
Financial Data Science Python Notebooks

Terence Lim

Mar 29, 2025

CONTENTS

1 Stock Prices	5
1.1 FinDS package	5
1.2 Stock price data	6
1.3 Properties of stock returns	14
2 Jegadeesh-Titman Rolling Portfolios	23
2.1 Price Momentum	24
2.2 Hypothesis testing	29
3 Fama-French Portfolio Sorts	35
3.1 Stock fundamentals data	36
3.2 Bivariate sorts	37
3.3 Linear regression	42
3.4 Structural break test	46
4 Fama-Macbeth Cross-sectional Regressions	49
4.1 Mean variance optimization	50
4.2 Implied alphas	55
4.3 Cross-sectional regressions	59
4.4 Nonlinear regression	65
5 Contrarian Trading	69
5.1 Mean reversion	70
5.2 Implementation shortfall	73
5.3 Structural break with unknown changepoint	74
5.4 Performance evaluation	76
6 Quant Factors	81
6.1 Factor investing	82
6.2 Cluster analysis	137
7 Event Study	147
7.1 Abnormal returns	148
7.2 Fourier transforms	155
7.3 Multiple testing	157
8 Economic Indicators	171
8.1 FRED	172
8.2 Retrieving data from websites	172
8.3 Revisions and vintage dates	181
8.4 Outliers	187

9 Linear Regression Diagnostics	191
9.1 Model assumptions	192
9.2 Residual diagnostics	195
10 Time Series Analysis	201
10.1 Seasonality	202
10.2 Stationarity	204
10.3 Autocorrelation	205
10.4 Time Series Models	205
10.5 Forecasting	209
11 Approximate Factor Models	219
11.1 Integration order	220
11.2 Factor selection	223
12 State Space Models	231
12.1 Hidden Markov Model	232
12.2 Gaussian Mixture Model	237
13 Term Structure of Interest Rates	239
13.1 Interest Rates	239
13.2 Yield Curve	243
13.3 Principal Component Analysis (PCA)	250
13.4 Singular Value Decomposition (SVD)	252
13.5 Low-Rank Approximations	254
14 Interest Rate Risk	257
14.1 Interest rate sensitivity	257
14.2 Risk factors	262
15 Options Pricing	273
15.1 Options	273
15.2 Binomial option pricing	279
15.3 Black-Scholes-Merton model	285
15.4 Monte Carlo simulation	292
16 Value at Risk	297
16.1 Risk measures	298
16.2 Conditional volatility models	307
16.3 Backtesting VaR	311
17 Market Microstructure	313
17.1 Tick data	314
17.2 Liquidity measures	316
17.3 By market capitalization	319
17.4 High frequency sampling	326
18 Event Risk	337
18.1 Earnings expectations	337
18.2 Poisson regression	340
19 Supply Chain Network Graphs	345
19.1 Principal customers	345
19.2 Graph properties	346

20 Industry Community Detection	353
20.1 Industry taxonomy	353
20.2 Community structure	357
21 Input-Output Graph Centrality	365
21.1 Centrality measures	365
21.2 BEA Input-Output Use Tables	366
22 Product Market Link Prediction	383
22.1 Product market linkages	383
22.2 Link prediction algorithms	386
22.3 Accuracy metrics	387
23 Earnings Spatial Regression	399
23.1 Earnings surprise	400
23.2 Spatial dependence models	401
24 FOMC Topic Modeling	407
24.1 FOMC meeting minutes	408
24.2 Text pre-processing	410
24.3 Topic Modeling	412
25 Management Sentiment Analysis	421
25.1 Sentiment analysis	422
25.2 10-K Management discussion and analysis	424
26 Machine Learning: Classification	435
26.1 Text classification	436
26.2 Classification models	438
26.3 Evaluation	441
27 Machine Learning: Regression	451
27.1 Macroeconomic forecasting	451
27.2 Regression models	454
28 Deep Learning	479
28.1 Industry text classification	480
28.2 Feedforward neural networks	483
29 Convolutional Neural Networks	493
29.1 Convolutions	493
29.2 Temporal convolutional networks (TCN)	495
29.3 Vector Autoregression	506
30 Recurrent Neural Networks	513
30.1 Sequence modeling	514
30.2 Elman network	515
30.3 Dynamic Factor Models	522
31 Reinforcement Learning	527
31.1 Retirement spending policy	528
31.2 Deep reinforcement learning	538
32 Language Modeling	543
32.1 Transformers	544
32.2 Language modeling	547

33 Large Language Models	555
33.1 OpenAI GPT models	556
33.2 DeepSeek-R1 model	557
33.3 Text summarization	559
34 Fine-tuning	611
34.1 Meta Llama-3.1 model	612
34.2 Supervised fine-tuning (SFT)	613
34.3 Industry text classification	616
35 Prompting	625
35.1 Sentiment analysis	625
35.2 Google Gemma 3 model	627
35.3 Structured dialogue	627
35.4 Vision language	635
35.5 Multi-lingual	636
36 Agents	643
36.1 Microsoft Phi-4 model	644
36.2 Chatbot	647
36.3 Retrieval-augmented generation (RAG)	651
36.4 Multi-agents	657
36.5 Multi-lingual	667
36.6 Vision language	668
36.7 Audio	670

As financial markets produce vast volumes of structured and unstructured data, the ability to extract insights and develop predictive models has become increasingly important. [Financial Data Science Python Notebooks](#) provide a practical guide for analysts, researchers, and data scientists looking to apply Python and its broad ecosystem of libraries, tools, frameworks, and community resources to financial analysis, econometrics, and machine learning.

Designed to support financial data science workflows, the companion [FinDS Python package](#) demonstrates how to use database engines such as SQL, Redis, and MongoDB to manage and access large datasets, including:

- Core financial databases such as CRSP, Compustat, IBES, and TAQ
- Public economic data APIs from sources like FRED and the Bureau of Economic Analysis (BEA)
- Structured and unstructured data from academic and research websites

In addition to data access, it provides practical examples and templates for applying:

- Financial econometrics and time series modeling
- Graph analytics, event studies, and backtesting strategies
- Machine learning for predictive analytics
- Natural language processing (NLP) to extract insights from financial text
- Neural networks and large language models (LLMs) for advanced decision-making

March 2025: Updated with data through early 2025 and incorporated the latest LLMs – Microsoft Phi-4-multimodal (released Feb 2025), Google Gemma-3-12B (March 2025), DeepSeek-R1-14b (January 2025), Meta Llama-3.1-8B (July 2024), GPT-4o-mini (July 2024).

Topics

notebook	Financial	Data	Science
1.1_stock_prices	Stock price properties	CRSP stocks	Statistical moments
1.2_jegadeesh_titman	Price momentum	CRSP stocks	Hypothesis testing, Newey-West estimator
1.3_fama_french	Value and size	CRSP stocks, Compustat	Linear regression
1.4_fama_macbeth	CAPM	Fama-French	Non-linear regression, Quadratic optimization
1.5_contrarian_trading	Mean reversion, Implementation shortfall	CRSP stocks	Structural breaks
1.6_quant_factors	Factor investing, Backtesting	CRSP stocks, Compustat, IBES	Cluster analysis
1.7_event_study	Event studies	S&P key developments	Multiple testing, Fourier transforms and convolutions
2.1_economic_indicators	Economic data revisions, Employment payrolls	ALFRED	Outlier detection
2.2_regression_diagnostics	Consumer and producer prices	FRED	Linear regression diagnostics
2.3_time_series	Industrial production and inflation	FRED	Time series analysis
2.4_approximate_factors	Approximate factor models	FRED-MD	Unit root test, EM Algorithm

continues on next page

notebook	Financial	Data	Science
2.5_economic_states	State space models	FRED-MD	Gaussian Mixtures, Hidden Markov Models
3.1_term_structure	Interest rates	FRED yield curve	Low rank approximation
3.2_bond_returns	Bonds risk factors	FRED bond returns	Principal component analysis
3.3_options_pricing	Binomial tree, Black-Scholes-Merton	simulated	Monte Carlo simulations
3.4_value_at_risk	Value-at-risk	FRED crypto-currencies	Conditional volatility
3.5_covariance_matrix	Portfolio risk	Fama-French industries	Covariance matrix estimation
3.6_market_microstructure	Market liquidity	TAQ tick data	High frequency volatility
3.7_event_risk	Earnings expectations	IBES	Poisson regression, generalized linear model
4.1_network_graphs	Supply chain	Compustat principal customers	Network graphs
4.2_community_detection	Industry taxonomy	Hoberg-Phillips	Community detection
4.3_graph_centrality	Input-output uses	Bureau of Economic Analysis	Graph centrality
4.4_link_prediction	Product markets	Hoberg-Phillips	Link prediction
4.5_spatial_regression	Earnings surprises	IBES, Hoberg-Phillips	Spatial regression
5.1_fomc_topics	FOMC meetings	Federal Reserve	Topic modeling
5.2_management_sentiment	Management discussions	SEC Edgar, Loughran-Macdonald	Sentiment analysis
5.3_business_textual	Business descriptions	SEC Edgar	Part-of-speech, Density-based clustering
6.1_classification_models	Industry classification	SEC Edgar	Classification
6.2_regression_models	Macroeconomic forecasts	FRED-MD	Regression
6.3_deep_learning	Industry classification	SEC Edgar	Neural networks, word embeddings
6.4_convolutional_net	Macroeconomic forecasts	FRED-MD	Convolutional Neural Nets, Vector autoregression
6.5_recurrent_net	Macroeconomic forecasts	FRED-MD	Recurrent Neural Nets, Dynamic factor models
6.6_reinforcement_learning	Retirement spending	SBBI	Reinforcement learning
6.7_language_modeling	Fedspeak	Federal Reserve	Language modeling, Transformers
7.1_large_language_models	Market risk disclosures	SEC Edgar	Text summarization
7.2_llm_finetuning	Industry classification	SEC Edgar	LLM fine-tuning
7.3_llm_prompting	Financial news sentiment	Kaggle	Prompt engineering
7.4_llm_agents	Corporate philanthropy	textbook	Multi-agents, chatbots, retrieval-augmented generation

Documentation

- [Financial Data Science Notebooks](#),
- [Download PDF](#)
- [FinDS API reference](#)

Github repos

- FinDS package
- Jupyter notebooks

Contact

<https://terence-lim.github.io>

STOCK PRICES

In physics it takes three laws to explain 99% of the data; in finance it takes more than 99 laws to explain about 3% - Andrew Lo

Stock price data encompasses historical prices as well as corporate actions such as stock splits, dividends, and delistings. The CRSP database is a standard resource in academic research due to its comprehensive coverage of both active and delisted stocks, which supports unbiased and representative analysis. Efficient storage, retrieval, and processing of such structured financial data are enabled by tools such as SQL, SQLAlchemy, Redis, and Pandas. We examine statistical moments, assume log-normal distributions, and explore alternative correlation measures for modeling stock return behavior and dependencies.

```
# By: Terence Lim, 2020-2025 (terence-lim.github.io)
import numpy as np
import scipy
import pandas as pd
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
import warnings
from tqdm import tqdm
from finds.database import SQL, RedisDB
from finds.structured import BusDay, CRSP, CRSPBuffer, Benchmarks
from finds.utils import Finder
from secret import credentials, CRSP_DATE, paths
VERBOSE = 0

# %matplotlib qt
```

1.1 FinDS package

Developed over a journey to support financial data science workflows, our *FinDS* Python package integrates:

- Use of database engines like SQL, Redis, and MongoDB to manage large structured and unstructured datasets
- Tools for accessing key financial datasets, including CRSP, Compustat, IBES, and TAQ
- Interfaces to public data sources such as FRED, SEC EDGAR, and the BEA
- Utilities for extracting data from academic and research websites
- Recipes for applying a range of statistical and machine learning methods, including network graphs, natural language processing and large language models.

1.1.1 SQL

Structured Query Language (SQL) is a popular tool for storing and managing relational data organized into tables with columns (fields) and rows (records). Open-source systems like **MySQL** are widely used to implement relational databases, and Python libraries such as **SQLAlchemy** provide convenient interfaces for interacting with them. Additionally, the **Pandas** library allows users to run SQL queries and load the results directly into DataFrames for efficient analysis.

1.1.2 Redis

Redis is an open-source, in-memory data store commonly used as a caching layer. It helps improve performance by storing frequently accessed data in memory, thereby reducing the load on slower, primary databases.

```
# open database connections
sql = SQL(**credentials['sql'], verbose=VERBOSE)
rdb = RedisDB(**credentials['redis'])
bd = BusDay(sql, endweek=3, verbose=VERBOSE)
crsp = CRSP(sql, bd, rdb=rdb, verbose=VERBOSE)
bench = Benchmarks(sql, bd, verbose=VERBOSE)
find = Finder(sql)
outdir = paths['scratch']
```

1.2 Stock price data

Besides the historical prices of stocks, their adjustments such as identifier changes, stock splits, dividends, mergers, and delistings must also be recorded to accurately analyze performance over time.

1.2.1 CRSP stocks

The Center for Research in Security Prices (CRSP) provides the most widely used data for academic research into US stocks. It includes both successful and unsuccessful entities, rather than just those that have “survived” over time. This helps avoid the pitfalls of focusing only on surviving entities which can lead to an overestimation of average performance and underestimation of risk. It also captures corporate actions by standardizing data on events such as name changes, distributions and delistings. This information is recorded and validated from official sources, integrated into its historical databases along with details like announcement and effective dates, adjustment factors, and impact on stock performance calculations.

```
# describe the database schema of the CRP Stocks `names` table
DataFrame(**sql.run('describe names'))
```

	Field	Type	Null	Key	Default	Extra
0	date	int	NO	PRI	None	
1	comnam	varchar(32)	YES		None	
2	ncusip	varchar(8)	YES	MUL	None	
3	shrccls	varchar(1)	YES		None	
4	ticker	varchar(5)	YES		None	
5	permno	int	NO	PRI	None	
6	nameendt	int	YES		None	
7	shrcd	smallint	YES		None	
8	exchcd	smallint	YES		None	
9	siccd	smallint	YES		None	

(continues on next page)

(continued from previous page)

```

10  tsymbol  varchar(7)  YES      None
11  naics     int      YES      None
12  primexch  varchar(1)  YES      None
13  trdstat   varchar(1)  YES      None
14  secstat   varchar(4)  YES      None
15  permco    int      YES      None

```

SQL select and join statements to retrieve Apple Computer's identifiers and corporate actions, such as stock splits and dividends. Commonly used SQL commands listed at the end of this notebook.

```

# double up the % when passing sql command string to pandas
names_df = pd.read_sql("select * from names where comnam like '%%APPLE COMPUTER%%'", con=sql.engine)
names_df

```

	date	comnam	ncusip	shrccls	ticker	permno	nameendt	\	
0	19801212	APPLE COMPUTER INC	03783310		AAPL	14593	19821031		
1	19821101	APPLE COMPUTER INC	03783310		AAPL	14593	20040609		
2	20040610	APPLE COMPUTER INC	03783310		AAPL	14593	20070110		
	shrcd	exchcd	siccd	tsymbol	naics	primexch	trdstat	secstat	permco
0	11	3	3573		0	Q	A	R	7
1	11	3	3573	AAPL	0	Q	A	R	7
2	11	3	3573	AAPL	334111	Q	A	R	7

```

# inner join of identifiers (names) and distributions (dist) tables
cmd = """
select distinct names.permno, divamt, facpr, exdt, comnam, ticker from names
inner join dist
on names.permno = dist.permno
where names.comnam like '%%APPLE COMPUTER%%'
"""
dist_df = pd.read_sql(cmd, con=sql.engine)
dist_df

```

	permno	divamt	facpr	exdt	comnam	ticker
0	14593	0.12	0.0	19870511	APPLE COMPUTER INC	AAPL
1	14593	0.06	0.0	19870810	APPLE COMPUTER INC	AAPL
2	14593	0.08	0.0	19871117	APPLE COMPUTER INC	AAPL
3	14593	0.08	0.0	19880212	APPLE COMPUTER INC	AAPL
4	14593	0.08	0.0	19880516	APPLE COMPUTER INC	AAPL
..
86	14593	0.00	1.0	19870616	APPLE COMPUTER INC	AAPL
87	14593	0.00	1.0	20000621	APPLE COMPUTER INC	AAPL
88	14593	0.00	1.0	20050228	APPLE COMPUTER INC	AAPL
89	14593	0.00	6.0	20140609	APPLE COMPUTER INC	AAPL
90	14593	0.00	3.0	20200831	APPLE COMPUTER INC	AAPL

[91 rows x 6 columns]

1.2.2 Stock splits and dividends

An investor's total holding returns (`ret` in CRSP) include gains from appreciated stock prices (`prc`), adjusted for stock splits (`facpr`), plus ordinary cash dividends (`divamt`). Specifically, on ex-dates t :

$$ret_t = \frac{prc_t (1 + facpr_t) + div_t}{prc_{t-1}}$$

The Factor to Adjust Price (`facpr`) values over time can be used to adjust prices for distributions such as stock dividends and splits so that stock prices before and after one or more distributions are comparable. Historical cumulative adjust factors are computed by adding 1 to and then taking cumulative product from current to earlier time periods. This cumulative factor between two dates is divided into the earlier raw stock price to derive comparable split-adjusted prices. Hence to split-adjust CRSP raw prices

- apply cumulative factor to raw prices before corresponding ex-date
- back-fill to dates prior to ex-date
- default factor after latest ex-date is 1

yfinance

The yfinance Python library enables users to access current financial data from Yahoo Finance.

```
import yfinance as yf
ticker = yf.Ticker('AAPL')
df = ticker.history(period='max')
df[df['Dividends'].gt(0) | df['Stock Splits'].ne(0)]
```

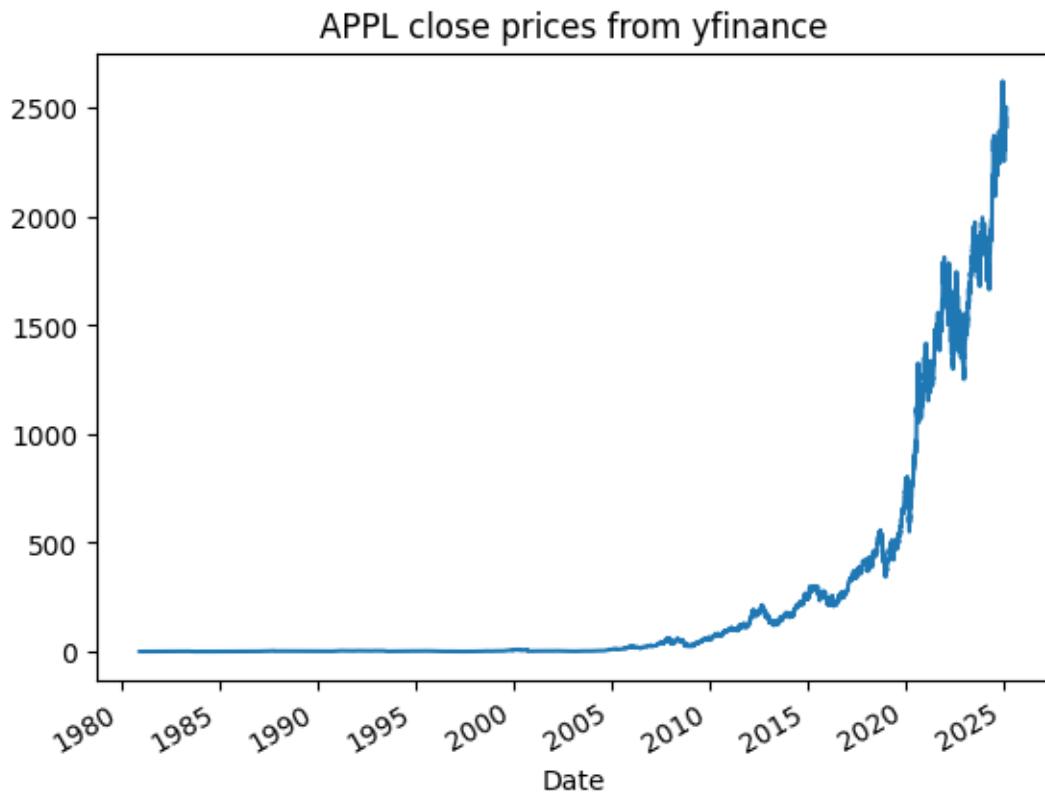
	Open	High	Low	Close	\
Date					
1987-05-11 00:00:00-04:00	0.264817	0.273415	0.263957	0.264817	
1987-06-16 00:00:00-04:00	0.285452	0.287171	0.261378	0.285452	
1987-08-10 00:00:00-04:00	0.332310	0.332310	0.315091	0.332310	
1987-11-17 00:00:00-05:00	0.253658	0.255384	0.241579	0.241579	
1988-02-12 00:00:00-05:00	0.280957	0.287009	0.280093	0.283551	
...
2024-02-09 00:00:00-05:00	187.763406	189.097120	187.116467	187.962479	
2024-05-10 00:00:00-04:00	184.280653	184.470019	181.519943	182.436859	
2024-08-12 00:00:00-04:00	215.595507	219.027939	215.126538	217.052292	
2024-11-08 00:00:00-05:00	226.920500	228.408869	226.161340	226.710739	
2025-02-10 00:00:00-05:00	229.570007	230.589996	227.199997	227.649994	
	Volume	Dividends	Stock Splits		
Date					
1987-05-11 00:00:00-04:00	197276800	0.000536	0.0		
1987-06-16 00:00:00-04:00	342720000	0.000000	2.0		
1987-08-10 00:00:00-04:00	77996800	0.000536	0.0		
1987-11-17 00:00:00-05:00	268800000	0.000714	0.0		
1988-02-12 00:00:00-05:00	137760000	0.000714	0.0		
...		
2024-02-09 00:00:00-05:00	45155200	0.240000	0.0		
2024-05-10 00:00:00-04:00	50759500	0.250000	0.0		
2024-08-12 00:00:00-04:00	38028100	0.250000	0.0		
2024-11-08 00:00:00-05:00	38328800	0.250000	0.0		
2025-02-10 00:00:00-05:00	33115600	0.250000	0.0		

[91 rows x 7 columns]

The daily `Close` prices from yFinance have been adjusted for stock splits and dividend payments. As a result, the plotted values directly represent cumulative total holding returns.

```
df['Close'].div(df['Close'].iloc[0]).plot(title="APPL close prices from yfinance")
```

```
<Axes: title={'center': 'APPL close prices from yfinance'}, xlabel='Date'>
```



To retrieve unadjusted historical prices accounting for stock splits and dividends, first apply the split factors to determine the original dividends per unadjusted share. The resulting values closely match the CRSP `divamt` figures, with minor differences likely due to cumulative numerical precision errors.

```
split = df['Stock Splits'].where(df['Stock Splits'] != 0.0, 1) \
    .shift(-1).fillna(1).iloc[::-1].cumprod().iloc[::-1]      # cumulate the split factors back in time
df['unadj_div'] = df['Dividends'] * split.shift(1).fillna(split.iloc[0])  # apply the split factors to dividends
df.set_index(df.index.strftime('%Y-%m-%d'))[(df['unadj_div'] > 0).values]
```

Date	Open	High	Low	Close	Volume
1987-05-11	0.264817	0.273415	0.263957	0.264817	197276800
1987-08-10	0.332310	0.332310	0.315091	0.332310	77996800
1987-11-17	0.253658	0.255384	0.241579	0.241579	268800000
1988-02-12	0.280957	0.287009	0.280093	0.283551	137760000
1988-05-16	0.280647	0.286711	0.277183	0.285845	74760000
...
2024-02-09	187.763406	189.097120	187.116467	187.962479	45155200

(continues on next page)

(continued from previous page)

2024-05-10	184.280653	184.470019	181.519943	182.436859	50759500
2024-08-12	215.595507	219.027939	215.126538	217.052292	38028100
2024-11-08	226.920500	228.408869	226.161340	226.710739	38328800
2025-02-10	229.570007	230.589996	227.199997	227.649994	33115600

Date	Dividends	Stock Splits	unadj_div
1987-05-11	0.000536	0.0	0.120064
1987-08-10	0.000536	0.0	0.060032
1987-11-17	0.000714	0.0	0.079968
1988-02-12	0.000714	0.0	0.079968
1988-05-16	0.000714	0.0	0.079968
...
2024-02-09	0.240000	0.0	0.240000
2024-05-10	0.250000	0.0	0.250000
2024-08-12	0.250000	0.0	0.250000
2024-11-08	0.250000	0.0	0.250000
2025-02-10	0.250000	0.0	0.250000

[86 rows x 8 columns]

Next, we work backward through time to reconstruct the original stock prices using daily holding returns and closing prices. Stock splits and dividends are accounted for on ex-dates by applying the rearranged formula:

$$prc_{t-1} = \frac{(1 + facpr_t)(prc_t + div_t)}{ret_t}$$

```
rets = df['Close'] / df['Close'].shift(1)
prc = df['Close'].iloc[-1]
div = 0
fac = 0
ret = 1
for i, t in enumerate(df.index[::-1]):      # iterate over days in reverse
    df.loc[t, 'unadj_prc'] = (((fac if fac else 1) * prc) + div) / ret
    prc = df.loc[t, 'unadj_prc']
    div = df.loc[t, 'unadj_div']
    fac = df.loc[t, 'Stock Splits']
    ret = rets.loc[t]
df.set_index(df.index.strftime('%Y-%m-%d'))
```

Date	Open	High	Low	Close	Volume	\
1980-12-12	0.098726	0.099155	0.098726	0.098726	469033600	
1980-12-15	0.094005	0.094005	0.093575	0.093575	175884800	
1980-12-16	0.087136	0.087136	0.086707	0.086707	105728000	
1980-12-17	0.088853	0.089282	0.088853	0.088853	86441600	
1980-12-18	0.091429	0.091858	0.091429	0.091429	73449600	
...	
2025-02-24	244.929993	248.860001	244.419998	247.100006	51326400	
2025-02-25	248.000000	250.000000	244.910004	247.039993	48013300	
2025-02-26	244.330002	244.979996	239.130005	240.360001	44433600	
2025-02-27	239.410004	242.460007	237.059998	237.300003	41153600	
2025-02-28	236.949997	242.089996	230.199997	241.839996	56796200	

Dividends	Stock Splits	unadj_div	unadj_prc
-----------	--------------	-----------	-----------

(continues on next page)

(continued from previous page)

```

Date
1980-12-12      0.0      0.0      0.0  28.757305
1980-12-15      0.0      0.0      0.0  27.257003
1980-12-16      0.0      0.0      0.0  25.256412
1980-12-17      0.0      0.0      0.0  25.881523
1980-12-18      0.0      0.0      0.0  26.631895
...
2025-02-24      0.0      0.0      0.0  247.100006
2025-02-25      0.0      0.0      0.0  247.039993
2025-02-26      0.0      0.0      0.0  240.360001
2025-02-27      0.0      0.0      0.0  237.300003
2025-02-28      0.0      0.0      0.0  241.839996

[11144 rows x 9 columns]

```

The unadjusted price computer and the original closing prices from CRSP during the early days of AAPL stock are nearly identical, with only small differences due to cumulative numerical precision errors.

```

price_df = pd.read_sql(f"select permno, date, abs(prc) from daily where permno={names_
˓→df['permno'][0]}", con=sql.engine)
price_df.head(10)

```

```

      permno      date  abs(prc)
0    14593  19801212  28.8125
1    14593  19801215  27.3125
2    14593  19801216  25.3125
3    14593  19801217  25.9375
4    14593  19801218  26.6875
5    14593  19801219  28.3125
6    14593  19801222  29.6875
7    14593  19801223  30.9375
8    14593  19801224  32.5625
9    14593  19801226  35.5625

```

1.2.3 Market capitalization

Following Fama and French (1992), academic research typically focuses on U.S.-domiciled stocks, specifically those with a share code (shrcd) of 10 or 11, that are listed on major exchanges (exchange code (exchcd) of 1, 2, or 3).

Stocks are often sorted into ten deciles based on market capitalization, with the smallest stocks placed in the 10th decile. These decile breakpoints are determined using only NYSE-listed stocks. For companies with multiple classes of securities, total market value is calculated by summing the market capitalization of all classes. Since CRSP reports shares outstanding (shrouth) in thousands, all market capitalization values must be multiplied by 1,000 to reflect their actual size.

Plot the number of stocks in and the market cap breakpoints of each size decile by year:

```

# retrieve universe of stocks annually from 1981
start = bd.endyr(19731231)
rebals = bd.date_range(start, CRSP_DATE, freq=12)
univs = {rebal: crsp.get_universe(date=rebal) for rebal in rebals}
num = dict()
for date, univ in univs.items():
    num[str(date//10000)] = {decile: sum(univ['decile']==decile)

```

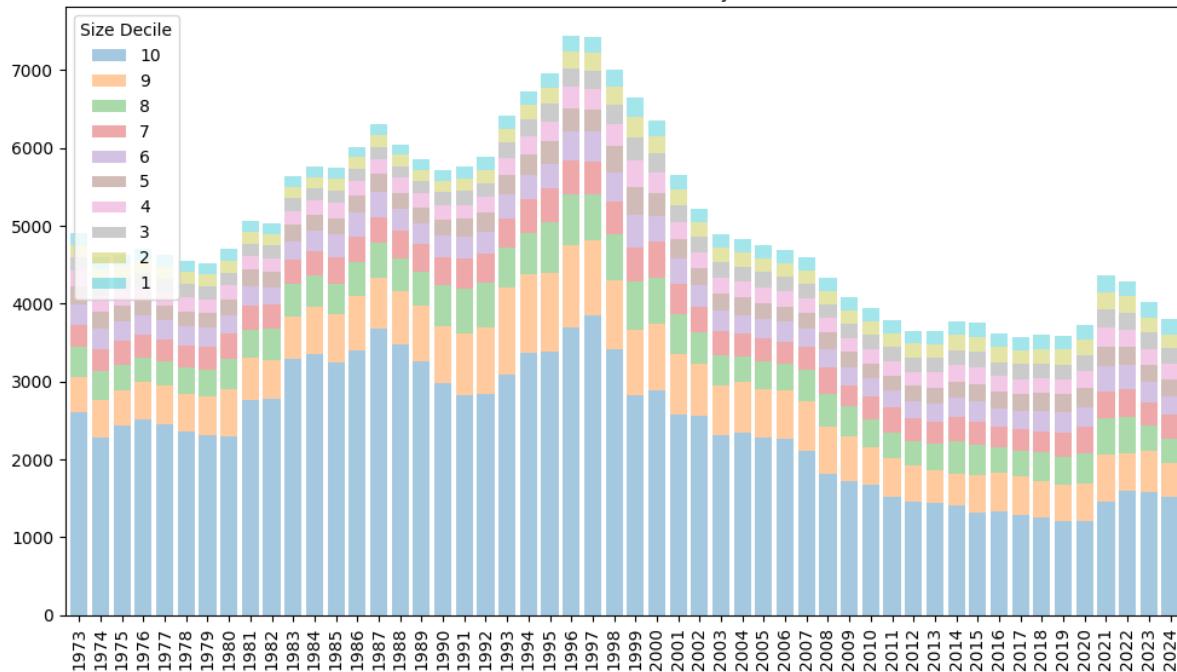
(continues on next page)

(continued from previous page)

```
for decile in range(10, 0, -1):
    num = DataFrame.from_dict(num, orient='index')
```

```
# plot number of stocks in each size decile
fig, ax = plt.subplots(figsize=(10, 6))
ax.set_title('Number of stocks in universe by size decile')
num.plot.bar(stacked=True, ax=ax, width=.8, alpha=0.4)
#set_xtickbins(ax=ax, nbins=len(cap)//10)
plt.legend(title='Size Decile', loc='upper left')
plt.tight_layout()
```

Number of stocks in universe by size decile



```
# If a company has multiple share classes, sum up its total market capitalization
# instead of separate capital values for each share class
names = find('ALPHABET', 'comnam').groupby('permno').tail(1).set_index('permno')
names
```

permno	date	comnam	ncusip	shrccls	ticker	nameendt	shrcd	\
14542	20231013	ALPHABET INC	02079K10	C	GOOG	20241231	11	
90319	20231013	ALPHABET INC	02079K30	A	GOOGL	20241231	11	
permno	exchcd	siccd	tsymbol	naics	primexch	trdstat	secstat	permco
14542	3	7375	GOOG	541511	Q	A	R	45483
90319	3	7375	GOOGL	541511	Q	A	R	45483

```
univ.loc[names.index] # market caps of share class (cap) and total company (capco)
```

	cap	capco	decile	nyse	siccd	prc	naics
permno							
14542	1.053895e+09	2.159975e+09		1	False	7375	190.44
90319	1.106080e+09	2.159975e+09		1	False	7375	189.30

1.2.4 Stock delistings

An important feature of the CRSP database is that it is free of survivorship-bias. It includes the historical records of stocks that have delisted from trading on the exchanges.

In CRSP Monthly, the Delisting Return is calculated from the last month ending price to the last daily trading price if no other delisting information is available. In this case the delisting payment date is the same as the delisting date. If the return is calculated from a daily price, it is a partial-month return. The partial-month returns are not truly Delisting Returns since they do not represent values after delisting, but allow the researcher to make a more accurate estimate of the Delisting Returns.

Following Bali, Engle, and Murray (2016) and Shumway (1997): we can construct returns adjusted for delistings, which result when a company is acquired, ceases operations, or fails to meet exchange listing requirements. The adjustment reflects the partial month of returns to investors who bought the stock in the month before the delisting. For certain delisting codes ([500, 520, 551..574, 580, 584]) where the delisting return is missing, a delisting return of -30% is assumed which reflects the average recovery amount after delisting.

```
# Show sample of original CRSP Monthly ret and dlret before and after adjustment
pd.read_sql('select * from monthly where dlstcd>100 and date=20041130', sql.engine) \
.set_index('permno') \
.rename(columns={'ret': 'original_ret'}) \
.join(crsp.get_ret(beg=20041101, end=20041130), how='left') \
.round(4)
```

	date	prc	original_ret	retx	dlstcd	dlret	ret
permno							
10275	20041130	NaN		NaN	584	-0.2703	-0.2703
10418	20041130	NaN		NaN	520	0.0509	0.0509
11194	20041130	NaN		NaN	233	0.0066	0.0066
12010	20041130	NaN		NaN	587	0.0490	0.0490
20459	20041130	NaN		NaN	231	0.0724	0.0724
32897	20041130	NaN		NaN	584	-0.2357	-0.2357
55589	20041130	NaN		NaN	233	-0.0114	-0.0114
64290	20041130	13.70	0.0178	0.0178	233	0.0000	0.0178
67708	20041130	NaN		NaN	233	0.0164	0.0164
69593	20041130	NaN		NaN	331	0.0998	0.0998
69681	20041130	NaN		NaN	584	-0.4853	-0.4853
70447	20041130	5.80	-0.2246	-0.2246	570	NaN	-0.4572
75606	20041130	NaN		NaN	520	-0.2466	-0.2466
75684	20041130	NaN		NaN	233	0.0094	0.0094
76306	20041130	NaN		NaN	241	-0.0329	-0.0329
76691	20041130	NaN		NaN	582	0.0127	0.0127
77838	20041130	NaN		NaN	520	-0.0041	-0.0041
79149	20041130	NaN		NaN	574	-0.2088	-0.2088
79523	20041130	NaN		NaN	233	0.0043	0.0043
80211	20041130	NaN		NaN	470	0.0186	0.0186
80714	20041130	NaN		NaN	332	0.0337	0.0337
82491	20041130	4.21	-0.0644	-0.0644	551	0.0689	0.0000

(continues on next page)

(continued from previous page)

83583	20041130	125.10	0.2810	0.2810	241	0.0624	0.3608
83702	20041130	NaN	NaN	NaN	587	0.2667	0.2667
83995	20041130	26.58	0.2374	0.2374	231	0.0394	0.2861
84047	20041130	16.35	0.0467	0.0467	241	-0.2661	-0.2318
86123	20041130	NaN	NaN	NaN	233	0.0059	0.0059
86307	20041130	NaN	NaN	NaN	233	0.0223	0.0223
86388	20041130	NaN	NaN	NaN	584	-0.0337	-0.0337
86991	20041130	NaN	NaN	NaN	233	0.0106	0.0106
87126	20041130	NaN	NaN	NaN	231	0.0784	0.0784
87158	20041130	NaN	NaN	NaN	233	0.0086	0.0086
87247	20041130	NaN	NaN	NaN	233	0.0046	0.0046
88670	20041130	NaN	NaN	NaN	470	0.0013	0.0013
89186	20041130	NaN	NaN	NaN	331	0.0696	0.0696
89385	20041130	NaN	NaN	NaN	231	0.2959	0.2959
89936	20041130	NaN	NaN	NaN	231	0.0719	0.0719
89939	20041130	NaN	NaN	NaN	233	0.0059	0.0059

1.3 Properties of stock returns

1.3.1 Long-run market averages

Calculate and plot the time series of annual cross-sectional averages of stock returns, where each year's average is cap-weighted, and the final time series is equal-weighted.

Over time, the contribution of dividend yield to total average stock returns has decreased, while share trading turnover has increased.

```
# Loop over the 20-year eras, and compute means of annual cap-weighted averages
years = range(1925, 2025, 20)
results = DataFrame(columns=['divyld', 'turnover', 'means'])
for era in tqdm(years):
    label = f"{era+1}-{era+20}"
    divyld, means, turnover = {}, {}, {}

    for year in bd.date_range(era, min(CRSP_DATE//10000 -1 , era+19), freq=12):
        # universe stocks at end of year
        univ = crsp.get_universe(bd.endyr(year))

        # retrieve cap-weighted average of next year's returns
        cmd = f"""
select permno, SUM(LOG(1+ret)) AS ret FROM daily
WHERE date > {bd.endyr(year)} AND date <= {bd.endyr(year, 1)}
GROUP BY permno
""".strip()
        data = pd.read_sql(cmd, sql.engine)
        df = data.set_index('permno').join(univ['cap'], how='right').dropna()
        means[year] = (np.exp(df['ret'])-1).dot(df['cap']) / df['cap'].sum()

        # retrieve cap-weighted average of annualized turnover
        cmd = f"""
select permno, 252*AVG(vol/(shrount*1000)) AS turnover FROM daily
WHERE date > {bd.endyr(year)} AND date <= {bd.endyr(year, 1)}
GROUP BY permno
"""

```

(continues on next page)

(continued from previous page)

```
""" .strip()
    data = pd.read_sql(cmd, sql.engine)
    df = data.set_index('permno').join(univ['cap'], how='right').dropna()
    turnover[year] = df['turnover'].dot(df['cap']) / df['cap'].sum()

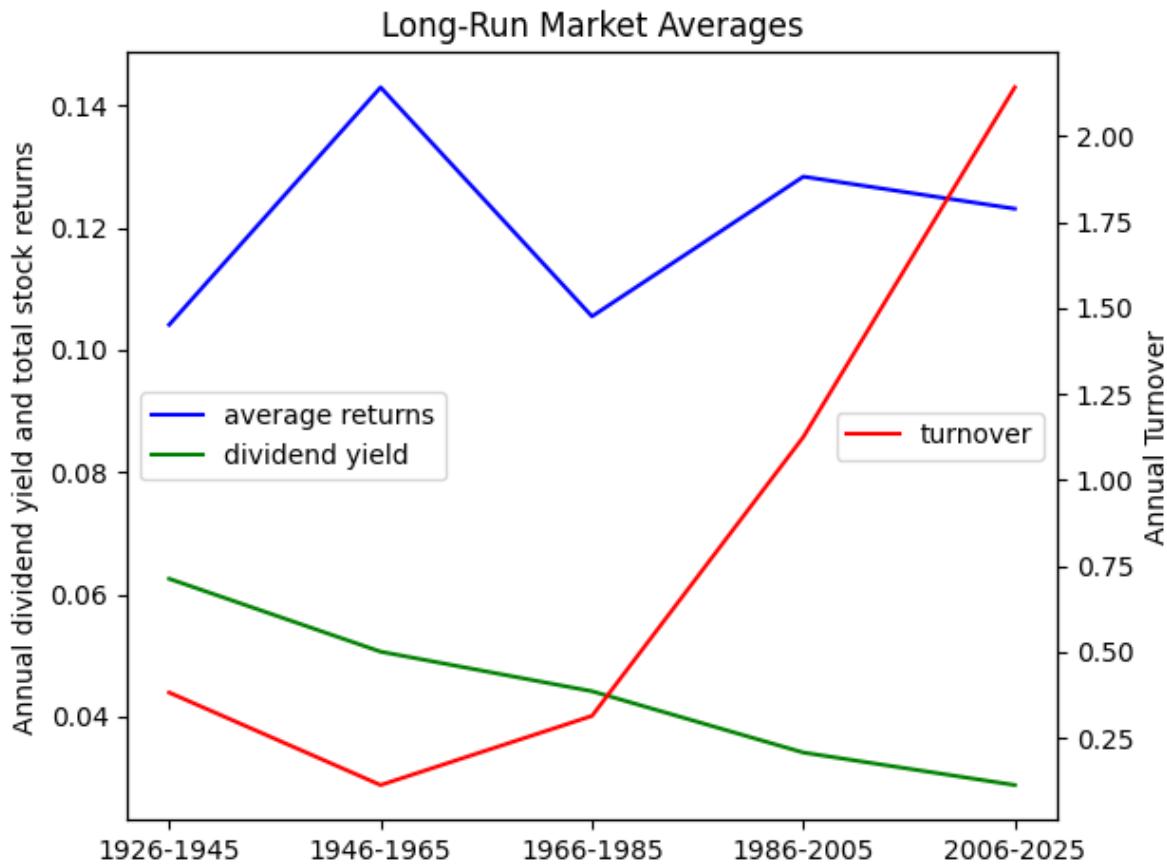
    # retrieve cap-weighted average of annual dividend amounts
    cmd = f"""
SELECT dist.permno as permno, SUM(daily.shrout * dist.divamt) AS divamt
FROM dist INNER JOIN daily
ON daily.permno = dist.permno AND daily.date = dist.exdt
WHERE dist.divamt > 0 AND dist.exdt > {bd.endyr(year)}
    AND dist.exdt <= {bd.endyr(year, 1)}
GROUP BY permno
""" .strip()
    data = pd.read_sql(cmd, sql.engine)
    df = data.set_index('permno').join(univ['cap'], how='right').dropna()
    divyld[year] = df['divamt'].sum() / df['cap'].sum()

    results.loc[label, 'turnover'] = np.mean(list(turnover.values()))
    results.loc[label, 'divyld'] = np.mean(list(divyld.values()))
    results.loc[label, 'means'] = np.mean(list(means.values()))

```

100% |██████████| 5/5 [1:03:37<00:00, 763.46s/it]

```
# Plot mean returns, dividend yield and turnover using both y-axes
fig, ax = plt.subplots()
ax.plot(results['means'], color="blue")
ax.plot(results['divyld'], color="green")
ax.legend(['average returns', 'dividend yield'], loc="center left")
ax.set_ylabel("Annual dividend yield and total stock returns")
bx = ax.twinx()
bx.plot(results['turnover'], color="red")
bx.set_ylabel("Annual Turnover")
bx.legend(['turnover'], loc="center right")
plt.title("Long-Run Market Averages")
plt.tight_layout()
```



1.3.2 Statistical moments

The volatility of an asset is usually measured using the standard deviation of the returns. The common practice is to report the annualized volatility using the square-root rule which assumes that variance scales linearly with time: e.g. daily by $\sqrt{252}$, weekly by $\sqrt{52}$, monthly by $\sqrt{12}$. Mean returns are annualized by multiplying by the respective number of periods in a year.

A normal distribution is symmetric and thin-tailed, and so has no skewness or excess kurtosis. However, many return series are both skewed and fat-tailed (kurtosis in excess of 3). A left-skewed distribution is longer on the left side of its peak than on its right. In other words, a left-skewed distribution has a long tail on its left side, where the mean typically lies to the left of its median. Left skew is also referred to as negative skew. Right or positive skew has the opposite properties.

Stock prices are often modeled with a log-normal distribution because prices cannot be negative. Large positive jumps are possible, but extreme negative moves are bounded at zero, hence the distribution of log-normal returns is positively-skewed with a long right tail. The skewness of a log-normal distribution is given by: $(e^{\sigma^2} + 2)\sqrt{e^{\sigma^2} - 1}$

By Jensen's inequality, the arithmetic mean is greater than the geometric mean. Under the assumption of log-normality, the amount by which the arithmetic mean exceeds the geometric mean of returns is half the volatility.

- Suppose the continuously compounded (log) returns (r_t) are normally distributed with mean (μ) and variance (σ^2): $r_t = \ln\left(\frac{P_t}{P_{t-1}}\right) \sim N(\mu, \sigma^2)$
- The geometric mean return is given by $\mu_G = \frac{1}{T} \sum_{t=1}^T r_t$, with expectation $\mathbb{E}[r_t] = \mu$
- The arithmetic mean of the simple returns is approximately given by $\mu_A \approx e^{\mu + \frac{1}{2}\sigma^2} - 1 \approx \mu_G + \frac{1}{2}\sigma^2$, using the first-order approximation $e^x \approx 1 + x$ for small x .

Jarque-Bera test

The Jarque-Bera test statistic can be used to test whether the sample skewness and kurtosis are compatible with an assumption that the returns are normally distributed. When returns are normally distributed, the skewness is asymptotically normally distributed with a variance of 6, so $(\text{skewness})^2/6$ has a χ_1^2 distribution. Meanwhile, the kurtosis is asymptotically normally distributed with mean 3 and variance of 24, and so $(\text{kurtosis} - 3)^2/24$ also has a χ_1^2 distribution. These two statistics are asymptotically independent (uncorrelated), and so their sum is χ_2^2

```
# Compute the stock returns sampled at various frequencies
intervals = {'annual': (12, 1), 'monthly': ('e', 12), 'weekly': ('w', 52), 'daily': (
    'd', 252)}
moments = []
begdate, enddate = bd.begyr(CRSP_DATE//10000 - 19), bd.endyr(CRSP_DATE//10000)

for dataset, (freq, annualize) in intervals.items():

    # If annual or monthly frequency, use StocksBuffer to pre-load all monthly returns
    if dataset not in ['daily', 'weekly']:
        stocks = CRSPBuffer(stocks=crsp, dataset='monthly', fields=['ret'],
                             beg=bd.begyr(begdate), end=enddate)
    if dataset in ['weekly']:
        # weekly returns already computed from CRSP Daily, and
        # cached
        stocks = crsp

    univ_year = bd.endyr(begdate - 1)  # universe as of end of previous calendar year
    dates = bd.date_range(bd.endyr(begdate), bd.endyr(enddate), freq=freq)
    allstocks = []
    for beg, end in tqdm(bd.date_tuples(dates)):

        # Update the investment universe every calendar year
        if bd.endyr(beg) != univ_year:
            univ = crsp.get_universe(univ_year)
            univ_year = bd.endyr(beg)

        # Use StocksBuffer to cache daily returns for the new calendar year
        if dataset in ['daily']:
            stocks = CRSPBuffer(stocks=crsp, dataset='daily', fields=['ret'],
                                 beg=bd.offset(beg, -1), end=bd.endyr(end))

        # retrieve returns for universe stocks
        ret = stocks.get_ret(beg=beg, end=end).reindex(univ.index)
        allstocks.append(ret.rename(end))

    # combine all years' stock returns, require stock in all years
    allstocks = pd.concat(allstocks, axis=1, join='inner')

    # compute annualized moments, ignoring small sample warnings
    with warnings.catch_warnings():
        warnings.simplefilter("ignore")
        out = {
            "MeanAnnualized": np.nanmedian(
                np.nanmean(np.log(1 + allstocks), axis=1)) * annualize,
            "VolAnnualized": np.nanmedian(
                np.nanstd(np.log(1 + allstocks), axis=1)) * np.sqrt(annualize)
        }
```

(continues on next page)

(continued from previous page)

```

        np.nanstd(allstocks, axis=1, ddof=0)) * np.sqrt(annualize),
        f"Skewness": np.nanmedian(
            scipy.stats.skew(allstocks, nan_policy='omit', axis=1)),
        f"ExcessKurtosis": np.nanmedian(
            scipy.stats.kurtosis(allstocks, nan_policy='omit', axis=1)),
        f"Count": len(allstocks),
    }
df = DataFrame({dataset: Series(out)}).T
moments.append(df)
moments = pd.concat(moments, axis=0) # accumulate df to results
→
moments.reset_index().to_json(outdir / 'moments.json')

```

```

100%|██████████| 19/19 [00:00<00:00, 37.61it/s]
100%|██████████| 228/228 [00:04<00:00, 53.61it/s]
100%|██████████| 991/991 [00:01<00:00, 517.26it/s]
100%|██████████| 4781/4781 [07:05<00:00, 11.23it/s]

```

As we move from annual to daily sampling, stock returns exhibit greater kurtosis (i.e. fat tails) and annualized standard deviation. Skewness is positive – the right tail of the distribution is longer than the left tail – but are U-shaped with daily and annual returns featuring more positive-skewness than weekly or monthly.

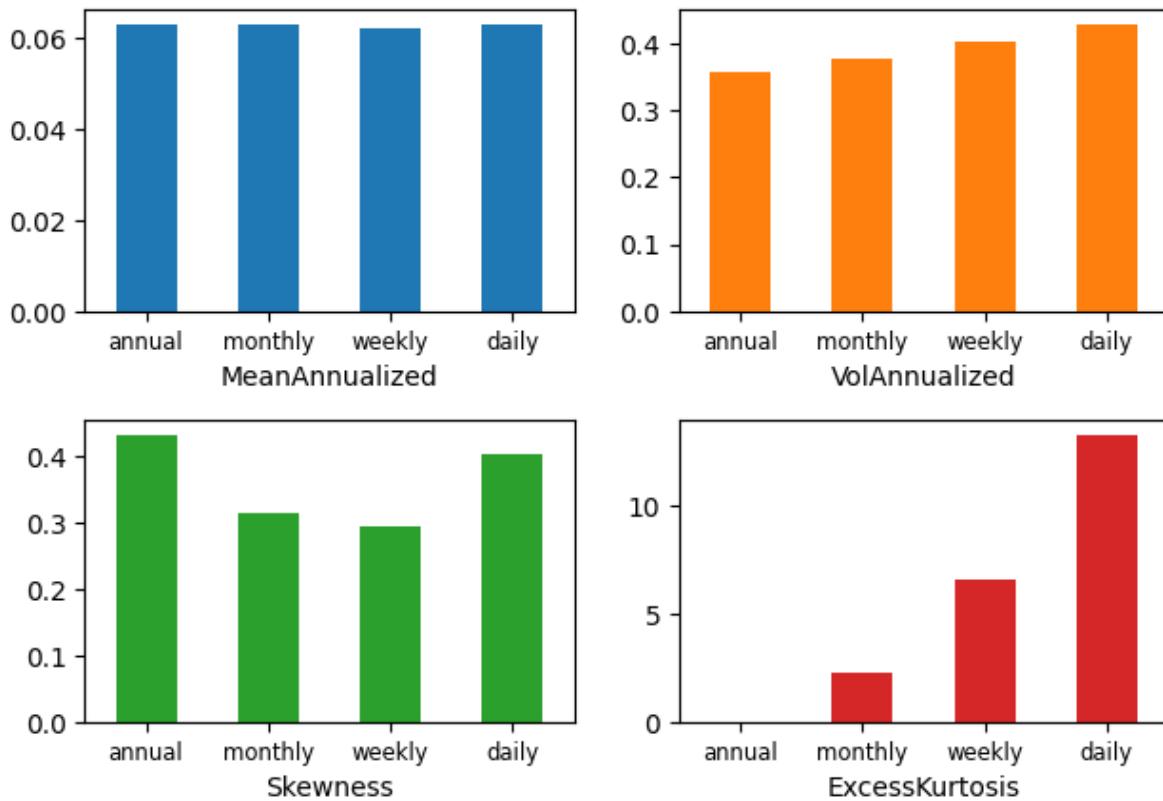
	MeanAnnualized	VolAnnualized	Skewness	ExcessKurtosis	Count
annual	0.063249	0.356397	0.434219	0.002332	1519.0
monthly	0.063249	0.376568	0.314758	2.242009	1519.0
weekly	0.062119	0.402871	0.293850	6.612277	1519.0
daily	0.063325	0.429472	0.404691	13.332600	1519.0

```

fig, axes = plt.subplots(2,2)
for ix, ax in enumerate(axes.flatten()):
    moments[moments.columns[[ix]]].plot.bar(ax=ax, color=f"C{ix}")
    ax.set_xticklabels(moments.index, rotation=0, fontsize='small')
    ax.set_xlabel(moments.columns[[ix]])
plt.suptitle(f"Statistical Moments of Stock Returns {begdate}-{enddate}")
plt.tight_layout()
plt.show()

```

Statistical Moments of Stock Returns 20050103-20241231



1.3.3 Correlations

Pearson's correlation, also known as the linear correlation estimator, measures the strength of a linear relationship between two variables. However, alternative methods can better capture nonlinear dependencies:

- **Spearman's rank correlation** applies Pearson's correlation to the ranked values of observations, measuring monotonic relationships while being less sensitive to outliers.
- **Kendall's τ** quantifies the relationship between two variables by comparing the number of concordant and discordant pairs – pairs that agree or disagree on ordering. It is robust to outliers and effective for skewed or non-normally distributed data.

Monthly SP500 and 30-year Treasury total market returns show positive correlation, which is fairly robust across the three methods.

```
# retrieve SP500 and 30-Year Treasury total market returns from CRSP Indexes
ret = bench.get_series(['sprtrn', 'b30ret(mo)'], field='ret').dropna()
```

permno	sprtrn	b30ret(mo)
date		
19620731	0.006917	-0.008187
19620831	0.007498	0.031939
19620928	0.008965	0.015465

(continues on next page)

(continued from previous page)

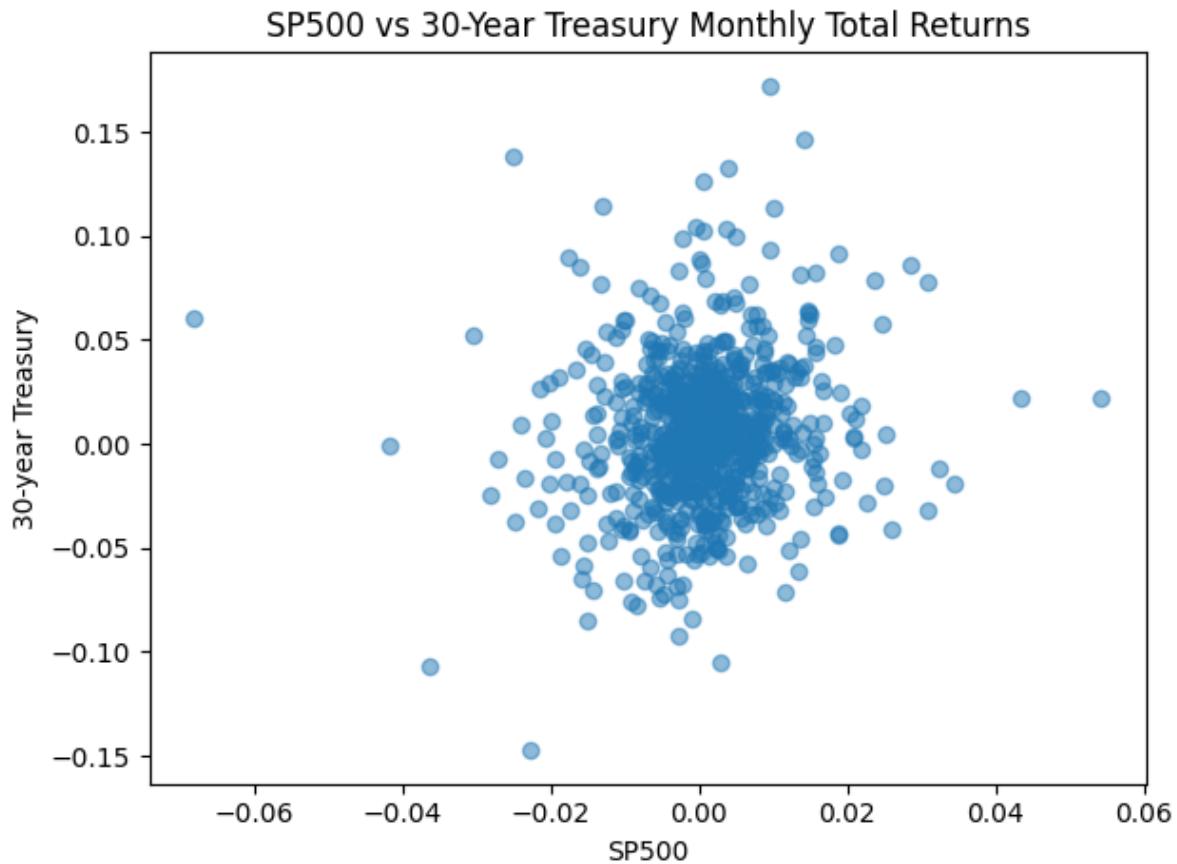
```
19621031 -0.000354    0.012171
19621130 -0.002403    0.003414
...
20240830  0.010093    0.020778
20240930  0.004237    0.015521
20241031 -0.018615   -0.053895
20241129  0.005608    0.021467
20241231 -0.004285   -0.063115
```

```
[749 rows x 2 columns]
```

```
# Compute alternative measures of correlation
DataFrame(dict(spearman=scipy.stats.spearmanr(ret['sprtrn'], ret['b30ret(mo)'])[0],
               kendall=scipy.stats.kendalltau(ret['sprtrn'], ret['b30ret(mo)'])[0],
               pearson=scipy.stats.pearsonr(ret['sprtrn'], ret['b30ret(mo)'])[0]),
               index=['Correlation of SP500 vs 30-Year Treasury Total Returns']).round(4)
```

	spearman	kendall	pearson
Correlation of SP500 vs 30-Year Treasury Total ...	0.1348	0.0933	0.1275

```
# Scatter plot of SP500 and and 30-year Treasury total market returns
fig, ax = plt.subplots()
ax.scatter(ret['sprtrn'], ret['b30ret(mo)'], alpha=0.5)
ax.set_xlabel('SP500')
ax.set_ylabel('30-year Treasury')
plt.title('SP500 vs 30-Year Treasury Monthly Total Returns')
plt.tight_layout()
```



APPENDIX

SQL commands

- Manage tables
 - CREATE DATABASE – Creates a new database.
 - CREATE TABLE – Creates a new table.
 - DELETE – Delete data from a table.
 - DROP COLUMN – Deletes a column from a table.
 - DROP DATABASE – Deletes the entire database.
 - DROP TABLE – Deletes a table from a database.
 - TRUNCATE TABLE – Deletes the data but does not delete the table.
- **Querying a table**
 - SELECT – Used to select data from a database, which is then returned in a results set.
 - SELECT DISTINCT – Same as SELECT, except duplicate values are excluded.
 - SELECT INTO – Copies data from one table and inserts it into another.
 - UNIQUE – This constraint ensures all values in a column are unique.
 - FROM – Specifies which table to select or delete data from.
 - AS – Renames a table or column with an alias value which only exists for the duration of the query.

- **Query conditions**

- WHERE – Filters results to only include data which meets the given condition.
- AND – Used to join separate conditions within a WHERE clause.
- BETWEEN – Selects values within the given range.
- IS NULL – Tests for empty (NULL) values.
- IS NOT NULL – The reverse of NULL. Tests for values that aren't empty / NULL.
- LIKE – Returns true if the operand value matches a pattern.
- NOT – Returns true if a record DOESN'T meet the condition.
- OR – Used alongside WHERE to include data when either condition is true.

- **Organize results**

- ORDER BY – Used to sort the result data in ascending (default) or descending order through the use of ASC or DESC keywords.
- GROUP BY – Used alongside aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the results.

- **Join tables**

- INNER JOIN returns rows that have matching values in both tables.
- LEFT JOIN returns all rows from the left table, and the matching rows from the right table.-
- RIGHT JOIN returns all rows from the right table, and the matching records from the left table
- FULL OUTER JOIN returns all rows when there is a match in either left table or right table.

References:

Fama, Eugene F., and Kenneth R. French. 1992. "The cross-section of expected stock returns." *The Journal of Finance* 47 (2): 427–65.

Shumway, Tyler. 1997. The Delisting Bias in CRSP Data. *The Journal of Finance*, 52, 327-340.

Bali, Turan G, Robert F Engle, and Scott Murray. 2016. *Empirical asset pricing: The cross section of stock returns*. John Wiley & Sons.

Stulz, René M., 2018, "The Shrinking Universe of Public Firms: Facts, Causes, and Consequences", National Bureau of Economics, 2 (June 2018)

FRM Exam Book Part I Quantitative Analysis Chapter 12

JEGADEESH-TITMAN ROLLING PORTFOLIOS

The future is just more of the past waiting to happen - Fred D'Aguiar.

The Jegadeesh-Titman rolling portfolios approach explores the phenomenon of price momentum in financial markets, focusing on strategies that involve buying stocks with recent strong performance and selling stocks with weak performance. Univariate spread portfolios are constructed, which help isolate the return differences between high- and low-ranked stocks. The following analysis covers key aspects such as overlapping and non-overlapping portfolio returns, the impact of autocorrelation on variance estimation, and statistical hypothesis testing. Additionally, it discusses the Newey-West correction for standard errors and evaluates the power of hypothesis tests.

```
import math
import numpy as np
import pandas as pd
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
import scipy
from scipy.stats import kurtosis, skew, norm
import statsmodels.formula.api as smf
import statsmodels.api as sm
from tqdm import tqdm
from finds.database import SQL, RedisDB
from finds.structured import BusDay, CRSP, CRSPBuffer
from finds.recipes import fractile_split
from finds.utils import plot_date
from secret import credentials, CRSP_DATE
```

```
VERBOSE = 0
#%matplotlib qt
```

```
# open connections
sql = SQL(**credentials['sql'], verbose=VERBOSE)
rdb = RedisDB(**credentials['redis'])
bd = BusDay(sql, verbose=VERBOSE)
crsp = CRSP(sql, bd, rdb=rdb, verbose=VERBOSE)
```

```
# date range and parameters to construct momentum portfolios
begrebal = 19260630          # first price date is 19251231
endrebal = bd.endmo(CRSP_DATE, -1)    # last rebal is one month before last CRSP
rebaldates = bd.date_range(begrebal, endrebal, 'endmo')
percentiles = [20, 80]    # quintile spread percentile breakpoints
maxhold = 6      # hold each monthly-rebalanced portfolio for 6 months
```

```
# preload monthly dataset into memory
monthly = CRSPBuffer(stocks=crsp, dataset='monthly',
                      fields=['ret', 'retx', 'prc'],
                      beg=bd.begmo(rebaldates[0], -6),
                      end=bd.endmo(rebaldates[-1], 1))
```

2.1 Price Momentum

2.1.1 Overlapping portfolio returns

First, we estimate the six-month returns of a momentum strategy by averaging monthly observations.

At the end of each month t , we calculate the sorting variable as the past six-month return of all stocks in the investment universe. The 20th and 80th percentiles of NYSE-listed stocks serve as thresholds: we go long on stocks in the top fractile and short those in the bottom fractile. Each fractile is weighted by market capitalization, while the spread portfolio return is the equal-weighted difference between the two sub-portfolios. The spread portfolio's returns over the next six months are recorded on a monthly basis by dividing by six.

A stock is eligible for inclusion if it meets the usual investment universe criteria at the end of the rebalance month and has a non-missing month-end price from six months prior.

```
stocks = monthly
mom = []
for rebaldate in tqdm(rebaldates):
    # determine pricing dates relative to rebaldate
    beg = bd.endmo(rebaldate, -6)    # require price at beg date
    end = bd.endmo(rebaldate, 0)      # require price at end date
    start = bd.offset(beg, 1)         # starting day of momentum signal

    # retrieve universe, prices, and momentum signal
    p = [crsp.get_universe(rebaldate),
          stocks.get_ret(beg=start, end=end).rename('mom'),
          stocks.get_section(fields=['prc'], date=beg)['prc'].rename('beg')]
    df = pd.concat(p, axis=1, join='inner').dropna()

    # quintile breakpoints are determined from NYSE subset
    tritile = fractile_split(values=df['mom'],
                              pct=percentiles,
                              keys=df.loc[df['nyse'], 'mom'])

    # construct cap-wtd tritile spread portfolios
    porthi, portlo = [df.loc[tritile==t, 'cap'] for t in [1, 3]]
    port = pd.concat((porthi/porthi.sum(), -portlo/portlo.sum()))

    # compute and store cap-weighted average returns over (up to) maxhold periods
    begret = bd.offset(rebaldate, 1)
    nhold = min(maxhold, len(rebaldates) - rebaldates.index(rebaldate))
    endret = bd.endmo(begret, nhold - 1)    # if maxhold is beyond end date
    rets = monthly.get_ret(begret, endret)
    ret = rets.reindex(port.index).fillna(0.).mul(port, axis=0).sum()
    mom.append(float(ret) / nhold)
```

0% | 0/1182 [00:00<?, ?it/s]

100% |██████████| 1182/1182 [11:41:22<00:00, 35.60s/it]

```
DataFrame({'mean': np.mean(mom), 'std': np.std(mom)}, index=['Overlapping Returns'])
```

	mean	std
Overlapping Returns	0.004526	0.024599

2.1.2 Non-overlapping portfolio returns

A spread portfolio is constructed at the end of each month in the same manner. However, instead of overlapping returns, the return recorded is the equal-weighted average of the following month's returns from six distinct portfolios formed between t and $t - 5$. Each month, the weights of stocks in the spread portfolios adjust according to their price changes, following a “buy-and-hold” approach over six months.

```
ports = [] # to roll 6 past portfolios
jt = []
stocks = monthly
for rebaldate in tqdm(rebaldates):

    # determine returns dates relative to rebaldate
    beg = bd.endmo(rebaldate, -6) # require price at beg date
    end = bd.endmo(rebaldate, 0) # require price at end date
    start = bd.offset(beg, 1) # starting day of momentum signal

    # retrieve universe, prices, and momentum signal
    p = [crsp.get_universe(rebaldate),
          stocks.get_ret(beg=start, end=end).rename('mom'),
          stocks.get_section(fields=['prc'], date=beg) ['prc'].rename('beg')]
    df = pd.concat(p, axis=1, join='inner').dropna()

    # quintile breakpoints determined from NYSE subset
    tritile = fractile_split(values=df['mom'],
                              pct=percentiles,
                              keys=df.loc[df['nyse'], 'mom'])

    # construct cap-wtd tritile spread portfolios
    porthi, portlo = [df.loc[tritile==t, 'cap'] for t in [1, 3]]
    port = pd.concat((porthi/porthi.sum(), -portlo/portlo.sum()))

    # retain up to 6 prior months of monthly-rebalanced portfolios
    ports.insert(0, port)
    if len(ports) > maxhold:
        ports.pop(-1)

    # compute all 6 portfolios' monthly capwtd returns, and store eqlwtd average
    begret = bd.offset(rebaldate, 1)
    endret = bd.endmo(begret)
    rets = stocks.get_ret(begret, endret)
    ret = np.mean([rets.reindex(p.index).fillna(0.).mul(p, axis=0).sum()
                  for p in ports])
    jt.append(ret)

    # adjust stock weights by monthly capital appreciation
```

(continues on next page)

(continued from previous page)

```
retx = stocks.get_ret(begret, endret, field='retx')
ports = [(1 + retx.reindex(p.index).fillna(0)).mul(p, axis=0)
          for p in ports]

DataFrame({'mean': np.mean(jt), 'std': np.std(jt)}, index=['Non-overlapping Returns'])
```

0% | 0/1182 [00:00<?, ?it/s]

100% |██████████| 1182/1182 [03:30<00:00, 5.62it/s]

	mean	std
Non-overlapping Returns	0.004502	0.051419

Correlation with lagged returns

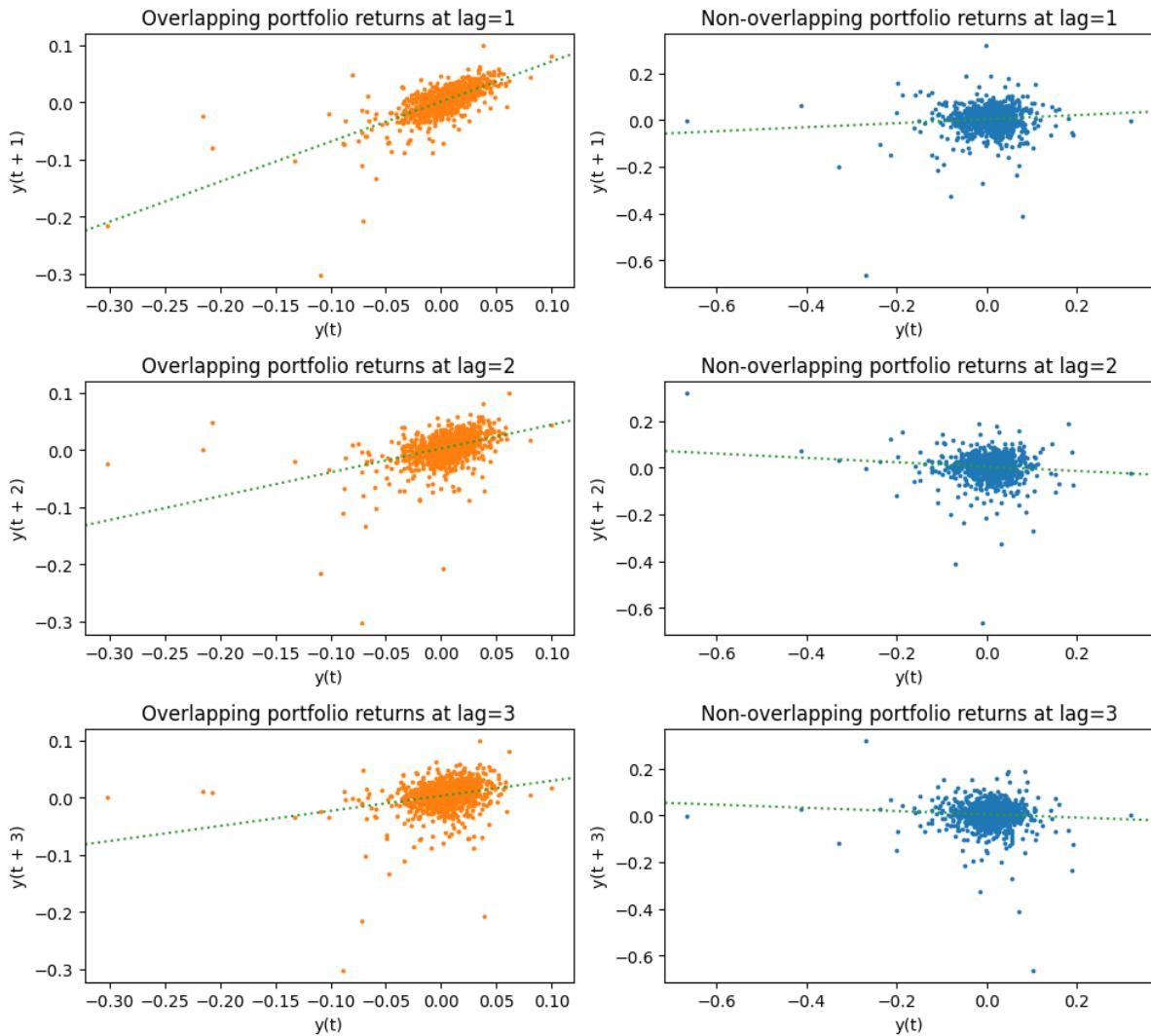
For the overlapping portfolios, each month's recorded return is (one-sixth of) a six-month return. Let r_t be the return at time t . The 6-month return at time t , denoted as R_t , is the sum of the past 6 monthly returns: $R_t = r_t + r_{t-1} + r_{t-2} + r_{t-3} + r_{t-4} + r_{t-5}$

Since we sample monthly, consecutive returns R_t and R_{t+1} overlap significantly. Up to 5/6 of adjacent months' returns actually reflect the same month's stock returns. Even returns recorded five months apart share one month of stock returns in common. Ignoring this overlap when estimating variance leads to underestimation of the true variance.

The Jegadeesh-Titman non-overlapping portfolio approach eliminates this issue.

```
fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(10, 9))
for lag, ax in zip(range(1, axes.shape[0]+1, 1), axes):
    pd.plotting.lag_plot(Series(mom), lag=lag, ax=ax[0], s=3, c="C1")
    ax[0].set_title(f"Overlapping portfolio returns at lag={lag}")
    r = scipy.stats.linregress(mom[lag:], mom[:-lag])
    ax[0].axline((0, r.intercept), slope=r.slope, ls=':', color="C2")

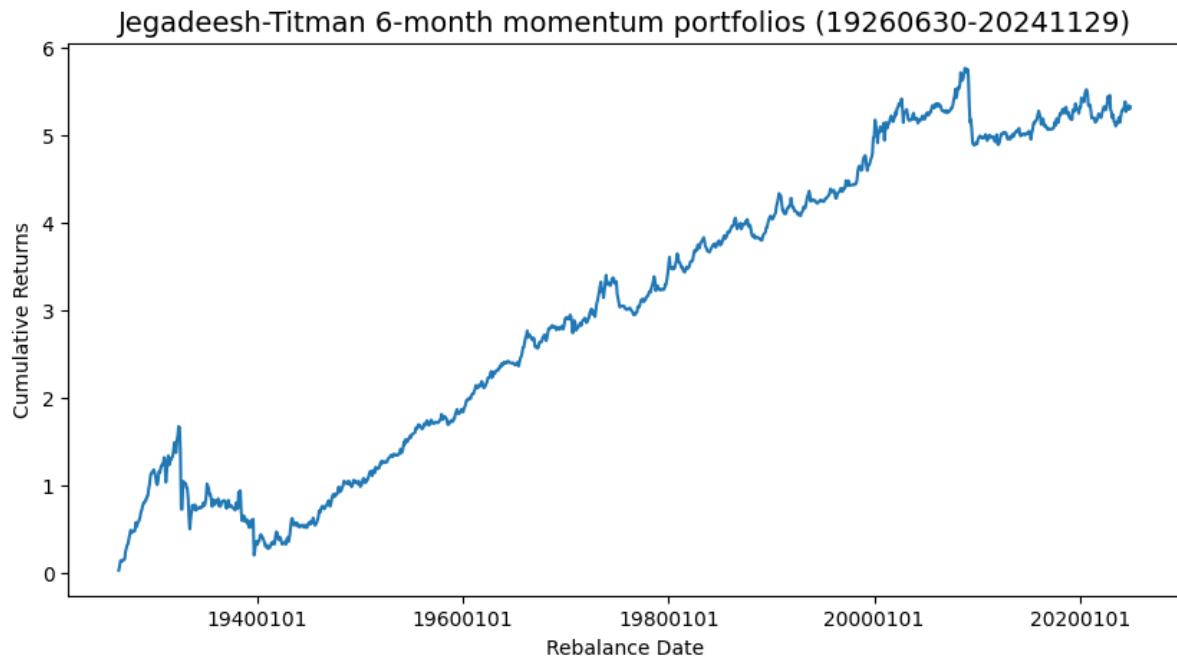
    pd.plotting.lag_plot(Series(jt), lag=lag, ax=ax[1], s=3, c="C0")
    ax[1].set_title(f"Non-overlapping portfolio returns at lag={lag}")
    r = scipy.stats.linregress(jt[lag:], jt[:-lag])
    ax[1].axline((0, r.intercept), slope=r.slope, ls=':', color="C2")
plt.tight_layout()
```



Plot cumulative monthly average returns

Jegadeesh-Titman non-overlapping 6-month momentum portfolio cumulative returns

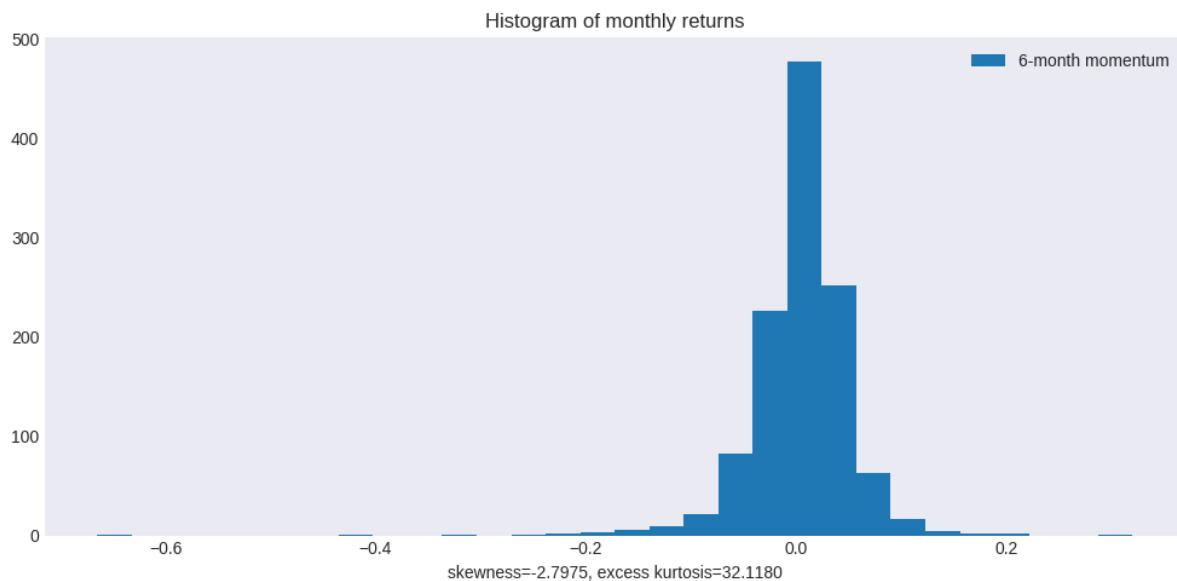
```
fig, ax = plt.subplots(figsize=(10, 5), clear=True)
plot_date(DataFrame(index=rebaldates, data=np.cumsum(jt), columns=['momentum']),
          ax=ax, fontsize=10, rotation=0,
          ylabel1='Cumulative Returns', xlabel='Rebalance Date',
          title=f'Jegadeesh-Titman 6-month momentum portfolios ({rebaldates[0]}-{rebaldates[-1]})')
plt.show()
```



Plot histogram of monthly returns

Distribution of Jegadeesh-Titman non-overlapping 6-month momentum portfolio returns

```
fig, ax = plt.subplots(1, 1, clear=True, figsize=(10, 5))
ax.hist(jt, bins=30)
ax.set_title(f"Histogram of monthly returns")
ax.legend(['6-month momentum'])
kurt = kurtosis(jt, bias=True, fisher=True) # excess kurtosis
skewness = skew(jt, bias=True)
ax.set_xlabel(f"skewness={skewness:.4f}, excess kurtosis={kurt:.4f}")
plt.tight_layout()
```



2.2 Hypothesis testing

A hypothesis test makes a precise statement about population parameters and evaluates the likelihood of observing the data under a given assumption.

- The *null hypothesis* specifies the true value of a parameter to be tested, often $H_0 : \hat{\mu} = \mu_0$
- The *test statistic* is a summary of the observed data that has a known distribution when the null hypothesis is true, e.g. $T - \frac{\hat{\mu} - \mu_0}{\sqrt{\sigma^2/n}} \sim N(0, 1)$
- The *alternative hypothesis* defines the range of values of the parameter where the null should be rejected, e.g. $H_a : \hat{\mu} \neq \mu_0$
 - In some testing problems, the alternative hypothesis is not the full complement of the null, for example, a *one-sided alternative* $H_a : \hat{\mu} > \mu_0$, which is used when the outcome of interest is only above or below the value assumed by the null.
- The *critical value* C_α marks the start of a range of values where the test statistic is unlikely to fall in, if the null hypothesis were true, e.g. $C_\alpha = \Phi^{-1}(1 - \alpha/2) = 1.96$ when $\alpha = 5\%$ for a two-sided test. This range is known as the *rejection region*.
- The *size* of the test is the probability of making a *Type I error* of rejecting null hypothesis that is actually true. A test is said to have *significance level* α if its *size* is less than or equal to α . This reflects the aversion to rejecting a null hypothesis that is, in fact, true.
- The *p-value* is the probability of obtaining a test statistic at least as extreme as the one we observed from the sample, if the null hypothesis were true, e.g. $p = 2(1 - \Phi(|T|))$ for a two-sided test.

2.2.1 Confidence Interval

A $1 - \alpha$ *confidence interval* contains the values surrounding the test statistic that cannot be rejected when using a test size of α , e.g. $[\hat{\mu} - C_\alpha \frac{\sigma^2}{\sqrt{n}}, \hat{\mu} + C_\alpha \frac{\sigma^2}{\sqrt{n}}]$ for a two-sided interval

2.2.2 Newey-West corrected t-stats

Standard errors are underestimated when assuming independent observations, as this assumption does not hold for overlapping returns. The **Newey-West (1987) estimator** corrects for heteroskedasticity and autocorrelation by specifying a “maximum lag” for autocorrelation control. A common choice is $L =$ the fourth root of the number of observations (e.g., Greene, *Econometric Analysis*, 7th ed., p. 960).

Applying the Newey-West correction nearly doubles the estimated standard error for overlapping portfolios, but it has a minimal effect on non-overlapping returns.

```
print('n =', len(mom), 'L =', math.ceil(len(mom)**(1/4)))
results = []
for rets, label in zip([mom, jt], ['Overlapping', 'Non-overlapping']):
    data = DataFrame(rets, columns=['ret'])

    # raw t-stats
    reg = smf.ols('ret ~ 1', data=data).fit()
    uncorrected = Series({stat: round(float(getattr(reg, stat).iloc[0]), 6)
                         for stat in ['params', 'bse', 'tvalues', 'pvalues']},
                         name='uncorrected') # coef, stderr, t-value, P>/z/
```

(continues on next page)

(continued from previous page)

```
# Newey-West correct t-stats
reg = smf.ols('ret ~ 1', data=data) \
    .fit(cov_type='HAC', cov_kwds={'maxlags': 6})
corrected = Series({stat: round(float(getattr(reg, stat).iloc[0]), 6)
                    for stat in ['params', 'bse', 'tvalues', 'pvalues']},
                   name='NeweyWest') # coef, stderr, t-value, P>/z

# merge into intermediate dataframe with multicolumn index
df = pd.concat([uncorrected, corrected], axis=1)
df.columns = pd.MultiIndex.from_product(([label], df.columns))
results.append(df)

pd.concat(results, axis=1).rename_axis('Standard Errors')
```

n = 1182 L = 6

	Overlapping		Non-overlapping	
	uncorrected	NeweyWest	uncorrected	NeweyWest
Standard Errors				
params	0.004526	0.004526	0.004502	0.004502
bse	0.000716	0.001285	0.001496	0.001463
tvalues	6.322644	3.522659	3.009069	3.078432
pvalues	0.000000	0.000427	0.002676	0.002081

2.2.3 Power of Test

A **Type II error** occurs when the alternative hypothesis is true but the null is not rejected. The probability of a Type II error is denoted by β , while **power** ($1 - \beta$) represents the probability of correctly rejecting a false null hypothesis.

Unlike test size, the power of a test depends on:

1. Sample size
2. Test size (α)
3. The distance between the true parameter value and the null hypothesis value

For a one-sided test $H_a : \hat{\mu} > \mu_0$, power is given by:

$$1 - \beta(\alpha) = \Phi \left(C_\alpha \frac{\sigma^2}{\sqrt{n}} \middle| \mu_a, \frac{\sigma^2}{\sqrt{n}} \right)$$

```
DataFrame(data={"True Null": ['correct', '(1 - alpha)', 'Type I Error', 'Size: (alpha)']
                ↵'],
              "False Null": ['Type II Error', '(beta)', 'correct', 'Power: (1-beta)']
                ↵'],
              index=['Accept Null', '', 'Reject Null', '']) \
    .rename_axis(index='Decision')
```

Decision	True Null	False Null
	Accept Null	correct

(continues on next page)

(continued from previous page)

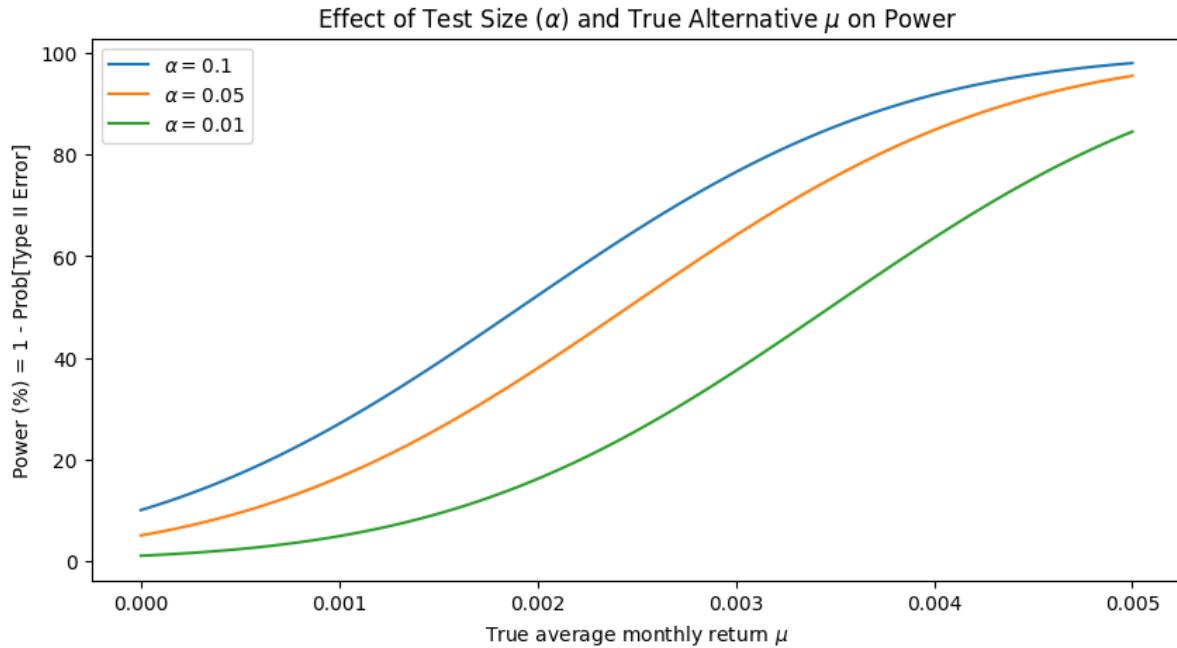
Reject Null	(1 - alpha)	(beta)
	Type I Error	correct
	Size: (alpha)	Power: (1-beta)

Effect of Test Size (alpha) and True Alternative (mu) on Power

```
# Assumptions
alternative = 0.06    # alternative hypothesis that annualized mean is as large as 6%
scale = np.std(jt) / np.sqrt(len(jt))  # assumed scale (std dev or volatility)
```

```
# Vary test size (alpha) and true mean (mu)
mu = np.linspace(0, alternative/12, 100)    # vary true mean
plt.figure(figsize=(10, 5))
for alpha in [0.1, 0.05, 0.01]:                # vary test size
    power = 1 - norm.cdf(norm.ppf(1 - alpha) * scale, loc=mu, scale=scale)
    plt.plot(mu, 100*power, label=f"$\alpha={alpha}$")
plt.title("Effect of Test Size ($\alpha$) and True Alternative ($\mu$) on Power")
plt.ylabel('Power (%) = 1 - Prob[Type II Error]')
plt.xlabel('True average monthly return $\mu$')
plt.legend()
```

<matplotlib.legend.Legend at 0x7f12c13d0950>



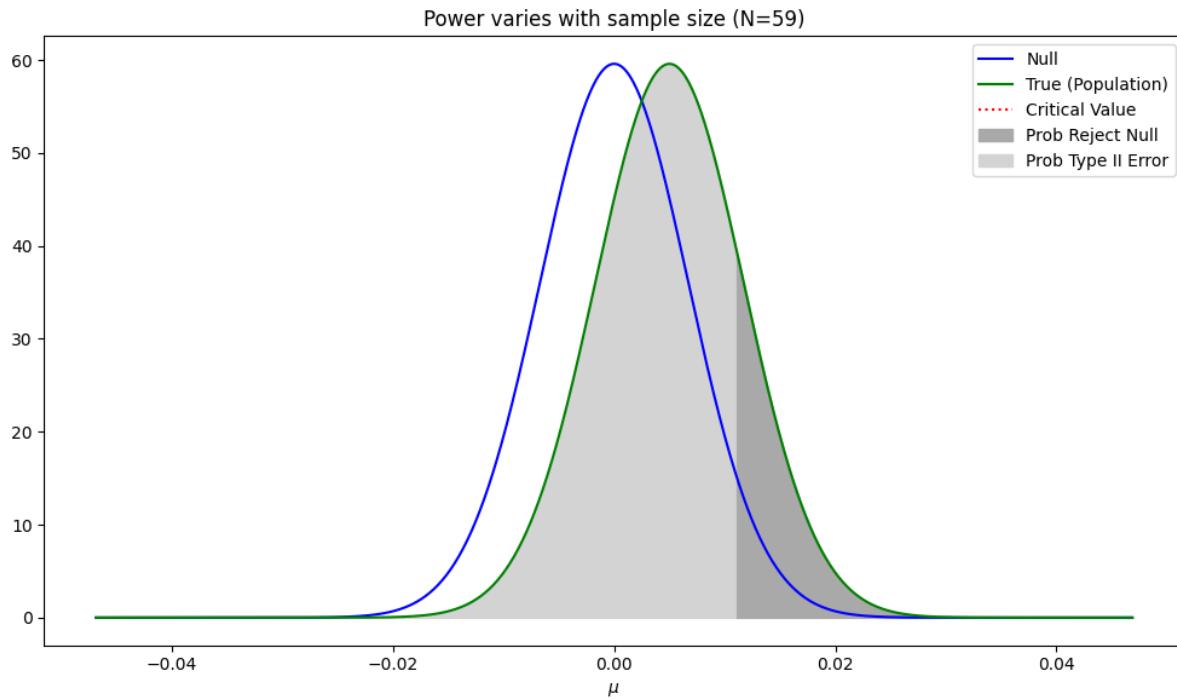
Effect of Sample Size on Power

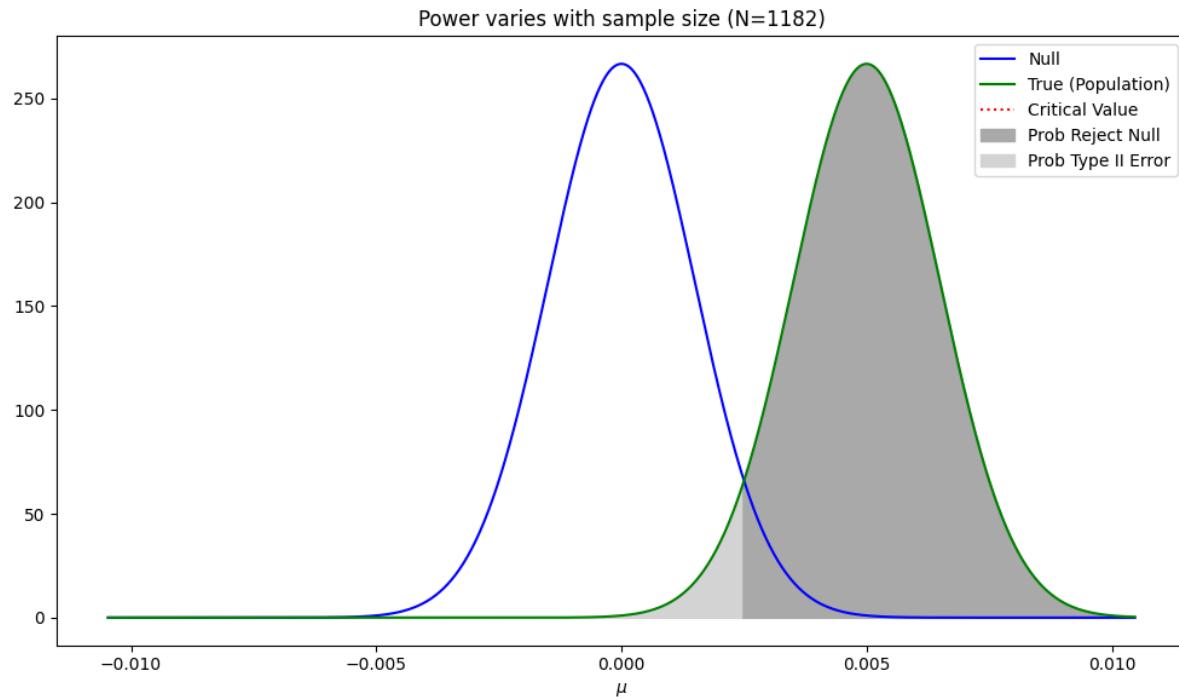
```
# Assumptions
volatility = np.std(jt)
alternative = 0.06/12    # mean of the alternate hypothesis
alpha = 0.05             # desired size of the test
```

```
# Compare large and small sample sizes
for N in [len(jt) // 20, len(jt)]:

    # define null and alternate distributions given sample size
    scale = volatility/np.sqrt(N)    # scaled by square root of sample size
    null_dist = norm(0, scale)
    alt_dist = norm(alternative, scale)
    critical_val = null_dist.ppf(1-alpha) # critical value to reject null

    fig, ax = plt.subplots(figsize=(10, 6))
    x = np.linspace(-7 * scale, 7 * scale, 1000)
    ax.plot(x, null_dist.pdf(x), color='blue') # plot null distribution
    ax.plot(x, alt_dist.pdf(x), color='green') # plot alt distribution
    ylim = plt.ylim()[0]
    ax.axvline(x=critical_val, ymax=ylim, ls=':', color='r') # critical value
    px = x[x > critical_val]
    ax.fill_between(px, alt_dist.pdf(px), color='darkgrey') # rejection region
    px = x[x < critical_val]
    ax.fill_between(px, alt_dist.pdf(px), color='lightgrey') # acceptance region
    ax.set_title(f"Power varies with sample size (N={N})")
    ax.set_xlabel("$\mu$")
    plt.legend(['Null', 'True (Population)', 'Critical Value',
               'Prob Reject Null', 'Prob Type II Error'])
    plt.tight_layout()
```





References:

Jegadeesh, Narasimhan, and Sheridan Titman (1993), "Returns to Buying Winners and Selling Losers: Implications for Stock Market Efficiency". *Journal of Finance*. March 1993, Volume 48, Issue 1, Pages 65-91.

Newey, Whitney K, West, Kenneth D (1987). "A Simple, Positive Semi-definite, Heteroskedasticity and Autocorrelation Consistent Covariance Matrix". *Econometrica*. 55 (3): 703–708.

Hong, Harrison, Terence Lim, Jeremy C. Stein, 2000, "Bad News Travels Slowly: Size, Analyst Coverage, and the Profitability of Momentum Strategies", Volume 55, Issue 1, Pages 265-295. <https://doi.org/10.1111/0022-1082.00206>

FRM Exam Book Part I Quantitative Analysis Chapter 6

MIT License. Copyright 2021-2025, Terence Lim

FAMA-FRENCH PORTFOLIO SORTS

The way to become rich is to put all your eggs in one basket and then watch that basket - Andrew Carnegie

The Fama-French portfolio sorting methodology is widely used in empirical asset pricing research, particularly in understanding the cross-section of stock returns. By classifying stocks based on fundamental characteristics such as book-to-market ratio and firm size, this approach provides insights into the risk and return dynamics of different investment strategies. This notebook also includes linear regression analysis to assess factor exposures, tests for the value and small-firm effects, and a structural break analysis using the Chow test.

```
import numpy as np
import scipy
from scipy.stats import skew, kurtosis
import pandas as pd
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
import statsmodels.formula.api as smf
from finds.database import SQL, RedisDB
from finds.structured import BusDay, CRSP, CRSPBuffer, Signals, Benchmarks, PSTAT
from finds.utils import plot_date
from finds.backtesting import bivariate_sorts, BackTest
from finds.utils import plot_date, plot_scatter, plot_hist
from tqdm import tqdm
from secret import credentials, CRSP_DATE
```

```
VERBOSE = 0
# %matplotlib qt
```

```
sql = SQL(**credentials['sql'], verbose=VERBOSE)
user = SQL(**credentials['user'], verbose=VERBOSE)
rdb = RedisDB(**credentials['redis'])
bd = BusDay(sql, verbose=VERBOSE)
crsp = CRSP(sql, bd, rdb=rdb, verbose=VERBOSE)
pstat = PSTAT(sql, bd, verbose=VERBOSE)
signals = Signals(user, verbose=VERBOSE)
bench = Benchmarks(sql, bd, verbose=VERBOSE)
backtest = BackTest(user, bench, rf='RF', max_date=CRSP_DATE, verbose=VERBOSE)
LAST_DATE = bd.endmo(CRSP_DATE, -1) # last monthly rebalance date
```

3.1 Stock fundamentals data

3.1.1 Compustat

Compustat is a database containing financial statements and market data for both active and inactive U.S. and international companies. It is commonly used in academic and industry research.

To compute book-to-price ratios from financial statements, we:

- Extract balance sheet items from the Compustat Annual dataset.
- Construct the High Minus Low (HML) factor by calculating book equity as shareholders' equity plus investment tax credits, minus preferred stock, divided by the market capitalization at the end of December.
- Apply a six-month reporting lag and require at least two years of history in Compustat.
- Exclude deferred taxes and investment tax credits from book equity for fiscal years ending in 1993 or later, following FASB 109, which improved the accounting treatment for deferred income taxes.

```
label = 'hml'
lag = 6                      # number of months to lag fundamental data
# retrieve data fields from compustat, linked by permno
df = pstat.get_linked(dataset = 'annual',
                      date_field = 'datadate',
                      fields = ['seq', 'pstk', 'pstkrv', 'pstkl', 'txditz'],
                      where = ("indfmt = 'INDL'" +
                               " AND datafmt = 'STD'" +
                               " AND curcd = 'USD'" +
                               " AND popsrc = 'D'" +
                               " AND consol = 'C'" +
                               " AND seq > 0"))
```

```
# subtract preferred stock
df[label] = np.where(df['pstkrv'].isna(), df['pstkl'], df['pstkrv'])
df[label] = np.where(df[label].isna(), df['pstk'], df[label])
df[label] = np.where(df[label].isna(), 0, df[label])
```

```
# do not add back deferred investment tax credit for fiscal years in 1993 or later
df[label] = (df['seq'] - df[label]
            + df['txditz'].fillna(0).where(df['datadate'] // 10000 <= 1993, 0))
df.dropna(subset = [label], inplace=True)
df = df[df[label] > 0][['permno', 'gvkey', 'datadate', label]]
```

```
# count years in Compustat
df = df.sort_values(by=['gvkey', 'datadate'])
df['count'] = df.groupby(['gvkey']).cumcount()
```

```
# construct b/m ratio
df['rebaldate'] = 0
for datadate in tqdm(sorted(df['datadate'].unique())):
    f = df['datadate'].eq(datadate)
    rebaldate = bd.endmo(datadate, abs(lag)) # 6 month lag
    capdate = bd.endyr(datadate) # Dec mktcap
    if rebaldate >= CRSP_DATE or capdate >= CRSP_DATE:
        continue
```

(continues on next page)

(continued from previous page)

```

df.loc[f, 'rebaldate'] = rebaldate
df.loc[f, 'cap'] = crsp.get_cap(capdate, use_permco=True) \
    .reindex(df.loc[f, 'permno']) \
    .values
df[label] /= df['cap']
df = df[df[label].gt(0) & df['count'].gt(1)] # 2+ years in Compustat
signals.write(df, label)

```

100% |██████████| 758/758 [00:02<00:00, 317.69it/s]

227642

3.2 Bivariate sorts

Independent bivariate sorts categorize stocks based on two characteristics: book-to-market ratio and market capitalization. Portfolios are formed at the end of each June and represent the intersections of:

- Two groups sorted by size (market equity, ME).
- Three groups sorted by book-to-market ratio (BE/ME).

The size breakpoint for year t is the median NYSE market equity at the end of June in that year. Stocks within each of the six resulting portfolios are weighted by market capitalization.

The two key factors derived from these sorts are:

- **HML (High Minus Low):** The equal-weighted average return of the two value portfolios minus the average return of the two growth portfolios.
- **SMB (Small Minus Big):** The equal-weighted average return of the three small-size portfolios minus the average return of the three large-size portfolios.

Causal Analysis

This sorting approach has conceptual parallels with causal analysis techniques. Specifically, propensity score matching is often used in statistical research to mitigate confounding effects when estimating treatment effects. Propensity scores, estimated via logistic regression, allow researchers to:

- **Stratify** subjects into groups based on similar propensity scores.
- **Match** treated and control subjects with comparable propensity scores.
- **Adjust** for imbalances using regression models.

Since firm size directly influences the book-to-market ratio (as its denominator), applying bivariate sorting ensures that value returns are estimated while controlling for the small-firm effect—similar to how propensity score matching controls for confounding variables in observational studies.

3.2.1 HML

Compute High Minus Low book-to-price monthly returns and compare to Fama-French research factor

```
label, benchname = 'hml', 'HML (mo)'
rebalend = LAST_DATE
rebalbeg = 19700101
```

```
# preload monthly dataset into memory
monthly = CRSPBuffer(stocks=crsp, dataset='monthly',
                      fields=['ret', 'retx', 'prc'],
                      beg=19251201, end=CRSP_DATE)
```

```
hml, smb = bivariate_sorts(stocks=monthly,
                           label=label,
                           signals=signals,
                           rebalbeg=rebalbeg,
                           rebalend=rebalend,
                           window=12,
                           months=[6])
```

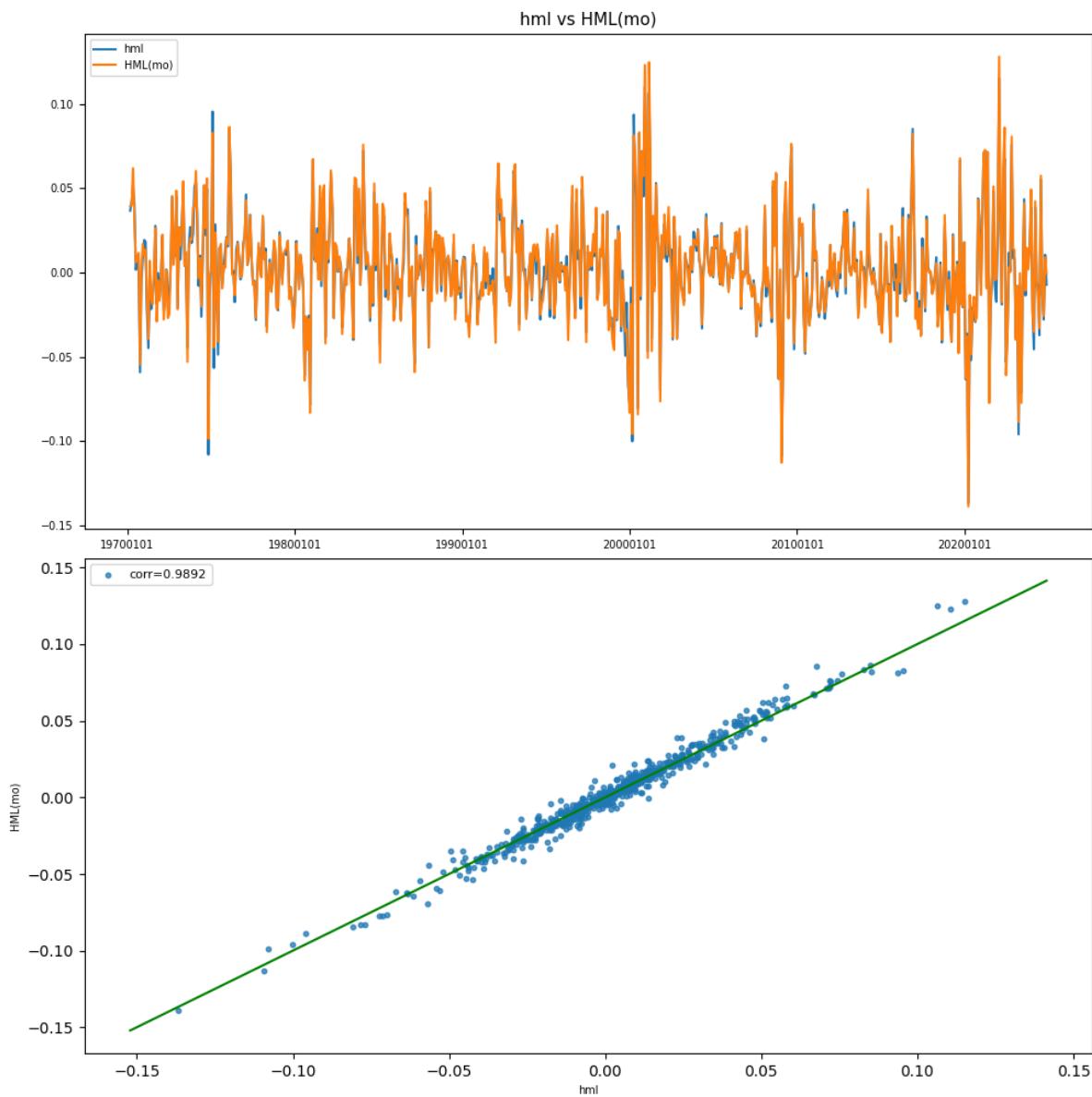
Helpers to show histograms and comparisons of portfolio returns

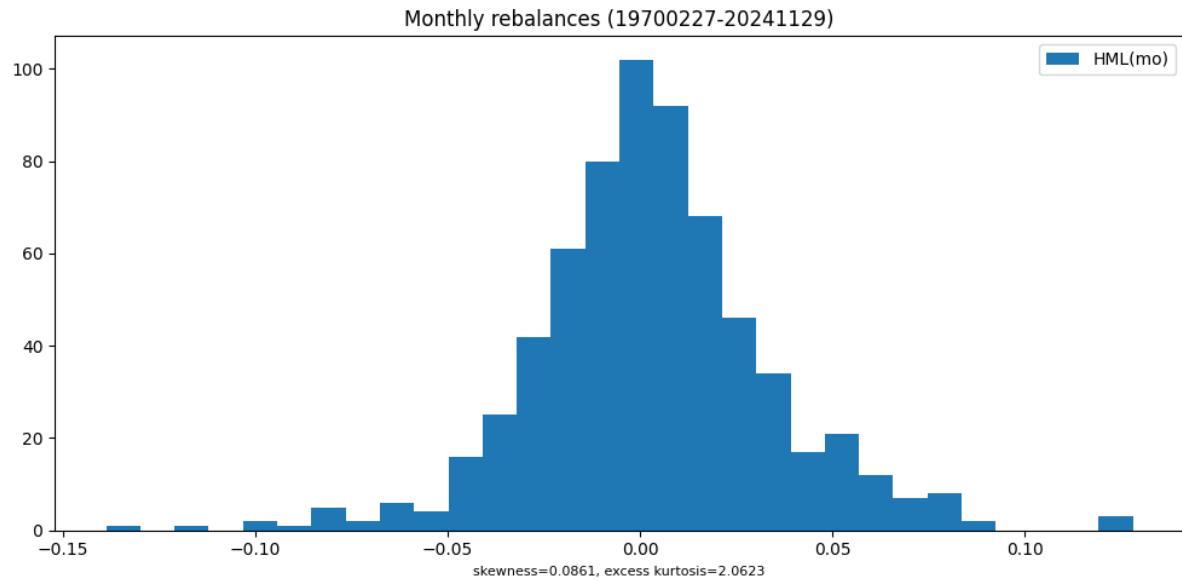
```
def plot_ff(y, label):
    """helper to scatter plot and compare portfolio returns"""
    y = y.rename(columns={'excess': label})
    corr = np.corrcoef(y, rowvar=False)[0, 1]
    fig, (ax1, ax2) = plt.subplots(2, 1, clear=True, figsize=(10, 10))
    plot_date(y, ax=ax1, title=f" vs ".join(y.columns), fontsize=7)
    plot_scatter(y.iloc[:, 0], y.iloc[:, 1], ax=ax2, abline=False, fontsize=7)
    plt.legend([f"corr={corr:.4f}"], fontsize=8)
    plt.tight_layout(pad=0.5)
    print(f"<R-squared of {label} vs {benchname}"
          f" ({y.index[0]} - {y.index[-1]}): {corr*corr:.4f}")
```

```
def plot_summary(y, label):
    """helper to plot histogram and statistics of portfolio returns"""
    y = y[label]
    kurt = kurtosis(y, bias=True, fisher=True) # excess kurtosis
    skewness = skew(y, bias=True)
    fig, ax = plt.subplots(1, 1, clear=True, figsize=(10, 5))
    ax.hist(y, bins=30)
    ax.set_title(f"Monthly rebalances ({y.index[0]}-{y.index[-1]})")
    ax.set_xlabel(f"skewness={skewness:.4f}, excess kurtosis={kurt:.4f}",
                  fontsize=8)
    plt.legend([label])
    plt.tight_layout()
```

```
# Plot histogram and comparison of HML returns
holdings = hml
result = backtest(monthly, holdings, label)
y = backtest.fit([benchname], rebalbeg, LAST_DATE)
plot_ff(y, label)
plot_summary(y, benchname)
```

<R-squared of hml vs HML (mo) (19700227 - 20241129): 0.9784



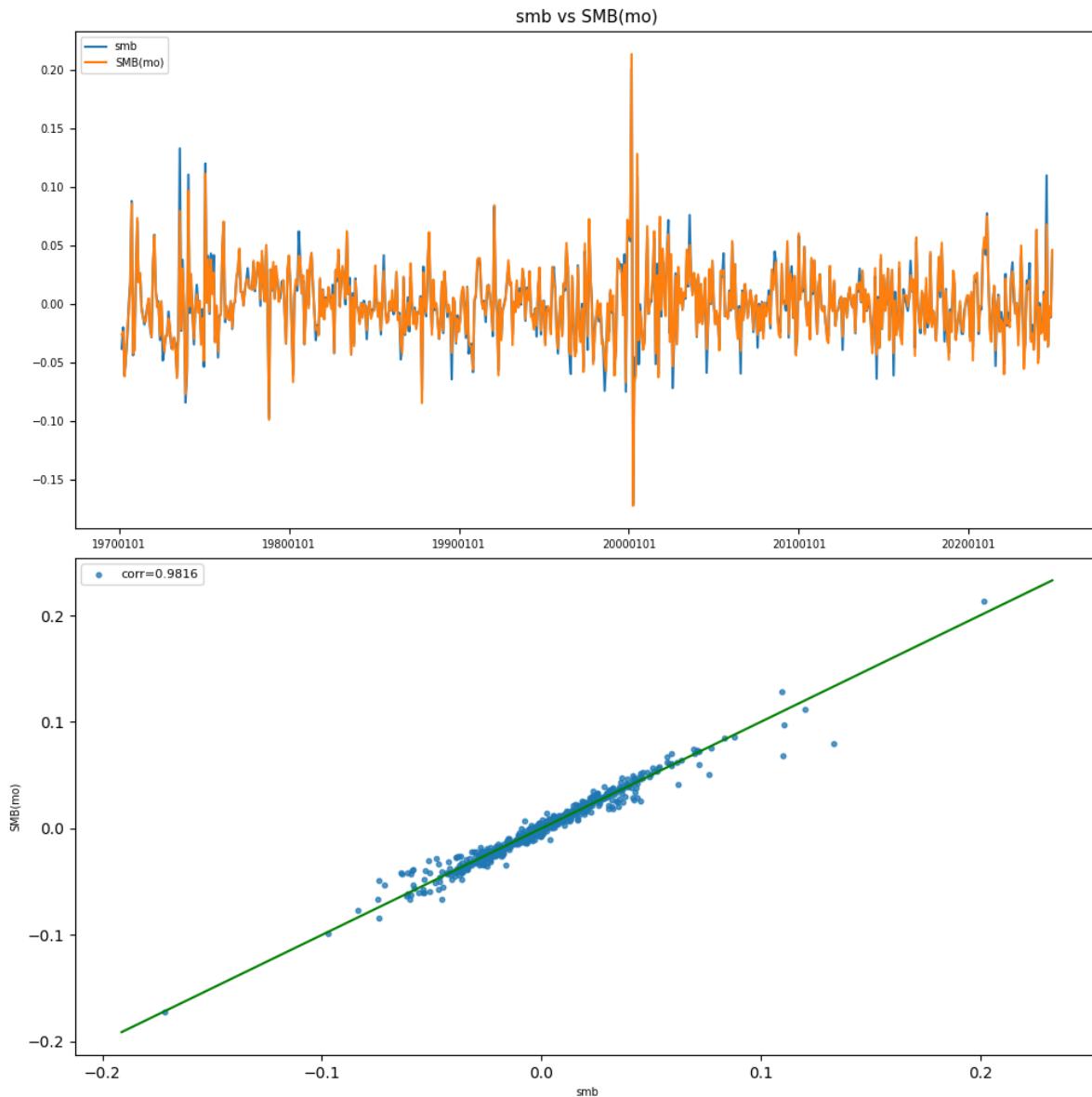


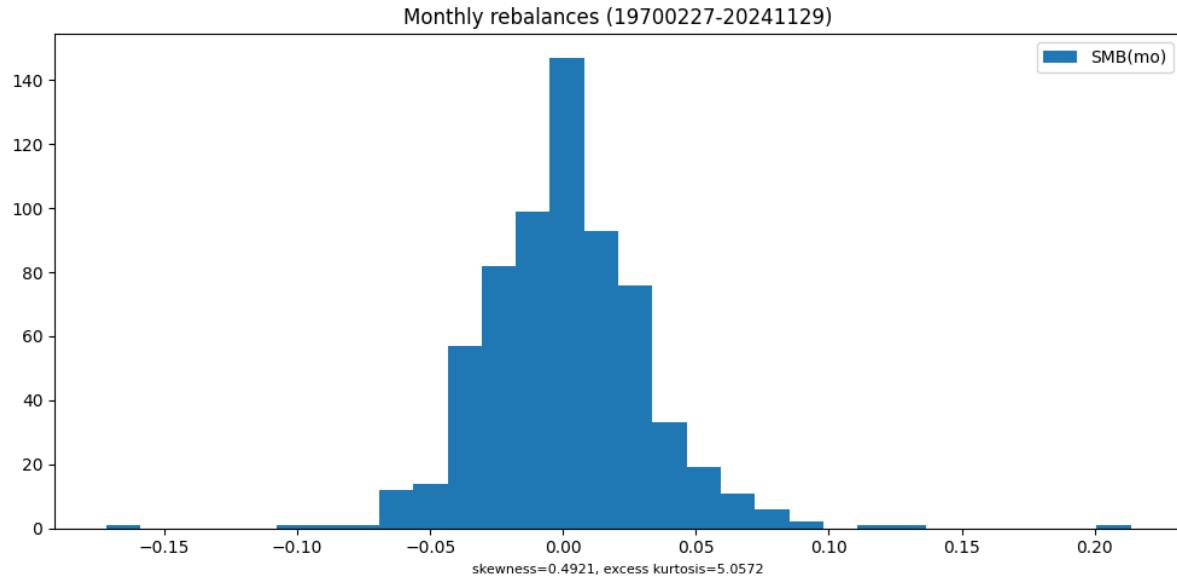
3.2.2 SMB

Compare Small-Minus-Big monthly returns and compare to Fama-French research factor

```
# Plot histogram and comparison of SMB returns
label, benchname = 'smb', 'SMB (mo)'
holdings = smb
result = backtest(monthly, holdings, label)
y = backtest.fit([benchname], rebalbeg, LAST_DATE)
plot_ff(y, label)
plot_summary(y, benchname)
```

<R-squared of smb vs SMB (mo) (19700227 - 20241129): 0.9636





3.3 Linear regression

Simple Linear Regression

The simple linear regression (SLR) model relates a continuous response (or dependent) variable y_i with one predictor (or explanatory or independent) variable x_i and an error term ϵ_i :

$$y_i = f(x_i) + \epsilon_i = a + bx_i + \epsilon_i$$

Coefficient estimates of the slope b and intercept a are chosen to minimize the residual sum of squares:

$$\hat{b} = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \hat{y})}{\sum_{i=1}^n (x_i - \bar{x})^2}$$

$$\hat{a} = \bar{y} - \hat{b}\bar{x}$$

Key concepts:

- Residuals are the difference between the observed response values and the response values predicted by the model, $e_i = y_i - \hat{y}_i$.
- Residual sum of squares (RSS) over all observations is $RSS = e_1^2 + e_2^2 + \dots + e_n^2$ or equivalently $RSS = \sum_{i=1}^n (y_i - \hat{y}_i)^2$.
- Mean square error (MSE) is an estimate of the variance of the residuals $s^2 = \hat{\sigma}^2 = \frac{1}{n-2} \sum_{i=1}^n (y_i - \hat{y}_i)^2$.
- Residual standard error (RSE) or residual standard deviation is the estimate of the (square root of the) variance of the residuals. Standard error tells us the average amount that the estimate differs from the actual value. The residual standard error is the estimate of the (square root of the) variance of the residuals $\hat{s} \equiv RSE = \sqrt{RSS/(n-2)}$.

Hypothesis testing and confidence intervals

The estimators of the coefficients follow a normal distribution in large samples. Therefore, tests of a hypothesis about a regression parameter are implemented using a t-test. The standard errors associated with linear regression coefficient and mean response estimates are:

- Slope: $se(\hat{b}) = \sqrt{\frac{s^2}{\sum_{i=1}^n (x_i - \bar{x})^2}}$
- Intercept: $se(\hat{a}) = \sqrt{s^2 \left[\frac{1}{n} + \frac{\bar{x}^2}{\sum_{i=1}^n (x_i - \bar{x})^2} \right]}$
- Mean response: $se(\hat{y}) = \sqrt{s^2 \left[\frac{1}{n} + \frac{(x - \bar{x})^2}{\sum_{i=1}^n (x_i - \bar{x})^2} \right]}$
- Confidence intervals can be computed from standard errors. A 95% confidence interval is defined as a range of values such that with 95% probability, the range will contain the true unknown value of the parameter. The confidence interval for coefficient estimates is $b_j \pm t_{n-(k+1), 1-\frac{\alpha}{2}} se(b_j)$, where k , the number of regressors, equals 1 for SLR.
- Prediction interval for a new response is $se(\hat{y}_{n+1}) = \sqrt{s^2 \left(1 + \frac{1}{n} + \frac{(x - \bar{x})^2}{\sum_{i=1}^n (x_i - \bar{x})^2} \right)}$

Multiple Linear Regression

With multiple regressors, the linear model $y = b_0 + b_1 x_1 + \dots + b_k x_k + \epsilon$ has coefficient estimates: $\hat{b} = (X^T X)^{-1} X^T y$

The coefficient b_j quantifies the association between the j 'th predictor and the response. It is the average effect on Y of a one unit increase in X_j , holding all other predictors fixed.

Additional concepts:

- Sum of squares total (SST) measures the total variance in the response Y, and can be thought of as the amount of variability inherent in the response before the regression is performed. $SST = \sum_{i=1}^n (y_i - \bar{y})^2$ measures the total variance in the response Y.
- Sum of squares regression (SSR) measures the total amount variance captured by the regression model: $SSR = \sum_{i=1}^n (\hat{y}_i - \bar{y})^2$
- Sum of squares error (SSE) measures the total variance of the response not explained by the regression model. In the linear regression context, we may interpret total deviation to equal the deviation not explained by the explanatory variables plus deviation explained by the explanatory variables: $(y_i - \bar{y}) = (y_i - \hat{y}_i) + (\hat{y}_i - \bar{y}_i)$. Squaring each side and summing over all observations yields for the total sum of squared deviations $SST = \sum_{i=1}^n (y_i - \hat{y}_i)^2 + \sum_{i=1}^n (\hat{y}_i - \bar{y}_i)^2 = SSE + SSR$, where the sum of the cross-product terms turns out to be zero.
- R^2 statistic or coefficient of determination measures the proportion of variability in Y that can be explained using X. It is also identical to the squared correlation between X and Y. An R^2 statistic that is close to 1 indicates that a large proportion of the variability in the response has been explained by the regression. A number near 0 indicates that the regression did not explain much of the variability in the response. The R^2 statistic provides a relative measure of the quality of a linear regression fit $R^2 = 1 - \frac{RSS}{SST}$, and always takes on a value between 0 and 1, and is independent of the scale. R^2 is identical to the squared correlation between X and Y .
- Adjusted R^2 : The usual R^2 always increases (since residual sum of squares RSS always decreases) as more variables are added. The intuition behind the adjusted R^2 is that once all of the correct variables have been included in the model, adding noise variables will lead to a decrease in the statistic. In theory, the model with the largest adjusted R^2 will have only correct variables and no noise variables.
- Partial correlation coefficients, which measure the correlation between y and the j 'th explanatory variable x_j controlling for other explanatory variables, can also be obtained by running only one regression: $r(y, x_j | x_1, x_2, \dots, x_{j-1}, x_{j+1}, \dots, x_k) = \frac{t(b_j)}{\sqrt{t(b_j)^2 + n - (k+1)}}$ where $t(b_j)$ is the t -ratio for b_j from a regression of y on x_1, \dots, x_k (including the variable x_j).

- t-statistic or t-ratio: $t(b_j) = \frac{b_j}{se(b_j)}$ can be interpreted to be the number of standard errors that b_j is away from zero. In a t-test, the null hypothesis ($H_0 : \beta_j = 0$) is rejected in favor of the alternative if the absolute value of the t-ratio $|t(b_j)|$ exceeds a t-value, denoted $t_{n-(k+1), 1-\frac{\alpha}{2}}$, equal to the $(1 - \frac{\alpha}{2})$ 'th percentile from the t-distribution using $df = n - (k + 1)$ degrees of freedom.

*Hypothesis tests

The t-test is not directly applicable when testing hypotheses that involve more than one parameter, because the parameter estimators can be correlated. Instead, a common alternative called the **F-test** compares the fit of the model (measured using the RSS) when the null hypothesis is true relative to the fit of the model without the restriction on the parameters assumed by the null. To test whether all regression slope coefficients are zero $H_0 : b_1 = \dots = b_p = 0$, versus the alternative $H_a : \text{at least one } b_j \text{ is non-zero}$, compute the statistic, which has a $F(p, n - p - 1)$ distribution:

$$F = \frac{(TSS - RSS)/p}{RSS/(n - p - 1)}$$

Partial F-test: Sometimes, we want to test that a particular subset of q of the coefficients are zero. In this case we fit a second model that uses all the variables except those last q , then compute residual sum of squares for that model and the appropriate F-statistic, which has a $F(p - q, n - p - 1)$ distribution:

$$F = \frac{(RSS_q - RSS)/(p - q)}{RSS/(n - p - 1)}$$

3.3.1 Value effect

The value effect refers to the observed tendency of value stocks (low price-to-book ratios) to outperform growth stocks (high price-to-book ratios) over time. This phenomenon supports value investing, a strategy linked to Benjamin Graham and David Dodd, which focuses on identifying undervalued stocks based on fundamental analysis.

```
# Linear regression on Mkt-Rf and intercept
x = ["HML(mo)", "Mkt-RF(mo)"]
formula = f'Q("{x[0]}") ~ ' + " + ".join(f'Q("{v}")' for v in x[1:])
data = bench.get_series(x, field='ret', beg=19620701, end=20991231)
lm = smf.ols(formula, data=data).fit(cov_type='HAC', cov_kwds={'maxlags': 6})
print(f"Period: {data.index[0]}-{data.index[-1]}")
print(lm.summary())
```

```
Period: 19620731-20241231
OLS Regression Results
=====
Dep. Variable: Q("HML(mo)") R-squared: 0.041
Model: OLS Adj. R-squared: 0.040
Method: Least Squares F-statistic: 8.701
Date: Sun, 02 Mar 2025 Prob (F-statistic): 0.00328
Time: 14:45:35 Log-Likelihood: 1587.2
No. Observations: 750 AIC: -3170.
Df Residuals: 748 BIC: -3161.
Df Model: 1
Covariance Type: HAC
=====
      coef    std err          z      P>|z|      [ 0.025      0.975
-----
Intercept    0.0037    0.001     2.615      0.009      0.001      0.006
Q("Mkt-RF (mo)") -0.1344    0.046    -2.950      0.003     -0.224     -0.045

```

(continues on next page)

(continued from previous page)

```
=====
Omnibus:                  54.218   Durbin-Watson:           1.670
Prob(Omnibus):            0.000   Jarque-Bera (JB):      243.740
Skew:                     -0.030   Prob(JB):                  1.18e-53
Kurtosis:                 5.792   Cond. No.                 22.3
=====
```

Notes:

[1] Standard Errors are heteroscedasticity and autocorrelation robust (HAC) using $\sqrt{6}$ lags and without small sample correction

3.3.2 Small firm effect

The small firm effect, identified by Rolf Banz in 1981, describes the tendency of small-cap stocks to generate higher risk-adjusted returns than large-cap stocks. This suggests that smaller companies may offer higher expected returns as compensation for their increased risk and lower liquidity.

```
# Linear regression on Mkt-Rf, HML and intercept
x = ["SMB(mo)", "HML(mo)", "Mkt-RF(mo)"]
formula = f'Q("{x[0]}") ~ ' + " + ".join(f'Q("{v}")' for v in x[1:])
data = bench.get_series(x, field='ret', beg=19620701, end=20991231)
lm = smf.ols(formula, data=data).fit(cov_type='HAC', cov_kwds={'maxlags': 6})
print(f"Period: {data.index[0]}-{data.index[-1]}")
print(lm.summary())
```

```
Period: 19620731-20241231
OLS Regression Results
=====
Dep. Variable: Q("SMB(mo)")   R-squared: 0.097
Model: OLS   Adj. R-squared: 0.095
Method: Least Squares   F-statistic: 28.64
Date: Sun, 02 Mar 2025   Prob (F-statistic): 1.03e-12
Time: 14:45:35   Log-Likelihood: 1593.2
No. Observations: 750   AIC: -3180.
Df Residuals: 747   BIC: -3167.
Df Model: 2
Covariance Type: HAC
=====

      coef    std err          z      P>|z|      [0.025      0.975]
-----
Intercept    0.0007    0.001    0.592     0.554    -0.002     0.003
Q("HML(mo)") -0.0937    0.088   -1.066     0.286    -0.266     0.079
Q("Mkt-RF(mo)") 0.1901    0.028    6.672     0.000     0.134     0.246
=====

Omnibus: 105.898   Durbin-Watson: 2.056
Prob(Omnibus): 0.000   Jarque-Bera (JB): 722.449
Skew: 0.406   Prob(JB): 1.33e-157
Kurtosis: 7.739   Cond. No. 34.8
=====
```

Notes:

[1] Standard Errors are heteroscedasticity and autocorrelation robust (HAC) using $\sqrt{6}$ lags and without small sample correction

3.4 Structural break test

The **Chow test** is used to detect structural breaks in time-series data. This involves estimating separate regression models before and after a specified breakpoint (e.g., the publication of the HML factor in 1993). The test statistic is:

$$\text{Chow} = \frac{(RSS - (RSS_1 + RSS_2))/K}{(RSS_1 + RSS_2)/(N - 2K)}$$

where the test follows an $F(K, N - 2K)$ distribution. If the test statistic exceeds a critical value, we reject the null hypothesis that the regression coefficients remain constant across both time periods.

```
# Run restricted and unregression models
x = ["HML(mo)", "Mkt-RF(mo)"]
formula = f'Q("{x[0]}") ~ ' + " + ".join(f'Q("{v}")' for v in x[1:])
#formula = f'Q("HML(mo)") ~ .'
bp = 19931231 # breakpoint date
lm1 = smf.ols(formula, data=data[data.index<=bp]).fit()
print(f'\nSub Model 1 ({data.index[0]}-{bp}):')
print(lm1.summary())
lm2 = smf.ols(formula, data=data[data.index>bp]).fit()
print(f'\nSub Model 2 ({bp}-{data.index[-1]}):')
print(lm2.summary())
lm0 = smf.ols(formula, data=data).fit()
print('\nRestricted Model (coefficient is equal):')
print(lm0.summary())
```

```
Sub Model 1 (19620731-19931231):
    OLS Regression Results
=====
Dep. Variable: Q("HML(mo)") R-squared: 0.125
Model: OLS Adj. R-squared: 0.122
Method: Least Squares F-statistic: 53.47
Date: Sun, 02 Mar 2025 Prob (F-statistic): 1.59e-12
Time: 16:34:00 Log-Likelihood: 874.79
No. Observations: 378 AIC: -1746.
Df Residuals: 376 BIC: -1738.
Df Model: 1
Covariance Type: nonrobust
=====
            coef    std err        t    P>|t|    [0.025    0.975]
-----
Intercept  0.0056    0.001     4.524    0.000     0.003    0.008
Q("Mkt-RF(mo)") -0.2021    0.028    -7.312    0.000    -0.256   -0.148
=====
Omnibus: 29.355 Durbin-Watson: 1.600
Prob(Omnibus): 0.000 Jarque-Bera (JB): 55.192
Skew: 0.462 Prob(JB): 1.04e-12
Kurtosis: 4.628 Cond. No. 22.4
=====
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
    specified.

Sub Model 2 (19931231-20241231):
    OLS Regression Results
```

(continues on next page)

(continued from previous page)

```
=====
Dep. Variable: Q("HML(mo)") R-squared: 0.007
Model: OLS Adj. R-squared: 0.005
Method: Least Squares F-statistic: 2.752
Date: Sun, 02 Mar 2025 Prob (F-statistic): 0.0980
Time: 16:34:00 Log-Likelihood: 738.00
No. Observations: 372 AIC: -1472.
Df Residuals: 370 BIC: -1464.
Df Model: 1
Covariance Type: nonrobust
=====
            coef    std err      t      P>|t|      [0.025      0.975]
-----
Intercept  0.0015    0.002    0.866    0.387    -0.002    0.005
Q("Mkt-RF (mo)") -0.0640    0.039   -1.659    0.098    -0.140    0.012
=====
Omnibus: 25.036 Durbin-Watson: 1.720
Prob(Omnibus): 0.000 Jarque-Bera (JB): 87.339
Skew: 0.058 Prob(JB): 1.08e-19
Kurtosis: 5.371 Cond. No. 22.3
=====
```

Notes:

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

Restricted Model (coefficient is equal):

OLS Regression Results

```
=====
Dep. Variable: Q("HML(mo)") R-squared: 0.041
Model: OLS Adj. R-squared: 0.040
Method: Least Squares F-statistic: 31.83
Date: Sun, 02 Mar 2025 Prob (F-statistic): 2.39e-08
Time: 16:34:00 Log-Likelihood: 1587.2
No. Observations: 750 AIC: -3170.
Df Residuals: 748 BIC: -3161.
Df Model: 1
Covariance Type: nonrobust
=====
            coef    std err      t      P>|t|      [0.025      0.975]
-----
Intercept  0.0037    0.001    3.421    0.001      0.002    0.006
Q("Mkt-RF (mo)") -0.1344    0.024   -5.642    0.000    -0.181   -0.088
=====
Omnibus: 54.218 Durbin-Watson: 1.670
Prob(Omnibus): 0.000 Jarque-Bera (JB): 243.740
Skew: -0.030 Prob(JB): 1.18e-53
Kurtosis: 5.792 Cond. No. 22.3
=====
```

Notes:

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

Test statistic with K parameters and N observations follows a $F(K, N - 2K)$ -distribution

```
# Compute test statistic
K = len(lm0.params)
N = len(data)
RSS = lm0.resid.dot(lm0.resid)
RSS1 = lm1.resid.dot(lm1.resid)
RSS2 = lm2.resid.dot(lm2.resid)
chow = ((RSS - (RSS1 + RSS2)) / K) / ((RSS1 + RSS2) / (N - 2*K))
chow, N, K
```

```
(5.425564122014983, 750, 2)
```

p-value of Chow test statistic

```
1 - scipy.stats.f.cdf(chow, dfn=K, dfd=N - 2*K)
```

```
0.0045780451491863605
```

5% critical value to reject null

```
scipy.stats.f.ppf(q=1 - 0.05, dfn=K, dfd=N - 2*K)
```

```
3.0077945872688696
```

References:

Eugene F. Fama and Kenneth R. French (1992), “The Cross-Section of Expected Stock Returns”, Journal of Finance, Volume 47, Issue 2, June 1992, pages 427-465

Eugene Fama and Kenneth French (2023), “Production of U.S. Rm-Rf, SMB, and HML in the Fama-French Data Library”, Chicago Booth Paper No. 23-22

FRM Exam I Book Quantitative Analysis Chapter 7-8

FAMA-MACBETH CROSS-SECTIONAL REGRESSIONS

If you don't risk anything, you risk even more – Erica Jong

The Fama-MacBeth (1973) cross-sectional regression methodology is a fundamental tool in empirical asset pricing, used to estimate risk factor loadings and associated risk premia while accounting for cross-sectional correlation in errors. By performing two-stage regressions, the method first estimates factor loadings for individual assets and then determines the associated risk premia over time. This approach has broad applications in testing asset pricing models, including the Capital Asset Pricing Model (CAPM) and multi-factor models. The following sections also analyze efficient frontier construction, Black-Litterman implied alphas and portfolio optimization, risk factor modeling, and non-linear regressions.

```
import numpy as np
from numpy import linalg as la
import pandas as pd
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
import statsmodels.formula.api as smf
from sklearn.kernel_ridge import KernelRidge
import random
from tqdm import tqdm
import cvxpy as cp
from finds.database import SQL, RedisDB
from finds.structured import (BusDay, Signals, Benchmarks, CRSP,
                               CRSPBuffer, SignalsFrame)
from finds.backtesting import RiskPremium
from finds.recipes import winsorize, least_squares
from finds.readers import FFReader
from secret import credentials, paths, CRSP_DATE
VERBOSE = 0
#%matplotlib qt
```

```
# open connections
sql = SQL(**credentials['sql'], verbose=VERBOSE)
user = SQL(**credentials['user'], verbose=VERBOSE)
rdb = RedisDB(**credentials['redis'])
bd = BusDay(sql, verbose=VERBOSE)
crsp = CRSP(sql, bd, rdb=rdb, verbose=VERBOSE)
signals = Signals(user, verbose=VERBOSE)
bench = Benchmarks(sql, bd, verbose=VERBOSE)
imgdir = paths['images']
LAST_DATE = bd.endmo(CRSP_DATE, -1)
```

```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

4.1 Mean variance optimization

Markowitz demonstrated that, given two investments with the same expected return (measured as the mean of returns), a risk-averse investor will prefer the one with lower risk (measured by variance). His theory relies on several assumptions, including the absence of market frictions (such as taxes or transaction costs) and normally distributed returns.

The assumption of normally distributed returns implies that rational investors should evaluate potential portfolio allocations based solely on the means and variances of their return distributions. Investors generally seek higher mean returns while minimizing variance. Diversification plays a crucial role in reducing portfolio risk by incorporating assets whose price movements are not perfectly correlated.

A key challenge in implementing this framework is estimating the necessary parameters—mean returns, variances, and asset correlations—using historical data. The choice of historical period or forecast assumptions can significantly impact the resulting allocation. To address this uncertainty, techniques such as robust portfolio optimization and the Black-Litterman model have been developed.

```
# Retrieve test asset returns and risk-free rate
symbol = '6_Portfolios_2x3'
ff = FFReader(symbol)
rf = FFReader('F-F_Research_Data_Factors')[0]['RF'] / 100 # risk-free rates
mktcaps = ff[4] * ff[5] # number of firms x average market cap
labels = [s.replace('ME1', 'BIG').replace('ME2', 'SMALL') for s in mktcaps.columns]
n = len(labels)
```

```
r = (ff[0]/100).sub(rf.fillna(0), axis=0) # excess, of the risk-free, returns
sigma = np.cov(r, rowvar=False)
mu = np.mean(r, axis=0).values
assets = DataFrame(data={'mean': mu, 'volatility': np.sqrt(np.diag(sigma))}, index=labels)
```

```
mkt = {'weights': (mktcaps.iloc[-1]/mktcaps.iloc[-1].sum()).values} # latest caps
mkt['mean'] = mkt['weights'].dot(mu)
mkt['variance'] = mkt['weights'].dot(sigma).dot(mkt['weights'])
pd.concat([assets.T, Series({'mean': mkt['mean'], 'volatility': np.sqrt(mkt['variance'])}), name='Mkt')], axis=1)
```

	SMALL	LoBM	BIG	BM2	SMALL	HiBM	BIG	LoBM	SMALL	BM2	BIG	HiBM	\
mean	0.007149	0.009629	0.011528	0.006860	0.006886	0.009252							
volatility	0.074748	0.069687	0.081045	0.052923	0.056215	0.071231							
	Mkt												
mean	0.007152												
volatility	0.053822												

4.1.1 Global minimum variance portfolio

The Global Minimum Variance (GMV) portfolio is the allocation that achieves the lowest possible risk based on estimated asset variances and correlations while disregarding expected returns. This optimization problem is convex (quadratic) and subject to the constraint that portfolio weights sum to one.

Mathematically, the GMV portfolio is obtained by solving:

$\min_w w^T \Sigma w$, subject to $w^T 1 = 1$ where Σ represents the covariance matrix of asset returns. This can be solved numerically with the `cvxpy` Python package for convex optimization.

```
W = cp.Variable(n)      # variable to optimize over - portfolio weights
Var = cp.quad_form(W, sigma)    # objective to minimize portfolio volatility
Ret = mu.T @ W           # objective to maximize portfolio return
```

```
obj = cp.Problem(cp.Minimize(Var), [cp.sum(W) == 1])
obj.solve()
gmv = dict(weights=W.value, variance=Var.value, mean=Ret.value,
            coords=(np.sqrt(Var.value), Ret.value))
```

The GMV portfolio weights can also be derived using a closed form solution by differentiating the (convex) objective function and setting the first-order conditions to zero: $\text{GMV} = \frac{\Sigma^{-1} 1}{1^T \Sigma^{-1} 1}$

```
def gmv_portfolio(sigma, mu=None):
    """Returns position weights of global minimum variance portfolio"""
    ones = np.ones((sigma.shape[0], 1))
    w = la.inv(sigma).dot(ones) / ones.T.dot(la.inv(sigma)).dot(ones)
    return {'weights': w, 'volatility': np.sqrt(w.T.dot(sigma).dot(w)),
            'mean': None if mu is None else w.T.dot(mu)}
```

```
w = gmv_portfolio(mu=mu, sigma=sigma) ['weights']
pd.concat([Series(gmv['weights']).rename('numerical'),
            Series(w.flatten()).rename('formula')], axis=1) \
    .set_index(assets.index).T
```

	SMALL LoBM	BIG BM2	SMALL HiBM	BIG LoBM	SMALL BM2	BIG HiBM
numerical	-0.485571	0.668181	-0.330072	0.789931	0.83042	-0.47289
formula	-0.485571	0.668181	-0.330072	0.789931	0.83042	-0.47289

4.1.2 Efficient frontier

Each point on the efficient frontier represents a portfolio that offers the highest expected return for a given level of risk, measured by the standard deviation of returns. A line drawn from the risk-free rate becomes tangent to the efficient frontier at the **tangency portfolio**, defining the **Capital Market Line (CML)**:

$$E(R_p) = r_f + \frac{E[R_M] - r_f}{\sigma_M} \sigma_p$$

Portfolios along this line dominate all other portfolios on the efficient frontier. This leads to the **Two-Fund Separation Theorem**, which states that all investors should allocate capital between the risk-free asset and the tangency portfolio.

```

var_ticks = np.linspace(gmv['variance'], 3*np.max(np.diag(sigma)), 200)
best_slope, tangency = 0, tuple()      # to find the tangency portfolio
efficient = []
for var in var_ticks:
    obj = cp.Problem(cp.Maximize(Ret), [cp.sum(W) == 1, Var <= var])
    obj.solve(verbose=False)

    # tangency portfolio has best slope
    risk = np.sqrt(var)
    slope = Ret.value / risk
    if slope > best_slope:
        best_slope = slope
        tangency = {'coords': (risk, Ret.value), 'weights': W.value}
    efficient.append(dict(mean=Ret.value, volatility=risk))

```

```

frontier = []      # inefficient frontier
for var in var_ticks:
    obj = cp.Problem(cp.Minimize(Ret), [cp.sum(W) == 1, Var <= var])
    obj.solve(verbose=False)
    frontier.append(dict(mean=Ret.value, volatility=np.sqrt(var)))

```

The efficient and tangency portfolios can be derived as:

- Efficient portfolio (target return μ_0) = $\Sigma^{-1}M(M^T\Sigma^{-1}M)^{-1}[\mu_0 \ 1]^T$, where $M = [\mu \ 1]$
- Tangency portfolio = $\frac{\Sigma^{-1}\mu}{1^T\Sigma^{-1}\mu}$

Any portfolio on the efficient frontier can be expressed as a linear combination of two other efficient portfolios.

```

def efficient_portfolio(mu, sigma, target):
    """Returns weights of minimum variance portfolio that exceeds target return"""
    mu = mu.flatten()
    n = len(mu)
    ones = np.ones((n, 1))
    M = np.hstack([mu.reshape(-1, 1), ones])
    B = M.T.dot(la.inv(sigma)).dot(M)
    w = la.inv(sigma).dot(M).dot(la.inv(B)).dot(np.array([[target], [1]]))
    return {'weights': w, 'volatility': np.sqrt(float(w.T.dot(sigma).dot(w))), 'mean': float(w.T.dot(mu))}

```

```

p = random.choice(efficient)
e = efficient_portfolio(mu, sigma, p['mean'])
df = DataFrame({'random efficient portfolio': p,
                'by formula': dict(mean=e['mean'], volatility=e['volatility'])})

```

```

/tmp/ipykernel_1785309/3388186085.py:9: DeprecationWarning: Conversion of an array
  ↵with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you
  ↵extract a single element from your array before performing this operation.
  ↵(Deprecated NumPy 1.25.)
  return {'weights': w, 'volatility': np.sqrt(float(w.T.dot(sigma).dot(w)))},
/tmp/ipykernel_1785309/3388186085.py:10: DeprecationWarning: Conversion of an
  ↵array with ndim > 0 to a scalar is deprecated, and will error in future. Ensure
  ↵you extract a single element from your array before performing this operation.
  ↵(Deprecated NumPy 1.25.)
  'mean': float(w.T.dot(mu)) }

```

```

def tangency_portfolio(mu, sigma):
    """Returns weights of tangency portfolio with largest slope (sharpe ratio)"""
    mu = mu.flatten()
    n = len(mu)
    ones = np.ones((n, 1))
    w = la.inv(sigma).dot(mu)/ones.T.dot(la.inv(sigma).dot(mu))
    return {'weights': w, 'mean': float(w.T.dot(mu)),
            'volatility': np.sqrt(float(w.T.dot(sigma).dot(w)))}

```

```
s = tangency_portfolio(mu, sigma)
```

```

# show numerical and formulas are same solution
DataFrame({'tangency portfolio': list(tangency['coords']),
            'tangency formula': [s['volatility'], s['mean']]},
           index=['volatility', 'mean']).join(df)

```

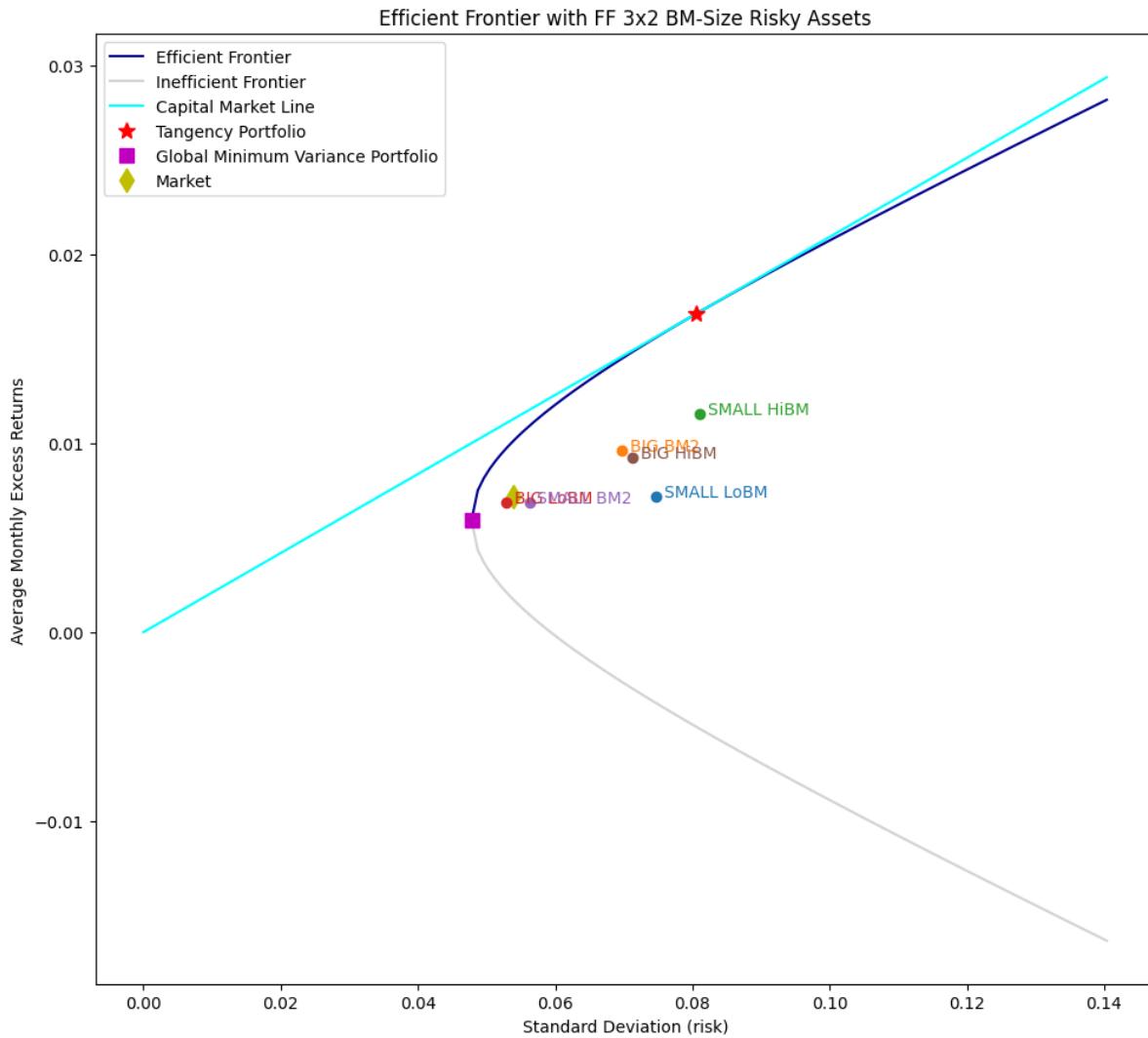
	tangency portfolio	tangency formula	random efficient portfolio	\
volatility	0.080541	0.080748	0.08782	
mean	0.016848	0.016891	0.01834	
	by formula			
volatility	0.08782			
mean	0.01834			

Plot efficient frontier and portfolios

```

fig, ax = plt.subplots(figsize=(10, 9))
DataFrame(efficient).set_index('volatility').plot(ax=ax, color='darkblue')
DataFrame(frontier).set_index('volatility').plot(ax=ax, color='lightgrey')
ax.plot([0, np.sqrt(max(var_ticks))], [0, np.sqrt(max(var_ticks))*best_slope],
        color='cyan') # capital market line
ax.plot(*tangency['coords'], "r*", ms=10) # tangency portfolio
ax.plot(np.sqrt(gmv['variance']), gmv['mean'], "ms", ms=8) # GMV portfolio
ax.plot(np.sqrt(mkt['variance']), mkt['mean'], "yd", ms=10) # market portfolio
plt.legend(['Efficient Frontier', 'Inefficient Frontier', 'Capital Market Line',
           'Tangency Portfolio', 'Global Minimum Variance Portfolio', 'Market'])
for c, r in enumerate(assets.itertuples()): # risky assets
    ax.plot(r.volatility, r.mean, marker='o', color=f"C{c}")
    ax.annotate(text=r.Index, xy=(r.volatility, r.mean),
                xytext=(0.5, 0), textcoords="offset fontsize", color=f"C{c}")
ax.set_xlabel('Standard Deviation (risk)')
ax.set_ylabel('Average Monthly Excess Returns')
ax.set_title('Efficient Frontier with FF 3x2 BM-Size Risky Assets')
plt.tight_layout()

```



4.1.3 CAPM

Sharpe, Lintner, and Mossin developed the CAPM, an equilibrium model that describes the relationship between risk and expected return for risky assets. The model assumes market efficiency and that investors optimize portfolios based on mean-variance principles.

The CAPM posits that market equilibrium is reached when all investors hold combinations of the risk-free asset and the market portfolio. An asset's expected return is determined by its contribution to the market portfolio's total risk, specifically its **systematic risk**, which cannot be diversified away. This risk is measured by beta:

$$\beta_i = \frac{\text{cov}(R_i, R_M)}{\text{var}(R_M)}$$

The **Security Market Line (SML)** represents the relationship between expected returns and beta:

$$E(R_i) = r_f + \beta_i(E[R_M] - r_f)$$

4.2 Implied alphas

If a known portfolio allocation W is an optimal solution to a mean-variance objective, the implied mean return inputs can be inferred given the covariance matrix. These **implied alphas**, proportional to $w^T \Sigma$, when used as expected returns in the optimization process, yield the same portfolio W .

