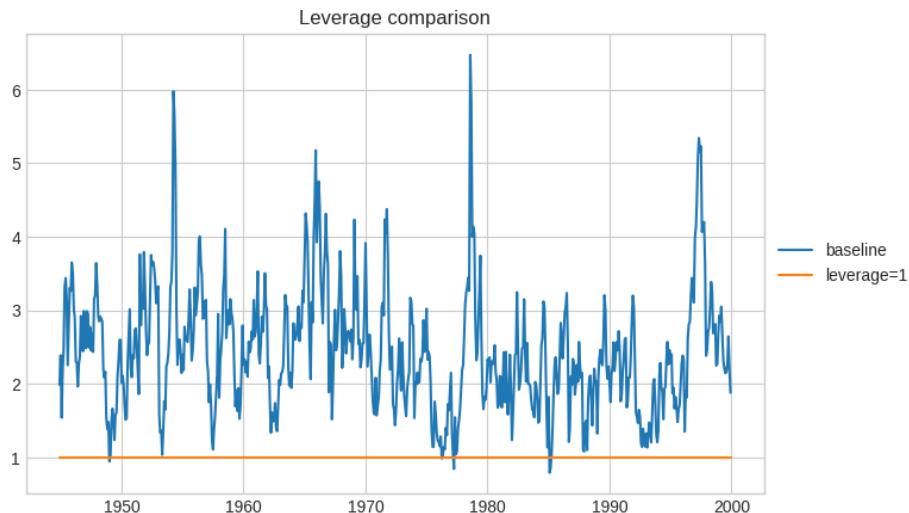
Wall time: 10.8 s

In this case, adding the leverage constraint slightly improves the Sharpe ratio (while decreasing the realized risk).

```
[7]: line(pd.concat([m0.pnl_, m.pnl_], axis=1), cumsum=True)
```

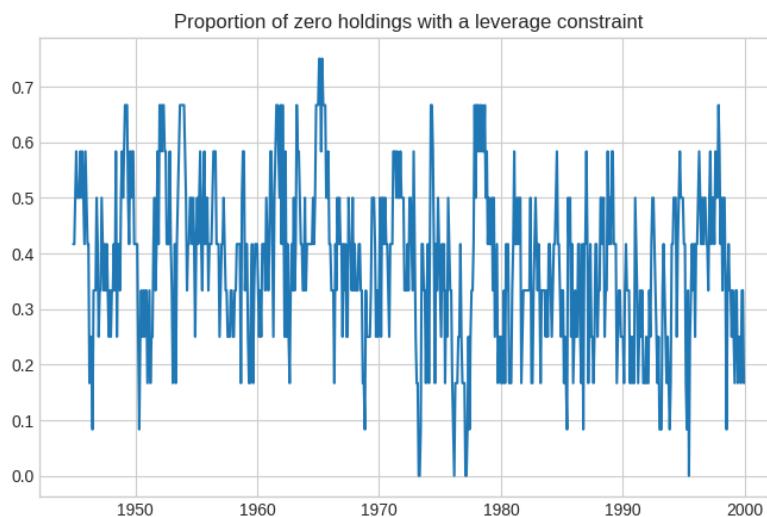


```
[8]: line(
    {"baseline": m0.h_.abs().sum(axis=1), "leverage=1": m.h_.abs().sum(axis=1)},
    title="Leverage comparison",
)
```



The graph above shows that the proportion of zero holdings with a leverage constraint is significant for all periods.

```
[9]: line(
    m.h_.pipe(lambda x: x.abs().le(1e-4)).mean(axis=1),
    legend=False,
    title="Proportion of zero holdings with a leverage constraint",
)
```



Chapter 11

Non-Normality in Portfolio Returns: Implications for Risk and Performance Evaluation

11.1 Non-normality

At the stock level, non-normality—such as skewness or kurtosis in return distributions—may not be a significant concern. This is because, within a diversified portfolio, it is possible to be short a stock with negative skewness, and some portion of stock-specific kurtosis can be diversified away. The collective effect is that idiosyncratic non-normality at the individual stock level does not typically pose a substantial problem for the overall portfolio.

However, non-normality at the factor level is much harder to diversify, particularly when dealing with strategies based on risk premia, which tend to be inherently non-normally distributed. In risk-off environments, by definition, these strategies often fail to deliver positive returns, resulting in particularly thick left tails in the return distribution and exposing the portfolio to “rare disaster” events.

In this section, we discuss how to document such non-normality by using statistical tests that focus on skewness and kurtosis, and by illustrating these methods with examples such as industry momentum strategies.

The statistics of higher moments:

- $skewness = \frac{E[(x-\bar{x})^3]}{\sigma_x^3}$
- $kurtosis = \frac{E[(x-\bar{x})^4]}{\sigma_x^4}$
- Jarque-Bera statistics $JB = \frac{T}{6} \left(skewness^2 + \frac{(kurtosis-3)^2}{4} \right)$

If the observations $\{x_1, \dots, x_T\}$ are independant and follow a Normal distribution, then

1. $\$skewness=0\$$
1. $\$kurtosis=3\$$
1. $\$JB\$$ follows of Chi-squared distribution with two degrees of freedom.

The third assertion provides a way to test whether a variable is Normally distributed.

```
[2]: from scipy.stats import jarque_bera, kurtosis, skew
```

Normally distributed random data:

```
[3]: (T, N) = ret.shape
x = np.random.normal(0, 1, T) # create random values based on a normal distribution

print(f"Excess kurtosis of normal distribution (should be 0): {kurtosis(x):.3f}")
print(f"Skewness of normal distribution (should be 0): {skew(x):.3f}")
print(f"Jarque beta of normal distribution (should be 0): {jarque_bera(x)[0]:.3f}")
```

Excess kurtosis of normal distribution (should be 0): 0.199

Skewness of normal distribution (should be 0): 0.083

Jarque beta of normal distribution (should be 0): 2.450

We can *not* reject the null hypothesis that the data follows is a Normal distribution!

With `fisher=False`, the function `kurtosis` computes the raw kurtosis (the default is `fisher=True`).

```
[4]: kurtosis(np.random.normal(0, 1, T), fisher=False)
```

```
[4]: 2.937930525039259
```

Uniformly distributed random data:

```
[5]: x = np.random.uniform(0, 1, T) # create random values based on a normal distribution

print(f"Excess kurtosis of normal distribution (should be 0): {kurtosis(x):.3f}")
print(f"Skewness of normal distribution (should be 0): {skew(x):.3f}")
print(f"Jarque beta of normal distribution (should be 0): {jarque_bera(x)[0]:.3f}")
```

Excess kurtosis of normal distribution (should be 0): -1.151

Skewness of normal distribution (should be 0): -0.079

Jarque beta of normal distribution (should be 0): 49.578

In this case, we can reject at any confidence level the null hypothesis that the data follows a Normal distribution!

Industry momentum data:

```
[6]: pnl = (
    Backtester(MeanVariance()).compute_holdings(features, target).compute_pnl(ret).pnl_
)
```

```
[7]: x = pnl
```

```
print(f"Excess kurtosis of normal distribution (should be 0): {kurtosis(x):.3f}")
print(f"Skewness of normal distribution (should be 0): {skew(x):.3f}")
print(f"Jarque beta of normal distribution (should be 0): {jarque_bera(x)[0]:.3f}")
```

Excess kurtosis of normal distribution (should be 0): 1.860

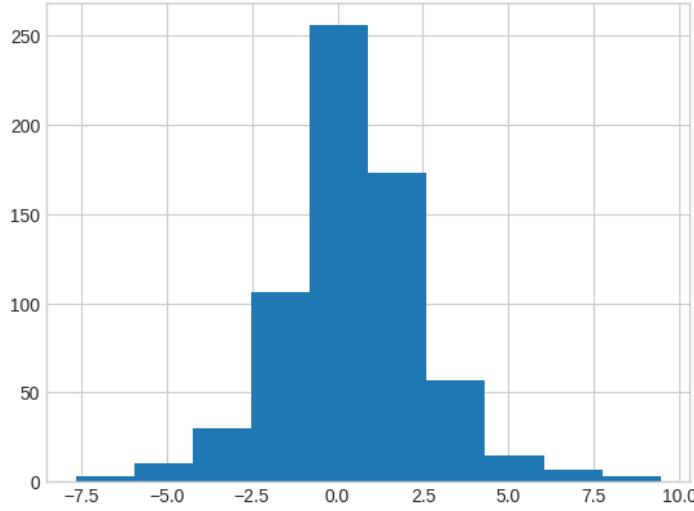
Skewness of normal distribution (should be 0): 0.049

Jarque beta of normal distribution (should be 0): 95.382

As in the uniform case, we can also reject the null hypothesis of a Normal distribution

```
[8]: print(f"Pnl std={pnl.std():.2f}")
pnl.hist();
```

Pn1 std=2.10



The test in the notebook shows that the normality assumption is easily rejected for the momentum strategy return – there are a lot of observations (T large) and significant deviations.

11.2 The statistics of rolling sharpe ratio

We now focus on 3-year rolling sharpe ratio:

- A 3-year horizon is the natural horizon to evaluate the performance of investment strategies.
- Significant underperformance over 3 years is almost always a show-stopper!
- In particular, we assess the impact of skewness and kurtosis on the rolling sharpe ratio.

Lemma [Lo, 1996]. When the underlying data is iid Normally distributed, the limiting distribution of the estimated monthly sharpe ratio \hat{S} (relative to the true unobserved S) is

$$\sqrt{T}(\hat{S} - S) \rightarrow N\left(0, 1 + \frac{S^2}{2}\right)$$

Proof. For independently Normally distributed returns, we have

$$\sqrt{T} \begin{bmatrix} \hat{\mu} - \mu \\ \hat{\sigma}^2 - \sigma^2 \end{bmatrix} \rightarrow N\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}; \begin{bmatrix} \sigma^2 & 0 \\ 0 & 2\sigma^4 \end{bmatrix}\right)$$

For $S = g(\mu, \sigma^2) = \frac{\mu}{\sqrt{\sigma^2}}$, the asymptotic variance of \hat{S} is

$$\begin{bmatrix} \frac{\partial g}{\partial \mu} \\ \frac{\partial g}{\partial \sigma^2} \end{bmatrix}^T \begin{bmatrix} \sigma^2 & 0 \\ 0 & 2\sigma^4 \end{bmatrix} \begin{bmatrix} \frac{\partial g}{\partial \mu} \\ \frac{\partial g}{\partial \sigma^2} \end{bmatrix} = \frac{1}{\sigma^2} \sigma^2 + \left(-\frac{\mu}{2\sigma^3}\right)^2 (2\sigma^4)$$

Corollary. The minimum monthly S_0 for a monthly strategy where the Sharpe ratio is always statistically different from 0 at the 5%-confidence level over a window of T months is given by

$$S_0 - 1.96 \sqrt{\frac{1 + \frac{S_0^2}{2}}{T}} = 0 \Leftrightarrow S_0 = \sqrt{\frac{1}{\frac{T}{1.96^2} - 1/2}}$$

- Rule of thumb: for a 36-month horizon, then the monthly $S_0 \approx 0.35$ or after annualization

$$S_0^{\text{annualized}} \approx .35 \times \sqrt{12} = 1.16$$

When the returns are not Normally distributed ($\text{skewness} \neq 0$ and (excess) $\text{kurtosis} \neq 0$), the Central Limit theorem still ensures the asymptotic normality. (In what follows, kurtosis refers to the “excess kurtosis.”)

Lemma [Mertens, 2002] The asymptotic distribution of \hat{S} is

$$\sqrt{T}(\hat{S} - S) \rightarrow N(0, V_\infty)$$

where $V_\infty = 1 + \frac{S^2}{2} - \text{skewness} \times S + \frac{\text{kurtosis} \times S^2}{4}$.

Proof. The asymptotic variance of $\begin{bmatrix} \hat{\mu} - \mu \\ \hat{\sigma}^2 - \sigma^2 \end{bmatrix}$ is:

$$E \begin{bmatrix} (r_t - \mu)^2 & (r_t - \mu)[(r_t - \mu)^2 - \sigma^2] \\ (r_t - \mu)[(r_t - \mu)^2 - \sigma^2] & [(r_t - \mu)^2 - \sigma^2]^2 \end{bmatrix} = \begin{bmatrix} \sigma^2 & E[(r_t - \mu)^3] \\ E[(r_t - \mu)^3] & 2E[(r_t - \mu)^4] - \sigma^4 \end{bmatrix}$$

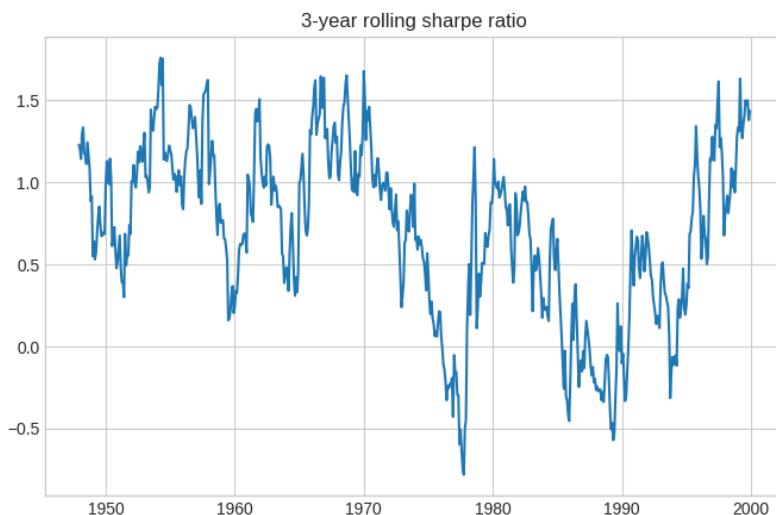
Corollary. The minimum $S_0(\text{skewness}, \text{kurtosis})$ for a monthly strategy where the information is always statistically different from 0 at the 5%-confidence level over a window of T months is increasing in kurtosis and decreasing in skewness .

Proof. The function $f(S, \text{skewness}, \text{kurtosis})$

$$= S - 1.96 \sqrt{\frac{1 + \frac{S^2}{2} - \text{skewness} \times S + \frac{\text{kurtosis} \times S^2}{4}}{T}}$$

is increasing in S and skewness and decreasing in kurtosis , so that S_0 defined by $f(S_0, \text{skewness}, \text{kurtosis}) = 0$ is decreasing in skewness and increasing in kurtosis .

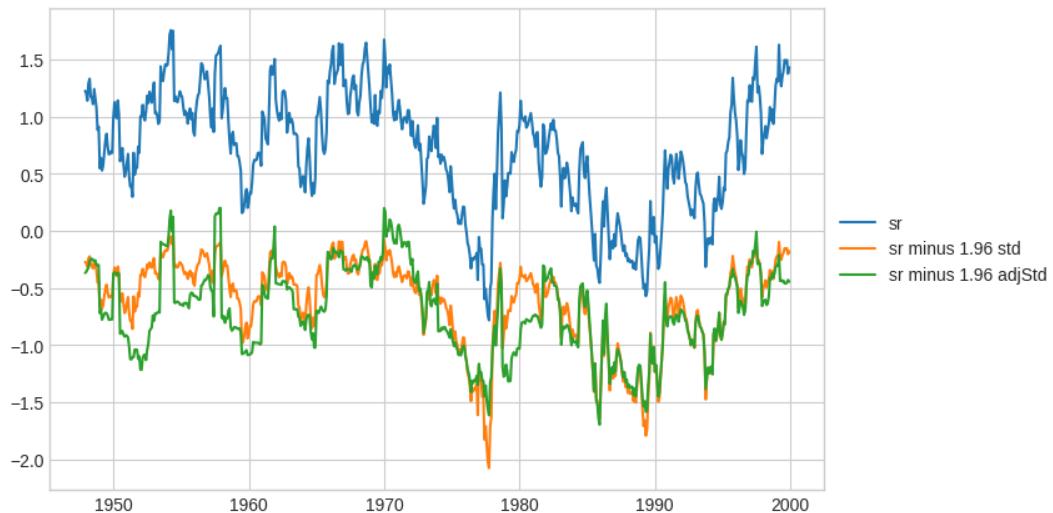
```
[9]: sr3y = pnl.rolling(36).apply(sharpe_ratio)
line(sr3y, legend=False, title="3-year rolling sharpe ratio", start_date="1945")
```



Below we compute the two lower bounds for the sharpe ratio.

```
[10]: sr3y_lb = sr3y - 1.96 * np.sqrt((1 + 0.5 * sr3y**2) / 36) * np.sqrt(12)
skew3y = pnl.rolling(36).apply(skew)
kurtosis3y = pnl.rolling(36).apply(kurtosis)
sr3y_lb_bis = sr3y - 1.96 * np.sqrt(
    (1 + 0.5 * sr3y**2 - skew3y * sr3y + 0.25 * kurtosis3y * sr3y**2) / 36
) * np.sqrt(12)
```

```
[11]: line(
    {"sr": sr3y, "sr minus 1.96 std": sr3y_lb, "sr minus 1.96 adjStd": sr3y_lb_bis},
    start_date="1945",
)
```



Even generic momentum strategies (including industry momentum) have a fairly strong Sharpe ratio...

- ... but the performance includes kurtosis/skewness risk premia, with occasional deep underperformance over a 3-year horizon.

Chapter 12

Style Analysis and Factor Decomposition: Evaluating Industry Momentum and Berkshire Hathaway's Alpha

In this section, we compare the performance of the Industry momentum derived in previous sections with other factors (e.g. Stock Momentum, Value, Size, etc.). In particular, we discuss the Style regression of Sharpe (1992).

12.1 Style analysis

As introduced by Sharpe (1992), Style Analysis is the process of determining what type of investment behaviour an investor or a money manager employs when making investment decisions.

A regression is used to determine the factor exposures $\langle \beta_1, \dots, \beta_K \rangle$ where:

$$r = \alpha + \beta_1 r_1^\Phi + \dots + \beta_K r_K^\Phi + \epsilon$$

Additional constraints might be added to “regularize” the regression such as non-negative exposures : $\beta_k \geq 0$ and/or sum equals 1: $\sum_{k=1}^K \beta_k = 1$.

Frazzini, Kabiller and Pedersen (2013) state that:

Berkshire Hathaway has realized a Sharpe ratio of 0.76, higher than any other stock or mutual fund with a history of more than 30 years, and Berkshire has a significant alpha to traditional risk factors.”

How did Warren Buffet do it? We use a “style analysis” approach applied to equity factors to address this question.

The main regression is:

$$r_t = \alpha + \beta_1 MKT_t + \beta_2 SMB_t + \beta_3 HML_t + \beta_4 UMD_t + \beta_5 BAB_t + \beta_6 QMJ_t + \epsilon_t$$

where the factors are

- r_t : excess return of the Berkshire Hathaway stock
- MKT_t (market): excess market return
- SMB_t (size): small minus big
- HML_t (value): high book-to-market minus low book-to-market
- UMD_t (momentum): up minus down
- BAB_t (betting-against-beta): safe (low beta) minus risky (high beta)
- QMJ_y (quality): quality minus junk

Can we replicate this finding? Fortunately Steve Lihn (on GitHub) already did it.

Data: github.com/slihn/buffetts_alpha_R/archive/master.zip

Table 4

Buffett's Exposures: What Kind of Companies does Berkshire Own?

<u>Berkshire stock 1976 - 2011</u>			
Alpha	12.1% (3.19)	9.2% (2.42)	6.3% (1.58)
MKT	0.84 (11.65)	0.83 (11.70)	0.95 (10.98)
SMB	-0.32 -3.05	-0.32 -3.13	-0.15 -1.15
HML	0.63 (5.35)	0.38 (2.79)	0.46 (3.28)
UMD	0.06 (0.90)	-0.03 -(0.40)	-0.05 -(0.71)
BAB		0.37 (3.61)	0.29 (2.67)
QMJ			0.43 (2.34)
R2 bar	0.25	0.27	0.28

- The characteristics of the investment of Warren Buffet: high loadings on replicable factors such as beta, size, value and quality – and a negative loading on momentum.
- At least in this replication of the paper's results (with slightly different data), the intercept is no longer statistically significant – it might still be economically significant!

```
[3]: from skfin.datasets_ import load_buffets_data

data = load_buffets_data(cache_dir="data").assign(
    excess_return=lambda x: x["BRK.A"] - x["RF"]
)
```

```
[4]: from statsmodels import api
```

```
m1 = api.OLS(data["excess_return"], api.add_constant(data["MKT"])).fit()
m1.summary()
```

```
[4]: <class 'statsmodels.iolib.summary.Summary'>
"""
                OLS Regression Results
=====
Dep. Variable:      excess_return    R-squared:           0.443
Model:                 OLS    Adj. R-squared:        0.440
Method:              Least Squares    F-statistic:         141.7
Date:      Mon, 29 Sep 2025    Prob (F-statistic):   2.08e-24
Time:          17:48:20    Log-Likelihood:     -471.71
No. Observations:      180    AIC:                  947.4
Df Residuals:          178    BIC:                  953.8
Df Model:                   1
Covariance Type:    nonrobust
=====
            coef      std err          t      P>|t|      [0.025      0.975]
-----
const      0.2431      0.250      0.972      0.332     -0.251      0.737
MKT       0.6242      0.052     11.903      0.000      0.521      0.728
=====
Omnibus:             55.501    Durbin-Watson:       2.104
Prob(Omnibus):        0.000    Jarque-Bera (JB):  251.454
Skew:             -1.075    Prob(JB):        2.50e-55
Kurtosis:            8.377    Cond. No.          4.79
=====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
"""
```

```
[5]: summaries = []
for cols in [
    ["MKT", "SMB", "HML", "UMD"],
    ["MKT", "SMB", "HML", "UMD", "BAB"],
    ["MKT", "SMB", "HML", "UMD", "BAB", "QMJ"],
]:
    m_ = api.OLS(data["excess_return"], api.add_constant(data[cols])).fit()
    summaries += [m_.summary()]
```

```
[6]: def prettify_table(tbl):
    df = pd.DataFrame(tbl.tables[1].data)
    idx = df.iloc[1:, 0]
    return pd.DataFrame(
        df.iloc[1:, [1, 3]].astype(float).values,
        index=idx.rename(None),
        columns=["coef", "tstat"],
    ).stack()
```

```
[7]: pd.concat([prettify_table(v) for v in summaries], axis=1).fillna(0).round(2)
```

	0	1	2	
const	0.28	0.25	0.07	
	tstat	1.32	1.21	0.33
MKT	coef	0.69	0.70	0.82
	tstat	14.41	14.31	12.31
SMB	coef	-0.32	-0.31	-0.20
	tstat	-4.28	-4.11	-2.32
HML	coef	0.49	0.45	0.46

```

      tstat    7.03   5.53   5.83
UMD    coef   -0.12  -0.13  -0.14
       tstat  -3.27  -3.40  -3.62
BAB    coef    0.00   0.06   0.02
       tstat  0.00   0.94   0.36
QMJ    coef    0.00   0.00   0.29
       tstat  0.00   0.00   2.61

```

The coefficients are qualitatively close to the results in the paper – with the except of the BAB coefficients not being statistically significant.

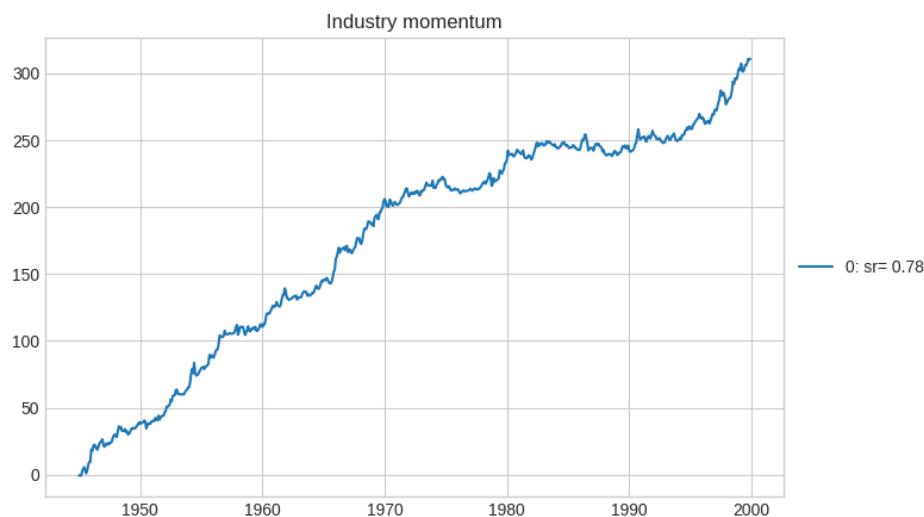
12.2 Industry momentum factor exposure

In this section, we go back to the Industry momentum backtest and decompose it on the factors as computed by Ken French.

```
[8]: returns_data = load_kf_returns(cache_dir="data")
ret = returns_data["Monthly"]["Average_Value_Weighted_Returns"][:"1999"]
```

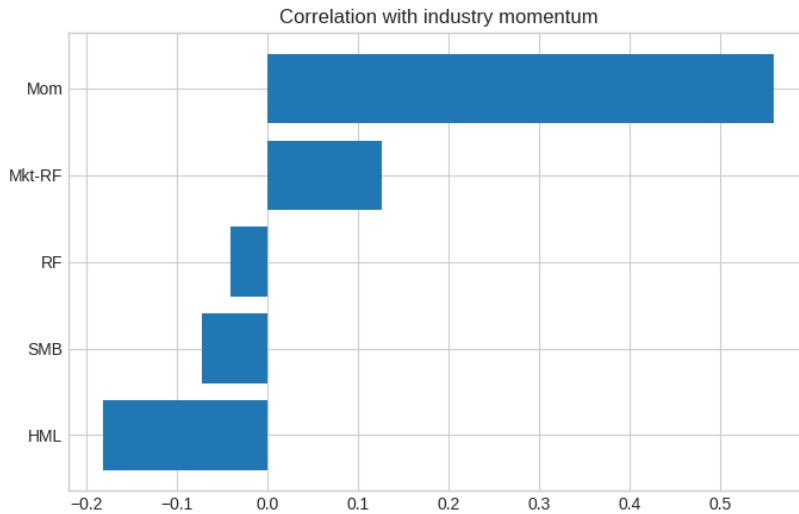
```
[9]: transform_X = lambda x: x.rolling(12).mean().fillna(0)
transform_y = lambda x: x.shift(-1)
features = transform_X(ret)
target = transform_y(ret)
```

```
[10]: pn10 = Backtester(MeanVariance()).train(features, target, ret)
line(pn10, cumsum=True, title="Industry momentum")
```



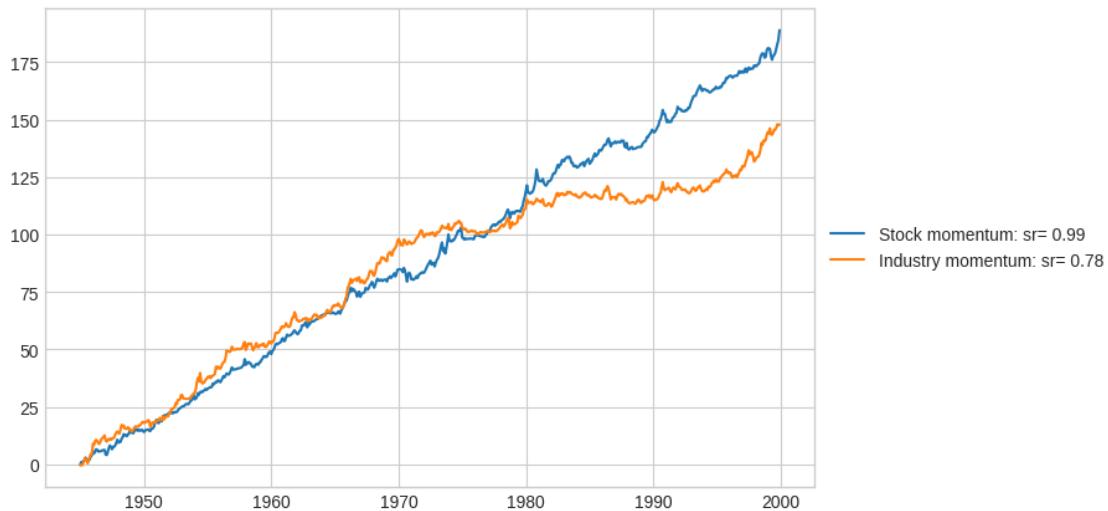
```
[11]: files = ["F-F_Research_Data_Factors", "F-F_Momentum_Factor"]
df = pd.concat([load_kf_returns(c)["Monthly"] for c in files], axis=1)[:"1945":"1999"]
```

```
[12]: bar(df.corrwith(pn10), horizontal=True, title="Correlation with industry momentum")
```



```
[13]: data_ = df.join(pnl0.rename("IndustryMom"))
```

```
[14]: line(
    pd.concat({"Stock momentum": df["Mom"], "Industry momentum": pnl0}, axis=1).pipe(
        lambda x: x.div(x.std())
    ),
    cumsum=True,
)
```



The main issue with this statistical decomposition is that the estimation is done “full sample”. In the next section, we estimate the Momentum loading on rolling windows.

12.3 Residual pnl

The residual pnl is defined as the difference between the pnl of a strategy and the pnl that can be attributed to the strategy’s exposure to a set of benchmarks. In other words, it is the portion

of the strategy's pnl that remains after removing the component explained by its projection onto the chosen benchmarks, reflecting the strategy's performance that is independent of those benchmarks.

To run the rolling estimation decomposition, we use the function `fit_predict` used in previous sections.

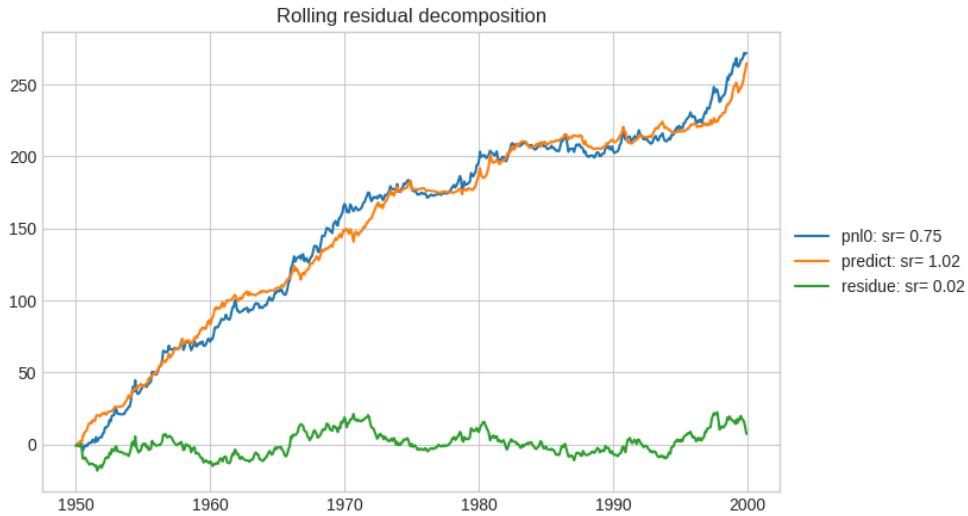
```
[15]: from skfin.backtesting import fit_predict
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import TimeSeriesSplit
```

```
[16]: start_date = "1945-01-01"
max_train_size = 60
test_size = 1
params = dict(max_train_size=max_train_size, test_size=test_size, gap=0)
params["n_splits"] = (len(data_) - max_train_size) // test_size
cv_ = TimeSeriesSplit(**params)
```

```
[19]: pnl_hat, estimator_ = zip(
    *[[
        fit_predict(
            estimator=LinearRegression(),
            X=data_.drop(["IndustryMom", "RF"], axis=1),
            y=data_["IndustryMom"],
            train=train,
            test=test,
            return_estimator=True,
        )
        for train, test in cv_.split(data_["IndustryMom"])
    ]
])
```

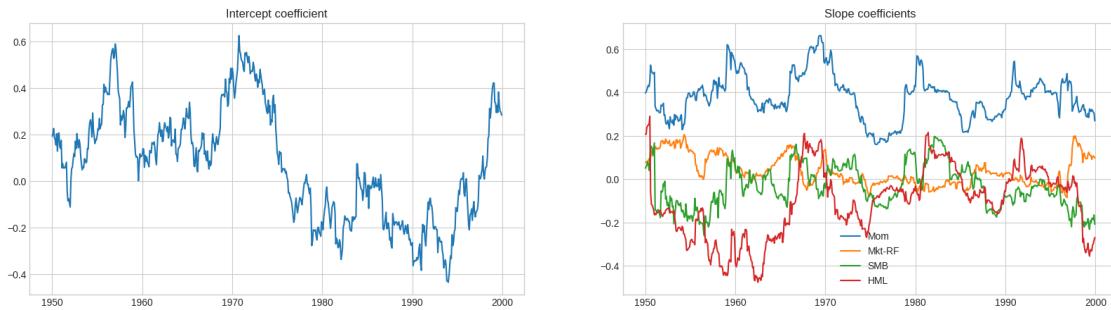
```
[20]: pnl_hat = pd.Series(
    np.concatenate(pnl_hat), index=data_["IndustryMom"].index[max_train_size:]
)
```

```
[21]: line(
    {
        "pnl0": pnl0[max_train_size:],
        "predict": pnl_hat,
        "residue": pnl0[max_train_size:] - pnl_hat,
    },
    cumsum=True,
    title="Rolling residual decomposition",
)
```



```
[22]: fig, ax = plt.subplots(1, 2, figsize=(20, 5))
line(
    pd.DataFrame([m.intercept_ for m in estimator_], index=pnl_hat.index),
    title="Intercept coefficient",
    ax=ax[0],
    legend=False,
)

line(
    pd.DataFrame(
        [m.coef_ for m in estimator_],
        columns=data_.drop(["IndustryMom", "RF"], axis=1).columns,
        index=pnl_hat.index,
    ),
    title="Slope coefficients",
    loc="best",
    ax=ax[1],
)
```



Over this period, the simple Industry momentum strategy seems to have zero residual relative to other factors.

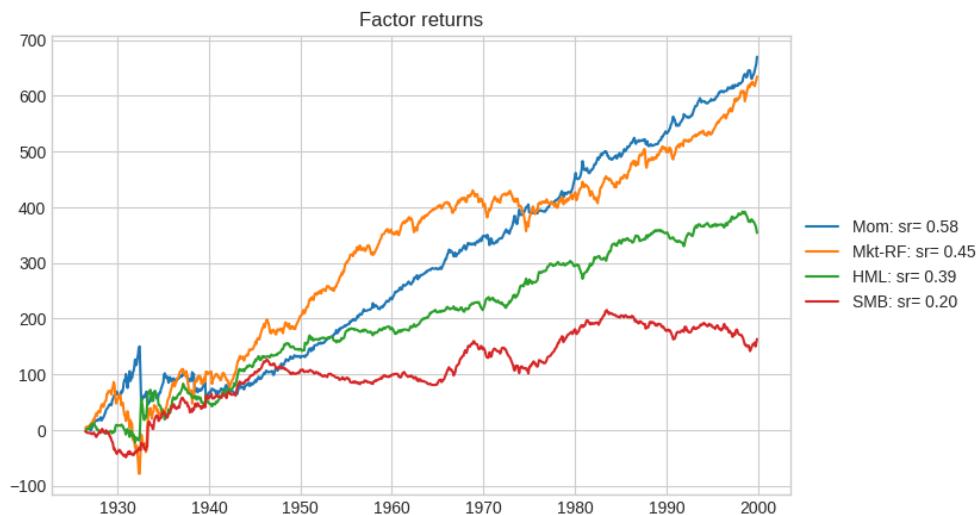
Chapter 13

Momentum-Based Factor Timing: Decomposition and Performance Analysis

In this section, we discuss a simple Momentum strategy to time the quantitative factors previously considered (Stock Momentum, Value, Size and the Market).

```
[2]: files = ["F-F_Research_Data_Factors", "F-F_Momentum_Factor"]
ret = pd.concat([load_kf_returns(c)[("Monthly") for c in files], axis=1].drop(
    "RF", axis=1
)[:, "1999"]]

[3]: line(ret, cumsum=True, title="Factor returns")
```



As in the industry Momentum case, we use the previous 12-month returns as the prediction of returns.

```
[4]: def transform_X(df, window=12):
    return df.rolling(window=window).mean()
```

```

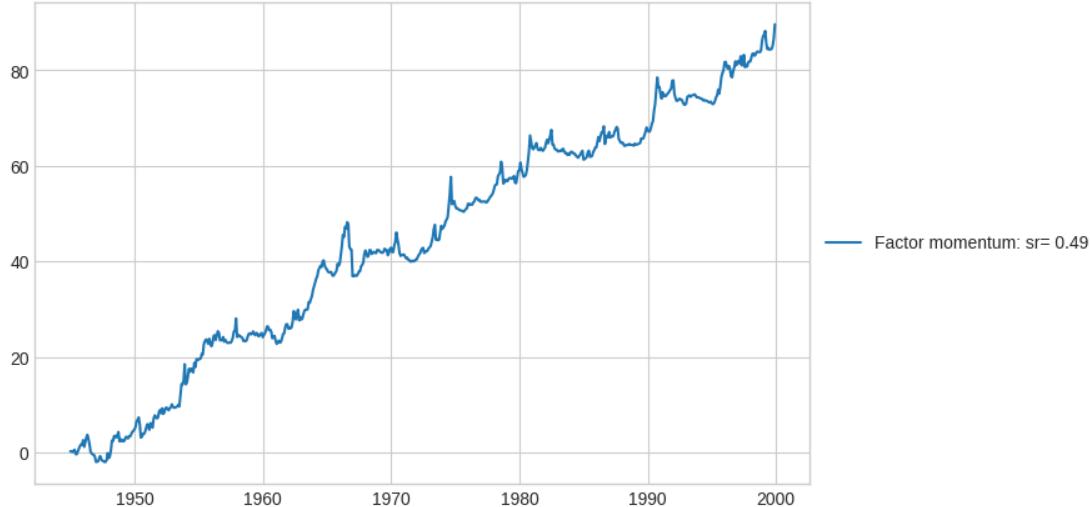
def transform_y(df):
    return df.shift(-1)

X = transform_X(ret)
y = transform_y(ret)

```

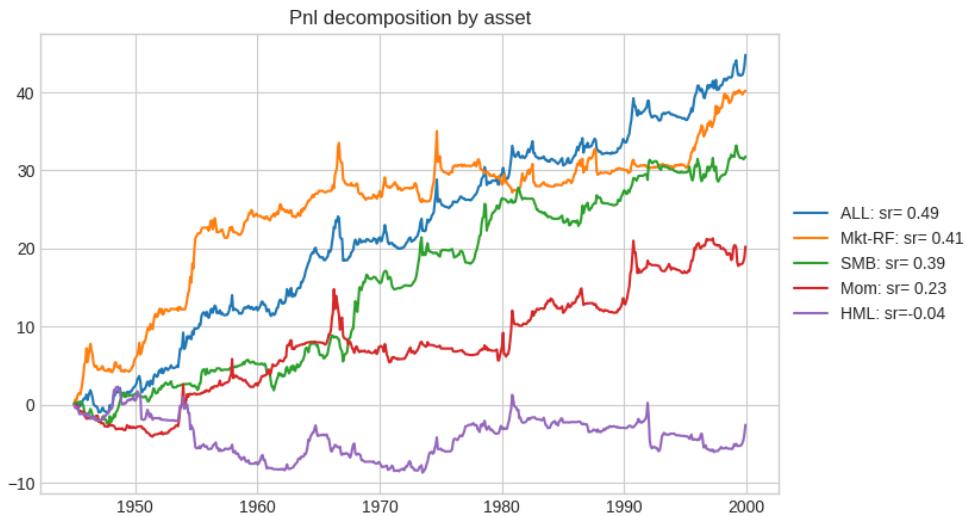
While in the sector rotation case, we imposed a “cash-neutrality constraint” (so that the sum of the positions on a given is zero and the strategy is long-short), this constraint is not necessary in the factor timing case. Below, we set the constraint vector A to `None` in the `MeanVariance` estimator.

```
[5]: m = Backtester(estimator=MeanVariance(A=None), name="Factor momentum")
pnl_ = m.train(X, y, ret)
line(pnl_, cumsum=True)
```



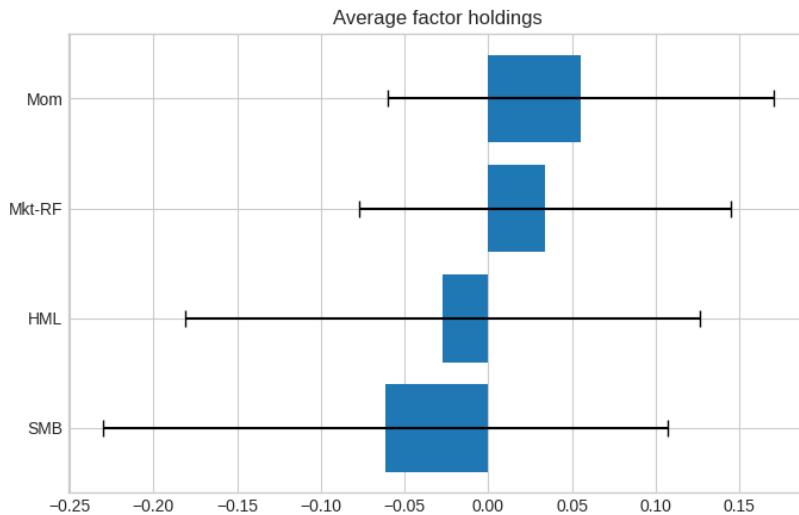
Below we decompose the strategy `pnl` by the contribution of each asset. (We also divide the total `pnl` by the square-root of the number of assets so that all the `pnl` shown below have approximately the same standard deviation.)

```
[6]: line(
    m.h_.shift(1)
    .mul(ret)
    .dropna(how="all", axis=0)
    .assign(ALL=lambda x: x.sum(axis=1).div(np.sqrt(ret.shape[1]))),
    cumsum=True,
    title="Pnl decomposition by asset",
)
```



The chart below shows the average positions on the factors is positive, but there is a significant standard deviation on the position.

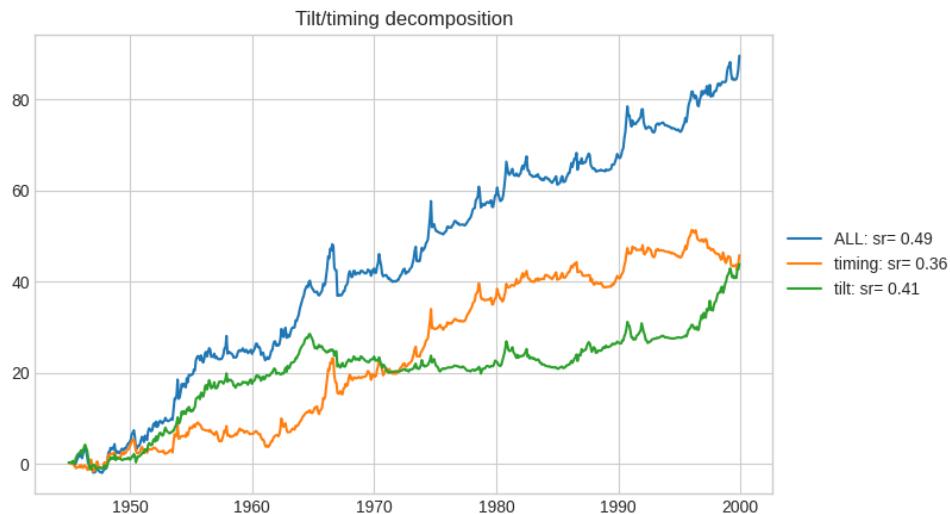
```
[7]: bar(m.h_.mean(), m.h_.std(), horizontal=True, title="Average factor holdings")
```



To see whether the performance is coming from factor “tilts”, we decompose the pnls into the slow positions (over a one-year halflife) and the faster residual positions (defined as the difference between the positions and slow positions). This tilt/timing decomposition below shows some contribution from timing.

```
[8]: line(
    {
        "ALL": pnl_,
        "tilt": m.h_.ewm(halflife=12)
            .mean()
            .shift(1)
            .mul(ret)
```

```
.sum(axis=1)
.replace(0, np.nan)
.dropna(),
"timing": m.h_.sub(m.h_.ewm(halflife=12).mean())
.shift(1)
.mul(ret)
.sum(axis=1)
.replace(0, np.nan)
.dropna(),
},
cumsum=True,
title="Tilt/timing decomposition",
)
```



Chapter 14

Combining Multiple Strategies: Ensemble Methods in Portfolio Construction

In previous sections (e.g. on hyperparameter tuning), we have described strategies to identify the best parameters for a given objective. In this section, rather than choosing a single estimator (or set of parameters) among many, we describe how to combine all multiple strategies. More precisely, the differences across strategies might be due to:

- different hyperparameters;
- different estimators;
- different data inputs.

Combination of strategies that are not perfectly correlated is actually powerful because it exploits gains from diversification. To combine weights, there are two setups:

1. either the weights are known
2. or they need to be learned.

In Finance (and more generally in Data Science), the “known weights” case is generally very strong, often as the equal-weight “1-over-N” baseline. This is generally due to the fact that in setups with a lot of noise, learning the weights is hard. In a latter section below, we introduce a setup with “learned weights”: the main challenge is that the weights have to be learned over rolling windows (jointly with the derivation of the individual strategy positions).

14.1 Ensemble with scikit-learn

In the spirit of this course, we first use `scikit-learn` to combine strategy with known weights, in particular, the `VotingRegressor` class.

Disclaimer: I have found that using these `Ensemble` classes from `scikit-learn` is quite cumbersome (see also the `StackingRegressor`), in particular because these classes do not interact well with `scikit-learn` pipelines.

```
[2]: from skfin.estimators import Ridge
      from sklearn.ensemble import VotingRegressor
```

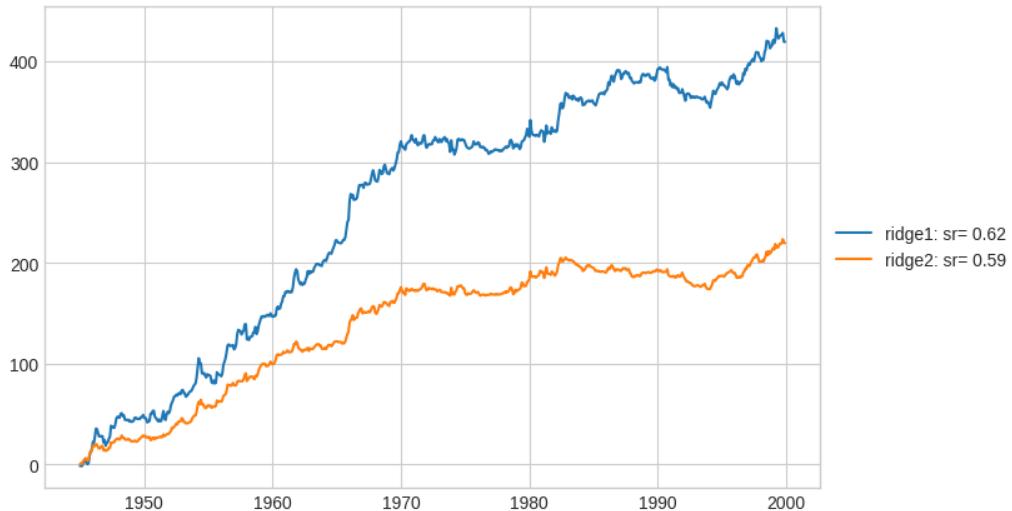
```
class VotingRegressor(VotingRegressor):
    def transform(self, X):
        return self.predict(X)
```

```
[3]: estimators_ = [
    ("ridge1", Ridge(alpha=1)),
    ("ridge2", Ridge(alpha=100)),
]
```

By default, `VotingRegressor` applies equal weights across regressors.

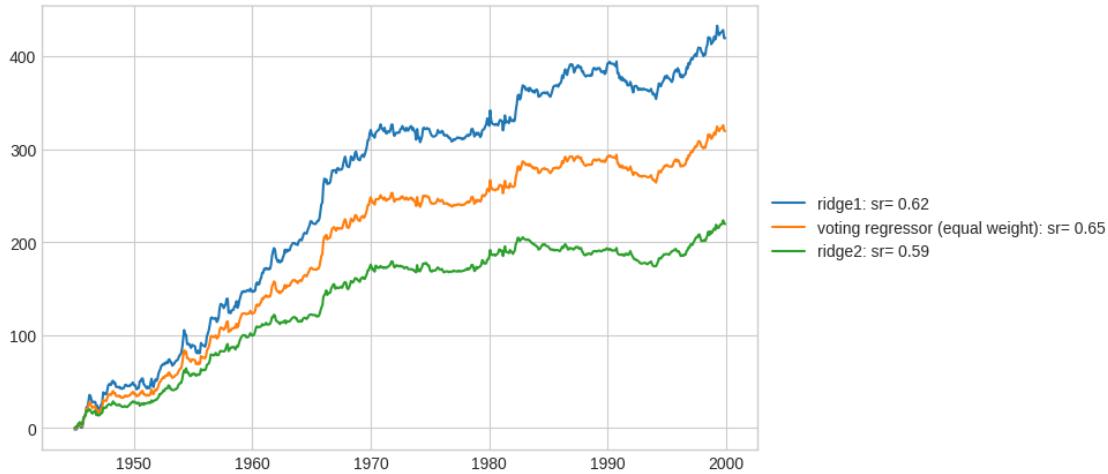
```
[4]: pnls0 = pd.concat(
{
    k: Backtester(
        make_pipeline(StandardScaler(with_mean=False), v, MeanVariance())
    ).train(X, y, ret)
    for k, v in estimators_
},
axis=1,
)
```

```
[5]: line(pnls0, cumsum=True)
```



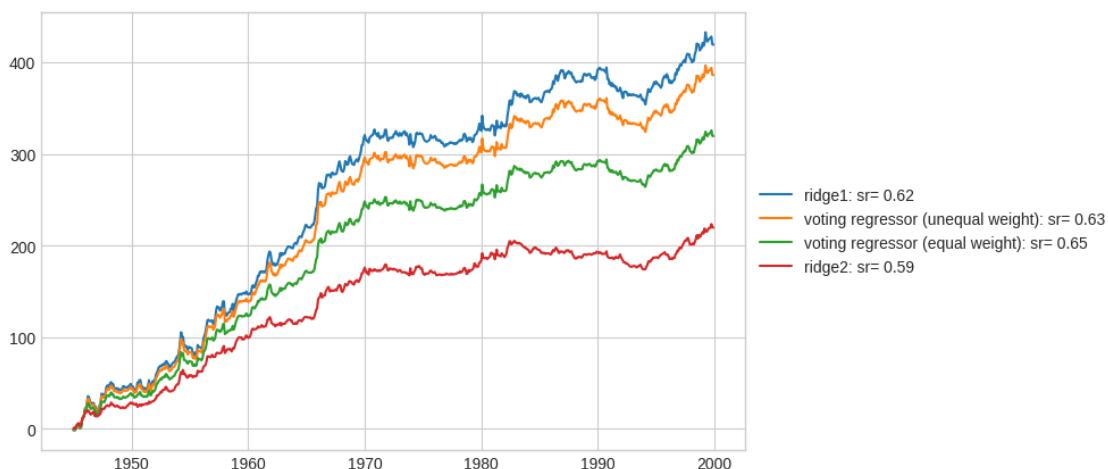
```
[6]: estimator = make_pipeline(
    StandardScaler(with_mean=False),
    MultiOutputRegressor(VotingRegressor(estimators=estimators_)),
    MeanVariance(),
)
```

```
[7]: pnls0["voting regressor (equal weight)"] = Backtester(estimator).train(X, y, ret)
line(pnls0, cumsum=True)
```



```
[8]: estimator2 = make_pipeline(
    StandardScaler(with_mean=False),
    MultiOutputRegressor(VotingRegressor(estimators_=estimators_, weights=[5, 1])),
    MeanVariance(),
)
```

```
[9]: pnls0["voting regressor (unequal weight)"] = Backtester(estimator2).train(X, y, ret)
line(pnls0, cumsum=True)
```

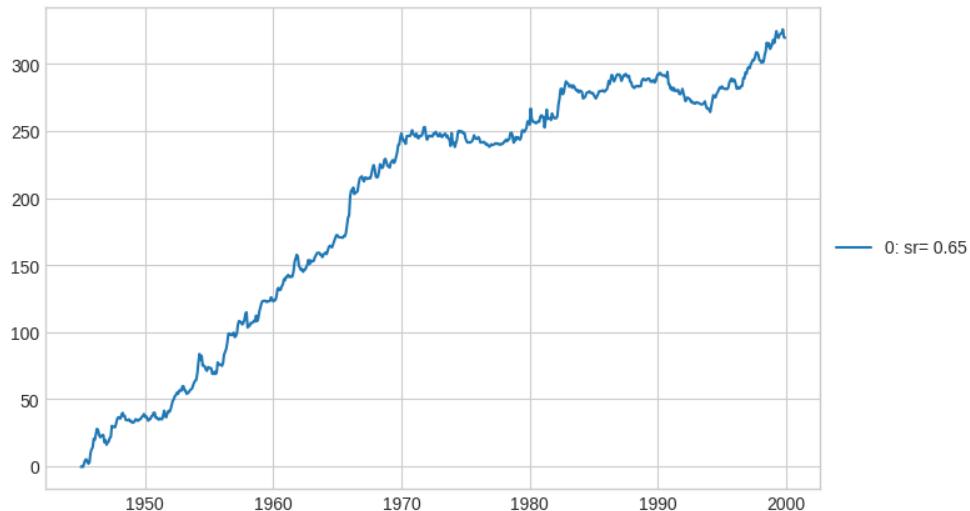


```
[10]: estimators_ = [
    ("ridge1", make_pipeline(StandardScaler(with_mean=False), Ridge(alpha=1))),
    ("ridge2", make_pipeline(StandardScaler(with_mean=False), Ridge(alpha=100))),
]

estimator = make_pipeline(
    MultiOutputRegressor(VotingRegressor(estimators=estimators_)),
    MeanVariance(),
)

pnl_ = Backtester(estimator).train(X, y, ret)
```

```
line(pnl_, cumsum=True)
```



14.2 Rolling ensemble backtest

In this section, we build a custom ensemble method to learn weights on different estimators from pnls.

14.2.1 StackingBacktester

We consider three estimators:

- the simple Industry momentum.
- a strategy that learns cross-industry effect with Ridge.
- a strategy that learns cross-industry effect with Lightgbm.

```
[11]: estimators = {
    "momentum": MeanVariance(),
    "ridge": make_pipeline(StandardScaler(with_mean=False), Ridge(), MeanVariance()),
    "lightgbm": make_pipeline(
        MultiOutputRegressor(
            LGBMRegressor(min_child_samples=5, n_estimators=25, n_jobs=1)
        ),
        MeanVariance()
    ),
}
```

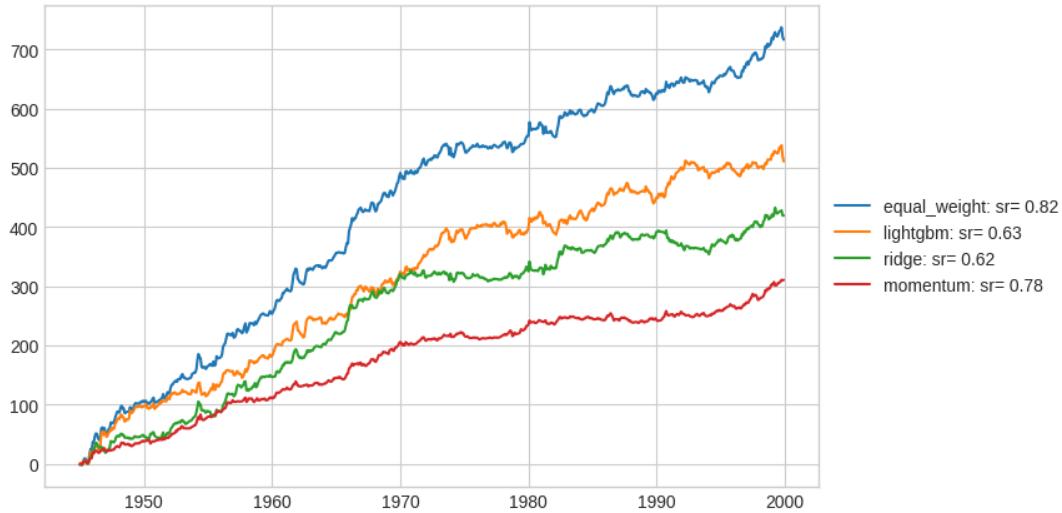
```
[12]: pnls = pd.concat(
    {k: Backtester(v).train(X, y, ret) for k, v in estimators.items()},
    axis=1,
)
```

To construct an equal-weight strategy, we scale the sum of the pnl by the square-root of the number of strategy:

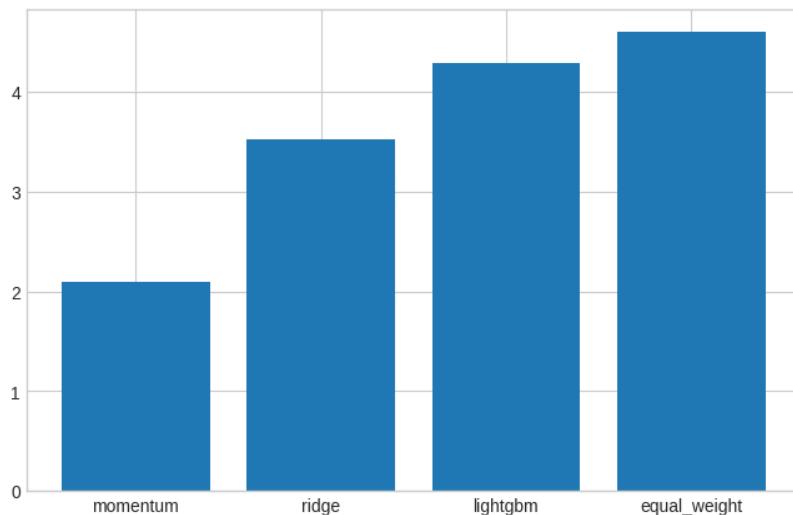
$$pnl_{\text{equal weight}} = \frac{\sum_{n=1}^{N=pnl} pnl_n}{\sqrt{N}}$$

In this case, if the individual pnls pnl_n are identically and independently distributed (with the same standard deviation), then $pnl_{\text{equal weight}}$ has the same ex-ante standard deviation.

```
[13]: pnls_ = pnls.assign(equal_weight=lambda x: x.sum(axis=1).div(np.sqrt(x.shape[1])))
line(pnls_, cumsum=True)
```



```
[14]: bar(pnls_.std())
```



The average correlation is not particularly high, which explains why some simple ensemble seems to help.

```
[15]: print(
    f"The average pnl correlation between estimators is {pnls.corr().stack().loc[lambda x: x != -1].mean():.2f}"
)
```

The average pnl correlation between estimators is 0.45

We introduce a StackingBacktester with the sklearn API.

```
[16]: %%writefile ../skfin/ensemble.py
from dataclasses import dataclass

import numpy as np
import pandas as pd
from skfin.mv_estimators import Mbj
from sklearn.base import BaseEstimator
from sklearn.model_selection import TimeSeriesSplit

@dataclass
class StackingBacktester:
    estimators: dict
    max_train_size: int = 36
    test_size: int = 1
    start_date: str = "1945-01-01"
    end_date: str = None
    window: int = 60
    min_periods: int = 60
    final_estimator: BaseEstimator = Mbj()

    def train(self, X, y, ret):
        cv = TimeSeriesSplit(
            max_train_size=self.max_train_size,
            test_size=self.test_size,
            n_splits=1 + len(X.loc[self.start_date : self.end_date]) // self.test_size,
        )
        N_estimators = len(self.estimators)
        cols = X.columns
        idx = X.index[np.concatenate([test for _, test in cv.split(X)])]

        _h = {k: [] for k in list(self.estimators.keys()) + ["ensemble"]}
        _next_pnls = {k: [] for k in self.estimators.keys()}
        _coef = []
        for i, (train, test) in enumerate(cv.split(X)):
            h_ = {}
            # each strategy position and next-period pnls
            for k, m in self.estimators.items():
                m.fit(X.iloc[train], y.iloc[train])
                h_[k] = m.predict(X.iloc[test])
                _h[k] += [h_[k]]
            if i + 1 < len(idx):
                _next_pnls[k] += [ret.loc[idx[i + 1]].dot(np.squeeze(h_[k]))]
            # compute coef from strategy pnls
            if i <= self.min_periods:
                _coef += [np.zeros(N_estimators)]
            else:
                pnl_window = np.stack(
                    [np.array(v[-self.window-1 :-1]) for k, v in _next_pnls.items()], axis=1
                )
                coef_ = self.final_estimator.fit(pnl_window).coef_
                _coef += [coef_]
        # ensemble
```

```

        if i <= self.min_periods:
            h_ensemble = np.zeros([len(cols), 1])
        else:
            h_ensemble = (
                np.stack([np.squeeze(v) for v in h_.values()], axis=1)
                .dot(coef_)
                .reshape(-1, 1)
            )
            V_ = m.named_steps["meanvariance"].V_
            h_ensemble = h_ensemble / np.sqrt(
                np.diag(h_ensemble.T.dot(V_.dot(h_ensemble))))
        )
    _h[ensemble] += [h_ensemble.T]

self.h_ = {
    k: pd.DataFrame(np.concatenate(_h[k]), index=idx, columns=cols)
    for k in _h.keys()
}
self.pnls_ = pd.concat(
{
    k: v.shift(1).mul(ret).sum(axis=1)[self.start_date :]
    for k, v in self.h_.items()
},
axis=1,
)
self.coef_ = pd.DataFrame(
    np.stack(_coef), index=idx, columns=self.estimators.keys()
)
self.cv = cv
return self

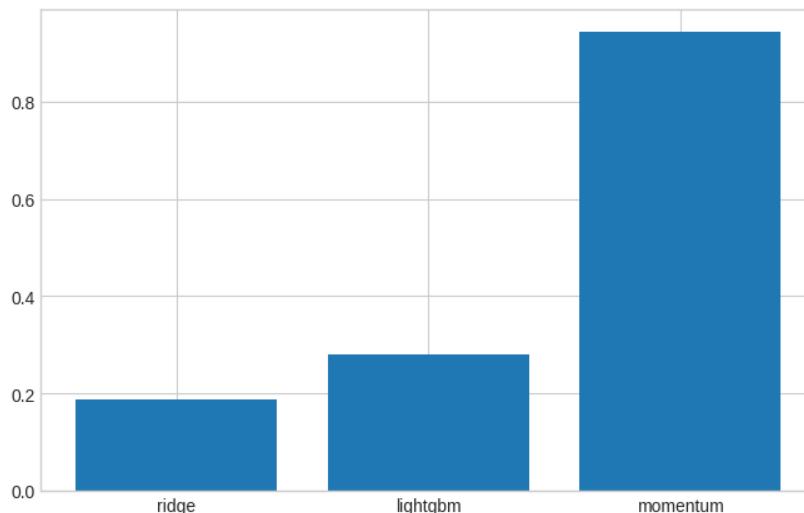
```

Overwriting ../skfin/ensemble.py

[17]: `from skfin.mv_estimators import Mbj`

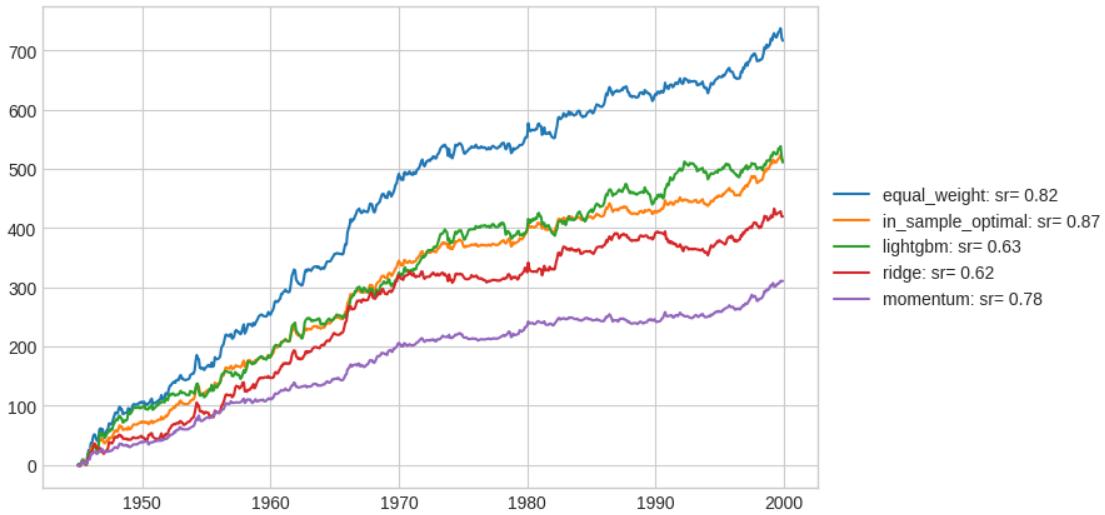
We first use the Britten-Jones (1999) trick (with the `Mbj` estimator) to compute the unconstrained mean-variance weights over the full sample:

[18]: `m = Mbj()
m.fit(pnls)
bar(pd.Series(m.coef_, index=pnls.columns))`



These in-sample optimal weights improve even more the sharpe ratio – but this is in-sample!

```
[19]: line(pnls_.assign(in_sample_optimal=Mbj().fit_transform(pnls)), cumsum=True)
```

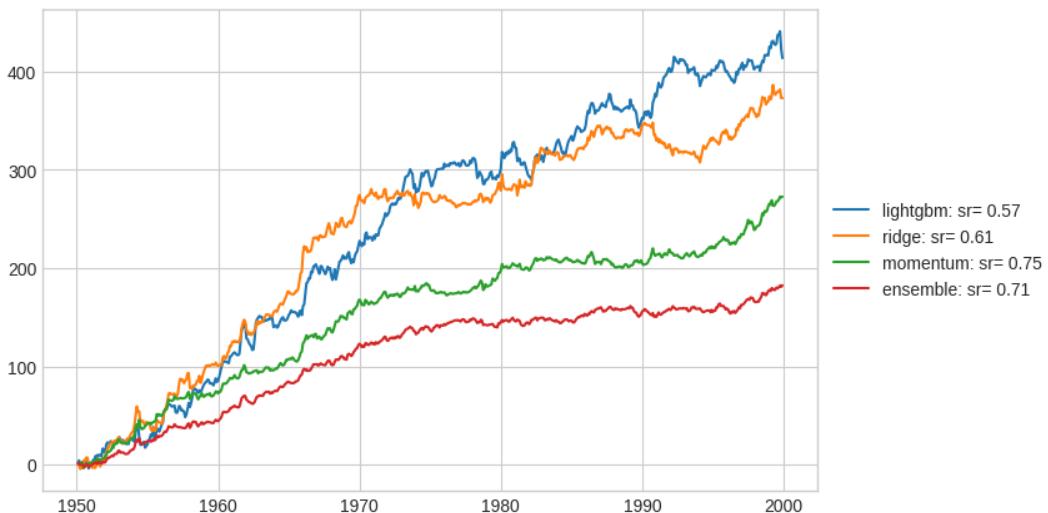


The StackingBacktester computes the performance with the learned weights over rolling windows of 60 months (5 years).

```
[20]: from skfin.ensemble import StackingBacktester
```

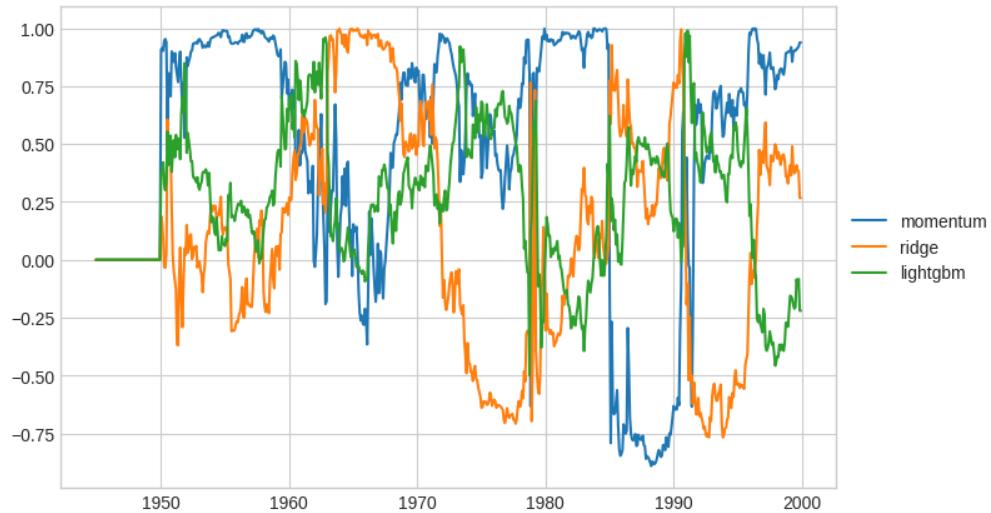
```
[21]: m = StackingBacktester(estimators=estimators, window=60, min_periods=60).train(
    X, y, ret
)
pnls = pnls.assign(ensemble_mbj=m.pnls_[ "ensemble" ])
```

```
[22]: line(m.pnls_[ "1950-02-01": ], cumsum=True)
```



To understand why the performance is lower, it is useful to look at the weights – in this case, the weights are often negative.

```
[23]: line(m.coef_)
```



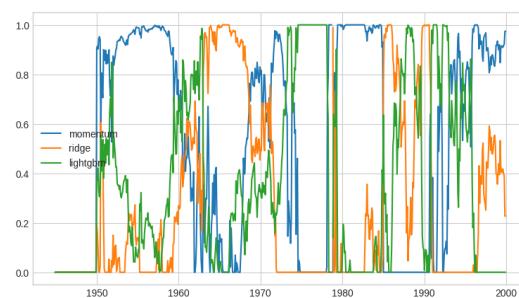
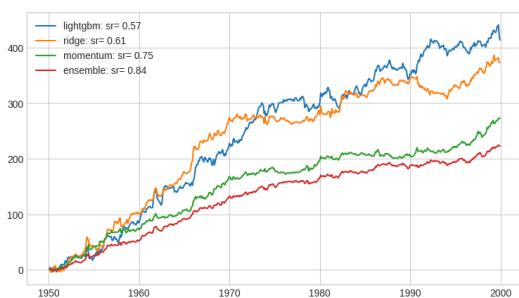
We redo the exercise with a positive-weight constraint.

```
[24]: from skfin.mv_estimators import Mbj
```

```
[25]: m = StackingBacktester(
    estimators=estimators,
    final_estimator=Mbj(positive=True),
    window=60,
    min_periods=60,
)
m.train(X, y, ret)
pnls["ensemble_mbj_positive"] = m.pnls_["ensemble"]
```

```
/home/schamponn/dev/projects/skfin/skfin/mv_estimators.py:139: RuntimeWarning:
invalid value encountered in divide
    self.coef_ = m.coef_ / np.sqrt(np.sum(m.coef_**2))
```

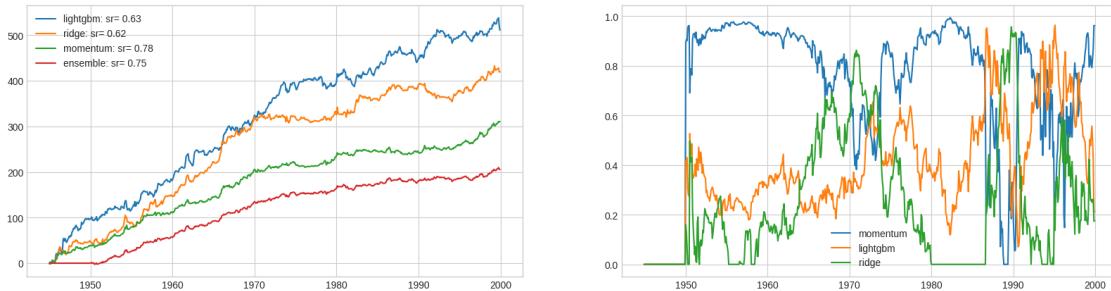
```
[26]: fig, ax = plt.subplots(1, 2, figsize=(20, 5))
line(m.pnls_[1950-02-01:""], cumsum=True, ax=ax[0], loc="best")
line(m.coef_, ax=ax[1], loc="best")
```



Over longer periods with positive constraints, the performance is closer to the industry momentum.

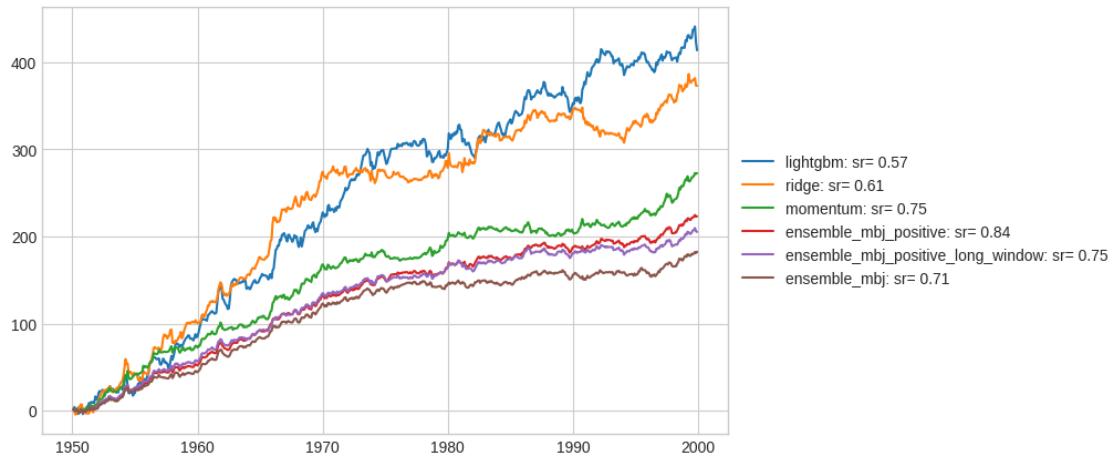
```
[27]: m = StackingBacktester(
    estimators=estimators,
    final_estimator=Mbj(positive=True),
    window=180,
    min_periods=60,
)
m.train(X, y, ret)
pnls["ensemble_mbj_positive_long_window"] = m.pnls_["ensemble"]
```

```
[28]: fig, ax = plt.subplots(1, 2, figsize=(20, 5))
line(m.pnls_, cumsum=True, ax=ax[0], loc="best")
line(m.coef_, ax=ax[1], loc="best")
```



Putting the different ensembles, we compare the pnls in the graph below

```
[29]: line(pnls["1950-02-01":], cumsum=True)
```



Chapter 15

Incorporating Transaction Costs and Market Microstructure in Portfolio Optimization

The goal of this section is to take transaction costs into account when evaluating a strategy. More precisely, discussing the cost of executing a trade requires an introduction to market microstructure. Beyond explicit costs, there are implicit trading costs that can significantly impact the pnl of a strategy. Intuititvely, the price will move against a party who wants to execute a significant trade – ie. the price will increase (resp. decrease) as a party wants to buy (resp. sell) a significant amount of security. Crucially, this can happen even if no trading happens.

In the context of this section, the impact of transaction is evaluated along two directions:

- evaluate whether the strategy pnl covers costs;
- improve the strategy so that the transaction costs are lower.

The latter point generally highlights a tradeoff between alpha (trading quickly to avoid alpha decay) and costs (avoid costs from execessing trading).

15.1 Order book and trades

In this section, we discuss the costs associated to trading following Section 3.7 in *Engineering Investment Process* by Ielpo, Merhy and Simon (2017). A rule of thumbs is

“It takes one day of volume to trade one day of volatility.”

In other words: *the bigger trade, the higher the cost*. There are three main categories of trade execution:

1. over-the-counter (OTC): the price of the transaction is directly negotiated with another market participant;
2. delegated-execution: the trade is delegated to a broker with a guarantee of execution price (e.g. price at close or volume-weighted average price as VWAP).
3. direct trade and execution on markets.

At any given point in time, some market participants can make public their willingness to trade a quantity at a given price in an *order book*. More precisely, an order book refers to an electronic list of buy and sell orders for a specific security or financial instrument organized by price level.

The bid-ask spread is the difference between the highest price that a buyer is willing to pay for an asset and the lowest price that a seller is willing to accept. More precisely, there are two types of orders:

1. market order: this is an instruction by an investor to a broker to buy or sell an assets at the best available price in the order book.
2. limit order: this is an instruction to a broker to buy or sell only a certain quantity at a certain price.

There is a tradeoff between trading immediately at a known (and potentially high) price and trading later, but at an unknown time/date and for an unknown price. In this context, there are two different types of costs:

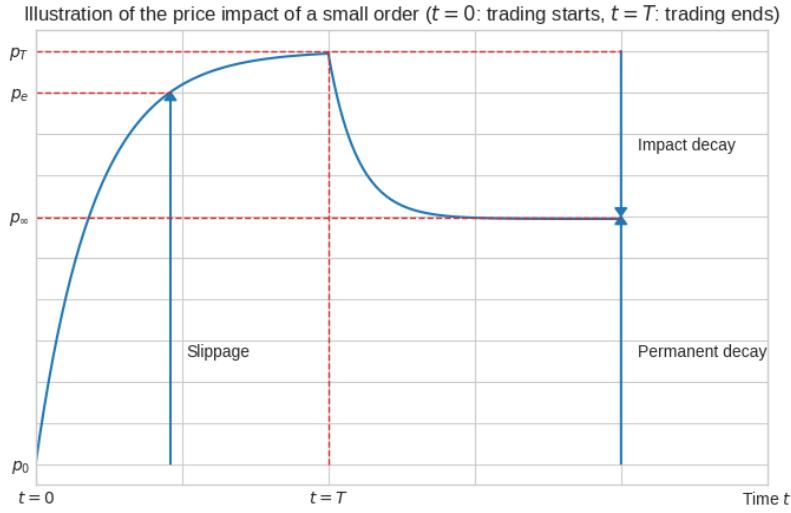
- direct/explicit costs: commissions, brokerage fees, settlement/clearing fees, exchange fees, taxes. This also includes shorting costs with easy-to-borrow / hard-to-borrow rates.
- indirect/implicit/impact costs: loosely, these costs capture the idea that the bigger the trade, the higher the cost. But typically, each investor faces impact costs depending on execution choice, execution pattern, latency, procedure speed and horizon. These impact costs are estimated based on (potentially private) past trades.

15.2 Impact costs

The figure below is from Ielpo et al. (2017, p. 239):

- p_0 : decision price
- p_T : final price
- p_e : average executing price
- p_∞ : after the trade, the price reverts to a limit value
- slippage = $\frac{p_e}{p_0} - 1$
- permanent impact = $\frac{p_\infty}{p_0} - 1$

Given that trades are motivated by a view of the market (ie. some alpha opportunity), it is difficult to distinguish alpha et pure impact.



In practice, we define the impact cost as a function I :

$$Q \mapsto I(Q),$$

where Q is the traded quantities. The price impact in dollars of trading the quantity Q is

$$\text{ImpactCost} = Q \times I(Q).$$

The fraction of traded volume is $\delta = \frac{Q_{n,t}}{\text{Liquidity}_{n,t}}$ can be interpreted as time – e.g. how many days (or fraction of a day) to trade $Q_{n,t}$ given the asset liquidity $\text{Liquidity}_{n,t}$. Over that period, the asset volatility scaled in $\sigma_{n,t}\sqrt{\delta}$, so that a simple impact model is:

$$I(Q) = \sigma \sqrt{\frac{Q}{\text{Liquidity}}}.$$

More generally, two parameters $\langle \gamma, \beta \rangle$ are fitted with the functional form:

$$I(Q) = \gamma \sigma \left(\frac{Q}{\text{Liquidity}} \right)^\beta.$$

A rule of thumb is as follows:

- for small horizons, $\beta \approx 0$, $I(Q)$ is constant (e.g. bid-ask spread) and the total cost is linear;
- for long horizons, $\beta \approx 1$ and the total impact cost $Q \times I(Q)$ is quadratic.

In analytically more complex (and potentially more realistic) setups, β is taken to be $1/2$ as motivated above.

In what follows, we focus on quadratic impact cost with $\beta = 1$.

15.3 Mean-variance optimisation with quadratic costs

With quadratic cost, the mean-variance optimisation can be solved in closed form. More precisely, the mean-variance utility at time t with transaction cost is

$$h_t^T \alpha - \frac{h_t^T V h_t}{2\lambda} - \frac{(h_t - h_{t-1})^T \Lambda (h_t - h_{t-1})}{2},$$

where Λ is a liquidity factor on each asset.

Lemma. Given the mean-variance utility with quadratic cost, the optimal portfolio is

$$h_t = \left[\frac{V}{\lambda} + \Lambda \right]^{-1} (\alpha + \Lambda h_{t-1}).$$

Proof. The mean-variance objective is rewritten as

$$h_t^T (\alpha + \Lambda h_{t-1}) - \frac{h_t^T (V + \lambda \Lambda) h_t}{2\lambda} - \frac{h_{t-1}^T \Lambda h_{t-1}}{2}$$

The optimal portfolio with transaction cost is a weighted average of two holdings:

$$h_t = (I + \lambda V^{-1} \Lambda)^{-1} \lambda V^{-1} \alpha + [I - (I + \lambda V^{-1} \Lambda)^{-1}] h_{t-1}.$$

Interpretation:

- the friction-less mean-variance portfolio $\lambda V^{-1} \alpha$ is the target portfolio
- the optimal portfolio h_t is a weighted-average of this target portfolio and the past holdings h_{t-1} :
- ideally the investor would like to hold the target portfolio but because of transaction cost, it only trades partially towards this target.
- when the transaction costs are lower and the risk-tolerance λ is higher, the investor loads more on the mean-variance target $\lambda V^{-1} \alpha$
- the transaction imply so form of (non-linear) exponential averaging.

15.4 Positions and turnover

In previous sections, the positions h_t have been rescaled so that

$$\sqrt{h_t^T V_t h_t} = 1$$

and

$$\sum_t h_{n,t} = 0.$$

More generally the positions are denominated in dollars as:

$$pos_t = \theta \times h_t,$$

where the scalar is a risk target in dollars: $\theta = \sqrt{\text{pos}_t^T V_t \text{pos}_t}$. Given this risk target, the (two-sided) gross leverage

$$\text{Leverage} = \sum_n |\text{pos}_{n,t}|$$

From (rescaled) positions in asset n are denoted $h_{n,t}$, the trades *quantities* are:

$$q_{n,t} = \text{pos}_{n,t} - (1 + r_t) \text{pos}_{n,t-1}.$$

- (One-sided) turnover

$$\text{TrueTurnover}_t = \sum_n |\text{pos}_{n,t} - (1 + r_t) \text{pos}_{n,t-1}| / 2$$

where $h_{n,t} - (1 + r_{n,t})h_{n,t-1}$ is the trade. In most of what we do, approximating the small returns away is good enough:

$$\text{Turnover}_t = \sum_n |\text{pos}_{n,t} - \text{pos}_{n,t-1}| / 2$$

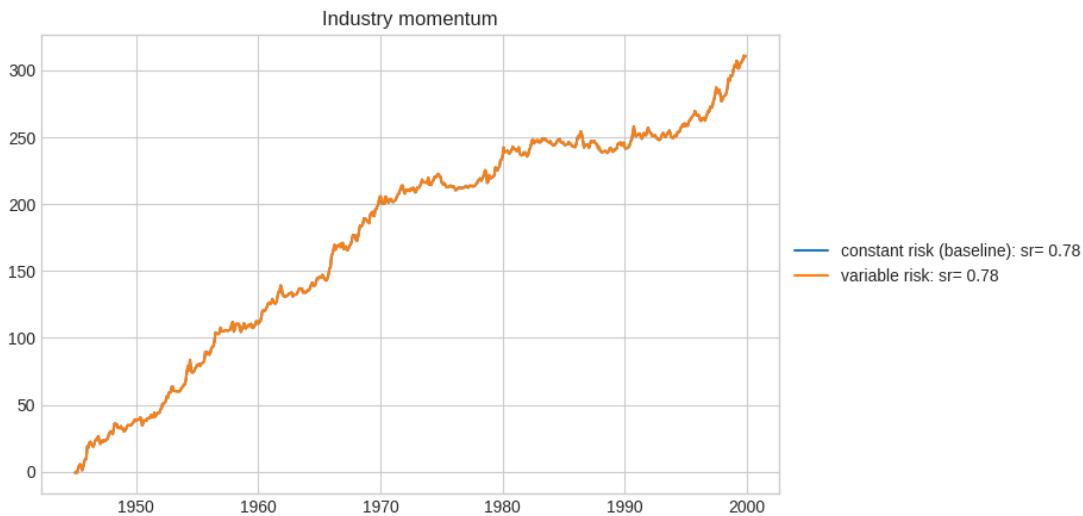
The portfolio average holding period refers to the average length of time that investments are held within a portfolio before being sold or replaced. It is a measure often used to assess the turnover rate of the portfolio, providing insight into the investment strategy being employed—whether it is more short-term or long-term in nature.

$$\text{HP} = \frac{\frac{1}{T} \sum_{t=1}^T \text{Leverage}_t}{\frac{1}{T} \sum_{t=1}^T \text{Turnover}_t}$$

15.5 Cost for the Industry Momentum backtest

Given that the data is monthly, we re-estimate the model every month and the maximum train window is 12 months.

```
[3]: estimators_ = {
    "constant risk (baseline)": MeanVariance(risk_target=1),
    "variable risk": MeanVariance(risk_target=None),
}
line(
    {k: Backtester(v).train(features, target, ret) for k, v in estimators_.items()},
    cumsum=True,
    title="Industry momentum",
)
```



15.5.1 Turnover

```
[4]: def average_holding_period(h):
    h = h.squeeze()
    if isinstance(h, pd.Series):
        return h.abs().mean() / h.diff().abs().div(2).mean()
    else:
        return h.abs().sum(axis=1).mean() / h.diff().abs().sum(axis=1).div(2).mean()
```

```
[5]: m0 = (
    Backtester(MeanVariance(risk_target=None))
    .compute_holdings(features, target)
    .compute_pnl(ret)
)

hp = average_holding_period(m0.h_)

print(f"The average holding period is {hp:.2f} months")
```

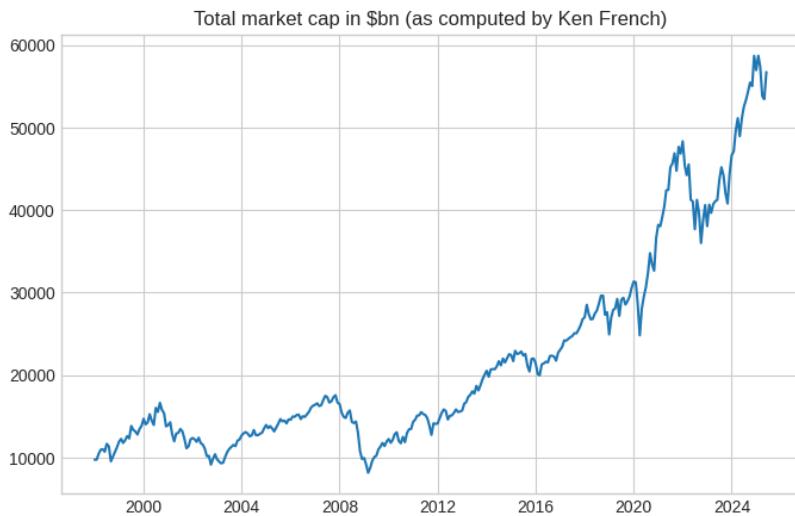
The average holding period is 5.53 months

15.5.2 Liquidity and market cap

Given that we do not have data on sector liquidity, we use the market capitalisation as a proxy for liquidity.

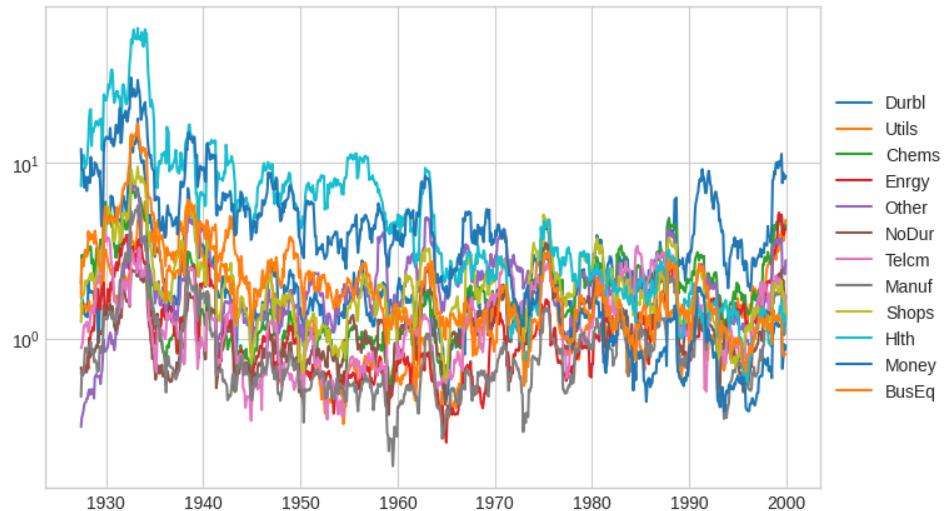
```
[6]: mcap = returns_data["Monthly"]["Average_Firm_Size"].mul(
    returns_data["Monthly"]["Number_of_Firms_in_Portfolios"]
)
```

```
[7]: line(
    mcap.loc["1998":].sum(axis=1).div(1e3),
    title="Total market cap in $bn (as computed by Ken French)",
    legend=False,
)
```



```
[8]: gamma = 1 / 20
mcap_norm = mcap.div(mcap.sum(axis=1), axis=0)
vol_liquidity_factor = 0.5 * gamma * ret.rolling(12).std().div(mcap_norm.loc[ret.index])
```

```
[9]: line(vol_liquidity_factor, yscale="log")
```



15.5.3 Mark-to-market and backtesting

In this case, Λ is the diagonal of the vector of return standard deviations over market capitalisation (as a proxy for liquidity – e.g. traded volume).

```
[10]: %%writefile ../skfin/backtesting_with_cost.py
from dataclasses import dataclass

import numpy as np
import pandas as pd
```

```

from sklearn.base import BaseEstimator, clone
from sklearn.model_selection import TimeSeriesSplit
from skfin.mv_estimators import MeanVariance

def compute_pnl_components(h, ret, vol_liquidity_factor=None):
    ret = ret[h.index[0] : h.index[-1]]
    vol_liquidity_factor = vol_liquidity_factor.loc[h.index[0] : h.index[-1]]

    pnl = h.shift(1).mul(ret).sum(axis=1)
    if vol_liquidity_factor is not None:
        impact_cost = h.diff().pow(2).mul(vol_liquidity_factor).sum(axis=1)
        return {
            "gross": pnl,
            "net = gross - impact cost": pnl.sub(impact_cost),
            "impact cost": -1 * impact_cost,
        }
    else:
        return pnl


def compute_batch_holdings_with_cost(
    pred, V, A=None, past_h=None, vol_liquidity_factor=None, lambda_=None, risk_target=None
):
    """
    compute markowitz holdings with return prediction "mu" and covariance matrix "V"
    mu: numpy array (shape N * K)
    V: numpy array (N * N)

    """
    if (lambda_ is None) & (vol_liquidity_factor is not None):
        lambda_ = 1
    N, _ = V.shape
    if isinstance(pred, pd.Series) | isinstance(pred, pd.DataFrame):
        pred = pred.values
    if pred.shape == (N,):
        pred = pred[:, None]
    elif pred.shape[1] == N:
        pred = pred.T

    if vol_liquidity_factor is not None:
        invV = np.linalg.inv(V / lambda_ + 2 * np.diag(vol_liquidity_factor))
    else:
        invV = np.linalg.inv(V)
    if A is None:
        M = invV
    else:
        U = invV.dot(A)
        if A.ndim == 1:
            M = invV - np.outer(U, U.T) / U.dot(A)
        else:
            M = invV - U.dot(np.linalg.inv(U.T.dot(A)).dot(U.T))
    if (vol_liquidity_factor is not None) & (past_h is not None):
        h = M.dot(pred + 2 * np.diag(vol_liquidity_factor).dot(past_h.T))
    else:
        h = M.dot(pred)
    return h.T

```

`@dataclass`

Chapter 15. Incorporating Transaction Costs and Market Microstructure in Portfolio Optimization

```
class MeanVarianceWithCost(MeanVariance):
    """
    Mean-variance optimization estimator with transaction cost considerations.
    """

    @staticmethod
    def compute_batch_holdings(pred, V, A, **kwargs):
        """
        Compute portfolio holdings considering transaction costs.

        Parameters:
            pred (np.ndarray): Predicted returns.
            V (np.ndarray): Covariance matrix.
            A (np.ndarray): Constraint matrix.
            **kwargs: Additional keyword arguments.

        Returns:
            np.ndarray: Portfolio holdings considering costs.
        """

        return compute_batch_holdings_with_cost(pred, V, A, **kwargs)

@dataclass
class BacktesterWithCost:
    estimator: BaseEstimator = MeanVarianceWithCost()
    vol_liquidity_factor: pd.DataFrame=None
    max_train_size: int=36
    test_size: int=1
    start_date: str="1945-01-01"
    end_date: str=None
    h_init: pd.Series=None
    return_pnl_component: bool=False

    def train(self, X, y, ret):
        X = X.loc[:self.end_date]
        cv = TimeSeriesSplit(
            max_train_size=self.max_train_size,
            test_size=self.test_size,
            n_splits=1 + len(X.loc[self.start_date : ]) // self.test_size,
        )

        _h = []
        past_h = self.h_init
        for train, test in cv.split(X):
            m = clone(self.estimator)
            m.fit(X.iloc[train], y.iloc[train])
            if self.vol_liquidity_factor is None:
                vlf = None
            else:
                vlf = np.squeeze(self.vol_liquidity_factor.values[test])
            current_h = m.predict(
                X.iloc[test], past_h=past_h, vol_liquidity_factor=vlf
            )
            _h += [current_h]
            past_h = current_h

        cols = X.columns
        idx = X.index[np.concatenate([test for _, test in cv.split(X)])]
        h_ = pd.DataFrame(np.concatenate(_h), index=idx, columns=cols)

        self.h_ = h_
        if self.return_pnl_component:
```

```

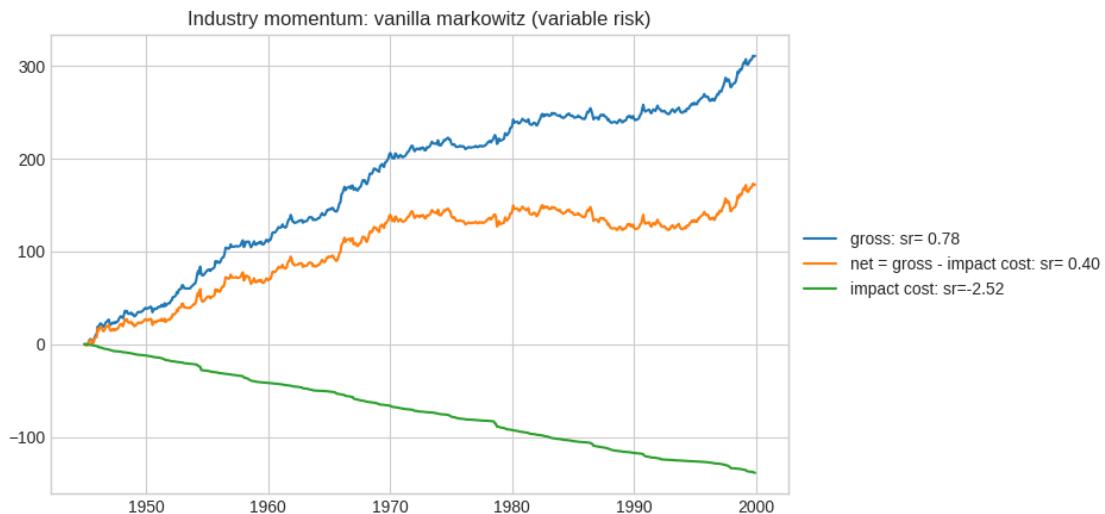
        self.pnl_ = compute_pnl_components(
            self.h_, ret, vol_liquidity_factor=self.vol_liquidity_factor
        )
    else:
        self.pnl_ = (
            h_.shift(1).mul(ret).sum(axis=1)[self.start_date : self.end_date]
        )
    return self

```

Overwriting .../skfin/backtesting_with_cost.py

```
[11]: from skfin.backtesting_with_cost import compute_pnl_components
```

```
[12]: pnlc = compute_pnl_components(m0.h_, ret, vol_liquidity_factor=vol_liquidity_factor)
line(pnlc, cumsum=True, title="Industry momentum: vanilla markowitz (variable risk)")
```



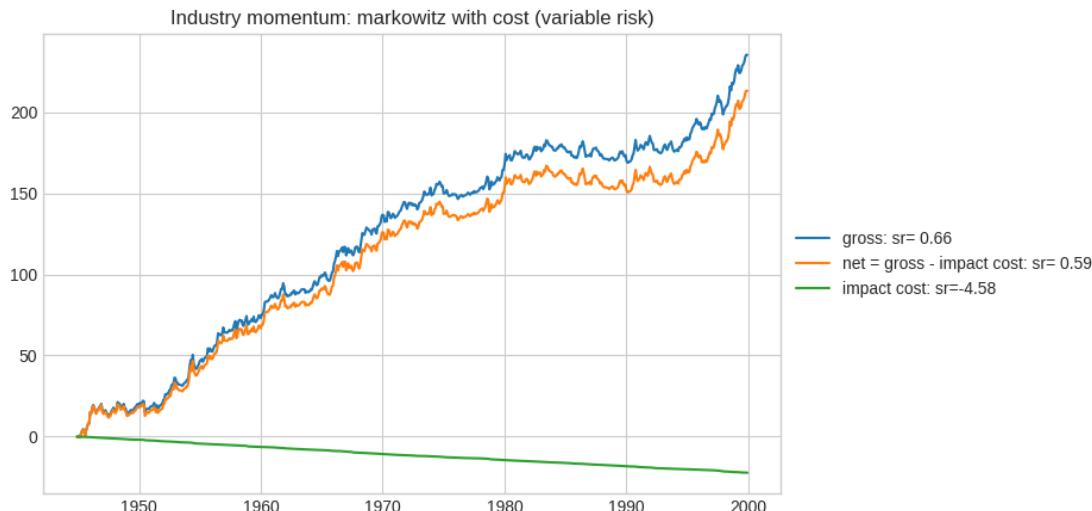
15.5.4 Backtesting with cost

```
[13]: from skfin.backtesting_with_cost import BacktesterWithCost, MeanVarianceWithCost
```

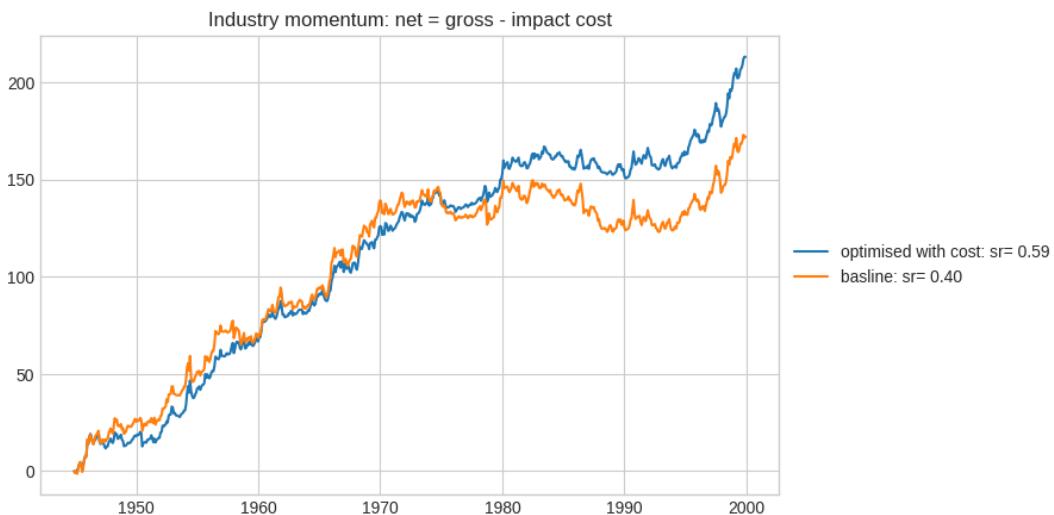
```
[14]: m0_ = BacktesterWithCost(MeanVarianceWithCost(), vol_liquidity_factor=None).train(
    features, target, ret
)
np.allclose(m0_.pnl_, m0.pnl_)
```

```
[14]: True
```

```
[15]: m = BacktesterWithCost(
    MeanVarianceWithCost(), vol_liquidity_factor, return_pnl_component=True
).train(features, target, ret)
line(
    m.pnl_, cumsum=True, title="Industry momentum: markowitz with cost (variable risk)"
)
```



```
[16]: c = "net = gross - impact cost"
line(
    {"baseline": pnlc[c], "optimised with cost": m.pnl_[c]},
    cumsum=True,
    title=f"Industry momentum: {c}",
)
```



Comparing the average holding periods, we see a sharp increase.

```
[17]: print(
    f"Average holding period (in month):\n - baseline = {average_holding_period(m0.h_):.1f},\n - optimised with quadratic cost = {average_holding_period(m.h_):.1f}.\n")
```

Average holding period (in month):
- baseline = 5.5,
- optimised with quadratic cost = 10.3.