```
# market cap-weighted portfolio implied expected returns
capm = mkt['weights'].dot(sigma) * 2
```

```
# HML implied alphas
hml = Series(0.0, index=assets.index)
hml['BIG HiBM'] = 0.5
hml['SMALL HiBM'] = 0.5
hml['BIG LoBM'] = -0.5
hml['SMALL LoBM'] = -0.5
#hml = {'weights': hml.values}
#hml['variance'] = hml['weights'].dot(sigma).dot(hml['weights'])
#hml['coords'] = (np.sqrt(hml['variance']), hml['weights'].dot(mu))
alphas = hml.dot(sigma) * 2
```

```
pd.concat([Series(hml.values).rename('HML weights'),
           Series(alphas).rename('HML implied-alpha'),
           Series(mkt['weights']).rename('Market weights'),
           Series(capm).rename('CAPM equilibrium returns'),
           Series(mu).rename('historical mu'),
           Series(Series(alphas)/Series(capm)).rename('implied/capm')],
           axis=1, ignore_index=False) \
.set_index(assets.index).T
```

	SMALL LoBM	BIG BM2	SMALL HiBM	BIG LoBM	\
HML weights	-0.500000	0.000000	0.500000	-0.500000	
HML implied-alpha	0.000742	0.001827	0.003132	0.000277	
Market weights	0.010631	0.017235	0.011753	0.693932	
CAPM equilibrium returns	0.007044	0.006705	0.007413	0.005624	
historical mu	0.007149	0.009629	0.011528	0.006860	
implied/capm	0.105320	0.272514	0.422451	0.049240	
	SMALL BM2	BIG HiBM			
HML weights	0.000000	0.500000			
HML implied-alpha	0.001634	0.002967			
Market weights	0.190550	0.075900			
CAPM equilibrium returns	0.005733	0.006865			
historical mu	0.006886	0.009252			
implied/capm	0.285082	0.432198			

```
# Correlations of alphas
DataFrame({'historical mu': np.corrcoef(alphas, mu)[0][-1],
           'capm equilibrium': np.corrcoef(alphas, capm)[0][-1]},
           index=['Correlation with implied alphas'])
```

	historical mu	capm equilibrium
Correlation with implied alphas	0.836893	0.610935

```
# Mean-variance optimization with HML-implied alphas
MeanVariance = alphas @ W - Var
obj = cp.Problem(cp.Maximize(MeanVariance))
obj.solve(verbose=False)
DataFrame.from_records([hml.values, W.value], columns=labels,
                      index=['HML weights', 'mean-variance weights']).round(6)
```

	SMALL LoBM	BIG BM2	SMALL HiBM	BIG LoBM	SMALL BM2	\
HML weights	-0.5	0.0	0.5	-0.5	0.0	
mean-variance weights	-0.5	0.0	0.5	-0.5	0.0	
	BIG HiBM					
HML weights		0.5				
mean-variance weights		0.5				

```
# Mean-variance optimization with CAPM-implied expected returns
MeanVariance = capm @ W - Var
obj = cp.Problem(cp.Maximize(MeanVariance))
obj.solve(verbose=False)
p = tangency_portfolio(mu=capm, sigma=sigma)
DataFrame.from_records([W.value, mkt['weights'], p['weights']], columns=labels,
                      index=['Market weights', 'mean-variance weights', 'formula'])
```

	SMALL LoBM	BIG BM2	SMALL HiBM	BIG LoBM	SMALL BM2	\
Market weights	0.010631	0.017235	0.011753	0.693932	0.19055	
mean-variance weights	0.010631	0.017235	0.011753	0.693932	0.19055	
formula	0.010631	0.017235	0.011753	0.693932	0.19055	
	BIG HiBM					
Market weights		0.0759				
mean-variance weights		0.0759				
formula		0.0759				

4.2.1 Black-Litterman Model

Mean-variance portfolios are highly sensitive to input estimates, particularly expected returns. Errors in estimating expected returns have a far greater impact than errors in estimating variances and covariances. Black and Litterman (1992) proposed *shrinking* investor expectations toward equilibrium market returns to reduce sensitivity to estimation errors.

```
active = tangency['weights'] - mkt['weights']
DataFrame.from_records([tangency['weights'], mkt['weights'], active], columns=labels,
                      index=['Tangency Portfolio Weights', 'Market Weights',
                             'Active Weights']).round(6)
```

	SMALL LoBM	BIG BM2	SMALL HiBM	BIG LoBM	\
Tangency Portfolio Weights	-2.807841	2.631802	1.023199	2.132014	
Market Weights	0.010631	0.017235	0.011753	0.693932	
Active Weights	-2.818472	2.614567	1.011446	1.438083	
	SMALL BM2	BIG HiBM			
Tangency Portfolio Weights	-1.466995	-0.51218			

(continues on next page)

(continued from previous page)

Market Weights	0.190550	0.07590
Active Weights	-1.657544	-0.58808

The Black-Litterman expected return estimates are computed as:

$$E[R] = [(\tau\Sigma)^{-1} + P^T\Omega^{-1}P]^{-1}[(\tau\Sigma)^{-1}\Pi + P^T\Omega^{-1}Q]$$

where:

- τ is a confidence scalar for investor views versus equilibrium returns.
- In Bayesian terms, τ represents uncertainty in equilibrium return estimation.

```
tau = 0.05 # He and Litterman (1992) for a moderate amount of active risk
k = 1
Pi = capm.reshape((n, 1)) # equilibrium views: CAPM implied excess returns
P = (tangency['weights']).reshape((k, n)) # view portfolio weights
Q = (tangency['weights']).dot(mu).reshape((k, k)) # portfolio view
Omega = np.diag(np.array(P.dot(sigma).dot(P.T)).reshape((k,k))) # uncertainty
```

```
def black_litterman(tau, Pi, Sigma, P, Q):
    """Returns black-litterman alphas"""
    def inv(x):
        """helper wraps over la.inv to handle scalar/1d inputs"""
        try:
            return la.inv(x)
        except:
            return np.array(1/x).reshape((1,1))
    return inv(inv(tau*Sigma)+P.T.dot(inv(Omega)).dot(P)) \
        .dot(inv(tau*Sigma).dot(Pi) + P.T.dot(inv(Omega)).dot(Q))
```

```
bl = {'alphas': black_litterman(tau=tau, Pi=Pi, Sigma=sigma, P=P, Q=Q)}
bl |= tangency_portfolio(mu=bl['alphas'], sigma=sigma)
bl['mean'] = bl['weights'].dot(mu) # express mean based on original mu
bl['tilt'] = bl['weights'] - mkt['weights']
print('Active Risk:', np.sqrt(bl['tilt'].T.dot(sigma).dot(bl['tilt'])))
DataFrame.from_dict({'Black-Litterman weights': bl['weights'],
                     'Market weights': mkt['weights'], 'Active weights': bl['tilt']},
                     columns=labels, orient='index').round(6)
```

Active Risk: 0.0024824588797282025

	SMALL LoBM	BIG BM2	SMALL HiBM	BIG LoBM	\
Black-Litterman weights	-0.101900	0.121624	0.052136	0.751349	
Market weights	0.010631	0.017235	0.011753	0.693932	
Active weights	-0.112531	0.104390	0.040383	0.057417	
	SMALL BM2	BIG HiBM			
Black-Litterman weights	0.124370	0.05242			
Market weights	0.190550	0.07590			
Active weights	-0.066179	-0.02348			

```
# BL tilts the optimal weights towards the active positions in the view portfolio
bl['tilt'] / active
```

```
array([0.03992611, 0.03992611, 0.03992611, 0.03992611, 0.03992611,
       0.03992611])
```

With numerical solvers, constraints like no short-selling can be incorporated, and additional constraints can be iteratively added to achieve more reasonable portfolio allocations.

```
Alpha = W @ mu
```

```
# Minimize variance to same expected return
obj = cp.Problem(cp.Minimize(Var), [cp.sum(W) == 1, W >= 0, Alpha >= bl['mean']])
obj.solve()
tilt = W.value - mkt['weights']
print('Minimize variance to achieve target return, with no short sales:')
print('Active Risk:', np.sqrt(tilt.T.dot(sigma).dot(tilt)))
DataFrame.from_dict({'Constrained weights': W.value,
                     'Market weights': mkt['weights'], 'Active weights': tilt},
                     columns=labels, orient='index').round(6)
```

```
Minimize variance to achieve target return, with no short sales:
Active Risk: 0.004263652598782702
```

	SMALL LoBM	BIG BM2	SMALL HiBM	BIG LoBM	SMALL BM2	\
Constrained weights	-0.000000	-0.000000	0.145177	0.790527	0.064296	
Market weights	0.010631	0.017235	0.011753	0.693932	0.190550	
Active weights	-0.010631	-0.017235	0.133425	0.096595	-0.126254	
	BIG HiBM					
Constrained weights	0.0000					
Market weights	0.0759					
Active weights	-0.0759					

```
obj = cp.Problem(cp.Maximize(Alpha),
                 [cp.sum(W) == 1, W >= 0, Var <= bl['volatility']**2])
obj.solve()
tilt = W.value - mkt['weights']
print('Maximize return within target variance, with no short sales:')
print('Active Risk (annualized):', np.sqrt(tilt.T.dot(sigma).dot(tilt) * 12))
DataFrame.from_dict({'Constrained weights': W.value,
                     'Market weights': mkt['weights'], 'Active weights': tilt},
                     columns=labels, orient='index').round(6)
```

```
Maximize return within target variance, with no short sales:
Active Risk (annualized): 0.009003930083077559
```

	SMALL LoBM	BIG BM2	SMALL HiBM	BIG LoBM	SMALL BM2	\
Constrained weights	0.000000	0.000001	0.079741	0.778467	0.141791	
Market weights	0.010631	0.017235	0.011753	0.693932	0.190550	
Active weights	-0.010631	-0.017234	0.067988	0.084535	-0.048759	

(continues on next page)

(continued from previous page)

	BIG HiBM
Constrained weights	0.0000
Market weights	0.0759
Active weights	-0.0759

4.3 Cross-sectional regressions

The Fama-MacBeth (1973) methodology estimates factor betas and risk premia in two steps:

1. **Time-Series Regressions:** Estimate each asset's beta by regressing its returns against proposed factor returns.
2. **Cross-Sectional Regressions:** Estimate factor risk premia by regressing all asset returns against their betas across multiple time periods.

This approach provides standard errors adjusted for cross-sectional correlation.

4.3.1 Testing the CAPM

We retrieve test asset returns and the risk-free rate. To test whether beta is priced, we examine whether higher-beta assets earn proportionally higher risk premia. We also test for **non-linearity** (beta-squared) and for the pricing of residual risk.

```

factors = FFReader('F-F_Research_Data_Factors')[0] / 100 # risk-free rates
test_assets = FFReader('25_Portfolios_ME_BETA_5x5')
df = test_assets[1] / 100
df = df.sub(factors['RF'], axis=0).dropna().copy()

# unpivot the wide table to a long one
rets = df.stack() \
    .reset_index(name='ret') \
    .rename(columns={'level_1':'port', 'level_0':'Date'})

# estimate test assets' market betas from their time-series of returns
data = df.join(factors[['Mkt-RF']], how='left')
betas = least_squares(data, y=df.columns, x=['Mkt-RF'], stdres=True)
betas = betas.rename(columns={'Mkt-RF': 'BETA'})[['BETA', '_stdres']]

# collect test asset mean returns and betas
assets_df = betas[['BETA']].join(df.mean().rename('premiums')).sort_values('BETA')

# Orthogonalize polynomial (quadratic) beta^2 and residual-volatility features
betas['BETA2'] = smf.ols("I(BETA**2) ~ BETA", data=betas).fit().resid
betas['RES'] = smf.ols("_stdres ~ BETA + BETA2", data=betas).fit().resid
r = rets.join(betas, on='port').sort_values(['port', 'Date'], ignore_index=True)

# run monthly Fama-MacBeth cross-sectional regressions
fm = r.groupby(by='Date') \
    .apply(least_squares, y=['ret'], x=['BETA', 'BETA2', 'RES'])

```

```
/tmp/ipykernel_1785309/3663017786.py:3: DeprecationWarning: DataFrameGroupBy.apply_
  ↪operated on the grouping columns. This behavior is deprecated, and in a future_
  ↪version of pandas the grouping columns will be excluded from the operation.|
  ↪Either pass `include_groups=False` to exclude the groupings or explicitly select_
  ↪the grouping columns after groupby to silence this warning.
  .apply(least_squares, y=['ret'], x=['BETA', 'BETA2', 'RES'])
```

```
# compute time-series means and standard errors of the Fama-MacBeth coefficients
out = DataFrame(dict(mean=fm.mean(), stderr=fm.sem(), tstat=fm.mean()/fm.sem())).T
```

```
print("Monthly Cross-sectional Regressions" +
      f" {min(rets['Date'])} to {max(rets['Date'])}")
out
```

Monthly Cross-sectional Regressions 1963-07 to 2024-12

	_intercept	BETA	BETA2	RES
mean	0.007817	0.000124	-0.008395	0.121528
stderr	0.001535	0.002255	0.002544	0.060507
tstat	5.091416	0.054945	-3.299643	2.008509

Clustered standard errors

Alternative corrections for standard errors account for both time-series and cross-sectional correlation, such as double clustering by firm and year.

```
### Compare uncorrected to robust cov
ls = smf.ols("ret ~ BETA + BETA2 + RES", data=r).fit()
print(ls.summary())
# print(ls.get_robustcov_results('HC0').summary())
# print(ls.get_robustcov_results('HAC', maxlags=6).summary())
```

OLS Regression Results						
Dep. Variable:	ret	R-squared:	0.000			
Model:	OLS	Adj. R-squared:	0.000			
Method:	Least Squares	F-statistic:	2.554			
Date:	Mon, 03 Mar 2025	Prob (F-statistic):	0.0535			
Time:	07:02:21	Log-Likelihood:	25722.			
No. Observations:	18450	AIC:	-5.144e+04			
Df Residuals:	18446	BIC:	-5.141e+04			
Df Model:	3					
Covariance Type:	nonrobust					
coef	std err	t	P> t	[0.025	0.975]	
Intercept	0.0078	0.002	4.174	0.000	0.004	0.011
BETA	0.0001	0.002	0.076	0.940	-0.003	0.003
BETA2	-0.0084	0.006	-1.390	0.165	-0.020	0.003
RES	0.1215	0.051	2.393	0.017	0.022	0.221
Omnibus:	1547.660	Durbin-Watson:				1.778
Prob(Omnibus):	0.000	Jarque-Bera (JB):				9926.401

(continues on next page)

(continued from previous page)

```
Skew:           -0.051   Prob (JB):           0.00
Kurtosis:      6.592    Cond. No.          174.
=====
```

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly ~~specified~~.

```
print(ls.get_robustcov_results('hac-panel',
                               groups=r['port'],
                               maxlags=6).summary())
# print(ls.get_robustcov_results('cluster', groups=r['port']).summary())
```

OLS Regression Results

```
=====
Dep. Variable:                  ret    R-squared:           0.000
Model:                          OLS    Adj. R-squared:      0.000
Method: Least Squares          F-statistic:          1.636
Date:  Mon, 03 Mar 2025        Prob (F-statistic):  0.207
Time: 07:02:21                 Log-Likelihood:     25722.
No. Observations:            18450   AIC:                 -5.144e+04
Df Residuals:                 18446   BIC:                 -5.141e+04
Df Model:                      3
Covariance Type:             hac-panel
=====
      coef    std err          t      P>|t|      [ 0.025    0.975]
-----
Intercept    0.0078    0.002      4.003      0.001      0.004    0.012
BETA        0.0001    0.002      0.066      0.948     -0.004    0.004
BETA2       -0.0084    0.006     -1.330      0.196     -0.021    0.005
RES         0.1215    0.066      1.847      0.077     -0.014    0.257
=====
Omnibus:            1547.660   Durbin-Watson:      1.778
Prob(Omnibus):      0.000    Jarque-Bera (JB):  9926.401
Skew:              -0.051    Prob(JB):            0.00
Kurtosis:           6.592    Cond. No.          174.
=====
```

Notes:

[1] Standard Errors are robust to cluster correlation (HAC-Panel)

4.3.2 Factor risk models

Beginning with Barra in the mid-1970's, industry practitioners have employed cross-sectional models to forecast risk premia and measure risk factors. Monthly cross-sectional regressions are run on standardized individual stock characteristics such as:

- **Size:** Market capitalization rank
- **Value:** Book-to-market ratio
- **Momentum:** 12-month return (excluding past month)
- **Reversal:** 1-month return reversal

The stock characteristic values are winsored at their 5% tails to reduce the influence of outliers. These risk premia are interpreted as returns on dollar-neutral portfolios with unit exposure to a characteristic and zero exposure to other characteristics.

Their estimated risk premiums are compared to Fama-French research factor returns (which are constructed as returns from long-short spread portfolios).

```
# Construct momentum and reversal signals

rebalbeg = 19640601
rebalend = LAST_DATE
# preload CRSP monthly into memory
monthly = CRSPBuffer(stocks=crsp, dataset='monthly',
                      fields=['ret', 'retx', 'prc'],
                      beg=bd.begmo(rebalbeg, -13),
                      end=bd.endmo(rebalend, 1))    # load with extra months around
rebaldates = rebaldates

intervals = {'mom': (2, 12, 1), 'strev': (1, 1, -1)} # signal: (start, end, sign)
for label, past in intervals.items():
    out = []
    rebaldates = bd.date_range(bd.endmo(rebalbeg), rebalend, 'endmo')
    for rebaldate in tqdm(rebaldates, total=len(rebaldates)):
        start = bd.endmo(rebaldate, -past[1])
        beg1 = bd.offset(start, 1)
        end1 = bd.endmo(rebaldate, 1-past[0])
        df = monthly.get_universe(end1)
        # require data available at start month and at last month (universe)
        df['start'] = monthly.get_section(dataset='monthly',
                                           fields=['ret'],
                                           date_field='date',
                                           date=start).reindex(df.index)
        df[label] = past[2] * monthly.get_ret(beg1, end1).reindex(df.index)
        df['permno'] = df.index
        df['rebaldate'] = rebaldate
        df = df.dropna(subset=['start'])
        out.append(df[['rebaldate', 'permno', label]]) # append rows
    out = pd.concat(out, axis=0, ignore_index=True)
    n = signals.write(out, label, overwrite=True)
```

```
100%|██████████| 726/726 [00:43<00:00, 16.53it/s]
100%|██████████| 726/726 [00:41<00:00, 17.50it/s]
```

```
rebalbeg = 19640601
rebalend = LAST_DATE
rebaldates = crsp.bd.date_range(rebalbeg, rebalend, 'endmo')
loadings = dict()

# preload signal values
sf = {key: SignalsFrame(signals.read(key)) for key in ['hml', 'mom', 'strev']}

for pordate in tqdm(rebaldates): # retrieve signal values every month
    date = bd.june_universe(pordate)
    univ = crsp.get_universe(date)
    smb = univ['capco'].rank(ascending=False).div(len(univ)).rename('smallsize')
    hml = sf['hml']('hml', date, bd.endmo(date, -12))['hml'].rename('value')
```

(continues on next page)

(continued from previous page)

```

#beta = signals('beta', pordate, bd.begmo(pordate))['beta']*2/3 + 1/3 #shrink
mom = sf['mom']('mom', pordate)['mom'].rename('momentum')
strev = sf['strev']('strev', pordate)['strev'].rename('reversal')
df = pd.concat((strev, hml, smb, mom), join='inner', axis=1) \
    .reindex(univ.index).dropna()
loadings[pordate] = winsorize(df, quantiles=[0.05, 0.95])

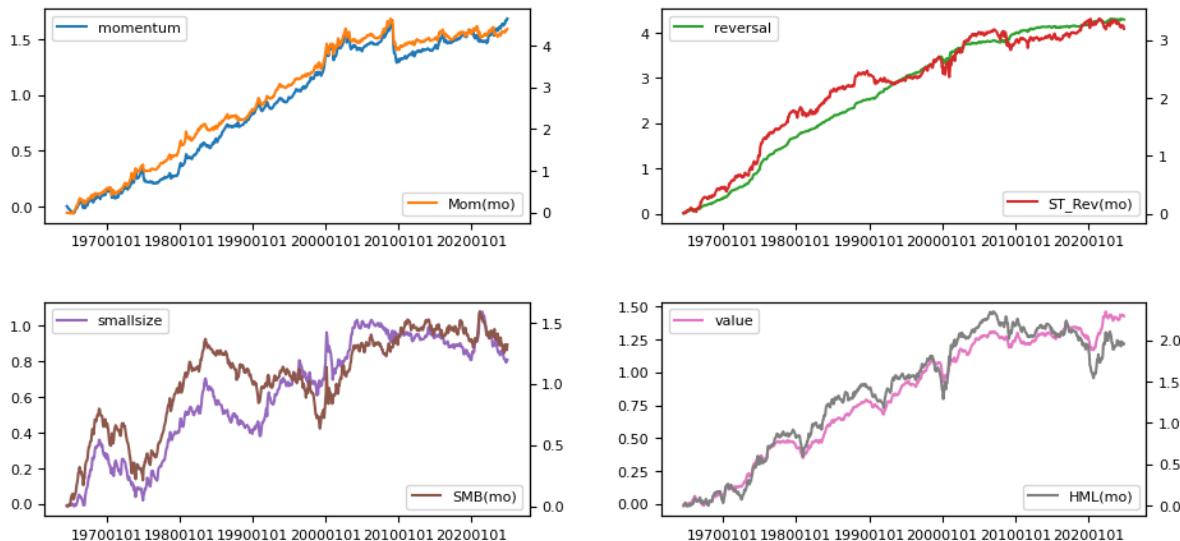
# Compute coefficients from FM cross-sectional regressions
monthly = CRSPBuffer(stocks=crsp, dataset='monthly', fields=['ret'],
                      beg=bd.begmo(rebalbeg, -13), end=bd.endmo(rebalend, 1))

riskpremium = RiskPremium(sql=user, bench=bench, rf='RF', end=LAST_DATE)
out = riskpremium(stocks=monthly, loadings=loadings,
                  standardize=['value', 'smallsize', 'momentum', 'reversal'])

# Compare time series of risk premiums to portfolio-sort benchmark eturns
benchnames = {'momentum': 'Mom(mo)',
              'reversal': 'ST_Rev(mo)',
              'smallsize': 'SMB(mo)',
              'value': 'HML(mo)'}
out = riskpremium.fit(benchnames.values()) # to compare portfolio-sorts
riskpremium.plot(benchnames)

```

100% |██████████| 726/726 [00:31<00:00, 22.93it/s]



```

# Summarize time-series means of Fama-Macbeth risk premiums
df = out[0]
df['tvalue'] = df['mean']/df['stderr']
df['sharpe'] = np.sqrt(12) * df['mean']/df['std']
print("Fama-MacBeth Cross-sectional Regression Risk Premiums")
df.round(4)

```

Fama-MacBeth Cross-sectional Regression Risk Premiums

Factor Returns	mean	stderr	std	count	tvalue	sharpe
reversal	0.0059	0.0005	0.0143	725	11.1314	1.4321
value	0.0020	0.0004	0.0112	725	4.7378	0.6095
smallsize	0.0011	0.0007	0.0178	725	1.6812	0.2163
momentum	0.0023	0.0007	0.0175	725	3.5671	0.4589

```
# Summarize time-series means of Fama-French portfolio-sort returns
df = out[2]
df['tvalue'] = df['mean']/df['stderr']
df['sharpe'] = np.sqrt(12) * df['mean']/df['std']
print("Fama-French Portfolio-Sorts")
df.round(4)
```

Fama-French Portfolio-Sorts

Benchmarks	mean	stderr	std	count	tvalue	sharpe
Mom (mo)	0.0061	0.0016	0.0423	725	3.8647	0.4972
ST_Rev (mo)	0.0044	0.0012	0.0315	725	3.7562	0.4833
SMB (mo)	0.0018	0.0011	0.0308	725	1.5941	0.2051
HML (mo)	0.0027	0.0011	0.0301	725	2.4116	0.3103

```
# Show correlation of returns
print('Correlation of FM-Cross-sectional Risk Premiums and FF-Sorted Portfolio Returns')
pd.concat([out[1].join(out[4]), out[4].T.join(out[3])], axis=0).round(3)
```

Correlation of FM-Cross-sectional Risk Premiums and FF-Sorted Portfolio Returns

	reversal	value	smallsize	momentum	Mom (mo)	ST_Rev (mo)	\
reversal	1.000	0.016	0.077	-0.444	-0.388	0.793	
value	0.016	1.000	-0.227	-0.194	-0.169	-0.019	
smallsize	0.077	-0.227	1.000	-0.003	0.133	0.008	
momentum	-0.444	-0.194	-0.003	1.000	0.884	-0.279	
Mom (mo)	-0.388	-0.169	0.133	0.884	1.000	-0.307	
ST_Rev (mo)	0.793	-0.019	0.008	-0.279	-0.307	1.000	
SMB (mo)	0.145	-0.203	0.517	-0.066	-0.047	0.178	
HML (mo)	0.051	0.810	-0.151	-0.207	-0.195	0.013	
			SMB (mo)	HML (mo)			
reversal	0.145	0.051					
value	-0.203	0.810					
smallsize	0.517	-0.151					
momentum	-0.066	-0.207					
Mom (mo)	-0.047	-0.195					
ST_Rev (mo)	0.178	0.013					
SMB (mo)	1.000	-0.150					
HML (mo)	-0.150	1.000					

4.4 Nonlinear regression

4.4.1 Feature transformations

A simple way to directly extend the linear model to accommodate non-linear relationships, using polynomial regression, is to include transformed versions of the predictors in the model, such as a quadratic term or several polynomial functions of the predictors, and use standard linear regression to estimate coefficients in order to produce a non-linear fit. The CAPM predicts that these coefficients on non-linear transformations of beta should be zero.

Raw polynomial terms may be highly correlated with each other: *Orthogonal polynomials* transform the raw data matrix of polynomial terms to another whose columns are a basis of orthogonal terms which span the same column space. For example, regress the second predictor on the first and replace its column with the residuals, then regress the third predictor on the first two and replace its column with the residuals, and so on.

Other feature transformation approaches include:

- dummy or binary indicator variable
- categorical variables with two or more levels
- binarization or turning a categorical variable into several binary variables (4)
- Legendre polynomials which are defined as a system of orthogonal polynomials over the interval $[-1, 1]$
- interaction term constructed by computing the product of the values of the two variables to capture the effect that response of one predictor is dependent on the value of another predictor.

4.4.2 Kernel regression

If there are already a large number of k features, then polynomial transformations, say up to degree d , may be computational expensive since we could be working in $O(k^d)$ dimensional space. Fortunately, many high-dimensional feature mappings, denoted $\phi(x)$, correspond to kernel functions K , where model fitting and prediction calculations only require inner products of these kernel matrices and we never need to explicitly represent vectors in the very high-dimensional feature space. For example, the kernel $K(x, y) = (x^T y + c)^d$, which requires only $O(k)$ to compute, expands to the feature space corresponding with all polynomial terms up to degree d of the features in x and y .

Kernels can be viewed as similarity metrics, that measure how close together the feature maps $\phi(x)$ and $\phi(y)$ are. The radial basis function (RBF), or Gaussian, kernel uses distance in Euclidean space which corresponds to an infinite-dimension feature mapping.

This application of Kernel functions that can be efficiently computed, where only their inner products are needed without ever explicitly computing their corresponding feature vectors in very high-dimensional space, has come to be known as the **kernel trick**.

factors

	Mkt-RF	SMB	HML	RF
Date				
1926-07	0.0296	-0.0256	-0.0243	0.0022
1926-08	0.0264	-0.0117	0.0382	0.0025
1926-09	0.0036	-0.0140	0.0013	0.0023
1926-10	-0.0324	-0.0009	0.0070	0.0032
1926-11	0.0253	-0.0010	-0.0051	0.0031
...
2024-08	0.0161	-0.0355	-0.0113	0.0048
2024-09	0.0174	-0.0017	-0.0259	0.0040

(continues on next page)

(continued from previous page)

```
2024-10 -0.0097 -0.0101  0.0089  0.0039
2024-11  0.0651  0.0463 -0.0005  0.0040
2024-12 -0.0317 -0.0273 -0.0295  0.0037

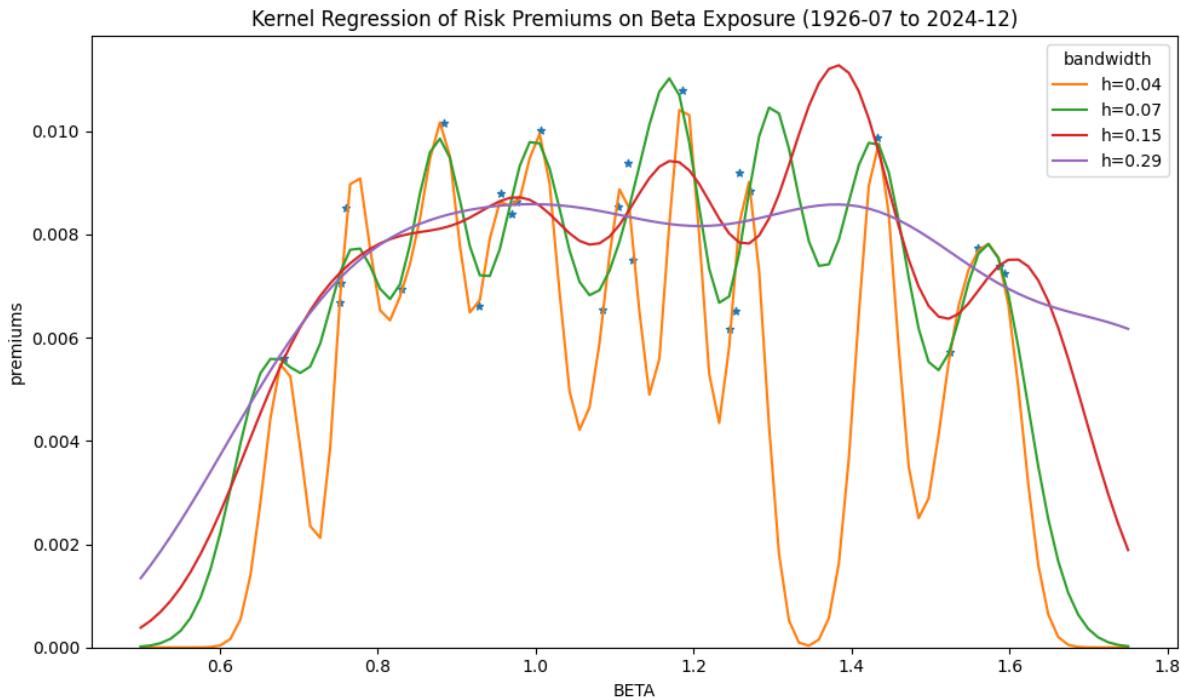
[1182 rows x 4 columns]
```

The concave shape of the fitted kernel regression curve is consistent with the negative average premiums observed for the squared-beta factor in earlier Fama-MacBeth tests.

```
y_train = assets_df[['premiums']].values
X_train = assets_df[['BETA']].values
X_test = np.linspace(0.5, 1.75, 100).reshape(-1, 1)
bandwidth = float((max(X_train) - min(X_train)) * 4 / len(X_train))

fig, ax = plt.subplots(figsize=(10, 6))
legend = []
color = 1
for h in [0.25, 0.5, 1, 2]:
    for alpha in [0.01]:
        model = KernelRidge(alpha=alpha, kernel='rbf', gamma=1/(h*bandwidth)**2)
        model.fit(X=X_train, y=y_train)
        y_pred = model.predict(X_test)
        ax.plot(X_test, y_pred, ls='-', color=f"C{color}")
        legend.append(f"h={h*bandwidth:.2f}")
        color += 1
# scatter plot actual
assets_df.plot(x='BETA', y='premiums', kind='scatter', ax=ax, marker="*", color="C0")
ax.set_xlim(left=0)
plt.legend(legend, loc='best', title='bandwidth')
plt.title('Kernel Regression of Risk Premiums on Beta Exposure' +
          f" ({factors.index[0]} to {factors.index[-1]})")
plt.tight_layout()
```

```
/tmp/ipykernel_1785309/182542393.py:4: DeprecationWarning: Conversion of an array
  with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you
  extract a single element from your array before performing this operation.
  (Deprecated NumPy 1.25.)
  bandwidth = float((max(X_train) - min(X_train)) * 4 / len(X_train))
```



References:

- H. M. Markowitz, "Portfolio Selection," *Journal of Finance* 7, 1952, pp. 77–91.
- W. F. Sharpe, "Capital Asset Prices: A Theory of Market Equilibrium under Conditions of Risk," *Journal of Finance* 19, 1964, pp. 425–442.
- J. Lintner, "Security Prices, Risk and Maximal Gains from Diversification," *Journal of Finance* 20, 1965, pp. 587–615, and J. Mossin, "Equilibrium in a Capital Asset Market," *Econometrica* 34, 1966, pp. 768–783.
- Fama, Eugene F.; MacBeth, James D. (1973). "Risk, Return, and Equilibrium: Empirical Tests". *Journal of Political Economy*. 81 (3): 607–636.
- Black, F., and R. Litterman. 1992. "Global Portfolio Optimization." *Financial Analysts Journal*, vol. 48, no. 5 (September/October): 28-43
- He, G., and R. Litterman. 1999. "The Intuition Behind Black-Litterman Model Portfolios." *Goldman Sachs Investment Management Series*.
- R. C. Jones, T. Lim and P. J. Zangari. "The Black-Litterman Model for Structured Equity Portfolios," *Journal of Portfolio Management*, Vol. 33, No. 2, 2007, pp. 24-43. doi:10.3905/jpm.2007.674791
- FRM Part I Exam Book Foundations of Risk Management Ch. 5.

CONTRARIAN TRADING

Fortune befriends the bold - Emily Dickinson

Contrarian trading strategies are based on the premise of mean reversion, which posits that asset prices tend to revert to their long-term average over time. This idea is integral to various investment strategies, as it suggests that prices that have deviated significantly from historical norms will eventually return to their equilibrium levels. In these strategies, mispricing can occur due to investor overreaction, leading to temporary opportunities for profitable trades. Common approaches, such as pairs trading and statistical arbitrage, capitalize on these price deviations by simultaneously taking opposing positions in correlated assets. This analysis examines the construction of a contrarian trading strategy, evaluates its performance using key risk-adjusted metrics like the Sharpe ratio, and assesses the effects of implementation shortfall and structural breaks in the strategy's effectiveness over time.

```
import numpy as np
import pandas as pd
from pandas import DataFrame, Series
from tqdm import tqdm
import matplotlib.pyplot as plt
import rpy2.robj as ro
from rpy2.robj.packages import importr
from finds.database import SQL, RedisDB
from finds.structured import BusDay, CRSP, Benchmarks
from finds.recipes import fractile_split, least_squares
from finds.utils import PyR, row_formatted
from secret import credentials, paths, CRSP_DATE
VERBOSE = 0
#%matplotlib qt

importr('strucchange')    # R package to use
```

```
rpy2.robj.packages.Package as a <module 'strucchange'>
```

```
# open connections
sql = SQL(**credentials['sql'], verbose=VERBOSE)
rdb = RedisDB(**credentials['redis'])
bd = BusDay(sql, verbose=VERBOSE)
crsp = CRSP(sql, bd, rdb=rdb, verbose=VERBOSE)
bench = Benchmarks(sql, bd)
```

5.1 Mean reversion

The concept of **mean reversion** forms the foundation of many trading strategies. According to the **law of one price**, similar assets should be priced similarly. When this does not hold, traders can exploit the mispricing through **arbitrage** by buying undervalued assets and short-selling overvalued ones, causing the prices to converge. Deviations from expected long-term values, such as yield spreads in fixed income assets, are typically not persistent. **Pairs trading** involves selecting two related securities and trading them to profit from their temporary price divergences. Meanwhile, **statistical arbitrage** (or **stat arb**) relies on complex algorithms to identify and exploit statistical relationships between securities. Although these relationships are not risk-free, stat arb strategies diversify across numerous positions to reduce exposure to risk. However, correlations between asset classes often increase during financial crises due to liquidity constraints and deleveraging, complicating risk management.

We evaluate a **contrarian strategy** based on weekly returns of US stocks, driven by the idea that stock mispricing arises from investor overreaction, as discussed by Lo and Mackinlay (1990). The **alpha** of this strategy can be measured by its **information coefficient** and **volatility** components, as outlined by Grinold (1994). To examine the strategy's effectiveness over time, we apply tests for **structural breaks** with unknown change points. A key consideration in this analysis is the discrepancy between theoretical portfolio returns, based on assumed execution at decision prices, and actual returns, influenced by factors such as trading costs. The concept of **implementation shortfall** (introduced by Perold in 1988) captures the total cost of executing an investment decision, accounting for both explicit and implicit costs like market impact and opportunity costs.

Weekly returns reversals

Daily returns from CRSP are compounded into weekly stock returns assuming Wednesday-close to Wednesday-close. This skips over weekend and holiday (which often occur over a long weekend) effects. The backtest starts in January 1974, after an expansion of the stocks universe the previous year, and excludes the smallest market cap quintile (based on NYSE breakpoints) comprising microcap stocks.

Let \bar{r}_t and σ_t be the cross-sectional mean and standard deviation of stock returns in week t . Define $\tilde{r}_t = r_t - \bar{r}_t$ as the vector of demeaned stock returns, and $X_t = -\tilde{r}_t/\sigma_t$ as the vector of normalized scores. In other words, stocks' "exposures" to (minus) their respective prior week's returns are standardized to have cross-sectional variance and standard deviation equal to 1.0.

Each week, a portfolio is rebalanced to hold an amount in each stock equal to their respective exposure values divided by the number of holdings $w_t = X_t/n$. hence the portfolio overall has unit exposure to prior week's returns $w_t^T X_t = 1$ and is dollar-neutral $\sum w_t = \sum X_t/n = 0$.

This portfolio construction approach can more generally incorporate additional signals where X may be a matrix with ones in the first column and standardized signal exposures in other columns. Then each row, except the first, of $W = (X'X)^{-1}X'$ contains stock weights of a long-short characteristic portfolio: a dollar-neutral, minimum-norm (in terms of squared weights) portfolio with unit exposure through long positions in stocks with positive exposure and short positions in stocks with negative exposure to the signal, and zero exposure to the other characteristics.

The portfolio's realized return in the following week $t + 1$ is

$$W_t' r_{t+1} = \frac{X_t' \tilde{r}_t}{n} + \frac{X_t' \hat{r}_t}{n} = -\frac{\tilde{r}_t' \tilde{r}_{t+1} \sigma_{t+1}}{n \sigma_t \sigma_{t+1}} = -\rho_{t,t+1} \sigma_{t+1}$$

that is the product of the (negative) cross-sectional correlation of stock returns times the amount of cross-sectional stock volatility at week $t + 1$.

```
weekday = 3           # wednesday close-to-close
bd = BusDay(sql, endweek=weekday)  # Generate weekly cal
begweek = 19740102  # increased stocks coverage in CRSP in Jan 1973
endweek = bd.endwk(CRSP_DATE, -1)
rebaldates = bd.date_range(begweek, endweek, freq='weekly')
retdates = bd.date_tuples(rebaldates)
```

(continues on next page)

(continued from previous page)

```

june_universe = 0 # to track date when reached a June end to update universe
year = 0 # to track new year to pre-load stocks datas in batch by year
results = []
lagged_weights = Series(dtype=float) # to track "turnover" of stock weights
for rebaldate, pastdates, nextdates in tqdm(zip(
    rebaldates[1:-1], retdates[:-1], retdates[1:]), total=len(rebaldates)-1):

    # screen universe each June: largest 5 size deciles
    d = bd.june_universe(rebaldate)
    if d != june_universe: # need next June's universe
        june_universe = d # update universe every June
        univ = crsp.get_universe(june_universe) # usual CRSP universe screen
        univ = univ[univ['decile'] <= 8] # drop smallest quintile stocks

    # retrieve new annual batch of daily prices and returns when start new year
    if bd.begyr(rebaldate) != year:
        year = bd.begyr(rebaldate)
        prc = crsp.get_range(dataset='daily',
            fields=['bidlo', 'askhi', 'prc', 'retx', 'ret'],
            date_field='date',
            beg=year,
            end=bd.offset(bd.endyr(year), 10),
            cache_mode="rw")

    # get past week's returns, require price at rebalance (decision) date
    past_week = prc[prc.index.get_level_values('date') == rebaldate]['prc']\
        .reset_index()\
        .set_index('permno')\
        .join(crsp.get_ret(*pastdates).reindex(univ.index))\
        .dropna()

    # convert past week's returns to desired standardized portfolio weights
    weights = ((past_week['ret'].mean() - past_week['ret']) / # mean
               (past_week['ret'].std(ddof=0) * len(past_week))) # std

    # adjust past week's holdings by change stock price
    lagged_weights = lagged_weights.mul(crsp.get_ret(*pastdates, field='retx')\
        .reindex(lagged_weights.index))\
        .fillna(0) + 1 # fillna(0) + 1

    # compute how much to buy (or sell) to achieve desired new portfolio weights
    chg_weights = pd.concat([weights, -lagged_weights], axis=1)\ # concat
        .fillna(0)\ # fillna(0)
        .sum(axis=1) # sum

    # calculate total abs weight as denominator for scaling turnover
    total_weight = weights.abs().sum() + lagged_weights.abs().sum()

    # get next week's gross returns
    next_week = crsp.get_ret(*nextdates).reindex(weights.index).fillna(0)

    # get next day's prices to compute one-day slippage cost
    next_day = prc[prc.index.get_level_values('date') == # get date
                  bd.offset(rebaldate, 1)]\ # offset
                  .reset_index()\

```

(continues on next page)

(continued from previous page)

```

.set_index('permno') \
.drop(columns='date') \
.reindex(chg_weights.index)

# if no trade next day, then enter position at askhi (buy) or bidlo (sell)
bidask = next_day['askhi'].where(chg_weights > 0, next_day['bidlo']).abs()

# spread is relevant askhi or bidlo divided by recorded close, minus 1
spread = next_day['prc'].where(next_day['prc'] > 0, bidask) \
    .div(next_day['prc'].abs()) \
    .sub(1) \
    .fillna(0)

# finally, trade_prc is the next day's close, or the relevant askhi or bidlo
trade_prc = next_day['prc'].where(next_day['prc'] > 0, bidask).fillna(0)

# drift is next day's trade price with dividends over today's decision price
# delay (positive is cost) will be chg_weights * drift
drift = trade_prc.div(next_day['prc'].abs()) \
    .mul(1 + next_day['ret']) \
    .sub(1) \
    .fillna(0)

# exit and enter delay should sum to chg_weights.dot(next_day['ret'])
exit1 = -lagged_weights.dot(next_day['ret']).reindex(lagged_weights.index) \
    .fillna(0)
enter1 = weights.dot(next_day['ret']).reindex(weights.index).fillna(0)

# accumulate weekly calculations
results.append(DataFrame(
    {'ret': weights.dot(next_week),
     'exit1': exit1,
     'enter1': enter1,
     'delay': chg_weights.dot(next_day['ret'].fillna(0)), # delay=enter+exit
     'spread': chg_weights.dot(spread),
     'slippage': chg_weights.dot(drift), # total slippage
     'ic': weights.corr(next_week),
     'n': len(next_week),
     'beg': nextdates[0],
     'end': nextdates[1],
     'absweight': np.sum(weights.abs()),
     'turnover': chg_weights.abs().sum() / total_weight,
     'vol': next_week.std(ddof=0)},
    index=[rebaldate])))

# carry forward to next week as lagged portfolio weights
lagged_weights = weights

```

100% [██████████] 2659/2660 [01:29<00:00, 29.74it/s]

```

# Combine accumulated computations and report
df = pd.concat(results, axis=0)
dates = df.index
df.index = pd.DatetimeIndex(df.index.astype(str))

```

(continues on next page)

(continued from previous page)

```
df['net'] = df['ret'].sub(df['slippage'])
# Show summary
cols = ['ic', 'vol', 'ret', 'slippage', 'net', 'exit1', 'enter1', 'delay',
        'spread', 'turnover']
indexes = ['Information coefficient', 'Cross-sectional Volatility',
           'Gross return (alpha)', 'Slippage cost', 'Net (of slippage) return',
           'Exit one day delay', 'Enter one day delay', 'Delay cost',
           'Spread cost', 'Portfolio turnover']
```

```
print(f'Summary of Weekly Mean Reversion Strategy {dates[0]}-{dates[-1]}\n')
pd.concat([df[cols].mean(axis=0).rename('mean'),
           df[cols].std(axis=0).rename('std')], axis=1)\n
.set_index(pd.Index(indexes)).round(4)
```

Summary of Weekly Mean Reversion Strategy 19740109–20241218

	mean	std
Information coefficient	0.0376	0.1022
Cross-sectional Volatility	0.0536	0.0191
Gross return (alpha)	0.0021	0.0080
Slippage cost	0.0017	0.0047
Net (of slippage) return	0.0005	0.0078
Exit one day delay	-0.0003	0.0029
Enter one day delay	0.0006	0.0036
Delay cost	0.0004	0.0041
Spread cost	0.0013	0.0023
Portfolio turnover	0.7363	0.0407

5.2 Implementation shortfall

Perold (1988) observed that the a paper portfolio based upon a well-known stock rankings system significantly outperformed the actual track record of funds that make use of the system. He defined implementation shortfall as the difference in return between a theoretical portfolio and the implemented portfolio, which captures explicit fees and commissions as well as market impact, delay and opportunity costs.

- **Decision price** is the price at the time the investment decision was made
- **Arrival price** is the midquote (mid-point of bid-ask prices) at the time the trader, broker or trading system received the order, or the trade decision is made.
- **Market drift** is the amount of buys (sells) multiplied by the increase (decrease) in execution price relative to the arrival price, due to execution delay.
- **Delay** is the adverse change in execution price relative to the decision price
- **Opportunity costs** are the profits lost due to trades that are cancelled or not executed.
- **Market impact costs** are bid-ask spreads as well as the amount that buying or selling moves the price against the buyer or seller

The **trader's dilemma** refers to the trade-off between market drift and impact: one can trade faster with more impact to minimize market drift, or trade slower to minimize market impact but at the risk of the market drifting away.

In practice, stocks in the CRSP database are not available at finer than daily frequency, so we assume that trades are executed at the next day's closing prices.

If stocks do not trade, we estimate execution prices based on bid-ask spreads. This approach helps address slippage and delays in execution, although transaction costs cannot be directly observed in historical backtests.

Unfortunately, stock prices in CRSP are not available at a finer than daily frequency. We adjust estimated profits for slippage by waiting a full day after the decision price then setting the execution price at the next day's closing price, when stock market exchanges typically experience the most liquidity at the close of a trading dollar. For stocks that did not trade during that day, we assumed the desired buys are executed at the (higher) ask price and sells are executed at the (lower) bid price (in CRSP for such cases, closing bid and ask quotes are recorded and the closing price is set to the negative of the bid-ask average). This approach helps address slippage and delays in execution, although transaction costs cannot be directly observed in historical backtests.

Over the full period, much of the profitability of this version of the strategy appears to be dissipated after considering a one-day execution delay and bid-ask spreads.

5.3 Structural break with unknown changepoint

In a linear regression, the Chow test is commonly used to test for the presence of a structural break in the model at a period known *a priori*; it essentially constructs a test of whether the true coefficients on the independent variable split into two subsets are equal. Welch's test that two populations have equal means is a special case with no independent variables and only an intercept whose true values are tested in the two time periods. However, when the breakpoints are unknown, these standard tests are not applicable. Andrews (1993) and others have developed alternative tests, based on *supremum statistics*, for identifying changes in mean that occur at unknown points in the time series.

The **R** library `strucchange` provides tools for detecting structural breaks, and the **rpy2** Python package facilitates integration with R. The `PyR` wrapper class in the `FinDS` package facilitates converting Pandas DataFrames to and from R objects and function calls.

```
# Structural Break Test with Unknown Changepoint

# Set up data and formulas for R
Y = df['ret']
formula = ro.Formula('y ~ 1')
formula.environment['y'] = PyR(Y.values).ro

# Call R strucchange routines to compute breakpoint statistics
fstats_r = ro.r['Fstats'](formula, **{'from': 1})      # Fstats at every break
breakpoints_r = ro.r['breakpoints'](formula)          # candidate breakpoints
confint_r = ro.r['confint'](breakpoints_r, breaks=1) # conf interval for 1 break
sctest_r = ro.r['sctest'](fstats_r, **{'type': 'aveF'})

# Extract output from R results
confint = PyR(confint_r[0]).frame.iloc[0].astype(int) - 1 # R index starts at 1

output = dict(zip(confint.index, df.index[confint]))      # confidence interval

for k,v in zip(sctest_r.slots['names'][:3], sctest_r[:3]): # significance values
    output[k] = PyR(v).values[0]
output['mean(pre)'] = Y[df.index <= output['breakpoints']].mean()
output['mean(post)'] = Y[df.index > output['breakpoints']].mean()

fstat = [0] + list(PyR(fstats_r[0]).values) + [0, 0] # pad before and after
```

(continues on next page)

(continued from previous page)

```
print("Structural break test with unknown changepoint")
DataFrame(output, index=['sctest'])
```

Structural break test with unknown changepoint

```
2.5 % breakpoints      97.5 %  statistic  p.value      method  \
sctest 1996-12-04  2001-05-30 2002-11-20  17.588886      0.0  aveF test

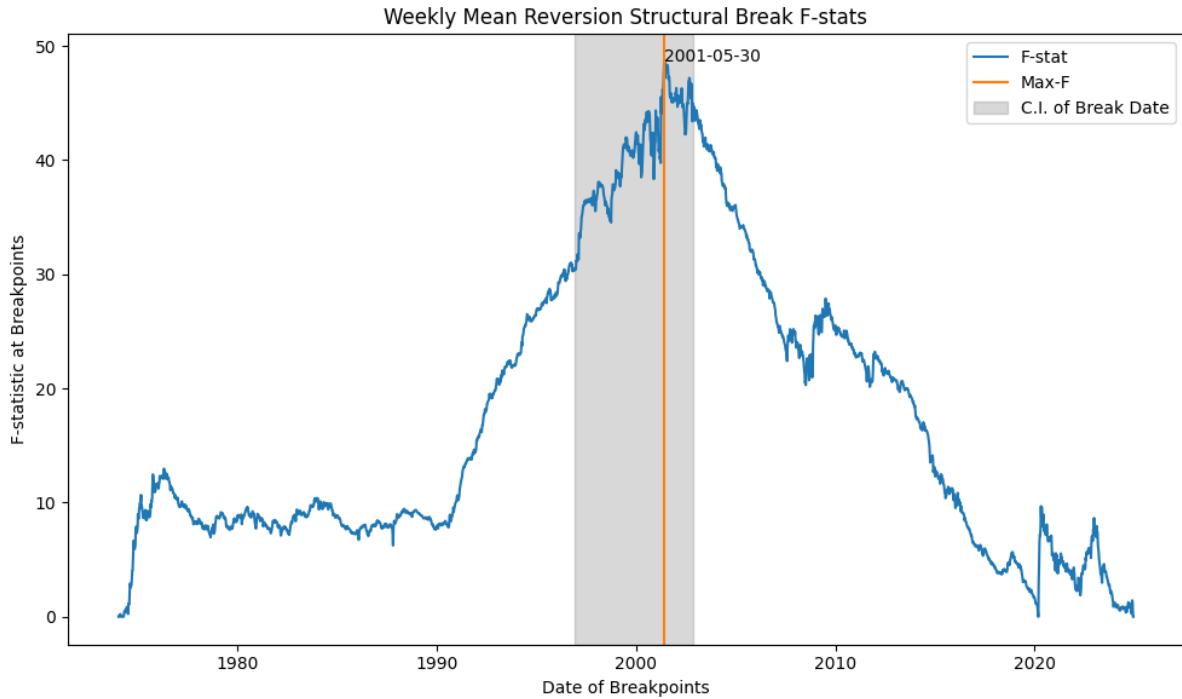
mean(pre)  mean(post)
sctest    0.003112    0.000969
```

Plot breakpoint F-stats

```
fig, ax = plt.subplots(num=2, clear=True, figsize=(10, 6))
ax.plot(df.index, fstat, color='C0')
argmax = np.nanargmax(fstat)                      # where maximum fstat
ax.axvline(df.index[argmax], color='C1')
ax.axvspan(df.index[confint[0]], df.index[confint[2]], alpha=0.3, color='grey')
ax.legend(['F-stat', 'Max-F', 'C.I. of Break Date'])
ax.annotate(
    df.index[argmax].strftime('%Y-%m-%d'), xy=(df.index[argmax], fstat[argmax]))
ax.set_ylabel('F-statistic at Breakpoints')
ax.set_xlabel('Date of Breakpoints')
ax.set_title('Weekly Mean Reversion Structural Break F-stats')
plt.tight_layout()
```

```
/tmp/ipykernel_1657351/3930510136.py:5: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
```

```
ax.axvspan(df.index[confint[0]], df.index[confint[2]], alpha=0.3, color='grey')
```



5.4 Performance evaluation

5.4.1 Information coefficient

Grinold (1994) linked the expected alpha of a signal to its information coefficient and an asset's signal score and idiosyncratic volatility: $\alpha = IC \times volatility \times score$.

- **IC:** The information coefficient can be understood as the correlation between a signal and residual returns. It tells how well forecasts align with actual returns and is a measure of manager forecasting skill.
- **Volatility:** The volatility can be understood as an asset's residual risk. This component allows for forecast alpha to be expressed in units of returns.
- **Score:** The score is a standardized measure of an asset's raw signal exposure, and reflects relative expectations about an asset. Standardization, by subtracting the cross-sectional mean and dividing by the cross-sectional standard deviation, allows assets to be compared to one another and over time.

Recall that the weekly reversal portfolio W_t was constructed to be dollar-neutral with exposure equal to 1.0, with weekly profitability $W_t' r_{t+1} = -\rho_{t,t+1} \sigma_{t+1}$. Hence the **IC** of this strategy can be computed using the negative cross-sectional correlation of stock returns over time. The rolling average of alpha and its components—information coefficient and volatility—reveal trends over the strategy's life. The data show that while volatility fluctuated over time, the **IC** sharply declined after reaching a peak in the mid-1990s, continuing its downward trajectory through 2010.

```
## Plot returns, and rolling avg information coefficient and cross-sectional vol
fig, ax = plt.subplots(num=1, clear=True, figsize=(10, 6))
df['ret'].cumsum().plot(ax=ax, ls='-', color='r', rot=0)
ax.legend(['cumulative returns'], loc='center left')
ax.set_ylabel('cumulative returns')
bx = ax.twinx()
roll = 250 # 250 week rolling average ~ 5 years
```

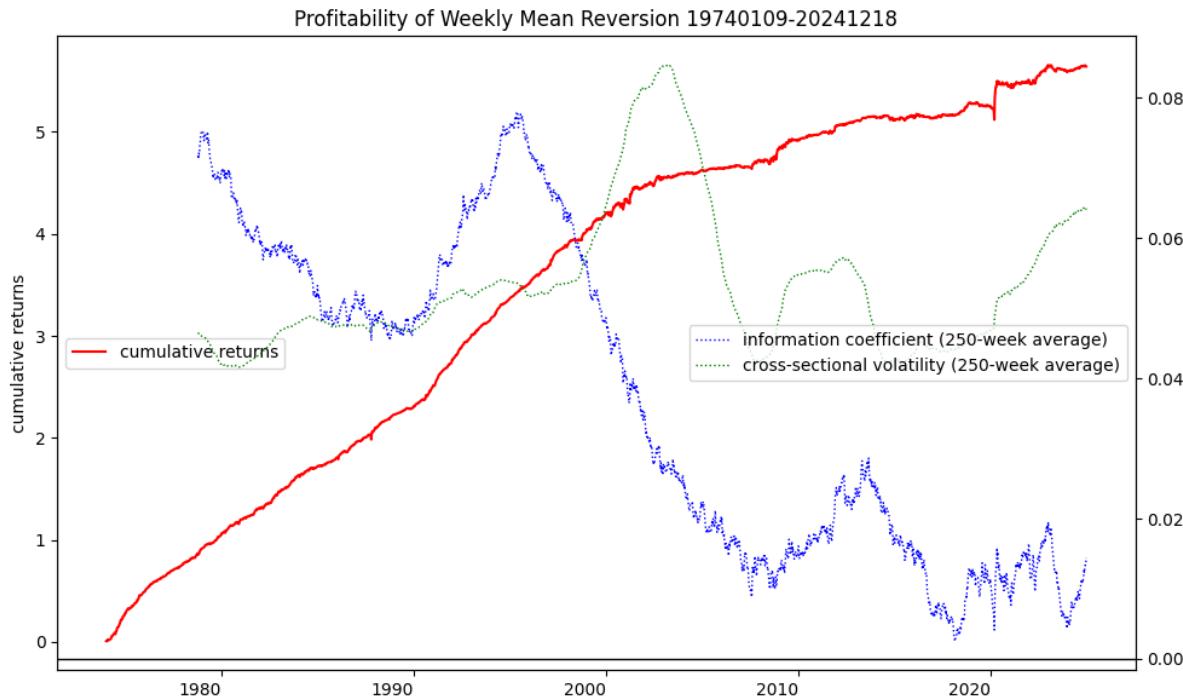
(continues on next page)

(continued from previous page)

```

df['ic'].rolling(roll).mean().plot(ax=bx, ls=':', lw=1, rot=0, color='b')
df['vol'].rolling(roll).mean().plot(ax=bx, ls=':', lw=1, rot=0, color='g')
#bx.axhline(df['ic'].mean(), linestyle='-', color='C0', lw=2)
bx.axhline(0, linestyle='-', color='black', lw=1)
bx.legend([f"information coefficient ({roll}-week average)",
           f"cross-sectional volatility ({roll}-week average)"],
          loc='center right')
ax.set_title(f'Profitability of Weekly Mean Reversion {dates[0]}-{dates[-1]}')
plt.tight_layout()

```



5.4.2 Risk-adjusted performance measures

In a CAPM equilibrium, no investor can achieve an abnormal return, and each investment yields an identical risk-adjusted return. In the real world, assets may yield a return in excess of, or below, that which fairly compensates for their risk exposure. To assess the strategy's risk-adjusted performance, we employ several key metrics:

- Sharpe ratio - slope of the capital market line is the fair equilibrium compensation: $\frac{R_P - r_f}{\sigma_P}$
- Treynor ratio - uses beta which is an appropriate measure of risk for a well-diversified portfolio: $\frac{R_P - r_f}{\beta_P}$
- Jensen's alpha - the intercept of a CAPM regression should be zero in equilibrium: $R_P - r_f - \beta_P(R_M - r_f)$
- Appraisal ratio - Jensen's alpha scaled by the volatility of residual returns $\frac{\alpha}{\sigma_{P-M}}$
- Sortino ratio - focuses on downside risk relative a target required rate of return T: $\frac{R_P - T}{\sqrt{\sum_t \min(0, r_t - T)^2 / N}}$

- Information ratio - adjusts performance relative to a benchmark, called the active return, scaled by the volatility of active returns, called tracking error:
$$\frac{\hat{R}_P - \hat{R}_B}{\sigma_{R_P - R_B}}$$
- M^2 (Modigliani-squared) - imagines that the given portfolio, P , is mixed with a position in T-bills so that the resulting portfolio P^* matches the volatility of the market portfolio. Because the market index and portfolio P^* have the same standard deviation, their performance may be compared by simply subtracting returns: $R_{P^*} - R_M$

```

market = bench.get_series(permnos=['Mkt-RF'], field='ret').reset_index()
breakpoint = BusDay.to_date(output['breakpoints'])
out = dict()
for select, period in zip([dates > 0, dates <= breakpoint, dates > breakpoint],
                         ['Full', 'Pre-break', 'Post-break']):
    res = df[select].copy()
    res.index = dates[select]

    # align market returns and compute market regression beta
    #res['date'] = res.index
    res['mkt'] = [(1 + market[market['date'].between(*dt)]['Mkt-RF']).prod() - 1
                  for dt in res[['beg', 'end']].itertuples(index=False)] 
    # model = lm(res['mkt'], res['ret'], flatten=True)
    model = least_squares(data=res, y=['ret'], x=['mkt'], stdres=True)

    # save df summary
    out[f'{period} Period'] = {
        'start date': min(res.index),
        'end date': max(res.index),
        'Sharpe Ratio': np.sqrt(52)*res['ret'].mean()/res['ret'].std(),
        'Average Gross Return': res['ret'].mean(),
        'Std Dev Returns': res['ret'].std(),
        'Market Beta': model.iloc[1],
        'Jensen Alpha (annualized)': model.iloc[0] * 52,
        'Appraisal Ratio': np.sqrt(52) * model.iloc[0] / model.iloc[2],
        'Information Coefficient': res['ic'].mean(),
        'Cross-sectional Vol': res['vol'].mean(),
        'Total Slippage Cost': res['slippage'].mean(),
        'Spread Cost': res['spread'].mean(),
        'Delay Cost': res['delay'].mean(),
        'Exit Delay Cost': res['exit1'].mean(),
        'Enter Delay Cost': res['enter1'].mean(),
        'Average Net Return': res['net'].mean(),
        'Portfolio Turnover': res['turnover'].mean(),
        #'Abs Weight': res['absweight'].mean(),
        'Average Num Stocks': int(res['n'].mean()),
    }
}

```

```

# Display as formatted DataFrame
fmts = dict.fromkeys(['start date', 'end date', 'Average Num Stocks'], '{:.0f}')
print("Subperiod Performance of Weekly Reversals")
row_formatted(DataFrame(out), formats=fmts, default='{:4f}')

```

Subperiod Performance of Weekly Reversals

	Full Period	Pre-break Period	Post-break Period	Period
start date	19740109	19740109	20010606	

(continues on next page)

(continued from previous page)

end date	20241218	20010530	20241218
Sharpe Ratio	1.9185	3.6668	0.7300
Average Gross Return	0.0021	0.0031	0.0010
Std Dev Returns	0.0080	0.0061	0.0096
Market Beta	0.0898	0.0686	0.1109
Jensen Alpha (annualized)	0.1030	0.1566	0.0407
Appraisal Ratio	1.8537	3.6607	0.6140
Information Coefficient	0.0376	0.0578	0.0140
Cross-sectional Vol	0.0536	0.0542	0.0529
Total Slippage Cost	0.0017	0.0028	0.0003
Spread Cost	0.0013	0.0023	0.0002
Delay Cost	0.0004	0.0005	0.0002
Exit Delay Cost	-0.0003	-0.0004	-0.0001
Enter Delay Cost	0.0006	0.0010	0.0002
Average Net Return	0.0005	0.0003	0.0006
Portfolio Turnover	0.7363	0.7393	0.7328
Average Num Stocks	1936	1981	1884

The structural break test identified a statistically significant change in the strategy's weekly returns around mid-2001. Comparing performance before and after this point reveals a clear decline in the annualized Sharpe ratio and average weekly returns, along with an increase in risk. This shift roughly coincides with the adoption of decimalization by the New York and American Stock Exchanges on January 29, 2001, which resulted in tighter bid-ask spreads. Prior to this change, U.S. markets quoted prices in fractions, with one-sixteenth (1/16) of a dollar being the smallest allowable price increment.

R usage notes:

```
/usr/bin/ld: cannot find -lgfortran
collect2: error: ld returned 1 exit status
```

- check versions are the same: gfortran --version and gcc --version
- select versions to be the same: sudo update-alternatives --config gcc

References:

- Andrews, D.W.K., "Tests for parameter instability and structural change with unknown change point", *Econometrica*, 61 (1993), 821-856.
- Grinold, Richard C., 1994, "Alpha is Volatility Times IC Times Score", *The Journal of Portfolio Management*, Summer 1994, 20(4), 9-16
- Lo, Andrew W. and MacKinlay, A. Craig, 1990, "When Are Contrarian Profits Due to Stock Market Overreaction?", *The Review of Financial Studies*, 3(2), 175–205
- Perold, Andre, 1988, "The Implementation Shortfall: Paper versus Reality", *The Journal of Portfolio Management*, 14(3), 4-9

QUANT FACTORS

Quants do it with models - Anonymous

Factor investing is a systematic approach to asset pricing and portfolio management based on the premise that various risk factors drive asset returns. Factor-based investing recognizes that besides broad market exposure, additional systematic risks—such as value, momentum, and volatility—play a crucial role in determining returns. This framework has evolved over time, beginning with early models like the Capital Asset Pricing Model (CAPM) and expanding to multifactor models, behavioral theories, and adaptive market perspectives. This analysis explores the empirical performance of different factor strategies, and the methodologies used to evaluate them. We also examine historical backtests and employ clustering techniques to group similar investment strategies, seeking to identify style factors and construct effective benchmarks.

```
from pandas import DataFrame, Series
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from sklearn.cluster import AgglomerativeClustering, KMeans
from scipy.cluster.hierarchy import dendrogram
from sklearn.metrics import silhouette_samples, silhouette_score
from tqdm import tqdm
import warnings
from datetime import datetime
from typing import List, Tuple
from finds.database import SQL, RedisDB
from finds.structured import (BusDay, Stocks, Benchmarks, Signals, SignalsFrame,
                               CRSP, PSTAT, IBES, CRSPBuffer)
from finds.backtesting import BackTest, univariate_sorts
from secret import credentials, paths, CRSP_DATE
VERBOSE = 0
if not VERBOSE:
    warnings.simplefilter(action='ignore', category=FutureWarning)

#%matplotlib qt
```

```
LAST_DATE = CRSP_DATE
# open connections
imgdir = paths['images']
sql = SQL(**credentials['sql'], verbose=VERBOSE)
user = SQL(**credentials['user'], verbose=VERBOSE)
rdb = RedisDB(**credentials['redis'])
bd = BusDay(sql, verbose=VERBOSE)
crsp = CRSP(sql, bd, rdb=rdb, verbose=VERBOSE)
pstat = PSTAT(sql, bd, verbose=VERBOSE)
```

(continues on next page)

(continued from previous page)

```
bench = Benchmarks(sql, bd, verbose=VERBOSE)
signals = Signals(user, verbose=VERBOSE)
ibes = IBES(sql, bd, verbose=VERBOSE)
backtest = BackTest(user, bench, 'RF', LAST_DATE, verbose=VERBOSE)
outdir = paths['scratch'] / 'output'
```

6.1 Factor investing

Factor investing posits that asset returns are driven by exposure to specific risk factors, which determine their risk premiums. The market itself is an investable factor, as described by the Capital Asset Pricing Model (CAPM), which asserts that market exposure is the sole driver of asset returns. However, additional factors such as interest rates, value-growth investing, low volatility strategies, and momentum portfolios have been widely recognized. Macroeconomic factors, including inflation and economic growth, also influence asset returns. Assets exhibit varying degrees of exposure to these risk factors, with greater exposure leading to higher risk premiums. Essentially, assets can be viewed as bundles of different factor exposures.

Early multifactor models include Stephen Ross's (1976) Arbitrage Pricing Theory (APT), which argues that risk factors cannot be arbitrated or diversified away, and Robert Merton's Intertemporal Capital Asset Pricing Model (ICAPM), which accounts for investors hedging risky positions over multiple time periods. Additionally, behavioral finance theories suggest that factor premiums arise due to investor biases, such as overreaction, underreaction, and bounded rationality.

6.1.1 Adaptive Markets Hypothesis

Andrew Lo's (2004) *Adaptive Markets Hypothesis* proposes that financial markets are shaped by principles of evolutionary biology rather than fixed physical laws. This perspective suggests that investment performance fluctuates as the financial ecosystem and market conditions evolve. Lo advocates studying financial markets by analyzing different “species” of investors—individuals and institutions that share common traits—and tracking their size, growth, interactions, and behavioral tendencies.

6.1.2 Factor Zoo

John Cochrane (2011) coined the term *Factor Zoo* to highlight the rapid proliferation of newly identified factors in academic research. In response, Green, Hand, and Zhang (2017) systematically examined nearly 100 firm characteristic factors, addressing issues such as microcap stock overweighting and data snooping biases. Their study assessed the predictive power of these factors across different time periods.

```
# preload monthly stocks data
monthly = CRSPBuffer(stocks=crsp, dataset='monthly', fields=['ret', 'retx', 'prc'],
                      beg=19251201, end=CRSP_DATE)
```

```
# signals to flip signs when forming spread portfolios
leverage = {'mom1m':-1, 'mom36m':-1, 'pricedelay':-1, 'absacc':-1, 'acc':-1,
            'agr':-1, 'chcsho':-1, 'egr':-1, 'mve_ia':-1, 'pctacc':-1,
            'aeavol':-1, 'disp':-1, 'stdacc':-1, 'stdcf':-1, 'secured':-1,
            'maxret':-1, 'ill':-1, 'zerotrade':-1, 'cashpr':-1, 'chinv':-1,
            'invest':-1, 'cinvest':-1, 'idiovol':-1, 'retvol':-1}
```

Helper functions

```
# to lag yearly characteristics
def as_lags(df, var, key, nlags):
    """Return dataframe with {nlags} of column {var}, same {key} value in row"""
    out = df[[var]].rename(columns={var: 0})           # first col: not shifted
    for i in range(1, nlags):
        prev = df[[key, var]].shift(i, fill_value=0) # next col: shifted i+1
        prev.loc[prev[key] != df[key], :] = np.nan    # require same {key} value
        out.insert(i, i, prev[var])
    return out
```

```
# rolling window of returns
def as_rolling(df, other, width=0, dropna=True):
    """join next dataframe to a sliding window with fixed number of columns"""
    df = df.join(other, how='outer', sort=True, rsuffix='r')
    if width and len(df.columns) > width:           # if wider than width
        df = df.iloc[:, (len(df.columns)-width):]      # then drop first cols
    if dropna:                                         # drop empty rows
        df = df[df.count(axis=1) > 0]
    df.columns = list(range(len(df.columns)))
    return df
```

```
# pipeline to run backtest
def backtest_pipeline(backtest: BackTest,
                      stocks: Stocks,
                      holdings: DataFrame,
                      label: str,
                      benchnames: List[str],
                      suffix: str = '',
                      overlap: int = 0,
                      outdir: str = '',
                      num: int = None) -> DataFrame:
    """wrapper to run a backtest pipeline
```

Args:

backtest: To compute backtest results
stocks: Where securities returns can be retrieved from (e.g. CRSP)
holdings: dict (key int date) of Series holdings (key permno)
label: Label of signal to backtest
benchnames: Names of benchmarks to attribute portfolio performance
overlap: Number of overlapping holdings to smooth
num: Figure num to plot to

Returns:

DataFrame of performance returns in rows

Notes:

graph and summary statistics are output to jpg and (appended) html
backtest object updated with performance and attribution data

"""

```
summary = backtest(stocks, holdings, label, overlap=overlap)
excess = backtest.fit(benchnames)
backtest.write(label)
backtest.plot(num=num, label=label + suffix)
if VERBOSE:
    print(pd.Series(backtest.annualized, name=label + suffix) \
          .to_frame().T.round(3).to_string())
```

(continues on next page)

(continued from previous page)

```

if outdir:
    # performance metrics from backtest to output
    sub = ['alpha', 'excess', 'appraisal', 'sharpe', 'welch-t', 'welch-p']
    with open(outdir / 'index.html', 'at') as f:
        f.write(f"<p><hr><h2>{label + suffix}</h2>\n<pre>\n")
        f.write(f"{{}-{} }\n".format(min(backtest.excess.index),
                                    max(backtest.excess.index),
                                    benchnames))
        f.write(f":12s ".format("Annualized"))
        f.write(f"\n".join(f" {k:10s}" for k in sub) + "\n")
        f.write(f":12s ".format(label + ":"))
        f.write(f"\n".join(f" {backtest.annualized[k]:10.4f}" for k in sub))
        f.write(f"\n</pre><p>{datetime.now()}\n")
    return summary

```

6.1.3 Past prices

Momentum and dividend yield data are sourced from CRSP monthly records

```

beg, end = 19251231, LAST_DATE
intervals = {'mom12m': (2, 12),
             'mom36m': (13, 36),
             'mom6m': (2, 6),
             'mom1m': (1, 1)}
for label, past in tqdm(intervals.items(), total=len(intervals)):
    out = []
    rebaldates = bd.date_range(bd.endmo(beg, past[1]), end, 'endmo')
    for rebaldate in rebaldates:
        start = bd.endmo(rebaldate, -past[1])
        beg1 = bd.offset(start, 1)
        end1 = bd.endmo(rebaldate, 1-past[0])
        df = crsp.get_universe(end1)
        # require data available as of start month and end month (universe)
        df['start'] = monthly.get_section(dataset='monthly',
                                           fields=['ret'],
                                           date_field='date',
                                           date=start).reindex(df.index)
        df[label] = monthly.get_ret(beg1, end1).reindex(df.index)
        df['permno'] = df.index
        df['rebaldate'] = rebaldate
        df = df.dropna(subset=['start'])
        out.append(df[['rebaldate', 'permno', label]]) # append rows
    out = pd.concat(out, axis=0, ignore_index=True)
    n = signals.write(out, label, overwrite=True)

beg, end = 19270101, LAST_DATE
columns = ['chmom', 'divyld', 'indmom']
out = []
for rebaldate in bd.date_range(beg, end, 'endmo'):
    start = bd.endmo(rebaldate, -12)
    beg1 = bd.offset(start, 1)
    end1 = bd.endmo(rebaldate, -6)
    beg2 = bd.offset(end1, 1)
    end2 = bd.endmo(rebaldate)

```

(continues on next page)

(continued from previous page)

```

df = crsp.get_universe(end1)
df['start'] = monthly.get_section(dataset='monthly',
                                    fields=['ret'],
                                    date_field='date',
                                    date=start).reindex(df.index)
df['end2'] = monthly.get_section(dataset='monthly',
                                    fields=['ret'],
                                    date_field='date',
                                    date=end2).reindex(df.index)
df['mom2'] = monthly.get_ret(beg2, end2).reindex(df.index)
df['mom1'] = monthly.get_ret(beg1, end1).reindex(df.index)
df['divyld'] = crsp.get_divamt(beg1, end2) \
    .reindex(df.index) ['divamt'] \
    .div(df['cap']) \
    .fillna(0)
df['chmom'] = df['mom1'] - df['mom2']

# 6-month two-digit sic industry momentum (group means of 'mom1')
df['sic2'] = df['siccd'] // 100
df = df.join(DataFrame(df.groupby(['sic2']) ['mom1'].mean()) \
    .rename(columns={'mom1': 'indmom'}),
    on='sic2', how='left')
df['permno'] = df.index
df['rebald'] = rebald
out.append(df.dropna(subset=['start', 'end2']) \
    [['rebald', 'permno'] + columns])
out = pd.concat(out, axis=0, ignore_index=True)
for label in columns: # save signal values to sql
    n = signals.write(out, label, overwrite=True)

```

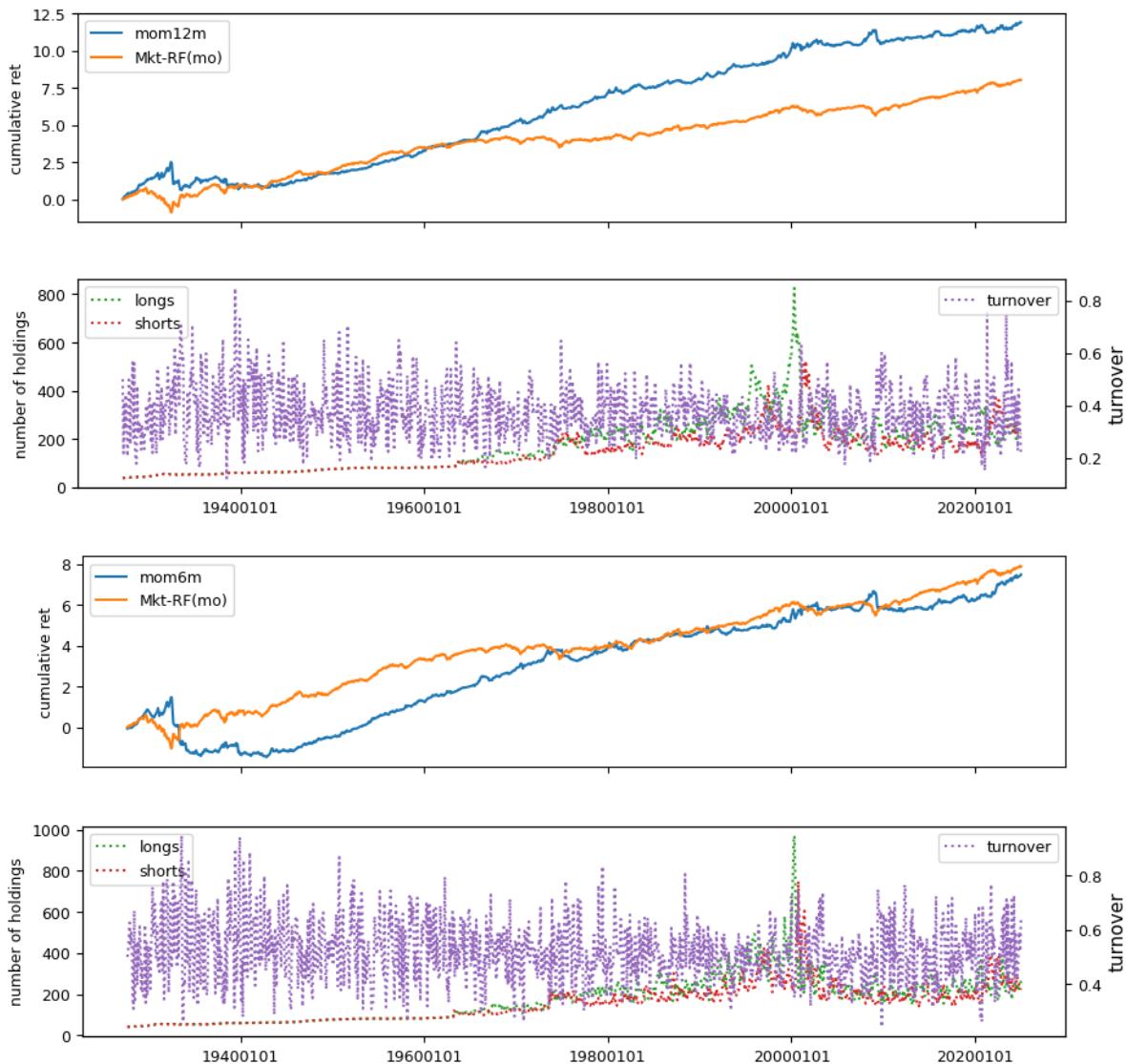
100% [██████████] 4/4 [47:03<00:00, 705.93s/it]

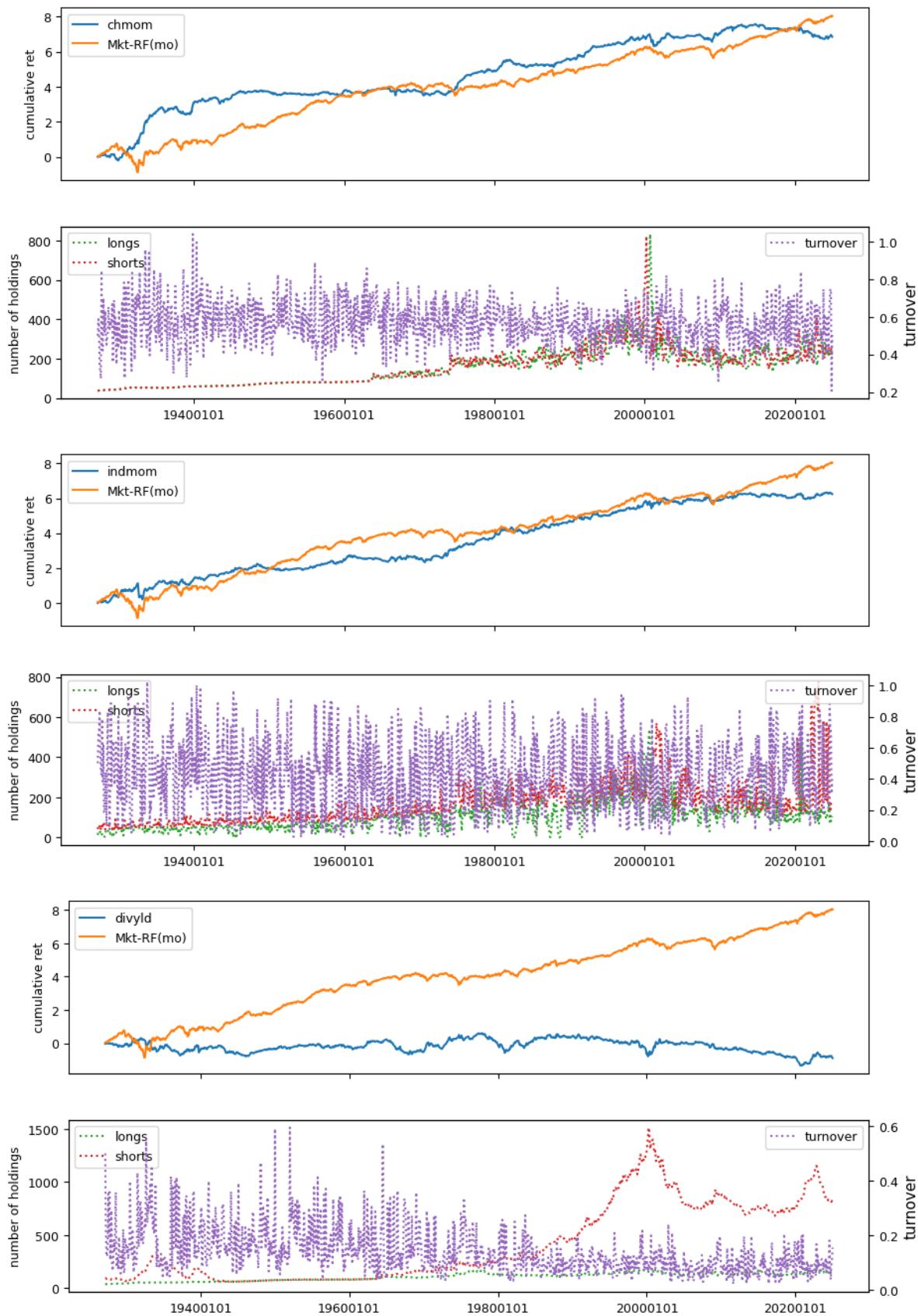
```

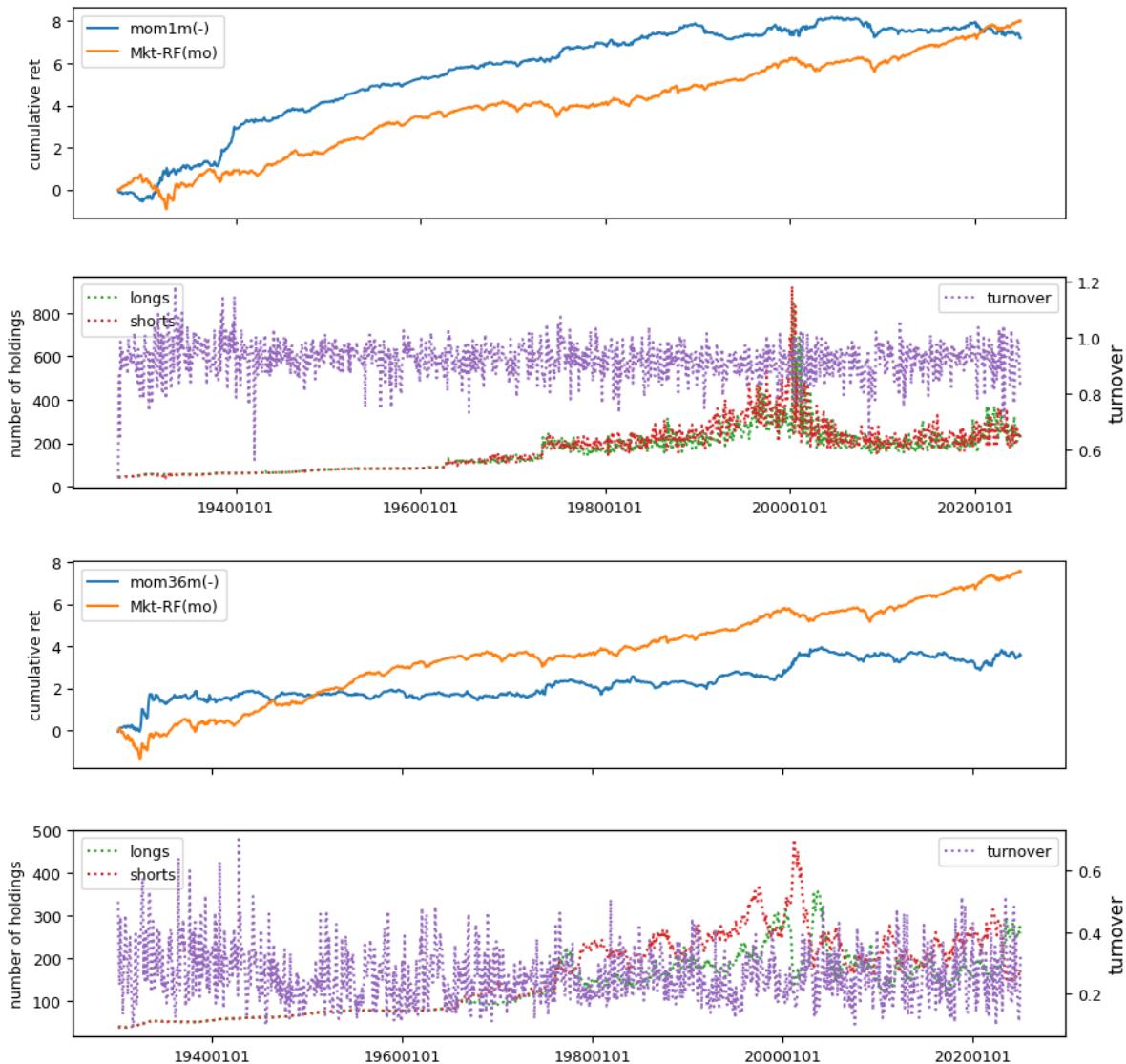
benchnames = ['Mkt-RF(mo)']
rebald, rebald = 19260101, LAST_DATE
columns = ['mom12m', 'mom6m', 'chmom', 'indmom', 'divyld', 'mom1m', 'mom36m']
for label in tqdm(columns):
    holdings = univariate_sorts(monthly,
                                 label,
                                 SignalsFrame(signals.read(label)),
                                 rebald,
                                 rebald,
                                 window=1,
                                 months=[],
                                 maxdecile=8,
                                 minprc=1.0,
                                 pct=(10.0, 90.0),
                                 leverage=leverage.get(label, 1))
    excess = backtest_pipeline(backtest,
                               monthly,
                               holdings,
                               label,
                               benchnames,
                               overlap=0,
                               outdir=outdir,
                               suffix=(leverage.get(label, 1) < 0) * '(-)')

```

100% |██████████| 7/7 [22:01<00:00, 188.74s/it]







```
# helper to calculate beta, idiovol and price delay from weekly returns
def regress(x: np.array, y: np.array) -> Tuple[float, float, float]:
    """helper method to calculate beta, idiovol and price delay

    Args:
        x: equal-weighted market returns (in ascending time order)
        y: stock returns (in ascending time order). NaN's will be discarded.

    Returns:
        beta: slope from regression on market returns and intercept
        idiovol: mean squared error of residuals
        pricedelay: increase of adjusted Rsq with four market lags over without
    """
    v = np.logical_not(np.isnan(y))
    y = y[v]
    x = x[v]
    n0 = len(y)
    A0 = np.vstack([x, np.ones(len(y))]).T
```

(continues on next page)

(continued from previous page)

```

b0 = np.linalg.inv(A0.T.dot(A0)).dot(A0.T.dot(y))    # univariate coeffs
sse0 = np.mean((y - A0.dot(b0))**2)
sst0 = np.mean((y - np.mean(y))**2)
if (sst0>0 and sse0>0):
    R0 = (1 - ((sse0 / (n0 - 2)) / (sst0 / (n0 - 1))))
else:
    R0 = 0
y4 = y[4:]
n4 = len(y4)
A4 = np.vstack([x[0:-4], x[1:-3], x[2:-2], x[3:-1], x[4:],
               np.ones(n4)]).T
b4 = np.linalg.inv(A4.T.dot(A4)).dot(A4.T.dot(y4))    # four lagged coeffs
sse4 = np.mean((y4 - A4.dot(b4))**2)
sst4 = np.mean((y4 - np.mean(y4))**2)
if sst4 > 0 and sse4 > 0:
    R4 = (1 - ((sse4 / (n4 - 6)) / (sst4 / (n4 - 1))))
else:
    R4 = 0
return [b0[0],
        sse0 or np.nan,
        (1 - (R0 / R4)) if R0>0 and R4>0 else np.nan]

```

Weekly price responses derived from CRSP daily records.

```

beg, end = 19260101, LAST_DATE
columns = ['beta', 'idiovol', 'pricedelay']
wd = BusDay(sql, endweek='Wed')    # custom weekly trading day calendar

```

```

width      = 3*52+1           # up to 3 years of weekly returns
mininvalid = 52                # at least 52 weeks required to compute beta
weekly     = DataFrame()        # rolling window of weekly stock returns
mkt        = DataFrame()        # to queue equal-weighted market returns
out        = []                 # to accumulate final calculations

```

```

for date in tqdm(wd.date_range(beg, end, 'weekly')):
    df = crsp.get_ret(wd.begwk(date), date)
    mkt = as_rolling(mkt,           # rolling window of weekly mkt returns
                     DataFrame(data=[df.mean()], columns=[date]),
                     width=width)
    weekly = as_rolling(weekly,    # rolling window of weekly stock returns
                        df.rename(date),
                        width=width)
    valid = weekly.count(axis=1) >= mininvalid    # require min number weeks
    if valid.any():
        result = DataFrame([regress(mkt.values[0], y)
                             for y in weekly.loc[valid].values],
                             columns=columns)
        result['permno'] = weekly.index[valid].values
        result['rebaldate'] = date
        if wd.ismonthend(date): # signal value from last week of month
            out.append(result)
out = pd.concat(out, axis=0, ignore_index=True)
for label in columns:
    signals.write(out, label, overwrite=True)

```

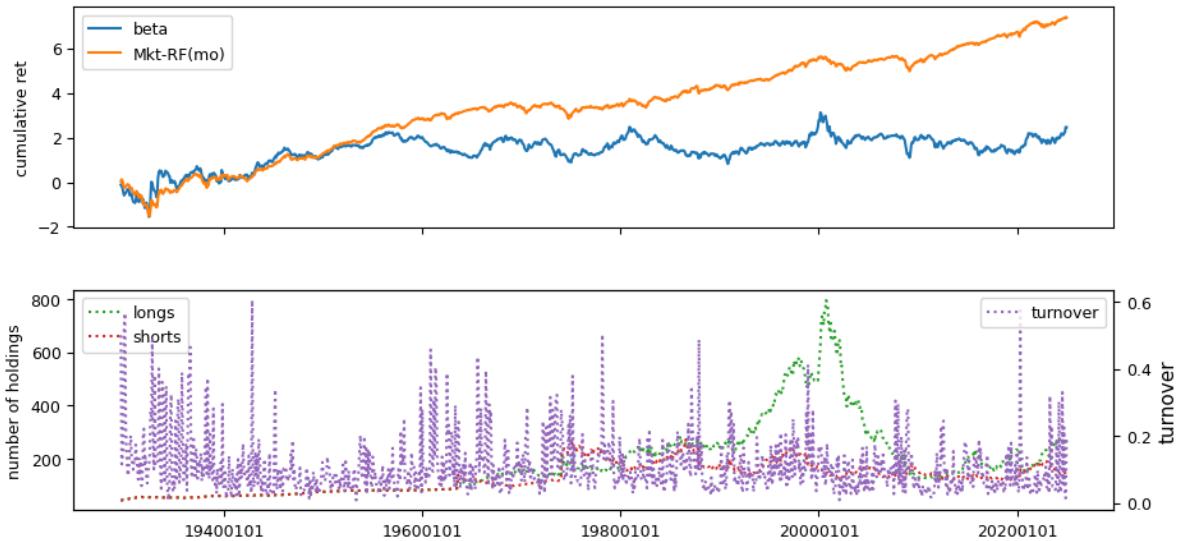
100% |██████████| 5165/5165 [31:49<00:00, 2.70it/s]

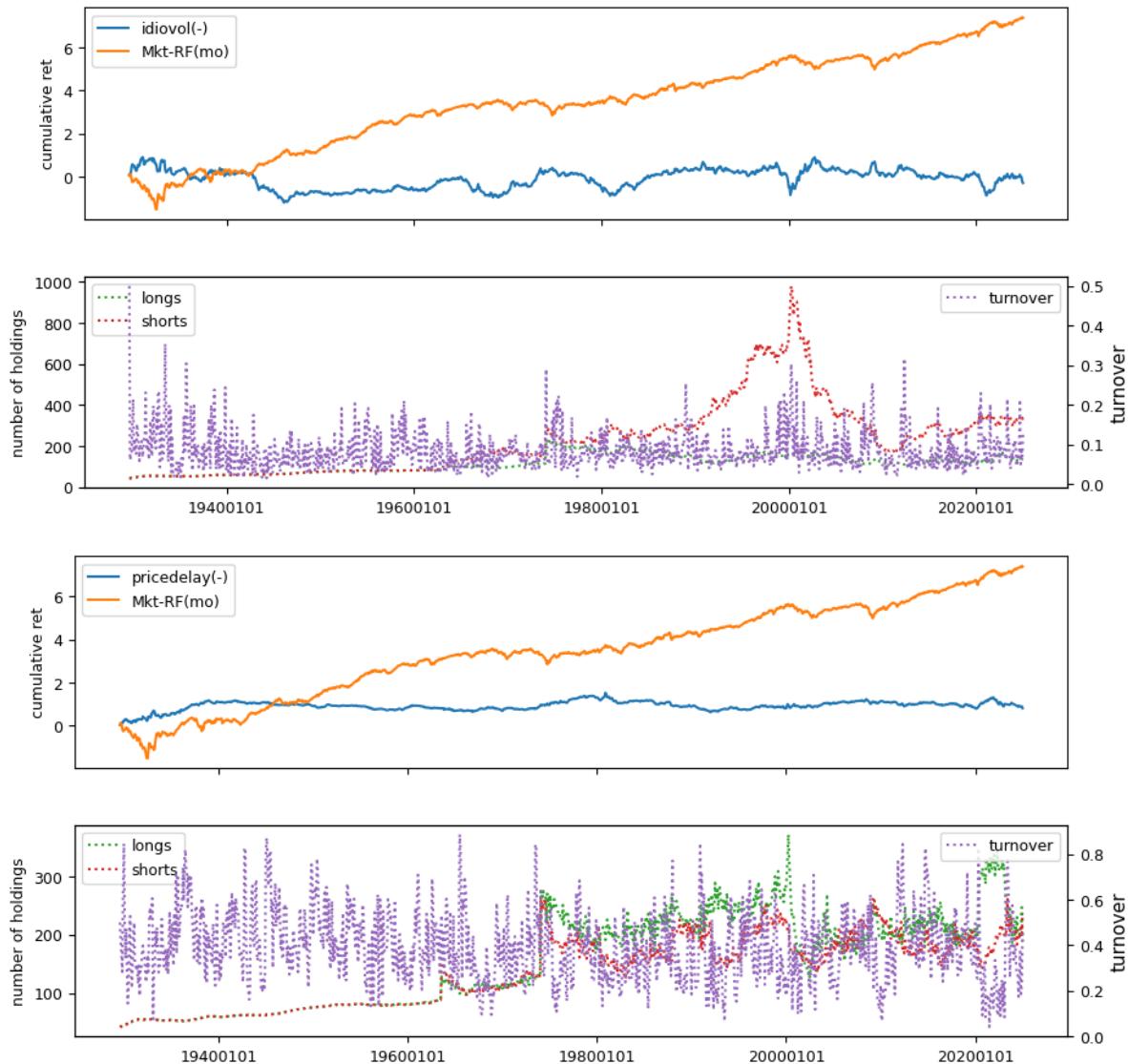
```

benchnames = ['Mkt-RF (mo)']
rebalbeg, rebalend = 19290601, LAST_DATE
for label in tqdm(columns):
    holdings = univariate_sorts(monthly,
                                 label,
                                 SignalsFrame(signals.read(label)),
                                 rebalbeg,
                                 rebalend,
                                 window=1,
                                 months=[],
                                 maxdecile=8,
                                 pct=(10., 90.),
                                 leverage=leverage.get(label, 1))
    excess = backtest_pipeline(backtest,
                               monthly,
                               holdings,
                               label,
                               benchnames,
                               overlap=0,
                               outdir=outdir,
                               suffix=(leverage.get(label, 1) < 0) * '(-)')

```

100% |██████████| 3/3 [07:48<00:00, 156.06s/it]





6.1.4 Liquidity

Liquidity signals are derived from daily stock return data.

```
beg, end = 19830601, LAST_DATE      # nasdaq/volume from after 1982
columns = ['ill', 'maxret', 'retvol', 'baspread', 'std_dolvol',
           'zerotrade', 'std_turn', 'turn']
```

```
out = []
dolvol = []
turn = DataFrame()      # to average turn signal over rolling 3-months
dt = bd.date_range(bd.begmo(beg,-3), end, 'endmo') # monthly rebalances
chunksize = 12          # each chunk is 12 months (1 year)
chunks = [dt[i:(i+chunksize)] for i in range(0, len(dt), chunksize)]
for chunk in tqdm(chunks):
    q = (f"SELECT permno, date, ret, askhi, bidlo, prc, vol, shrouut "
```

(continues on next page)

(continued from previous page)

```

f" FROM {crsp['daily'].key}"
f" WHERE date>={bd.begmo(chunk[0])}"
f" AND date<={chunk[-1]}")      # retrieve a chunk
f = crsp.sql.read_dataframe(q).sort_values(['permno', 'date'])
f['baspread'] = ((f['askhi'] - f['bidlo']) / ((f['askhi'] + f['bidlo']) / 2))
f['dolvol'] = f['prc'].abs() * f['vol']
f['turn1'] = f['vol'] / f['shroud']
f.loc[f['dolvol']>0, 'ldv'] = np.log(f.loc[f['dolvol']>0, 'dolvol'])
f['ill'] = 1000000 * f['ret'].abs() / f['dolvol']

for rebaldate in chunk:          # for each rebaldate in the chunk
    grouped = f[f['date'].ge(bd.begmo(rebaldate))
                & f['date'].le(rebaldate)].groupby('permno')
    df = grouped[['ret']].max().rename(columns={'ret': 'maxret'})
    df['retvol'] = grouped['ret'].std()
    df['baspread'] = grouped['baspread'].mean()
    df['std_dolvol'] = grouped['ldv'].std()
    df['ill'] = grouped['ill'].mean()
    dv = grouped['dolvol'].sum()
    df.loc[dv > 0, 'dolvol'] = np.log(dv[dv > 0])
    df['turn1'] = grouped['turn1'].sum()
    df['std_turn'] = grouped['turn1'].std()
    df['countzero'] = grouped['vol'].apply(lambda v: sum(v==0))
    df['ndays'] = grouped['prc'].count()

    turn = as_rolling(turn, df[['turn1']], width=3)
    df['turn'] = turn.reindex(df.index).mean(axis=1, skipna=False)
    df.loc[df['turn1'].le(0), 'turn1'] = 0
    df.loc[df['ndays'].le(0), 'ndays'] = 0
    df['zerotrade'] = ((df['countzero'] + ((1/df['turn1'])/480000))
                        * 21/df['ndays'])

    df['rebaldate'] = rebaldate
    df = df.reset_index()
    out.append(df[['permno', 'rebaldate']] + columns)
    if rebaldate < bd.endmo(end):
        df['rebaldate'] = bd.endmo(rebaldate, 1)
        dolvol.append(df[['permno', 'rebaldate', 'dolvol']])
out = pd.concat(out, axis=0, ignore_index=True)
dolvol = pd.concat(dolvol, axis=0, ignore_index=True)

```

100% |██████████| 42/42 [25:56<00:00, 37.05s/it]

```

for label in columns:
    n = signals.write(out, label, overwrite=True)
n = signals.write(dolvol, 'dolvol', overwrite=True)

```

```

rebalbeg, rebalend = 19830601, LAST_DATE
benchnames = ['Mkt-RF(mo)']
for label in tqdm(columns + ['dolvol']):
    holdings = univariate_sorts(monthly,
                                 label,
                                 SignalsFrame(signals.read(label)),
                                 rebalbeg,

```

(continues on next page)

(continued from previous page)

```

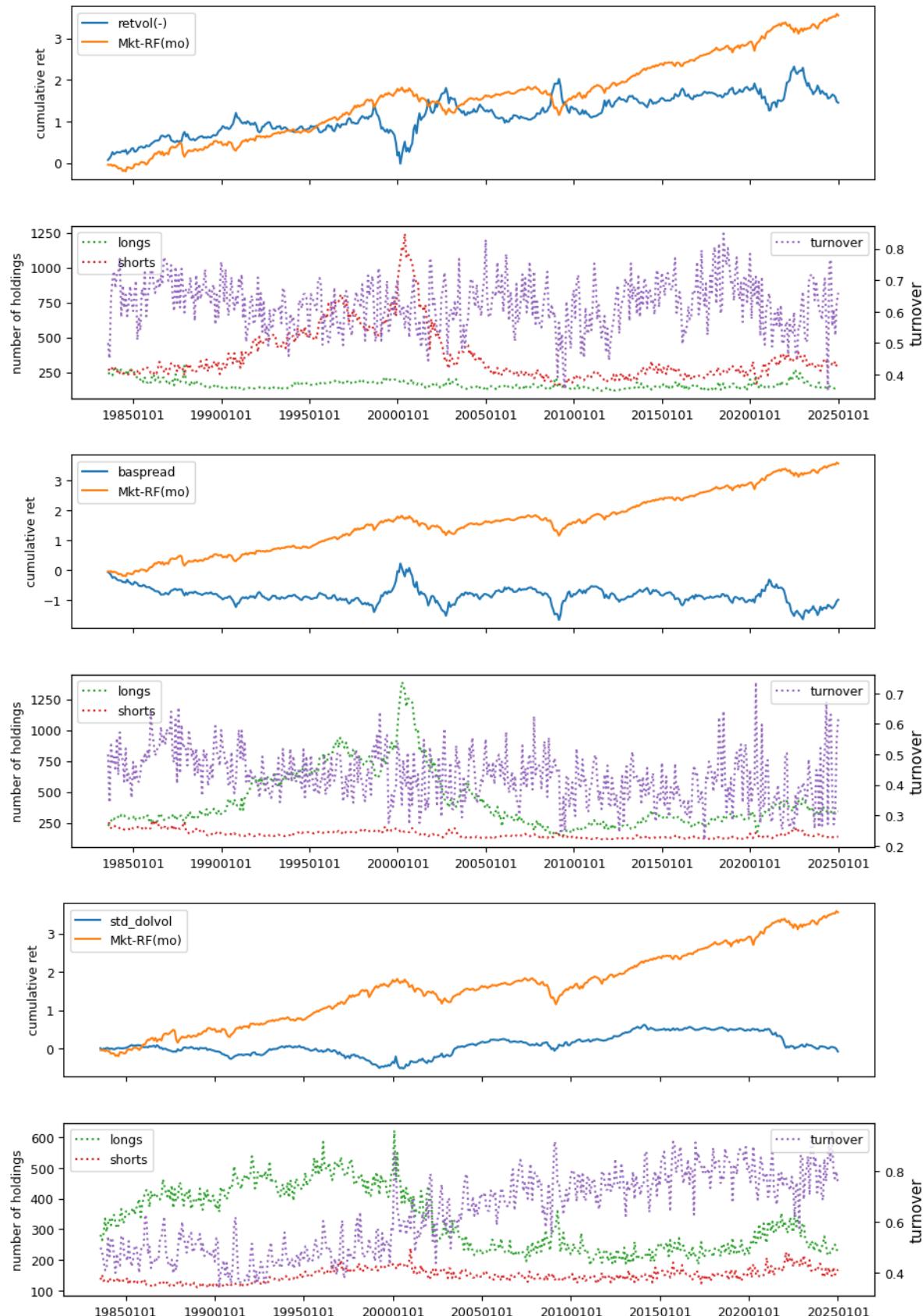
        rebalend,
        window=1,
        months=[],
        maxdecile=8,
        pct=(10., 90.),
        leverage=leverage.get(label, 1))

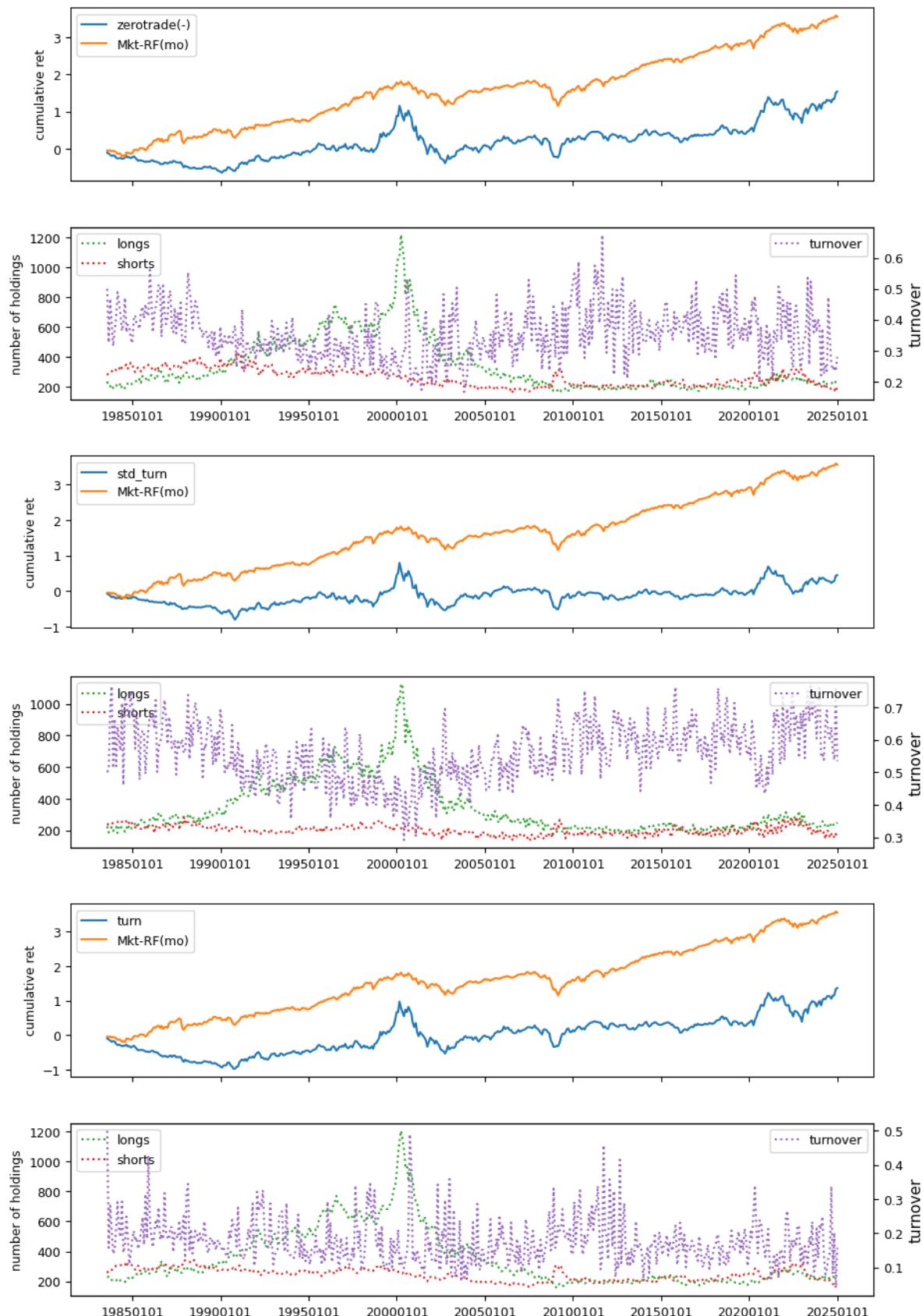
excess = backtest_pipeline(backtest,
                           monthly,
                           holdings,
                           label,
                           benchnames,
                           overlap=0,
                           outdir=outdir,
                           suffix=(leverage.get(label, 1) < 0) * '(-)')

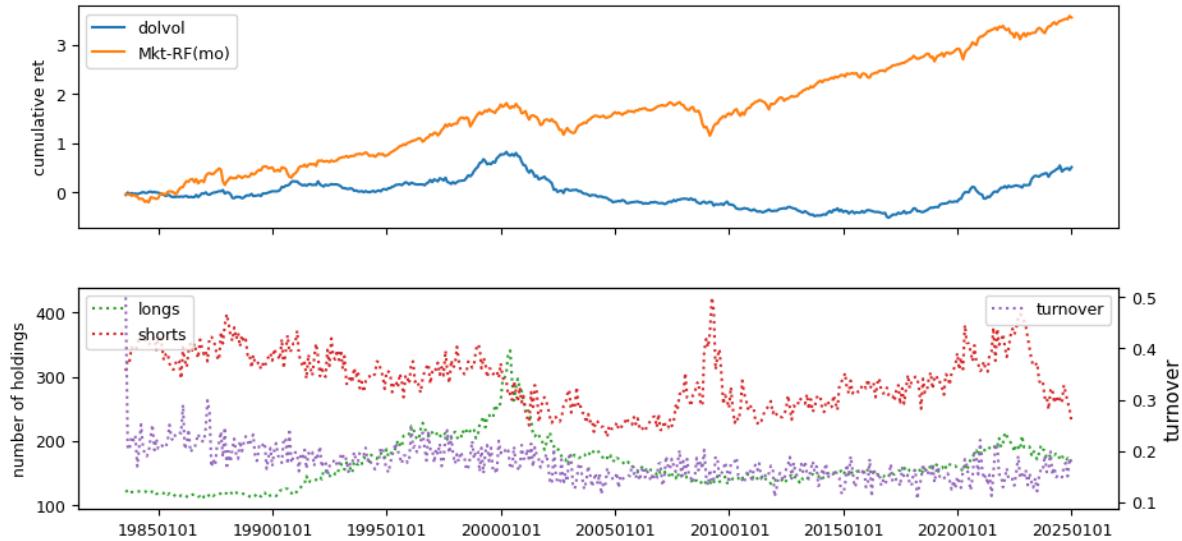
```

100% |██████████| 9/9 [09:46<00:00, 65.17s/it]









6.1.5 Fundamentals

Fundamental signals are collected from Compustat, using both annual and quarterly datasets.

```
columns = ['absacc', 'acc', 'agr', 'bm', 'cashpr', 'cfp', 'chcsho',
           'chinv', 'depr', 'dy', 'egr', 'ep', 'gma', 'grcapx',
           'grltnoa', 'hire', 'invest', 'lev', 'lgr',
           'pchdepr', 'pchgpm_pchsale', 'pchquick',
           'pchsale_pchinv', 'pchsale_pchrect', 'pchsale_pchxsga',
           'pchsaleinv', 'pctacc', 'quick', 'rd_sale', 'rd_mve',
           'realestate', 'salecash', 'salerec', 'saleinv', 'secured',
           'sgr', 'sp', 'tang', 'bm_ia', 'cfp_ia', 'chatoia', 'chpmia',
           'pchcapx_ia', 'chempia', 'mve_ia']
numlag = 6          # number of months to lag data for rebalance
end = LAST_DATE    # last data date
```

```
# retrieve annual, keep [permno, datadate] with non null prccq if any
fields = ['sic', 'fyear', 'ib', 'oancf', 'at', 'act', 'che', 'lct',
          'dlc', 'dltt', 'prcc_f', 'csho', 'invt', 'dp', 'ppent',
          'dvt', 'ceq', 'txp', 'revt', 'cogs', 'rect', 'aco', 'intan',
          'ao', 'ap', 'lco', 'lo', 'capx', 'emp', 'ppegt', 'lt',
          'sale', 'xsga', 'xrd', 'fatb', 'fatl', 'dm']
df = pstat.get_linked(dataset='annual',
                      fields=fields,
                      date_field='datadate',
                      where=(f"indfmt = 'INDL' "
                             f" AND datafmt = 'STD' "
                             f" AND curcd = 'USD' "
                             f" AND popsrc = 'D' "
                             f" AND consol = 'C' "
                             f" AND datadate <= (end//100)31"))
fund = df.sort_values(['permno', 'datadate', 'ib'])\
    .drop_duplicates(['permno', 'datadate'])\
    .dropna(subset=['ib'])
fund.index = list(zip(fund['permno'], fund['datadate'])) # multi-index
fund['rebaldate'] = bd.endmo(fund.datadate, numlag)
```

```
# precompute, and lag common metrics: mve_f avg_at sic2
fund['sic2'] = np.where(fund['sic'].notna(), fund['sic'] // 100, 0)
fund['fyear'] = fund['datadate'] // 10000      # can delete this
fund['mve_f'] = fund['prcc_f'] * fund['csho']
```

```
lag = fund.shift(1, fill_value=0)
lag.loc[lag['permno'] != fund['permno'], fields] = np.nan
fund['avg_at'] = (fund['at'] + lag['at']) / 2
```

```
lag2 = fund.shift(2, fill_value=0)
lag2.loc[lag2['permno'] != fund['permno'], fields] = np.nan
lag['avg_at'] = (lag['at'] + lag2['at']) / 2
```

```
fund['bm'] = fund['ceq'] / fund['mve_f']
fund['cashpr'] = (fund['mve_f'] + fund['dltt'] - fund['at'])/fund['che']
fund['depr'] = fund['dp'] / fund['ppent']
fund['dy'] = fund['dvt'] / fund['mve_f']
fund['ep'] = fund['ib'] / fund['mve_f']
fund['lev'] = fund['lt'] / fund['mve_f']
fund['quick'] = (fund['act'] - fund['invt']) / fund['lct']
fund['rd_sale'] = fund['xrd'] / fund['sale']
fund['rd_mve'] = fund['xrd'] / fund['mve_f']
fund['realestate'] = ((fund['fatb'] + fund['fatl']) /
                      np.where(fund['ppeg'] .notna(),
                               fund['ppeg'], fund['ppent']))
fund['salecash'] = fund['sale'] / fund['che']
fund['salerec'] = fund['sale'] / fund['rect']
fund['saleinv'] = fund['sale'] / fund['invt']
fund['secured'] = fund['dm'] / fund['dltt']
fund['sp'] = fund['sale'] / fund['mve_f']
fund['tang'] = (fund['che'] + fund['rect'] * 0.715 + fund['invt'] * 0.547
                + fund['ppent'] * 0.535) / fund['at']
```

```
# changes: agr chcsho chinv egr gma egr grcapx grltnoa emp invest lgr
fund['agr'] = (fund['at'] / lag['at'])
fund['chcsho'] = (fund['csho'] / lag['csho'])
fund['chinv'] = ((fund['invt'] - lag['invt']) / fund['avg_at'])
fund['egr'] = (fund['ceq'] / lag['ceq'])
fund['gma'] = ((fund['revt'] - fund['cogs']) / lag['at'])
fund['grcapx'] = (fund['capx'] / lag2['capx'])
fund['grltnoa'] = (((fund['rect'] +
                      + fund['invt'] +
                      + fund['ppent'] +
                      + fund['aco'] +
                      + fund['intan'] +
                      + fund['ao'] -
                      - fund['ap'] -
                      - fund['lco'] -
                      - fund['lo']) /
                      (lag['rect'] +
                      + lag['invt'] +
                      + lag['ppent'] +
                      + lag['aco'] +
                      + lag['intan'])
```

(continues on next page)

(continued from previous page)

```

+ lag['ao']
- lag['ap']
- lag['lco']
- lag['lo']))
- ((fund['rect']
+ fund['invt']
+ fund['aco']
- fund['ap']
- fund['lco']))
- (lag['rect']
+ lag['invt']
+ lag['aco']
- lag['ap']
- lag['lco']))) / fund['avg_at']
fund['hire'] = ((fund['emp'] / lag['emp']) - 1).fillna(0)
fund['invest'] = (((fund['ppegt'] - lag['ppegt'])
+ (fund['invt'] - lag['invt'])) / lag['at'])
fund['invest'] = fund['invest'].where(fund['invest'].notna(),
((fund['ppent'] - lag['ppent'])
+ (fund['invt'] - lag['invt'])) / lag['at']))
fund['lgr'] = (fund['lt'] / lag['lt'])
fund['pchdepr'] = ((fund['dp'] / fund['ppent']) / (lag['dp'] / lag['ppent']))
fund['pchgm_pchsale'] = (((fund['sale'] - fund['cogs']) / (lag['sale'] - lag['cogs']))
- (fund['sale'] / lag['sale']))
fund['pchquick'] = (((fund['act'] - fund['invt']) / fund['lct'])
/ ((lag['act'] - lag['invt']) / lag['lct']))
fund['pchsaled_pchinv'] = ((fund['sale'] / lag['sale']) - (fund['invt'] / lag['invt'
-']))
fund['pchsaled_pchrect'] = ((fund['sale'] / lag['sale']) - (fund['rect'] / lag['rect'
-']))
fund['pchsaled_pchxsga'] = ((fund['sale'] / lag['sale']) - (fund['xsga'] / lag['xsga'
-']))
fund['pchsaledinv'] = ((fund['sale'] / fund['invt']) / (lag['sale'] / lag['invt']))
fund['sgr'] = (fund['sale'] / lag['sale'])

```

```

fund['chato'] = ((fund['sale'] / fund['avg_at']) - (lag['sale'] / lag['avg_at']))
fund['chpm'] = (fund['ib'] / fund['sale']) - (lag['ib'] / lag['sale'])
fund['pchcapx'] = fund['capx'] / lag['capx']

```

```

# compute signals with alternative definitions: acc absacc cfp
fund['_acc'] = (((fund['act'] - lag['act']) - (fund['che'] - lag['che']))
- ((fund['lct'] - lag['lct']) - (fund['dlc'] - lag['dlc']))
- (fund['txp'] - lag['txp']) - fund['dp']))
fund['cfp'] = ((fund['ib'] - (((fund['act'] - lag['act']) - (fund['che'] - lag['che'
-'])))
- ((fund['lct'] - lag['lct'])
- (fund['dlc'] - lag['dlc'])
- (fund['txp'] - lag['txp'])
- fund['dp']))) / fund['mve_f'])
g = fund['oancf'].notnull()
fund.loc[g, 'cfp'] = fund.loc[g, 'oancf'] / fund.loc[g, 'mve_f']
fund.loc[g, '_acc'] = fund.loc[g, 'ib'] - fund.loc[g, 'oancf']
fund['acc'] = fund['_acc'] / fund['avg_at']
fund['absacc'] = abs(fund['_acc']) / fund['avg_at']
fund['pctacc'] = fund['_acc'] / abs(fund['ib'])

```

(continues on next page)

(continued from previous page)

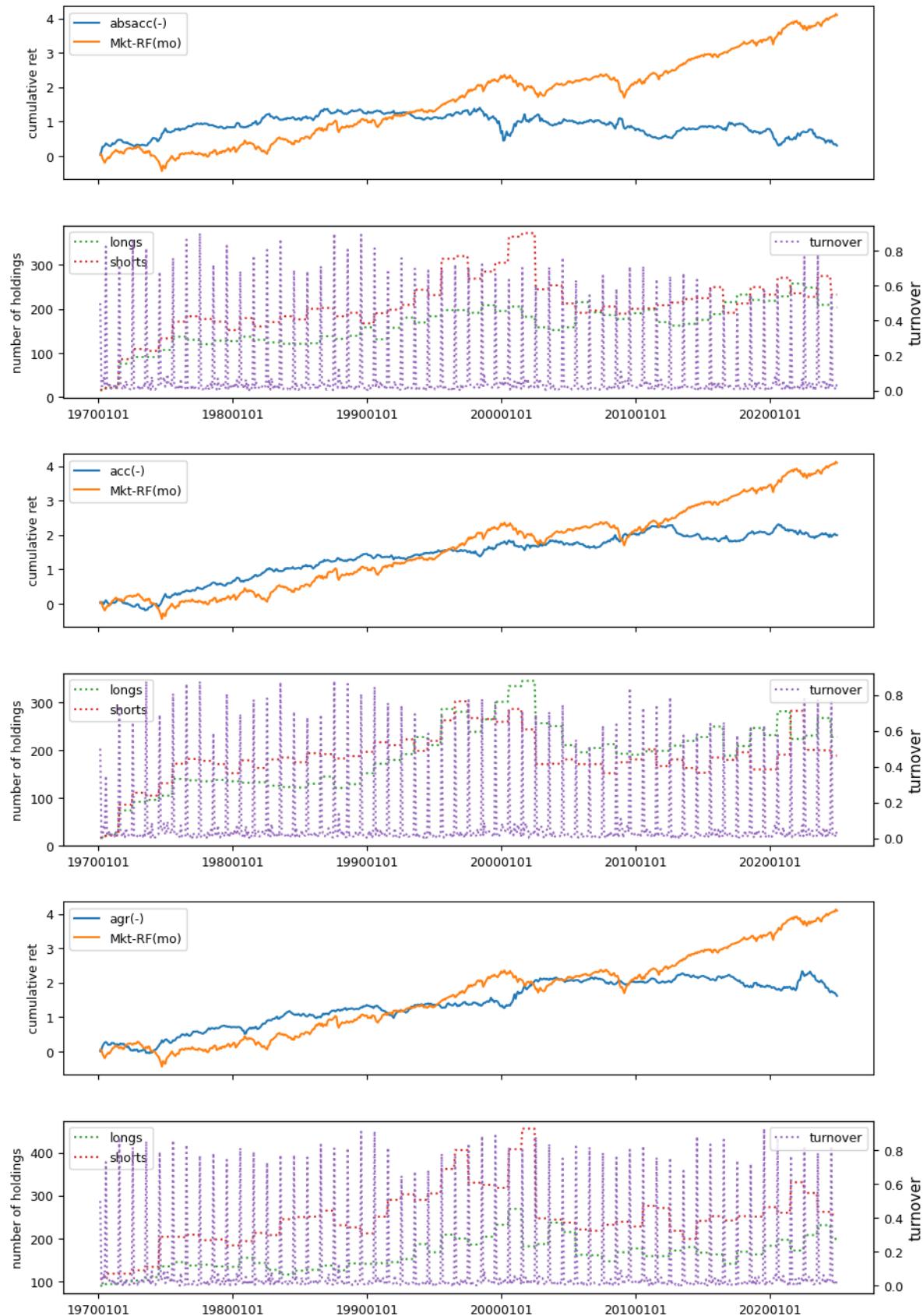
```
h = (fund['ib'].abs() <= 0.01)
fund.loc[h, 'pctacc'] = fund.loc[h, '_acc'] / 0.01
```

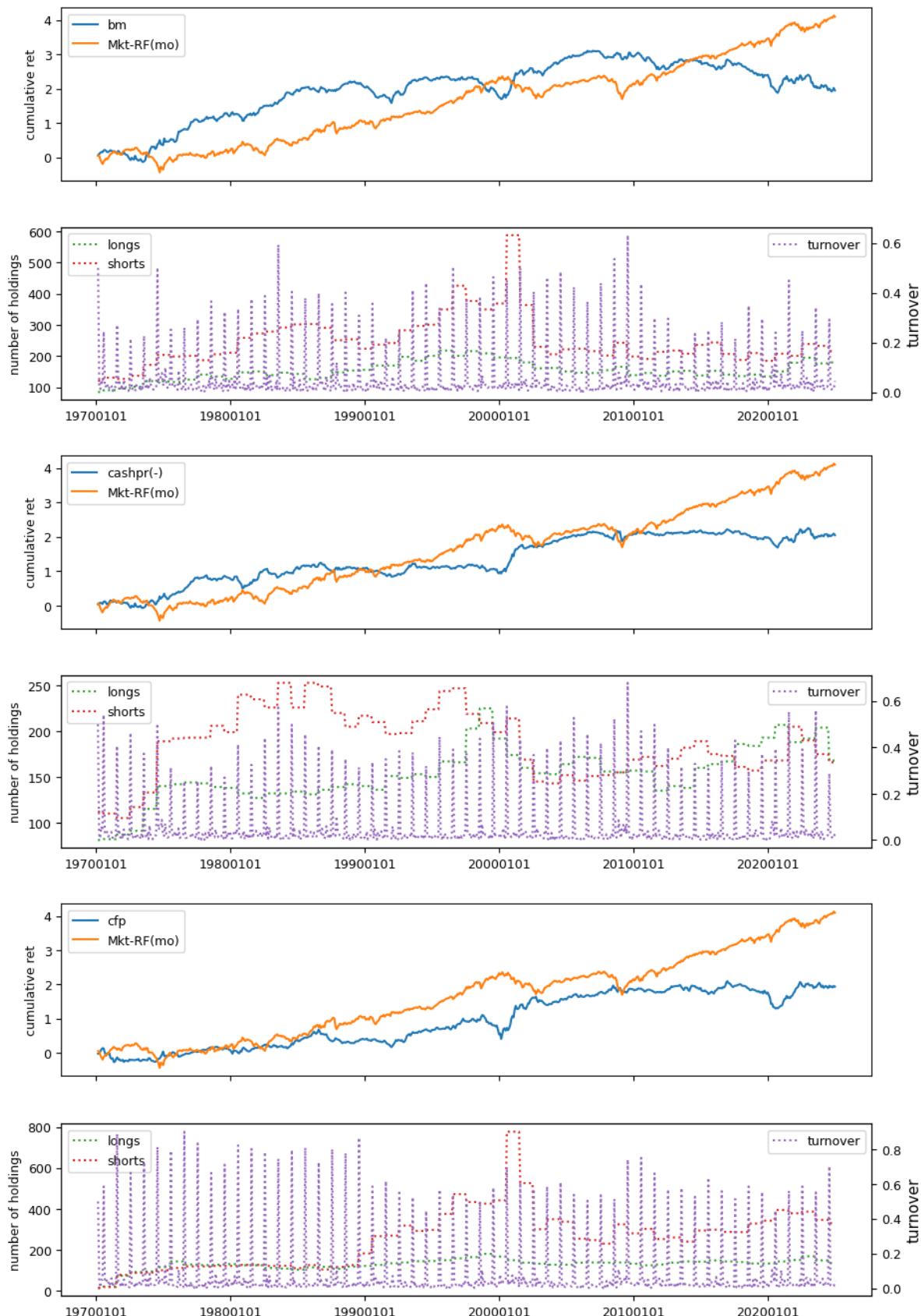
```
# industry-adjusted
cols = {'bm_ia': 'bm', 'cfp_ia': 'cfp', 'chatoia': 'chato',
        'chpmia': 'chpm', 'pchcapx_ia': 'pchcapx',
        'chempia': 'hire', 'mve_ia': 'mve_f'}
group = fund.groupby(['sic2', 'fyear'])
for k,v in cols.items():
    fund[k] = fund[v] - group[v].transform('mean')
```

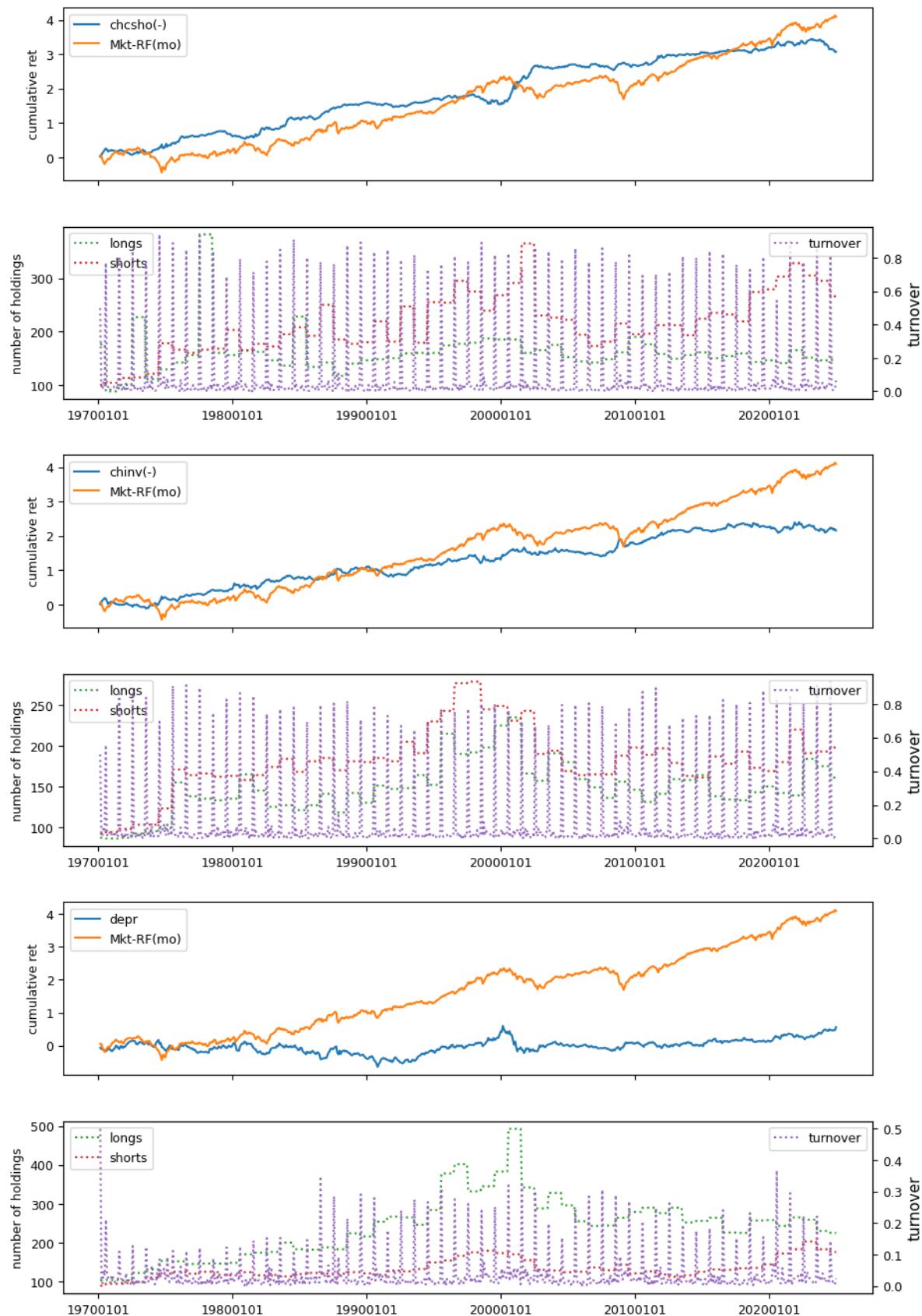
```
for label in columns:
    signals.write(fund, label, overwrite=True)
```

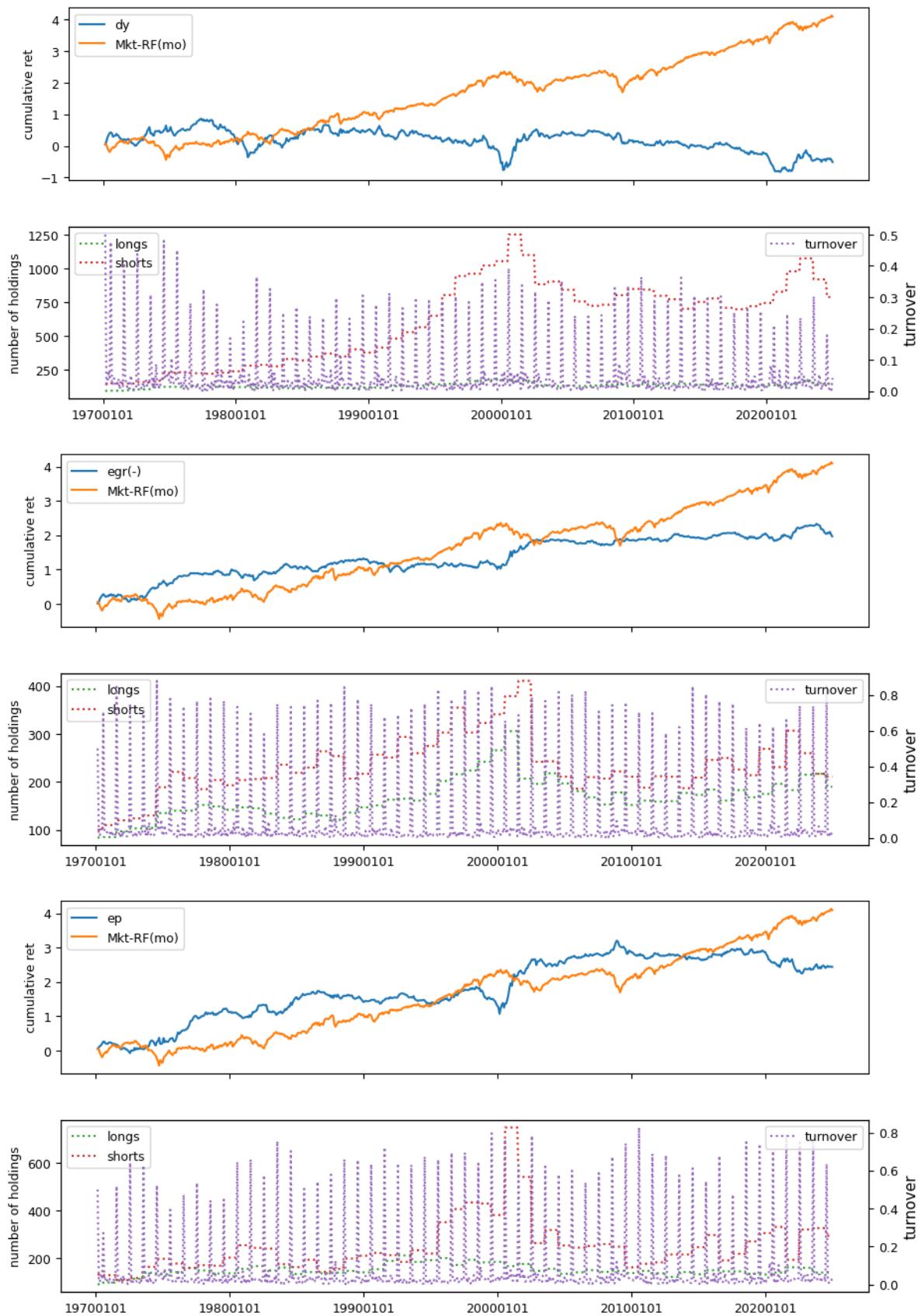
```
rebalbeg, rebalend = 19700101, LAST_DATE
benchnames = ['Mkt-RF(mo)'] #['Mom'] #['ST_Rev(mo)'] #
for label in tqdm(columns):
    holdings = univariate_sorts(monthly,
                                  label,
                                  SignalsFrame(signals.read(label)),
                                  rebalbeg,
                                  rebalend,
                                  window=12,
                                  months=[6],
                                  maxdecile=8,
                                  pct=(10., 90.),
                                  leverage=leverage.get(label, 1))
    excess = backtest_pipeline(backtest,
                               monthly,
                               holdings,
                               label,
                               benchnames,
                               overlap=0,
                               outdir=outdir,
                               suffix=(leverage.get(label, 1) < 0) * '(-)')
```

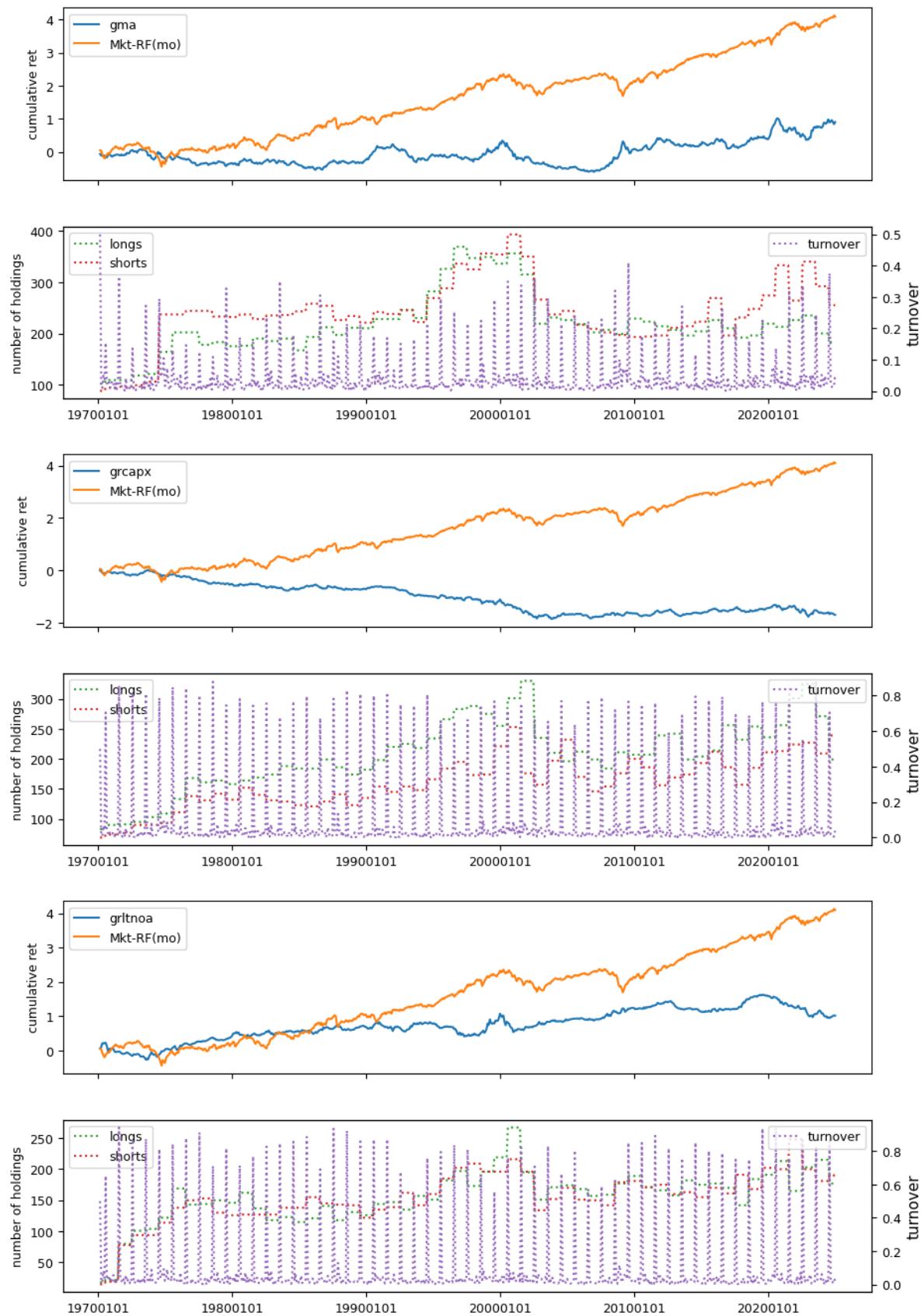
```
44%|██████████| 20/45 [29:40<39:56, 95.88s/it] /home/terence/Dropbox/github/data-
  ↵science-notebooks/finds/backtesting/backtest.py:310: RuntimeWarning: More than
  ↵20 figures have been opened. Figures created through the pyplot interface
  ↵(`matplotlib.pyplot.figure`) are retained until explicitly closed and may
  ↵consume too much memory. (To control this warning, see the rcParam `figure.max_
  ↵open_warning`). Consider using `matplotlib.pyplot.close()` .
    fig, (ax1, ax2) = plt.subplots(2, 1, sharex=True, clear=True,
100%|██████████| 45/45 [1:09:08<00:00, 92.19s/it]
```

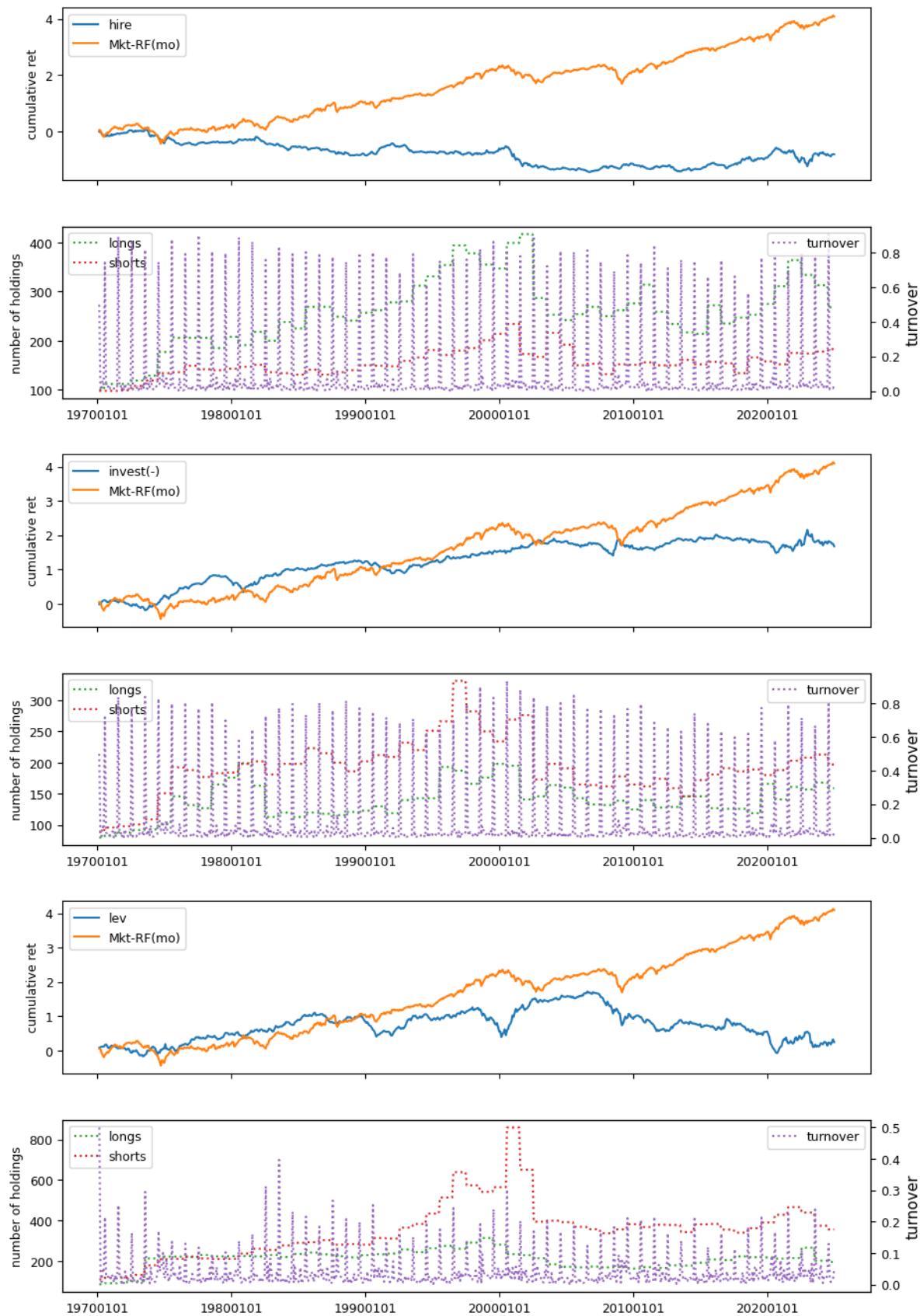


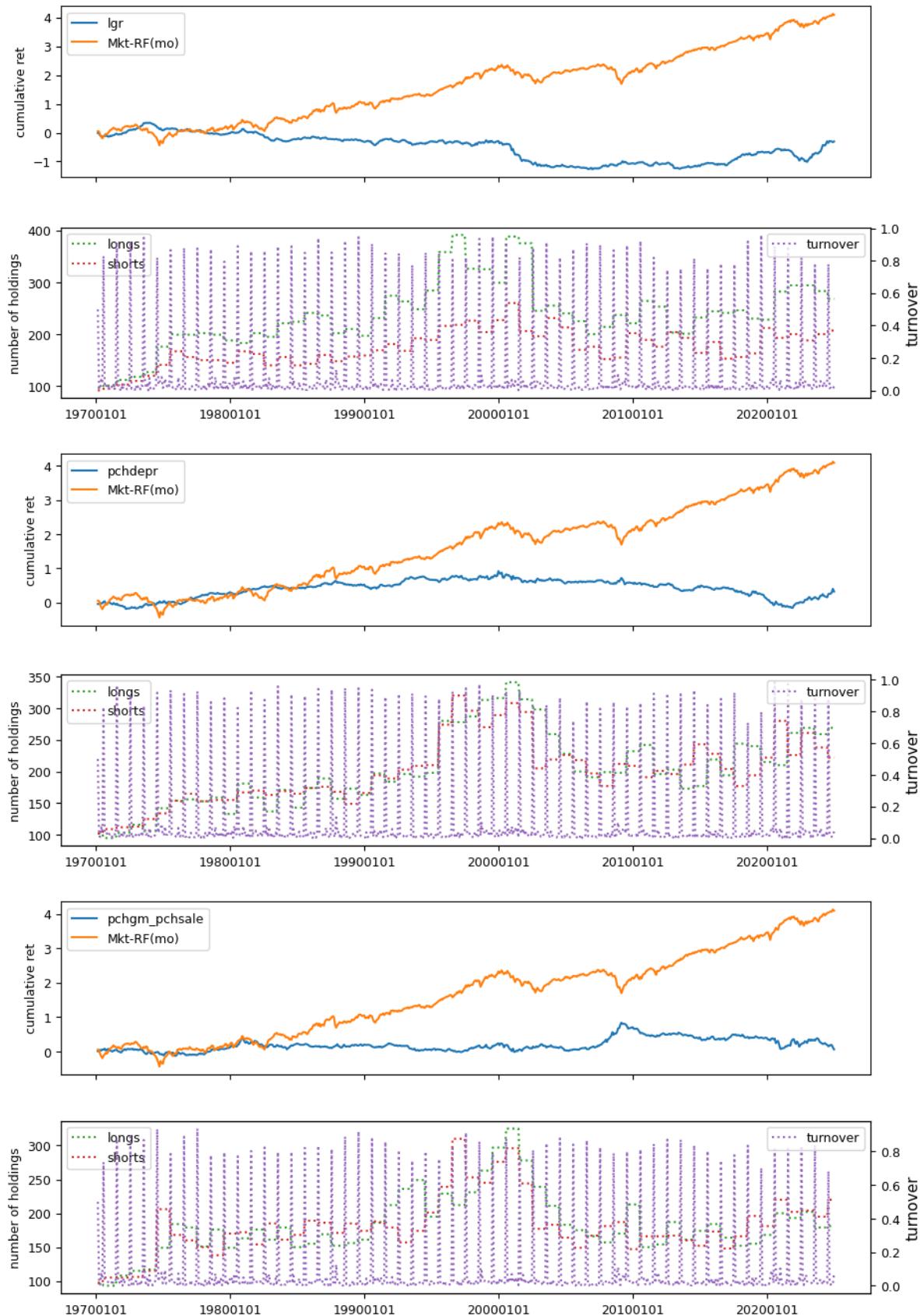


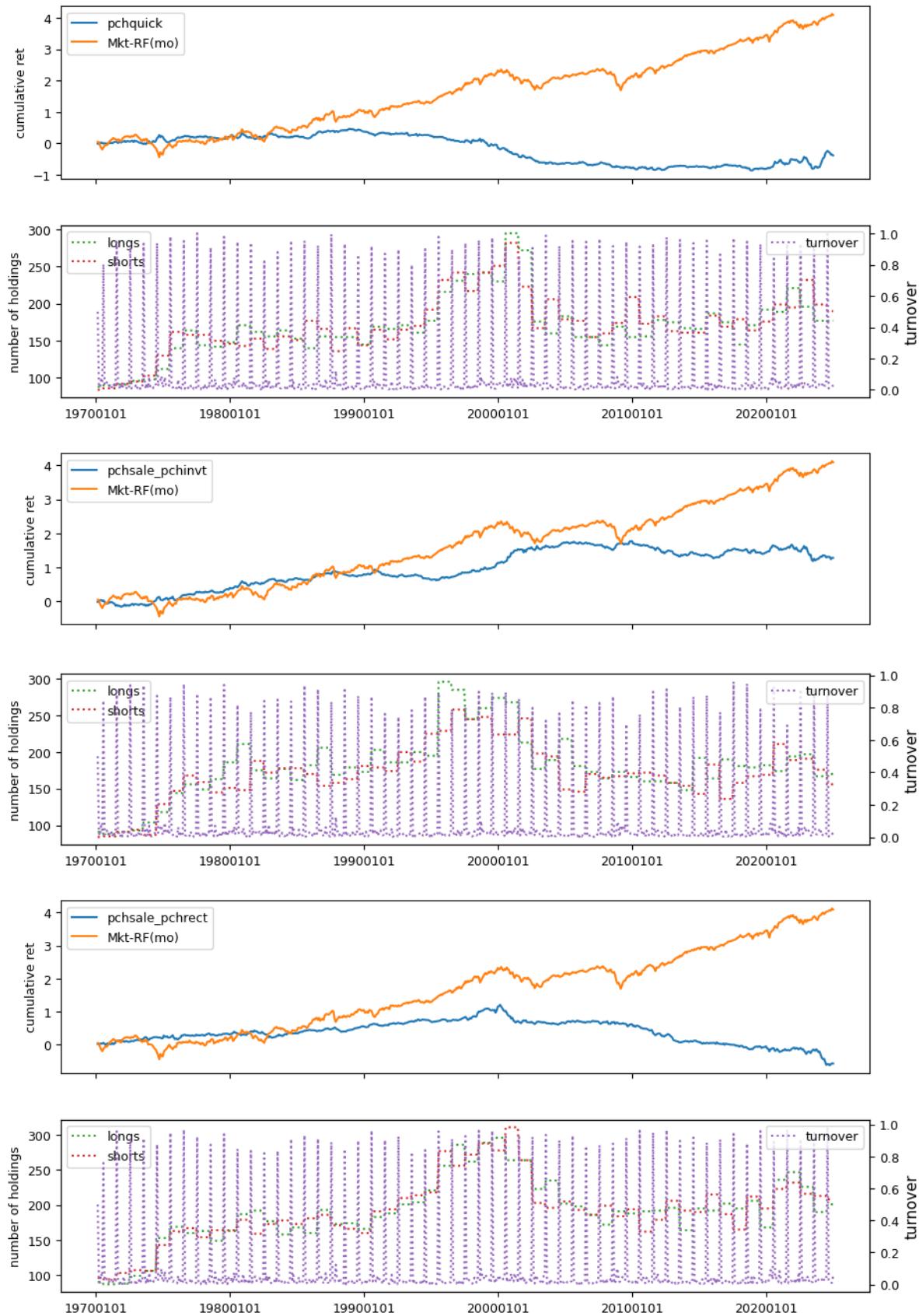


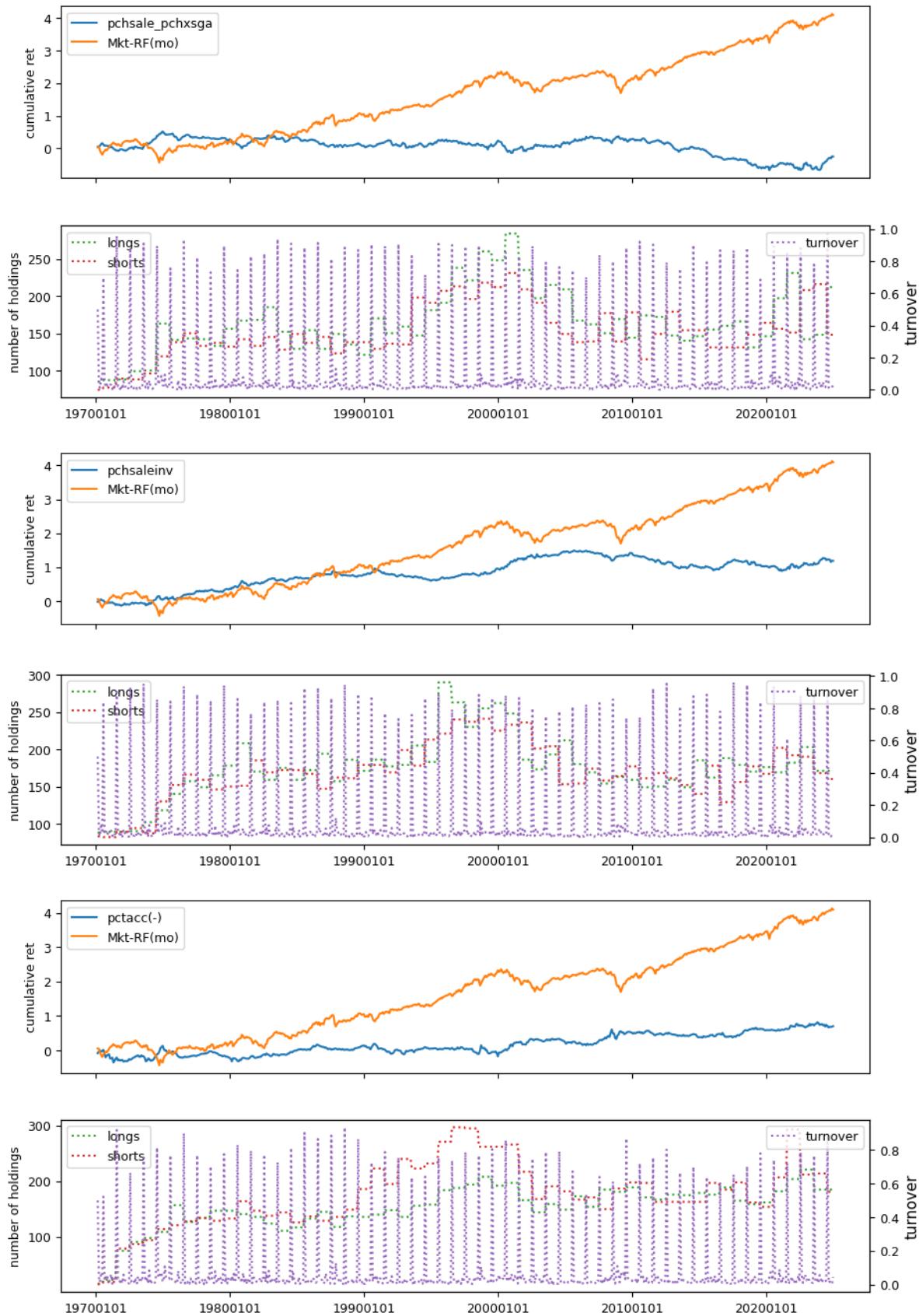


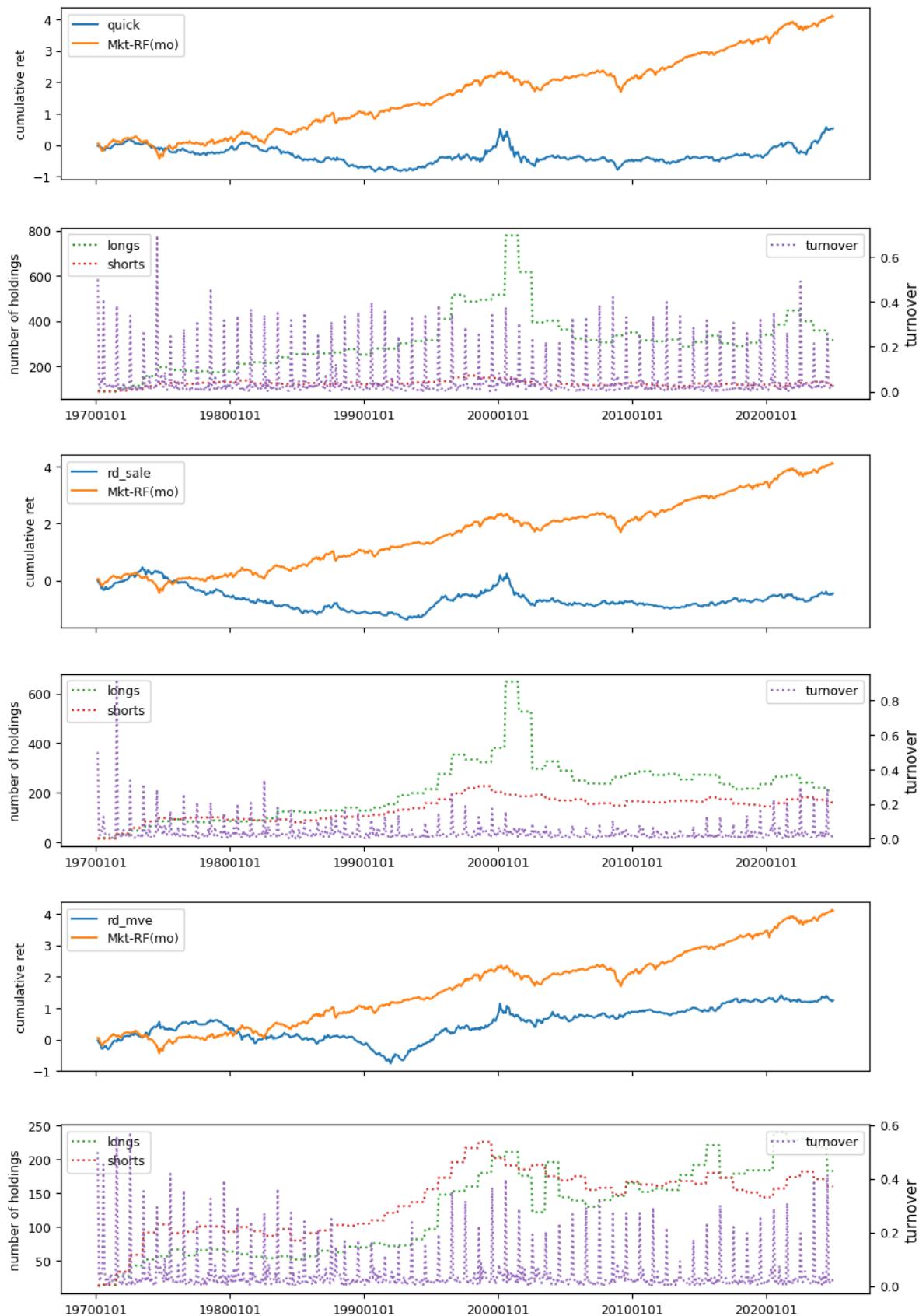


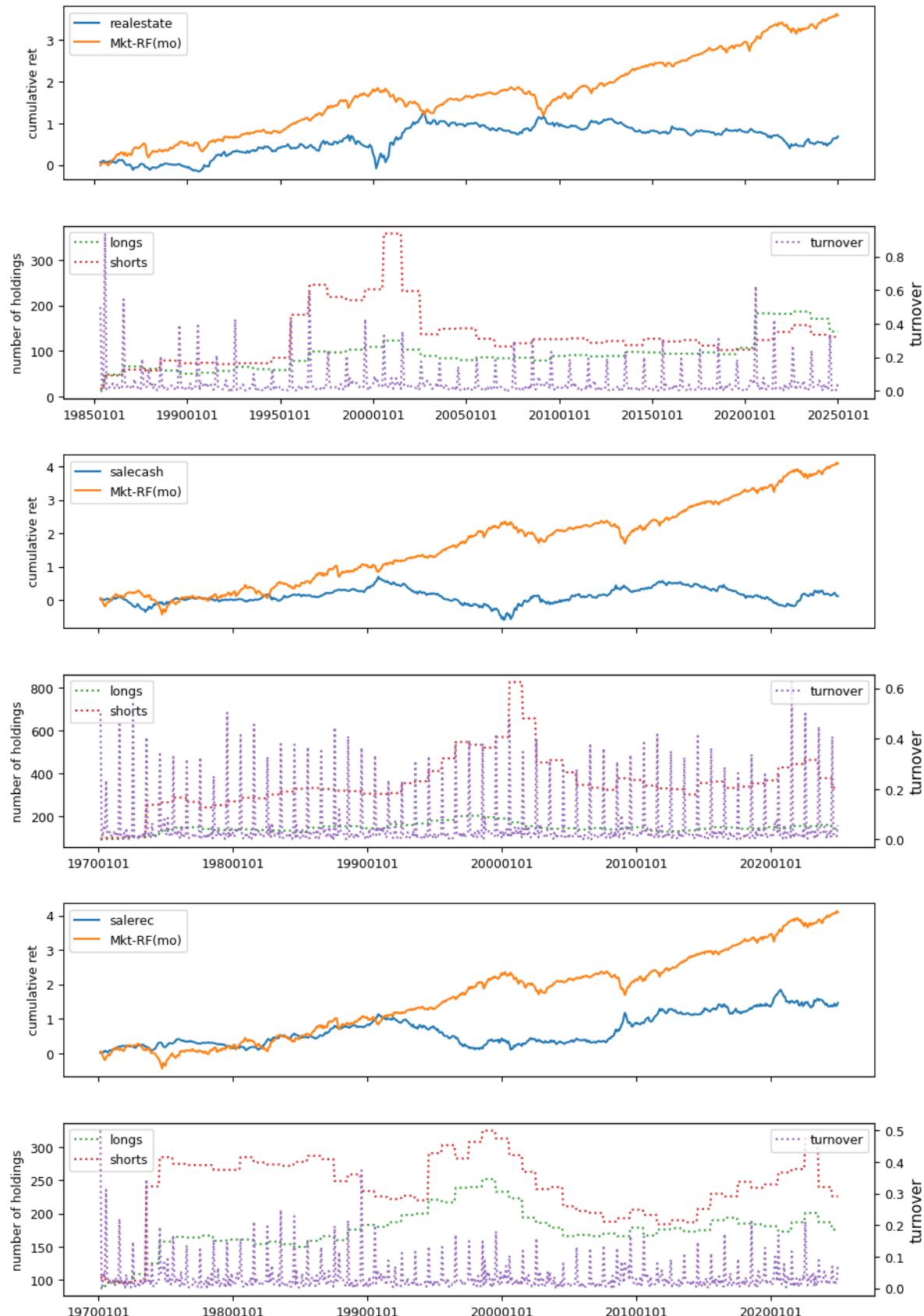


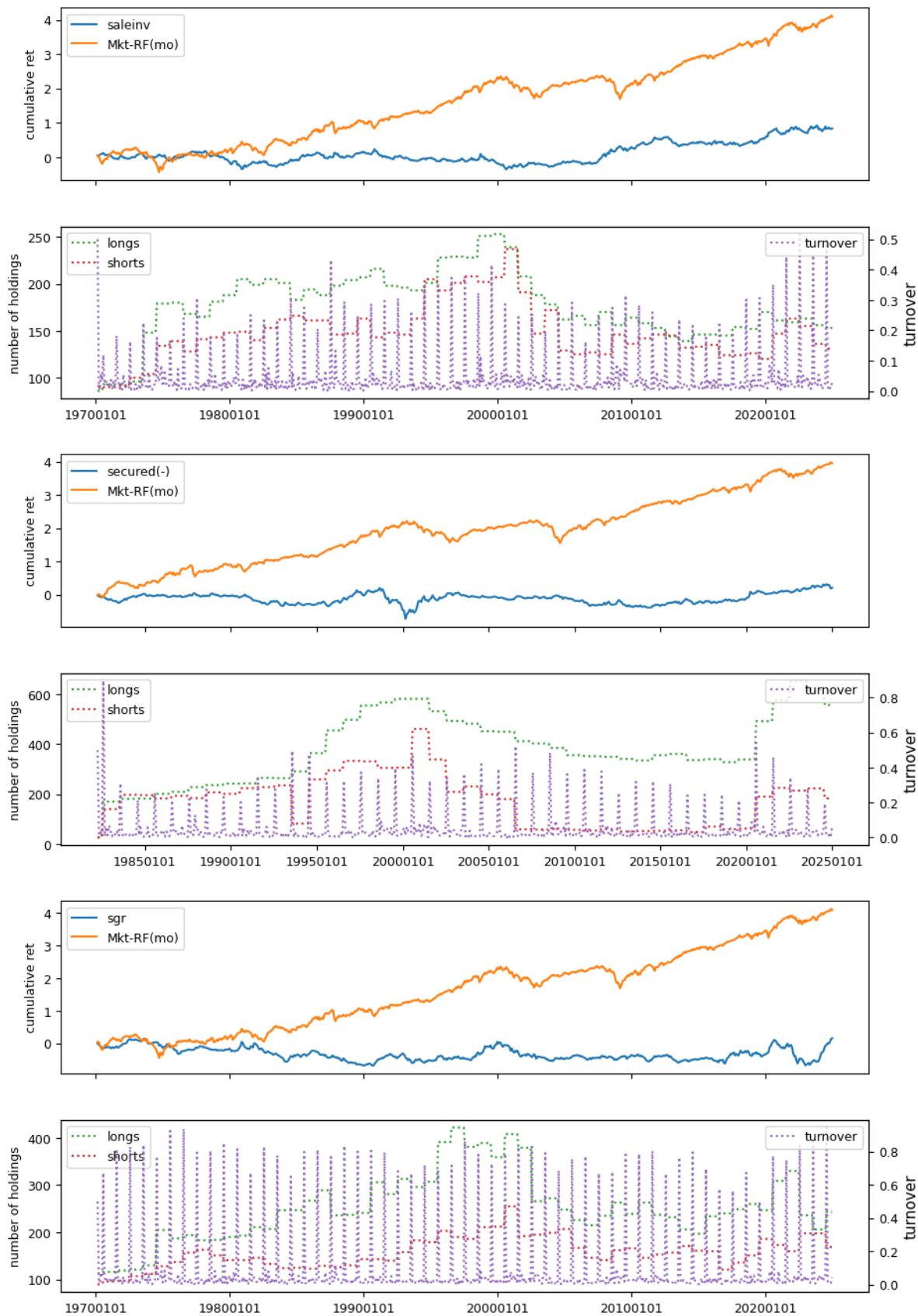




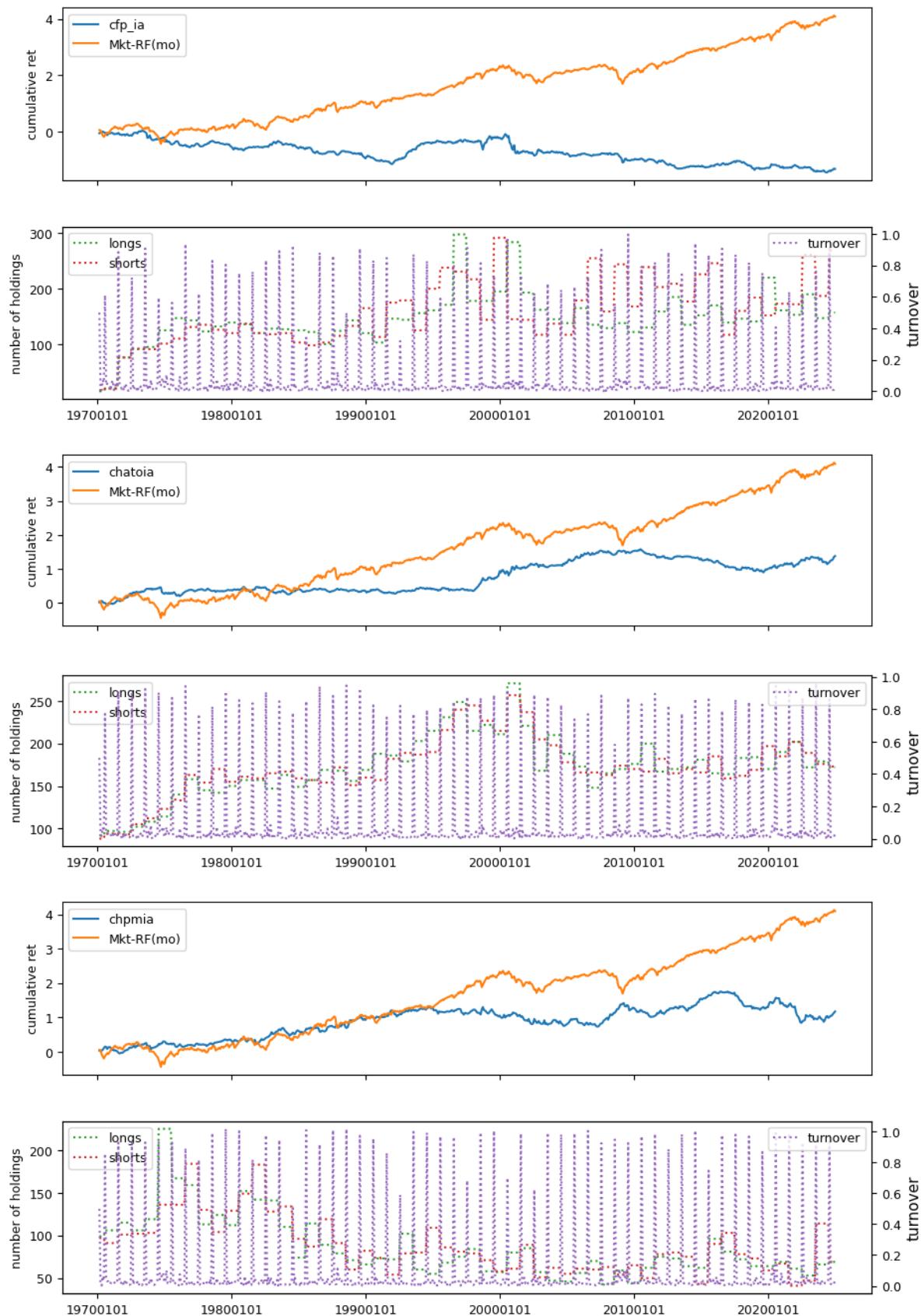


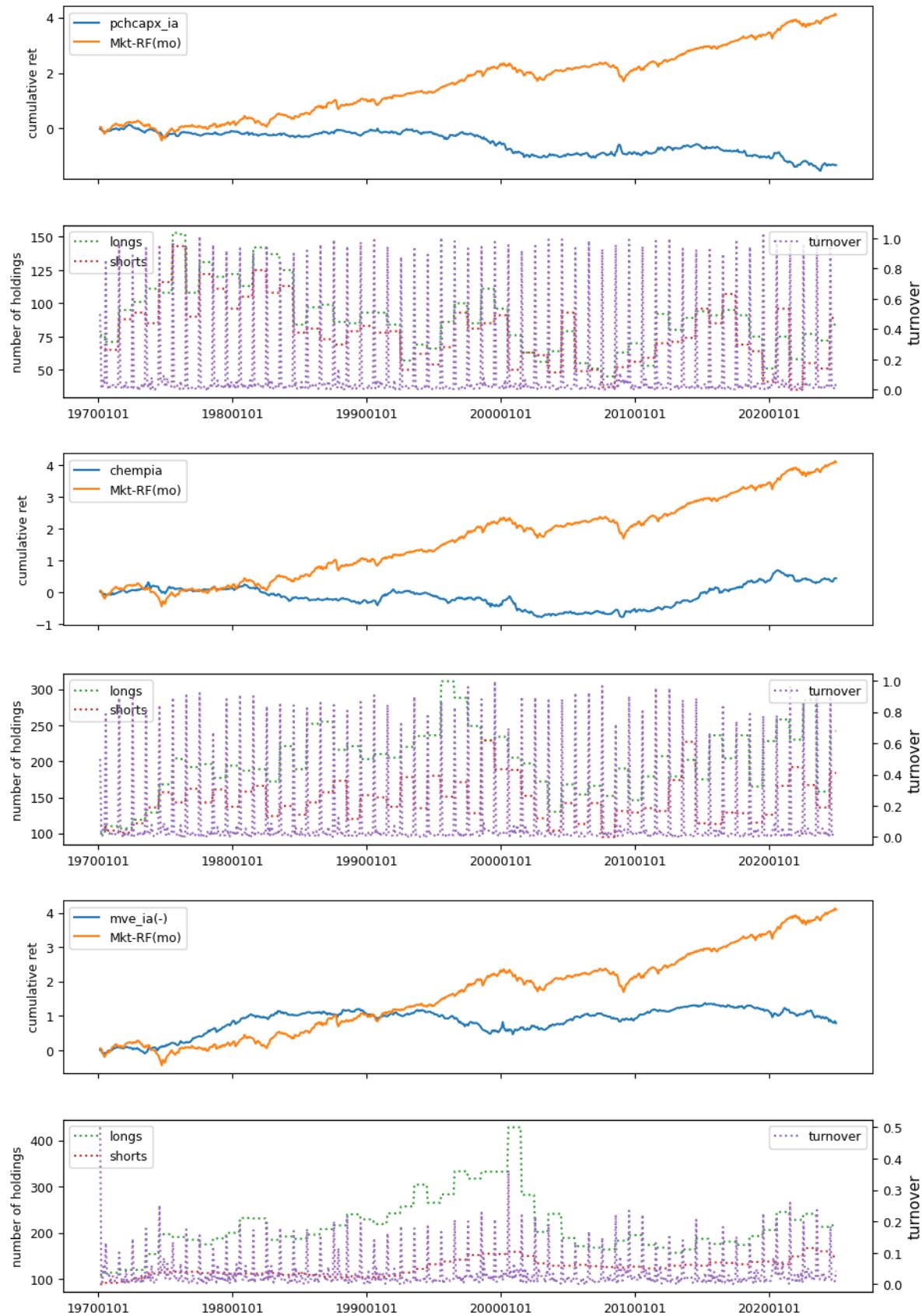












Fundamental signals from Compustat Quarterly