The scatterplot below shows at the industry level:

- the average liquidity factor
- the average holding period

Intuitive, the more expensive it is to trade a sector, the longer the holding period should be.

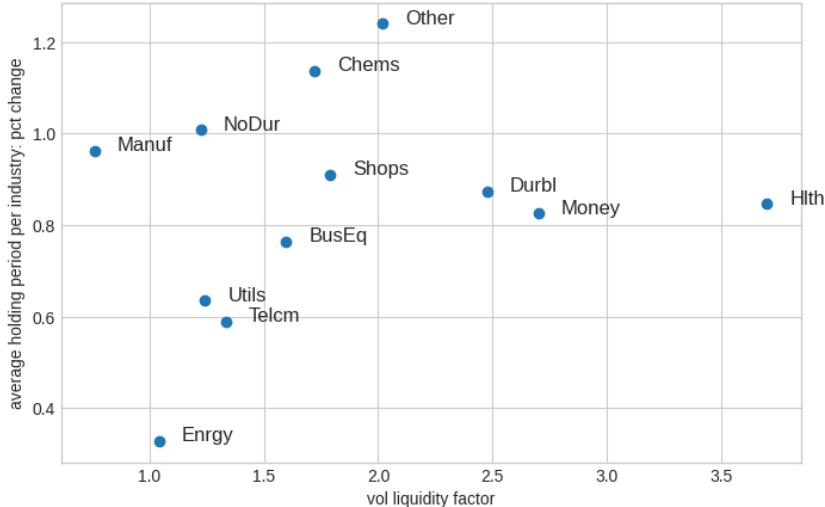
```
[18]: fig, ax = plt.subplots(1, 1, figsize=(8, 5))

avp = (
    pd.Series({c: average_holding_period(m.h_[c]) for c in m0.h_.columns})
    .div(pd.Series({c: average_holding_period(m0.h_[c]) for c in m0.h_.columns}))
    .sub(1)
)

df = pd.concat(
{
    "vol liquidity factor": vol_liquidity_factor.loc[m0.h_.index[0] :].mean(),
    "average holding period per industry: pct change": avp,
},
axis=1,
)

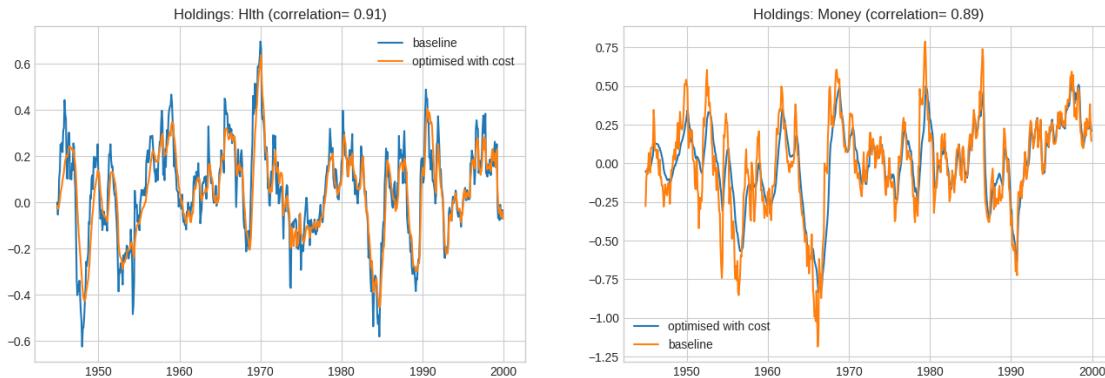
cols = list(df.columns)
idx = list(df.index)

ax.scatter(df.iloc[:, 0].values, df.iloc[:, 1].values)
ax.set_xlabel(cols[0])
ax.set_ylabel(cols[1])
xshift, yshift, rotation = 0.1, 0, 0
for i, txt in enumerate(idx):
    ax.text(
        df.iloc[i, 0] + xshift,
        df.iloc[i, 1] + yshift,
        txt,
        fontsize=12,
        rotation=rotation,
    )
)
```



Chapter 15. Incorporating Transaction Costs and Market Microstructure in Portfolio Optimization

```
[19]: cols = ["Hlth", "Money"]
fig, ax = plt.subplots(1, 2, figsize=(16, 5))
for i, c in enumerate(cols):
    df = pd.concat({"baseline": m0.h_[c], "optimised with cost": m.h_[c]}, axis=1)
    line(
        df,
        title=f"Holdings: {c} (correlation={df.corr().iloc[0, 1]: .2f})",
        ax=ax[i],
        loc="best",
    )
```



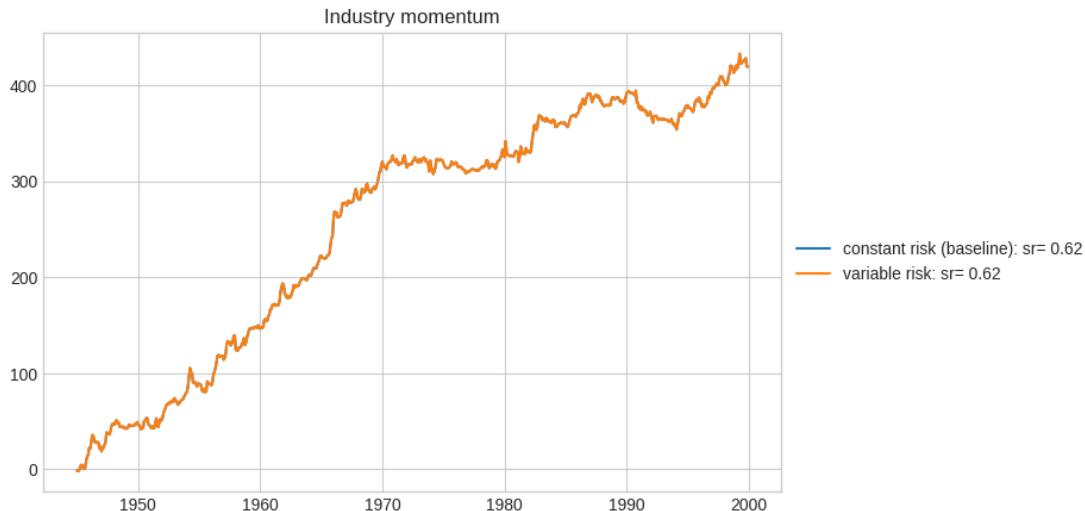
For healthcare, the volatility/liquidity factor seems high on average, but decreases sharply over the period – which explains that the average holding period increases less than expected.

15.5.5 Ridge backtest with cost

```
[20]: from skfin.estimators import Ridge
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
```

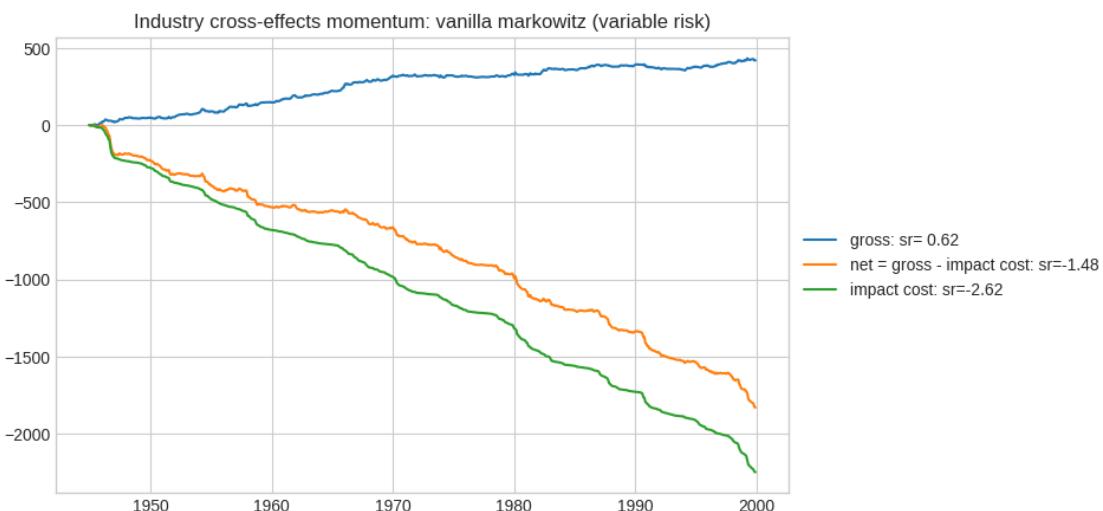


```
[21]: estimators_ = {
    "constant risk (baseline)": make_pipeline(
        StandardScaler(with_mean=False), Ridge(), MeanVariance(risk_target=1)
    ),
    "variable risk": make_pipeline(
        StandardScaler(with_mean=False), Ridge(), MeanVariance(risk_target=None)
    ),
}
line(
    {k: Backtester(v).train(features, target, ret) for k, v in estimators_.items()},
    cumsum=True,
    title="Industry momentum",
)
```



```
[22]: estimator_ = make_pipeline(
    StandardScaler(with_mean=False), Ridge(), MeanVarianceWithCost()
)
m0_ = BacktesterWithCost(estimator_, vol_liquidity_factor=None).train(
    features, target, ret
)

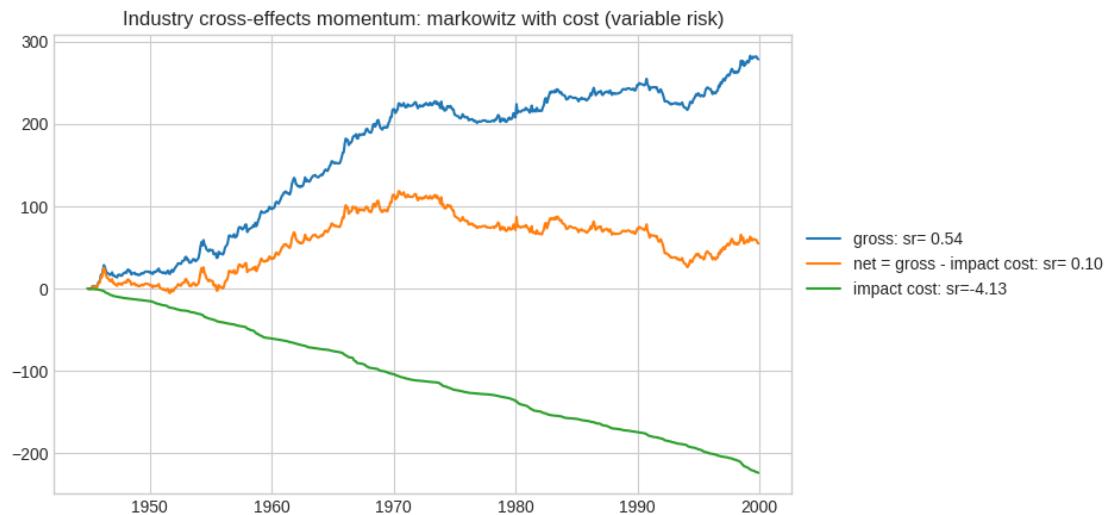
pnlc = compute_pnl_components(m0_.h_, ret, vol_liquidity_factor=vol_liquidity_factor)
line(
    pnlc,
    cumsum=True,
    title="Industry cross-effects momentum: vanilla markowitz (variable risk)",
)
```



```
[23]: m_ = BacktesterWithCost(
    estimator_,
    vol_liquidity_factor=vol_liquidity_factor,
```

Chapter 15. Incorporating Transaction Costs and Market Microstructure in Portfolio Optimization

```
    return_pnl_component=True,
).train(features, target, ret)
line(
    m_.pnl_,
    cumsum=True,
    title="Industry cross-effects momentum: markowitz with cost (variable risk)",
)
```



A key difference between the baseline industry momentum strategy and the strategies that lean cross-effects (e.g. learning with a Ridge estimator) is these strategies are faster. In this case, taking cost into account to slow down the strategy is even more important.

```
[24]: print(
    f"Average holding period (in month):\n - baseline Ridge = {average_holding_period(m0_.h_):.1f},\n - optimised with quadratic cost = {average_holding_period(m_.h_):.1f}.\")
```

Average holding period (in month):
- baseline Ridge = 2.5,
- optimised with quadratic cost = 4.5.

Chapter 16

Data Challenges in Calculating Historical Stock Returns

```
[2]: import pandas as pd
from skfin.plot import *
from tqdm.auto import tqdm
```

In this section, we discuss two issues related to stock data: - survivorship bias - matching datasets (e.g. tickers and company names).

Survivorship bias in data analysis refers to the error that occurs when only the “surviving” or currently existing subjects are considered, while those that have ceased to exist or dropped out are ignored. This leads to overly optimistic results and conclusions because the analysis does not account for entities that failed, exited, or disappeared from the sample. For example, in financial studies, analyzing only funds that exist today ignores those that closed down due to poor performance, thereby distorting performance statistics.

Matching stock tickers and company names presents several challenges. First, ticker symbols can change over time due to corporate actions such as mergers, acquisitions, or rebranding, making it difficult to maintain a consistent mapping. Additionally, different stock exchanges may assign the same or similar tickers to entirely different companies, leading to ambiguity, especially in multinational datasets. Company names themselves can vary due to abbreviations, punctuation, company suffixes (like Inc., Corp., Ltd.), or changes following corporate events. Name spelling differences, typographical errors, and the use of aliases or former names further complicate matching.

16.1 Returns data

16.1.1 Scikit-learn returns

The main issue with this dataset compiled by Scikit-learn is we have no information on how the returns were constructed (and how the stocks were chosen at the time).

```
[3]: from skfin.dataloaders.constants.mappings import symbol_dict
from skfin.datasets_ import load_sklearn_stock_returns

clip = lambda x: x.T.clip(
    lower=x.quantile(q=0.01, axis=0), upper=x.quantile(0.99, axis=0), axis=0
).T
rescale = lambda x: x.div(x.std())
```

```
[4]: ret = load_sklearn_stock_returns(cache_dir="data") / 100
ret_clip = ret.pipe(clip)
```

16.1.2 Yahoo finance returns

The main issue with Yahoo finnace returns is the survivorship bias because the deslisted firms are no longer available in the past.

```
[5]: import yfinance as yf
```

```
[6]: tickers = list(symbol_dict.keys())
start, end = ret.index[0].strftime("%Y-%m-%d"), ret.index[-1].strftime("%Y-%m-%d")
```

```
[7]: data = yf.download(tickers, start=start, end=end, auto_adjust=True, progress=False)
data = data.dropna(how="all", axis=1)
```

```
[8]: tickers_ = data.columns.get_level_values(1).unique()
dividends = {}
for ticker in tickers_:
    stock = yf.Ticker(ticker)
    div = stock.dividends[start:end]
    if div.dtype == "O":
        div = div.str.replace(" USD", "").astype(float)
    dividends[ticker] = div

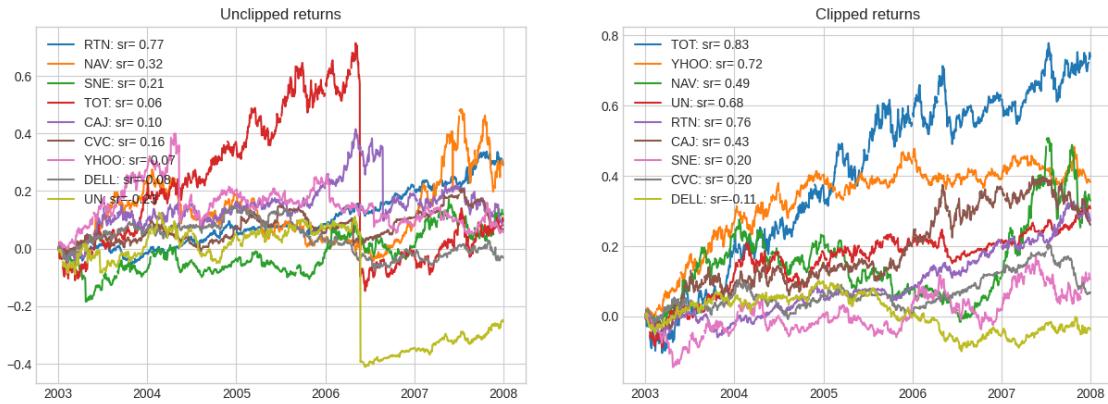
div = (
    pd.concat(dividends, axis=1)
    .pipe(lambda x: x.set_index(x.index.tz_localize(None)))
    .reindex_like(data["Close"])
    .fillna(0)
    .astype(float)
    .div(data["Close"]))
)
```

16.2 Survivorship bias and non-US firms

```
[9]: cols = ["CAJ", "SNE", "UN", "NAV", "TOT", "RTN", "DELL", "YHOO", "CVC"]
{c: symbol_dict[c] for c in cols}
```

```
[9]: {'CAJ': 'Canon',
'SNE': 'Sony',
'UN': 'Unilever',
'NAV': 'Navistar',
'TOT': 'Total',
'RTN': 'Raytheon',
'DELL': 'Dell',
'YHOO': 'Yahoo',
'CVC': 'Cablevision'}
```

```
[10]: fig, ax = plt.subplots(1, 2, figsize=(15, 5))
line(ret[cols], cumsum=True, ax=ax[0], title="Unclipped returns", loc="best")
line(ret_clip[cols], cumsum=True, ax=ax[1], title="Clipped returns", loc="best")
```



16.2.1 Dirty data and jumps

Total returns are defined as:

$$\text{total_returns}_t = \frac{\text{price}_t + \text{dividend}_t - \text{price}_{t-1}}{\text{price}_{t-1}}.$$

Adjustments are necessary when there are stock splits (or reverse splits) that alter the number of shares outstanding and the nominal price per share.

```
[11]: data.columns.get_level_values(0).unique()

[11]: Index(['Close', 'High', 'Low', 'Open', 'Volume'], dtype='object', name='Price')

[12]: pret = data["Close"].pct_change().dropna(how="all", axis=1)
       tret = pret.add(div)

[13]: rets = {
          "price returns": pret,
          "total returns": tret,
      }

[14]: axis = 0
       pd.DataFrame({k: v.corrwith(ret, axis=axis) for k, v in rets.items()}).mean()

[14]: price returns    0.815361
      total returns     0.805144
      dtype: float64

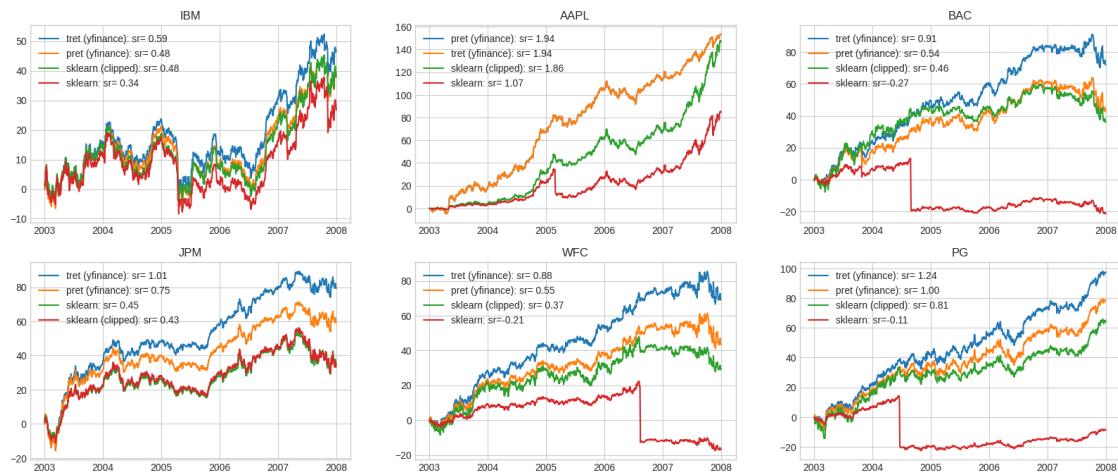
[15]: axis = 0
       pd.DataFrame({k: v.corrwith(ret_clip, axis=axis) for k, v in rets.items()}).mean()

[15]: price returns    0.943665
      total returns     0.931119
      dtype: float64

[16]: cols = ["IBM", "AAPL", "BAC", "JPM", "WFC", "PG"]
       fig, axs = plt.subplots(2, 3, figsize=(20, 8))
       axs = axs.ravel()
       for i, c in enumerate(cols):
           line(
```

```

pd.concat(
{
    "pret (yfinance)": pret[c],
    "tret (yfinance)": tret[c],
    "sklearn": ret[c],
    "sklearn (clipped)": ret_clip[c],
},
axis=1,
)
.fillna(0)
.pipe(rescale),
cumsum=True,
title=c,
ax=axs[i],
loc="best",
)
    
```



16.3 Matching to other datasets

Matching company names is often a time-consuming task: below we use the package `rapiddfuzz` to check the candidate matches.

```
[17]: from skfin.datasets_ import load_10X_summaries
from skfin.dataloaders.constants.mappings import mapping_10X

df = load_10X_summaries()
```

```
[18]: v = mapping_10X["CVC"]
print(v)
df.loc[lambda x: x.CoName.isin(v if isinstance(v, list) else [v])].loc[
    ret.index[0] : ret.index[-1]
].loc["2006"].iloc[:, :10]
```

CABLEVISION SYSTEMS CORP /NY

```
[18]:      CIK  FILING_DATE          ACC_NUM        CPR FORM_TYPE \
date
2006-05-10  1053112     20060510  0001104659-06-032803  20060331      10-Q
2006-05-10    784681     20060510  0001104659-06-032803  20060331      10-Q
```

2006-09-21	1053112	20060921	0001104659-06-062197	20051231	10-K-A
2006-09-21	784681	20060921	0001104659-06-062197	20051231	10-K-A
2006-09-21	1053112	20060921	0001104659-06-062201	20060331	10-Q-A
2006-09-21	784681	20060921	0001104659-06-062201	20060331	10-Q-A
2006-11-08	1053112	20061108	0001104659-06-072875	20060930	10-Q
2006-11-08	784681	20061108	0001104659-06-072875	20060930	10-Q

		CoName	SIC	FFInd	N_Words	N_Unique_Words
date						
2006-05-10	CABLEVISION SYSTEMS CORP /NY	4841	32	38566	2327	
2006-05-10	CABLEVISION SYSTEMS CORP /NY	4841	32	38566	2327	
2006-09-21	CABLEVISION SYSTEMS CORP /NY	4841	32	92495	2946	
2006-09-21	CABLEVISION SYSTEMS CORP /NY	4841	32	92495	2946	
2006-09-21	CABLEVISION SYSTEMS CORP /NY	4841	32	35794	1966	
2006-09-21	CABLEVISION SYSTEMS CORP /NY	4841	32	35794	1966	
2006-11-08	CABLEVISION SYSTEMS CORP /NY	4841	32	45265	2148	
2006-11-08	CABLEVISION SYSTEMS CORP /NY	4841	32	45265	2148	

```
[19]: CoName = list(
    df.assign(CoName=lambda x: x.CoName.str.upper())
    .groupby(["date", "CoName"])["FILING_DATE"]
    .count()
    .loc[ret.index[0] : ret.index[-1]]
    .groupby(level=1)
    .count()
    .index
)

from rapidfuzz import fuzz

pd.Series({c: fuzz.token_set_ratio("CABLEVISION", c) for c in CoName}).sort_values(
    ascending=False
).head(5)
```

```
[19]: CABLEVISION SYSTEMS CORP /NY      100.000000
PARKERVISION INC                  66.666667
CAGLES INC                         66.666667
ABLEST INC                          66.666667
LATIN TELEVISION INC                64.516129
dtype: float64
```

Chapter 17

Risk Management and Mean-Reversion Strategies in Hedge Funds: Lessons from the 2007 Quant Crisis

Hedge funds generate returns by taking on risks, which can be categorized into idiosyncratic risk and aggregate risk. Idiosyncratic risk is specific to individual assets or sectors and can be mitigated through diversification. Aggregate risk, affecting the broader market, cannot be diversified away and requires hedging strategies. The returns earned by hedge funds are the reward for navigating and exploiting these risks effectively.

In this section, we discuss one particular risk-taking strategy on stocks – liquidity provision (or mean-reversion) – and discuss its performance during the period of elevated aggregated risk in August 2007.

Hedge funds need to consider two key factors in their overall approach to risk-taking:

- Risk management: Effective risk management often relies on improved risk models. We will examine the use of risk-model shrinkage as a method to decrease the over-realization of risk.
- Leverage management: another aspect of risk is the management of leverage and we discuss how to reduce the risk-per-unit of leverage by concentrating the positions or the predictors.

17.1 Liquidity and autocorrelation of stock returns



JOURNAL OF INVESTMENT MANAGEMENT, Vol. 5, No. 4, (2007), pp. 5–54
 © JOIM 2007

WHAT HAPPENED TO THE QUANTS IN AUGUST 2007?

Amir E. Khandani^a and Andrew W. Lo^b

During the week of August 6, 2007, a number of quantitative long/short equity hedge funds experienced unprecedented losses. Based on TASS hedge-fund data and simulations of a specific long/short equity strategy, we hypothesize that the losses were initiated by the rapid “unwind” of one or more sizable quantitative equity market-neutral portfolios. Given the speed and price impact with which this occurred, it was likely the result of a forced liquidation by a multi-strategy fund or proprietary-trading desk, possibly due to a margin call or a risk reduction. These initial losses then put pressure on a broader set of long/short and long-only equity portfolios, causing further losses by triggering stop/loss and de-leveraging policies. A significant rebound of these strategies occurred on August 10th, which is also consistent with the unwind hypothesis. This dislocation was apparently caused by forces outside the long/short equity sector—in a completely unrelated set of markets and instruments—suggesting that systemic risk in the hedge-fund industry may have increased in recent years.

Hedge funds as provider of liquidity is trend perfected by Long-Term Capital Management (until its fatal crash in 1998...):

“by taking long positions in stocks that have declined and short positions in stocks that have advanced over the previous day, the strategy actively provides liquidity to the marketplace;”

This type of strategies generally requires a very high leverage, so anticipating market dislocations is key. Khandani-Lo (2007) argue that hedge funds have overtook traditional market makers as liquidity providers, so much so that when hedge funds retract, the whole market collapses...

Mean-reversion is a contrarian strategy where the holdings positive in past losers and negative in past winners:

$$h_{n,t} = -1 \times \frac{1}{N} (r_{n,t} - r_{\text{Market},t}),$$

where $r_{\text{Market},t} = \frac{1}{N} \sum_n r_{n,t}$ is the cross-sectional average. By construction, such a strategy is cash-neutral:

$$\sum_n h_{n,t} = 0.$$

Khandani and Lo (2007) show that a market-making liquidity provision exhibited extreme losses during the week of August 6, 2007 suggesting market-wide deleveraging.

“We wish to acknowledge at the outset that the hypotheses advanced in this paper are speculative, tentative, and based solely on indirect evidence.”

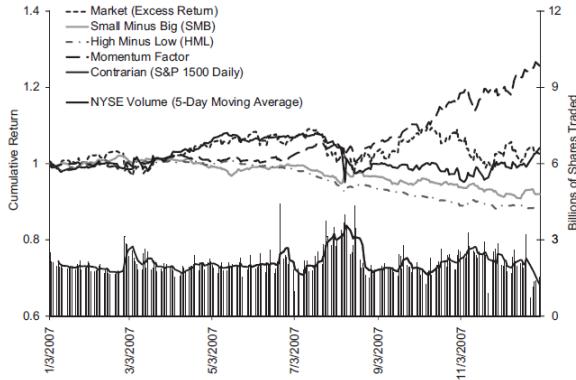


Fig. 1. Factor performance and volume in 2007. The cumulative daily returns for the Market, Small Minus Big (SMB), High Minus Low (HML), Momentum factors as well as the contrarian strategy of Lehmann (1990) and Lo and MacKinlay (1990) for January 3, 2007, to December 31, 2007. Data for Market, SMB, HML, and Momentum factors were obtained from Kenneth French's website (see footnote 8 for details). The contrarian strategy was implemented as in Khandani and Lo (2007) using daily return for stocks listed in the S&P 1500 on January 3, 2007. Volume represents the Daily NYSE Group Volume in NYSE-listed securities obtained from the NYSE Euronext website.

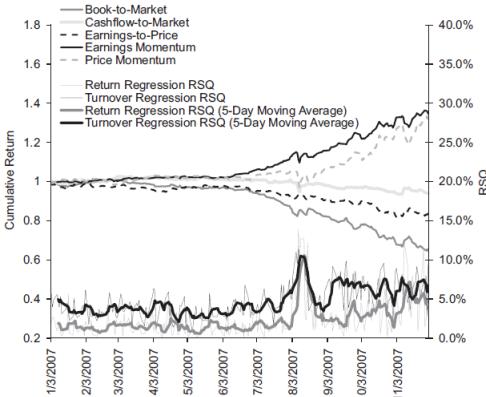


Fig. 2. Cumulative performance and RSQ of cross-sectional regression for quant factors. Cumulative performance of five long/short equity market-neutral portfolios constructed from commonly used equity-valuation factors, from January 3, 2007, to December 31, 2007. Also plotted are daily R^2 's and their 5-day moving averages of the following cross-sectional regressions of returns and turnover: $R_{it} = \alpha_i + \sum_{j=1}^J \beta_{jt} D_{ij} + \varepsilon_{it}$, and $TO_{it} = \gamma_i + \sum_{j=1}^J \delta_{jt} D_{ij} - 5.51 + \eta_{it}$, where R_{it} is the return for security i on day t , D_{ij} is the decile ranking of security i according to factor j , and TO_{it} the turnover for security i on day t , is defined as the ratio of the number of shares traded to the shares outstanding.

Table 1
Performance of contrarian strategy at intraday and daily frequency.

Panel B: Daily Contrarian Strategy

Construction Date

Return (%)

	1 Day	2 Days	3 Days	4 Days	5 Days
8/1/2007	0.14	-1.03	-2.69	-2.57	-0.34
8/2/2007	-0.76	-1.62	-2.57	-2.63	-2.79
8/3/2007	-0.30	-0.57	0.65	0.29	2.04
8/6/2007	-1.47	-1.79	-1.75	1.24	3.44
8/7/2007	-2.88	-4.49	1.38	4.00	4.52
8/8/2007	-3.99	3.79	7.81	8.31	8.20
8/9/2007	6.85	10.12	9.83	9.47	8.96
8/10/2007	-1.46	-1.71	-1.48	-1.84	-1.49
8/13/2007	0.19	0.82	3.79	4.61	3.77
8/14/2007	-0.95	-0.83	0.22	0.34	0.56
8/15/2007	-1.34	-0.58	0.31	0.76	1.69
January to July 2007 Sigma	0.36	0.49	0.59	0.66	0.69

Quantitative equity funds were hit in August 2007 specifically on August 7th–9th with massive reversal on August 10th. Some of the most consistently profitable funds lost too. Moreover, it seemed to only affect quants and it was not related to any real market news.

Kandhani-Lo “Unwind hypothesis:”

- many quant funds use similar factor models
- if one fund has to unwind a large position, that hurts other funds
- this causes these other funds to sell as well...

Sequence of event:

- initial losses “initiated by the rapid unwind of one or more sizeable quantitative equity market-neutral portfolios (likely the result of a force liquidation by a multi-strategy fund or proprietary desk, possibly due to a margin call or a risk reduction)”
- “these initial losses put pressure on a broader set of long/short and long-only equity portfolios, causing further losses by triggering stop/loss and deleveraging policies”
- “on Friday, August 10th, sharp reversals in all five strategies erased nearly all of the losses of the previous four days, returning portfolio values back to their levels on the morning of August 6th. Of course, this assumes that portfolio leverage did not change during this tumultuous week, which is an unlikely assumption given the enormous losses during the first few days.”

The Review of Financial Studies / v 00 n 0 2008

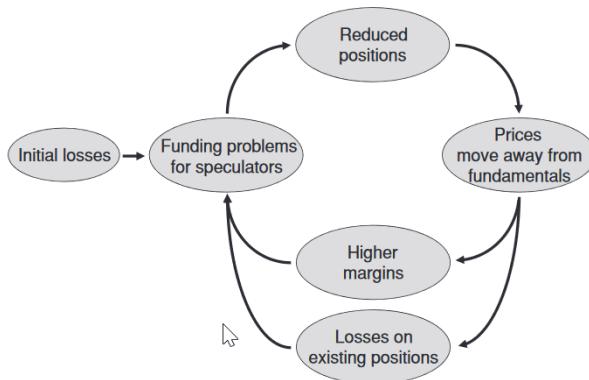


Figure 2
Liquidity spirals
The figure shows the loss spiral and the margin/haircut spiral.

17.2 Sample

```
[8]: from skfin.datasets import load_sklearn_stock_returns
ret = load_sklearn_stock_returns(cache_dir="data")
```

This stock return is found in the example folder on scikit-learn github repository and contains returns from 2003 to 2008. Interestingly, it includes not only US companies, but also European (e.g. Total, SAP, Novartis, etc) and Japanese (Honda, Sony, Toyota) ones. Moreover, some companies are no longer publically traded (e.g. Dell) or have been split (DuPont).

In particular, removing firms that are no longer publicly traded would create a survival bias in the sample. More generally, it is important to make sure that the firms that would have been available for trading are dynamically included in the backtest sample and removed when there are no longer traded (or not liquid enough to be traded given transaction costs).

17.3 Mean-reversion strategy

We load the same functions as the ones we used for the industry backtest.

```
[9]: from skfin.backtesting import Backtester
from skfin.mv_estimators import MeanVariance
```

Rather than setting the positions as $-1 \times$ returns (as Khandani and Lo do), we feed the $-1 \times$ EMA(returns) into the Backtester class to get mean-variance positions.

```
[10]: def xs_score(x, clip=5):
        return (
            x.sub(x.mean(axis=1), axis=0)
            .div(x.std(axis=1), axis=0)
            .clip(lower=-clip, upper=clip)
        )
```

```
[11]: transform_X = lambda x: -1 * x.pipe(xs_score).ewm(halflife=5).mean().fillna(0)
transform_y = lambda x: x.shift(-1).fillna(0)
features = transform_X(ret)
target = transform_y(ret)
```

In contrast to the industry backtests (which were at the Monthly frequency), these mean-reversion backtests are at the (business) day frequency.

```
[12]: ret.equals(ret.resample("B").mean())
```

```
[12]: True
```

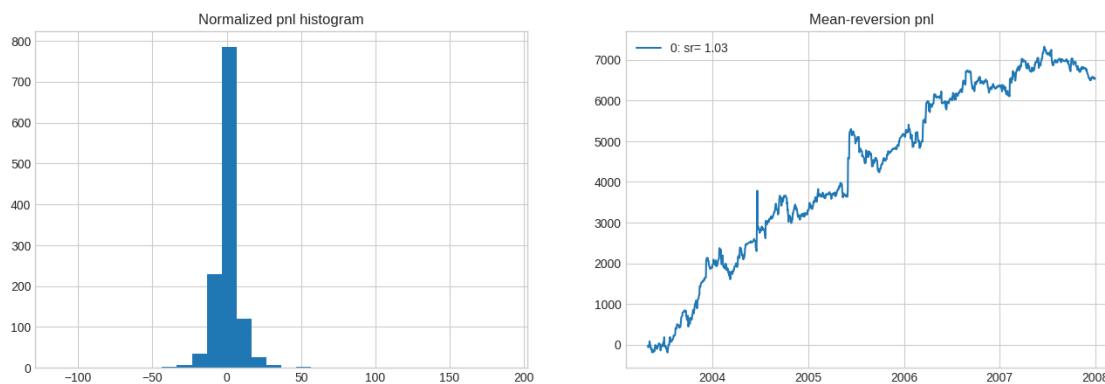
The backtest is setup so that the risk-model is computed over 3-months (=63 business days).

```
[13]: risk_model_window = 63
```

```
[14]: m = (
    Backtester(
        MeanVariance(), max_train_size=risk_model_window, start_date="2003-05-01"
    )
    .compute_holdings(features, target)
    .compute_pnl(ret)
)

hs = {"no-shrinkage": m.h_}
```

```
[15]: fig, ax = plt.subplots(1, 2, figsize=(16, 5))
m.pnl_.div(np.sqrt(risk_model_window)).hist(bins=30, ax=ax[0])
ax[0].set_title("Normalized pnl histogram")
line(
    m.pnl_,
    cumsum=True,
    start_date="2003-05-01",
    title="Mean-reversion pnl",
    ax=ax[1],
    loc="best",
)
```



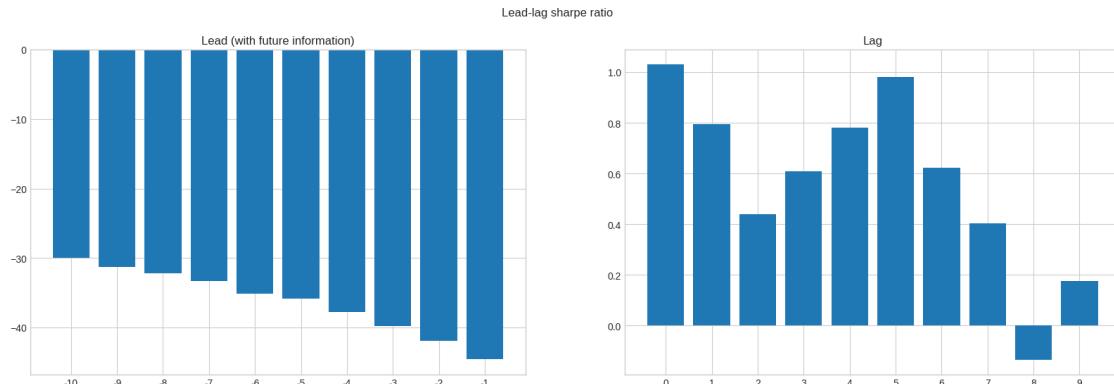
The graph below shows the lead-lag sharpe ratio. The right graph shows the lag: the 0 lag is the tradable lag with a sharpe ratio equal to 1.06 (as in the cumulative pnl graph above). Lagging trading by several business days (up to five) maintain sharpe ratio approximately constant.

The left graph shows the “lead” sharpe ratios – that is, the counterfactual of what the sharpe ratio would have been with future information. Given that the strategy is strong contrarian, this lead

sharpe ratio is very negative.

```
[16]: fig, ax = plt.subplots(1, 2, figsize=(20, 6))
fig.suptitle("Lead-lag sharpe ratio")
bar(
    {
        i: m.h_.shift(1 + i).mul(ret).sum(axis=1).pipe(sharpe_ratio)
        for i in range(-10, 0)
    },
    sort=False,
    title="Lead (with future information)",
    ax=ax[0],
)

bar(
    {
        i: m.h_.shift(1 + i).mul(ret).sum(axis=1).pipe(sharpe_ratio)
        for i in range(0, 10)
    },
    sort=False,
    title="Lag",
    ax=ax[1],
)
```



```
[17]: m_ = Backtester(
    MeanVariance(), max_train_size=risk_model_window, start_date="2003-05-01"
)
pnls_ = []
for clip in [2, 3, 4, 5]:
    transform_X_ = lambda x: -1 * x.pipe(xs_score, clip=clip).ewm(
        halflife=5
    ).mean().fillna(0)
    features_ = transform_X_(ret)
    pnls_[f"clip={clip}"] = m_.train(features_, target, ret)
line(pnls_, cumsum=True, title="Robustness: return clipping")
```



17.4 Mean-reversion with risk-model shrinkage

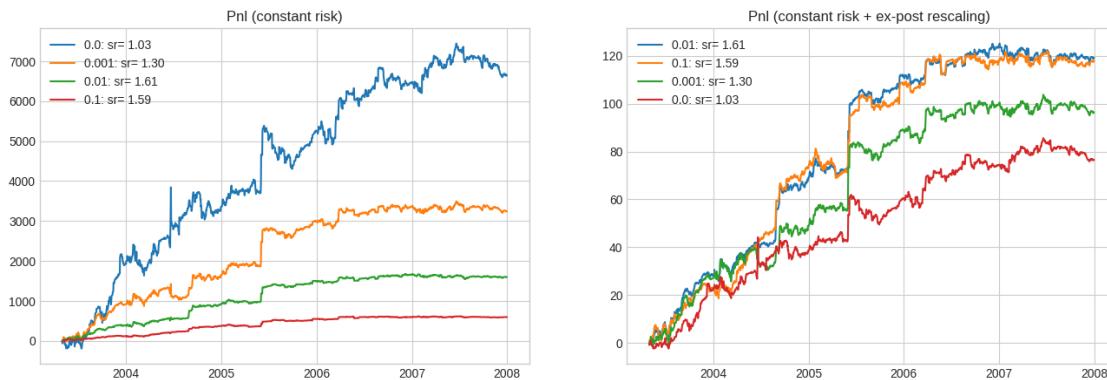
Intuitively, risk-model shrinkage is more necessary and useful with more assets (e.g. many stocks versus few industries). We show how the shrinkage of the covariance matrix helps reducing the over-realisation of risk.

```
[18]: from sklearn.covariance import ShrunkCovariance

pnls_ = {}
for shrinkage in [0, 0.001, 0.01, 0.1]:
    transform_V_ = lambda x: ShrunkCovariance(shrinkage=shrinkage).fit(x).covariance_
    pnls_[shrinkage] = Backtester(
        MeanVariance(transform_V=transform_V_),
        max_train_size=risk_model_window,
        start_date="2003-05-01",
    ).train(features, target, ret)
```

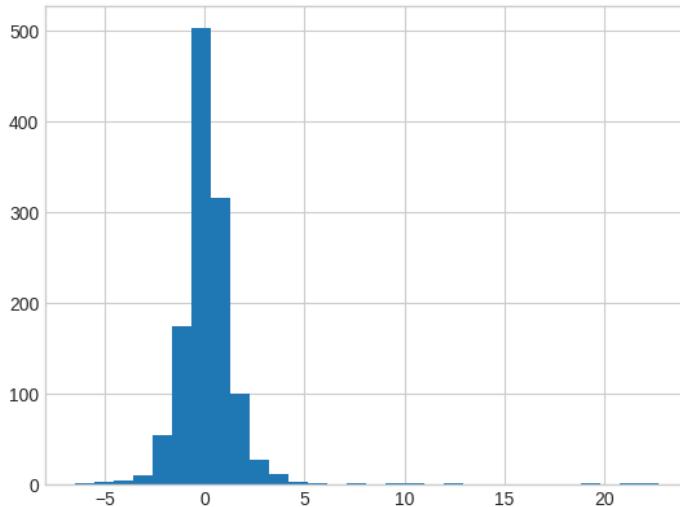
```
[19]: fig, ax = plt.subplots(1, 2, figsize=(16, 5))

line(pnls_, cumsum=True, title="Pnl (constant risk)", loc="best", ax=ax[0])
line(
    pd.concat(pnls_, axis=1).pipe(lambda x: x.div(x.std())),
    cumsum=True,
    title="Pnl (constant risk + ex-post rescaling) ",
    loc="best",
    ax=ax[1],
)
```



We see that with no risk-model shrinkage, the over-realisation of risk is very significant. With some shrinkage, there are less pnl outliers – as shown in the histogram below.

```
[20]: pnls_[0.01].div(np.sqrt(risk_model_window)).hist(bins=30);
```



17.5 Leverage of mean-reversion strategies

Concentrating portfolio positions—in other words, setting smaller positions close to zero—can help increase the risk per unit of leverage. Intuitively, when a portfolio is very diversified across many positions, the realized risk is low – which is great to generate a high Sharpe ratio, but not necessarily good to generate high returns.

In this section, we discuss how we can increase the level of risk per unit of leverage by concentrating the positions.

$$\text{risk_per_unit_of_leverage} = \text{STD} \left[\frac{h_t^T r_t}{\|h_t\|_1} \right]$$

As the ratio of a L2 and L1 norms, the risk-per-unit-of-leverage is related to other concentration measures such as the Herfindahl-Hirschman Index or the Gini coefficient.

```
[21]: from skfin.metrics import sharpe_ratio

leverage_scaling = lambda x: x.div(x.abs().sum(axis=1), axis=0)

def risk_per_unit_of_leverage(h, ret, return_sharpe_ratio=False):
    pnl = h.pipe(leverage_scaling).shift(1).mul(ret).sum(axis=1).loc[h.index]
    if return_sharpe_ratio:
        return {"std": pnl.std() * np.sqrt(252), "sharpe_ratio": pnl.pipe(sharpe_ratio)}
    else:
        return pnl.std() * np.sqrt(252)
```

```
[22]: shrinkage = 0.01
transform_V_ = lambda x: ShrunkCovariance(shrinkage).fit(x).covariance_
m_ = (
    Backtester(
        MeanVariance(transform_V=transform_V_),
        max_train_size=risk_model_window,
        start_date="2003-05-01",
    )
    .compute_holdings(features, target)
    .compute_pnl(ret)
)

hs["shrinkage=0.01"] = m_.h_
```

```
[23]: from sklearn.base import TransformerMixin

def apply_quantile_filter(dataframe, q):
    """
    Apply a quantile-based filtering to a DataFrame.

    This function sets values within each row of the DataFrame to zero if they are not
    in the range defined by the quantiles specified along the rows of the DataFrame. Values that
    are greater than or equal to the upper quantile or less than or equal to the lower quantile
    remain unchanged.

    Parameters:
    - dataframe (pd.DataFrame): The input DataFrame to apply the filter to.
    - q (float): The quantile threshold to use for filtering. It should be a float
      between 0 and 1. Values are retained if they are above `q` or below `1-q`.

    Returns:
    - pd.DataFrame: A DataFrame with values set to zero if they don't meet the quantile
      condition.
    """
    return dataframe.where(
        dataframe.ge(dataframe.quantile(q=q, axis=1), axis=0)
        | dataframe.le(dataframe.quantile(q=1 - q, axis=1), axis=0),
        0,
    )

class Concentrate(TransformerMixin):
    """
    A transformer that filters data based on quantile thresholds across rows.
    """
```

The `Concentrate` transformer is designed to modify a DataFrame by zeroing out values that fall between the specified quantile thresholds for each row. This highlights outliers or concentrations along each row.

Attributes:

- `quantile (float)`: The quantile threshold set during initialization.

Methods:

- `fit(X, y=None)`: Returns the transformer itself. It doesn't learn any parameters.
- `transform(X, return_dataframe=False)`: Transforms the input data based on quantile filtering.

Parameters:

- `q (float)`: The quantile threshold to determine which values to keep. The default is 0.75. If `'q'` is less than or equal to 0.5, it automatically adjusts to `'1-q'`.

Example:

```
>>> transformer = Concentrate(q=0.75)
>>> transformed_data = transformer.transform(dataframe)
>>> transformed_data[:5] # View the top 5 rows
"""

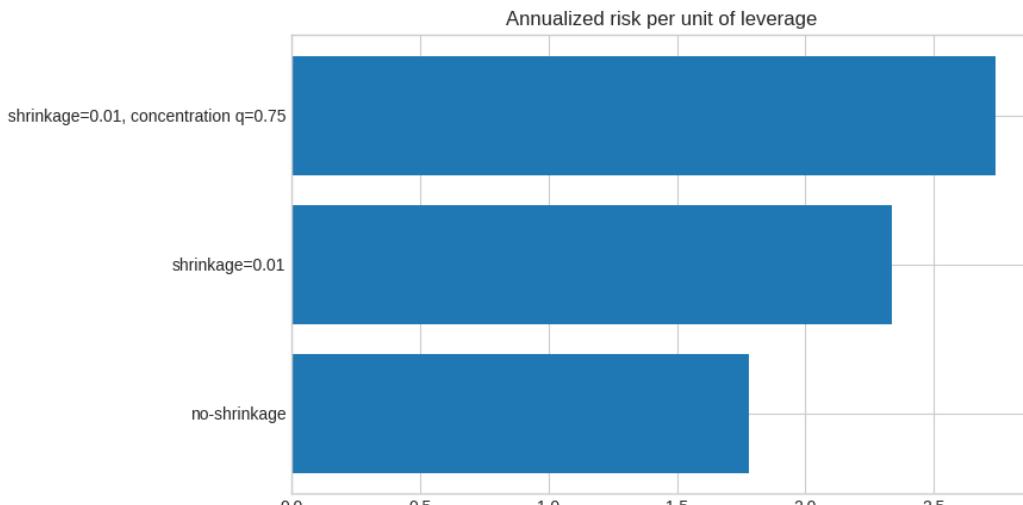
def __init__(self, q=0.75):
    self.quantile = q if q > 0.5 else 1 - q

def fit(self, X, y=None):
    return self

def transform(self, X, return_dataframe=False):
    if not isinstance(X, pd.DataFrame):
        X = pd.DataFrame(X)
    filtered_data = apply_quantile_filter(X, q=self.quantile)
    return filtered_data if return_dataframe else filtered_data.values
```

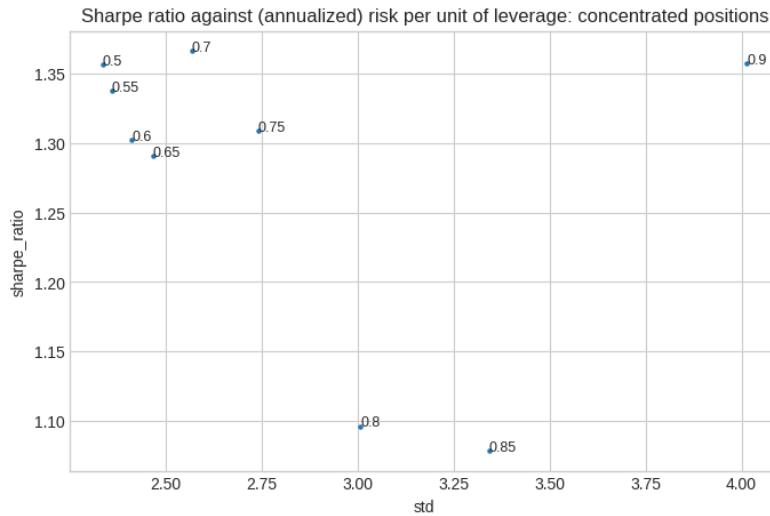
[24]: `q = 0.75
hs[f"shrinkage=0.01, concentration q={q}"] = m_.h_.pipe(apply_quantile_filter, q=q)`

[25]: `bar(
 pd.Series({k: risk_per_unit_of_leverage(v, ret) for k, v in hs.items()}),
 horizontal=True,
 title="Annualized risk per unit of leverage",
)`



```
[26]: df = pd.DataFrame.from_dict(
    {
        q: m_.h_.pipe(apply_quantile_filter, q=q).pipe(
            risk_per_unit_of_leverage, ret=ret, return_sharpe_ratio=True
        )
        for q in np.arange(0.5, 0.95, 0.05)
    },
    orient="index",
)

fig, ax = plt.subplots(1, 1, figsize=(8, 5))
ax.scatter(df.iloc[:, 0], df.iloc[:, 1], s=5, c="tab:blue")
ax.set_xlabel(df.columns[0])
ax.set_ylabel(df.columns[1])
ax.set_title(
    "Sharpe ratio against (annualized) risk per unit of leverage: concentrated positions"
)
for i, txt in enumerate(df.index):
    ax.text(df.iloc[i, 0], df.iloc[i, 1], round(txt, 3), fontsize=9)
```



17.6 Concentrating the predictor instead of the positions

Instead of applying the concentration operator on positions, we can also apply on the predictor. The results are similar.

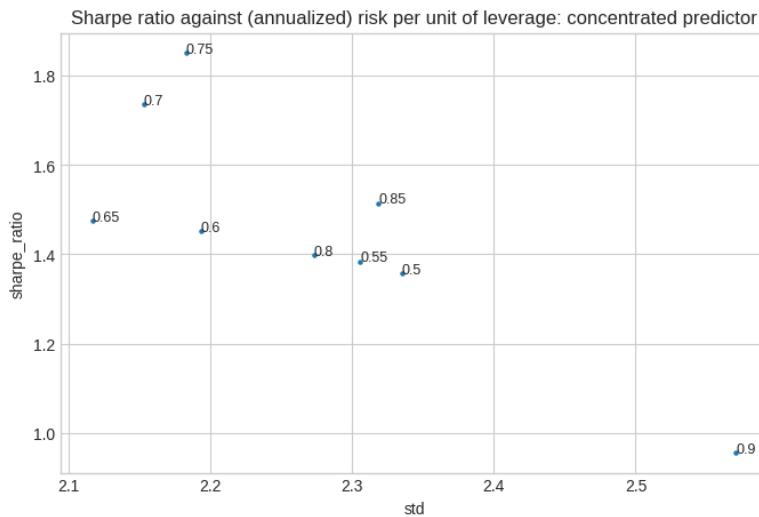
```
[27]: from sklearn.pipeline import make_pipeline

[28]: res = []
for q in np.arange(0.5, 0.95, 0.05):
    estimator = make_pipeline(Concentrate(q=q), MeanVariance(transform_V=transform_V_))
    m2_ = Backtester(
        estimator, max_train_size=risk_model_window, start_date="2003-05-01"
    ).compute_holdings(features, target)
```

```
res[q] = m2_.h_.pipe(risk_per_unit_of_leverage, ret=ret, return_sharpe_ratio=True)
```

```
[29]: df = pd.DataFrame.from_dict(res, orient="index")

fig, ax = plt.subplots(1, 1, figsize=(8, 5))
ax.scatter(df.iloc[:, 0], df.iloc[:, 1], s=5, c="tab:blue")
ax.set_xlabel(df.columns[0])
ax.set_ylabel(df.columns[1])
ax.set_title(
    "Sharpe ratio against (annualized) risk per unit of leverage: concentrated predictor"
)
for i, txt in enumerate(df.index):
    ax.text(df.iloc[i, 0], df.iloc[i, 1], round(txt, 3), fontsize=9)
```



Chapter 18

Trading Around Earnings Announcements: Volatility, Risk, and Regulatory Filings

In this section, we discuss the trading around corporate events, specifically earning announcements. For instance in the US, firms releases their earnings generally on a quarterly frequency and this is a significant piece of information that subsequently gets incorporated into prices.

18.1 Earnings announcement premium



THE JOURNAL OF FINANCE • VOL. LXXI, NO. 1 • FEBRUARY 2016

Earnings Announcements and Systematic Risk

PAVEL SAVOR and MUNGO WILSON*

ABSTRACT

Firms scheduled to report earnings earn an annualized abnormal return of 9.9%. We propose a risk-based explanation for this phenomenon, whereby investors use announcements to revise their expectations for nonannouncing firms, but can only do so imperfectly. Consequently, the covariance between firm-specific and market cash flow news spikes around announcements, making announcers especially risky. Consistent with our hypothesis, announcer returns forecast aggregate earnings. The announcement premium is persistent across stocks, and early (late) announcers earn higher (lower) returns. Nonannouncers' response to announcements is consistent with our model, both over time and across firms. Finally, exposure to announcement risk is priced.

Savor and Wilson (2016) highlight several key points: the earnings announcement premium is 9.9% annually. They also suggest that firms making these announcements are considered “risky,” which necessitates compensation for the associated risk. Furthermore, the study indicates that firm earnings contain information about market cash-flow risk, which has implications for aggregate risk. Our emphasis will be on the first point.

Table I
Summary Statistics with Expected Announcement Dates

This table presents summary statistics for calendar-time announcer and nonannouncer portfolios (Panel A) and event-time announcer returns (Panel B). *Ret Ann* is the weekly excess return (value- or equal-weighted) of a portfolio consisting of all firms announcing in a given week, based on expected announcement dates. *Ret Non* is the weekly excess return (value- or equal-weighted) of a portfolio consisting of all nonannouncing firms. *N(Ann)* is the number of firms in the announcer portfolio, and *N(Non)* is the number of firms in the nonannouncer portfolio. *Excess Ret* is an announcing firm's excess return, and *Ab. Ret* is the same firm's return in excess of the market return. The sample is from 1974 to 2012. Returns are expressed in percentage points.

	Mean	Median	Min	Perc. 10	Perc. 90	Max	t-stat	SD	Skew.	Kurt.
Panel A: Calendar Time (<i>N</i> = 2,035)										
Ret Ann (vw)	0.32	0.40	-17.27	-2.94	3.35	17.54	5.00	2.86	-0.11	4.08
Ret Non (vw)	0.13	0.28	-18.77	-2.42	2.52	16.09	2.45	2.35	-0.41	6.79
Ann - Non (vw)	0.19	0.11	-9.10	-1.41	1.88	12.48	5.27	1.62	0.65	8.23
Ret Ann (ew)	0.35	0.40	-18.43	-2.52	3.06	17.34	6.06	2.60	-0.53	6.13
Ret Non (ew)	0.22	0.34	-19.06	-2.13	2.44	15.81	4.37	2.25	-0.84	8.79
Ann - Non (ew)	0.13	0.11	-4.31	-0.97	1.30	12.48	6.06	0.98	0.15	2.43
<i>N(Ann)</i>	308	176	0	55	747	1,509		305	1.6	2.0
<i>N(Non)</i>	6,305	6,393	4,250	4,730	7,970	9,047		1,114	0.2	-0.5
Panel B: Event Time (<i>N</i> = 626,567)										
Excess Ret	0.26	-0.11	-96.47	-8.50	9.03	473.28	21.73	9.50	3.26	76.60
Ab. Ret	0.15	-0.24	-97.35	-8.26	8.53	473.86	13.14	9.50	3.26	76.60

Table II
Earnings Announcement Premium

This table presents calendar-time abnormal returns for the long-short earnings announcement factor portfolio. Every week all stocks are divided into those that announce earnings and those that do not, based on their expected announcement dates. Portfolio returns equal those of a strategy that buys all announcing stocks and sells short nonannouncing stocks. Alphas are computed using the CAPM, the Fama-French three-factor model, and the Fama-French + momentum model. Returns are expressed in percentage points. *t*-statistics are given in brackets.

	Excess Ret.	Alpha	Mktfrf	SMB	HML	UMD	R ²
Panel A: Value-Weighted Earnings Announcement Premium (%)							
1974–2012	0.19 [5.27]	0.19 [5.18]	0.02 [1.54]				0.12
1974–2012	0.19 [5.27]	0.19 [5.40]	0.01 [0.52]	0.06 [1.93]	-0.10 [-3.27]		0.90
1974–2012	0.19 [5.27]	0.19 [5.19]	0.01 [0.88]	0.06 [1.88]	-0.08 [-2.66]	0.03 [1.82]	1.06
Panel B: Equal-Weighted Earnings Announcement Premium (%)							
1974–2012	0.13 [6.06]	0.12 [5.66]	0.10 [10.61]				5.25
1974–2012	0.13 [6.06]	0.12 [5.49]	0.10 [10.61]	0.00 [0.21]	0.03 [1.55]		5.36
1974–2012	0.13 [6.06]	0.12 [5.54]	0.10 [10.24]	0.00 [0.23]	0.02 [1.29]	-0.01 [-0.77]	5.39
Panel C: Value-Weighted Earnings Announcement Premium (Subsamples) (%)							
1974–1986	0.11 [2.28]	0.10 [2.15]	0.08 [3.30]	0.08 [1.83]	0.06 [1.17]	0.01 [0.17]	1.93
1987–1999	0.26 [4.92]	0.24 [4.44]	0.03 [0.94]	0.13 [2.86]	-0.10 [-1.51]	0.11 [2.43]	3.63
2000–2012	0.20 [2.48]	0.21 [2.59]	-0.02 [-0.47]	0.02 [0.29]	-0.10 [-1.76]	0.02 [0.58]	0.84

18.2 Regulatory filings

The main regulatory filings to the U.S. Securities and Exchange Commission (SEC) are known as the 10-Ks and 10-Qs.

- The 10-K is an annual report required by the U.S. Securities and Exchange Commission (SEC) that gives a summary of the company's financial performance (and includes information such as company history, executive compensation, etc).
- The 10-Q is a quarterly report and contains similar information as the 10-K, but with less details.

The regulatory filings and the earning conference calls take place typically on the same day, so that all the market-moving information is disclosed to the market at the same time.

18.3 Filing dates from 10-Ks/10-Qs

In this section, we use the sample of firms from the daily stock return dataset and match it to the McDonald repository (<https://sraf.nd.edu/>) used in particular in the Loughran-McDonald paper.

```
[6]: from skfin.datasets_ import load_10X_summaries, load_sklearn_stock_returns
from skfin.dataloaders.constants.mappings import mapping_10X
from skfin.plot import bar, line

ret = load_sklearn_stock_returns(cache_dir="data")

[7]: df = load_10X_summaries()

[8]: df.sample(n=5).iloc[:, :12]
```

	CIK	FILING_DATE	ACC_NUM	CPR	FORM_TYPE	\
date						
2021-02-11	720875	20210211	0001654954-21-001483	20201231	10-Q	
2018-05-08	1001039	20180508	0001001039-18-000079	20180331	10-Q	
1998-05-11	1013272	19980511	0000946275-98-000289	19980331	10-Q	
2016-11-03	886744	20161103	0001104659-16-154548	20160930	10-Q	
1998-08-03	717423	19980803	0000717423-98-000008	19980630	10-Q	

	CoName	SIC	FFInd	N_Words	N_Uneque_Words	\
date						
2021-02-11	DYNATRONICS CORP	3841	12	7034	1045	
2018-05-08	WALT DISNEY CO/	4841	32	42614	2731	
1998-05-11	NORWOOD FINANCIAL CORP	6022	44	4326	862	
2016-11-03	GERON CORP	2834	13	65722	3706	
1998-08-03	MURPHY OIL CORP /DE	2911	30	5159	950	

	N_Negative	N_Positive
date		
2021-02-11	86	13
2018-05-08	689	184
1998-05-11	39	23
2016-11-03	1891	657
1998-08-03	52	9

The mapping of stock tickers to company name is `mapping_10X`:

- given that the name of firms can change (e.g. “Dell computer corp” becoming “Dell inc”), all the possible names need to be tracked.

```
[9]: random.choices(list(mapping_10X.items()), k=10)

[9]: [('CAT', 'CATERPILLAR INC'),
('PEP', 'PEPSI BOTTLING GROUP INC'),
('YHOO', 'YAHOO INC'),
('CMCSA', 'COMCAST CORP'),
('COP', 'CONOCOPHILLIPS'),
('DELL', ['DELL COMPUTER CORP', 'DELL INC']),
('AIG', 'AMERICAN INTERNATIONAL GROUP INC'),
('CL', 'COLGATE PALMOLIVE CO'),
('AXP', 'AMERICAN EXPRESS CO'),
('CL', 'COLGATE PALMOLIVE CO')]
```

The table below shows the number of regulatory filings over time for the selected firms.

```
[10]: pd.DataFrame.from_dict(
{}
```

```

k: df.loc[lambda x: x.CoName.isin(v if isinstance(v, list) else [v])]
    .loc[ret.index[0] : ret.index[-1]]
    .groupby(["FORM_TYPE"])["FILING_DATE"]
    .count()
    for k, v in mapping_10X.items():
},
orient="index",
).assign(
**{
    "10_K_Q": lambda x: x["10-K"] + x["10-Q"],
    "restatements": lambda x: x["10-K-A"] + x["10-Q-A"],
}
).sort_values(["10_K_Q", "restatements"]).fillna(0).astype(int)

```

[10]:

	10-K	10-Q	10-K-A	10-Q-A	10_K_Q	restatements
NAV	3	9	0	0	12	0
CVC	6	12	6	2	18	8
BA	5	15	1	1	20	2
PEP	5	15	1	1	20	2
F	5	15	1	3	20	4
GE	5	15	2	3	20	5
WFC	5	15	2	3	20	5
JPM	5	15	5	1	20	6
TWX	5	15	2	4	20	6
AIG	5	15	2	5	20	7
XRX	5	15	3	4	20	7
CAT	5	15	7	2	20	9
AAPL	5	15	0	0	20	0
AMZN	5	15	0	0	20	0
AXP	5	15	0	0	20	0
BAC	5	15	0	0	20	0
CL	5	15	0	0	20	0
CMCSA	5	15	0	0	20	0
COP	5	15	0	1	20	0
CSCO	5	15	0	0	20	0
CVS	5	15	0	0	20	0
CVX	5	15	0	0	20	0
DD	5	15	0	1	20	0
DELL	5	15	0	1	20	0
GD	5	15	1	0	20	0
GS	5	15	0	0	20	0
HD	5	15	0	0	20	0
IBM	5	15	0	1	20	0
K	5	15	0	0	20	0
KMB	5	15	0	0	20	0
KO	5	15	0	0	20	0
MAR	5	15	0	0	20	0
MCD	5	15	0	0	20	0
MMM	5	15	0	0	20	0
MSFT	5	15	0	1	20	0
NOC	5	15	0	0	20	0
PFE	5	15	0	0	20	0
PG	5	15	0	0	20	0
R	5	15	0	0	20	0
RTN	5	15	1	0	20	0
TXN	5	15	0	0	20	0
VLO	5	15	0	2	20	0
WMT	5	15	1	0	20	0
XOM	5	15	0	0	20	0
YHOO	5	15	0	0	20	0
HPQ	6	15	0	0	21	0

18.4 Stock returns on filing dates

```
[11]: ret_norm = ret.pipe(lambda x: x.div(x.ewm(halflife=63, min_periods=21).std())).dropna(
    how="all", axis=0
)

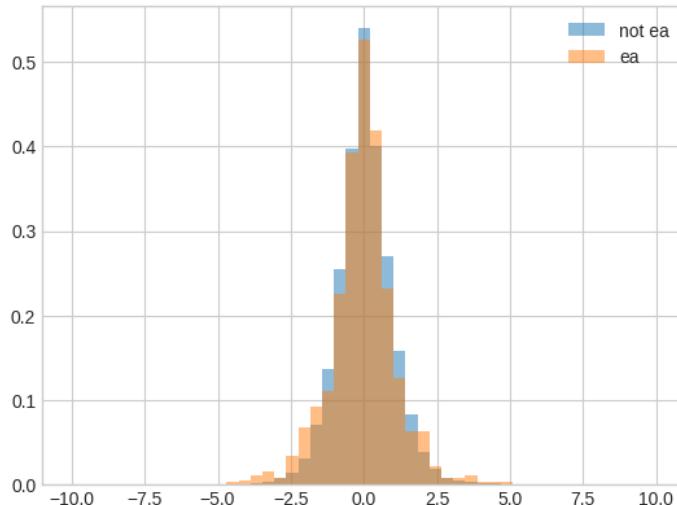
mask = (
    pd.concat(
        {
            k: df.loc[lambda x: x.CoName.isin(v if isinstance(v, list) else [v])].loc[
                "2002-01-01" : ret.index[-1]
            ]["FORM_TYPE"]
            for k, v in mapping_10X.items()
        }
    )
    .groupby(level=[1, 0])
    .count()
)

funcs = {
    "ea": lambda x: x.loc[x.FORM_TYPE.notna()].drop("FORM_TYPE", axis=1),
    "not_ea": lambda x: x.loc[x.FORM_TYPE.isna()].drop("FORM_TYPE", axis=1),
}

ret_ea = pd.concat(
    {
        k: ret_norm.stack().rename("ret").to_frame().join(mask).pipe(v).squeeze()
        for k, v in funcs.items()
    },
    axis=1,
)
)
```

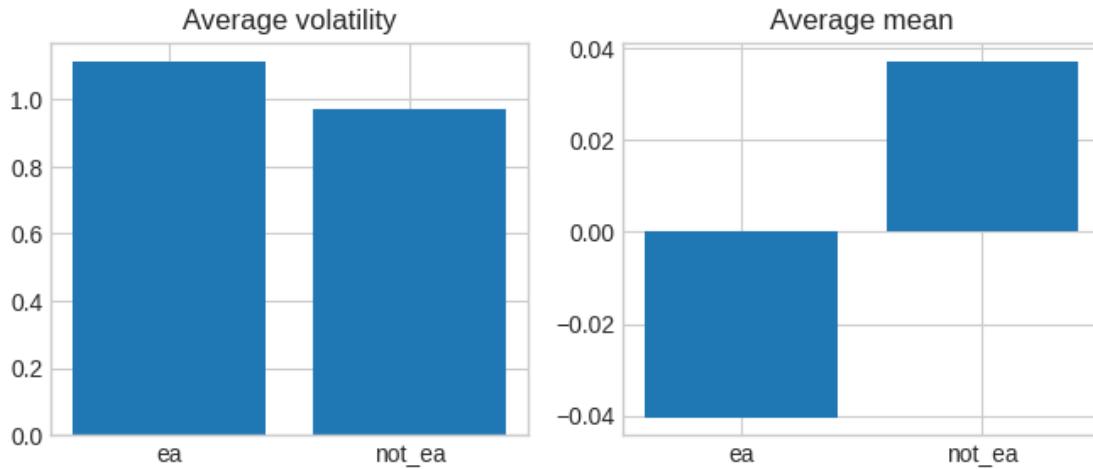
The histogram below shows more extreme returns on filing dates.

```
[12]: bins = np.linspace(-10, 10, 50)
plt.hist(ret_ea["not_ea"].dropna(), bins, density=True, alpha=0.5, label="not ea")
plt.hist(ret_ea["ea"].dropna(), bins, density=True, alpha=0.5, label="ea")
plt.legend(loc="upper right")
plt.show()
```



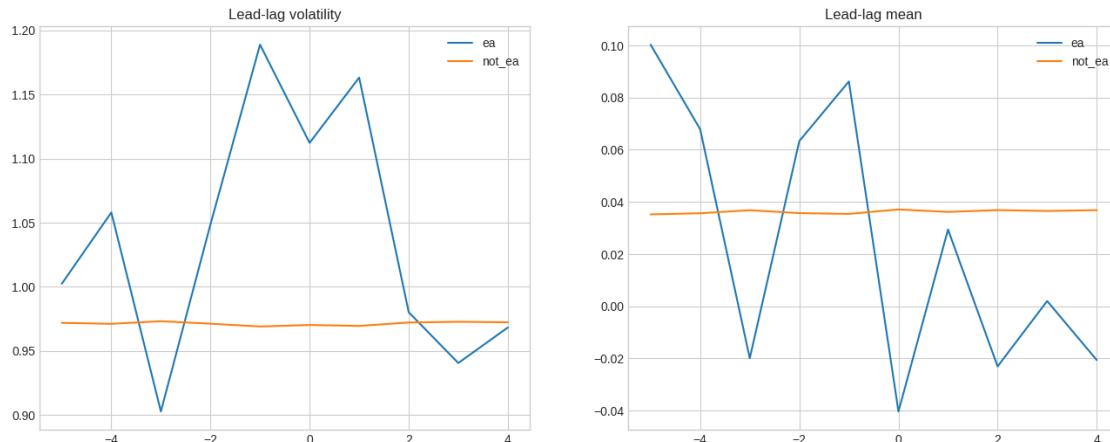
Statistically, this is confirmed with a higher volatility on filing dates. On this sample, the average returns do not seem different.

```
[13]: fig, ax = plt.subplots(1, 2, figsize=(8, 3))
bar(ret_ea.std(), title="Average volatility", ax=ax[0], sort=False)
bar(ret_ea.mean(), title="Average mean", ax=ax[1], sort=False)
```



```
[14]: ea_std, ea_mean = {}, {}
for i in range(-5, 5):
    mask_ = mask.unstack().reindex(ret.index).shift(i).stack().rename("FORM_TYPE")
    ret_ea_ = pd.concat(
        {
            k: ret_norm.stack().rename("ret").to_frame().join(mask_).pipe(v).squeeze()
            for k, v in funcs.items()
        },
        axis=1,
    )
    ea_std[i] = ret_ea_.std()
    ea_mean[i] = ret_ea_.mean()
```

```
[15]: fig, ax = plt.subplots(1, 2, figsize=(16, 6))
line(
    pd.DataFrame(ea_std).T,
    title="Lead-lag volatility",
    sort=False,
    ax=ax[0],
    bbox_to_anchor=None,
    loc="best",
)
line(
    pd.DataFrame(ea_mean).T,
    title="Lead-lag mean",
    sort=False,
    ax=ax[1],
    bbox_to_anchor=None,
    loc="best",
)
```



Chapter 19

Measuring Financial Sentiment in 10-K Filings: Dictionary and Model-Based Approaches

In this section, we describe two strategies to measure sentiment in text. A first strategy is based a dictionary of positive and negative words: simple word counts capture the polarity of the document. A second strategy relies on fitting a linear model with target as returns and features as word counts: in this case, the sign and amplitude of the learned coefficients is used to determined the document sentiment.

Two main papers:

- Loughran and McDonald (2011): “When is a Liability not a Liability? Textual Analysis, Dictionaries and 10-Ks,” *Journal of Finance*
- Jegadeesh and Wu (2013): “Word Power: A New Approach for Content Analysis,” *Journal of Financial Economics*

19.1 Rule-based sentiment

Loughran-McDonalds (2011): textual analysis in finance/accounting to examine the tone and sentiment of corporate 10-K reports. Two statements:

1. a Finance-specific dictionary of negative words matters
2. weighting (e.g. tf.idf weights) matters

Bag of words method: parse the 10-K documents into vectors of words and word counts.

- Dictionaries: http://www3.nd.edu/~mcdonald/Word_Lists.html:
- sentiment negative and positive words
- uncertainty (e.g. approximate, contingency, depend, fluctuate, indefinite, uncertain, and variability)
- litigation (e.g. claimant, deposition, interlocutory, testimony, and tort.)
- modal words are related to levels of confidence: strong modal words (e.g. always, highest, must, and will) and weak modal words (e.g. could, depending, might)

The table below shows the most frequent sentiment words in the full 10-K document in the “Management Discussion and Analysis” subsection.

Table III—Continued

Panel B: Fin-Neg					
Full 10-K Document			MD&A Subsection		
Word in H4N- Inf	Word	% of Total Fin-Neg Word Count	Word in H4N- Inf	Word	% of Total Fin-Neg Word Count
✓	LOSS	9.73%	✓	LOSS	9.51%
✓	LOSSES	5.67%	✓	LOSSES	7.58%
✓	CLAIMS	3.15%	✓	IMPAIRMENT	4.71%
✓	IMPAIRMENT	3.08%	✓	RESTRUCTURING	2.88%
✓	AGAINST	2.50%	✓	DECLINE	2.88%
✓	ADVERSE	2.44%	✓	CLAIMS	2.71%
✓	RESTATED	2.09%	✓	ADVERSE	2.44%
✓	ADVERSELY	1.75%	✓	AGAINST	2.01%
✓	RESTRUCTURING	1.72%	✓	ADVERSELY	1.94%
✓	LITIGATION	1.67%	✓	LITIGATION	1.67%
✓	DISCONTINUED	1.57%	✓	CRITICAL	1.63%
✓	TERMINATION	1.35%	✓	DISCONTINUED	1.62%
✓	DECLINE	1.19%	✓	DECLINED	1.30%
✓	CLAIMING	1.08%	✓	TERMINATION	1.03%
✓	FAILURE	0.92%	✓	NEGATIVE	0.96%
✓	UNABLE	0.84%	✓	FAILURE	0.89%
✓	DAMAGES	0.82%	✓	UNABLE	0.91%
✓	DOUBTFUL	0.77%	✓	CLOSING	0.86%
✓	LIMITATIONS	0.75%	✓	NONPERFORMING	0.81%
✓	FORCE	0.74%	✓	IMPAIRED	0.81%
✓	VOLATILITY	0.73%	✓	VOLATILITY	0.79%
✓	CRITICAL	0.73%	✓	FORCE	0.75%
✓	IMPAIRED	0.70%	✓	NEGATIVELY	0.73%
✓	TERMINATED	0.70%	✓	DOUBTFUL	0.72%
✓	COMPLAINT	0.68%	✓	CLOSED	0.70%
✓	DEFENDANT	0.67%	✓	DIFFICULT	0.69%
✓	NEGATIVE	0.51%	✓	DECLINES	0.63%
✓	DEFENDANTS	0.51%	✓	EXPOSED	0.60%
✓	PLAINTIFFS	0.51%	✓	DEFAULT	0.59%
✓	DIFFICULT	0.50%	✓	DELAYS	0.56%

The following table shows that sentiment words represent about 1% of all words. In these sections, there are more words deemed negative (mean=1.39%) than positive (mean=0.75%).

Full 10-K Document
(N = 50,115)

Variable	Mean	Median	Standard Deviation
<i>Word Lists</i>			
H4N-Inf (H4N w/ inflections)	3.79%	3.84%	0.76%
Fin-Neg (negative)	1.39%	1.36%	0.55%
Fin-Pos (positive)	0.75%	0.74%	0.21%
Fin-Unc (uncertainty)	1.20%	1.20%	0.32%
Fin-Lit (litigious)	1.10%	0.95%	0.53%
MW-Strong (strong modal words)	0.26%	0.24%	0.11%
MW-Weak (weak modal words)	0.43%	0.39%	0.21%

The following table shows the predictability of sentiment for 4-day forward value-weighted excess return (1994-2008). More precisely, the Fin-Neg (negative) sentiment predicts returns with a t-stat from -2.64 to -3.11 after controlling for risk factors.

	Proportional Weights		tf.idf Weights	
	(1)	(2)	(3)	(4)
<i>Word Lists</i>				
H4N-Inf (Harvard-IV-4-Neg with inflections)	-7.422 (-1.35)		-0.003 (-3.16)	
Fin-Neg (negative)		-19.538 (-2.64)		-0.003 (-3.11)
<i>Control Variables</i>				
Log(size)	0.123 (2.87)	0.127 (2.93)	0.131 (2.96)	0.132 (2.97)
Log(book-to-market)	0.279 (3.35)	0.280 (3.45)	0.273 (3.37)	0.277 (3.41)
Log(share turnover)	-0.284 (-2.46)	-0.269 (-2.36)	-0.254 (-2.32)	-0.255 (-2.31)
Pre_FFAAlpha	-2.500 (-0.06)	-3.861 (-0.09)	-5.319 (-0.12)	-6.081 (-0.14)
Institutional ownership	0.278 (0.93)	0.261 (0.86)	0.254 (0.87)	0.255 (0.87)
NASDAQ dummy	0.073 (0.86)	0.073 (0.87)	0.083 (0.97)	0.080 (0.94)
Average R^2	2.44%	2.52%	2.64%	2.63%

The table below shows the same results for several word dictionaries.

Dependent Variable	H4N-Inf	Finance Dictionaries					
		Negative	Positive	Uncertainty	Litigious	Modal Strong	Modal Weak
Panel A: Proportional Weights							
Event period excess return	-7.422 (-1.35)	-19.538 (-2.64)	-21.696 (-1.18)	-42.026 (-4.13)	9.705 (1.17)	-149.658 (-3.82)	-60.230 (-2.43)
Event period abnormal volume (coefficient /100)	2.735 (2.02)	6.453 (3.11)	-1.957 (-0.20)	2.220 (0.48)	0.057 (0.02)	21.430 (1.67)	4.300 (0.74)
Postevent return volatility	11.336 (8.59)	34.337 (12.59)	18.803 (3.47)	33.973 (8.34)	-0.299 (-0.23)	152.312 (12.32)	59.239 (8.58)
Panel B: tf.idf Weights							
Event period excess return	-0.003 (-3.16)	-0.003 (-3.11)	-0.011 (-2.27)	-0.022 (-4.04)	-0.001 (-0.62)	-0.065 (-2.28)	-0.080 (-3.44)
Event period abnormal volume	0.086 (4.30)	0.098 (4.40)	0.159 (1.03)	0.409 (2.50)	0.135 (2.60)	0.046 (0.03)	0.864 (1.21)
Postevent return volatility	0.004 (12.91)	0.004 (11.87)	0.014 (12.52)	0.020 (8.95)	0.006 (10.10)	0.073 (7.47)	0.069 (8.21)

19.2 Learning-based sentiment

Jegadeesh and Wu (2013) discuss how to fit word weights to better identify terms that drive returns.

- The idea is to identify sentiment words associated to significant return moves (either positive or negative) when firm file 10Ks with the SEC.

Learning for document d :

$$r_{d,t \rightarrow t+3} = a + \sum_{v \in LM} b_v \frac{count_{d,v}}{length_v} + e$$

where the terms v are in the base sentiment vocabulary LM from Loughran and McDonalds.

Out-of-sample forecast:

$$Score_d = \sum_v \left(\frac{b_v - \bar{b}}{\sqrt{Var(b_j)}} \right) \frac{count_{d,v}}{length_d}$$

$$r_{d,t+5 \rightarrow t+w} = \alpha + \beta Score_d + \epsilon$$

where the statistical significance of β is evaluated using Fama-MacBeth statistics.

The table below shows the most impactful words from the regressions.

Table 3

Top five most positive and negative words within frequency quintiles.

This table presents the five positive and words with the largest word power weights within each term frequency quintile. Term frequency of each word is the percentage of 10-Ks in which the word appears. Frequency Quintile 1 contains the quintile of words with the lowest frequency and Frequency Quintile 5 contains the quintile of words with the highest frequency. This table reports the word power weights computed using Eqs. (6) and (7) fitted over the sample period of 1995–2010.

Frequency Quintiles				
1	2	3	4	5
<i>Panel A: Top five most positive words</i>				
ingenuity	influential	exceptional	adequately	favorable
acclaimed	optimistic	proficient	highest	strong
revolutionize	enthusiastic	transparency	progress	gain
courteous	excited	versatile	desirable	efficiency
incredible	regain	compliment	encouraged	opportunity
<i>Panel B: Top five most negative words</i>				
imperil	turbulent	disapprove	unplanned	unresolved
disavow	overestimate	reluctant	illegal	unsuccessful
insubordination	underinsured	uncontrollable	wasteful	discourage
bailout	aggravate	setback	misuse	unauthorized
dismal	unfortunate	tumult	strain	insufficient

The table below shows taht the words identified in the regressions are not the same as the one with high tfidf weights.

Table 4

Comparison of word power weights and *idf* term weights.

This table presents the top and bottom ten positive and negative words based on word power weights and their *idf* term weights. This table reports the words with the largest word power weights based on Eqs. (6) and (7) fitted over the sample period of 1995–2010. The *idf* term weighting scheme assigns weights inversely proportional to document frequency, as described in Eq. (1). The document frequency of each word is the percentage of 10-Ks in which the word appears. Panels A and B present the top and bottom positive and negative words, respectively.

	Most impactful words		Least impactful words	
	WP rank	<i>idf</i> rank	WP rank	<i>idf</i> Rank
<i>Panel A: Positive words</i>				
ingenuity	1	14	lucrative	123
acclaimed	2	7	tremendous	122
influential	3	26	worthy	121
revolutionize	4	19	happy	120
optimistic	5	42	spectacular	119
enthusiasm	6	29	beautiful	118
excited	7	48	smooth	117
courteous	8	20	conducive	116
regain	9	39	receptive	115
incredible	10	3	proactive	114
<i>Panel B: Negative words</i>				
imperil	1	18	dispossess	718
disavow	2	22	ridicule	717
insubordination	3	20	mischief	716
bailout	4	31	derogatory	715
dismal	5	10	disorderly	714
untruthful	6	39	disassociate	713
unwelcome	7	5	immoral	712
turbulent	8	140	irreconcilable	711
vitiating	9	38	disgrace	710
undocumented	10	55	extenuating	709

The table shows that the learned sentiment (as the WP or word power score) predicts 4-day forward returns – even after controlling for known risk factors.

	Models			
	Combined LM Lexicon (1)	LM Lexicon (2)	Global Lexicon (3)	Global Lexicon (4)
<i>Term Weighting Scheme</i>				
WP	0.343 (2.67)	0.192 (3.81)	0.294 (2.44)	0.190 (3.58)
<i>Control Variables</i>				
Size		-0.018 (-0.21)		-0.019 (-0.14)
BM		2.631 (1.45)		2.461 (1.00)
Volatility		-0.312 (-1.75)		-0.334 (-1.84)
Turnover		-0.117 (-1.56)		-0.123 (-1.62)
EAD-Ret		0.575 (5.80)		0.546 (6.38)
Accruals		-0.312 (-1.75)		-0.312 (-1.62)

19.3 10-Ks

We use the 10-K/10-Q summary file from the McDonalds data repository to test some insights from the Loughran-McDonalds paper. The sentiment metric is: - sentiment = (#positive - #negative) / #words

```
[10]: from skfin.datasets_ import load_10X_summaries, load_sklearn_stock_returns
from skfin.dataloaders.constants.mappings import mapping_10X
from skfin.metrics import sharpe_ratio

ret = load_sklearn_stock_returns(cache_dir="data")

[11]: df = load_10X_summaries()

[12]: df.columns

[12]: Index(['CIK', 'FILING_DATE', 'ACC_NUM', 'CPR', 'FORM_TYPE', 'CoName', 'SIC',
       'FFInd', 'N_Words', 'N_Uncertainty', 'N_Litigious', 'N_Constraining',
       'N_Negation', 'NonTextDocTypeChars', 'HTMLChars', 'XBRLChars', 'XMLChars',
       'N_Exhibits'],
       dtype='object')

[13]: sentiment_func = lambda x: (x.N_Positive - x.N_Negative) / x.N_Words

sent = (
    pd.concat(
        {
            k: df.loc[lambda x: x.CoName.isin(v if isinstance(v, list) else [v])].loc["2002-01-01": ret.index[-1]].pipe(sentiment_func).for k, v in mapping_10X.items()
        }
    )
    .groupby(level=[0, 1])
    .mean()
```

)

From the sentiment metrics, we follow the steps to construct a predictor:

- take the sentiment difference from one filing to the previous to measure improvement or deterioration (and remove biases)
- forward-fill the value for one-month (=21 business days) to have an active position over a limited window
- remove the cross-sectional mean (and standardise) so that the predictor have long-short risk-managed positions.

```
[14]: pred = (
    sent.sort_index(level=[0, 1])
    .groupby(level=0)
    .transform("diff")
    .unstack(level=0)
    .resample("B")
    .last()
    .ffill(limit=21)
    .pipe(lambda x: x.sub(x.mean(axis=1), axis=0).div(x.std(axis=1), axis=0))
    .reindex(ret.index)
)
```

```
[15]: line(pred.shift(2).mul(ret).sum(axis=1), cumsum=True)
```



Interesting to note that over this period, two firms contribute disproportionately to the pnl – Apple and Goldman Sachs.

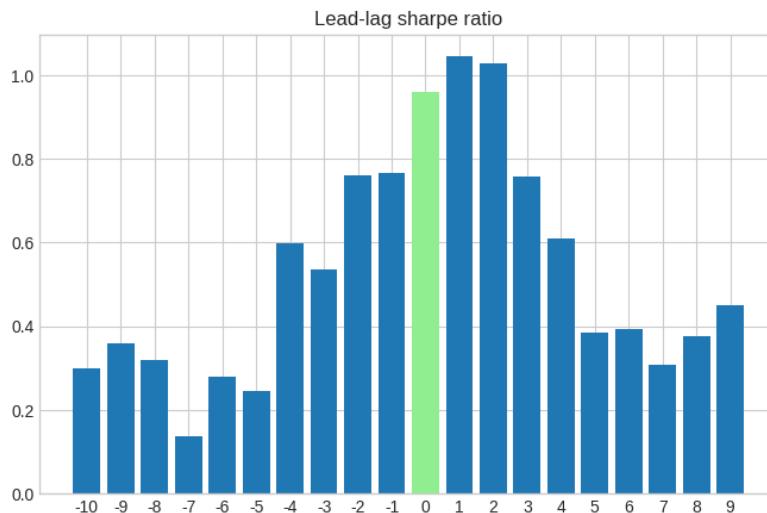
```
[16]: pred.shift(2).mul(ret).dropna(how="all", axis=1).sum().pipe(
    lambda x: pd.concat([x.nlargest(), x.sort_values(ascending=False).tail(5)])
).rename("Stock-level pnl contribution").to_frame()
```

	Stock-level pnl contribution
AAPL	109.887671
GS	97.708109
WFC	27.697014
YHOO	26.637468
DELL	25.705414

TWX	-9.531156
AIG	-10.113330
GD	-21.962501
BAC	-27.134267
MSFT	-35.148272

To assess the alpha decay, the graph below shows the sharpe ratio when the predictor is lagged (or led) by multiple business days.

```
[17]: bar(
    {
        i: pred.shift(2 + i).mul(ret).sum(axis=1).pipe(sharpe_ratio)
        for i in range(-10, 10)
    },
    sort=False,
    baseline=0,
    title="Lead-lag sharpe ratio",
)
```



We can test different sentiment construct as shown in the graph below.

```
[18]: pnls_ = {}
for c in ["N_Litigious", "N_Constraining", "N_Words"]:
    sentiment_func_ = lambda x: (x.N_Positive - x.N_Negative) / x[c]
    sent_ = (
        pd.concat(
            {
                k: df.loc[lambda x: x.CoName.isin(v if isinstance(v, list) else [v])].
                    loc["2002-01-01" : ret.index[-1]].
                    pipe(sentiment_func_)
                    for k, v in mapping_10X.items()
            }
        )
        .groupby(level=[0, 1])
        .mean()
    )

    pred_ = (
        sent_.sort_index(level=[0, 1])
```

```

    .groupby(level=0)
    .transform("diff")
    .unstack(level=0)
    .resample("B")
    .last()
    .ffill(limit=21)
    .pipe(lambda x: x.sub(x.mean(axis=1), axis=0).div(x.std(axis=1), axis=0))
    .reindex(ret.index)
)

pnls_[f"sent/{c}"] = pred_.shift(2).mul(ret).sum(axis=1)

for c in ["N_Negative", "N_Negation", "N_WeakModal"]:
    sentiment_func_ = lambda x: -1 * x[c] / x.N_Words
    sent_ = (
        pd.concat(
            {
                k: df.loc[lambda x: x.CoName.isin(v if isinstance(v, list) else [v])]
                .loc["2002-01-01" : ret.index[-1]]
                .pipe(sentiment_func_)
                for k, v in mapping_10X.items()
            }
        )
        .groupby(level=[0, 1])
        .mean()
    )

    pred_ = (
        sent_.sort_index(level=[0, 1])
        .groupby(level=0)
        .transform("diff")
        .unstack(level=0)
        .resample("B")
        .last()
        .ffill(limit=21)
        .pipe(lambda x: x.sub(x.mean(axis=1), axis=0).div(x.std(axis=1), axis=0))
        .reindex(ret.index)
    )

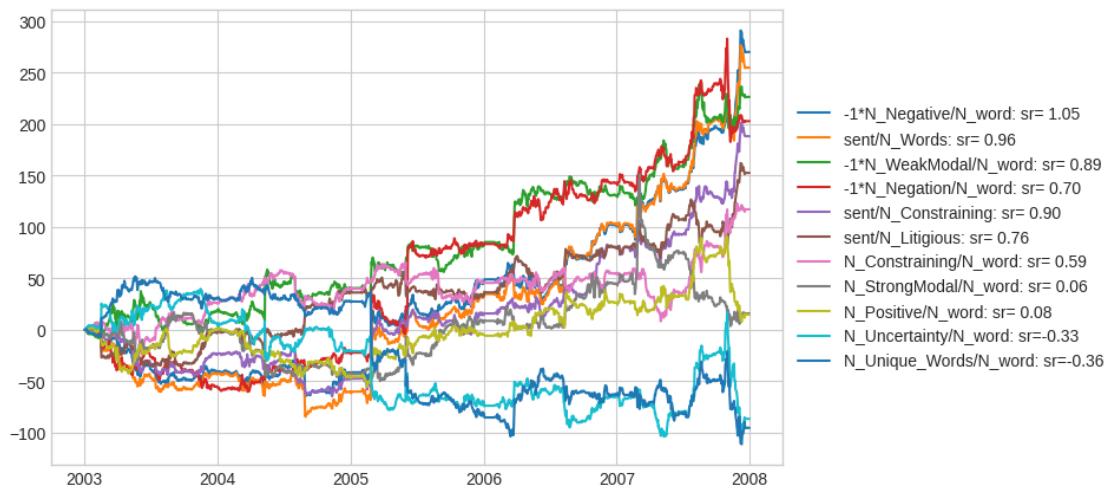
    pnls_[f"-1*{c}/N_word"] = pred_.shift(2).mul(ret).sum(axis=1)

for c in [
    "N_Unique_Words",
    "N_Positive",
    "N_Uncertainty",
    "N_StrongModal",
    "N_Constraining",
]:
    sentiment_func_ = lambda x: x[c] / x.N_Words
    sent_ = (
        pd.concat(
            {
                k: df.loc[lambda x: x.CoName.isin(v if isinstance(v, list) else [v])]
                .loc["2002-01-01" : ret.index[-1]]
                .pipe(sentiment_func_)
                for k, v in mapping_10X.items()
            }
        )
        .groupby(level=[0, 1])
    )

```

Chapter 19. Measuring Financial Sentiment in 10-K Filings: Dictionary and Model-Based Approaches

```
.mean()  
)  
  
pred_ = (  
    sent_.sort_index(level=[0, 1])  
    .groupby(level=0)  
    .transform("diff")  
    .unstack(level=0)  
    .resample("B")  
    .last()  
    .ffill(limit=21)  
    .pipe(lambda x: x.sub(x.mean(axis=1), axis=0).div(x.std(axis=1), axis=0))  
    .reindex(ret.index)  
)  
  
pnls_[f"{{c}}/{N_word}"] = pred_.shift(2).mul(ret).sum(axis=1)  
line(pnls_, cumsum=True)
```



Chapter 20

Out-of-Sample Predictability of the S&P 500: A Critical Assessment

In previous sections, we studied the predictability of industry and stock returns in a long-short “cash-neutral” setting. In this section, we shift to the predictability of a single asset (ie. the “market” as the S&P 500 US index).