```

columns = ['stdacc', 'stdcf', 'roavol', 'sgrvol', 'cinvest', 'chtx',
           'rsup', 'roaq', 'cash', 'nincr']
numlag = 4          # require 4 month lag of fiscal data
end = LAST_DATE

# retrieve quarterly, keep [permno, datadate] key with non null prccq
fields = ['ibq', 'actq', 'cheq', 'lctq', 'dlcq', 'saleq', 'prccq',
          'cshoq', 'atq', 'txtq', 'ppentq']
df = pstat.get_linked(dataset='quarterly',
                      fields=fields,
                      date_field='datadate',
                      where=(f"datadate > 0 "
                             f"and datadate <= {end//100}31"))
fund = df.sort_values(['permno', 'datadate', 'ibq']) \
    .drop_duplicates(['permno', 'datadate']) \
    .dropna(subset=['ibq'])
fund.index = list(zip(fund['permno'], fund['datadate']))
rebalddate = bd.endmo(fund.datadate, numlag)

# compute current and lagged: scf sacc roaq nincr cinvest cash rsup chtx
lag = fund.shift(1, fill_value=0)
lag.loc[lag['permno'] != fund['permno'], fields] = np.nan
fund['_saleq'] = fund['saleq']
fund.loc[fund['_saleq'].lt(0.01), '_saleq'] = 0.01

fund['sacc'] = (((fund['actq'] - lag['actq']) - (fund['cheq'] - lag['cheq']))
                 - ((fund['lctq'] - lag['lctq'])
                     - (fund['dlcq'] - lag['dlcq']))) / fund['_saleq']
fund['cinvest'] = (fund['ppentq'] - lag['ppentq']) / fund['_saleq']
fund['nincr'] = (fund['ibq'] > lag['ibq']).astype(int)
fund['scf'] = (fund['ibq'] / fund['_saleq']) - fund['sacc']
fund['roaq'] = (fund['ibq'] / lag['atq'])
fund['cash'] = (fund['cheq'] / fund['atq'])

lag4 = fund.shift(4, fill_value=0)
lag4.loc[lag4['permno'] != fund['permno'], fields] = np.nan
fund['rsup'] = ((fund['saleq'] - lag4['saleq'])
                 / (fund['prccq'].abs() * fund['cshoq'].abs()))
fund['chtx'] = (fund['txtq'] - lag4['txtq']) / lag4['atq']

# for each var: make dataframe of 15 lags (column names=[0,...,15])
lags = {col : as_lags(fund, var=col, key='permno', nlags=16)
        for col in ['sacc', 'scf', 'roaq', 'rsup', 'cinvest', 'nincr']}
for i in range(1, 16):                      # lags[ninrc][i]=1 iff ibq
    lags['nincr'][i] *= lags['nincr'][i-1]  # increasing all prior qtrs

# compute signals from the 15 lags
fund['rebalddate'] = rebalddate
fund['stdacc'] = lags['sacc'].std(axis=1, skipna=False)
fund['stdcf'] = lags['scf'].std(axis=1, skipna=False)
fund['roavol'] = lags['roaq'].std(axis=1, skipna=False)
fund['sgrvol'] = lags['rsup'].std(axis=1, skipna=False)

```

(continues on next page)

(continued from previous page)

```

fund['cinvest'] = (fund['cinvest'] -
                    lags['cinvest'][[1, 2, 3, 4]].mean(axis=1, skipna=False))

# count number of consecutive increasing quarters
fund['nincr'] = lags['nincr'][np.arange(8)].sum(axis=1)

```

```

/home/terence/env3.11/lib/python3.11/site-packages/numpy/core/_methods.py:49:-
  ↵RuntimeWarning: invalid value encountered in reduce
    return umr_sum(a, axis, dtype, out, keepdims, initial, where)
/home/terence/env3.11/lib/python3.11/site-packages/pandas/core/nanops.py:1016:-
  ↵RuntimeWarning: invalid value encountered in subtract
    sqr = _ensure_numeric((avg - values) ** 2)
/home/terence/env3.11/lib/python3.11/site-packages/numpy/core/_methods.py:49:-
  ↵RuntimeWarning: invalid value encountered in reduce
    return umr_sum(a, axis, dtype, out, keepdims, initial, where)
/home/terence/env3.11/lib/python3.11/site-packages/pandas/core/nanops.py:1016:-
  ↵RuntimeWarning: invalid value encountered in subtract
    sqr = _ensure_numeric((avg - values) ** 2)
/home/terence/env3.11/lib/python3.11/site-packages/numpy/core/_methods.py:49:-
  ↵RuntimeWarning: invalid value encountered in reduce
    return umr_sum(a, axis, dtype, out, keepdims, initial, where)
/home/terence/env3.11/lib/python3.11/site-packages/pandas/core/nanops.py:1016:-
  ↵RuntimeWarning: invalid value encountered in subtract
    sqr = _ensure_numeric((avg - values) ** 2)
/home/terence/env3.11/lib/python3.11/site-packages/numpy/core/_methods.py:49:-
  ↵RuntimeWarning: invalid value encountered in reduce
    return umr_sum(a, axis, dtype, out, keepdims, initial, where)
/home/terence/env3.11/lib/python3.11/site-packages/pandas/core/nanops.py:1016:-
  ↵RuntimeWarning: invalid value encountered in subtract
    sqr = _ensure_numeric((avg - values) ** 2)
/home/terence/env3.11/lib/python3.11/site-packages/numpy/core/_methods.py:49:-
  ↵RuntimeWarning: invalid value encountered in reduce
    return umr_sum(a, axis, dtype, out, keepdims, initial, where)
/home/terence/env3.11/lib/python3.11/site-packages/pandas/core/nanops.py:1016:-
  ↵RuntimeWarning: invalid value encountered in subtract
    sqr = _ensure_numeric((avg - values) ** 2)
/home/terence/env3.11/lib/python3.11/site-packages/numpy/core/_methods.py:49:-
  ↵RuntimeWarning: invalid value encountered in reduce
    return umr_sum(a, axis, dtype, out, keepdims, initial, where)
/home/terence/env3.11/lib/python3.11/site-packages/pandas/core/nanops.py:1016:-
  ↵RuntimeWarning: invalid value encountered in subtract
    sqr = _ensure_numeric((avg - values) ** 2)
/home/terence/env3.11/lib/python3.11/site-packages/numpy/core/_methods.py:49:-
  ↵RuntimeWarning: invalid value encountered in reduce
    return umr_sum(a, axis, dtype, out, keepdims, initial, where)
/home/terence/env3.11/lib/python3.11/site-packages/pandas/core/nanops.py:1016:-
  ↵RuntimeWarning: invalid value encountered in subtract
    sqr = _ensure_numeric((avg - values) ** 2)
/home/terence/env3.11/lib/python3.11/site-packages/numpy/core/_methods.py:49:-
  ↵RuntimeWarning: invalid value encountered in reduce
    return umr_sum(a, axis, dtype, out, keepdims, initial, where)
/home/terence/env3.11/lib/python3.11/site-packages/pandas/core/nanops.py:1016:-
  ↵RuntimeWarning: invalid value encountered in subtract
    sqr = _ensure_numeric((avg - values) ** 2)
/home/terence/env3.11/lib/python3.11/site-packages/numpy/core/_methods.py:49:-
  ↵RuntimeWarning: invalid value encountered in reduce
    return umr_sum(a, axis, dtype, out, keepdims, initial, where)
/home/terence/env3.11/lib/python3.11/site-packages/pandas/core/nanops.py:1016:-
  ↵RuntimeWarning: invalid value encountered in subtract
    sqr = _ensure_numeric((avg - values) ** 2)
/home/terence/env3.11/lib/python3.11/site-packages/numpy/core/_methods.py:49:-
  ↵RuntimeWarning: invalid value encountered in reduce
    return umr_sum(a, axis, dtype, out, keepdims, initial, where)
/home/terence/env3.11/lib/python3.11/site-packages/pandas/core/nanops.py:1016:-
  ↵RuntimeWarning: invalid value encountered in subtract
    sqr = _ensure_numeric((avg - values) ** 2)

```

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

```

/home/terence/env3.11/lib/python3.11/site-packages/numpy/core/_methods.py:49:-
  ↪RuntimeWarning: invalid value encountered in reduce
    return umr_sum(a, axis, dtype, out, keepdims, initial, where)
/home/terence/env3.11/lib/python3.11/site-packages/pandas/core/nanops.py:1016:-
  ↪RuntimeWarning: invalid value encountered in subtract
    sqr = _ensure_numeric((avg - values) ** 2)
/home/terence/env3.11/lib/python3.11/site-packages/numpy/core/_methods.py:49:-
  ↪RuntimeWarning: invalid value encountered in reduce
    return umr_sum(a, axis, dtype, out, keepdims, initial, where)
/home/terence/env3.11/lib/python3.11/site-packages/pandas/core/nanops.py:1016:-
  ↪RuntimeWarning: invalid value encountered in subtract
    sqr = _ensure_numeric((avg - values) ** 2)
/home/terence/env3.11/lib/python3.11/site-packages/numpy/core/_methods.py:49:-
  ↪RuntimeWarning: invalid value encountered in reduce
    return umr_sum(a, axis, dtype, out, keepdims, initial, where)
/home/terence/env3.11/lib/python3.11/site-packages/pandas/core/nanops.py:1016:-
  ↪RuntimeWarning: invalid value encountered in subtract
    sqr = _ensure_numeric((avg - values) ** 2)

```

```

for label in columns:
    signals.write(fund, label, overwrite=True)

```

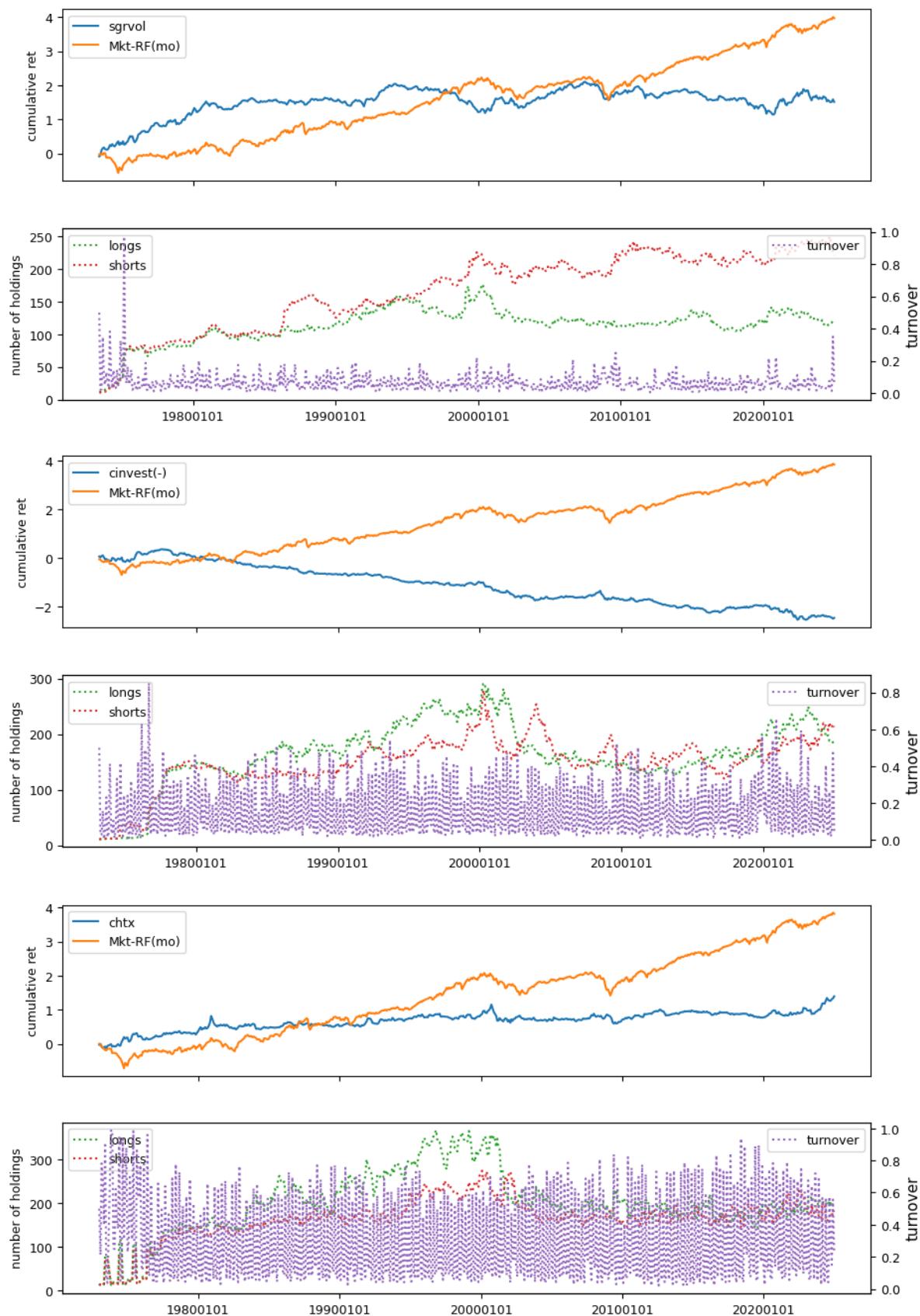
```

rebalbeg, rebalend = 19700101, LAST_DATE
benchnames = ['Mkt-RF (mo)']
for label in tqdm(columns):
    holdings = univariate_sorts(monthly,
                                 label,
                                 SignalsFrame(signals.read(label)),
                                 rebalbeg,
                                 rebalend,
                                 window=3,
                                 months=[],
                                 maxdecile=8,
                                 pct=(10., 90.),
                                 leverage=leverage.get(label, 1))
    excess = backtest_pipeline(backtest,
                               monthly,
                               holdings,
                               label,
                               benchnames,
                               overlap=0,
                               outdir=outdir,
                               suffix='(-)'*(leverage.get(label, 1) < 0))

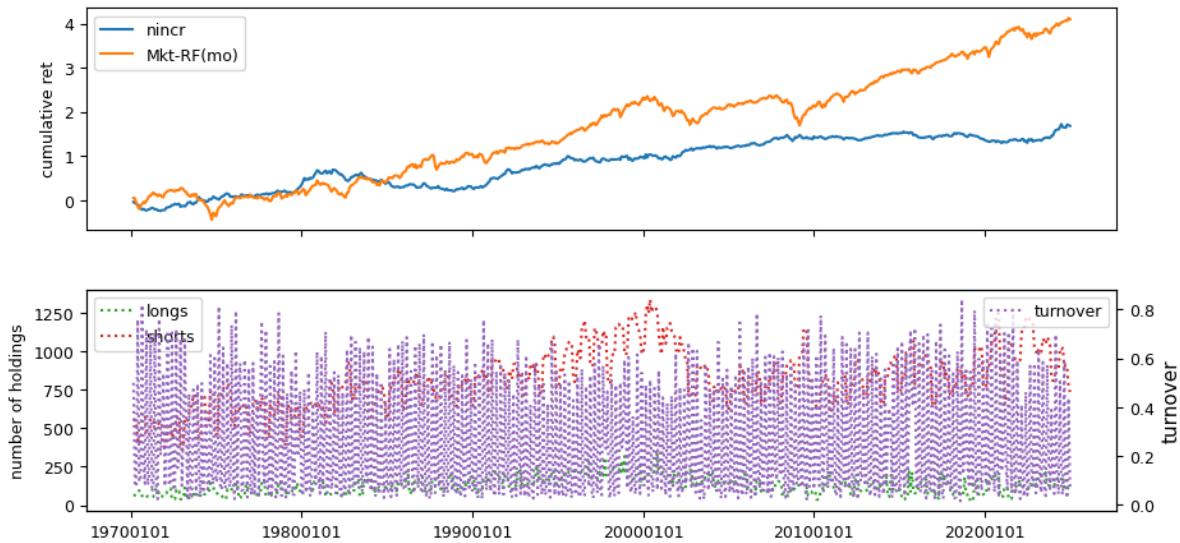
```

100% |██████████| 10/10 [11:53<00:00, 71.38s/it]









6.1.6 Earnings Estimates

Earnings estimate signals are drawn from IBES data, including fiscal year 1 projections, long-term growth forecasts, and announcement dates linked to CRSP daily prices and Compustat quarterly.

```

columns = ['chfeps', 'chnanalyst', 'disp']

df = ibes.get_linked(dataset='statsum',
                      fields=['fpedats', 'meanest', 'medest',
                              'stdev', 'numest'],
                      date_field = 'statpers',
                      where=("meanest IS NOT NULL "
                             " AND fpedats IS NOT NULL "
                             " AND statpers IS NOT NULL"
                             " AND fpi = '1'"))
out = df.sort_values(['permno', 'statpers', 'fpedats', 'meanest']) \
    .drop_duplicates(['permno', 'statpers', 'fpedats'])
out['rebaldate'] = bd.endmo(out['statpers'])

out['disp'] = out['stdev'] / abs(out['meanest'])
out.loc[abs(out['meanest']) < 0, 'disp'] = out['stdev'] / 0.01

lag1 = out.shift(1, fill_value=0)
f1 = (lag1['permno'] == out['permno'])
out.loc[f1, 'chfeps'] = out.loc[f1, 'meanest'] - lag1.loc[f1, 'meanest']

lag3 = out.shift(3, fill_value=0)
f3 = (lag3['permno'] == out['permno'])
out.loc[f3, 'chnanalyst'] = out.loc[f3, 'numest']-lag3.loc[f3, 'numest']

for label in columns:
    signals.write(out, label, overwrite=True)

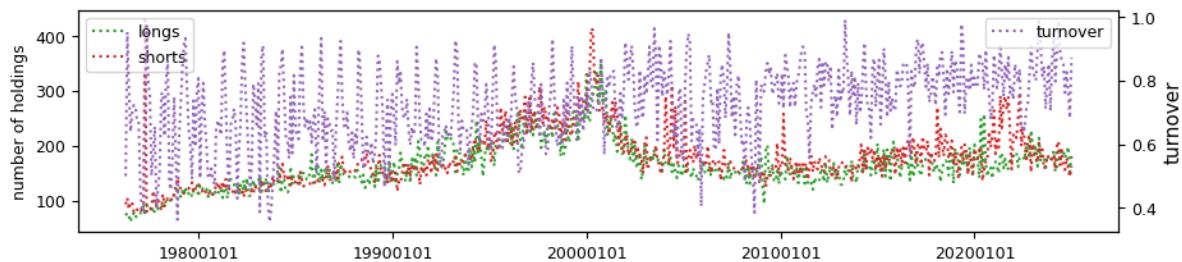
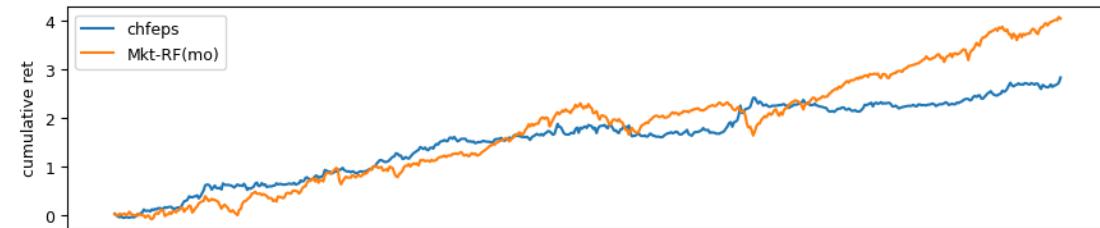
```

```

rebalbeg, rebalend = 19760101, LAST_DATE
benchnames = ['Mkt-RF(mo)']
for label in tqdm(columns):
    holdings = univariate_sorts(monthly,
                                 label,
                                 SignalsFrame(signals.read(label)),
                                 rebalbeg,
                                 rebalend,
                                 window=3,
                                 months=[],
                                 maxdecile=8,
                                 pct=(10., 90.),
                                 leverage=leverage.get(label, 1))
    excess = backtest_pipeline(backtest,
                               monthly,
                               holdings,
                               label,
                               benchnames,
                               overlap=0,
                               outdir=outdir,
                               suffix=(leverage.get(label, 1) < 0) * '(-)')

```

100% |██████████| 3/3 [03:39<00:00, 73.25s/it]





IBES Long-term Growth signals

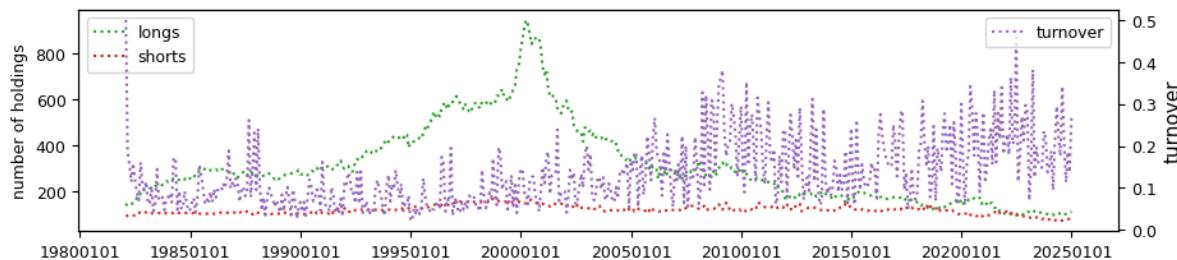
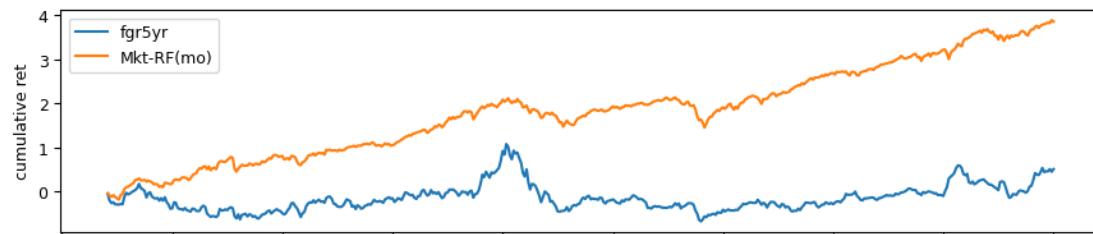
```
columns = ['fgr5yr']

df = ibes.get_linked(dataset='statsum',
                      fields = ['meanest'],
                      date_field = 'statpers',
                      where=("meanest IS NOT NULL "
                            "AND fpi = '0'"
                            "AND statpers IS NOT NULL"))
out = df.sort_values(['permno', 'statpers', 'meanest'])\
    .drop_duplicates(['permno', 'statpers'])\
    .dropna()
out['rebaldate'] = bd.endmo(out['statpers'])
out['fgr5yr'] = out['meanest']
signals.write(out, 'fgr5yr', overwrite=True)
```

1319938

```
rebalbeg, rebalend = 19760101, LAST_DATE
benchnames = ['Mkt-RF (mo)']
for label in tqdm(columns):
    holdings = univariate_sorts(monthly,
                                 label,
                                 SignalsFrame(signals.read(label)),
                                 rebalbeg,
                                 rebalend,
                                 window=3,
                                 months=[],
                                 maxdecile=8,
                                 pct=(10., 90.),
                                 leverage=leverage.get(label, 1))
    excess = backtest_pipeline(backtest,
                               monthly,
                               holdings,
                               label,
                               benchnames,
                               overlap=0,
                               outdir=outdir,
                               suffix=(leverage.get(label, 1) < 0) * '(-)')
```

100% |██████████| 1/1 [01:06<00:00, 66.27s/it]



Announcement date in Quarterly, linked to CRSP daily

```
columns = ['ear', 'aeavol']
```

```
# retrieve rdq, and set rebalance date to at least one month delay
df = pstat.get_linked(dataset='quarterly',
                      fields=['rdq'],
                      date_field='datadate',
                      where=('rdq > 0'))
```

(continues on next page)

(continued from previous page)

```
fund = df.sort_values(['permno', 'rdq', 'datadate'])\
    .drop_duplicates(['permno', 'rdq'])\
    .dropna()
fund['rebaldate'] = bd.offset(fund['rdq'], 2)
```

```
# ear is compounded return around 3-day window
out = crsp.get_window(dataset='daily',
                       field='ret',
                       date_field='date',
                       permnos=fund['permno'],
                       dates=fund['rdq'],
                       left=-1,
                       right=1)
fund['ear'] = (1 + out).prod(axis=1).values
```

```
# aeavol is avg volume in 3-day window over 20-day average ten-days prior
actual = crsp.get_window(dataset='daily',
                         field='vol',
                         date_field='date',
                         permnos=fund['permno'],
                         dates=fund['rdq'],
                         left=-1,
                         right=1)
normal = crsp.get_window(dataset='daily',
                         field='vol',
                         date_field='date',
                         permnos=fund['permno'],
                         dates=fund['rdq'],
                         left=-30,
                         right=-11,
                         avg=True)
fund['aeavol'] = normal['vol'].values
```

```
signals.write(fund, 'ear', overwrite=True)
signals.write(fund, 'aeavol', overwrite=True)
```

968315

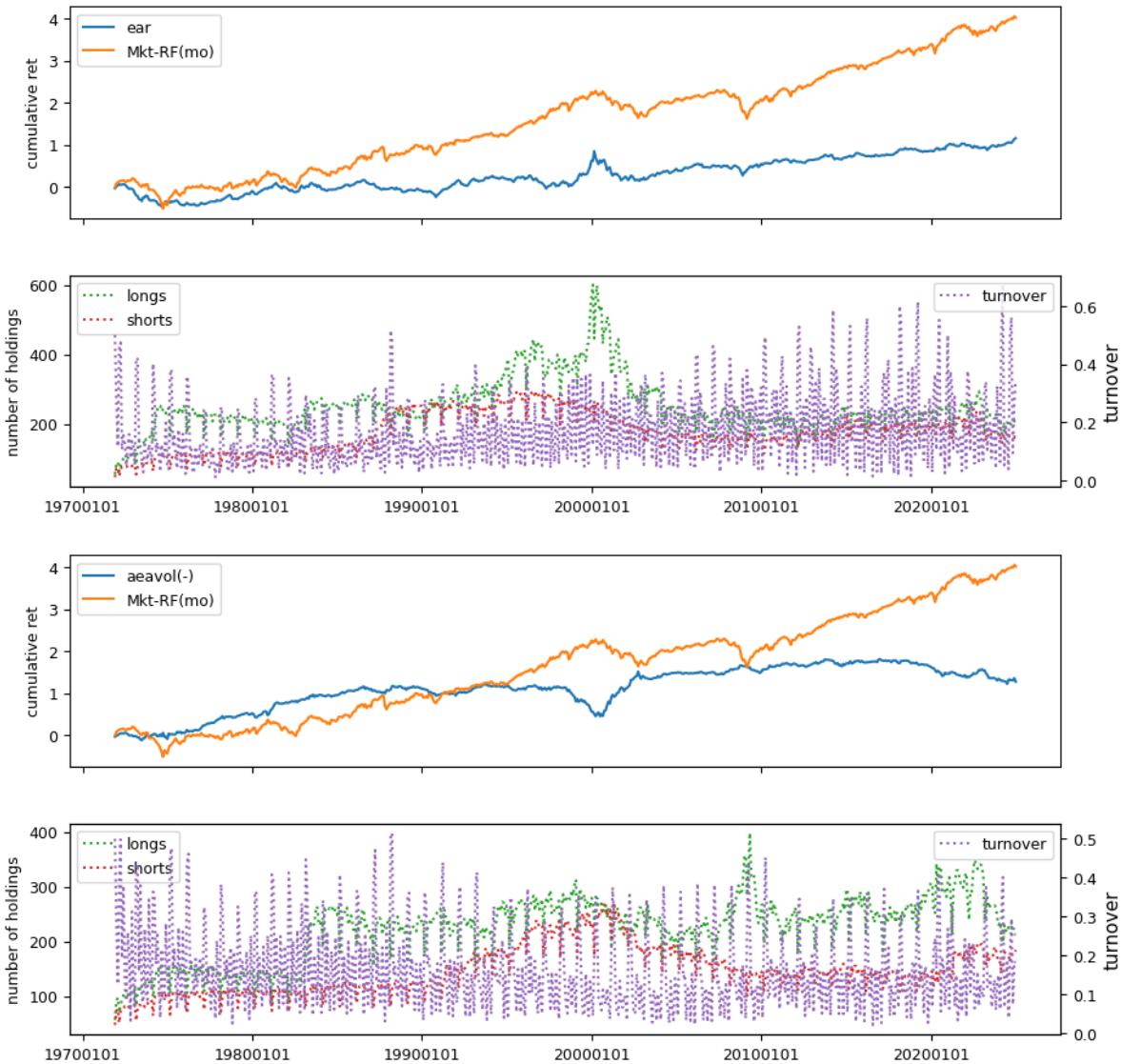
```
rebalbeg, rebalend = 19700101, LAST_DATE
benchnames = ['Mkt-RF(mo)']
for label in tqdm(columns):
    holdings = univariate_sorts(monthly,
                                 label,
                                 SignalsFrame(signals.read(label)),
                                 rebalbeg,
                                 rebalend,
                                 window=3,
                                 months=[],
                                 maxdecile=8,
                                 pct=(10., 90.),
                                 leverage=leverage.get(label, 1))
excess = backtest_pipeline(backtest,
                           monthly,
```

(continues on next page)

(continued from previous page)

```
holdings,
label,
benchnames,
overlap=0,
outdir=outdir,
suffix=(leverage.get(label, 1) < 0) * '(-)'
```

100% |██████████| 2/2 [02:07<00:00, 63.88s/it]



IBES Fiscal Year 1 linked to Quarterly PSTAT

```
beg, end = 19760101, LAST_DATE
monthnum = lambda d: ((d//10000)-1900)*12 + ((d//100)%100) - 1
```

```
df = pstat.get_linked(dataset='quarterly',
```

(continues on next page)

(continued from previous page)

```

        fields=['prccq'],
        date_field='datadate')

df = df.dropna()\
    .sort_values(['permno', 'datadate'])\
    .drop_duplicates(['permno', 'datadate'])

out = ibes.get_linked(dataset='statsum',
                      fields=['fpedats', 'meanest'],
                      date_field='statpers',
                      where="fpi='1'")

out = out.dropna()\
    .sort_values(['permno', 'statpers', 'fpedats'])\
    .drop_duplicates(['permno', 'statpers'])
out['monthnum'] = monthnum(out['statpers'])
out = out.set_index(['permno', 'monthnum'], drop=False)
out['sfeq'] = np.nan

```

```

for num in range(4): # match ibes statpers to any datadate in last 4 mos
    df['monthnum'] = monthnum(df['datadate']) - num
    df = df.set_index(['permno', 'monthnum'], drop=False)
    out = out.join(df[['prccq']], how='left')
    out['sfeq'] = out['sfeq'].where(out['sfeq'].notna(),
                                      out['meanest'] / out['prccq'].abs())
    out = out.drop(columns=['prccq'])

    out['rebalddate'] = bd.endmo(out['statpers'])
    n = signals.write(out.reset_index(drop=True), 'sfeq', overwrite=True)

```

IBES Fiscal Year 1 linked to IBES price history

```
beg, end = 19760101, LAST_DATE
```

```

# retrieve monthly price history
df = ibes.get_linked(dataset='actpsum',
                      fields=['price'],
                      date_field='statpers')
hist = df.dropna()\
    .sort_values(['permno', 'statpers'])\
    .drop_duplicates(['permno', 'statpers'], keep='last')\
    .set_index(['permno', 'statpers'])

```

```

# retrieve monthly FY1 mean estimate
df = ibes.get_linked(dataset='statsum',
                      fields=['fpedats', 'meanest'],
                      date_field='statpers',
                      where="fpi='1' AND statpers <= fpedats")

out = df.dropna()\
    .sort_values(['permno', 'statpers', 'fpedats'])\
    .drop_duplicates(['permno', 'statpers'])\
    .set_index(['permno', 'statpers'])

```

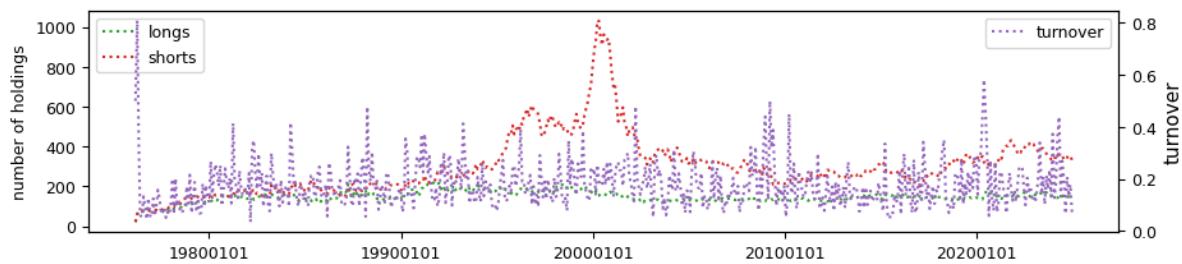
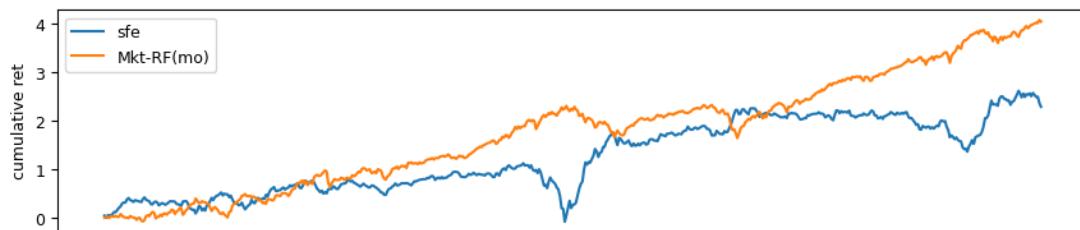
```
# join on [permno, statpers], and reindex on [permno, rebalddate]
```

(continues on next page)

(continued from previous page)

```
out = out.join(hist[['price']], how='left').reset_index()
out['rebaldate'] = bd.endmo(out['statpers'])
out = out.set_index(['permno', 'rebaldate'])
out['sfe'] = out['meanest'].div(out['price'].abs())
n = signals.write(out.reset_index(), 'sfe', overwrite=True)
```

```
rebalbeg, rebalend = 19760101, LAST_DATE
benchnames = ['Mkt-RF(mo)']
label = 'sfe'
holdings = univariate_sorts(monthly,
                             label,
                             SignalsFrame(signals.read(label)),
                             rebalbeg,
                             rebalend,
                             window=3,
                             months=[],
                             maxdecile=8,
                             pct=(10., 90.),
                             leverage=leverage.get(label, 1))
excess = backtest_pipeline(backtest,
                           monthly,
                           holdings,
                           label,
                           benchnames,
                           overlap=0,
                           outdir=outdir,
                           suffix=(leverage.get(label, 1) < 0) * '(-)')
```



IBES Fiscal Quarter 1, linked to Quarterly

```
columns = ['sue']
numlag = 4
end = LAST_DATE
```

```
# retrieve quarterly, keep [permno, datadate] key with non null prccq
df = pstat.get_linked(dataset='quarterly',
                      fields=['prccq', 'cshoq', 'ibq'],
                      date_field='datadate',
                      where=f"datadate <= {end//100}31")
fund = df.dropna(subset=['ibq']) \
    .sort_values(['permno', 'datadate', 'cshoq']) \
    .drop_duplicates(['permno', 'datadate'])
fund['rebalddate'] = bd.endmo(fund['datadate'], numlag)
fund = fund.set_index(['permno', 'rebalddate'], drop=False)
```

```
# retrieve ibes Q1 where forecast period <= fiscal date, keep latest
df = ibes.get_linked(dataset='statsum',
                      fields=['fpedats', 'medest', 'actual'],
                      date_field='statpers',
                      where=" fpi = '6' AND statpers <= fpedats")
summ = df.dropna() \
    .sort_values(['permno', 'fpedats', 'statpers']) \
    .drop_duplicates(['permno', 'fpedats'], keep='last')
summ['rebalddate'] = bd.endmo(summ['fpedats'], numlag)
summ = summ.set_index(['permno', 'statpers'])
```

```
# retrieve ibes price, then left join
df = ibes.get_linked(dataset='actpsum',
                      fields=['price'],
                      date_field='statpers')
hist = df.dropna() \
    .sort_values(['permno', 'statpers']) \
    .drop_duplicates(['permno', 'statpers'], keep='last')
hist = hist.set_index(['permno', 'statpers'])
summ = summ.join(hist[['price']], how='left')
summ = summ.reset_index() \
    .set_index(['permno', 'rebalddate']) \
    .reindex(fund.index)
```

```
# sue with ibes surprise and price
fund['sue'] = (summ['actual'] - summ['medest']) / summ['price'].abs()
```

```
# sue with ibes surprice and compustat quarterly price
fund['sue'] = fund['sue'].where(
    fund['sue'].notna(), (summ['actual'] - summ['medest']) / fund['prccq'].abs())
```

```
# sue with lag(4) difference in compustat quarterly and price
lag = fund.shift(4, fill_value=0)
fund['sue'] = fund['sue'].where(
    fund['sue'].notna() | (lag['permno'] != fund['permno']),
    ((fund['ibq'] - lag['ibq']) / (fund['prccq'] * fund['cshoq']).abs()))
```

```
signals.write(fund.reset_index(drop=True), 'sue', overwrite=True)
```

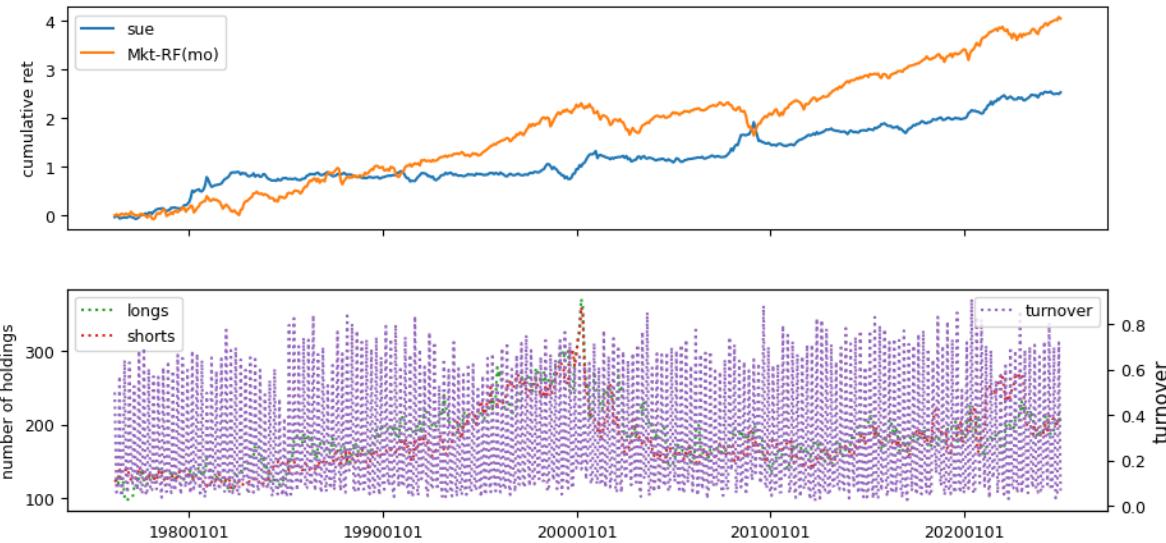
1089089

```

rebalbeg, rebalend = 19760101, LAST_DATE
benchnames = ['Mkt-RF(mo)']
for label in tqdm(columns):
    holdings = univariate_sorts(monthly,
                                 label,
                                 SignalsFrame(signals.read(label)),
                                 rebalbeg,
                                 rebalend,
                                 window=3,
                                 months=[],
                                 maxdecile=8,
                                 pct=(10., 90.),
                                 leverage=leverage.get(label, 1))
    excess = backtest_pipeline(backtest,
                               monthly,
                               holdings,
                               label,
                               benchnames,
                               overlap=0,
                               outdir=outdir,
                               suffix=(leverage.get(label, 1) < 0) * '(-)')

```

100% |██████████| 1/1 [01:14<00:00, 74.37s/it]



6.1.7 Backtests

All backtests are conducted on univariate, dollar-neutral spreads between the top and bottom deciles of a given characteristic. Stocks are value-weighted within each decile, excluding the smallest quintile of NYSE-listed firms by market capitalization. Spread portfolios are rebalanced monthly, with fundamental data lagged in the standard manner—six months for annual data and four months for quarterly data.

Each backtest generates a time series of cumulative spread portfolio and market index returns, along with metrics such as monthly turnover rates and the number of long and short positions. Results are ranked based on Welch's t -statistic, which tests for differences in mean returns before 2002 and after 2003. This aligns with findings from Green et al. (2017) and others, who observed a significant decline in the predictive power of many factors after this period.

Additionally, we compute the **maximum drawdown** for each strategy, defined as the largest peak-to-trough loss in cumulative returns before a new peak is reached. This measure provides insight into the worst-case historical performance of each strategy.

```
def maximum_drawdown(x: Series, is_price_level: bool = False) -> Series:
    """Compute max drawdown (max loss from previous high) period and returns"""
    cumsum = np.log(1 + x).cumsum()
    cummax = cumsum.cummax()
    end = (cummax - cumsum).idxmax()
    beg = cumsum[cumsum.index <= end].idxmax()
    dd = cumsum.loc[[beg, end]]
    return np.exp(dd)
```

```
zoo = backtest.read().sort_values(['begret', 'permno'])
r = []
rets = []
for label in zoo.index:
    perf = backtest.read(label)
    rets.append(perf['ret'].rename(label))
    excess = {'ret': backtest.fit(['Mkt-RF(mo)'])}
    excess['annualized'] = backtest.annualized
    excess['dd'] = maximum_drawdown(backtest.perf['excess'])
    post = {'ret': backtest.fit(['Mkt-RF(mo)'],
                                beg=20020101).copy()}
    post['annualized'] = backtest.annualized.copy()
    s = label + ('(-)' if leverage.get(label, 1) < 0 else '')
    r.append(DataFrame({
        '#Start': excess['ret'].index[0],
        'Sharpe Ratio': excess['annualized']['sharpe'],
        'Alpha': excess['annualized']['alpha'],
        'Appraisal Ratio': excess['annualized']['appraisal'],
        'Avg Ret': excess['ret']['excess'].mean(),
        'Vol': excess['ret']['excess'].std(ddof=0),
        'Welch-t': excess['annualized']['welch-t'],
        'Appraisal2002': post['annualized']['appraisal'],
        'Ret2002': post['ret']['excess'].mean(),
        'Drawdown': (excess['dd'].iloc[1]/excess['dd'].iloc[0]) - 1,
    }, index=[s]))
df = pd.concat(r, axis=0).round(4).sort_values('Welch-t')
```

```
pd.set_option("display.max_colwidth", None, 'display.max_rows', None)
df#.sort_values('Sharpe Ratio', ascending=False)
```

	Sharpe Ratio	Alpha	Appraisal Ratio	Avg Ret	Vol	\
pchsale_pchrect	-0.1169	-0.0121	-0.1368	-0.0009	0.0256	
mom1m(-)	0.3820	0.0568	0.3132	0.0061	0.0543	
bm	0.2233	0.0339	0.2141	0.0029	0.0457	
bm_ia	0.2413	0.0216	0.1611	0.0027	0.0393	
agr(-)	0.2493	0.0480	0.4320	0.0025	0.0341	
pchsale_pchinvt	0.2434	0.0229	0.2392	0.0019	0.0276	
chmom	0.4007	0.0509	0.3114	0.0058	0.0494	
pchsaleinv	0.2446	0.0197	0.2238	0.0018	0.0254	
lev	0.0261	-0.0002	-0.0012	0.0004	0.0488	
absacc(-)	0.0380	0.0212	0.1520	0.0005	0.0414	
ep	0.2831	0.0639	0.4231	0.0037	0.0452	

(continues on next page)

(continued from previous page)

chcsho(-)	0.5415	0.0725	0.7490	0.0047	0.0298
invest(-)	0.2479	0.0450	0.3776	0.0025	0.0356
acc(-)	0.3358	0.0377	0.3500	0.0030	0.0312
indmom	0.3518	0.0643	0.3708	0.0053	0.0512
sp	0.2303	0.0329	0.2220	0.0028	0.0428
pchdepr	0.0617	0.0048	0.0506	0.0005	0.0273
egr(-)	0.3157	0.0539	0.5072	0.0030	0.0327
sgrvol	0.1918	0.0082	0.0562	0.0024	0.0440
mom12m	0.4699	0.1532	0.6550	0.0101	0.0732
realestate	0.1084	0.0508	0.3445	0.0014	0.0458
mom36m(-)	0.1778	0.0191	0.0953	0.0031	0.0601
cashpr(-)	0.2962	0.0475	0.3843	0.0031	0.0363
divyld	-0.0470	0.0377	0.2455	-0.0008	0.0552
aeavol(-)	0.2160	0.0381	0.3568	0.0020	0.0320
ill(-)	0.3323	0.0504	0.4209	0.0034	0.0350
dy	-0.0473	0.0382	0.2262	-0.0008	0.0569
chatoia	0.2567	0.0292	0.2994	0.0021	0.0282
chinv(-)	0.3601	0.0506	0.4775	0.0033	0.0314
retvol(-)	0.1306	0.1269	0.6067	0.0029	0.0772
maxret(-)	0.1807	0.1144	0.6383	0.0034	0.0646
pricedelay(-)	0.0819	0.0030	0.0302	0.0007	0.0292
chfeps	0.4741	0.0673	0.5567	0.0048	0.0353
stdcf(-)	0.2132	0.0613	0.5020	0.0024	0.0391
idiovol(-)	-0.0126	0.0514	0.2623	-0.0003	0.0697
mve_ia(-)	0.1372	0.0025	0.0244	0.0012	0.0302
chpmia	0.1598	0.0266	0.2003	0.0018	0.0385
disp(-)	0.3267	0.1025	0.6640	0.0048	0.0508
cfp	0.2466	0.0511	0.3691	0.0029	0.0412
stdacc(-)	0.1414	0.0431	0.3801	0.0014	0.0354
pchsale_pchxsga	-0.0415	-0.0053	-0.0480	-0.0004	0.0320
grltnoa	0.1595	0.0208	0.1793	0.0015	0.0335
sfe	0.2535	0.0763	0.4340	0.0039	0.0532
pchgm_pchsale	0.0106	-0.0035	-0.0337	0.0001	0.0297
cfp_ia	-0.1814	-0.0331	-0.2501	-0.0020	0.0386
mom6m	0.3185	0.1085	0.5001	0.0064	0.0682
beta	0.0807	-0.0733	-0.3806	0.0022	0.0906
dolvol	0.1131	0.0017	0.0161	0.0010	0.0315
nincr	0.3410	0.0242	0.2728	0.0026	0.0259
cinvest(-)	-0.4026	-0.0447	-0.3815	-0.0039	0.0339
cash	0.2751	0.0143	0.1021	0.0035	0.0438
chtx	0.2162	0.0211	0.1716	0.0022	0.0358
salecash	0.0170	0.0199	0.1773	0.0002	0.0342
rd_mve	0.1412	0.0114	0.0716	0.0019	0.0463
chnanalyst	0.0875	0.0021	0.0240	0.0006	0.0255
fgr5yr	0.0537	-0.0485	-0.2487	0.0010	0.0640
sue	0.4709	0.0557	0.5087	0.0043	0.0317
pchcapx_ia	-0.1942	-0.0262	-0.2113	-0.0020	0.0359
roavol	0.0768	-0.0063	-0.0425	0.0010	0.0440
basspread	-0.0851	-0.1231	-0.5790	-0.0020	0.0804
pchquick	-0.0755	-0.0085	-0.0918	-0.0006	0.0268
rd_sale	-0.0530	-0.0146	-0.0945	-0.0007	0.0446
pctacc(-)	0.1174	0.0248	0.2364	0.0011	0.0312
secured(-)	0.0364	0.0337	0.2793	0.0004	0.0376
std_turn	0.0539	-0.0490	-0.2884	0.0009	0.0582
std_dolvol	-0.0176	0.0007	0.0068	-0.0002	0.0315
rsup	0.1644	0.0237	0.1859	0.0018	0.0369

(continues on next page)

(continued from previous page)

depr	0.0696	-0.0201	-0.1534	0.0008	0.0422
zerotrade (-)	0.1663	-0.0351	-0.1939	0.0031	0.0644
sgr	0.0225	-0.0138	-0.1111	0.0002	0.0372
salerec	0.2028	0.0405	0.3172	0.0022	0.0378
roaq	0.2858	0.0610	0.4358	0.0035	0.0422
turn	0.1420	-0.0412	-0.2184	0.0027	0.0669
ear	0.1909	-0.0019	-0.0185	0.0018	0.0329
quick	0.0603	-0.0241	-0.1670	0.0008	0.0467
gma	0.1117	0.0201	0.1363	0.0014	0.0426
hire	-0.1215	-0.0324	-0.2790	-0.0012	0.0353
tang	0.2328	0.0149	0.1259	0.0024	0.0351
grcapx	-0.2991	-0.0421	-0.4188	-0.0026	0.0298
saleinv	0.1401	0.0364	0.3711	0.0013	0.0312
chempia	0.0698	-0.0084	-0.0768	0.0007	0.0331
lgr	-0.0573	-0.0162	-0.1720	-0.0005	0.0279
	Welch-t	Appraisal2002	Ret2002	Drawdown	
pchsale_pchrect	-3.1904	-0.6200	-0.0047	-0.8591	
mom1m(-)	-2.7253	-0.3205	-0.0017	-0.7606	
bm	-2.3167	-0.3007	-0.0019	-0.7651	
bm_ia	-2.2583	-0.2118	-0.0012	-0.7011	
agr(-)	-2.1174	0.0200	-0.0008	-0.5362	
pchsale_pchinv	-2.1010	-0.1343	-0.0008	-0.4944	
chmom	-1.9158	-0.0634	0.0010	-0.6159	
pchsaleinv	-1.7332	-0.1297	-0.0003	-0.4843	
lev	-1.6920	-0.3338	-0.0035	-0.8698	
absacc(-)	-1.6578	-0.1800	-0.0027	-0.7840	
ep	-1.4698	0.2254	0.0007	-0.6637	
chcsho(-)	-1.4605	0.5437	0.0027	-0.3253	
invest(-)	-1.3936	0.1366	0.0001	-0.4268	
acc(-)	-1.3486	0.0828	0.0011	-0.3966	
indmom	-1.3305	0.2007	0.0020	-0.6892	
sp	-1.2746	-0.0372	0.0004	-0.5726	
pchdepr	-1.2651	-0.1436	-0.0011	-0.6979	
egr(-)	-1.2025	0.3026	0.0013	-0.3384	
sgrvol	-1.1672	-0.2804	0.0001	-0.6935	
mom12m	-1.1488	0.5282	0.0058	-0.9670	
realestate	-1.1485	0.2426	-0.0007	-0.6426	
mom36m(-)	-1.0582	-0.0673	-0.0000	-0.7280	
cashpr(-)	-1.0456	0.0969	0.0014	-0.4235	
divyld	-1.0071	-0.1090	-0.0032	-0.9650	
aeavol(-)	-0.9899	0.2019	0.0006	-0.5583	
ill(-)	-0.8358	0.4068	0.0022	-0.6150	
dy	-0.7118	-0.0399	-0.0025	-0.9190	
chatoia	-0.6699	0.1768	0.0012	-0.5082	
chinv(-)	-0.6658	0.2930	0.0023	-0.2747	
retvol(-)	-0.6457	0.5100	0.0009	-0.7990	
maxret(-)	-0.6188	0.5778	0.0017	-0.7173	
pricedelay(-)	-0.5810	-0.1441	-0.0002	-0.6155	
chfeps	-0.5669	0.5971	0.0040	-0.2969	
stdcf(-)	-0.5562	0.4330	0.0015	-0.4787	
idiovol(-)	-0.4946	0.3285	-0.0021	-0.9875	
mve_ia(-)	-0.4777	-0.0749	0.0005	-0.5605	
chpmia	-0.4676	0.1088	0.0009	-0.6303	
disp(-)	-0.3850	0.7927	0.0039	-0.5300	
cfp	-0.3045	0.2216	0.0024	-0.5708	

(continues on next page)

(continued from previous page)

stdacc(-)	-0.2576	0.3357	0.0011	-0.5056
pchsale_pchxsga	-0.2270	-0.1208	-0.0007	-0.7740
grltnoa	-0.2164	0.1620	0.0012	-0.5122
sfe	-0.1952	0.3435	0.0034	-0.7417
pchgm_pchsale	-0.1935	0.0227	-0.0002	-0.5895
cfp_ia	-0.1337	-0.3944	-0.0022	-0.8613
mom6m	-0.1285	0.5223	0.0060	-0.9885
beta	-0.1225	-0.4818	0.0016	-0.9586
dolvol	-0.1030	0.1331	0.0009	-0.7624
nincr	-0.0929	0.3654	0.0024	-0.4061
cinvest(-)	0.0387	-0.3699	-0.0039	-0.9587
cash	0.0479	0.0796	0.0036	-0.7076
chtx	0.1783	0.3262	0.0025	-0.4779
salecash	0.2132	0.2326	0.0005	-0.7463
rd_mve	0.2541	0.0853	0.0024	-0.7868
chnanalyst	0.2806	0.1425	0.0010	-0.5443
fgr5yr	0.3620	-0.0030	0.0020	-0.8761
sue	0.3800	0.6849	0.0048	-0.4062
pchcapx_ia	0.4287	-0.0762	-0.0013	-0.8750
roavol	0.4640	0.0964	0.0019	-0.7812
basspread	0.4727	-0.5203	-0.0005	-0.9539
pchquick	0.5811	-0.0879	0.0001	-0.7586
rd_sale	0.6114	-0.0313	0.0005	-0.8706
pctacc(-)	0.6117	0.2568	0.0019	-0.3803
secured(-)	0.6222	0.3922	0.0014	-0.6406
std_turn	0.6391	-0.2285	0.0024	-0.8064
std_dolvol	0.7879	0.0782	0.0008	-0.5301
rsup	0.8444	0.4582	0.0032	-0.5353
depr	0.8697	0.0616	0.0024	-0.6522
zerotrade(-)	0.8812	-0.0779	0.0054	-0.8344
sgr	0.9168	0.0682	0.0018	-0.7062
salerec	0.9547	0.5194	0.0040	-0.6761
roaq	0.9856	0.7546	0.0054	-0.4532
turn	0.9945	-0.0831	0.0054	-0.8291
ear	1.0453	0.2661	0.0033	-0.5423
quick	1.2014	0.0018	0.0033	-0.7923
gma	1.2529	0.4330	0.0039	-0.6364
hire	1.4328	0.0264	0.0011	-0.8262
tang	1.4843	0.3043	0.0047	-0.7361
grcapx	1.5208	-0.1320	-0.0004	-0.8714
saleinv	1.9087	0.6251	0.0040	-0.4900
chempia	2.0683	0.3022	0.0038	-0.7319
lgr	2.2177	0.2295	0.0024	-0.8286

```
X = pd.concat(rets, join='outer', axis=1).dropna()
X = X/X.std()      # standardize to unit variance
corr = X.corr()
dist = np.sqrt(1 - corr)
```

6.2 Cluster analysis

We use historical backtest returns as input features for cluster analysis, enabling the identification of peer benchmarks for different investment strategies. Strategies that exhibit similar return patterns tend to load on the same “style” factors and should be evaluated against one another.

The correlation between two return series is closely related to their Euclidean distance. When returns are standardized to have unit variance, the squared Euclidean norm between two series x and y can be expressed as:

$$d^2 = \sum_i (x_i - y_i)^2 = \sum_i x_i^2 + \sum_i y_i^2 - 2 \sum_i x_i y_i = n + n - 2n\rho$$

since $\sum_i x_i^2 = \sum_i y_i^2 = n$, and $\rho = \sum_i x_i y_i / n$, where ρ represents the correlation between the two series. This relationship allows correlation to be rewritten as:

$$\rho = 1 - \frac{d^2}{2n}$$

6.2.1 Hierarchical clustering

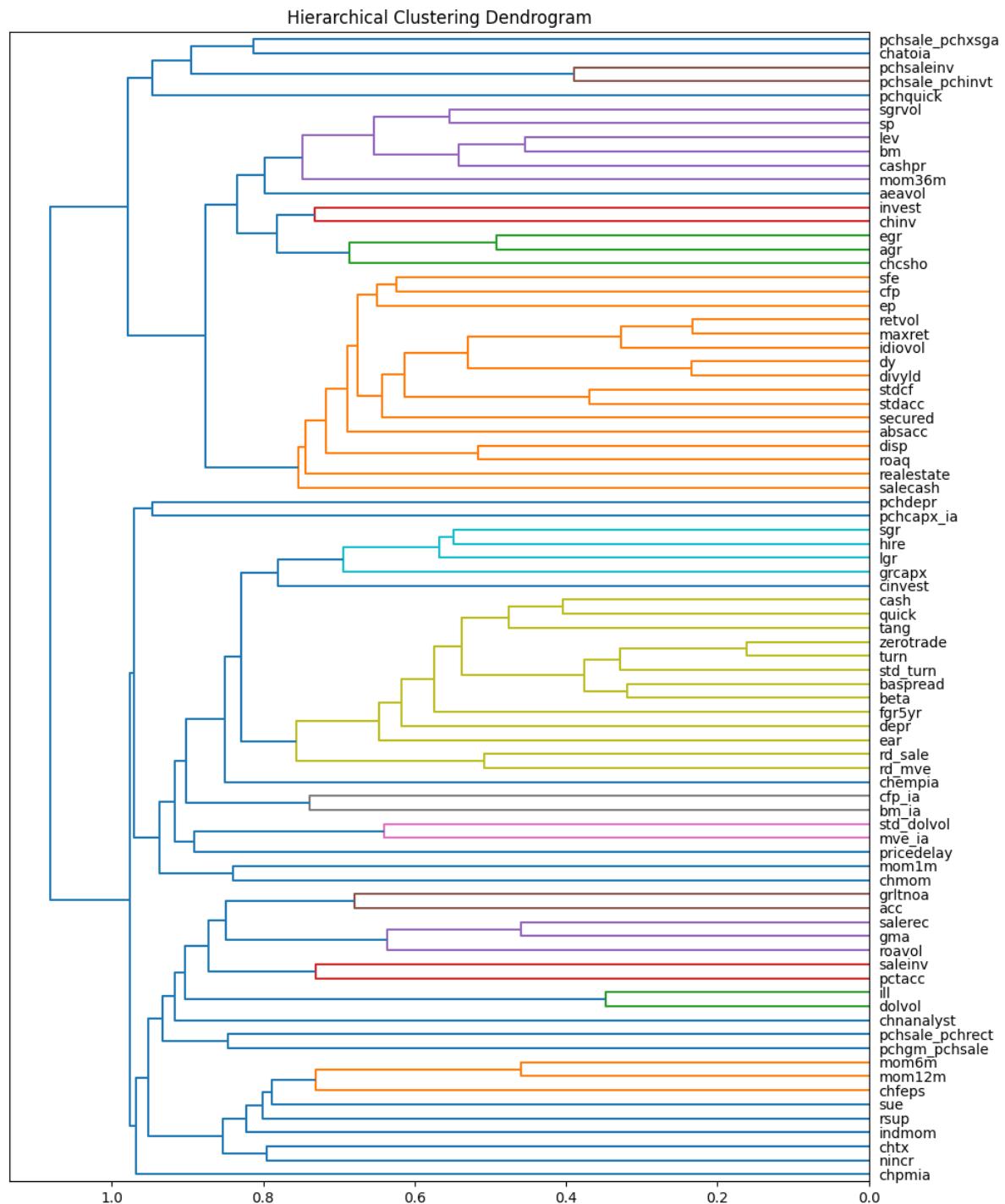
Hierarchical clustering builds a tree-like structure by iteratively merging clusters based on their similarity. Different linkage methods determine how clusters are merged:

- **Single Linkage:** Merges clusters based on the closest points, leading to elongated clusters.
- **Complete Linkage:** Uses the farthest points between clusters, producing compact, spherical clusters.
- **Average Linkage:** Merges clusters based on the average distance between all points, balancing between single and complete linkage.
- **Ward’s Method:** Minimizes within-cluster variance, resulting in clusters of similar size and shape.

```
model = AgglomerativeClustering(metric='precomputed', linkage='average',
                                 distance_threshold=0, n_clusters=None).fit(dist)
```

```
fig, ax = plt.subplots(figsize=(10, 12))
plt.title("Hierarchical Clustering Dendrogram")
# Create linkage matrix and then plot the dendrogram
# scikit-learn: "Plot Hierarchical Clustering Dendrogram"
counts = np.zeros(model.children_.shape[0])
n_samples = len(model.labels_)
# create the counts of samples under each node
for i, merge in enumerate(model.children_):
    current_count = 0
    for child_idx in merge:
        if child_idx < n_samples:
            current_count += 1 # leaf node
        else:
            current_count += counts[child_idx - n_samples]
    counts[i] = current_count

linkage_matrix = np.column_stack([model.children_, model.distances_, counts]) \
    .astype(float)
# Plot the corresponding dendrogram
dendrogram(linkage_matrix, orientation='left', labels=dist.columns, leaf_font_size=10)
plt.tight_layout()
```



6.2.2 K-means clustering

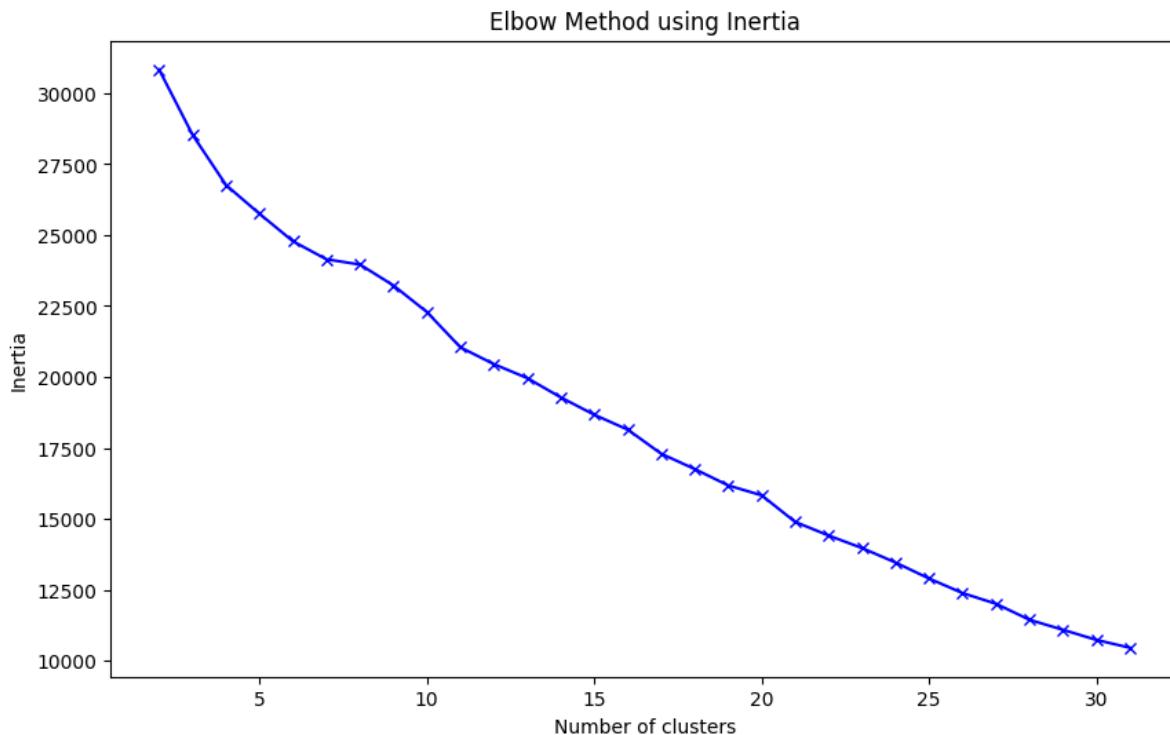
K-means is a widely used unsupervised learning algorithm that partitions data into K non-overlapping clusters. The process involves:

1. Selecting an initial number of clusters K .
2. Randomly assigning initial cluster centroids.
3. Iteratively assigning each data point to the nearest centroid.
4. Recalculating centroids based on the mean of points in each cluster.
5. Repeating until convergence.

K-means minimizes the within-cluster sum of squared distances, making it well-suited for datasets where clusters are roughly spherical and of similar size.

The Elbow Method is commonly used to determine the optimal number of clusters by plotting the within-cluster sum of squares for different values of K . The optimal K is where the graph forms an “elbow,” indicating diminishing returns from adding more clusters.

```
# Selecting number of centers with elbow method
inertias = []
n_clusters = list(range(2, 32))
for n_cluster in n_clusters:
    kmeans = KMeans(n_clusters=n_cluster, random_state=0, n_init="auto").fit(X.T)
    inertias.append(kmeans.inertia_)
plt.figure(figsize=(10, 6))
plt.plot(n_clusters, inertias, 'bx-')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.title('Elbow Method using Inertia')
plt.show()
```



Silhouette Analysis provides an alternative evaluation method by measuring how well-separated clusters are. The silhouette score compares cohesion (similarity within a cluster) to separation (difference from other clusters), with values ranging from -1 to 1. Higher scores indicate better-defined clusters, and the number of clusters maximizing the average silhouette score is considered optimal.

```
# scikit-learn: "Selecting the number of clusters with silhouette analysis on KMeans clustering"
scores = {}
fig, ax = plt.subplots(ncols=4, nrows=int(np.round(len(n_clusters)/4)), figsize=(10, 15))
ax = ax.flatten()
for n_cluster, ax1 in zip(n_clusters, ax):
    clusterer = KMeans(n_clusters=n_cluster, n_init='auto', random_state=10)
    cluster_labels = clusterer.fit_predict(X)

    # The silhouette_score gives the average value for all the samples.
    # This gives a perspective into the density and separation of the formed
    # clusters
    silhouette_avg = silhouette_score(X, cluster_labels)
    print("For n_clusters =", n_cluster,
          "The average silhouette_score is :", silhouette_avg)
    scores[n_cluster] = silhouette_avg
    # Compute the silhouette scores for each sample
    sample_silhouette_values = silhouette_samples(X, cluster_labels)

    ax1.set_title(f"{n_cluster} clusters: {silhouette_avg:.4f}")
    ax1.set_xlim([-0.1, .4])
    ax1.set_ylim([0, len(X) + (n_cluster + 1) * 10])

    y_lower = 10
    for i in range(n_cluster):
        # Aggregate the silhouette scores for samples belonging to
        # cluster i, and sort them
        ith_cluster_silhouette_values = sample_silhouette_values[cluster_labels == i]

        ith_cluster_silhouette_values.sort()

        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i

        color = cm.nipy_spectral(float(i) / n_cluster)
        ax1.fill_betweenx(
            np.arange(y_lower, y_upper),
            0,
            ith_cluster_silhouette_values,
            facecolor=color,
            edgecolor=color,
            alpha=0.7,
        )

        # Label the silhouette plots with their cluster numbers at the middle
        ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

        # Compute the new y_lower for next plot
        y_lower = y_upper + 10 # 10 for the 0 samples
    ax1.set_ylabel("Cluster label")
```

(continues on next page)

(continued from previous page)

```

# The vertical line for average silhouette score of all the values
ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

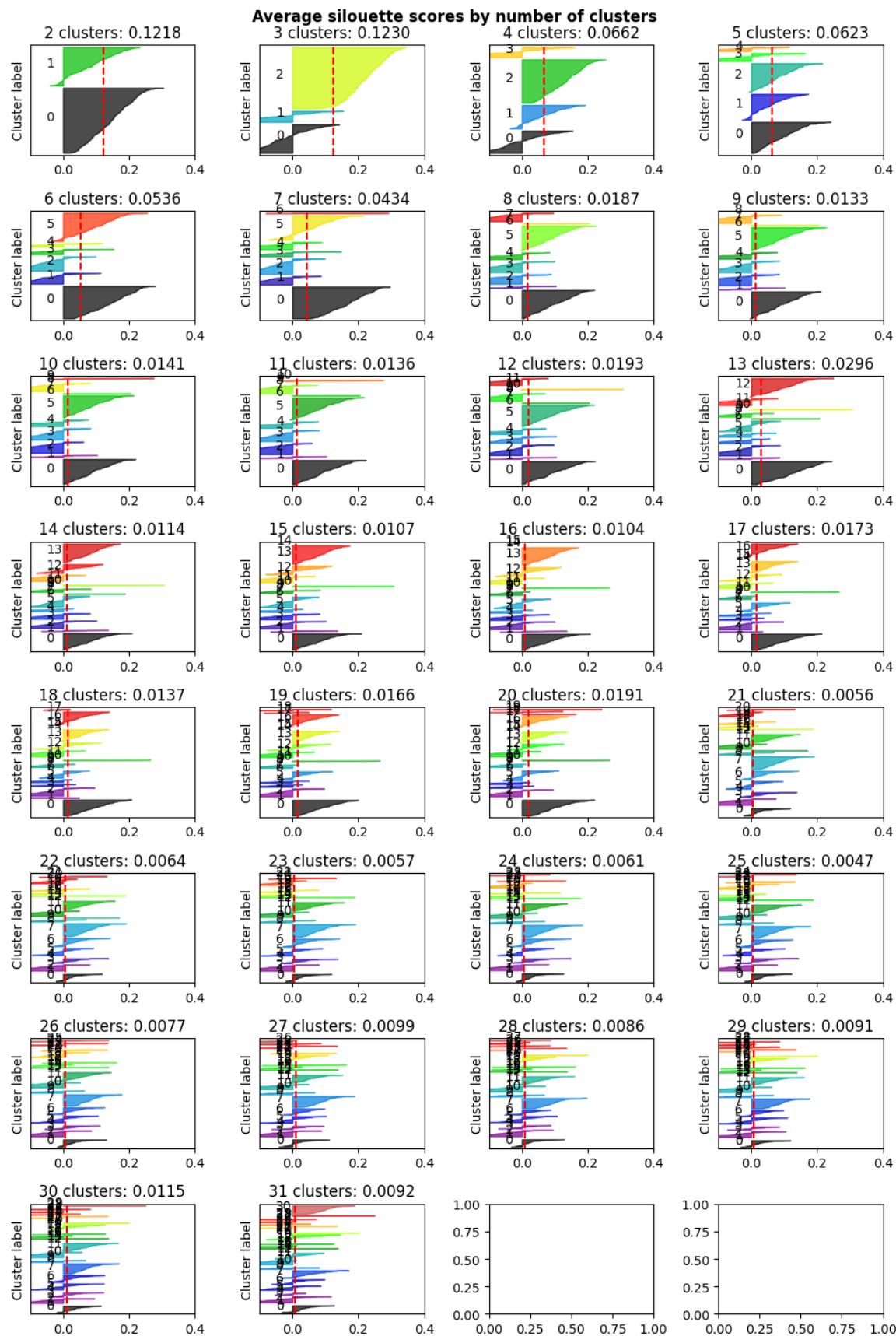
ax1.set_yticks([]) # Clear the yaxis labels / ticks
# ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])
plt.suptitle("Average silhouette scores by number of clusters", fontweight="bold")
plt.tight_layout()
plt.show()

```

```

For n_clusters = 2 The average silhouette_score is : 0.12182520066378914
For n_clusters = 3 The average silhouette_score is : 0.12298455415373394
For n_clusters = 4 The average silhouette_score is : 0.06615920430551211
For n_clusters = 5 The average silhouette_score is : 0.062263203571522374
For n_clusters = 6 The average silhouette_score is : 0.05360073591339586
For n_clusters = 7 The average silhouette_score is : 0.043399891992264766
For n_clusters = 8 The average silhouette_score is : 0.01865524677198136
For n_clusters = 9 The average silhouette_score is : 0.013274893057161832
For n_clusters = 10 The average silhouette_score is : 0.014055548382794032
For n_clusters = 11 The average silhouette_score is : 0.013559353746117705
For n_clusters = 12 The average silhouette_score is : 0.019298192911994215
For n_clusters = 13 The average silhouette_score is : 0.029625503519344714
For n_clusters = 14 The average silhouette_score is : 0.011439750949169072
For n_clusters = 15 The average silhouette_score is : 0.010659851615184082
For n_clusters = 16 The average silhouette_score is : 0.010407061841623768
For n_clusters = 17 The average silhouette_score is : 0.017270693194767258
For n_clusters = 18 The average silhouette_score is : 0.01369810719820829
For n_clusters = 19 The average silhouette_score is : 0.016630208862778856
For n_clusters = 20 The average silhouette_score is : 0.019144464436508506
For n_clusters = 21 The average silhouette_score is : 0.005561006651677671
For n_clusters = 22 The average silhouette_score is : 0.006354452315559386
For n_clusters = 23 The average silhouette_score is : 0.005722873193904617
For n_clusters = 24 The average silhouette_score is : 0.006108458004688574
For n_clusters = 25 The average silhouette_score is : 0.004737653906771348
For n_clusters = 26 The average silhouette_score is : 0.007702186499322102
For n_clusters = 27 The average silhouette_score is : 0.009876913034750879
For n_clusters = 28 The average silhouette_score is : 0.008628670013359775
For n_clusters = 29 The average silhouette_score is : 0.00908060151674441
For n_clusters = 30 The average silhouette_score is : 0.01149725260958716
For n_clusters = 31 The average silhouette_score is : 0.009230744450581462

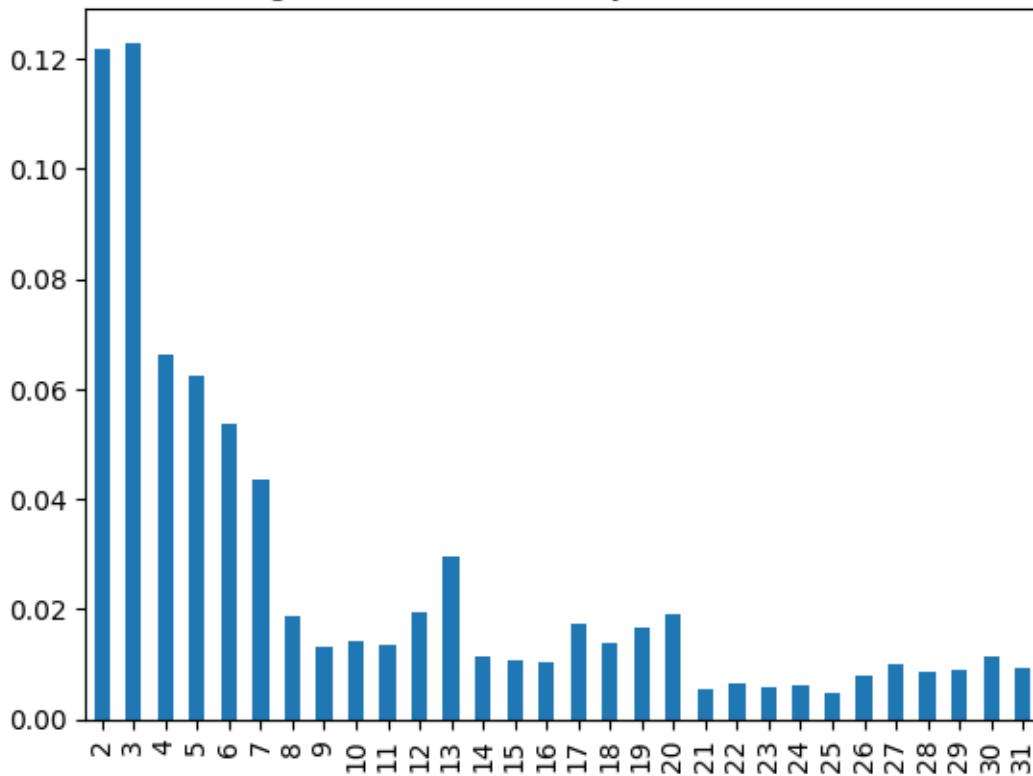
```



```
Series(scores).plot.bar(title='Average Silhouette Score by Number of Clusters')
```

```
<Axes: title={'center': 'Average Silhouette Score by Number of Clusters'}>
```

Average Silhouette Score by Number of Clusters

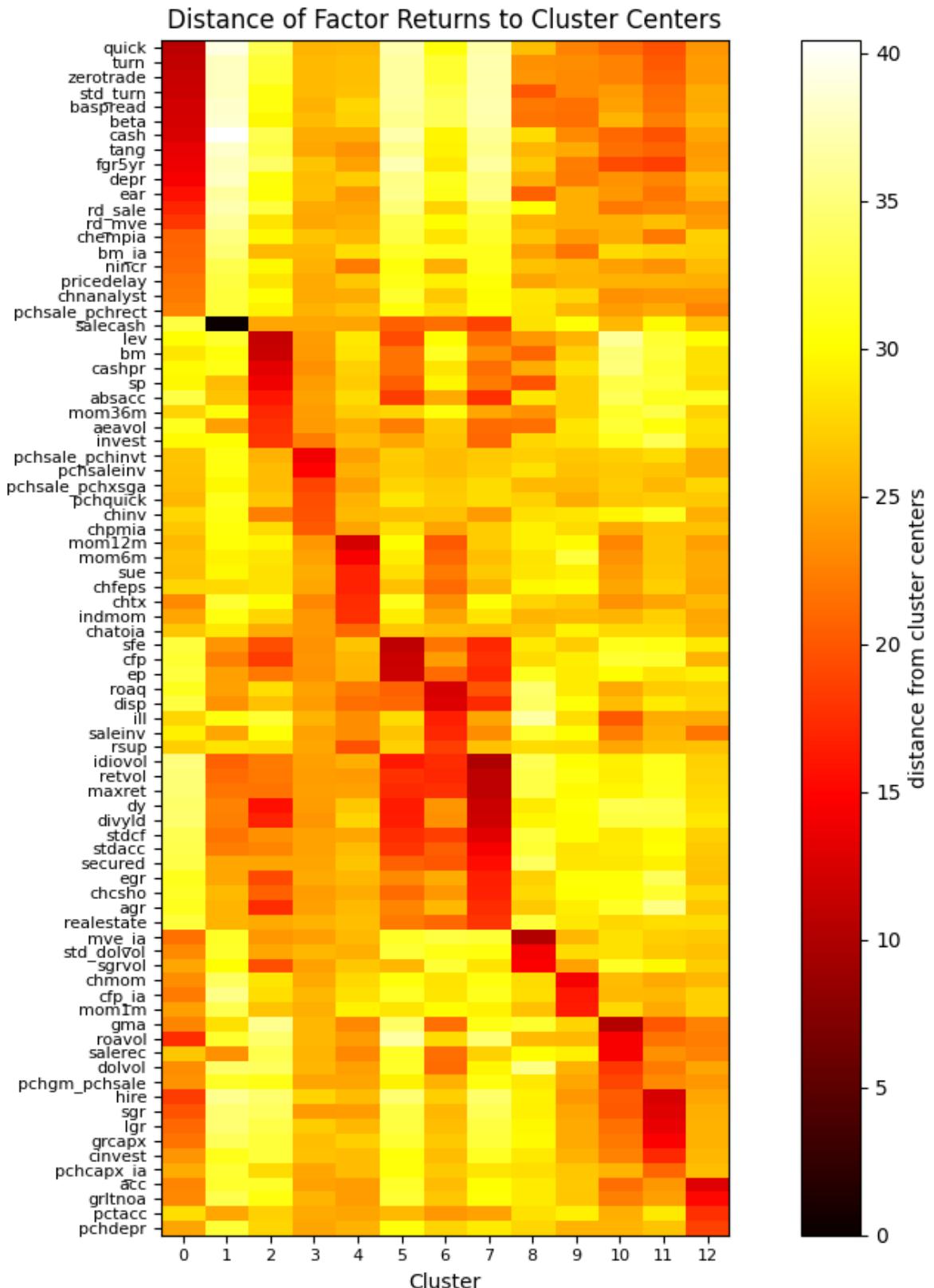


Heatmap of strategy correlations

A heatmap visualizes the distances between strategy returns (y-axis) and their respective cluster centers (x-axis), with clusters determined using the K-Means algorithm (K=20). The darkest-colored values appear along the jagged diagonal, indicating that strategy returns are closest to the centers of their assigned clusters. This visualization aids in identifying common style factors and risk exposures across different investment strategies.

```
kmeans = KMeans(n_clusters=13, random_state=0, n_init="auto").fit(X.T)
Z = DataFrame(kmeans.transform(X.T), index=X.columns)
Z['distance'] = np.min(Z.iloc[:, :kmeans.n_clusters], axis=1)
Z['cluster'] = np.argmin(Z.iloc[:, :kmeans.n_clusters], axis=1)
Z = Z.sort_values(['cluster', 'distance']) # group by cluster and distance to center
```

```
fig, ax = plt.subplots(figsize=(10, 9))
plt.title('Distance of Factor Returns to Cluster Centers')
plt.imshow(Z.iloc[:, :kmeans.n_clusters], cmap='hot', aspect=1/3)
plt.yticks(range(len(Z)), labels=Z.index, fontsize=8)
plt.xticks(range(kmeans.n_clusters), labels=range(kmeans.n_clusters), fontsize=8)
plt.xlabel('Cluster')
plt.colorbar(label='distance from cluster centers')
plt.tight_layout()
plt.show()
```



References:

Stephen Ross, 1976, “The arbitrage theory of capital asset pricing”, Journal of Economic Theory, Volume 13, Issue 3, December 1976, Pages 341-360

Robert C. Merton, 1973, “An Intertemporal Capital Asset Pricing Model”, Econometrica, Vol. 41, No. 5 (Sep., 1973), pp. 867-887

Jeremiah Green, John R. M. Hand, X. Frank Zhang, 2017, “The Characteristics that Provide Independent Information about Average U.S. Monthly Stock Returns”, The Review of Financial Studies, Volume 30, Issue 12, December 2017, Pages 4389–4436

John H. Cochrane, 2011, “Presidential Address: Discount Rates”, The Journal of Finance, Volume 66, Issue 4, August 2011, Pages 1047-1108

Lo, Andrew W. (2004). “The Adaptive Markets Hypothesis: Market Efficiency from an Evolutionary Perspective”. Journal of Portfolio Management. 5. 30: 15–29

Andrew Lo, 2017, “Adaptive Markets: Financial Evolution at the Speed of Thought”.

Andrew Ang, 2014, “Asset Management: A Systematic Approach to Factor Investing”

FRM Part II Exam Book Investment Manager Ch. 1-3.

EVENT STUDY

All strange and terrible events are welcome, but comforts we despise - Cleopatra

Event studies are a fundamental tool in financial research used to assess how specific events impact stock returns, see Mackinlay (1997). These events, such as mergers, earnings announcements, or regulatory changes, can cause deviations in stock prices known as abnormal returns. By comparing stock returns to a market benchmark, event studies help analysts determine whether an event had a statistically significant impact on a company's stock price.

This notebook explores key methodologies used in event studies, including the computation of abnormal returns, cumulative abnormal returns, and buy-and-hold abnormal returns. Additionally, it delves into the announcement effect, which examines how stock prices react immediately around an event, and investigates potential pre- and post-announcement drifts. To account for overlapping event windows and cross-sectional dependencies, we apply statistical corrections based on the work of Kolari et al. (2010, 2018).

We also introduce frequency domain methods, leveraging the Fourier Transform and Convolution Theorem to efficiently compute cross-correlations between time series. Finally, we address the multiple testing problem, ensuring that statistical inferences remain valid when analyzing a large number of events.

```
import numpy as np
import pandas as pd
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
from scipy.stats import norm
from statsmodels.stats.multitest import multipletests
from tqdm import tqdm
import warnings
from finds.database import SQL, RedisDB
from finds.structured import BusDay, Benchmarks, Stocks, CRSP, PSTAT
from secret import credentials, paths, CRSP_DATE
VERBOSE = 0
if not VERBOSE:
    warnings.simplefilter(action='ignore', category=FutureWarning)
#%matplotlib qt
```

```
# open connections
sql = SQL(**credentials['sql'], verbose=VERBOSE)
user = SQL(**credentials['user'], verbose=VERBOSE)
bd = BusDay(sql)
rdb = RedisDB(**credentials['redis'])
keydev = PSTAT(sql, bd, verbose=VERBOSE)
crsp = CRSP(sql, bd, rdb=rdb, verbose=VERBOSE)
bench = Benchmarks(sql, bd, verbose=VERBOSE)
imgdir = paths['images'] / 'events'
```

7.1 Abnormal returns

Event studies analyze how stock returns respond to specific company events, such as mergers or public announcements. These studies examine deviations from expected returns, known as **abnormal returns**, in the days surrounding the news release.

Let the event date be $t = 0$, and define the stock return for firm i at time t as $R_{i,t}$. To provide a benchmark, we also compute the returns of a relevant market or sector index, denoted $R_{m,t}$. Then:

- **Abnormal returns (AR):** The deviation of a stock's return from the market return: $AR_{i,t} = R_{i,t} - R_{m,t}$
- **Average abnormal returns (AAR):** The mean abnormal return across all firms: $AAR_t = \frac{1}{N} \sum_i^N AR_{i,t}$
- **Cumulative average abnormal returns (CAR):** The sum of average abnormal returns over time: $CAR_T = \sum_t AAR_t$
- **Buy-and-hold average abnormal returns (BHAR):** The average buy-and-hold abnormal returns across firms: $BHAR_T = \frac{1}{N} \sum_i^N [\prod_t (1 + AR_{i,t}) - \prod_t (1 + R_{m,t})]$

7.1.1 Announcement effect

The **announcement effect** refers to the immediate stock price movement around a news release. The nature of this reaction varies:

- An **efficient reaction** occurs when the stock price stabilizes at its new level after the event.
- An **under-reaction** happens when the price continues trending in the same direction post-announcement.
- An **over-reaction** is when the price initially spikes but then reverses course.

In historical data, pinpointing the exact announcement time can be challenging. To account for this uncertainty, studies often consider an **announcement window**, typically covering one day before and after the event.

- A **pre-announcement drift** may indicate information leakage before the official announcement.
- A **post-announcement drift** may suggest that investors either under- or over-reacted to the news.

When analyzing multiple events, researchers often aggregate returns for stocks with the same announcement date. However, Kolari et al. (2010) caution that even small correlations among abnormal returns can lead to significant over-rejection of the null hypothesis of zero abnormal returns. To correct for this cross-sectional dependence, the unbiased estimate of abnormal return variance is:

$$\sigma^2 = \frac{s^2}{n} (1 + (n - 1)\hat{r})$$

where:

- \hat{r} is the average sample cross-sectional correlation of residuals,
- s^2 is the (biased) cross-sectional standard deviation of abnormal returns.

In the next subsection, we explore how cross-correlations can be measured efficiently using the Fourier Transform technique from the field of signal processing.

7.1.2 FinDS eventstudy module

The `eventstudy` module in the FinDS package provides a comprehensive tool for analyzing announcement effects.

```
from finds.backtesting import EventStudy
eventstudy = EventStudy(user, bench=bench, stocks=crsp, max_date=CRSP_DATE)
```

```
# event window parameters
left, right, post = -1, 1, 21
end = bd.offset(CRSP_DATE, left - post)
beg = 20020101
mindays = 1000
```

- Retrieve event dates and company roles from CapitalIQ Key Developments

```
# sorted list of all eventids and roleids, provided in keydev class
events = sorted(keydev.event.index)
roles = sorted(keydev.role.index)
```

```
# str formatter to pretty print descriptions, provided in keydev class
eventformat = lambda e, r: "{event} ({eventid}) {role} [{roleid}]"\ \
    .format(event=keydev.event[e], eventid=e, role=keydev.role[r], roleid=r)
```

- Helper functions to retrieve size decile of universe stocks

```
class Universe:
    """Helper to lookup prevailing size decile of universe stocks"""
    def __init__(self, beg: int, end: int, stocks: Stocks, annual: bool = False):
        # whether to offset previous month or year end when lookup
        self.offset = stocks.bd.endyr if annual else stocks.bd.endmo

        # populate dictionary, keyed by permno and date, with size decile
        self.decile = dict()
        date_range = stocks.bd.date_range(start=self.offset(beg, -1), end=end,
                                           freq=12 if annual else 'e')
        for date in date_range:
            univ = crsp.get_universe(date)
            for permno in univ.index:
                self.decile[(permno, date)] = univ.loc[permno, 'decile']

    def __getitem__(self, item):
        """Returns size decile, else 0 if not universe stock"""
        permno, date = item
        return self.decile.get((permno, self.offset(date, -1)), 0)

univ = Universe(beg=beg, end=end, stocks=crsp)
```

- Define the event study pipeline:

1. retrieve announcement dates for the desired event and test period
2. filter the universe
3. retrieve announcement window daily stock returns, and compute event study metrics

```

# run event study after screening stock universe
def event_pipeline(eventstudy: EventStudy,
                   beg: int,
                   end: int,
                   eventid: int,
                   roleid: int,
                   left: int,
                   right: int,
                   post: int) -> DataFrame:
    """helper to screen stock universe, and merge keydev events and crsp daily"""

    # Retrieve announcement dates for this event
    df = keydev.get_linked(
        dataset='keydev',
        date_field='announcedate',
        fields=['keydevid',
                'keydeveventtypeid',
                'keydevtoobjectroletypeid'],
        where=(f"announcedate >= {beg} "
               f" and announcedate <= {end} "
               f" and keydeveventtypeid = {eventid} "
               f" and keydevtoobjectroletypeid = {roleid}"))\
            .drop_duplicates(['permno', 'announcedate'])\
            .set_index(['permno', 'announcedate'], drop=False)

    # Require universe size decile
    df['size'] = [univ[row.permno, row.announcedate] for row in df.itertuples()]

    # Call eventstudy to compute daily abnormal returns, with named label
    rows = eventstudy(label=f'{eventid}_{roleid}',
                       df=df[df['size'].gt(0)],
                       left=left,
                       right=right,
                       post=post,
                       date_field='announcedate')
    return df.loc[rows.to_records(index=False).tolist()] # restrict successful rows

```

Examine and show subsample plots for events that exhibited large post-announcement drift

```

events_list = [[93, 1], [232, 1]] # largest drift returns
for i, (eventid, roleid) in enumerate(events_list):
    df = event_pipeline(eventstudy,
                        eventid=eventid,
                        roleid=roleid,
                        beg=beg,
                        end=end,
                        left=left,
                        right=right,
                        post=post)
    halfperiod = np.median(df['announcedate'])
    sample = {'First Half': df['announcedate'].lt(halfperiod).values,
              'Second Half': df['announcedate'].ge(halfperiod).values,
              'Large': df['size'].le(5).values,
              'Small': df['size'].gt(5).values,
              'ALL': []}
    fig, axes = plt.subplots(nrows=2, ncols=2, clear=True, figsize=(12, 6),

```

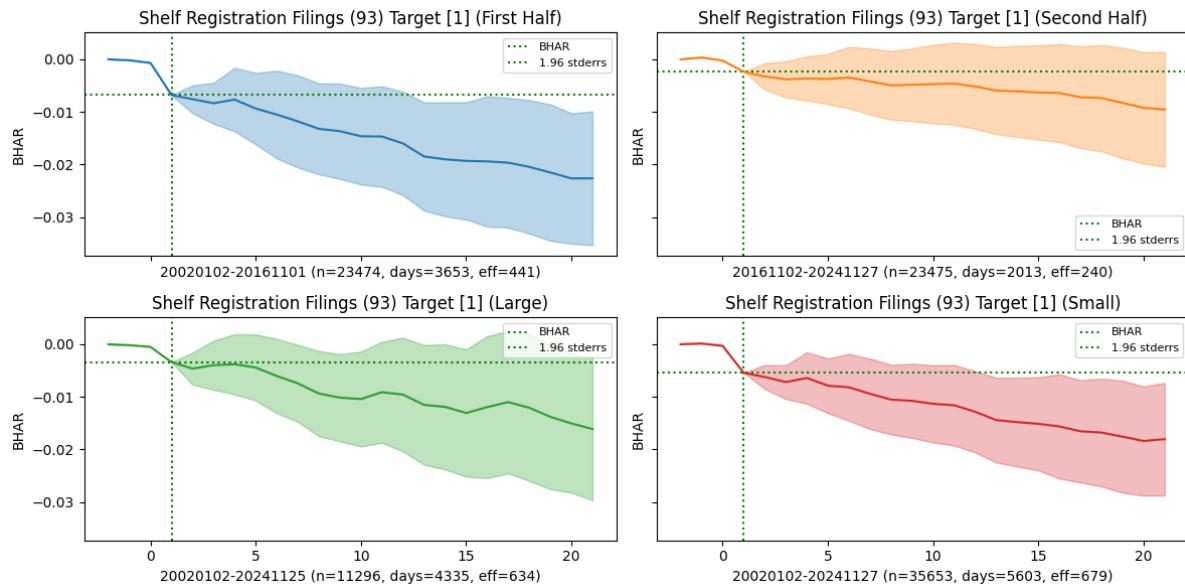
(continues on next page)

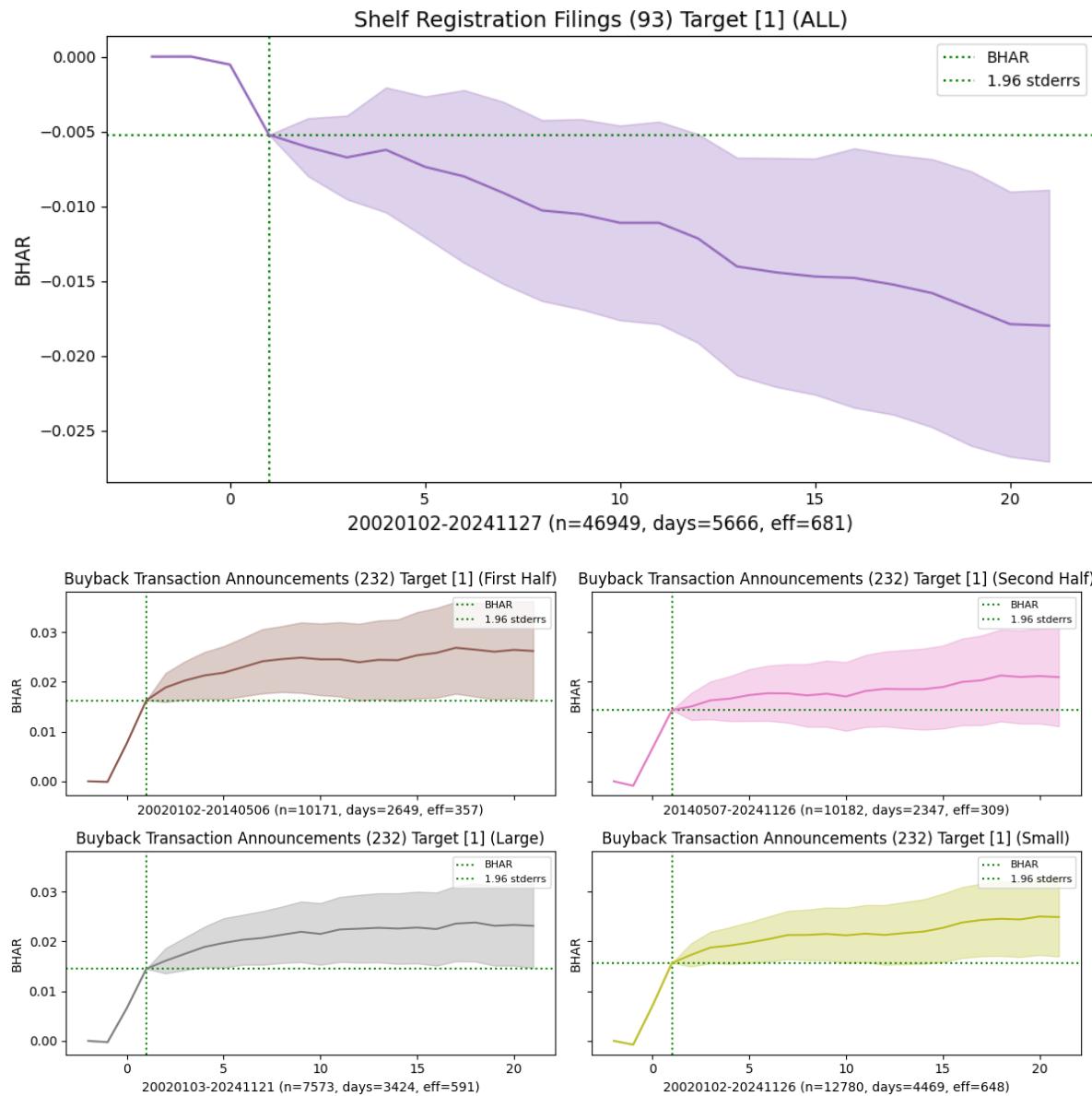
(continued from previous page)

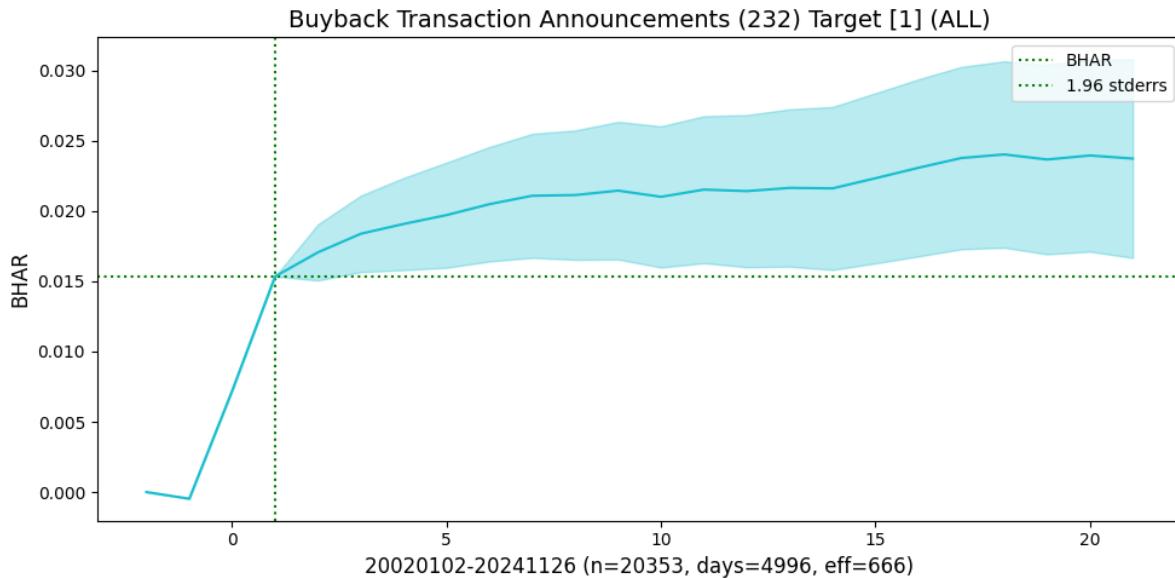
```

        sharex=True, sharey=True)
axes = axes.flatten()
for ifig, (label, rows) in enumerate(sample.items()):
    if ifig >= len(axes):
        plt.show()
        fig, ax = plt.subplots(clear = True, figsize=(10, 5))
    else:
        ax = axes[ifig]
bhar = eventstudy.fit(model='sbhar', rows=rows)
eventstudy.plot(model='sbhar',
                 title=eventformat(eventid, roleid) + f" ({label})",
                 drift=True,
                 ax=ax,
                 fontsize=8 if ifig < len(axes) else 10,
                 c=f"C{i*5+ifig}")
plt.tight_layout()
plt.savefig(imgdir / (label + f"{eventid}_{roleid}.jpg"))
plt.show()

```







Examine and show subsample plots for events that exhibited large announcement window returns

```

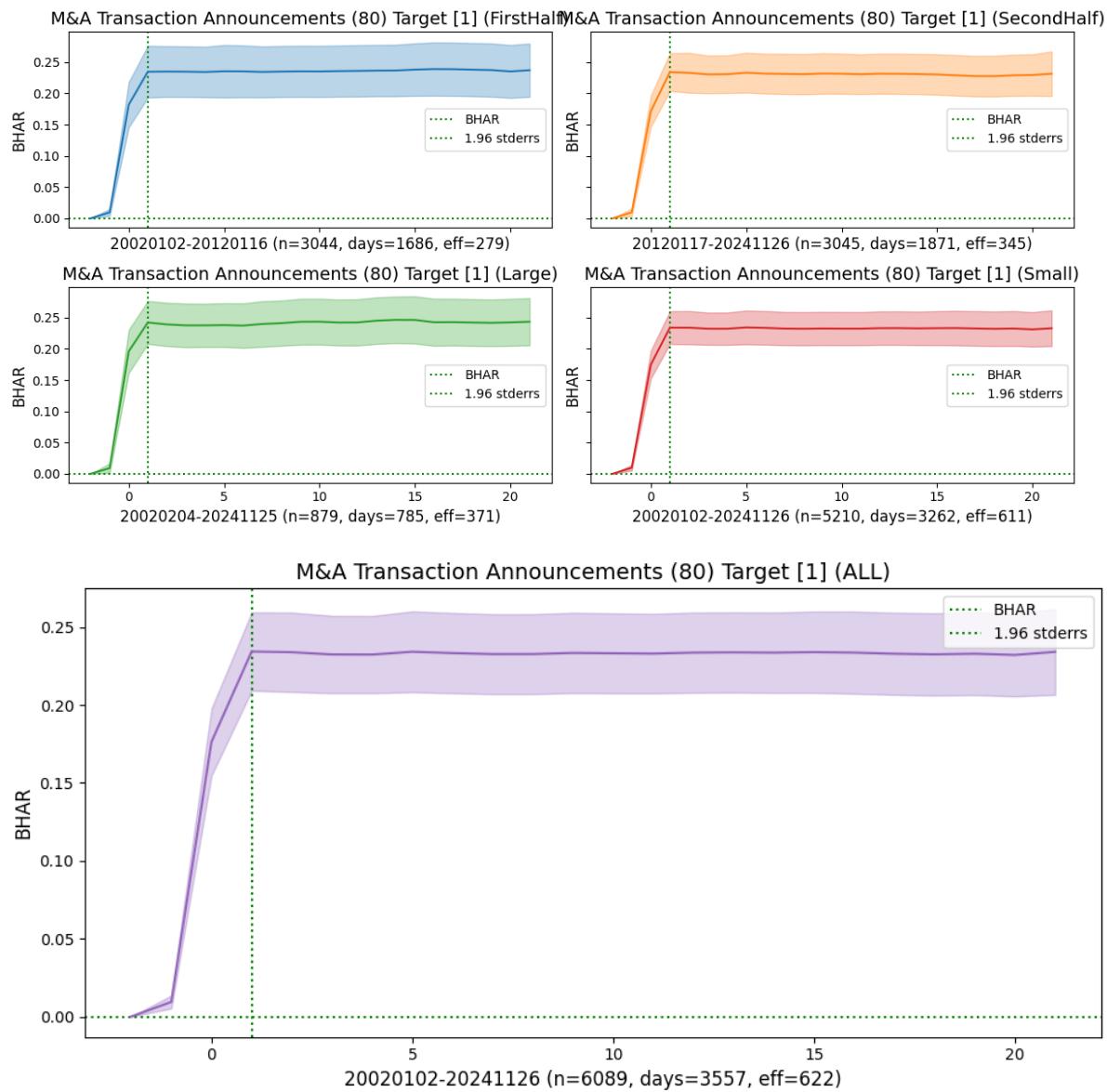
events_list = [[80, 1], [26, 1]] # largest announcement window returns
for i, (eventid, roleid) in enumerate(events_list):
    #eventid, roleid = 50, 1
    #eventid, roleid = 83, 1
    df = event_pipeline(eventstudy,
                        eventid=eventid,
                        roleid=roleid,
                        beg=beg,
                        end=end,
                        left=left,
                        right=right,
                        post=post)
    halfperiod = np.median(df['announcedate'])
    sample = {'FirstHalf': df['announcedate'].lt(halfperiod).values,
              'SecondHalf': df['announcedate'].ge(halfperiod).values,
              'Large': df['size'].le(5).values,
              'Small': df['size'].gt(5).values,
              'ALL': []}
    fig, axes = plt.subplots(nrows=2, ncols=2, clear=True, figsize=(12, 6),
                            sharex=True, sharey=True)
    axes = axes.flatten()
    for ifig, (label, rows) in enumerate(sample.items()):
        if ifig >= len(axes):
            plt.show()
            fig, ax = plt.subplots(clear = True, figsize=(10, 5))
        else:
            ax = axes[ifig]
        bhar = eventstudy.fit(model='sbhar', rows=rows)
        eventstudy.plot(model='sbhar',
                        title=eventformat(eventid, roleid) + f" ({label})",
                        drift=False,
                        ax=ax,
                        c=f"C{i*5+ifig}")
    plt.tight_layout()
    plt.savefig(imgdir / (label + f" {eventid}_{roleid}.jpg"))

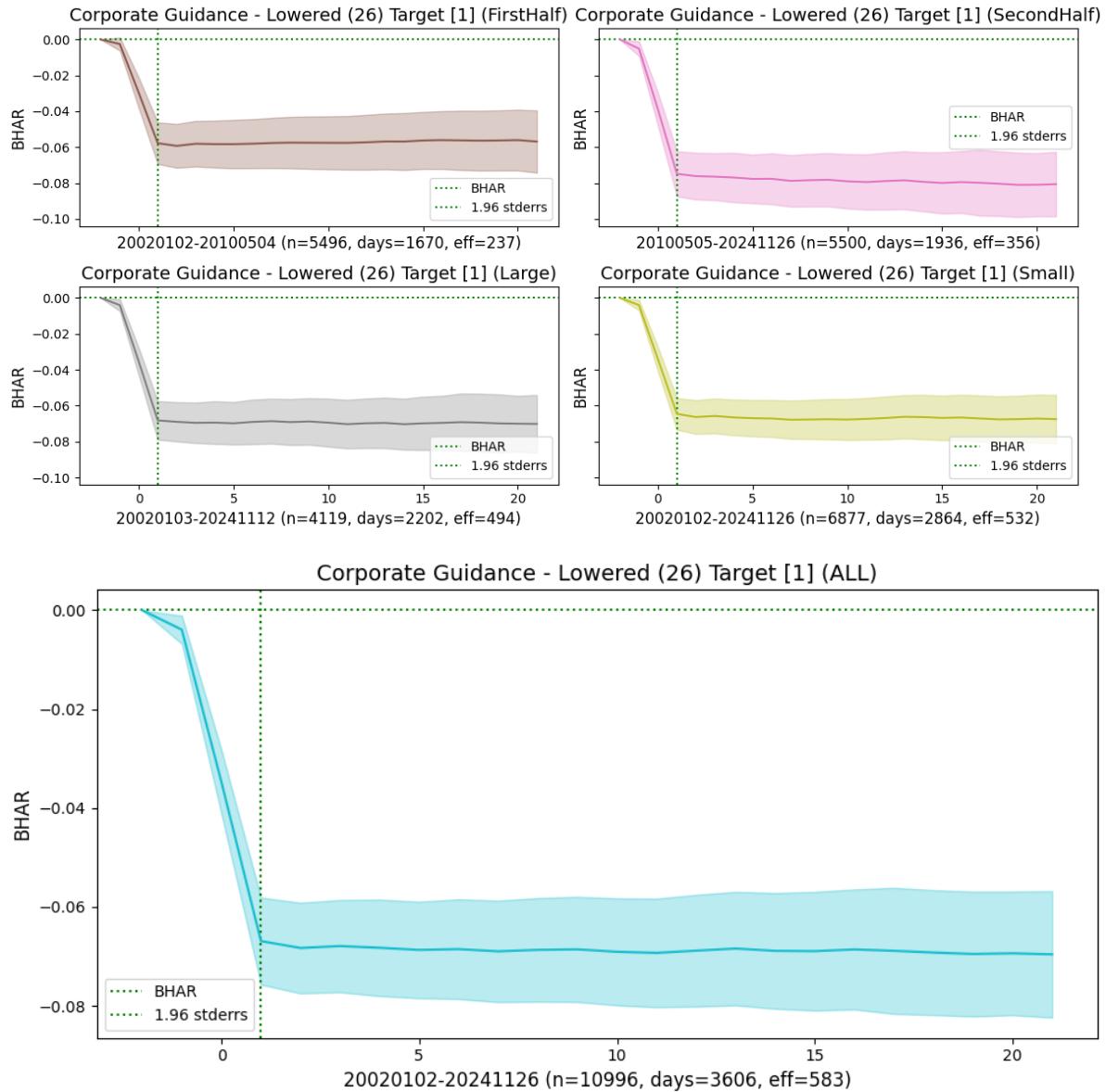
```

(continues on next page)

(continued from previous page)

```
plt.show()
```





7.2 Fourier transforms

7.2.1 Cross-sectional Correlations

Kolari et al. (2018) address event windows that **partially overlap in calendar time**. To account for this overlap and cross-sectional correlation effects, they adjust the variance estimate as follows:

$$\sigma^2 = \frac{s^2}{n} m (1 + \tau(n-1)\rho)$$

where:

- m = length of the event window,
- τ = proportion of overlapping days between event windows,

- ρ = ratio of average covariance between abnormal returns to their variance.

If all events share the same calendar date, such that all event windows fully overlap, then ρ simplifies to the average cross-sectional correlation of residuals during the estimation period.

To estimate ρ , we assume cross-sectional correlations arise mainly from time misalignment. By shifting and aligning event window residuals for each stock pair to maximize their correlation, we approximate ρ as the average of these best-aligned correlations. We leverage the methodology of Fourier transforms and convolutions to efficiently compute all pairs of correlations.

7.2.2 Fast Fourier Transforms

Fourier transforms allow us to express a time series (or any function) as a sum of sinusoids. Given a function over an interval of length (L), we can decompose it into sinusoidal components with periods ($L, L/2, L/3, \dots$) etc. Each component has:

- A magnitude (scale factor)
- A phase (shift)
- A frequency (inverse of the period)

This means that any function can be represented as a weighted sum of sinusoidal waves. By using complex numbers, we can efficiently store and manipulate these sinusoidal components in Fourier space rather than the original time domain. The **Fast Fourier Transform (FFT)** is an efficient algorithm for computing the **Discrete Fourier Transform (DFT)**. The DFT converts a sequence of time-domain data points into its sinusoidal coefficients, converting it from real space to frequency space. The **Inverse Fast Fourier Transform** reconstructs the original function.

7.2.3 Convolution Theorem

Convolution is a mathematical operation that combines two functions (f and g) to produce a third function ($f * g$), expressing how one function modifies the other:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(s)g(t-s)ds$$

In **one dimension**, convolution acts like a **moving average**, smoothing a time series by weighting neighboring values. In **two dimensions**, it applies filters (e.g., blurring, edge detection) in image processing.

In signal processing, frequency-domain techniques efficiently compute cross-correlations across various time shifts. Instead of directly computing correlations for every possible lag (which is computationally expensive), we leverage the **Convolution Theorem**. A key property of Fourier transforms is that convolution in the time domain corresponds to multiplication in the frequency domain.

$$\mathcal{F}\{f(t) * g(t)\} = \mathcal{F}\{f(t)\}\mathcal{F}\{g(t)\}$$

Since direct convolution requires $O(N^2)$ operations, computing it directly is slow. However:

1. We can first apply the Fast Fourier Transform (FFT) to convert both functions into frequency space in $O(N \log N)$ time.
2. In frequency space, convolution is just element-wise multiplication of Fourier coefficients, which is very fast.
3. We then apply the Inverse FFT (IFFT) to obtain the final result in $O(N \log N)$ time.