20.1 Timing the market

A Comprehensive Look at The Empirical Performance of Equity Premium Prediction

Ivo Welch

Brown University Department of Economics NBER

Amit Goyal

Emory University Goizueta Business School

Our article comprehensively reexamines the performance of variables that have been suggested by the academic literature to be good predictors of the equity premium. We find that by and large, these models have predicted poorly both in-sample (IS) and out-of-sample (OOS) for 30 years now; these models seem unstable, as diagnosed by their out-of-sample predictions and other statistics; and these models would not have helped an investor with access only to available information to profitably time the market. (*JEL* G12, G14)

To evaluate the out-of-sample predictability of a variable, Welch-Goyal (2008) compare two regressions:

- conditional regression (based on the predictor)
- unconditional regression (based on a rolling mean)

The comparison between the two regression provides a test of whether the predictor has any value.

The main intuitions for why the predictability of some variables for the S&P 500 are related to

valuations:

- “low” prices relative to dividends forecast higher subsequent returns;
- other ratios (earnings, book value, moving average of past prices instead of dividends) should also work;
- expected returns vary over the business cycle and higher risk premium required to get people to hold stocks at the bottom of a recession: dividend-price ratios can be interpreted a state-variable capturing business cycle risk.

The main critical question that Welch-Goyal (2008) ask is whether in-sample results also hold out-of-sample.

Data for the following graphs:

- dividend price ratio (“d/p”): difference between the log of dividends and the log of prices
- dividend yield (“d/y”): difference between the log of dividends and the log of lagged prices
- percent equity issuing (“equis”): ratio of equity issuing activity as a fraction of total issuing equity

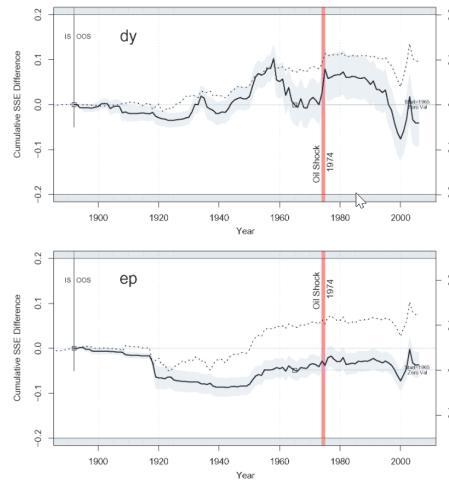


Figure 1
Annual performance of IS insignificant predictors.

Table 1

Forecasts at annual frequency

This table presents statistics on forecast errors in-sample (IS) and out-of-sample (OOS) for log equity premium forecasts at annual frequency (both in the forecasting equation and forecast). Variables are explained in Section 2. Stock returns are price changes, including dividends, of the S&P500. All numbers are in percent per year, except except \bar{R}^2 and power which are simple percentages. A star next to $IS-\bar{R}^2$ denotes significance of the in-sample regression as measured by F -statistics (critical values of which are obtained empirically from bootstrapped distributions). The column ‘IS for OOS’ gives the $IS-\bar{R}^2$ for the OOS period. ARMSE is the RMSE (root mean square error) difference between the unconditional forecast and the conditional forecast for the same sample/forecast period. Positive numbers signify superior out-of-sample conditional forecast. The $OOS-\bar{R}^2$ is defined in Equation 6. A star next to $OOS-\bar{R}^2$ is based on significance of MSE-F statistic by McCracken (2004), which tests for equal MSE of the unconditional forecast and the conditional forecast. One-sided critical values of MSE statistics are obtained empirically from bootstrapping distributions, except for caya and all models where they are obtained from McCracken (2004). Critical values for the ms model are not calculated. Power is calculated as the fraction of draws where the simulated ARMSE is greater than the empirically calculated 95% critical value. The two numbers under the power column are for all simulations and for those simulations in which the in-sample estimate was significant at the 95% level. Significance levels at 90%, 95%, and 99% are denoted by one, two, and three stars, respectively.

Variable	Data	Full Sample										1927–2005	
		Forecasts begin 20 years after sample					Forecasts begin 1965					IS	\bar{R}^2
		IS	IS for	OOS		IS for	OOS	OOS	\bar{R}^2	ΔR^2	Power		
		\bar{R}^2	$OOS\bar{R}^2$	\bar{R}^2	$\Delta RMSE$	Power	$OOS\bar{R}^2$	\bar{R}^2	$\Delta RMSE$	Power	IS	\bar{R}^2	
Full Sample, Not Significant IS													
dfy	Default yield spread	1919–2005	-1.18	-3.29	-0.14		-4.15	-0.12			-1.31		
infl	Inflation	1919–2005	-1.00	-4.07	-0.20		-3.56	-0.08			-0.99		
svar	Stock variance	1885–2005	-0.76	-27.14	-2.33		-2.44	+0.01			-1.32		
d/e	Dividend payout ratio	1872–2005	-0.75	-4.33	-0.31		-4.99	-0.18			-1.24		
lty	Long term yield	1919–2005	-0.63	-7.72	-0.47		-12.57	-0.76			-0.94		
tms	Term spread	1920–2005	0.16	-2.42	-0.07		-2.96	-0.03			0.89		
tbl	Treasury-bill rate	1920–2005	0.34	-3.37	-0.14		-4.90	-0.18			0.15		
dfr	Default return spread	1926–2005	0.40	-2.16	-0.03		-2.82	-0.02			0.32		
d/p	Dividend price ratio	1872–2005	0.49	-2.06	-0.11		-3.69	-0.09			1.67		
dy	Dividend yield	1872–2005	0.91	-1.93	-0.10		-6.68	-0.31			2.71*		
ltr	Long term return	1926–2005	0.99	-11.79	-0.76		-18.38	-1.18			0.92		
e/p	Earning price ratio	1872–2005	1.08	-1.78	-0.08		-1.10	0.11			3.20*		

Welch-Goyal summary: very little predictability and the oil shock 1974 important in explaining results in the literature.

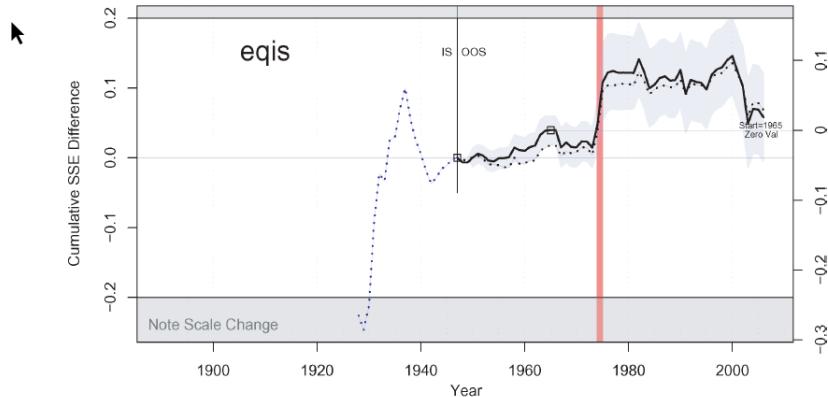


Figure 2
Annual performance of predictors that are not in-sample significant
Explanation: See Figure 1.

Responding to Welch-Goyal (2008), Campbell-Thompson (2008): impose “sign-restrictions”

“in practice, an investor would not use a perverse coefficient but would likely conclude that the coefficient is zero, in effect imposing prior knowledge on the output of the regression” (p. 1516)

Sign restrictions

- set the regression coefficient to zero whenever it has the “wrong” sign (different from the theoretically expected sign estimated over the sample)
- set the forecast equity premium to zero whenever it is negative

Summary: does dividend yield predict returns?

- Yes: dividend yield is a strong predictor in the 1970s and 1980s (in-sample!)
- No: the relationship became weaker in 1990s
- No: the statistical evidence is much weaker when adjusting for fact that regressors highly persistent
- No: dividend yield is also weak predictor out-of-sample –and rarely better than a moving-average.

20.2 Data

The data provided by Amit Goyal on the S&P 500 is essentially identical to the one provided by Ken French.

```
[6]: from skfin.datasets import load_ag_features, load_kf_returns
df = load_ag_features()[:"1999"]
```

ReadMe
Annual

Monthly
Quarterly

```
[7]: ret = load_kf_returns(filename="F-F_Research_Data_Factors")["Monthly"][:"1999"]

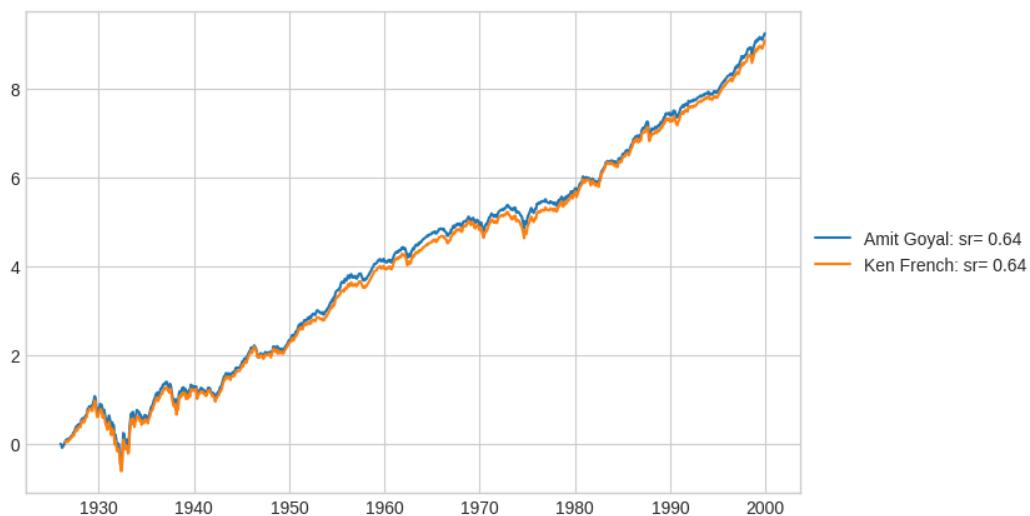
[8]: corr_ = df[[["ret"]]].corrwith(ret.assign(Mkt=lambda x: x["Mkt-RF"] + x["RF"])["Mkt"])
      "ret"
]
print(f"Correlation data Ken French/Amit Goyal:{corr_.:.2f}")

Correlation data Ken French/Amit Goyal:0.99
```

```
[9]: df["ret"].std() * np.sqrt(12)

[9]: 0.19677293731410958
```

```
[10]: line(
      {
        "Amit Goyal": df["ret"],
        "Ken French": ret.assign(Mkt=lambda x: x["Mkt-RF"] + x["RF"])["Mkt"] / 100,
      },
      cumsum=True,
)
```



20.3 Timing backtest

```
[11]: from skfin.estimators import Ridge, RidgeCV
from skfin.mv_estimators import TimingMeanVariance
from sklearn.model_selection import TimeSeriesSplit
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

[12]: start_date = "1945-01-01"
test_size = 1
params = dict(max_train_size=36, test_size=test_size, gap=0)
params["n_splits"] = 1 + len(ret[:"1999"].loc[start_date:]) // test_size
```

```

cv = TimeSeriesSplit(**params)

[13]: cols = [
    "d12",
    "e12",
    "b/m",
    "tbl",
    "AAA",
    "BAA",
    "lty",
    "ntis",
    "Rfree",
    "infl",
    "ltr",
    "corpr",
    "svar",
    "csp",
]
ret_ = ret["Mkt-RF"]
target = ret_
features = df.loc[ret_.index, cols].fillna(0)

```

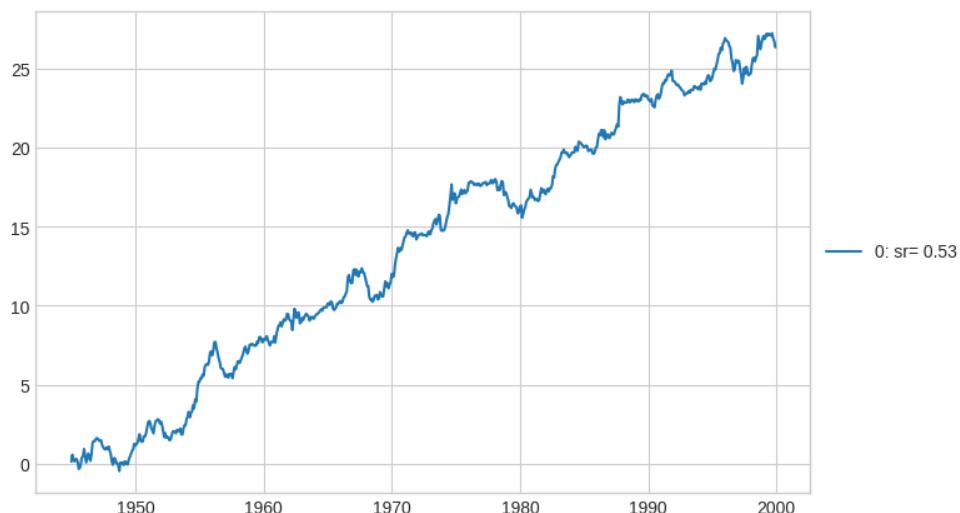
```

[14]: m = make_pipeline(
    StandardScaler(), Ridge(), TimingMeanVariance(a_min=-0.25, a_max=0.25)
)

_h = []
for train, test in cv.split(ret):
    m.fit(features.iloc[train], target.iloc[train])
    _h += [m.predict(features.iloc[test])]

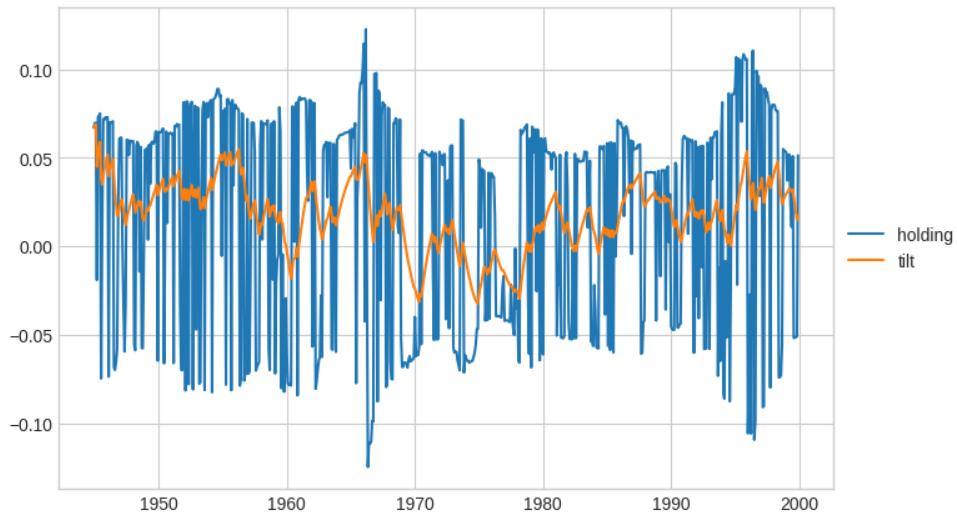
idx = ret.index[np.concatenate([test for _, test in cv.split(ret)])]
h = pd.Series(np.concatenate(_h), index=idx)
pnl = h.shift(1).mul(ret_).dropna()
line(pnl, cumsum=True)

```



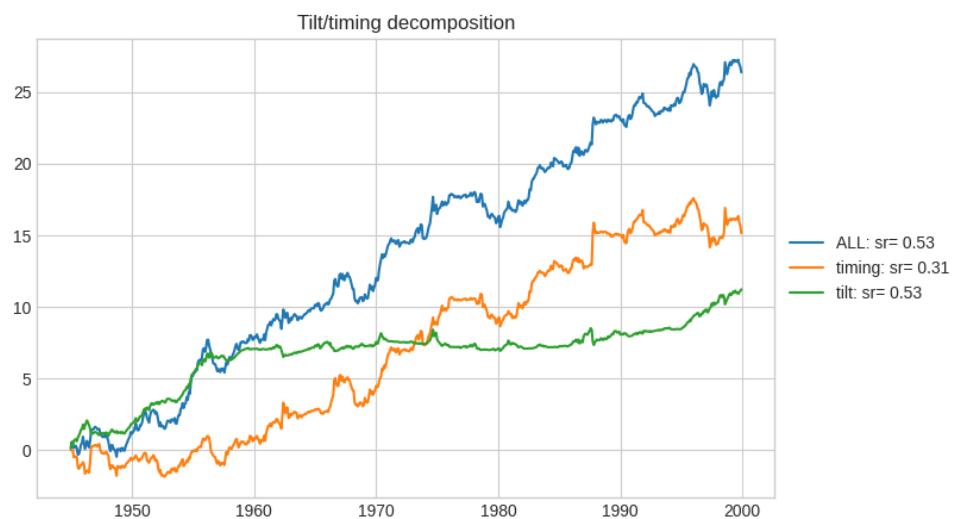
We can plot the holdings and in this case, we see that the positions vary significantly and that there is a significant positive tilt (defined as the exponential average over the positions with a 252-day halflife).

```
[15]: line({"holding": h, "tilt": h.ewm(halflife=12).mean()})
```



Decomposing the pnl attributed to the `tilt` and the `timing` (defined as the difference between the positions and the `tilt`), we see that both contribute – although the `timing` pnl has a lower sharpe ratio.

```
[16]: line(
    {
        "ALL": pnl,
        "tilt": h.ewm(halflife=12).mean().shift(1).mul(ret_).dropna(),
        "timing": h.sub(h.ewm(halflife=12).mean()).shift(1).mul(ret_).dropna(),
    },
    cumsum=True,
    title="Tilt/timing decomposition",
)
```



In what follows, we use the `Backtester` clas with the timing pipeline.

```
[17]: from skfin.backtesting import Backtester

estimator = make_pipeline(
    StandardScaler(), Ridge(), TimingMeanVariance(a_min=-0.25, a_max=0.25)
)

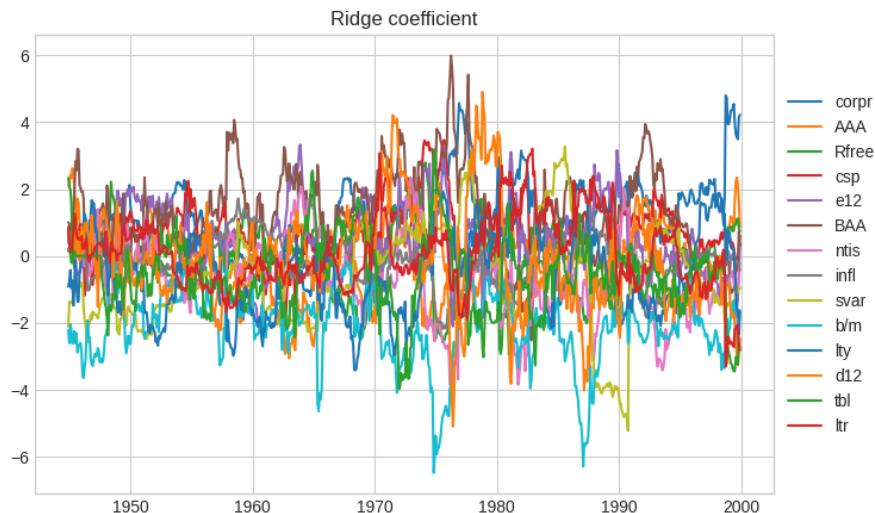
m = Backtester(estimator=estimator)
m.compute_holdings(features, target).compute_pnl(ret_)

np.allclose(h, m.h_), np.allclose(pnl, m.pnl_)
```

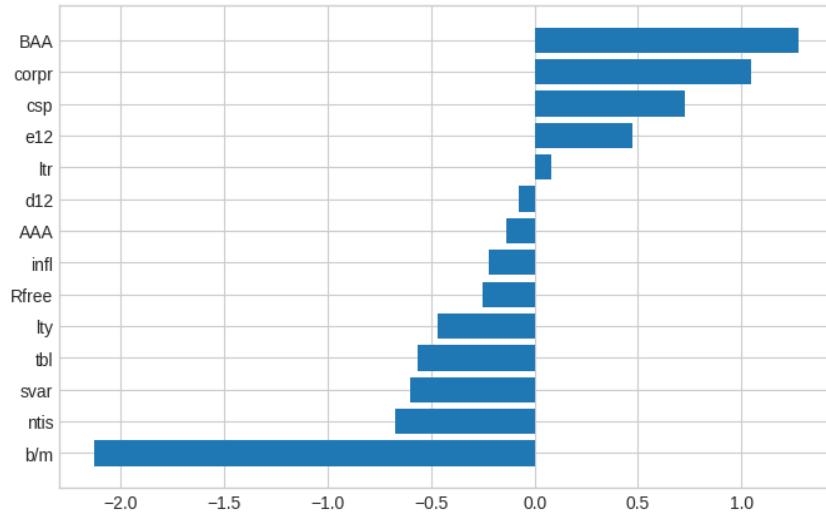
[17]: (True, True)

20.4 Other timing backtest statistics

```
[18]: coef = pd.DataFrame(
    [m_.steps[1][1].coef_ for m_ in m.estimators_], columns=cols, index=m.h_.index
)
line(coef, title="Ridge coefficient")
```

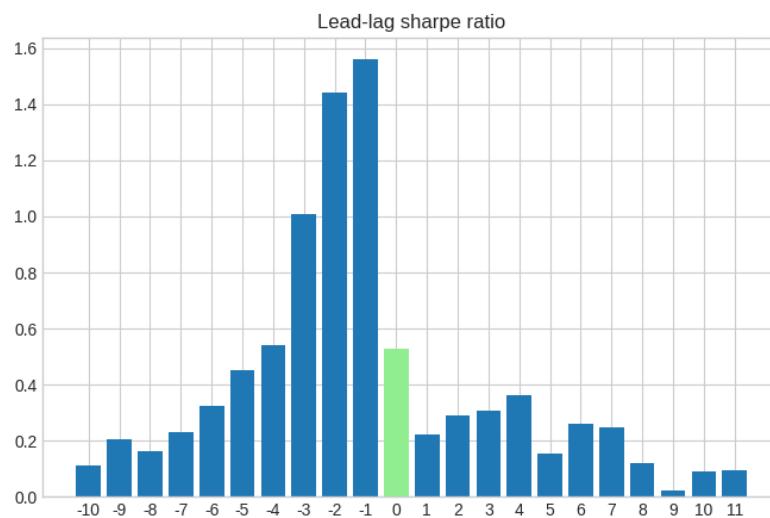


```
[19]: bar(coef.mean(), horizontal=True)
```

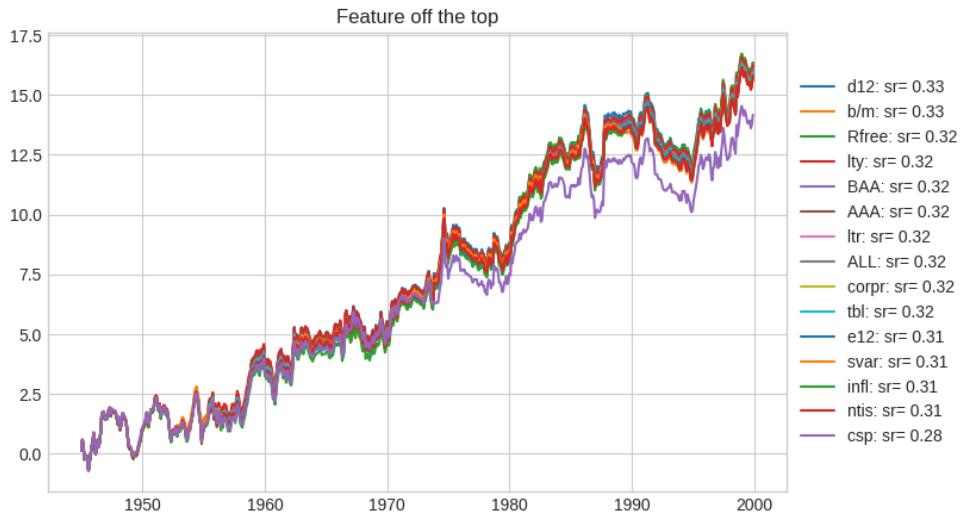


```
[20]: from skfin.metrics import sharpe_ratio
```

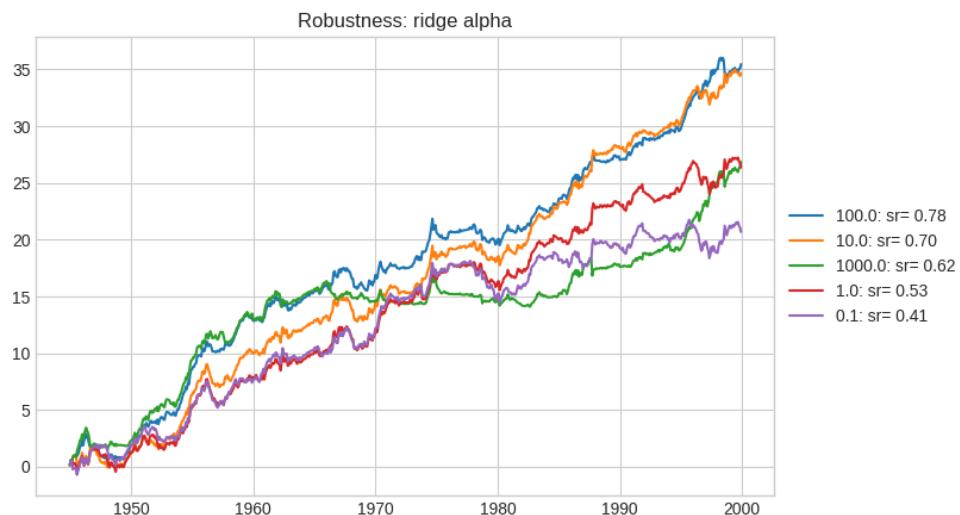
```
[21]: sr = {i: m.h_.shift(1 + i).mul(ret_).pipe(sharpe_ratio) for i in range(-10, 12)}
bar(sr, baseline=0, sort=False, title="Lead-lag sharpe ratio")
```



```
[22]: pnls_ = {}
for c in cols + ["ALL"]:
    features_ = df.loc[ret.index].drop(c, axis=1, errors="ignore").fillna(0)
    pnls_[c] = Backtester(estimator=estimator).train(features_, target, ret=ret_)
line(pnls_, cumsum=True, title="Feature off the top")
```

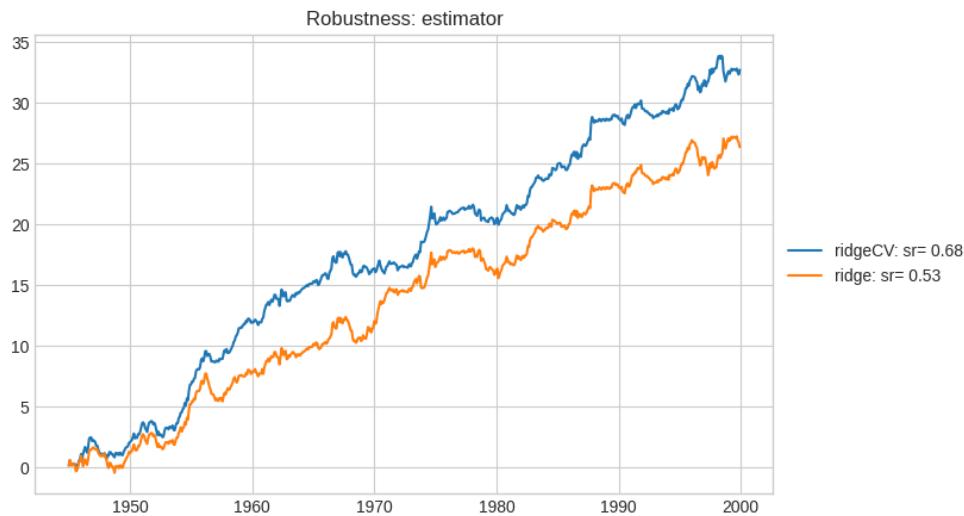


```
[23]: pnls_ = {}
for alpha in [0.1, 1, 10, 100, 1000]:
    estimator_ = make_pipeline(
        StandardScaler(),
        Ridge(alpha=alpha),
        TimingMeanVariance(a_min=-0.25, a_max=0.25),
    )
    pnls_[alpha] = Backtester(estimator=estimator_).train(features, target, ret=ret_)
line(pnls_, cumsum=True, title="Robustness: ridge alpha")
```



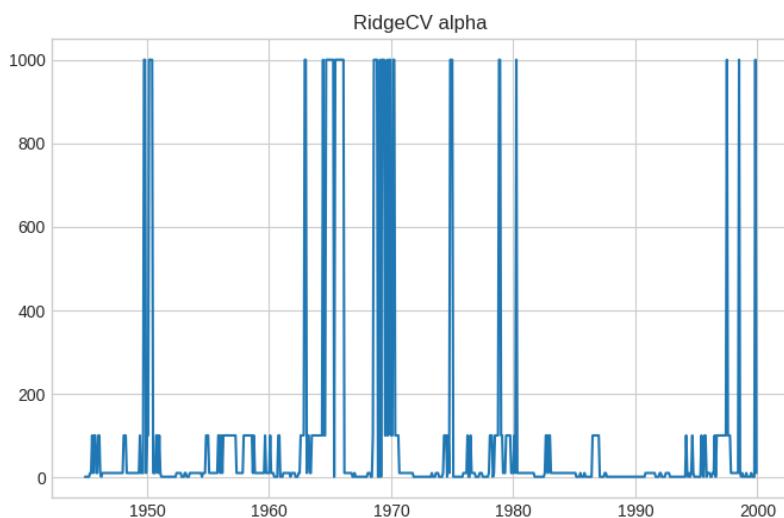
```
[24]: estimator_ = make_pipeline(
    StandardScaler(),
    RidgeCV(alphas=[1, 10, 100, 1000]),
    TimingMeanVariance(a_min=-0.25, a_max=0.25),
)
m_ = Backtester(estimator=estimator_)
```

```
m_.compute_holdings(features, target).compute_pnl(ret_)
line({"ridge": m_.pnl_, "ridgeCV": m_.pnl_}, cumsum=True, title="Robustness: estimator")
```



The following graph shows the regularization parameter alpha estimated by cross-validation by the RidgeCV estimator.

```
[25]: alpha = pd.Series([m_.steps[1][1].alpha_ for m_ in m_.estimators_], index=m_.h_.index)
line(alpha, legend=False, title="RidgeCV alpha")
```



Chapter 21

Politics and the stock market

In this section, we explore the relationship between political control in the United States government and stock market performance, focusing on the phenomenon known as the “Government Puzzle.” We first discuss the “Presidential Puzzle”—the observation that U.S. stock market excess returns have historically been higher under Democratic presidents.

We then examine whether the alignment or division of political power across the executive and legislative branches (i.e., United vs. Divided government) has an even greater impact on financial markets and economic growth.

21.1 The Presidential puzzle and the division of the government

THE JOURNAL OF FINANCE • VOL. LVIII, NO. 5 • OCTOBER 2003

The Presidential Puzzle: Political Cycles and the Stock Market

PEDRO SANTA-CLARA and ROSSEN VALKANOV*

ABSTRACT

The excess return in the stock market is higher under Democratic than Republican presidencies: 9 percent for the value-weighted and 16 percent for the equal-weighted portfolio. The difference comes from higher real stock returns and lower real interest rates, is statistically significant, and is robust in subsamples. The difference in returns is not explained by business-cycle variables related to expected returns, and is not concentrated around election dates. There is no difference in the riskiness of the stock market across presidencies that could justify a risk premium. The difference in returns through the political cycle is therefore a puzzle.

Divided Government and the Stock Market*

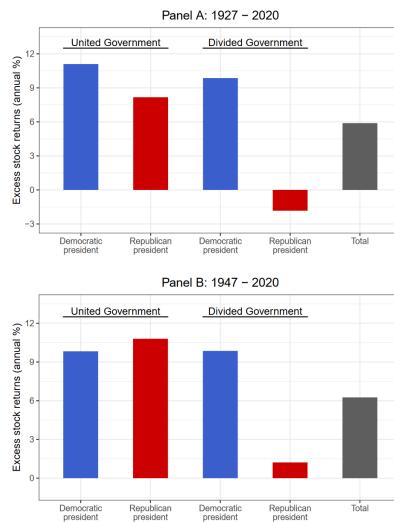
Theofanis Papamichalis[†] Dean Ryu[‡] Mungo Wilson[§]

August 2024

Abstract

We show that during United governments, where the same political party controls the White House, the Senate, and the House of Representatives, the U.S. stock market earns substantially higher equity premia and the U.S. economy experiences higher economic growth than during Divided governments. Our government cycle sheds light on many asset pricing patterns that are not supported by the presidential cycle alone (Santa-Clara and Valkanov, Journal of Finance 2003), such as treasury premium gap and September effect. Consistent with the political gridlock mechanism, the government cycle has a more pronounced impact on small firms than on large firms, shown by a substantial 7% annual disparity in SMB that can account for the recent vanishing of the size effect. Our findings are causal, as demonstrated by closely contested election results, and are in line with theoretical models that link lower returns during Divided-Republican governments to increasing political uncertainty and heightened sentiment.

The figure below from Papamichalis, Ryu and Wilson (2024) shows a two-way sort of average annual excess stock returns by both the president's party and whether the government is united or divided, using post-war data (1927–2020). The main takeaway is that Republican and Democratic presidents deliver strong stock market returns when they have unified control of government, while Republican presidents under divided government are associated with particularly weak market performance—demonstrating that the government cycle, not just presidential party, is key to explaining stock market outcomes.



21.2 Politics over century of data

```
[5]: from skfin.dataloaders.constants.dates import load_us_politics_dates
end_date = "2022-10-01" # Midterm election during Biden's term
```

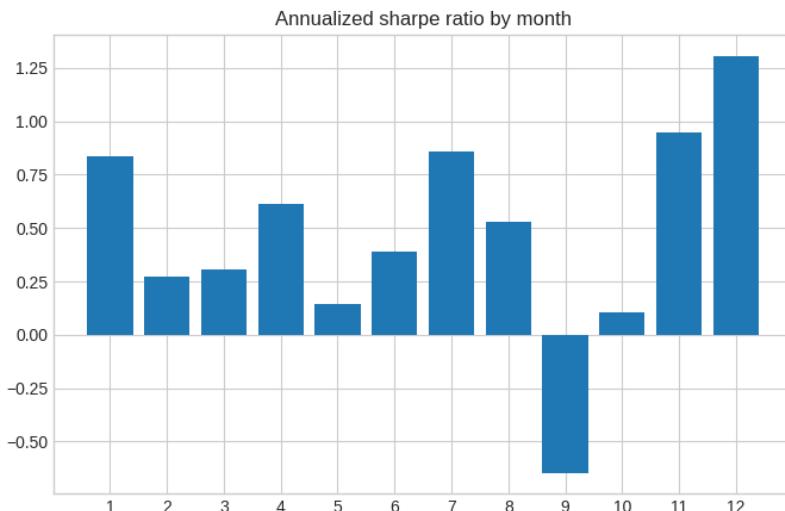
```

cols = ["Presidency", "House", "Senate"]
cols_ = ret.columns
government_ffill_shift = (
    lambda x, cols: x[cols]
    .shift(1)
    .ffill()
    .join(x.drop(cols, axis=1).dropna(), how="right")
)
df = (
    load_us_politics_dates()
    .resample("MS")
    .last()
    .join(ret, how="outer")
    .pipe(government_ffill_shift, cols)
    .assign(
        united_government=lambda x: x[cols].apply(
            lambda row: row.nunique() == 1, axis=1
        ),
        democratic_president=lambda x: x["Presidency"] == "Democratic",
    )
    .loc[:end_date]
)

```

We first look whether there is a monthly seasonality in returns.

```
[6]: bar(
    df.groupby(lambda x: x.month)[["Mkt-RF"]].apply(sharpe_ratio, num_period_per_year=12),
    sort=False,
    title="Annualized sharpe ratio by month",
)
```

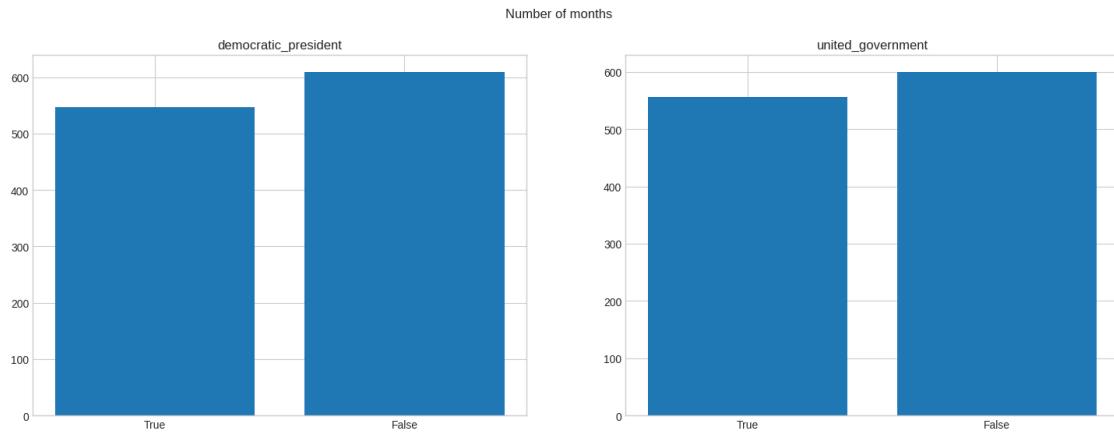


In particular, the bar chart below shows that there is no strong imbalance in the number of months in each subcategories.

```
[7]: vars = ["democratic_president", "united_government"]

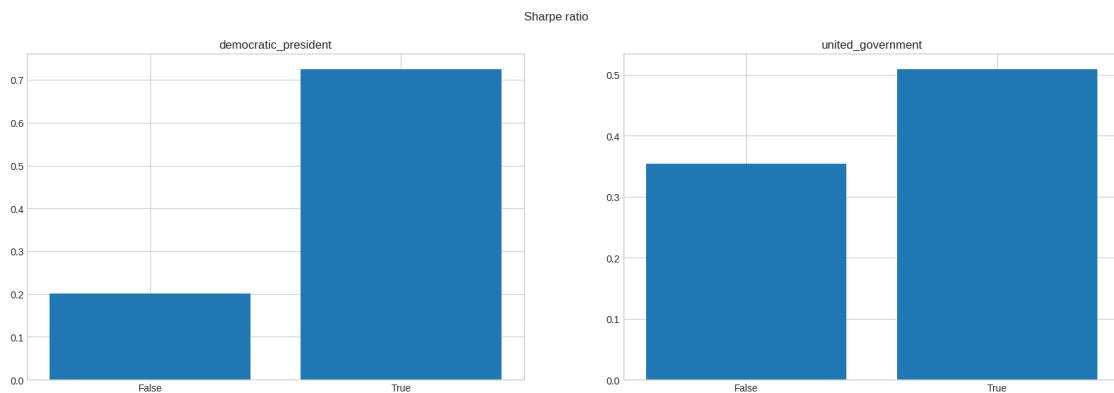
fig, ax = plt.subplots(1, 2, figsize=(18, 6))
fig.suptitle("Number of months")
for i, c in enumerate(vars):
```

```
bar(df.groupby(c)[ "RF" ].count(), ax=ax[i], title=c)
```

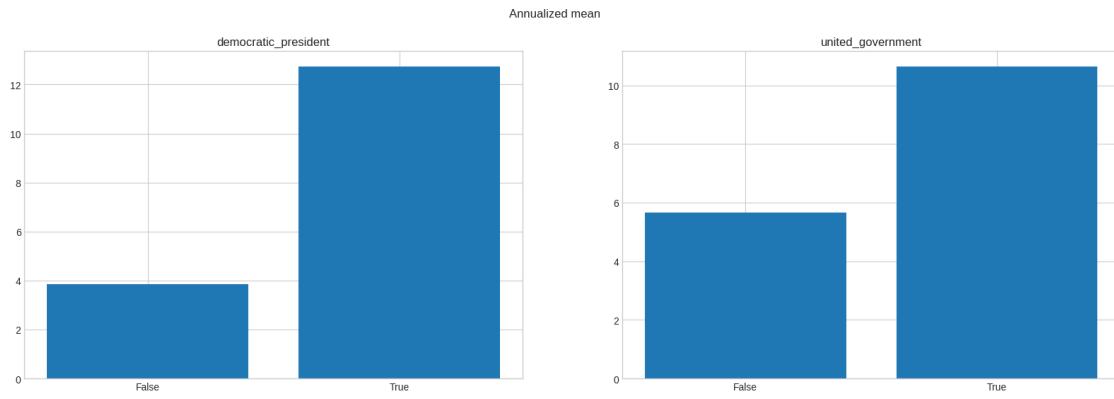


The two graphs below essentially replicates the results in Santa-Clara and Valkanov (2003) and Papamichalis et al. (2024): higher Sharpe ratio (and gain) for democratic presidents and unified governments.

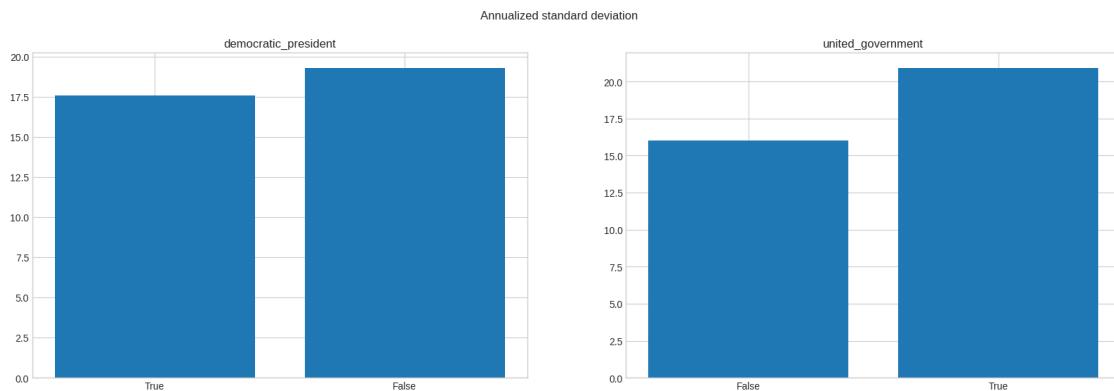
```
[8]: fig, ax = plt.subplots(1, 2, figsize=(20, 6))
fig.suptitle("Sharpe ratio")
for i, c in enumerate(vars):
    bar(
        df.groupby(c)[ "Mkt-RF" ].apply(sharpe_ratio, num_period_per_year=12),
        ax=ax[i],
        title=c,
    )
```



```
[9]: fig, ax = plt.subplots(1, 2, figsize=(20, 6))
fig.suptitle("Annualized mean")
for i, c in enumerate(vars):
    bar(df.groupby(c)[ "Mkt-RF" ].mean().mul(12), ax=ax[i], title=c)
```



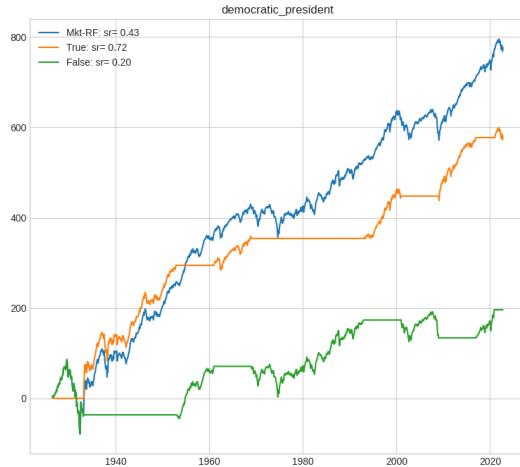
```
[10]: fig, ax = plt.subplots(1, 2, figsize=(20, 6))
fig.suptitle("Annualized standard deviation")
for i, c in enumerate(vars):
    bar(df.groupby(c)["Mkt-RF"].std().mul(np.sqrt(12)), ax=ax[i], title=c)
```



```
[11]: q = "Mkt-RF"
c = "united_government"

dfs = {
    c: df.assign(**{c: lambda x: x[c].apply(str)})
        .set_index(c, append=True)[q]
        .unstack()
        .fillna(0)
        .join(ret[q])
        for c in vars
}

fig, ax = plt.subplots(1, 2, figsize=(20, 8))
for i, (k, v) in enumerate(dfs.items()):
    line(v, cumsum=True, title=k, ax=ax[i], loc="best")
```



Chapter 22

Market Reactions to Scheduled Macroeconomic Announcements: Evidence from FOMC and Other Key Events

In this section, we describe trading around specific scheduled macroeconomic events.

22.1 Macroeconomic events

JOURNAL OF FINANCIAL AND QUANTITATIVE ANALYSIS Vol. 48, No. 2, Apr. 2013, pp. 343–375
COPYRIGHT 2013, MICHAEL G. FOSTER SCHOOL OF BUSINESS, UNIVERSITY OF WASHINGTON, SEATTLE, WA 98195
doi:10.1017/S002210901300015X

How Much Do Investors Care About Macroeconomic Risk? Evidence from Scheduled Economic Announcements

Pavel Savor and Mungo Wilson*

Three statements in Savor and Wilson (2013):

- U.S. stock market returns are significantly higher on days when important macroeconomic news is scheduled to be announced.
- Inversely, the returns on T-Bills is significantly lower on these days.
- The announcement surprises do not seem to predict the returns.

These facts are consistent with risk-averse investors demanding higher return for holding event

risk.

Data:

- 157 prescheduled CPI announcements from Jan. 1958 to Jan. 1971 and 467 for the PPI from Feb. 1971 to Dec. 2009.
- 621 employment announcements from Jan. 1958 to Dec. 2009.
- FOMC interest rate announcements start in Jan. 1978 and end in Dec. 2009.

Any unscheduled announcements are excluded, leaving us with 279 FOMC observations.

TABLE 1
Summary Statistics for Daily Stock Market Excess Returns

Table 1 presents the distribution of stock market excess returns on announcement days and nonannouncement days. Announcement days are those trading days when CPI/PPI (CPI before 1971 and PPI afterward) numbers, employment numbers, and FOMC interest rate decisions are scheduled for release. The sample covers the period 1958–2009. Market excess returns are computed as the difference between the CRSP value-weighted market return and the risk-free rate. The daily risk-free rate is derived from the 1-month risk-free rate provided by CRSP. t-statistics are given in square brackets. All numbers are expressed in basis points, and the numbers in bold are of special interest.

	Panel A. All Obs.			Panel B. Excl. Outliers (< 1 percentile or > 99 percentile)		
	Ann.	Nonann.	Diff.	Ann.	Nonann.	Diff.
Mean	11.4 [4.41]	1.1 [1.29]	10.3 [3.77]	11.7 [5.39]	1.3 [1.88]	10.4 [4.55]
1st percentile	−258.2	−257.7	−1.2	−208.4	−203.2	−5.1
25th percentile	−34.7	−40.8	6.1	−33.3	−39.3	6.0
Median	13.5	4.3	9.2	13.5	4.3	9.2
75th percentile	59.0	45.2	13.7	57.8	44.3	13.5
99th percentile	292.6	242.4	50.3	233.4	191.3	42.1
Std. dev.	98.6	94.6		81.8	75.4	
Skewness	−0.6	−0.6		0.0	−0.2	
Kurtosis	8.5	19.7		1.1	0.8	
N	1,450	11,641		1,420	11,407	

TABLE 2
Regression Analysis: Daily Stock Market Excess Returns

Table 2 presents the results of OLS regressions of daily stock market excess returns on an announcement-day dummy variable and various other controls. Ann. day is a dummy variable equaling 1 if day t is an announcement day, and 0 otherwise. Market excess returns (MKTRF) are computed as the difference between the CRSP value-weighted market return and the risk-free rate (expressed in basis points). Monday–Thursday are dummy variables for the corresponding days of the week. t-statistics are calculated using Newey-West (1987) standard errors (with 5 lags) and are given in square brackets. The numbers in bold are of special interest.

Variable	Panel A. All Obs.			Panel B. Excl. Outliers (< 1 percentile or > 99 percentile)		
	1	2	3	1	2	3
Intercept	1.133 [1.24]	−0.064 [−0.07]	3.448 [2.00]	1.324 [1.71]	0.647 [0.83]	4.670 [3.11]
Ann. day	10.291 [3.55]	10.439 [3.61]	7.093 [2.69]	10.38 [4.18]	10.653 [4.34]	8.462 [3.41]
MKTRF $_{t-1}$	0.081 [4.91]	0.081 [4.93]		0.094 [8.75]	0.094 [8.74]	
(MKTRF $_{t-1}$) 2	0.0001 [1.61]	0.0001 [1.59]		0.0000 [0.47]	0.0000 [0.45]	
Monday		−13.605 [−5.05]			−11.228 [−5.31]	
Tuesday		−2.611 [−1.06]			−5.662 [−2.71]	
Wednesday		2.151 [0.89]			1.129 [0.56]	
Thursday		−2.558 [−1.07]			−3.519 [−1.71]	
N	13,091	13,090	13,090	12,827	12,826	12,826
R ² (%)	0.1	0.9	1.2	0.2	1.4	1.7

The return (resp. standard deviation) is 11.4bps (resp. 98.6 bps) on announcement days and 1.1bps (resp. 94.6 bps) on non-announcement days, so that the annualized Sharpe ratios very different, 1.8 and .18 respectively. The next two figures show similar statements (with inverse sign) for the risk-free asset (= T-Bills) – impacted by precautionary saving demand.

TABLE 3
Summary Statistics for Daily 30-Day T-Bill Returns

Table 3 presents the distribution of daily 30-day T-bill returns on announcement days and nonannouncement days. Announcement days are those trading days when CPMPI (CPI before 1971 and PPI afterward) numbers, employment numbers, and FOMC interest rate decisions are scheduled for release. The sample covers the period 1961–2009. The 30-day T-bill returns are defined as the return of the T-bill issue whose length of maturity is closest to 30 days (daily T-bill quotes are obtained from CRSP starting in June 1961). *t*-statistics are given in square brackets. All numbers are expressed in basis points, and the numbers in bold are of special interest.

	Panel A. All Obs.			Panel B. Excl. Outliers (< 1 percentile or > 99 percentile)		
	Ann.	Nonann.	Diff.	Ann.	Nonann.	Diff.
Mean	1.5 [34.69]	2.3 [94.84]	−0.7 [−14.49]	1.5 [51.29]	2.2 [123.03]	−0.7 [−21.40]
1st percentile	−0.9	−0.7	−0.2	−0.3	−0.2	−0.2
25th percentile	0.8	1.0	−0.2	0.8	1.0	−0.2
Median	1.3	1.6	−0.3	1.3	1.6	−0.3
75th percentile	1.9	2.9	−1.0	1.9	2.9	−1.0
99th percentile	6.8	10.2	−3.5	5.5	8.5	−3.0
Std. dev.	1.6	2.5		1.1	1.8	
Skewness	8.5	0.2		1.5	1.5	
Kurtosis	163.1	55.3		3.8	2.3	
N	1,370	10,711		1,342	10,495	

TABLE 4
Regression Analysis: Daily 30-Day T-Bill Returns

Table 4 presents the results of OLS regressions of daily 30-day T-bill returns on an announcement-day dummy variable and various other controls. Ann. day is a dummy variable equaling 1 if day *t* is an announcement day, and 0 otherwise. The 30-day T-bill returns (TBILL) are defined as the return of the T-bill issue whose length of maturity is closest to 30 days (expressed in basis points). Monday–Thursday are dummy variables for the corresponding days of the week. *t*-statistics are calculated using Newey-West (1987) standard errors (with 5 lags) and are given in square brackets. The numbers in bold are of special interest.

Variable	Panel A. All Obs.			Panel B. Excl. Outliers (< 1 percentile or > 99 percentile)		
	1	2	3	1	2	3
Intercept	2.255 [69.77]	1.936 [37.28]	1.057 [13.71]	2.200 [84.54]	1.902 [55.12]	1.092 [24.45]
Ann. day	−0.726 [−15.42]	−0.707 [−15.08]	−0.106 [−2.46]	−0.724 [−22.70]	−0.694 [−22.44]	−0.141 [−4.93]
TBILL _{t−1}	0.131 [6.35]	0.214 [6.37]		0.119 [10.6]	0.209 [9.04]	
(TBILL _{t−1}) ²	−0.0053 [−0.32]	−0.0032 [−0.15]		0.0012 [1.62]	0.0017 [0.97]	
Monday		2.846 [31.89]			2.647 [57.48]	
Tuesday		−0.124 [−1.18]			−0.214 [−2.83]	
Wednesday		0.250 [5.01]			0.210 [7.32]	
Thursday		0.311 [5.89]			0.280 [10.09]	
N	12,080	11,941	11,941	11,836	11,710	11,710
R ² (%)	0.9	2.6	22.5	1.7	4.5	36.0

Variable	Panel A. Stock Market Excess Returns			Panel B. T-Bill Returns		
	Model Expect.	SPF Expect.		Model Expect.	SPF Expect.	
Ann. day	7.607 [2.60]	8.076 [2.75]	11.278 [2.51]	9.638 [2.16]	−0.150 [−3.45]	−0.156 [−3.61]
Inflation Surprise	0.015 [0.82]		0.007 [0.37]		−0.000 [−0.23]	0.000 [0.63]
Unemployment Surprise	0.088 [0.36]		0.151 [0.62]		0.000 [0.11]	0.003 [0.54]
FED Funds Surprise	−1.472 [−0.77]		−1.347 [−0.74]		−0.039 [−2.16]	−0.039 [−2.47]
Infl. Sur. × Expansion	0.013 [0.88]		0.012 [0.55]		0.000 [0.33]	0.000 [−0.31]
Unemp. Sur. × Expansion	0.310 [1.28]		−0.167 [−0.86]		−0.002 [−0.56]	−0.001 [−0.12]
FED Sur. × Expansion	−1.915 [−1.20]		−1.869 [−1.19]		−0.052 [−2.82]	−0.049 [−2.76]
Infl. Sur. × Recessions	0.018 [0.43]		−0.002 [−0.07]		−0.000 [−0.42]	0.000 [1.09]
Unemp. Sur. × Recessions	−0.768 [−1.03]		1.005 [1.61]		0.013 [1.15]	0.011 [1.67]
FED Sur. × Recessions	0.161 [0.02]		0.439 [0.07]		0.008 [0.3]	−0.003 [−0.16]
N	12,987	12,987	7,008	7,008	11,841	11,841
R ² (%)	1.2	1.2	0.3	0.4	22.6	22.6
					14.7	14.7
					6,864	6,864

Other events from the follow-up paper: Savor and Wilson (2014): “Asset pricing: a Tale of Two Days,”

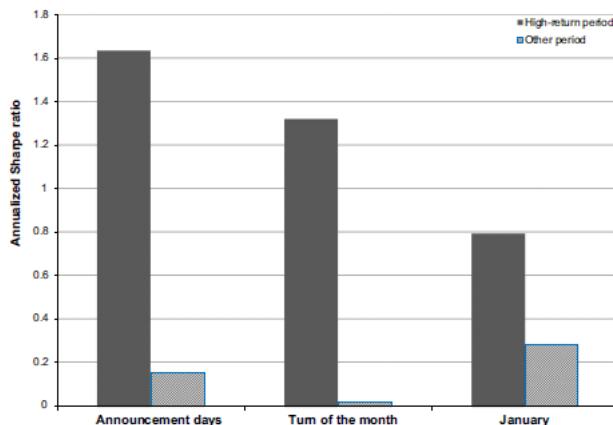


Fig. 9. Annualized market Sharpe ratio across different periods. This figure plots the annualized Sharpe ratio for the value-weighted index of all NYSE, Amex, and Nasdaq stocks separately for announcement days (days on which inflation, employment, or Federal Open Market Committee interest rate decisions are scheduled to be announced) versus all other days; the turn of the month (the last trading day of a month and the first four trading days of the following month) versus all other days; and January versus other months.

22.2 FOMC dates

The Federal Open Market Committee (FOMC) is a component of the Federal Reserve System, responsible for overseeing the country's open market operations and essentially setting U.S. monetary policy. The FOMC makes key decisions regarding interest rates and the growth of the United States money supply, influencing economic activity, employment, and inflation.

FOMC statements are important for financial markets because they provide insights into the committee's economic outlook and monetary policy intentions. Investors, analysts, and economists closely examine these statements for clues about the future direction of interest rates and other policy measures. Any changes or signals in the FOMC's approach can lead to significant movements in financial markets, affecting everything from stock prices and bond yields to currency exchange rates and commodity prices.

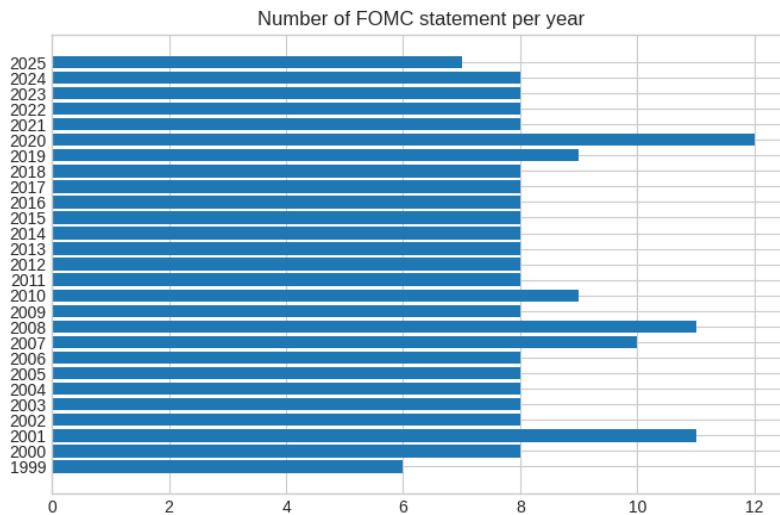
In this section, in this section, we focus only on the dates when statements are released – ignoring what was said in the statement. We will cover later how to extract information from these statements.

```
[10]: from skfin.datasets_ import load_fomc_statements
from skfin.plot import bar, line
from skfin.text import show_text

[11]: statements = load_fomc_statements(force_reload=False)

[12]: bar(
    statements.groupby(lambda x: x.year)[“text”].count(),
    sort=False,
    horizontal=True,
    title=“Number of FOMC statement per year”,
```

)



22.3 Returns on statement days

```
[13]: import numpy as np
import pandas as pd
from pandas.tseries.offsets import BDay
from skfin.datasets_ import load_kf_returns

ret = load_kf_returns(filename="F-F_Research_Data_Factors_daily")["Daily"]
```

```
[14]: ret_norm = ret.div(ret.ewm(halflife=252).std())[statements.sort_index().index[0] :].asfreq("B")
```

Some FOMC meetings on Sundays or when the markets are closed:

- ‘2008-01-21’: Luther King Jr Day
- ‘2010-05-09’: Sunday
- ‘2020-03-15’: (note the release time!!!)

```
[15]: special_days = ["2008-01-22", "2010-05-09", "2020-03-15"]
idx0 = pd.to_datetime(pd.Index(special_days))
idx = statements.index.difference(idx0).union(idx0 + BDay(1))
```

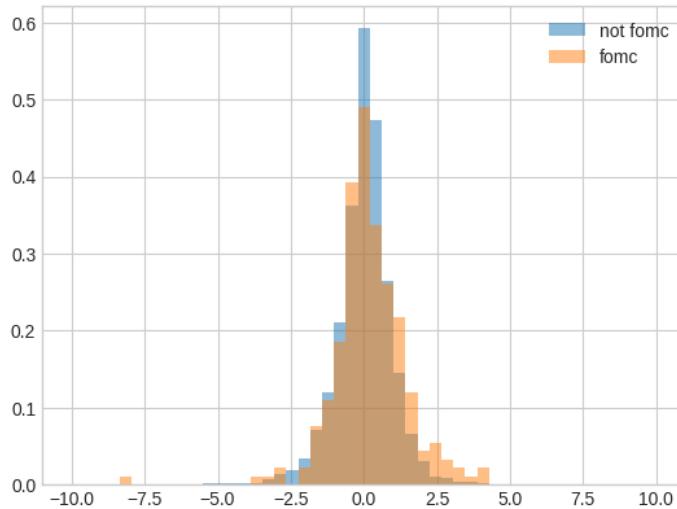
```
[16]: ret_fomc = pd.concat(
{
    "fomc": ret_norm.loc[ret_norm.index.intersection(idx)],
    "not fomc": ret_norm.loc[ret_norm.index.difference(idx)],
},
axis=1,
)
```

```
[17]: bins = np.linspace(-10, 10, 50)
plt.hist(
    ret_fomc["not fomc"]["Mkt-RF"].dropna(),
    bins,
```

```

density=True,
alpha=0.5,
label="not fomc",
)
plt.hist(
    ret_fomc["fomc"]["Mkt-RF"].dropna(), bins, density=True, alpha=0.5, label="fomc"
)
plt.legend(loc="upper right")
plt.show()

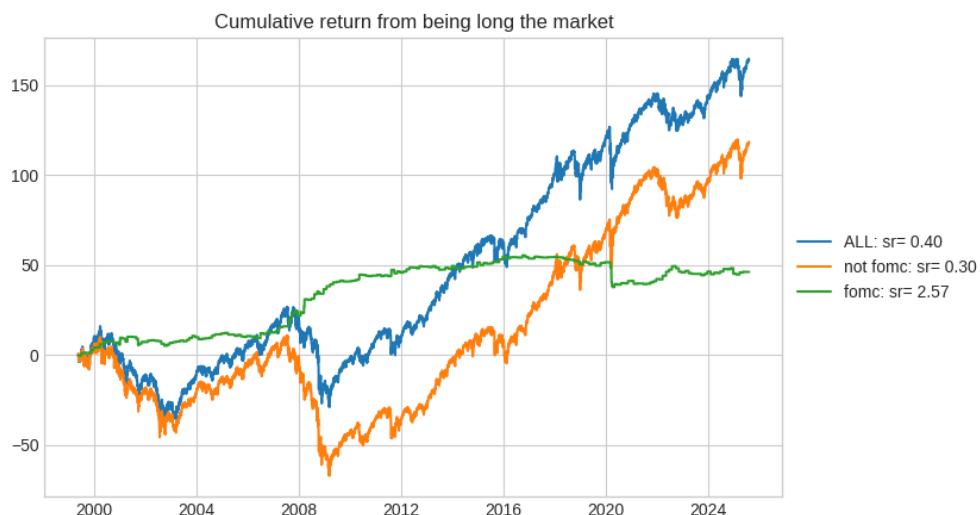
```



```

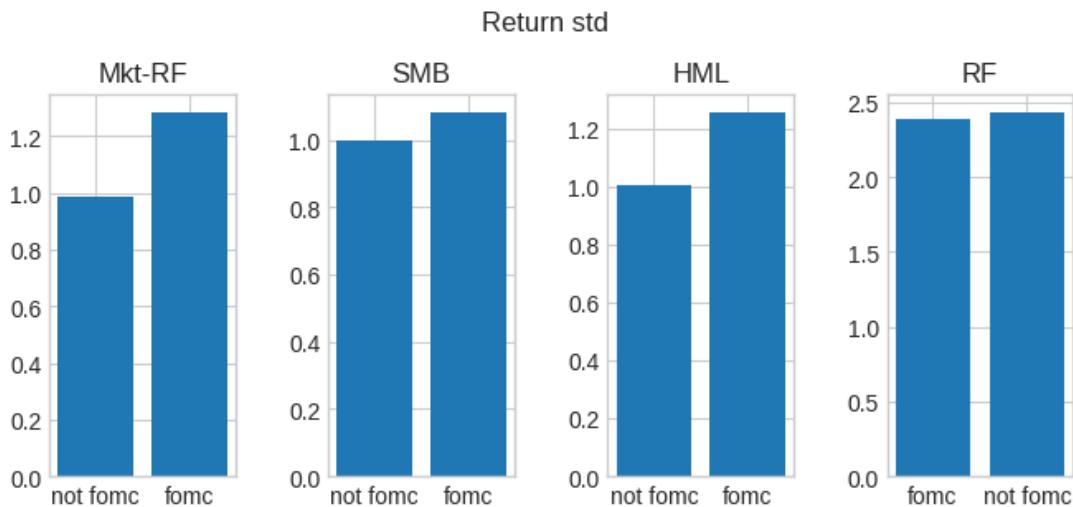
[18]: line(
    ret_fomc.xs("Mkt-RF", axis=1, level=1)
    .fillna(0)
    .assign(ALL=lambda x: x.sum(axis=1))["1996":],
    cumsum=True,
    title="Cumulative return from being long the market",
)

```

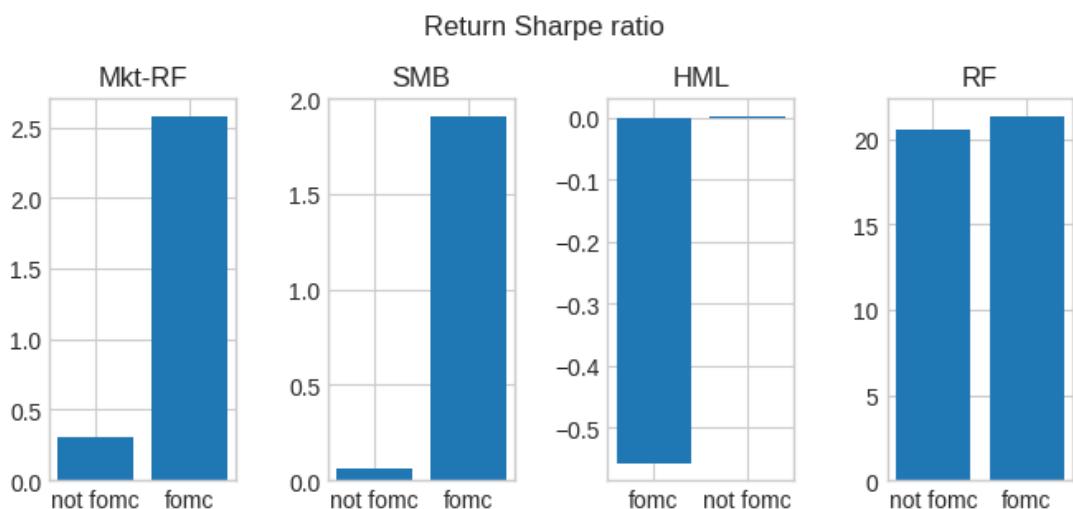


```
[19]: from skfin.metrics import sharpe_ratio
def bars(df, wspace=.5, suptitle='', y=1):
    T, N = df.shape
    fig, axs = plt.subplots(1, N, figsize=(8, 3))
    fig.subplots_adjust(wspace=wspace)
    fig.suptitle(suptitle, y=y, fontsize=12)
    for i, c in enumerate(df.columns):
        bar(df[c], ax=axs[i], title=c)
```

```
[20]: bars(ret_fomc.std().unstack(), suptitle='Return std', y=1.05)
```

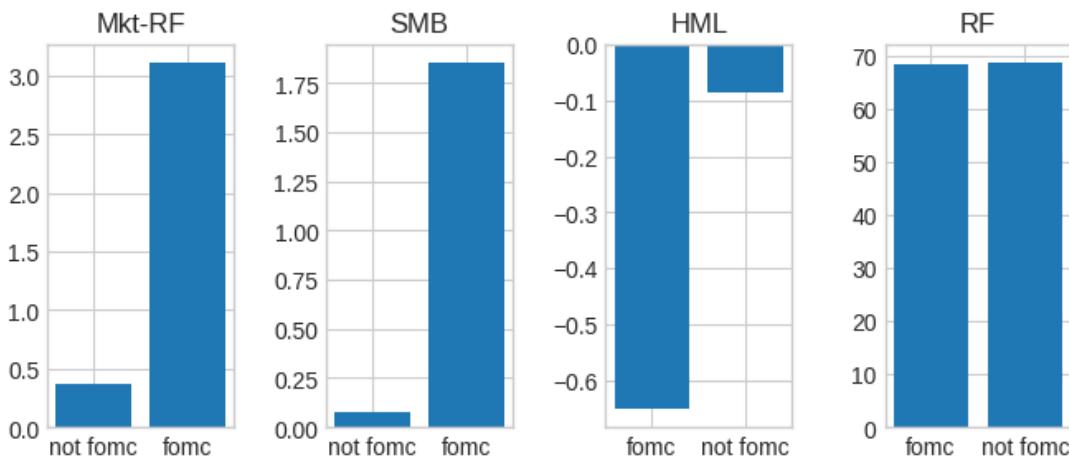


```
[21]: bars(ret_fomc.pipe(sharpe_ratio).unstack(), suptitle='Return Sharpe ratio', y=1.05)
```



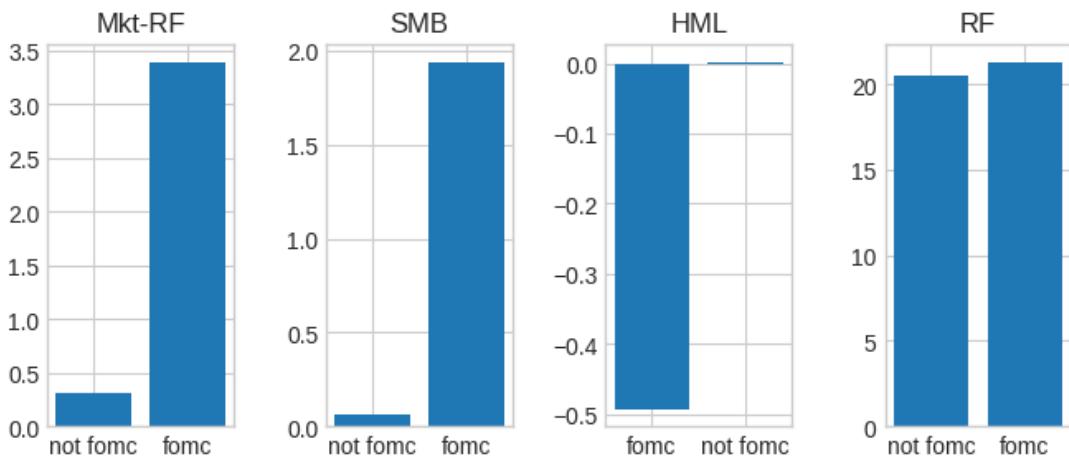
```
[22]: bars(ret_fomc.clip(lower=-3, upper=3).pipe(sharpe_ratio).unstack(), suptitle='Return Sharpe ratio (3-sigma clipping)', y=1.05)
```

Return Sharpe ratio (3-sigma clipping)



```
[23]: bars(ret_fomc.drop(pd.to_datetime("2020-03-16")).asfreq("B").pipe(sharpe_ratio).unstack(),  
        suptitle='Return Sharpe ratio (dropping March 16th 2020)', y=1.05)
```

Return Sharpe ratio (dropping March 16th 2020)



Chapter 23

Text Representation and Sentiment Analysis of FOMC Statements: From Bag-of-Words to Deep Learning Embeddings

In the previous section, we focused only on the dates of Federal Open Market Committee (FOMC) statements. We now turn to the task of extracting information from these statements – in particular by providing a vector representation of each document.

Representations of text have evolved significantly over time, starting from simple approaches such as the bag of words model, which treats text as an unordered collection of words and relies on exogenous dictionaries to map words to features. This method, however, lacks context and semantic understanding.

To address these limitations, sparse representations like Term Frequency-Inverse Document Frequency (TF-IDF) were developed, which weigh words based on their importance across documents. These sparse matrices can be transformed into dense representations using techniques like Principal Component Analysis (PCA), Non-negative Matrix Factorization (NMF), or k-means clustering, improving efficiency and interpretability.

More recently, advances in natural language processing have led to the emergence of dense representations through encoder transformer language models, such as BERT and GPT. These models capture deep contextual and semantic information by encoding words in multiple dimensions, revolutionizing how text is understood and processed in computational linguistics.

23.1 Loading the FOMC statements

As discussed in the previous section, the FOMC statements can be loaded with the function `load_fomc_statements` and inspected with the function `show_text`.

```
[2]: from skfin.datasets_ import load_fomc_statements
from skfin.text import show_text, plot_document_embeddings, plot_word_embeddings

statements = load_fomc_statements(force_reload=False)
```



```
[3]: special_days = ["2008-01-22", "2010-05-09", "2020-03-15"]
```

```
[4]: show_text(statements.loc[special_days], n=3)
```

```
<IPython.core.display.HTML object>
```

23.2 Bag of words: Sentiment in FOMC statements

In this section, we measure sentiment with the Loughran-McDonalds sentiment dictionary in two ways:

- sentiment = (#positive - #negative) / (#positive + #negative)
- sentiment = (#positive - #negative) / (#words)

In the first case, short documents (with few or no sentiment words) might lead to biased estimates.

```
[5]: import pandas as pd
from skfin.datasets_ import load_fomc_statements, load_loughran_mcdonald_dictionary
from skfin.plot import line
from skfin.text import coefs_plot, show_text
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
```

```
[6]: statements = load_fomc_statements()
lm = load_loughran_mcdonald_dictionary()
```

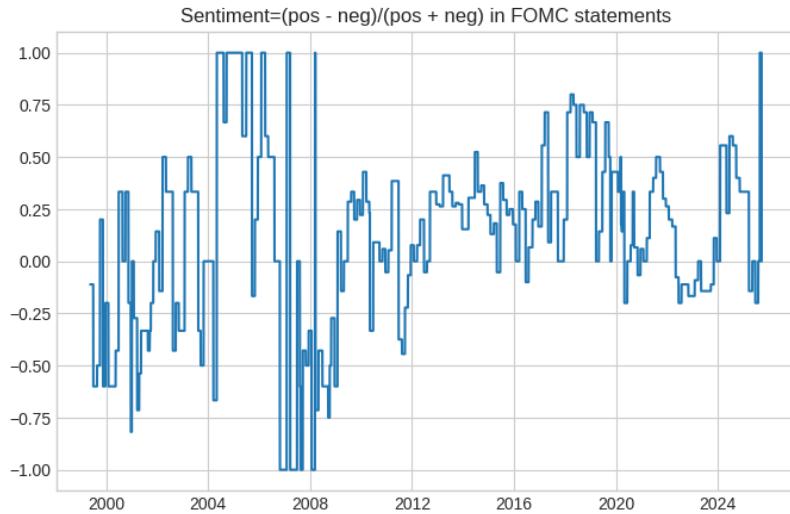
```
[7]: X = statements["text"]
```

```
[8]: funcs = {
    "negative": lambda x: x.Negative > 0,
    "positive": lambda x: x.Positive > 0,
    "all": lambda x: x.Word.notna(),
}

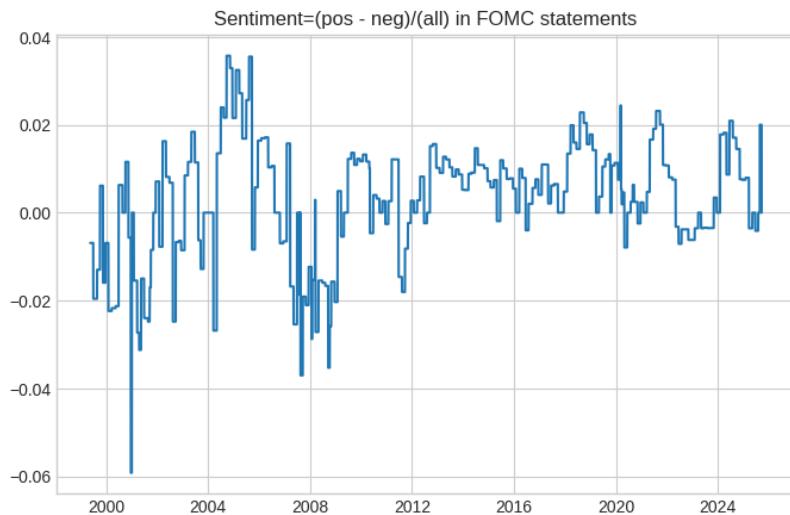
def get_total_count(X, lm, func):
    m = CountVectorizer(vocabulary=lm.loc[func].Word.str.lower().values)
    return pd.DataFrame(m.fit_transform(X).toarray(), index=X.index).sum(axis=1)

lm_counts = pd.concat({k: get_total_count(X, lm, v) for k, v in funcs.items()}, axis=1)
```

```
[9]: line(
    lm_counts.pipe(lambda x: (x.positive - x.negative) / (x.positive + x.negative))
    .resample("B")
    .last()
    .ffill(),
    legend=False,
    title="Sentiment=(pos - neg)/(pos + neg) in FOMC statements",
)
```



```
[10]: line(
    lm_counts.pipe(lambda x: (x.positive - x.negative) / x["all"])
    .resample("B")
    .last()
    .ffill(),
    legend=False,
    title="Sentiment=(pos - neg)/(all) in FOMC statements",
)
```



```
[11]: lm_lexica = {
    "negative": pd.Series(1, lm.loc[lm.Negative > 0].Word.str.lower().values),
    "positive": pd.Series(1, lm.loc[lm.Positive > 0].Word.str.lower().values),
}
show_text(
    statements.loc[["2000-12-19", "2013-12-18", "2014-01-29"]], lexica=lm_lexica, n=None
)
```

<IPython.core.display.HTML object>

23.3 Sparse text representation: TFIDF

The progress of Natural Language Processing has been based on coming up with progressively better representations of text. In this section, we first discuss “word counting” in a given corpus and then embedding derived from pretrained language models.

In order to extract features from text, the simplest way is to count words. In `scikit-learn`, this is done with the function `CountVectorizer`. A slightly more advanced feature is to select words based on a TFIDF score, defined as the product of the term frequency (TF) and the inverse document frequency (IDF). More precisely, the TFIDF score trades off: - the terms that are frequent and therefore important in a corpus: - the terms that appear in almost all documents and therefore are not helping to discriminate across documents.

In `TfidfVectorizer`, terms can be filtered additionally with: - a stop word list - min and max document frequencies or counts - some token pattern (e.g. that eliminates the short tokens).

```
[12]: from sklearn.decomposition import NMF
from sklearn.cluster import KMeans
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import Pipeline
```

```
[13]: vectorizer = TfidfVectorizer(
    stop_words="english",
    min_df=5,
    max_df=0.8,
    ngram_range=(1, 3),
    token_pattern=r"\b[a-zA-Z]{3,}\b",
)
X_ = vectorizer.fit_transform(statements[["text"]].values)
```

```
[14]: cols = vectorizer.get_feature_names_out()
print(len(cols))
list(cols)[:10]
```

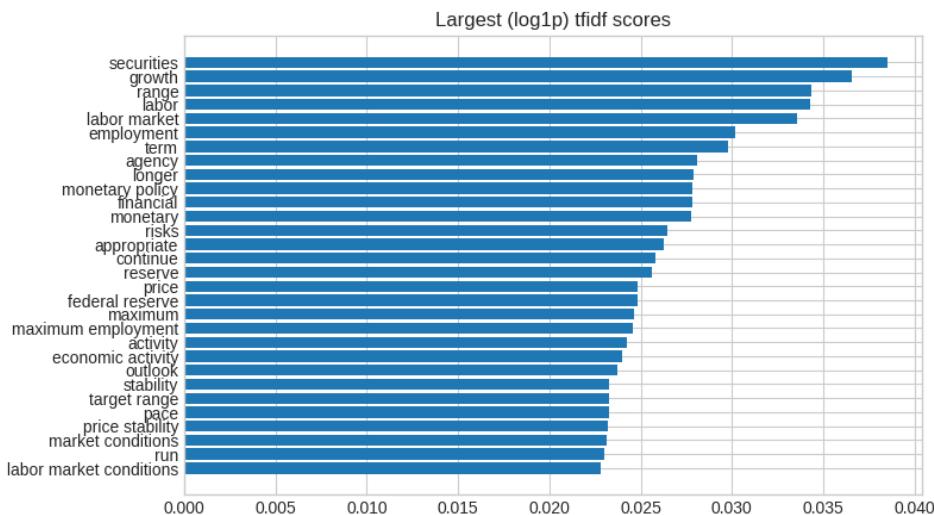
4355

```
[14]: ['abate',
'abating',
'abroad',
'accommodation',
'accommodation balanced',
'accommodation balanced approach',
'accommodation economic',
'accommodation economic activity',
'accommodation economic growth',
'accommodation removed']
```

In what follows, to reduce the impact over large `tfidf` coefficients, we use the log transformation $x \mapsto \log(1 + x)$

```
[15]: X_tfidf = pd.DataFrame(
    np.log1p(X_.toarray()), index=statements[["text"]].index, columns=cols
)
```

```
[16]: bar(
    X_tfidf.mean().sort_values(ascending=False).head(30),
    horizontal=True,
    title="Largest (log1p) tfidf scores",
)
```



23.3.1 Principal component exploration

In terms of representation of text, one key feature of the tfidf representation is that it is very sparse (with many zeros). For the tfidf representation X (where each row is a document and each column is an n-gram term out of the tfidf), the idea of a low-rank approximation is \hat{X} and H such that

$$X \approx \hat{X}H^T,$$

where \hat{X} is a “denser” matrix than X because intuitively, some columns have been combined.

We first perform Principal Component Analysis (PCA) using the singular-value decomposition (svd) function in numpy.

```
[17]: u, s, w = np.linalg.svd(X_tfidf)
```

A singular value decomposition yields:

$$X = U \times \text{Diag}(s) \times W^T,$$

where s is the vector of eigenvalues and U and W are the matrix of eigenvectors. For a number of modes n , define: s_n the vector with the first (largest) n eigenvalues; and U_n and W_n the first n columns of U and W .

Then with $\hat{X} = U_n \times \text{Diag}(\sqrt{s_n})$ and $H_n = W_n \times \text{Diag}(\sqrt{s_n})$, we have:

$$X \approx \hat{X}H_n^T.$$

```
[18]: n_modes = 6

signed_sqrt_eigv = np.diag(
    np.sqrt(s[:n_modes]) * np.sign(np.mean(u[:, :n_modes], axis=0))
)
X_pca = pd.DataFrame(u[:, :n_modes].dot(signed_sqrt_eigv), index=statements.index)
H_ = pd.DataFrame(w[:n_modes, :].T.dot(signed_sqrt_eigv), index=cols)
```

Chapter 23. Text Representation and Sentiment Analysis of FOMC Statements: From Bag-of-Words to Deep Learning Embeddings

We can compute a distance between the features X and the approximation $\hat{X}_n H_n^T$. More precisely, the Frobinus norm is the sum of squared coefficients of the matrix and it can be computed with `scipy`.

```
[19]: norm_pca = scipy.linalg.norm(X_pca.dot(H_.T).sub(X_tfidf), ord="fro")
```

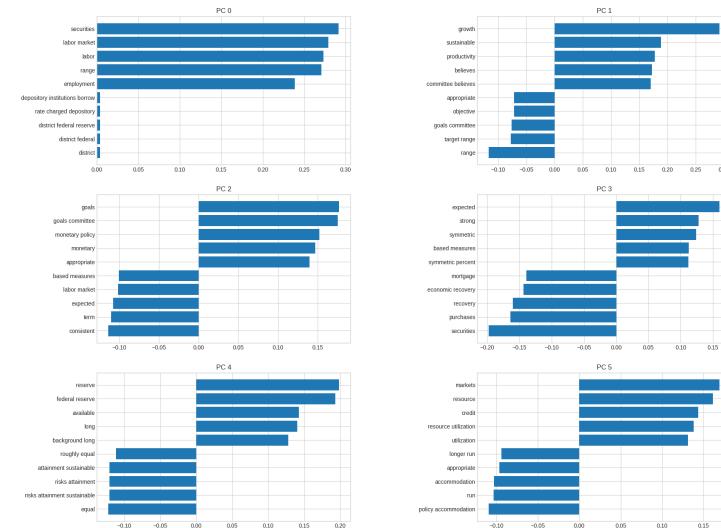
```
[20]: np.allclose(norm_pca, np.sqrt(X_pca.dot(H_.T).sub(X_tfidf).pow(2).sum().sum()))
```

```
[20]: True
```

```
[21]: def topbottom(x, n=5):
    return pd.concat([x.nlargest(n=n), x.sort_values(ascending=False).tail(n)])
```



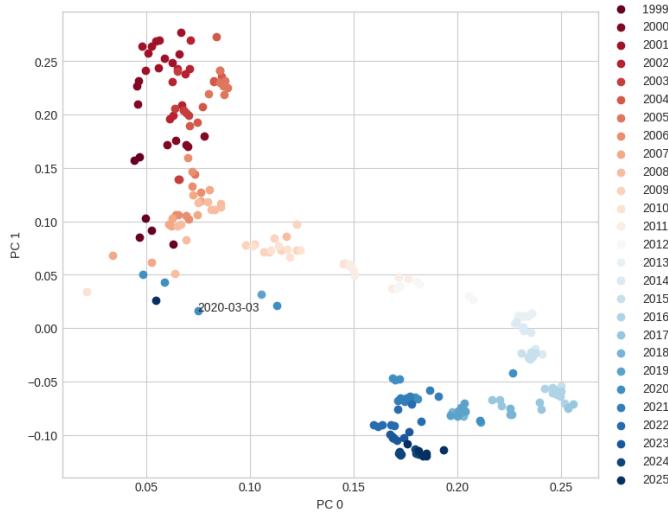
```
fig, ax = plt.subplots(int(n_modes / 2), 2, figsize=(20, 16))
ax = ax.ravel()
plt.subplots_adjust(wspace=0.5)
for i in range(n_modes):
    bar(H_[i].pipe(topbottom, n=5), horizontal=True, ax=ax[i], title=f"PC {i}")
```



The plot above shows that the first principal component PC0 is related to the labor market and the second principal component PC1 is relate to economic growth. The graph below shows the loadings on these factor over time by document.

- in the earlier years (1999-2009), the statements talk more about economic growth (high loading on PC1, low loading on PC0), but there is a switch in later years (2010-2020).
- interestingly the last part of the sample (2021-2023) are in the middle and do not seem well explained by the loadings.

```
[22]: plot_document_embeddings(X_pca)
```



23.3.2 Clustering

```
[23]: m = KMeans(n_clusters=6).fit(X_tfidf)
X_kmeans = pd.get_dummies(pd.Series(m.labels_, index=statements.index))

H3_ = pd.DataFrame(m.cluster_centers_.T, index=cols)
norm_kmeans = scipy.linalg.norm(X_kmeans.dot(H3_.T).dot(X_tfidf), ord="fro")
```

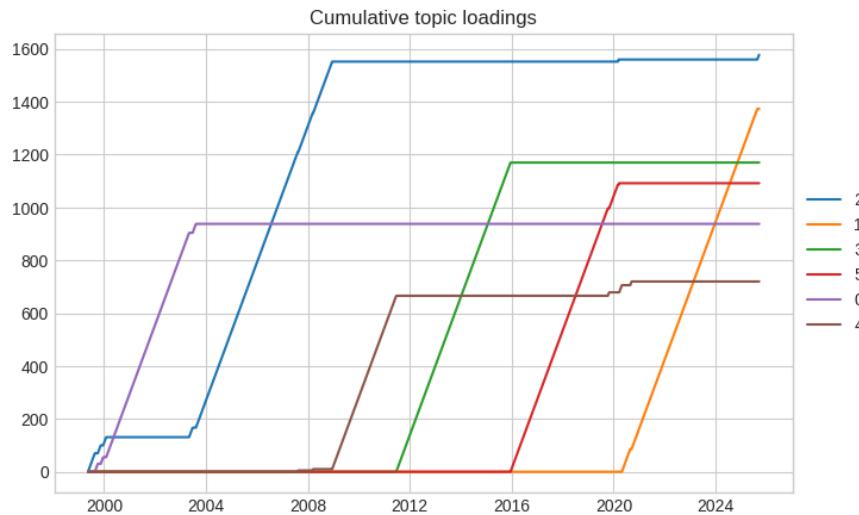
```
[24]: print(
    f"The norm of the approximations are: pca = {norm_pca:.2f}; kmeans = {norm_kmeans:.2f}."
```

The norm of the approximations are: pca = 11.19; kmeans = 11.48.

```
[25]: plot_word_embeddings(H3_)
```



```
[26]: line(
    X_kmeans.resample("B").last().ffill(),
    cumsum=True,
    title="Cumulative topic loadings",
    legend_sharpe_ratio=False,
)
```



Does the sentence transformers embeddings yield a decomposition which is less clustered in time?

23.3.3 Non-negative matrix factorization

It is often informative to group tokens into topics that explain differences across documents. A powerful algorithm is the non-negative matrix factorisation (NMF): for a non-negative matrix X (such as the one with tfidf scores), NMF finds two other non-negative matrices such that

$$X \approx \hat{X}_n H_n^T.$$

The number of topics (called `n_components` in the `scikit-learn` implementation) determines the number of columns in X_n and the number of rows in H_n .

```
[27]: n_components = 6
m = NMF(
    n_components=n_components,
    init="nndsvd",
    solver="cd",
    beta_loss="frobenius",
    random_state=1,
    alpha_W=0,
    l1_ratio=0,
    max_iter=500,
).fit(X_tfidf)
```

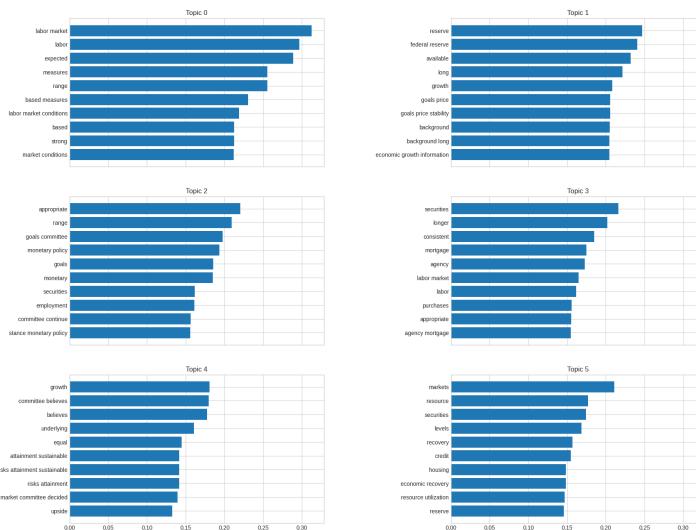
```
[28]: H2_ = pd.DataFrame(m.components_.T, index=cols)
X_nmf = pd.DataFrame(m.transform(X_tfidf), index=statements.index)
```

The non-negative matrix factorization provides a slightly better approximation as measured by the Frobinus norm.

```
[29]: norm_nmf = scipy.linalg.norm(X_nmf.dot(H2_.T).sub(X_tfidf), ord="fro")
print(
    f"The norm of the approximations are: pca = {norm_pca:.2f}; ; kmeans = {norm_kmeans:.2f}; nmf = {norm_nmf:.2f}." )
)
```

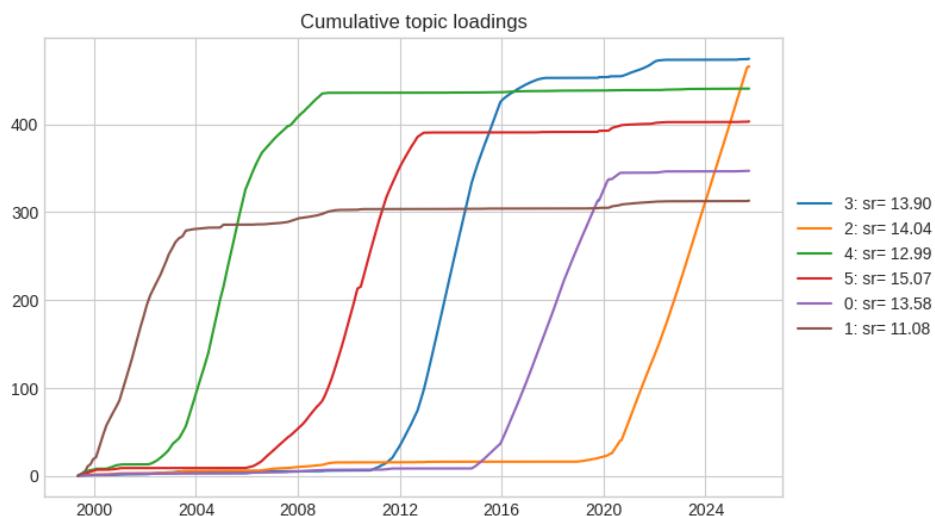
The norm of the approximations are: pca = 11.19; ; kmeans = 11.48; nmf = 11.23.

```
[30]: plot_word_embeddings(H2_)
```



Are these topics interesting? This is a matter of interpretation, but at least, the graph below shows that these topics capture a strong element of time-clustering which makes it a bit less useful.

```
[31]: line(X_nmf.resample("B").last().ffill(), cumsum=True, title="Cumulative topic loadings")
```



23.4 Deep-learning embeddings: sentence transformers

Deep-learning models have been used heavily to power NLP applications, in particular with transformers architecture starting with Delvin et al. (2018): “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. Sentence Transformers are language models fine-tuned from pretrained language models to specifically generate meaningful text representations (as embeddings).