```

# best alignment and cross-correlation using convolution theorem method
from scipy.fft import rfft, irfft
def fft_align(X):
    """Find best alignment, max cross-correlation and indices of all pairs of columns"""
    """

    def _normalize(X: np.ndarray) -> np.ndarray:
        """Helper to demean columns and divide by norm"""
        X = X - np.mean(X, axis=0)
        X = X / np.linalg.norm(X, axis=0)
        return X

    N, M = X.shape
    X = np.pad(_normalize(X), [(0, N), (0, 0)]) # normalize and zero pad
    Y = rfft(np.flipud(X), axis=0) # FFT of all series flipped
    X = rfft(X, axis=0) # FFT of all original series
    corr, disp, cols = [], [], [] # to accumulate results
    for col in range(M-1): # at each iter: compute column col * all remaining columns
        conv = irfft(X[:, [col]] * Y[:, col+1:], axis=0) # inverse of product of FFT
        corr.append(np.max(conv, axis=0))
        shift = (N//2) + 1 # displacement location relative to center
        disp.append((np.argmax(conv, axis=0) + shift) % N - shift + 1)
        cols.append([(col, j) for j in range(col+1, M)])
    return corr, disp, cols

```

```

Z = np.random.uniform(size=(10000, 1))
fft_align(np.hstack((Z[:-1], Z[1:])))

```

```

([0.9998967076018761], [1], [(0, 1)])

```

```

#fft_align(np.hstack(np.hstack((Z[:-5], Z[5:]))[:,2:], Z[:-2]))
fft_align(np.hstack((np.hstack((Z[:-5], Z[5:]))[:-2], Z[7:])))

```

```

([0.9997503757740265, 0.9993943885689067, 0.9996440440611477],
 [5, 7, 2],
 [(0, 1), (0, 2), (1, 2)])

```

7.3 Multiple testing

When testing a huge number of null hypotheses, we are bound to get some very testing small p-values by chance. **Data snooping** occurs when the analyst fits a great number of models to a dataset. When explanatory variables are selected using the data, t-ratios and F-ratios will be too large, thus overstating the importance of variables in the model. **Multiple testing** adjusts the hurdle for significance because some tests will appear significant by chance. The downside of doing this is that some truly significant strategies might be overlooked because they did not pass the more stringent hurdle. This is the classic tension between Type I errors and Type II errors. The Type I error is the false discovery (investing in an unprofitable trading strategy). The Type II error is missing a truly profitable trading strategy.

The **Family-Wise Error Rate (FWER)** controls the probability of making **at least one** false discovery. If all m null hypotheses are independent and true:

$$\text{FWER}(\alpha) = 1 - (1 - \alpha)^m$$

To maintain a stricter significance threshold, the **Bonferroni correction** divides the confidence level by the number of tests:

$$\alpha' = \frac{\alpha}{m}$$

This ensures that across multiple tests, the probability of making even **one** false discovery remains below α . However, it can be overly conservative.

The **false discovery rate (FDR)** controls the proportion of false positives among rejected hypotheses. The **Benjamini-Hochberg (BH) procedure** is a widely used method for FDR control. Instead of adjusting individual significance thresholds like Bonferroni, BH ranks p-values and sets a significance cutoff where false discoveries remain within acceptable limits.

Compute BHAR and CAR of all events

```
restart_event = 0 # 75,1
for i, eventid in tqdm(enumerate(events), total=len(events)):
    if eventid <= restart_event: # kludge to resume loop
        continue
    for roleid in roles:
        # retrieve all returns observations of this eventid, roleid
        df = event_pipeline(eventstudy,
                            beg=beg,
                            end=end,
                            eventid=eventid,
                            roleid=roleid,
                            left=left,
                            right=right,
                            post=post)
        if df['announcedate'].nunique() < mindays: # require min number of dates
            continue

        # compute both BHAR and CAR averages, then plot and save
        bhar = eventstudy.fit(model='sbhar')
        car = eventstudy.fit(model='scar')
        #eventstudy.write()
        eventstudy.write_summary()
        #print(eventstudy.label, eventid, roleid)

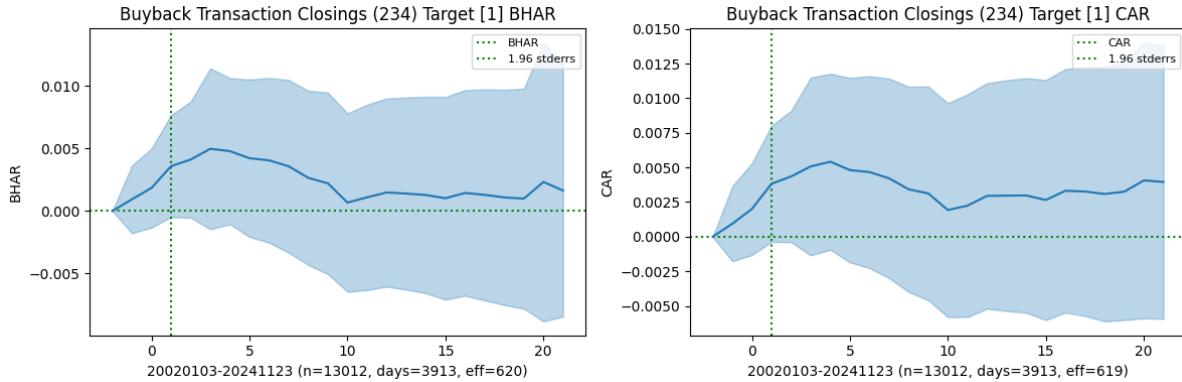
        fig, axes = plt.subplots(1, 2, clear=True, figsize=(12, 4), num=1)
        eventstudy.plot(model='sbhar', ax=axes[0],
                        title=eventformat(eventid, roleid) + ' BHAR',
                        fontsize=8, vline=[right])
        eventstudy.plot(model='scar', ax=axes[1],
                        title=eventformat(eventid, roleid) + ' CAR',
                        fontsize=8, vline=[right])
        plt.tight_layout()
        plt.savefig(imgdir / f"{eventid}_{roleid}.jpg")
```

```
45%|██████████| 52/116 [20:04:50<22:46:35, 1281.17s/it]/home/terence/Dropbox/
github/data-science-notebooks/finds/recipes/filters.py:28: RuntimeWarning:_
invalid value encountered in divide
X = X / np.linalg.norm(X, axis=0)
53%|██████████| 62/116 [23:27:18<18:41:48, 1246.45s/it]/home/terence/Dropbox/
github/data-science-notebooks/finds/recipes/filters.py:28: RuntimeWarning:_
invalid value encountered in divide
X = X / np.linalg.norm(X, axis=0)
```

(continues on next page)

(continued from previous page)

```
61%|██████████| 71/116 [26:38:56<16:18:29, 1304.66s/it]/home/terence/Dropbox/
  ↵github/data-science-notebooks/finds/recipes/filters.py:28: RuntimeWarning:_
  ↵invalid value encountered in divide
  X = X / np.linalg.norm(X, axis=0)
100%|██████████| 116/116 [42:41:25<00:00, 1324.87s/it]
```



Summarize BHAR's of all events

```
# Create (eventid, roleid) multiindex for table of all BHAR's
df = eventstudy.read_summary(model='sbhar') \
    .set_index('label') \
    .drop(columns=['rho', 'tau', 'created'])
df = df[df['effective'] > 400].sort_values('post_t') # sorted by post drift t-stat
multiIndex = DataFrame(df.index.str.split('_').to_list()).astype(int)
df.index = pd.MultiIndex.from_frame(multiIndex, names=['eventid', 'roleid'])
df['event'] = keydev.event[df.index.get_level_values(0)].values
df['role'] = keydev.role[df.index.get_level_values(1)].values
print("Post-Announcement Drift")
pd.set_option('display.max_rows', None)
df.set_index(['event', 'role']).drop(columns=['model'])
```

Post-Announcement Drift

event	role	beg	end	\
Shelf Registration Filings	Target	20020102	20241127	
Special/Extraordinary Shareholders Meeting	Target	20040106	20241127	
Name Changes	Target	20020430	20241121	
Changes in Company Bylaws/Rules	Target	20020430	20241127	
Product-Related Announcements	Target	20020102	20241127	
M&A Transaction Closings	Target	20020108	20241125	
Auditor Changes	Target	20020103	20241127	
Executive/Board Changes - Other	Target	20020101	20241127	
Executive Changes - CFO	Target	20020102	20241127	
Business Expansions	Target	20020102	20241127	
Special Calls	Target	20030211	20241127	
Executive Changes - CEO	Target	20020102	20241127	
M&A Transaction Cancellations	Buyer	20020107	20241122	
Client Announcements	Target	20020102	20241127	
Follow-on Equity Offerings	Target	20020102	20241127	

(continues on next page)

(continued from previous page)

Investor Activism - Target Communication	Target	20020402	20241127
Earnings Calls	Target	20020108	20241127
Strategic Alliances	Target	20020102	20241127
Lawsuits & Legal Issues	Target	20020101	20241127
Investor Activism - Activist Communication	Target	20020102	20241126
Corporate Guidance - Lowered	Target	20020102	20241126
Private Placements	Buyer	20020103	20241127
M&A Transaction Announcements	Seller	20020101	20241127
M&A Transaction Closings	Buyer	20020101	20241127
Company Conference Presentations	Target	20041101	20241127
M&A Calls	Target	20031027	20241125
Index Constituent Adds	Target	20020102	20241127
Buyback Transaction Closings	Target	20020103	20241123
M&A Rumors and Discussions	Target	20020918	20241127
Shareholder/Analyst Calls	Target	20020108	20241127
Labor-related Announcements	Target	20020111	20241121
Business Reorganizations	Target	20020102	20241122
Conferences	Participant	20070125	20241127
M&A Transaction Announcements	Buyer	20020101	20241127
Delayed SEC Filings	Target	20020102	20241127
Earnings Release Date	Target	20020411	20241127
M&A Transaction Closings	Seller	20020101	20241127
M&A Transaction Announcements	Target	20020102	20241126
Discontinued Operations/Downsizings	Target	20020102	20241127
Fixed Income Offerings	Target	20020101	20241127
Corporate Guidance - New/Confirmed	Target	20020101	20241127
Private Placements	Target	20020103	20241127
Seeking Acquisitions/Investments	Target	20020103	20241125
Board Meeting	Target	20020204	20241120
Considering Multiple Strategic Alternatives	Target	20040324	20241122
Announcements of Sales/Trading Statement	Target	20020102	20241119
Debt Financing Related	Target	20020102	20241127
Annual General Meeting	Target	20020122	20241127
Seeking to Sell/Divest	Target	20020110	20241125
End of Lock-Up Period	Target	20020115	20241126
Buyback Tranche Update	Target	20020107	20241127
Announcements of Earnings	Target	20020101	20241127
Analyst/Investor Day	Target	20040204	20241126
Investor Activism - Proxy/Voting Related	Target	20020107	20241126
Delistings	Target	20020102	20241127
Impairments/Write Offs	Target	20020408	20241126
Corporate Guidance - Raised	Target	20020103	20241126
Index Constituent Drops	Target	20020107	20241126
Buyback - Change in Plan Terms	Target	20020102	20241126
Buyback Transaction Announcements	Target	20020102	20241126

event	role	rows	days	\
Shelf Registration Filings	Target	46949	5666	
Special/Extraordinary Shareholders Meeting	Target	6183	3353	
Name Changes	Target	1563	1329	
Changes in Company Bylaws/Rules	Target	27834	4825	
Product-Related Announcements	Target	209897	5767	
M&A Transaction Closings	Target	1508	1234	
Auditor Changes	Target	9563	3888	
Executive/Board Changes - Other	Target	222258	5766	

(continues on next page)

(continued from previous page)

			effective	window	\
event	role				
Shelf Registration Filings	Target		681.0	-0.005224	
Executive Changes - CFO	Target	22715	5544		
Business Expansions	Target	58224	5709		
Special Calls	Target	16738	4501		
Executive Changes - CEO	Target	17079	5269		
M&A Transaction Cancellations	Buyer	1508	1258		
Client Announcements	Target	212795	5761		
Follow-on Equity Offerings	Target	21917	5219		
Investor Activism - Target Communication	Target	5248	3059		
Earnings Calls	Target	236084	5442		
Strategic Alliances	Target	29517	5463		
Lawsuits & Legal Issues	Target	41158	5599		
Investor Activism - Activist Communication	Target	8556	3875		
Corporate Guidance - Lowered	Target	10996	3606		
Private Placements	Buyer	11631	4362		
M&A Transaction Announcements	Seller	12325	4804		
M&A Transaction Closings	Buyer	43177	5699		
Company Conference Presentations	Target	339716	4975		
M&A Calls	Target	7218	3305		
Index Constituent Adds	Target	29372	2248		
Buyback Transaction Closings	Target	13012	3913		
M&A Rumors and Discussions	Target	21747	4605		
Shareholder/Analyst Calls	Target	24420	4417		
Labor-related Announcements	Target	3975	2485		
Business Reorganizations	Target	4398	2766		
Conferences	Participant	296636	3676		
M&A Transaction Announcements	Buyer	25352	5501		
Delayed SEC Filings	Target	13297	2078		
Earnings Release Date	Target	226653	5294		
M&A Transaction Closings	Seller	18467	5138		
M&A Transaction Announcements	Target	6089	3557		
Discontinued Operations/Downsizings	Target	18782	4447		
Fixed Income Offerings	Target	28237	5403		
Corporate Guidance - New/Confirmed	Target	148835	5649		
Private Placements	Target	15216	5057		
Seeking Acquisitions/Investments	Target	37787	5239		
Board Meeting	Target	6563	2705		
Considering Multiple Strategic Alternatives	Target	4017	2491		
Announcements of Sales/Trading Statement	Target	12367	3306		
Debt Financing Related	Target	41158	5708		
Annual General Meeting	Target	71528	5094		
Seeking to Sell/Divest	Target	5250	2922		
End of Lock-Up Period	Target	7107	3293		
Buyback Tranche Update	Target	113233	5073		
Announcements of Earnings	Target	343641	5712		
Analyst/Investor Day	Target	6664	3353		
Investor Activism - Proxy/Voting Related	Target	7449	3165		
Delistings	Target	17906	4928		
Impairments/Write Offs	Target	23529	3527		
Corporate Guidance - Raised	Target	20044	4162		
Index Constituent Drops	Target	18202	3039		
Buyback - Change in Plan Terms	Target	6897	3412		
Buyback Transaction Announcements	Target	20353	4996		

(continues on next page)

(continued from previous page)

Special/Extraordinary Shareholders Meeting	Target	534.0	-0.002029
Name Changes	Target	454.0	0.012626
Changes in Company Bylaws/Rules	Target	592.0	-0.000070
Product-Related Announcements	Target	686.0	0.005712
M&A Transaction Closings	Target	430.0	0.010726
Auditor Changes	Target	609.0	-0.003402
Executive/Board Changes - Other	Target	685.0	-0.000533
Executive Changes - CFO	Target	677.0	-0.007618
Business Expansions	Target	688.0	0.002006
Special Calls	Target	597.0	0.014764
Executive Changes - CEO	Target	671.0	-0.005253
M&A Transaction Cancellations	Buyer	455.0	-0.000106
Client Announcements	Target	687.0	0.008237
Follow-on Equity Offerings	Target	667.0	-0.030227
Investor Activism - Target Communication	Target	525.0	0.008690
Earnings Calls	Target	654.0	0.000499
Strategic Alliances	Target	679.0	0.010685
Lawsuits & Legal Issues	Target	682.0	-0.000897
Investor Activism - Activist Communication	Target	576.0	0.014757
Corporate Guidance - Lowered	Target	583.0	-0.066909
Private Placements	Buyer	645.0	0.000440
M&A Transaction Announcements	Seller	662.0	0.009878
M&A Transaction Closings	Buyer	687.0	0.004541
Company Conference Presentations	Target	592.0	0.001272
M&A Calls	Target	566.0	0.059589
Index Constituent Adds	Target	517.0	-0.004618
Buyback Transaction Closings	Target	620.0	0.003593
M&A Rumors and Discussions	Target	584.0	0.013315
Shareholder/Analyst Calls	Target	585.0	0.001265
Labor-related Announcements	Target	485.0	0.001723
Business Reorganizations	Target	519.0	-0.001773
Conferences	Participant	451.0	-0.000775
M&A Transaction Announcements	Buyer	682.0	0.010533
Delayed SEC Filings	Target	519.0	-0.017189
Earnings Release Date	Target	632.0	-0.000074
M&A Transaction Closings	Seller	672.0	0.003516
M&A Transaction Announcements	Target	622.0	0.234380
Discontinued Operations/Downsizing	Target	594.0	-0.009840
Fixed Income Offerings	Target	677.0	-0.004932
Corporate Guidance - New/Confirmed	Target	687.0	-0.001445
Private Placements	Target	657.0	0.023461
Seeking Acquisitions/Investments	Target	668.0	-0.000697
Board Meeting	Target	473.0	0.002824
Considering Multiple Strategic Alternatives	Target	479.0	-0.008435
Announcements of Sales/Trading Statement	Target	588.0	0.004235
Debt Financing Related	Target	684.0	0.004347
Annual General Meeting	Target	620.0	0.001460
Seeking to Sell/Divest	Target	504.0	-0.002623
End of Lock-Up Period	Target	565.0	-0.060075
Buyback Tranche Update	Target	646.0	0.000906
Announcements of Earnings	Target	685.0	0.000446
Analyst/Investor Day	Target	548.0	0.008374
Investor Activism - Proxy/Voting Related	Target	530.0	-0.001100
Delistings	Target	625.0	-0.020833
Impairments/Write Offs	Target	521.0	-0.003683
Corporate Guidance - Raised	Target	631.0	0.033832

(continues on next page)

(continued from previous page)

Index Constituent Drops	Target	507.0	-0.003266	
Buyback - Change in Plan Terms	Target	605.0	0.009150	
Buyback Transaction Announcements	Target	666.0	0.015343	
		window_t	post	\
event	role			
Shelf Registration Filings	Target	-2.542320	-0.012755	
Special/Extraordinary Shareholders Meeting	Target	-0.537886	-0.018210	
Name Changes	Target	1.944510	-0.023076	
Changes in Company Bylaws/Rules	Target	-0.024033	-0.010242	
Product-Related Announcements	Target	4.608120	-0.004112	
M&A Transaction Closings	Target	1.184190	-0.019140	
Auditor Changes	Target	-1.173250	-0.009535	
Executive/Board Changes - Other	Target	-0.537347	-0.003171	
Executive Changes - CFO	Target	-3.312440	-0.006563	
Business Expansions	Target	1.663570	-0.003755	
Special Calls	Target	1.100980	-0.014743	
Executive Changes - CEO	Target	-1.572760	-0.007032	
M&A Transaction Cancellations	Buyer	-0.019428	-0.010588	
Client Announcements	Target	7.380600	-0.002300	
Follow-on Equity Offerings	Target	-5.005380	-0.007275	
Investor Activism - Target Communication	Target	2.362150	-0.004881	
Earnings Calls	Target	0.481335	-0.002042	
Strategic Alliances	Target	2.626260	-0.003538	
Lawsuits & Legal Issues	Target	-0.413061	-0.002723	
Investor Activism - Activist Communication	Target	4.288130	-0.004063	
Corporate Guidance - Lowered	Target	-14.887900	-0.002680	
Private Placements	Buyer	0.272817	-0.002208	
M&A Transaction Announcements	Seller	2.630630	-0.002869	
M&A Transaction Closings	Buyer	3.186290	-0.001394	
Company Conference Presentations	Target	1.292770	-0.001116	
M&A Calls	Target	8.955770	-0.002523	
Index Constituent Adds	Target	-1.557500	-0.002675	
Buyback Transaction Closings	Target	1.725140	-0.001955	
M&A Rumors and Discussions	Target	4.876590	-0.001790	
Shareholder/Analyst Calls	Target	0.428922	-0.002059	
Labor-related Announcements	Target	0.546605	-0.002090	
Business Reorganizations	Target	-0.381681	-0.002317	
Conferences	Participant	-0.643617	-0.000928	
M&A Transaction Announcements	Buyer	3.253760	-0.001108	
Delayed SEC Filings	Target	-3.797660	-0.001306	
Earnings Release Date	Target	-0.071140	-0.000190	
M&A Transaction Closings	Seller	1.461850	-0.000303	
M&A Transaction Announcements	Target	18.265800	-0.000214	
Discontinued Operations/Downsizings	Target	-2.844410	-0.000069	
Fixed Income Offerings	Target	-2.922310	0.000168	
Corporate Guidance - New/Confirmed	Target	-0.883251	0.000366	
Private Placements	Target	4.328510	0.002351	
Seeking Acquisitions/Investments	Target	-0.305334	0.001316	
Board Meeting	Target	0.980749	0.002164	
Considering Multiple Strategic Alternatives	Target	-1.135750	0.012629	
Announcements of Sales/Trading Statement	Target	1.329410	0.005084	
Debt Financing Related	Target	2.013270	0.002883	
Annual General Meeting	Target	0.950287	0.004452	
Seeking to Sell/Divest	Target	-0.531814	0.006912	
End of Lock-Up Period	Target	-9.088620	0.007541	

(continues on next page)

(continued from previous page)

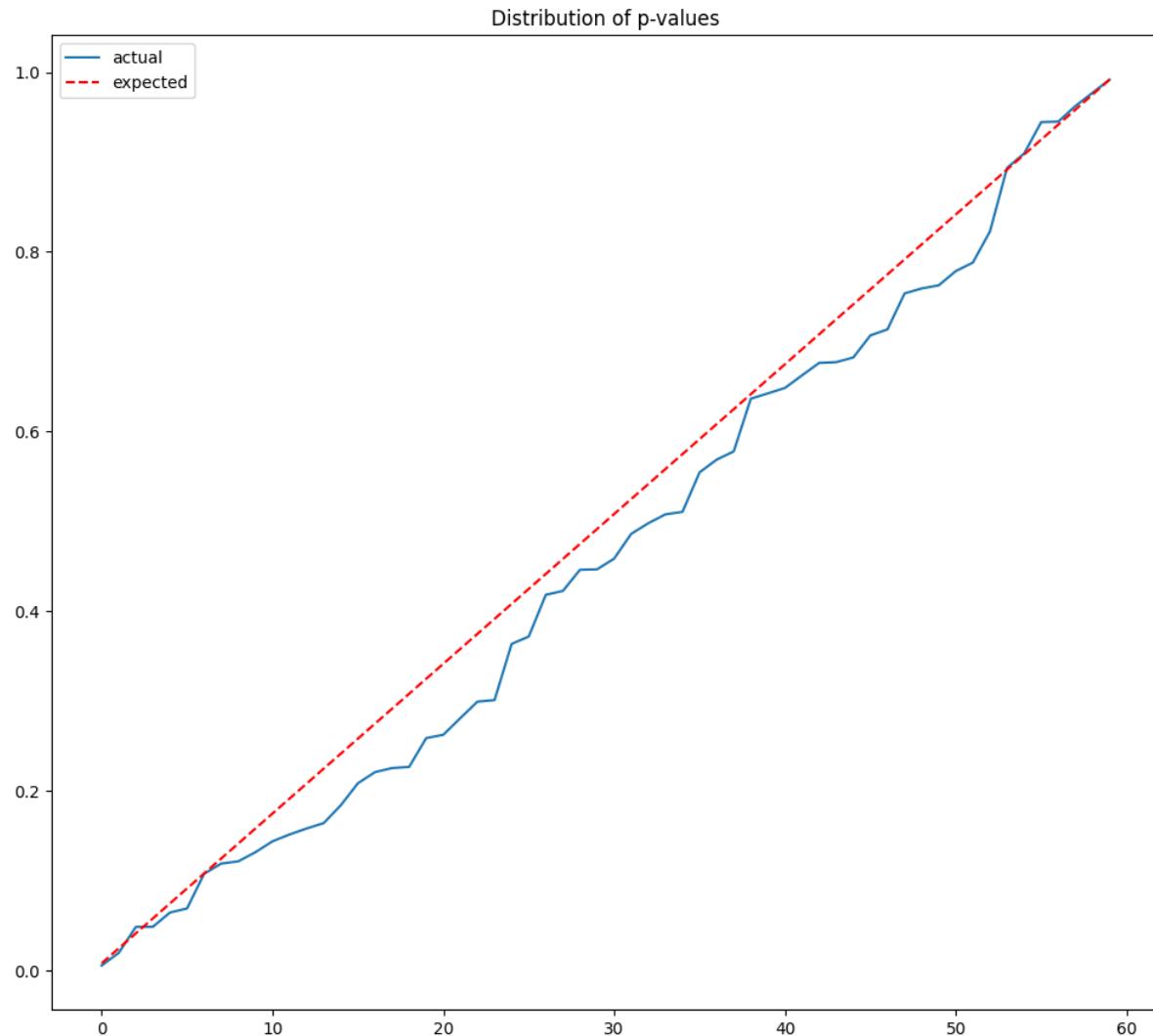
Buyback Tranche Update	Target	0.559217	0.002991
Announcements of Earnings	Target	0.305990	0.002854
Analyst/Investor Day	Target	3.238440	0.006292
Investor Activism - Proxy/Voting Related	Target	-0.472323	0.011292
Delistings	Target	-3.871270	0.016593
Impairments/Write Offs	Target	-1.122100	0.008273
Corporate Guidance - Raised	Target	11.737600	0.006226
Index Constituent Drops	Target	-0.751127	0.017488
Buyback - Change in Plan Terms	Target	3.837660	0.008018
Buyback Transaction Announcements	Target	7.037260	0.008389
			post_t
event	role		
Shelf Registration Filings	Target	-2.753130	
Special/Extraordinary Shareholders Meeting	Target	-1.970180	
Name Changes	Target	-1.968600	
Changes in Company Bylaws/Rules	Target	-1.815650	
Product-Related Announcements	Target	-1.607030	
M&A Transaction Closings	Target	-1.506510	
Auditor Changes	Target	-1.433420	
Executive/Board Changes - Other	Target	-1.410500	
Executive Changes - CFO	Target	-1.390520	
Business Expansions	Target	-1.327890	
Special Calls	Target	-1.257180	
Executive Changes - CEO	Target	-1.120140	
M&A Transaction Cancellations	Buyer	-1.037350	
Client Announcements	Target	-0.908251	
Follow-on Equity Offerings	Target	-0.892820	
Investor Activism - Target Communication	Target	-0.801679	
Earnings Calls	Target	-0.761636	
Strategic Alliances	Target	-0.741016	
Lawsuits & Legal Issues	Target	-0.677482	
Investor Activism - Activist Communication	Target	-0.657731	
Corporate Guidance - Lowered	Target	-0.590799	
Private Placements	Buyer	-0.569800	
M&A Transaction Announcements	Seller	-0.472392	
M&A Transaction Closings	Buyer	-0.464066	
Company Conference Presentations	Target	-0.455673	
M&A Calls	Target	-0.436285	
Index Constituent Adds	Target	-0.417274	
Buyback Transaction Closings	Target	-0.416104	
M&A Rumors and Discussions	Target	-0.409002	
Shareholder/Analyst Calls	Target	-0.313663	
Labor-related Announcements	Target	-0.306394	
Business Reorganizations	Target	-0.301770	
Conferences	Participant	-0.281036	
M&A Transaction Announcements	Buyer	-0.268724	
Delayed SEC Filings	Target	-0.113672	
Earnings Release Date	Target	-0.069660	
M&A Transaction Closings	Seller	-0.069015	
M&A Transaction Announcements	Target	-0.028792	
Discontinued Operations/Downsizings	Target	-0.010337	
Fixed Income Offerings	Target	0.047205	
Corporate Guidance - New/Confirmed	Target	0.133957	
Private Placements	Target	0.223887	
Seeking Acquisitions/Investments	Target	0.366793	

(continues on next page)

(continued from previous page)

Board Meeting	Target	0.375795
Considering Multiple Strategic Alternatives	Target	0.556299
Announcements of Sales/Trading Statement	Target	0.662073
Debt Financing Related	Target	0.696258
Annual General Meeting	Target	0.760826
Seeking to Sell/Divest	Target	0.809188
End of Lock-Up Period	Target	1.033670
Buyback Tranche Update	Target	1.077510
Announcements of Earnings	Target	1.128810
Analyst/Investor Day	Target	1.208400
Investor Activism - Proxy/Voting Related	Target	1.211770
Delistings	Target	1.223670
Impairments/Write Offs	Target	1.460700
Corporate Guidance - Raised	Target	1.547110
Index Constituent Drops	Target	1.557680
Buyback - Change in Plan Terms	Target	1.846240
Buyback Transaction Announcements	Target	2.327460

```
# Expected p-values (with continuity correction)
pvals = norm.cdf(-df['post_t'].abs()) * 2
fig, ax = plt.subplots(1, 1, clear=True, figsize=(10, 9))
ax.plot(sorted(pvals))
ax.plot([0, len(pvals)-1], [0.5/len(pvals), (len(pvals)-0.5)/len(pvals)], 'r--')
ax.set_title('Distribution of p-values')
ax.legend(['actual', 'expected'])
plt.tight_layout()
```



```
# Number of rejections with uncorrected pvals
argmin = np.argmax(pvals)
header = df.iloc[argmin][['event', 'role', 'days', 'effective', 'post_t']]
alpha = 0.05
uncorrected = DataFrame({'rejections': sum(pvals < alpha),
                        'min p-value': min(pvals)},
                        index=['uncorrected'])
uncorrected.round(4)
```

	rejections	min p-value
uncorrected	4	0.0059

7.3.1 Bonferroni correction

The best-known FWER test is called the Bonferroni test which adjusts for the multiple tests. Given the chance that one test could randomly show up as significant, the Bonferroni requires the confidence level to increase. Instead of 5%, you take the 5% and divide by the number of tests, that is, $5\%/10 = 0.5\%$. Again equivalently, you need to be 99.5% confident with 10 tests that you are not making a single false discovery. In terms of the t-statistic, the Bonferroni requires a statistic of at least 2.8 for 10 tests. For 1,000 tests, the statistic must exceed 4.1.

It sets the threshold for rejecting each hypothesis to α/m , by applying the union bound inequality:

$$\text{FWER}(\alpha) = \Pr(\bigcup_{j=1}^m A_j) \leq \sum_{j=1}^m \Pr(A_j) \leq m \times \frac{\alpha}{m} = \alpha$$

where A_j denotes the probability of rejecting the j -th hypothesis. The Bonferroni correction can be quite conservative, in the sense that the true FWER is often quite a bit lower than the nominal (or target) FWER;

```
# The Bonferroni p-values bonf are simply the uncorrected pvalues multiplied by
# the number of samples and truncated to be less than or equal to 1.
reject, bonf_corrected, _, _ = multipletests(pvals, alpha=alpha, method='bonferroni')
bonf = DataFrame({'rejections': np.sum(bonf_corrected < alpha),
                  'min p-value': np.min(bonf_corrected)},
                  index=['bonferroni'])
pd.concat([uncorrected, bonf], axis=0).round(4)
```

	rejections	min p-value
uncorrected	4	0.0059
bonferroni	0	0.3542

7.3.2 Holm's step-down procedure

Holm's method, also known as Holm's step-down procedure or the Holm-Bonferroni method, is an alternative to the Bonferroni procedure. Holm's method controls the FWER, but it is less conservative than Bonferroni, in the sense that it will reject more null hypotheses, typically resulting in fewer Type II errors and hence greater power. Holm's method makes no independence assumptions about the hypothesis tests, and is uniformly more powerful than the Bonferroni method – it will always reject at least as many null hypotheses as Bonferroni – and so it should always be preferred. It is worth noting that in Holm's procedure, the threshold that we use to reject each null hypothesis actually depends on the values of all the p-values.

The Holm method begins by sorting the tests from the lowest p-value (most significant) to the highest (least significant), and comparing a threshold computed with the Holm function. In contrast to the Bonferroni, which has a single threshold for all tests, the other tests will have a different hurdle under Holm. Starting from the first test, we sequentially compare the p-values with their hurdles. When we first come across the test such that its p-value fails to meet the hurdle, we reject this test and all others with higher p-values.

```
# Holm method
reject, holm_corrected, _, _ = multipletests(pvals, alpha=alpha, method='holm')
holm = DataFrame({'rejections': np.sum(holm_corrected < alpha),
                  'min p-value': np.min(holm_corrected)},
                  index=['holm'])
pd.concat([uncorrected, bonf, holm], axis=0).round(4)
```

	rejections	min p-value
uncorrected	4	0.0059

(continues on next page)

(continued from previous page)

bonferroni	0	0.3542
holm	0	0.3542

7.3.3 Benjamin-Hochberg procedure

The false discovery rate approach allows an expected proportional error rate. As such, it is less stringent than both the Bonferroni and the Holm test. FDR control is much milder – and more powerful – than FWER control, in the sense that it allows us to reject many more null hypotheses, with a cost of substantially more false positives.

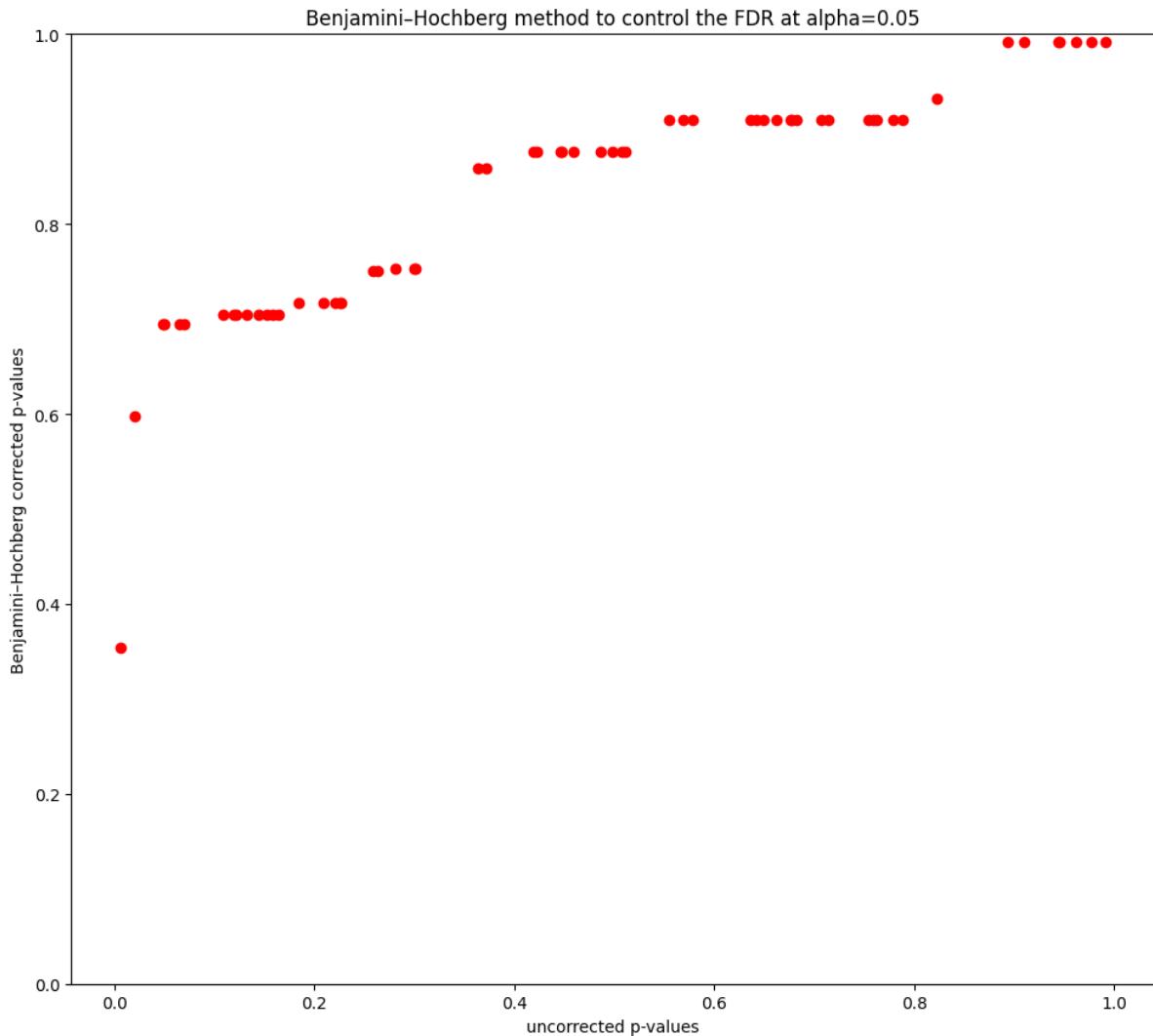
Similar to the Holm test, Benjamin-Hochberg also relies on the distribution of test statistics. However, in contrast to the Holm test that begins with the most significant test, the Benjamin-Hochberg approach starts with the least significant. We sort the tests from the lowest p-value (most significant) to the highest (least significant). Starting from the last test, we sequentially compare the p-values with their Benjamin-Hochberg thresholds. When we first come across the test such that its p-value falls below its threshold, we declare this test significant and all tests that have a lower p-value.

The `multipletests` function can be used to carry out the Benjamini-Hochberg procedure. The q-values output by the Benjamini-Hochberg procedure can be interpreted as the smallest FDR threshold at which we would reject a particular null hypothesis. For instance, a q-value of 0.1 indicates that we can reject the corresponding null hypothesis at an FDR of 10% or greater, but that we cannot reject the null hypothesis at an FDR below 10%.

```
# Benjamini-Hochberg method
reject, fdr_bh_corrected, _, _ = multipletests(pvals, alpha=alpha, method='fdr_bh')
fdr_bh = DataFrame({'rejections': np.sum(fdr_bh_corrected < alpha),
                    'min p-value': np.min(fdr_bh_corrected)},
                   index=['fdr_bh'])
pd.concat([uncorrected, bonf, holm, fdr_bh], axis=0).round(4)
```

	rejections	min p-value
uncorrected	4	0.0059
bonferroni	0	0.3542
holm	0	0.3542
fdr_bh	0	0.3542

```
# Plot uncorrected and corrected p-values from Benjamini-Hochberg method
fig, ax = plt.subplots(figsize=(10, 9))
ax.scatter(pvals, fdr_bh_corrected, color='red')
ax.set_xlim(bottom=0, top=1)
ax.set_title(f"Benjamini-Hochberg method to control the FDR at alpha={alpha}")
ax.set_xlabel('uncorrected p-values')
ax.set_ylabel('Benjamini-Hochberg corrected p-values')
plt.tight_layout()
```



References

Kolari, James W. and Pynnonen, Seppo, 2010, “Event Study Testing with Cross-sectional Correlation of Abnormal Returns”. *The Review of Financial Studies*, 23(11), 3996-4025

Kolari, James W., Paper, Bernd, and Pynnonen, Seppo, 2018, “Event Study Testing with Cross-sectional Correlation Due to Partially Overlapping Event Windows”, working paper

MacKinlay, A. Craig “Event Studies in Economics and Finance”, *Journal of Economic Literature*, March 1997.

MIT License. Copyright 2021-2025 Terence Lim

ECONOMIC INDICATORS

What we learn from history is that people don't learn from history - Warren Buffett

Economic data is fundamental to financial analysis, policymaking, and investment strategies. However, many economic indicators are subject to revisions, meaning initial estimates may change over time as more accurate data becomes available. Understanding these revisions is crucial for interpreting past economic conditions, refining forecasting models, and making informed decisions. We explore retrieving data from online sources such as the Federal Reserve Economic Data (FRED), its archival counterpart (ALFRED), and key derived datasets such as FRED-MD and FRED-QD. Additionally, we examine the impact of data revisions on critical economic indicators like Total Nonfarm Payrolls (PAYEMS), and methods for detecting outliers in historical data.

```
# By: Terence Lim, 2020-2025 (terence-lim.github.io)
import numpy as np
import pandas as pd
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
import textwrap
from finds.readers import Alfred, fred_md, fred_qd
from finds.utils import plot_date, plot_groupbar
from finds.recipes import is_outlier
from datetime import datetime
from pprint import pprint
from secret import credentials
VERBOSE = 0
# %matplotlib qt
```

8.1 FRED

Federal Reserve Economic Data (FRED) is a widely used online database maintained by the Federal Reserve Bank of St. Louis, providing access to hundreds of thousands of economic data series from national and international sources. Users can retrieve data via the website, an Excel add-in, or API calls.

8.2 Retrieving data from websites

Economic data can be retrieved from the web through several methods:

1. **Downloading structured files** – Many websites provide data in formats like CSV, Excel, or JSON, making it easy to import into analytical tools.
2. **Web scraping** – Extracting information directly from web pages by identifying specific HTML tags or text patterns.
3. **Using APIs** – Some platforms, including FRED, offer APIs that allow developers to automate data retrieval via structured queries.

8.2.1 Download structured files

Many economic data providers allow users to download pre-structured files containing historical and current data. These files often include metadata, timestamps, and adjustment information.

```
# This URL is the location of the FRED-MD csv file from the St Louis FRED
url = 'https://www.stlouisfed.org/-/media/project/frbstl/stlouisfed/research/fred-md/
˓monthly/current.csv'
```

```
# Pandas has several built-in readers for csv, xml, json, excel and even html files
˓
df = pd.read_csv(url, header=0)
df
```

	sasdate	RPI	W875RX1	DPCERA3M086SBEA	CMRMTSPLx	\\
0	Transform:	5.000	5.0	5.000	5.000000e+00	
1	1/1/1959	2583.560	2426.0	15.188	2.766768e+05	
2	2/1/1959	2593.596	2434.8	15.346	2.787140e+05	
3	3/1/1959	2610.396	2452.7	15.491	2.777753e+05	
4	4/1/1959	2627.446	2470.0	15.435	2.833627e+05	
..
788	8/1/2024	20007.209	16322.1	121.052	1.530317e+06	
789	9/1/2024	20044.142	16333.7	121.690	1.541305e+06	
790	10/1/2024	20128.752	16397.9	121.948	1.539382e+06	
791	11/1/2024	20161.687	16432.8	122.519	1.544190e+06	
792	12/1/2024	20184.060	16457.8	123.013	NaN	
	RETAILx	INDPRO	IPFPNSS	IPFINAL	IPCONGD	...
0	5.00000	5.0000	5.0000	5.0000	5.0000	...
1	18235.77392	21.9616	23.3868	22.2620	31.6664	...
2	18369.56308	22.3917	23.7024	22.4549	31.8987	...
3	18523.05762	22.7142	23.8459	22.5651	31.8987	...
4	18534.46600	23.1981	24.1903	22.8957	32.4019	...
..

(continues on next page)

(continued from previous page)

```

788 710038.00000 103.0135 100.9825 100.9803 102.2118 ...
789 716388.00000 102.5969 100.3826 100.0630 101.9696 ...
790 720393.00000 102.0854 99.5434 98.9267 101.3127 ...
791 725925.00000 102.2549 99.8216 99.4970 101.7893 ...
792 729191.00000 103.1942 100.5351 100.1302 102.2582 ...

DNDGRG3M086SBEA DSERRG3M086SBEA CES0600000008 CES2000000008 \
0 6.000 6.000 6.00 6.00
1 18.294 10.152 2.13 2.45
2 18.302 10.167 2.14 2.46
3 18.289 10.185 2.15 2.45
4 18.300 10.221 2.16 2.47
...
788 119.653 128.291 31.26 35.81
789 119.220 128.682 31.44 36.00
790 119.064 129.169 31.55 36.22
791 119.112 129.375 31.61 36.21
792 119.689 129.760 31.73 36.44

CES3000000008 UMCSENTx DTCOLNVHFNM DTCTHFNM INVEST VIXCLSp
0 6.00 2.0 6.00 6.00 6.0000 1.0000
1 2.04 NaN 6476.00 12298.00 84.2043 NaN
2 2.05 NaN 6476.00 12298.00 83.5280 NaN
3 2.07 NaN 6508.00 12349.00 81.6405 NaN
4 2.08 NaN 6620.00 12484.00 81.8099 NaN
...
788 27.97 67.9 551667.22 933066.90 5327.6461 19.6750
789 28.11 70.1 553347.06 934283.59 5368.5818 17.6597
790 28.14 70.5 554377.25 937299.96 5407.3304 19.9478
791 28.29 71.8 555000.61 938899.31 5382.4019 15.9822
792 28.34 74.0 NaN NaN 5370.6184 15.6997

```

[793 rows x 127 columns]

8.2.2 Web scraping

Web scraping involves extracting data from unstructured web pages by identifying patterns in the HTML structure. This method is useful when structured data files are unavailable, but it requires compliance with website policies.

```
# URL that displays the most popular series in the FRED economic data web site
url = "https://fred.stlouisfed.org/tags/series?ob=pv&pageID=1"
```

```
# use requests package to retrieve the web page
import requests
data = requests.get(url)
data # a response code of 200 indicates the request has succeeded
```

<Response [200]>

```
# the content is just a byte-string that you can parse with Python string (or other) ↵
data.content[:200]
```

```
b'<!DOCTYPE html>\n<html lang="en">\n<head>\n    <meta http-equiv="X-UA-Compatible\n    content="IE=edge">\n    <meta charset="utf-8">\n    <title>\nEconomic Data Series by Tag | FRED | St. Louis Fed</'
```

```
# use the BeautifulSoup package to parse html formats
from bs4 import BeautifulSoup
soup = BeautifulSoup(data.content, 'lxml')

# based on this snippet, we want to extract the href property of the series-title
# class tag
print(soup.decode()[39000:40000])
```

```
s="series-title pager-series-title-gtm" href="/series/T10Y2Y" id="titleLink" style=
    "font-size:1.2em; padding-bottom: 2px">10-Year Treasury Constant Maturity Minus
    2-Year Treasury Constant Maturity</a></h3>
</div>
<div class="display-results-popularity-bar d-none d-sm-block col-sm-2">
<span aria-label="popularity 100% popular" class="popularity-bar-span-parent" data-
    target="popularity-bar-span-T10Y2Y" tabindex="0" title="100% popular">
<span aria-hidden="true" class="popularity_bar" style="padding-top: 3px; padding-
    left:60px;"> </span> <span aria-hidden="true" class="popularity_bar_background"-
    id="popularity-bar-span-T10Y2Y"> </span></span>
</div>
</td>
</tr>
<tr class="series-pager-attr">
<td colspan="2">
<div class="series-meta series-group-meta">
<span class="attributes">Percent, Not Seasonally Adjusted</span>
<br class="clear"/>
</div>
<div class="series-meta">
<input aria-labelledby="unitLinkT10Y2Y" class="pager-item-checkbox pager-check-
    series-gtm" name="sids[0]" type="checkbox" v
```

```
# identify all the tags whose class starts with 'series-title'
tags = soup.findAll(name='a', attrs={'class': 'series-title'})
tags[0] # show first tag found
```

```
<a class="series-title pager-series-title-gtm" href="/series/T10Y2Y" id="titleLink
    " style="font-size:1.2em; padding-bottom: 2px">10-Year Treasury Constant
    Maturity Minus 2-Year Treasury Constant Maturity</a>
```

```
# extract desired substring (which is a data series mnemonic) from the href property
details = [tag.get('href').split('/')[-1] for tag in tags] # only want substring
# after last '/'
details[0] # show first mnemonic string found
```

```
'T10Y2Y'
```

8.2.3 Using APIs

APIs (Application Programming Interfaces) enable direct communication with data servers, allowing for real-time data retrieval. Many economic research institutions, including the St Louis Fed, offer APIs to access macroeconomic data programmatically.

```
# an API call is simply a URL string containing your parameters for the request
url = "{root}?series_id={series_id}&file_type={file_type}&api_key={api_key}".format(
    root="https://api.stlouisfed.org/fred/series",
    series_id=details[0], # mnemonic of the data series to
    ↪retrieve
    file_type='json', # request data be returned in json
    ↪format
    api_key=credentials['fred']['api_key']) # private api key (obtain from
    ↪FRED for free)
```

```
# make the API call to retrieve the data
data = requests.get(url)
data.content
```

```
b'{"realtime_start": "2025-02-28", "realtime_end": "2025-02-28", "serieses": [{"id": "T10Y2Y", "realtime_start": "2025-02-28", "realtime_end": "2025-02-28", "title": "10-Year Treasury Constant Maturity Minus 2-Year Treasury Constant Maturity", "observation_start": "1976-06-01", "observation_end": "2025-02-28", "frequency": "Daily", "frequency_short": "D", "units": "Percent", "units_short": "%", "seasonal_adjustment": "Not Seasonally Adjusted", "seasonal_adjustment_short": "NSA", "last_updated": "2025-02-28 16:02:07-06", "popularity": 100, "notes": "Starting with the update on June 21, 2019, the Treasury bond data used in calculating interest rate spreads is obtained directly from the U.S. Treasury Department (https://www.treasury.gov/resource-center/data-chart-center/interest-rates/Pages/Textview.aspx?data=yield). \r\nSeries is calculated as the spread between 10-Year Treasury Constant Maturity (BC_10YEAR) and 2-Year Treasury Constant Maturity (BC_2YEAR). Both underlying series are published at the U.S. Treasury Department (https://www.treasury.gov/resource-center/data-chart-center/interest-rates/Pages/Textview.aspx?data=yield)."}]}'
```

```
# use the json package to convert byte-string data content
import json
v = json.loads(data.content)
v
```

```
{'realtime_start': '2025-02-28',
'realtime_end': '2025-02-28',
'serieses': [{"id': 'T10Y2Y',
'realtime_start': '2025-02-28',
'realtime_end': '2025-02-28',
'title': '10-Year Treasury Constant Maturity Minus 2-Year Treasury Constant Maturity',
'observation_start': '1976-06-01',
'observation_end': '2025-02-28',
'frequency': 'Daily',
'frequency_short': 'D',
'units': 'Percent',
'units_short': '%',
```

(continues on next page)

(continued from previous page)

```
'seasonal_adjustment': 'Not Seasonally Adjusted',
'seasonal_adjustment_short': 'NSA',
'last_updated': '2025-02-28 16:02:07-06',
'popularity': 100,
'notes': 'Starting with the update on June 21, 2019, the Treasury bond data used in calculating interest rate spreads is obtained directly from the U.S. Treasury Department (https://www.treasury.gov/resource-center/data-chart-center/interest-rates/Pages/TextView.aspx?data=yield). \r\nSeries is calculated as the spread between 10-Year Treasury Constant Maturity (BC_10YEAR) and 2-Year Treasury Constant Maturity (BC_2YEAR). Both underlying series are published at the U.S. Treasury Department (https://www.treasury.gov/resource-center/data-chart-center/interest-rates/Pages/TextView.aspx?data=yield). '}]}
```

```
# Pandas can create a DataFrame directly from a dict data structure
df = DataFrame(v['series'])
df
```

```
id realtime_start realtime_end \
0 T10Y2Y 2025-02-28 2025-02-28

title observation_start \
0 10-Year Treasury Constant Maturity Minus 2-Yea... 1976-06-01

observation_end frequency frequency_short units units_short \
0 2025-02-28 Daily D Percent %

seasonal_adjustment seasonal_adjustment_short last_updated \
0 Not Seasonally Adjusted NSA 2025-02-28 16:02:07-06

popularity notes
0 100 Starting with the update on June 21, 2019, the...
```

8.2.4 ALFRED (Archival FRED)

ALFRED extends FRED's functionality by preserving historical versions of economic data. This allows researchers to track how data revisions impact economic narratives over time.

```
today = int(datetime.today().strftime('%Y%m%d'))
alf = Alfred(api_key=credentials['fred']['api_key'], verbose=VERBOSE)
```

8.2.5 Popular FRED series

FRED organizes its data into **categories, frequencies, and seasonal adjustments**. Some of the most frequently accessed series include employment figures, inflation metrics, and GDP growth rates. A current list of the most popular FRED series can be found [here](#).

```
# scrape FRED most popular page
popular = {}
titles = Alfred.popular(1)
for title in titles:
```

(continues on next page)

(continued from previous page)

```

series = alf.request_series(title)    # requests 'series' FRED api
if not series.empty:
    popular[title] = series.iloc[-1][['title', 'popularity']]
print(f"Most Popular Series in FRED, retrieved {today}")
DataFrame.from_dict(popular, orient='index')

```

Most Popular Series in FRED, retrieved 20250302

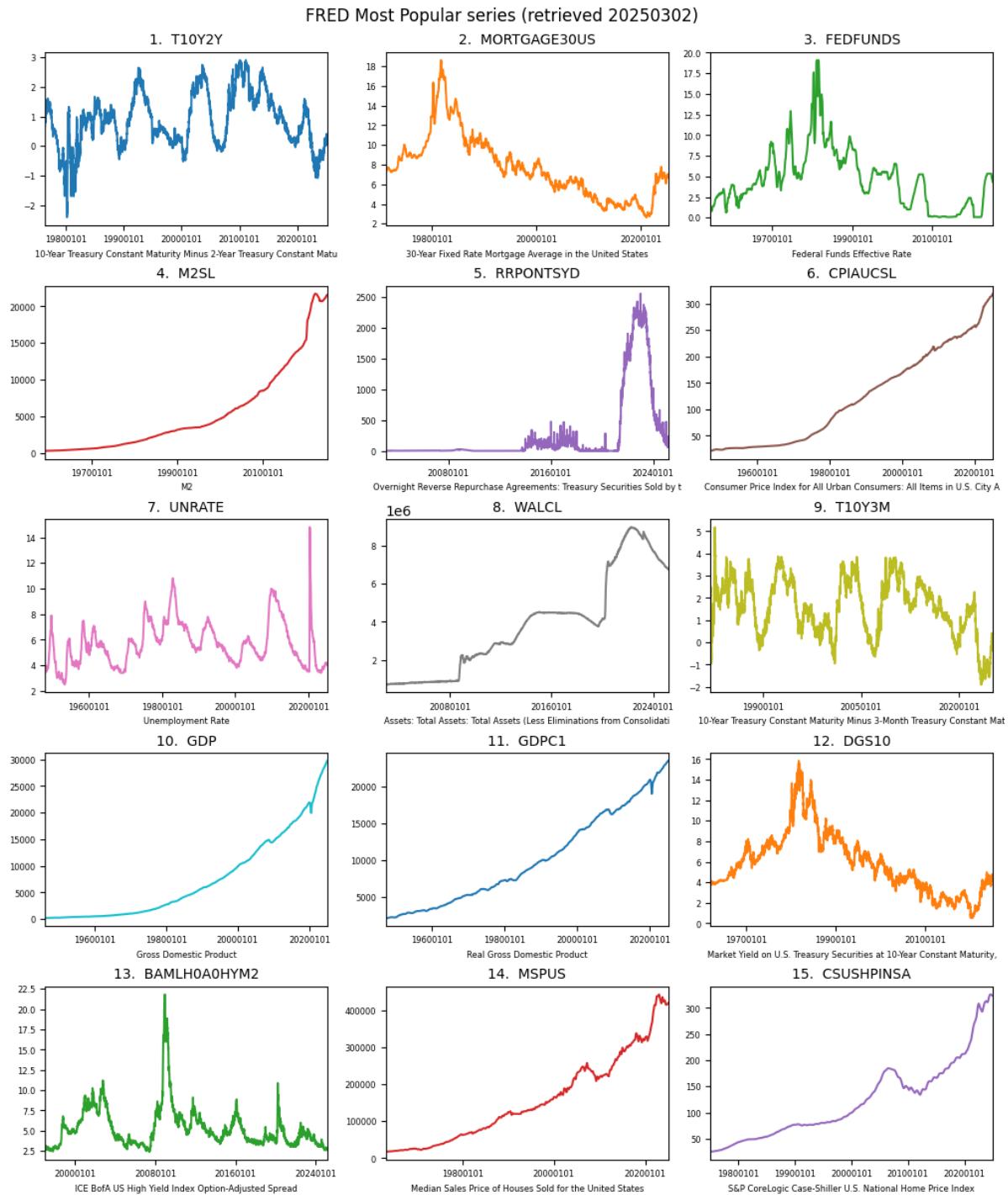
		title	popularity
T10Y2Y	10-Year Treasury Constant Maturity Minus 2-Yea...	100	
MORTGAGE30US	30-Year Fixed Rate Mortgage Average in the Uni...	99	
FEDFUNDS	Federal Funds Effective Rate	98	
M2SL	M2	93	
RRPONTSYD	Overnight Reverse Repurchase Agreements: Treas...	95	
CPIAUCSL	Consumer Price Index for All Urban Consumers: ...	95	
UNRATE	Unemployment Rate	95	
WALCL	Assets: Total Assets: Total Assets (Less Elimi...	94	
T10Y3M	10-Year Treasury Constant Maturity Minus 3-Mon...	94	
GDP	Gross Domestic Product	93	
GDPC1	Real Gross Domestic Product	92	
DGS10	Market Yield on U.S. Treasury Securities at 10...	92	
BAMLH0A0HYM2	ICE BofA US High Yield Index Option-Adjusted S...	92	
MSPUS	Median Sales Price of Houses Sold for the Unit...	90	
CSUSHPINSA	S&P CoreLogic Case-Shiller U.S. National Home ...	88	
T10YIE	10-Year Breakeven Inflation Rate	89	
FPCPITOLZGUSA	Inflation, consumer prices for the United States	85	
M1SL	M1	84	

```

# plot popular series
fig, axes = plt.subplots(ncols=3, nrows=5, figsize=(10, 12), layout='constrained')
for cn, (ax, title) in enumerate(zip(np.ravel(axes), titles[:15])):
    series = alf(title)
    plot_date(series, ax=ax, title=f"{cn+1}. {title}", xlabel=alf.header(title)[:70],
              fontsize=6, ls='-', cn=cn, nbins=4)
plt.suptitle(f"FRED Most Popular series (retrieved {today})")

```

Text(0.5, 0.98, 'FRED Most Popular series (retrieved 20250302)')



8.2.6 FRED series categories

One of the most closely watched FRED series is Total Nonfarm Payroll Employment (PAYEMS), a key labor market indicator. This series belongs to broader employment-related categories.

```
# Retrieve grandparent, parent and siblings of series
series_id, freq = 'PAYEMS', 'M'
category = alf.categories(series_id).iloc[0]
grand_category = alf.get_category(category['parent_id'])
parent_category = alf.get_category(category['id'])
category.to_frame().T
```

	id	name	parent_id
PAYEMS	32305	Total Nonfarm	11

```
print(f"Super category {grand_category['id']}: {grand_category['name']} ")
if 'notes' in grand_category:
    print(textwrap.fill(grand_category['notes']))
```

Super category 11: Current Employment Statistics (Establishment Survey)
The establishment survey provides data on employment, hours, and earnings by industry. Numerous conceptual and methodological differences between the current population (household) and establishment surveys result in important distinctions in the employment estimates derived from the surveys. Among these are: The household survey includes agricultural workers, the self-employed, unpaid family workers, and private household workers among the employed. These groups are excluded from the establishment survey. The household survey includes people on unpaid leave among the employed. The establishment survey does not. The household survey is limited to workers 16 years of age and older. The establishment survey is not limited by age. The household survey has no duplication of individuals, because individuals are counted only once, even if they hold more than one job. In the establishment survey, employees working at more than one job and thus appearing on more than one payroll are counted separately for each appearance. For more information, visit <http://www.bls.gov/news.release/empstat.tn.htm>.

```
print("Parent categories:")
for child in grand_category['children']:
    node = alf.get_category(child['id'])
    if node:
        print(f" {node['id']}: {node['name']} "
              f" (children={len(node['children'])}, series={len(node['series'])})")
```

Parent categories:

32305: Total Nonfarm	(children=0, series=5)
32306: Total Private	(children=0, series=27)
32307: Goods-Producing	(children=0, series=27)
32326: Service-Providing	(children=0, series=1)
32308: Private Service-Providing	(children=0, series=27)
32309: Mining and Logging	(children=0, series=39)
32310: Construction	(children=0, series=41)
32311: Manufacturing	(children=0, series=31)

(continues on next page)

(continued from previous page)

```

32312: Durable Goods (children=0, series=63)
32313: Nondurable Goods (children=0, series=55)
32314: Trade, Transportation, and Utilities (children=0, series=27)
32315: Wholesale Trade (children=0, series=33)
32316: Retail Trade (children=0, series=55)
32317: Transportation and Warehousing (children=0, series=47)
32318: Utilities (children=0, series=27)
32319: Information (children=0, series=39)
32320: Financial Activities (children=0, series=51)
32321: Professional and Business Services (children=0, series=55)
32322: Education and Health Services (children=0, series=51)
32323: Leisure and Hospitality (children=0, series=41)
32324: Other Services (children=0, series=33)
32325: Government (children=0, series=23)

```

```

print("Sibling series:")
for child in parent_category['series']:
    if child['id'] == series_id:
        node = child
    print(f" {child['id']}: {child['title']} {child['seasonal_adjustment']}"
          f" (popularity={child['popularity']}"))

```

Sibling series:

```

CES0000000010: Women Employees, Total Nonfarm Seasonally Adjusted (popularity=4)
CES0000000039: Women Employees-To-All Employees Ratio: Total Nonfarm Seasonally Adjusted (popularity=16)
CEU0000000010: Women Employees, Total Nonfarm Not Seasonally Adjusted (popularity=1)
PAYEMS: All Employees, Total Nonfarm Seasonally Adjusted (popularity=83)
PAYNSA: All Employees, Total Nonfarm Not Seasonally Adjusted (popularity=47)

```

```

print(f"{node['id']}: {node['title']} {node['seasonal_adjustment']}",
      f" ({node['observation_start']}-{node['observation_end']})")
print()
print(textwrap.fill(node['notes']))

```

PAYEMS: All Employees, Total Nonfarm Seasonally Adjusted (1939-01-01-2025-01-01)

All Employees: Total Nonfarm, commonly known as Total Nonfarm Payroll, is a measure of the number of U.S. workers in the economy that excludes proprietors, private household employees, unpaid volunteers, farm employees, and the unincorporated self-employed. This measure accounts for approximately 80 percent of the workers who contribute to Gross Domestic Product (GDP). This measure provides useful insights into the current economic situation because it can represent the number of jobs added or lost in an economy. Increases in employment might indicate that businesses are hiring which might also suggest that businesses are growing. Additionally, those who are newly employed have increased their personal incomes, which means (all else constant) their disposable incomes have also increased, thus fostering further economic expansion. Generally, the U.S. labor force and levels of employment and unemployment are subject to fluctuations due to seasonal changes in weather, major holidays, and the opening and

(continues on next page)

(continued from previous page)

closing of schools. The Bureau of Labor Statistics (BLS) adjusts the data to offset the seasonal effects to show non-seasonal changes: for example, women's participation in the labor force; or a general decline in the number of employees, a possible indication of a downturn in the economy. To closely examine seasonal and non-seasonal changes, the BLS releases two monthly statistical measures: the seasonally adjusted All Employees: Total Nonfarm (PAYEMS) and All Employees: Total Nonfarm (PAYNSA), which is not seasonally adjusted. The series comes from the 'Current Employment Statistics (Establishment Survey).' The source code is: CES0000000001

8.3 Revisions and vintage dates

Economic data revisions occur as new information becomes available, improving the accuracy of initial estimates. The Bureau of Labor Statistics (BLS), for instance, releases an initial estimate of Total Nonfarm Payroll Employment (PAYEMS) on the first Friday of each month. However, this figure is a very rough estimate, which is then revised in subsequent months as more firm-level data is collected.

These revisions can be significant, sometimes altering economic assessments. ALFRED, the archival FRED tool, allows users to compare initial estimates with later revisions. For the monthly values of PAYEMS in 2023, we examine the total amount of changes at each subsequent revision.

```
start, end = 20230101, 20231231
data = {}
print(f'{alf.header(series_id)} (retrieved {today}):')
latest = alf(series_id, start=start, end=end, freq=freq, realtime=True)
latest
```

All Employees, Total Nonfarm (retrieved 20250302):

	PAYEMS	realtime_start	realtime_end
date			
20230131	154780	20250207	99991231
20230228	155086	20250207	99991231
20230331	155171	20250207	99991231
20230430	155387	20250207	99991231
20230531	155614	20250207	99991231
20230630	155871	20250207	99991231
20230731	156019	20250207	99991231
20230831	156176	20250207	99991231
20230930	156334	20250207	99991231
20231031	156520	20250207	99991231
20231130	156661	20250207	99991231
20231231	156930	20250207	99991231

```
print("First Release:")
data[0] = alf(series_id, release=1, start=start, end=end, freq=freq, realtime=True)
data[0]
```

First Release:

	PAYEMS	realtime_start	realtime_end
date			
20230131	155073	20230203	20230309
20230228	155350	20230310	20230406
20230331	155569	20230407	20230504
20230430	155673	20230505	20230601
20230531	156105	20230602	20230706
20230630	156204	20230707	20230803
20230731	156342	20230804	20230831
20230831	156419	20230901	20231005
20230930	156874	20231006	20231102
20231031	156923	20231103	20231207
20231130	157087	20231208	20240104
20231231	157232	20240105	20240201

```
print("Second Release:")
data[1] = alf(series_id, release=2, start=start, end=end, freq=freq, realtime=True)
data[1]
```

Second Release:

	PAYEMS	realtime_start	realtime_end
date			
20230131	155039	20230310	20230406
20230228	155333	20230407	20230504
20230331	155420	20230505	20230601
20230430	155766	20230602	20230706
20230531	155995	20230707	20230803
20230630	156155	20230804	20230831
20230731	156232	20230901	20231005
20230831	156538	20231006	20231102
20230930	156773	20231103	20231207
20231031	156888	20231208	20240104
20231130	157016	20240105	20240201
20231231	157347	20240202	20240307

```
print("Third Release:")
data[2] = alf(series_id, release=3, start=start, end=end, freq=freq, realtime=True)
data[2]
```

Third Release:

	PAYEMS	realtime_start	realtime_end
date			
20230131	155007	20230407	20240201
20230228	155255	20230505	20240201
20230331	155472	20230602	20240201
20230430	155689	20230707	20240201
20230531	155970	20230804	20240201
20230630	156075	20230901	20240201
20230731	156311	20231006	20240201
20230831	156476	20231103	20240201
20230930	156738	20231208	20240201

(continues on next page)

(continued from previous page)

20231031	156843	20240105	20240201
20231130	157014	20240202	20250206
20231231	157304	20240308	20250206

```
print("Fourth Release:")
data[3] = alf(series_id, release=4, start=start, end=end, freq=freq, realtime=True)
data[3]
```

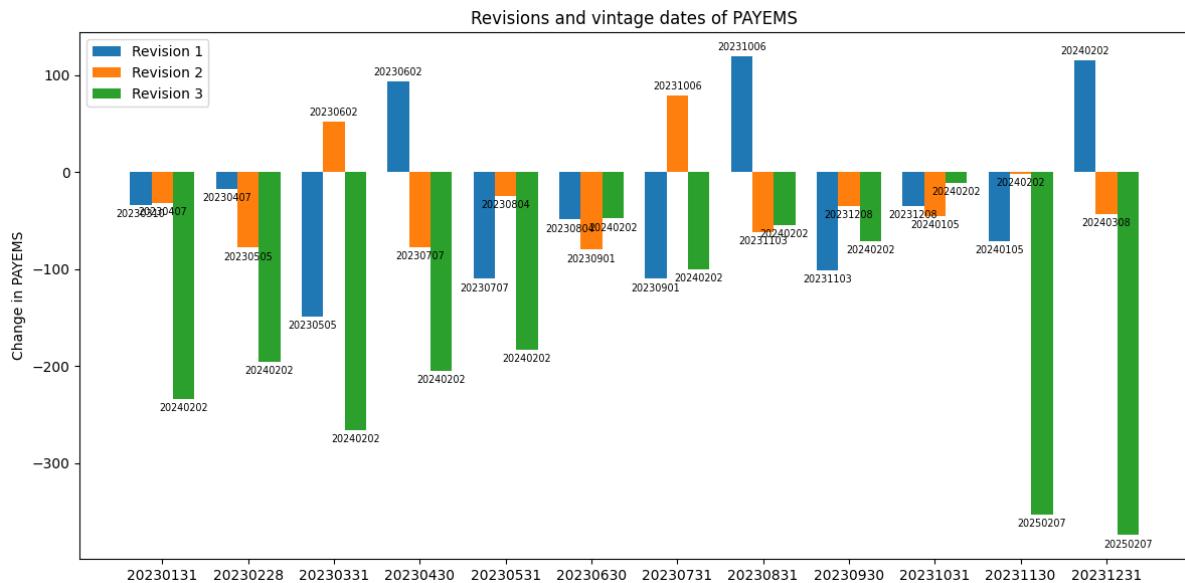
Fourth Release:

	PAYEMS	realtime_start	realtime_end
date			
20230131	154773	20240202	20250206
20230228	155060	20240202	20250206
20230331	155206	20240202	20250206
20230430	155484	20240202	20250206
20230531	155787	20240202	20250206
20230630	156027	20240202	20250206
20230731	156211	20240202	20250206
20230831	156421	20240202	20250206
20230930	156667	20240202	20250206
20231031	156832	20240202	20250206
20231130	156661	20250207	99991231
20231231	156930	20250207	99991231

```
df = pd.concat([(data[i][series_id] - data[i-1][series_id]).rename(f"Revision {i}")  
               for i in range(1, len(data))], axis=1)  
labels = pd.concat([data[i]['realtime_start'].rename(f"Revision {i}")  
                   for i in range(1, len(data))], axis=1).fillna(0).astype(int)  
DataFrame(df.sum(axis=0).rename("Total revisions ('000)"))
```

	Total revisions ('000)
Revision 1	-349
Revision 2	-348
Revision 3	-2095

```
#df = pd.concat([data[i][series_id].rename(f"Revision {i}")  
               for i in range(1, len(data))], axis=1)  
#labels = pd.concat([data[i]['realtime_start'].rename(f"Revision {i}")  
#                   for i in range(1, len(data))], axis=1).fillna(0).astype(int)  
fig, ax = plt.subplots(figsize=(12, 6))  
plot_groupbar(df, labels=labels, ax=ax)  
plt.legend()  
plt.ylabel(f'Change in {series_id}')  
plt.title(f'Revisions and vintage dates of {series_id}')  
plt.tight_layout()  
plt.show()
```



8.3.1 FRED-MD and FRED-QD

FRED-MD (Monthly Database) and FRED-QD (Quarterly Database) are curated datasets that streamline access to macroeconomic indicators. These datasets mimic the coverage of macroeconomic datasets used in the research literature and are updated in real-time, relieving users from the task of incorporating data changes and revisions. Historical monthly snap-shots of the datasets are also available.

8.3.2 Release dates

The timing of data releases is crucial for market participants and policymakers.

```
md_df, md_transform = fred_md()
end = md_df.index[-1]
out = {}
for i, title in enumerate(md_df.columns):
    out[title] = alf(series_id=title,
                      release=1,
                      start=end, # within 4 days of monthend
                      end=end,
                      realtime=True)
    if title.startswith('S&P'): # stock market data available same day close
        out[title] = Series({end: end}, name='realtime_start').to_frame()
    elif title in alf.splice_: # these series were renamed or spliced
        if isinstance(Alfred.splice_[title], str): # if renamed
            out[title] = alf(series_id=Alfred.splice_[title],
                              release=1,
                              start=end-4, # within 4 days of monthend
                              end=end,
                              realtime=True)
    else: # if FRED-MD series was spliced
        out[title] = pd.concat([alf(series_id=sub,
                                     reglease=1,
                                     start=end-4, # within 4 days of monthend
                                     end=end,
                                     realtime=True)
                               for sub in Alfred.splice_[title]])
(continues on next page)
```

(continued from previous page)

```
        end=end,
        realtime=True)
for sub in Alfred.splice_[title][1:]]
```

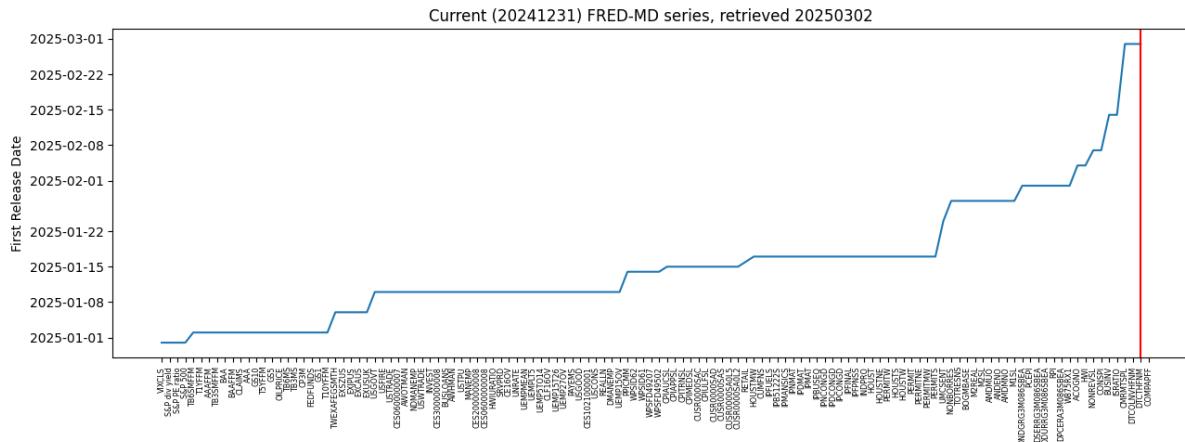
FRED-MD vintage: monthly/current.csv

```
# date convention of Consumer Sentiment
df = alf('UMCSENT', release=1, realtime=True)
out['UMCSENT'] = df[df['realtime_start'] > end - 4].iloc[:1]
```

```
# weekly averages of Claims
df = alf('ICNSA', release=1, realtime=True)
out['CLAIMS'] = df[df['realtime_start'] > end - 4].iloc[:1]
```

```
# Plot release dates of series in FRED-MD
release = Series({k: str(min(v['realtime_start'])) if v is not None and len(v)
                  else None for k,v in out.items()}).sort_values()
```

```
fig, ax = plt.subplots(clear=True, num=1, figsize=(13, 5))
ax.plot(pd.to_datetime(release, errors='coerce'))
ax.axvline(release[~release.isnull()].index[-1], c='r')
ax.set_title(f"Current ({end}) FRED-MD series, retrieved {today}")
ax.set_ylabel('First Release Date')
ax.set_xticks(np.arange(len(release)))
ax.set_xticklabels(release.index, rotation=90, fontsize='xx-small')
plt.tight_layout()
```



```
# Check if recently released data available to update latest FRED-MD
md_missing = md_df.iloc[-1]
md_missing = md_missing[md_missing.isnull()]
print("Recent values available to update missing in current FRED-MD")
for series_id in md_missing.index:
    print(alf.splice(series_id).iloc[-3:])
```

```
Recent values available to update missing in current FRED-MD
date
20241031    1538666.0
20241130    1544822.0
20241231    1555153.0
Name: CMRMTSPL, dtype: float64
date
20241031    7839
20241130    8156
20241231    7600
Name: HWI, dtype: int64
date
20241130    1.145345
20241231    1.103689
20250131      NaN
Name: HWIURATIO, dtype: float64
date
20241031    248120.0
20241130    248160.0
20241231    248851.0
Name: ACOGNO, dtype: float64
date
20241031    2585582.0
20241130    2588757.0
20241231    2584314.0
Name: BUSINV, dtype: float64
date
20241031    1.37
20241130    1.37
20241231    1.35
Name: ISRATIO, dtype: float64
date
20241031    3736897.53
20241130    3745366.76
20241231    3763355.59
Name: NONREVSL, dtype: float64
date
20241130    149.697308
20241231    149.793644
20250131      NaN
Name: CONSPI, dtype: float64
date
20241231    37.90
20250131    37.66
20250228    37.53
Name: S&P PE ratio, dtype: float64
date
20241031    554951.25
20241130    556075.09
20241231    558854.68
Name: DTCOLNVHFN, dtype: float64
date
20241031    938525.34
20241130    941204.79
20241231    946489.00
Name: DTCTHFNM, dtype: float64
```

```
# Find any missing series observations, if any, now available to update current FRED-  
→MD  
Series(release.values, index=[(s, alf.header(s)) for s in release.index])\  
.tail(len(md_missing))
```

```
(W875RX1, Real personal income excluding current transfer receipts) →  
↳ 20250131  
(ACOGNO, Manufacturers' New Orders: Consumer Goods) →  
↳ 20250204  
(HWI, Help Wanted Index for United States) →  
↳ 20250204  
(NONREVSL, Nonrevolving Consumer Credit Owned and Securitized) →  
↳ 20250207  
(CONSPPI, Nonrevolving consumer credit to Personal Income) →  
↳ 20250207  
(BUSINV, Total Business Inventories) →  
↳ 20250214  
(ISRATIO, Total Business: Inventories to Sales Ratio) →  
↳ 20250214  
(CMRMTSPL, Real Manufacturing and Trade Industries Sales) →  
↳ 20250228  
(DTCOLNVHFNM, Consumer Motor Vehicle Loans Owned by Finance Companies, Level) →  
↳ 20250228  
(DTCTHFN, Total Consumer Loans and Leases Owned and Securitized by Finance  
↳ Companies, Level) 20250228  
(COMPAPFF, 3-Month Commercial Paper Minus FEDFUNDS) →  
↳ None  
dtype: object
```

8.4 Outliers

1. **Interquartile Range (IQR) Approach** – Filters data within $\text{median} \pm 10$ times the interquartile range to minimize extreme values.
2. **Tukey's Rule** – Proposed by John Tukey, this method classifies data points as “outliers” if they fall beyond 1.5 times the interquartile range (IQR) of the first or third quartile, that is outside of $[\text{Q1} - 1.5(\text{Q3}-\text{Q1}), \text{Q3} + 1.5(\text{Q3}-\text{Q1})]$, and as “far out” if beyond 3 times the IQR.

```
payems = alf('PAYEMS', freq=freq, realtime=True, diff=1, log=1).dropna().iloc[:, 0]  
payems
```

```
date  
19390228 0.005898  
19390331 0.005962  
19390430 -0.006162  
19390531 0.006789  
19390630 0.006678  
...  
20240930 0.001517  
20241031 0.000278  
20241130 0.001647  
20241231 0.001934
```

(continues on next page)

(continued from previous page)

```
20250131    0.000899
Name: PAYEMS, Length: 1032, dtype: float64
```

```
for method in ['tukey', 'farout', 'iq10']:
    print(f"Outliers fraction detected by {method}:", np.mean(is_outlier(payems, method=method)).round(4))
payems.iloc[is_outlier(payems, method='iq10')]
```

```
Outliers fraction detected by tukey: 0.0969
Outliers fraction detected by farout: 0.0329
Outliers fraction detected by iq10: 0.0029
```

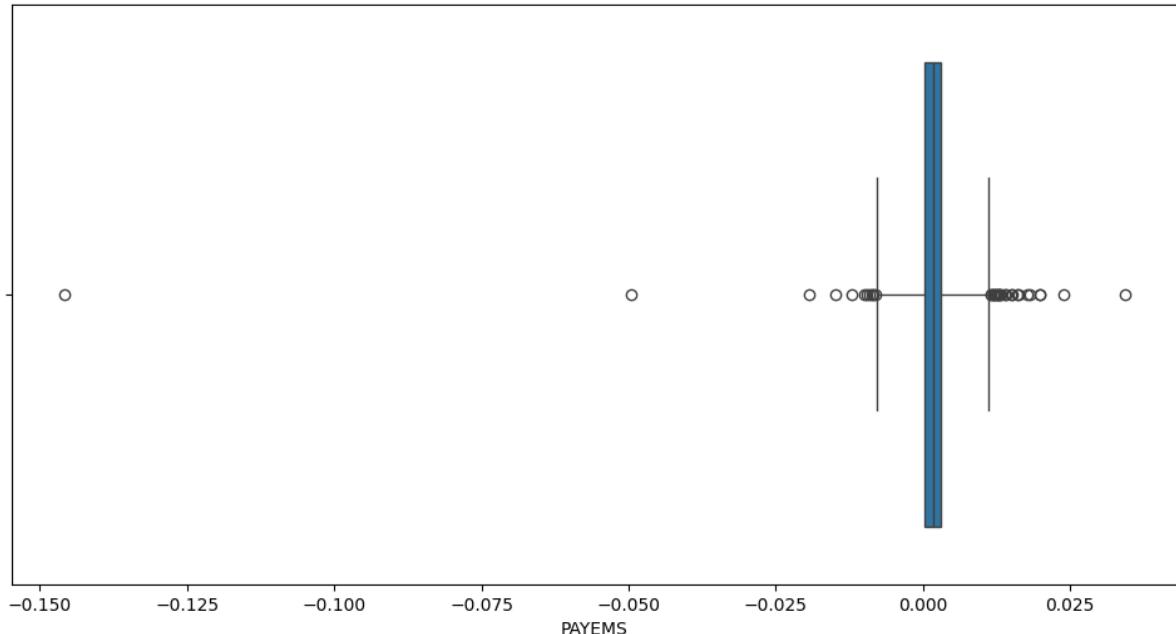
```
date
19450930   -0.049622
20200430   -0.145794
20200630    0.034217
Name: PAYEMS, dtype: float64
```

Box-and-whiskers plot

A box plot shows the quartiles of the data while the whiskers extend to show the rest of the distribution, except for points that are determined to be “outliers”, which are more than some multiple of the inter-quartile range (IQR) beyond the first and third quartiles.

```
import seaborn as sns
fig, ax = plt.subplots(figsize=(12, 6))
sns.boxplot(payems, ax=ax, orient='h', whis=3) # whiskers at 3xIQR
```

```
<Axes: xlabel='PAYEMS'>
```



References:

<https://fred.stlouisfed.org/>

<https://www.stlouisfed.org/research/economists/mccracken/fred-databases>

McCracken, M. W., & Ng, S. (2016). FRED-MD: A Monthly Database for Macroeconomic Research. *Journal of Business & Economic Statistics*, 34(4), 574–589.

McCracken, M.W., Ng, S., 2020. FRED-QD: A Quarterly Database for Macroeconomic Research, Federal Reserve Bank of St. Louis Working Paper 2020- 005

Katrina Stierholz, 2018, Economic Data Revisions: What They Are and Where to Find Them <https://journals.ala.org/index.php/dttp/article/view/6383/8404>

LINEAR REGRESSION DIAGNOSTICS

In economics, the majority is always wrong - John Kenneth Galbraith

For a linear regression model to produce reliable and interpretable results, it must satisfy certain assumptions. Diagnosing potential issues such as heteroskedasticity, multicollinearity, omitted variables, and influential data points is important for model validity. We explore key diagnostic tests for linear regression, including methods for detecting violations of assumptions, evaluating the impact of outliers, and assessing model fit through residual plots. Additionally, we discuss techniques for robust standard errors when assumptions are violated, such as heteroskedasticity- and autocorrelation-consistent (HAC) estimators.

```
# By: Terence Lim, 2020-2025 (terence-lim.github.io)
import numpy as np
import pandas as pd
import pandas as pd
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
import patsy
import statsmodels.formula.api as smf
from statsmodels.stats.outliers_influence import variance_inflation_factor
from finds.readers import Alfred
from finds.utils import plot_fitted, plot_leverage, plot_scale, plot_qq
from secret import credentials
VERBOSE = 0
# matplotlib qt
```

```
alf = Alfred(api_key=credentials['fred']['api_key'])
```

For this analysis, we retrieve monthly Consumer Price Index (CPI) data as the dependent (endogenous) variable and Producer Price Index (PPI) data as the independent (exogenous) variable. The model uses the monthly differences of the logarithms of both series to account for changes over time.

```
# difference of logs of CPI and PPI monthly series from FRED
series_id, freq, start = 'CPIAUCSL', 'M', 0    #19740101
exog_id = 'WPSFD4131'

data = pd.concat([alf(s, start=start) for s in [series_id, exog_id]], axis=1)
data.index = pd.DatetimeIndex(data.index.astype(str))
data = np.log(data).diff().dropna()  # model the changes in logs of the series
DataFrame.from_dict({s: alf.header(s) for s in [series_id, exog_id]}, orient='index', columns=['Description'])
```

	Description
CPIAUCSL	Consumer Price Index for All Urban Consumers: ...
WPSFD4131	Producer Price Index by Commodity: Final Deman...

9.1 Model assumptions

A valid linear regression model must satisfy the following assumptions:

1. **Linearity:** The expected value of y_i follows a linear function of the independent variables:

$$E[y_i] = b_0 + b_1 x_{i1} + \dots + b_k x_{ik}$$
2. **Exogeneity:** The explanatory variables $\{x_{i1}, \dots, x_{ik}\}$ are non-stochastic and not correlated with the error term.
3. **Homoscedasticity:** The variance of the dependent variable remains constant:

$$Var(y_i) = \sigma^2$$
4. **Independence:** The observations $\{y_i\}$ are independent of each other.
5. **Normality:** The error terms follow a normal distribution.

When assumptions 1-4 hold, the least squares estimator:

- Provides an **unbiased** estimate of the regression coefficients:

$$\hat{b} = (X'X)^{-1}X'y$$
- Has a variance-covariance matrix:

$$Var(\hat{b}) = \sigma^2(X'X)^{-1}$$
- The standard error for each coefficient b_j is:

$$se(b_j) = \sigma \sqrt{(X'X)^{-1}_{[j+1,j+1]}}$$

When all five assumptions hold, the least squares estimator follows a normal distribution, enabling valid statistical inference.

```
# Run Linear Regression (with exog and 2 lags)
dmf = (f'{series_id} ~ {series_id}.shift(1) + {series_id}.shift(2) + {exog_id}.
       ↪shift(1) ^)
model = smf.ols(formula=dmf, data=data).fit()
print(model.summary())
```

OLS Regression Results						
Dep. Variable:	CPIAUCSL	R-squared:	0.453			
Model:	OLS	Adj. R-squared:	0.451			
Method:	Least Squares	F-statistic:	167.5			
Date:	Sun, 02 Mar 2025	Prob (F-statistic):	4.43e-79			
Time:	22:25:01	Log-Likelihood:	2815.2			
No. Observations:	610	AIC:	-5622.			
Df Residuals:	606	BIC:	-5605.			
Df Model:	3					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.
975]						

(continues on next page)

(continued from previous page)

Intercept	0.0009	0.000	6.276	0.000	0.001	0.
↳001						
CPIAUCSL.shift(1)	0.5821	0.041	14.199	0.000	0.502	0.
↳663						
CPIAUCSL.shift(2)	-0.0479	0.041	-1.174	0.241	-0.128	0.
↳032						
WPSFD4131.shift(1)	0.2011	0.036	5.609	0.000	0.131	0.
↳271						
<hr/>						
Omnibus:	116.912	Durbin-Watson:	2.051			
Prob(Omnibus):	0.000	Jarque-Bera (JB):	584.706			
Skew:	-0.752	Prob(JB):	1.08e-127			
Kurtosis:	7.555	Cond. No.	522.			
<hr/>						

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly
↳specified.

9.1.1 Heteroskedasity and HAC robust errors

If the variance of residuals is not constant (**heteroskedasticity**), the usual **Ordinary Least Squares (OLS)** standard error formula, $\sigma^2(X'X)^{-1}$, no longer holds. While OLS coefficient estimates remain **consistent**, their standard errors may be misestimated. A more general form of the variance-covariance matrix is:

$$(X'X)^{-1}(X'\Omega X)(X'X)^{-1}$$

where different choices of Ω provide **robust standard error estimators**.

- **White's (1980) heteroskedasticity-consistent estimator** (also known as the sandwich estimator) uses the diagonal of squared residuals.
- Alternative estimators account for leverage effects in the design matrix.

If error terms exhibit **serial correlation**, standard heteroskedasticity-robust errors may still be misleading. **Newey and West (1987)** introduced the **Heteroskedasticity and Autocorrelation Consistent (HAC)** estimator, which applies a weighting scheme to correct for autocorrelation. The truncation parameter for lag selection is often chosen as:

$$m = 0.75T^{1/3}$$

where autocorrelation coefficients are weighted as follows:

$$1 + 2 \sum_{j=1}^{m-1} \frac{m-j}{m} \hat{\rho}_j$$

```
robust = model.get_robustcov_results(cov_type='HAC', use_t=None, maxlags=0)
print(robust.summary())
```

OLS Regression Results

Dep. Variable:	CPIAUCSL	R-squared:	0.453
Model:	OLS	Adj. R-squared:	0.451
Method:	Least Squares	F-statistic:	108.8

(continues on next page)

(continued from previous page)

```

Date: Sun, 02 Mar 2025 Prob (F-statistic): 2.25e-56
Time: 22:25:01 Log-Likelihood: 2815.2
No. Observations: 610 AIC: -5622.
Df Residuals: 606 BIC: -5605.
Df Model: 3
Covariance Type: HAC
=====
            coef    std err      t    P>|t|    [0.025    0.
c975]
-----
Intercept  0.0009  0.000  5.779  0.000  0.001  0.
c001
CPIAUCSL.shift(1)  0.5821  0.074  7.903  0.000  0.437  0.
c727
CPIAUCSL.shift(2) -0.0479  0.053 -0.902  0.367 -0.152  0.
c056
WPSFD4131.shift(1)  0.2011  0.055  3.652  0.000  0.093  0.
c309
=====
Omnibus: 116.912 Durbin-Watson: 2.051
Prob(Omnibus): 0.000 Jarque-Bera (JB): 584.706
Skew: -0.752 Prob(JB): 1.08e-127
Kurtosis: 7.555 Cond. No. 522.
=====

Notes:
[1] Standard Errors are heteroscedasticity and autocorrelation robust (HAC) using
    0 lags and without small sample correction
    
```

9.1.2 Multicollinearity and variance inflation factors

Multicollinearity arises when explanatory variables are highly correlated, leading to unstable coefficient estimates. The **Variance Inflation Factor (VIF)** quantifies the degree of multicollinearity by measuring how much a predictor's variance is inflated due to correlation with other predictors:

$$VIF_i = \frac{1}{1 - R_i^2}$$

where R_i^2 is obtained by regressing X_i on all other predictors.

A $VIF > 5$ or 10 suggests that the variable is highly collinear with other explanatory variables, potentially leading to large standard errors and unreliable estimates.

```

Y, X = patsy.dmatrices(dmf + ' - 1', data=data) # exclude intercept term
print("Variance Inflation Factors")
Series({X.design_info.column_names[i]: variance_inflation_factor(X, i)
        for i in range(X.shape[1])}, name='VIF').to_frame()
    
```

Variance Inflation Factors

	VIF
CPIAUCSL.shift(1)	3.414717

(continues on next page)

(continued from previous page)

```
CPIAUCSL.shift(2)    3.404137
WPSFD4131.shift(1)  2.285974
```

9.1.3 Omitted variables

Leaving out an important variable from the regression model can lead to biased estimates. The consequences are:

1. **Bias in Included Variables:** If the omitted variable is correlated with included variables, their regression coefficients will capture some of its effect, leading to inconsistent estimates.
2. **Inflated Residual Variance:** The estimated residuals will include both true shocks and the effects of the omitted variable, reducing model accuracy.

Conversely, including an extraneous (irrelevant) variable does not introduce bias but increases standard errors, making it harder to detect significant effects.

9.2 Residual diagnostics

Residual plots help evaluate model fit, identify outliers and detect potential specification issues.

An ideal model would have residuals that are not systematically related to any of the included explanatory variables. Standardized residuals may alternatively be used so that the magnitude of deviation is more apparent.

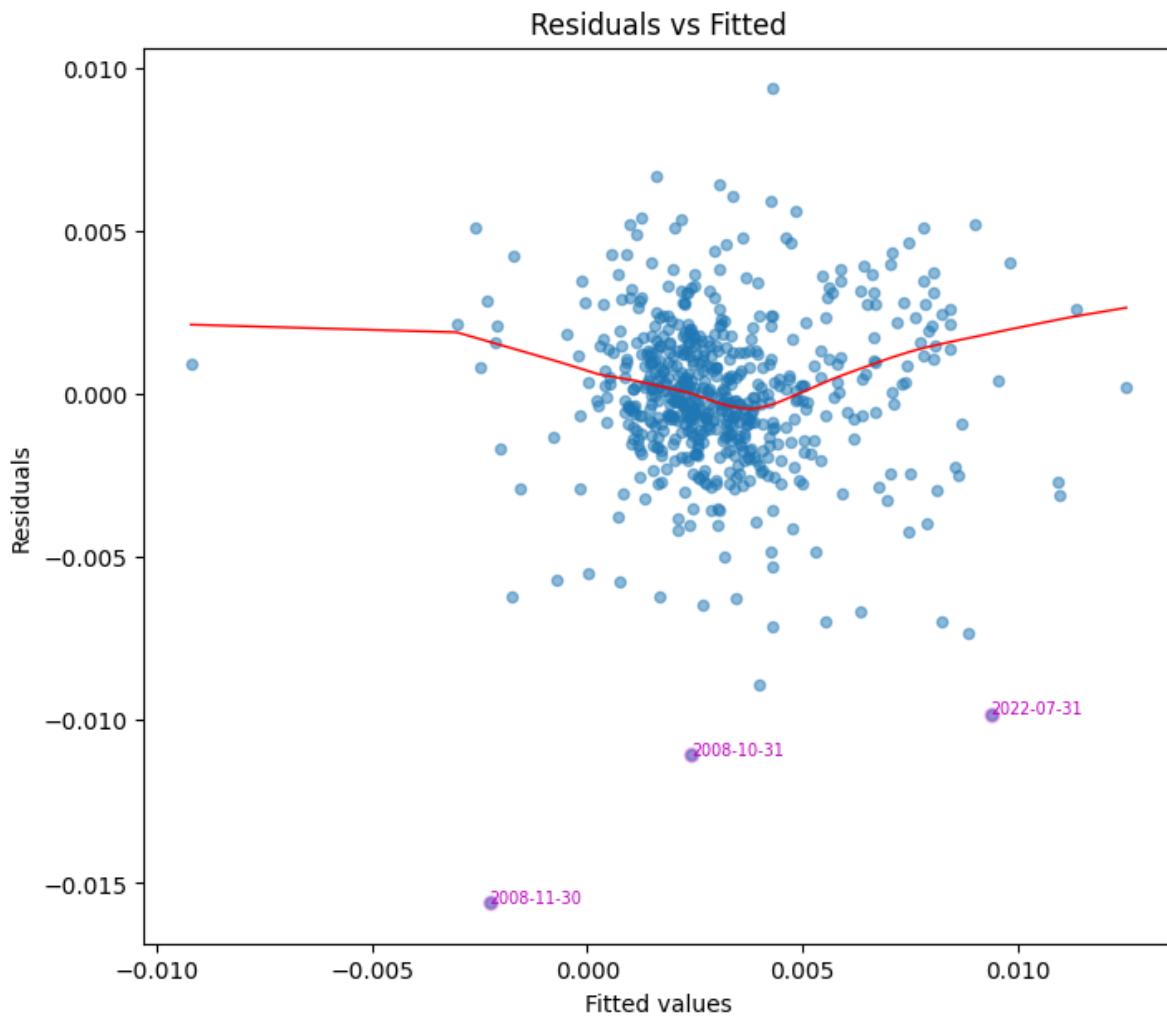
9.2.1 Residuals vs fitted plot

This plot assesses whether residuals exhibit nonlinear patterns. Ideally, residuals should be randomly scattered around zero, with no discernible trend. A systematic pattern suggests model misspecification or omitted variables.

```
# Plot residuals and identify outliers
fig, ax = plt.subplots(clear=True, figsize=(8, 7))
z = plot_fitted(fitted=model.fittedvalues,
                 resid=model.resid,
                 ax=ax)
print("Residual Outliers")
z.to_frame().T
```

Residual Outliers

date	2022-07-31	2008-10-31	2008-11-30
outliers	-0.009815	-0.011066	-0.015599



9.2.2 Normal QQ plot

A **quantile-quantile (Q-Q) plot** compares residuals to a normal distribution. If residuals are normally distributed, data points should align along a 45-degree reference line. **Outliers** may appear as deviations from this line, indicating potential issues such as a heavy-tailed distribution.

```
fig, ax = plt.subplots(clear=True, figsize=(8, 7))
plot_qq(model.resid, ax=ax)
```

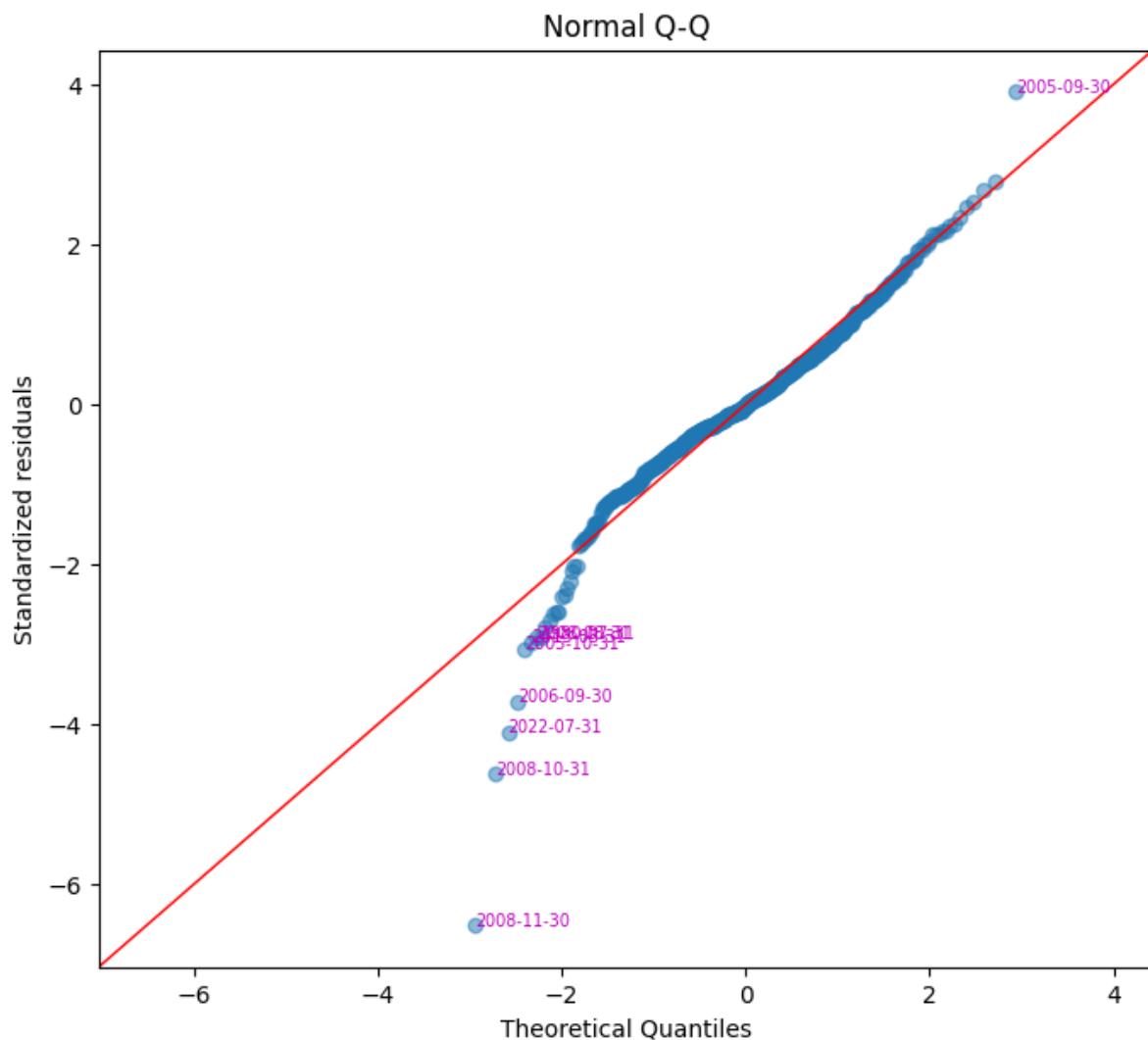
```
/home/terence/env3.11/lib/python3.11/site-packages/statsmodels/graphics/gofplots.
  ↪py:1043: UserWarning: color is redundantly defined by the 'color' keyword
  ↪argument and the fmt string "b" (-> color=(0.0, 0.0, 1.0, 1)). The keyword
  ↪argument will take precedence.
  ax.plot(x, y, fmt, **plot_style)
```

	residuals	standardized
date		
2008-11-30	-0.015599	-6.511279

(continues on next page)

(continued from previous page)

2008-10-31	-0.011066	-4.619178
2022-07-31	-0.009815	-4.097059
2006-09-30	-0.008903	-3.716483
2005-10-31	-0.007321	-3.056102
2013-03-31	-0.007108	-2.967065
2008-08-31	-0.007003	-2.923170
1980-07-31	-0.006997	-2.920733
2005-09-30	0.009362	3.907884

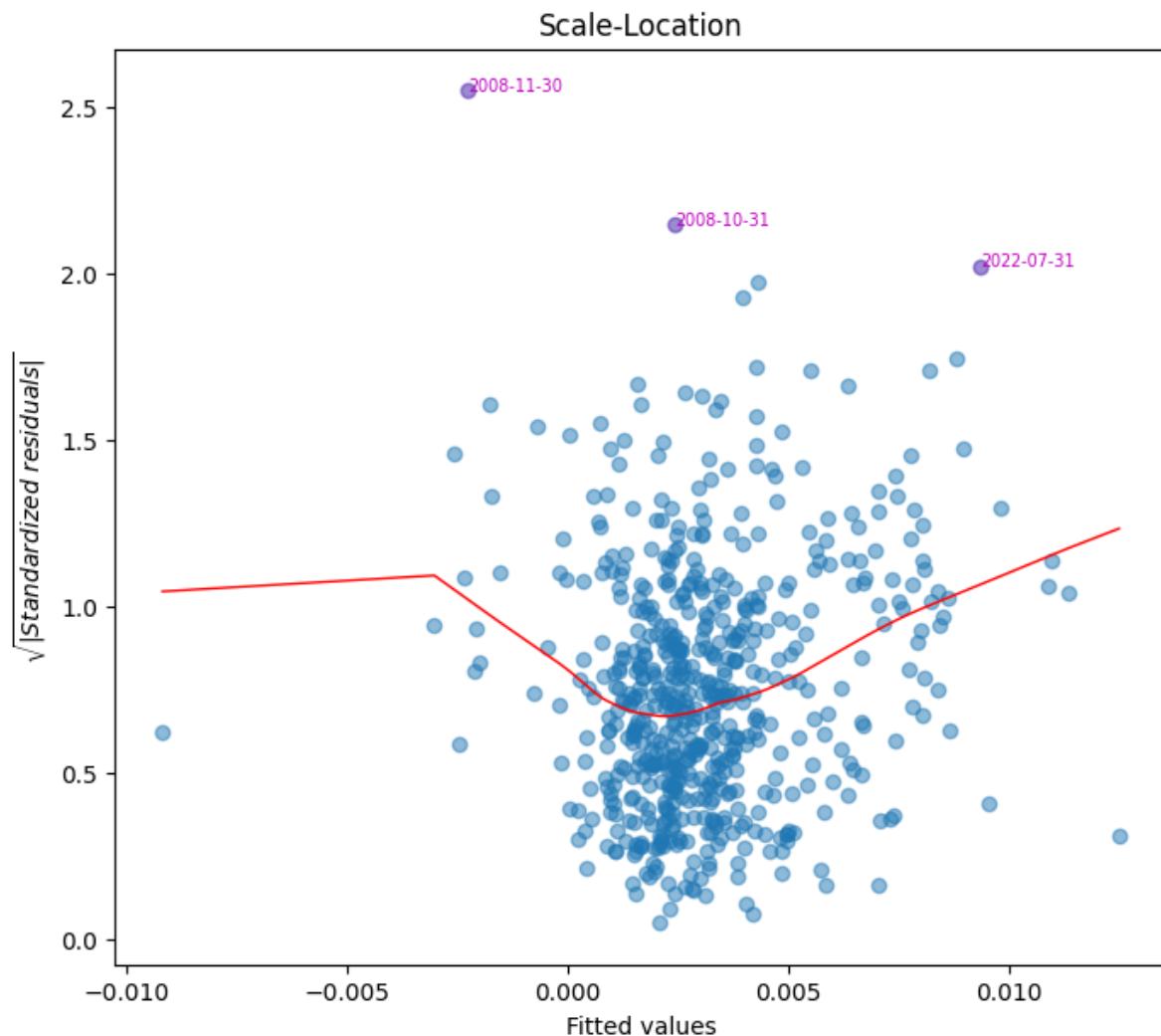


9.2.3 Scale-location plot

This plot checks for homoscedasticity (constant variance). Residuals should be evenly spread across predictor values. A funnel-shaped pattern suggests heteroskedasticity, requiring robust standard errors.

```
fig, ax = plt.subplots(clear=True, figsize=(8, 7))
plot_scale(model.fittedvalues, model.resid, ax=ax)
```

```
array([579, 414, 415])
```



9.2.4 Leverage and influential points

Certain data points can disproportionately affect regression estimates. The projection matrix from the least squares estimator, also called the **hat matrix**, $H = X(X^T X)^{-1} X^T$ identifies **leverage points**, where the diagonal element h_{ii} measures the influence of the i -th observation.

A point may have **high leverage** but not necessarily influence the regression results significantly. **Cook's Distance** measures influence based on both residual magnitude and leverage:

$$D_i = \frac{1}{p} t_i^2 \frac{h_{ii}}{1 - h_{ii}}$$

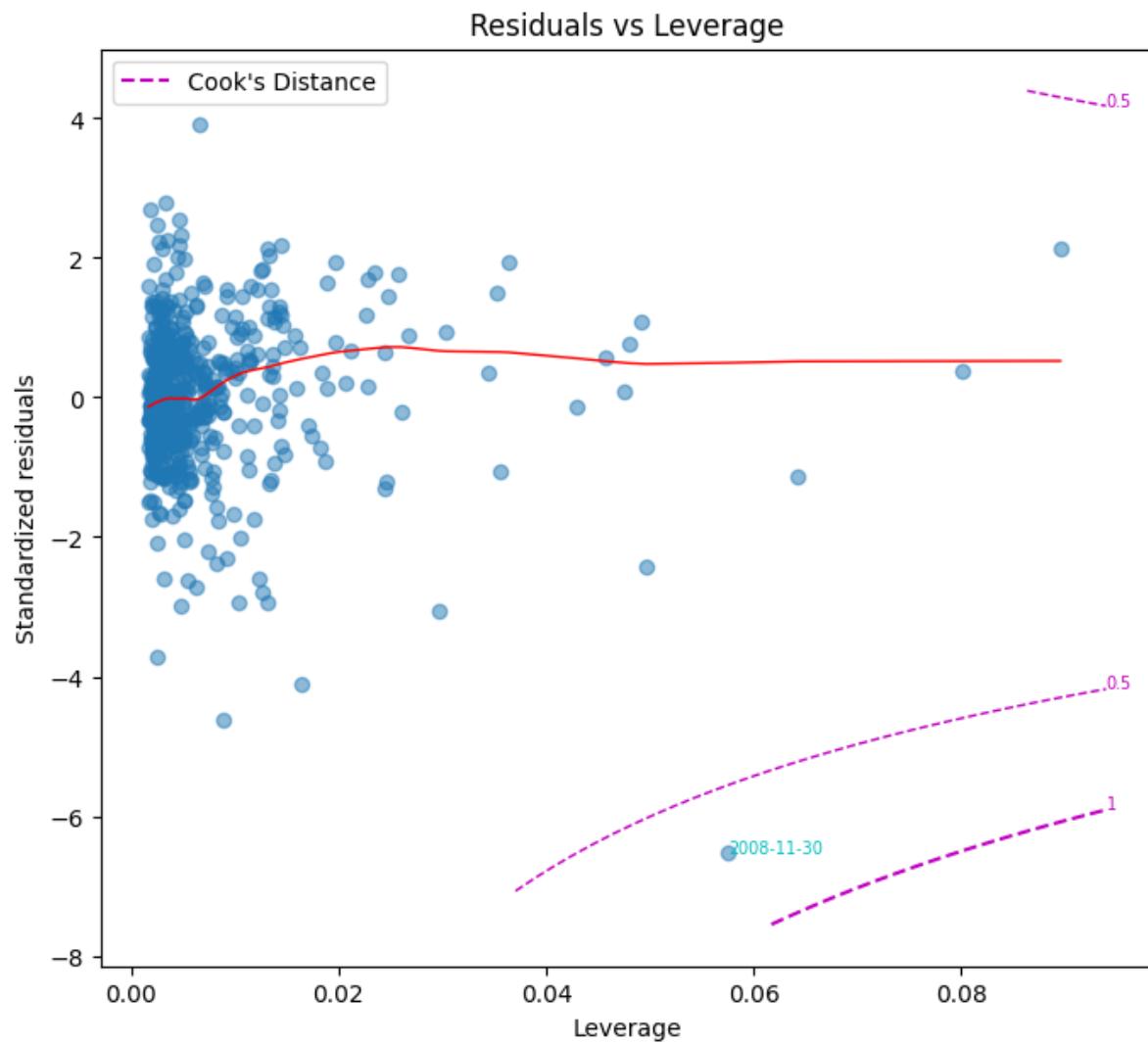
where:

- p is the number of regression parameters.
- $t_i = \frac{\hat{\epsilon}_i}{\hat{\sigma}(1-h_{ii})}$ is the *studentized residual*, which accounts for non-constant variance.
- $\hat{\sigma} = \sqrt{\sum_{j=1}^n \hat{\epsilon}_j^2 / n}$.

$D_i > 1$ suggests an influential point that may need further investigation. The **Residuals vs Leverage plot** helps visualize these influential points.

```
fig, ax = plt.subplots(clear=True, figsize=(8, 7))
plot_leverage(model.resid, model.get_influence().hat_matrix_diag,
               model.get_influence().cooks_distance[0],
               ddof=len(model.params), ax=ax)
```

```
Empty DataFrame
Columns: [influential, cook's D, leverage]
Index: []
```



References:

White, Halbert (1980). "A Heteroskedasticity-Consistent Covariance Matrix Estimator and a Direct Test for Heteroskedasticity". *Econometrica*. 48 (4): 817–838.

Newey, Whitney K., and Kenneth D. West. 1987. "A Simple, Positive Semi-Definite, Heteroskedasticity and Autocorrelation Consistent Covariance Matrix." *Econometrica* 55: 703–8.

<https://library.virginia.edu/data/articles/diagnostic-plots>

FRM Part I Exam Book Quantitative Analysis Ch 9

TIME SERIES ANALYSIS

The only thing we know about the future is that it will be different - Peter Drucker

Time series analysis is a fundamental statistical technique used to analyze data points collected over time. A time series can exhibit multiple components, such as long-term trends, cyclical fluctuations, seasonal variations, and irregular random movements. Understanding these components enables analysts to build accurate predictive models. We explore key concepts in time series analysis, including trend patterns, seasonality, stationarity, autocorrelation, and various statistical models such as autoregressive (AR), moving average (MA), and autoregressive moving average (ARMA) models. We also examine forecasting methods and statistical tests used to assess time series properties.

```
# By: Terence Lim, 2020-2025 (terence-lim.github.io)
import numpy as np
import pandas as pd
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.tsa.ar_model import AutoReg, ar_select_order
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf, acf
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tsa.stattools import grangercausalitytests, adfuller
from statsmodels.tsa.api import VAR
import statsmodels.formula.api as smf
from sklearn.metrics import mean_squared_error
from finds.readers import Alfred
from secret import credentials
VERBOSE = 0
# %matplotlib qt
```

We retrieve the Industrial Production (IP) total index monthly time series from FRED.

```
series_id, freq, start = 'IPB50001N', 'ME', 0 # not seasonally adjusted
alf = Alfred(api_key=credentials['fred']['api_key'])
df = alf(series_id, log=1, freq=freq, start=start).dropna()
date_title = f" ({df.index[0]//100}-{df.index[-1]//100})"
df.index = pd.to_datetime(df.index, format='%Y%m%d')
df.index.freq = freq # set index to datetime type with 'M frequency
alf.header(series_id)
```

'Industrial Production: Total Index'

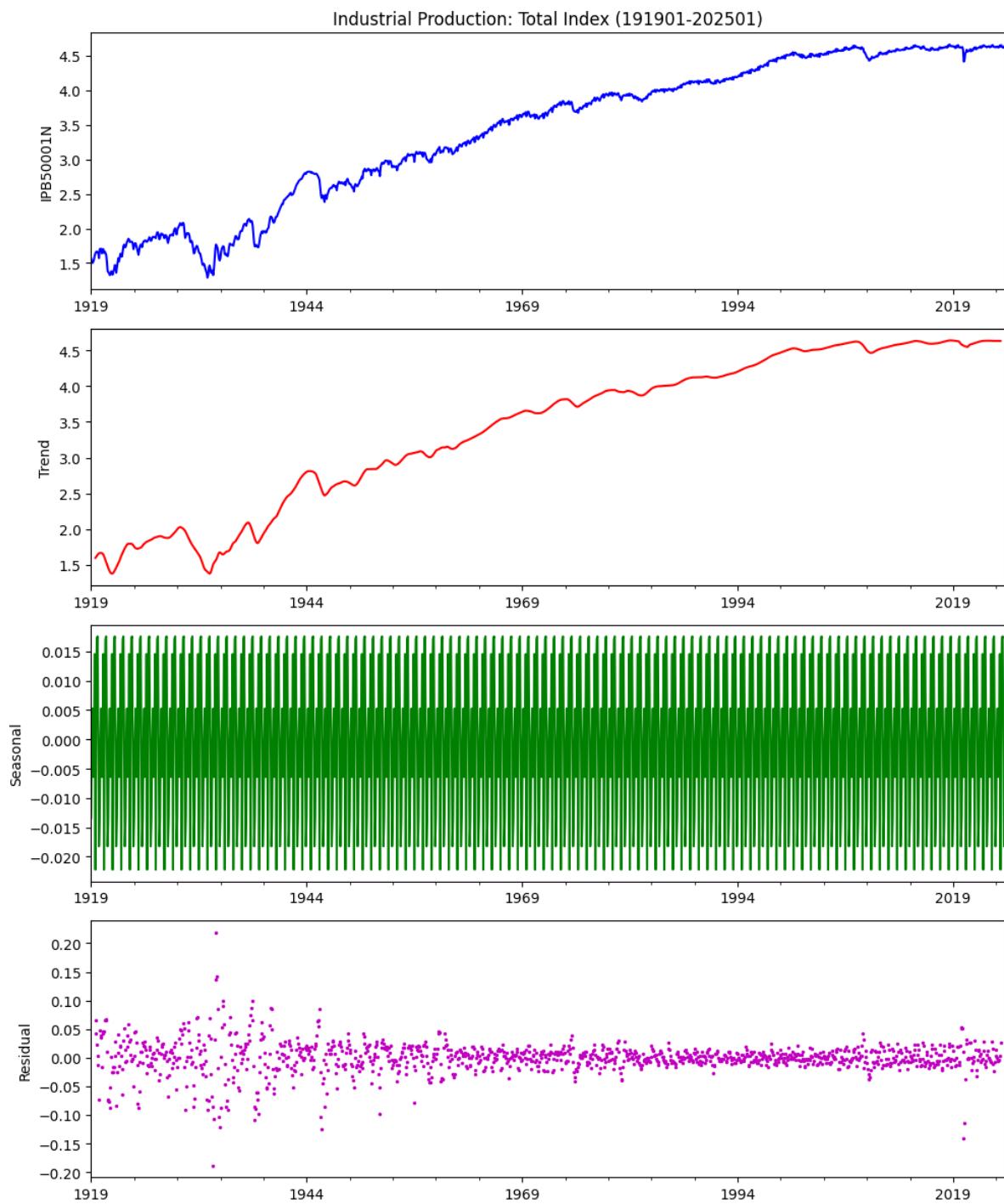
10.1 Seasonality

A time series can be broken down into three key components:

- **Trend** – Represents long-term movements in the data.
- **Seasonality** – Captures predictable and recurring changes within a specific time frame.
- **Cyclical** – Encompasses longer-term cycles that are influenced by broader economic or structural factors.

If a time series Y_t exhibits seasonality, its mean can differ across periods, repeating every s periods (e.g., quarterly with $s = 4$ or monthly with $s = 12$). Deterministic seasonal effects can be effectively modeled using dummy variables.

```
## Seasonality Decomposition Plot
result = seasonal_decompose(df, model='add')
fig, ax = plt.subplots(nrows=4, ncols=1, clear=True, figsize=(10, 12))
result.observed.plot(ax=ax[0], title=alf.header(result.observed.name) + date_title,
                      ylabel=result.observed.name, xlabel='', c='b')
result.trend.plot(ax=ax[1], ylabel='Trend', xlabel='', c='r')
result.seasonal.plot(ax=ax[2], ylabel='Seasonal', xlabel='', c='g')
result.resid.plot(ax=ax[3], ls=' ', ms=3, marker='.', c='m',
                  ylabel='Residual', xlabel='')
plt.tight_layout()
```



10.2 Stationarity

A time series is considered **covariance-stationary** if it meets three key conditions:

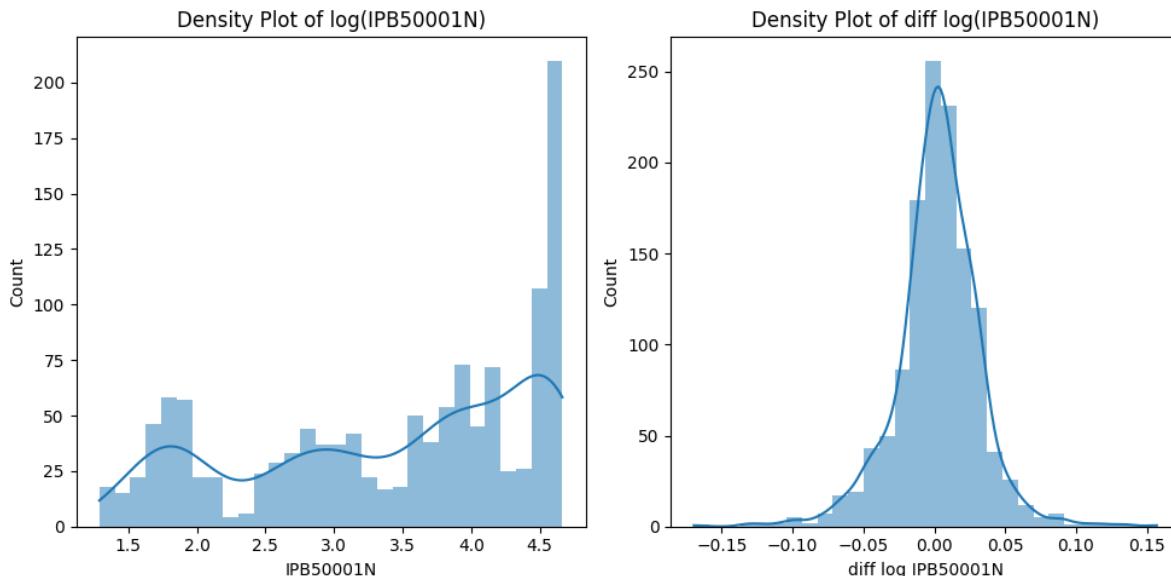
1. The mean remains constant over time, i.e., $E[Y_t] = m$ for all t .
2. The variance is finite and does not change over time, i.e., $V[Y_t] = \gamma_0 < \infty$.
3. The autocovariance depends only on the time lag h and not on t , i.e., $Cov[Y_t, Y_{t-h}] = \gamma_h$.

Covariance stationarity depends on the first two moments of a time series. Stationarity is crucial for modeling and forecasting because stationary series exhibit stable statistical properties. The **Augmented Dickey-Fuller (ADF) test** helps determine whether a time series has a unit root, with the null hypothesis stating that a unit root is present. If the p-value is below a critical threshold, the null hypothesis can be rejected, indicating stationarity.

```
values = df.diff().dropna().values.squeeze()
adf = adfuller(values)
DataFrame.from_dict({"I(1)": list(adf[:4]) + list(adf[4].values())},
                    orient='index',
                    columns=['Test Statistic', 'p-value', 'Lags Used', 'Obs Used'] +
                    [f"critical({k})" for k in adf[4].keys()]).round(3)
```

	Test Statistic	p-value	Lags Used	Obs Used	critical(1%)	\
I (1)	-7.266	0.0	23	1248	-3.436	
	critical(5%)	critical(10%)				
I (1)	-2.864	-2.568				

```
# Histogram Plot and Kernel Density Estimate
fig, axes = plt.subplots(1, 2, clear=True, figsize=(10,5))
sns.histplot(df.dropna(), bins=30, lw=0, kde=True, ax=axes[0])
axes[0].set_title(f"Density Plot of log({series_id})")
sns.histplot(df.diff().dropna().rename(f"diff log {series_id}"),
             bins=30, lw=0, kde=True, ax=axes[1]) #line_kws={"color": "r"}
axes[1].set_title(f"Density Plot of diff log({series_id})")
plt.tight_layout()
```



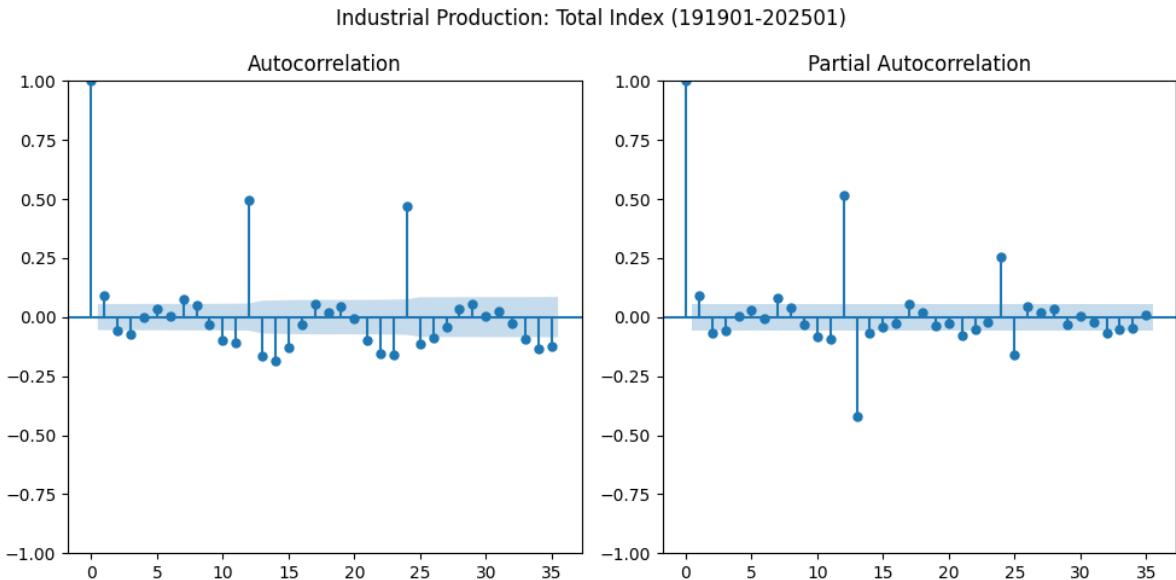
10.3 Autocorrelation

Autocorrelation measures the relationship between time series observations at different points in time. The autocorrelation function (ACF) at lag h is given by:

$$\rho_h = \frac{\gamma_h}{\gamma_0} \text{ where } \gamma_h \text{ is the autocovariance at lag } h, \text{ and } \gamma_0 \text{ is the variance.}$$

The **partial autocorrelation function (PACF)** isolates the direct relationship between observations separated by lag h , removing the influence of intermediate lags (i.e., $Y_{t-1}, Y_{t-2}, \dots, Y_{t-h+1}$).

```
values = df.diff().dropna().values.squeeze()
fig, axes = plt.subplots(1, 2, clear=True, figsize=(10, 5))
plot_acf(values, lags=35, ax=axes[0])
plot_pacf(values, lags=35, ax=axes[1], method='ywm')
plt.suptitle(alf.header(result.observed.name) + date_title)
plt.tight_layout()
```



10.4 Time Series Models

10.4.1 Autoregressive (AR) models

Autoregressive (AR) models describe a time series in terms of its past values. An AR(1) process follows:

$$Y_t = \delta + \phi Y_{t-1} + \epsilon_t \text{ where } \delta \text{ is the intercept, } \phi \text{ is the autoregressive coefficient, and } \epsilon_t \text{ is white noise.}$$

For a higher-order AR(p) model:

$$Y_t = \mu + \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \dots + \phi_p Y_{t-p} + \epsilon_t$$

The PACF for an AR process is nonzero only for the first p lags, while the ACF gradually decays ($\rho(h) = \phi^{|h|}$)

10.4.2 Moving average (MA) models**

A moving average model expresses Y_t as a function of past shocks:

$$Y_t = \mu + \theta_1 \epsilon_{t-1} + \epsilon_t \text{ An MA}(q) \text{ model extends this to include } q \text{ lags of } \epsilon_t: Y_t = \mu + \epsilon_t + \theta_1 \epsilon_{t-1} + \dots + \theta_q \epsilon_{t-q}$$

The ACF of an MA process is zero for lags greater than q , while the PACF has a more complex structure.

10.4.3 ARMA models

An **ARMA(p, q)** model combines AR and MA components. For example, a simple ARMA(1,1) evolves according to:

$$Y_t = \mu + \phi Y_{t-1} + \theta \epsilon_{t-1} + \epsilon_t$$

- The mean of this process is $\mu = \delta / (1 - \phi)$
- The variance is $\gamma_0 = \sigma^2 (1 + 2\phi\theta + \theta^2) / (1 - \phi^2)$

An ARMA(1,1) process is covariance-stationary if $|\phi| < 1$. The MA coefficient plays no role in determining whether the process is covariance-stationary, because any MA is covariance-stationary as the MA component only affects a single lag of the shock. The AR component, however, affects all lagged shocks and so if ϕ_1 is too large, then the time series is not covariance-stationary.

10.4.4 Lag lengths

Determining the appropriate lag lengths for the AR and MA components (i.e., p and q , respectively) is a key challenge when building an ARMA model. The first step in model building is to inspect the sample autocorrelation and sample PACFs.

The **Box-Pierce** test statistic is the sum of the squared autocorrelations scaled by the sample size T : $Q_{BP} = T \sum_{i=1}^h \left(\frac{T+1}{T-1} \right) \hat{\rho}_i^2$

When the null is true, Q_{BP} is asymptotically distributed as a χ_h^2 variable.

The **Ljung-Box** statistic is a modified version of the Box-Pierce statistic that works better in smaller samples, and is defined as: $Q_{LB} = T \sum_{i=1}^h \left(\frac{T+2}{T-i} \right) \hat{\rho}_i^2$

10.4.5 Seasonal component

Seasonal components can be added to the short-term components of an ARMA(p,q) model, by using lags only at the seasonal frequency. A seasonal ARMA combines these two components into a single specification:

$$ARMA(p, q) \times (p_s, q_s)_f$$

where p and q are the orders of the short-run lag polynomials, p_s and q_s represent seasonal lag orders, and f denotes the seasonal horizon (e.g., every 3 or 12 months with monthly observations).

10.4.6 Unit Roots

Random walks are most important source of **non-stationarity** in economic time series. A simple random walk process evolves according to:

$$Y_t = Y_{t-1} + \epsilon_t$$

Unit roots generalize random walks by adding short-run stationary dynamics to the long-run random walk.

Spurious relationships can occur when non-stationary series are regressed against each other; this produces a coefficient estimate that is large and seemingly statistically different from zero when using conventional statistical distributions to obtain the critical values.

Differencing removes unit roots and prevents such misleading results. If Y_t has a unit root (i.e. has **integration order** equal to 1), then the difference: $\Delta Y_t = Y_t - Y_{t-1}$ does not.

10.4.7 SARIMAX model

The **Seasonal AutoRegressive Integrated Moving Average with eXogenous Regressors (SARIMAX)** model is denoted:

$(p, d, q) \times (P, D, Q)_s$

where:

- (p, d, q) are the orders of AR, differencing, and MA components.
- (P, D, Q, s) are the seasonal counterparts and periodicity.

```
split_date = df.index[-12]    # train/test split date
# df_train = df.loc[:split_date].dropna()
```

```
# Fit a SARIMA(1,1,3) with seasonal order (0, 0, 0, 12)
pdq = (1, 1, 1)    #(12, 1, 0)
seasonal_pdq = (0, 0, 0, 12)
arima = SARIMAX(df, order=pdq, seasonal_order=seasonal_pdq, trend='c').fit()
fig = arima.plot_diagnostics(figsize=(10, 6), lags=36)
plt.tight_layout()
arima.summary()
```

RUNNING THE L-BFGS-B CODE

```
* * *
Machine precision = 2.220D-16
N =           4      M =           10
At X0           0 variables are exactly at the bounds
At iterate      0      f= -2.09476D+00      |proj g|=  3.46166D-01
At iterate      5      f= -2.09484D+00      |proj g|=  2.67321D-02
At iterate     10      f= -2.09486D+00      |proj g|=  1.29274D-01
* * *
```

(continues on next page)

(continued from previous page)

```

Tit  = total number of iterations
Tnf  = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

* * *

```

N	Tit	Tnf	Tnint	Skip	Nact	Projg	F
4	14	18	1	0	0	8.413D-04	-2.095D+00
F =	-2.0948681083990448						

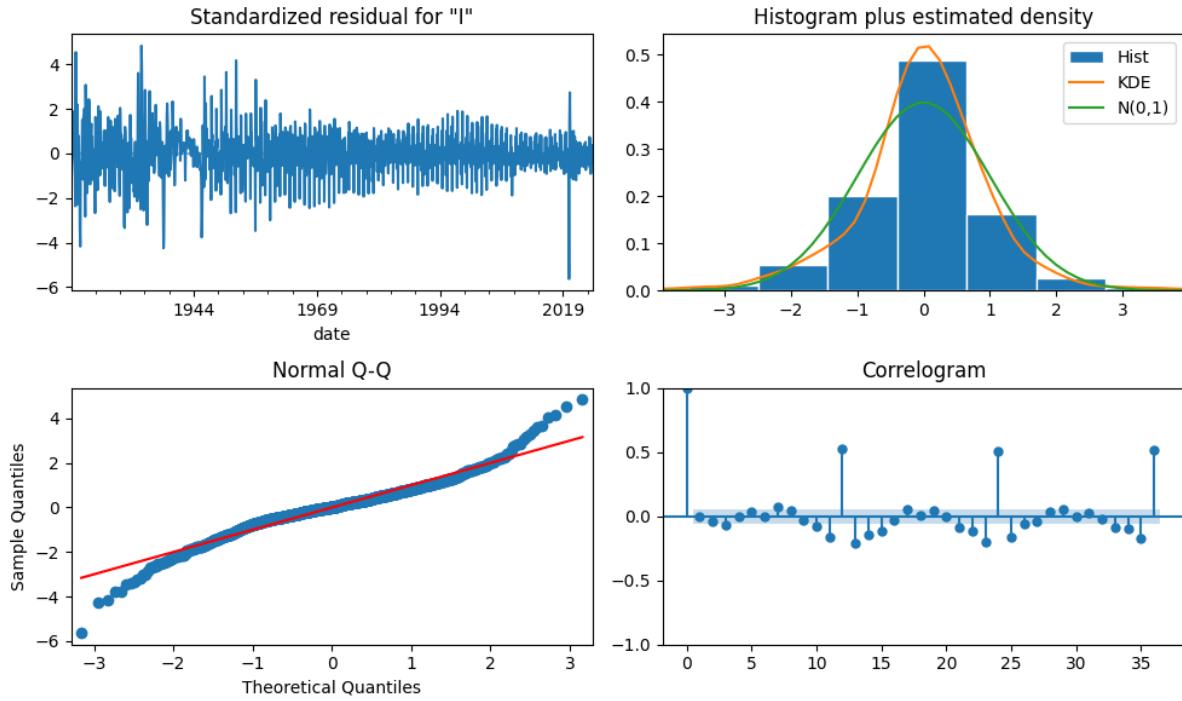
CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH

This problem is unconstrained.

Dep. Variable:	IPB50001N	No. Observations:	1273			
Model:	SARIMAX(1, 1, 1)	Log Likelihood	2666.767			
Date:	Mon, 03 Mar 2025	AIC	-5325.534			
Time:	10:54:09	BIC	-5304.941			
Sample:	01-31-1919 - 01-31-2025	HQIC	-5317.799			
Covariance Type:	opg					
	coef	std err	z	P> z	[0.025	0.975]
intercept	0.0027	0.001	2.396	0.017	0.000	0.005
ar.L1	-0.1167	0.223	-0.524	0.600	-0.553	0.320
ma.L1	0.2195	0.222	0.987	0.323	-0.216	0.655
sigma2	0.0009	2.21e-05	39.946	0.000	0.001	0.001
Ljung-Box (L1) (Q):	0.02	Jarque-Bera (JB):	563.24			
Prob(Q):	0.87	Prob(JB):	0.00			
Heteroskedasticity (H):	0.31	Skew:	-0.20			
Prob(H) (two-sided):	0.00	Kurtosis:	6.23			

Warnings:

[1] Covariance matrix calculated using the outer product of gradients (complex-step).



10.5 Forecasting

```
series_id, start = 'INDPRO', 0
df_all = alf(series_id, log=1, diff=1, start=start).dropna()
df_all.index = pd.to_datetime(df_all.index, format='%Y%m%d')
df_all.index.freq = freq
df_train = df_all[df_all.index <= split_date]
df_test = df_all[df_all.index > split_date]
```

10.5.1 AR lag order selection

The most natural measure of fit is the sample variance of the estimated residuals, also known as the Mean Squared Error (MSE) of the model. Unfortunately, choosing a model to minimize MSE also selects a specification that is far too large. The solution to this overfitting problem is to add a penalty to the MSE that increases each time a new parameter is added, employing criteria like **Akaike Information Criterion (AIC)** and **Bayesian Information Criterion (BIC)**.

- AIC is calculated by $T \ln \hat{\sigma}^2 + 2k$, where T is the sample size and k is the number of parameters.
- BIC alters the penalty and is computed by $T \ln \hat{\sigma}^2 + k \ln T$

Unlike the AIC, the BIC has a cost per parameter that slowly increases with T . Hence BIC always selects a model that is no larger than the model selected by the AIC (assuming $\ln T > 2$), and is a consistent model selection criterion (i.e., the true model is selected as $T \rightarrow \infty$).

```
lags = ar_select_order(df_train, maxlag=36, ic='bic', old_names=False).ar_lags
print('(BIC) lags= ', len(lags), ':', lags)
```

```
(BIC) lags= 1 : [1]
```

```
# Train final model on train split
model = AutoReg(df_train, lags=lags, old_names=False).fit()
print(model.summary())
```

```
AutoReg Model Results
=====
Dep. Variable: INDPRO   No. Observations: 1261
Model: AutoReg(1)   Log Likelihood: 3366.312
Method: Conditional MLE   S.D. of innovations: 0.017
Date: Mon, 03 Mar 2025   AIC: -6726.625
Time: 10:54:10   BIC: -6711.208
Sample: 03-31-1919   HQIC: -6720.831
- 02-29-2024
=====
      coef    std err      z   P>|z|   [ 0.025   0.975]
-----
const    0.0013    0.000    2.705    0.007    0.000    0.002
INDPRO.L1  0.4859    0.025   19.794    0.000    0.438    0.534
Roots
=====
      Real      Imaginary      Modulus      Frequency
-----
AR.1    2.0581    +0.0000j    2.0581    0.0000
```

10.5.2 One-step forecast

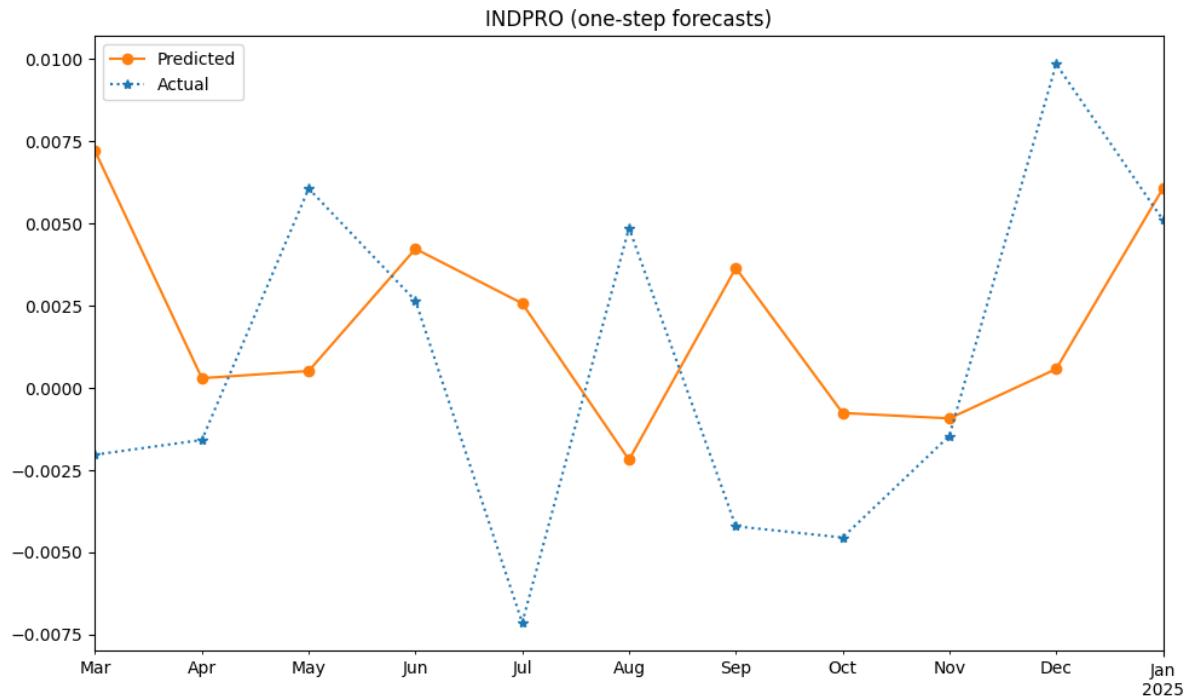
One-step forecast predicts Y_{T+1} using all available data up to time T , including the entire prior history of Y (Y_T, Y_{T-1}, \dots), as well as all values of any other variables that occurred at time T or earlier.

```
# Observations to predict are from the oos test split
test = AutoReg(df_all, lags=lags, old_names=False)
```

```
# Use model params from train split, start predictions from last train row
df_pred = test.predict(model.params, start=df_test.index[0])
mse = mean_squared_error(df_test, df_pred)
#var = np.mean(np.square(df_test - df_train.mean()))
print(f"ST Forecast({len(df_pred)}): rmse={np.sqrt(mse)}")
```

```
ST Forecast(11): rmse=0.006247508053669628
```

```
fig, ax = plt.subplots(clear=True, num=1, figsize=(10, 6))
df_pred.plot(ax=ax, c='C1', ls='-', marker='o', label='Predicted')
df_test.plot(ax=ax, c='C0', ls=':', marker='*', label='Actual')
ax.legend()
ax.set_title(series_id + " (one-step forecasts)")
ax.set_xlabel('')
plt.tight_layout()
```

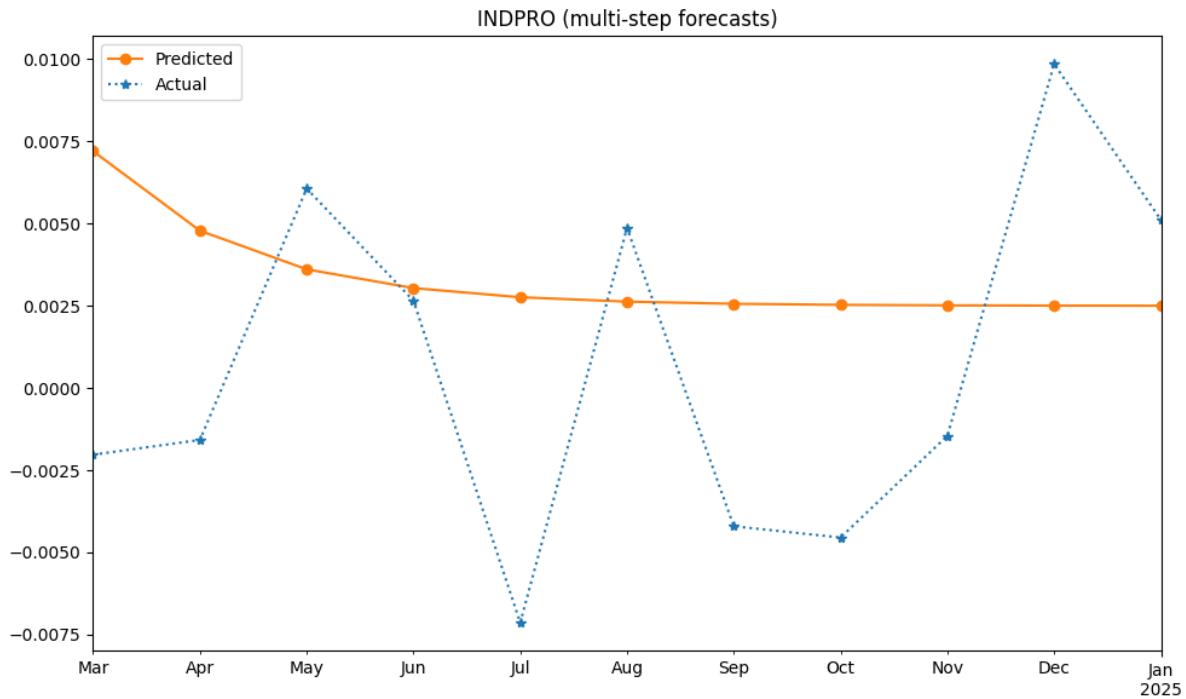


10.5.3 Multi-step forecast

Multi-step forecasts recursively predict values at future horizons, starting with $E_T[Y_{T+1}]$. The forecast horizon h depends on forecasts from earlier steps ($E_T[Y_{T+1}]$, ..., $E_T[Y_{T+h-1}]$). When these quantities appear in the forecast for period $T+h$, they are replaced by the forecasts computed for horizons 1, 2, ..., $h-1$.

```
# set dynamic=True for multi-step ahead predictions
df_pred = test.predict(model.params, dynamic=True,
                       start=df_test.index[0], end=df_test.index[-1])
mse = mean_squared_error(df_test, df_pred)
var = np.mean(np.square(df_test - df_train.mean()))
print(f"Long-term Forecasts: rmse={np.sqrt(mse):.6f}")
fig, ax = plt.subplots(clear=True, num=2, figsize=(10, 6))
df_pred.plot(ax=ax, c='C1', ls='-', marker='o', label='Predicted')
df_test.plot(ax=ax, c='C0', ls=':', marker='*', label='Actual')
ax.legend()
ax.set_title(series_id + " (multi-step forecasts)")
ax.set_xlabel('')
plt.tight_layout()
```

Long-term Forecasts: rmse=0.006085



10.5.4 Granger causality

Granger causality tests whether past values of one time series help predict another.

```
# Granger Causality: INDPRO vs CPI
variables = ['INDPRO', 'CPIAUCSL']
start = 19620101
for series_id, exog_id in zip(variables, list(reversed(variables))):
    df = pd.concat([alf(s, start=start, log=1)
                    for s in [series_id, exog_id]], axis=1)
    df.index = pd.DatetimeIndex(df.index.astype(str))
    df.index.freq = freq
    data = df.diff().dropna()

    print(f"Null Hypothesis: {exog_id} granger-causes {series_id}")
    res = grangercausalitytests(data, maxlag=3)
    print()

    dmf = (f'{series_id} ~ {series_id}.shift(1) '
            f'+ {exog_id}.shift(1) '
            f'+ {exog_id}.shift(2) '
            f'+ {exog_id}.shift(3) ')
    model = smf.ols(formula=dmf, data=data).fit()
    robust = model.get_robustcov_results(cov_type='HAC', use_t=None, maxlags=0)
    print(robust.summary())
```

Null Hypothesis: CPIAUCSL granger-causes INDPRO

Granger Causality
number of lags (no zero) 1

(continues on next page)

(continued from previous page)

```

ssr based F test:      F=0.4754 , p=0.4907 , df_denom=752, df_num=1
ssr based chi2 test:  chi2=0.4773 , p=0.4896 , df=1
likelihood ratio test: chi2=0.4772 , p=0.4897 , df=1
parameter F test:      F=0.4754 , p=0.4907 , df_denom=752, df_num=1

```

```

Granger Causality
number of lags (no zero) 2
ssr based F test:      F=7.1786 , p=0.0008 , df_denom=749, df_num=2
ssr based chi2 test:  chi2=14.4530 , p=0.0007 , df=2
likelihood ratio test: chi2=14.3162 , p=0.0008 , df=2
parameter F test:      F=7.1786 , p=0.0008 , df_denom=749, df_num=2

```

```

Granger Causality
number of lags (no zero) 3
ssr based F test:      F=5.4544 , p=0.0010 , df_denom=746, df_num=3
ssr based chi2 test:  chi2=16.5167 , p=0.0009 , df=3
likelihood ratio test: chi2=16.3381 , p=0.0010 , df=3
parameter F test:      F=5.4544 , p=0.0010 , df_denom=746, df_num=3

```

OLS Regression Results

```

=====
Dep. Variable:          INDPRO    R-squared:           0.090
Model:                 OLS        Adj. R-squared:      0.085
Method:                Least Squares  F-statistic:         2.729
Date:      Mon, 03 Mar 2025  Prob (F-statistic):    0.0283
Time:      10:54:12          Log-Likelihood:      2472.9
No. Observations:      753        AIC:                 -4936.
Df Residuals:          748        BIC:                 -4913.
Df Model:                  4
Covariance Type:        HAC
=====

            coef      std err          t      P>|t|      [ 0.025      0.
975]
-----
Intercept      0.0019      0.001      1.993      0.047      2.94e-05      0.
004
INDPRO.shift (1)  0.2553      0.141      1.817      0.070      -0.020      0.
531
CPIAUCSL.shift (1)  0.4106      0.208      1.973      0.049      0.002      0.
819
CPIAUCSL.shift (2)  -0.4210      0.193     -2.178      0.030      -0.800      -0.
042
CPIAUCSL.shift (3)  -0.1652      0.134     -1.228      0.220      -0.429      0.
099
-----
Omnibus:            721.664    Durbin-Watson:        1.981
Prob(Omnibus):      0.000    Jarque-Bera (JB):    120296.612
Skew:                 -3.763    Prob(JB):            0.00
Kurtosis:                64.462    Cond. No.            565.
=====
```

Notes:

[1] Standard Errors are heteroscedasticity and autocorrelation robust (HAC) using 0 lags and without small sample correction
 Null Hypothesis: INDPRO granger-causes CPIAUCSL

(continues on next page)

(continued from previous page)

```
Granger Causality
number of lags (no zero) 1
ssr based F test:      F=0.0450 , p=0.8321 , df_denom=752, df_num=1
ssr based chi2 test:  chi2=0.0452 , p=0.8317 , df=1
likelihood ratio test: chi2=0.0452 , p=0.8317 , df=1
parameter F test:      F=0.0450 , p=0.8321 , df_denom=752, df_num=1
```

```
Granger Causality
number of lags (no zero) 2
ssr based F test:      F=0.0150 , p=0.9851 , df_denom=749, df_num=2
ssr based chi2 test:  chi2=0.0301 , p=0.9850 , df=2
likelihood ratio test: chi2=0.0301 , p=0.9850 , df=2
parameter F test:      F=0.0150 , p=0.9851 , df_denom=749, df_num=2
```

```
Granger Causality
number of lags (no zero) 3
ssr based F test:      F=0.0926 , p=0.9641 , df_denom=746, df_num=3
ssr based chi2 test:  chi2=0.2804 , p=0.9637 , df=3
likelihood ratio test: chi2=0.2803 , p=0.9637 , df=3
parameter F test:      F=0.0926 , p=0.9641 , df_denom=746, df_num=3
```

OLS Regression Results

```
=====
Dep. Variable:          CPIAUCSL    R-squared:                   0.386
Model:                 OLS         Adj. R-squared:             0.383
Method:                Least Squares F-statistic:                 51.93
Date:      Mon, 03 Mar 2025   Prob (F-statistic):           1.28e-38
Time:          10:54:12        Log-Likelihood:              3452.5
No. Observations:      753        AIC:                      -6895.
Df Residuals:          748        BIC:                      -6872.
Df Model:                   4
Covariance Type:        HAC
=====
            coef    std err          t      P>|t|      [ 0.025      0.
975]
-----
Intercept      0.0012      0.000      6.847      0.000      0.001      0.
002
CPIAUCSL.shift(1)  0.6210      0.046     13.623      0.000      0.532      0.
711
INDPRO.shift(1)   -0.0021      0.014     -0.147      0.883     -0.030      0.
025
INDPRO.shift(2)    0.0005      0.017      0.032      0.975     -0.032      0.
033
INDPRO.shift(3)    0.0061      0.009      0.676      0.499     -0.012      0.
024
=====
Omnibus:            88.983    Durbin-Watson:                 2.119
Prob(Omnibus):      0.000    Jarque-Bera (JB):            750.324
Skew:                  0.030    Prob(JB):                  1.17e-163
Kurtosis:                 7.890    Cond. No.                      319.
=====
```

Notes:

(continues on next page)

(continued from previous page)

```
[1] Standard Errors are heteroscedasticity and autocorrelation robust (HAC) using
    ↵0 lags and without small sample correction
```

10.5.5 Impulse response function

Impulse response functions (IRF) analyze how shocks propagate through a system, often in **Vector Autoregression (VAR)** models.

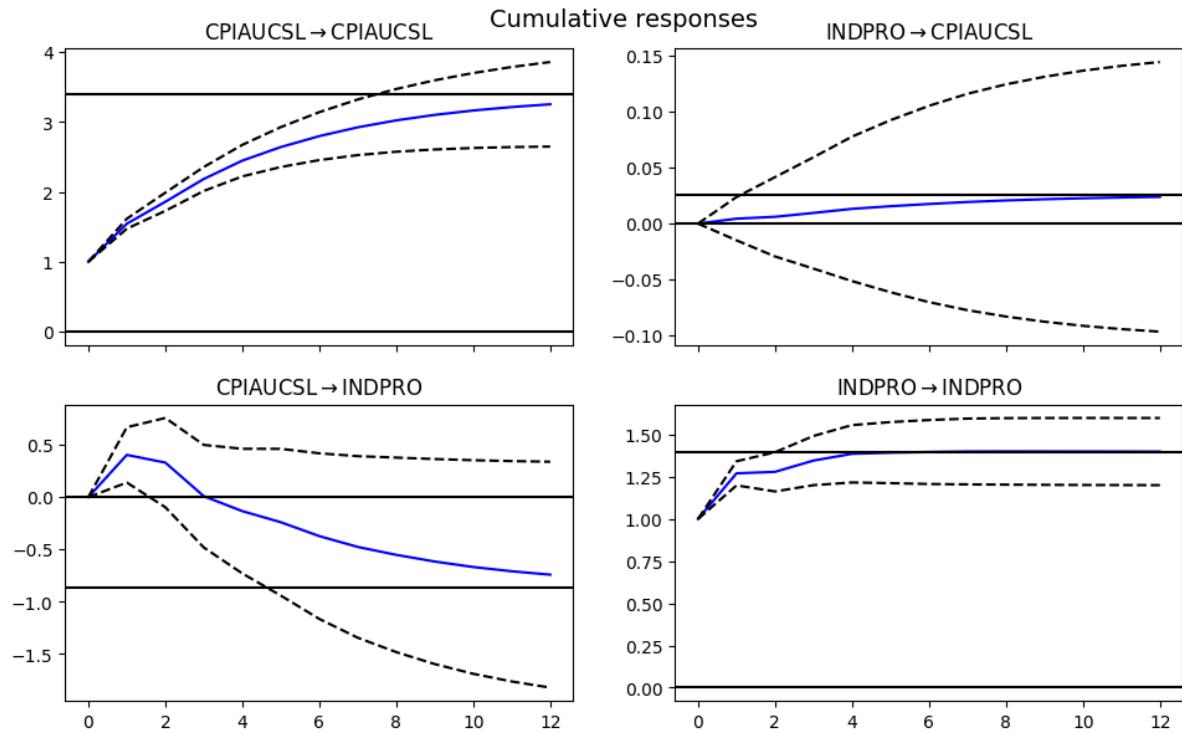
```
# Vector Autoregression: Impulse Response Function
model = VAR(data)
results = model.fit(maxlags=3)
print(results.summary())
irf = results.irf(12)
#irf.plot(orth=False)
irf.plot_cum_effects(orth=False, figsize=(10, 6))
```

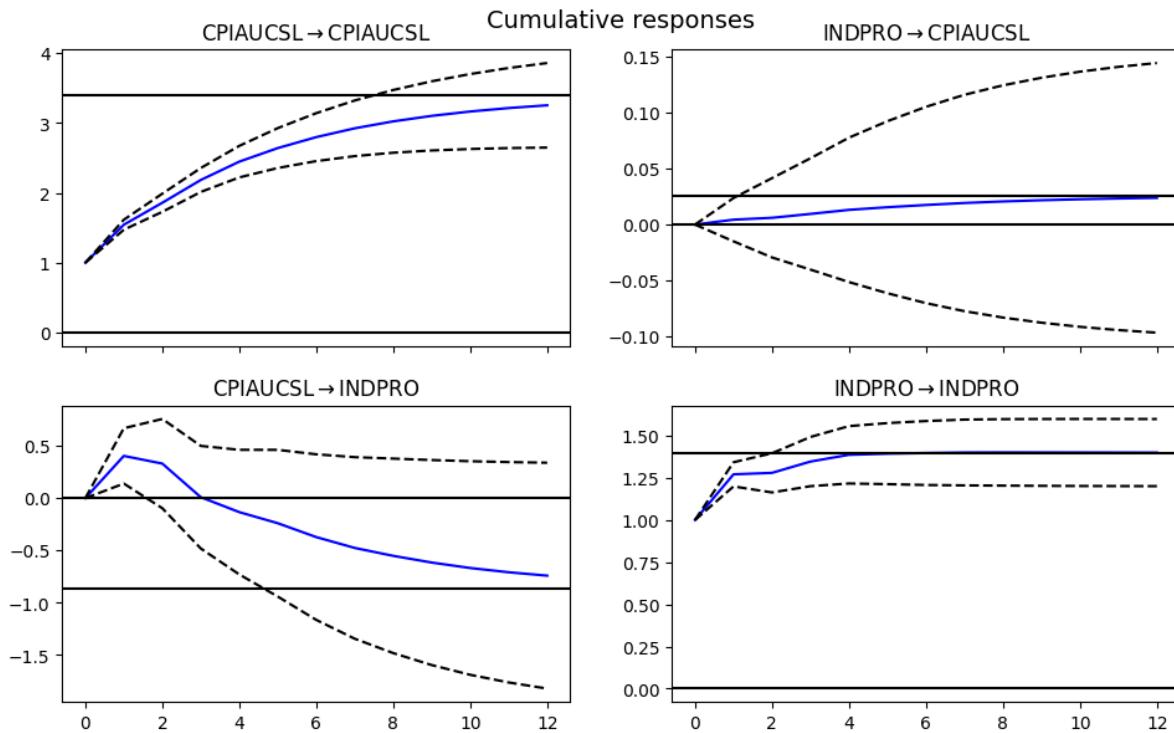
```
Summary of Regression Results
=====
Model:                         VAR
Method:                        OLS
Date:      Mon, 03, Mar, 2025
Time:      10:54:12
-----
No. of Equations:      2.00000   BIC:          -21.3411
Nobs:                  753.000   HQIC:          -21.3939
Log likelihood:        5944.36    FPE:          4.94710e-10
AIC:                  -21.4270   Det(Omega_mle): 4.85639e-10
-----
Results for equation CPIAUCSL
=====
      coefficient      std. error      t-stat      prob
-----
const      0.000909    0.000139      6.528      0.000
L1.CPIAUCSL  0.544275    0.036383     14.960      0.000
L1.INDPRO   0.004382    0.009861      0.444      0.657
L2.CPIAUCSL  0.016181    0.041512      0.390      0.697
L2.INDPRO   -0.001898    0.010116     -0.188      0.851
L3.CPIAUCSL  0.147062    0.036710      4.006      0.000
L3.INDPRO   0.002953    0.009745      0.303      0.762
-----
Results for equation INDPRO
=====
      coefficient      std. error      t-stat      prob
-----
const      0.001899    0.000517      3.673      0.000
L1.CPIAUCSL  0.404130    0.135092      2.992      0.003
L1.INDPRO   0.270682    0.036614      7.393      0.000
L2.CPIAUCSL  -0.402895   0.154135     -2.614      0.009
L2.INDPRO   -0.066262    0.037563     -1.764      0.078
L3.CPIAUCSL  -0.182529   0.136306     -1.339      0.181
L3.INDPRO   0.083606    0.036182      2.311      0.021
```

(continues on next page)

(continued from previous page)

```
Correlation matrix of residuals
      CPIAUCSL    INDPRO
CPIAUCSL  1.000000  0.101927
INDPRO     0.101927  1.000000
```



**References:**

FRM Part I Exam Book Quantitative Analysis Ch 10-11

APPROXIMATE FACTOR MODELS

It is better to be vaguely right than precisely wrong - Carveth Read

Approximate factor models provide a simplified yet effective way to represent time series data by identifying common factors that explain variation across observed variables. A critical step in time series analysis is ensuring stationarity, typically assessed using tests like the Augmented Dickey-Fuller (ADF) test. Transformations such as differencing and logarithmic scaling are often applied to remove non-stationary components. Principal Component Analysis (PCA) is commonly used for factor extraction, and the optimal number of components can be selected using criteria like the Bayesian Information Criterion (BIC). To handle missing data, imputation methods such as the Expectation-Maximization (EM) algorithm are used.