- <https://www.sbert.net/>

```
[32]: from sentence_transformers import SentenceTransformer

def count_trainable_parameters(model):
    model_parameters = filter(lambda p: p.requires_grad, model.parameters())
    params = sum([np.prod(p.size()) for p in model_parameters])
    return params
```

We use here a specific pretrained model and use it to derive embeddings for the corpus of Fed statements.

```
[33]: lm_name = "all-distilroberta-v1"
m = SentenceTransformer(lm_name, device="cpu", trust_remote_code=True)
X_sbert = m.encode(statements[["text"]].values, batch_size=2);
```

```
[34]: print(
    f"Model card:\n - model name: {lm_name}\n - number of parameters:{count_trainable_parameters(m) / 1e6:.1f}m\n - embedding size: {m.get_sentence_embedding_dimension()}"
)
```

Model card:
- model name: all-distilroberta-v1
- number of parameters: 82.1m
- embedding size: 768

23.5 UMAP

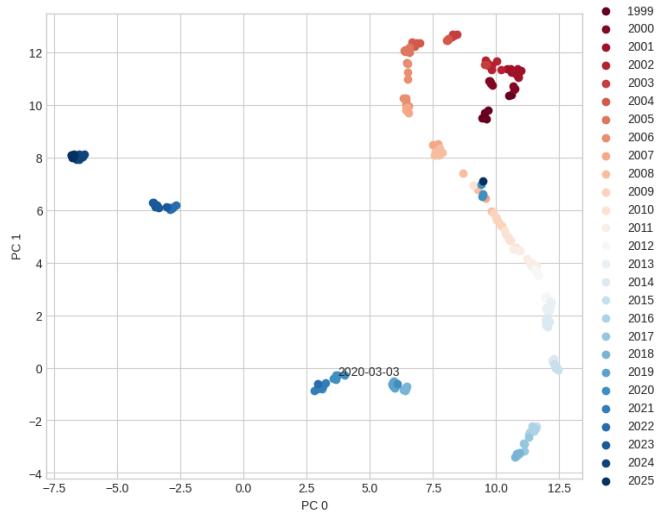
Uniform Manifold Approximation and Projection (UMAP) is a non-linear dimensionality reduction technique. It works by constructing a high-dimensional graph representation of the data and then optimizing a low-dimensional graph to be as structurally similar as possible. UMAP is useful because it effectively preserves both local and global structures in the data, which makes it particularly good for visualizing clusters and relationships in high-dimensional datasets.

- <https://umap-learn.readthedocs.io/en/latest/>

```
[35]: from umap import UMAP
```

```
[36]: embedding_ = UMAP().fit_transform(X_tfidf)
X_umap = pd.DataFrame(embedding_, index=statements.index)
```

```
[37]: plot_document_embeddings(X_umap)
```



Chapter 24

Supervised Learning on FOMC Statements: Predicting Rate Decisions and Market Reactions

24.1 Supervised learning: vector representation + Elastic net

In this section, we use the corpus of FOMC statements for supervised learning. More precisely, we match the text of the statements to the decision of the committee to raise rates, decrease rates or do nothing.

In practice, this implemented by using scikit-learn pipelines and chaining the TfidfVectorizer with a logistic regression.

```
[2]: from skfin.datasets import load_fomc_statements
from skfin.dataloaders.constants.dates import load_fomc_change_date

statements = load_fomc_statements(force_reload=False)

fomc_change_up, fomc_change_dw = load_fomc_change_date()
```

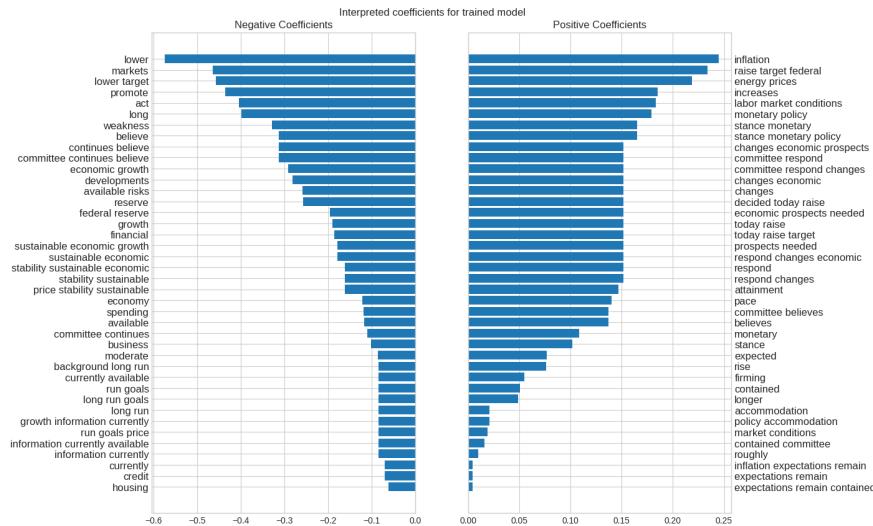
```
[3]: other = {
    "other_dt_change": ["2003-01-09", "2008-03-16", "2011-06-22"],
    "statements_dt_change_other": ["2007-08-16"],
    "qe1": ["2008-11-25", "2008-12-01", "2008-12-16", "2009-03-18"],
    "qe2": ["2010-11-03"],
    "twist": ["2011-09-21", "2012-06-20"],
    "qe3": ["2012-09-13", "2012-12-12", "2013-12-13"],
    "corona": ["2020-03-20"],
}
```

```
[4]: dates = {
    "up": fomc_change_up,
    "dw": fomc_change_dw,
    "other": [d for c in other.values() for d in c],
}
dates["no change"] = statements.index.difference([d for c in dates.values() for d in c])
```

```
[5]: from skfin.text import coefs_plot, show_text
from sklearn.linear_model import ElasticNet, LogisticRegression
from sklearn.preprocessing import FunctionTransformer
```

```
[6]: est = Pipeline(
    [
        (
            "tfidf",
            TfidfVectorizer(
                vocabulary=None,
                ngram_range=(1, 3),
                max_features=500,
                stop_words="english",
                token_pattern=r"\b[a-zA-Z]{3,}\b",
            ),
        ),
        ("log1p", FunctionTransformer(np.log1p)),
        (
            "reg",
            LogisticRegression(
                C=1, l1_ratio=0.35, penalty="elasticnet", solver="saga", max_iter=500
            ),
        ),
    ],
)
X, y = pd.concat([
    statements.loc[fomc_change_up].assign(change=1),
    statements.loc[fomc_change_dw].assign(change=-1),
])
).pipe(lambda df: (df["text"], df["change"]))
est.fit(X, y)
vocab_ = pd.Series(est.named_steps["tfidf"].vocabulary_).sort_values().index
```

```
[7]: interpret_coef = pd.DataFrame(np.transpose(est.named_steps["reg"].coef_), index=vocab_)
coefs_plot(interpret_coef, title="Interpreted coefficients for trained model")
```



A trick is that using a linear regression (e.g. ElasticNet) instead of a logistic regression is faster and as efficient (even sometimes better)

```
[8]: est = Pipeline(
    [
        (

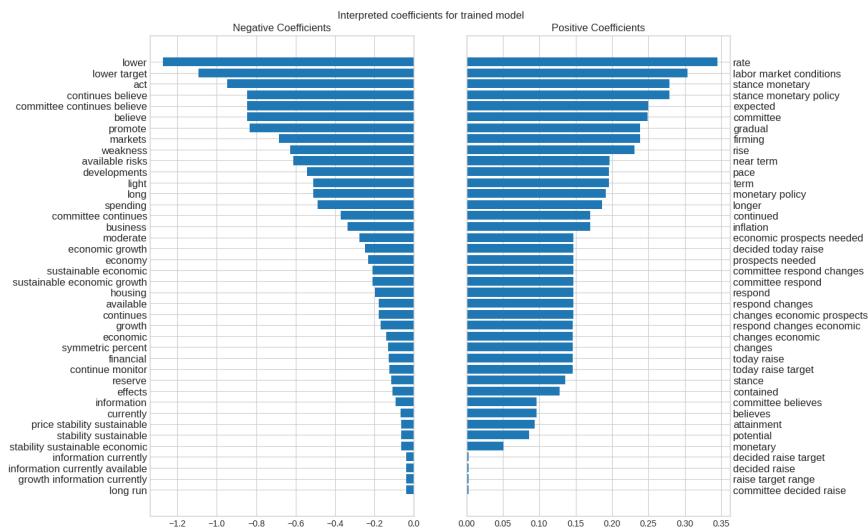
```

```

        "tfidf",
        TfidfVectorizer(
            vocabulary=None,
            ngram_range=(1, 3),
            max_features=500,
            stop_words="english",
            token_pattern=r"\b[a-zA-Z]{3,}\b",
        ),
    ),
    ("log1p", FunctionTransformer(np.log1p)),
    ("reg", ElasticNet(alpha=0.01)),
]
)
X, y = pd.concat([
    statements.loc[fomc_change_up].assign(change=1),
    statements.loc[fomc_change_dw].assign(change=-1),
])
).pipe(lambda df: (df[["text"]], df[["change"]]))
est.fit(X, y)
vocab_ = pd.Series(est.named_steps["tfidf"].vocabulary_).sort_values().index

```

[9]: `interpret_coef = pd.DataFrame(np.transpose(est.named_steps["reg"].coef_), index=vocab_)`
`coefs_plot(interpret_coef, title="Interpreted coefficients for trained model")`

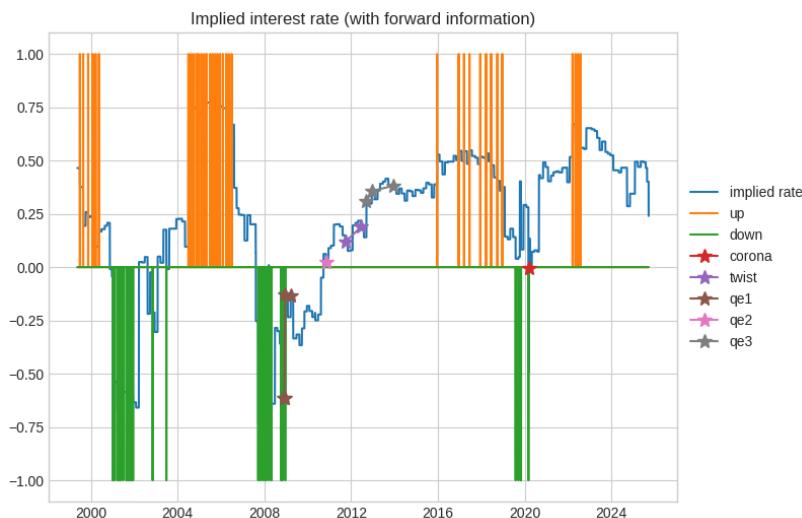


[10]: `fig, ax = plt.subplots(figsize=(8, 6))`
`pred_tfidf =`
`pd.Series(est.predict(statements[["text"]]), index=statements.index)`
`.resample("B")`
`.last()`
`.ffill()`
`)`
`line(`
`pred_tfidf.rename("implied rate")`
`.to_frame()`
`.join(`
`pd.Series(1, index=fomc_change_up)`
`.reindex(pred_tfidf.index)`
`.fillna(0)`

```

        .rename("up")
    )
    .join(
        pd.Series(-1, index=fomc_change_dw)
        .reindex(pred_tfidf.index)
        .fillna(0)
        .rename("dw")
    ),
    sort=False,
    ax=ax,
    title="Implied interest rate (with forward information)",
)
cols = ["corona", "twist", "qe1", "qe2", "qe3"]
for c in cols:
    ax.plot(pred_tfidf.loc[other[c]], marker="*", ms=10)
ax.legend(
    ["implied rate", "up", "down"] + cols, loc="center left", bbox_to_anchor=(1, 0.5)
);

```



```
[11]: lexica = {
    "positive": interpret_coef.squeeze().nlargest(n=10),
    "negative": interpret_coef.squeeze().nsmallest(n=10),
}
```

```
[12]: idx_ = (
    pd.Series(est.predict(X), index=X.index)
    .sort_values()
    .pipe(lambda x: [x.index[0], x.index[-1]])
)
show_text(statements.loc[idx_], lexica=lexica, n=None)
```

<IPython.core.display.HTML object>

24.1.1 comparison with sentence transformer embeddings

To test the usefulness of these SentenceTransformer, we run a regression of the embeddings on the rate decision. Warning: this is a full sample regression, so this is just an illustration, not a statistical test.

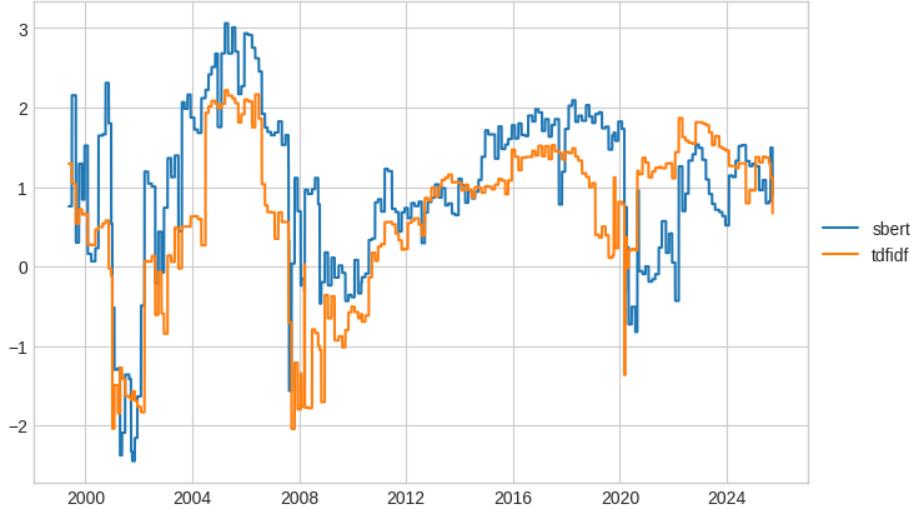
```
[13]: from sentence_transformers import SentenceTransformer
lm_name = "all-distilroberta-v1"
m = SentenceTransformer(lm_name, device="cpu", trust_remote_code=True)
X_sbert = m.encode(statements["text"].values, batch_size=2)

[14]: df = pd.DataFrame(X_sbert, index=statements.index)
m = ElasticNet(alpha=0.01)
X_, y_ = pd.concat([
    [df.loc[fomc_change_up].assign(change=1), df.loc[fomc_change_dw].assign(change=-1)]
]).pipe(lambda df: (df.drop("change", axis=1), df["change"]))
m.fit(X_, y_)
pred_sbert = (
    pd.Series(m.predict(df), index=statements.index).resample("B").last().ffill()
)

[15]: corr_tfidf_sbert = (
    pd.concat({"sbert": pred_sbert, "tfidf": pred_tfidf}, axis=1).corr().iloc[0, 1]
)
print(
    f"The correlation of the in-sample prediction for the decisions of the Fed for the two text representations (tfidf and sbert) is {corr_tfidf_sbert:.2f}."
)
```

The correlation of the in-sample prediction for the decisions of the Fed for the two text representations (tfidf and sbert) is 0.68.

```
[16]: line(
    pd.concat({"sbert": pred_sbert, "tfidf": pred_tfidf}, axis=1).pipe(
        lambda x: x.div(x.std())
    )
)
```



24.2 Sentiment in FOMC statements: supervised learning

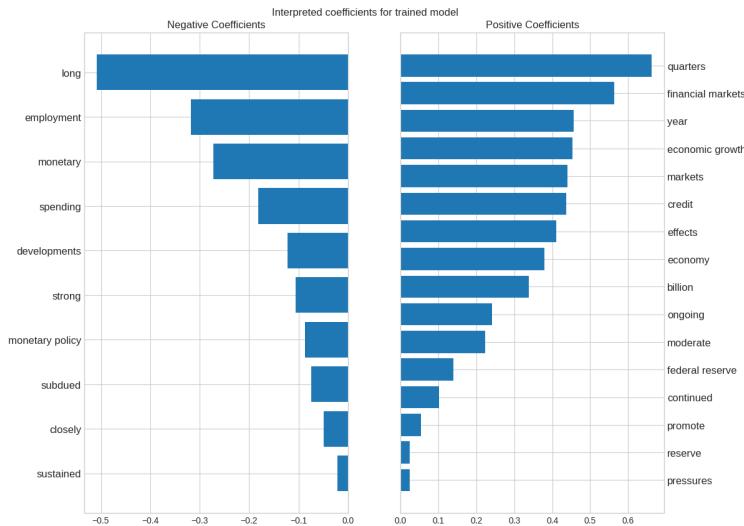
Building on previous analyses, we build here a scikit-learn pipeline with a Tfidfvectorizer and a regularized regressionElasticNet. The target is the return of the market on the day of the

statement.

```
[17]: ret = load_kf_returns(filename="F-F_Research_Data_Factors_daily")["Daily"]
```

```
[18]: special_days = ["2008-01-22", "2010-05-09", "2020-03-15"]
idx0 = pd.to_datetime(pd.Index(special_days))
idx = statements.index.difference(idx0).union(idx0 + BDay(1))
ret_fomc = ret.div(ret.ewm(252).std()).loc[ret.index.intersection(idx)]
```

```
[19]: est = Pipeline(
    [
        (
            "tfidf",
            TfidfVectorizer(
                vocabulary=None,
                ngram_range=(1, 3),
                max_features=500,
                stop_words="english",
                token_pattern=r"\b[a-zA-Z]{3,}\b",
            ),
        ),
        ("reg", ElasticNet(alpha=0.0075)),
    ]
)
y = ret_fomc["Mkt-RF"].dropna()
X = statements["text"]
idx_ = y.index.intersection(X.index)
X, y = X.loc[idx_], y.loc[idx_]
est.fit(X, y)
vocab_ = pd.Series(est.named_steps["tfidf"].vocabulary_).sort_values().index
interpret_coef = pd.DataFrame(np.transpose(est.named_steps["reg"].coef_), index=vocab_)
coefs_plot(interpret_coef, title="Interpreted coefficients for trained model")
```



```
[20]: lexica = {
    "positive": interpret_coef.squeeze().nlargest(n=10),
    "negative": interpret_coef.squeeze().nsmallest(n=10),
}
```

```
[21]: idx_ = (
    pd.Series(est.predict(X), index=X.index)
    .sort_values()
    .pipe(lambda x: [x.index[0], x.index[-1]])
)
show_text(statements.loc[idx_], lexica=lexica, n=None)
```

<IPython.core.display.HTML object>

Chapter 25

Leveraging Large Language Models to Generate Economic Expectations from Historical News and Fed Statements

This chapter introduces a novel methodology for generating economic expectations using large language models (LLMs) applied to historical news data. The central goal is to demonstrate how LLMs, specifically models like OpenAI's GPT-3.5, can process and analyze vast quantities of news articles to estimate economic sentiment and expectations.

By deriving these LLM-generated expectations, we review their usefulness in understanding broader economic sentiment and the dynamics underlying behavioral theories, such as bubbles. Furthermore, these generated expectations are systematically compared to more traditional survey-based measures, highlighting the potential and implications of leveraging advanced language models in economic forecasting and sentiment analysis.

In this notebook, we also illustrate how to use Large Language Models (LLM), in particular models by Openai to extract information for each document of a corpus. The corpus that we will use is the collection of statements by the US Federal Reserve since 1999.

25.1 Generating estimates from news

The Ghost in the Machine: Generating Beliefs with Large Language Models *

J. Leland Bybee

Yale University

First Draft: February 16, 2023

This Draft: December 9, 2023

Updated Regularly: [Latest Version](#)

The main objective of the paper is to introduce a methodology for generating economic expectations using large language models (LLMs) applied to historical news data, and to explore the implications of these generated expectations for understanding economic sentiment and behavioral theories of bubbles.

More precisely, the research uses OpenAI's GPT-3.5, to generate economic expectations from historical news articles. The generated expectations are then compared to existing survey measures to evaluate their accuracy and used to construct a measure of economic sentiment

Figure 2: Prompt Format

Here is a piece of news:

"%s"

Do you think this news will increase or decrease %s?

Write your answer as:

```
{increase/decrease/uncertain}:
{confidence (0-1)}:
{magnitude of increase/decrease (0-1)}:
{explanation (less than 25 words)}
```

Note. Reports the prompt format for queries made to GPT. "%s" indicates where in the prompt the headline and target text are inserted.

Table 1: Summary Statistics of GPT Survey

Series	Prompt	Date Range	Count	Inc. %	Dec. %	Unc. %
SNP	the S&P 500 index	1984-2021	136345	15.13	26.84	58.02
CPI	the consumer price index in the United States	1984-2021	132736	7.86	6.45	85.69
IIS	housing starts in the United States	1984-2021	132212	2.50	5.68	91.82
IP	industrial production in the United States	1984-2021	132892	10.11	11.72	78.17
DEFL	the GDP price deflator in the United States	1984-2021	132760	9.63	12.50	77.88
AAA	the AAA-rated bond's rate in the United States	1984-2021	133467	11.09	14.88	74.03
C	real consumption in the United States	1984-2021	131574	11.53	17.67	70.80
GF	federal government consumption in the United States	1984-2021	132839	9.86	10.88	79.26
GY	the real GDP of the United States	1984-2021	132148	20.54	20.91	58.56
NRI	real nonresidential investment in the United States	1984-2021	132961	17.46	22.94	59.59
RI	real residential investment in the United States	1984-2021	133157	8.66	16.49	74.85
GS	state and local government consumption in the United States	1984-2021	131428	13.44	17.30	69.26
3TB	the 3-month treasury bill rate	1984-2021	134609	15.45	11.21	73.34
UE	employment in the United States	1984-2021	120102	9.81	11.24	78.95

Note. Reports summaries of the surveyed labels, the text passed to GPT, the date range, the number of non-missing responses, the proportion of responses that were "increase", the proportion of responses that were "decrease", and the proportions that were "uncertain".

Table 2: Example Responses

Series	Direction	Headline/Response
(1989-06-29) U.S. Reaches Accords Widening Access To the Mobile Phone Business in Japan		
S&P	1	The news signals potential growth opportunities for US mobile phone companies, which could positively impact the S&P 500 index.
UE	-1	Increased access to the Japanese mobile phone market will likely create new job opportunities in the U.S. telecommunications industry.
CPI	-1	Increased competition and access to the Japanese mobile phone market may lead to lower prices for U.S. consumers.
(2001-02-28) Bush Offers Tax Cuts and Tight Budgets to Aid 'Faltering' Economy		
S&P	1	Tax cuts stimulate the economy, which could lead to increased corporate profits and higher stock prices.
UE	0	Tax cuts may stimulate investment, but tight budgets could reduce government spending and slow job growth.
CPI	1	Tax cuts usually stimulate spending, which can increase demand and raise prices, but budget tightening may counteract that effect.

Note. Reports a sample of example article headlines and the corresponding responses from GPT. The first column reports the series queried. The second column reports the direction of the query. The third column reports the headline and the corresponding response from GPT.

The study finds that generated expectations closely match existing survey measures and capture deviations from full-information rational expectations. The generated measure of economic sentiment is positively correlated with past returns and negatively correlated with future returns, indicating its potential as a predictor of market bubbles.

Figure 4: Survey Correlations with Existing Moments

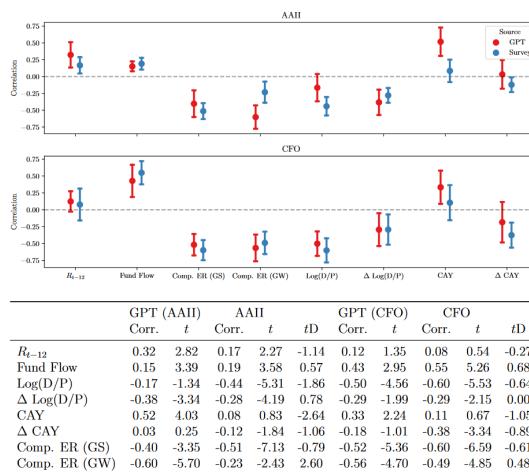
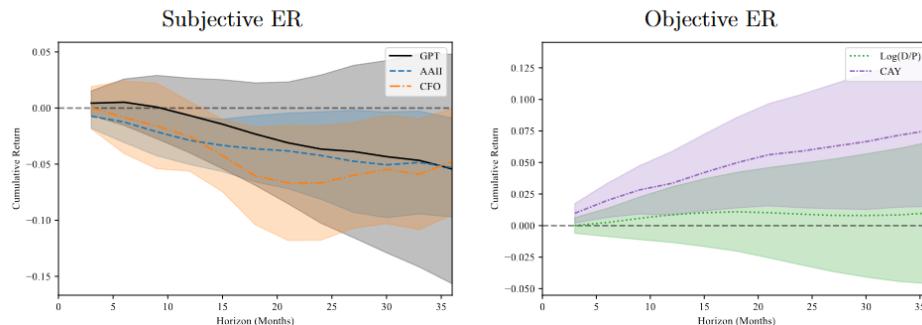


Figure 5: Predictive Return Regressions



Note. Reports the coefficients for a series of predictive regressions of future cumulative returns over the given horizon on subjective and objective expected return proxies. Shaded bands report 90% confidence intervals using Newey-West standard errors with the corresponding horizon as the number of lags.

25.2 LLM wrapper

The code below is based on the Openai package. Given that Openai only proposes commercial LLMs, you need to have created an account and been provided a key. In the cell below you can paste the key: `api_key=sk-....`. Alternatively (as done below), you can put in a file call “key.” and call the file.

The following functions in `llm.py` allows to access language models by `openai`.

```
[8]: %%writefile ../skfin/llm.py
import os
import ssl
from concurrent.futures import ThreadPoolExecutor
from functools import partial

import pandas as pd
from openai import OpenAI
from tenacity import retry, stop_after_attempt, wait_exponential
import httpx

ssl._create_default_https_context = ssl._create_unverified_context

@retry(wait=wait_exponential(multiplier=1, min=6, max=10), stop=stop_after_attempt(5))
def apply_prompt_single_text(
    text,
    content_function,
    response_format,
    nested_response=False,
    model="gpt-4o-mini",
    max_new_tokens=2048,
    temperature=0,
    api_key=None,
):
    """Apply a specified prompt to a single text input using an OpenAI GPT model, and format the response."""
    # ... (rest of the function implementation)
```

This function retries on failure with an exponential backoff strategy up to 5 attempts.

Args:

- text (str): The input text to be processed by the prompt.*
- content_func (callable): A function that takes a single string argument (text) and returns a formatted string.*
- response_format (ResponseFormat): An instance specifying the desired format of the response.*
- model (str, optional): The identifier for the model to use. Defaults to "gpt-4o-mini".*
- max_new_tokens (int, optional): The maximum number of new tokens to generate in the response. Defaults to 2048.*
- temperature (float, optional): Sampling temperature to use for the response generation. A value closer to 0 makes the output more deterministic. Defaults to 0.*
- api_key (str, optional): OpenAI key. Defaults to None, in which case, it loaded in the local folder.*

Returns:

- dict: A dictionary where keys correspond to the properties specified in the response format's JSON schema, and values are the parsed responses from the model. Returns 'NA' if no output is generated.*

Raises:

- openai.error.OpenAIError: If fails to get a response after the specified retry attempts.*

```

messages = [
    {
        "role": "system",
        "content": "Assume the role of a financial analyst.",
    },
    {
        "role": "user",
        "content": content_function(text),
    },
]
if api_key is None:
    with open("key", "r") as f:
        api_key = f.read()
client = OpenAI(api_key=api_key,
               http_client=httpx.Client(verify= os.environ.get("REQUESTS_CA_BUNDLE")))
completion = client.beta.chat.completions.parse(
    model=model,
    messages=messages,
    response_format=response_format,
    max_tokens=max_new_tokens,
    temperature=0,
    seed=42,
    logprobs=True,
    top_logprobs=10,
    top_p=1,
)
message = completion.choices[0].message
if message.parsed:
    keys = list(response_format.model_json_schema()["properties"].keys())
    output = {k: message.parsed.__dict__[k] for k in keys}
    if nested_response:
        assert len(output) == 1
        output = output[list(output.keys())[0]]
        assert isinstance(output, list)
    k, v = output[0].model_json_schema()["properties"].keys()

```

```
        output = pd.Series({c.__dict__[k]: c.__dict__[v] for c in output})
    else:
        print("No output")
        output = "NA"

    return output

def apply_prompt(
    dataframe,
    content_function,
    response_format,
    nested_response=False,
    model="gpt-4o-mini",
    text_column="text",
    max_workers=25,
    sequential=False,
    return_dataframe=True,
    stacked_dataframe=False,
    api_key=None,
):
    """
    Apply a specified prompt to a DataFrame containing text data using an OpenAI GPT model.

    This function can process the text data either sequentially or concurrently using a thread-
    pool.

    Args:
        dataframe (pd.DataFrame): The input DataFrame containing the text data to be processed.
        content_function (callable): A function that takes a single string argument (text) and_
        →returns a formatted string.
        response_format (ResponseFormat): An instance specifying the desired format of the_
        →response.
        text_columns (str, optional): The column name in the DataFrame that contains the text_
        →data. Defaults to 'text'.
        max_workers (int, optional): The maximum number of worker threads to use for concurrent_
        →execution. Defaults to 25.
        sequential (bool, optional): If True, process the text data sequentially. If False,_
        →process the data concurrently. Defaults to False.
        return_dataframe (bool, optional): If True, return the results as a new DataFrame. If_
        →False, return the results as a list. Defaults to True.
        api_key (str, optional): OpenAI key. Defaults to None, in which case, it loaded in the_
        →local folder.

    Returns:
        pd.DataFrame or list: The processed output. A DataFrame if `return_dataframe` is True,_
        →otherwise a list.
    """

    texts = dataframe[text_column].values

    if sequential:
        results = [
            apply_prompt_single_text(
                text,
                content_function=content_function,
                response_format=response_format,
                nested_response=nested_response,
                model=model,
                api_key=api_key,
            )
        ]
```

```

        for text in texts
    ]
else:
    apply_prompt_single_text_partial = partial(
        apply_prompt_single_text,
        content_function=content_function,
        response_format=response_format,
        nested_response=nested_response,
        model=model,
        api_key=api_key,
    )
    with ThreadPoolExecutor(max_workers=max_workers) as executor:
        results = list(executor.map(apply_prompt_single_text_partial, texts))

if return_dataframe:
    if stacked_dataframe:
        return pd.concat({dataframe.index[i]: pd.DataFrame(d) for i, d in
←enumerate(results)})
    else:
        return pd.DataFrame(results, index=dataframe.index)
else:
    return results

```

Overwriting .../skfin/llm.py

In the function above (and the analysis below), the baseline model is gpt-4o-mini.

25.3 Measuring hawkishness in FOMC statements

[9]: statements = load_fomc_statements(force_reload=False)

[10]: text = statements["text"].iloc[0]
text

[10]: "The Federal Reserve released the following statement after today's Federal Open Market Committee meeting: While the FOMC did not take action today to alter the stance of monetary policy, the Committee was concerned about the potential for a buildup of inflationary imbalances that could undermine the favorable performance of the economy and therefore adopted a directive that is tilted toward the possibility of a firming in the stance of monetary policy. Trend increases in costs and core prices have generally remained quite subdued. But domestic financial markets have recovered and foreign economic prospects have improved since the easing of monetary policy last fall. Against the background of already-tight domestic labor markets and ongoing strength in demand in excess of productivity gains, the Committee recognizes the need to be alert to developments over coming months that might indicate that financial conditions may no longer be consistent with containing inflation."

With the following prompt, we ask the model to qualify Fed statements in terms of hawkishness/dovishness. We also insist on receiving a single-word answer.

[11]: from skfin.llm import apply_prompt
from typing import Literal
from pydantic import BaseModel

```

def content_hawkishness(text):
    return f"""Act as a financial analyst. What is the monetary policy hawkishness of this text?

```

Chapter 25. Leveraging Large Language Models to Generate Economic Expectations from Historical News and Fed Statements

```
Please choose an answer from hawkish, dovish, neutral or unknown and provide a probability ↴and a short explanation.
```

```
Text: {text}"""
```

```
class format_hawkishness(BaseModel):
    hawkishness: Literal["hawkish", "neutral", "dovish", "unknown"]
    probability: float
    explanation: str
```

```
[12]: df = apply_prompt(
    statements,
    content_function=content_hawkishness,
    response_format=format_hawkishness,
    text_column="text",
    api_key=api_key,
)
```

```
[13]: show_text(df, text_column="explanation", n=5)
```

```
<IPython.core.display.HTML object>
```

Checking that indeed we received at most the four options. (It seems that the models rarely says “I don’t know.”)

```
[14]: df["hawkishness"].value_counts()
```

```
dovish      119
neutral      63
hawkish      45
Name: hawkishness, dtype: int64
```

```
[15]: df["probability"].value_counts()
```

```
0.85      122
0.75      68
0.70      34
0.60       1
75.00      1
0.65       1
Name: probability, dtype: int64
```

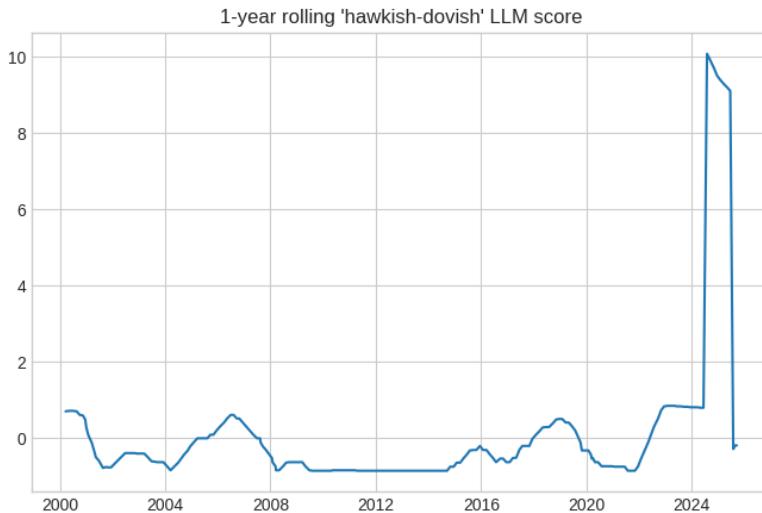
To get an intuition from the output of the LLM, we problem the cumulative difference of the dummies hawkish - dovish. We see clearly the tightening and accomodating phases of monetary policy:

- 1999 - 2001: tightening until the dot-com burst
- 2001 - 2004: loosening of monetary policy
- 2004 - 2007: tightening
- 2007 - 2017: long experiment with loosening, including with Quantitative Easing
- 2017 - 2019: Quantitative tightening
- 2019 - 2021: economic slowdown (2019) preceding the Covid intervention
- 2022 - today: tightening due to high inflation

The picture below shows hawkish-dovish LLM score with a rolling 1-year (= 8 FOMC meetings).

```
[16]: statements_ = statements.join(
    pd.get_dummies(df["hawkishness"], dtype=float).mul(df["probability"], axis=0)
).join(df["explanation"])
```

```
line(
    statements_.pipe(lambda x: x["hawkish"].sub(x["dovish"])).rolling(window=8).mean(),
    legend=False,
    title="1-year rolling 'hawkish-dovish' LLM score",
)
```



25.4 Explaining hawkishness

Beyond the simple validation of the LLM score in the previous section, we can go deeper and identify the words associated with hawkishness (or dovishness) as interpreted by the language model. To do so, we run a regression where the target is the LLM score and the features are tfidf values for the main tokens.

```
[17]: from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import ElasticNet
from sklearn.pipeline import Pipeline
```

```
[18]: est = Pipeline(
    [
        (
            "tfidf",
            TfidfVectorizer(
                vocabulary=None,
                ngram_range=(1, 3),
                max_features=500,
                stop_words="english",
                token_pattern=r"\b[a-zA-Z]{3,}\b",
            ),
        ),
        ("reg", ElasticNet(alpha=0.0075)),
    ]
)
```

```
[19]: X = statements_[ "text" ]
interpret_coefs = {}
for c in [ "hawkish", "neutral", "dovish" ]:
```

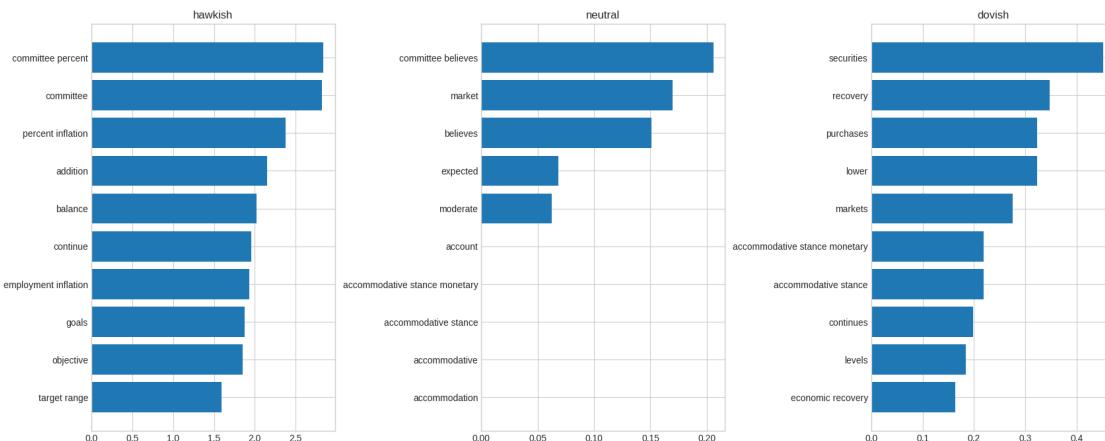
```

y = statements_[c]
est.fit(X, y)
vocab_ = pd.Series(est.named_steps["tfidf"].vocabulary_).sort_values().index
interpret_coefs[c] = pd.Series(
    np.transpose(est.named_steps["reg"].coef_), index=vocab_
)
d = {k: v.nlargest(n=10) for k, v in interpret_coefs.items()}

```

Words associated to tightening are on the hawkish side (“raise target”, “pressures”, “inflation”, etc). A bit more mixed results on the other side.

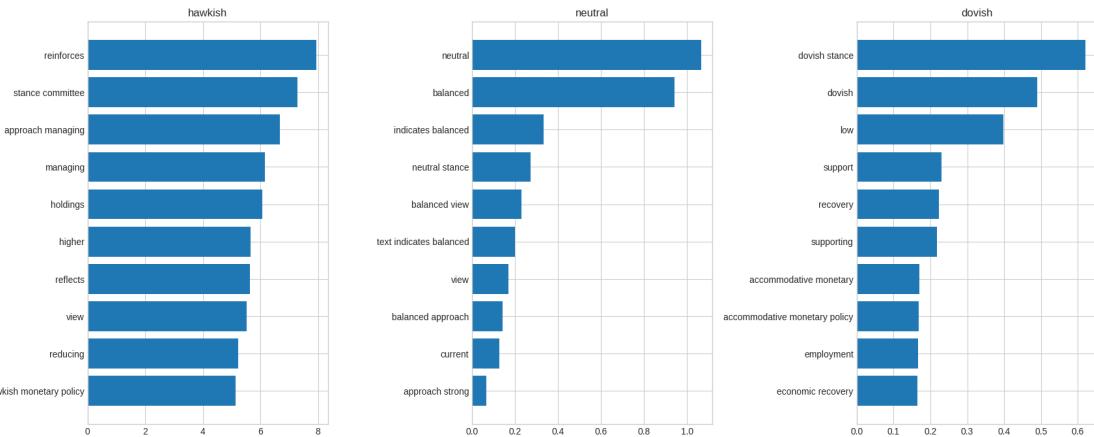
```
[20]: fig, ax = plt.subplots(1, 3, figsize=(20, 8))
fig.subplots_adjust(wspace=0.6)
for i, (k, v) in enumerate(d.items()):
    bar(v, horizontal=True, ax=ax[i], title=k)
```



Instead of using the actual text, we can use the explanation provided by chatGPT.

```
[21]: X = statements_[ "explanation" ]
interpret_coefs = {}
for c in [ "hawkish", "neutral", "dovish" ]:
    y = statements_[c]
    est.fit(X, y)
    vocab_ = pd.Series(est.named_steps["tfidf"].vocabulary_).sort_values().index
    interpret_coefs[c] = pd.Series(
        np.transpose(est.named_steps["reg"].coef_), index=vocab_
    )
d = {k: v.nlargest(n=10) for k, v in interpret_coefs.items()}

fig, ax = plt.subplots(1, 3, figsize=(20, 8))
fig.subplots_adjust(wspace=0.6)
for i, (k, v) in enumerate(d.items()):
    bar(v, horizontal=True, ax=ax[i], title=k)
```



25.5 Topics in FOMC statements

```
[22]: def content_topics(text):
    return f"""Please assess the importance for each of the following topics in the text below:
    inflation, employment, economic growth, financial stability, interest rates and the yield_
    curve, fiscal policy, consumer confidence, and market expectations.
    Provide a score between 0 and 1 for each.

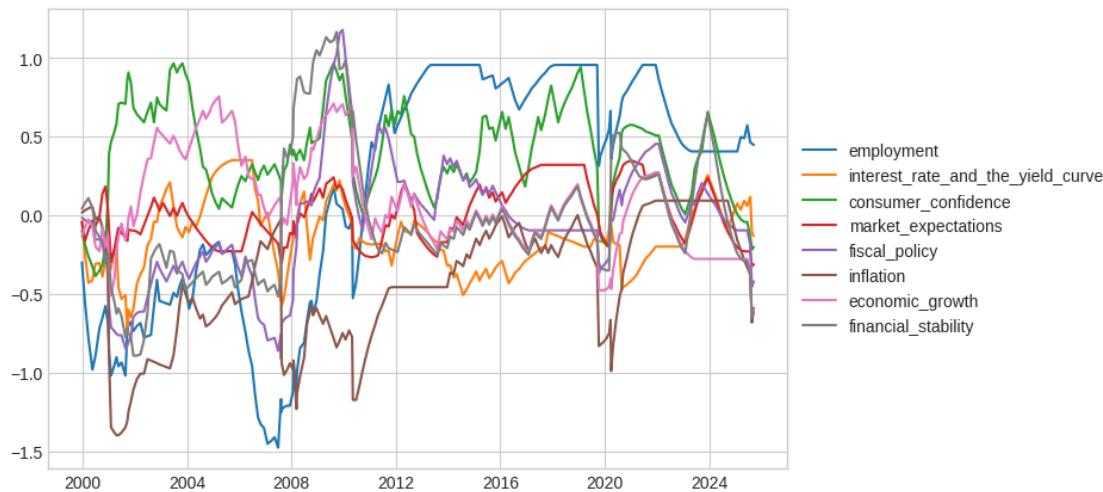
Text: {text}"""

    
```

```
class format_topics(BaseModel):
    inflation: float
    employment: float
    economic_growth: float
    financial_stability: float
    interest_rate_and_the_yield_curve: float
    fiscal_policy: float
    consumer_confidence: float
    market_expectations: float
```

```
[23]: df2 = apply_prompt(
    statements,
    content_function=content_topics,
    response_format=format_topics,
    text_column="text",
    api_key=api_key,
)
```

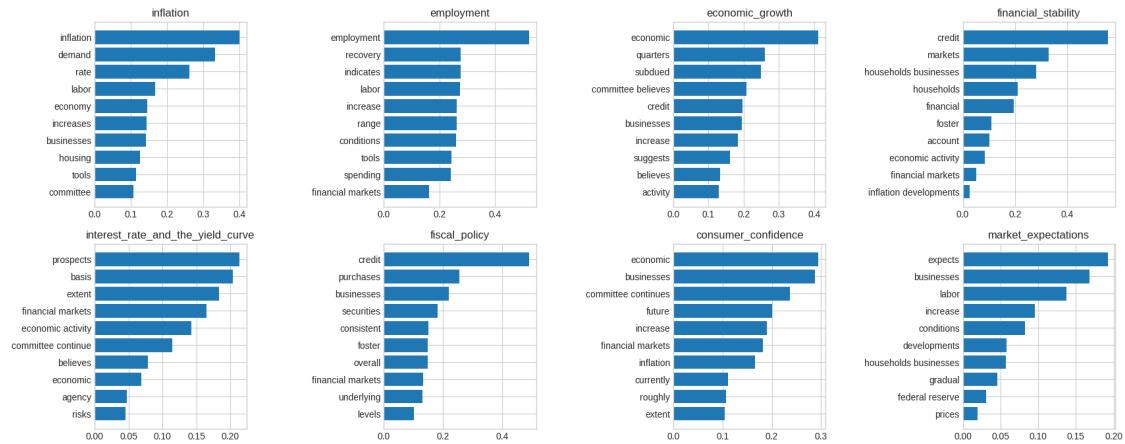
```
[24]: line(
    df2.pipe(lambda x: x.sub(x.rolling(window=12, min_periods=6).mean())),
    cumsum=True,
    legend_sharpe_ratio=False,
)
```



```
[25]: est_ = Pipeline(
    [
        (
            "tfidf",
            TfidfVectorizer(
                vocabulary=None,
                ngram_range=(1, 3),
                max_features=500,
                stop_words="english",
                token_pattern=r"\b[a-zA-Z]{3,}\b",
            ),
        ),
        ("reg", ElasticNet(alpha=5 * 1e-4)),
    ]
)
```

```
[26]: statements2_ = statements.join(df2)
X = statements2_[["text"]]
interpret_coefs = {}
for c in df2.columns:
    y = statements2_[c]
    est_.fit(X, y)
    vocab_ = pd.Series(est_.named_steps["tfidf"].vocabulary_).sort_values().index
    interpret_coefs[c] = pd.Series(
        np.transpose(est_.named_steps["reg"].coef_), index=vocab_
    )
d = {k: v.nlargest(n=10) for k, v in interpret_coefs.items()}

fig, ax = plt.subplots(2, 4, figsize=(20, 8))
ax = ax.ravel()
fig.subplots_adjust(wspace=0.9)
for i, (k, v) in enumerate(d.items()):
    bar(v, horizontal=True, ax=ax[i], title=k)
```



25.6 Predicting returns with FOMC statements

```
[27]: def content_returns(text):
    return f"""Here is a piece of news: {text}.
Do you think this news will increase or decrease the S&P500? Write your answer as:
- impact: increase/decrease/uncertain:
- confidence (0-1):
- magnitude of impact (0-1):
- explanation (less than 25 words)
"""

```

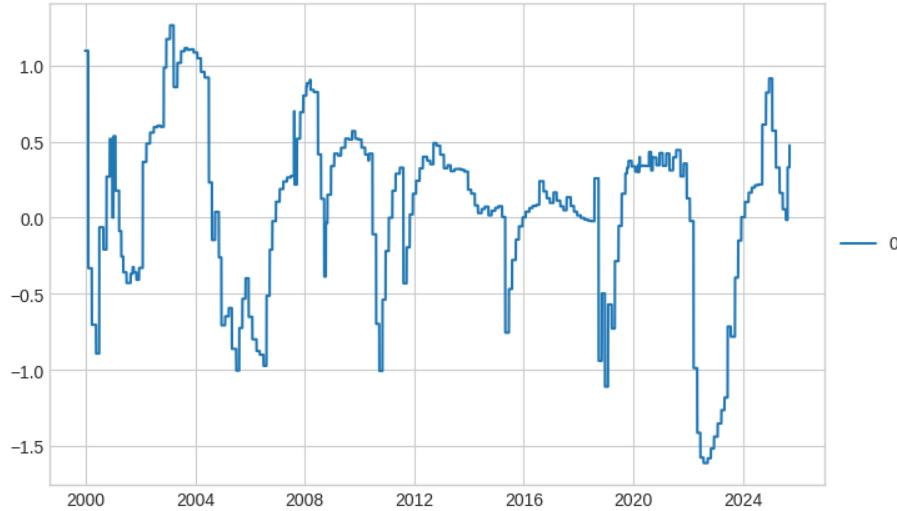
```
class format_returns(BaseModel):
    sp_impact: Literal["increase", "decrease", "uncertain"]
    confidence: float
    magnitude: float
    explanation: str
```

```
[28]: df3 = apply_prompt(
    statements,
    content_function=content_returns,
    response_format=format_returns,
    text_column="text",
    api_key=api_key,
)
```

```
[29]: raw_alpha = (
    pd.get_dummies(df3["sp_impact"], dtype=float)
    .mul(df3["confidence"] * df3["magnitude"], axis=0)
    .pipe(lambda x: x["increase"] - x["decrease"])
    .resample("B")
    .last()
)
```

```
[30]: alpha = (
    raw_alpha.pipe(lambda x: x.sub(x.ewm(halflife=6, ignore_na=True).mean()))
    .pipe(lambda x: x.div(x.ewm(12, min_periods=6, ignore_na=True).std()))
    .ewm(halflife=63)
    .mean()
)
```

```
line(alpha)
```



```
[31]: from skfin.backtesting import Backtester
from skfin.mv_estimators import TimingMeanVariance
from skfin.datasets_ import load_kf_returns

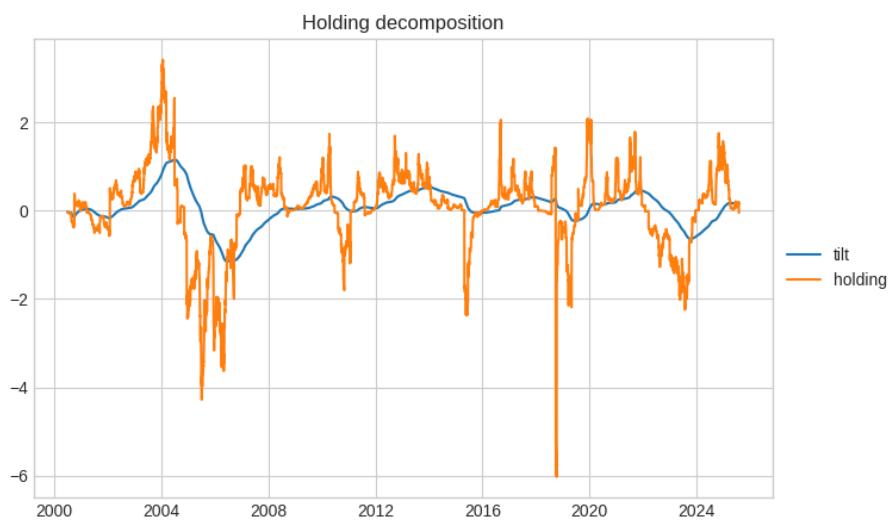
def transform_y(df):
    return df.shift(-2)

ret = (
    load_kf_returns(filename="F-F_Research_Data_Factors_daily")["Daily"]
    .resample("B")
    .last()
)
data = ret.join(alpha.rename("alpha"), how="inner").fillna(0)
```

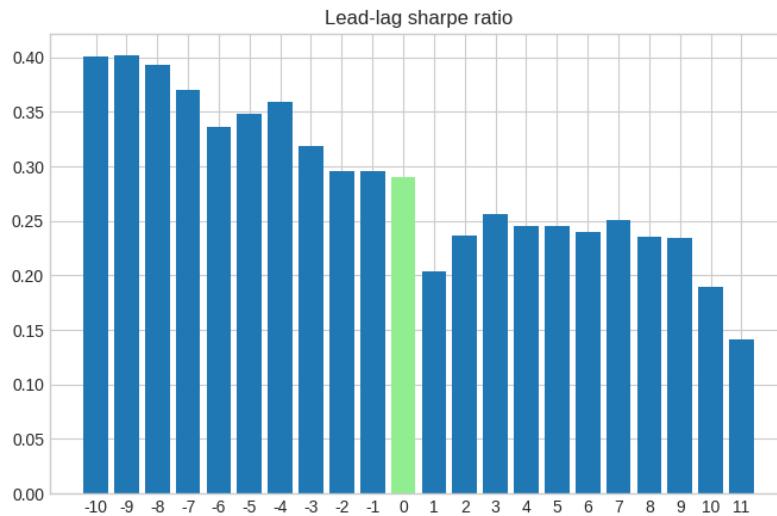
```
[32]: ret_ = data["Mkt-RF"]
y = transform_y(data["Mkt-RF"])
m = Backtester(estimator=TimingMeanVariance(), name="gpt", start_date="2000-06-30")
pnl_ = m.train(data["alpha"], y, ret_)
line(pnl_, cumsum=True)
```



```
[33]: line(
    {"holding": m.h_, "tilt": m.h_.ewm(halflife=252).mean()},
    title="Holding decomposition",
)
```



```
[34]: sr = {i: m.h_.shift(1 + i).mul(ret_).pipe(sharpe_ratio) for i in range(-10, 12)}
bar(sr, baseline=0, sort=False, title="Lead-lag sharpe ratio")
```



```
[35]: line(
    {
        "ALL": pnl_,
        "tilt": m.h_.ewm(halflife=12).mean().shift(1).mul(ret_).dropna(),
        "timing": m.h_.sub(m.h_.ewm(halflife=12).mean()).shift(1).mul(ret_).dropna(),
    },
    cumsum=True,
    title="Tilt/timing decomposition",
)
```



```
[36]: line(
    {
        "ALL": pnl_,
        "long": m.h_.clip(lower=0).shift(1).mul(ret_).dropna(),
        "short": m.h_.clip(upper=0).shift(1).mul(ret_).dropna(),
    },
    cumsum=True,
```

```
    title="Long/short decomposition",  
)
```



Chapter 26

Conclusion

There are several packages that are natural extensions of the ideas described here:

- jax: <https://github.com/google/jax>
- cvxpy: <https://github.com/cvxpy/cvxpy>
- optuna: <https://github.com/optuna/optuna> or <https://github.com/automl/auto-sklearn>

Several recent significant progress were made in Natural Language Processing, Computer Vision, Audio, and Graph Neural Networks.