```
# By: Terence Lim, 2020-2025 (terence-lim.github.io)
import numpy as np
import pandas as pd
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from finds.readers import Alfred, fred_md, fred_qd
from finds.recipes import (approximate_factors, mrsq, select_baing, integration_order,
                           remove_outliers, is_outlier)
from secret import credentials
VERBOSE = 0
#%matplotlib qt
```

```
alf = Alfred(api_key=credentials['fred']['api_key'], verbose=VERBOSE)
```

```
## Retrieve recession periods from FRED
vspan = alf.date_spans('USREC')
DataFrame(vspan, columns=['Start', 'End'])
```

	Start	End
0	1854-12-31	1854-12-31
1	1857-06-30	1858-12-31
2	1860-10-31	1861-06-30
3	1865-04-30	1867-12-31
4	1869-06-30	1870-12-31
5	1873-10-31	1879-03-31
6	1882-03-31	1885-05-31
7	1887-03-31	1888-04-30
8	1890-07-31	1891-05-31
9	1893-01-31	1894-06-30

(continues on next page)

(continued from previous page)

```

10 1895-12-31 1897-06-30
11 1899-06-30 1900-12-31
12 1902-09-30 1904-08-31
13 1907-05-31 1908-06-30
14 1910-01-31 1912-01-31
15 1913-01-31 1914-12-31
16 1918-08-31 1919-03-31
17 1920-01-31 1921-07-31
18 1923-05-31 1924-07-31
19 1926-10-31 1927-11-30
20 1929-08-31 1933-03-31
21 1937-05-31 1938-06-30
22 1945-02-28 1945-10-31
23 1948-11-30 1949-10-31
24 1953-07-31 1954-05-31
25 1957-08-31 1958-04-30
26 1960-04-30 1961-02-28
27 1969-12-31 1970-11-30
28 1973-11-30 1975-03-31
29 1980-01-31 1980-07-31
30 1981-07-31 1982-11-30
31 1990-07-31 1991-03-31
32 2001-03-31 2001-11-30
33 2007-12-31 2009-06-30
34 2020-02-29 2020-04-30

```

11.1 Integration order

11.1.1 Augmented Dickey-Fuller test

The **Augmented Dickey-Fuller (ADF) test** is one of the most widely used methods for detecting unit roots in a time series. It operates by performing an ordinary least squares (OLS) regression, where the first difference of the series is regressed on its lagged level, along with deterministic components and lagged differences. The general form of the ADF regression is:

$$\Delta Y_t = \gamma Y_{t-1} + (\delta_0 + \delta_1 t) + \lambda_1 \Delta Y_{t-1} + \dots + \lambda_p \Delta Y_{t-p}$$

If $\gamma = 0$, then Y_t follows a **random walk** and is non-stationary, implying the presence of a unit root. The alternative hypothesis, $\gamma < 0$, suggests that Y_t is **covariance-stationary**. This is a one-sided test, as positive values of γ would indicate an autoregressive coefficient greater than one, leading to instability.

When the test fails to reject the null hypothesis of a unit root, differencing is required to transform the series into a stationary form. Best practice involves repeatedly applying the ADF test to the differenced data until stationarity is achieved.

```

qd_df, qd_codes = fred_qd() # 202004
md_df, md_codes = fred_md() # 201505
qd_date = max(qd_df.index)
md_date = max(md_df.index)

```

```

FRED-QD vintage: quarterly/current.csv
FRED-MD vintage: monthly/current.csv

```

11.1.2 Transformations

The FRED-MD and FRED-QD datasets include transformation codes that indicate appropriate preprocessing steps, such as applying logarithmic transformations or differencing, to ensure stationarity. The number of differences required depends on the presence of unit roots.

```
print(f"Number of series by suggested tcode transformations ({md_date}):")
tcodes = DataFrame.from_dict({i: alf.tcode[i] for i in range(1, 8)},
                             orient='index') \
    .join(qd_codes['transform'].value_counts().rename('fred-qd')) \
    .join(md_codes['transform'].value_counts().rename('fred-md')) \
    .fillna(0) \
    .astype(int) \
    .rename_axis(index='tcode')
tcodes
```

Number of series by suggested tcode transformations (20250131):

tcode	diff	log	pct_change	fred-qd	fred-md
1	0	0	0	22	11
2	1	0	0	32	19
3	2	0	0	0	0
4	0	1	0	0	10
5	1	1	0	140	52
6	2	1	0	49	33
7	1	0	1	1	1

```
# For each series, compare fitted integration order with tcode
out = {}
stationary_out = {}
for label, df, transforms in [[['md', md_df, md_codes['transform']],
                               ['qd', qd_df, qd_codes['transform']]],
                               stationary = dict()
for series_id, tcode in transforms.items():
    # apply transformation if series tcode is valid
    if tcode in range(1, 8):
        # take logs of series if specified by tcode
        s = np.log(df[series_id]) if tcode in [4, 5, 6] else df[series_id]

        # estimate integration order
        order = integration_order(s.dropna(), pvalue=0.05)

        # expected order specified by tcode
        expected_order = 2 if tcode == 7 else ((tcode - 1) % 3)

        # accumulate results for this series
        stationary[series_id] = {'tcode': tcode,
                                'I(p)': order,
                                'different': order - expected_order,
                                'title': alf.header(series_id)}
    #
    # print(series_id, tcode, expected_order, order)

    # collect results for display
    stationary = DataFrame.from_dict(stationary, orient='index')
```

(continues on next page)

(continued from previous page)

```
stationary = stationary.sort_values(stationary.columns.to_list())
c = stationary.groupby(['tcode', 'I(p)'])['title'].count().reset_index()
out[label] = c.pivot(index='tcode', columns='I(p)', values='title')\
    .fillna(0).astype(int)
out[label].columns=[f"I({p})" for p in out[label].columns]
stationary_out[label] = stationary[stationary['different'] > 0]
```

```
print('Series by tcode, transformations and estimated order of integration:')
results = pd.concat([tcodes.drop(columns='fred-md'),
                     out['qd'],
                     tcodes['fred-md'],
                     out['md']], axis=1).fillna(0).astype(int)
print('Integration order by transformation')
results
```

Series by tcode, transformations and estimated order of integration:
Integration order by transformation

tcode	diff	log	pct_change	fred-qd	I(0)	I(1)	I(2)	fred-md	I(0)	I(1)	\
1	0	0	0	22	17	5	0	11	11	11	0
2	1	0	0	32	11	19	2	19	4	4	15
3	2	0	0	0	0	0	0	0	0	0	0
4	0	1	0	0	0	0	0	10	7	7	3
5	1	1	0	140	30	107	3	52	14	38	
6	2	1	0	49	0	29	20	33	0	30	
7	1	0	1	1	0	1	0	1	0	1	

tcode	I(2)
1	0
2	0
3	0
4	0
5	0
6	3
7	0

```
print('FRED-MD Series with unit root after transformation')
stationary_out['md']
```

FRED-MD Series with unit root after transformation

tcode	I(p)	different	\
PERMITMW	4	1	1
HOUSTMW	4	1	1
HOUSTNE	4	1	1

		title
PERMITMW	New Privately-Owned Housing Units Authorized i...	
HOUSTMW	New Privately-Owned Housing Units Started: Tot...	
HOUSTNE	New Privately-Owned Housing Units Started: Tot...	

```
print('FRED-QD Series with unit root after transformation')
stationary_out['qd']
```

FRED-QD Series with unit root after transformation

	tcode	I(p)	different	\	
GS1TB3M	1	1	1		
NWPI	1	1	1		
TLBSNNBBDI	1	1	1		
TLBSNNCBBDI	1	1	1		
HWI	1	1	1		
GFDEBTN	2	2	1		
S&P div yield	2	2	1		
CES2000000008	5	2	1		
TLBSHNO	5	2	1		
OPHMFG	5	2	1		
					title
GS1TB3M					*** GS1TB3M ***
NWPI					*** NWPI ***
TLBSNNBBDI					*** TLBSNNBBDI ***
TLBSNNCBBDI					*** TLBSNNCBBDI ***
HWI					Help Wanted Index for United States
GFDEBTN					Federal Debt: Total Public Debt
S&P div yield					S&P's Composite Common Stock: Dividend Yield
CES2000000008					Average Hourly Earnings of Production and Nons...
TLBSHNO					Households and Nonprofit Organizations; Total ...
OPHMFG					Manufacturing Sector: Labor Productivity (Outp...

11.2 Factor selection

Bai and Ng (2002) provide a systematic approach for extracting and determining the optimal number of factors in an approximate factor model of time series, and imputing missing or outlier observations in the dataset.

11.2.1 Principal component analysis (PCA)

Principal component analysis finds a low-dimensional representation of a data set that contains as much as possible of the variation of its p features. Each principal component is a linear combination of the original features, capturing the most significant patterns in the data.

The first principal component is the normalized linear combination of features:

$$Z_1 = \phi_{11}X_1 + \phi_{21}X_2 + \dots + \phi_{p1}X_p$$

which maximizes variance. Subsequent principal components, such as Z_2 and Z_3 , are determined iteratively, ensuring that each is uncorrelated with the previously computed components while explaining the next highest level of variance.

11.2.2 BIC criterion

Determining the appropriate number of factors is a crucial step in approximate factor modeling. The **Bayesian Information Criterion (BIC)** selects the optimal number of components by balancing model fit and complexity. Lower BIC values indicate a better trade-off between variance explanation and model parsimony.

11.2.3 Data imputation

When dealing with missing data in time series, the **Expectation-Maximization (EM) algorithm** can be employed to estimate missing values from factors iteratively. After each imputation iteration, factors are recovered as projections from PCA, ensuring that the underlying structure of the data is captured.

```
# Verify BaiNg implementation on original FRED-MD and FRED-QD
qd_df, qd_codes = fred_qd('assets/FRED-QD_2020m04.csv', url='')
md_df, md_codes = fred_md('assets/FRED-MD_2015m5.csv', url='')
for freq, df, transforms in [['monthly', md_df, md_codes['transform']],
                             ['quarterly', qd_df, qd_codes['transform']]]:
    # Apply tcode transformations
    transformed = []
    for col in df.columns:
        transformed.append(alf.transform(df[col],
                                         tcode=transforms[col],
                                         freq=freq[0]))
    data = pd.concat(transformed, axis=1).iloc[2:]
    cols = list(data.columns)
    sample = data.index[((np.count_nonzero(np.isnan(data)), axis=1)==0)
                         | (data.index <= 20141231)
                         & (data.index >= 19600301)]

    # set missing and outliers to NaN
    x = data.loc[sample]
    x = remove_outliers(x)      # default fence 'iq10' is 10 times IQ

    # compute factors EM and auto select number of components, r
    Z = approximate_factors(x, p=2, verbose=VERBOSE)
    r = select_baing(Z, p=2)

    # show marginal R2's of series to each component
    mR2 = mrsq(Z, r).to_numpy()
    print(f"FRED-{freq[0].upper()}D {freq} series:")
    print(DataFrame({'selected': r,
                      'variance explained': np.sum(np.mean(mR2[:, :r], axis=0)),
                      'start': min(sample),
                      'end': max(sample),
                      'obs': Z.shape[0],
                      'series': Z.shape[1]},
                    index=[f'factors']))

    for k in range(r):
        args = np.argsort(-mR2[:, k])
        print(f"Factor:{1+k} Variance Explained={np.mean(mR2[:, k]):.4f}")
        print(DataFrame.from_dict({mR2[arg, k].round(4):
                                    {'series': cols[arg],
                                     'description': alf.header(cols[arg])}
                                    for arg in args[:10]}),
              orient='index'))
```

```

FRED-QD vintage: assets/FRED-QD_2020m04.csv
FRED-MD vintage: assets/FRED-MD_2015m5.csv
FRED-MD monthly series:
    selected  variance explained      start      end  obs  series
factors          8          0.485832  19600331  20141231  658     134
Factor:1 Variance Explained=0.1613
    series                      description
0.7424  USGOOD          All Employees, Goods-Producing
0.7235  PAYEMS          All Employees, Total Nonfarm
0.7002  MANEMP          All Employees, Manufacturing
0.6565  NAPM             *** NAPM ***
0.6540  IPMANSICS       Industrial Production: Manufacturing (SIC)
0.6513  DMANEMP         All Employees, Durable Goods
0.6314  INDPERO         Industrial Production: Total Index
0.6037  NAPMNOI         *** NAPMNOI ***
0.6026  NAPMPI          *** NAPMPI ***
0.5601  CUMFNS          Capacity Utilization: Manufacturing (SIC)
Factor:2 Variance Explained=0.0703
    series                      description
0.6259  T10YFFM         10-Year Treasury Constant Maturity Minus Feder...
0.6164  AAAFFM          Moody's Seasoned Aaa Corporate Bond Minus Fede...
0.5856  BAAFFM          Moody's Seasoned Baa Corporate Bond Minus Fede...
0.5832  T5YFFM          5-Year Treasury Constant Maturity Minus Feder...
0.4785  TB6SMFFM        6-Month Treasury Bill Minus Federal Funds Rate
0.4779  TB3SMFFM        3-Month Treasury Bill Minus Federal Funds Rate
0.4322  T1YFFM          1-Year Treasury Constant Maturity Minus Feder...
0.2367  COMPAPFF        3-Month Commercial Paper Minus FEDFUNDS
0.2258  BUSINV          Total Business Inventories
0.1896  NAPMPRI         *** NAPMPRI ***
Factor:3 Variance Explained=0.0652
    series                      description
0.7192  CUSR0000SAC     Consumer Price Index for All Urban Consumers: ...
0.7066  DNDGRG3M086SBEA Personal consumption expenditures: Nondurable ...
0.6700  CPIAUCSL        Consumer Price Index for All Urban Consumers: ...
0.6392  CUSR0000SA0L5   Consumer Price Index for All Urban Consumers: ...
0.6115  CUUR0000SA0L2   Consumer Price Index for All Urban Consumers: ...
0.5923  PCEPI            Personal Consumption Expenditures: Chain-type ...
0.5907  CPITRNSL        Consumer Price Index for All Urban Consumers: ...
0.5482  CPIULFSL        Consumer Price Index for All Urban Consumers: ...
0.4863  PPIFCG           Producer Price Index by Commodity for Finished...
0.4779  PPIITM           Producer Price Index by Commodity Intermediate...
Factor:4 Variance Explained=0.0547
    series                      description
0.4294  GS1               Market Yield on U.S. Treasury Securities at 1-...
0.4228  GS5               Market Yield on U.S. Treasury Securities at 5-...
0.4123  AAA               Moody's Seasoned Aaa Corporate Bond Yield
0.3995  TB6MS             6-Month Treasury Bill Secondary Market Rate, D...
0.3893  GS10              Market Yield on U.S. Treasury Securities at 10...
0.3621  BAA               Moody's Seasoned Baa Corporate Bond Yield
0.3179  TB3MS             3-Month Treasury Bill Secondary Market Rate, D...
0.3139  CP3M              3-Month AA Financial Commercial Paper Rates
0.2681  HOUST              New Privately-Owned Housing Units Started: Tot...
0.2562  HOUSTW             New Privately-Owned Housing Units Started: Tot...
Factor:5 Variance Explained=0.0425
    series                      description
0.2481  PERMITW           New Privately-Owned Housing Units Authorized i...
0.2374  PERMIT              New Privately-Owned Housing Units Authorized i...

```

(continues on next page)

(continued from previous page)

```

0.2316 HOUSTW New Privately-Owned Housing Units Started: Tot...
0.2225 GS5 Market Yield on U.S. Treasury Securities at 5-...
0.2221 GS1 Market Yield on U.S. Treasury Securities at 1-...
0.2146 HOUST New Privately-Owned Housing Units Started: Tot...
0.2028 GS10 Market Yield on U.S. Treasury Securities at 10...
0.1951 PERMITMW New Privately-Owned Housing Units Authorized i...
0.1949 T1YFFM 1-Year Treasury Constant Maturity Minus Federa...
0.1932 TB6MS 6-Month Treasury Bill Secondary Market Rate, D...
Factor:6 Variance Explained=0.0365
    series                                     description
0.2186 IPCONGD          Industrial Production: Consumer Goods
0.1793 ISRATIO          Total Business: Inventories to Sales Ratio
0.1736 NAPMEI           *** NAPMEI ***
0.1628 IPDCONGD         Industrial Production: Durable Consumer Goods:...
0.1577 IPFINAL          Industrial Production: Final Products
0.1545 TB6SMFFM         6-Month Treasury Bill Minus Federal Funds Rate
0.1425 NAPM              *** NAPM ***
0.1416 NAPMII            *** NAPMII ***
0.1414 ACOGNO           Manufacturers' New Orders: Consumer Goods
0.1373 IPFPNSS          Industrial Production: Final Products and Noni...
Factor:7 Variance Explained=0.0292
    series                                     description
0.5165 S&P 500           S&P's Common Stock Price Index: Composite
0.5159 S&P: indust       S&P's Common Stock Price Index: Industrials
0.4002 S&P div yield    S&P's Composite Common Stock: Dividend Yield
0.2764 S&P PE ratio     S&P's Composite Common Stock: Price-Earnings R...
0.2564 UMCSENT          University of Michigan: Consumer Sentiment
0.1030 IPCONGD          Industrial Production: Consumer Goods
0.1019 EXCAUS            Canadian Dollars to U.S. Dollar Spot Exchange ...
0.0728 IPFINAL          Industrial Production: Final Products
0.0644 IPDCONGD         Industrial Production: Durable Consumer Goods:...
0.0565 IPFPNSS          Industrial Production: Final Products and Noni...
Factor:8 Variance Explained=0.0262
    series                                     description
0.5375 TWEXMMTH         Nominal Major Currencies U.S. Dollar Index (Go...
0.2309 EXSZUS            Swiss Francs to U.S. Dollar Spot Exchange Rate
0.2111 EXUSUK            U.S. Dollars to U.K. Pound Sterling Spot Excha...
0.1497 EXJPUS             Japanese Yen to U.S. Dollar Spot Exchange Rate
0.1332 SRVPRD            All Employees, Service-Providing
0.1199 CES0600000008    Average Hourly Earnings of Production and Nons...
0.1156 ACOGNO            Manufacturers' New Orders: Consumer Goods
0.1128 CES3000000008    Average Hourly Earnings of Production and Nons...
0.1104 USGOVT            All Employees, Government
0.1010 USTRADE           All Employees, Retail Trade
FRED-QD quarterly series:
    selected variance explained      start      end  obs  series
factors        7        0.4981  19600331  20191231  240      248
Factor:1 Variance Explained=0.2014
    series                                     description
0.8382 USPRIV            All Employees, Total Private
0.8184 USGOOD             All Employees, Goods-Producing
0.8165 OUTMS              Manufacturing Sector: Real Sectoral Output for...
0.8124 IPMANSICS         Industrial Production: Manufacturing (SIC)
0.8120 PAYEMS             All Employees, Total Nonfarm
0.8057 INDPRO             Industrial Production: Total Index
0.7758 MANEMP             All Employees, Manufacturing

```

(continues on next page)

(continued from previous page)

0.7717 HOANBS Nonfarm Business Sector: Hours Worked for All ...
 0.7667 DMANEMP All Employees, Durable Goods
 0.7651 UNRATE Unemployment Rate

Factor:2 Variance Explained=0.0824

	series	description
0.4995	AAAFFM	Moody's Seasoned Aaa Corporate Bond Minus Fede...
0.4761	T5YFFM	5-Year Treasury Constant Maturity Minus Federa...
0.4622	PERMIT	New Privately-Owned Housing Units Authorized i...
0.4404	BUSINV	Total Business Inventories
0.4231	HOUST	New Privately-Owned Housing Units Started: Tot...
0.4055	PERMITS	New Privately-Owned Housing Units Authorized i...
0.3925	S&P div yield	S&P's Composite Common Stock: Dividend Yield
0.3849	TCU	Capacity Utilization: Total Index
0.3674	CPF3MTB3M	3-Month Commercial Paper Minus 3-Month Treasur...
0.3633	GS10TB3M	*** GS10TB3M ***

Factor:3 Variance Explained=0.0727

	series	description
0.7569	CUSR0000SA0L2	Consumer Price Index for All Urban Consumers: ...
0.7405	CUSR0000SAC	Consumer Price Index for All Urban Consumers: ...
0.7368	DGDSRG3Q086SBEA	Personal consumption expenditures: Goods (chai...
0.7212	PCECTPI	Personal Consumption Expenditures: Chain-type ...
0.7065	CPITRNSL	Consumer Price Index for All Urban Consumers: ...
0.6963	DNDGRG3Q086SBEA	Personal consumption expenditures: Nondurable ...
0.6798	CUSR0000SA0L5	Consumer Price Index for All Urban Consumers: ...
0.6712	CPIAUCSL	Consumer Price Index for All Urban Consumers: ...
0.6472	WPSID61	Producer Price Index by Commodity: Intermediat...
0.6352	CPIULFSL	Consumer Price Index for All Urban Consumers: ...

Factor:4 Variance Explained=0.0467

	series	description
0.4044	IMFSL	Institutional Money Market Funds (DISCONTINUED)
0.3457	CES9093000001	All Employees, Local Government
0.3259	CES9092000001	All Employees, State Government
0.2484	EXUSEU	U.S. Dollars to Euro Spot Exchange Rate
0.2482	USGOVT	All Employees, Government
0.2364	GFDEBTN	Federal Debt: Total Public Debt
0.2251	REVOLSL	Revolving Consumer Credit Owned and Securitized
0.2202	USSERV	All Employees, Other Services
0.2128	COMPRMS	Manufacturing Sector: Real Hourly Compensation...
0.2126	NWPI	*** NWPI ***

Factor:5 Variance Explained=0.0375

	series	description
0.3664	OPHMFG	Manufacturing Sector: Labor Productivity (Outp...
0.3093	HWI	Help Wanted Index for United States
0.2989	NWPI	*** NWPI ***
0.2961	AWHMAN	Average Weekly Hours of Production and Nonsupe...
0.2703	OPHPBS	Business Sector: Labor Productivity (Output pe...
0.2358	OPHNFB	Nonfarm Business Sector: Labor Productivity (O...
0.2307	UNRATELT	*** UNRATELT ***
0.2189	UNLPNBS	Nonfarm Business Sector: Unit Nonlabor Payment...
0.2131	ULCMFG	Manufacturing Sector: Unit Labor Costs for All...
0.1989	TLBSNNCBBDI	*** TLBSNNCBBDI ***

Factor:6 Variance Explained=0.0303

	series	description
0.2669	CONSPI	Nonrevolving consumer credit to Personal Income
0.2388	ULCBS	Business Sector: Unit Labor Costs for All Workers
0.2379	ULCNFB	Nonfarm Business Sector: Unit Labor Costs for ...

(continues on next page)

(continued from previous page)

0.2173	CONSUMER	Consumer Loans, All Commercial Banks
0.2016	EXUSEU	U.S. Dollars to Euro Spot Exchange Rate
0.1979	AHETPI	Average Hourly Earnings of Production and Nons...
0.1865	NONREVSL	Nonrevolving Consumer Credit Owned and Securit...
0.1613	TOTALSSL	Total Consumer Credit Owned and Securitized
0.1453	B020RE1Q156NBEA	Shares of gross domestic product: Exports of g...
0.1361	UMCSENT	University of Michigan: Consumer Sentiment
Factor:7 Variance Explained=0.0270		
	series	description
0.2663	USEPUINDXM	Economic Policy Uncertainty Index for United S...
0.1954	TNWBSHNO	Households and Nonprofit Organizations; Net Wo...
0.1889	TABSHNO	Households and Nonprofit Organizations; Total ...
0.1872	S&P 500	S&P's Common Stock Price Index: Composite
0.1856	S&P: indust	S&P's Common Stock Price Index: Industrials
0.1808	TFAABSHNO	Households and Nonprofit Organizations; Total ...
0.1756	NASDAQCOM	NASDAQ Composite Index
0.1623	GS10TB3M	*** GS10TB3M ***
0.1394	OPHPBS	Business Sector: Labor Productivity (Output pe...

```
# Compute approximate factors from current FRED-MD
data, t = fred_md()      # fetch dataframe of current fred-md and transform codes
data.index = pd.to_datetime(data.index, format='%Y%m%d')
transforms = t['transform']
data = pd.concat([alf.transform(data[col], tcode=transforms[col], freq='m')
                  for col in data.columns], axis=1)

# remove outliers and impute using Bai-Ng approach
r = 8      # fix number of factors = 8
X = remove_outliers(data, method='farout')
X = approximate_factors(X, kmax=r, p=0, verbose=VERBOSE)
```

FRED-MD vintage: monthly/current.csv

Recover factors as the projections from PCA

```
# Extract factors from PCA projections on imputed data
y = StandardScaler().fit_transform(X)
pca = PCA(n_components=r).fit(y)
factors = DataFrame(pca.transform(y), index=data.index, columns=range(1, 1+r))
```

```
# Plot extracted factors
fig, axes = plt.subplots(nrows=4, ncols=2, figsize=(10, 12), sharex=True, sharey=True)
axes = axes.flatten()
for col, ax in zip(factors.columns, axes):
    flip = -np.sign(max(factors[col]) + min(factors[col])) # try match sign
    (flip*factors[col]).plot(ax=ax, color=f"C{col}")
    for a,b in vspans:
        if b >= min(factors.index):
            ax.axvspan(max(a, min(factors.index)),
                        min(b, max(factors.index)),
                        alpha=0.4,
                        color='grey')
    ax.spines["top"].set_visible(False)
    ax.spines["right"].set_visible(False)
```

(continues on next page)

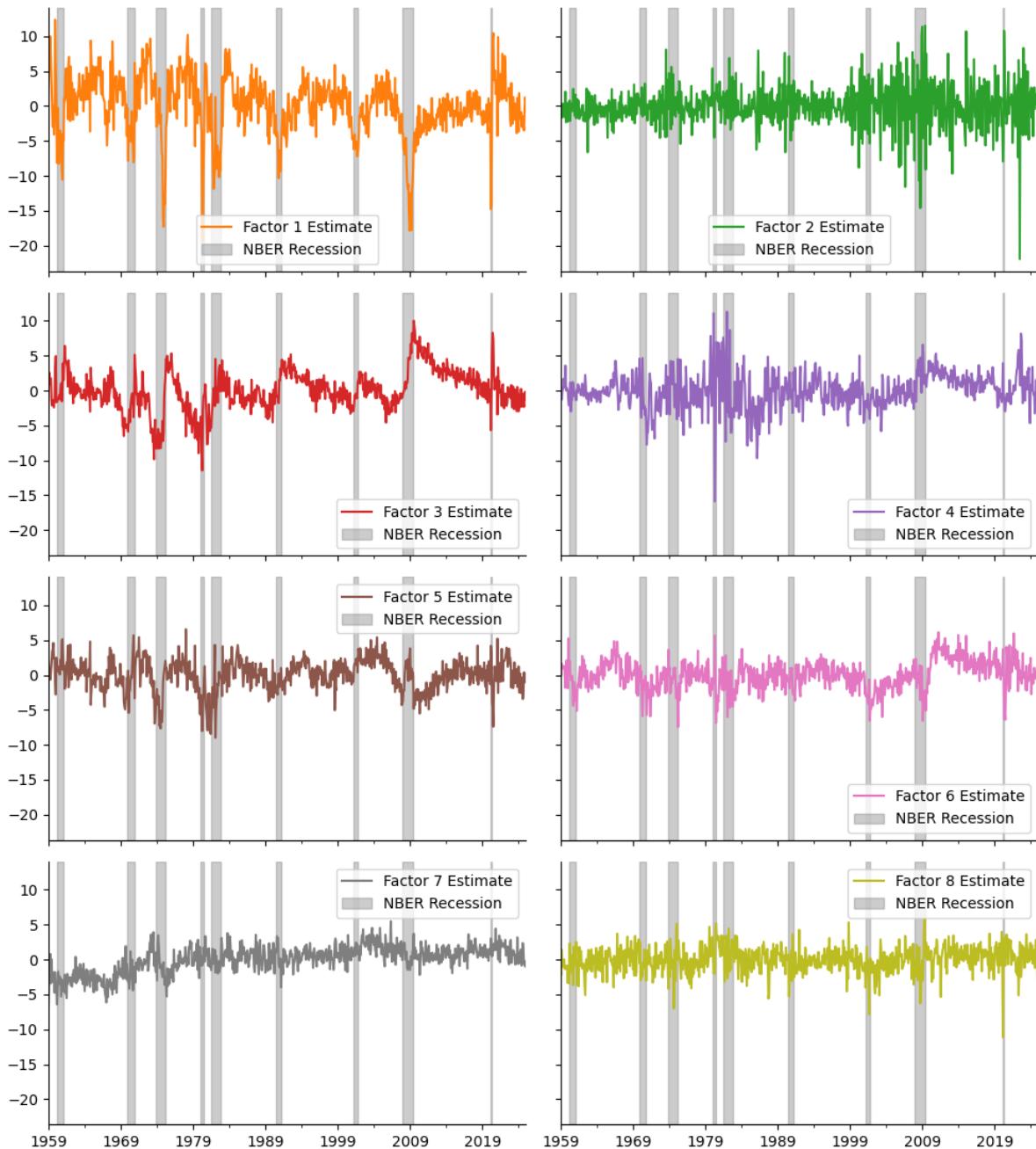
(continued from previous page)

```

    ax.legend([f"Factor {col} Estimate", 'NBER Recession'])
    plt.tight_layout(rect=[0, 0.03, 1, 0.95])
    plt.suptitle(f"Factor Estimates {factors.index[0]}:{factors.index[-1]}")
    plt.show()

```

Factor Estimates Jan-1959:Jan-2025

**References:**

Bai, Jushan and Ng, Serena, 2002, Determining the Number of Factors in Approximate Factor Models, *Econometrica*

70:1, 191-2211

McCracken, Michael W., and Serena Ng, 2016. FRED-MD: A monthly database for macroeconomic research, *Journal of Business & Economic Statistics*, 34(4), 574-589.

Michael W. McCracken, Serena Ng, 2020, FRED-QD: A Quarterly Database for Macroeconomic Research.

STATE SPACE MODELS

If you want to understand today, you have to search yesterday - Pearl Buck

State space models provide a powerful framework for modeling time series data, particularly when the observed data are generated by an underlying system of **hidden states**.

A fundamental example of a state space model is the **Hidden Markov Model (HMM)**, which represents a system where an unobserved sequence of states follows a **Markov process**, and each state emits observations according to some probability distribution.

The **Gaussian Mixture Model (GMM)** is a probabilistic method that represents data as a combination of multiple Gaussian distributions, allowing for the estimation of the likelihood of an unknown state.

```
# By: Terence Lim, 2020-2025 (terence-lim.github.io)
import numpy as np
import pandas as pd
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
import seaborn as sns
from hmmlearn import hmm
from sklearn.preprocessing import StandardScaler
from sklearn.mixture import GaussianMixture
from finds.readers import fred_qd, fred_md, Alfred
from finds.recipes import approximate_factors, remove_outliers
from secret import paths, credentials
VERBOSE = 0
# %matplotlib qt
```

```
# Load and pre-process time series from FRED
alf = Alfred(api_key=credentials['fred']['api_key'], verbose=VERBOSE)
vspan = alf.date_spans('USREC') # to indicate recession periods in the plots
```

```
# FRED-MD
freq = 'M'
df, t = fred_md() if freq == 'M' else fred_qd()
transforms = t['transform']
data = pd.concat([alf.transform(df[col], tcode=transforms[col], freq=freq),
                  for col in df.columns],
                  axis=1)\n    .iloc[1:] # apply transforms and drop first row
cols = list(data.columns)

# remove outliers and impute using Bai-Ng approach
r = 8 # fix number of factors = 8
```

(continues on next page)

(continued from previous page)

```
data = remove_outliers(data)
data = approximate_factors(data, kmax=r, p=0, verbose=VERBOSE)
```

```
FRED-MD vintage: monthly/current.csv
```

```
# standardize inputs
X = StandardScaler().fit_transform(data.values)
```

12.1 Hidden Markov Model

A **Hidden Markov Model (HMM)** is a type of multivariate time series model where the system is governed by a set of **unobserved (hidden) states** that follow a Markov process. The model is characterized by three key components:

1. **Transition probabilities:** The likelihood of moving from one state to another (or remaining in the same state), represented by a **transition matrix A** .
2. **Emission probabilities:** The probability of observing a certain value given a hidden state, denoted as θ .
3. **Initial state distribution:** A probability vector π that defines the starting state of the system.

Thus, an HMM is fully specified by the parameter set $\lambda = (A, \theta, \pi)$. The observed sequence X is generated by an unobserved state sequence Z , making inference about Z a central problem in HMM applications.

To work with HMMs in Python, we use the `hmmlearn` package, which provides efficient algorithms for solving the three fundamental HMM tasks:

- **Evaluation:** Computing the likelihood of an observation sequence given the model, $P(X|\lambda)$. This is solved using the **Forward-Backward algorithm**, implemented via the `.score()` method.
- **Decoding:** Determining the most probable sequence of hidden states Z given the observations X . This is accomplished using the **Viterbi algorithm**, accessed through the `.predict()` method.
- **Training:** Estimating the model parameters $\lambda = (A, \theta, \pi)$ by maximizing the likelihood of the observed data. This is done using the **Baum-Welch algorithm**, a specific case of the **Expectation-Maximization (EM) algorithm**, via the `.fit()` method.

The `GaussianHMM` model in `hmmlearn` assumes a **Gaussian distribution** for emissions. The `covariance_type` parameter controls the structure of the covariance matrix, with the following options (ordered by increasing complexity):

- "spherical": A single variance value is used for all features within each state.
- "diag": Each state has a diagonal covariance matrix, allowing feature-specific variances.
- "tied": A single full covariance matrix is shared across all states.
- "full": Each state has its own full covariance matrix, providing the most flexibility.

To select the optimal model, an **information criterion** such as the **Bayesian Information Criterion (BIC)** can be used to balance model complexity and goodness of fit.

```
def hmm_summary(markov, X, lengths, matrix=False):
    """Helper to return summary statistics from fitting Hidden Markov Model

    Args:
        markov: Fitted GaussianHMM
        X: Input data of shape (nsamples, nfeatures)
```

(continues on next page)

(continued from previous page)

```

lengths: Lengths of the individual sequences in X, sum is nsamples
matrix: Whether to return the transition and stationary matrices

Returns:
    Dictionary of results in {'aic', 'bic', 'parameters', 'NLL'}
"""
logL = markov.score(X, lengths)
T = np.sum(lengths)      # n_samples
n = markov.n_features    # number of features ~ dim of covariance matrix
m = markov.n_components  # number of states
k = dict(diag=m*n,      # parms in mean and cov matrix
         full=m*n*(n-1)/2,
         tied=n*(n-1)/2,
         spherical=m)[markov.covariance_type] + markov.n_features
p = m**2 + (k * m) - 1  # number of independent parameters of the model

results = {'aic': -2 * logL + (2 * p),
           'bic': -2 * logL + (p * np.log(T)),
           'parameters': p,
           'NLL' : -logL}
if matrix:    # whether to return the transition and stationary matrix
    matrix = DataFrame(markov.transmat_) \
        .rename_axis(columns='Transition Matrix:')
    matrix['Stationary'] = markov.get_stationary_distribution()
    results.update({'matrix': matrix})  # return matrix as DataFrame
return results

```

```

# Compare covariance types in Gaussian HMM models
out = []
for covariance_type in ["full", "diag", "tied", "spherical"]:
    for n_components in range(1, 8):
        if VERBOSE:
            print('=====', covariance_type, n_components, '=====')
        markov = hmm.GaussianHMM(n_components=n_components,
                                  covariance_type=covariance_type,
                                  verbose=VERBOSE,
                                  tol=1e-6,
                                  random_state=0,
                                  n_iter=500) \
            .fit(X, [X.shape[0]])
        result = hmm_summary(markov, data, [X.shape[0]])
        #print(n_components, Series(results, name=covariance_type).to_frame().T)
        result.update({'cov_type': covariance_type,
                       'n_components': n_components})
        out.append(Series(result))
results = pd.concat(out, axis=1).T.convert_dtypes()

```

```

Model is not converging. Current: -48374.028067592284 is not greater than -48374.
-02806594439. Delta is -1.6478952602483332e-06
Model is not converging. Current: -125173.83118863673 is not greater than -125173.
-83118797389. Delta is -6.628397386521101e-07
Model is not converging. Current: -123662.93419332648 is not greater than -123662.
-93417683932. Delta is -1.648715988267213e-05

```

```
# Show best bic's
best_bic = []
for covariance_type in ["full", "diag", "tied", "spherical"]:
    result = results[results['cov_type'] == covariance_type]
    argmin = np.argmin(result['bic'])
    best_bic.append(result.iloc[argmin])
best_bic = pd.concat(best_bic, axis=0)
best_type = best_bic.iloc[np.argmin(best_bic['bic'])]['cov_type']
print(f"HMM best bic type: {best_type}")
best_bic.round(0)
```

HMM best bic type: spherical

	aic	bic	parameters	NLL	cov_type	n_components
0	212991056.0	213028448.0	8001	106487527.0	full	1
12	16508012.0	16532906.0	5327	8248679.0	diag	6
14	212991056.0	213028448.0	8001	106487527.0	tied	1
23	14544672.0	14546518.0	395	7271941.0	spherical	3

```
# fit model with best_bic
n_components = best_bic[best_bic['cov_type'] == best_type]['n_components']
n_components = int(n_components.iloc[0])
markov = hmm.GaussianHMM(n_components=n_components, covariance_type=best_type,
                         verbose=False, tol=1e-6, random_state=0, n_iter=100) \
    .fit(X, [X.shape[0]])
pred_markov = Series(markov.predict(X),
                      name='state',
                      index=pd.to_datetime(data.index, format="%Y%m%d"))
matrix = hmm_summary(markov, X, [X.shape[0]], matrix=True)['matrix']
```

```
# Compute average change in INDPRO by state
df = alf('INDPRO', freq=freq, log=1, diff=1)
df.index = pd.DatetimeIndex(df.index.astype(str), freq='infer')
indpro = pd.concat([df, pred_markov], join='inner', axis=1).groupby('state').mean(
    ↴ 'INDPRO')

# and HMM transition and stationary probabilities
print("HMM transition and stationary probabilities, and average INDPRO value by state
    ↴")
matrix.join(indpro).round(4)
```

HMM transition and stationary probabilities, and average INDPRO value by state

	0	1	2	Stationary	INDPRO
0	0.6485	0.1775	0.1740	0.1720	-0.0020
1	0.0677	0.8595	0.0728	0.4482	0.0053
2	0.0793	0.0854	0.8352	0.3798	-0.0002

Plot predicted states

```
def plot_states(modelname, labels, num=1, series_id='INDPRO', freq='M'):
    """helper to plot predicted states 'IPMANSICS'"""
    (continues on next page)
```

(continued from previous page)

```

# n_components markers
n_components = len(np.unique(labels))
markers = ["o", "s", "d", "X", "P", "8", "H", "*", "x", "+"][:n_components]

fig, (bx, ax) = plt.subplots(nrows=2, ncols=1, figsize=(10, 6))

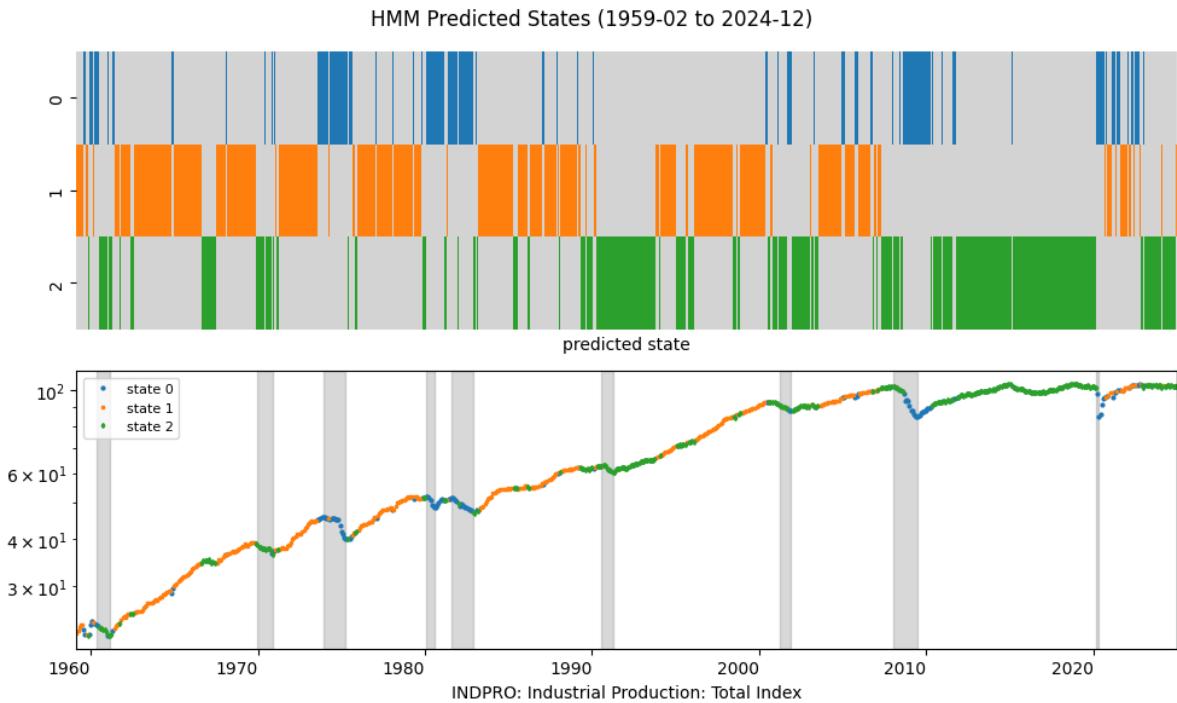
# plot series, with states colored
df = alf(series_id, freq=freq)
df.index = pd.DatetimeIndex(df.index.astype(str), freq='infer')
df = df[(df.index >= min(labels.index)) & (df.index <= max(labels.index))]
for i, marker in zip(range(n_components), markers):
    df.loc[labels == i].plot(ax=ax, style=marker, markersize=2, color=f"C{i}", ↴
    rot=0)
    ax.set_xlabel(f"{series_id}: {alf.header(series_id)}")
    ax.set_xlim(left=min(df.index), right=max(df.index))
    for a,b in vspans: # shade economic recession periods
        if (b > min(df.index)) & (a < max(df.index)):
            ax.axvspan(max(a, min(df.index)), min(b, max(df.index)),
                        alpha=0.3, color='grey')
    ax.legend([f"state {i}" for i in range(n_components)], fontsize=8)
    ax.set_yscale('log')

s = np.zeros((n_components, len(labels)))
for i, j in enumerate(labels.values.flatten()):
    s[j][i] = j + 1
sns.heatmap(s, vmin=0, vmax=n_components, ax=bx, cbar=False, xticklabels=False,
            cmap=["lightgrey"] + [f"C{i}" for i in range(n_components)])
bx.set_xlabel('predicted state')
date_str = f"({str(df.index[0])[:7]} to {str(df.index[-1])[:7]})"
fig.suptitle(f"{modelname.upper()} Predicted States" + date_str)

plt.tight_layout()

```

```
plot_states('HMM', pred_markov, num=1, freq=freq)
```



12.1.1 Markov Chains

Markov Chains are a fundamental class of state space models where the future state of a system depends only on the current state and not on past states. Key concepts in Markov Chains include:

- Geometric Runs: The probability distribution of the time until a particular state recurs, which often follows a geometric distribution.
- Irreducibility: A property of a Markov Chain where it is possible to reach any state from any other state, ensuring long-run stability in the system.

The `Graphviz` python package can be used to visualize the states and transition probabilities of a Hidden Markov Model (HMM), providing a representation of how states evolve over time.

```
# Visualize HMM transitions and states
from sklearn.preprocessing import minmax_scale
colors = minmax_scale(indpro).flatten()

import graphviz
def fillcolor(r, g, b):
    def scale(x):
        return int((1 - x) * 256 * .5 + 64)
    return f"#{scale(r):02x}{scale(g):02x}{scale(b):02x}"
```

```
dot = graphviz.Digraph(engine='circo', graph_attr={'splines': 'true'})
dot.graph_attr['size'] = '10,10'
dot.graph_attr['ratio'] = 'expand'
for i in range(len(matrix)): # Add nodes
    dot.node(name=str(i), label=f"{matrix.iloc[i, i]:.03f}",
              style='filled', fillcolor=fillcolor(0, colors[i], colors[i]))
for i in range(len(matrix)): # Add edges
```

(continues on next page)

(continued from previous page)

```

for j in range(len(matrix)):
    if i != j:
        dot.edge(tail_name=str(i), head_name=str(j),
                 label=f" {matrix.iloc[i, j]:.03f} ", color='red')

```

matrix

Transition Matrix:		0	1	2	Stationary
0		0.648460	0.177506	0.174035	0.171962
1		0.067656	0.859514	0.072830	0.448240
2		0.079320	0.085432	0.835248	0.379799

```

dot
# dot.format = 'png'
# dot.view(filename='digraph')  # Visualize the graph

```

```
<graphviz.graphs.Digraph at 0x7f3131652c10>
```

12.2 Gaussian Mixture Model

A **Gaussian Mixture Model (GMM)** is a probabilistic model that assumes that the data is generated from a mixture of multiple Gaussian distributions. Each component in the mixture represents a different subpopulation within the data. The following are the parameters that need to be estimated:

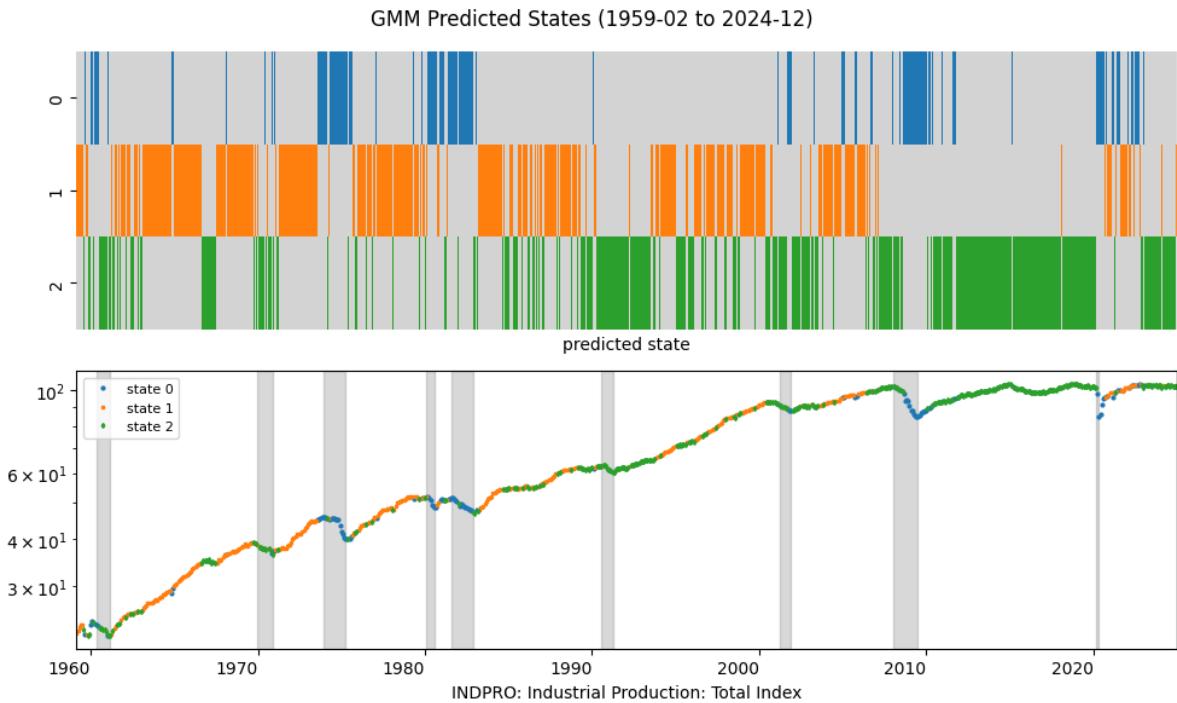
1. Mixing Coefficients (Weights) π_k – These represent the proportion of each Gaussian component in the overall mixture and must sum to 1.
2. Mean Vectors μ_k – The center of each Gaussian component in the feature space, indicating where each cluster is located.
3. Covariance Matrices Σ_k – These define the shape and spread of each Gaussian distribution, determining how data points are dispersed around the mean.

These parameters are typically estimated using the **Expectation-Maximization (EM) algorithm**, by iteratively optimizing the likelihood of the observed data.

```

gmm = GaussianMixture(n_components=n_components,
                      random_state=0,
                      covariance_type='best_type').fit(X)
pred_gmm = Series(gmm.predict(X),
                   name='state',
                   index=pd.to_datetime(data.index, format="%Y%m%d"))
plot_states('GMM', pred_gmm, num=2, freq=freq)

```



12.2.1 Persistance

Persistence in an HMM refers to the likelihood that the system remains in the same hidden state over time rather than transitioning to a different state. It is determined by the self-transition probabilities on the diagonal of the transition matrix.

In GMMs, persistence refers to the likelihood of a data point belonging to the same Gaussian component across different observations. Since GMMs do not model temporal dependencies like HMMs, persistence is inferred from the posterior probability of a point belonging a particular cluster.

```
# Compare persistance of HMM and GMM
dist = DataFrame({
    'Hidden Markov': ([np.mean(pred_markov[:-1].values == pred_markov[1:].values)]
                    + matrix.iloc[:, -1].tolist()),
    'Gaussian Mixture': ([np.mean(pred_gmm[:-1].values == pred_gmm[1:].values)]
                        + (pred_gmm.value_counts().sort_index() / len(pred_gmm)).tolist()),
    index=[['Average persistance of states']
          + [f'Stationary prob of state {s}' for s in range(n_components)]])
print("Compare HMM with GMM:")
dist
```

Compare HMM with GMM:

	Hidden Markov	Gaussian Mixture
Average persistance of states	0.820253	0.732911
Stationary prob of state 0	0.171962	0.152971
Stationary prob of state 1	0.448240	0.424779
Stationary prob of state 2	0.379799	0.422250

TERM STRUCTURE OF INTEREST RATES

The best thing about the future is that it comes one day at a time - Abraham Lincoln

The term structure of interest rates describes how interest rates and bond yields vary across different maturities, typically illustrated using a yield curve. We examine basic concepts such as spot rates, forward rates, par rates, and yield-to-maturity, along with techniques for modeling the term structure, including yield curve construction, splines, and bootstrapping. To gain intuitive insights into the dynamics of interest rate movements, we also apply low-rank approximation methods such as Principal Component Analysis (PCA) and Singular Value Decomposition (SVD).

```
# By: Terence Lim, 2020-2025 (terence-lim.github.io)
import re
from typing import List, Dict
from datetime import datetime
import numpy as np
import numpy.linalg as la
from scipy.interpolate import CubicSpline
import pandas as pd
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.decomposition import PCA
from finds.readers import Alfred
from secret import credentials, paths
VERBOSE = 0
# %matplotlib qt

alf = Alfred(api_key=credentials['fred']['api_key'], convert_date=0, verbose=VERBOSE)
```

13.1 Interest Rates

The frequency at which interest is compounded determines how an interest rate is measured. For example, an annual compounding rate assumes interest is compounded once per year, while a semi-annual compounding rate assumes two compounding periods per year.

The **Annualized Percentage Yield (APY)** or **effective annual yield** accounts for the effects of compounding. The conversion formula is:

$$APY = \left(1 + \frac{R_m}{m}\right)^m - 1$$

where m represents the number of compounding periods per year.

To convert an interest rate R_m with compounding frequency m to an equivalent **continuously compounded rate** r , we use the formula:

$$e^{rt} = \left(1 + \frac{R_m}{m}\right)^{mt}$$

The **spot rate** for a given maturity is the zero-coupon rate applicable to that period. It is directly related to the discount factor:

$$\text{Discount factor} = \left(1 + \frac{R_m}{m}\right)^{-t}$$

The **forward rate** is the implied future interest rate derived from observed spot rates. Longer-term spot rates can be determined by compounding forward rates.

A bond's **yield to maturity (YTM)** is the single discount rate that equates the bond's present value of cash flows to its market price.

The **par rate** is the coupon rate at which a bond is issued at par value. A bond priced at par has a yield to maturity equal to its coupon rate.

The **term structure of interest rates** represents the relationship between yields and maturities. When plotted, it forms a **yield curve**. An upward-sloping curve indicates that forward rates are higher than spot and par rates, while a downward-sloping curve suggests the opposite.

Swap rates represent fixed interest rates exchanged for floating rates in an interest rate swap and are considered par rates.

```
# retrieve Constant Maturity Treasuries, excluding inflation-indexed and discontinued
cat = alf.get_category(115)  # Fed H.15 Selected Interest Rates
print('Retrieved category:', cat['id'], cat['name'])
```

Retrieved category: 115 Treasury Constant Maturity

```
treas = DataFrame.from_dict(
    {s['id']: [s['observation_start'], s['frequency'], s['title'].split(',')[0][44:]]
     for s in cat['series'] if 'Inflation' not in s['title'] and
     'DISCONT' not in s['title'] and s['frequency'] in ['Daily', 'Monthly']},
    columns = ['start', 'freq', 'title'],
    orient='index').sort_values(['freq', 'start'])
print("Constant Maturity Treasuries in FRED")
pd.set_option('display.max_colwidth', None)
treas
```

Constant Maturity Treasuries in FRED

	start	freq	title
DGS1	1962-01-02	Daily	1-Year Constant Maturity
DGS10	1962-01-02	Daily	10-Year Constant Maturity
DGS20	1962-01-02	Daily	20-Year Constant Maturity
DGS3	1962-01-02	Daily	3-Year Constant Maturity
DGS5	1962-01-02	Daily	5-Year Constant Maturity
DGS7	1969-07-01	Daily	7-Year Constant Maturity
DGS2	1976-06-01	Daily	2-Year Constant Maturity
DGS30	1977-02-15	Daily	30-Year Constant Maturity
DGS3MO	1981-09-01	Daily	3-Month Constant Maturity
DGS6MO	1981-09-01	Daily	6-Month Constant Maturity

(continues on next page)

(continued from previous page)

DGS1MO	2001-07-31	Daily	1-Month	Constant	Maturity
GS1	1953-04-01	Monthly	1-Year	Constant	Maturity
GS10	1953-04-01	Monthly	10-Year	Constant	Maturity
GS20	1953-04-01	Monthly	20-Year	Constant	Maturity
GS3	1953-04-01	Monthly	3-Year	Constant	Maturity
GS5	1953-04-01	Monthly	5-Year	Constant	Maturity
GS7	1969-07-01	Monthly	7-Year	Constant	Maturity
GS2	1976-06-01	Monthly	2-Year	Constant	Maturity
GS30	1977-02-01	Monthly	30-Year	Constant	Maturity
GS3M	1981-09-01	Monthly	3-Month	Constant	Maturity
GS6M	1981-09-01	Monthly	6-Month	Constant	Maturity
GS1M	2001-07-01	Monthly	1-Month	Constant	Maturity

```
# infer maturity from label
daily = pd.concat([alf(s, freq='D')
                   for s in treas.index if treas.loc[s, 'freq']=='Daily'],
                   axis=1, join='outer')
daily.columns = [int(re.sub('\D', '', col)) * (1 if col[-1].isalpha() else 12) # MO
                 ↪or M
                   for col in daily.columns] # infer maturity in months from label
daily = daily.rename_axis(columns='maturity').sort_index(axis=1)
daily
```

maturity	1	3	6	12	24	36	60	84	120	240	360
date											
1962-01-02	NaN	NaN	NaN	3.22	NaN	3.70	3.88	NaN	4.06	4.07	NaN
1962-01-03	NaN	NaN	NaN	3.24	NaN	3.70	3.87	NaN	4.03	4.07	NaN
1962-01-04	NaN	NaN	NaN	3.24	NaN	3.69	3.86	NaN	3.99	4.06	NaN
1962-01-05	NaN	NaN	NaN	3.26	NaN	3.71	3.89	NaN	4.02	4.07	NaN
1962-01-08	NaN	NaN	NaN	3.31	NaN	3.71	3.91	NaN	4.03	4.08	NaN
...
2025-02-21	4.36	4.32	4.30	4.15	4.19	4.19	4.26	4.35	4.42	4.69	4.67
2025-02-24	4.36	4.31	4.30	4.15	4.13	4.17	4.23	4.32	4.40	4.69	4.66
2025-02-25	4.34	4.30	4.28	4.12	4.07	4.08	4.12	4.21	4.30	4.59	4.55
2025-02-26	4.35	4.31	4.28	4.12	4.05	4.04	4.06	4.16	4.25	4.55	4.51
2025-02-27	4.38	4.32	4.28	4.13	4.07	4.05	4.09	4.19	4.29	4.59	4.56

[15774 rows x 11 columns]

```
monthly = pd.concat([alf(s, freq='M')
                      for s in treas.index if treas.loc[s, 'freq']=='Monthly'],
                      axis=1, join='outer')
monthly.columns = [int(re.sub('\D', '', col)) * (1 if col[-1].isalpha() else 12) # MO or M
                   for col in monthly.columns] # infer maturity in months from label
monthly = monthly.rename_axis(columns='maturity').sort_index(axis=1)
monthly
```

maturity	1	3	6	12	24	36	60	84	120	240	360
date											
1953-04-30	NaN	NaN	NaN	2.36	NaN	2.51	2.62	NaN	2.83	3.08	NaN
1953-05-31	NaN	NaN	NaN	2.48	NaN	2.72	2.87	NaN	3.05	3.18	NaN
1953-06-30	NaN	NaN	NaN	2.45	NaN	2.74	2.94	NaN	3.11	3.21	NaN

(continues on next page)

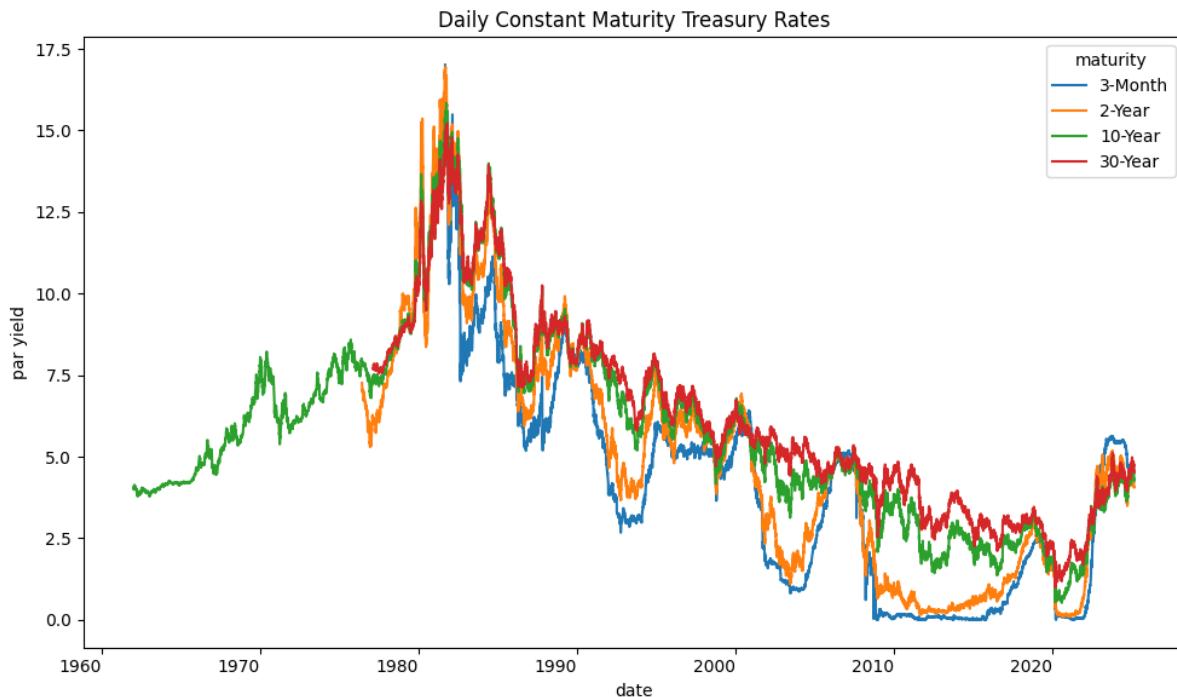
(continued from previous page)

1953-07-31	NaN	NaN	NaN	2.38	NaN	2.62	2.75	NaN	2.93	3.12	NaN
1953-08-31	NaN	NaN	NaN	2.28	NaN	2.58	2.80	NaN	2.95	3.10	NaN
...
2024-09-30	5.06	4.92	4.55	4.03	3.62	3.51	3.50	3.60	3.72	4.10	4.04
2024-10-31	4.92	4.72	4.44	4.20	3.97	3.90	3.91	3.99	4.10	4.44	4.38
2024-11-30	4.71	4.62	4.43	4.33	4.26	4.21	4.23	4.29	4.36	4.63	4.54
2024-12-31	4.50	4.39	4.32	4.23	4.23	4.22	4.25	4.32	4.39	4.66	4.58
2025-01-31	4.42	4.34	4.26	4.18	4.27	4.33	4.43	4.53	4.63	4.92	4.85

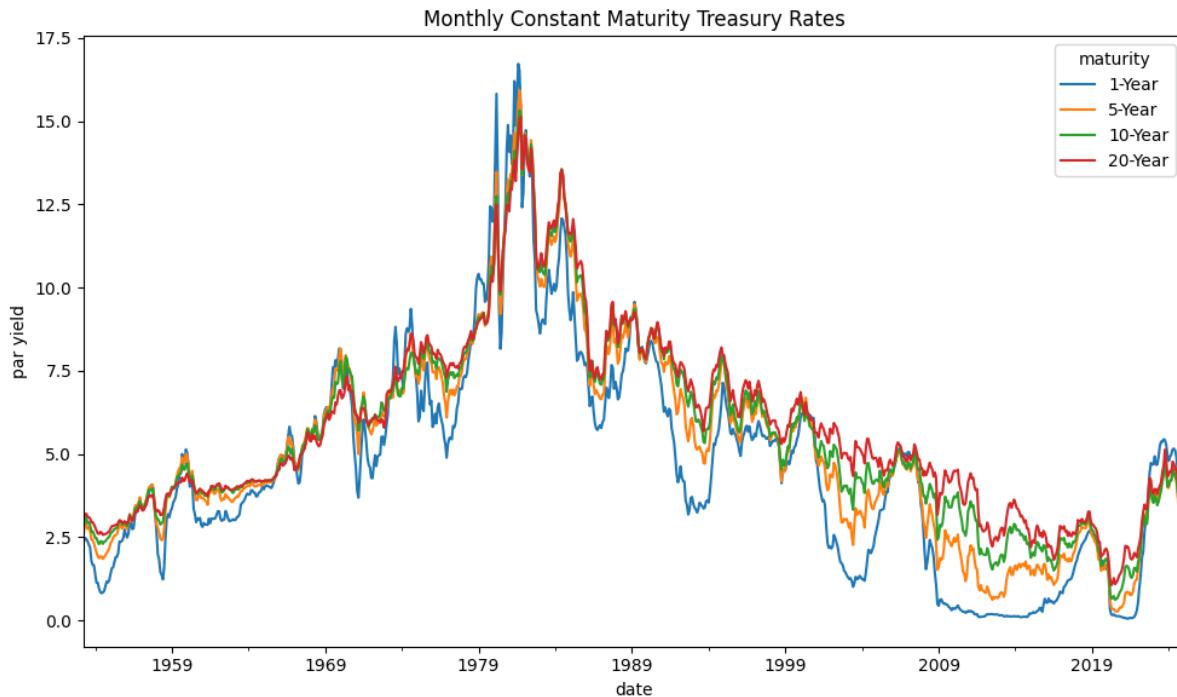
[862 rows x 11 columns]

```
# mapper to display maturity months as labels
mapper = lambda month: f"{month}-Month" if month < 12 else f"{int(month/12)}-Year"
```

```
cols = [3, 24, 120, 360]
fig, ax = plt.subplots(figsize=(10, 6))
daily[cols].rename(columns=mapper).plot(ax=ax, rot=0)
plt.title('Daily Constant Maturity Treasury Rates')
plt.ylabel('par yield')
plt.tight_layout()
```



```
cols = [12, 60, 120, 240]
fig, ax = plt.subplots(figsize=(10, 6))
monthly[cols].rename(columns=mapper).plot(ax=ax, rot=0)
plt.title('Monthly Constant Maturity Treasury Rates')
plt.ylabel('par yield')
plt.tight_layout()
```



13.2 Yield Curve

The U.S. Treasury's official **yield curve** is a par yield curve constructed using a monotone convex method. This curve is based on indicative bid-side price quotations collected by the Federal Reserve Bank of New York at approximately 3:30 PM each trading day.

Historically, the Treasury used a quasi-cubic Hermite spline (HS) method to construct the yield curve. This approach interpolated yields directly from observed market data under the assumption that the resulting curve was a par yield curve. However, since December 6, 2021, the monotone convex method has been the standard.

Constant Maturity Treasury (CMT) yields are derived from the Treasury's par yield curve and represent bond-equivalent yields for semiannual interest-paying securities. These yields are expressed on a simple annualized basis rather than an effective annual yield (APY) basis, which incorporates compounding effects. To convert a CMT yield to APY, use:

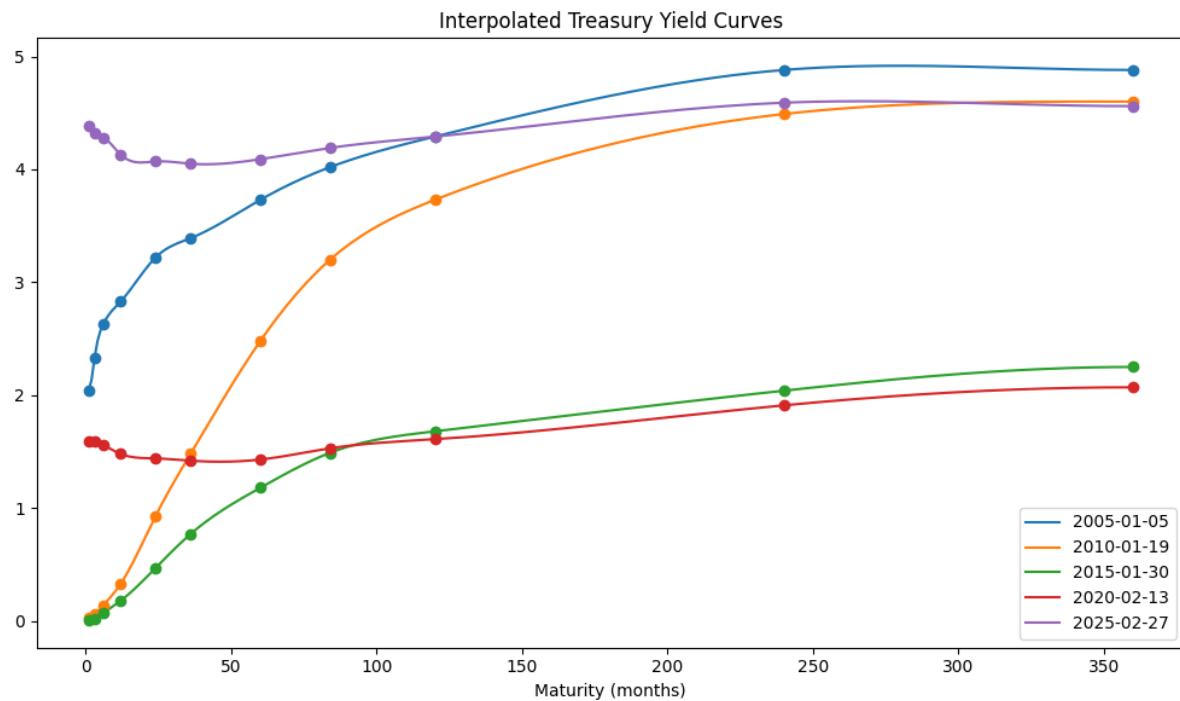
$$APY = \left(1 + \frac{y}{2}\right)^2 - 1$$

13.2.1 Splines

Spline interpolation is used to estimate yields at maturities not directly observed in the market. A **piecewise cubic polynomial** ensures smooth transitions between maturities and maintains twice continuous differentiability.

```
yield_curve = dict()
curve_dates = sorted(daily.dropna().index[-1:0:-(5*252)])
for date in curve_dates:
    yield_curve[date] = CubicSpline(
        x=daily.columns.to_list(), y=daily.loc[date].values, bc_type='clamped')
```

```
# Plot historical yield curves
fig, ax = plt.subplots(figsize=(10, 6))
X = list(range(1, 361))
for col, (date, curve) in enumerate(yield_curve.items()):
    ax.plot(X, curve(X), label=date.strftime('%Y-%m-%d'), color=f'C{col}')
plt.legend()
for col, (date, curve) in enumerate(yield_curve.items()):
    daily.loc[date].plot(ax=ax, marker='o', ls='', color=f'C{col}', label=None)
plt.title('Interpolated Treasury Yield Curves')
plt.xlabel('Maturity (months)')
plt.tight_layout()
```



13.2.2 Bootstrap Method

Bootstrapping is a process used to derive spot rates by progressively solving for zero-coupon rates using available bond data. This iterative approach fits spot rates to increasingly longer maturities.

```
# To bootstrap 6-month spot rates from 6-month par yields
m = 2      # compounding periods per year
```

```
# list of 6-monthly maturities
maturities = list(range(int(12/m)), daily.columns[-1]+1, int(12/m)))
```

```
# Helper to bootstrap spot rates from par yield curve
def bootstrap_spot(coupon: float, spots: List[float], m: int, price: float=1) -> float:
    """Compute spot rate to maturity of par bond from yield and sequence of spots
```

(continues on next page)

(continued from previous page)

```

Args:
    coupon : Annual coupon rate
    spots : Simple annual spot rates each period (excluding last period before_
    ↪maturity
    m : Number of compounding periods per year
    price: Present value of bond

Returns:
    Simple spot interest rate till maturity
"""
if not spots:           # trivial one-period bond
    return coupon / price
n = len(spots) + 1      # number of coupons through maturity

# discount factors from given spot rates
discount = [(1 + spot/m)**(-(1+t)) for t, spot in enumerate(spots)]

# nominal amount of last payment
last_payment = 1 + coupon/m

# infer present value of last coupon and principal
last_pv = price - np.sum(discount) * coupon/m

# compute discount factor and annualize the effective rate
spot = ((last_payment/last_pv)**(1/n) - 1) * m
return spot

```

```

# select most recent yield curve
curve_date = curve_dates[-1]
yields = [yield_curve[curve_date](t) / 100 for t in maturities]
spots = []
for coupon in yields:
    spots.append(bootstrap_spot(coupon=coupon, spots=spots, m=m))
DataFrame({curve_date.strftime('%Y-%m-%d'): spots}, index=maturities).head()

```

	2025-0227
6	0.042800
12	0.041285
18	0.040591
24	0.040684
30	0.040639

Helper to compute bond prices

```

def bond_price(coupon: float, n: int, m: int, yields: float | List[float],
               par: float = 1) -> float:
    """Compute present value of bond given spot rates or yield-to-maturity

Args:
    coupon : Annual coupon rate
    n : Number of remaining coupons
    m : Number of compounding periods per year
    yields : Simple annual yield-to-maturity or spot rates each period
    par : face or par value of bond

```

(continues on next page)

(continued from previous page)

```

Returns:
    Present value of bond
"""
if not pd.api.types.is_list_like(yields):
    yields = [yields] * n           # same yield-to-maturity is spot rate every_
→period
assert len(yields) == n, "Number of yields must equal number of coupons"
    pv = [(1 + yields[t-1]/m)**(-t) * (coupon/m + (par if t == n else 0)))
          for t in range(1, n+1)]      # discount every period's payment, plus last face
return np.sum(pv)

# Sanity-check par bond price given spots
for t in range(len(yields)):
    price = bond_price(coupon=yields[t], n=t+1, m=2, yields=spots[:t+1])
    assert np.allclose(price, 1.0)    # discounted payments at spot rates must equal_
→price

# Compute forward rates from spot rates
def forwards_from_spots(spots: List[float], m: int, skip: int=0) -> List[float]:
    """Compute forward rates given spot rates

Args:
    spots : Sequence of simple annual spot rates
    m : Number of compounding periods per year
    skip: Number of initial periods skipped

Returns:
    List of forward rates, excluding first period of spot rates input
"""
    result = []
    assert len(spots) >= 2, "Require at least two spot rates as input"
    for t in range(1, len(spots)):
        n = skip + t
        numerator = (1 + spots[n]/m)**n           # discounted value of period n
        denominator = (1 + spots[n-1]/m)**(n-1)    # discounter value of period n-1
        result.append(((numerator / denominator) - 1) * m)
    return result

forwards = [spots[0]] + forwards_from_spots(spots=spots, m=m)

```

Plot current yield curve

```

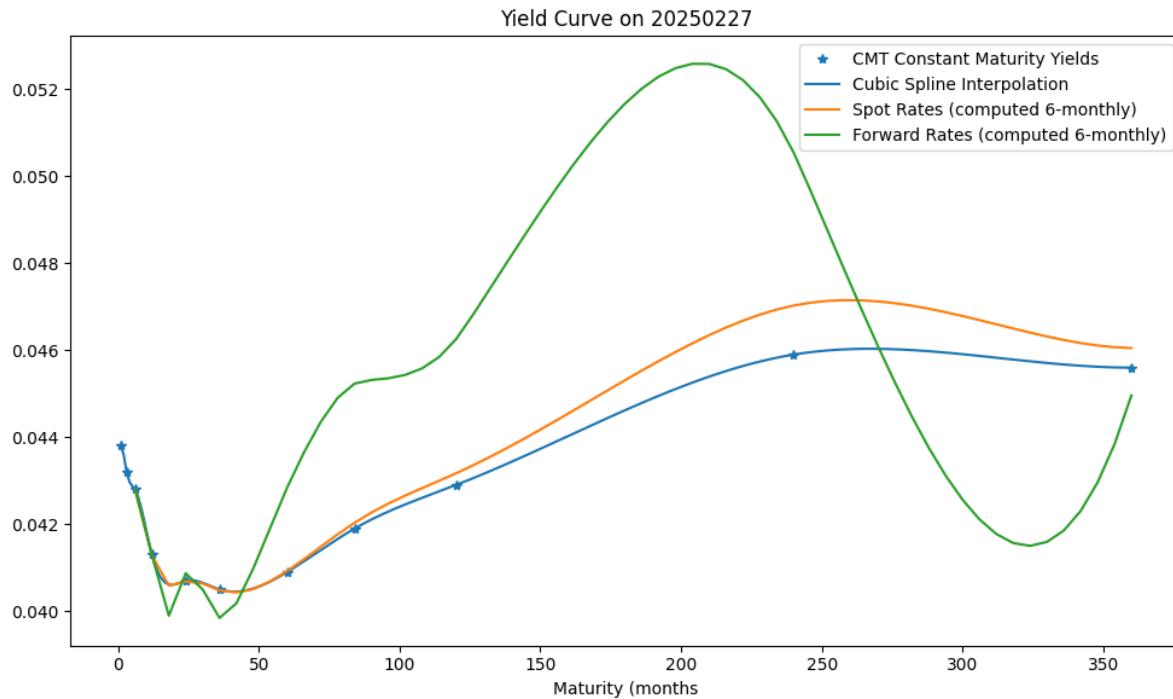
fig, ax = plt.subplots(figsize=(10, 6))
(daily.loc[curve_date] / 100).plot(marker='*', ls='', color="C0")
X = range(1, maturities[-1]+1)
ax.plot(X, [yield_curve[curve_date](t) / 100 for t in X], marker='', ls='-', color="C0"
→)
ax.plot(maturities, spots, marker='', ls='-', color="C1")
ax.plot(maturities, forwards, marker='', ls='-', color="C2")
plt.legend(['CMT Constant Maturity Yields', 'Cubic Spline Interpolation',
           'Spot Rates (computed 6-monthly)', 'Forward Rates (computed 6-monthly)'])
plt.title(f"Yield Curve on {curve_date.strftime('%Y-%m-%d')}")
plt.xlabel('Maturity (months)')

```

(continues on next page)

(continued from previous page)

```
plt.tight_layout()
```



There are more sophisticated methods for fitting yield curves to treasury prices, such as this model by Liu and Wu (2021)

```
# Download reconstructed yield curve data by Liu and Wu (2021)
# file_id from https://sites.google.com/view/jingcynthiawu/yield-data
file_id = '1-wmStGZHLx55dSYi3gQK2vb3F8dMw_Nb' # monthly
file_id = '11Hsxll_u2tBNt3FyN5iXGsIKLwxvVz7t' # daily
src = "https://drive.google.com/uc?export=download&id={}".format(file_id) # to load
# from gdrive
df = pd.ExcelFile(src).parse()
dates = np.where(df.iloc[:, 0].astype(str).str[0].str.isdigit())[0] # locate first
# date cell
```



```
liuwu = DataFrame(np.exp(df.iloc[dates, 1:361].astype(float).values/100) - 1,
                  index=pd.to_datetime(df.iloc[dates, 0], format="%Y%m")
                  + pd.offsets.MonthEnd(1)),
                  columns=np.arange(1, 361))
```



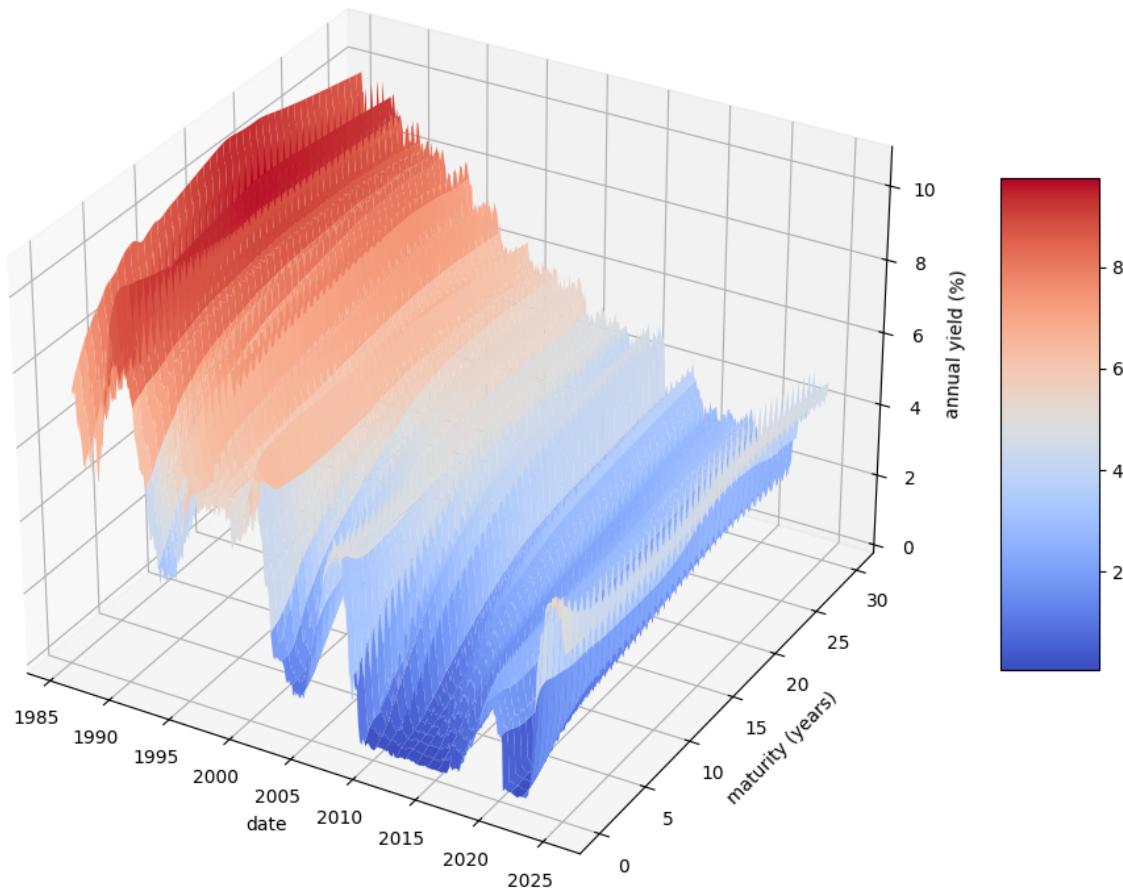
```
# Plot historical reconstructed rates as 3D surface
from mpl_toolkits.mplot3d import Axes3D
r = liuwu.dropna()
X, Y = np.meshgrid(r.index.map(lambda x: float(str(x)[:4]) + float(str(x)[5:7])/12),
                    r.columns.astype(float)/12)
#X, Y = np.meshgrid((r.index//10000) + (((r.index//100)%100)-1)/12,
#                    r.columns.astype(float)/12)
Z = r.T.to_numpy()*100
fig = plt.figure(num=1, clear=True, figsize=(10, 8))
ax = plt.axes(projection='3d')
```

(continues on next page)

(continued from previous page)

```
f = ax.plot_surface(X, Y, Z, cmap='coolwarm', linewidth=0, antialiased=True)
ax.set_title('Reconstructed Treasury Interest Rates [Liu and Wu (2020)]')
ax.set_xlabel('date')
ax.set_ylabel('maturity (years)')
ax.set_zlabel('annual yield (%)')
fig.colorbar(f, shrink=0.5, aspect=5)
plt.tight_layout()
```

Reconstructed Treasury Interest Rates [Liu and Wu (2020)]



```
# Plot historical Yield Curves
curve_dates = sorted(liuwu.index[-1:0:-(7*12)])[-4:]
for date in curve_dates:
    yield_curve[date] = CubicSpline(
        x=monthly.columns.to_list(), y=monthly.loc[date].values, bc_type='clamped')
```

```
fig, axes = plt.subplots(2,2, figsize=(12, 8), sharey=True, sharex=True)
axes = axes.flatten()
for num, (curve_date, ax) in enumerate(zip(curve_dates, axes)):
    # fit yields
    yields = [yield_curve[curve_date](t) / 100 for t in maturities]
```

(continues on next page)

(continued from previous page)

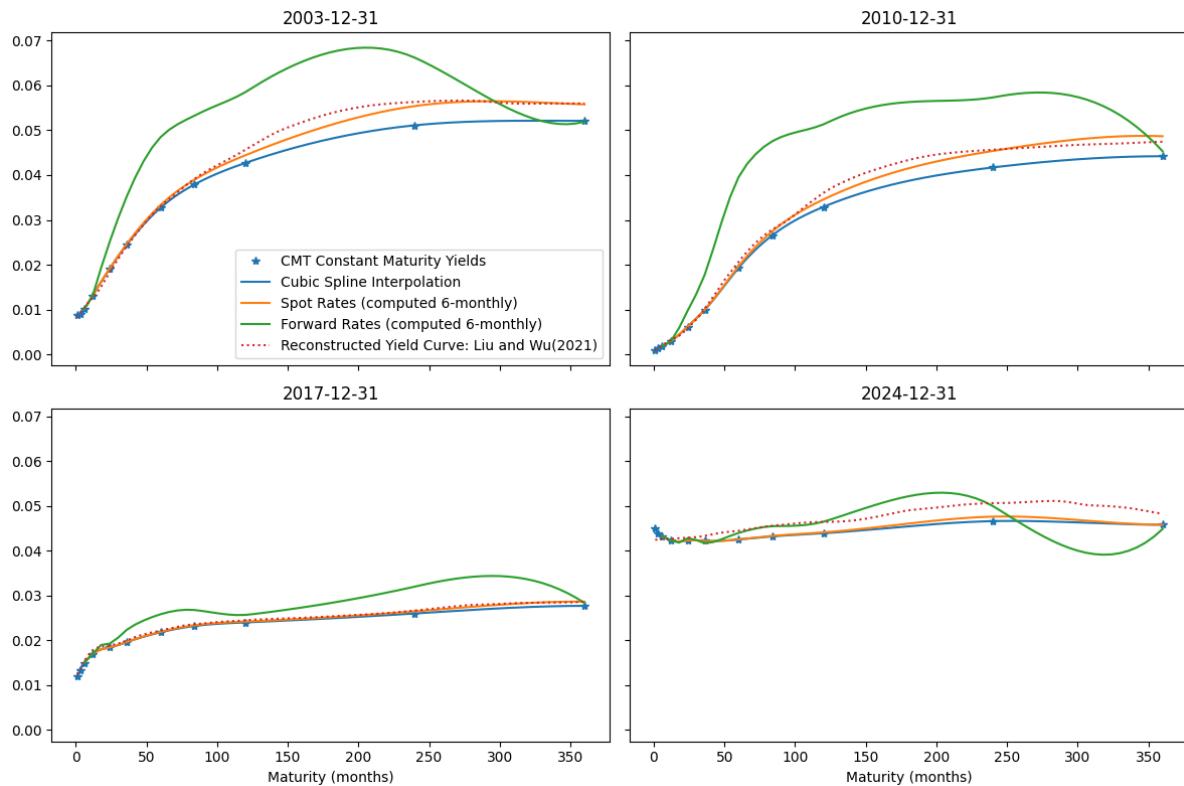
```

# compute spots
spots = []
for coupon in yields:
    spots.append(bootstrap_spot(coupon=coupon, spots=spots, m=m))

# compute forwards
forwards = [spots[0]] + forwards_from_spots(spots=spots, m=m)

# plot
(monthly.loc[curve_date] / 100).plot(marker='*', ls='', color="C0", ax=ax) # CMT
yield
X = range(1, maturities[-1]+1)
ax.plot(X, [yield_curve[curve_date](t) / 100 for t in X], marker='.', ls='-', color="C0")
ax.plot(maturities, spots, marker='.', ls='-', color="C1")
ax.plot(maturities, forwards, marker='.', ls='-', color="C2")
liuwu.loc[curve_date].plot(ax=ax, marker=':', ls=':', color="C3") # Liu and Wu
ax.set_title(f'{curve_date.strftime("%Y-%m-%d")}')
ax.set_xlabel('Maturity (months)')
if not num:
    ax.legend(['CMT Constant Maturity Yields', 'Cubic Spline Interpolation',
              'Spot Rates (computed 6-monthly)', 'Forward Rates (computed 6-monthly)',
              'Reconstructed Yield Curve: Liu and Wu(2021)'])
plt.tight_layout()

```



13.3 Principal Component Analysis (PCA)

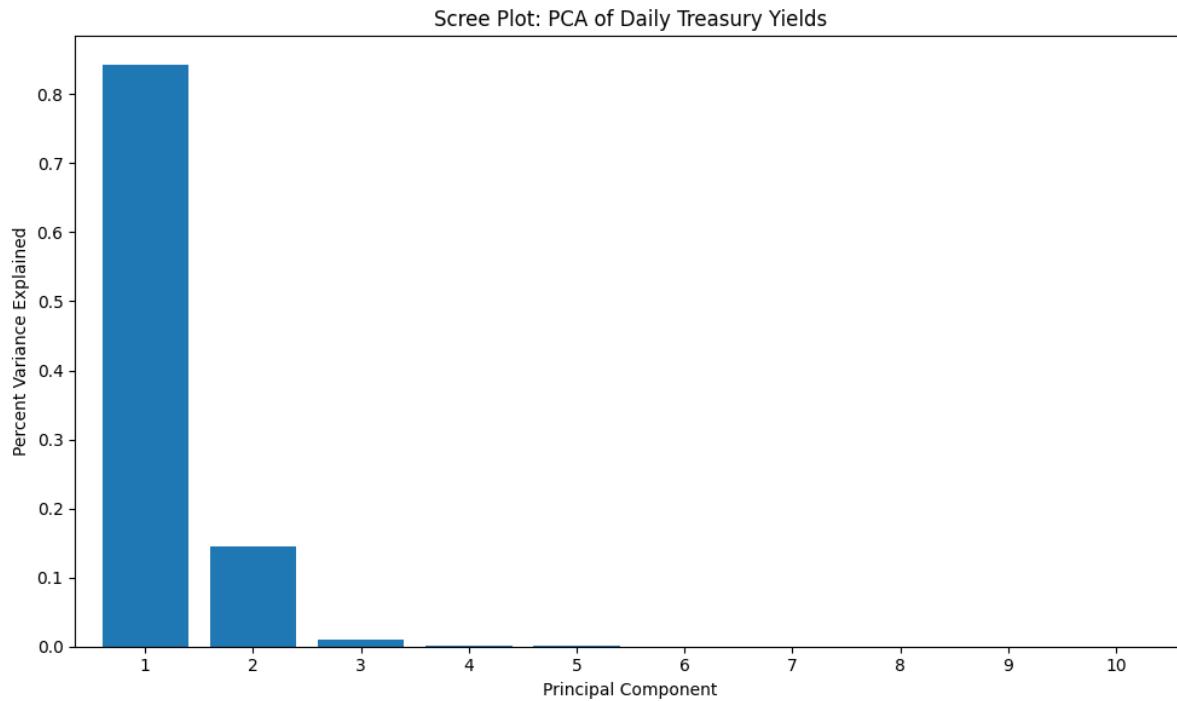
Principal Component Analysis (PCA), also referred to as **eigendecomposition**, involves rotating the data space so that each axis captures the maximum variance. We apply PCA to identify the key factors influencing daily interest rate in Constant Maturity Treasury (CMT) rates.

The number of factors in PCA equals the number of interest rate maturities analyzed. The first principal component accounts for 84% of daily yield variance, while the first two components explain almost 99%. The first three components together account for approximately 99.9%, meaning most of the uncertainty in yield changes can be attributed to these factors.

```
X = daily.dropna()
Y = StandardScaler().fit_transform(X)
pca = PCA().fit(Y)
DataFrame({'Cumulative Variance Explained': pca.explained_variance_ratio_.cumsum()},
          index=[f"PC {c+1}" for c in range(pca.n_components_)])
```

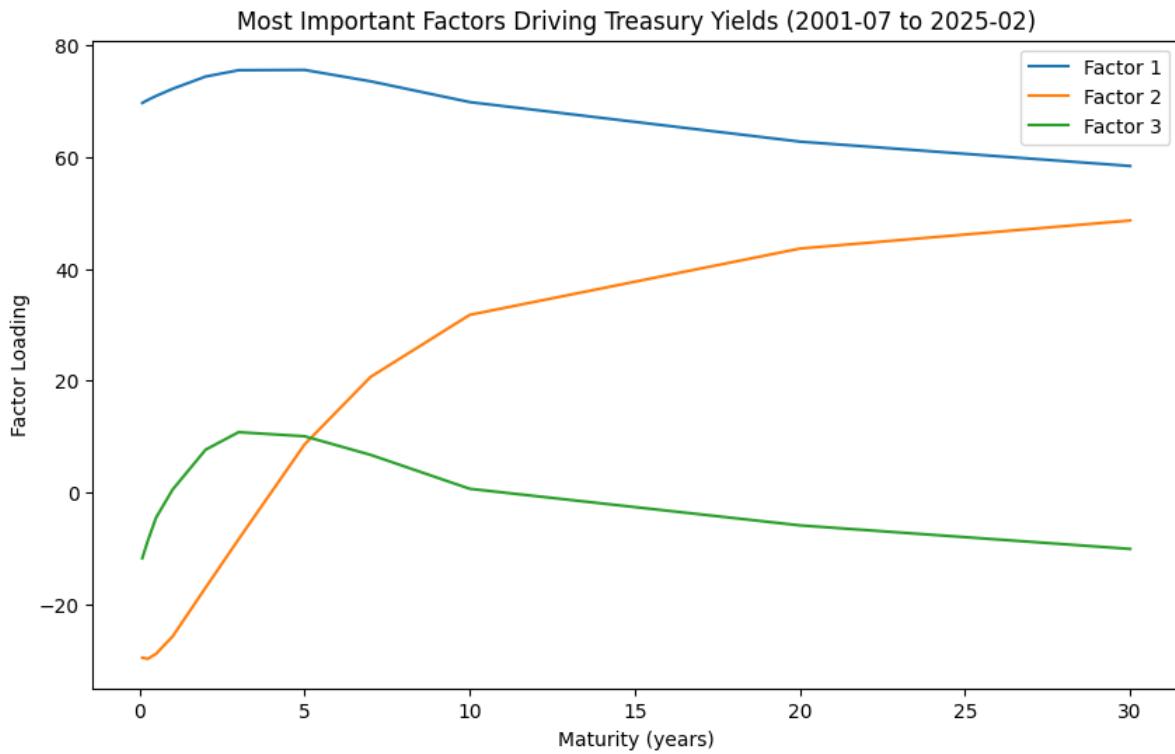
	Cumulative Variance Explained
PC 1	0.841980
PC 2	0.986916
PC 3	0.997595
PC 4	0.998925
PC 5	0.999401
PC 6	0.999727
PC 7	0.999866
PC 8	0.999929
PC 9	0.999965
PC 10	0.999986
PC 11	1.000000

```
# Scree plot
scree = Series(pca.explained_variance_ratio_,
               index=np.arange(1, Y.shape[1] + 1))
fig, ax = plt.subplots(figsize=(10, 6))
scree[:10].plot(kind='bar', rot=0, width=.8, ax=ax)
ax.set_title('Scree Plot: PCA of Daily Treasury Yields')
ax.set_ylabel("Percent Variance Explained")
ax.set_xlabel("Principal Component")
plt.tight_layout()
```



```
# Factor Loadings
Z = pca.components_[:, :].T * pca.singular_values_[:3].reshape(1, 3)
loadings = DataFrame(Z, columns=[f"Factor {n+1}" for n in range(3)], index=X.columns[12])
fig, ax = plt.subplots(figsize=(10, 6))
loadings.plot(ax=ax)
ax.set_title(f"Most Important Factors Driving Treasury Yields"
             f" ({str(X.index[0])[:7]} to {str(X.index[-1])[:7]})")
ax.set_xlabel('Maturity (years)')
ax.set_ylabel('Factor Loading')
```

```
Text(0, 0.5, 'Factor Loading')
```



Key Factors Affecting Treasury Yields

1. **Level Factor** – Represents a parallel shift in yields across all maturities.
2. **Slope Factor** – Indicates changes in the steepness of the yield curve, where short-term and long-term rates move in opposite directions.
3. **Twist Factor** – Describes changes in the curvature of the yield curve, where intermediate rates move differently from short- and long-term rates.

13.4 Singular Value Decomposition (SVD)

Singular Value Decomposition (SVD) is a factorization method that decomposes a matrix into three components:

$$A = USV^T$$

where:

- U and V are orthogonal matrices representing rotations.
- S is a diagonal matrix containing singular values (scaling factors).

SVD generalizes eigenvalue decomposition:

$$A^T A = V \Lambda V^T$$

where Λ contains the eigenvalues.

A goal of both PCA and SVD is to approximate the original data matrix with a lower-dimensional presentation, with the most important eigen- and singular vectors associated with the largest eigenvalues and singular values respectively.

- **Eigenvalues** (λ) in PCA correspond to squared singular values in SVD.

- **Principal components** (columns of V) are the **eigenvectors** or **right singular vectors**.
- **Loadings** are obtained by multiplying each principal component by its corresponding singular value.
- **Projections (scores)** are computed by projecting data onto the principal components.

```
# A is num_samples (N) by num_features (K) data matrix, standardized by column
A = daily.dropna().values
A = (A - A.mean(axis=0)) / A.std(axis=0) # subtract mean, divide by std
N, K = A.shape
A.shape
```

(5896, 11)

```
# svd x is related to pca of x'x
u, s, vT = np.linalg.svd(A, full_matrices=False)
v = vT.T # transposed right singular vector was returned
```

The **eigenvalues** (λ) of PCA

- can be retrieved as (N-1) times `pca.explained_variance_`
- are equal to the squares of the singular values (s from SVD)

```
# s**2 = lambda = N * explained_variance
assert np.allclose((N-1) * pca.explained_variance_, s**2), 'eigenvalues values'
assert np.allclose(pca.singular_values_, s), "singular values"
```

The **components** (columns of V) of an eigendecomposition

- are also called the **eigenvectors**, or **right singular vectors** from the SVD
- can be retrieved from the rows of `pca.components_`, or
- are the rows of V^T (identically, the columns of V from SVD)

Relatedly:

- **loadings** are computed by multiplying each component by its corresponding singular value $v \cdot s$

```
# components and right singular vectors are identical up to sign flip
for pc in range(K):
    assert (np.allclose(vT[pc, :], pca.components_[pc, :]) or
            np.allclose(vT[pc, :], -pca.components_[pc, :]))
```

```
# square of loadings is same as square of data matrix, i.e. the covariance matrix
loadings = np.diag(pca.singular_values_) @ pca.components_
assert np.allclose(A.T @ A, loadings.T @ loadings), 'square matrix'
```

The **projections** of PCA

- are also known as the **scores** or **co-ordinates**
- are computed by projecting the data matrix on the components $A \cdot V$
- or by scaling each left singular vector by its corresponding singular value $u \cdot s$
- can be retrieved by calling `pca.transform()` on the data matrix

```
# assert: x @ v == transform(x) (aka projection on components)
y = pca.transform(A)
for pc in range(K):
    assert np.allclose((A @ v)[:,pc], -y[:,pc]) or np.allclose((A @ v)[:,pc], y[:,pc])
    assert np.allclose(u[:,pc] * s[pc], -y[:,pc]) or np.allclose(u[:,pc] * s[pc], y[:,pc])
```

13.5 Low-Rank Approximations

PCA and SVD allow approximating the original data matrix with a lower-dimensional representation. A rank- k approximation is a technique to find a matrix A_k with lower rank k that is as close as possible to the original matrix A in terms of some measure, typically the Frobenius norm. This approximation reduces the complexity of the data while retaining its most important features.

13.5.1 Low-rank approximation by PCA

$$A \approx A'_k A_k = V_{[k]} D_{[k]} V'_{[k]}$$

```
ATA = A.T.dot(A)
eigval, eigvec = (N-1)*pca.explained_variance_, pca.components_.T
assert np.allclose(eigvec.dot(np.diag(eigval)).dot(eigvec.T), ATA), "pca error"
```

```
print('rank-K PCA approximation:')
DataFrame.from_dict({k: (la.norm(
    eigvec[:, :k].dot(np.diag(eigval[:k])).dot(eigvec[:, :k].T) - ATA) / la.norm(ATA))
    for k in range(1, 5)}, orient='index', columns=['Frobenius Norm']) \
    .rename_axis(index='K')
```

rank-K PCA approximation:

K	Frobenius Norm
1	0.170097
2	0.012614
3	0.001707
4	0.000700

13.5.2 Low-rank approximation by SVD

$$A \approx A_k = U_{[k]} S_{[k]} V'_{[k]}$$

```
assert np.allclose(u.dot(np.diag(s)).dot(v.T), A), "svd error"
```

```
print('rank-K SVD approximation:')
DataFrame.from_dict({k: la.norm(
    u[:, :k].dot(np.diag(s[:k])).dot(v[:, :k].T) - A) / la.norm(A)
    for k in range(1, 5)}, orient='index', columns=['Frobenius Norm']) \
    .rename_axis(index='K')
```

rank-K SVD approximation:

Frobenius Norm
K
1 0.397517
2 0.114383
3 0.049039
4 0.032781

References:

FRM Part 1 Exam Book Valuation and Risk Models, Chapter 12-13

Yan Liu and Jing Cynthia Wu “Reconstructing the Yield Curve”, Journal of Financial Economics, 2021, 142 (3), 1395-1425.

<https://home.treasury.gov/policy-issues/financing-the-government/interest-rate-statistics/treasury-yield-curve-methodology>

INTEREST RATE RISK

The essence of investment management is the management of risks, not the management of returns - Benjamin Graham

Changes in interest rates directly affect the value of bonds and fixed-income portfolios. Various measures of interest rate sensitivity, such as duration, convexity, DV01, and key rate shifts help quantify how bond prices respond to fluctuations in the yield curve. Additionally, we explore the statistical approach to identifying key risk factors that explain interest rate movements.

```
# By: Terence Lim, 2020-2025 (terence-lim.github.io)
from typing import List
import numpy as np
import pandas as pd
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.decomposition import PCA
import statsmodels.formula.api as smf
from finds.readers import Alfred
from finds.recipes import bond_price
from secret import credentials
# %matplotlib qt
VERBOSE = 0
```

14.1 Interest rate sensitivity

14.1.1 Duration

Duration, or **Macaulay Duration**, measures how a bond's price P changes in response to an instantaneous change in its yield y .

Consider a bond with price P and yield y . To simplify the duration formula, we first express the yield using continuous compounding. The bond's price is given by:

$$P = \sum_{t=1}^n c_t e^{-yt}$$

Differentiating this equation, we obtain:

$$\Delta P = - \sum_{t=1}^n c_t t e^{-yt} \Delta y$$

Yield-based duration is defined as the proportional change in bond price for a small change in yield:

$$D = \frac{\Delta P}{P \Delta y} = \sum_{t=1}^n t \frac{c_t e^{-yt}}{P}$$

This formulation provides an alternative interpretation of duration: it represents the weighted average time at which cash flows are received, with each time weighted by the proportion of the bond's total value received at that time. This explains why the term “duration” is used to describe sensitivity to yield changes—duration effectively measures how long an investor must wait to receive the bond's cash flows.

Modified duration accounts for different compounding conventions. If yields are measured with semi-annual compounding instead of continuous compounding, Macaulay duration must be adjusted by dividing by $(1 + y_2/2)$ (or $1 + y_m/m$ for m -thly compounding).

Effective duration, on the other hand, measures the percentage change in the price of a bond with embedded options due to a small shift in all interest rates.

14.1.2 DV01

DV01 (Dollar Value of a 01) quantifies the impact of a one-basis-point change in interest rates on a bond or portfolio's value. It is given by:

$$DV01 = -\frac{\Delta P}{\Delta r}$$

where Δr represents a small parallel shift in the interest rate term structure (expressed in basis points).

14.1.3 Convexity

Duration and convexity appear as the first two terms in a Taylor expansion of a bond's price with respect to interest rates. While duration provides a linear estimate of price sensitivity, convexity measures the curvature of the bond price-yield relationship, refining estimates for larger rate movements.

Convexity accounts for the fact that bond price changes are not perfectly linear with respect to interest rate movements. **Yield-based convexity**, when yields are expressed with continuous compounding, is given by:

$$C = \frac{1}{P} \frac{1}{(1 + r/m)^2} \sum_{t=1}^T (t/2m + (t/m)^2) \frac{C_t}{(1 + r/m)^t}$$

This formula represents a weighted average of the squared time to maturity. When yields are expressed with semi-annual compounding, these expressions must be divided by $(1 + y_2/2)^2$ (or $(1 + y_m/m)^2$ for m -thly compounding), and the result is known as **modified convexity**.

Effective convexity measures how duration itself changes in response to interest rate shifts and is calculated as:

$$C = \frac{1}{P} \left[\frac{P^+ + P^- - 2P}{(\Delta r)^2} \right]$$

```
# Helpers to calculate duration and convexity
def macaulay_duration(coupon: float, n: int, m: int, price: float,
                      yields: float | List[float], par: float = 1, **kwargs) -> float:
    """Compute macaulay duration of a bond given spot rates or yield-to-maturity

    Args:
        coupon : Annual coupon rate
    """
    # ... (implementation details) ...

```

(continues on next page)

(continued from previous page)

```

n : Number of remaining coupons
m : Number of compounding periods per year
price : current market price of bond
yields : Simple annual yield-to-maturity or spot rates each period
par : face or par value of bond

Returns:
    Macaulay duration
"""
if not pd.api.types.is_list_like(yields):
    yields = [yields] * n           # same spot rate every period
assert len(yields) == n, "Number of spot rates must equal number of coupons"
pv = [(1 + yields[t-1]/m)**(-t) * (t/m) * (coupon/m + par*(t == n))
      for t in range(1, n+1)]      # discount every period's time-weighted payment
return np.sum(pv) / price

```

```

def modified_duration(coupon: float, n: int, m: int, price: float,
                      yields: float | List[float], par: float = 1, **kwargs) -> float:
    """Compute modified duration of a bond given spot rates or yield-to-maturity

```

Args:

```

    coupon : Annual coupon rate
    n : Number of remaining coupons
    m : Number of compounding periods per year
    price : current market price of bond
    yields : Simple annual yield-to-maturity or spot rates each period
    par : face or par value of bond

```

Returns:

```

    Modified duration
"""
assert not pd.api.types.is_list_like(yields), "Not Implemented"
ytm = yields
return (macaulay_duration(coupon=coupon, n=n, m=m, price=price, yields=yields,_
                           par=par)
        / (1 + ytm/2))

```

```

def modified_convexity(coupon: float, n: int, m: int, price: float,
                      yields: float | List[float], par: float = 1, **kwargs) ->_
float:
    """Compute modified convexity of a bond given spot rates or yield-to-maturity

```

Args:

```

    coupon : Annual coupon rate
    n : Number of remaining coupons
    m : Number of compounding periods per year
    price : current market price of bond
    yields : Simple annual yield-to-maturity or spot rates each period
    par : face or par value of bond

```

Returns:

```

    Modified convexity
"""
assert not pd.api.types.is_list_like(yields), "Not Implemented"
ytm = yields

```

(continues on next page)

(continued from previous page)

```

if not pd.api.types.is_list_like(yields):
    yields = [yields] * n           # same spot rate every period
assert len(yields) == n, "Number of spot rates must equal number of coupons"
pv = [(1 + yields[t-1]/m)**(-t) * ((t/m)**2 + t/(2*m)) * (coupon/m + par*(t == n))
      for t in range(1, n+1)]      # discount every period's time-weighted payment
return np.sum(pv) / (price * (1 + ytm/m)**2)
    
```

14.1.4 Barbells and bullets

Positive convexity benefits bondholders when there is a parallel shift in interest rates. Consider two fixed-income strategies:

- A barbell strategy, which consists of holding short- and long-maturity bonds
- A bullet strategy, which focuses on medium-term bonds

Both strategies may have the same yield (4%) and duration (8.1758), but the barbell strategy typically outperforms when interest rates shift in parallel. This creates an arbitrage opportunity:

1. Invest a given USD amount in the barbell strategy
2. Short the same USD amount in the bullet strategy

If term structure shifts were always parallel, this approach would be consistently profitable.

```

# Compute prices, duration and convexity for 3 bonds (FRM Valuation and Risk Models
# → 12.7)
bond5Y = dict(coupon=2, m=2, n=2*5, par=100, yields=0.04)
bond10Y = dict(coupon=4, m=2, n=2*10, par=100, yields=0.04)
bond20Y = dict(coupon=6, m=2, n=2*20, par=100, yields=0.04)
for bond in [bond5Y, bond10Y, bond20Y]:
    bond |= dict(price=bond_price(**bond))
    bond |= dict(duration=modified_duration(**bond))
    bond |= dict(convexity=modified_convexity(**bond))
bonds = DataFrame.from_dict(
    {f"b['n']/b['m']:.0f}-year, {b['coupon']}% coupon": [
        b['price'], b['duration'], b['convexity']]
    for b in [bond5Y, bond10Y, bond20Y]},
    orient='index',
    columns=['Value', 'Effective Duration', 'Effective Convexity'])
    
```

```

print("Table 12.4 Effective Durations and Convexities of Three Bonds")
bonds.round(4)
    
```

Table 12.4 Effective Durations and Convexities of Three Bonds

	Value	Effective Duration	Effective Convexity
5-year, 2% coupon	91.0174	4.6764	24.8208
10-year, 4% coupon	100.0000	8.1757	78.8979
20-year, 6% coupon	127.3555	12.6233	212.4587

```

# Compute 5Y and 20Y weights of barbell portfolio, with same duration as bullet 10Y
barbell = ((bond10Y['duration'] - bond20Y['duration']) /
    
```

(continues on next page)

(continued from previous page)

```
(bond5Y['duration'] - bond20Y['duration']))
print(f"Barbell: weight in 5Y = {barbell:.4f}, weight in 20Y = {1-barbell:.4f}")
```

Barbell: weight in 5Y = 0.5597, weight in 20Y = 0.4403

```
# Compare durations and convexities
DataFrame.from_dict(dict(Bullet=np.array([0,1,0]).dot(bonds),
                           Barbell=np.array([barbell, 0, 1-barbell]).dot(bonds)),
                           orient='index', columns=bonds.columns)\n                           [['Effective Duration', 'Effective Convexity']])
```

	Effective Duration	Effective Convexity
Bullet	8.175717	78.897925
Barbell	8.175717	107.444902

14.1.5 Key rate shifts

Consider three key spot rates: the two-year, five-year, and ten-year rates. Each influences rates in its surrounding maturity range, and together, their combined movements contribute to an overall one-basis-point shift in the yield curve.

These shifts, known as **key rate shifts**, allow for a more detailed decomposition of DV01. The impact of these shifts is captured by **partial 01s** or **key rate 01s (KR01s)**, defined as follows:

- **KR01₁**: The reduction in portfolio value from a one-basis-point increase in the two-year spot rate
- **KR01₂**: The reduction in portfolio value from a one-basis-point increase in the five-year spot rate
- **KR01₃**: The reduction in portfolio value from a one-basis-point increase in the ten-year spot rate

Since these shifts sum to the overall DV01, we have:

$$DV01 = KR01_1 + KR01_2 + KR01_3$$

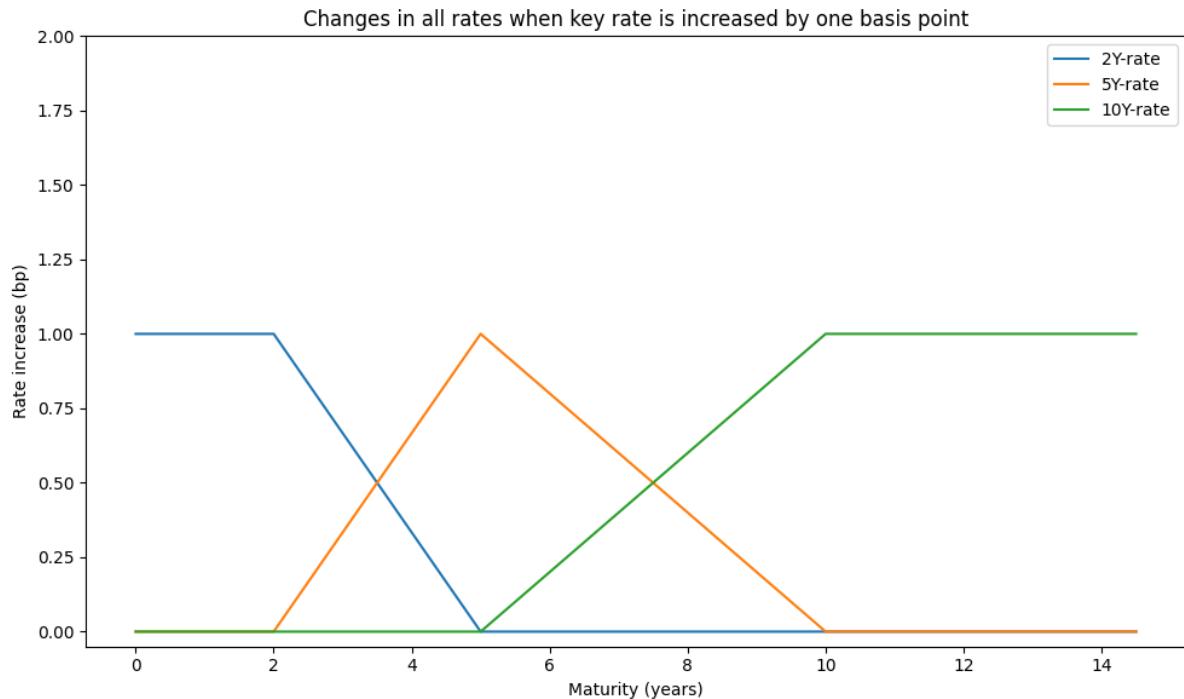
By analyzing these key rate sensitivities, investors can better estimate portfolio values under various term structure movements and hedge against specific interest rate risks.

```
KR = [2, 5, 10]
DV01 = dict()
for maturity in np.arange(0, 15, 0.5):
    if maturity <= KR[0]:
        changes = [1, 0, 0]
    elif maturity >= KR[2]:
        changes = [0, 0, 1]
    elif maturity < KR[1]:
        diff = (maturity - KR[0]) / (KR[1] - KR[0])
        changes = [1 - diff, diff, 0]
    else:
        diff = (KR[2] - maturity) / (KR[2] - KR[1])
        changes = [0, diff, 1 - diff]
    DV01[maturity] = changes
```

```

fig, ax = plt.subplots(figsize=(10, 6))
DataFrame.from_dict(DV01, orient='index', columns=[f"({y})Y-rate" for y in KR]) \
    .plot(ax=ax)
ax.legend()
ax.set_title('Changes in all rates when key rate is increased by one basis point')
ax.set_ylabel('Rate increase (bp)')
ax.set_xlabel('Maturity (years)')
ax.set_ylim(top=2)
plt.tight_layout()

```



14.2 Risk factors

We examine the risk factors which drive the daily returns of the Merrill Lynch Total Bond Indexes.

```
alf = Alfred(api_key=credentials['fred']['api_key'], verbose=VERBOSE)
```

```
# get Merrill Lynch bond indexes
freq = 'D'      # periodicity 'M' or 'D'
cat = alf.get_category(32413)
print(cat['id'], cat['name'])
```

32413 BofA Merrill Lynch Total Bond Return Index Values

```
# get bond index returns
bonds = []      # to accumulate bond returns
for s in cat['series']:
```

(continues on next page)

(continued from previous page)

```
bonds.append(alf(s['id'], start=19961231, freq=freq) )
bonds_df = pd.concat(bonds, axis=1).sort_index()
```

```
# show blocks of data availability
counts = bonds_df.notna().sum(axis=1).rename('count')
counts = pd.concat([counts, (counts != counts.shift()).cumsum().rename('notna')], axis=1)
counts = counts.reset_index().groupby(['notna', 'count'])['date'].agg(['first', 'last'])
counts
```

		first	last
notna	count		
1	15	19961231	19971230
2	16	19971231	19981230
3	33	19981231	20031230
4	48	20031231	20181108
5	46	20181109	20181109
6	48	20181112	20250228

Choose start date with good data availability

```
start_date = 19981231
rets = bonds_df.loc[bonds_df.index >= start_date,
                    bonds_df.loc[start_date].notna().values].iloc[:-1]
rets = pd.concat([alf.transform(rets[col], log=1, diff=1)
                  for col in rets.columns], axis=1).dropna()
date_str = f" ({rets.index[0]}-{rets.index[-1]})"
rets
```

	BAMLCC0A0CMTRIV	BAMLCC0A1AAATTRIV	BAMLCC0A2AATTRIV	BAMLCC0A3AATTRIV	\
date					
19990104	-0.001053	-0.001421	-0.000660	-0.001166	
19990105	-0.003846	-0.004194	-0.003500	-0.003857	
19990106	0.002388	0.002099	0.002102	0.002495	
19990107	-0.002694	-0.002813	-0.002413	-0.002573	
19990108	-0.002583	-0.002941	-0.002419	-0.002658	
...
20250221	0.004058	0.005418	0.004519	0.004090	
20250224	0.001879	0.002273	0.001855	0.001842	
20250225	0.005450	0.007181	0.006098	0.005468	
20250226	0.002036	0.002643	0.002200	0.002056	
20250227	-0.002092	-0.003094	-0.002047	-0.002056	
	BAMLCC0A4BBBTRIV	BAMLCC1A013YTRIV	BAMLCC2A035YTRIV	\	
date					
19990104	-0.000995	0.000076	0.000473		
19990105	-0.003952	-0.000844	-0.002452		
19990106	0.002419	0.000466	0.000901		
19990107	-0.002996	0.000101	-0.000853		
19990108	-0.002504	-0.000920	-0.002006		
...		
20250221	0.003918	0.001278	0.002482		
20250224	0.001917	0.000683	0.001165		

(continues on next page)

(continued from previous page)

20250225	0.005274	0.001025	0.002846		
20250226	0.001964	0.000422	0.001190		
20250227	-0.002106	0.000216	-0.000200		
	BAMLCC3A057YTRIV	BAMLCC4A0710YTRIV	BAMLCC7A01015YTRIV	...	\
date					
19990104	0.000534	-0.000519	-0.001209	...	
19990105	-0.003343	-0.004276	-0.004990	...	
19990106	0.001695	0.001967	0.002537	...	
19990107	-0.001561	-0.002662	-0.003371	...	
19990108	-0.003309	-0.003773	-0.004368	...	
...	
20250221	0.003648	0.004107	0.005264	...	
20250224	0.001481	0.001797	0.002408	...	
20250225	0.004326	0.005516	0.007383	...	
20250226	0.001797	0.002089	0.002762	...	
20250227	-0.000742	-0.001944	-0.002688	...	
	BAMLEMPTPRVICRPITRIV	BAMLEMRCRPIASIATRIV	BAMLEMRECRPIEMEATRIV		\
date					
19990104	0.001599	-0.000700	-0.005616		
19990105	0.002692	0.000400	0.008113		
19990106	0.007144	0.003395	0.009135		
19990107	-0.001484	-0.003095	0.007386		
19990108	-0.000594	0.001998	0.000392		
...	
20250221	0.001815	0.002360	0.001548		
20250224	0.001124	0.001355	0.001095		
20250225	0.002267	0.003170	0.002267		
20250226	0.001417	0.001305	0.001557		
20250227	-0.000023	0.000111	0.000705		
	BAMLEMRLCRPILATRIV	BAMLEMUBCRPIUSTRIV	BAMLHE00EHYITRIV		\
date					
19990104	0.002297	0.001099	0.000195		
19990105	0.002392	0.002394	0.012287		
19990106	0.006251	0.006060	0.000577		
19990107	0.000198	-0.000694	0.003932		
19990108	-0.002079	-0.000892	0.000000		
...	
20250221	0.001954	0.002056	0.000502		
20250224	0.000737	0.001161	0.000422		
20250225	0.002544	0.002939	0.000026		
20250226	0.002419	0.001755	0.000659		
20250227	-0.000991	-0.000062	0.000527		
	BAMLHYH0A0HYM2TRIV	BAMLHYH0A1BBTRIV	BAMLHYH0A2BTRIV		\
date					
19990104	0.001490	0.000250	0.001999		
19990105	0.000674	-0.000834	0.000781		
19990106	0.001487	0.001583	0.001041		
19990107	-0.000112	-0.001250	0.000520		
19990108	0.001737	-0.000918	0.002509		
...	
20250221	-0.000726	-0.000524	-0.001033		
20250224	0.001040	0.001016	0.001033		

(continues on next page)

(continued from previous page)

```

20250225      0.001028      0.001456      0.000758
20250226      0.001984      0.001797      0.001809
20250227     -0.000006      -0.000065      0.000063

      BAMLHYH0A3CMTRIV
date
19990104      0.003234
19990105      0.005428
19990106      0.003480
19990107      0.001005
19990108      0.007643
...
20250221     -0.000715
20250224      0.001213
20250225     -0.000062
20250226      0.003319
20250227      0.000046

[6830 rows x 33 columns]

```

Select these bond return indexes

```

pd.set_option('display.max_colwidth', None)
print("Bond Index Total Returns")
Series(alf.header(rrets.columns), index=rrets.columns, name='title') \
    .to_frame().rename_axis('series')

```

Bond Index Total Returns

	title
series	
BAMLCC0A0CMTRIV	ICE BofA US
↳Corporate Index Total Return Index Value	
BAMLCC0A1AAATRIV	ICE BofA AAA US
↳Corporate Index Total Return Index Value	
BAMLCC0A2AATRIV	ICE BofA AA US
↳Corporate Index Total Return Index Value	
BAMLCC0A3ATRIV	ICE BofA Single-A US
↳Corporate Index Total Return Index Value	
BAMLCC0A4BBBTRIV	ICE BofA BBB US
↳Corporate Index Total Return Index Value	
BAMLCC1A013YTRIV	ICE BofA 1-3 Year US
↳Corporate Index Total Return Index Value	
BAMLCC2A035YTRIV	ICE BofA 3-5 Year US
↳Corporate Index Total Return Index Value	
BAMLCC3A057YTRIV	ICE BofA 5-7 Year US
↳Corporate Index Total Return Index Value	
BAMLCC4A0710YTRIV	ICE BofA 7-10 Year US
↳Corporate Index Total Return Index Value	
BAMLCC7A01015YTRIV	ICE BofA 10-15 Year US
↳Corporate Index Total Return Index Value	
BAMLCC8A015PYTRIV	ICE BofA 15+ Year US
↳Corporate Index Total Return Index Value	

(continues on next page)

(continued from previous page)

BAMLEM1BRRAAA2ACRPITRIV	ICE BofA AAA-A Emerging Markets
↳ Corporate Plus Index Total Return Index Value	
BAMLEM2BRRBBBCRPITRIV	ICE BofA BBB Emerging Markets
↳ Corporate Plus Index Total Return Index Value	
BAMLEM3BRRBBCRPITRIV	ICE BofA BB Emerging Markets
↳ Corporate Plus Index Total Return Index Value	
BAMLEM4BRRBLCRPITRIV	ICE BofA B & Lower Emerging Markets
↳ Corporate Plus Index Total Return Index Value	
BAMLEM5BCOCRPITRIV	ICE BofA Crossover Emerging Markets
↳ Corporate Plus Index Total Return Index Value	
BAMLEMCBPITRIV	ICE BofA Emerging Markets
↳ Corporate Plus Index Total Return Index Value	
BAMLEMEBCRPITRIV	ICE BofA Euro Emerging Markets
↳ Corporate Plus Index Total Return Index Value	
BAMLEMSFCSRPITRIV	ICE BofA Private Sector Financial Emerging Markets
↳ Corporate Plus Index Total Return Index Value	
BAMLEMHBHYCRPITRIV	ICE BofA High Yield Emerging Markets
↳ Corporate Plus Index Total Return Index Value	
BAMLEMIBHGCRPITRIV	ICE BofA High Grade Emerging Markets
↳ Corporate Plus Index Total Return Index Value	
BAMLEMNSNFCRPITRIV	ICE BofA Non-Financial Emerging Markets
↳ Corporate Plus Index Total Return Index Value	
BAMLEMPBPUBSICRPITRIV	ICE BofA Public Sector Issuers Emerging Markets
↳ Corporate Plus Index Total Return Index Value	
BAMLEMPPTPRVICRPITRIV	ICE BofA Private Sector Issuers Emerging Markets
↳ Corporate Plus Index Total Return Index Value	
BAMLEMRACRPIASIATRIV	ICE BofA Asia Emerging Markets
↳ Corporate Plus Index Total Return Index Value	
BAMLEMRECRPIEMEATRIV	ICE BofA EMEA Emerging Markets
↳ Corporate Plus Index Total Return Index Value	
BAMLEMRLCRPILATRIV	ICE BofA Latin America Emerging Markets
↳ Corporate Plus Index Total Return Index Value	
BAMLEMUBCRPIUSTRIV	ICE BofA US Emerging Markets
↳ Corporate Plus Index Total Return Index Value	
BAMLHE00EHYITRIV	ICE BofA Euro
↳ High Yield Index Total Return Index Value	
BAMLHYH0A0HYM2TRIV	ICE BofA US
↳ High Yield Index Total Return Index Value	
BAMLHYH0A1BBTRIV	ICE BofA BB US
↳ High Yield Index Total Return Index Value	
BAMLHYH0A2BTRIV	ICE BofA Single-B US
↳ High Yield Index Total Return Index Value	
BAMLHYH0A3CMTRIV	ICE BofA CCC & Lower US
↳ High Yield Index Total Return Index Value	

14.2.1 Statistical risk factors

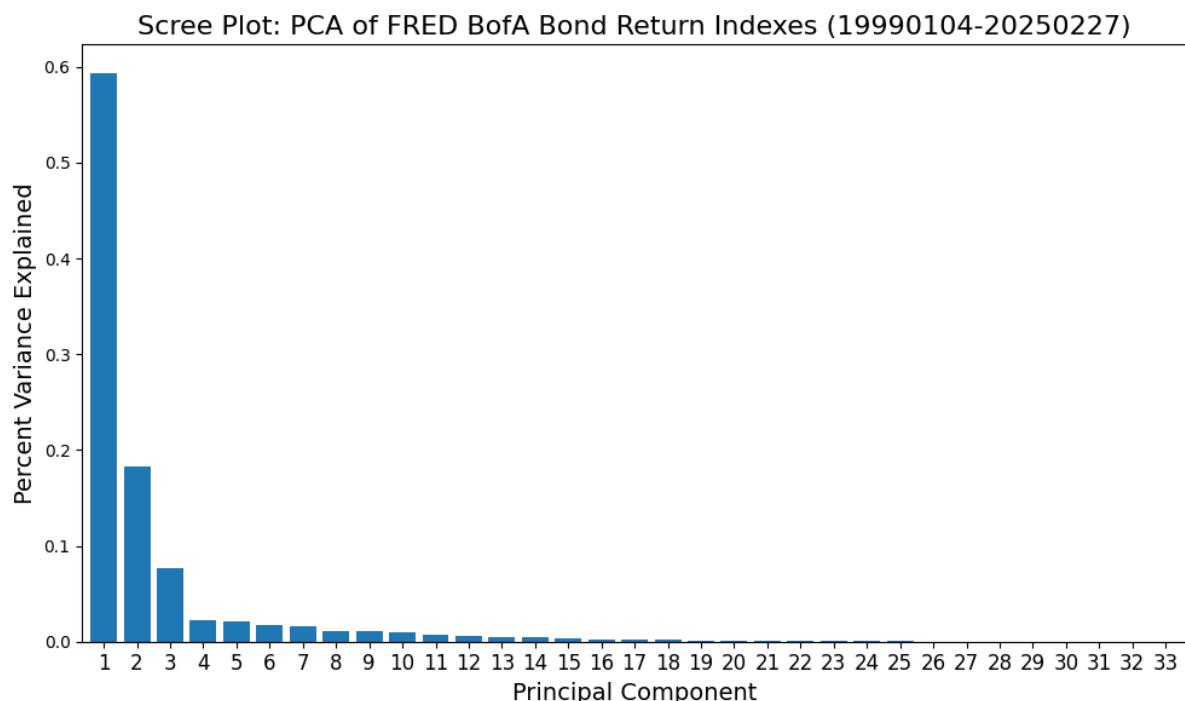
Extract principal components of bond index returns

```
pipe = Pipeline([('scaler', StandardScaler()), ('pca', PCA())])
pipe.fit(rets)
print(pipe.named_steps['pca'].explained_variance_ratio_) # sanity check
scree = Series(pipe.named_steps['pca'].explained_variance_ratio_,
               index=np.arange(1, rets.shape[1]+1))
DataFrame(scree.cumsum()).rename('Cumulative Variance Ratio Explained')).iloc[:10]
```

```
[5.93480704e-01 1.83414224e-01 7.60999397e-02 2.29460554e-02
 2.08374926e-02 1.73252063e-02 1.58102554e-02 1.11222261e-02
 1.07270153e-02 9.46337690e-03 7.12268856e-03 5.78660305e-03
 5.08617829e-03 4.56329291e-03 3.87130739e-03 2.53236096e-03
 2.40159698e-03 2.17255450e-03 1.28094767e-03 9.31393646e-04
 7.70330791e-04 6.01123712e-04 3.93600105e-04 3.60165273e-04
 3.03684121e-04 2.70804862e-04 9.27216331e-05 8.30786301e-05
 5.32780919e-05 3.92254907e-05 3.55714421e-05 1.53907425e-05
 5.60549950e-06]
```

	Cumulative Variance Ratio Explained
1	0.593481
2	0.776895
3	0.852995
4	0.875941
5	0.896778
6	0.914104
7	0.929914
8	0.941036
9	0.951763
10	0.961226

```
# Scree plot
fig, ax = plt.subplots(num=1, clear=True, figsize=(10, 6))
scree.plot(kind='bar', rot=0, width=.8, ax=ax)
ax.set_title('Scree Plot: PCA of FRED BofA Bond Return Indexes' + date_str,..  
→fontsize=16)
ax.xaxis.set_tick_params(labelsize=12)
ax.set_ylabel("Percent Variance Explained", fontsize=14)
ax.set_xlabel("Principal Component", fontsize=14)
plt.tight_layout()
```



14.2.2 Explainability of statistical risk factors

```
# Extract factor returns from bond indexes with PCA projection
K = 4
factors = DataFrame(pipe.transform(rets)[:, :K],
                     columns=[f"PC{c+1}" for c in range(K)],
                     index=pd.DatetimeIndex(rets.index.astype(str), freq='infer'))
```

Construct interest rate spreads to compare with the statistical factors:

- **Level:** The average of the two-year and ten-year Treasury rates
- **Slope:** The difference between the ten-year and two-year Treasury rates
- **Twist:** The difference between (ten-year minus five-year) and (five-year minus two-year) rates
- **Credit Spread:** The difference between BAA corporate bond yields and ten-year Treasury rates

```
# Construct interest rate spread changes
spreads = pd.concat([alf(s, freq=freq) for s in ['BAA10Y', 'DGS10', 'DGS5', 'DGS2']], axis=1) \
    .sort_index()
spreads.index = pd.DatetimeIndex(spreads.index.astype(str), freq='infer')
spreads['level'] = 0.5 * (spreads['DGS2'] + spreads['DGS10'])
spreads['credit'] = spreads['BAA10Y']
spreads['slope'] = spreads['DGS10'] - spreads['DGS2']
spreads['twist'] = ((spreads['DGS10'] - spreads['DGS5']) - (spreads['DGS5'] - spreads['DGS2']))
spreads = spreads.drop(columns=['BAA10Y', 'DGS10', 'DGS5', 'DGS2']) \
    .ffill() \
    .diff() \
    .dropna()
spreads
```

	level	credit	slope	twist
date				
1986-01-03	0.010	-0.04	1.776357e-15	1.776357e-15
1986-01-06	0.015	-0.01	1.000000e-02	-1.000000e-02
1986-01-07	-0.100	0.06	-6.000000e-02	-8.881784e-16
1986-01-08	0.155	-0.14	7.000000e-02	3.000000e-02
1986-01-09	0.160	-0.05	-4.000000e-02	-8.000000e-02
...
2025-02-24	-0.040	-0.01	4.000000e-02	-2.000000e-02
2025-02-25	-0.080	0.02	-4.000000e-02	6.000000e-02
2025-02-26	-0.035	0.01	-3.000000e-02	5.000000e-02
2025-02-27	0.030	0.01	2.000000e-02	0.000000e+00
2025-02-28	-0.065	0.05	3.000000e-02	-1.000000e-02

[9795 rows x 4 columns]

```
# Show correlations between bond factor returns and spread changes
data = pd.concat([spreads, factors], axis=1, join='inner')
corr = data.corr()
plt.imshow(corr**2, vmin=0, vmax=1, cmap='Purples')
plt.imshow(corr, vmin=-1, vmax=1, cmap='seismic')
plt.xticks(range(len(corr)), corr.index)
plt.yticks(range(len(corr)), corr.index)
```

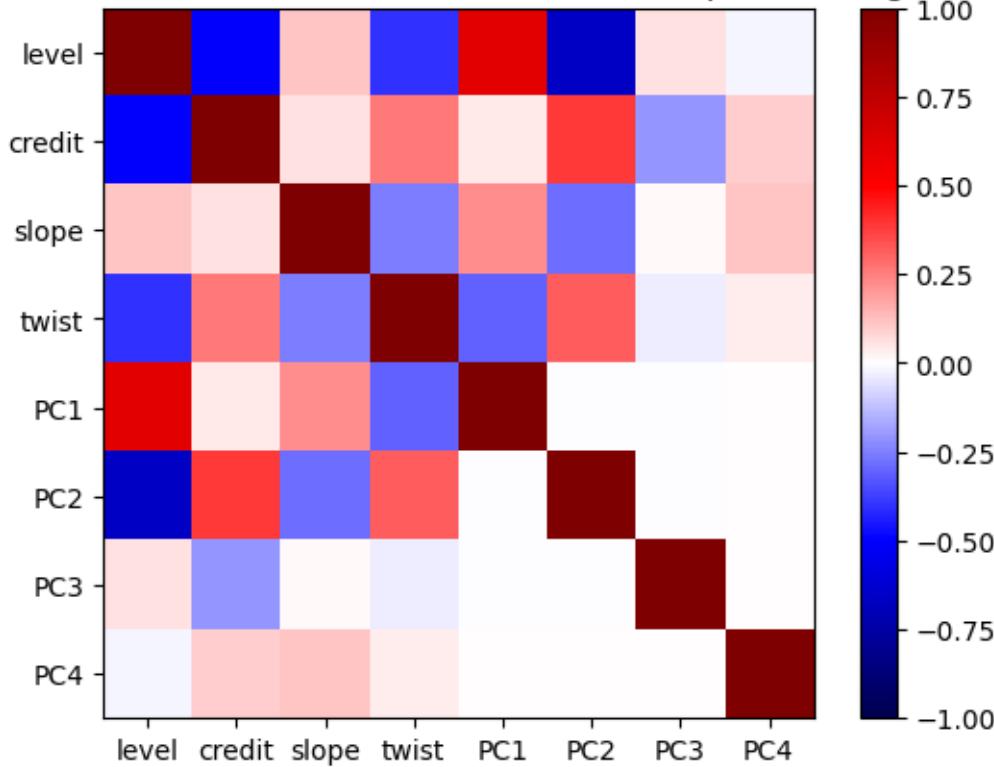
(continues on next page)

(continued from previous page)

```
plt.colorbar()
plt.title('Correlation of bond factors and interest rate spread changes')
```

```
Text(0.5, 1.0, 'Correlation of bond factors and interest rate spread changes')
```

Correlation of bond factors and interest rate spread changes



```
# Show regression fits
for pc in range(K):
    print(smf.ols(f"PC{pc+1} ~ credit + level + slope + twist", data=data) \
        .fit(cov_type='HAC', cov_kwds={'maxlags': 63}) \
        .summary())
```

```
OLS Regression Results
=====
Dep. Variable:                  PC1    R-squared:                 0.547
Model:                          OLS    Adj. R-squared:            0.546
Method: Least Squares          F-statistic:              449.8
Date: Mon, 03 Mar 2025          Prob (F-statistic):        0.00
Time: 17:15:31                  Log-Likelihood:          -16545.
No. Observations:              6533   AIC:                     3.310e+04
Df Residuals:                  6528   BIC:                     3.313e+04
Df Model:                      4
Covariance Type:                HAC
=====
            coef      std err      z      P>|z|      [ 0.025      0.975 ]
=====

```

(continues on next page)

(continued from previous page)

Intercept	0.0260	0.089	0.293	0.770	-0.148	0.200
credit	67.2252	10.644	6.316	0.000	46.363	88.087
level	67.5805	3.099	21.810	0.000	61.507	73.654
slope	9.3162	2.473	3.767	0.000	4.469	14.164
twist	-13.9266	2.182	-6.384	0.000	-18.202	-9.651
<hr/>						
Omnibus:	4287.230	Durbin-Watson:	1.169			
Prob(Omnibus):	0.000	Jarque-Bera (JB):	232868.764			
Skew:	2.485	Prob(JB):	0.00			
Kurtosis:	31.823	Cond. No.	40.8			
<hr/>						

Notes:

[1] Standard Errors are heteroscedasticity and autocorrelation robust (HAC) using \sim 63 lags and without small sample correction

OLS Regression Results

Dep. Variable:	PC2	R-squared:	0.486			
Model:	OLS	Adj. R-squared:	0.486			
Method:	Least Squares	F-statistic:	276.0			
Date:	Mon, 03 Mar 2025	Prob (F-statistic):	1.62e-219			
Time:	17:15:31	Log-Likelihood:	-13117.			
No. Observations:	6533	AIC:	2.624e+04			
Df Residuals:	6528	BIC:	2.628e+04			
Df Model:	4					
Covariance Type:	HAC					
<hr/>						

	coef	std err	z	P> z	[0.025	0.975]
Intercept	0.0026	0.048	0.055	0.956	-0.092	0.097
credit	8.5003	3.413	2.491	0.013	1.811	15.190
level	-27.6696	1.648	-16.794	0.000	-30.899	-24.440
slope	-14.4220	1.193	-12.090	0.000	-16.760	-12.084
twist	0.0005	1.291	0.000	1.000	-2.530	2.531
<hr/>						
Omnibus:	4199.401	Durbin-Watson:	1.263			
Prob(Omnibus):	0.000	Jarque-Bera (JB):	212780.844			
Skew:	2.428	Prob(JB):	0.00			
Kurtosis:	30.534	Cond. No.	40.8			
<hr/>						

Notes:

[1] Standard Errors are heteroscedasticity and autocorrelation robust (HAC) using \sim 63 lags and without small sample correction

OLS Regression Results

Dep. Variable:	PC3	R-squared:	0.048			
Model:	OLS	Adj. R-squared:	0.048			
Method:	Least Squares	F-statistic:	9.825			
Date:	Mon, 03 Mar 2025	Prob (F-statistic):	6.37e-08			
Time:	17:15:32	Log-Likelihood:	-12252.			
No. Observations:	6533	AIC:	2.451e+04			
Df Residuals:	6528	BIC:	2.455e+04			
Df Model:	4					
Covariance Type:	HAC					
<hr/>						

(continues on next page)

(continued from previous page)

	coef	std err	z	P> z	[0.025	0.975]
<hr/>						
Intercept	-0.0047	0.025	-0.189	0.850	-0.053	0.044
credit	-12.9602	2.326	-5.571	0.000	-17.519	-8.401
level	-2.1745	1.302	-1.671	0.095	-4.726	0.377
slope	1.7327	1.218	1.422	0.155	-0.655	4.120
twist	0.4779	1.097	0.436	0.663	-1.671	2.627
<hr/>						
Omnibus:	3598.010	Durbin-Watson:			1.563	
Prob(Omnibus):	0.000	Jarque-Bera (JB):			548862.079	
Skew:	1.601	Prob(JB):			0.00	
Kurtosis:	47.789	Cond. No.			40.8	
<hr/>						

Notes:

[1] Standard Errors are heteroscedasticity and autocorrelation robust (HAC) using
→63 lags and without small sample correction

OLS Regression Results

Dep. Variable:	PC4	R-squared:	0.024
Model:	OLS	Adj. R-squared:	0.024
Method:	Least Squares	F-statistic:	13.23
Date:	Mon, 03 Mar 2025	Prob (F-statistic):	9.77e-11
Time:	17:15:32	Log-Likelihood:	-8402.4
No. Observations:	6533	AIC:	1.681e+04
Df Residuals:	6528	BIC:	1.685e+04
Df Model:	4		
Covariance Type:	HAC		
<hr/>			

	coef	std err	z	P> z	[0.025	0.975]
<hr/>						
Intercept	-0.0016	0.013	-0.125	0.901	-0.027	0.024
credit	2.5655	1.691	1.517	0.129	-0.748	5.879
level	0.5677	0.753	0.753	0.451	-0.909	2.044
slope	2.7974	0.428	6.532	0.000	1.958	3.637
twist	1.6629	0.452	3.679	0.000	0.777	2.549
<hr/>						
Omnibus:	2769.335	Durbin-Watson:			1.940	
Prob(Omnibus):	0.000	Jarque-Bera (JB):			176343.662	
Skew:	1.201	Prob(JB):			0.00	
Kurtosis:	28.339	Cond. No.			40.8	
<hr/>						

Notes:

[1] Standard Errors are heteroscedasticity and autocorrelation robust (HAC) using
→63 lags and without small sample correction

References:

FRM Part I Exam Book Valuation and Risk Models Ch12-13

OPTIONS PRICING

Derivatives are financial weapons of mass destruction - Warren Buffett

We explore the basics of options, common strategies, and the foundational pricing models used in the financial markets. The price of an option is influenced by various factors, including the underlying asset's price, the time remaining until expiration, the volatility of the asset, interest rates, and the strike price. The valuation of options can be approached through techniques like binomial tree pricing and Monte Carlo simulations, or models like the Black-Scholes-Merton model.

```
# By: Terence Lim, 2020-2025 (terence-lim.github.io)
from pandas import DataFrame, Series
import pandas as pd
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
from tqdm import tqdm
import time
from finds.utils import row_formatted
from finds.readers import Alfred
from secret import credentials
VERBOSE = 0
#%matplotlib qt
```

15.1 Options

An **European** call or put option grants the buyer the right to buy or sell an asset at a specified price on a given expiration date. An **American** call or put option, on the other hand, allows the buyer to exercise the option at any time before or on the expiration date. The specified date is known as the **expiration** (or **maturity**) date, while the price at which the asset can be traded is referred to as the **strike** (or **exercise**) price.

Options can be categorized based on their **moneyness**, meaning the relationship between the option's strike price and the current price of the underlying asset. An option is considered **in-the-money** if it would result in a positive payoff if exercised immediately, **out-of-the-money** if it would lead to a negative payoff, and **at-the-money** if the strike price equals the current market price.

The value of an option depends on several factors, including:

- The price of the underlying asset (S)
- The strike price (K)
- The risk-free rate (r)
- The volatility of the asset's price (σ)

- The time to maturity (T)
- Any dividends to be paid during the life of the option (q)

Put-call parity defines the relationship between the prices of European call and put options with identical strike prices and expiration dates: $S - C = PV(K) - P$.

15.1.1 Option strategies

Various trading strategies involve entering positions in multiple options simultaneously, and sometimes incorporating the underlying asset.

- A **protective put** strategy involves buying a put option while holding the underlying asset. This combined position offers a payoff similar to that of a call option, along with the amount of cash equivalent to the present value of the strike price.
- A **covered call** strategy entails owning an asset and selling a call option on it. Typically, the call option is out-of-the-money, and the strategy generates income from the option premium while sacrificing potential upside gains beyond the strike price.
- A **bull spread** is employed by an investor expecting an increase in the asset's price. The strategy involves buying a European call with a lower strike price (K1) and selling a European call with a higher strike price (K2).
- A **bear spread** involves buying a European put option with a higher strike price (K2) and selling a European put option with a lower strike price (K1).
- A **box spread** is a portfolio consisting of a bull spread (using call options) and a bear spread (using put options). Both spreads use the same strike prices and expiration dates.
- A **butterfly spread** involves a combination of three options, created using either calls or puts. A common version involves one long call with a lower strike price (K1), one long call with a higher strike price (K3), and two short calls at a strike price (K2), where $K2 = (K1 + K3)/2$.
- A **calendar spread** strategy consists of buying a long call option with an expiration date at time T^* and selling a short call option with an earlier expiration date, both at the same strike price (K).
- A **straddle** is created by holding both a long call and a long put with the same strike price and expiration date.
- A **strangle** is similar to a straddle but reduces the cost by selecting a call option with a higher strike price than the put option.

```
# define call and put payoffs at maturity
def call_payoff(K):
    return lambda s: s - K if s > K else 0

def put_payoff(K):
    return lambda s: K - s if s < K else 0
```

```
# helpers to plot final payoff of option strategy
def plot_payoff(payoff, S=np.linspace(70, 130, 60), ax=None, label=''):
    """helper to plot final payoff over range of stock price S"""
    ax = ax or plt.gca()
    df = DataFrame([payoff(s) for s in S], index=S)
    zeros = (df == 0).all(axis=0)
    df = df.drop(columns=df.columns[zeros])
    df = df + np.sign(df.sum(axis=0))*0.4    # jigger so can display separate
    y = df.sum(axis=1).rename(label)
    y.plot(ax=ax, lw=3)
```

(continues on next page)

(continued from previous page)

```

df.plot(marker='.', ls='', ms=4, ax=ax)
ax.legend([label] + list(df.columns))

def _ls(n):
    """helper to label as long or short position"""
    return 'long' if n >=0 else 'short'

# Define and plot the options strategies
fig, ax = plt.subplots(nrows=4, ncols=2, figsize=(10,12))
ax = ax.flatten()

def options_strategy(K, calls=0, puts=0, stocks=0):
    return lambda s: {f"_{ls(calls)} call {K}": calls*call_payoff(K)(s),
                      f"_{ls(puts)} put {K}": puts*put_payoff(K)(s),
                      "stock": stocks*s}

plot_payoff(options_strategy(K=100, puts=1, stocks=1), ax=ax[0], label='Protective Put
    ↪')

plot_payoff(options_strategy(K=100, calls=-1, stocks=1), ax=ax[1], label='Covered Call
    ↪')

def bull_spread(K1, K2):
    assert K2 > K1, "K2 must be greater than K1"
    return lambda s: {f"long call {K1}": call_payoff(K1)(s),
                      f"short call {K2}": -call_payoff(K2)(s)}

plot_payoff(bull_spread(K1=95, K2=105), ax=ax[2], label='Bull Spread')

def bear_spread(K1, K2):
    assert K2 > K1, "K2 must be greater than K1"
    return lambda s: {f"short put {K1}": -put_payoff(K1)(s),
                      f"long put {K2}": put_payoff(K2)(s)}

plot_payoff(bear_spread(K1=95, K2=105), ax=ax[3], label='Bear Spread')

def box_spread(K1, K2):
    assert K2 > K1, "K2 must be greater than K1"
    return lambda s: {f"short put {K1}": -put_payoff(K1)(s),
                      f"long call {K1}": call_payoff(K1)(s),
                      f"long put {K2}": put_payoff(K2)(s),
                      f"short call {K2}": -call_payoff(K2)(s)}

plot_payoff(box_spread(K1=95, K2=105), ax=ax[4], label='Box Spread')

def butterfly_spread(K1, K3):
    assert K3 > K1, "K3 must be greater than K1"
    K2 = (K1 + K3) / 2
    return lambda s: {f"long call {K1}": call_payoff(K1)(s),
                      f"short calls {K2:g}": -2*call_payoff(K2)(s),
                      f"long call {K3}": call_payoff(K3)(s)}

plot_payoff(butterfly_spread(K1=90, K3=110), ax=ax[5], label='Butterfly Spread')

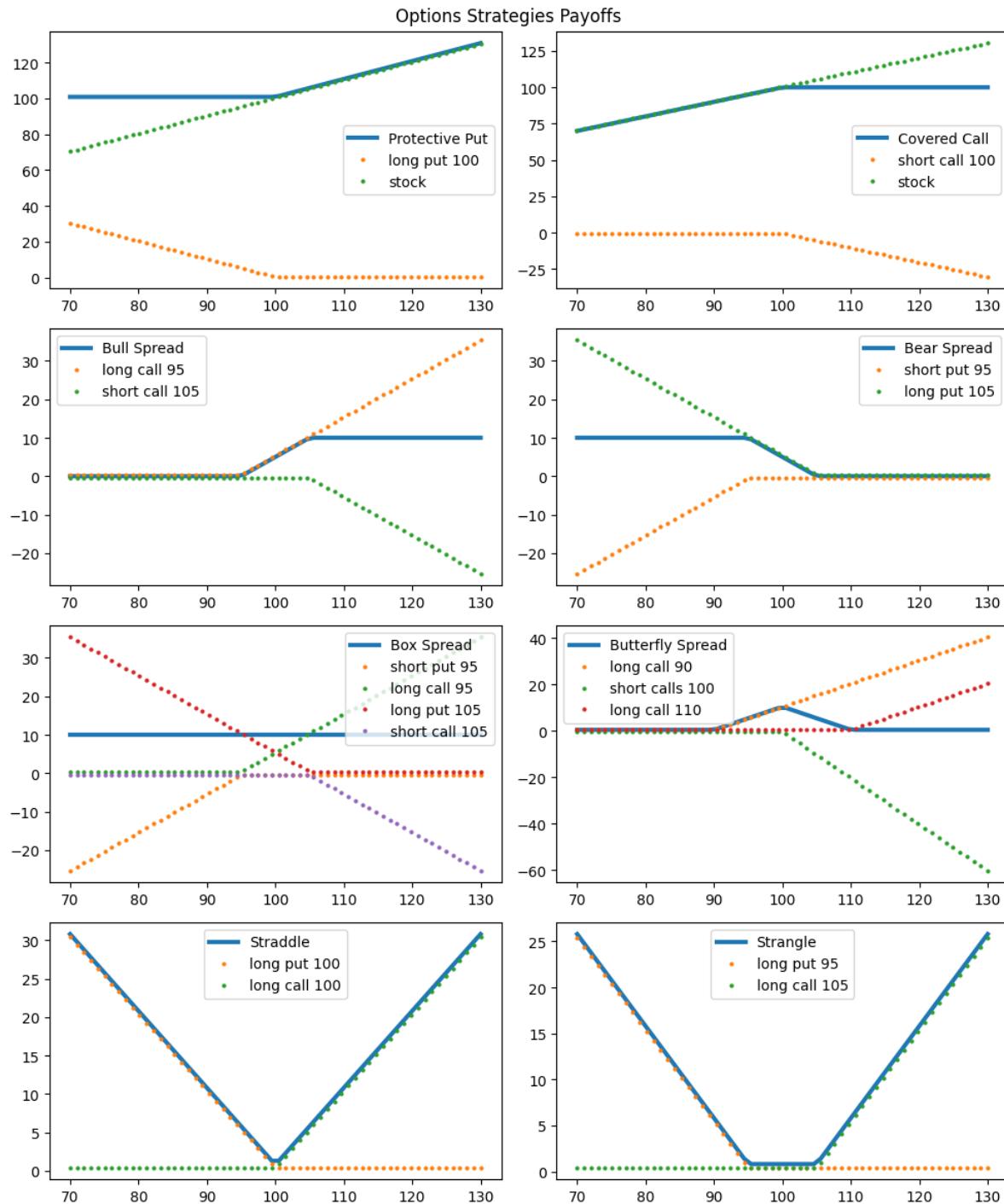
def straddle(K):
    return lambda s: {f"long put {K}": put_payoff(K)(s),
                      f"long call {K}": call_payoff(K)(s)}

```

(continues on next page)

(continued from previous page)

```
f"long call {K}": call_payoff(K)(s) }  
plot_payoff(straddle(K=100), ax=ax[6], label='Straddle')  
  
def strangle(K1, K2):  
    assert K2 > K1, "K2 must be greater than K1"  
    return lambda s: {f"long put {K1}": put_payoff(K1)(s),  
                  f"long call {K2}": call_payoff(K2)(s) }  
plot_payoff(strangle(K1=95, K2=105), ax=ax[7], label='Strangle')  
  
plt.suptitle('Options Strategies Payoffs')  
plt.tight_layout()
```



15.1.2 Exotic options

In addition to standard European and American options (which are termed plain vanilla options), there are **exotic options** (or simply **exotics**) which have more complex structures and non-standard features.

- A **Bermudan** option is one where the exercise of the option is restricted to certain predetermined dates.
- A **forward start** option is an option that starts at a future date and is usually at-the-money at the time it begins.
- A **gap** option is a European option where the price triggering a payoff is different from the price used to calculate the payoff.
- A **cliquet** option is a series of forward start options, each with its own rules for determining strike prices.
- **Binary** options (also called **digital** options) pay a fixed amount or asset if the option's price exceeds (or falls below) the strike price, otherwise, they pay nothing.
- **Asian** options provide payoffs based on the arithmetic average of the asset's price during the option's life.
- A **lookback** option's payoff depends on the maximum or minimum price reached by the asset during its lifetime. A floating lookback option gives a payoff based on the difference between the final asset price and the minimum (or maximum) price reached, while a fixed lookback option bases the payoff on the difference between the maximum (or minimum) price and the strike price.
- **Barrier** options have payoffs that depend on whether the price of the asset reaches a specific level, with various types like down-and-out, down-and-in, up-and-out, and up-and-in.
- A **compound** option is an option on another option, which results in two strike prices and two expiration dates.
- An **asset-exchange** option allows the holder to exchange one asset for another.
- A **volatility swap** is a forward contract based on the realized volatility of an asset during a specified period. Traders exchange the realized volatility at the end of the period for a pre-specified volatility rate.
- The exercise of a **Bermudan** option is restricted to certain dates.
- A **forward start** option is an option that will begin at a future time. It is usually stated that the option will be at-the-money at the time it starts.
- A **gap** option is a European call or put option where the price triggering a payoff is different from the price used in calculating the payoff.
- A **cliquet** option is a series of forward start options with certain rules for determining the strike prices.
- **Binary** call (put) options may a fixed amount of cash or an asset when its price is above (below) the strike price, or nothing otherwise. Cash-or-nothing options are sometimes referred to as **digital** options.
- **Asian** options provide a payoff dependent on an arithmetic average of the underlying asset price during the life of the option.
- The payoff from a **lookback** option depends on the maximum or minimum asset price reached during the life of the option. A floating lookback call (put) gives a payoff equal difference between the final asset price and minimum (maximum) price. The payoff of a fixed lookback call (put) is based on the difference between the maximum (minimum) price and the strike price.
- **Barrier** options come into existence or ceases to exist depending on whether the asset price reaches a particular barrier. There are down-and-out, down-and-in, up-and-out and up-and-in variants.
- A **compound** option is an option on another option. Thus, there are two strike prices and two maturity dates.
- In an **asset-exchange** option, the holder has the right to exchange one asset for another.
- A **volatility swap** is a forward contract on the realized volatility of an asset during a certain period. A trader agrees to exchange a pre-specified volatility for the realized volatility at the end of the period.

15.2 Binomial option pricing

The **binomial option pricing model**, proposed by Cox, Ross, and Rubinstein (1979), is widely used to value American-style options and other derivatives.

15.2.1 No-arbitrage

The no-arbitrage principle assumes there are no opportunities for riskless profit in the market. The **law of one price** stipulates that if two portfolios produce the same cash flows at the same times, they should have the same price. Binomial trees use this no-arbitrage principle to model option pricing.

For example, assuming a non-dividend-paying stock with price S , the stock can either increase to S_u or decrease to S_d within time T . The portfolio is structured with:

- A short position in the derivative, and
- A position in the stock which we set equal to $\Delta = \frac{f_u - f_d}{S_u - S_d}$

The value of the portfolio at time T is

- $S_u\Delta - f_u$ if the stock price increases, and
- $S_d\Delta - f_d$ if the stock price decreases.

The value of the portfolio today is $S\Delta - f$, where f is the value of the derivative today. Suppose r is the risk-free rate for maturity T . For no arbitrage, we must have

$$S\Delta - f = \frac{f_u d - f_d u}{u - d} e^{-rT}$$

Substituting for Δ , gives:

$$f = e^{-rT}[p f_u + (1 - p) f_d]$$

$$\text{where } p = \frac{e^{rT} - d}{u - d}$$

15.2.2 Risk neutral pricing

Suppose we choose to interpret the variable p as the probability of an upward movement (with $1 - p$ being the probability of a downward movement), then the expected stock price grows at the risk-free rate. It also means that p is the probability of an upward movement in a risk-neutral world.

The **risk-neutral** valuation approach applies a probability p for upward movements and values the option by its expected payoff, discounted at the risk-free rate. Risk-neutral pricing assumes that all assets earn the risk-free rate of return in the market. Put another way, a risk-neutral world is one where all tradable assets have an expected return equal to the risk-free interest rate. The probabilities of different outcomes in a risk-neutral world are therefore based on this assumption, and a risk-neutral investor has no preference between assets with different risks. This methodology simplifies the valuation process by treating all market participants as indifferent to risk.

It should be emphasized that the risk-neutral valuation is nothing more than an artificial way of valuing derivatives. We are not assuming that the world is actually risk-neutral. We are instead arguing that the price of a derivative is the same in the real world as it would be in the risk-neutral world

15.2.3 Binomial tree

In practice, the binomial model is extended to multi-step trees, where each step reflects changes in the asset's price over time. Parameters like Δt (time between steps), and u, d (upward and downward movements) are chosen based on the asset's volatility:

- the length of a tree step as Δt
- the parameters u and d should be chosen to reflect the volatility of the stock price. If we denote the volatility per year by σ , then appropriate values for the parameters are
 - $u = e^{\sigma\sqrt{\Delta t}}$
 - $d = e^{-\sigma\sqrt{\Delta t}}$

where Δt is measured in years.

- hence $f = e^{-r\Delta t}[pf_u + (1-p)f_d]$ where $p = \frac{e^{(r-q)\Delta t} - d}{u - d}$
- the delta, or position taken in the stock to hedge a short position in the derivative, is $\Delta = \frac{f_u - f_d}{S_u - S_d}$

Dividends: For assets paying dividends, the probability p is adjusted to account for the dividend yield q :

$$p = \frac{e^{(r-q)\Delta t} - d}{u - d}$$

Currency Options: A currency can be considered as an asset providing a yield at the foreign risk-free rate. Therefore, the analysis we presented for a stock paying a continuous dividend yield applies, with q equal to the foreign risk-free rate.

Futures: Because it costs nothing to enter into a futures contract, the return on a futures contract in a risk-neutral world must be zero. This means we can treat a futures contract like a stock, paying a continuous dividend yield equal to the risk free rate.

```
def binomial_tree(S, sigma, r, T, steps, payoff=None, q=0, american=False,
                  verbose=True):
    delta_t = T / steps
    u = np.exp(sigma * np.sqrt(delta_t))
    d = np.exp(-sigma * np.sqrt(delta_t))
    p = (np.exp((r - q) * delta_t) - d) / (u - d)
    result = dict(value=None, u=u, d=d, p=p, delta_t=delta_t)
    if payoff is not None:
        label = "STEP {:.5d}".format # to label output of each step
        # initialize price vectors at last step
        prices = DataFrame(0.0, index=np.arange(steps+1),
                           columns=['stock', 'option', 'delta'])
        for downs in range(steps+1):
            s = d**downs * u**(steps-downs) * S # price after number of downs
            prices.loc[downs, 'stock'] = s
            prices.loc[downs, 'option'] = payoff(s) # option value at expiry
        print(row_formatted(prices.T.rename_axis(columns=label(steps)),
                            default=".4f"))
        # roll back one time step at a time
        for step in range(steps - 1, -1, -1):
            # update all scenarios in this time step
            for downs in range(step+1):
                # stock price after this number of downs
                result['value'] = result['u'] * result['value'] * (1 + r * delta_t) - result['d'] * result['value'] * (1 - q * delta_t)
                result['u'] = u
                result['d'] = d
                result['p'] = p
                result['delta_t'] = delta_t
            result['value'] = payoff(result['value'])
            result['value'] = result['value'] * (1 + r * delta_t) - result['value'] * (1 - q * delta_t)
            result['u'] = u
            result['d'] = d
            result['p'] = p
            result['delta_t'] = delta_t
    return result
```

(continues on next page)

(continued from previous page)

```

s = d**downs * u** (step - downs) * S
prices.loc[downs, 'stock'] = s

# value of option is max of roll back or exercise (if American)
exercise = payoff(s) if american else 0 # if exercise American
f_u = prices.loc[downs, 'option'] # option value after up
f_d = prices.loc[downs + 1, 'option'] # option value after down
f = np.exp(-r * delta_t) * (p * f_u + (1 - p) * f_d)
prices.loc[downs, 'option'] = max(f, exercise)
prices.loc[downs, 'delta'] = (f_u - f_d) / (S*u - S*d)

# Display this time step
prices = prices.iloc[:-1]
print()
print(row_formatted(prices.T.rename_axis(columns=label(step)),
                     default=".4f"))
result['value'] = prices.loc[0, 'option']
return DataFrame(result, index=[steps]).rename_axis(columns='STEPS').round(4)

```

European call option in 2 steps

```

# Figure 14.3 of FRM Part I Exam Book "Valuation and Risk Models"
binomial_tree(S=29, sigma=0.25, r=0.03, T=1, steps=2,
               payoff=call_payoff(K=30))

```

STEP 2	0	1	2
stock	41.2995	29.0000	20.3635
option	11.2995	0.0000	0.0000
delta	0.0000	0.0000	0.0000

STEP 1	0	1
stock	34.6076	24.3010
option	5.5483	0.0000
delta	1.0963	0.0000

STEP 0	0
stock	29.0000
option	2.7243
delta	0.5383

STEPS	value	u	d	p	delta_t
2	2.7243	1.1934	0.838	0.4984	0.5

European put option in 2 steps

```

# Figure 14.4 of FRM Part I Exam Book "Valuation and Risk Models"
binomial_tree(S=29, sigma=0.25, r=0.03, T=1, steps=2,
               payoff=put_payoff(K=30))

```

STEP 2	0	1	2
stock	41.2995	29.0000	20.3635
option	0.0000	1.0000	9.6365
delta	0.0000	0.0000	0.0000

(continues on next page)

(continued from previous page)

	0	1
stock	34.6076	24.3010
option	0.4941	5.2523
delta	-0.0970	-0.8380

	0
stock	29.0000
option	2.8377
delta	-0.4617

STEPS	value	u	d	p	delta_t
2	2.8377	1.1934	0.838	0.4984	0.5

American put option in 4 steps

```
# Figure 14.5 of FRM Part I Exam Book "Valuation and Risk Models"
binomial_tree(S=29, sigma=0.25, r=0.03, T=1, steps=2,
              payoff=put_payoff(K=30), american=True)
```

	0	1	2
stock	41.2995	29.0000	20.3635
option	0.0000	1.0000	9.6365
delta	0.0000	0.0000	0.0000

	0	1
stock	34.6076	24.3010
option	0.4941	5.6990
delta	-0.0970	-0.8380

	0
stock	29.0000
option	3.0584
delta	-0.5050

STEPS	value	u	d	p	delta_t
2	3.0584	1.1934	0.838	0.4984	0.5

American put option in 4 steps

```
# Figure 14.6 of FRM Part I Exam Book "Valuation and Risk Models"
binomial_tree(S=29, sigma=0.25, r=0.03, T=1, steps=4,
              payoff=put_payoff(K=30), american=True)
```

	0	1	2	3	4
stock	47.8129	37.2367	29.0000	22.5852	17.5894
option	0.0000	0.0000	1.0000	7.4148	12.4106
delta	0.0000	0.0000	0.0000	0.0000	0.0000

	0	1	2	3
stock	42.1948	32.8613	25.5924	19.9314
option	0.0000	0.4974	4.4076	10.0686
delta	0.0000	-0.1376	-0.8825	-0.6873

(continues on next page)

(continued from previous page)

	0	1	2
stock	37.2367	29.0000	22.5852
option	0.2474	2.4387	7.4148
delta	-0.0684	-0.5379	-0.7788

	0	1
stock	32.8613	25.5924
option	1.3356	4.8958
delta	-0.3015	-0.6846

	0
stock	29.0000
option	3.0966
delta	-0.4898

STEPS	value	u	d	p	delta_t
4	3.0966	1.1331	0.8825	0.4988	0.25

European call on index with dividend yield

```
# Figure 14.7 of FRM Part I Exam Book "Valuation and Risk Models"
binomial_tree(S=2500, sigma=0.15, r=0.03, q=0.02, T=0.5, steps=3,
              payoff=call_payoff(K=2500))
```

	0	1	2	3
stock	3004.1734	2657.8778	2351.5002	2080.4391
option	504.1734	157.8778	0.0000	0.0000
delta	0.0000	0.0000	0.0000	0.0000

	0	1	2
stock	2825.7257	2500.0000	2211.8212
option	328.7911	78.2792	0.0000
delta	1.1303	0.5153	0.0000

	0	1
stock	2657.8778	2351.5002
option	202.0979	38.8125
delta	0.8177	0.2555

	0
stock	2500.0000
option	119.5793
delta	0.5330

STEPS	value	u	d	p	delta_t
3	119.5793	1.0632	0.9406	0.4983	0.1667

American call option on foreign currency

```
# Figure 14.8 of FRM Part I Exam Book "Valuation and Risk Models"
binomial_tree(S=0.78, sigma=0.12, r=0.02, q=0.06, T=1, steps=4,
              payoff=call_payoff(K=0.8))
```

STEP 4	0	1	2	3	4
stock	0.9916	0.8794	0.7800	0.6918	0.6136
option	0.1916	0.0794	0.0000	0.0000	0.0000
delta	0.0000	0.0000	0.0000	0.0000	0.0000

STEP 3	0	1	2	3
stock	0.9338	0.8282	0.7346	0.6515
option	0.1239	0.0318	0.0000	0.0000
delta	1.1972	0.8483	0.0000	0.0000

STEP 2	0	1	2
stock	0.8794	0.7800	0.6918
option	0.0685	0.0127	0.0000
delta	0.9837	0.3394	0.0000

STEP 1	0	1
stock	0.8282	0.7346
option	0.0350	0.0051
delta	0.5955	0.1358

STEP 0	0
stock	0.7800
option	0.0170
delta	0.3191

STEPS	value	u	d	p	delta_t
4	0.017	1.0618	0.9418	0.4021	0.25

American put option on futures contract

```
# Figure 14.9 of FRM Part I Exam Book "Valuation and Risk Models"
binomial_tree(S=38, sigma=0.2, r=0.04, q=0.04, T=9/12, steps=3,
payoff=put_payoff(K=40), american=True)
```

STEP 3	0	1	2	3
stock	51.2946	41.9965	34.3838	28.1511
option	0.0000	0.0000	5.6162	11.8489
delta	0.0000	0.0000	0.0000	0.0000

STEP 2	0	1	2
stock	46.4133	38.0000	31.1118
option	0.0000	2.9190	8.8882
delta	0.0000	-0.7377	-0.8187

STEP 1	0	1
stock	41.9965	34.3838
option	1.5172	5.9925
delta	-0.3834	-0.7841

STEP 0	0
stock	38.0000
option	3.8282
delta	-0.5879

STEPS	value	u	d	p	delta_t
3	3.8282	1.1052	0.9048	0.475	0.25

15.3 Black-Scholes-Merton model

The **Black-Scholes-Merton model**, introduced in two papers in 1973, revolutionized options pricing and remains one of the most widely used models for pricing European options. Based on assumptions like lognormal stock price distributions, continuous trading, and no-arbitrage conditions, the model provides a formula for calculating the price of call and put options. In one of the papers, Black and Scholes used the capital asset pricing model (CAPM) to derive the relationship between the return from a stock and the return from an option on the stock. In the other, Merton used no-arbitrage arguments similar to those of the binomial tree approach. Both derived that the price evolution of derivatives satisfies the same partial differential equation:

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS \frac{\partial V}{\partial S} - rV = 0$$

To value an option, we simply apply boundary conditions. For a European call option with time to maturity T and strike price K , the boundary condition is that the value of the option is $\max(S - K, 0)$ at time T . For a European put, this boundary condition is $\max(K - S, 0)$. Other derivatives give rise to other boundary conditions.

The solutions for the prices of a European call and put are the Black-Scholes-Merton formulas:

$$C = S e^{-qT} \Phi(d_1) - K e^{-rT} \Phi(d_2)$$

$$p = K e^{-rT} \Phi(-d_2) - S e^{-qT} \Phi(-d_1)$$

where:

- $d_1 = \frac{\ln(S_0/K) + (r - q + \sigma^2/2)T}{\sigma\sqrt{T}}$
- $d_2 = \frac{\ln(S_0/K) + (r - q - \sigma^2/2)T}{\sigma\sqrt{T}} = d_1 - \sigma\sqrt{T}$
- S is the current stock price,
- K is the strike price,
- T is the time to maturity in years,
- r is the risk-free rate per year (continuously compounded),
- q is the dividend yield (or foreign risk-free rate for currency options)
- σ is an estimated volatility per year over the next T years,
- Φ is the cumulative normal distribution function

Suppose discrete dividends are expected to be paid with ex-dates during the life of the option. The option can be valued by replacing stock price S with $S - PV(D)$, the present value of those dividends.

Since American call options on a non-dividend paying stock should never be exercised early, the pricing formula for American call options on non-dividend paying stocks as well as for European call options are the same. But it may be optimal to exercise early when there are discrete dividends, but only immediately before an ex-dividend date. American put options on stocks and all American options on stock indices, currencies, and futures should not be valued as European options. Binomial trees can be used in these cases.

Black-Scholes-Merton option pricing formulas

```
def _d1(S, K, sigma, r, T, q):
    """helper to compute d1 term in Black-Scholes-Merton formula"""
    return (np.log(S/K) + (r - q + sigma**2/2) * T) / (sigma * np.sqrt(T))
```

```
def call(S, K, sigma, r, T, q=0.):
    """Black-Scholes-Merton call option value"""
    d1 = _d1(S=S, K=K, sigma=sigma, r=r, T=T, q=q)
    d2 = d1 - sigma * np.sqrt(T)
    return S*np.exp(-q*T)*stats.norm.cdf(d1) - K*np.exp(-r*T)*stats.norm.cdf(d2)
```

```
def put(S, K, sigma, r, T, q=0.):
    """Black-Scholes-Merton put option value"""
    d1 = _d1(S=S, K=K, sigma=sigma, r=r, T=T, q=q)
    d2 = d1 - sigma * np.sqrt(T)
    return K*np.exp(-r*T)*stats.norm.cdf(-d2) - S*np.exp(-q*T)*stats.norm.cdf(-d1)
```

```
print(call(S=56, K=60, r=0.05, sigma=0.3, T=18/12)) # 8.3069
print(put(S=56, K=60, r=0.05, sigma=0.3, T=18/12)) # 7.9715
```

```
8.306909593824336
7.971518773537515
```

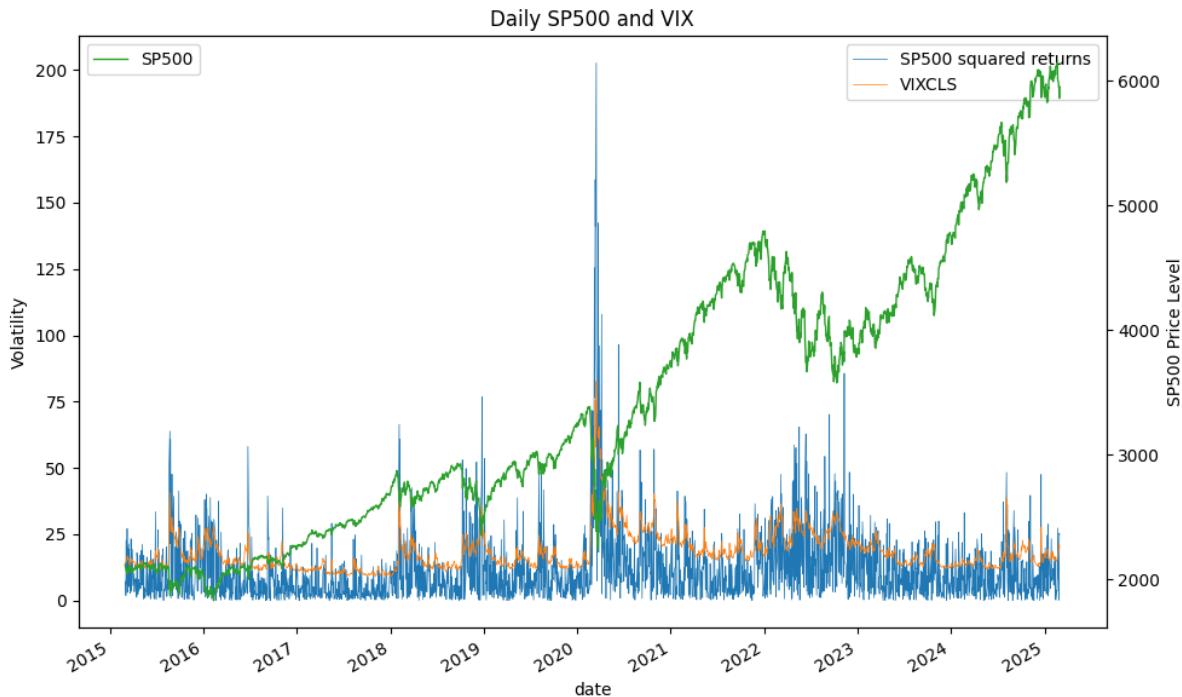
15.3.1 Implied volatility

Implied volatility is the volatility figure derived from an option's market price and used to infer expectations about future price movements.

The Chicago Board Options Exchange has developed indices that track volatilities. The most popular of these is the SPX VIX index, which tracks the volatilities of 30-day options on the S&P 500. Traders monitor implied volatilities carefully and often use them to communicate prices.

```
alf = Alfred(api_key=credentials['fred']['api_key'], verbose=VERBOSE, convert_date=0)
vix = alf('VIXCLS')
sp500_sq = alf('SP500', log=1, diff=1).rename('SP500 squared returns')
sp500 = alf('SP500')
df = pd.concat([100*np.sqrt(252*sp500_sq**2), vix], axis=1, join='inner').dropna()

fig, ax = plt.subplots(figsize=(10, 6))
df.plot(ax=ax, lw=.5)
ax.set_ylabel('Volatility')
bx = ax.twinx()
sp500.plot(ax=bx, lw=1, color="C2")
bx.set_ylabel('SP500 Price Level')
plt.title('Daily SP500 and VIX')
plt.legend()
plt.tight_layout()
```



15.3.2 Volatility smile

If the assumptions underlying the Black-Scholes-Merton model held exactly, all options on an asset would have the same implied volatility at all times. In practice, implied volatility varies with the strike price and time to maturity. The **volatility smile** refers to this observed pattern, where options with extreme strike prices (both high and low) tend to have higher implied volatilities. Because of put-call parity, the implied volatility of a European call option is the same as that of a European put option when they have the same strike price and time to maturity.

In equity options, the volatility smile generally slopes downward. This means that out-of-the-money puts and in-the-money calls tend to have higher implied volatilities, while out-of-the-money calls and in-the-money puts have lower implied volatilities. This phenomenon is often referred to as a volatility skew. There is a negative correlation between equity prices and volatility, which means that as stock prices fall, volatility increases, and as stock prices rise, volatility decreases.

For foreign currency options, the volatility smile takes a U-shape. Both out-of-the-money and in-the-money options tend to have higher implied volatilities compared to at-the-money options. The volatility of exchange rates is not constant and can be subject to sudden jumps, often driven by central bank actions. These nonconstant volatilities and jumps make extreme price movements more likely.

Traders also often use a volatility term structure, where the implied volatility of an option depends on its time to maturity. When volatility smiles and term structures are combined, they form a **volatility surface**. This surface shows implied volatility as a function of both the strike price and the time to maturity. When quoting option prices, traders interpolate between known implied volatilities to estimate the implied volatility for the option in question. This estimated volatility is then plugged into the Black-Scholes-Merton model to calculate the option price. This approach helps address the fact that the market does not always price options in line with the assumptions of the Black-Scholes-Merton model.

15.3.3 The “Greeks”

The Greek letters, or **Greeks** as they are often called, are metrics used to measure the sensitivity of option prices to different factors such as changes in the price of the underlying asset, volatility, time decay, and interest rates.

- **Delta** measures the sensitivity to changes in the price of the underlying asset:

$$\Delta_c = e^{-qt} \Phi(d_1)$$

$$\Delta_p = e^{-qt} [\Phi(d_1) - 1]$$

- **Gamma** measures the sensitivity of a portfolio's delta to changes in the price of the underlying asset:

$$\Gamma = \frac{e^{-qt}}{S\sigma\sqrt{t}} \phi(d_1)$$

- **Theta** measures the sensitivity to time to expiration

$$\Theta_c = -\frac{S\sigma e^{-qt}}{2\sqrt{t}} \phi(d_1) - rKe^{-rt} \Phi(d_2) + qSe^{-qt} \Phi(d_1)$$

$$\Theta_p = -\frac{S\sigma e^{-qt}}{2\sqrt{t}} \phi(d_1) + rKe^{-rt} \Phi(-d_2) - qSe^{-qt} \Phi(-d_1)$$

- **Vega** measures the sensitivity to the implied volatility

$$V = Se^{-qt} \sqrt{t} \phi(d_1)$$

- **Rho** measures the sensitivity to changes in the level of interest rate

$$\rho_c = Kte^{-rt} \Phi(d_2)$$

$$\rho_p = Kte^{-rt} \Phi(-d_2)$$

Any of the Greek letters for a portfolio of derivatives dependent on the same asset can be calculated as the weighted sum of the Greek letters for each portfolio component.

The Black-Scholes-Merton analysis can be used to show that $\$ \Theta + (r - q)S\Delta + \frac{1}{2}\sigma^2 S^2 \Gamma = (r - q)C\$$

```
# Helper to plot a greek over range of stock price S"""
def plot_greek(K, sigma, r, T, greek, S=np.linspace(50, 200, 100),
               ax=None, label='', q=0., c="C0"):

    """helper to plot a greek over range of stock price S"""
    y = [greek(S=s, K=K, sigma=sigma, r=r, T=T, q=q) for s in S]
    ax = ax or plt.gca()
    ax.plot(S, y, color=c)
    #ax.set_xlabel('Stock Price')
    ax.legend([label])
```

Define and plot options sensitivities

```
# Plot options sensitivities
fig, ax = plt.subplots(nrows=4, ncols=2, figsize=(10, 12))
ax = ax.flatten()

# call option parameters
opt = dict(K=105, r=0.04, sigma=0.25, T=1)

def delta_call(S, K, sigma, r, T, q=0.):
    """Black-Scholes-Merton call option delta"""
    d1 = _d1(S=S, K=K, sigma=sigma, r=r, T=T, q=q)
```

(continues on next page)

(continued from previous page)

```

    return np.exp(-q*T) * stats.norm.cdf(d1)
def delta_put(S, K, sigma, r, T, q=0.):
    """Black-Scholes-Merton put option delta"""
    d1 = _d1(S=S, K=K, sigma=sigma, r=r, T=T, q=q)
    return -np.exp(-q*T) * stats.norm.cdf(-d1)
print(delta_call(S=100, **opt)) # 0.5358

plot_greek(greek=delta_call, label='Call Delta', ax=ax[0], c="C0", **opt)
plot_greek(greek=delta_put, label='Put Delta', ax=ax[1], c="C1", **opt)

def vega(S, K, sigma, r, T, q=0.):
    """Black-Scholes-Merton call or put option delta"""
    d1 = _d1(S=S, K=K, sigma=sigma, r=r, T=T, q=q)
    return S * np.exp(-q*T) * np.sqrt(T) * stats.norm.pdf(d1)
print(vega(S=100, **opt)) # 39.73

plot_greek(greek=vega, label='Vega', ax=ax[2], c="C2", **opt)

def gamma(S, K, sigma, r, T, q=0.):
    """Black-Scholes-Merton call or put option gamma"""
    d1 = _d1(S=S, K=K, sigma=sigma, r=r, T=T, q=q)
    return np.exp(-q*T) * stats.norm.pdf(d1) / (S * sigma * np.sqrt(T))
print(gamma(S=100, **opt)) # 0.0159

plot_greek(greek=gamma, label='Gamma', ax=ax[3], c="C3", **opt)

def theta_call(S, K, sigma, r, T, q=0.):
    """Black-Scholes-Merton call option theta"""
    d1 = _d1(S=S, K=K, sigma=sigma, r=r, T=T, q=q)
    d2 = d1 - sigma * np.sqrt(T)
    return ((-S * np.exp(-q*T) * stats.norm.pdf(d1) * sigma / (2 * np.sqrt(T)))
            - (r * K * np.exp(-r*T) * stats.norm.cdf(d2)) +
            (q * S * np.exp(-q*T) * stats.norm.cdf(d1)))
def theta_put(S, K, sigma, r, T, q=0.):
    """Black-Scholes-Merton put option theta"""
    d1 = _d1(S=S, K=K, sigma=sigma, r=r, T=T, q=q)
    d2 = d1 - sigma * np.sqrt(T)
    return ((-S * np.exp(-q*T) * stats.norm.pdf(d1) * sigma / (2 * np.sqrt(T)))
            + (r * K * np.exp(-r*T) * stats.norm.cdf(-d2)) -
            (q * S * np.exp(-q*T) * stats.norm.cdf(-d1)))
print(theta_call(S=100, **opt)) # 6.73

plot_greek(greek=theta_call, label='Call Theta', ax=ax[4], c="C4", **opt)
plot_greek(greek=theta_put, label='Put Theta', ax=ax[5], c="C5", **opt)

def rho_call(S, K, sigma, r, T, q=0.):
    """Black-Scholes-Merton call option rho"""
    d1 = _d1(S=S, K=K, sigma=sigma, r=r, T=T, q=q)
    d2 = d1 - sigma * np.sqrt(T)
    return K * T * np.exp(-r * T) * stats.norm.cdf(d2)
def rho_put(S, K, sigma, r, T, q=0.):
    """Black-Scholes-Merton put option rho"""
    d1 = _d1(S=S, K=K, sigma=sigma, r=r, T=T, q=q)
    d2 = d1 - sigma * np.sqrt(T)
    return -K * T * np.exp(-r * T) * stats.norm.cdf(-d2)

```

(continues on next page)

(continued from previous page)

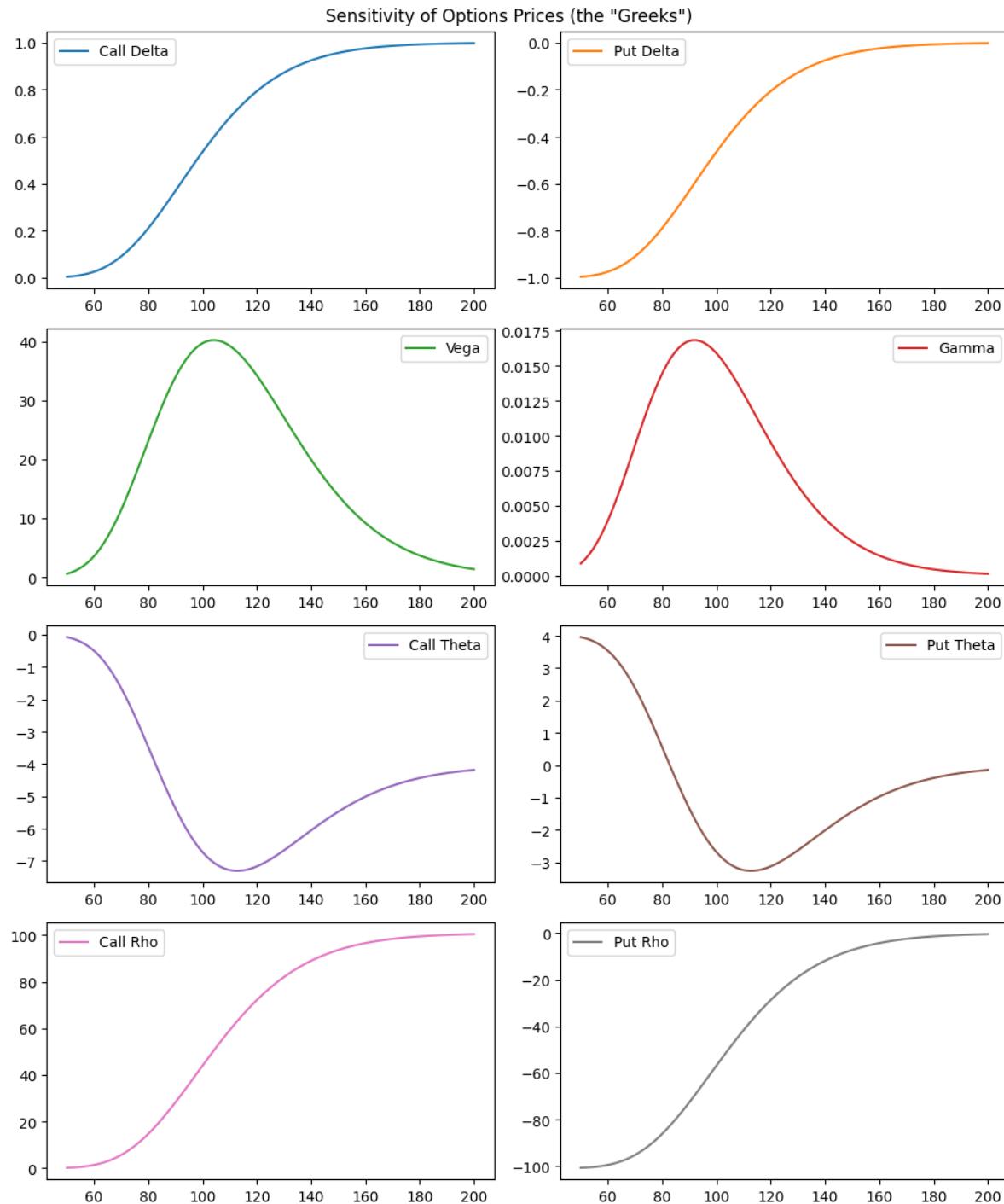
```
print(rho_call(S=100, **opt)) # 44

plot_greek(greek=rho_call, label='Call Rho', ax=ax[6], c="C6", **opt)

plot_greek(greek=rho_put, label='Put Rho', ax=ax[7], c="C7", **opt)

plt.suptitle('Sensitivity of Options Prices (the "Greeks")')
plt.tight_layout()
```

```
0.5357925584332519
39.73355715246575
0.0158934228609863
-6.727614629584682
44.02299963816157
```



```
# Check that: theta + r S delta + 1/2 sigma^2 S^2 gamma = r call
S = 100
print(theta_call(S=S, **opt) + opt['r'] * S * delta_call(100, **opt) +
      0.5 * opt['sigma']**2 * S**2 * gamma(S=S, **opt),
      opt['r'] * call(S=S, **opt))
```

```
0.3822502482065442 0.38225024820654485
```

15.4 Monte Carlo simulation

Simulation is an effective way to estimate expectations that are difficult or impossible to compute analytically. Consider a random variable X that can be simulated (for example, from a normal distribution), and a function g that can be evaluated at realizations of X . To estimate the expected value of $g(X)$, we take multiple simulated draws. Since these draws are independent and identically distributed (iid), the expectation can be approximated as:

$$\hat{E}[g(X)] = \frac{1}{b} \sum_{i=1}^b g(X_i)$$

where b is the number of simulated samples. By the Law of Large Numbers (LLN), the approximation improves as b increases:

$$\lim_{b \rightarrow \infty} \hat{E}[g(X)] = E[g(X)]$$

Additionally, the Central Limit Theorem (CLT) implies that the distribution of the simulated estimate approaches a normal distribution as the number of simulations grows. The variance of the simulated estimate can be approximated by:

$$V[\hat{E}[g(X)]] = \frac{\sigma_b^2}{b}$$

where σ_b^2 is the sample variance. This variance can be estimated as:

$$\sigma_g^2 = \frac{1}{b} \sum_{i=1}^b (g(X_i) - E[g(X_i)])^2$$

which is the standard variance estimator for iid samples. The standard error of the estimate, $\frac{\sigma_g}{\sqrt{b}}$, indicates the accuracy of the approximation. This allows us to adjust b to achieve any desired level of precision.

15.4.1 Antithetic Variates

Antithetic variates are a technique to improve the accuracy of simulation by generating a second set of random variables that are negatively correlated with the original set of random variables. These pairs of variables are created using a single uniform random number. Each uniform variable U_i generates its counterpart $1 - U_i$, both of which are then mapped through the inverse cumulative distribution function (CDF) to generate correlated random variables.

Using antithetic variates reduces the simulation error, but only if the function $g(X)$ is monotonic, which ensures that the negative correlation between the paired variables results in reduced error. The benefit of this approach comes from the negative correlation, which decreases the standard error and thus increases the accuracy of the simulation.

15.4.2 Control Variates

Control variates are another technique to enhance simulation accuracy. A control variate is a derived random variable $h(X_i)$, which is correlated with the primary variable $g(X_i)$ but has a known mean, typically zero. To be effective, a good control variate must satisfy two criteria:

1. It should be computationally inexpensive to construct from the data x_i , offering a more cost-effective alternative to increasing the number of simulations.

2. It should exhibit a high correlation with the function $g(X)$, making it useful for reducing the error in the primary estimate.

The optimal parameter β , which minimizes the approximation error, is typically found using regression:

$$g(x_i) = \alpha + \beta h(x_i)$$

By adjusting β , the control variate technique helps improve the accuracy of the estimated value of $g(X)$.

15.4.3 Simulating option prices

To simulate the price of a financial option, we assume that the logarithm of the stock price follows a normal distribution. The log of the stock price at time T , denoted s_T , is given by:

$$s_T = s_0 + T \left(r_f - \frac{\sigma^2}{2} \right) + \sqrt{T} x_i$$

where x_i is a random variable sampled from a normal distribution $N(0, \sigma^2)$, r_f is the risk-free rate, and σ^2 is the variance of the stock return. The final stock price is then $S_T = \exp(s_T)$, and the present value of the option's payoff is:

$$C = \exp(-r_f T) \max(S_T - K, 0)$$

where K is the strike price of the option. The expected price of a call option can be estimated by averaging the simulated payoffs:

$$E[C] = \bar{C} = \frac{1}{b} \sum_{i=1}^b C_i$$

where C_i represents the simulated payoff for each instance. This approach is particularly useful for complex pricing scenarios where analytical solutions may not exist.

```
# helpers for random number generator and monte carlo simulation
class RNG:
    """Helper to generate random normal variables, with optional antithetic variates"""
    def __init__(self, seed=None, antithetic=False, ppf=stats.norm.ppf):
        self.ppf = ppf
        self.antithetic = antithetic
        self._prev = None # to track if antithetic pair available to return next
        self.seed = seed
        self.rng = np.random.default_rng(seed)

    def __call__(self, shape=1, **kwargs):
        _shape = (shape, ) if isinstance(shape, int) else shape
        n = np.prod(_shape)
        if self.antithetic: # generate half as many rv's, by returning 1-rv
            new = int((n + 1) / 2) if self._prev is None else int(n / 2)
            new = self.rng.random((new,))
            rem = 1 - new[:n - len(new) - int(self._prev is not None)]
            last = new[-1] if len(rem) < len(new) else None # if last pair unused
            out = [] if self._prev is None else [1 - self._prev]
            #out.extend(new)
            #out.extend(rem)
            for x, y in zip(new[:len(rem)], rem):
                out.extend([x, y])
            self._prev = last
        else:
            new = self.rng.random(_shape)
            out = new
        return out
```

(continues on next page)

(continued from previous page)

```

if last is not None:
    out.extend([last])
out = np.array(out)
self._prev = last
else:
    out = self.rng.random(_shape)
if shape == 1:
    return self.ppf(out[0], **kwargs)
else:
    return self.ppf(out.reshape(_shape), **kwargs)

def monte_carlo(rng, S, K, sigma, r, T, control=None):
    """Helper to price European call option by Monte Carlo Simulation
    Args:
        control : True price of European put option, as control variate
    """
    if rng.antithetic:
        label = 'Both' if control else 'Antithetic'
    else:
        label = 'Control' if control else 'Standard'
    result = {}
    for b in [50*4**i for i in range(9)]:
        tic = time.time()
        x = rng(b, scale=sigma)
        s = S * np.exp(T * (r - sigma**2/2) + np.sqrt(T) * x)
        c = np.exp(-r * T) * np.maximum(s - K, 0)
        if control is not None: # to apply control variate method
            error = np.exp(-r * T) * np.maximum(K - s, 0) - control # error of put
        else:
            ols = stats.linregress(x=error, y=c) # compute best hedge to minimize error
            c = c - ols.slope * error
        result[b] = dict(Price=np.mean(c).round(2),
                         StdErr=(np.std(c) / np.sqrt(b)).round(2),
                         elapsed = np.round(time.time() - tic, 4))
    return DataFrame.from_dict(result, orient='index').rename_axis(columns=label)

RNG(antithetic=True)((2, 3)) # generate normal r.v.'s with antithetic variates

```

```
array([[-0.72886259,  0.72886259,  0.25403329],
       [-0.25403329, -0.81873243,  0.81873243]])
```

Standard simulation

```

S = 2500
K = 2500
sigma = 0.164
r = 0.02
T = 2

seed = 42
rng = RNG(seed=seed)
monte_carlo(rng, S=S, K=K, sigma=sigma, r=r, T=T)

```

Standard	Price	StdErr	elapsed
50	297.91	54.77	0.0005
200	247.65	25.97	0.0003
800	283.95	15.23	0.0003
3200	279.62	7.59	0.0004
12800	278.25	3.73	0.0009
51200	279.45	1.85	0.0027
204800	278.93	0.93	0.0112
819200	278.31	0.46	0.0491
3276800	278.59	0.23	0.2490

Antithetic Variates

```
rng = RNG(seed=seed, antithetic=True)
monte_carlo(rng, S=S, K=K, sigma=sigma, r=r, T=T)
```

Antithetic	Price	StdErr	elapsed
50	280.86	54.41	0.0004
200	242.12	25.04	0.0006
800	276.97	14.15	0.0013
3200	284.38	7.63	0.0013
12800	277.30	3.72	0.0014
51200	279.28	1.87	0.0052
204800	278.15	0.93	0.0225
819200	278.46	0.46	0.1279
3276800	278.48	0.23	0.4977

Control Variates

```
control = put(S, K, sigma, r, T)    # Black-Scholes-Merton put price
rng = RNG(seed=seed)
monte_carlo(rng, S=S, K=K, sigma=sigma, r=r, T=T, control=control)
```

Control	Price	StdErr	elapsed
50	263.59	47.04	0.0010
200	251.85	22.76	0.0003
800	287.49	13.52	0.0002
3200	279.92	6.76	0.0003
12800	278.43	3.32	0.0010
51200	279.01	1.64	0.0022
204800	279.02	0.83	0.0109
819200	278.22	0.41	0.0441
3276800	278.64	0.21	0.3454

Both Antithetic and Control Variates

```
control = put(S, K, sigma, r, T)    # Black-Scholes-Merton put price
rng = RNG(seed=seed, antithetic=True)
monte_carlo(rng, S=S, K=K, sigma=sigma, r=r, T=T, control=control)
```

Both	Price	StdErr	elapsed
50	286.14	45.71	0.0006
200	227.29	22.03	0.0008
800	277.91	12.32	0.0003

(continues on next page)

(continued from previous page)

3200	286.51	6.79	0.0008
12800	276.56	3.31	0.0018
51200	279.47	1.66	0.0058
204800	277.88	0.83	0.0244
819200	278.41	0.41	0.1019
3276800	278.41	0.21	0.5569

References:

F. Black and M. Scholes, "The Pricing of Options and Corporate Liabilities," *Journal of Political Economy*, 81 (May/June 1973): 637–59

R. C. Merton, "Theory of Rational Option Pricing," *Bell Journal of Economics and Management Science* 4 (Spring 1973): 141–183.

Cox, J. C., Ross, S. A., & Rubinstein, M. (1979). Option Pricing: A Simplified Approach. *Journal of Financial Economics*, 7(3), 229-263.

Terence Lim, Andrew W. Lo, Robert C. Merton and Myron S. Scholes (2006), "The Derivatives Sourcebook", Foundations and Trends in Finance: Vol. 1: No. 5–6, pp 365-572. <http://dx.doi.org/10.1561/0500000005>

FRM Part I Exam Book Financial Markets and Products Ch. 14-15

FRM Part I Exam Book Quantitative Analysis Ch. 13

FRM Part I Exam Book Valuation and Risk Models Ch. 14-16

FRM Part II Exam Book Market Risk Measurement and Management Ch. 15

CHAPTER
SIXTEEN

VALUE AT RISK

The only constant in life is change - Heraclitus

Value at Risk (VaR) is a widely used risk measure in financial risk management that quantifies the potential loss in a portfolio over a given time period with a specified confidence level. However, VaR has limitations, including its inability to capture the severity of losses beyond its threshold. To address this, alternative measures such as Expected Shortfall (ES) have been introduced. We explore different methodologies for calculating VaR, including parametric, historical, and Monte Carlo simulation approaches, as well as advanced techniques such as stressed VaR and bootstrapping. Additionally, we examine conditional volatility models, including EWMA and GARCH, to account for changing market conditions. Finally, we discuss backtesting methods, such as the Kupiec Likelihood Ratio test and conditional coverage tests, to validate the accuracy of VaR models in real-world scenarios.

```
# By: Terence Lim, 2020-2025 (terence-lim.github.io)
from typing import Dict
import numpy as np
from scipy import stats
import pandas as pd
from pandas import DataFrame, Series
import statsmodels.api as sm
import matplotlib.pyplot as plt
from finds.readers import Alfred
from finds.utils import row_formatted
from secret import credentials
#pd.set_option('display.max_rows', None)
VERBOSE = 0
#%matplotlib qt
```

16.1 Risk measures

Value at Risk (VaR) is a fundamental risk measure that estimates the maximum potential loss in a portfolio over a specific period at a given confidence level. A VaR at an (α %) confidence level represents the loss threshold that has an (α %) probability of being exceeded.

16.1.1 Coherence

A major limitation of VaR is that it does not provide insight into the magnitude of losses beyond its threshold. Artzner et al. proposed four essential properties that a risk measure should satisfy:

1. **Monotonicity:** If one portfolio consistently performs worse than another under all conditions, it should have a higher risk measure.
2. **Translation Invariance:** Adding a risk-free cash amount K to a portfolio should reduce its risk measure by K .
3. **Homogeneity:** Scaling a portfolio by a factor of X should scale its risk measure by the same factor.
4. **Subadditivity:** The risk measure of a merged portfolio should not exceed the sum of the individual risk measures, ensuring diversification benefits.

A risk measure that satisfies all four properties is considered **coherent**. **Expected Shortfall (ES)** is a coherent risk measure, whereas VaR lacks subadditivity. ES is calculated as the probability-weighted average of losses beyond the VaR threshold. Other coherent risk measures can be derived by applying a risk aversion function to weight quantiles, with ES being a special case where tail quantiles receive equal weighting.

Retrieve crypto currency index returns from FRED

```
# retrieve crypto currency index from FRED, as log returns
alf = Alfred(api_key=credentials['fred']['api_key'], verbose=VERBOSE, convert_date=0)
cat = alf.get_category(33913)
cryptos = [alf(s['id']), log=1, diff=1]
    for s in cat['series'] if 'DISCONT' not in s['title']]
cryptos = pd.concat(cryptos, axis=1).sort_index()
date_str = f" ({str(cryptos.index[0])[:10]} to {str(cryptos.index[-1])[:10]})"
titles = Series({s['id']: s['title']}
    for s in cat['series'] if 'DISCONT' not in s['title']},
name=cat['name'])
```



```
# crypto index names
titles.to_frame().rename_axis(index=cat['id'])
```

Cryptocurrencies	
33913	
CBBCHUSD	Coinbase Bitcoin Cash
CBBTCUSD	Coinbase Bitcoin
CBETHUSD	Coinbase Ethereum
CBLTCUSD	Coinbase Litecoin

```
# recent crypto log returns
cryptos
```

```

      CCBCHUSD  CCBTCUSD  CBETHUSD  CBLTCUSD
date
2014-12-01      NaN        NaN        NaN        NaN
2014-12-02      NaN  0.021391        NaN        NaN
2014-12-03      NaN  0.000000        NaN        NaN
2014-12-04      NaN -0.002384        NaN        NaN
2014-12-06      NaN  0.002384        NaN        NaN
...
2025-02-26  0.010099  0.006085 -0.011953  0.018231
2025-02-27  0.062384 -0.003877 -0.030322  0.009735
2025-02-28 -0.014295  0.020210 -0.008916 -0.029413
2025-03-01  0.071640  0.091550  0.127686  0.028240
2025-03-02 -0.003326 -0.010081 -0.019098  0.015676

[3710 rows x 4 columns]

```

16.1.2 Parametric method

Assuming returns follow a normal distribution, VaR and ES at a confidence level α are given by:

$$VaR = -\mu + \sigma z_{1-\alpha}$$

$$ES = -\mu + \sigma \frac{\exp(-z_{1-\alpha}^2/2)}{(1-\alpha)\sqrt{2\pi}}$$

where z_α is the standard normal quantile corresponding to α .

For example, at a 95% confidence level ($\alpha = 0.95$), $z_\alpha = -1.645$, which represents the lower 5% quantile. In practice, μ and σ are estimated from historical data, but since short-term mean returns are difficult to measure accurately, μ is often assumed to be zero.

Under the assumption that geometric returns are normally distributed, then arithmetic returns follow a **lognormal distribution**. The skewness of the lognormal $(\exp(\sigma^2) + 2)\sqrt{(\exp(\sigma^2) - 1)}$ is always positive, hence the lognormal has a long right-tail. The lognormal has kurtosis $\exp(\sigma^2)^4 + 2\exp(\sigma^2)^3 + 3\exp(\sigma^2)^2 - 3$ which exceeds 3 and increases with volatility, hence exhibits fatter tails than the normal distribution.

```

# Helper to compute parametric VaR and ES
def parametric_risk(sigma: float | Series, alpha: float) -> Dict:
    """Calculate parametric gaussian VaR and ES"""
    var = -sigma * stats.norm.ppf(1 - alpha)
    es = sigma * stats.norm.pdf(stats.norm.ppf(1 - alpha)) / (1 - alpha)
    return dict(value_at_risk=var, expected_shortfall=es)

```

```

alpha = 0.95
volatility = {label: np.std(cryptos[label]) for label in cryptos}
parametric = DataFrame({label: Series(parametric_risk(std, alpha=alpha))
                           for label, std in volatility.items()})
print(f"Parametric Risk Measures (alpha={alpha})")
pd.concat([DataFrame.from_dict({'volatility': volatility}, orient='index'),
           parametric], axis=0).round(4)

```

```
Parametric Risk Measures (alpha=0.95)
```

	CBCHUSD	CBBTCUSD	CBETHUSD	CBLTCUSD
volatility	0.0576	0.0396	0.0505	0.0543
value_at_risk	0.0948	0.0651	0.0831	0.0894
expected_shortfall	0.1189	0.0817	0.1042	0.1121

16.1.3 Delta-Normal method

The delta-normal method provides an approximation for non-linear portfolios by assuming that underlying asset returns are normally distributed. The risk of the portfolio is modeled using **delta**, which measures the sensitivity of portfolio value to changes in underlying asset prices. A more refined approximation includes **gamma**, the second derivative of portfolio value with respect to asset prices, known as the **delta-gamma method**.

16.1.4 Monte Carlo simulation method

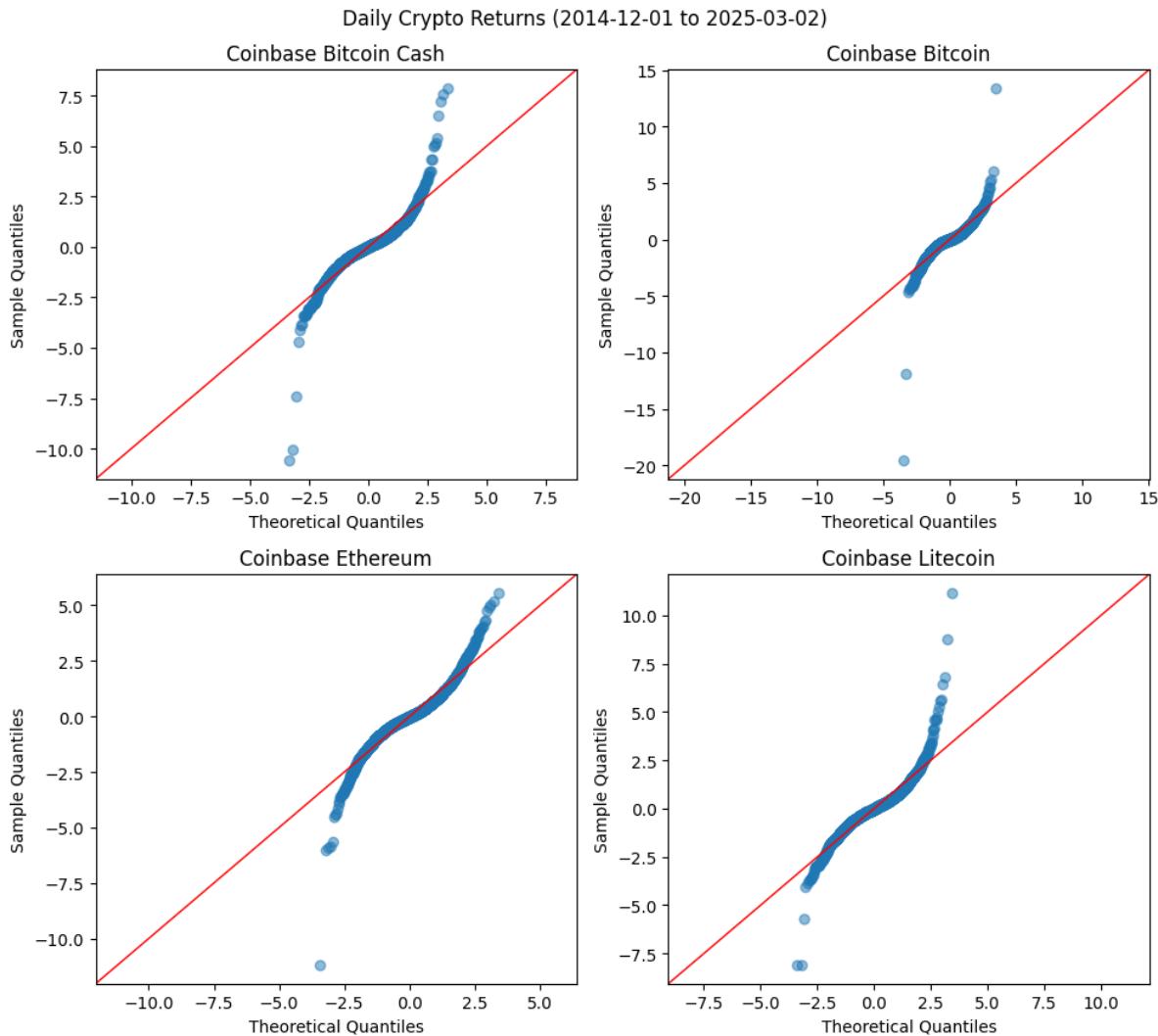
Monte Carlo simulation applies to both linear and non-linear portfolios. This approach generates random scenarios based on an assumed distribution for underlying risk factors. If 1,000 scenarios are simulated, the 95% VaR is estimated as the 50th worst loss (i.e., the 5th percentile), while Expected Shortfall is calculated as the average of the 49 worst losses.

However, standard Monte Carlo models assume normality and independence, which may not always reflect real-world financial data, necessitating the use of **non-parametric approaches**.

A **QQ plot** compares the empirical distribution of returns to a theoretical normal distribution. If returns exhibit heavier tails than the reference distribution, the QQ plot will have steeper slopes at the extremes.

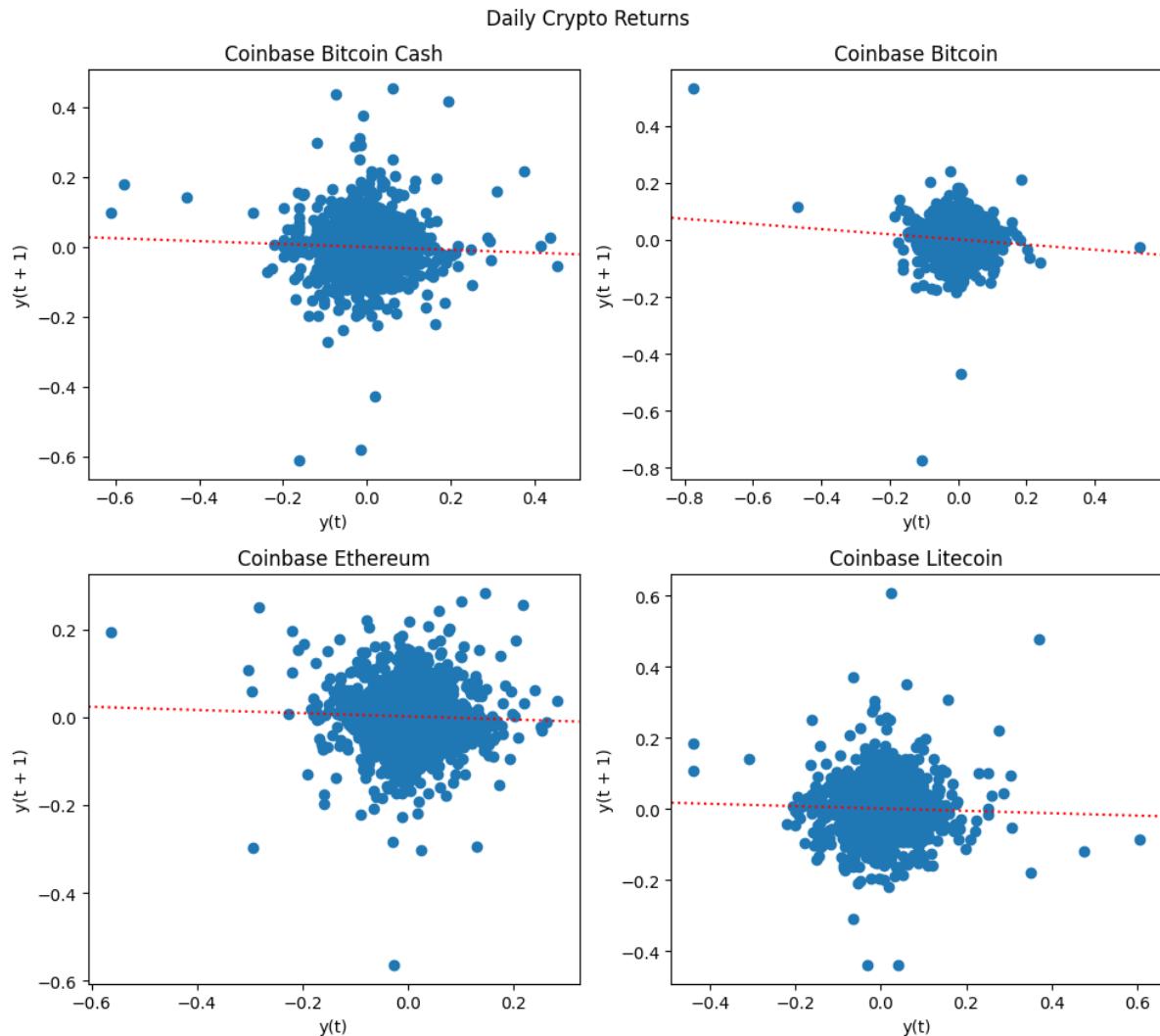
```
# QQ Plot for Gaussian assumption
from statsmodels.graphics.gofplots import ProbPlot
fig, axes = plt.subplots(2, 2, figsize=(10, 9))
for label, ax in zip(cryptos, axes.flatten()):
    pp = ProbPlot(cryptos[label].dropna(), fit=True)
    pp.qqplot(ax=ax, color='C0', alpha=.5)
    sm.qqline(ax=ax, fmt='r--', line='45', lw=1)
    ax.set_title(f"{titles[label]}")
plt.suptitle(f"Daily Crypto Returns" + date_str)
plt.tight_layout()
```

```
/home/terence/env3.11/lib/python3.11/site-packages/statsmodels/graphics/gofplots.
  ↪py:1043: UserWarning: color is redundantly defined by the 'color' keyword
  ↪argument and the fmt string "b" (-> color=(0.0, 0.0, 1.0, 1)). The keyword
  ↪argument will take precedence.
    ax.plot(x, y, fmt, **plot_style)
```



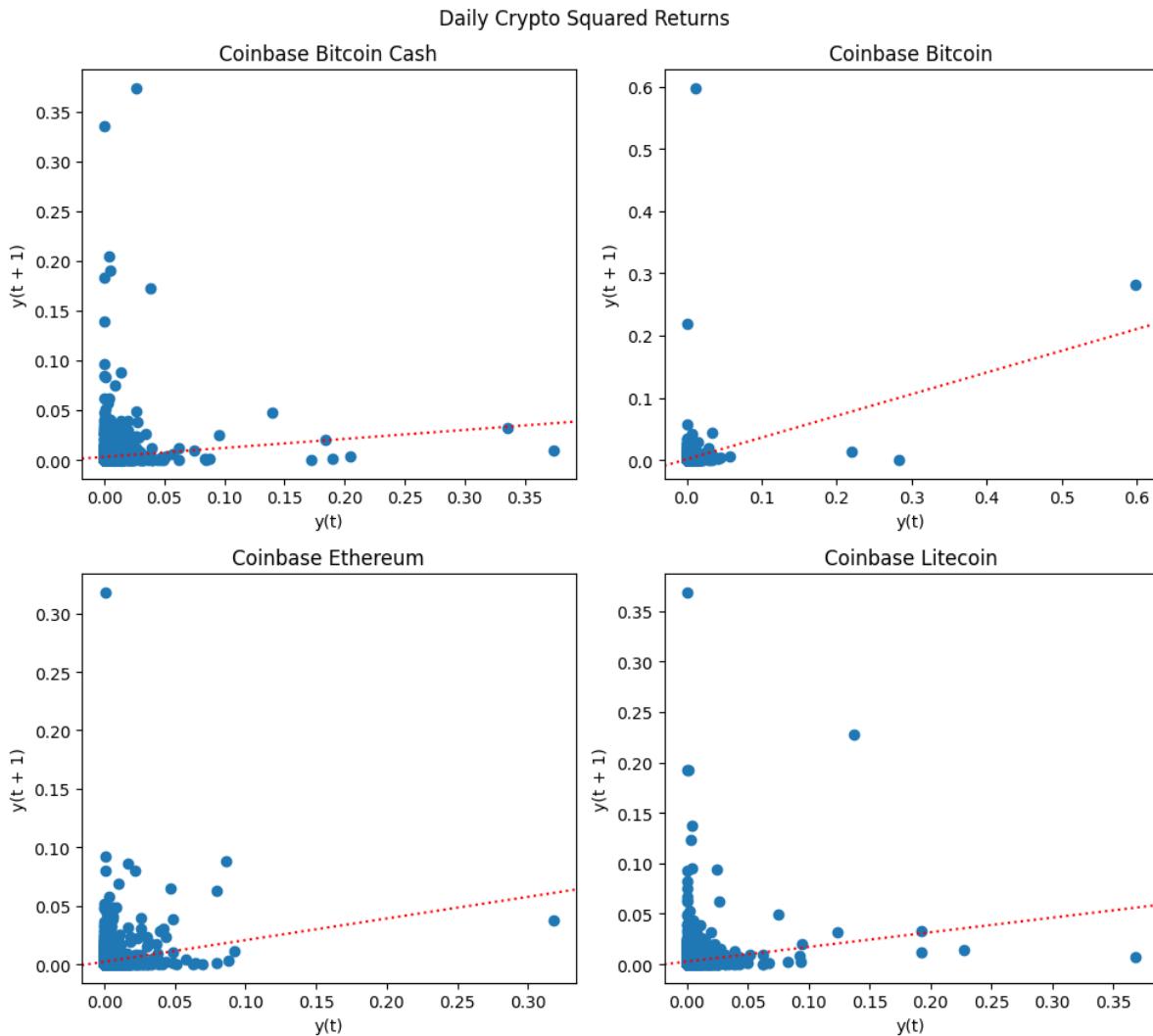
Lag plot of daily returns

```
# Autocorrelation of returns
import statsmodels.api as sm
fig, axes = plt.subplots(2, 2, figsize=(10, 9))
for label, ax in zip(cryptos, axes.flatten()):
    X = cryptos[label].dropna()
    pd.plotting.lag_plot(X, lag=1, ax=ax)
    r = stats.linregress(X.values[1:], X.values[:-1])
    ax.axline((0, r.intercept), slope=r.slope, ls=':', color="red")
    ax.set_title(f"\"{titles[label]}\"")
plt.suptitle("Daily Crypto Returns")
plt.tight_layout()
```



Lag plot of squared daily returns

```
# Autocorrelation of squared returns
fig, axes = plt.subplots(2, 2, figsize=(10, 9))
for label, ax in zip(cryptos, axes.flatten()):
    X = cryptos[label].dropna()**2
    pd.plotting.lag_plot(X, lag=1, ax=ax)
    r = stats.linregress(X.values[1:], X.values[:-1])
    ax.axline((0, r.intercept), slope=r.slope, ls=':', color="red")
    ax.set_title(f"\"{titles[label]}\"")
plt.suptitle("Daily Crypto Squared Returns")
plt.tight_layout()
```



16.1.5 Historical Simulation method

Historical simulation (HS) is the simplest non-parametric approach that estimates VaR by ordering historical losses and selecting the appropriate quantile. Expected Shortfall is calculated as the average of losses beyond the VaR threshold. This method accommodates non-normal features such as skewness and fat tails, making it robust in capturing market risks. However, its accuracy depends on whether the historical dataset adequately represents future market conditions.

```
# Helper to compute VaR, ES and sample moments from historical simulation
def historical_risk(X: Series, alpha: float):
    """Calculate historical VaR, ES, and sample moments"""
    X = X.dropna()
    N = len(X)
    var = -np.percentile(X, 100 * (1 - alpha))
    es = -np.mean(X[X < var])
    vol = np.std(X, ddof=0)
    skew = stats.skew(X)
    kurt = stats.kurtosis(X)
    jb = stats.jarque_bera(X) [0]
```

(continues on next page)

(continued from previous page)

```
    jbp = stats.jarque_bera(X) [1]
    return dict(N=N, value_at_risk=var, expected_shortfall=es, volatility=vol,
    skewness=skew, excess_kurtosis=kurt-3, jb_statistic=jb, jb_pvalue=jbp)
```

```
hist = DataFrame({label: historical_risk(cryptos[label], alpha=alpha)
    for label in cryptos})
print(f"Historical Risk Measures (alpha={alpha})")
row_formatted(hist.round(4), {'N': '{:.0f}'})
```

Historical Risk Measures (alpha=0.95)

	CBBCHUSD	CBBTCUSD	CBETHUSD	CBLTCUSD
N	2623	3709	3207	3117
value_at_risk	0.0831	0.0583	0.074	0.0822
expected_shortfall	0.0078	0.0032	0.0055	0.0061
volatility	0.0576	0.0396	0.0505	0.0543
skewness	-0.2394	-1.822	-0.4136	0.7306
excess_kurtosis	13.159	52.3855	6.0638	10.3101
jb_statistic	28562.5232	476116.6693	11069.1899	23285.8936
jb_pvalue	0.0	0.0	0.0	0.0

16.1.6 Stressed VaR method

During periods of market stress, volatility and correlations tend to rise, often leading to *correlation breakdowns* where asset prices move more synchronously. It is sometimes stated that “in stressed markets all correlations go to one.” Standard VaR models may not capture these dynamics accurately. **Stressed VaR** is calculated using historical periods of extreme market distress, such as the 2008 Financial Crisis or the 2021–2022 Crypto Winter, as opposed to simply the most recent number of years. Stress testing is designed to identify vulnerabilities, particularly those involving periods of high volatility.

```
beg, end = '2021-11-01', '2022-11-21' # dubbed "crypto winter"
stress = DataFrame({label: historical_risk(cryptos.loc[beg:end, label], alpha=alpha)
    for label in cryptos})
print(f"Stressed Risk Measures ({beg} to {end})")
row_formatted(stress.round(4).rename_axis(index='(alpha=0.05)'), {'N': '{:.0f}'})
```

Stressed Risk Measures (2021-11-01 to 2022-11-21)

	CBBCHUSD	CBBTCUSD	CBETHUSD	CBLTCUSD
(alpha=0.05)				
N	386	386	386	386
value_at_risk	0.0818	0.0624	0.0773	0.0911
expected_shortfall	0.0078	0.0058	0.0074	0.0053
volatility	0.0464	0.0348	0.0461	0.0478
skewness	-0.3194	-0.5904	-0.3785	-0.417
excess_kurtosis	-0.5048	0.7985	-0.6065	-1.0258
jb_statistic	106.6976	254.4822	101.3559	73.8708
jb_pvalue	0.0	0.0	0.0	0.0

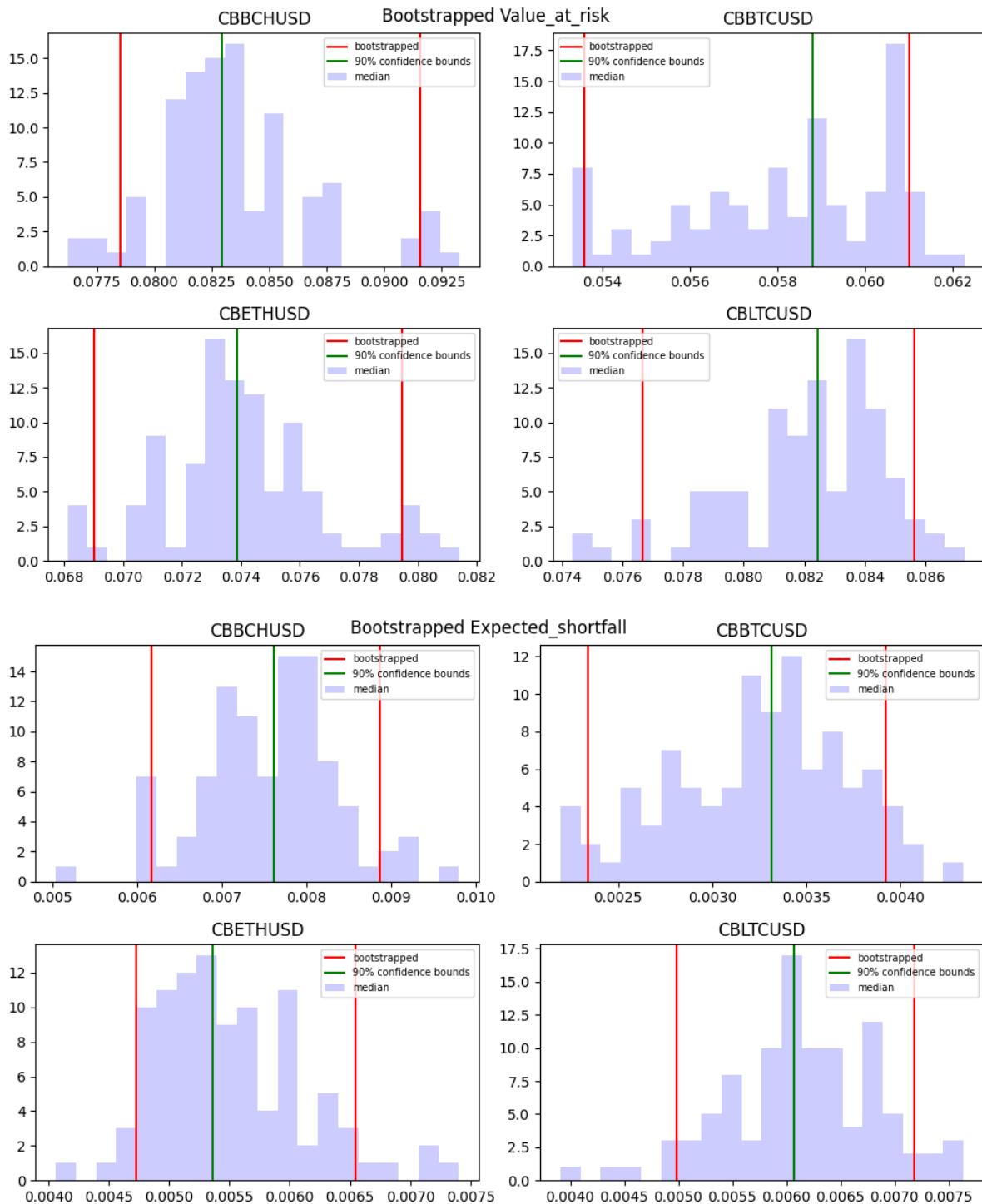
16.1.7 Bootstrap method

Bootstrap resampling improves historical simulation by repeatedly drawing random samples from the dataset with replacement. Each resampled dataset provides a new VaR or ES estimate, and the distribution of these estimates is used to compute confidence intervals. However, basic bootstrap methods assume that observations are independent over time, which may not hold for financial returns. Modifications such as the **block bootstrap** preserve time dependencies by sampling blocks of consecutive observations.

```
def bootstrap_risk(X: Series, alpha: float, n: int) -> dict:
    """Calculate bootstrap VaR, ES, confidence and plot VaR histogram"""
    X = X.dropna()
    N = len(X)
    bootstraps = []
    for _ in range(n):
        Z = Series(np.random.choice(X, N), index=X.index)
        bootstraps.append(historical_risk(Z, alpha=alpha))
    bootstraps = DataFrame.from_records(bootstraps)
    return bootstraps
```

```
def confidence_intervals(X: Series, confidence: float) -> dict:
    """Extracts confidence intervals and median from a series"""
    lower = (1 - confidence) / 2
    upper = lower + confidence
    return np.quantile(X, [lower, 0.5, upper], method='inverted_cdf')
```

```
# Run and plot bootstrapped VaR and ES
n = 100
confidence = 0.9
intervals = dict()
for measure in ['value_at_risk', 'expected_shortfall']:
    intervals[measure] = dict()
    fig, axes = plt.subplots(2, 2, figsize=(10, 6))
    for label, ax in zip(cryptos, axes.flatten()):
        bootstraps = bootstrap_risk(cryptos[label].dropna(), alpha=alpha, n=n)
        interval = confidence_intervals(bootstraps[measure], confidence=confidence)
        intervals[measure][label] = interval.tolist()
        ax.hist(bootstraps[measure], color='blue', alpha=0.2, bins=int(n/5))
        ax.axvline(x=interval[0], color='red')
        ax.axvline(x=interval[1], color='green')
        ax.legend(['bootstrapped', f'{confidence*100:.0f}% confidence bounds',
                  'median'], fontsize='x-small')
        ax.axvline(x=interval[2], color='red')
        ax.set_title(label)
    plt.tight_layout()
    plt.suptitle('Bootstrapped ' + measure.capitalize())
```



```
# display confidence intervals of VaR
DataFrame(intervals['value_at_risk'], index=['lower', 'median', 'upper'])\n    .rename_axis(index='Value at Risk')
```

	CBBCHUSD	CBBTCUSD	CBETHUSD	CBLTCUSD
Value at Risk				

(continues on next page)

(continued from previous page)

lower	0.078517	0.053589	0.069026	0.076687
median	0.082939	0.058818	0.073875	0.082435
upper	0.091576	0.061019	0.079481	0.085629

```
# display confidence intervals of VaR
DataFrame(intervals['expected_shortfall'], index=['lower', 'median', 'upper'])\
    .rename_axis(index='Expected Shortfall')
```

	CBCHUSD	CBTCUSD	CBETHUSD	CBLTCUSD
Expected Shortfall				
lower	0.006168	0.002339	0.004729	0.004981
median	0.007616	0.003314	0.005362	0.006072
upper	0.008870	0.003924	0.006546	0.007182

16.2 Conditional volatility models

A return distribution's characteristics may change over time. A mixture model of normal distributions with varying volatilities produces more peakedness and fatter tails than a simple normal distribution. In a model where returns are conditionally normal, the distribution is normal each day, while the standard deviation of the return varies over time. This leads to an unconditional distribution with fat tails.

16.2.1 EWMA model

Volatility can be estimated using an equal-weighted moving average of squared returns. However, this method suffers from sudden jumps when large returns enter or exit the dataset.

The **Exponentially Weighted Moving Average (EWMA)** model addresses this by applying exponentially decreasing weights to past returns:

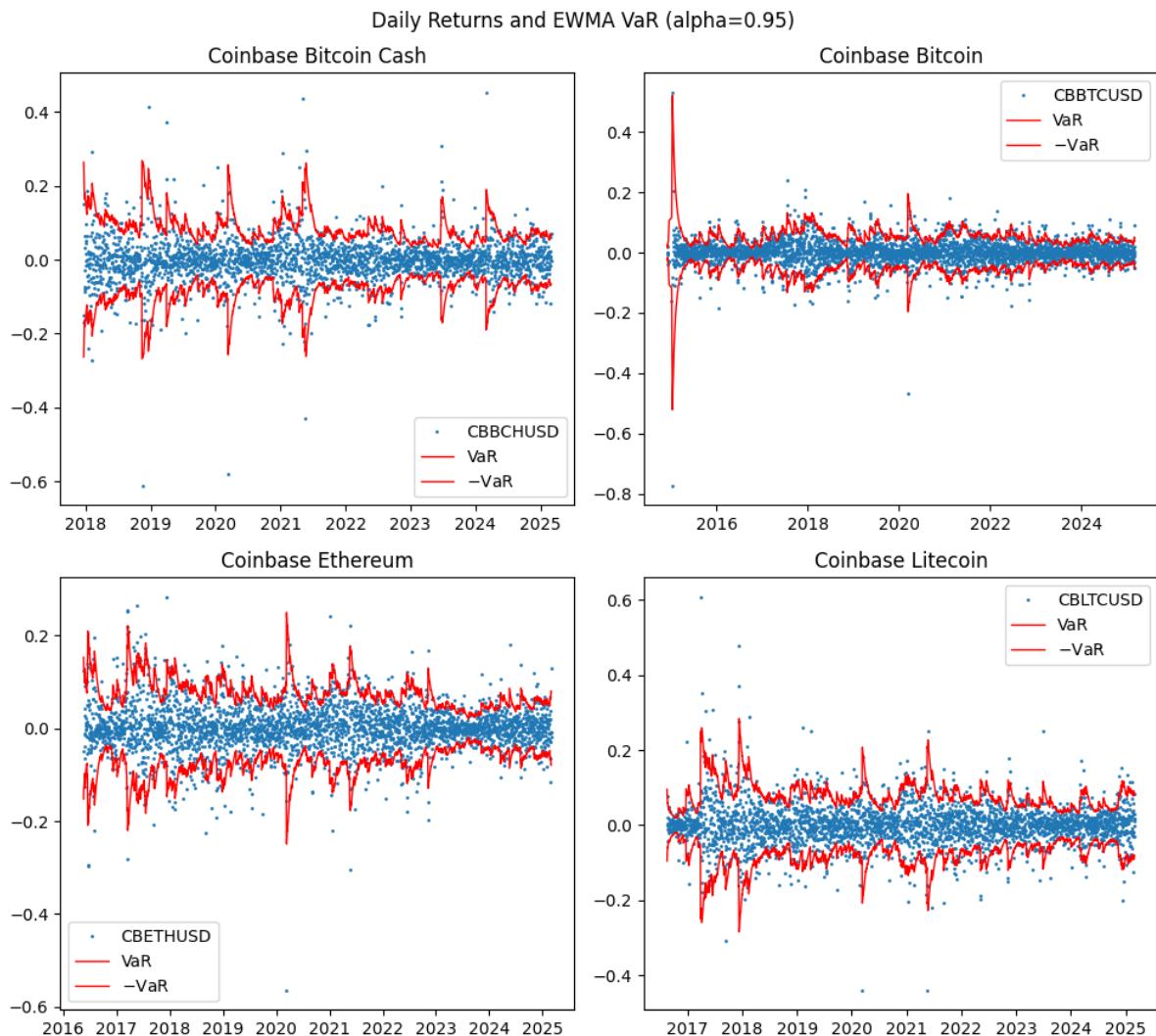
$$\sigma_t^2 = (1 - \lambda)r_{t-1}^2 + \lambda\sigma_{t-1}^2$$

where λ is a positive value less than 1 which determines the decay rate of past observations. This formula provides a very simple way of implementing EWMA. The new estimate of the variance rate on day t is a weighted average of the estimate of the variance rate made for the previous day $t - 1$, and the most recent observation of the squared return on day $t - 1$. In the 1990s, JP Morgan's **RiskMetrics** suggests $\lambda = 0.94$ for daily market volatility estimation.

```
# Estimate EWMA (lambda=0.94) rolling model predictions for all cryptos
lambda_ = 0.94
ewma = {label: np.sqrt((cryptos[label]**2).dropna().ewm(alpha=1-lambda_).mean())
        for label in cryptos}
```

```
# Helper to plot predicted VaR vs actual returns
def plot_var(X: Series, VaR: Series, ax: plt.Axes):
    """Helper to plot returns and VaR predictions"""
    ax.plot(X, ls=' ', marker='.', markersize=2)
    ax.plot(-VaR.shift(-1), lw=1, ls='-', c='r')
    ax.plot(VaR.shift(-1), lw=1, ls='-', c='r')
    ax.legend([X.name, 'VaR', '$-$VaR'])
```

```
# Plot daily returns and EWMA predicted VaR of all cryptos
fig, axes = plt.subplots(2, 2, figsize=(10, 9))
for label, ax in zip(cryptos, axes.flatten()):
    Z = pd.concat([cryptos[label].dropna(),
                   parametric_risk(ewma[label], alpha=alpha) ['value_at_risk']\n                   .rename('VaR')],\n                   join='inner', axis=1).dropna()
    plot_var(Z[label], VaR=Z['VaR'], ax=ax)
    ax.set_title(titles[label])
plt.suptitle(f"Daily Returns and EWMA VaR (alpha={alpha})")
plt.tight_layout()
```



```
# Properties of EWMA normalized returns for all cryptos
ewma_hist = dict()
for label in cryptos:
    X = (cryptos[label] / ewma[label].shift(-1)) # normalize by predict vol
    ewma_hist[label] = Series(historical_risk(X, alpha=0.95)).rename(label)
print("Normalized by EWMA predicted volatility (alpha=0.95)")
DataFrame(ewma_hist).round(4)
```

Normalized by EWMA predicted volatility (alpha=0.95)

	CBBCHUSD	CBBTCUSD	CBETHUSD	CBLTCUSD
N	2622.0000	3708.0000	3206.0000	3116.0000
value_at_risk	1.5056	1.4695	1.5831	1.5697
expected_shortfall	0.1241	0.0717	0.0828	0.0896
volatility	0.9269	0.9252	0.9432	0.9425
skewness	0.1035	-0.0464	-0.0177	0.0048
excess_kurtosis	-1.5298	-1.5609	-2.1157	-1.8466
jb_statistic	240.8139	321.2816	104.6335	172.7197
jb_pvalue	0.0000	0.0000	0.0000	0.0000

16.2.2 GARCH model

The **Generalized Autoregressive Conditional Heteroskedasticity (GARCH)** model, developed by Robert Engle and Tim Bollerslev, can be intuitively regarded as an extension of EWMA. In **GARCH (1,1)**, we also give some weight to a long run average variance $\hat{\sigma}$.

$$\sigma_t^2 = \alpha r_{t-1}^2 + \beta \sigma_{t-1}^2 + \gamma \hat{\sigma}$$

where $\alpha + \beta + \gamma = 1$. This introduces **mean reversion**, where the $\hat{\sigma}$ term provides a “pull” toward the long-run average, ensuring that volatility stabilizes over time. And just as with ARMA specifications of time series models, more complex **GARCH(p,q)** models can incorporate additional lags of q squared returns and p variance estimates for improved accuracy.

rugarch package in R

```
# Estimate GARCH(1, 1) by calling R's rugarch library
import rpy2.robj as ro
from rpy2.robj.packages import importr
from finds.utils import PyR

def rugarch(X: Series, savefig: str = '', verbose=VERBOSE) -> Series:
    """GARCH(1,1) wrapper over rugarch"""
    rugarch_ro = importr('rugarch') # to use library rugarch
    c_ = ro.r['c']
    list_ = ro.r['list']
    spec = ro.r['ugarchspec'](mean_model=list_(armaOrder=c_(0,0), include_mean=False))
    model = ro.r['ugarchfit'](spec, data=PyR(X.values).ro)
    if verbose:
        ro.r['show'](model)
    if savefig:
        for which in [4, 5, 10, 11]:
            ro.r['plot'](model, which=which)
            PyR.savefig(f'{savefig}{which}.png', display=None)
    return Series(PyR(ro.r['sigma'](model)).values.flatten(),
                 index=X.index, name=X.name)
```

```
# Estimate GARCH(1,1) full period model for all cryptos
garch = {label: rugarch(cryptos[label].dropna()) for label in cryptos}
```

```
# Plot daily returns and GARCH predicted VaR
alpha = 0.95 # VaR parameter
fig, axes = plt.subplots(2, 2, figsize=(10, 9))
```

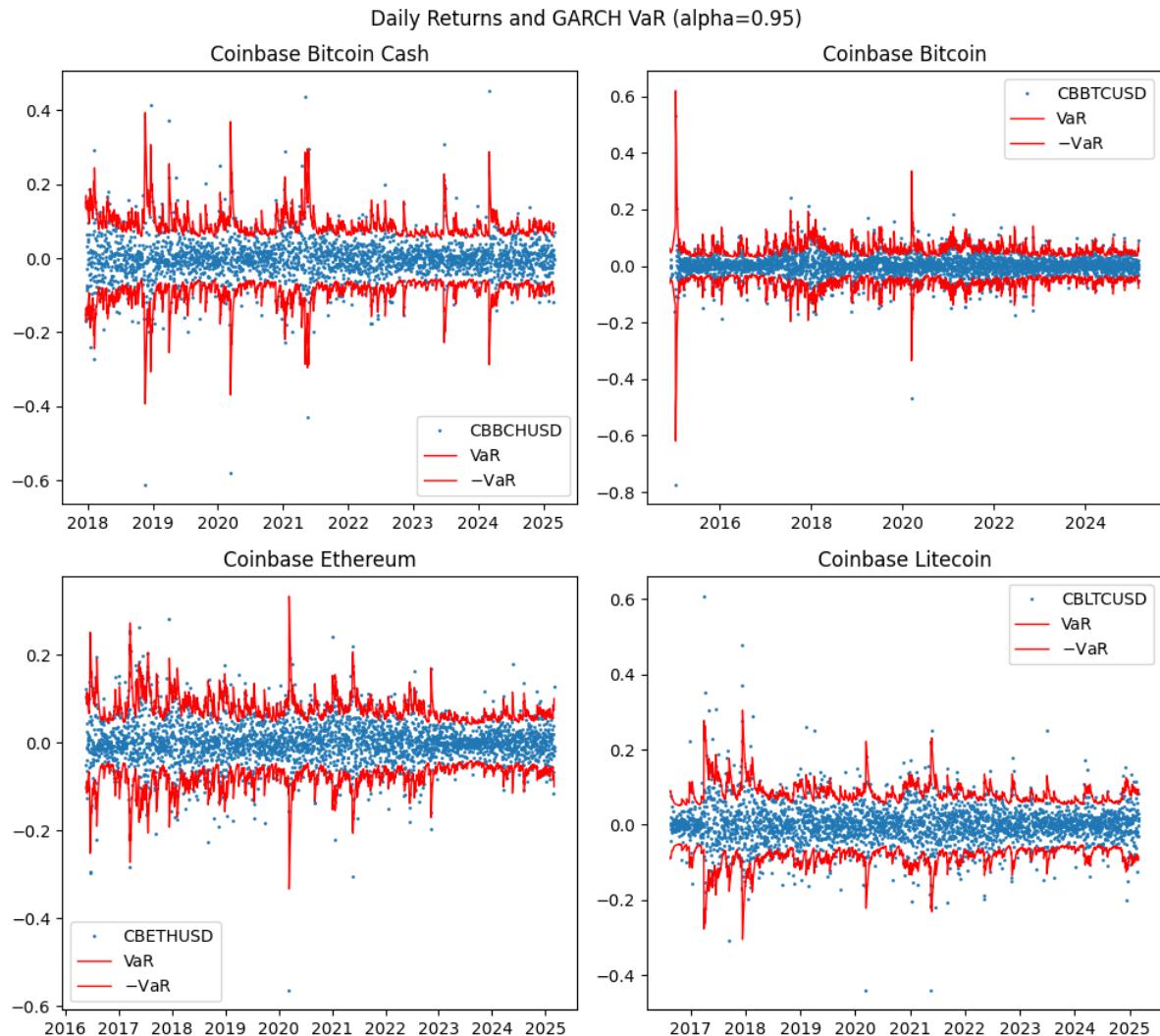
(continues on next page)

(continued from previous page)

```

for label, ax in zip(cryptos, axes.flatten()):
    Z = pd.concat([cryptos[label],
                  parametric_risk(garch[label], alpha=alpha) ['value_at_risk']\ 
                  .rename('VaR')],
                  join='inner', axis=1).dropna()
    plot_var(Z[label], VaR=Z['VaR'], ax=ax)
    ax.set_title(titles[label])
plt.suptitle(f"Daily Returns and GARCH VaR (alpha={alpha})")
plt.tight_layout()

```



```

# Properties of GARCH normalized returns
garch_hist = dict()
for label in cryptos:
    X = (cryptos[label] / garch[label].shift(-1)) # normalize by predict vol
    garch_hist[label] = Series(historical_risk(X, alpha=0.95)).rename(label)
print("Normalized by GARCH predicted volatility (alpha=0.95)")
DataFrame(garch_hist).round(4)

```

Normalized by GARCH predicted volatility (alpha=0.95)

	CBBCHUSD	CBBTCUSD	CBETHUSD	CBLTCUSD
N	2622.0000	3708.0000	3206.0000	3116.0000
value_at_risk	1.3382	1.2965	1.4123	1.3956
expected_shortfall	0.1062	0.0504	0.0789	0.0843
volatility	0.7901	0.7740	0.8411	0.8454
skewness	0.0185	-0.0364	-0.0033	0.0350
excess_kurtosis	-2.5903	-2.9028	-2.7553	-1.9190
jb_statistic	18.4912	2.2795	8.0058	152.3519
jb_pvalue	0.0001	0.3199	0.0183	0.0000

16.3 Backtesting VaR

The simplest method for validating a VaR model is failure rate analysis, which counts the proportion of times actual losses exceed the VaR estimate. If the model is accurate, these exceedances should follow a binomial distribution with probability $p = 1 - \alpha$, where α is the VaR confidence level. The VaR model can be rejected in two regions, both when the number of observed violations is too few or too many.

16.3.1 Kupiec Likelihood Ratio test

Kupiec (1995) proposed a likelihood ratio test for backtesting VaR based on the number of observed violations:

$$LR = -2[(N - S) \ln(1 - p) + S \ln(p)] + 2[(N - S) \ln(1 - S/N) + S \ln(S/N)]$$

where S is the number of VaR exceedances in N observations.

```
def kupiec_LR(alpha: float, s: int, n: int):
    """Compute Kupiec likelihood ratio given s violations in n trials

    Returns:
        Dict of likelihood statistic and pvalue
    """
    p = 1 - alpha      # prob of violation
    num = np.log(1 - p)*(n - s) + np.log(p)*s
    den = np.log(1 - (s/n))*(n - s) + np.log(s/n)*s
    lr = -2 * (num - den)
    return {'lr': lr, 'violations': s, 'N': n,
            '# 5%_critical': stats.chi2.ppf(0.95, df=1),
            'pvalue': 1 - stats.chi2.cdf(lr, df=1)}
```

```
def kupiec(X: Series, VaR: Series, alpha: float) -> Dict:
    """Kupiec Likelihood Ratio test of VaR

    Returns:
        Dict of likelihood statistic and pvalue
    """
    Z = pd.concat([X, VaR], axis=1).dropna()
    n = len(Z)
    s = np.sum(Z.iloc[:, 0] < -Z.iloc[:, 1])  # number of violations < -VaR
    return kupiec_LR(alpha=alpha, s=s, n=n)
```

```
# Kupiec likelihood ratio test for EWMA
row_formatted(DataFrame(
    {label: kupiec(cryptos[label],
        VaR=parametric_risk(ewma[label], alpha=alpha) ['value_at_risk'],
        alpha=alpha) for label in cryptos})\
    .rename_axis(index=f"EWMA({lambda_:})", columns="Kupiec LR Test:")\
    .round(4),
    {'N': '{:.0f}', 'violations': '{:.0f}'})
```

	CBBCHUSD	CBBTCUSD	CBETHUSD	CBLTCUSD
EWMA(0.94)				
lr	1.9136	3.8477	1.3918	3.0693
violations	116	160	146	135
N	2623	3709	3207	3117
pvalue	0.1666	0.0498	0.2381	0.0798

```
# Kupiec likelihood ratio test for GARCH(1,1)
row_formatted(DataFrame(
    {label: kupiec(cryptos[label],
        VaR=parametric_risk(garch[label], alpha=alpha) ['value_at_risk'],
        alpha=alpha) for label in cryptos})\
    .rename_axis(index=f"GARCH(1,1)", columns="Kupiec LR Test:").round(4),
    {'N': '{:.0f}', 'violations': '{:.0f}'})
```

	CBBCHUSD	CBBTCUSD	CBETHUSD	CBLTCUSD
GARCH(1,1)				
lr	6.8446	5.5607	3.4345	12.3497
violations	103	155	138	115
N	2623	3709	3207	3117
pvalue	0.0089	0.0184	0.0638	0.0004

16.3.2 Conditional coverage tests

Value-at-Risk (VaR) violations should occur randomly over time; if violations cluster, it may suggest that the model is misspecified. Conditional coverage tests, which extend Kupiec's test, evaluate not only the frequency but also the independence of exceptions over time.

```
# TODO: conditional likelihood test
```

References:

Jorion, Phillippe. Value at Risk.

P. Artzner, F. Delbaen, J.-M. Eber, and D. Heath, “Coherent Measures of Risk”, Mathematical Finance 9 (1999): 203–228.

Kupiec, P. (1995). Techniques for Verifying the Accuracy of Risk Measurement Models. Journal of Derivatives, 3, 73-84.

FRM Part I Exam Book Ch. 1-3

FRM Part II Exam Book Market Risk Measurement and Management Ch. 1-2

CHAPTER
SEVENTEEN

MARKET MICROSTRUCTURE

Beware of little expenses. A small leak will sink a great ship - Benjamin Franklin

Market microstructure focuses on the mechanics of how securities are traded, analyzing factors such as price formation, liquidity, and trading costs. The NYSE Trade and Quote (TAQ) dataset is a widely used source of tick data, containing detailed records of executed trades and best bid and offer quotes. We analyze how key liquidity measures vary across market capitalization and during the trading day. We also examine intraday volatility patterns through the variance ratio and high-frequency estimators.

```
# By: Terence Lim, 2020-2025 (terence-lim.github.io)
import numpy as np
import pandas as pd
import time
import os
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
from tqdm import tqdm
import multiprocessing
from finds.database import SQL, RedisDB
from finds.structured import CRSP, BusDay
from finds.readers import opentaq, itertaq, bin_trades, bin_quotes, TAQ, \
    clean_trade, clean_nbbo, align_trades, plot_taq
from finds.utils import plot_time, Store, row_formatted
from finds.recipes import weighted_average, hl_vol, ohlc_vol
from secret import credentials, paths
import warnings
VERBOSE = 0
if not VERBOSE: # Suppress FutureWarning messages
    warnings.simplefilter(action='ignore', category=FutureWarning)

%matplotlib inline
```

```
sql = SQL(**credentials['sql'], verbose=VERBOSE)
user = SQL(**credentials['user'], verbose=VERBOSE)
bday = BusDay(sql)
rdb = RedisDB(**credentials['redis'])
crsp = CRSP(sql, bday, rdb=rdb, verbose=VERBOSE)
```

```
taqdir = paths['taq']
storedir = paths['scratch'] / 'ticks'
open_t = pd.to_datetime('1900-01-01T9:30')    # exclude <=
close_t = pd.to_datetime('1900-01-01T16:00')  # exclude >
```

(continues on next page)

(continued from previous page)

```
EPSILON = 1e-15
dates = [20191007, 20191008, 20180305, 20180306]
```

17.1 Tick data

17.1.1 NYSE TAQ

The **NYSE Trade and Quote (TAQ) dataset** contains tick-by-tick intraday trading data for U.S. equities, providing information on executed trades and best bid/offer quotes from various exchanges. There are three primary types of TAQ daily files:

1. Trades (EQY_US_ALL_TRADE_YYYYMMDD.gz) – containing executed trades data, including price, volume, microsecond timestamps, sale conditions and trade correction indicators
2. National Best Bid and Offer NBBO (EQY_US_ALL_NBBO_YYYYMMDD.gz) - containing best bid/offer price, size, microsecond timestamps, quote conditions and market center identifiers
3. Master (EQY_US_ALL_REF_MASTER_YYYYMMDD.gz) – containing reference information about securities, including symbol, CUSIP, security description, shares outstanding and primary exchange

The `taq` module in the FinDS package provides tools for processing NYSE TAQ data, with functionalities including:

- File reading & indexing: `open_taq`, `iterataq`, `taq_from_csv`
- Data cleaning & filtering: `clean_trades`, `clean_nbbo`
- Tick-level analysis: `TAQ` class
- Resampling & binning: `bin_trades`, `bin_quotes`, `align_trades`
- Visualization: `plot_taq`

Bad trades and quotes records, such as invalid prices, duplicate records, and specific sale conditions, can be filtered out to ensure data integrity.

Retrieve and visualize tick-level trades and quotes for a selected stock (e.g., VOO, an ETF tracking the S&P 500).

```
# Plot VOO tick-by-tick
master, trades, quotes = opentaq(dates[0], taqdir)
symbol = "VOO"
t = trades[symbol]
q = quotes[symbol]
ct = clean_trade(t, close_t=close_t + np.timedelta64('5', 'm'))
cq = clean_nbbo(q)
align_trades(ct, cq, inplace=True)

plot_taq(ct[['Trade_Price', 'Prevailing_Mid']].groupby(level=0).last(),
         ct[['Trade_Volume']].groupby(level=0).last(),
         (cq['Best_Offer_Price'] - cq['Best_Bid_Price']) \
         .rename('Quoted_Spread').groupby(level=0).last(),
         ((cq['Best_Bid_Size'] + cq['Best_Offer_Size']) / 2) \
         .rename('Depth').groupby(level=0).last(),
         open_t=open_t,
         close_t=close_t + np.timedelta64('5', 'm'),
         num=1,
         title=f"Tick Prices, Volume, Quotes, Spreads, and Depths ({dates[0]})"
)
```

```
(<Axes: title={'center': 'Tick Prices, Volume, Quotes, Spreads, and Depths',
  xlabel='VOO', ylabel='Trade_Price'>,
<Axes: xlabel='VOO', ylabel='Quoted Spread'>)
```



The data is preprocessed to extract a universe of U.S.-domiciled common stocks, and indexed by ticker symbol for efficient access.

```

for d, date in enumerate(dates):
    store = Store(storedir / str(date), verbose=VERBOSE)
    master, trades, quotes = opentaq(date, taqdir)

    # screen on CRSP universe
    univ = crsp.get_universe(date) \
        .join(crsp.get_section(dataset='names',
                               fields=['ncusip', 'permco', 'exchcd'],
                               date_field='date',
                               date=date,
                               start=0), how='inner') \
        .sort_values(['permco', 'ncusip'])

    # drop duplicate share classes (same permco): keep largest cap
    dups = master['CUSIP'].str \
        .slice(0, 8) \
        .isin(univ.loc[univ.duplicated(['permco'], keep=False),
                      'ncusip'])

    #shareclass.extend(master[dups].to_dict(orient='index').values())
    univ = univ.sort_values(['permco', 'cap'], na_position='first') \
        .drop_duplicates(['permco'], keep='last') \
        .reset_index() \
        .set_index('ncusip', drop=False)

    # Iterate by symbol over Daily Taq trades, nbbo and master files
    for ct, cq, mast in itertaq(trades,
                                 quotes,
                                 master,
                                 cusips=univ['ncusip'],
                                 open_t=open_t,
                                 close_t=None,
                                 verbose=VERBOSE):
        header = {'date':date}
        header.update(univ.loc[mast['CUSIP'][:8],
                               ['permno', 'decile', 'exchcd', 'siccd']])
        header.update(mast[['Symbol', 'Round_Lot']])
        store[header['Symbol']] = dict(header=header, ct=ct, cq=cq, mast=mast)
        quotes.close()
        trades.close()

```

17.2 Liquidity measures

- **Depth:** The average bid and ask size at the best quotes, indicating the available liquidity at the current market price.
- **Quoted Spread:** The difference between the best ask and best bid prices, representing the cost of immediacy for traders.
- **Effective Spread:** A trade-based measure of execution cost, calculated as twice the absolute difference between the trade price and the midquote.
- **Price Impact:** The change in the midquote price after a trade, reflecting how much a trade moves the market.
- **Realized Spread:** The difference between the trade price and the midquote price a few minutes later, measuring the profitability of liquidity providers.
- **Lee-Ready Tick Test:** A method for classifying trades as buyer- or seller-initiated, using trade price movements relative to the prevailing midquote.

- **Volume Weighted Average Price (VWAP)** – A common trade execution benchmark, though it can be influenced by the trade itself – achieving zero slippage to VWAP while accounting for 100% of market volume does not necessarily indicate good execution.

17.2.1 Intraday liquidity

Intraday liquidity is analyzed computing liquidity measures across various time intervals: 1-second, 2-second, 5-second, 15-second, 30-second, 1-minute, 2-minute, and 5-minute bins.

```

intervals = ([(v, 's') for v in [1, 2, 5, 15, 30]] + [(v, 'm') for v in [1, 2, 5]])
max_num = 100000
bin_keys = ['effective', 'realized', 'impact',
            'quoted', 'volume', 'offersize', 'bidsize',
            'ret', 'retq', 'counts']

# helper call run liquidity calculations by date, parallelizable
def intraday(date):
    """Compute intraday liquidity for a date"""
    store = Store(storedir / str(date), verbose=VERBOSE)
    symbols = sorted(store)
    daily_all = []
    bins_all = {k: [] for k in bin_keys}
    for num, symbol in enumerate(symbols):
        if num >= max_num:      # set small max_num for debugging
            break
        header, ct, cq, mast = store[symbol].values()

        # Compute and collect daily and bin statistics at all intervals
        daily = header.copy()    # to collect current stock's daily stats

        # Compute effective spreads by large and small trade sizes
        med_volume = mast['Round_Lot'] * (cq['Best_Bid_Size'].median()
                                           + cq['Best_Offer_Size'].median()) / 2.
        data = ct.loc[(ct.index > open_t) & (ct.index < close_t),
                      ['Trade_Price', 'Prevailing_Mid', 'Trade_Volume']]
        eff_spr = data['Trade_Price'].div(data['Prevailing_Mid']).sub(1).abs()
        eff_large = eff_spr[data['Trade_Volume'].ge(med_volume).to_numpy()]
        daily['large_trades'] = len(eff_large)
        daily['large_volume'] = data.loc[data['Trade_Volume'].ge(med_volume),
                                           'Trade_Volume'].mean()
        daily['large_spread'] = eff_large.mean()
        eff_small = eff_spr[data['Trade_Volume'].lt(med_volume)]
        daily['small_trades'] = len(eff_small)
        daily['small_volume'] = data.loc[data['Trade_Volume'].lt(med_volume),
                                           'Trade_Volume'].mean()
        daily['small_spread'] = eff_small.mean()

        v, u = intervals[-1]
        for (v, u) in intervals:
            bt = bin_trades(ct, v, u, open_t=open_t, close_t=close_t)
            bq = bin_quotes(cq, v, u, open_t=open_t, close_t=close_t)
            daily[f"tvar{v}{u}"] = bt['ret'].var(ddof=0) * len(bt)
            daily[f"tvarHL{v}{u}"] = ((hl_vol(bt['maxtrade'], bt['mintrade'])**2
                                       * len(bt)))
            daily[f"tvarOHLC{v}{u}"] = ((ohlc_vol(bt['first'],

```

(continues on next page)

(continued from previous page)

```

        bt['maxtrade'],
        bt['mintrade'],
        bt['last'])**2)
    * len(bt))
daily[f"qvar{v}{u}"] = bq['retq'].var(ddof=0) * len(bq)
daily[f"qvarHL{v}{u}"] = ((hl_vol(bq['maxmid'], bq['minmid'])**2)
    * len(bq))
daily[f"qvarOHLC{v}{u}"] = ((ohlc_vol(bq['firstmid'],
    bq['maxmid'],
    bq['minmid'],
    bq['mid']))**2)
    * len(bq))
daily[f"tunch{v}{u}"] = np.mean(np.abs(bt['ret']) < EPSILON)
daily[f"qunch{v}{u}"] = np.mean(np.abs(bq['retq']) < EPSILON)
daily[f"tzero{v}{u}"] = np.mean(bt['counts'] == 0)

# Collect final (i.e. 5 minute bins) bt and bq intraday series
df = bq.join(bt, how='left')
for s in ['effective', 'realized', 'impact', 'quoted']:
    bins_all[s].append({**header,
        **(df[s]/df['mid']).to_dict()})
for s in ['volume', 'offersize', 'bidsize', 'ret', 'retq', 'counts']:
    bins_all[s].append({**header,
        **df[s].to_dict()})

# Collect daily means
daily.update(df[['bidsize', 'offersize', 'quoted', 'mid']].mean())
daily.update(df[['volume', 'counts']].sum())
daily.update(weighted_average(df[['effective', 'impact', 'realized',
    'vwap', 'volume']],
    weights='volume'))
daily_all.append(daily)

return DataFrame(daily_all), {k: DataFrame(bins_all[k]) for k in bin_keys}

```

To optimize performance, **multiprocessing** package is used to parallelize computations, allowing efficient distribution of input data across multiple processes using its **Pool** API.

```

# Run each day as a parallel thread, must be called as context manager or with close_
    ↵command
with multiprocessing.Pool(processes=min(os.cpu_count(), len(dates))) as p:
    data = p.map(intraday, dates)

```

```

# Combine data
daily_df = pd.concat([data[j][0] for j in range(len(data))], ignore_index=True)
bins_df = {k: pd.concat([data[j][1][k] for j in range(len(data))], ignore_index=True)
    for k in bin_keys}

```

```

# Store in scratch folder
store = Store(paths['scratch'])

```

```

store['tick.daily'] = daily_df
store['tick.bins'] = bins_df

```

```
# Fetch extracted data
daily_df = store['tick.daily']
bins_df = store['tick.bins']
```

17.3 By market capitalization

The daily averages of liquidity measures are analyzed across different market capitalization categories.

```
# group by market cap (NYSE deciles 1-3, 4-6, 7-9, 10) and exchange listed
daily_df['Size'] = pd.cut(daily_df['decile'],
                           [0, 3.5, 6.5, 9.5, 11],
                           labels=['large', 'medium', 'small', 'tiny'])
groupby = daily_df.groupby(['Size'], observed=False)
```

```
# collect results for each metric
results = {}      # to collect results as dict of {column: Series}
formats = {}      # and associated row formatter string
results.update(groupby['mid']\
               .count()\
               .rename('Number of Stock/Days').to_frame())
formats.update({'Number of Stock/Days': '{:.0f}'})
```

```
result = groupby[['mid', 'vwap']].mean()    # .quantile(), and range
result.columns = ['Midquote Price', "VWAP"]
formats.update({k: '{:.2f}' for k in result.columns})
results.update(result)
```

```
result = groupby[['counts', 'volume']].mean()
result.columns = ['Number of trades', "Volume (shares)"]
formats.update({k: '{:.0f}' for k in result.columns})
results.update(result)
```

```
# volatility from 5m intervals
result = np.sqrt(groupby[['tvar5m', 'qvar5m', 'tvarHL5m', 'qvarHL5m',
                           'tvarOHLC5m', 'qvarOHLC5m']].mean())
result.columns = ['Volatility(trade price)', "Volatility(midquote)",
                  'Volatility(HL trade price)', "Volatility(HL midquote)",
                  'Volatility(OHLC trade price)', "Volatility(OHLC midquote)"]
formats.update({k: '{:.4f}' for k in result.columns})
results.update(result)
```

```
result = groupby[['offersize', 'bidsize']].mean()
result.columns = [s.capitalize() + ' (lots)' for s in result.columns]
formats.update({k: '{:.1f}' for k in result.columns})
results.update(result)
```

```
spr = ['quoted', 'effective', 'impact', 'realized']
result = groupby[spr].mean()
result.columns = [s.capitalize() + ' $ spread' for s in spr]
```

(continues on next page)

(continued from previous page)

```
formats.update({k: '{:.4f}' for k in result.columns})
results.update(result)
```

```
rel = [s.capitalize() + ' (% price)' for s in spr]
daily_df[rel] = daily_df[spr].div(daily_df['mid'], axis=0) # scale spreads
result = 100*groupby[rel].mean()
formats.update({k: '{:.4f}' for k in result.columns})
results.update(result)
```

```
# summarize large and small trade effective spreads
spr = ['large_spread', 'small_spread']
result = 100*groupby[spr].mean()
result.columns = ['Large trade (% spread)', 'Small trade (% spread)']
formats.update({k: '{:.4f}' for k in result.columns})
results.update(result)
```

```
spr = ['large_trades', 'small_trades']
result = groupby[spr].mean()
result.columns = ['Large trade (# trades)', 'Small trade (# trades)']
formats.update({k: '{:.0f}' for k in result.columns})
results.update(result)
```

```
spr = ['large_volume', 'small_volume']
result = groupby[spr].mean()
result.columns = ['Large trade (avg volume)', 'Small trade (avg volume)']
formats.update({k: '{:.0f}' for k in result.columns})
results.update(result)
```

```
# display table of results
print("Average Liquidity by Market Cap")
row_formatted(DataFrame(results).T, formats)
```

Average Liquidity by Market Cap

Size	large	medium	small	tiny
Number of Stock/Days	2063	2572	4297	4618
Midquote Price	128.98	64.91	27.76	7.92
VWAP	128.97	64.92	27.75	7.90
Number of trades	25178	8407	3658	837
Volume (shares)	3031056	995483	533736	254773
Volatility(trade price)	0.0145	0.0211	0.0359	0.0807
Volatility(midquote)	0.0152	0.0223	0.0355	0.0918
Volatility(HL trade price)	0.0176	0.0217	0.0335	0.0690
Volatility(HL midquote)	0.0144	0.0198	0.0288	0.0594
Volatility(OHLC trade price)	0.0184	0.0218	0.0324	0.0612
Volatility(OHLC midquote)	0.0138	0.0183	0.0255	0.0452
Offersize (lots)	8.8	9.8	11.9	13.6
Bidsize (lots)	8.9	17.0	14.1	16.5
Quoted \$ spread	0.0630	0.0672	0.0781	0.0841
Effective \$ spread	0.0379	0.0442	0.0406	0.0457
Impact \$ spread	0.0273	0.0244	0.0248	0.0186

(continues on next page)

(continued from previous page)

Realized \$ spread	0.0106	0.0199	0.0158	0.0270
Quoted (% price)	0.0350	0.0834	0.2500	1.1582
Effective (% price)	0.0200	0.0421	0.1287	0.6638
Impact (% price)	0.0165	0.0345	0.0904	0.2669
Realized (% price)	0.0036	0.0077	0.0384	0.3973
Large trade (% spread)	0.0191	0.0420	0.1310	0.6398
Small trade (% spread)	0.0190	0.0400	0.1304	0.6755
Large trade (# trades)	3133	1140	423	108
Small trade (# trades)	22045	7267	3236	729
Large trade (avg volume)	1734	1608	2635	2324
Small trade (avg volume)	61	57	71	85

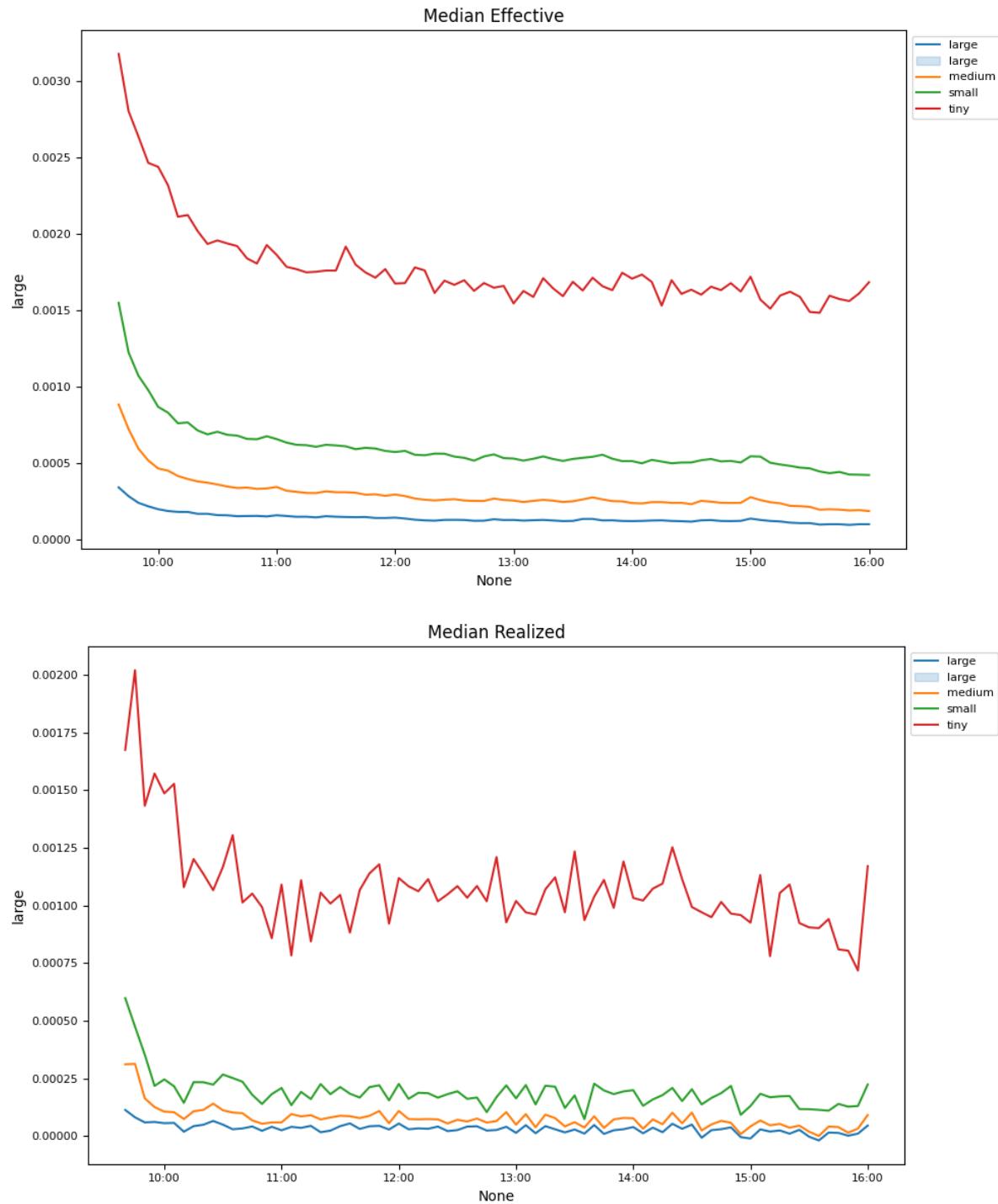
17.3.1 By time of day

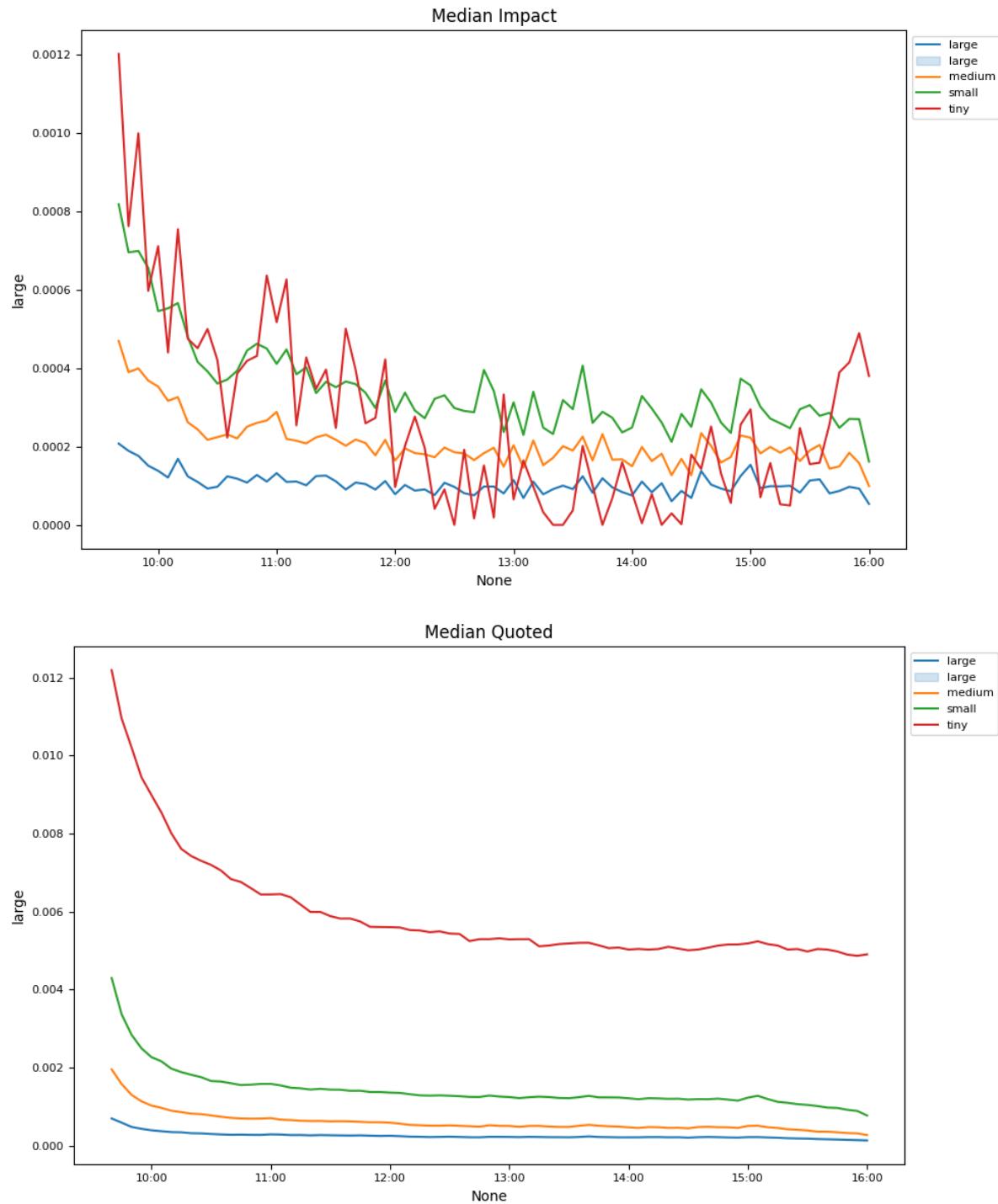
Market liquidity changes are examined over the trading day.

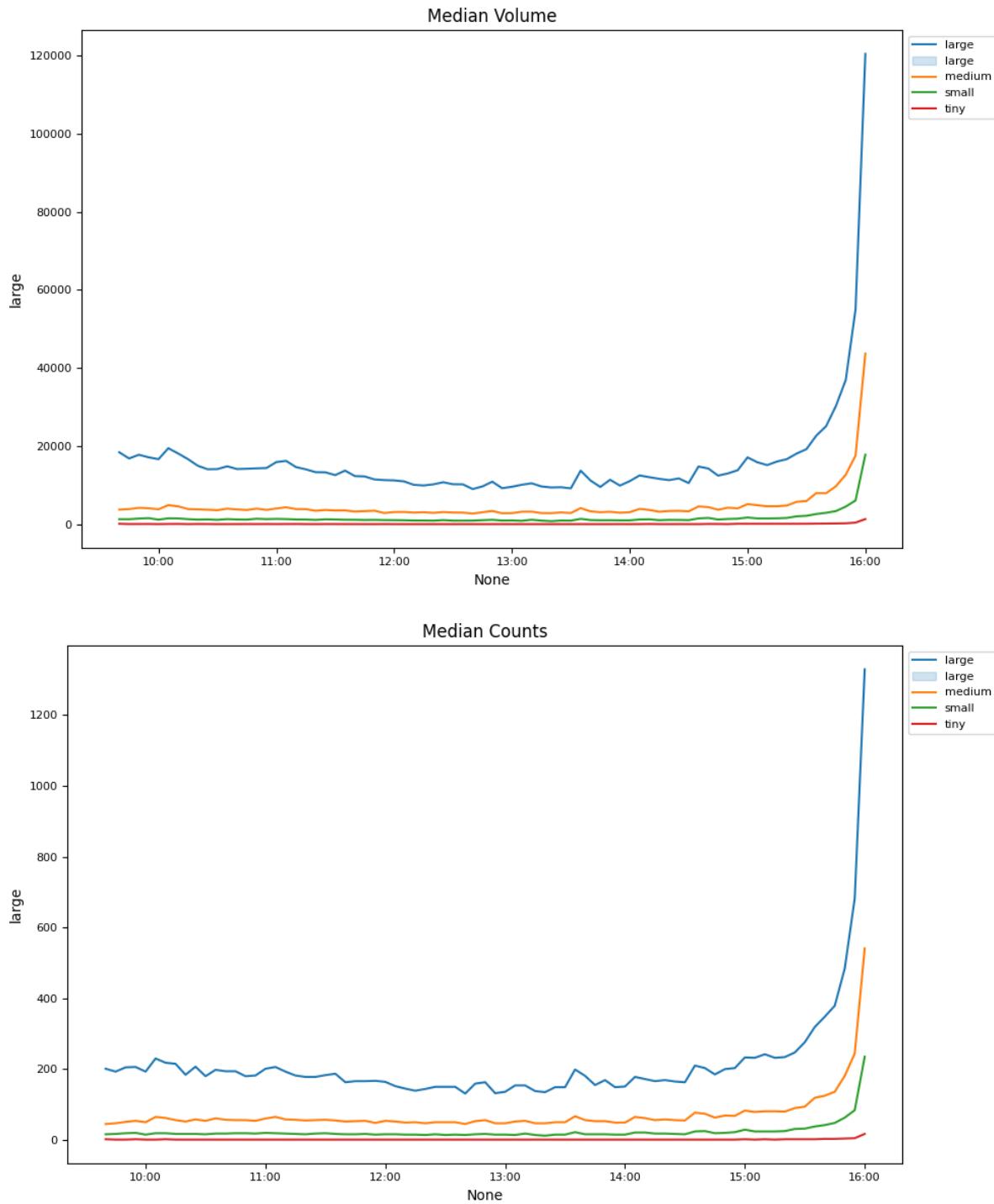
```
# Plot intraday spreads, depths and volumes
keys = ['effective', 'realized', 'impact', 'quoted',
        'volume', 'counts', 'offersize', 'bidsize']
for num, key in enumerate(keys):
    df = bins_df[key].drop(columns=['Round_Lot', 'Symbol'])
    df.index = list(zip(df['permno'], df['date']))

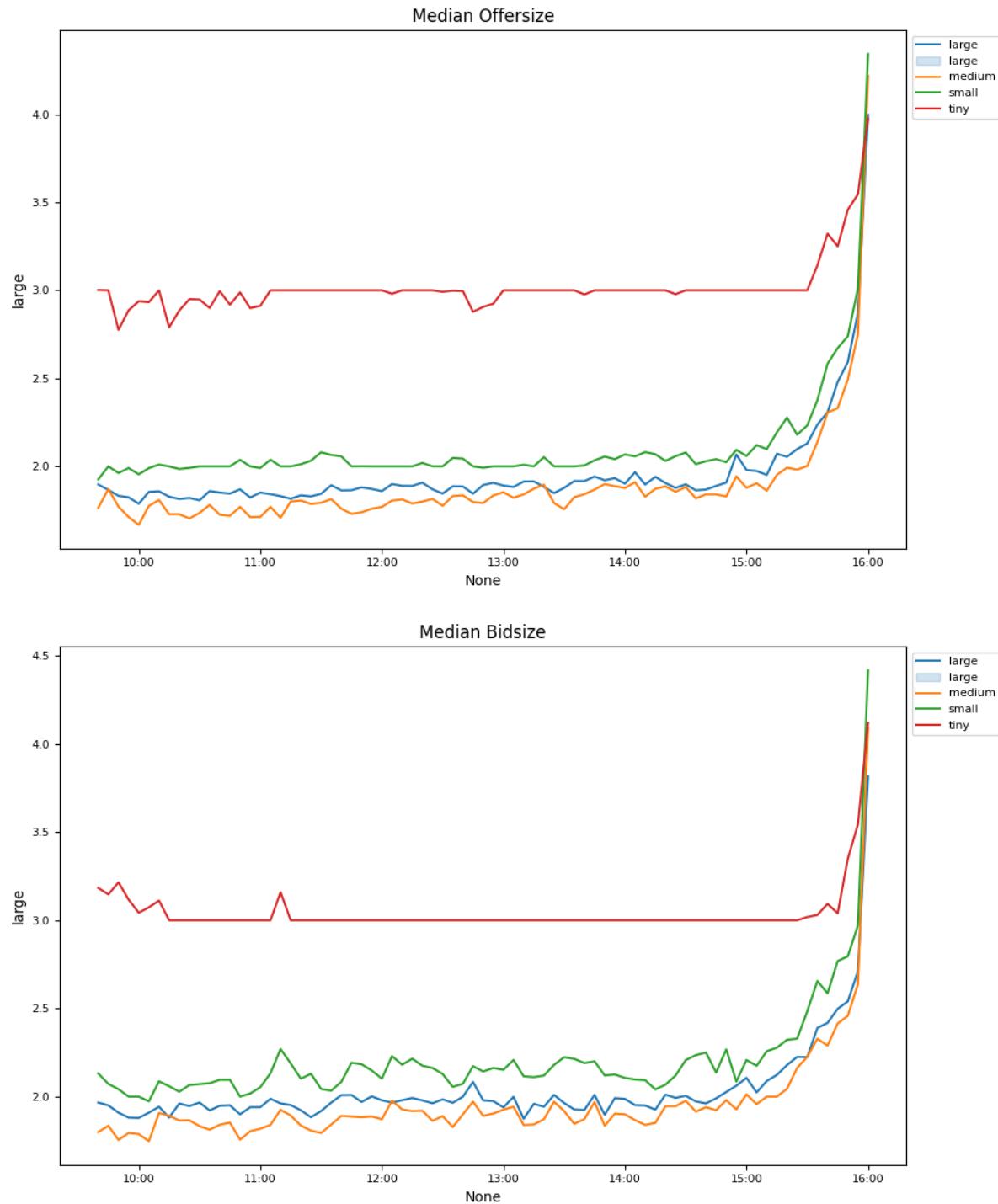
    # Group by market cap
    df['Size'] = pd.cut(df['decile'],
                         [0, 3.5, 6.5, 9.5, 11],
                         labels=['large', 'medium', 'small', 'tiny'])
    df = df.drop(columns=['date', 'permno', 'decile', 'exchcd', 'siccd']) \
        .dropna() \
        .groupby(['Size'], observed=False) \
        .median().T

    fig, ax = plt.subplots(1, 1, num=num+1, clear=True, figsize=(10, 6))
    plot_time(df.iloc[1:], ax=ax, fontsize=8)
    ax.legend(['large'] + list(df.columns),
              loc='upper left', bbox_to_anchor=(1.0, 1.0),
              fontsize=8)
    ax.set_title('Median ' + key.capitalize())
    plt.subplots_adjust(right=0.8)
    plt.tight_layout()
```









17.4 High frequency sampling

17.4.1 Variance ratio

Tick data often exhibits spurious autocorrelation due to irregularly spaced trades and quotes rather than continuous trading:

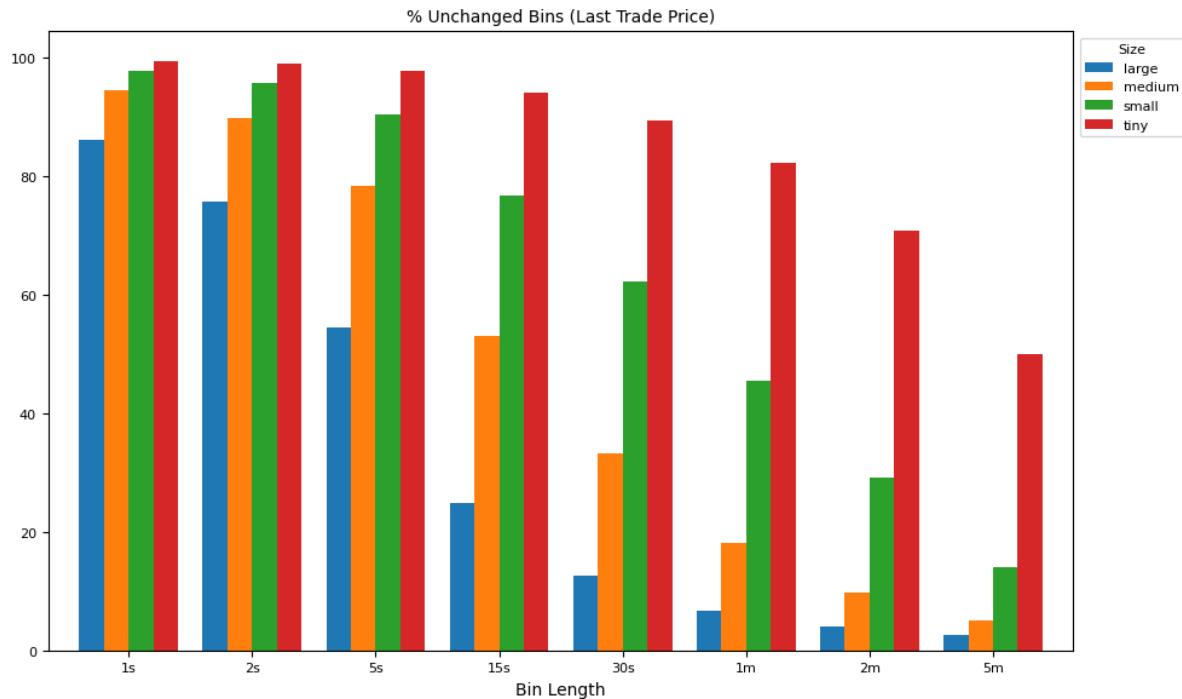
- Non-Continuous Trading: Trades and quotes occur discretely, often clustering around news or market events.
- Order Flow Clustering: Market participants submit bursts of orders, creating short-term price autocorrelation.
- Bid-Ask Bounce: Trades alternate between the bid and ask prices, creating an illusion of mean-reverting returns.

The variance ratio test (Lo & MacKinlay, 1988) checks for mean reversion or momentum by comparing **multi-period return variance to single-period return variance, scaled by the number of periods.

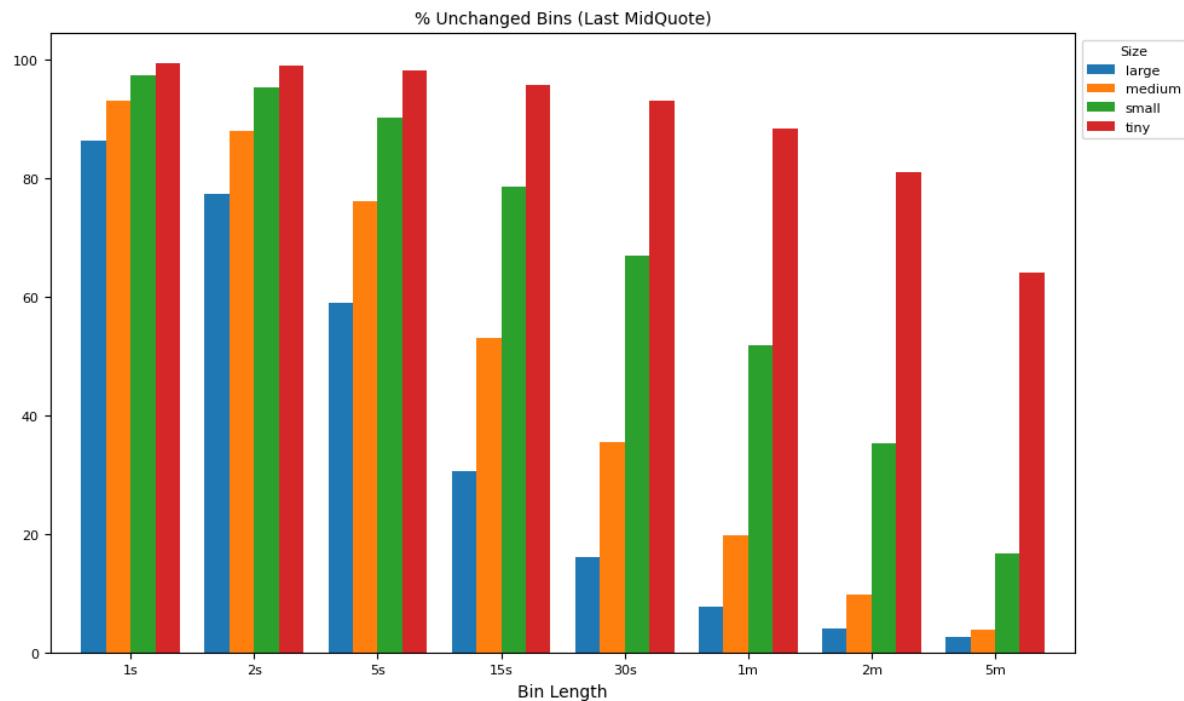
```
def plot_helper(result, xticks, keys, legend, xlabel, title, ylim=[], figsize=(10, 6), num=1, fontsize=8):
    """helper to plot bar graphs by sampling frequency"""
    fig, ax = plt.subplots(num=num, clear=True, figsize=figsize)
    result.plot(kind='bar',
                fontsize=fontsize,
                rot=0,
                width=0.8,
                xlabel='',
                ax=ax)
    if ylim:
        ax.set_ylim(*ylim)
    ax.set_xticklabels(xticks, fontsize=fontsize)
    ax.legend(keys, loc='upper left', bbox_to_anchor=(1.0, 1.0),
              fontsize=fontsize, title=legend, title_fontsize=8)
    ax.set_xlabel(xlabel, fontsize=fontsize + 2)
    ax.set_title(title, fontsize=fontsize + 2)
    plt.subplots_adjust(right=0.8, bottom=0.15)
    plt.tight_layout()
    return ax
```

```
xticks = [f"v{u}" for v, u in intervals]      # x-axis: bin lengths
keys = list(groupby.indices.keys())             # legend labels
```

```
labels = [f"tunch{v}{u}" for v, u in intervals]
result = groupby[labels].median()*100
ax = plot_helper(result.T,
                  title="% Unchanged Bins (Last Trade Price)",
                  xticks=xticks,
                  xlabel="Bin Length",
                  keys=keys,
                  legend='Size',
                  num=1)
```



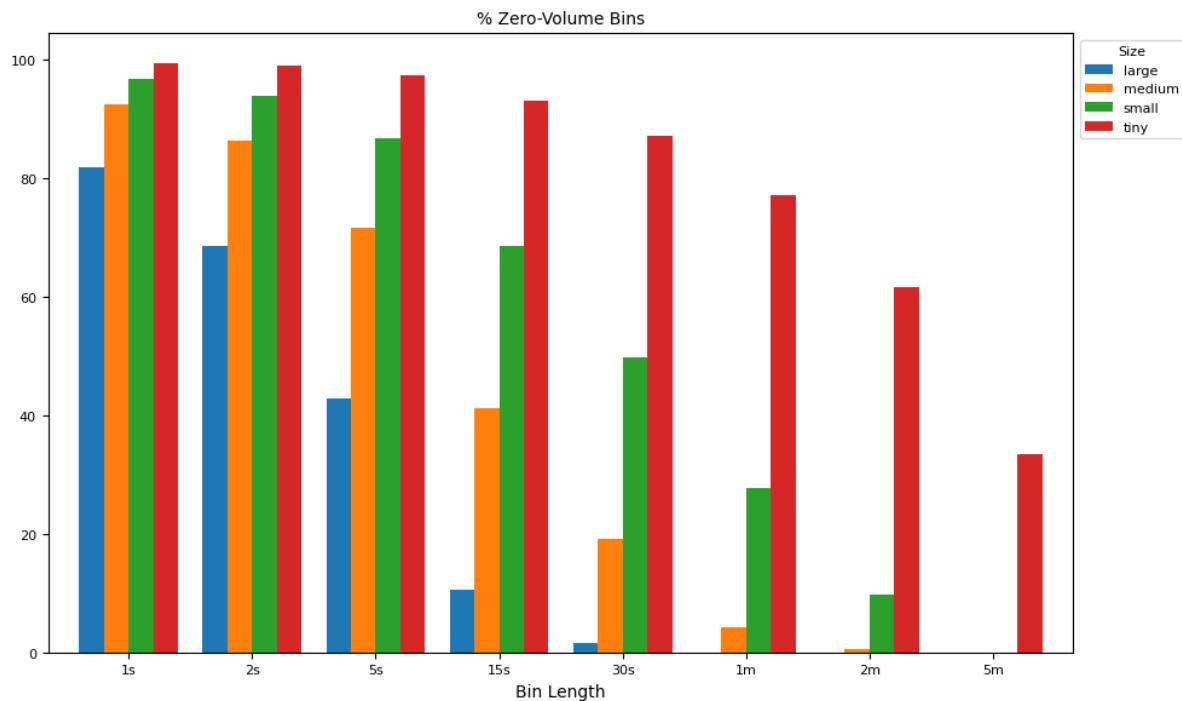
```
labels = [f"qunch{v}{u}" for v, u in intervals]
result = groupby[labels].median()*100
ax = plot_helper(result.T,
                  title="% Unchanged Bins (Last MidQuote)",
                  xticks=xticks,
                  xlabel="Bin Length",
                  keys=keys,
                  legend='Size',
                  num=2)
```



```

labels = [f"tzero{v}{u}" for v, u in intervals]
result = groupby[labels].median()*100
ax = plot_helper(result.T,
                  title="% Zero-Volume Bins",
                  xticks=xticks,
                  xlabel="Bin Length",
                  keys=keys,
                  legend='Size',
                  num=3)

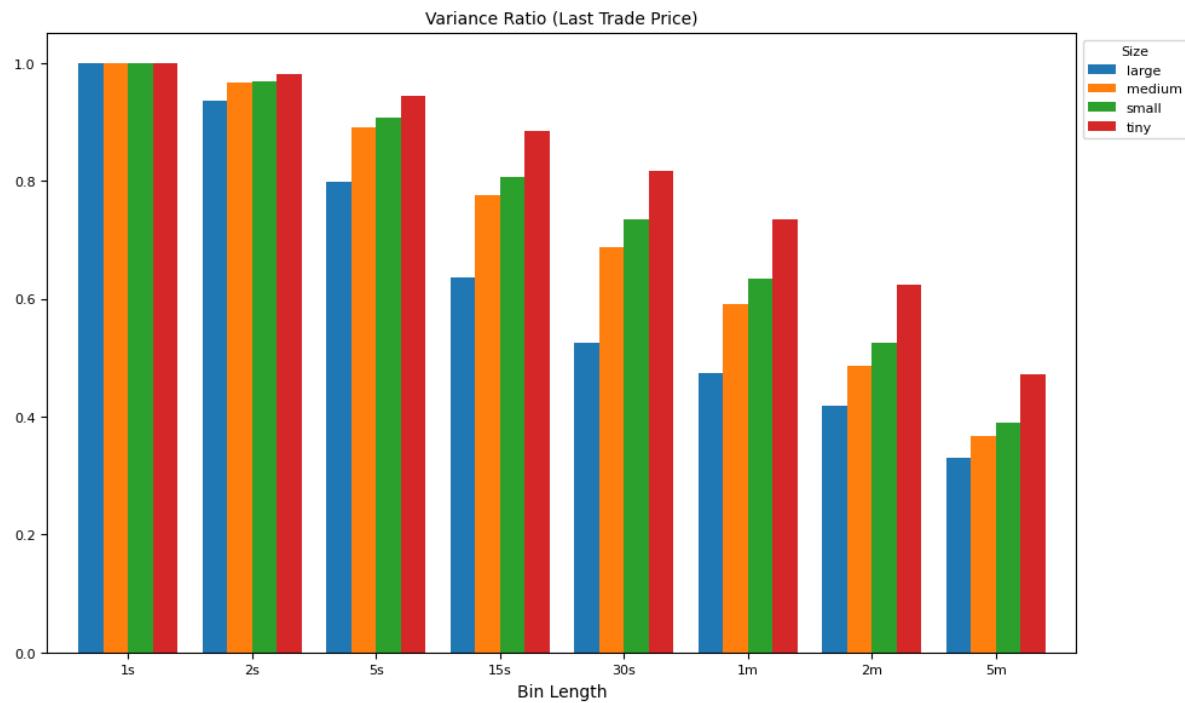
```



```

labels = [f"tvar{v}{u}" for v, u in intervals]
result = groupby[labels].median()
result = result.div(result['tvar1s'].values, axis=0)
ax = plot_helper(result.T,
                  title="Variance Ratio (Last Trade Price)",
                  xticks=xticks,
                  xlabel="Bin Length",
                  keys=keys,
                  legend='Size',
                  num=4)

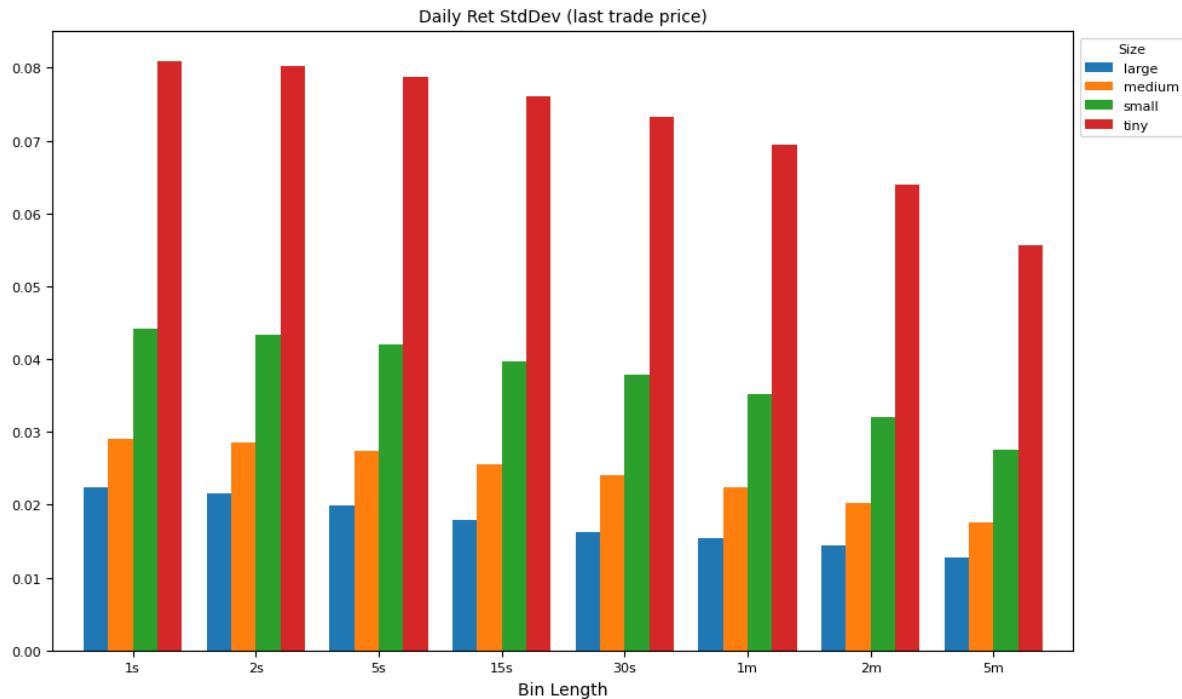
```



```

labels = [f"tvar{v}{u}" for v, u in intervals]
result = np.sqrt(groupby[labels].median())
ax = plot_helper(result.T,
                  title="Daily Ret StdDev (last trade price)",
                  xticks=xticks,
                  xlabel="Bin Length",
                  keys=keys,
                  legend='Size',
                  num=5)

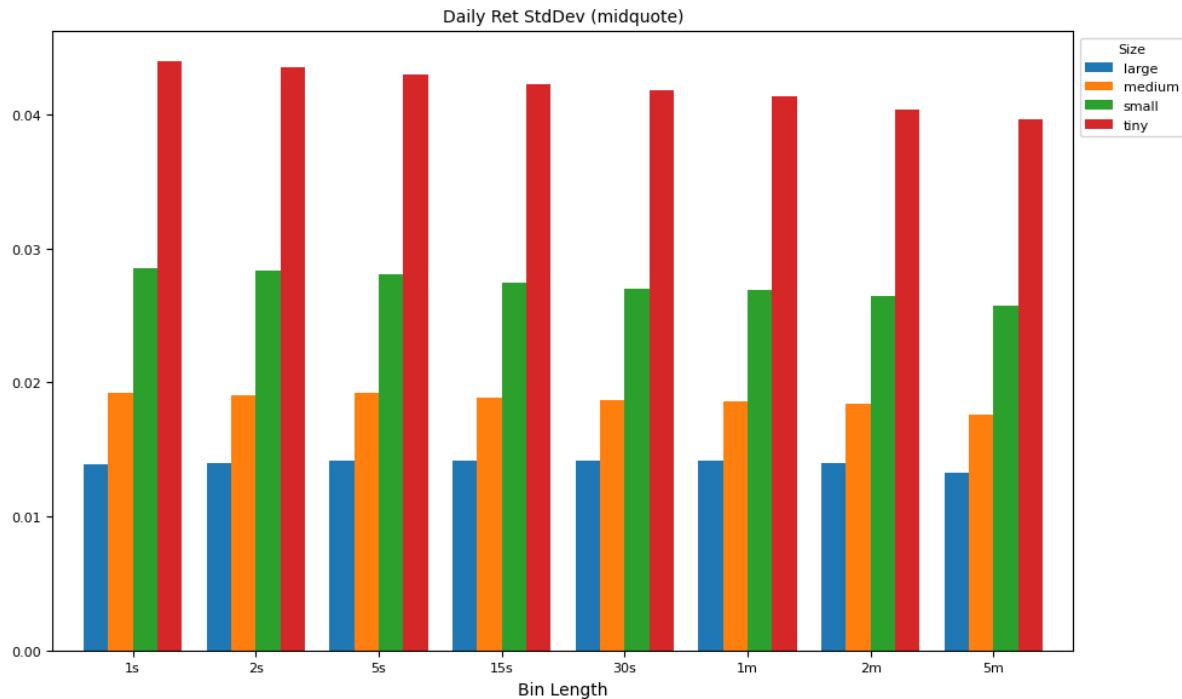
```



```

labels = [f"qvar{v}{u}" for v, u in intervals]
result = np.sqrt(groupby[labels].median())
ax = plot_helper(result.T,
                  title="Daily Ret StdDev (midquote)",
                  xticks=xticks,
                  xlabel="Bin Length",
                  keys=keys,
                  legend='Size',
                  num=6)

```



17.4.2 Volatility measures

Parkinson's (HL) Volatility estimator uses the high-low price range to estimate volatility, assuming a geometric Brownian motion without a drift term:

$$\sigma_{HL} = \frac{\sqrt{\ln 2}}{\sqrt{4T \ln 2}} \times \frac{H-L}{\sqrt{\Delta t}}$$

This method is more efficient than close-to-close volatility but assumes continuous trading and no jumps.

The **Klass-Garman (OHLC) Volatility** estimator improves on Parkinson's by incorporating open, high, low, and close prices for better accuracy:

$$\sigma_{OHLC}^2 = 0.511(H-L)^2 - 0.019(C-O)(H+L-2O) - 0.383(C-O)^2$$

This model accounts for both overnight jumps and intra-day movements.