Moreover, similar analyses could be extended to several asset classes (for instance by leveraging yfinance: <https://github.com/ranaroussi/yfinance> or pandas-datareader: <https://github.com/pydata/pandas-datareader>):

- options
- futures contracts (including commodity futures)
- non-US stocks (e.g. Chinese stocks)
- bitcoin and other crypto currencies

Chapter 27

Appendix: Custom Helper Functions for Data Exploration and Visualization

Here we create some helper functions that will be used across notebooks using the magic `%%writefile`.

27.1 Data visualisation

Data exploration, in particular based on visualisation, is crucial to modern data science. Pandas has a lot of plotting functionalities (e.g. see the graph below), but we will find it usefull to use a custom plot set of functions.

```
[2]: %%writefile ./skfin/plot.py
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from skfin.metrics import sharpe_ratio

plt.style.use("seaborn-whitegrid")

def set_axis(ax=None, figsize=(8, 5), title=None, fig=None):
    if ax is None:
        fig, ax = plt.subplots(1, 1, figsize=figsize)
    if title is not None:
        ax.set_title(title)
    return fig, ax

def line(
    df,
    sort=True,
    figsize=(8, 5),
    ax=None,
    title="",
    cumsum=False,
    loc="center left",
    bbox_to_anchor=(1, 0.5),
    legend_sharpe_ratio=None,
    legend=True,
    yscale=None,
```

```

        start_date=None,
):
    df = df.copy()
    if loc == "best":
        bbox_to_anchor = None
    if isinstance(df, dict) | isinstance(df, list):
        df = pd.concat(df, axis=1)
    if isinstance(df, pd.Series):
        df = df.to_frame()
    if start_date is not None:
        df = df[start_date:]
    if cumsum & (legend_sharpe_ratio is None):
        legend_sharpe_ratio = True
    if legend_sharpe_ratio:
        df.columns = [f"{c}: sr={sharpe_ratio(df[c]): 3.2f}" for c in df.columns]
    if cumsum:
        df = df.cumsum()
    if sort:
        df = df.loc[:, lambda x: x.iloc[-1].sort_values(ascending=False).index]
    if ax is None:
        fig, ax = set_axis(ax=ax, figsize=figsize)
    if title != '':
        ax.set_title(title)
    ax.plot(df.index, df.values)
    if legend:
        ax.legend(df.columns, loc=loc, bbox_to_anchor=bbox_to_anchor)
    if yscale == "log":
        ax.set_yscale("log")

def bar(
    df,
    err=None,
    sort=True,
    figsize=(8, 5),
    ax=None,
    title=None,
    horizontal=False,
    baseline=None,
    rotation=0,
):
    if isinstance(df, pd.DataFrame):
        df = df.squeeze()
    if isinstance(df, dict):
        df = pd.Series(df)
    if sort:
        df = df.sort_values()
    if err is not None:
        err = err.loc[df.index]
    labels = df.index
    x = np.arange(len(labels))
    fig, ax = set_axis(ax=ax, figsize=figsize, title=title)
    if horizontal:
        ax.barh(x, df.values, xerr=err, capsize=5)
        ax.set_yticks(x)
        ax.set_yticklabels(labels, rotation=0)
        if baseline in df.index:
            df_ = df.copy()
            df_[df.index != baseline] = 0
            ax.barh(x, df_.values, color="lightgreen")
    else:

```

```

        ax.bar(x, df.values, yerr=err, capsize=5)
        ax.set_xticks(x)
        ax.set_xticklabels(labels, rotation=0)
        if baseline in df.index:
            df_ = df.copy()
            df_[df.index != baseline] = 0
            ax.bar(x, df_.values, color="lightgreen")
    ax.set_title(title)

def heatmap(
    df,
    ax=None,
    fig=None,
    figsize=(8, 5),
    title=None,
    vmin=None,
    vmax=None,
    vcompute=True,
    cmap="RdBu",
):
    labels_x = df.index
    x = np.arange(len(labels_x))
    labels_y = df.columns
    y = np.arange(len(labels_y))
    if vcompute:
        vmax = df.abs().max().max()
        vmin = -vmax
    fig, ax = set_axis(ax=ax, figsize=figsize, title=title, fig=fig)
    pos = ax.imshow(
        df.T.values, cmap=cmap, interpolation="nearest", vmax=vmax, vmin=vmin
    )
    ax.set_xticks(x)
    ax.set_yticks(y)
    ax.set_xticklabels(labels_x, rotation=90)
    ax.set_yticklabels(labels_y)
    ax.grid(True)
    fig.colorbar(pos, ax=ax)

def scatter(
    df,
    ax=None,
    xscale=None,
    yscale=None,
    xlabel=None,
    ylabel=None,
    xticks=None,
    yticks=None,
    figsize=(8, 5),
    title=None,
):
    fig, ax = set_axis(ax=ax, figsize=figsize, title=title)
    ax.scatter(df, df.index, facecolors="none", edgecolors="b", s=50)
    if xlabel is not None:
        ax.set_xlabel(xlabel)
    if ylabel is not None:
        ax.set_ylabel(ylabel)
    if xscale is not None:
        ax.set_xscale(xscale)
    if yscale is not None:

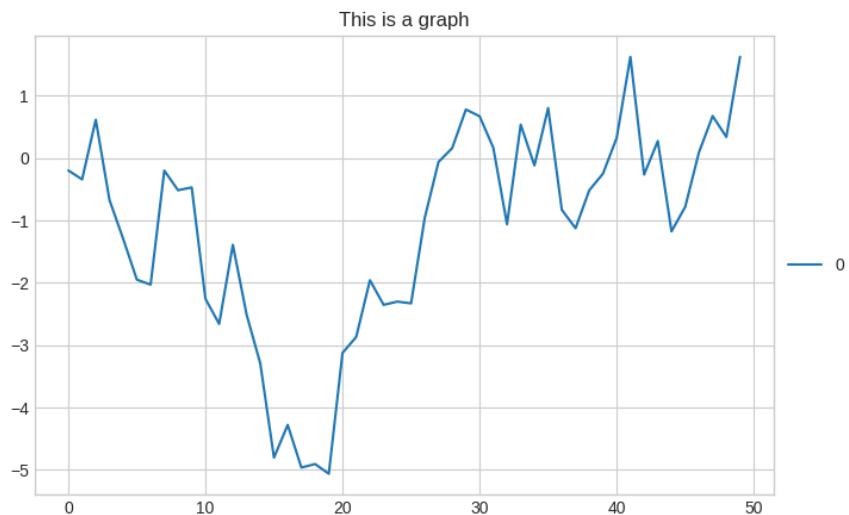
```

```
    ax.set_yscale(yscale)
if yticks is not None:
    ax.set_yticks(yticks)
    ax.set_yticklabels(yticks)
if xticks is not None:
    ax.set_xticks(xticks)
    ax.set_xticklabels(xticks)
```

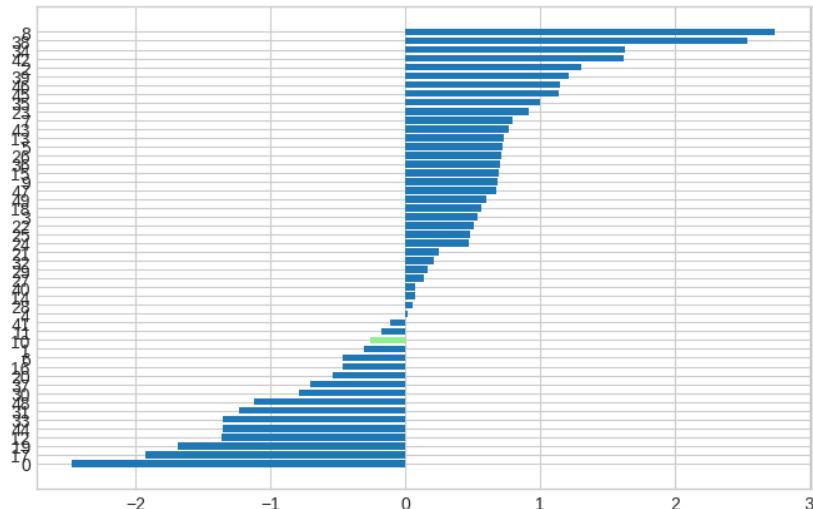
Overwriting .../skfin/plot.py

```
[3]: from skfin.plot import bar, heatmap, line
```

```
[4]: line(
    pd.Series(np.random.normal(size=50)),
    cumsum=True,
    title="This is a graph",
    legend_sharpe_ratio=False,
)
```



```
[5]: bar(pd.Series(np.random.normal(size=50)), baseline=10, horizontal=True)
```



27.2 Dates and mappings

```
[6]: %%writefile ../skfin/dataloaders/constants/mappings.py
"""
Mapping dictionaries for various datasets and symbols.
"""

symbol_dict = {
    "TOT": "Total",
    "XOM": "Exxon",
    "CVX": "Chevron",
    "COP": "ConocoPhillips",
    "VLO": "Valero Energy",
    "MSFT": "Microsoft",
    "IBM": "IBM",
    "TWX": "Time Warner",
    "CMCSA": "Comcast",
    "CVC": "Cablevision",
    "YHOO": "Yahoo",
    "DELL": "Dell",
    "HPQ": "HP",
    "AMZN": "Amazon",
    "TM": "Toyota",
    "CAJ": "Canon",
    "SNE": "Sony",
    "F": "Ford",
    "HMC": "Honda",
    "NAV": "Navistar",
    "NOC": "Northrop Grumman",
    "BA": "Boeing",
    "KO": "Coca Cola",
    "MMM": "3M",
    "MCD": "McDonald's",
    "PEP": "Pepsi",
    "K": "Kellogg",
    "UN": "Unilever",
    "MAR": "Marriott",
    "PG": "Procter Gamble",
    "CL": "Colgate-Palmolive",
    "GE": "General Electrics",
    "WFC": "Wells Fargo",
    "JPM": "JPMorgan Chase",
    "AIG": "AIG",
    "AXP": "American express",
    "BAC": "Bank of America",
    "GS": "Goldman Sachs",
    "AAPL": "Apple",
    "SAP": "SAP",
    "CSCO": "Cisco",
    "TXN": "Texas Instruments",
    "XRX": "Xerox",
    "WMT": "Wal-Mart",
    "HD": "Home Depot",
    "GSK": "GlaxoSmithKline",
    "PFE": "Pfizer",
    "SNY": "Sanofi-Aventis",
    "NVS": "Novartis",
}
```

```

    "KMB": "Kimberly-Clark",
    "R": "Ryder",
    "GD": "General Dynamics",
    "RTN": "Raytheon",
    "CVS": "CVS",
    "CAT": "Caterpillar",
    "DD": "DuPont de Nemours",
}

mapping_10X = {
    "AAPL": ["APPLE COMPUTER INC", "APPLE INC"],
    "AIG": "AMERICAN INTERNATIONAL GROUP INC",
    "AMZN": "AMAZON COM INC",
    "AXP": "AMERICAN EXPRESS CO",
    "BA": "BOEING CO",
    "BAC": "BANK OF AMERICA CORP /DE/",
    "CAT": "CATERPILLAR INC",
    "CL": "COLGATE PALMOLIVE CO",
    "CMCSA": "COMCAST CORP",
    "COP": "CONOCOPHILLIPS",
    "CSCO": "CISCO SYSTEMS INC",
    "CVC": "CABLEVISION SYSTEMS CORP /NY",
    "CVS": ["CVS CORP", "CVS/CAREMARK CORP", "CVS CAREMARK CORP"],
    "CVX": ["CHEVRONTEXACO CORP", "CHEVRON CORP"],
    "DD": "DUPONT E I DE NEMOURS & CO",
    "DELL": ["DELL COMPUTER CORP", "DELL INC"],
    "F": "FORD MOTOR CO",
    "GD": "GENERAL DYNAMICS CORP",
    "GE": "GENERAL ELECTRIC CO",
    "GS": "GOLDMAN SACHS GROUP INC/",
    "HD": "HOME DEPOT INC",
    "HPQ": "HEWLETT PACKARD CO",
    "IBM": "INTERNATIONAL BUSINESS MACHINES CORP",
    "JPM": "J P MORGAN CHASE & CO",
    "K": "KELLOGG CO",
    "KMB": "KIMBERLY CLARK CORP",
    "KO": "COCA COLA CO",
    "MAR": "MARRIOTT INTERNATIONAL INC /MD/",
    "MCD": "MCDONALDS CORP",
    "MMM": "3M CO",
    "MSFT": "MICROSOFT CORP",
    "NAV": "NAVISTAR INTERNATIONAL CORP",
    "NOC": "NORTHROP GRUMMAN CORP /DE/",
    "PEP": "PEPSI BOTTLING GROUP INC",
    "PFE": "PFIZER INC",
    "PG": "PROCTER & GAMBLE CO",
    "R": "RYDER SYSTEM INC",
    "RTN": "RAYTHEON CO/",
    "TWX": ["AOL TIME WARNER INC", "TIME WARNER INC"],
    "TXN": "TEXAS INSTRUMENTS INC",
    "VLO": "VALERO ENERGY CORP/TX",
    "WFC": "WELLS FARGO & CO/MN",
    "WMT": "WAL MART STORES INC",
    "XOM": "EXXON MOBIL CORP",
    "XRX": "XEROX CORP",
    "YHOO": "YAHOO INC",
}
}

```

Overwriting ../skfin/dataloaders/constants/mappings.py

```
[7]: %%writefile ../skfin/dataloaders/constants/dates.py
import pandas as pd

def load_fomc_change_date(as_datetime=True):
    """
    Return the dates of FOMC interest rate changes.

    Args:
        as_datetime: If True, convert dates to datetime format

    Returns:
        Tuple of two lists: (increase dates, decrease dates)
    """
    change_up = [
        "1999-06-30",
        "1999-08-24",
        "1999-11-16",
        "2000-02-02",
        "2000-03-21",
        "2000-05-16",
        "2004-06-30",
        "2004-08-10",
        "2004-09-21",
        "2004-11-10",
        "2004-12-14",
        "2005-02-02",
        "2005-03-22",
        "2005-05-03",
        "2005-06-30",
        "2005-08-09",
        "2005-09-20",
        "2005-11-01",
        "2005-12-13",
        "2006-01-31",
        "2006-03-28",
        "2006-05-10",
        "2006-06-29",
        "2015-12-16",
        "2016-12-14",
        "2017-03-15",
        "2017-06-14",
        "2017-12-13",
        "2018-03-21",
        "2018-06-13",
        "2018-09-26",
        "2018-12-19",
        "2022-03-16",
        "2022-05-04",
        "2022-06-15",
        "2022-07-27",
    ]
    change_dw = [
        "2001-01-03",
        "2001-01-31",
        "2001-03-20",
        "2001-04-18",
        "2001-05-15",
        "2001-06-27",
        "2001-08-21",
    ]
```

```
"2001-09-17",
"2001-10-02",
"2001-11-06",
"2001-12-11",
"2002-11-06",
"2003-06-25",
"2007-09-18",
"2007-10-31",
"2007-12-11",
"2008-01-22",
"2008-01-30",
"2008-03-18",
"2008-04-30",
"2008-10-08",
"2008-10-29",
"2008-12-16",
"2019-07-31",
"2019-09-18",
"2019-10-30",
"2020-03-03",
"2020-03-15",
]
if as_datetime:
    change_up, change_dw = pd.to_datetime(change_up), pd.to_datetime(change_dw)

return change_up, change_dw

def load_us_politics_dates():
    us_politics = {
        "03-1925": {
            "Presidency": "Republican",
            "House": "Republican",
            "Senate": "Republican",
        },
        "03-1933": {
            "Presidency": "Democratic",
            "House": "Democratic",
            "Senate": "Democratic",
        },
        "01-1947": {
            "Presidency": "Democratic",
            "House": "Republican",
            "Senate": "Republican",
        },
        "01-1949": {
            "Presidency": "Democratic",
            "House": "Democratic",
            "Senate": "Democratic",
        },
        "01-1953": {
            "Presidency": "Republican",
            "House": "Republican",
            "Senate": "Republican",
        },
        "01-1955": {
            "Presidency": "Republican",
            "House": "Democratic",
            "Senate": "Democratic",
        },
        "01-1961": {
```

```

    "Presidency": "Democratic",
    "House": "Democratic",
    "Senate": "Democratic",
  },
  "01-1969": {
    "Presidency": "Republican",
    "House": "Democratic",
    "Senate": "Democratic",
  },
  "01-1981": {
    "Presidency": "Republican",
    "House": "Democratic",
    "Senate": "Republican",
  },
  "01-1987": {
    "Presidency": "Republican",
    "House": "Democratic",
    "Senate": "Democratic",
  },
  "01-1993": {
    "Presidency": "Democratic",
    "House": "Democratic",
    "Senate": "Democratic",
  },
  "01-1995": {
    "Presidency": "Democratic",
    "House": "Republican",
    "Senate": "Republican",
  },
  "01-2001": {
    "Presidency": "Republican",
    "House": "Republican",
    "Senate": "50-50 Split",
  },
  "06-2001": {
    "Presidency": "Republican",
    "House": "Republican",
    "Senate": "Democratic",
  },
  "01-2003": {
    "Presidency": "Republican",
    "House": "Republican",
    "Senate": "Republican",
  },
  "01-2007": {
    "Presidency": "Republican",
    "House": "Democratic",
    "Senate": "Democratic",
  },
  "01-2009": {
    "Presidency": "Democratic",
    "House": "Democratic",
    "Senate": "Democratic",
  },
  "01-2011": {
    "Presidency": "Democratic",
    "House": "Republican",
    "Senate": "Democratic",
  },
  "01-2015": {
    "Presidency": "Democratic",
    "House": "Democratic",
    "Senate": "Democratic",
  },
  "01-2017": {
    "Presidency": "Democratic",
    "House": "Democratic",
    "Senate": "Democratic",
  }
}

```

```

        "House": "Republican",
        "Senate": "Republican",
    },
    "01-2017": {
        "Presidency": "Republican",
        "House": "Republican",
        "Senate": "Republican",
    },
    "01-2019": {
        "Presidency": "Republican",
        "House": "Democratic",
        "Senate": "Republican",
    },
    "01-2021": {
        "Presidency": "Democratic",
        "House": "Democratic",
        "Senate": "Democratic",
    },
}
return pd.DataFrame.from_dict(us_politics, orient="index").pipe(
    lambda d: d.set_index(pd.to_datetime(d.index, format="%m-%Y"))
)

```

Overwriting .../skfin/dataloaders/constants/dates.py

27.3 Data utils

```
[8]: %%writefile ../skfin/dataloaders/cache.py
import logging
import sys
from pathlib import Path
from typing import Any, Union, Callable

import pandas as pd

from skfin.dataloaders.io_utils import clean_directory_path, load_dict, save_dict

logging.basicConfig(stream=sys.stdout, level=logging.CRITICAL)
logger = logging.getLogger(__name__)

class CacheManager:
    """Manages loading and saving data to cache."""

    def __init__(self, cache_dir: str = "data"):
        self.cache_dir = clean_directory_path(cache_dir)
        self.logger = logging.getLogger(__name__)

    def get_cached_dataframe(
        self,
        filename: Union[str, Path],
        loader_func: Callable,
        force_reload: bool = False,
        **kwargs,
    ) -> Any:
        """
        Load a DataFrame from cache or get it via a loader function.

        Args:
            ...
        """

```

```

filename: Cache filename
loader_func: Function to call to load data if not present in cache
force_reload: If True, ignore cache and reload data
**kwargs: Additional arguments to pass to loader_func

>Returns:
    Loaded DataFrame or other object
"""

if isinstance(filename, str):
    filename = Path(filename)

full_path = self.cache_dir / filename

if (full_path.exists()) & (not force_reload):
    self.logger.info(f"Loading from cache : {full_path}")
    if filename.suffix == ".parquet":
        return pd.read_parquet(full_path)
    elif filename.suffix == ".csv":
        return pd.read_csv(full_path)
    elif filename.suffix == ".xlsx":
        if "sheet_name" in kwargs:
            return pd.read_excel(full_path, sheet_name=kwargs["sheet_name"])
        else:
            return pd.read_excel(full_path)
    else:
        return load_dict(full_path)
else:
    self.logger.info("Loading from an external source")
    data = loader_func(**kwargs)
    self.save_to_cache(data, full_path)
    return data

def save_to_cache(self, data: Any, path: Path) -> None:
    """Saving to cache."""
    self.logger.info(f"Saving to the cache : {path}")
    if isinstance(data, pd.DataFrame):
        if path.suffix == ".parquet":
            data.to_parquet(path)
        elif path.suffix == ".csv":
            data.to_csv(path)
    else:
        save_dict(data, path)

```

Overwriting ../skfin/dataloaders/cache.py

```

[9]: %%writefile ../skfin/dataloaders/cleaners.py
from typing import Dict, List, Optional

import numpy as np
import pandas as pd
from bs4 import BeautifulSoup

class DataCleaner:
    """Utilities for data cleaning."""

    @staticmethod
    def clean_kf_dataframes(df: pd.DataFrame, multi_df: bool = False) -> Dict:
        """Extract annual and monthly dataframes from the CSV file with specific formatting."""
        idx = [-2] + list(np.where(df.notna().sum(axis=1) == 0)[0])

```

```

if multi_df:
    cols = [" Average Value Weighted Returns -- Monthly"] + list(
        df.loc[df.notna().sum(axis=1) == 0].index
    )
returns_data = {"Annual": {}, "Monthly": {}}
for i in range(len(idx)):
    if multi_df:
        c_ = (
            cols[i]
            .replace("-- Annual", "")
            .replace("-- Monthly", "")
            .strip()
            .replace("/", " ")
            .replace(" ", "_")
        )
    if i != len(idx) - 1:
        v = df.iloc[idx[i] + 2 : idx[i + 1] - 1].astype(float)
        v.index = v.index.str.strip()
        if len(v) != 0:
            if len(v.index[0]) == 6:
                v.index = pd.to_datetime(v.index, format="%Y%m")
            if multi_df:
                returns_data["Monthly"][c_] = v
            else:
                returns_data["Monthly"] = v
            continue
        if len(v.index[0]) == 4:
            v.index = pd.to_datetime(v.index, format="%Y")
            if multi_df:
                returns_data["Annual"][c_] = v
            else:
                returns_data["Annual"] = v
    return returns_data

@staticmethod
def sent_cleaner(s: str) -> str:
    """Clean a text string by removing line breaks and excess spaces."""
    return s.replace("\n", " ").replace("\r", " ").replace("\t", " ").strip()

@staticmethod
def bs_cleaner(
    bs: BeautifulSoup, html_tag_blocked: Optional[List[str]] = None
) -> List[str]:
    """Extract text from a BeautifulSoup object excluding certain HTML tags."""
    if html_tag_blocked is None:
        html_tag_blocked = [
            "style",
            "script",
            "[document]",
            "meta",
            "a",
            "span",
            "label",
            "strong",
            "button",
            "li",
            "h0",
            "font",
            "h1",
            "h2",
            "h3",
        ]

```

```

        "h5",
        "h4",
        "em",
        "body",
        "head",
        "sup",
    ]
return [
    DataCleaner.sent_cleaner(t)
    for t in bs.find_all(text=True)
    if (t.parent.name not in html_tag_blocked)
    & (len(DataCleaner.sent_cleaner(t)) > 0)
]

```

Overwriting ../skfin/dataloaders/cleaners.py

```

[10]: %%writefile ../skfin/dataloaders/fomc.py
import re
from typing import List, Optional

import numpy as np
import pandas as pd
import requests
from bs4 import BeautifulSoup


class FomcUtils:
    """Utilities for processing FOMC data."""

    regexp = re.compile(r"\s+", re.UNICODE)

    @staticmethod
    def get_fomc_urls(
        from_year: int = 1999, switch_year: Optional[int] = None
    ) -> List[str]:
        """Get URLs of FOMC statements from a specific year."""
        if switch_year is None:
            from datetime import datetime

            switch_year = datetime.now().year - 5

        calendar_url = "https://www.federalreserve.gov/monetarypolicy/fomccalendars.htm"
        r = requests.get(calendar_url)
        soup = BeautifulSoup(r.text, "html.parser")
        contents = soup.find_all(
            "a", href=re.compile("^/newsevents/pressreleases/monetary\d{8}[ax].htm$")
        )
        urls_ = [content.attrs["href"] for content in contents]

        for year in range(from_year, switch_year):
            yearly_contents = []
            fomc_yearly_url = f"https://www.federalreserve.gov/monetarypolicy/
            ↪fomchistorical{year}.htm"
            r_year = requests.get(fomc_yearly_url)
            soup_yearly = BeautifulSoup(r_year.text, "html.parser")
            yearly_contents = soup_yearly.findAll("a", text="Statement")
            for yearly_content in yearly_contents:
                urls_.append(yearly_content.attrs["href"])

        urls = ["https://www.federalreserve.gov" + url for url in urls_]
        return urls

```

```

@staticmethod
def feature_extraction(
    corpus: List, sent_filters: Optional[List[str]] = None
) -> pd.DataFrame:
    """Extract features from FOMC statements."""
    if sent_filters is None:
        sent_filters = [
            "Board of Governors",
            "Federal Reserve System",
            "20th Street and Constitution Avenue N.W., Washington, DC 20551",
            "Federal Reserve Board - Federal Reserve issues FOMC statement",
            "For immediate release",
            "Federal Reserve Board - FOMC statement",
            "DO NOT REMOVE: Wireless Generation",
            "For media inquiries",
            "or call 202-452-2955.",
            "Voting",
            "For release at",
            "For immediate release",
            "Last Update",
            "Last update",
        ]
    else:
        sent_filters = set(sent_filters)

    corpus = [c if "Release Date: " in c[1] else c[9:] for c in corpus]

    text = [
        " ".join([
            FomcUtils.regexp.sub(" ", s)
            for i, s in enumerate(c)
            if (i > 1) & np.all([q not in s for q in sent_filters])
        ])
        for c in corpus
    ]

    release_date = [
        pd.to_datetime(c[1].replace("Release Date: ", "")) for c in corpus
    ]
    last_update = [
        pd.to_datetime([
            s.replace("Last update:", "").replace("Last Update:", "").strip()
            for s in c
            if "last update:" in s.lower()
        ][0])
        for c in corpus
    ]
    voting = [" ".join([s for s in c if "Voting" in s]) for c in corpus]
    release_time = [
        " ".join([
            s
            for s in c
            if ("For release at" in s) | ("For immediate release" in s)
        ])
        for c in corpus
    ]

```

```

    return pd.DataFrame(
        {
            "release_date": release_date,
            "last_update": last_update,
            "text": text,
            "voting": voting,
            "release_time": release_time,
        }
    )
)

```

Overwriting .../skfin/dataloaders/fomc.py

```

[11]: %%writefile ../skfin/dataloaders/io_utils.py
import os
from pathlib import Path
from typing import Dict, Union, Any

import pandas as pd


def clean_directory_path(
    cache_dir: Union[str, Path, None], default_dir: str = "data"
) -> Path:
    """
    Ensure a directory path exists, creating it if necessary.

    Args:
        cache_dir: Directory path to clean/create
        default_dir: Default directory name if cache_dir is None

    Returns:
        Path object to the clean directory
    """
    if cache_dir is None:
        cache_dir = Path(os.getcwd()) / default_dir
    if isinstance(cache_dir, str):
        cache_dir = Path(cache_dir)
    if not cache_dir.is_dir():
        os.makedirs(cache_dir)
    return cache_dir


def save_dict(data: Dict[str, Any], output_dir: Path) -> None:
    """
    Recursively save a dictionary structure to disk.

    Args:
        data: Dictionary to save
        output_dir: Directory path where to save data
    """
    assert isinstance(data, dict)
    if not output_dir.is_dir():
        os.makedirs(output_dir)
    for k, v in data.items():
        if isinstance(v, pd.DataFrame):
            v.to_parquet(output_dir / f"{k}.parquet")
        else:
            save_dict(v, output_dir=output_dir / k)

```

```
def load_dict(input_dir: Path) -> Dict[str, Any]:
    """
    Recursively load a dictionary structure from disk.

    Args:
        input_dir: Directory path from which to load data

    Returns:
        Dictionary with loaded data
    """
    data = {}
    for o in os.scandir(input_dir):
        if o.name.endswith(".parquet"):
            k = o.name.replace(".parquet", "")
            data[k] = pd.read_parquet(o)
        elif o.is_dir():
            data[o.name] = load_dict(o)
    return data
```

Overwriting ../skfin/dataloaders/io_utils.py

```
[12]: %%writefile ../skfin/dataloaders/web_utils.py
from io import BytesIO
from zipfile import ZipFile

import requests
from bs4 import BeautifulSoup

class WebUtils:
    """Utilities for web requests and parsing."""

    @staticmethod
    def get_response(url: str) -> requests.Response:
        """Perform a GET request and return the response."""
        return requests.get(url)

    @staticmethod
    def download_zip_content(url: str) -> ZipFile:
        """Download and return the content of a ZIP file from a URL."""
        r = WebUtils.get_response(url)
        return ZipFile(BytesIO(r.content))

    @staticmethod
    def parse_html(content: str) -> BeautifulSoup:
        """Parse HTML content with BeautifulSoup."""
        return BeautifulSoup(content, "html.parser")
```

Overwriting ../skfin/dataloaders/web_utils.py

Chapter 28

Appendix: Custom Helper Functions for Text Analysis

```
[2]: %%writefile ../skfin/text.py
import logging
import sys

import numpy as np
import pandas as pd
from matplotlib import cm
from matplotlib import pyplot as plt
from IPython.display import HTML, display
from skfin.plot import bar

# Set up logging
logging.basicConfig(stream=sys.stdout, level=logging.INFO)
logger = logging.getLogger(__name__)

# Configure Pandas and Matplotlib settings
pd.options.display.max_colwidth = None
plt.style.use("seaborn-whitegrid")

def show_text(dataframe, lexica=None, text_column="text", n=2):
    """
    Displays a sample of the text column in a DataFrame with optional lexical highlighting.

    Args:
        dataframe (pd.DataFrame): The DataFrame containing the text data.
        lexica (dict): Dictionary containing 'positive' and 'negative' words for highlighting.
        text_column (str): The name of the column containing text data.
        n (int): The number of samples to display. If None, all data is displayed.
    """
    if n is not None:
        dataframe = dataframe.sample(n=n)

    dataframe[text_column] = (
        dataframe[text_column]
        .str.replace("$", "\$", regex=False)
        .str.replace("\n", " ", regex=False)
    )

    if lexica is not None:
```

```
        dataframe[text_column] = dataframe[text_column].apply(
            highlight_lexica, lexica=lexica
        )

    display(HTML(dataframe.to_html(escape=False)))

def green_text(text):
    return f"<b><font color='green'>{text}</font></b>"

def red_text(text):
    return f"<b><font color='red'>{text}</font></b>"

def color_text(word, lexica):
    """
    Wraps a word with HTML to color it based on lexica values.

    Args:
        word (str): The word to color.
        lexica (dict): Dictionary containing 'positive' and 'negative' words for highlighting.

    Returns:
        str: HTML-colored word if found in lexica, otherwise the word itself.
    """
    word_lower = word.lower()
    if word_lower in lexica["positive"]:
        return green_text(word)
    elif word_lower in lexica["negative"]:
        return red_text(word)
    return word

def highlight_lexica(text, lexica):
    """
    Highlights lexica in the text by applying HTML coloring.

    Args:
        text (str or list): Text to highlight. If list, it takes the first element.
        lexica (dict): Dictionary containing 'positive' and 'negative' words for highlighting.

    Returns:
        str: Text with colored words according to lexica.
    """
    if isinstance(text, list):
        text = text[0]
    text = text.replace("<br /><br />", "")
    return " ".join(color_text(word, lexica) for word in text.split())

def plot_document_embeddings(embeddings):
    """
    Plots document embeddings using a scatter plot.

    Args:
        embeddings (pd.DataFrame): DataFrame containing embeddings with index as dates.
    """
    fig, ax = plt.subplots(figsize=(8, 7))
    unique_years = [str(year) for year in embeddings.index.year.unique()]
    colors = cm.RdBu(np.linspace(0, 1, len(unique_years)))
```

```

for i, year in enumerate(unique_years):
    ax.scatter(
        x=embeddings.loc[year][0], y=embeddings.loc[year][1], color=colors[i]
    )

ax.legend(unique_years, loc="center left", bbox_to_anchor=(1, 0.5))
ax.set_xlabel("PC 0")
ax.set_ylabel("PC 1")

specific_date = "2020-03-03"
ax.text(
    x=embeddings.loc[specific_date][0],
    y=embeddings.loc[specific_date][1],
    s=specific_date,
)

```

def plot_word_embeddings(embeddings, num_plots=6):

Plots word embeddings using bar charts.

Args:

 embeddings (*pd.DataFrame*): DataFrame containing word embeddings.
 num_plots (int): Number of plots to generate.

Plots positive and negative coefficients of features.

**def coefs_plot(
 coefficients, top_n=40, fontsize=12, rotation=0, title=None, filename=None
):**

Plots positive and negative coefficients of features.

Args:

 coefficients (*pd.Series*): Series of coefficients indexed by feature names.
 top_n (int): Number of top positive and negative coefficients to plot.
 fontsize (int): Font size for the y-tick labels.
 rotation (int): Rotation angle for the y-tick labels.
 title (str): Title of the entire figure.
 filename (str): If provided, the plot is saved with this file name.

fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12, 10))

 coefficients = coefficients.squeeze()
 positive_coefs = (
 coefficients.loc[coefficients > 0].sort_values(ascending=True).head(top_n)
)
 negative_coefs = (
 coefficients.loc[coefficients < 0].sort_values(ascending=False).tail(top_n)
)

```
axes[0].barh(np.arange(len(negative_coefs)), negative_coefs.values, capsize=5)
axes[0].set_yticks(np.arange(len(negative_coefs)))
axes[0].set_yticklabels(negative_coefs.index, rotation=rotation, fontsize=fontsize)
axes[0].set_title("Negative Coefficients")

axes[1].barh(np.arange(len(positive_coefs)), positive_coefs.values, capsize=5)
axes[1].set_yticks(np.arange(len(positive_coefs)))
axes[1].set_yticklabels(positive_coefs.index, rotation=rotation, fontsize=fontsize)
axes[1].yaxis.tick_right()
axes[1].set_title("Positive Coefficients")

if title:
    fig.suptitle(title, y=0.92)

if filename:
    plt.savefig(f"{filename}.png", orientation="landscape", bbox_inches="tight")

def error_analysis_plot(data, lexica, sample_size=5):
    """
    Plots a subset of data with the largest prediction errors, highlighting lexica.

    Args:
        data (pd.DataFrame): DataFrame containing 'label', 'pred', and 'text' columns.
        lexica (dict): Dictionary containing 'positive' and 'negative' words for highlighting.
        sample_size (int): The number of samples to display from the top and bottom errors.
    """
    data = data.assign(diff=lambda df: df["label"] - df["pred"]).sort_values("diff")
    if sample_size is not None:
        data = pd.concat([data.head(sample_size), data.tail(sample_size)])
    show_text(data, lexica)
```

Overwriting ../skfin/text.py

Chapter 29

Appendix: A Structured Framework for Presenting Empirical Research Findings

We outline a structured approach for presenting research findings. The framework is divided into several key segments:

1. Introduction
2. Dataset overview
3. Analytics and learning strategies
4. Empirical results: baseline and robustness
5. Conclusion

The opening segment encompasses four essential elements:

- Contextual Background: What is the larger setting of the study? What makes this area of inquiry compelling? What are the existing gaps or limitations within the current body of research? What are some unanswered yet noteworthy questions?
- Project Contributions: What are the specific advancements made by this study, such as in data acquisition, algorithmic development, parameter adjustments, etc.?
- Summary of the main empirical results: What is the main statistical statement? Is it significant (e.g. statistically or economically)?
- Literature and Resource Citations: What are related academic papers? What are the GitHub repositories, expert blogs, or software packages that used in this project?

In the dataset profile, one should consider:

- The origin and composition of data utilized in the study. If the dataset is original, then provide the source code to ensure reproducibility.
- The chronological accuracy of the data points, verifying that the dates reflect the actual availability of information.
- A detailed analysis of descriptive statistics, with an emphasis on discussing the importance of the chosen graphs or metrics.

The analytics and machine learning methodologies section accounts for:

- A detailed explanation of the foundational algorithm.
- A description of the data partitioning strategy for training, validation and test.

- An overview of the parameter selection and optimization process.

To effectively convey the empirical findings, separate the baseline results from the additional robustness tests. Within the primary empirical outcomes portion, include:

- Key statistical evaluations (for instance, if presenting a backtest – provide a pnl graph alongside the Sharpe ratio).
- Insights into what primarily influences the results, such as specific characteristics or assets that significantly impact performance.

The robustness of empirical tests section should detail:

- Evaluation of the stability of the principal finding against variations in hyperparameters or algorithmic modifications.

Finally, the conclusive synthesis should recapitulate the primary findings, consider external elements that may influence the results, and hint at potential directions for further investigative work.

Google Scholar, SSRN, and arXiv are key resources for discovering new research ideas.

1. **Google Scholar** (<https://scholar.google.com/>): A free search engine indexing scholarly literature across various formats and disciplines, it helps researchers find academic papers, theses, and books, offering a broad view of existing research and citation tracking.
2. **Social Science Research Network** (<https://www.ssrn.com/>): Focuses on social sciences, providing early-stage research and working papers. It's invaluable for accessing the latest findings in fields like economics, law, and business, enabling researchers to spot emerging trends and collaborative opportunities.
3. **arXiv** (<https://arxiv.org/>): A preprint repository covering areas like physics, mathematics, and computer science, arXiv offers immediate access to unreviewed preprints. This facilitates rapid dissemination of new ideas and access to pioneering research developments.

These platforms are essential for generating new research ideas, promoting collaboration, and staying updated on advances in various fields.

Bibliography

- J.-P. Bouchaud, J. Bonart, J. Donier, and M. Gould. *Trades, quotes and prices: financial markets under the microscope*. Cambridge University Press, 2018.
- M. Britten-Jones. The sampling error in estimates of mean-variance efficient portfolio weights. *The Journal of Finance*, 54(2):655–671, 1999.
- J. Y. Campbell and S. B. Thompson. Predicting excess stock returns out of sample: Can anything beat the historical average? *The Review of Financial Studies*, 21(4):1509–1531, 2008.
- T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 785–794, 2016.
- A. Frazzini, D. Kabiller, and L. H. Pedersen. Buffett’s alpha. *Financial Analysts Journal*, 74(4):35–55, 2018.
- T. Hastie, R. Tibshirani, J. H. Friedman, and J. H. Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.
- F. Ielpo, C. Merhy, and G. Simon. *Engineering Investment Process: Making Value Creation Repeatable*. Elsevier, 2017.
- M. Isichenko. *Quantitative Portfolio Management: The Art and Science of Statistical Arbitrage*. Wiley, 2022.
- S. Jansen. *Machine Learning for Algorithmic Trading: Predictive models to extract signals from market and alternative data for systematic trading strategies with Python*. Packt Publishing Ltd, 2020.
- N. Jegadeesh and D. Wu. Word power: A new approach for content analysis. *Journal of financial economics*, 110(3):712–729, 2013.
- R. N. Kahn. *The future of investment management*. CFA Institute Research Foundation, 2018.
- A. Karpathy. A recipe for training neural networks. 2019.
- A. E. Khandani and A. W. Lo. What happened to the quants in august 2007? evidence from factors and transactions data. *Journal of Financial Markets*, 14(1):1–46, 2011.
- O. Ledoit and M. Wolf. Honey, i shrunk the sample covariance matrix. *The Journal of Portfolio Management*, 30(4):110–119, 2004.
- A. W. Lo. The statistics of sharpe ratios. *Financial analysts journal*, 58(4):36–52, 2002.
- A. W. Lo and A. C. MacKinlay. When are contrarian profits due to stock market overreaction? *The review of financial studies*, 3(2):175–205, 1990.

Bibliography

- T. Loughran and B. McDonald. When is a liability not a liability? textual analysis, dictionaries, and 10-ks. *The Journal of finance*, 66(1):35–65, 2011.
- N. Maslej, L. Fattorini, R. Perrault, V. Parli, A. Reuel, E. Brynjolfsson, J. Etchemendy, K. Ligett, T. Lyons, J. Manyika, J. C. Niebles, Y. Shoham, R. Wald, and J. Clark. Artificial intelligence index report 2024, 2024. URL <https://arxiv.org/abs/2405.19522>.
- T. J. Moskowitz and M. Grinblatt. Do industries explain momentum? *The Journal of finance*, 54(4):1249–1290, 1999.
- K. P. Murphy. *Probabilistic machine learning: an introduction*. MIT press, 2022.
- T. Papamichalis, D. Ryu, and M. Wilson. Divided government and the stock market. Technical report, Working Paper, University of Cambridge, 2024.
- L. H. Pedersen. *Efficiently inefficient: how smart money invests and market prices are determined*. Princeton University Press, 2019.
- P. Santa-Clara and R. Valkanov. The presidential puzzle: Political cycles and the stock market. *The Journal of Finance*, 58(5):1841–1872, 2003.
- P. Savor and M. Wilson. How much do investors care about macroeconomic risk? evidence from scheduled economic announcements. *Journal of Financial and Quantitative Analysis*, 48(2):343–375, 2013.
- W. F. Sharpe. Asset allocation: Management style and performance measurement. *Journal of portfolio Management*, 18(2):7–19, 1992.
- I. Welch and A. Goyal. A comprehensive look at the empirical performance of equity premium prediction. *The Review of Financial Studies*, 21(4):1455–1508, 2008.