```
# Compare methods of volatility estimates, by interval and market cap
for ifig, (split_label, split_df) in enumerate(groupby):
    vol_df = np.sqrt(split_df[[c for c in daily_df.columns if "qvar" in c or "tvar" in c]])
    result = []
    for col in [c for c in vol_df.columns if "qvar" in c or "tvar" in c]:
        if col[4] == 'H':
            m = 'HL'
        elif col[4] == 'O':
            m = 'OHLC'
        else:
            m = 'Close'
        result.append({'method': {'t': 'Last Trade', 'q': 'Mid Quote'}[col[0]] + ' ' +
                      m,
                      'interval': (int("".join(filter(str.isdigit, col))) *
                                   (60 if col[-1] == 'm' else 1)),
                      'val': vol_df[col].median()})
    ifig += 1
```

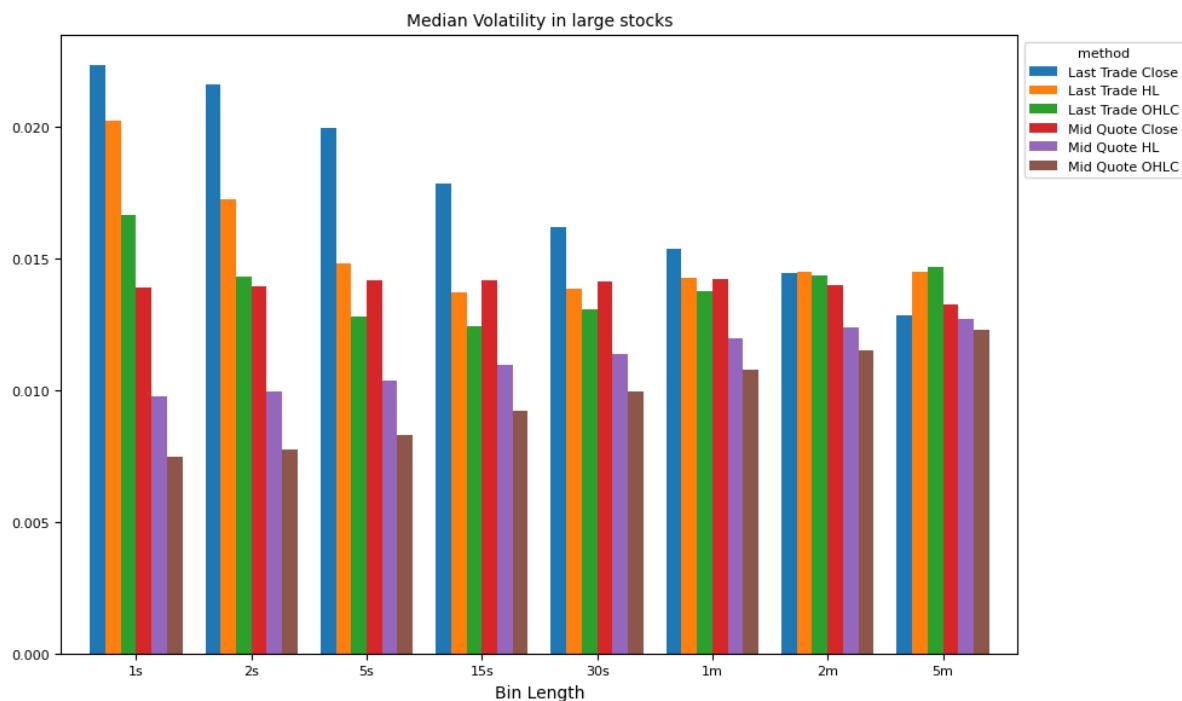
(continues on next page)

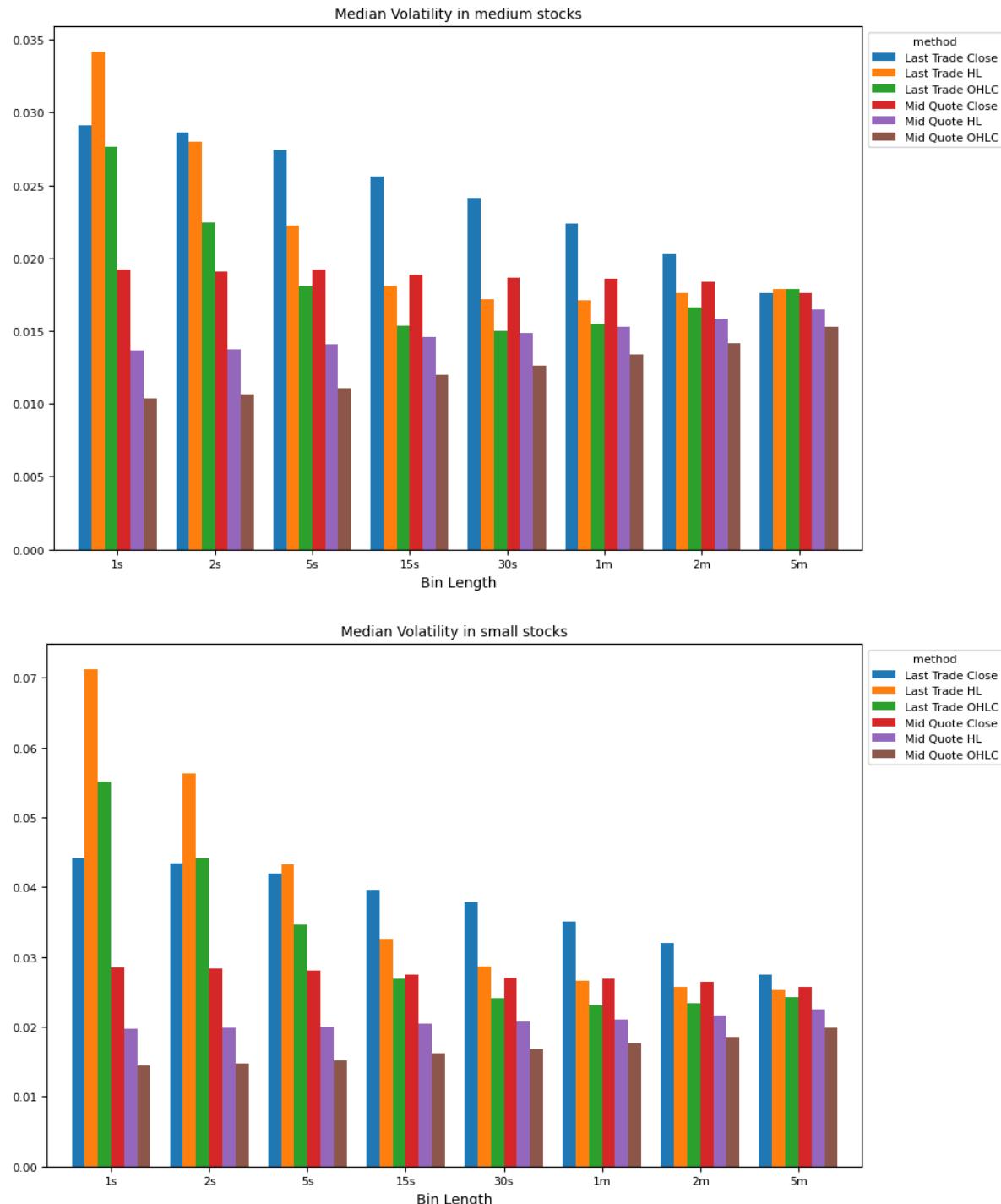
(continued from previous page)

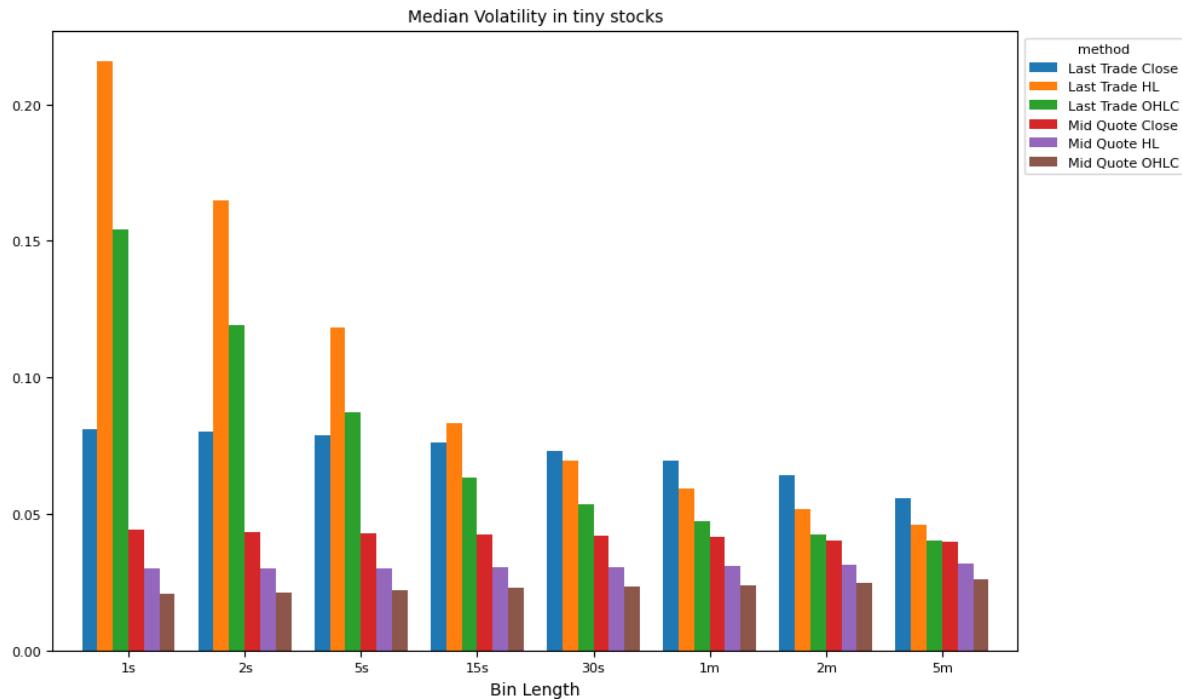
```

result = DataFrame.from_dict(result, orient='columns') \
    .pivot(index='interval', columns='method', values='val') \
    .sort_index()
ax = plot_helper(result,
                  title="Median Volatility in " + " ".join(split_label) + " stocks"
                  ,
                  xticks=xticks,
                  xlabel="Bin Length",
                  keys=result.columns,
                  num=iifig+1,
                  legend='method')

```







References:

Andrew W. Lo, A. Craig MacKinlay, Stock Market Prices Do Not Follow Random Walks: Evidence from a Simple Specification Test, The Review of Financial Studies, Volume 1, Issue 1, January 1988, Pages 41–66, <https://doi.org/10.1093/rfs/1.1.41>

Parkinson, M. (1980) The Extreme Value Method for Estimating the Variance of the Rate of Return. Journal of Business, 53, 61-65. <http://dx.doi.org/10.1086/296071>

EVENT RISK

Don't worry about failure; you only have to be right once — Drew Houston

Event risk refers to the potential for a sudden and significant impact on an asset's price or a portfolio's value due to an unforeseen event. These can be company-specific or broad market events that introduce volatility and uncertainty. For example, a company missing or exceeding analysts' earnings expectations can cause sharp price movements. Other types of event risk in finance include credit defaults and downgrades, mergers and acquisitions, or monetary policy changes. We explore how **Poisson regression**, an example of a **generalized linear model (GLM)** suited for count data, can be used to model the frequency of events such as earnings misses.

```
# By: Terence Lim, 2020-2025 (terence-lim.github.io)
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
from finds.database import SQL, RedisDB
from finds.structured import BusDay, SignalsFrame, CRSP, IBES
from finds.readers import Alfred
from secret import credentials, paths, CRSP_DATE
VERBOSE = 0
#%matplotlib qt
```

```
# open connections
sql = SQL(**credentials['sql'], verbose=VERBOSE)
user = SQL(**credentials['user'], verbose=VERBOSE)
rdb = RedisDB(**credentials['redis'])
bd = BusDay(sql, verbose=VERBOSE)
crsp = CRSP(sql, bd, rdb=rdb, verbose=VERBOSE)
ibes = IBES(sql, bd, verbose=VERBOSE)
LAST_DATE = CRSP_DATE
```

18.1 Earnings expectations

The consensus analyst estimate for a company's quarterly earnings is derived from the average of all the analyst forecasts covering that company. The number of analysts following a particular company typically correlates with the company's size and the liquidity of its stock. Large, well-known companies often have many analysts providing coverage, resulting in a consensus estimate derived from various opinions. In contrast, smaller or less-recognized companies may have only a few analysts covering them, or none at all.

When a company reports earnings that fall below the consensus expectation, this is referred to as an *earnings miss*. On the other hand, if the company's earnings exceed the analysts' expectations, it is called an *earnings beat*.

18.1.1 Standardized unexpected earnings (SUE)

To quantify earnings surprises, the Standardized Unexpected Earnings (SUE) is calculated. This is determined by subtracting the median of the analysts' estimates for a particular fiscal quarter from the company's reported earnings, then scaling it by the stock price.

```
# retrieve ibes Q1 where forecast period <= fiscal date, and keep latest
df = ibes.get_linked(dataset='statsum',
                      fields=['fpedats', 'medest', 'actual'],
                      date_field='statpers',
                      where=" fpi = '6' AND statpers <= fpedats")
```

```
summ = df.dropna() \
    .sort_values(['permno', 'fpedats', 'statpers']) \
    .drop_duplicates(['permno', 'fpedats'], keep='last')
summ['rebalddate'] = bd.endmo(summ['fpedats'])
summ = summ.set_index(['permno', 'statpers'])
```

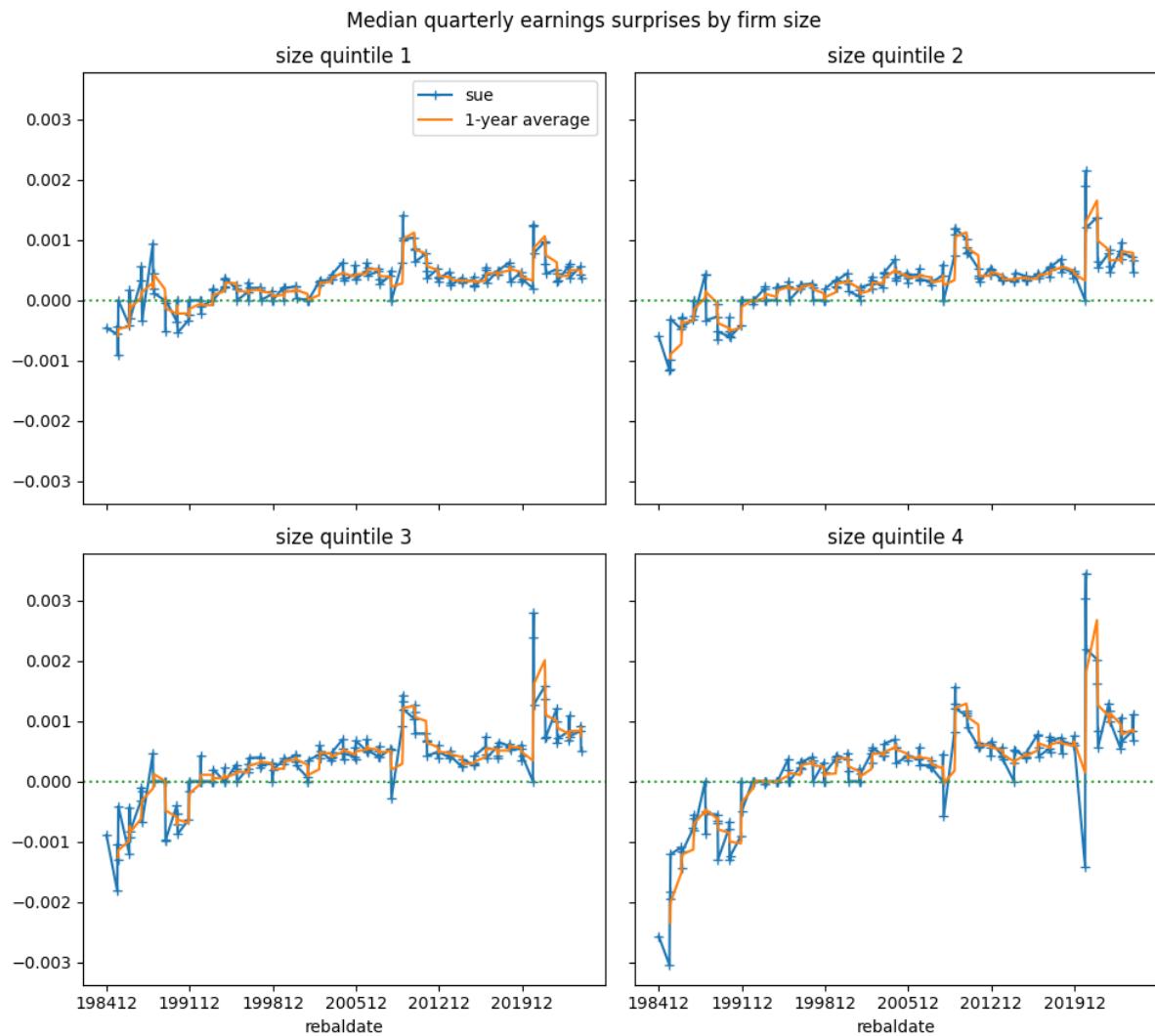
```
# retrieve ibes price
df = ibes.get_linked(dataset='actpsum',
                      fields=['price'],
                      date_field='statpers')
```

```
hist = df.dropna() \
    .sort_values(['permno', 'statpers']) \
    .drop_duplicates(['permno', 'statpers'], keep='last')
hist = hist.set_index(['permno', 'statpers'])
```

```
# calculate sue with ibes surprise and price
summ = summ.join(hist[['price']], how='inner')
summ['sue'] = (summ['actual'] - summ['medest']).div(summ['price'])
```

```
# define large earnings surprises as 5% of price
START = 19841201
signals = SignalsFrame(summ.reset_index(drop=False))
rebaldates = bd.date_range(START, LAST_DATE-100, freq='quarterly')
label = 'sue'
out = []
for rebaldate in rebaldates:
    univ = crsp.get_universe(rebaldate)      # get this quarter's universe
    univ = univ[(abs(univ['prc']) >= 5.0) & (univ['decile'] < 9)]  # no small low-
    ↪price
    signal = signals(label=label,
                      date=rebaldate,
                      start=bd.endqr(rebaldate, quarters=-1)) \
                      .reindex(univ.index) \
                      .dropna() \
                      .reset_index()
    signal['rebaldate'] = rebaldate // 100
    signal['miss'] = signal['sue'] < -0.05  # large earnings misses
    out.append(signal.set_index('permno').join(univ['decile'], how='inner') \
                      .reset_index())
out = pd.concat(out)
```

```
# median quarterly earnings surprise by firm size
fig, axes = plt.subplots(2, 2, figsize=(10, 9), sharex=True, sharey=True)
axes = axes.flatten()
for label, (ax, dec) in enumerate(zip(axes, [[1,2], [3,4], [5,6], [7,8]])):
    y = out[out['decile'].isin(dec)].groupby('rebalance') ['sue'].median()
    y.plot(ax=ax, marker='+', color="C0")
    y.rolling(4).mean().plot(ax=ax, color="C1")
    ax.set_title(f"size quintile {label+1}")
    if not label:
        ax.legend(['sue', '1-year average'])
    ax.axhline(0, color='C2', ls=':')
    ax.set_xticks(y.index[::28])
    ax.minorticks_off()
plt.suptitle('Median quarterly earnings surprises by firm size')
plt.tight_layout()
plt.show()
```



Historically, the median earnings surprise was negative (i.e. estimates were too optimistic), especially for smaller stocks, until the early-1990s, after which the trend generally shifted to positive earnings surprises

18.1.2 Earnings misses

We define earnings misses to be large negative earnings surprises exceeding 5% of stock price.

```
# count number of stocks with large earnings misses
frac = out.groupby('rebaldate')['miss'].mean().rename('frac')
count = out.groupby('rebaldate')['miss'].sum().rename('count')
exposures = out['rebaldate'].value_counts().sort_index().rename('exposures')
Y = pd.concat([exposures, frac, count], axis=1)
Y.index = Y.index.astype(str)
print("Earnings misses")
Y
```

Earnings misses

rebaldate	exposures	frac	count
198412	1327	0.051997	69
198503	1302	0.038402	50
198506	1384	0.040462	56
198509	1384	0.033960	47
198512	1398	0.063662	89
...
202309	1845	0.008672	16
202312	1812	0.009934	18
202403	1764	0.003401	6
202406	1776	0.004505	8
202409	1737	0.005757	10

[160 rows x 3 columns]

18.2 Poisson regression

The Poisson distribution is often used to model count data, or events that occur in a fixed period of time or space. Poisson regression models the expected count of events as a function of certain covariates. Specifically, it is expressed as:

$$\log(E[Y|X]) = \beta X$$

which is equivalent to:

$$E[Y|X] = e^{\beta X}$$

In this model, the interpretation of the coefficient β is that an increase in X_i by one unit is associated with a multiplicative change in the expected value of Y_i by a factor of e^{β_i} . The variance of Y_i under the Poisson model is equal to its expected value, i.e., $Var[Y_i] = E[Y_i]$. Importantly, the Poisson model ensures that the fitted values are non-negative, as it only allows for counts of zero or more.

Poisson regression can also be useful for modeling rates, where the event count is divided by some measure of exposure. For example, when modeling the number of stocks with large earnings misses each quarter, the exposure would be the total number of stocks, denoted N . The log of the total number of stocks, called the *offset* variable, is included in the regression equation with a coefficient constrained to 1.

The modified equation would then look like:

$$\log \left[\frac{E[Y|X]}{\text{exposure}} \right] = X\beta - \log(\text{exposure})$$

18.2.1 Generalized Linear Models

Linear and Poisson regression models are both types of Generalized Linear Models (GLMs). GLMs are a class of models where the response variable is modeled as a function of the predictors through a link function. Specifically, the mean of the response variable Y_i is transformed using the link function, which ensures the mean of the response is related linearly to the predictors.

For linear regression, the link function is the identity function, which simply states that the transformed mean is equal to the expected value: $\nu(E[Y]) = E[Y]$. In Poisson regression, the link function is the log function, so $\nu(E[Y]) = \log(E[Y])$, which transforms the expected count to the log scale. Specifically, Poisson regression models the response as coming from the Poisson distribution (with support $0, 1, 2, \dots$), with its canonical link function being the log function $X\beta = \log E[Y]$, and the mean function being the exponential $E[Y] = e^{X\beta}$.

Both linear and Poisson regression are based on the assumption that, conditional on X , the response variable Y comes from a member of the exponential family of distributions. The exponential family includes both discrete and continuous distributions such as the Gaussian, Poisson, Bernoulli, Multinomial, Exponential, Gamma, and Negative Binomial distributions. A GLM links the transformed mean of the distribution to the predictors, enabling flexible modeling of different types of data.

```
# retrieve and transform economics series
alf = Alfred(api_key=credentials['fred']['api_key'], verbose=VERBOSE)
```

```
X = pd.concat([alf(s, log=1, diff=1, freq='Q') for s in ['INDPRO', 'NASDAQCOM']], axis=1).dropna()
X.index = (X.index // 100).astype(str)
X
```

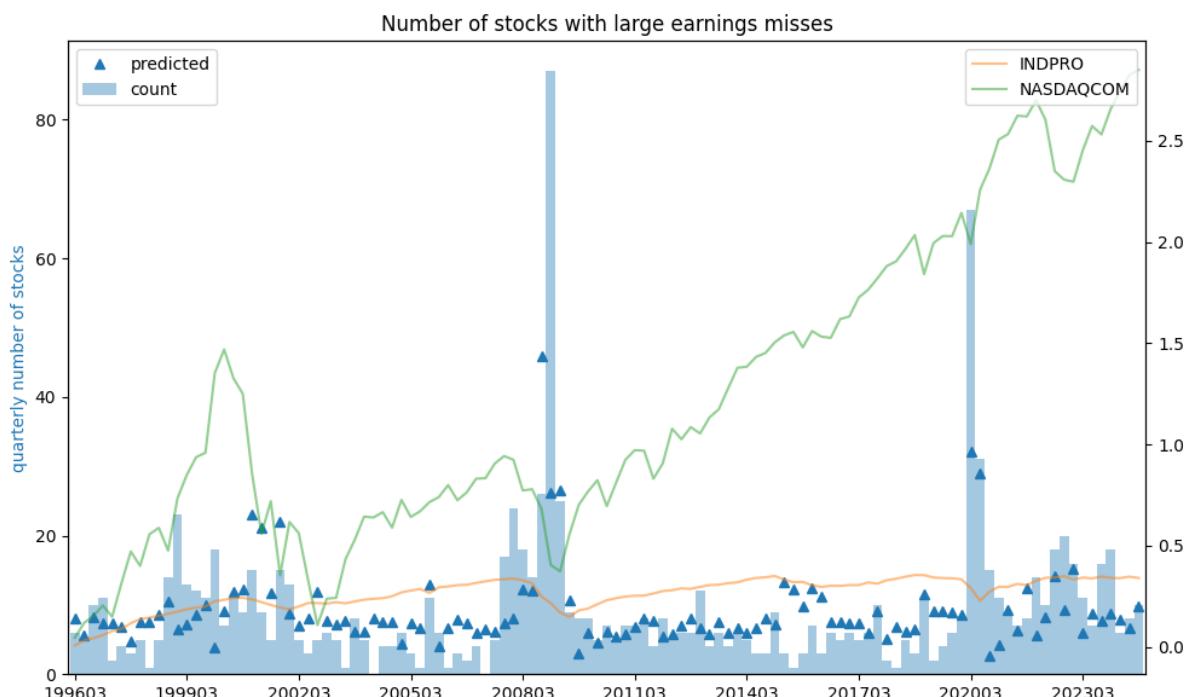
	INDPRO	NASDAQCOM
date		
197106	0.014847	0.017122
197109	0.007394	0.011345
197112	0.023171	0.045627
197203	0.040696	0.115873
197206	0.012599	0.015026
...
202403	-0.001095	0.087222
202406	0.007142	0.079377
202409	-0.006472	0.025422
202412	0.003853	0.059838
202503	0.005111	-0.024295

[216 rows x 2 columns]

```
# Run GLM regression with Poisson family and Log link
Z = Y.loc[Y.index > '199512'].join(X, how='inner')
Z['const'] = 1
glm = sm.GLM(exog=Z[['const'] + list(X.columns)],
              endog=Z['count'],
              family=sm.families.Poisson(link=sm.families.links.Log()),
              exposure=Z['exposures'])\
              .fit()
y_pred = glm.predict(exog=Z[['const'] + list(X.columns)],
                     exposure=Z['exposures']).rename('predicted')
print(glm.summary())
```

Generalized Linear Model Regression Results						
Dep. Variable:	count	No. Observations:	115			
Model:	GLM	Df Residuals:	112			
Model Family:	Poisson	Df Model:	2			
Link Function:	Log	Scale:	1.0000			
Method:	IRLS	Log-Likelihood:	-473.08			
Date:	Mon, 03 Mar 2025	Deviance:	519.46			
Time:	21:57:30	Pearson chi2:	622.			
No. Iterations:	5	Pseudo R-squ. (CS):	0.9417			
Covariance Type:	nonrobust					
coef	std err	z	P> z	[0.025	0.975]	
const	-5.3070	0.032	-167.739	0.000	-5.369	-5.245
INDPRO	-24.3445	1.378	-17.665	0.000	-27.046	-21.643
NASDAQCOM	-1.4022	0.209	-6.724	0.000	-1.811	-0.994

```
# Plot predicted and predictors
fig, ax = plt.subplots(figsize=(10, 6))
bx = ax.twinx()
Z['count'].plot(kind='bar', width=1.0, alpha=0.4, color="C0", ax=ax)
y_pred.plot(ls=' ', marker='^', color="C0", ax=ax)
ax.set_ylabel('quarterly number of stocks', color="C0")
Z['INDPRO'].cumsum().plot(color="C1", alpha=.5, ax=bx)
Z['NASDAQCOM'].cumsum().plot(color="C2", alpha=.5, ax=bx)
ax.legend(loc='upper left')
bx.legend(loc='upper right')
ax.set_xticks(np.arange(0, len(Z), 12), Z.index[::12])
ax.set_title('Number of stocks with large earnings misses')
plt.tight_layout()
```



Credit losses

Suppose a financial institution has a portfolio N loans, and:

- **default risk** p_i is the probability of default of the i th loan
- **loss severity** or loss given default L_i is the portion of the i th loan lost in the event of default. This is often assumed known with certainty

Then Expected loss is the sum of Default Probability \times Loss given default over all loans = $\sum_{i=1}^N p_i L_i$

The standard deviation of the expected loss on an individual loan, by applying the “Bernoulli shortcut” is $\sigma_i = \sqrt{p_i(1-p_i)L_i}$.

The standard deviation of the loss of the portfolio depends on the correlation of defaults between loans $\sigma_P = \sqrt{\sum_i \sum_j \rho_{ij} \sigma_i \sigma_j}$. For tractability, the correlations may be simplified to be constant or determined by a (Gaussian) copula, though neither assumption suffices to model real markets.

Actuarial Loss:

- Severity
- Frequency

References:

<https://sites.google.com/site/zoubin019/teaching/math-5639-actuarial-loss-models>

Lim, Terence, 2001, “Rationality and Analysts’ Forecast Bias”, Journal of Finance, Volume 56, Issue 1, Pages 369-385.
<https://doi.org/10.1111/0022-1082.00329>

SUPPLY CHAIN NETWORK GRAPHS

Forget about your competitors, just focus on your customers - Jack Ma

Supply chain networks comprise the interconnected relationships between suppliers and their customers. An effective way to study these relationships is by constructing graphs that visually and mathematically represent supply chain structures. We introduce key properties of graphical networks, such as degree distribution, clustering coefficients, and path lengths, and explore how to identify central and connected firms.

```
# By: Terence Lim, 2020-2025 (terence-lim.github.io)
import math
import numpy as np
import pandas as pd
import networkx as nx
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
from finds.database import SQL
from finds.recipes import graph_draw
from secret import credentials
import warnings
# %matplotlib qt
VERBOSE = 0
if not VERBOSE:
    warnings.simplefilter(action='ignore', category=FutureWarning)
sql = SQL(**credentials['sql'], verbose=VERBOSE)
```

19.1 Principal customers

The relationships between upstream suppliers and downstream customers, collectively known as “supply chain” relationships, form a crucial component of economic linkages, representing the movement of goods, services, and financial transactions across industries and individual firms.

Principal customers are those that contribute more than 10% of a firm’s total sales, as defined by Regulation S-K and SFAS 131. Under these disclosure requirements, suppliers are often significantly smaller than their corporate principal customers.

Retrieve principal customer relationships from the database:

```
# retrieve principal customers info
year = 2022
cust = sql.read_dataframe(f"select gvkey, cgvkey, stic, ctic, conm, cconnm "
                           f"  from customer "
```

(continues on next page)

(continued from previous page)

```
f"      where srcdate >= {year}0101 "
f"      and srcdate <= {year}1231")
```

Construct a lookup table to map company tickers to their full names:

```
# construct Series to lookup company full name from ticker
lookup = pd.concat([Series(cust['conm'].values, cust['stic'].values),
                    Series(cust['cconm'].values, cust['ctic'].values)]) \
    .drop_duplicates()
```

19.2 Graph properties

In a **graph**, **nodes** (also called vertices) represent entities, while **edges** denote relationships between them. The **degree** of a node is the number of edges directly connected to it.

Density measures the proportion of existing edges relative to the maximum possible edges:

- In undirected graphs: $\frac{2m}{n(n-1)}$
- In directed graphs: $\frac{m}{n(n-1)}$

where m is the number of edges and n is the number of nodes.

A **simple graph** has no self-loops (edges connecting a node to itself) and does not contain multiple edges between the same pair of nodes.

A **subgraph** is a subset of nodes and the edges that connect them. A **component** is a subgraph in which every node is connected to every other node by some path.

Key graph attributes include:

- **Weighted vs. Unweighted:** Indicates whether numerical values are assigned to edges.
- **Directed vs. Undirected:** Determines whether edges have a defined direction.
- **Cyclic vs. Acyclic:** A cyclic graph contains at least one cycle (a path that starts and ends at the same node), while an acyclic graph does not.
- **Connected vs. Disconnected:** A graph is connected if a path exists between any two nodes.
- **Weakly vs. Strongly Connected Components:** Weakly connected components have a path between every pair of nodes when edge direction is ignored, whereas strongly connected components require directed paths between every pair of nodes.

To model supply chains, we construct a directed graph with edges pointing from suppliers to their principal customers and analyze its structural properties.

```
# nodes are companies, with directed edges from supplier to customer
vertices = set(cust['stic']).union(cust['ctic'])
edges = cust[['stic', 'ctic']].values.tolist() # supplier --> customer
```

```
# Populate networkx directed graph with nodes and edges
DG = nx.DiGraph()
DG.add_nodes_from(vertices)
DG.add_edges_from(edges)
```

Display graph properties:

```

# Helper to display graph properties
def graph_info(G):
    out = dict()
    out['is_directed'] = nx.is_directed(G)
    out['num_edges'] = nx.number_of_edges(G)
    out['num_nodes'] = nx.number_of_nodes(G)
    out['num_selfloops'] = nx.number_of_selfloops(G)
    out['density'] = nx.density(G)

    out['is_weighted'] = nx.is_weighted(G)
    if nx.is_weighted(G):
        out['is_negatively_weighted'] = nx.is_negatively_weighted(G)

    # Components
    if nx.is_directed(G):
        out['is_weakly_connected'] = nx.is_weakly_connected(G)
        out['weakly_connected_components'] = nx.number_weakly_connected_components(G)
        out['size_largest_weak_component'] = len(max(
            nx.weakly_connected_components(G), key=len))
        out['is_strongly_connected'] = nx.is_strongly_connected(G)
        out['strongly_connected_components'] = nx.number_strongly_connected_
        ↪components(G)
        out['size_largest_strong_component'] = len(max(
            nx.strongly_connected_components(G), key=len))
    else:
        out['is_connected'] = nx.is_connected(G)
        out['connected_components'] = nx.number_connected_components(G)
        out['size_largest_component'] = len(max(
            nx.connected_components(G), key=len))
    return out

```

```
Series(graph_info(DG)).rename('Principal Customers Graph').to_frame()
```

Principal Customers Graph	
is_directed	True
num_edges	2561
num_nodes	1696
num_selfloops	5
density	0.000891
is_weighted	False
is_weakly_connected	False
weakly_connected_components	111
size_largest_weak_component	1398
is_strongly_connected	False
strongly_connected_components	1692
size_largest_strong_component	3

```

# remove self-loops, if any
DG.remove_edges_from(nx.selfloop_edges(DG))

```

19.2.1 Clustering coefficient

The clustering coefficient measures how nodes in a graph tend to cluster together.

- A **triplet** consists of three nodes connected by two (open triplet) or three (closed triplet) edges.
- A **triangle** comprises three closed triplets.
- The **clustering coefficient** of a node is given by:

$$\frac{2 \times \# \text{ of triangles}}{\text{degree} \times (\text{degree} - 1)}$$

- **Graph transitivity** is the fraction of all possible triangles that are actually present: $3 \times \frac{\# \text{ of triangles}}{\# \text{ of triads}}$

```
# as an undirected graph
G = nx.Graph(DG)
DataFrame({'transitivity': nx.transitivity(G),
           'average clustering': nx.average_clustering(G)},
          index=['clustering'])
```

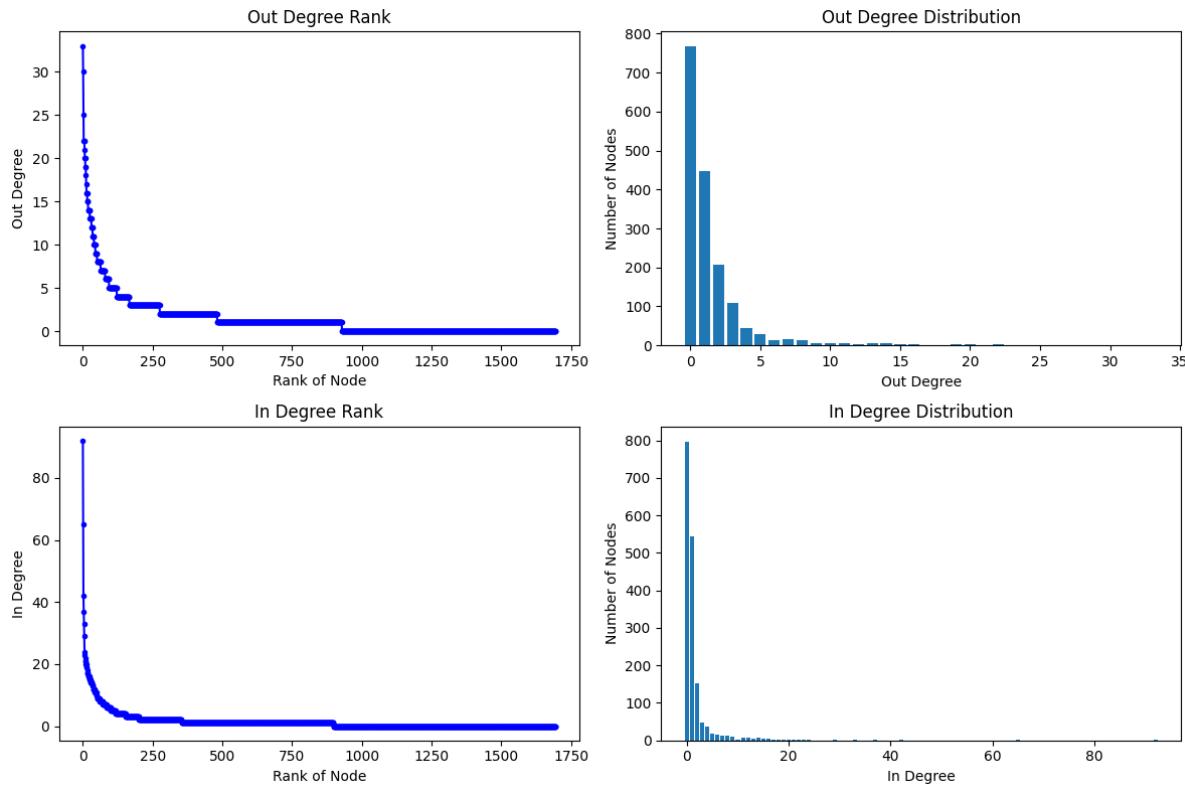
	transitivity	average clustering
clustering	0.007702	0.011649

19.2.2 Degree analysis

In directed graphs:

- **Out-degree** represents the number of edges pointing out from a node.
- **In-degree** represents the number of edges pointing into a node.

```
# Plot distribution of degrees
degrees = {'Out Degree': sorted((d for n, d in DG.out_degree()), reverse=True),
           'In Degree': sorted((d for n, d in DG.in_degree()), reverse=True)}
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(12, 8))
for direction in zip(axes, degrees):
    ax[0].plot(degrees[direction], "b-", marker=".")
    ax[0].set_title(f"{direction} Rank")
    ax[0].set_ylabel(f"{direction}")
    ax[0].set_xlabel("Rank of Node")
    ax[1].bar(*np.unique(degrees[direction], return_counts=True))
    ax[1].set_title(f"{direction} Distribution")
    ax[1].set_xlabel(f"{direction}")
    ax[1].set_ylabel("Number of Nodes")
plt.tight_layout()
plt.show()
```



19.2.3 Ego network

An **ego** is a focal node in a network. Each node in a network can serve as an ego. A **neighborhood** consists of the ego and all nodes directly connected to it.

For analysis, we select the node with the highest degree as the focal node and construct its one-step neighborhood, capturing immediate suppliers and principal customers.

```
# find node with greatest degree
ego, degree = max(G.degree(), key=lambda x: x[1])

# build subgraph of ego and neighbors
all_neighbors = list(nx.all_neighbors(DG, ego)) # predecessors and successors
neighbors = list(nx.neighbors(DG, ego)) # successors only
ego_graph = DG.subgraph([ego] + all_neighbors).copy()

# Plot ego graph
node_color = (dict.fromkeys(all_neighbors, 'b') # plot predecessor (supplier) nodes
              | dict.fromkeys(neighbors, 'g') # plot successor (customer) nodes
              | {ego: 'cyan'}) # plot ego node

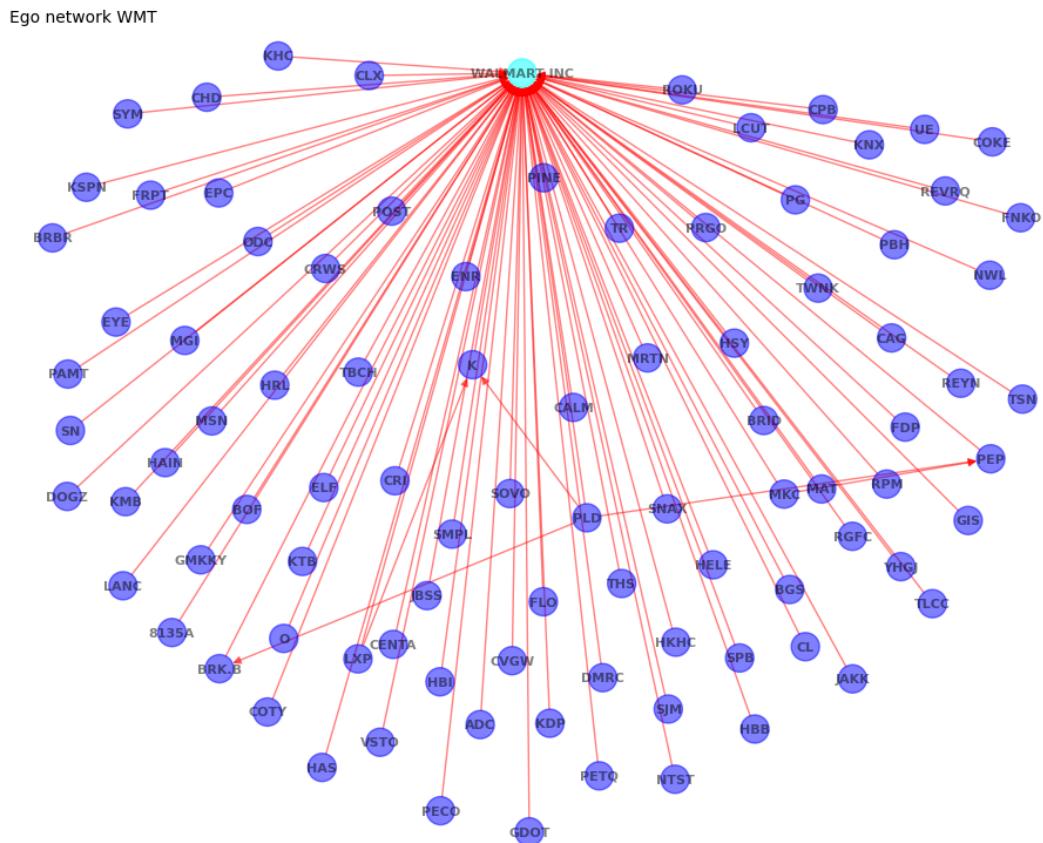
labels = {ticker: ticker for ticker in ego_graph.nodes}
          | {ticker: lookup[ticker] for ticker in [ego] + neighbors})

graph_draw(ego_graph, figsize=(10, 8), node_size=300, seed=42,
```

(continues on next page)

(continued from previous page)

```
width=1, node_color=node_color, labels=labels, style='-'  
title=f"Ego network {ego}")  
plt.show()
```



19.2.4 Path lengths

The **distance** or **path length** between two nodes is the shortest path (fewest number of edges) between them.

- **Eccentricity** of a node is the maximum distance from that node to any other node in the graph.
 - **Diameter** is the maximum eccentricity across all nodes in the graph.

To analyze path lengths, we identify the largest connected component of the graph and compute the longest undirected path within it.

```
# find component with the longest diameter
components = list(nx.weakly_connected_components(DG))
nodes = components[np.argmax([nx.diameter(G.subgraph(c)) for c in components])]
```

```
# compute all shortest path lengths, and determine the longest
best_length = 0
```

(continues on next page)

(continued from previous page)

```

for src, targets in dict(nx.shortest_path(G.subgraph(nodes))).items():
    for tgt, path in targets.items():
        length = len(path)
        if length > best_length:
            best_length = length
            best_path = path

```

```

# Archimedean spiral: r = b * theta
pos = {ticker: ((n + 5) * math.cos(n), (n + 5) * math.sin(n))
       for n, ticker in enumerate(best_path)}

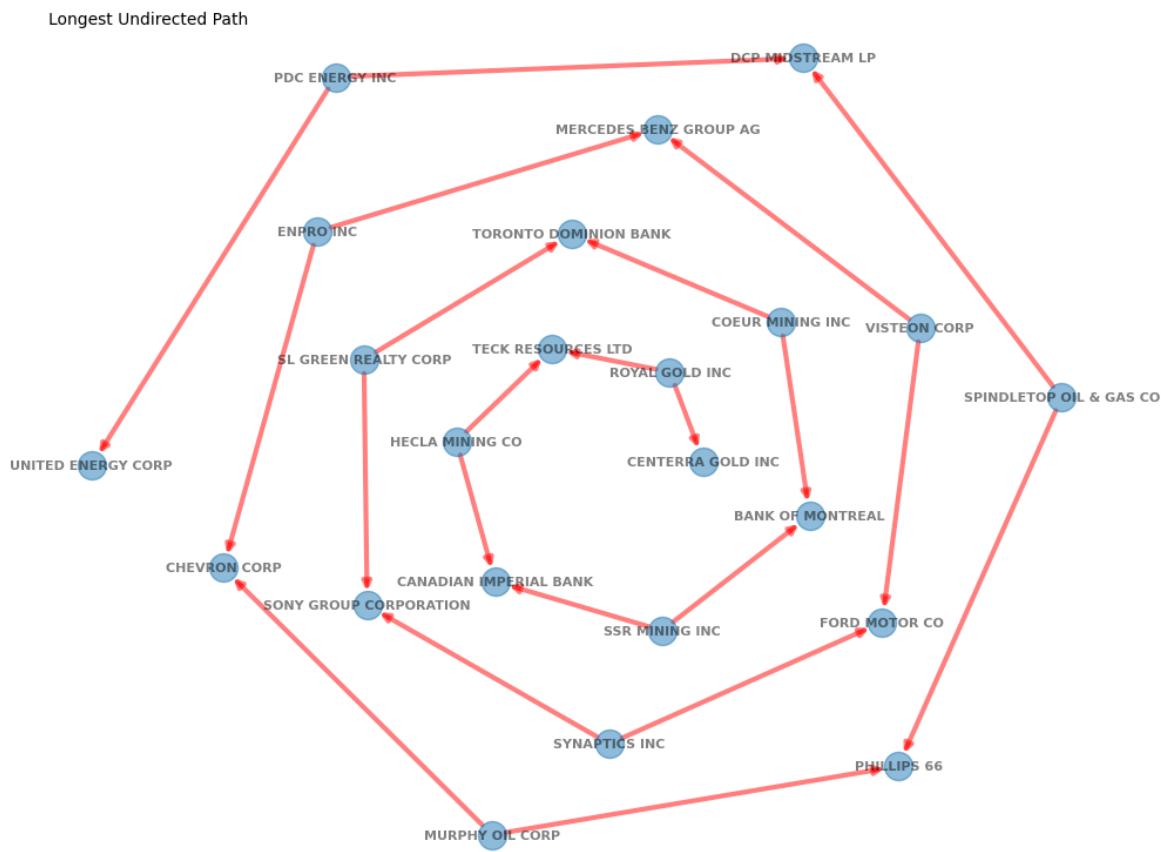
# Plot nodes in a spiral graph
labels = {ticker: lookup[ticker] for ticker in best_path}
graph_draw(DG.subgraph(best_path), figsize=(10, 8), node_size=300,
           width=3, labels=labels, style='-', pos=pos,
           title=f"Longest Undirected Path")

```

```

{'CGAU': (5.0, 0.0),
 'RGLD': (3.2418138352088386, 5.048825908847379),
 'TECK': (-2.9130278558299967, 6.365081987779772),
 'HL': (-7.919939972803563, 1.1289600644789377),
 'CM': (-5.882792587772507, -6.811222457771354),
 'SSRM': (2.8366218546322624, -9.589242746631385),
 'BMO': (10.561873153154025, -3.0735704801881845),
 'CDE': (9.046827052119655, 7.883839184625469),
 'TD': (-1.8915004395119759, 12.861657206103963),
 'SLG': (-12.755823666385478, 5.769658793384592),
 'SONY': (-12.586072936146786, -8.160316663340547),
 'SYNA': (0.07081116780881257, -15.999843304811256),
 'F': (14.345517298452366, -9.121739606007393),
 'VC': (16.33404206610353, 7.563006662879537),
 'MBGYY': (2.5980071459488383, 18.821539758202537),
 'NPO': (-15.193758257176427, 13.005756803142337),
 'CVX': (-20.110849086791077, -6.045969649966372),
 'MUR': (-6.053593437135133, -21.15074482135025),
 'PSX': (15.187284289613844, -17.272706675748548),
 'SPND': (23.72891083648006, 3.597053031910856),
 'DCP': (10.202051545334799, 22.82363126819069),
 'PDCE': (-14.240960765830978, 21.75304660193746),
 'UNRG': (-26.998942312655203, -0.23898535084090466)}

```



References:

- Ling Cen and Sudipto Dasgupta, 2021, The Economics and Finance of Customer-Supplier Relationships, Oxford Research Encyclopedia of Economics and Finance.
- Cen, Ling, et al. "Customer-supplier relationships and corporate tax avoidance." *Journal of Financial Economics* 123.2 (2017): 377-394.
- Cohen, Lauren, and Andrea Frazzini. "Economic links and predictable returns." *The Journal of Finance* 63.4 (2008): 1977-2011.

INDUSTRY COMMUNITY DETECTION

Realize that everything connects to everything else - Leonardo DaVinci

Traditional industry classification systems, such as SIC and NAICS, group firms based on production processes or product similarities. Natural language processing techniques can be leveraged to analyze product descriptions and capture dynamic changes in industry structures over time, as proposed by Hoberg and Phillips (2016). Industry communities can be detected through network analysis, where firms are modeled as nodes in a graph, and connections between them are determined by similarities in their product and market descriptions.

```
# By: Terence Lim, 2020-2025 (terence-lim.github.io)
import zipfile
import io
from itertools import chain
import numpy as np
import pandas as pd
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
import seaborn as sns
import networkx as nx
from finds.database import SQL
from finds.readers import requests_get, Sectoring
from finds.structured import BusDay, PSTAT
from finds.recipes import graph_info
from secret import credentials
# %matplotlib qt
VERBOSE = 0
```

```
sql = SQL(**credentials['sql'], verbose=VERBOSE)
bd = BusDay(sql)
pstat = PSTAT(sql, bd, verbose=VERBOSE)
```

20.1 Industry taxonomy

Industry classification, or industry taxonomy, organizes companies into groups based on shared characteristics such as production processes, product offerings, or financial market behaviors.

20.1.1 Text-based industry classification

Hoberg and Phillips (2016) developed a text-based measure of firm similarity by analyzing product descriptions in 10-K filings. They construct firm-by-firm similarity scores using word vectors, filtering out common words and focusing on nouns and proper nouns, while excluding geographic terms. The similarity between firms is quantified using cosine similarity, creating a pairwise similarity matrix across firms and years.

Since Item 101 of Regulation S-K mandates that firms accurately describe their key products in their 10-K filings, the TNIC scheme, based on textual similarity, provides a dynamic classification system that evolves with market changes. This method offers a more flexible alternative to traditional classification systems, capturing shifts in product markets over time.

Source: [Hoberg and Phillips Industry Classification](#)

The TNIC pair-wise firm similarities are retrieved from the Hoberg and Phillips website:

```
# Retrieve TNIC scheme from Hoberg and Phillips website
tnic_scheme = 'tnic3'
root = 'https://hobergphillips.tuck.dartmouth.edu/idata/'
source = root + tnic_scheme + '_data.zip'

if source.startswith('http'):
    response = requests_get(source)
    source = io.BytesIO(response.content)

# extract the csv file from zip archive
with zipfile.ZipFile(source).open(tnic_scheme + "_data.txt") as f:
    tnic_data = pd.read_csv(f, sep='\s+')

# extract latest year of tnic as data frame
year = max(tnic_data['year']) # [1989, 1999, 2009, 2019]
tnic = tnic_data[tnic_data['year'] == year].dropna()
tnic
```

	year	gvkey1	gvkey2	score
26307358	2023	1004	1823	0.0127
26307359	2023	1004	4091	0.0087
26307360	2023	1004	5567	0.0063
26307361	2023	1004	9698	0.0075
26307362	2023	1004	10519	0.0191
...
26973403	2023	351038	329141	0.0684
26973404	2023	351038	331856	0.0769
26973405	2023	351038	332115	0.1036
26973406	2023	351038	347007	0.0731
26973407	2023	351038	349972	0.0871

[666050 rows x 4 columns]

20.1.2 Industry classification

Industry classification systems such as SIC and NAICS follow hierarchical structures to categorize firms based on their economic activities:

- **Standard Industrial Classification (SIC):** Uses a 2-digit, 3-digit, and 4-digit hierarchy to classify industries.
- **North American Industry Classification System (NAICS):** Expands classification granularity from 2-digit to 6-digit levels.

```
# populate dataframe of nodes with gvkey (as index), permno, sic and naics
nodes = DataFrame(index=sorted(set(tnic['gvkey1']).union(tnic['gvkey2']))) \
    .rename_axis(index='gvkey')
for code in ['lpermno', 'sic', 'naics']:
    lookup = pstat.build_lookup('gvkey', code, fillna=0)
    nodes[code] = lookup(nodes.index)
Series(np.sum(nodes > 0, axis=0)).rename('Non-missing').to_frame().T
```

	lpermno	sic	naics
Non-missing	3829	3829	3827

```
# supplement naics and sic with crosswalks in Sectoring class
naics = Sectoring(sql, 'naics', fillna=0)    # supplement from crosswalk
sic = Sectoring(sql, 'sic', fillna=0)
nodes['naics'] = nodes['naics'].where(nodes['naics'] > 0, naics[nodes['sic']])
nodes['sic'] = nodes['sic'].where(nodes['sic'] > 0, sic[nodes['naics']])
Series(np.sum(nodes > 0, axis=0)).rename('Non-missing').to_frame().T
```

	lpermno	sic	naics
Non-missing	3829	3829	3829

20.1.3 Sector groups

Industry taxonomies group detailed classifications into broader sectors for economic analysis:

- Fama and French aggregate 4-digit SIC codes into industry groups consisting of 5, 10, 12, 17, 30, 38, 48, or 49 sectors.
- The Bureau of Economic Analysis (BEA) consolidates 6-digit NAICS codes into summary-level industry groups, with updates in 1947, 1963, and 1997.

```
# include sectoring schemes
codes = {'sic': ([f"codes{c}" for c in [5, 10, 12, 17, 30, 38, 48, 49]] \
    + ['sic2', 'sic3']), \
    'naics': ['bea1947', 'bea1963', 'bea1997']}
sectorings = {}    # store Sectoring objects
for key, schemes in codes.items():
    for scheme in schemes:
        if scheme not in sectorings:
            # missing value is integer 0 sic2 and sic3 shemes, else string ''
            fillna = 0 if scheme.startswith('sic') else ''
            # load the sectoring class from SQL
```

(continues on next page)

(continued from previous page)

```

sectorings[scheme] = Sectoring(sql, scheme, fillna=fillna)

# apply the sectoring scheme to partition the nodes
nodes[scheme] = sectorings[scheme][nodes[key]]

# keep nodes with non-missing data
nodes = nodes[nodes[scheme].ne(sectorings[scheme].fillna)]
print(len(nodes), scheme)
nodes

```

```

3845 codes5
3845 codes10
3845 codes12
3845 codes17
3845 codes30
3845 codes38
3845 codes48
3845 codes49
3829 sic2
3829 sic3
3561 bea1947
3561 bea1963
3561 bea1997

```

	lpermno	sic	naics	codes5	codes10	codes12	codes17	codes30	codes38	\
gvkey										
1004	54594	5080	423860	Cnsmr	Shops	Shops	Machn	Whlsl	Whlsl	
1045	21020	4512	481111	Other	Durbl	Durbl	Trans	Trans	Trans	
1050	11499	3564	333413	Manuf	Manuf	Manuf	Machn	FabPr	Machn	
1076	10517	6141	522220	Other	Other	Money	Finan	Fin	Money	
1078	20482	3845	334510	Hlth	Hlth	Hlth	Other	Hlth	Instr	
...	
345980	20333	5961	455110	Cnsmr	Shops	Shops	Rtail	Rtail	Rtail	
347007	15533	2836	325414	Hlth	Hlth	Hlth	Other	Hlth	Chems	
349337	20867	3845	334510	Hlth	Hlth	Hlth	Other	Hlth	Instr	
349972	15642	2836	325414	Hlth	Hlth	Hlth	Other	Hlth	Chems	
351038	16161	2834	325412	Hlth	Hlth	Hlth	Cnsum	Hlth	Chems	
				codes48	codes49	sic2	sic3	bea1947	bea1963	bea1997
gvkey										
1004	Whlsl	Whlsl	50	508	42	42	42			
1045	Trans	Trans	45	451	48	481	481			
1050	Mach	Mach	35	356	333	333	333			
1076	Banks	Banks	61	614	52	521CI	521CI			
1078	MedEq	MedEq	38	384	334	334	334			
...	
345980	Rtail	Rtail	59	596	44RT	44RT	4A0			
347007	Drugs	Drugs	28	283	325	325	325			
349337	MedEq	MedEq	38	384	334	334	334			
349972	Drugs	Drugs	28	283	325	325	325			
351038	Drugs	Drugs	28	283	325	325	325			

[3561 rows x 16 columns]

20.2 Community structure

In network analysis, **community structure** refers to the clustering of nodes (firms) into partitions (groups) based on connectivity patterns. Identifying these communities helps reveal hidden industry relationships and competitive dynamics.

```
# populate undirected graph with tnic edges
edges = tnic[tnic['gvkey1'].isin(nodes.index) & tnic['gvkey2'].isin(nodes.index)]
edges = list(
    edges[['gvkey1', 'gvkey2', 'score']].itertuples(index=False, name=None))
G = nx.Graph()
G.add_weighted_edges_from(edges)
G.remove_edges_from(nx.selfloop_edges(G)) # remove self-loops: not necessary
Series(graph_info(G, fast=True)).rename(year).to_frame()
```

	2023
transitivity	0.877035
average_clustering	0.575643
connected	False
connected_components	9
size_largest_component	3523
directed	False
weighted	True
negatively_weighted	False
edges	320352
nodes	3541
selfloops	0
density	0.051113

20.2.1 Measuring partitions

The quality of graph partitions can be evaluated using modularity, a measure that assesses the strength of community structures by comparing observed connections within clusters to a random network model.

```
# evaluate modularity of sectoring schemes on TNIC graph
def community_quality(G, communities):
    """helper to measure quality of partitions"""
    out = {'communities': len(communities)}
    out['modularity'] = nx.community.modularity(G, communities)
    (out['coverage'],
     out['performance']) = nx.community.partition_quality(G, communities)
    return out
```

```
modularity = {} # to collect measurements of each scheme
for scheme in sorted(chain(*codes.values())):
    communities = nodes.loc[list(G.nodes), scheme] \
        .reset_index() \
        .groupby(scheme)[['gvkey']] \
        .apply(list) \
        .to_list() # list of lists of node labels
    modularity[scheme] = community_quality(G, communities)
df = DataFrame.from_dict(modularity, orient='index').sort_index()
print(f"Quality of sectoring schemes on TNIC graph ({year})")
df
```

Quality of sectoring schemes on TNIC graph (2023)

	communities	modularity	coverage	performance
bea1947	40	0.330481	0.779187	0.925859
bea1963	58	0.324246	0.734745	0.948296
bea1997	61	0.324169	0.734514	0.948689
codes10	10	0.335843	0.940503	0.850069
codes12	12	0.336655	0.938187	0.878268
codes17	17	0.285847	0.766719	0.794675
codes30	30	0.335544	0.934385	0.899115
codes38	36	0.333800	0.793237	0.890785
codes48	48	0.331168	0.752610	0.944559
codes49	49	0.331003	0.751526	0.951096
codes5	5	0.337074	0.945045	0.818984
sic2	67	0.327541	0.743694	0.942476
sic3	226	0.288389	0.690952	0.958297

20.2.2 Detecting partitions

Community detection in graphs can be performed using various algorithms, including:

- **Label Propagation Algorithm:** This method assigns an initial label to each node and iteratively updates labels based on the majority of its neighbors' labels, allowing communities to form dynamically. It is fast and works well for large networks but may produce different results on different runs due to randomness.
- **Louvain Method:** This hierarchical clustering algorithm optimizes modularity by iteratively merging small communities into larger ones, maximizing intra-community connections while minimizing inter-community edges.
- **Greedy Algorithm:** This algorithm builds communities by iteratively merging pairs of nodes or groups that result in the largest modularity gain, prioritizing locally optimal choices.

```
# Run community detection algorithms
def community_detection(G):
    """Helper to run community detection algorithms on an undirected graph"""
    out = {}
    out['label'] = nx.community.label_propagation_communities(G)
    out['louvain'] = nx.community.louvain_communities(G, resolution=1)
    out['greedy'] = nx.community.greedy_modularity_communities(G, resolution=1)
    return out
```

```
communities = community_detection(G)
quality = {}
for key, community in communities.items():
    quality[key] = community_quality(G, community)
```

```
df = DataFrame.from_dict(quality, orient='index').sort_index()
print(f"Modularity of community detection algorithms on TNIC graph ({year})")
df
```

Modularity of community detection algorithms on TNIC graph (2023)

	communities	modularity	coverage	performance
greedy	51	0.323848	0.989711	0.689093
label	101	0.347795	0.990485	0.909486
louvain	19	0.354818	0.824855	0.838868

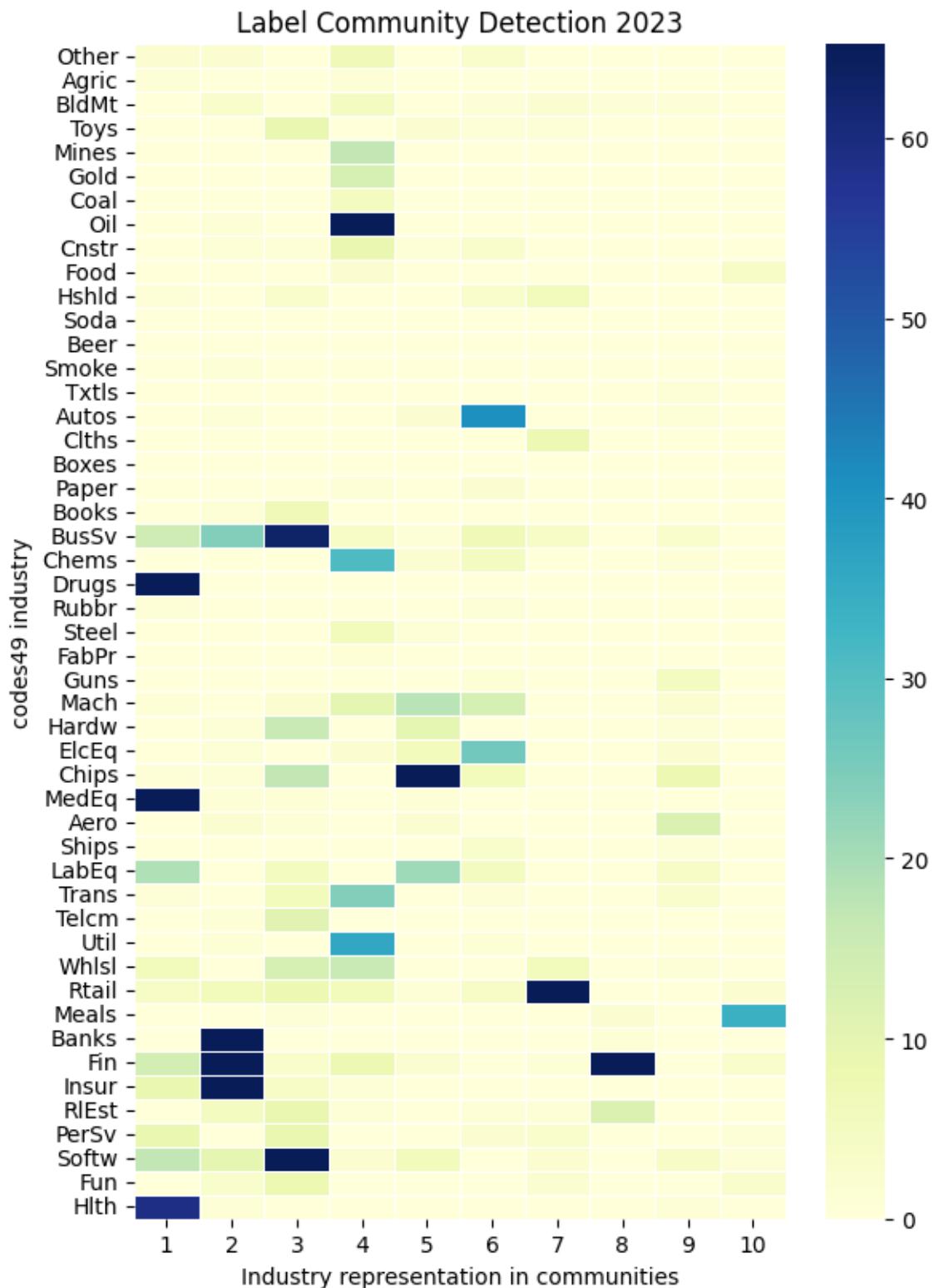
```

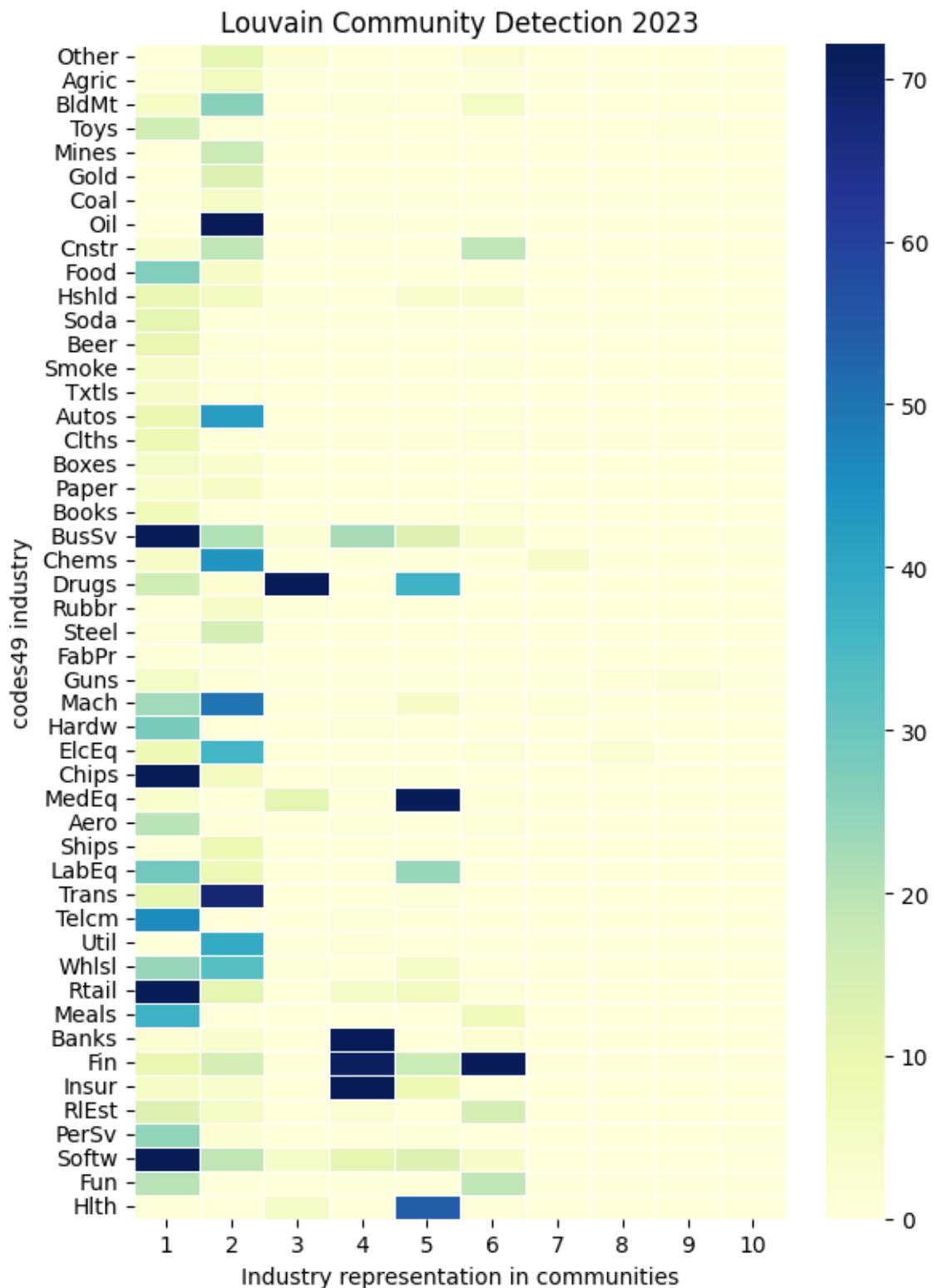
# Visualize Fama-French 49-industries in the detected communities
key = 'codes49'
for ifig, detection in enumerate(communitys.keys()):

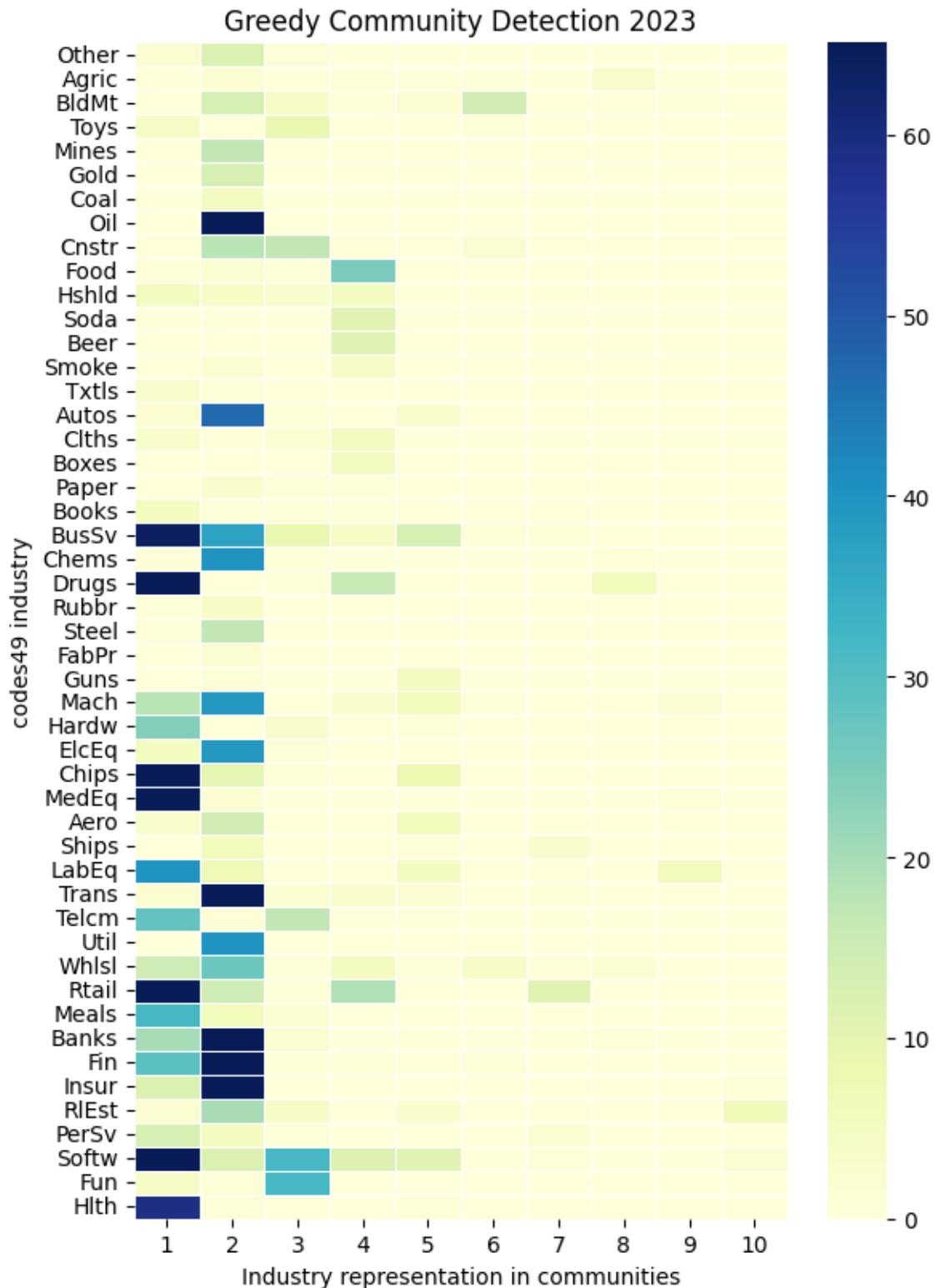
    # count industries represented in each partition
    industry = []
    communitys_sequence = sorted(communitys[detection], key=len, reverse=True)
    for i, community in enumerate(communitys_sequence):
        industry.append(nodes[key][list(community)].value_counts().rename(i+1))
    names = sectorings[key].sectors['name'].drop_duplicates(keep='first')
    df = pd.concat(industry, axis=1) \
        .dropna(axis=0, how='all') \
        .fillna(0) \
        .astype(int) \
        .reindex(names)

    # display as heatmap
    fig, ax = plt.subplots(num=ifig+1, clear=True, figsize=(6, 8))
    sns.heatmap(df.iloc[:, :10],
                square=False,
                linewidth=.5,
                ax=ax,
                yticklabels=1,
                cmap="YlGnBu",
                robust=True)
    if scheme.startswith('bea'):
        ax.set_yticklabels(Sectoring._bea_industry[df.index], size=10)
    else:
        ax.set_yticklabels(df.index, size=10)
    ax.set_title(f'{detection.capitalize()} Community Detection {year}')
    ax.set_xlabel("Industry representation in communities")
    ax.set_ylabel(f'{key} industry")
    fig.subplots_adjust(left=0.4)
    plt.tight_layout(pad=0)

```







References

Gerard Hoberg and Gordon Phillips, 2016, Text-Based Network Industries and Endogenous Product Differentiation. *Journal of Political Economy* 124 (5), 1423-1465.

Gerard Hoberg and Gordon Phillips, 2010, Product Market Synergies and Competition in Mergers and Acquisitions: A Text-Based Analysis. *Review of Financial Studies* 23 (10), 3773-3811.

INPUT-OUTPUT GRAPH CENTRALITY

And so we all matter - maybe less then a lot but always more than none - John Green

Input-output analysis in economics models the interdependencies between sectors by tracking the flow of goods and services. Graph centrality measures are valuable tools for analyzing the structure and dynamics of such networks. By representing input-output data published by the Bureau of Economic Analysis (BEA) as a directed graph (with sectors as nodes and transactions as edges from producers to consumers), we can apply centrality metrics to identify the most influential sectors driving overall economic activity.

```
# By: Terence Lim, 2020-2025 (terence-lim.github.io)
import pandas as pd
from pandas import DataFrame, Series
import networkx as nx
from finds.database import RedisDB
from finds.readers import BEA
from finds.recipes import graph_info, graph_draw
from secret import credentials
# %matplotlib qt
VERBOSE = 0
pd.set_option('display.max_rows', None)
pd.set_option('display.max_colwidth', 200)
```

```
rdb = RedisDB(**credentials['redis'])
bea = BEA(rdb, **credentials['bea'], verbose=VERBOSE)
years = [1947, 2023]
vintages = [1997, 1963, 1947]    # when sectoring schemes were revised
vintage = min(vintages)
```

21.1 Centrality measures

Graph centrality measures help quantify the importance of nodes within a network, providing insight into their roles and influence.

- **Degree Centrality** measures the number of edges connected to a node, indicating its direct level of activity.
- **Betweenness Centrality** quantifies the extent to which a node lies on the shortest paths between other nodes, capturing its role in network connectivity.
- **Closeness Centrality** assesses how close a node is to all other nodes based on shortest path distances, highlighting its accessibility.
- **Eigenvector Centrality** evaluates a node's importance based on the centrality of its neighbors, giving higher scores to nodes connected to other influential nodes.

- **PageRank Centrality** was originally developed to rank web pages and operates as a random walk process, estimating the long-term likelihood of visiting each node by following edges.
- **Hubs** are nodes with many outgoing edges, acting as facilitators of flow to other parts of the network.
- **Authorities** are nodes with many incoming edges, representing highly influential entities that are frequently referenced by other nodes.

```
# Helper to compute centrality measures
def nodes_centrality(G, weight='weight', cost=False, alpha=0.99):
    """Return dict of vertex centrality measures

    Args:
        G: Graph may be directed or undirected, weighted or unweighted
        weight: name of edge attribute for weights, Set to None for unweighted
        cost: If True, then weights are costs; else weights are importances
    """
    out = {}
    # Degree centrality, for directed and undirected graphs
    if nx.is_directed(G):
        out['in_degree'] = nx.in_degree_centrality(G)
        out['out_degree'] = nx.out_degree_centrality(G)
    else:
        out['degree'] = nx.degree_centrality(G)

    # Hubs and Authorities
    out['hub'], out['authority'] = nx.hits(G)

    # if weights are costs, then Eigenvector and PageRank ignore weights
    if not cost and nx.is_weighted(G):
        out['eigenvector'] = nx.eigenvector_centrality(G, weight=weight, max_iter=1000)
        out['pageRank'] = nx.pageRank(G, weight=weight, alpha=alpha)
    else:
        out['eigenvector'] = nx.eigenvector_centrality(G, max_iter=1000)
        out['pageRank'] = nx.pageRank(G, alpha=alpha)

    # if weights are importances, then Betweenness and Closeness ignore weights
    if cost and nx.is_weighted(G):
        out['betweenness'] = nx.betweenness_centrality(G, weight=weight)
        out['closeness'] = nx.closeness_centrality(G, distance=weight)
    else:
        out['betweenness'] = nx.betweenness_centrality(G)
        out['closeness'] = nx.closeness_centrality(G)
    return out
```

21.2 BEA Input-Output Use Tables

Input-output analysis is an economic modeling technique used to study interdependencies among different sectors. It involves constructing input-output tables that track the flow of goods and services between industries, quantifying how changes in one sector impact others.

```
# Read IOUse tables from BEA website
ioUses = {year: bea.read_ioUse(year=year, vintage=vintage) for year in years}
```

To analyze these relationships, we construct a directed graph from the latest BEA input-output flows, where edges flow from user sectors to make sectors. This allows us to visualize sectoral interactions and determine the most influential industries.

```
## Direction of edges point from user industry to maker, i.e. follows the money
tail = 'colcode' # edges follow flow of payments, from column to row
head = 'rowcode'
drop = ('F', 'T', 'U', 'V', 'Other') # drop these codes

# Display node centrality measures from latest year
ioUse = ioUses[max(years)]
data = ioUse[(~ioUse['rowcode'].str.startswith(drop) &
             ~ioUse['colcode'].str.startswith(drop))].copy()

# extract cross data; generate and load edges (as tuples) to graph
data = data[(data['colcode'] != data['rowcode'])]
data['weights'] = data['datavalue'] / data['datavalue'].sum()
edges = data.loc[data['weights'] > 0, [tail, head, 'weights']].values.tolist()
G = nx.DiGraph()
G.add_weighted_edges_from(edges, weight='weight')

# Display graph properties
Series(graph_info(G)).rename('Properties').to_frame()
```

Properties	
transitivity	0.903811
average_clustering	0.849259
weakly_connected	True
weakly_connected_components	1
size_largest_weak_component	47
strongly_connected	False
strongly_connected_components	4
size_largest_strong_component	44
directed	True
weighted	True
negatively_weighted	False
edges	1689
nodes	47
selfloops	0
density	0.781221

Vizualise graphs of input-output flows for the earliest and latest years available, highlighting the top five sectors with the highest PageRank centrality scores.

```
colors = ['lightgrey', 'darkgreen', 'lightgreen']

# Populate and plot graph of first and last table years
for ifig, year in enumerate(years):
    # keep year, drop invalid rows
    ioUse = ioUses[year]
    data = ioUse[(~ioUse['rowcode'].str.startswith(drop) &
                  ~ioUse['colcode'].str.startswith(drop))].copy()

    # create master table of industries and measurements
    master = data[data['rowcode'] == data['colcode']][['rowcode', 'datavalue']]\
        .set_index('rowcode')\
```

(continues on next page)

(continued from previous page)

```

.rename(columns={'datavalue': 'self'})

# extract cross data; generate and load edges (as tuples) to graph
data = data[(data['colcode'] != data['rowcode'])]
data['weights'] = data['datavalue'] / data['datavalue'].sum()
edges = data.loc[data['weights'] > 0, [tail, head, 'weights']]\
    .values\
    .tolist()

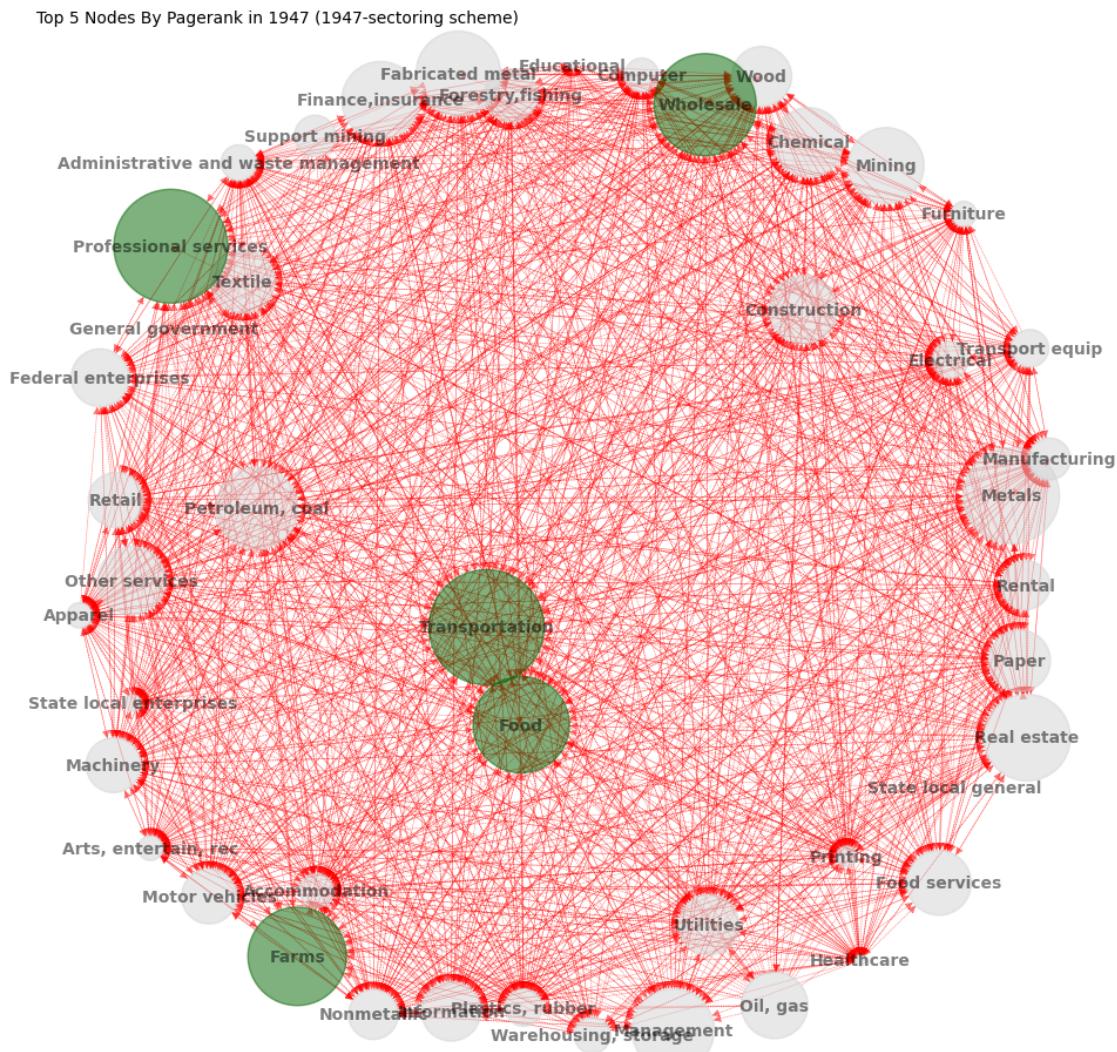
G = nx.DiGraph()
G.add_weighted_edges_from(edges, weight='weight')
nx_labels = BEA.short_desc[list(G.nodes)].to_dict()

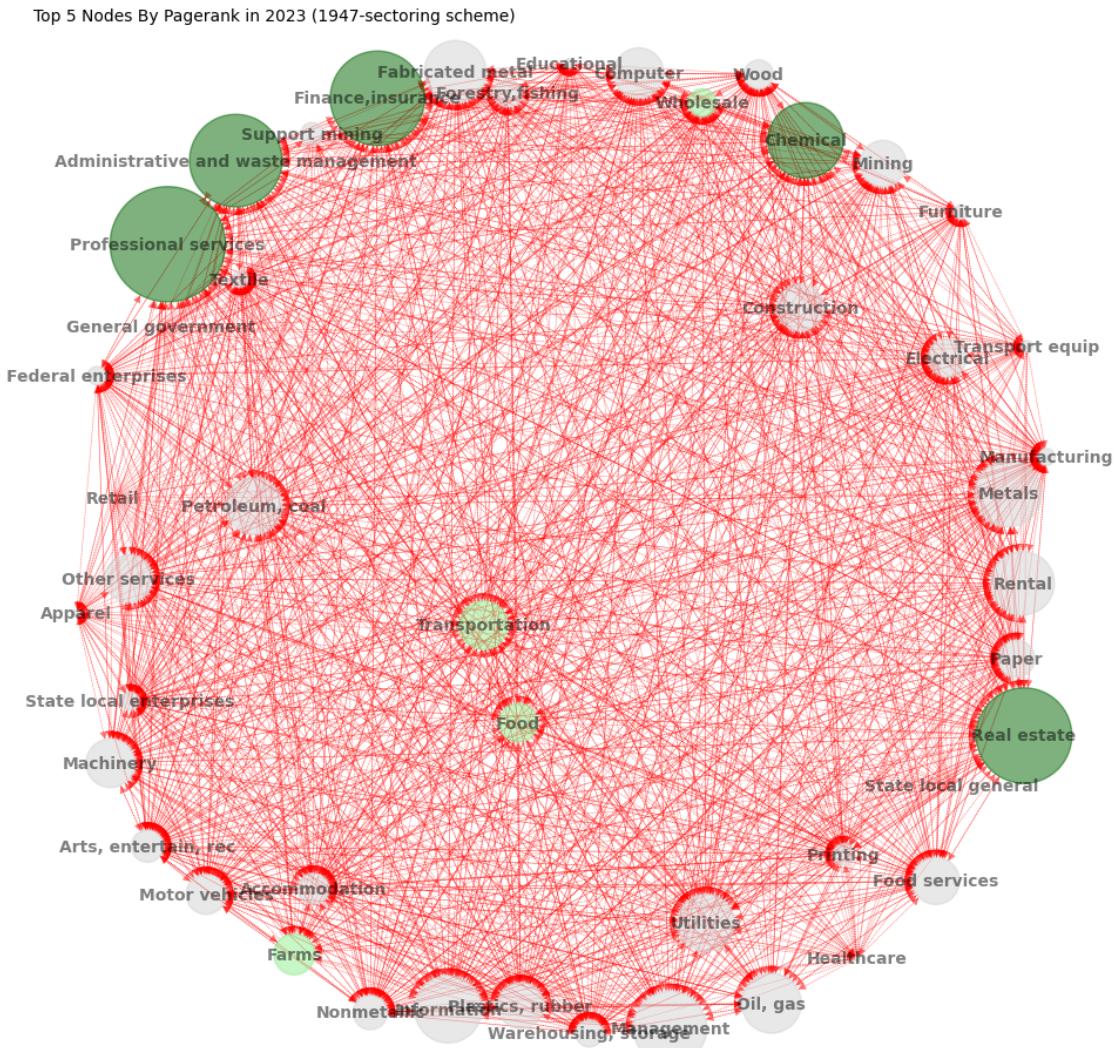
# update master table industry flow values
master = master.join(data.groupby(['colcode'])['datavalue'].sum(), how='outer')\
    .rename(columns={'datavalue': 'user'})
master = master.join(data.groupby(['rowcode'])['datavalue'].sum(), how='outer')\
    .rename(columns={'datavalue': 'maker'})
master = master.fillna(0).astype(int)

# inweight~supply~authority~eigenvector~pagerank, outweight~demand~hub
centrality = DataFrame(nodes_centrality(G)) # compute centrality metrics
master = master.join(centrality, how='left')
master['bea'] = BEA.short_desc[master.index].to_list()

# visualize graph
score = centrality['pagerank']
node_size = score.to_dict()
node_color = {node: colors[0] for node in G.nodes()}
if ifig == 0:
    center_name = score.index[score.argmax()]
else:
    node_color.update({k: colors[2] for k in top_color})
top_color = list(score.index[score.argsort()[-5:]])
node_color.update(dict.fromkeys(top_color, colors[1]))
pos = graph_draw(G,
                  num=ifig+1,
                  figsize=(10, 10),
                  center_name=center_name,
                  node_color=node_color,
                  node_size=node_size,
                  edge_color='r',
                  k=3,
                  pos=(pos if ifig else None),
                  font_size=10,
                  font_weight='semibold',
                  labels=master['bea'].to_dict(),
                  title=f"Top 5 Nodes By Pagerank in {year} ({vintage}-sectoring_{scheme})")

```





Display centrality scores for all BEA sectors based on the latest year's input-output data:

```

# show industry flow values and graph centrality measures
master = pd.concat(
    (data[data['rowcode'] == data['colcode']][['rowcode', 'datavalue']]\
     .set_index('rowcode')\
     .rename(columns={'datavalue': 'self'}),\
     data.groupby(['colcode'])['datavalue'].sum().rename('user'),\
     data.groupby(['rowcode'])['datavalue'].sum().rename('maker')),\
     join='outer', axis=1).fillna(0).astype(int)
master = master.join(DataFrame(nodes_centrality(G)), how='left')
master['bea'] = BEA.short_desc[master.index].to_list()

print(f"Node Centrality of BEA Input-Output Use Table {year}")
master.drop(columns=['self']).round(3)

```

Node Centrality of BEA Input-Output Use Table 2023

	user	maker	in_degree	out_degree	hub	authority	\
111CA	183083	413260	0.478	0.783	0.005	0.008	
113FF	4764	83642	0.543	0.652	0.000	0.001	
211	189659	491570	0.674	0.674	0.008	0.004	
212	43330	125323	0.870	0.804	0.002	0.003	
213	29107	23233	0.087	0.652	0.002	0.000	
22	148366	426549	1.000	0.717	0.008	0.026	
23	1108514	381314	1.000	0.761	0.037	0.027	
311FT	610392	378716	0.630	0.804	0.013	0.022	
313TT	22726	46617	0.848	0.761	0.001	0.002	
315AL	9831	7442	0.435	0.674	0.000	0.000	
321	64039	187546	0.957	0.804	0.002	0.010	
322	72249	177891	0.978	0.717	0.002	0.007	
323	42407	84936	0.739	0.804	0.002	0.007	
324	514797	537972	1.000	0.761	0.005	0.026	
325	207051	659545	1.000	0.804	0.009	0.031	
326	174402	358413	1.000	0.761	0.006	0.018	
327	61298	227005	0.957	0.739	0.002	0.010	
331	81903	335633	0.891	0.717	0.003	0.004	
332	195632	501561	1.000	0.761	0.006	0.017	
333	203600	266998	1.000	0.761	0.005	0.009	
334	53901	373613	0.978	0.696	0.003	0.020	
335	74954	201230	1.000	0.717	0.002	0.008	
3361MV	333656	199393	0.978	0.783	0.008	0.010	
3364OT	129087	85122	0.391	0.717	0.005	0.003	
337	44645	54948	0.522	0.717	0.001	0.003	
339	69034	113798	0.935	0.804	0.003	0.010	
42	1134476	73996	0.848	0.848	0.085	0.003	
44RT	992525	0	0.000	0.913	0.080	-0.000	
48	532651	399454	1.000	0.848	0.030	0.031	
493	55326	180519	0.913	0.696	0.004	0.014	
51	715155	648393	1.000	0.848	0.062	0.043	
52	714298	1199853	1.000	0.674	0.070	0.092	
531	1209091	1565039	1.000	0.739	0.096	0.113	
532RL	275723	508226	1.000	0.696	0.021	0.029	
54	772709	2108318	1.000	0.935	0.045	0.152	
55	279051	754756	0.891	0.848	0.025	0.051	
56	496066	1206168	1.000	0.913	0.033	0.099	
61	155822	31243	0.413	0.848	0.013	0.002	
62	1211742	24110	0.130	0.848	0.090	0.002	
71	182097	135112	1.000	0.913	0.016	0.010	
721	127238	104038	0.978	0.826	0.008	0.008	
722	614168	322353	1.000	0.826	0.038	0.028	
81	350192	348957	1.000	0.913	0.029	0.024	
GFE	40482	73578	0.761	0.739	0.002	0.006	
GFG	479004	0	0.000	0.848	0.030	0.000	
GSLE	296865	42462	0.891	0.783	0.016	0.003	
GSLG	1162737	0	0.000	0.870	0.066	-0.000	
	eigenvector	pagerank	betweenness	closeness	\		
111CA	0.039	0.015	0.001	0.657			
113FF	0.007	0.007	0.001	0.687			
211	0.072	0.032	0.004	0.754			
212	0.019	0.019	0.008	0.885			

(continues on next page)

(continued from previous page)

213	0.003	0.005	0.000	0.523
22	0.085	0.029	0.002	1.000
23	0.111	0.020	0.004	1.000
311FT	0.039	0.013	0.002	0.730
313TT	0.004	0.002	0.002	0.868
315AL	0.001	0.000	0.000	0.639
321	0.037	0.007	0.003	0.958
322	0.030	0.012	0.002	0.979
323	0.022	0.004	0.003	0.793
324	0.081	0.028	0.003	1.000
325	0.140	0.051	0.003	1.000
326	0.057	0.019	0.003	1.000
327	0.046	0.010	0.002	0.958
331	0.043	0.039	0.001	0.902
332	0.090	0.034	0.002	1.000
333	0.051	0.021	0.003	1.000
334	0.076	0.022	0.001	0.979
335	0.043	0.013	0.002	1.000
3361MV	0.041	0.014	0.004	0.979
3364OT	0.004	0.001	0.000	0.622
337	0.014	0.002	0.001	0.676
339	0.009	0.003	0.004	0.939
42	0.010	0.007	0.004	0.868
44RT	0.000	0.000	0.000	0.000
48	0.097	0.021	0.005	1.000
493	0.021	0.007	0.001	0.920
51	0.212	0.040	0.005	1.000
52	0.400	0.079	0.002	1.000
531	0.368	0.081	0.003	1.000
532RL	0.147	0.036	0.001	1.000
54	0.531	0.118	0.018	1.000
55	0.172	0.048	0.009	0.902
56	0.449	0.076	0.012	1.000
61	0.002	0.001	0.001	0.630
62	0.000	0.000	0.000	0.535
71	0.053	0.009	0.012	1.000
721	0.052	0.009	0.004	0.979
722	0.118	0.020	0.004	1.000
81	0.099	0.020	0.012	1.000
GFE	0.014	0.003	0.004	0.807
GFG	0.000	0.000	0.000	0.000
GSLE	0.012	0.003	0.004	0.902
GSLG	0.000	0.000	0.000	0.000

		bea
111CA		Farms
113FF		Forestry, fishing
211		Oil, gas
212		Mining
213		Support mining
22		Utilities
23		Construction
311FT		Food
313TT		Textile
315AL		Apparel
321		Wood

(continues on next page)

(continued from previous page)

322	Paper
323	Printing
324	Petroleum, coal
325	Chemical
326	Plastics, rubber
327	Nonmetallic
331	Metals
332	Fabricated metal
333	Machinery
334	Computer
335	Electrical
3361MV	Motor vehicles
3364OT	Transport equip
337	Furniture
339	Manufacturing
42	Wholesale
44RT	Retail
48	Transportation
493	Warehousing, storage
51	Information
52	Finance, insurance
531	Real estate
532RL	Rental
54	Professional services
55	Management
56	Administrative and waste management
61	Educational
62	Healthcare
71	Arts, entertain, rec
721	Accommodation
722	Food services
81	Other services
GFE	Federal enterprises
GFG	General government
GSLE	State local enterprises
GSLG	State local general

Compare the 1947 and 1997 sectoring schemes to examine how BEA's industry groupings have evolved over time.

```
# Compare 1947 and 1997 sector schemes (BEA "summary"-level industry groups)
v1947 = BEA.sectoring(1947).rename(columns={'description': '1947'})
v1997 = BEA.sectoring(1997).rename(columns={'description': '1997'})
df = v1947[['title', '1947']].join(v1997['1997'])
df[df['1947'] != df['1997']] # changes in the sectoring scheme
```

```
title \
code
441000
  vehicle and parts dealers
442000
  All other retail
443000
  All other retail
444000
  Building material and garden
```

(continues on next page)

(continued from previous page)

↳ equipment and supplies dealers	
445000	↳
↳ Food and beverage stores	
446000	↳
↳ and personal care stores	Health
447000	↳
↳ Gasoline stations	
448000	↳
↳ clothing accessories stores	Clothing and
451000	↳
↳ All other retail	
452000	↳
↳ General merchandise stores	
453000	↳
↳ All other retail	
454000	↳
↳ Nonstore retailers	
481000	↳
↳ Air transportation	
482000	↳
↳ Rail transportation	
483000	↳
↳ Water transportation	
484000	↳
↳ Truck transportation	
485000	↳
↳ passenger transportation	Transit and ground
486000	↳
↳ Pipeline transportation	
487000	↳
↳ and support activities	Scenic and sightseeing transportation
488000	↳
↳ and support activities	Scenic and sightseeing transportation
492000	↳
↳ Couriers and messengers	
511000	↳
↳ internet (includes software)	Publishing industries, except
511110	↳
↳ Newspaper publishers	
511120	↳
↳ Periodical publishers	
511130	↳
↳ Book publishers	Directory, mailing list,
511140	↳
↳ and other publishers	Directory, mailing list,
511190	↳
↳ and other publishers	
511210	↳
↳ Software publishers	
512000	↳
↳ sound recording industries	Motion picture and
512100	↳
↳ picture and video industries	Motion
512200	↳
↳ Sound recording industries	
513000	↳
	Broadcasting

(continues on next page)

(continued from previous page)

↳ and telecommunications	
514000	Data processing, internet publishing, and...
↳ other information services	
515100	Radio and...
↳ television broadcasting	
515200	Cable and other...
↳ subscription programming	
517100	Wired...
↳ telecommunications carriers	
517200	Wireless telecommunications...
↳ carriers (except satellite)	
517400	Satellite, telecommunications resellers, and all...
↳ other telecommunications	
518200	Data processing, hosting,
↳ and related services	
519110	News syndicates, libraries, archives and all...
↳ other information services	
519130	Internet publishing and broadcasting...
↳ and Web search portals	
521000	Monetary authorities and depository...
↳ credit intermediation	
522100	Monetary authorities and depository...
↳ credit intermediation	
522200	Nondepository credit intermediation...
↳ and related activities	
523000	Securities, commodity...
↳ contracts, and investments	
523100	Securities and commodity contracts...
↳ intermediation and brokerage	
523900	Other financial...
↳ investment activities	
524000	Insurance carriers...
↳ and related activities	
524113	Direct...
↳ life insurance carriers	
524114	Insurance...
↳ carriers, except direct life	
524120	Insurance...
↳ carriers, except direct life	
524130	Insurance...
↳ carriers, except direct life	
524200	Insurance agencies, brokerages...
↳ and related activities	
525000	Funds, trusts, and...
↳ other financial vehicles	
541100	Legal services
↳ Legal services	
541200	Accounting, tax preparation, bookkeeping,
↳ and payroll services	
541300	Architectural, engineering,
↳ and related services	
541400	...
↳ Specialized design services	
541500	Computer systems...
↳ design and related services	
541511	Custom...

(continues on next page)

(continued from previous page)

↳ computer programming services	
541512	Computer
↳ systems design services	
541513	Other computer related services, including
↳ facilities management	
541519	Other computer related services, including
↳ facilities management	
541610	
↳ Management consulting services	
541620	Environmental and other
↳ technical consulting services	
541690	Environmental and other
↳ technical consulting services	
541700	Scientific research
↳ and development services	
541800	Advertising, public relations,
↳ and related services	
541910	All other miscellaneous professional, scientific,
↳ and technical services	
541920	
↳ Photographic services	
541930	All other miscellaneous professional, scientific,
↳ and technical services	
541940	
↳ Veterinary services	
541990	All other miscellaneous professional, scientific,
↳ and technical services	
561000	
↳ Administrative and support services	
561100	Office
↳ administrative services	
561200	
↳ Facilities support services	
561300	
↳ Employment services	
561400	
↳ Business support services	
561500	Travel arrangement
↳ and reservation services	
561600	Investigation
↳ and security services	
561700	Services to
↳ buildings and dwellings	
561900	
↳ Other support services	
562000	Waste management
↳ and remediation services	
621000	
↳ Ambulatory health care services	
621100	
↳ Offices of physicians	
621200	
↳ Offices of dentists	
621300	Offices of
↳ other health practitioners	
621400	

(continues on next page)

(continued from previous page)

↳ Outpatient care centers	
621500	Medical and...
↳ diagnostic laboratories	
621600	
↳ Home health care services	
621900	Other...
↳ ambulatory health care services	
622000	
↳ Hospitals	
623000	Nursing and...
↳ residential care facilities	
623100	Nursing and...
↳ community care facilities	
623200	Residential mental health, substance abuse, and other...
↳ residential care facilities	
623300	Nursing and...
↳ community care facilities	
623900	Residential mental health, substance abuse, and other...
↳ residential care facilities	
624000	
↳ Social assistance	
624100	
↳ Individual and family services	
624200	Community food, housing, and other relief services, including vocational...
↳ rehabilitation services	
624400	
↳ Child day care services	
711000	Performing arts, spectator sports, museums,...
↳ and related activities	
711100	
↳ Performing arts companies	
711200	
↳ Spectator sports	
711300	Promoters of performing arts and sports and...
↳ agents for public figures	
711500	Independent artists,...
↳ writers, and performers	
712000	Museums, historical...
↳ sites, zoos, and parks	
713000	Amusements, gambling, and...
↳ recreation industries	
713100	
↳ Amusement parks and arcades	
713200	Gambling industries...
↳ (except casino hotels)	
713900	Other amusement and...
↳ recreation industries	
1947	\
code	
441000	RETAIL TRADE
442000	RETAIL TRADE
443000	RETAIL TRADE
444000	RETAIL TRADE
445000	RETAIL TRADE
446000	RETAIL TRADE

(continues on next page)

(continued from previous page)

447000	RETAIL TRADE
448000	RETAIL TRADE
451000	RETAIL TRADE
452000	RETAIL TRADE
453000	RETAIL TRADE
454000	RETAIL TRADE
481000	Transportation
482000	Transportation
483000	Transportation
484000	Transportation
485000	Transportation
486000	Transportation
487000	Transportation
488000	Transportation
492000	Transportation
511000	INFORMATION
511110	INFORMATION
511120	INFORMATION
511130	INFORMATION
511140	INFORMATION
511190	INFORMATION
511210	INFORMATION
512000	INFORMATION
512100	INFORMATION
512200	INFORMATION
513000	INFORMATION
514000	INFORMATION
515100	INFORMATION
515200	INFORMATION
517100	INFORMATION
517200	INFORMATION
517400	INFORMATION
518200	INFORMATION
519110	INFORMATION
519130	INFORMATION
521000	FINANCE AND INSURANCE
522100	FINANCE AND INSURANCE
522200	FINANCE AND INSURANCE
523000	FINANCE AND INSURANCE
523100	FINANCE AND INSURANCE
523900	FINANCE AND INSURANCE
524000	FINANCE AND INSURANCE
524113	FINANCE AND INSURANCE
524114	FINANCE AND INSURANCE
524120	FINANCE AND INSURANCE
524130	FINANCE AND INSURANCE
524200	FINANCE AND INSURANCE
525000	FINANCE AND INSURANCE
541100	PROFESSIONAL AND TECHNICAL SERVICES
541200	PROFESSIONAL AND TECHNICAL SERVICES
541300	PROFESSIONAL AND TECHNICAL SERVICES
541400	PROFESSIONAL AND TECHNICAL SERVICES
541500	PROFESSIONAL AND TECHNICAL SERVICES
541511	PROFESSIONAL AND TECHNICAL SERVICES
541512	PROFESSIONAL AND TECHNICAL SERVICES
541513	PROFESSIONAL AND TECHNICAL SERVICES

(continues on next page)

(continued from previous page)

541519	PROFESSIONAL AND TECHNICAL SERVICES
541610	PROFESSIONAL AND TECHNICAL SERVICES
541620	PROFESSIONAL AND TECHNICAL SERVICES
541690	PROFESSIONAL AND TECHNICAL SERVICES
541700	PROFESSIONAL AND TECHNICAL SERVICES
541800	PROFESSIONAL AND TECHNICAL SERVICES
541910	PROFESSIONAL AND TECHNICAL SERVICES
541920	PROFESSIONAL AND TECHNICAL SERVICES
541930	PROFESSIONAL AND TECHNICAL SERVICES
541940	PROFESSIONAL AND TECHNICAL SERVICES
541990	PROFESSIONAL AND TECHNICAL SERVICES
561000	ADMINISTRATIVE AND WASTE SERVICES
561100	ADMINISTRATIVE AND WASTE SERVICES
561200	ADMINISTRATIVE AND WASTE SERVICES
561300	ADMINISTRATIVE AND WASTE SERVICES
561400	ADMINISTRATIVE AND WASTE SERVICES
561500	ADMINISTRATIVE AND WASTE SERVICES
561600	ADMINISTRATIVE AND WASTE SERVICES
561700	ADMINISTRATIVE AND WASTE SERVICES
561900	ADMINISTRATIVE AND WASTE SERVICES
562000	ADMINISTRATIVE AND WASTE SERVICES
621000	HEALTH CARE AND SOCIAL ASSISTANCE
621100	HEALTH CARE AND SOCIAL ASSISTANCE
621200	HEALTH CARE AND SOCIAL ASSISTANCE
621300	HEALTH CARE AND SOCIAL ASSISTANCE
621400	HEALTH CARE AND SOCIAL ASSISTANCE
621500	HEALTH CARE AND SOCIAL ASSISTANCE
621600	HEALTH CARE AND SOCIAL ASSISTANCE
621900	HEALTH CARE AND SOCIAL ASSISTANCE
622000	HEALTH CARE AND SOCIAL ASSISTANCE
623000	HEALTH CARE AND SOCIAL ASSISTANCE
623100	HEALTH CARE AND SOCIAL ASSISTANCE
623200	HEALTH CARE AND SOCIAL ASSISTANCE
623300	HEALTH CARE AND SOCIAL ASSISTANCE
623900	HEALTH CARE AND SOCIAL ASSISTANCE
624000	HEALTH CARE AND SOCIAL ASSISTANCE
624100	HEALTH CARE AND SOCIAL ASSISTANCE
624200	HEALTH CARE AND SOCIAL ASSISTANCE
624400	HEALTH CARE AND SOCIAL ASSISTANCE
711000	ARTS, ENTERTAINMENT, AND RECREATION
711100	ARTS, ENTERTAINMENT, AND RECREATION
711200	ARTS, ENTERTAINMENT, AND RECREATION
711300	ARTS, ENTERTAINMENT, AND RECREATION
711500	ARTS, ENTERTAINMENT, AND RECREATION
712000	ARTS, ENTERTAINMENT, AND RECREATION
713000	ARTS, ENTERTAINMENT, AND RECREATION
713100	ARTS, ENTERTAINMENT, AND RECREATION
713200	ARTS, ENTERTAINMENT, AND RECREATION
713900	ARTS, ENTERTAINMENT, AND RECREATION

1997

code	
441000	Motor vehicle and parts dealers
442000	Other retail
443000	Other retail
444000	Other retail

(continues on next page)

(continued from previous page)

445000	Food and beverage stores
446000	Other retail
447000	Other retail
448000	Other retail
451000	Other retail
452000	General merchandise stores
453000	Other retail
454000	Other retail
481000	Air transportation
482000	Rail transportation
483000	Water transportation
484000	Truck transportation
485000	Transit and ground passenger transportation
486000	Pipeline transportation
487000	Other transportation and support activities
488000	Other transportation and support activities
492000	Other transportation and support activities
511000	Publishing industries, except internet (includes software)
511110	Publishing industries, except internet (includes software)
511120	Publishing industries, except internet (includes software)
511130	Publishing industries, except internet (includes software)
511140	Publishing industries, except internet (includes software)
511190	Publishing industries, except internet (includes software)
511210	Publishing industries, except internet (includes software)
512000	Motion picture and sound recording industries
512100	Motion picture and sound recording industries
512200	Motion picture and sound recording industries
513000	Broadcasting and telecommunications
514000	Data processing, internet publishing, and other information services
515100	Broadcasting and telecommunications
515200	Broadcasting and telecommunications
517100	Broadcasting and telecommunications
517200	Broadcasting and telecommunications
517400	Broadcasting and telecommunications
518200	Data processing, internet publishing, and other information services
519110	Data processing, internet publishing, and other information services
519130	Data processing, internet publishing, and other information services
521000	Federal Reserve banks, credit intermediation, and related activities
522100	Federal Reserve banks, credit intermediation, and related activities
522200	Federal Reserve banks, credit intermediation, and related activities
523000	Securities, commodity contracts, and investments
523100	Securities, commodity contracts, and investments
523900	Securities, commodity contracts, and investments
524000	Insurance carriers and related activities
524113	Insurance carriers and related activities
524114	Insurance carriers and related activities
524120	Insurance carriers and related activities
524130	Insurance carriers and related activities
524200	Insurance carriers and related activities
525000	Funds, trusts, and other financial vehicles
541100	Legal services
541200	Miscellaneous professional, scientific, and technical services
541300	Miscellaneous professional, scientific, and technical services
541400	Miscellaneous professional, scientific, and technical services
541500	Computer systems design and related services
541511	Computer systems design and related services

(continues on next page)

(continued from previous page)

541512	Computer systems design and related services
541513	Computer systems design and related services
541519	Computer systems design and related services
541610	Miscellaneous professional, scientific, and technical services
541620	Miscellaneous professional, scientific, and technical services
541690	Miscellaneous professional, scientific, and technical services
541700	Miscellaneous professional, scientific, and technical services
541800	Miscellaneous professional, scientific, and technical services
541910	Miscellaneous professional, scientific, and technical services
541920	Miscellaneous professional, scientific, and technical services
541930	Miscellaneous professional, scientific, and technical services
541940	Miscellaneous professional, scientific, and technical services
541990	Miscellaneous professional, scientific, and technical services
561000	Administrative and support services
561100	Administrative and support services
561200	Administrative and support services
561300	Administrative and support services
561400	Administrative and support services
561500	Administrative and support services
561600	Administrative and support services
561700	Administrative and support services
561900	Administrative and support services
562000	Waste management and remediation services
621000	Ambulatory health care services
621100	Ambulatory health care services
621200	Ambulatory health care services
621300	Ambulatory health care services
621400	Ambulatory health care services
621500	Ambulatory health care services
621600	Ambulatory health care services
621900	Ambulatory health care services
622000	Hospitals
623000	Nursing and residential care facilities
623100	Nursing and residential care facilities
623200	Nursing and residential care facilities
623300	Nursing and residential care facilities
623900	Nursing and residential care facilities
624000	Social assistance
624100	Social assistance
624200	Social assistance
624400	Social assistance
711000	Performing arts, spectator sports, museums, and related activities
711100	Performing arts, spectator sports, museums, and related activities
711200	Performing arts, spectator sports, museums, and related activities
711300	Performing arts, spectator sports, museums, and related activities
711500	Performing arts, spectator sports, museums, and related activities
712000	Performing arts, spectator sports, museums, and related activities
713000	Amusements, gambling, and recreation industries
713100	Amusements, gambling, and recreation industries
713200	Amusements, gambling, and recreation industries
713900	Amusements, gambling, and recreation industries

References:

Jason Choi & Andrew T. Foerster, 2017. "The Changing Input-Output Network Structure of the U.S. Economy," Economic Review, Federal Reserve Bank of Kansas City, issue Q II, pages 23-49

<https://www.bea.gov/industry/input-output-accounts-data>

<https://www.bea.gov/information-updates-national-economic-accounts>

PRODUCT MARKET LINK PREDICTION

A hidden connection is stronger than an obvious one - Heraclitus

Link prediction in network analysis seeks to infer missing or future connections between nodes based on observed relationships. In the context of product markets, firms are interconnected through shared product similarities and competitive interactions. We construct firm networks from text-based product market similarity data, and explore the application of several link prediction algorithms. Additionally, accuracy metrics such as precision-recall and ROC curves are examined to assess the performance of these predictions.

```
# By: Terence Lim, 2020-2025 (terence-lim.github.io)
import zipfile
import io
import numpy as np
import pandas as pd
from pandas import DataFrame, Series
from sklearn import metrics
import matplotlib.pyplot as plt
import networkx as nx
from finds.database import SQL, RedisDB
from finds.structured import CRSP, BusDay, PSTAT
from finds.readers import requests_get
from finds.recipes import graph_info
from secret import credentials, paths
# %matplotlib qt
VERBOSE = 0
```

```
sql = SQL(**credentials['sql'], verbose=VERBOSE)
bd = BusDay(sql, verbose=VERBOSE)
rdb = RedisDB(**credentials['redis'])
pstat = PSTAT(sql, bd, verbose=VERBOSE)
crsp = CRSP(sql, bd, rdb, verbose=VERBOSE)
```

22.1 Product market linkages

Hoberg and Phillips (2016) developed a text-based measure of firm similarity by analyzing product descriptions in 10-K filings. Their methodology constructs firm-by-firm similarity scores using word vectors, filtering out common words and focusing on nouns and proper nouns while excluding geographic terms. This approach captures shifts in product markets over time, as revealed by their business descriptions in annual company filings.

The **TNIC-3** dataset is calibrated to align with the granularity of three-digit SIC codes, providing a structured industry classification. The **TNIC-2** dataset represents a more comprehensive version, including all firm pairs, even those with weak relationships.

Source: Hoberg and Phillips Industry Classification

The TNIC-2 and TNIC-3 datasets are retrieved from the Hoberg and Phillips website.

```
root = 'https://hobergphillips.tuck.dartmouth.edu/idata/'
tnic_data = {}
for scheme in ['tnic2', 'tnic3']:
    source = root + scheme + '_data.zip'
    if source.startswith('http'):
        response = requests_get(source)
        source = io.BytesIO(response.content)
    with zipfile.ZipFile(source).open(scheme + "_data.txt") as f:
        tnic_data[scheme] = pd.read_csv(f, sep='\s+')
for k,v in tnic_data.items():
    print(k, v.shape)
```

```
tnic2 (52812348, 4)
tnic3 (27161830, 4)
```

```
# extract last year of both tnic schemes, merge in permno, and require in univ
year = max(tnic_data['tnic2']['year'])
capsize = 10 # large cap (large than NYSE median)
univ = crsp.get_universe(bd.endyr(year))
univ = univ[univ['decile'] <= capsize]
lookup = pstat.build_lookup('gvkey', 'lpermno', fillna=0)
nodes = {}
tnic = {}
edges = {}
for scheme in ['tnic2', 'tnic3']:
    tnic[scheme] = tnic_data[scheme][tnic_data[scheme].year == year].dropna()
    gvkeys = sorted(set(tnic[scheme]['gvkey1']).union(tnic[scheme]['gvkey2']))
    df = DataFrame(index=gvkeys, data=lookup(gvkeys), columns=['permno'])
    nodes[scheme] = df[df['permno'].gt(0)
                        & df['permno'].isin(univ.index)].drop_duplicates()
    nodes['tnic2'] = nodes['tnic2'][nodes['tnic2'].index.isin(nodes['tnic3'].index)]
    nodes['tnic3'] = nodes['tnic3'][nodes['tnic3'].index.isin(nodes['tnic2'].index)]
```

Using the TNIC-2 and TNIC-3 datasets, undirected graphs are constructed where nodes represent firms and edges indicate product market similarities based on the chosen granularity of the classification schemes.

```
# create graphs of tnic2 (denser graph) and tnic3 (sparser graph) schemes
for scheme in ['tnic2', 'tnic3']:
    e = tnic[scheme][tnic[scheme]['gvkey1'].isin(nodes[scheme].index) &
                    tnic[scheme]['gvkey2'].isin(nodes[scheme].index)]
    edges[scheme] = list(e[['gvkey1', 'gvkey2', 'score']].\
        itertuples(index=False, name=None))
```

```
results = {}
G = {}
for (scheme, node), (_, edge) in zip(nodes.items(), edges.items()):
    print(scheme, 'nodes = ', len(node), 'edges = ', len(edge))

    # populate graph
    g = nx.Graph()
    g.add_nodes_from(node.index)
```

(continues on next page)

(continued from previous page)

```

g.add_weighted_edges_from(edge)

# remove self-loops: not necessary
g.remove_edges_from(nx.selfloop_edges(g))

# graph info
results[scheme] = Series(graph_info(g, fast=True))

# Plot degree distribution
fig, ax = plt.subplots(clear=True, figsize=(10, 6))
degree = nx.degree_histogram(g)
degree = DataFrame(data={'degree': degree[1:]}, # exclude degree 0
                   index=np.arange(1, len(degree)))
degree['bin'] = (degree.index // (2*capsize) + 1) * (2*capsize)
degree.groupby('bin').sum().plot(kind='bar', ax=ax, fontsize=6)
ax.set_title(f'Degree Distribution of {scheme.upper()} links {year}')
plt.tight_layout()

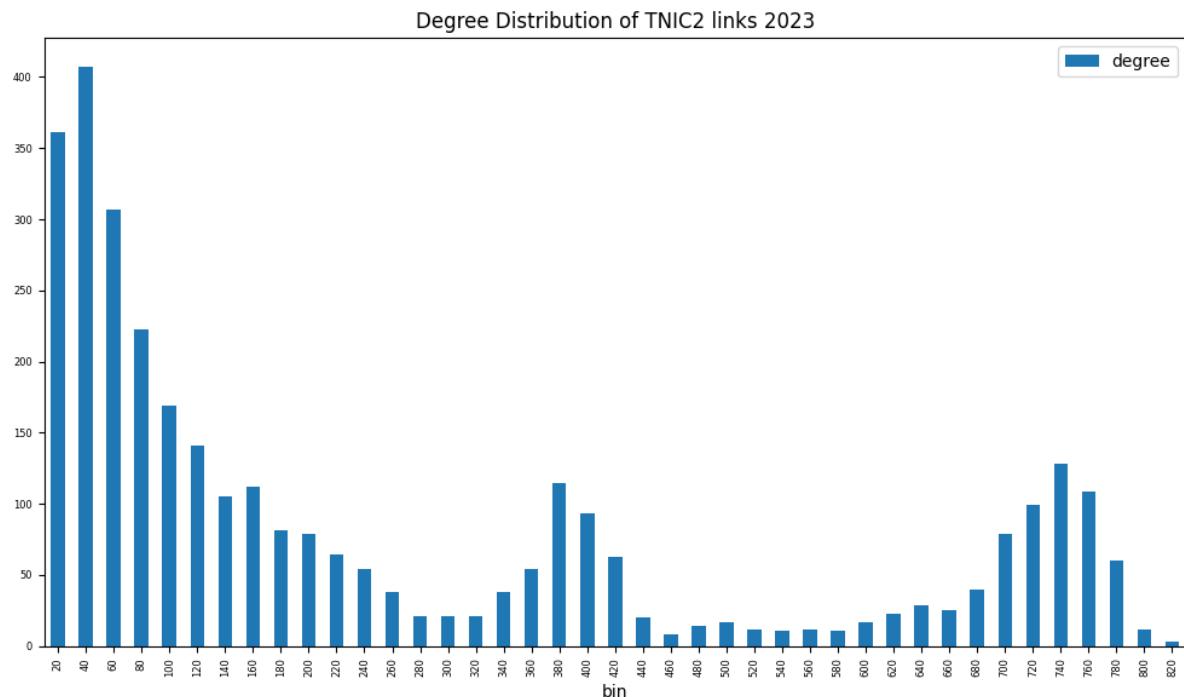
G[scheme] = g

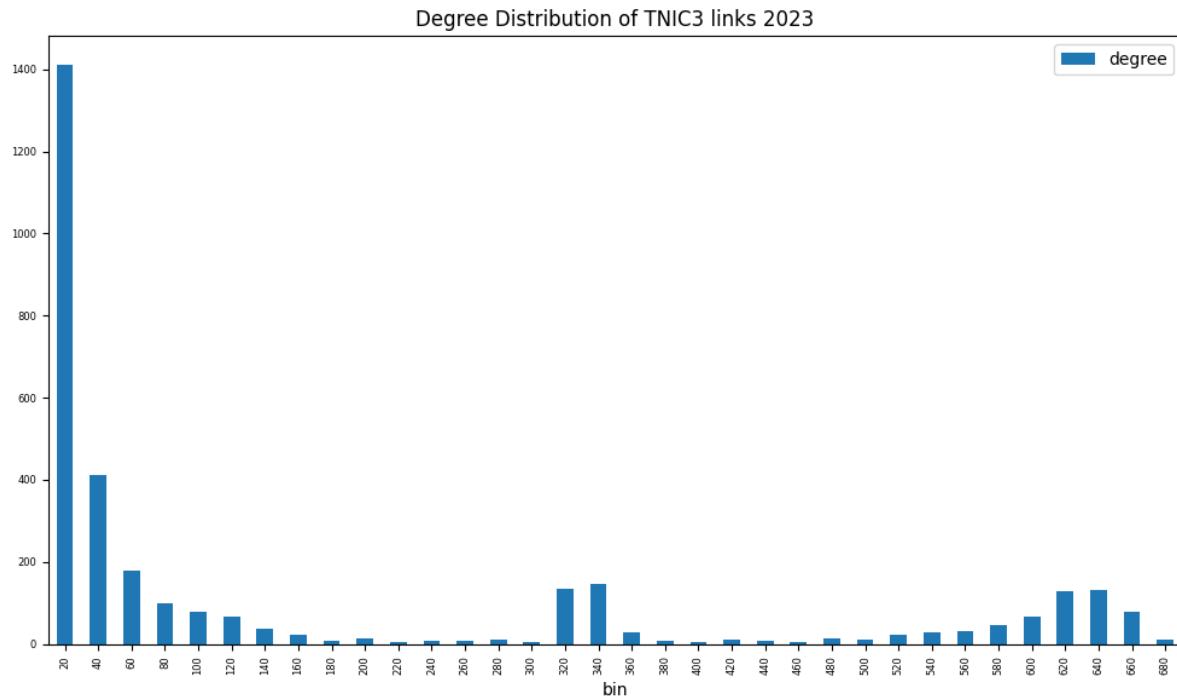
```

```

tnic2 nodes = 3296 edges = 819984
tnic3 nodes = 3296 edges = 528012

```





```
print(f"Graph properties of TNIC schemes {year}")
DataFrame(results)
```

Graph properties of TNIC schemes 2023

	tnic2	tnic3
transitivity	0.834713	0.881587
average_clustering	0.592383	0.571318
connected	True	False
connected_components	1	25
size_largest_component	3296	3255
directed	False	False
weighted	True	True
negatively_weighted	False	False
edges	409992	264006
nodes	3296	3296
selfloops	0	0
density	0.075503	0.048618

22.2 Link prediction algorithms

Link prediction aims to identify missing or future connections between nodes in a network. Given a partially observed network, these algorithms infer which links are most likely to be added or missing based on existing connections and network structure.

Common link prediction algorithms include:

- **Jaccard Coefficient:** Measures the similarity between two nodes by comparing their shared neighbors relative to their total number of neighbors.

- **Resource Allocation:** Assigns a higher likelihood of connection between nodes that share many common neighbors, emphasizing smaller-degree nodes.
- **Adamic-Adar:** Enhances the Resource Allocation approach by weighting common neighbors based on their overall connectivity.
- **Preferential Attachment:** Predicts new links based on the idea that nodes with higher degrees are more likely to form new connections.

```
# helper to call link prediction algorithms
def link_prediction(G):
    """Predict link scores for all nonexistent edges in graph"""

    def links(links):
        """returns list of edge-score 3-tuples sorted by highest score"""
        return sorted(links, key=lambda x: x[2], reverse=True)

    resource = links(nx.resource_allocation_index(G))
    jaccard = links(nx.jaccard_coefficient(G))
    adamic = links(nx.adamic_adar_index(G))
    preferential = links(nx.preferential_attachment(G))
    return {'resource_allocation': resource,
            'jaccard_coefficient': jaccard,
            'adamic_adar': adamic,
            'preferential_attachment': preferential}
```

```
links = link_prediction(G['tnic3'])
```

22.3 Accuracy metrics

Common metrics to assess the accuracy of link prediction (or any binary classification) models:

- **Precision:** The fraction of predicted links that are actual links.
- **Recall:** The fraction of actual links that were correctly predicted.
- **Accuracy:** The overall correctness of predictions.
- **Confusion Matrix:** A summary of prediction outcomes.
- **F1 Score:** A balanced measure between precision and recall, useful for imbalanced datasets. It is calculated as:

$$2 \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} = \frac{TP}{TP + (FP + FN)/2}$$

where

- False Positive (FP): Incorrectly predicted links.
- False Negative (FN): Missed actual links.
- True Positive (TP): Correctly predicted links.
- True Negative (TN): Correctly identified non-links.

22.3.1 ROC Curve

The **Receiver Operating Characteristic (ROC) curve** and the **Area Under the Curve (AUC)** evaluates model performance by measuring the trade-off between true positive and false positive rates at various classification score thresholds.

The ROC curve plots the following metrics on the two axes:

- **True Positive Rate (TPR) (or Sensitivity)** = $\frac{TP}{TP+FN}$: Measures how many actual positives are correctly identified.
- **False Positive Rate (FPR)** = $\frac{FP}{FP+TN}$: Measures how many negative cases are incorrectly classified as positive.

The AUC measures the area under the ROC curve and provides a single number to quantify model performance. A higher AUC means the model is better at distinguishing between positive and negative classes.

```
def make_sample(prediction, edges):
    """helper to extract predicted scores and gold labels"""
    names = [e[:2] for e in prediction]           # node-pairs of nonexistent edges
    scores = [e[-1] for e in prediction]          # predicted scores (ordered)
    gold = [e[:2] in edges for e in prediction]   # gold labels of nonexistent edges
    return gold, scores, names # actual, predicted score, names
```

```
report = {}
for ifig, (method, pred) in enumerate(links.items()):

    # extract predicted scores and gold labels of nonexistent edges
    # y, scores, names = make_sample(pred, G['tnic2'].edges)

    names = [e[:2] for e in pred]                 # node-pairs of nonexistent edges
    scores = [e[-1] for e in pred]                # predicted scores of edge
    y = [e[:2] in G['tnic2'].edges for e in pred] # gold labels of nonexistent edges

    # plot roc curve
    metrics.RocCurveDisplay.from_predictions(y_true=y, y_pred=scores,
                                              plot_chance_level=True)
    plt.title(f"ROC Curve: {method}")
    plt.tight_layout()

    # set classification threshold at class proportion
    thresh = scores[sum(y)]
    y_pred = [score >= thresh for score in scores]

    # generate and plot confusion matrix
    cm = metrics.confusion_matrix(y_true=y, y_pred=y_pred, normalize='all')
    disp = metrics.ConfusionMatrixDisplay(confusion_matrix=cm)
    disp.plot()
    plt.title(f"Confusion Matrix: {method}")
    plt.tight_layout()

    # generate classification report
    report[method] = metrics.classification_report(y_true=y, y_pred=y_pred)
    print(f"Classification Report: {method}")
    print(report[method])
```

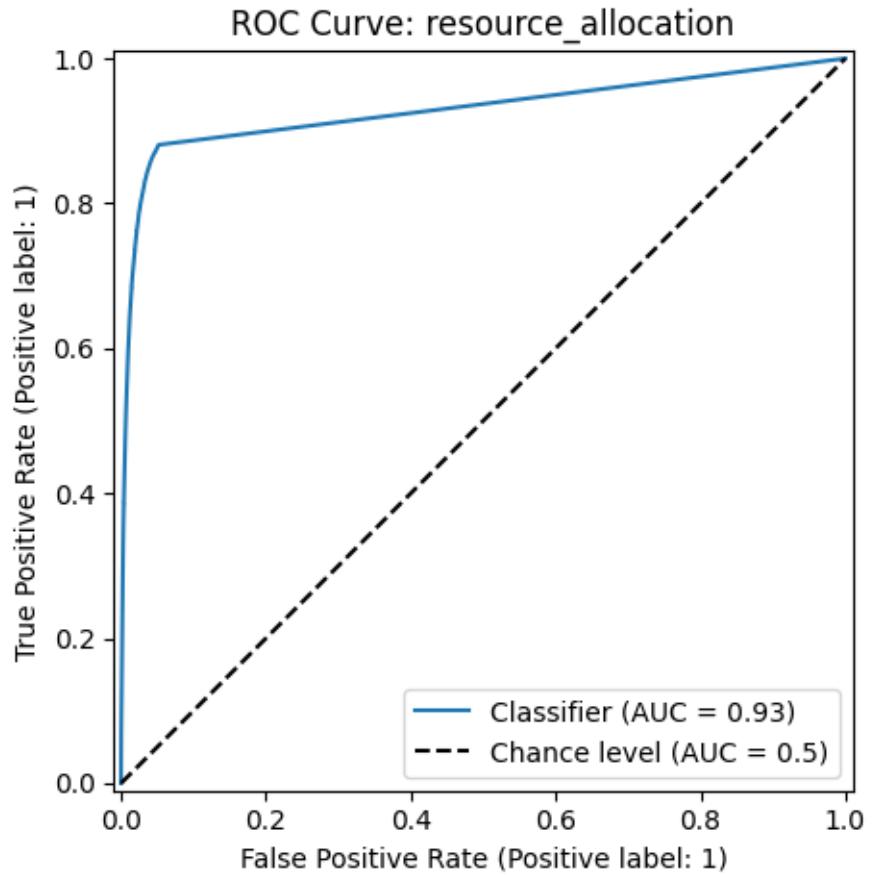
```
Classification Report: resource_allocation
      precision    recall  f1-score   support

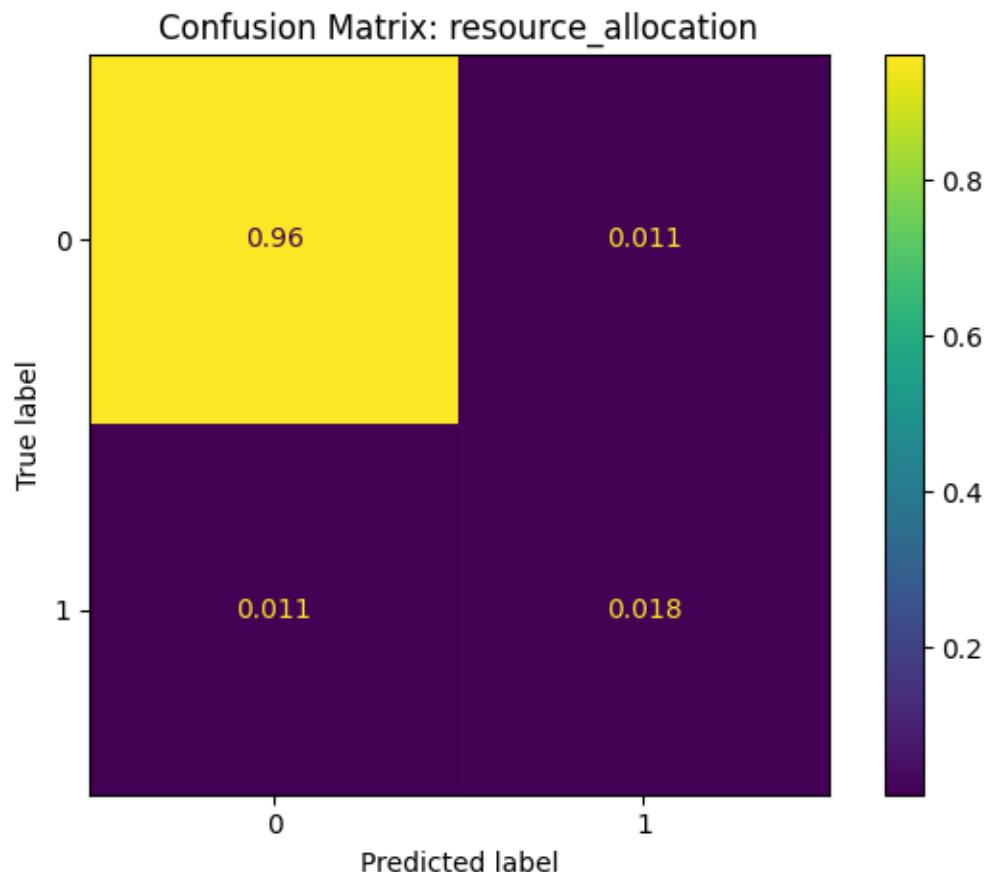

```

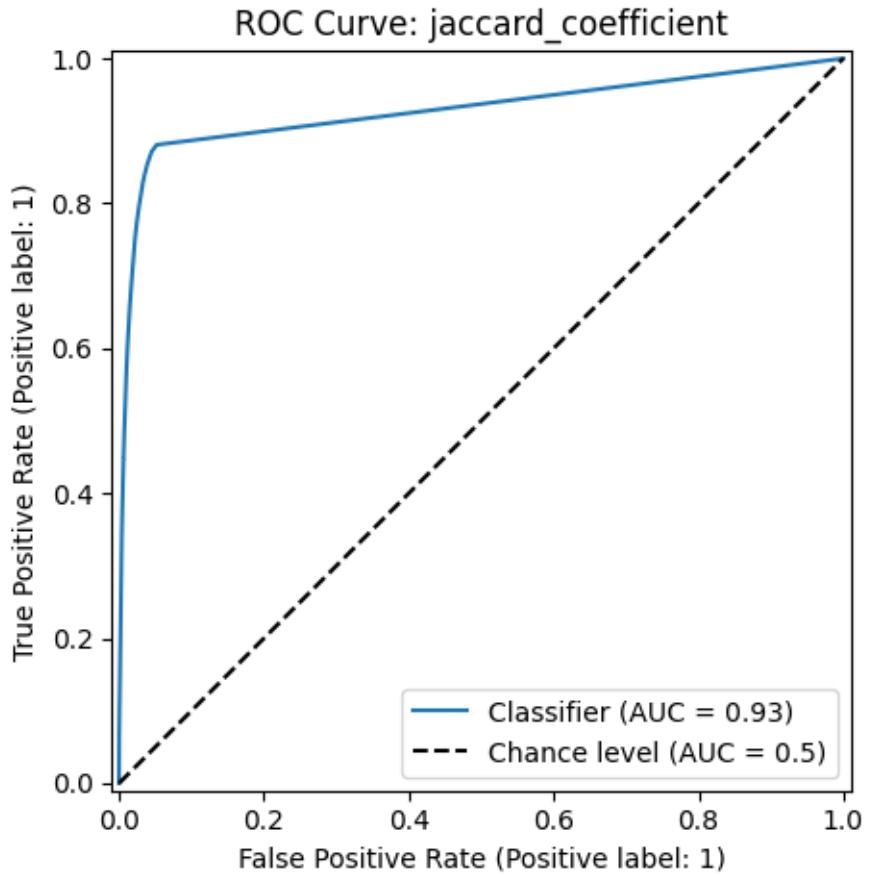
(continues on next page)

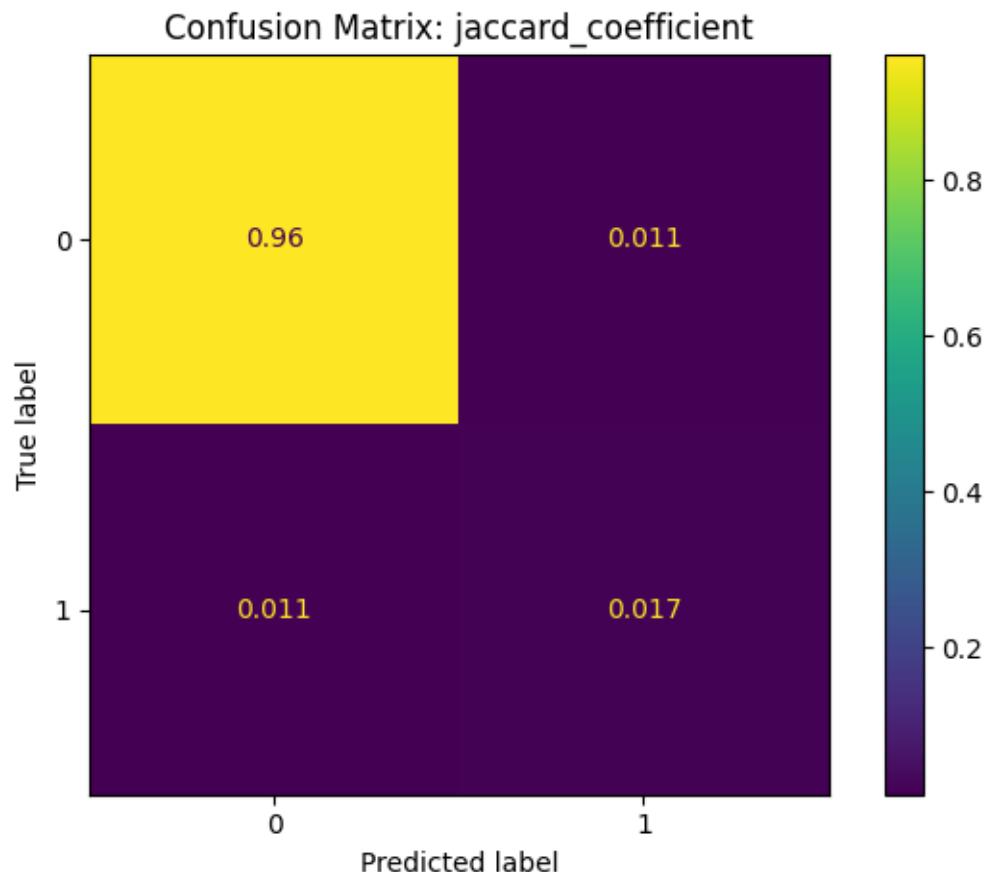
(continued from previous page)

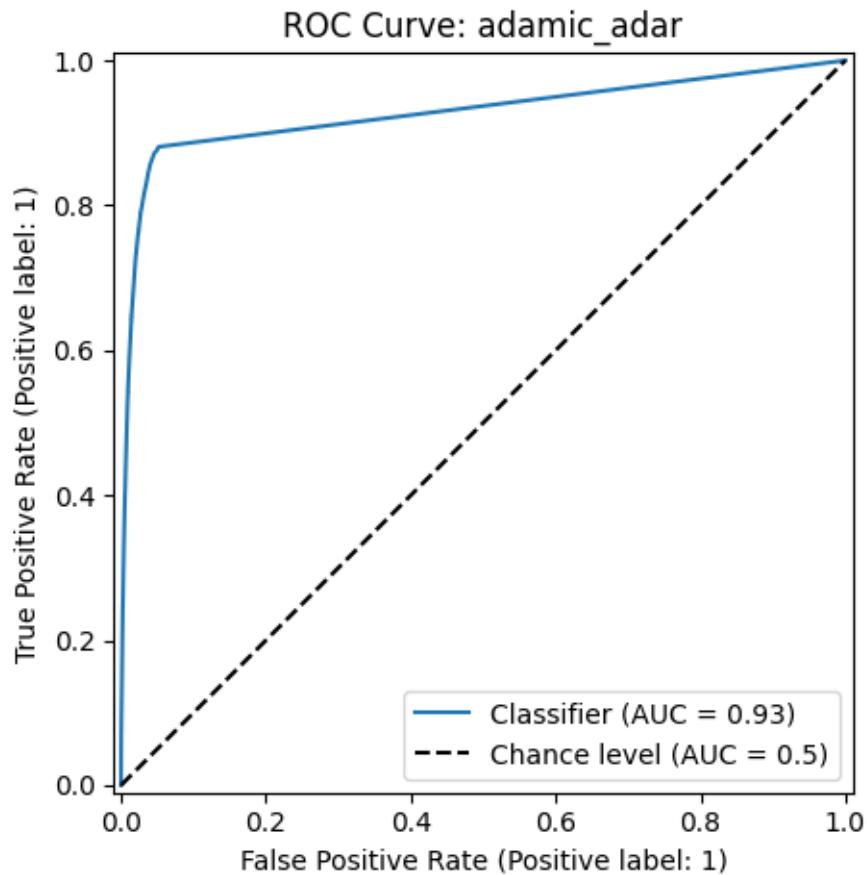
False	0.99	0.99	0.99	5020168
True	0.62	0.62	0.62	145986
accuracy			0.98	5166154
macro avg	0.80	0.80	0.80	5166154
weighted avg	0.98	0.98	0.98	5166154
Classification Report: jaccard_coefficient				
	precision	recall	f1-score	support
False	0.99	0.99	0.99	5020168
True	0.60	0.60	0.60	145986
accuracy			0.98	5166154
macro avg	0.79	0.79	0.79	5166154
weighted avg	0.98	0.98	0.98	5166154
Classification Report: adamic_adar				
	precision	recall	f1-score	support
False	0.99	0.99	0.99	5020168
True	0.60	0.60	0.60	145986
accuracy			0.98	5166154
macro avg	0.79	0.79	0.79	5166154
weighted avg	0.98	0.98	0.98	5166154
Classification Report: preferential_attachment				
	precision	recall	f1-score	support
False	0.97	0.97	0.97	5020168
True	0.09	0.09	0.09	145986
accuracy			0.95	5166154
macro avg	0.53	0.53	0.53	5166154
weighted avg	0.95	0.95	0.95	5166154

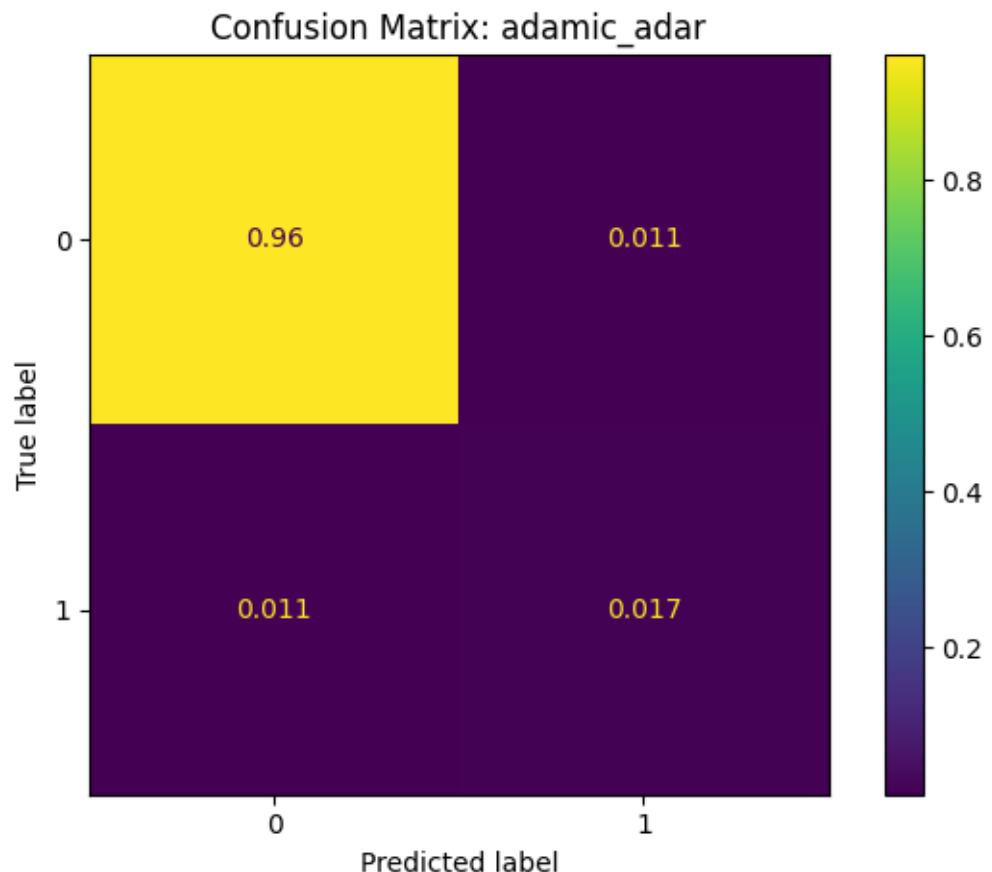


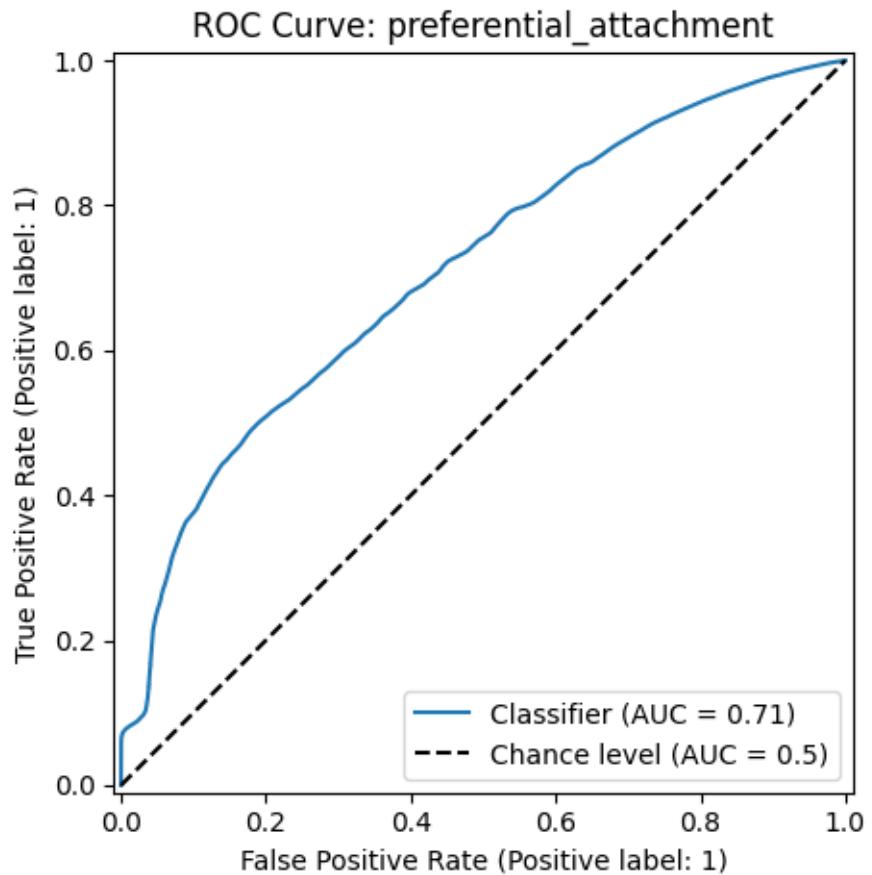


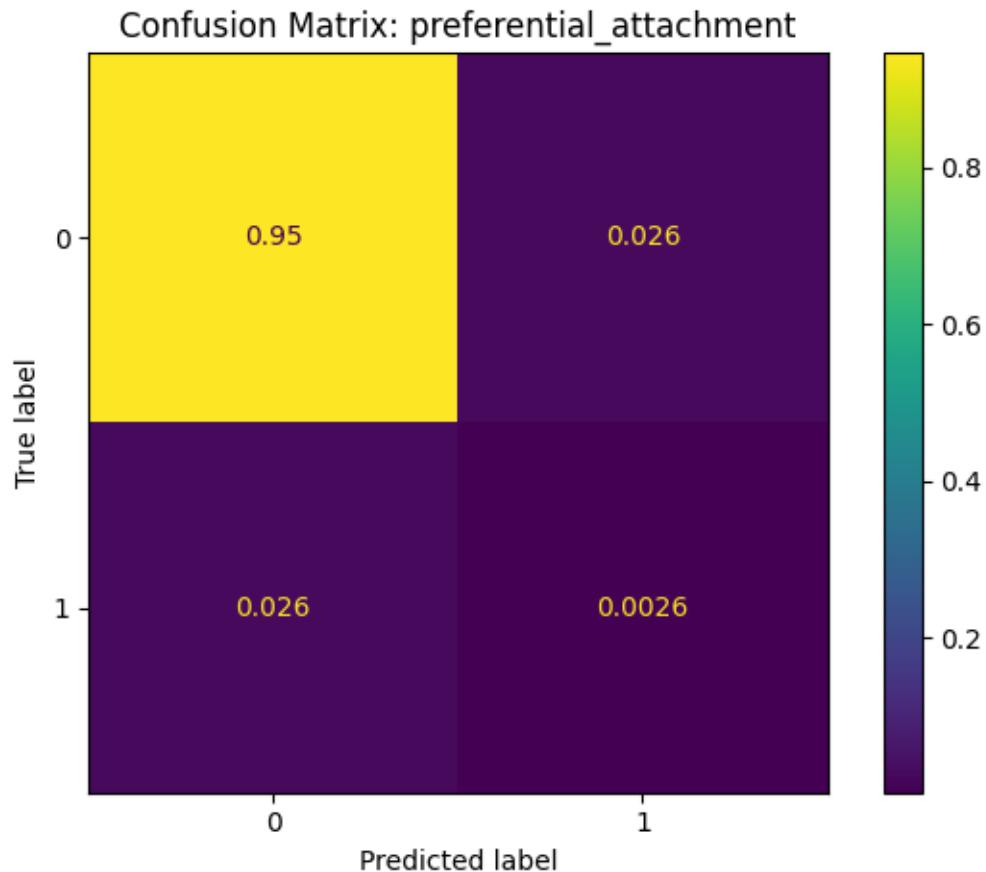










**References:**

Gerard Hoberg and Gordon Phillips, 2016, Text-Based Network Industries and Endogenous Product Differentiation. *Journal of Political Economy* 124 (5), 1423-1465.

Gerard Hoberg and Gordon Phillips, 2010, Product Market Synergies and Competition in Mergers and Acquisitions: A Text-Based Analysis. *Review of Financial Studies* 23 (10), 3773-3811.

CHAPTER
TWENTYTHREE

EARNINGS SPATIAL REGRESSION

Everything is related to everything else. But near things are more related than distant things - Waldo Tobler

Earnings surprises occur when a company reports earnings that are significantly different from what analysts had predicted. Spatial regression models help analyze how these surprises propagate through interconnected firms, particularly when firms operate within related industries. By using text-based firm similarity scores as spatial weights, we examine how earnings surprises cluster and spread across firms with similar product market characteristics.

```
# By: Terence Lim, 2020-2025 (terence-lim.github.io)
from pandas import DataFrame, Series
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm
import io
import zipfile
from libpysal.weights import W
fromesda.moran import Moran
import spreg
import networkx as nx
import statsmodels.formula.api as smf
import warnings
import requests
from finds.database import SQL, RedisDB
from finds.structured import BusDay, CRSP, PSTAT
from finds.readers import Alfred
from finds.recipes import remove_outliers
from finds.utils import Store
from secret import credentials, paths, CRSP_DATE
#pd.set_option('display.max_rows', 50)
VERBOSE = 0
#%matplotlib qt
```

```
# open connections
sql = SQL(**credentials['sql'], verbose=VERBOSE)
user = SQL(**credentials['user'], verbose=VERBOSE)
rdb = RedisDB(**credentials['redis'])
bd = BusDay(sql, verbose=VERBOSE)
crsp = CRSP(sql, bd, rdb=rdb, verbose=VERBOSE)
pstat = PSTAT(sql, bd, verbose=VERBOSE)
store = Store(paths['scratch'])
```

```
LAST_DATE = CRSP_DATE
scheme = 'tnic3'
# fetch NBER recession dates
alf = Alfred(api_key=credentials['fred']['api_key'], verbose=VERBOSE)
vspan = alf.date_spans('USREC') # to indicate recession periods in the plots
```

```
# create table to lookup gvkey by permno
lookup = pstat.build_lookup(source='lpermno', target='gvkey', fillna=0)
```

23.1 Earnings surprise

An earnings surprise, or unexpected earnings, is the difference between the reported earnings and expected earnings. The unexpected earnings per share when scaled by the stock price, at the fiscal end date, is more comparable across stocks of different market size and price levels.

```
# require and get actual quarterly earnings, reindex by (permno, rebaldate)
df = pstat.get_linked(dataset='quarterly',
                      fields=['prccq', 'cshoq', 'ibq'],
                      date_field='datadate',
                      where=f"datadate <= {LAST_DATE}")
# store['fund'] = df
```

```
fund = df.dropna(subset=['ibq']) \
    .sort_values(['permno', 'datadate', 'cshoq']) \
    .drop_duplicates(['permno', 'datadate']) \
    .reset_index()
fund['rebaldate'] = bd.endmo(fund['datadate'])

# calculate sue with lag(4) difference in compustat quarterly and price
lag = fund.shift(4, fill_value=0)
keep = ((lag['permno'] == fund['permno']) &
        (fund['prccq'] > 5)).values
fund.loc[keep, 'sue'] = ((fund.loc[keep, 'ibq'] - lag.loc[keep, 'ibq']) / 
                        abs(fund.loc[keep, 'prccq'] * fund.loc[keep, 'cshoq']))
print('with pstat earnings', np.sum(~fund['sue'].isna()))
```

with pstat earnings 740370

```
sue = fund.loc[~fund['sue'].isna(), ['permno', 'rebaldate', 'sue']] \
    .reset_index(drop=True)
sue['rebaldate'] // 100
```

23.2 Spatial dependence models

23.2.1 Moran's I

Moran's I is a statistical measure that quantifies spatial autocorrelation, assessing whether earnings surprises exhibit clustering across firms. The expected value of Moran's I under a random distribution is:

$$E(I) = \frac{-1}{N-1}$$

A positive Moran's I value suggests earnings surprises tend to cluster among similar firms, while a negative value indicates dispersion. This test provides a conservative benchmark by comparing results to a null hypothesis of zero spatial dependence, which helps identify significant deviations from randomness.

More details: [Moran's I \(Wikipedia\)](#)

23.2.2 Spatial lag model

The **spatial lag model** extends traditional regression analysis by incorporating the influence of neighboring observations. The model is expressed as:

The **spatial lag model** extends traditional regression analysis by incorporating the influence of neighboring observations. The model is expressed as:

$$Y_i = \beta X_i + \rho W Y_j + \epsilon_i$$

where:

- Y_i is the dependent variable (e.g., earnings surprise) for firm i ,
- Y_j represents the corresponding values from other firms,
- W is a matrix of spatial weights that assigns higher values to firms that are more closely related to firm i ,
- ρ captures the strength of spatial dependence, and
- ϵ_i is the error term.

Further reading:

- [Spatial Lag Model \(Lost Stats\)](#)
- [spreg.ML_Lag \(PySAL\)](#)

We use text-based firm similarity scores as spatial weights: the TNIC-3 scores, developed by Hoberg and Phillips (2016), are derived from product descriptions in 10-K filings. Their methodology constructs firm-by-firm similarity scores using word vectors while filtering out common words and focusing on nouns and proper nouns, excluding geographic terms.

```
# Retrieve TNIC linkages from Hoberg and Phillips website
root = 'https://hobergphillips.tuck.dartmouth.edu/idata/'
source = root + scheme + '_data.zip'
if source.startswith('http'):
    response = requests.get(source)
    source = io.BytesIO(response.content)
with zipfile.ZipFile(source).open(scheme + "_data.txt") as f:
    tnic_df = pd.read_csv(f, sep='\s+')
    store['spatial'] = (fund, tnic_df)
    tnic_df['year'].value_counts().sort_index().to_frame()
```

year	count
1988	311616
1989	640221
1990	663746
1991	699152
1992	788029
1993	940368
1994	1027776
1995	1095721
1996	1197991
1997	1138937
1998	1093821
1999	1174065
2000	1121918
2001	956591
2002	843735
2003	781521
2004	719787
2005	699340
2006	709914
2007	683662
2008	615445
2009	532159
2010	509755
2011	499887
2012	484441
2013	513159
2014	579649
2015	597923
2016	580988
2017	591097
2018	644292
2019	669597
2020	700725
2021	861057
2022	823596
2023	670149

```

# Compute quarterly spatial regression coefficients
out = dict(moran={}, rho={}, beta={})
years = range(1988, 2024)
for year in tqdm(years):
    tnic = tnic_df[tnic_df.year == year].dropna()    # extract the year's tnic links

    # populate graph from tnic edges, with gvkey as nodes
    graph = nx.Graph()
    graph.add_edges_from(tnic[['gvkey1', 'gvkey2']].values)
    graph.remove_edges_from(nx.selfloop_edges(graph))    # not necessary

    for qtr in [12, 103, 106, 109]:    # loop over next four quarters of sue
        rebaldate = year*100 + qtr

        # extract sue's with in this fiscal quarter
        y = sue[sue['rebaldate'] == rebaldate].set_index('permno')[['sue']]

```

(continues on next page)

(continued from previous page)

```

# lookup gvkeys
y['gvkey'] = lookup(y.index, date=bd.endmo(rebaldate))

# merge in stock returns in the quarter
y = y.join(crsp.get_ret(bd.begmo(rebaldate, months=-2),
                        bd.endmo(rebaldate)),
            how='left') \
      .set_index('gvkey')

# require available and not outlier
y = remove_outliers(y[~y.index.duplicated() & (y.index > 0)]).dropna()

# extract subgraph, its nodes and their neighbors
G = graph.subgraph(y.index)
G = G.subgraph(max(nx.connected_components(G), key=len))
neighbors = {node: list(G.neighbors(node)) for node in G.nodes()}
w = W(neighbors)
y = y.loc[sorted(neighbors.keys())]

# compute Moran's I and spatial lag model of sue on past stock returns
mi = Moran(y['sue'].values, w)
with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    spatial = spreg.ML_Lag(y=y['sue'].values, x=y[['ret']].values, w=w,
                           name_x=['ret'], name_y='sue', name_w=scheme,
                           name_ds=str(rebaldate))
out['moran'][rebaldate] = mi.I
out['rho'][rebaldate] = float(spatial.rho)
out['beta'][rebaldate] = float(spatial.betas[1][0])

```

100% |██████████| 36/36 [17:49<00:00, 29.70s/it]

```

# Show latest quarter's results
print(spatial.summary)

```

REGRESSION

```

-----
SUMMARY OF OUTPUT: MAXIMUM LIKELIHOOD SPATIAL LAG (METHOD = FULL)
-----
Data set          : 202409
Weights matrix   : tnic3
Dependent Variable : sue
                    Number of Observations: 2177
Mean dependent var : -0.0003
                    Number of Variables   : 3
S.D. dependent var : 0.0242
                    Degrees of Freedom   : 2174
Pseudo R-squared   : 0.0204
Spatial Pseudo R-squared: 0.0040
Sigma-square ML    : 0.001
                    Log likelihood      : 5026.
S.E of regression  : 0.024
                    Akaike info criterion : -10047.
                    Schwarz criterion    : -10030.

(continues on next page)

```

(continued from previous page)

↳ 303

```
↳-
      Variable      Coefficient      Std.Error      z-Statistic
↳Probability
-----
↳-
      CONSTANT      -0.0008998      0.0005767      -1.5602720      0.
↳1186956
      ret          0.0067306      0.0022802      2.9516958      0.
↳0031603
      W_sue        0.2247160      0.0440540      5.1009164      0.
↳0000003
=====
===== END OF REPORT
=====
```

```
# Show spatial regression results
ols = spreg.OLS(y=y['sue'].values, x=y[['ret']].values, w=w,
                 robust='white', spat_diag=True, moran=True,
                 name_x=['ret'], name_y='sue', name_w=scheme,
                 name_ds=str(rebaldate))
print(ols.summary)
```

```
REGRESSION
-----
SUMMARY OF OUTPUT: ORDINARY LEAST SQUARES
-----
Data set          : 202409
Weights matrix   : tnic3
Dependent Variable : sue
                   Number of Observations:  ↳
                   ↳2177
Mean dependent var : -0.0003
                   Number of Variables :  ↳
                   ↳2
S.D. dependent var : 0.0242
                   Degrees of Freedom :  ↳
                   ↳2175
R-squared          : 0.0042
Adjusted R-squared : 0.0037
Sum squared residual: 1.273
                   F-statistic      : 9.
                   ↳0806
Sigma-square      : 0.001
                   Prob(F-statistic) : 0.
                   ↳002613
S.E. of regression : 0.024
                   Log likelihood   : 5014.
                   ↳003
Sigma-square ML   : 0.001
                   Akaike info criterion : -10024.
                   ↳005
S.E. of regression ML: 0.0242
                   Schwarz criterion  : -10012.
                   ↳634

White Standard Errors
-----
      Variable      Coefficient      Std.Error      t-Statistic
```

(continues on next page)

(continued from previous page)

```

↳Probability
-----
↳-
CONSTANT      -0.00011401      0.0006542      -1.7428829      0.
↳0814954      ret      0.00069264      0.0032943      2.1025231      0.
↳0356221

-----
↳-
REGRESSION DIAGNOSTICS
MULTICOLLINEARITY CONDITION NUMBER      1.628

TEST ON NORMALITY OF ERRORS
TEST                      DF      VALUE      PROB
Jarque-Bera                2      4787.738      0.0000

DIAGNOSTICS FOR HETEROSKEDASTICITY
RANDOM COEFFICIENTS
TEST                      DF      VALUE      PROB
Breusch-Pagan test          1      63.692      0.0000
Koenker-Bassett test         1      13.760      0.0002

DIAGNOSTICS FOR SPATIAL DEPENDENCE
TEST                      MI/DF      VALUE      PROB
Moran's I (error)          0.0516      5.283      0.0000
Lagrange Multiplier (lag)  1      26.684      0.0000
Robust LM (lag)            1      0.627      0.4285
Lagrange Multiplier (error) 1      27.229      0.0000
Robust LM (error)          1      1.172      0.2790
Lagrange Multiplier (SARMA) 2      27.856      0.0000

===== END OF REPORT
=====

```

```

# Show Moran's I
print(f"Quarter: {rebaldate}")
DataFrame({ "Moran's I": mi.I, "Expected": mi.EI, "p_norm": mi.p_norm },
          index=[ "under assumption of normality" ])

```

Quarter: 202409

	Moran's I	Expected	p_norm
under assumption of normality	0.051519	-0.00046	1.401822e-07

```

# Show ordinary regression results
results = DataFrame(out)
results.index = bd.to_datetime(bd.endmo(results.index))
summary = dict()
for col in results.columns:
    model = smf.ols(f'{col} ~ 1', data=results).fit()
    robust = smf.ols(f'{col} ~ 1', data=results) \
        .fit(cov_type='HAC', cov_kwds={'maxlags': 6})

```

(continues on next page)

(continued from previous page)

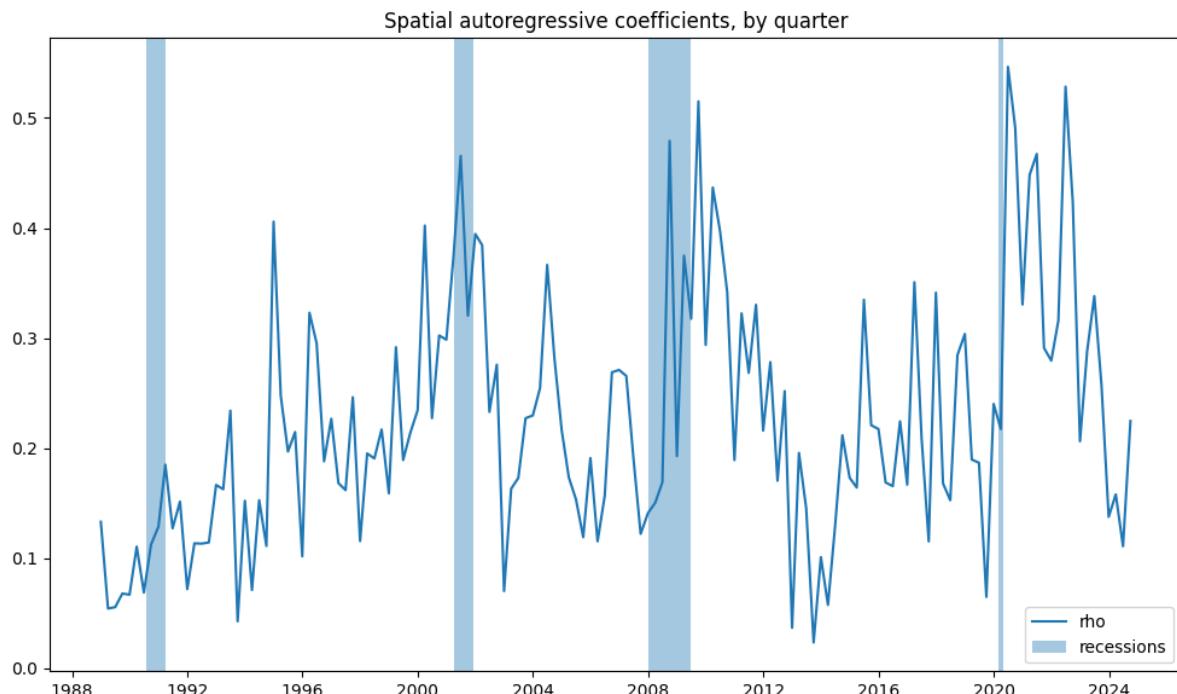
```
summary[col] = dict(mean=float(model.params.iloc[0]),
                     stderr=float(model.bse.iloc[0]),
                     t=float(model.tvalues.iloc[0]),
                     nw_stderr=float(robust.bse.iloc[0]),
                     nw_t=float(robust.tvalues.iloc[0]))
print("Tests of statistical significance")
DataFrame(summary)
```

Tests of statistical significance

	moran	rho	beta
mean	0.053146	0.224189	0.011726
stderr	0.002577	0.009331	0.000690
t	20.623401	24.027336	17.006094
nw_stderr	0.004606	0.017202	0.000952
nw_t	11.538987	13.032706	12.323118

To track changes over time, we plot the strength of spatial coefficients on a quarterly basis. Additionally, we highlight U.S. recession periods in blue, as times of economic stress may amplify the strength of linkages.

```
# Visualize autoregressive coefficients by quarter
fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(results['rho'])
ax.set_title("Spatial autoregressive coefficients, by quarter")
for a,b in vspans:
    if a >= min(results.index):
        ax.axvspan(a, min(b, max(results.index)), alpha=0.4)
plt.legend(['rho', 'recessions'])
plt.tight_layout()
plt.show()
```



CHAPTER
TWENTYFOUR

FOMC TOPIC MODELING

Our discussions of the economy may sometimes ring in the ears of the public with more certainty than is appropriate - Jerome Powell

The Federal Open Market Committee (FOMC) meeting minutes may reveal the Federal Reserve's economic outlook, policy decisions, and potential future actions. Analyzing these minutes can help identify key topics discussed over time, shedding light on trends in monetary policy and their implications for financial markets. We review methods for accessing and pre-processing textual data, including the use MongoDB for storing and retrieving unstructured data. Several topic modeling algorithms, including Latent Semantic Analysis (LSA), Latent Dirichlet Allocation (LDA), Non-negative Matrix Factorization (NMF), and Probabilistic Latent Semantic Indexing (PLSI), are applied to uncover patterns within the minutes.

```
# By: Terence Lim, 2020-2025 (terence-lim.github.io)
import numpy as np
import pandas as pd
from pandas import Series, DataFrame
import os
import sklearn.feature_extraction, sklearn.decomposition
from sklearn.decomposition import TruncatedSVD, LatentDirichletAllocation, NMF
from sklearn.cluster import KMeans
from scipy.special import softmax
from wordcloud import WordCloud
import wordcloud
import matplotlib.pyplot as plt
from finds.database import MongoDB
from finds.unstructured import Unstructured
from finds.utils import Store
from finds.readers import FOMCReader, Alfred
from pprint import pprint
from secret import credentials, paths
# %matplotlib qt
VERBOSE = 0
```

```
## retrieve recessions dates for plotting
alf = Alfred(api_key=credentials['fred']['api_key'])
vspan = alf.date_spans(series_id='USREC')
```

24.1 FOMC meeting minutes

The FOMC holds eight scheduled meetings per year, with additional meetings as necessary. The minutes from these meetings are released three weeks after the policy decision, offering crucial insights into the Federal Reserve's economic stance and possible future monetary policy actions. These insights can influence interest rates, inflation expectations, and overall market conditions.

For official meeting schedules and minutes, visit: [Federal Reserve FOMC Calendar](#).

24.1.1 FinDS fomcreader module

The `fomcreader` module in the FinDS packages offers functions for searching and retrieving meeting minutes from the Federal Reserve website.

```
# Helper class to scrape FOMC meeting minutes from Federal Reserve website
minutes = FOMCReader()

# Show number and date range of minutes available from website
DataFrame({'dates': len(minutes), 'start': min(minutes), 'end': max(minutes)},
          index=['FOMC minutes'])
```

```
100%|██████████| 27/27 [00:16<00:00,  1.66it/s]
```

	dates	start	end
FOMC minutes	256	19930203	20250129

24.1.2 MongoDB

MongoDB is a NoSQL (“not only SQL”) document-oriented database designed to efficiently manage unstructured and semi-structured data. Unlike traditional relational databases, MongoDB does not enforce a fixed schema, allowing for greater flexibility in handling diverse data formats. Each document (record) within a collection (table equivalent) can contain different fields and structures, including key-value pairs, arrays, and embedded subdocuments.

```
# store unstructured minutes text data in MongoDB
from pprint import pprint
mongodb = MongoDB(**credentials['mongodb'], verbose=VERBOSE)
print('uptime:', mongodb.client.admin.command("serverStatus")['uptime'])
fomc = Unstructured(mongodb, 'FOMC')
```

```
177.0
```

```
# retrieve keys (dates) of minutes previously retrieved and stored locally
dates = fomc['minutes'].distinct('date')
```

```
# fetch new minutes from FOMC site
docs = {d: minutes[d] for d in minutes if d not in dates}
print("New minutes:")
pprint([f"{k}: {len(v)} chars" for k,v in docs.items()])
```

```
New minutes:
['20250129: 54324 chars',
 '20240612: 43860 chars',
 '20240731: 44839 chars',
 '20240918: 44372 chars',
 '20241107: 49178 chars',
 '20241218: 45712 chars']
```

Preprocessing Meeting Minutes

1. Extract relevant sections of the minutes, starting from:
 - “Review of Monetary Policy Strategy, Tools, and Communications”
 - “Developments in Financial Markets”
 - “Discussion of Guidelines for Policy Normalization”
 - “Financial Developments and Open Market Operations”
 - “Discussion of the Economic Outlook”
 - “The information reviewed at this meeting”
 - “The staff presented several briefings”
2. Remove text following the adjournment line or the date scheduled for the next meeting. Exclude:
 - Notation votes, approvals of minutes, signatures, and footnotes
 - Intermeeting conference call discussions

Once extracted, store and retrieve all meeting minutes from MongoDB for further processing.

```
# Helper function to trim minutes text
def edit(text: str) -> str:
    """Helper to spawn editor and write/edit/read to tempfile"""
    import subprocess
    import tempfile
    with tempfile.NamedTemporaryFile(suffix=".tmp") as f: # save temp file
        f.write(text.encode("utf-8"))
        f.flush()
        subprocess.call([os.environ.get('EDITOR', 'emacs'), "-nw", f.name])
        f.seek(0)
    return f.read().decode("utf-8") # keep edited text
```

```
if docs:
    # to edit out head and tail of each document
    results = list()
    for date, initial_message in docs.items():
        edited_text = edit(initial_message)
        results.append({'date': date, 'text' : edited_text})
    results = sorted(results, key = lambda x: x['date']) # sort by date

    # save edited docs
    Store(paths['scratch'] / 'fomc', ext='gz').dump(results, f"{{max(docs.keys())}}.json")
    for doc in results: # store docs for new dates
        fomc.insert('minutes', doc, keys=['date'])
```

Retrieve all minutes that were stored locally in MongoDB

```
docs = Series({doc['date']: doc['text'] for doc in fomc.select('minutes')},  
              name='minutes').sort_index()  
DataFrame(docs)
```

```
minutes  
19930203 The Manager of the System Open Market Account ...  
19930323 The Deputy Manager for Domestic Operations rep...  
19930518 The Manager of the System Open Market Account ...  
19930707 The Deputy Manager for Domestic Operations rep...  
19930817 The Deputy Manager for Domestic Operations rep...  
...  
20240731 Developments in Financial Markets and Open Mar...  
20240918 Developments in Financial Markets and Open Mar...  
20241107 Developments in Financial Markets and Open Mar...  
20241218 Developments in Financial Markets and Open Mar...  
20250129 Developments in Financial Markets and Open Mar...
```

[256 rows x 1 columns]

24.2 Text pre-processing

Text preprocessing involves cleaning and preparing raw text data by removing noise, standardizing formats, and converting text into a structured form suitable for analysis. This process typically includes tokenization, removing stop words, and vectorizing the text.

24.2.1 Tokenization

Tokenization splits text into smaller units called **tokens**, typically words, though they may also be phrases or subwords. The pattern of tokenization is often defined using regular expressions.

The original **casing** of words may be preserved if it carries meaningful information, such as distinguishing proper nouns, acronyms, or sentence boundaries. Some models may interpret all uppercase words as organizations.

24.2.2 Regular expressions

Regular expressions (**regex**) define search patterns for text processing, often used in tokenization to find word boundaries and remove unwanted characters.

Basic Regular Expressions

Expression	Description
\d	Matches a digit (0-9)
\w	Matches a word character (ASCII letter, digit, or underscore)
\s	Matches a whitespace character (space, tab, newline)
\D	Matches a non-digit character
\W	Matches a non-word character
\S	Matches a non-whitespace character
[...]	Matches one of the characters in the brackets
[a-zA-Z]	Matches any letter (uppercase or lowercase)
[^a]	Matches any character except a
\b	Matches a word boundary
.	Matches any character except a line break
\.	Matches a period character
^	Matches the start of a string
\$	Matches the end of a string
+	Matches one or more occurrences
*	Matches zero or more occurrences
?	Matches zero or one occurrence
	Acts as an OR operator
(...)	Defines a capturing group

24.2.3 Stopwords

Common, uninformative words (e.g., “the,” “is,” “and”) can be removed to focus on meaningful content in the analysis. Corpus-specific uninformative words (such as calendar-related terms in FOMC minutes) can also be excluded.

```
# ignore these stop words
StopWords = [w for w in set(wordcloud.STOPWORDS) if "!" not in w]
StopWords += ['january', 'february', 'march', 'april', 'may', 'june',
              'july', 'august', 'september', 'october', 'november',
              'december', 'first', 'second', 'third', 'fourth', 'twelve',
              'participants', 'members', 'meeting']
```

24.2.4 Vectorization

Vectorization transforms text into numerical representations suitable for analysis.

- **Bag-of-Words (BoW):** Represents a document by the frequency of its words, ignoring grammar and word order.
- **N-grams:** Treats consecutive words as distinct items rather than isolated words.
- Indexing: Assigns unique integer indexes to words in the corpus.
- **TF-IDF (Term Frequency-Inverse Document Frequency):** Measures word importance based on its frequency in a document relative to its occurrence across all documents.

- The `scikit-learn` package provides `TfidfVectorizer` for TF-IDF vectorization and `CountVectorizer` for raw word counts.

```
# To vectorize the input words
#ngram_range = (1, 1)    # unigrams
#ngram_range = (2, 2)    # bigrams
ngram_range = (1, 2)    # unigrams and bigrams
max_df, min_df, max_features = 0.5, 6, 5000 # some reasonable constraints
tfidf_vectorizer = sklearn.feature_extraction.text.TfidfVectorizer(
    strip_accents='unicode',
    lowercase=True,
    stop_words=StopWords,
    ngram_range=ngram_range,
    max_df=max_df,
    min_df=min_df,
    max_features=max_features,
    token_pattern=r"\b[\^\d\W] [\^\d\W] [\^\d\W]+\b") #r'\b[\^\d\W]+\b'
tf_vectorizer = sklearn.feature_extraction.text.CountVectorizer(
    strip_accents='unicode',
    lowercase=True,
    stop_words=StopWords,
    ngram_range=ngram_range,           # (2, 2) for bigrams
    max_df=max_df,
    min_df=min_df,
    max_features=max_features,
    token_pattern=r"\b[\^\d\W] [\^\d\W] [\^\d\W]+\b")
```

24.3 Topic Modeling

Topic modeling applies statistical techniques to discover latent topics within a collection of documents.

24.3.1 Latent Semantic Analysis (LSA)

LSA uses singular value decomposition (SVD) to analyze relationships between terms and documents. By reducing dimensionality, it groups terms and documents that frequently co-occur in a lower-dimensional space, revealing underlying topics.

24.3.2 Latent Dirichlet Allocation (LDA)

LDA assumes each document is a mixture of topics, and each topic is a mixture of words. It iteratively assigns words to topics based on probability distributions, refining topic assignments over multiple iterations.

24.3.3 Non-negative Matrix Factorization (NMF)

NMF decomposes a term-document matrix into two non-negative matrices:

- A basis matrix representing topics
- A coefficient matrix representing the distribution of topics in documents

By optimizing these matrices, NMF extracts meaningful topics from the text.

24.3.4 Probabilistic Latent Semantic Indexing (PLSI)

PLSI models documents as mixtures of topics, estimating probability distributions of words and topics iteratively until convergence. It can be implemented using NMF with generalized Kullback-Leibler divergence as the loss function.

- scikit-learn decomposition documentation
- Example of topic extraction with NMF & LDA

```
# Define models
n_components = 4      # fix number of latent topics
algos = {
    'LSA': (TruncatedSVD(n_components=n_components),
              tfidf_vectorizer),
    'LDA': (LatentDirichletAllocation(n_components=n_components,
                                       learning_method='batch', #'online',
                                       # learning_offset = 50.0,
                                       max_iter = 40,
                                       random_state = 42),
              tf_vectorizer),
    'PLSI': (NMF(n_components=n_components,
                  beta_loss='kullback-leibler',
                  solver='mu',
                  alpha_W=0.00005,
                  alpha_H=0.00005,
                  l1_ratio=0.5,
                  max_iter=1000,
                  random_state = 42),
              tfidf_vectorizer),
    'NMF': (NMF(n_components=n_components,
                  random_state=42,
                  beta_loss='frobenius',
                  alpha_W=0.00005,
                  alpha_H=0.00005,
                  l1_ratio=0.5),
              tfidf_vectorizer)}
```

Once topics are extracted, they can be visualized over time to observe trends in FOMC discussions.

```

# Fit and plot models
scores = dict()      # to save model coefficients
topics = dict()      # to save dates of primary topic
for ifig, (name, (base, vectorizer)) in enumerate(algos.items()):

    # vectorize the input words
    vectorized = vectorizer.fit_transform(docs.to_list())
    feature_names = vectorizer.get_feature_names_out()

    # fit model and transform inputs
    model = base.fit(vectorized)
    transformed = model.transform(vectorized)

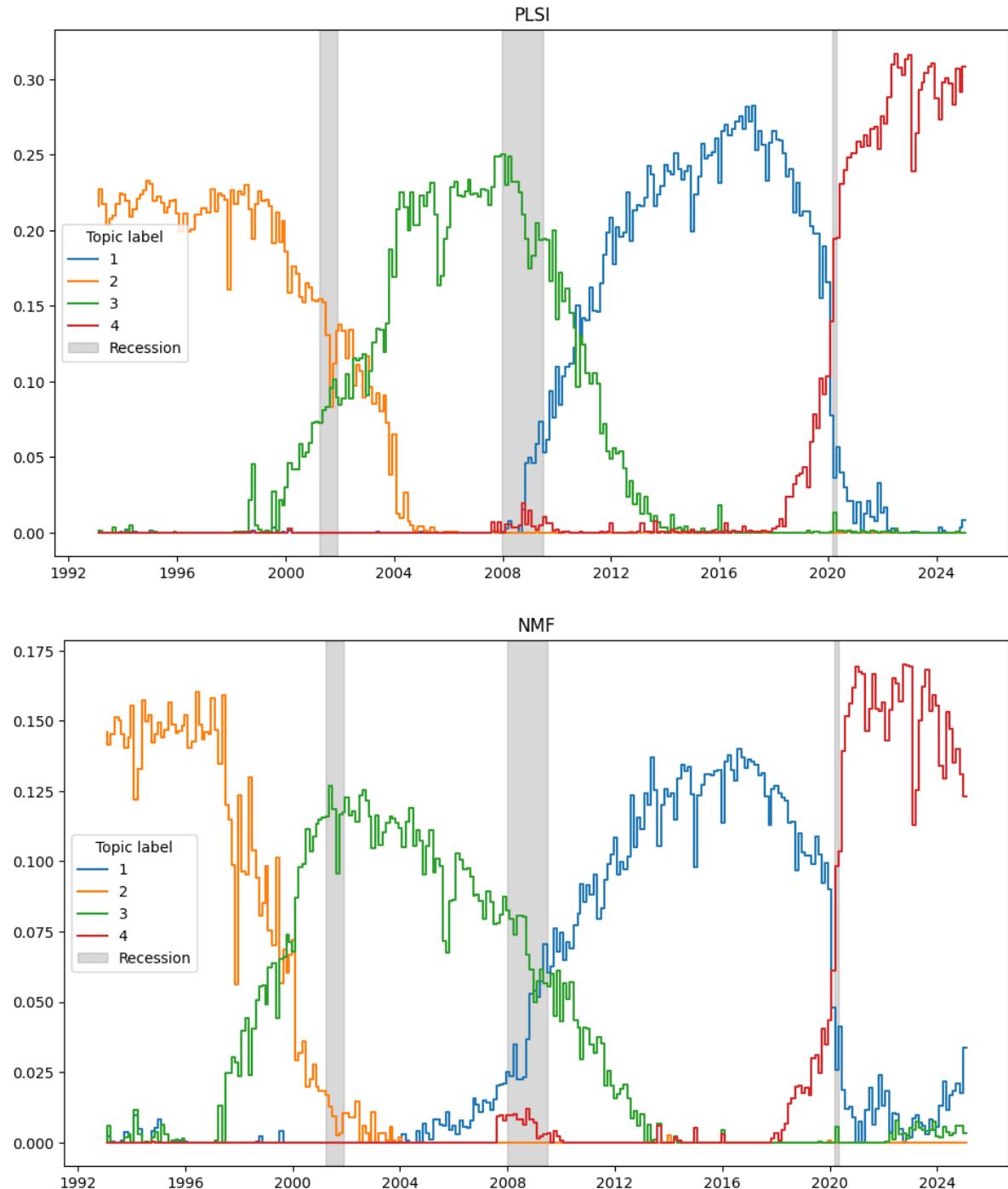
    # save the model fitted coefficients
    if name == 'LSA':  # Additional step for LSA
        kmeans = KMeans(n_clusters=n_components, n_init=5, random_state=37) \
            .fit(transformed)      # find centroids of the latent factors
        transformed = kmeans.transform(transformed)  # distance to centroid
        transformed = -(transformed / transformed.sum(axis=1, keepdims=True))
        scores[name] = softmax(model.components_, axis=1)  # scale word scores
    else:
        scores[name] = model.components_

    # plot topic scores over time
    fig, ax = plt.subplots(num=1 + ifig, clear=True, figsize=(10, 6))
    dates = pd.DatetimeIndex(docs.index.astype(str))
    ax.step(dates, transformed, where='pre')
    for a,b in vspans:
        if b >= min(dates):
            ax.axvspan(a, min(b, max(dates)), alpha=0.3, color='grey')
    ax.set_title(name)
    ax.legend([f"i+1" for i in range(n_components)] + ['Recession'],
              loc='center left', title='Topic label')
    plt.tight_layout(pad=2)

    # save dates of primary topic
    for topic in range(transformed.shape[1]):
        arg = DataFrame({'t': np.argmax(transformed, axis=1),
                         'dt': docs.index})
        dates = (arg!=arg.shift()).cumsum().groupby('t').agg(['first', 'last']) - 1
        dates['topic'] = arg.loc[dates.iloc[:,1], 't'].values
        topics[name] = {topic: [(arg['dt'].iloc[row[0]], arg['dt'].iloc[row[1]]) \
            for row in dates.itertuples(index=False, name=None) \
            if row[2] == topic]
                       for topic in range(transformed.shape[1])}

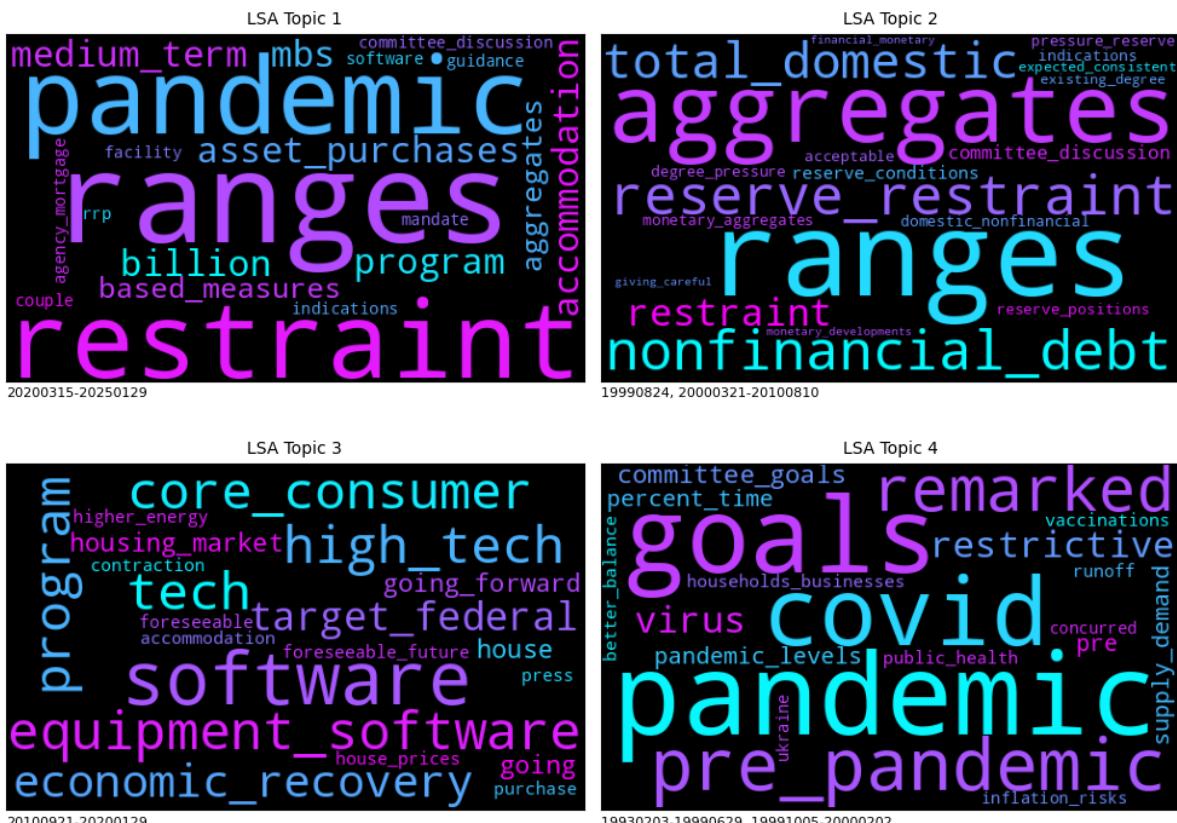
```



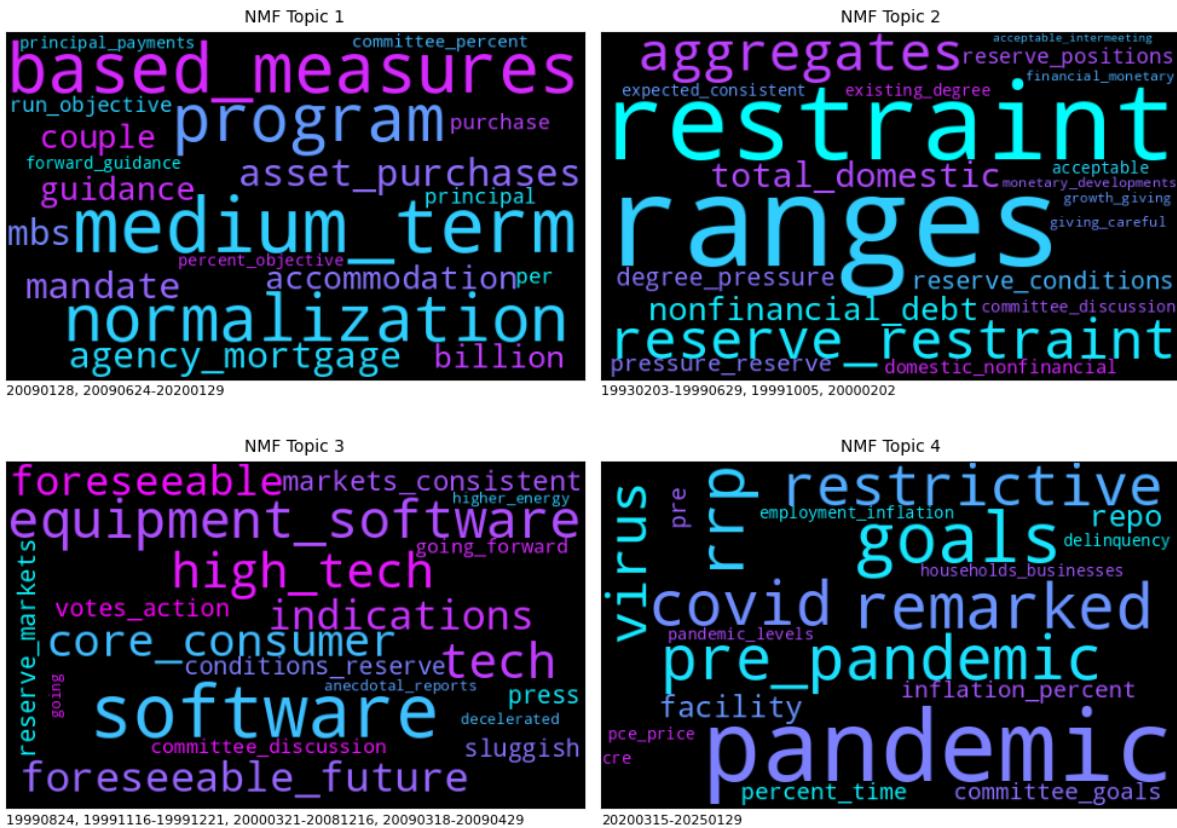


24.3.5 Feature importance

The most significant words for each topic are identified and visualized using the `WordCloud` package, providing an intuitive representation of key themes.







CHAPTER
TWENTYFIVE

MANAGEMENT SENTIMENT ANALYSIS

Life is a math equation. In order to gain the most, you have to know how to convert negatives into positives - Anonymous

Sentiment analysis helps quantify the tone of financial disclosures, revealing whether a company's management expresses optimism, caution, or concern. This analysis can be conducted using dictionary-based methods, which rely on predefined sentiment word lists, or more sophisticated machine learning models, such as large language models (LLMs). We explore the application of sentiment analysis in financial documents, focusing on the Loughran-MacDonald dictionary and the SEC's EDGAR system for retrieving company filings. Specifically, we analyze sentiment trends in 10-K Management Discussion and Analysis (MD&A) sections and their relationship with stock returns.

```
# By: Terence Lim, 2020-2025 (terence-lim.github.io)
import numpy as np
import pandas as pd
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import CountVectorizer
from tqdm import tqdm
from finds.database import SQL, RedisDB
from finds.structured import BusDay, CRSP, Signals
from finds.unstructured import Edgar
from finds.readers import Alfred
from finds.recipes import weighted_average, fractile_split
from finds.utils import Store
from secret import credentials, paths, CRSP_DATE
# %matplotlib qt
VERBOSE = 0
LAST_YEAR = CRSP_DATE // 10000
```

```
sql = SQL(**credentials['sql'], verbose=VERBOSE)
user = SQL(**credentials['user'], verbose=VERBOSE)
bd = BusDay(sql)
rdb = RedisDB(**credentials['redis'])
crsp = CRSP(sql=sql, bd=bd, rdb=rdb, verbose=VERBOSE)
signals = Signals(user)
store = Store(paths['scratch'], ext='pkl')
```

25.1 Sentiment analysis

Dictionary-based (or lexicon-based) sentiment analysis relies on predefined word lists, where words are assigned sentiment scores along with intensity levels. While this approach is straightforward and computationally efficient, it struggles with complex linguistic phenomena such as negations, irony, and sarcasm, which can lead to reduced accuracy. On the other hand, large language models excel at understanding context, syntax, and semantics, making them more effective for sentiment analysis of intricate financial texts. However, despite their limitations, dictionary-based methods remain popular due to their ease of implementation and transparent results.

25.1.1 Loughran-MacDonald dictionary

Loughran and McDonald (2011) found that general-purpose sentiment lexicons were not well-suited for financial statement analysis. To address this, they developed a dictionary based on company 10-K filings, categorizing words into seven sentiment-related groups relevant to finance: “negative,” “positive,” “litigious,” “uncertainty,” “constraining,” and “superfluous.” This domain-specific approach provides a more accurate reflection of sentiment in financial disclosures.

- Loughran-MacDonald Master Dictionary

The dictionary includes:

- A list of positive sentiment words
- A list of negative sentiment words
- A set of stop words to ignore

```
# to download from google drive, need this prefix and the source file id
_prefix = "https://drive.google.com/uc?export=download&id="
source = _prefix + '1ptUgGVeeUGhCbaKL14Ri3Xi5xOKkPkUD'
# source = "Loughran-McDonald_MasterDictionary_1993-2023.csv"
df = pd.read_csv(source, sep=', ')
```

```
# sets of positive and negative sentiment words
words = {'positive': set(df.loc[df['Positive'] != 0, 'Word'].str.lower()),
          'negative': set(df.loc[df['Negative'] != 0, 'Word'].str.lower())}
```

Positive sentiment words

```
DataFrame(words['positive'], columns=['Positive Words'])
```

```
Positive Words
0      outperform
1      rebounded
2      efficiency
3      honored
4      leadership
...
349      improves
350  collaborations
351      innovators
352      advantageous
353      honoring
```

[354 rows x 1 columns]

Negative sentiment words

```
DataFrame(words['negative'], columns=['Negative Words'])
```

```

  Negative Words
0      threatening
1      immature
2  falsifications
3  uncontrolled
4      panic
...
2350    objections
2351      coercion
2352  extenuating
2353      lying
2354 contradiction

[2355 rows x 1 columns]
```

List of stop words to ignore

```
# stopwords
generic = ['ME', 'MY', 'MYSELF', 'WE', 'OUR', 'OURS', 'OURSELVES',
           'YOU', 'YOUR', 'YOURS', 'YOURSELF', 'YOURSELVES',
           'HE', 'HIM', 'HIS', 'HIMSELF', 'SHE', 'HER', 'HERS',
           'HERSELF', 'IT', 'ITS', 'ITSELF', 'THEY', 'THEM',
           'THEIR', 'THEIRS', 'THEMSELVES', 'WHAT', 'WHICH',
           'WHO', 'WHOM', 'THIS', 'THAT', 'THESE', 'THOSE',
           'AM', 'IS', 'ARE', 'WAS', 'WERE', 'BE', 'BEEN',
           'BEING', 'HAVE', 'HAS', 'HAD', 'HAVING', 'DO',
           'DOES', 'DID', 'DOING', 'AN', 'THE', 'AND', 'BUT',
           'IF', 'OR', 'BECAUSE', 'AS', 'UNTIL', 'WHILE', 'OF',
           'AT', 'BY', 'FOR', 'WITH', 'ABOUT', 'BETWEEN',
           'INTO', 'THROUGH', 'DURING', 'BEFORE', 'AFTER',
           'ABOVE', 'BELOW', 'TO', 'FROM', 'UP', 'DOWN', 'IN',
           'OUT', 'ON', 'OFF', 'OVER', 'UNDER', 'AGAIN',
           'FURTHER', 'THEN', 'ONCE', 'HERE', 'THERE', 'WHEN',
           'WHERE', 'WHY', 'HOW', 'ALL', 'ANY', 'BOTH', 'EACH',
           'FEW', 'MORE', 'MOST', 'OTHER', 'SOME', 'SUCH',
           'NO', 'NOR', 'NOT', 'ONLY', 'OWN', 'SAME', 'SO',
           'THAN', 'TOO', 'VERY', 'CAN', 'JUST', 'SHOULD',
           'NOW', 'AMONG']
```

```
# CountVectorizer to tokenize and code counts of inputs words
vectorizer = CountVectorizer(strip_accents='unicode',
                             lowercase=True,
                             stop_words=generic,
                             token_pattern=r"\b[\^d\W][^\d\W][^\d\W]+\b")
```

```
# vectorizer to encode sentiment dictionary words
sentiment_vectorizer = CountVectorizer(strip_accents='unicode',
                                       lowercase=True,
                                       token_pattern=r"\b[\^d\W][^\d\W][^\d\W]+\b")
sentiment_vectorizer.fit([" ".join(words['positive'].union(words['negative']))])

# create a lookup Series and a score vector for computing net sentiment
```

(continues on next page)

(continued from previous page)

```
features = Series(sentiment_vectorizer.get_feature_names_out())
sentiment_points = (features.isin(words['positive'])).astype(int).values
                  - features.isin(words['negative'])).astype(int).values)
```

25.2 10-K Management discussion and analysis

A 10-K is a comprehensive annual report that publicly traded companies must file with the SEC. It provides an in-depth view of the company's financial health and operational performance. Key sections include the business description, management discussion and analysis (MD&A), risk factors, and financial statements.

25.2.1 SEC Edgar website

The SEC EDGAR (Electronic Data Gathering, Analysis, and Retrieval) system is an online database managed by the U.S. Securities and Exchange Commission (SEC). It provides public access to financial filings, including annual and quarterly reports (Forms 10-K and 10-Q), significant event disclosures (Form 8-K), beneficial ownership reports (Schedule 13D), insider sales reports (Form 144), proxy statements, and registration statements (S-1).

- SEC EDGAR Search and Access

Additionally, the SEC has recently released an EDGAR Application Programming Interface (API): [EDGAR API](#)

Loughran and MacDonald have also made SEC EDGAR data files and other textual resources available in their research repository: [SEC EDGAR Data](#)

25.2.2 FinDS edgar module

The `edgar` module in the FinDS package provides functions for

- Searching and retrieving company filings from the SEC Edgar website
- Storing and indexing data locally
- Identifying and extracting specific sections of text, such as MD&A

Retrieve MD&A section text from 10-K's:

```
# open 10-K archive of MD&A text
ed = Edgar(paths['10X'], zipped=True, verbose=VERBOSE)
item, form = 'mدا10K', '10-K'
rows = DataFrame(ed.open(form=form, item=item))
Series((rows['date'] // 10000).astype(int)).value_counts().sort_index().to_frame()
```

	count
date	
1993	2
1994	1362
1995	1674
1996	2960
1997	4461
1998	4501
1999	4364
2000	4287

(continues on next page)

(continued from previous page)

2001	4168
2002	4400
2003	5478
2004	5203
2005	5090
2006	4999
2007	4954
2008	5033
2009	5273
2010	5050
2011	4863
2012	4730
2013	4617
2014	4643
2015	4720
2016	4612
2017	4488
2018	4431
2019	4417
2020	4388
2021	4615
2022	4766
2023	4641
2024	4501

For all investment universe stocks between 1993 through the present, compute the average sentiment, and change in sentiment, of 10-K's

```
# retrieve usual universe at end of each year
univs = {year: crsp.get_universe(bd.endyr(year-1)).assign(year=year)
         for year in range(1992, LAST_YEAR + 1)}
permnos = rows['permno'].unique().astype(int)
DataFrame({'permnos': len(permnos),
           'documents': len(rows),
           'first': min(rows['date']),
           'last': max(rows['date'])},
           index=['10K-mdas'])
```

	permnos	documents	first	last
10K-mdas	14696	137691	19931129	20241231

We assume up to a 3-month lag when the 10-Ks are made available. For example, filings submitted between January and March of 2024 are assigned to fiscal year 2023.

```
# Compute average sentiment and change in sentiment for all companies and years
results = []
for permno in tqdm(permnos): # Loop over all permnos

    # retrieve all valid mda's for this permno by year
    mdas, dates = {}, {} # to collect mdas and doc dates for this permno
    docs = rows[rows['permno'].eq(permno)].to_dict('records')
    for doc in docs:
        year = bd.endmo(doc['date'], -3) // 10000 # assign fiscal year
        if (year in univs and (year not in mdas or doc['date'] < dates[year])):
            tokens = ed[doc['pathname']]
```

(continues on next page)

(continued from previous page)

```

if len(tokens):
    mdas[year] = tokens
    dates[year] = doc['date']

# compute sentiment as net sentiment word counts divided by doc length
if len(mdas):
    X = sentiment_vectorizer.transform(list(mdas.values())) \
        .dot(sentiment_points)
    X = np.divide(X, vectorizer.fit_transform(list(mdas.values())).sum())
    sentiment = {k:x for k,x in zip(mdas.keys(), X)}

# derive sentiment change and similarity scores by year
for year in sorted(mdas.keys()):
    result = {'year': year, 'permno': permno, 'date': dates[year]}
    result['mdasent'] = sentiment[year]
    result['currlen'] = len(mdas[year])
    if year-1 in mdas:
        result['prevlen'] = len(mdas[year-1])
        result['mdachg'] = sentiment[year] - sentiment[year-1]

    corpus = [mdas[year], mdas[year-1]]
    results.append(result)

```

```

0% | 0/14696 [00:00<?, ?it/s]/home/terence/env3.11/lib/python3.11/site-
  packages/sklearn/feature_extraction/text.py:408: UserWarning: Your stop_words_
  may be inconsistent with your preprocessing. Tokenizing the stop words generated_
  tokens ['about', 'above', 'after', 'again', 'all', 'among', 'and', 'any', 'are',
  'because', 'been', 'before', 'being', 'below', 'between', 'both', 'but', 'can',
  'did', 'does', 'doing', 'down', 'during', 'each', 'few', 'for', 'from', 'further
  ', 'had', 'has', 'have', 'having', 'her', 'here', 'hers', 'herself', 'him',
  'himself', 'his', 'how', 'into', 'its', 'itself', 'just', 'more', 'most', 'myself
  ', 'nor', 'not', 'now', 'off', 'once', 'only', 'other', 'our', 'ours', 'ourselves
  ', 'out', 'over', 'own', 'same', 'she', 'should', 'some', 'such', 'than', 'that',
  'the', 'their', 'theirs', 'them', 'themselves', 'then', 'there', 'these', 'they
  ', 'this', 'those', 'through', 'too', 'under', 'until', 'very', 'was', 'were',
  'what', 'when', 'where', 'which', 'while', 'who', 'whom', 'why', 'with', 'you',
  'your', 'yours', 'yourself', 'yourselves'] not in stop_words.
  warnings.warn(
100% |██████████| 14696/14696 [13:49<00:00, 17.71it/s]

```

```

# save in signals database
data = DataFrame.from_records(results)
data['rebaldate'] = bd.offset(data['date'])
print(signals.write(data, 'mdasent', overwrite=True),
      signals.write(data, 'mdachg', overwrite=True))

```

131045 107665

```

# right join data with univ, to identify universe companies with missing mda
data = pd.concat([data[data['year']==year]\n    .drop(columns=['year'])\n    .set_index('permno')\n    .join(univ[['year']], how='right')\n

```

(continues on next page)

(continued from previous page)

```

    .reset_index()
    for year, univ in univs.items() if year < LAST_YEAR],
    ignore_index=True)
# store temporary
store['sentiment'] = data

data = store['sentiment']

```

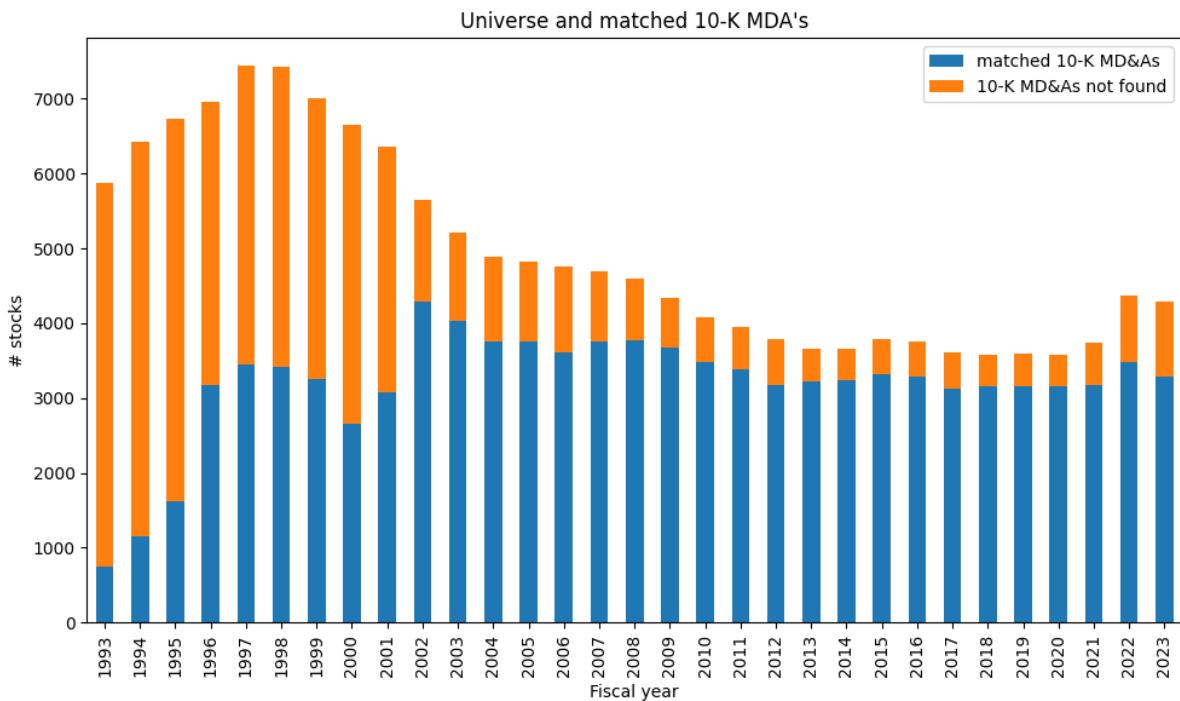
Exploratory analysis

Universe coverage by year:

```

# Stacked Bar Plot of universe coverage by year
y1 = data[data['mdasent'].notna()].groupby('year')['permno'].count()
y0 = data[data['mdasent'].isna()].groupby('year')['permno']\
    .count()\
    .reindex(y1.index)
fig, ax = plt.subplots(1, 1, clear=True, figsize=(10, 6))
y1.plot(kind='bar', label='matched 10-K MD&As', color='C0', ax=ax, rot=90)
y0.plot(kind='bar', label='10-K MD&As not found', color='C1', ax=ax, rot=90, \
    bottom=y1)
ax.set_ylabel('# stocks')
ax.set_xlabel('Fiscal year')
ax.set_title("Universe and matched 10-K MDA's")
ax.legend()
plt.tight_layout()

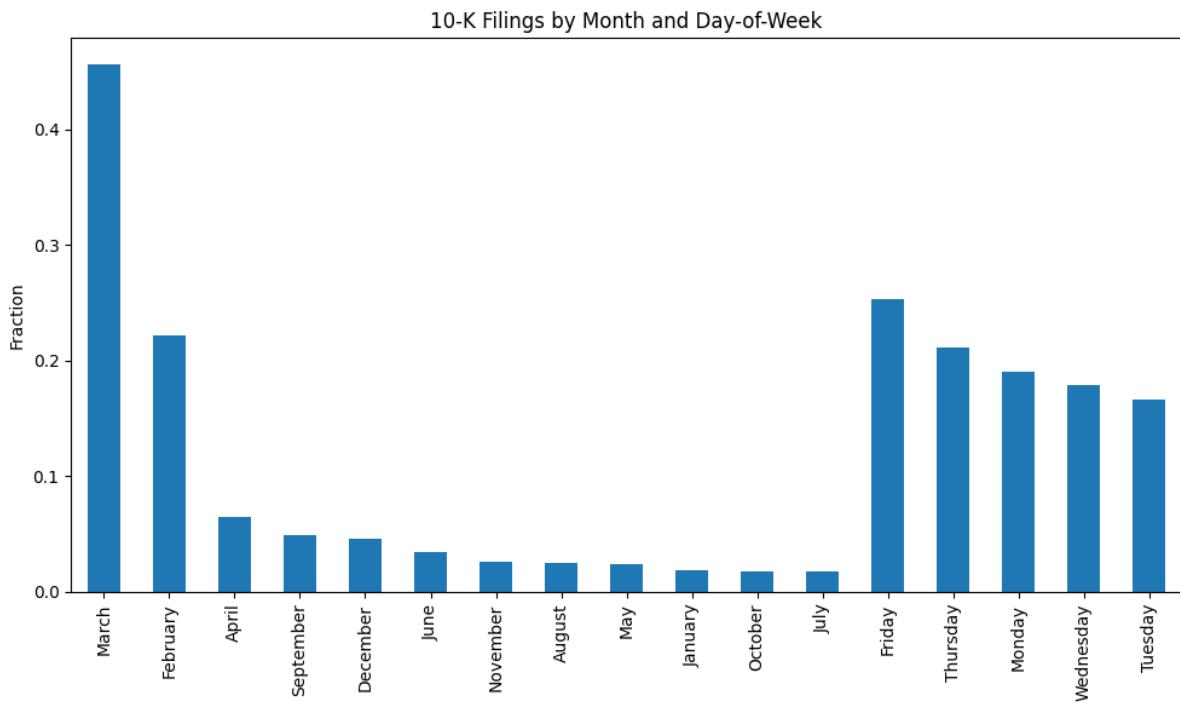
```



When are 10-K's filed?

```
# Stacked Bar Plot of filings date, by month and day-of-week
y = DataFrame.from_records([{'date': int(d),
                             'day': bd.datetime(int(d)).strftime('%A'),
                             'month': bd.datetime(int(d)).strftime('%B')}
                            for d in data.loc[data['mdasent'].notna(), 'date']])

z = pd.concat([y['month'].value_counts()/len(y),
               y['day'].value_counts()/len(y)])
fig, ax = plt.subplots(1, 1, clear=True, figsize=(10, 6))
z.plot(kind='bar', color='C0', ax=ax, rot=90)
ax.set_ylabel('Fraction')
ax.set_title("10-K Filings by Month and Day-of-Week")
plt.tight_layout()
```



Median sentiment and change in sentiment vs total corporate profits (of all US companies), by year:

```
alf = Alfred(api_key=credentials['fred']['api_key'], verbose=VERBOSE)
series_id = 'CP' # Corporate Profits
econ = alf(series_id).to_frame()
econ = econ.assign(year=econ.index // 10000).groupby('year').sum()

for sent, ylab in {'mdasent': 'sentiment', 'mdachg': 'sentiment change'}.items():
    print(sent, ylab)
    g = data[data['currlen'].gt(500)].dropna().groupby('year')
    iq1, iq2, iq3 = [g[sent].quantile(p) for p in [.25, .5, .75]]
    y = iq2.index.astype(int)
    fig, ax = plt.subplots(1, 1, clear=True, figsize=(10, 6))
    ax.step(y, iq2, ls='-', color='C1', where='pre')
    ax.fill_between(y, iq1, iq3, alpha=0.2, color='C1', step='pre')
    ax.set_title(f'{sent.upper()} by Fiscal Year of 10-K Filing')
    ax.set_xlabel("Fiscal Year")
    ax.set_ylabel(ylab)
    ax.legend([f'{sent.upper()} median', f"inter-quartile range"],
```

(continues on next page)

(continued from previous page)

```

        fontsize='small', loc='upper left')

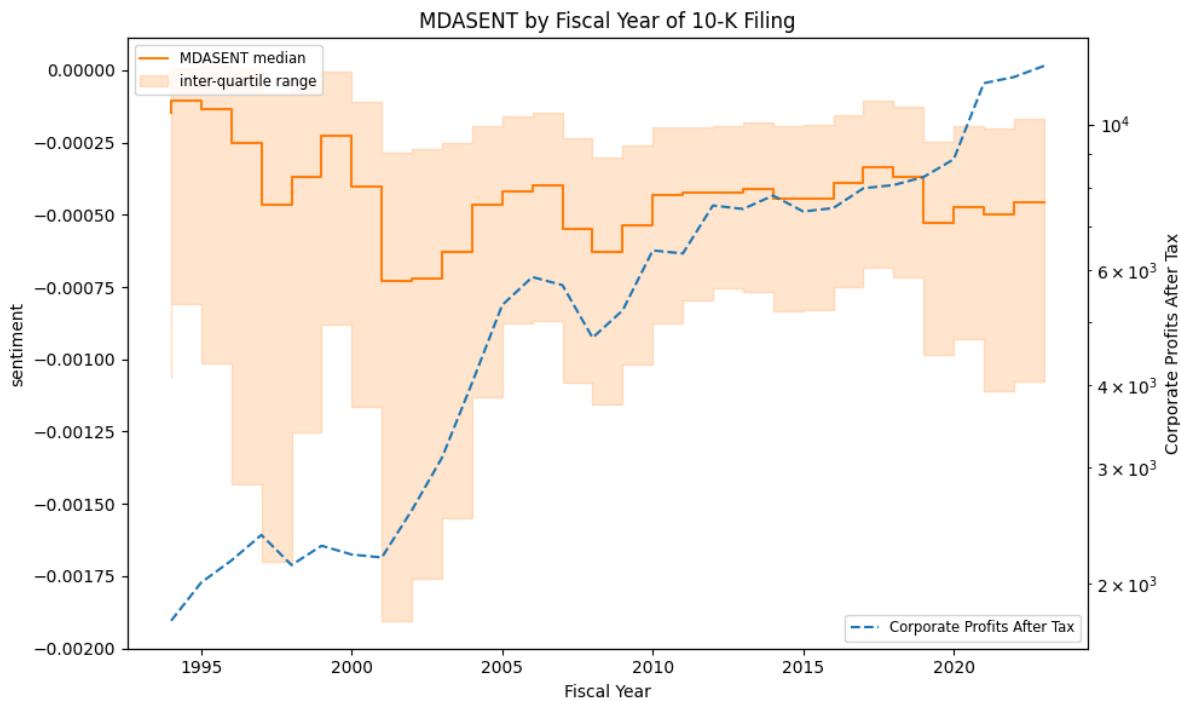
bx = ax.twinx()
econ[(econ.index >= min(y)) & (econ.index <= max(y))].plot(ls='--', ax=bx)
bx.legend([alf.header(series_id)[:27]], fontsize='small', loc='lower right')
bx.set_ylabel(alf.header(series_id)[:27])
bx.set_yscale('log')
plt.tight_layout()

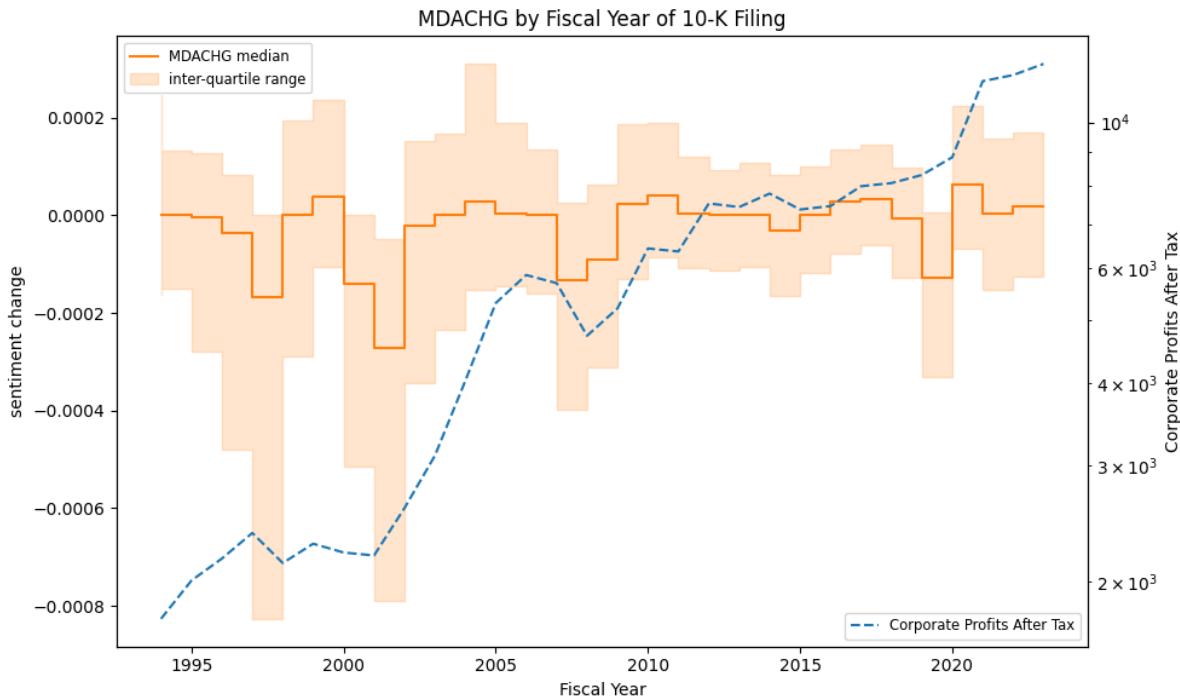
```

```

mdasent sentiment
mdachg sentiment change

```





25.2.3 Management sentiment and stock returns

We analyze the relationship between sentiment in 10-K filings and subsequent stock returns by constructing cap-weighted decile-spread returns. These are calculated for two periods:

1. The same calendar year (January–December) to examine contemporaneous relationships between sentiment and stock performance.
2. The following year (April–March) to explore an investable strategy based on sentiment changes after the release of 10-K reports for the prior fiscal year.

```

for ifig, key in enumerate(['mdasent', 'mdachg']):
    ret1 = {}      # to collect year-ahead spread returns
    ret0 = {}      # to collect current-year spread returns
    for year in tqdm(range(1999, max(data['year'])+1)):  # loop over years

        # compute current year average spread returns
        begyr = bd.begyr(year)
        endyr = bd.endyr(year)
        univ = data[data['year'] == year] \
            .dropna(subset=[key]) \
            .set_index('permno') \
            .join(crsp.get_cap(bd.offset(begyr, -1)), how='inner') \
            .join(crsp.get_ret(begyr, endyr), how='left')
        if len(univ):
            sub = fractile_split(univ[key], [10, 90])
            pos = weighted_average(univ.loc[sub==1, ['cap', 'ret']], 'cap')['ret']
            neg = weighted_average(univ.loc[sub==3, ['cap', 'ret']], 'cap')['ret']
            ret0[endyr] = {'ret': pos-neg, 'npos': sum(sub==1), 'nneg': sum(sub==3) }

    # compute year ahead average spread returns

```

(continues on next page)

(continued from previous page)

```

beg = bd.begmo(endyr, 4)
end = bd.endmo(endyr, 15)
univ = data[data['year'] == year] \
    .dropna(subset=[key]) \
    .set_index('permno') \
    .join(crsp.get_cap(bd.offset(beg, -1)), how='inner') \
    .join(crsp.get_ret(beg, end), how='left')
if len(univ):
    sub = fractile_split(univ[key], [10, 90])
    pos = weighted_average(univ.loc[sub==1, ['cap', 'ret']], 'cap')['ret']
    neg = weighted_average(univ.loc[sub==3, ['cap', 'ret']], 'cap')['ret']
    ret1[end] = {'ret': pos-neg, 'npos': sum(sub==1), 'nneg': sum(sub==3)}

# collect same-year and year-ahead average spread returns
r0 = DataFrame.from_dict(ret0, orient='index').sort_index()
r0.index = r0.index // 10000
r1 = DataFrame.from_dict(ret1, orient='index').sort_index()
r1.index = (r1.index // 10000) - 2

# plot same-year average spread returns
fig, ax = plt.subplots(nrows=2, clear=True, figsize=(10, 8), sharey=True)
r0['ret'].plot(kind='bar', ax=ax[0], width=.85, color="C0")
ax[0].axhline(r0['ret'].median(), linestyle=':', color='C0')
ax[0].axhline(r0['ret'].mean(), linestyle='-.', color='C0')
ax[0].set_title(f"Same Year (Jan-Dec) Returns")
ax[0].set_ylabel('annual returns')
ax[0].legend(['mean', 'median', 'Annual Spread Returns'])

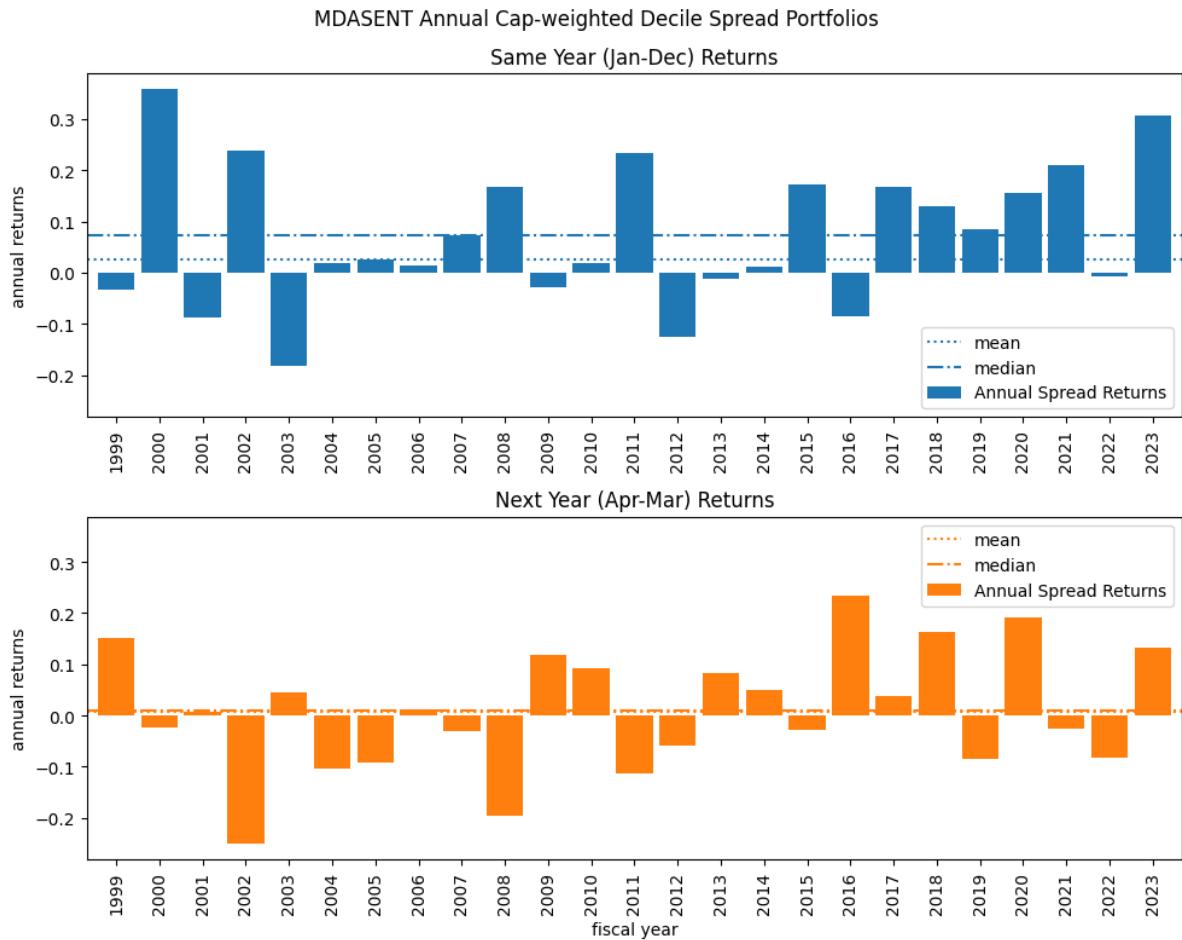
# plot year-ahead average spread returns
r1['ret'].plot(kind='bar', ax=ax[1], width=.85, color="C1")
ax[1].axhline(r1['ret'].median(), linestyle=':', color='C1')
ax[1].axhline(r1['ret'].mean(), linestyle='-.', color='C1')
ax[1].set_title(f"Next Year (Apr-Mar) Returns")
ax[1].set_ylabel('annual returns')
ax[1].legend(['mean', 'median', 'Annual Spread Returns'])
ax[1].set_xlabel('fiscal year')

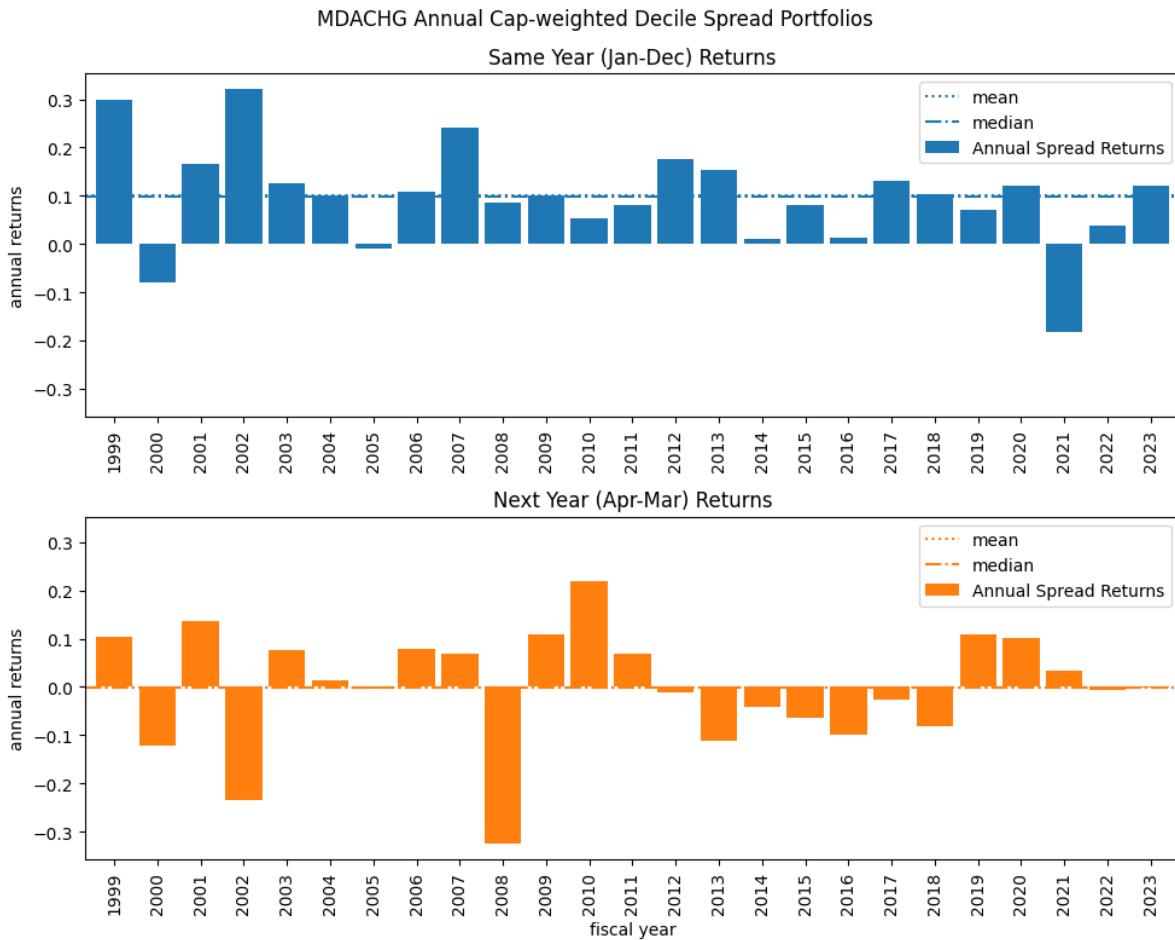
plt.suptitle(f"{key.upper()} Annual Cap-weighted Decile Spread Portfolios")
plt.tight_layout()

```

0% | 0/25 [00:00<?, ?it/s]

100% | ██████████ | 25/25 [06:21<00:00, 15.26s/it]
100% | ██████████ | 25/25 [00:48<00:00, 1.92s/it]





References:

Tim Loughran and Bill McDonald, 2011, When is a Liability not a Liability? Textual Analysis, Dictionaries, and 10-Ks, Journal of Finance, 66:1, 35-65.

Cohen, Malloy and Nguyen (2020), Lazy Prices, Journal of Finance, Volume 75, Issue3, June 2020, Pages 1371-1415

MACHINE LEARNING: CLASSIFICATION

When you come to a fork in the road, take it - Yogi Berra

We apply supervised learning models to a text classification task, using natural language processing (NLP) techniques to analyze business descriptions from the latest 10-K filings. Our goal is to predict firms' industry classifications, evaluating models such as Naive Bayes, Perceptron, Support Vector Machine (SVM), and Logistic Regression. We assess model performance using metrics like the confusion matrix and examine interpretability by visualizing feature importances.

```
# By: Terence Lim, 2020-2025 (terence-lim.github.io)
import time
import numpy as np
import pandas as pd
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.feature_extraction import text
from sklearn.linear_model import LogisticRegression, Perceptron
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import MultinomialNB
from sklearn import metrics
from nltk.tag import pos_tag
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from tqdm import tqdm
from finds.database import SQL, RedisDB
from finds.unstructured import Edgar
from finds.structured import BusDay, CRSP, PSTAT
from finds.readers import Sectoring
from finds.utils import Store
from secret import paths, credentials, CRSP_DATE
# %matplotlib qt
VERBOSE = 0
store = Store('assets', ext='pkl')
```

```
sql = SQL(**credentials['sql'], verbose=VERBOSE)
user = SQL(**credentials['user'], verbose=VERBOSE)
bd = BusDay(sql)
rdb = RedisDB(**credentials['redis'])
crsp = CRSP(sql, bd, rdb, verbose=VERBOSE)
pstat = PSTAT(sql, bd, verbose=VERBOSE)
ed = Edgar(paths['10X'], zipped=True, verbose=VERBOSE)
```

26.1 Text classification

Hoberg and Phillips (2016) proposed a system for classifying firms based on their business descriptions in 10-K filings, using these descriptions to measure firm similarity. We extend their analysis for text-based industry classification, focusing on U.S.-domiciled common stocks. The text data for each firm is drawn from the most recent year's Business Description section of their 10-K filings.

```
# Retrieve universe of stocks, as of beginning of latest year
univ = crsp.get_universe(bd.endyr(CRSP_DATE, -1))

# Construct table to lookup company names
comnam = crsp.build_lookup(source='permno', target='comnam', fillna="")
univ['comnam'] = comnam(univ.index)

# Construct table to lookup ticker symbols
ticker = crsp.build_lookup(source='permno', target='ticker', fillna="")
univ['ticker'] = ticker(univ.index)

# Construct table to lookup sic codes from Compustat, and map to FF 10-sector code
sic = pstat.build_lookup(source='lpermno', target='sic', fillna=0)
industry = Series(sic[univ.index], index=univ.index)
industry = industry.where(industry > 0, univ['siccd'])
sectors = Sectoring(sql, scheme='codes10', fillna='') # supplement from crosswalk
univ['sector'] = sectors[industry]
```

26.1.1 Text pre-processing

The pre-processing step involves several key operations to clean and prepare the text for analysis. We begin by extracting the Business Description text from the most recent 10-K filings of all stocks in our dataset.

```
# retrieve latest year's bus10K's
item, form = 'bus10K', '10-K'
rows = DataFrame(ed.open(form=form, item=item))
found = rows[rows['date'].between(bd.begyr(CRSP_DATE), bd.endyr(CRSP_DATE))]\n    .drop_duplicates(subset=['permno'], keep='last')\n    .set_index('permno')
```

- The text is then **lemmatized** using WordNet's built-in `morphy` function, which reduces words to their base or root form. This step helps standardize the text by consolidating variations of words into their common form.

```
# !nltk.download('averaged_perceptron_tagger')
lemmatizer = WordNetLemmatizer()
```

100% |██████████| 4488/4488 [11:25<00:00, 6.55it/s]

- Next, we apply the **part-of-speech (POS) tagger** from the `nltk` library, retaining only nouns, which are the most informative about for industry classification tasks.

```
bus = {}
for permno in tqdm(found.index):
```

(continues on next page)

(continued from previous page)

```

if permno not in univ.index:
    continue
doc = word_tokenize(ed[found.loc[permno, 'pathname']].lower())
tags = pos_tag(doc)
nouns = [lemmatizer.lemmatize(w[0]) for w in tags
         if w[1] in ['NN', 'NNS'] and w[0].isalpha() and len(w[0]) > 2]
if len(nouns) > 100:
    bus[permno] = nouns
store['nouns'] = bus

```

```

bus = store.load('nouns')
permnos = list(bus.keys())
labels = univ.loc[permnos, 'sector']
data = [" ".join(list(nouns)) for nouns in bus.values()]
classes = sorted(np.unique(labels))

```

- Finally, we split the corpus into training and testing samples, stratifying the data is to maintain the distribution of class labels in both sets.

```

test_size = 0.2
X_train, X_test, y_train, y_test = train_test_split(
    data, labels, test_size=test_size, random_state=42, stratify=labels)
summary = Series(y_train).value_counts().rename('n_train').to_frame()
summary['n_test'] = Series(y_test).value_counts()
summary['frac_train'] = summary['n_train'] / summary['n_train'].sum()
summary['frac_test'] = summary['n_test'] / summary['n_test'].sum()
print('Stratified Train/Test Split by Event')
summary.sort_values('n_train', ascending=False).round(2)

```

Stratified Train/Test Split by Event

sector	n_train	n_test	frac_train	frac_test
Hlth	657	164	0.24	0.24
Other	612	153	0.22	0.22
HiTec	554	139	0.20	0.20
Manuf	275	69	0.10	0.10
Shops	246	62	0.09	0.09
Durbl	131	33	0.05	0.05
NoDur	114	28	0.04	0.04
Enrgy	81	20	0.03	0.03
Utils	72	18	0.03	0.03
Telcm	37	9	0.01	0.01

26.1.2 Text vectorization

After pre-processing the text, we convert the textual data into numerical features that can be fed into machine learning models. This is achieved through the **Term Frequency-Inverse Document Frequency** (TF-IDF) method. TF-IDF weights terms based on their importance in a document relative to their frequency in a collection of documents (corpus).

- **Term Frequency (TF)** measures how often a word appears in a document. A higher frequency suggests the word is more important within that document.
- **Inverse Document Frequency (IDF)** adjusts the weight of a term based on its rarity across the entire corpus. Terms that appear frequently across many documents are given less weight, while terms that appear less frequently are given greater importance.

To focus on the most relevant and informative words, we filter out extremely common words that appear in more than 50% of the documents (using `max_df=0.5`), exclude rare words that appear in fewer than 200 documents (using `min_df=200`), and limit the vocabulary to the 10,000 most frequent remaining terms (`max_features=10000`).

```
# TfIdf vectorizer
max_df, min_df, max_features = 0.5, 10, 20000
tfidf_vectorizer = text.TfidfVectorizer(
    encoding='latin-1',
    strip_accents='unicode',
    lowercase=True,
    stop_words=stop_words,
    max_df=max_df,
    min_df=min_df,
    max_features=max_features,
    token_pattern=r'\b[a-z_]+\b',
)
x_train = tfidf_vectorizer.fit_transform(X_train)      # sparse array
x_test = tfidf_vectorizer.transform(X_test)
feature_names = tfidf_vectorizer.get_feature_names_out()
print("n_sample x n_features")
DataFrame([[x_train.shape, x_test.shape]],
          index=['data shape:'],
          columns=['train', 'test'])
```

n_sample x n_features

	train	test
data shape:	(2779, 10060)	(695, 10060)

26.2 Classification models

In machine learning, there are generally two types of approaches for classification tasks:

- **Generative models** estimate a probability distribution and define the classifier based on these estimates. An example of this is the Naive Bayes classifier.
- **Discriminative models** directly define a decision boundary between classes. Examples of discriminative models include Logistic Regression, Perceptron, and Support Vector Machines (SVM).

Define a helper function to compute and save accuracy scores for both the training and testing samples.

```

results = dict()

def update_results(name, clf, elapsed):
    """helper to update results dict with train and test accuracy"""
    tic = time.time()
    test_score = clf.score(x_test, y_test)
    toc = time.time() - tic
    results[name] = dict(model=clf,
                          train_score=clf.score(x_train, y_train),
                          test_score=test_score,
                          test_time=toc,
                          train_time=elapsed)
    #store['classification'] = results
    print('Accuracy')
    return DataFrame.from_dict(results, orient='index').iloc[:, 1:]

```

26.2.1 Naive Bayes

The Naive Bayes classifier is a simple yet effective method for text classification. It assumes that the features (words) are conditionally independent given the class, meaning that the occurrence of one word in a document does not affect the occurrence of another. Despite this strong assumption, Naive Bayes performs surprisingly well on many real-world classification tasks.

- **Binomial Naive Bayes** is used for binary classification problems, where each document is assigned to one of two classes.
- **Multinomial Naive Bayes** is typically used for multi-class classification, where documents can belong to more than two classes.

Since Naive Bayes relies on the multiplication of probabilities for each feature, it can encounter problems when a feature has a zero probability in the training set. This is addressed through **Laplace smoothing**, which adds a small constant to all feature counts to avoid zero probabilities.

The basic formula for Naive Bayes classification is: $P(f_1, f_2, \dots, f_n | c) = P(f_1 | c) \cdot P(f_2 | c) \cdot \dots \cdot P(f_n | c)$

where $P(f_i | c)$ is the probability of feature f_i occurring in class c . The classification decision is made by selecting the class that maximizes the likelihood of the observed features, i.e.: $\hat{c} = \arg \max_c (\log P(c) + \sum_{i=1}^n \log P(f_i | c))$

where the maximum likelihood estimate of the probability of frequency of word w_i is $P(w_i | c) = \frac{\text{count}(w_i, c)}{\sum_{w \in V} \text{count}(w, c)}$

```

clf = MultinomialNB(alpha=1.0)
tic = time.time()
clf.fit(x_train, y_train)
toc = time.time() - tic
update_results('naivebayes', clf, toc)

```

Accuracy

	train_score	test_score	test_time	train_time
naivebayes	0.751709	0.738129	0.002441	0.013695

26.2.2 Perceptron

The Perceptron is a linear classifier that updates weights based on classification errors. It uses a 0-1 loss function, which assigns a loss of zero for correct classifications and a loss of one for incorrect ones. The update rule for the Perceptron is as follows:

- If the predicted label \hat{y} is different from the true label y , the weight vector w is updated: $w \leftarrow w + \alpha x (y - \hat{y})$ Where α is the learning rate and x is the feature vector of the document.

In multiclass classification, the **One-vs-Rest (OVR)** approach is used, where the Perceptron is trained to distinguish between each class and the rest of the classes.

```
clf = Perceptron(penalty='elasticnet',
                  #n_jobs=4, # -1
                  random_state=0,
                  verbose=VERBOSE)

tic = time.time()
clf.fit(x_train, y_train)
toc = time.time() - tic
update_results('perceptron', clf, toc)
```

Accuracy

	train_score	test_score	test_time	train_time
naivebayes	0.751709	0.738129	0.002441	0.013695
perceptron	0.949982	0.761151	0.004099	0.866820

26.2.3 Support Vector Machine

Support Vector Machines (SVM) are powerful classifiers that work by finding the decision boundary that maximizes the margin between classes. The decision boundary is chosen to minimize classification errors, and SVM uses **hinge loss** to penalize misclassifications: $\text{Loss} = \max(0, 1 - y(w \cdot x))$

SVM can handle both linear and non-linear decision boundaries by using different kernel functions. In this analysis, we use the **LinearSVC** kernel, which optimizes the classification with a linear decision boundary. For multiclass classification, SVM uses the One-vs-Rest (OVR) method.

```
clf = LinearSVC(multi_class='ovr',
                  penalty='l2',
                  verbose=VERBOSE)

tic = time.time()
clf.fit(x_train, y_train)
toc = time.time() - tic
update_results('linearsvc', clf, toc)
```

```
/home/terence/env3.11/lib/python3.11/site-packages/sklearn/svm/_classes.py:31:_
  FutureWarning: The default value of `dual` will change from `True` to `auto`_
  in 1.5. Set the value of `dual` explicitly to suppress the warning.
  warnings.warn(
```

Accuracy

	train_score	test_score	test_time	train_time
naivebayes	0.751709	0.738129	0.002441	0.013695
perceptron	0.949982	0.761151	0.004099	0.866820
linearsvc	0.990284	0.831655	0.002280	0.348787

26.2.4 Logistic Regression

Logistic Regression is a widely used model for binary and multi-class classification tasks. It uses **cross-entropy loss** to measure the difference between predicted probabilities and actual class labels. The logistic regression model updates weights iteratively based on the gradient of the loss function:

- For a binary classification problem, the probability of class 1 is given by: $P(y = 1|x) = \frac{1}{1+e^{-wx}}$

The update step is:

- $P(y = 1|x) \leftarrow 1/(1 + e^{-wx})$
- $w \leftarrow w + \alpha x (1 - P(y = 1|x))$ if $y = 1$
- $w \leftarrow w - \alpha x (1 - P(y = 0|x))$ if $y = 0$

$$\Leftrightarrow w \leftarrow w + \alpha x (y - P(y = 1|x)) \text{ where } y \in \{0, 1\}$$

In multiclass problems, **softmax** is used to generalize logistic regression, providing a probability distribution over all classes. The model's weights are updated using gradient descent to minimize the cross-entropy loss.

```
clf = LogisticRegression(verbose=VERBOSE,
                         penalty='l2',
                         multi_class='multinomial',
                         # n_jobs=-1,           # when multi_class='ovr'
                         max_iter=1000)

tic = time.time()
clf.fit(x_train, y_train)
toc = time.time() - tic
update_results('logistic', clf, toc)
```

Accuracy

	train_score	test_score	test_time	train_time
naivebayes	0.751709	0.738129	0.002441	0.013695
perceptron	0.949982	0.761151	0.004099	0.866820
linearsvc	0.990284	0.831655	0.002280	0.348787
logistic	0.896366	0.833094	0.005731	3.705192

26.3 Evaluation

26.3.1 Overfitting

Evidence of overfitting can be observed when a model performs well on the training data but poorly on the test data. In such cases, the model may have learned to memorize the training data instead of generalizing to new examples. Overfitting can be mitigated by using techniques like cross-validation and regularization.

26.3.2 Accuracy Metrics

To evaluate model performance, we consider several metrics:

- **Precision:** The proportion of true positive predictions among all positive predictions.
- **Recall:** The proportion of true positive predictions among all actual positive cases.
- **F1 Score:** The harmonic mean of precision and recall, which balances the two metrics and provides a single score for model performance.

In multiclass and multilabel scenarios, the F1 score is computed as a weighted average of the F1 scores for each class.

```
# Compute precision, recall, f1 and confusion matrix
res = DataFrame.from_dict(results, orient='index')
models = {k: v['model'] for k,v in results.items()}

scores, cf_test, cf_train = {}, {}, {}
for ifig, (name, clf) in enumerate(models.items()):
    train_pred = clf.predict(x_train)
    test_pred = clf.predict(x_test)
    scores[name] = {
        'train': metrics.precision_recall_fscore_support(
            y_train, train_pred, average='macro')[:3],
        'test': metrics.precision_recall_fscore_support(
            y_test, test_pred, average='macro')[:3]
    }
    cf = DataFrame(confusion_matrix(y_train, train_pred, labels=classes),
                   index=pd.MultiIndex.from_product([[ 'Actual'], classes]),
                   columns=pd.MultiIndex.from_product([[ 'Predicted'], classes]))
    cf_train[name] = cf
    cf = DataFrame(confusion_matrix(y_test, test_pred, labels=classes),
                   index=pd.MultiIndex.from_product([[ 'Actual'], classes]),
                   columns=pd.MultiIndex.from_product([[ 'Predicted'], classes]))
    cf_test[name] = cf
```

```
/home/terence/env3.11/lib/python3.11/site-packages/sklearn/metrics/_classification.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/terence/env3.11/lib/python3.11/site-packages/sklearn/metrics/_classification.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
caption="Model Accuracy"
DataFrame({(metric, sample): [score[sample][i] for score in scores.values()]
           for i, metric in enumerate(['Precision', 'Recall', 'F1-score'])
           for sample in ['train', 'test']}),
           index=scores.keys())
```

	Precision		Recall		F1-score	
	train	test	train	test	train	test
naivebayes	0.758517	0.747060	0.522357	0.498908	0.541932	0.513666

(continues on next page)

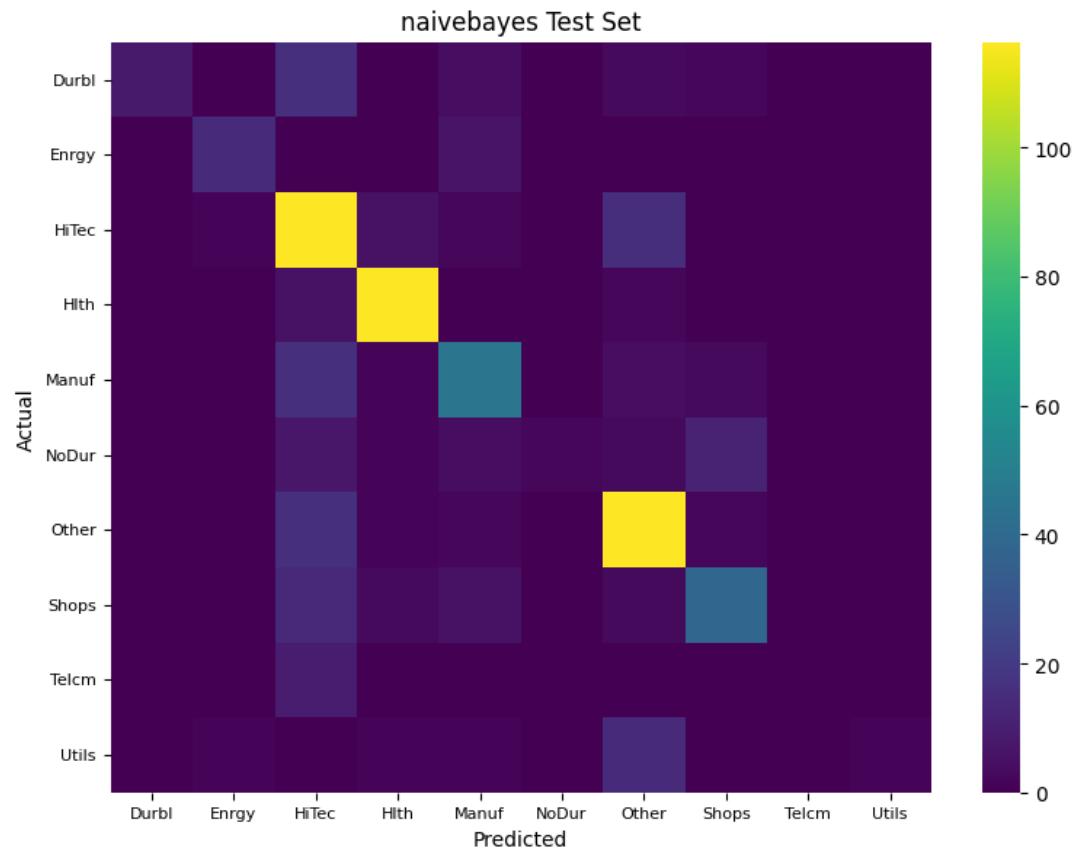
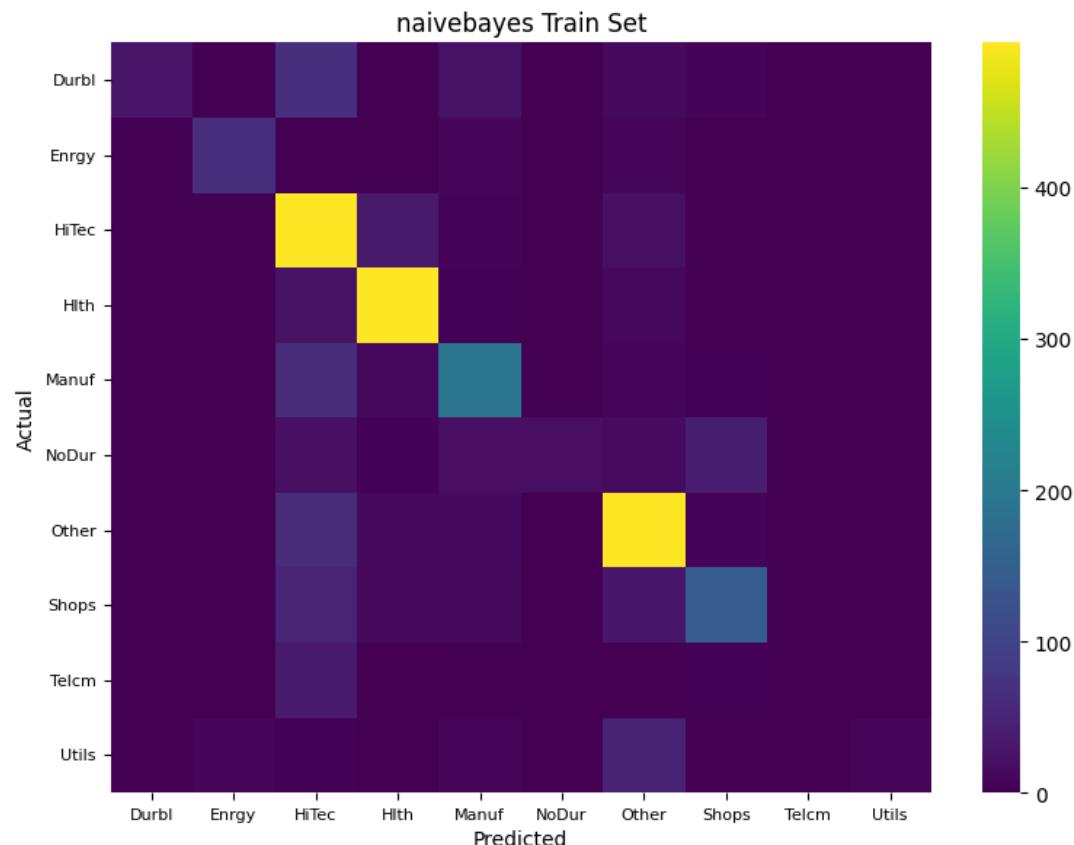
(continued from previous page)

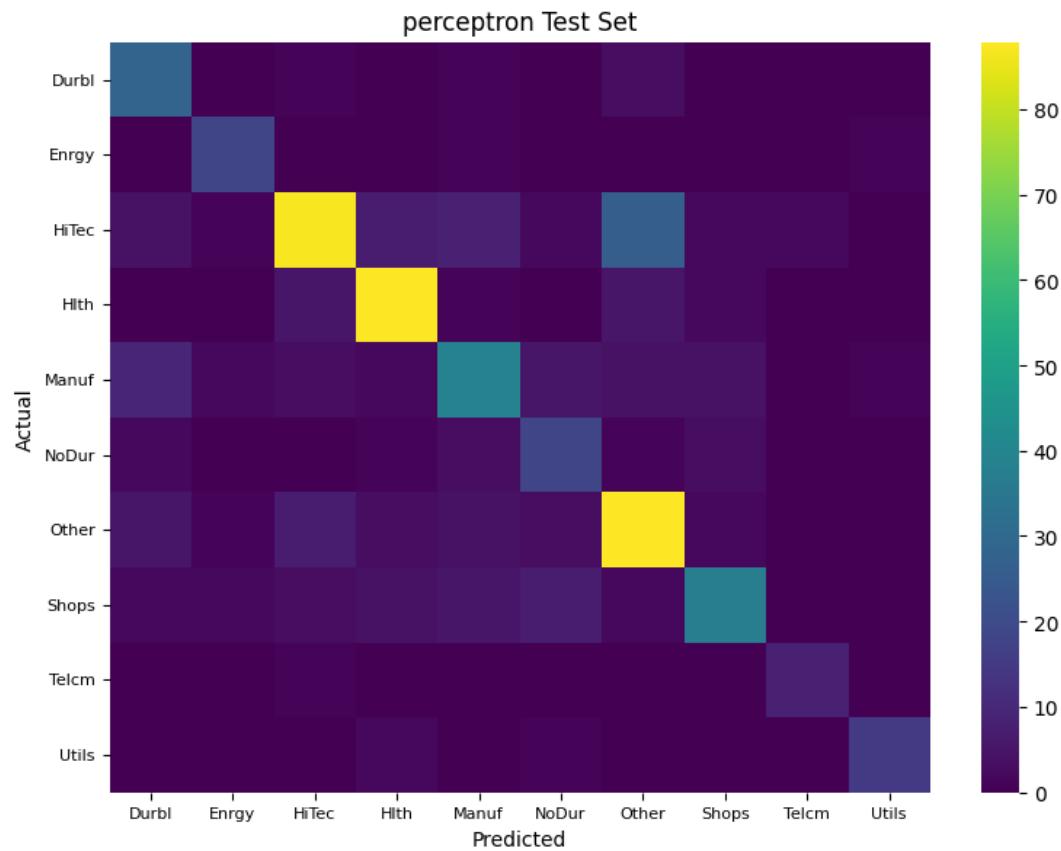
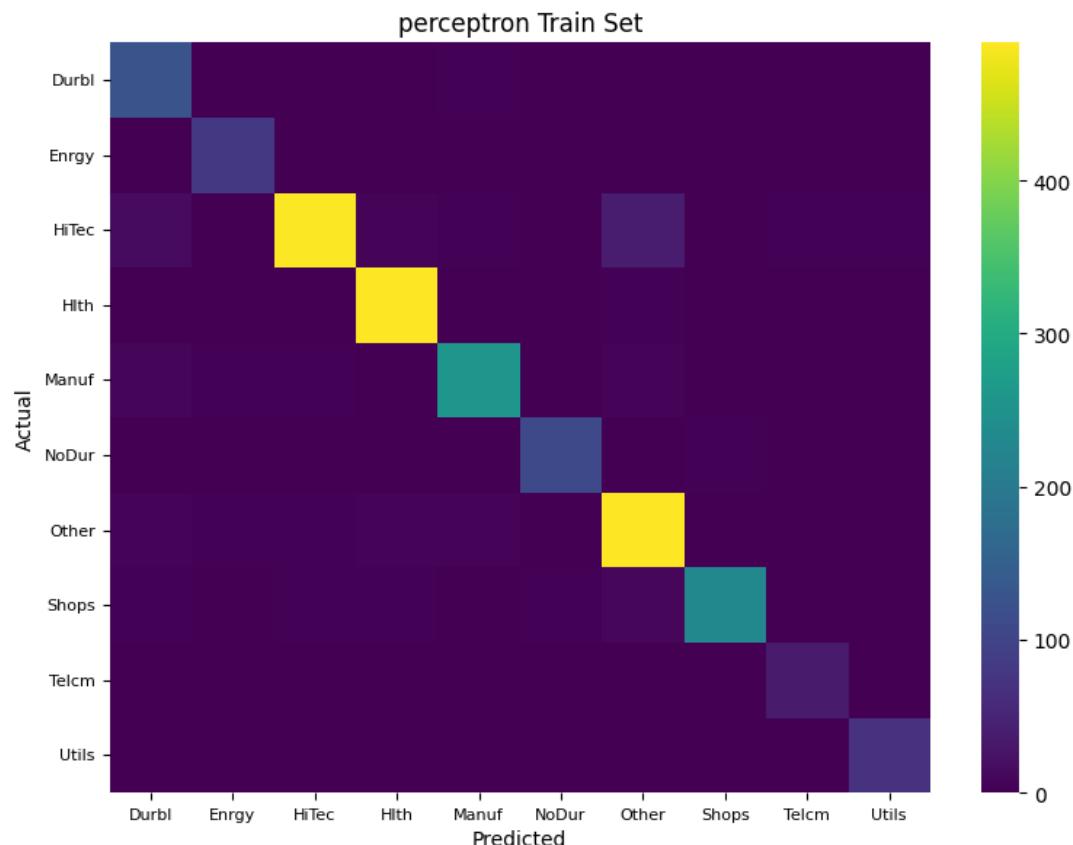
perceptron	0.942079	0.732010	0.953285	0.765879	0.946463	0.741730
linearsvc	0.990133	0.809224	0.993392	0.811619	0.991729	0.806204
logistic	0.901001	0.823311	0.844640	0.793319	0.867906	0.805434

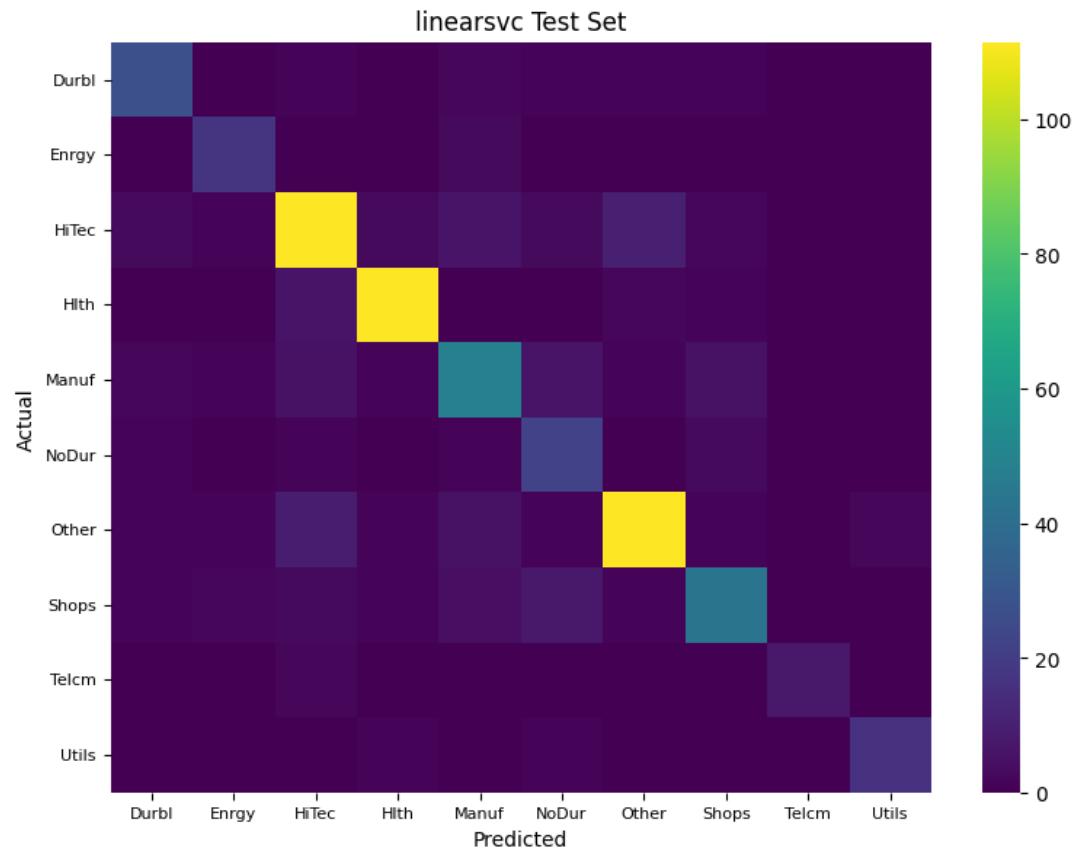
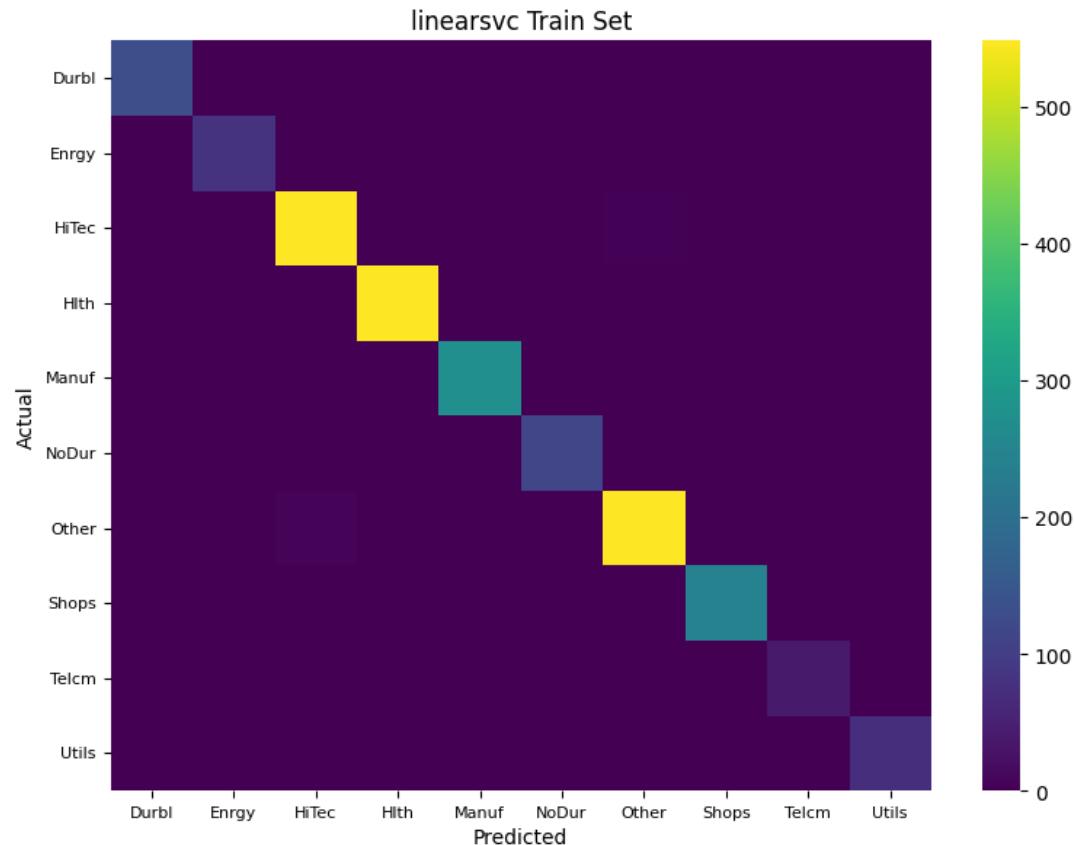
26.3.3 Confusion Matrix

A confusion matrix provides a detailed breakdown of the classification results, showing the true positive, false positive, true negative, and false negative counts for each class.

```
for name, model in models.items():
    fig, axes = plt.subplots(nrows=2, figsize=(8, 12))
    for label, cf, ax in [('Train Set', cf_train[name], axes[0]),
                          ('Test Set', cf_test[name], axes[1])]:
        sns.heatmap(cf, ax=ax, annot=False, fmt='d', cmap='viridis', robust=True,
                    yticklabels=model.classes_,
                    xticklabels=model.classes_)
        ax.set_title(f'{name} {label}')
        ax.set_xlabel('Predicted')
        ax.set_ylabel('Actual')
        ax.yaxis.set_tick_params(labelsize=8, rotation=0)
        ax.xaxis.set_tick_params(labelsize=8, rotation=0)
        #plt.subplots_adjust(left=0.35, bottom=0.25)
    plt.tight_layout()
plt.close()
```







26.3.4 Feature importance

Feature importance can be assessed by examining the weights or probabilities assigned to each term in the model, allowing us to visualize the most important terms for each class. For Naive Bayes, the weights can be exponentiated to probabilities

Word clouds, facilitated by packages such as `WordCloud`, can help visualize the most frequent and important words in the dataset, highlighting the key terms that influence classification decisions.

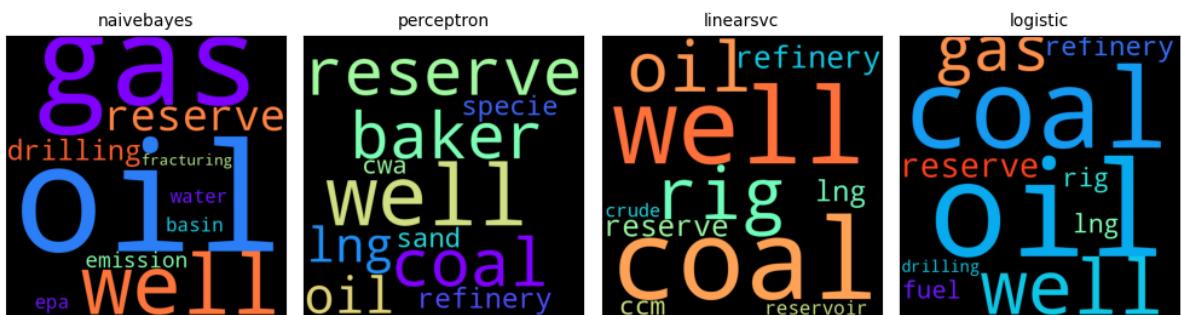
```
wc = WordCloud(height=500, width=500, prefer_horizontal=1.0, colormap='rainbow')

top_n = 10
for topic in classes:    # loop over classes
    fig, axes = plt.subplots(ncols=len(models), nrows=1, figsize=(10, 4))
    fig.suptitle(topic)
    for imodel, (ax, name, clf) in enumerate(zip(
        axes, models.keys(), models.values())):
        assert hasattr(clf, 'coef_') or hasattr(clf, 'feature_log_prob_')
        k = clf.classes_.tolist().index(topic)
        #print("Event %d %s:" % (topic, events_[clf.classes_[topic]]))
        if hasattr(clf, 'coef_'):
            importance = clf.coef_[k, :]
        else:
            importance = np.exp(clf.feature_log_prob_[k, :])
        words = {feature_names[i]: importance[i]
                 for i in importance.argsort()[-top_n:]}
        ax.imshow(wc.generate_from_frequencies(words))
        #Series(words).plot(kind='barh', color=f"C{imodel}", ax=ax)
        #ax.yaxis.set_tick_params(labelsize=7)
        ax.axes.yaxis.set_visible(False)    # make axes ticks invisible
        ax.xaxis.set_ticks([])
        ax.xaxis.set_ticklabels([])
        ax.set_title(name, fontdict={'fontsize':10})
    fig.tight_layout()
plt.close()
```

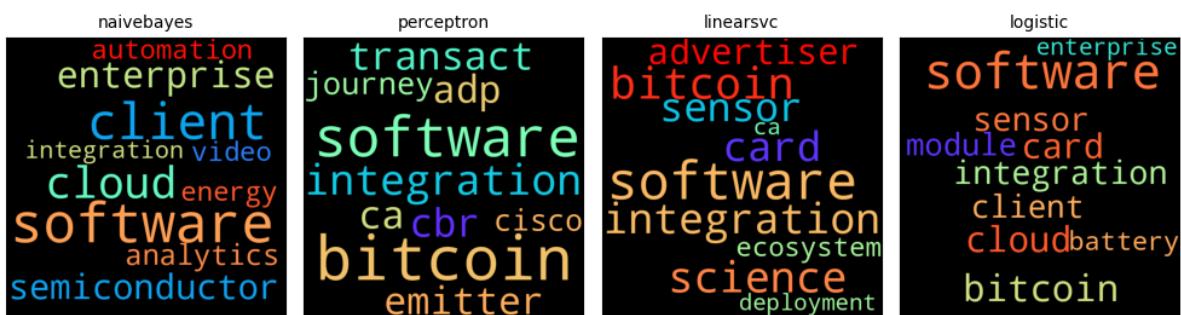
Durbl



Enrgy



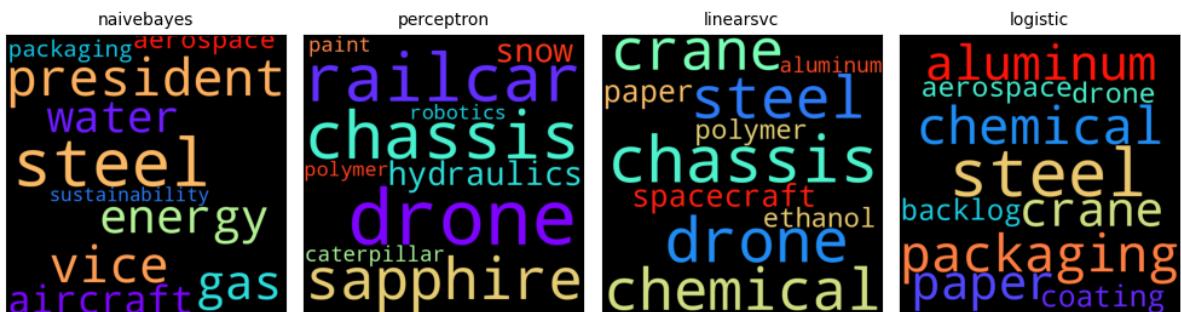
HiTec



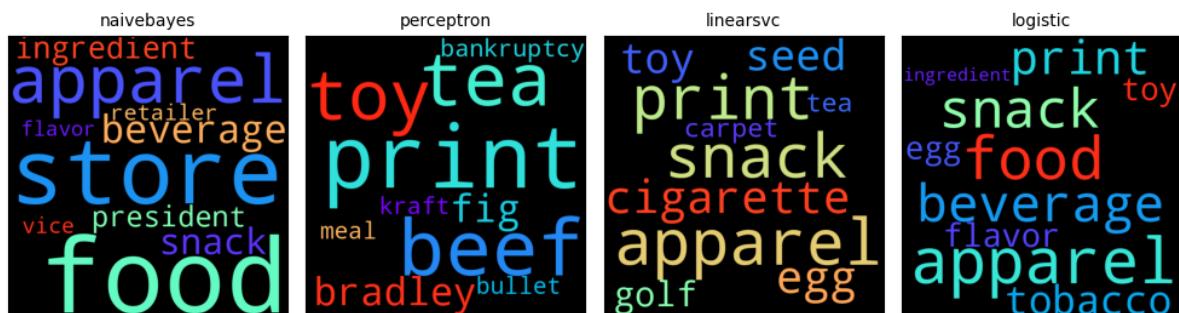
Hlth



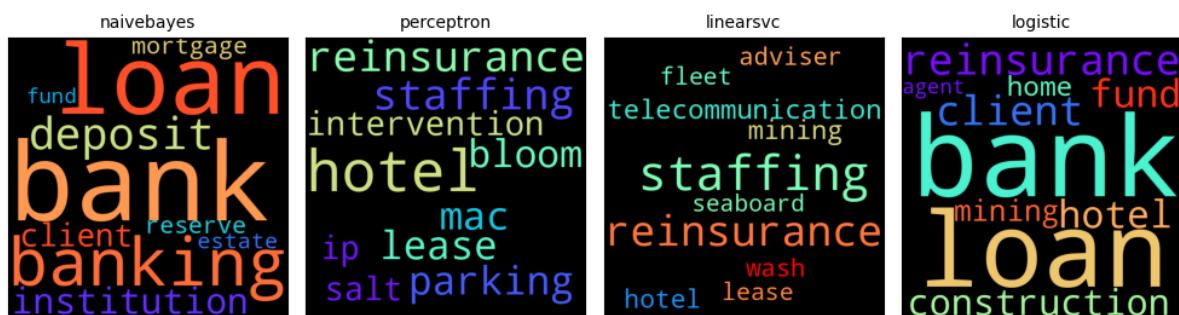
Manuf



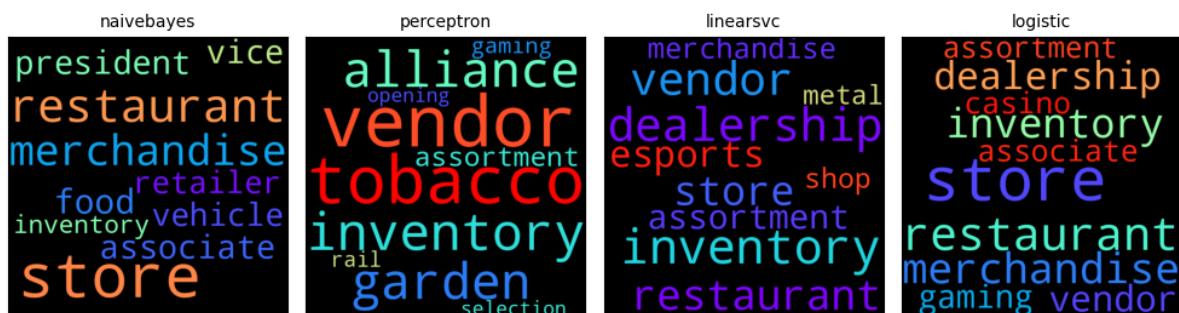
NoDur



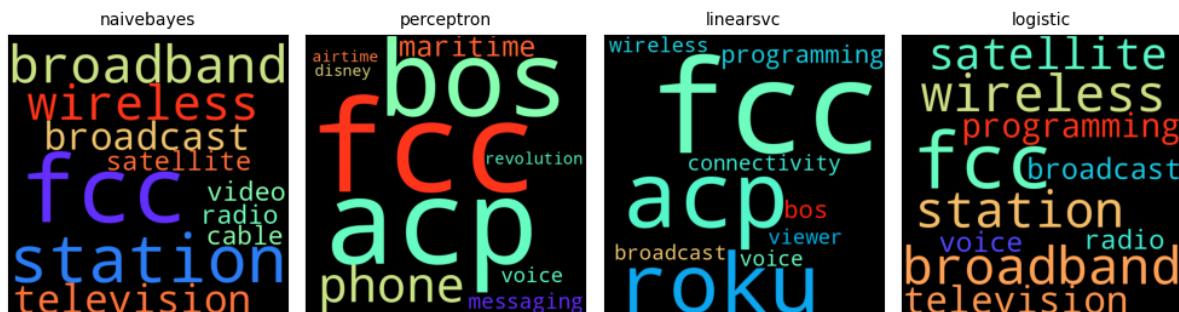
Other



Shops



Telcm



References:

Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani. "An Introduction to Statistical Learning with Applications in R". New York, Springer, 2013.

Gerard Hoberg and Gordon Phillips, 2016, Text-Based Network Industries and Endogenous Product Differentiation. *Journal of Political Economy* 124 (5), 1423-1465.

Gerard Hoberg and Gordon Phillips, 2010, Product Market Synergies and Competition in Mergers and Acquisitions: A Text-Based Analysis. *Review of Financial Studies* 23 (10), 3773-3811.

CHAPTER
TWENTYSEVEN

MACHINE LEARNING: REGRESSION

Whereof what's past is prologue - Shakespeare

We explore supervised machine learning and regression models for macroeconomic forecasting, with a focus on predicting the industrial production index (INDPRO) using a wide range of economic indicators. To improve generalization and predictive performance, we investigate methods such as subset selection, penalized regression, decision trees, and ensemble learning, aiming to strike a balance between model complexity and accuracy.

```
# By: Terence Lim, 2020-2025 (terence-lim.github.io)
import numpy as np
import pandas as pd
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.metrics import mean_squared_error
from finds.readers.alfred import Alfred, fred_md
from finds.utils import plot_date
from secret import credentials
# %matplotlib qt
VERBOSE = 0
```

27.1 Macroeconomic forecasting

For the candidate regression models, the independent variables consist of up to three lags of each economic indicator. The objective is to predict the rate of change in the industrial production index (INDPRO) for the following month while minimizing the **mean squared error (MSE)** loss function, which is appropriate for continuous-valued targets.

Monthly macroeconomic data is retrieved from FRED-MD, with any missing recent values supplemented from other public sources. Suggested transformations are applied to each time series to ensure stationarity. The pre-2023 period is used for training, while the post-2023 period is reserved for testing.

```
alf = Alfred(api_key=credentials['fred']['api_key'], verbose=-1)
```

```
# Get latest FRED-MD data
freq = 'M'
beg = 19640701 # 19620701
df, t = fred_md() #
transforms = t['transform']
end = df.index[-2]
split_date = end - 20000
print("Train/test date ranges:", beg, split_date, end)
```

```
FRED-MD vintage: monthly/current.csv
Train/test date ranges: 19640701 20221231 20241231
```

```
# Splice in common updates: source of PE ratio, Commercial Paper
for col in ['S&P PE ratio']:
    df[col] = alf.splice(col)
df['COMPAPFF'] = df['COMPAPFF'].ffill() # forward fill 20200430
df['CP3M'] = df['CP3M'].ffill() # forward fill 20200430
```

```
# Apply time series transformations according to FRED-MD
transformed = []
for col in df.columns:
    transformed.append(alf.transform(df[col], tcode=transforms[col], freq=freq))
data = pd.concat(transformed, axis=1).iloc[2:]
c = list(data.columns)
data = data.loc[(data.index >= beg) & (data.index <= end)]
```

```
# Drop columns with missing data
missing = []
for series_id in df.columns:
    g = data[series_id].notna()
    missing.extend([(date, series_id) for date in data.index[~g]])
missing_per_row = data.isna().sum(axis=1)
missing = DataFrame.from_records(missing, columns=['date', 'series_id'])
print('original:', data.shape, 'dropna:', data.dropna(axis=1).shape)
data = data.dropna(axis=1) # drop columns where missing values
print(missing['series_id'].value_counts())
data
```

```
original: (726, 126) dropna: (726, 122)
series_id
ACOGNO      332
UMCSENT     163
TWEXAFEGSMTH 103
ANDENO      44
Name: count, dtype: int64
```

	RPI	W875RX1	DPCEA3M086SBEA	CMRMTSPL	RETAIL	INDPRO	\
19640731	0.005422	0.005404	0.007343	0.019986	0.004947	0.006552	
19640831	0.005644	0.006061	0.005992	-0.020139	0.013974	0.006506	
19640930	0.004123	0.004205	-0.004164	0.027173	0.009372	0.003703	
19641031	0.000555	0.000650	0.006499	-0.013866	-0.039421	-0.013947	
19641130	0.006703	0.007352	-0.009111	-0.009745	0.009335	0.030429	
...
20240831	0.000066	-0.000117	0.000711	0.000375	-0.001144	0.004869	
20240930	0.001458	0.000240	0.005257	0.007155	0.008903	-0.004205	
20241031	0.003690	0.003488	0.001757	-0.001714	0.005575	-0.004547	
20241130	0.002184	0.002756	0.004346	0.003993	0.006484	-0.001453	
20241231	0.001234	0.001391	0.005441	0.006665	0.007175	0.009853	
	IPFPNSS	IPFINAL	IPCONGD	IPDCONGD	...	DDURRG3M086SBEA	\
19640731	0.010342	0.011313	0.014079	0.016438	...	0.000667	
19640831	-0.000936	-0.000939	-0.001865	0.007219	...	-0.002031	

(continues on next page)

(continued from previous page)

19640930	-0.004690	-0.003759	-0.011267	-0.023656	...	0.000728
19641031	-0.010404	-0.013269	-0.020030	-0.109875	...	-0.001712
19641130	0.029043	0.031929	0.036884	0.128121	...	0.004137
...
20240831	0.004515	0.007094	0.009218	0.047547	...	0.000600
20240930	-0.005809	-0.008928	-0.002317	-0.001441	...	0.005721
20241031	-0.007220	-0.010970	-0.006158	-0.028598	...	-0.002965
20241130	-0.001180	-0.000155	-0.004246	0.018794	...	-0.000986
20241231	0.007300	0.006569	0.003359	-0.013311	...	-0.004215
	DNDGRG3M086SBEA	DSERRG3M086SBEA	CES0600000008	CES2000000008	\	
19640731	-0.000369	-0.000090	-0.000016	0.003231		
19640831	-0.002398	0.001406	-0.000016	-0.003263		
19640930	0.003749	-0.001674	-0.000015	-0.006462		
19641031	-0.002344	0.000525	-0.011757	0.016093		
19641130	0.000985	-0.000352	0.011772	-0.019272		
...	
20240831	-0.002222	-0.000061	-0.002580	0.001376		
20240930	-0.002598	0.000553	0.004137	0.001906		
20241031	0.003609	0.000788	-0.003839	-0.005006		
20241130	0.000117	-0.002176	-0.000639	-0.002760		
20241231	0.004218	0.002086	0.002521	0.005776		
	CES3000000008	DTCOLNVHFN	DTCTHFN	INVEST	VIXCLS	
19640731	-0.004158	-0.003798	-0.002430	-0.002831	11.2238	
19640831	0.004141	-0.005958	-0.001818	0.011673	13.6898	
19640930	0.004090	-0.011142	-0.003802	0.010582	10.5167	
19641031	-0.024760	0.005974	-0.000704	-0.003579	11.0924	
19641130	0.024828	-0.009946	-0.002806	-0.002694	12.0087	
...	
20240831	-0.006110	-0.001974	-0.000766	0.001445	19.6750	
20240930	0.004652	-0.002368	-0.001586	-0.001644	17.6597	
20241031	-0.002154	-0.000146	0.003227	-0.000473	19.9478	
20241130	0.003190	-0.000872	-0.001679	-0.011753	15.9822	
20241231	-0.001439	0.002963	0.002748	0.002421	15.6997	

[726 rows x 122 columns]

Data beyond the split date is excluded from model training to ensure a proper evaluation of generalization performance.

```
# Split time series train and test set
def ts_split(X, Y, end=split_date):
    """helper to split train/test time series"""
    return X[Y.index<=end], X[Y.index>end], Y[Y.index<=end], Y[Y.index>end]
```

```
def columns_map(columns, lag):
    """helper to create lagged column names"""
    return {col: col + '_' + str(lag) for col in columns}
```

```
def columns_unmap(columns):
    """helper to extract lagged column names"""
    cols = [col.split('_') for col in columns]
    return [col[0] for col in cols], [int(col[1]) for col in cols]
```

```
target_id = 'INDPRO'
lags = 3 # Include up to 3 lags of exogs
Y = data[target_id].iloc[lags:]
X = pd.concat([data.shift(lag).iloc[lags:] \
    .rename(columns=columns_map(data.columns, lag)) \
    for lag in range(1, lags+1)], \
    axis=1)
```

```
# new table to collect final fitted models
test = Series(name='test', dtype=float) # collect test and train errors
train = Series(name='train', dtype=float)
final_models = {}
```

27.2 Regression models

Complex regression models with numerous features or parameters risk overfitting, capturing noise rather than true patterns. Several techniques help mitigate overfitting and improve model generalization:

- **Cross-validation:** Hyperparameter tuning using cross-validation helps ensure models generalize well across different data subsets. **K-fold Cross-Validation (K-fold CV)** divides the dataset into K parts, training the model on $K - 1$ sections and testing it on the remaining part, iterating through all folds and averaging results. **Leave-One-Out Cross-Validation (LOOCV)** is a special case where each data point serves as the test set once, helping refine hyperparameter selection.
- **Information criteria penalty:** The **Akaike Information Criterion (AIC)** and **Bayesian Information Criterion (BIC)** aid model selection by balancing goodness of fit and complexity. AIC, defined as $AIC = 2k - 2\ln(L)$, where k is the number of parameters and L is the likelihood, favors models that explain the data with minimal complexity. BIC, given by $BIC = k\ln(n) - 2\ln(L)$, introduces a stronger penalty for sample size n , preferring simpler models for large datasets. Lower AIC or BIC values indicate better models.
- **Ensemble learning:** By combining multiple weak learners, ensemble methods enhance model robustness. Bagging averages predictions from multiple models trained on different data subsets, reducing variance. Boosting sequentially corrects model errors, improving overall accuracy.

27.2.1 Forward Subset Selection

Forward Subset Regression is a stepwise variable selection method used in regression modeling to identify the best subset of predictors. It begins with no variables in the model and iteratively adds the most significant predictor at each step, based on a chosen criterion, and stops when adding more variables does not significantly improve model performance. Penalized selection criteria, such as AIC or BIC, helps avoid overfitting. Forward selection is computationally more efficient than exhaustive search methods but may miss the optimal subset if an initially excluded variable becomes relevant later in combination with others.

```
def forward_select(Y, X, selected, ic='aic'):
    """helper to forward select next regressor, using sm.OLS"""
    remaining = [x for x in X.columns if x not in selected]
    results = []
    for x in remaining:
        r = sm.OLS(Y, X[selected + [x]]).fit()
        results.append({'select': x,
```

(continues on next page)

(continued from previous page)

```

        'aic': r.aic,
        'bic': r.bic,
        'rsquared': r.rsquared,
        'rsquared_adj': r.rsquared_adj})
return DataFrame(results).sort_values(by=ic).iloc[0].to_dict()

```

```

# find best bic, and show selection criteria scores
ic = 'bic'    # select by information criterion
X_train, X_test, Y_train, Y_test = ts_split(X, Y)
selected = []
models = {}
for i in range(1, 32):
    select = forward_select(Y_train,
                             X_train,
                             selected,
                             ic=ic)
    models[i] = select
    selected.append(select['select'])
selected = DataFrame.from_dict(models, orient='index')
best = selected[[ic]].iloc[selected[ic].argmin()]
subset = selected.loc[:best.name].round(3)
subset.index = [alf.header(s.split('_')[0]) for s in subset['select']]
print('Forward Selection Subset')
subset

```

Forward Selection Subset

```

Initial Claims
Real M2 Money Stock
All Employees, Wholesale Trade
All Employees, Service-Providing
5-Year Treasury Constant Maturity Minus Federal...
Total Business: Inventories to Sales Ratio
All Employees, Service-Providing
All Employees, Financial Activities
All Employees, Trade, Transportation, and Utili...
Consumer Motor Vehicle Loans Owned by Finance C...
Industrial Production: Nondurable Energy Consum...
Industrial Production: Nondurable Goods Materials
Canadian Dollars to U.S. Dollar Spot Exchange Rate
M1
Nonrevolving consumer credit to Personal Income
Market Yield on U.S. Treasury Securities at 10-...
Capacity Utilization: Manufacturing (SIC)
Industrial Production: Manufacturing (SIC)
All Employees, Nondurable Goods
Average Hourly Earnings of Production and Nonsu...
Average Weekly Overtime Hours of Production and...
Industrial Production: Equipment: Business Equi...
Industrial Production: Total Index
Real Estate Loans, All Commercial Banks
All Employees, Mining, Quarrying, and Oil and G...
Moody's Seasoned Baa Corporate Bond Yield

```

```

select      aic \
CLAIMS_1  -4772.162
M2REAL_2  -4814.414
USWTRADE_2 -4843.063
SRVPRD_2  -4864.114
T5YFFM_3  -4881.249
ISRATIO_2  -4895.191
SRVPRD_1  -4911.828
USFIRE_2  -4928.888
USTPU_1   -4945.576
DTCOLNVHFN_3 -4952.630
IPB51222S_3 -4960.975
IPNMAT_3   -4974.770
EXCAUS_3   -4983.489
M1SL_3    -4993.678
CONSPI_2   -5001.564
GS10_1    -5008.976
CUMFNS_2   -5014.366
IPMANSICS_2 -5041.837
NDMANEMP_1 -5056.048
CES2000000008_1 -5065.393
AWOTMAN_2  -5075.474
IPBUSEQ_2  -5083.967
INDPRO_2   -5091.928
REALLN_1   -5096.759
CES1021000001_2 -5102.029
BAA_1     -5106.126

```

(continues on next page)

(continued from previous page)

Moody's Seasoned Aaa Corporate Bond Yield	AAA_1	-5110.792
All Employees, Mining, Quarrying, and Oil and G...	CES1021000001_1	-5116.399
Industrial Production: Materials	IPMAT_1	-5120.986

	bic	rsquared	\
Initial Claims	-4767.612	0.355	
Real M2 Money Stock	-4805.315	0.394	
All Employees, Wholesale Trade	-4829.414	0.420	
All Employees, Service-Providing	-4845.916	0.439	
5-Year Treasury Constant Maturity Minus Federal...	-4858.500	0.454	
Total Business: Inventories to Sales Ratio	-4867.893	0.467	
All Employees, Service-Providing	-4879.981	0.481	
All Employees, Financial Activities	-4892.490	0.495	
All Employees, Trade, Transportation, and Utili...	-4904.629	0.508	
Consumer Motor Vehicle Loans Owned by Finance C...	-4907.133	0.514	
Industrial Production: Nondurable Energy Consum...	-4910.928	0.521	
Industrial Production: Nondurable Goods Materials	-4920.175	0.532	
Canadian Dollars to U.S. Dollar Spot Exchange Rate	-4924.343	0.539	
M1	-4929.983	0.547	
Nonrevolving consumer credit to Personal Income	-4933.319	0.554	
Market Yield on U.S. Treasury Securities at 10-...	-4936.181	0.560	
Capacity Utilization: Manufacturing (SIC)	-4937.022	0.564	
Industrial Production: Manufacturing (SIC)	-4959.943	0.582	
All Employees, Nondurable Goods	-4969.605	0.592	
Average Hourly Earnings of Production and Nonsu...	-4974.400	0.598	
Average Weekly Overtime Hours of Production and...	-4979.931	0.605	
Industrial Production: Equipment: Business Equi...	-4983.875	0.611	
Industrial Production: Total Index	-4987.286	0.617	
Real Estate Loans, All Commercial Banks	-4987.568	0.620	
All Employees, Mining, Quarrying, and Oil and G...	-4988.288	0.624	
Moody's Seasoned Baa Corporate Bond Yield	-4987.835	0.627	
Moody's Seasoned Aaa Corporate Bond Yield	-4987.951	0.631	
All Employees, Mining, Quarrying, and Oil and G...	-4989.008	0.635	
Industrial Production: Materials	-4989.046	0.638	

	rsquared_adj
Initial Claims	0.354
Real M2 Money Stock	0.393
All Employees, Wholesale Trade	0.418
All Employees, Service-Providing	0.436
5-Year Treasury Constant Maturity Minus Federal...	0.450
Total Business: Inventories to Sales Ratio	0.462
All Employees, Service-Providing	0.475
All Employees, Financial Activities	0.489
All Employees, Trade, Transportation, and Utili...	0.502
Consumer Motor Vehicle Loans Owned by Finance C...	0.507
Industrial Production: Nondurable Energy Consum...	0.514
Industrial Production: Nondurable Goods Materials	0.524
Canadian Dollars to U.S. Dollar Spot Exchange Rate	0.531
M1	0.538
Nonrevolving consumer credit to Personal Income	0.544
Market Yield on U.S. Treasury Securities at 10-...	0.549
Capacity Utilization: Manufacturing (SIC)	0.553
Industrial Production: Manufacturing (SIC)	0.571
All Employees, Nondurable Goods	0.580
Average Hourly Earnings of Production and Nonsu...	0.586

(continues on next page)

(continued from previous page)

Average Weekly Overtime Hours of Production and...	0.593
Industrial Production: Equipment: Business Equi...	0.598
Industrial Production: Total Index	0.604
Real Estate Loans, All Commercial Banks	0.607
All Employees, Mining, Quarrying, and Oil and G...	0.610
Moody's Seasoned Baa Corporate Bond Yield	0.613
Moody's Seasoned Aaa Corporate Bond Yield	0.616
All Employees, Mining, Quarrying, and Oil and G...	0.620
Industrial Production: Materials	0.623

```
DataFrame.from_dict({n: {'series_id': s.split('_')[0],
                         'lag' : s.split('_')[1],
                         'description': alf.header(s.split('_')[0])}
                     for n, s in selected.loc[:best.name, 'select'].items()},
                     orient='index').set_index('series_id')
```

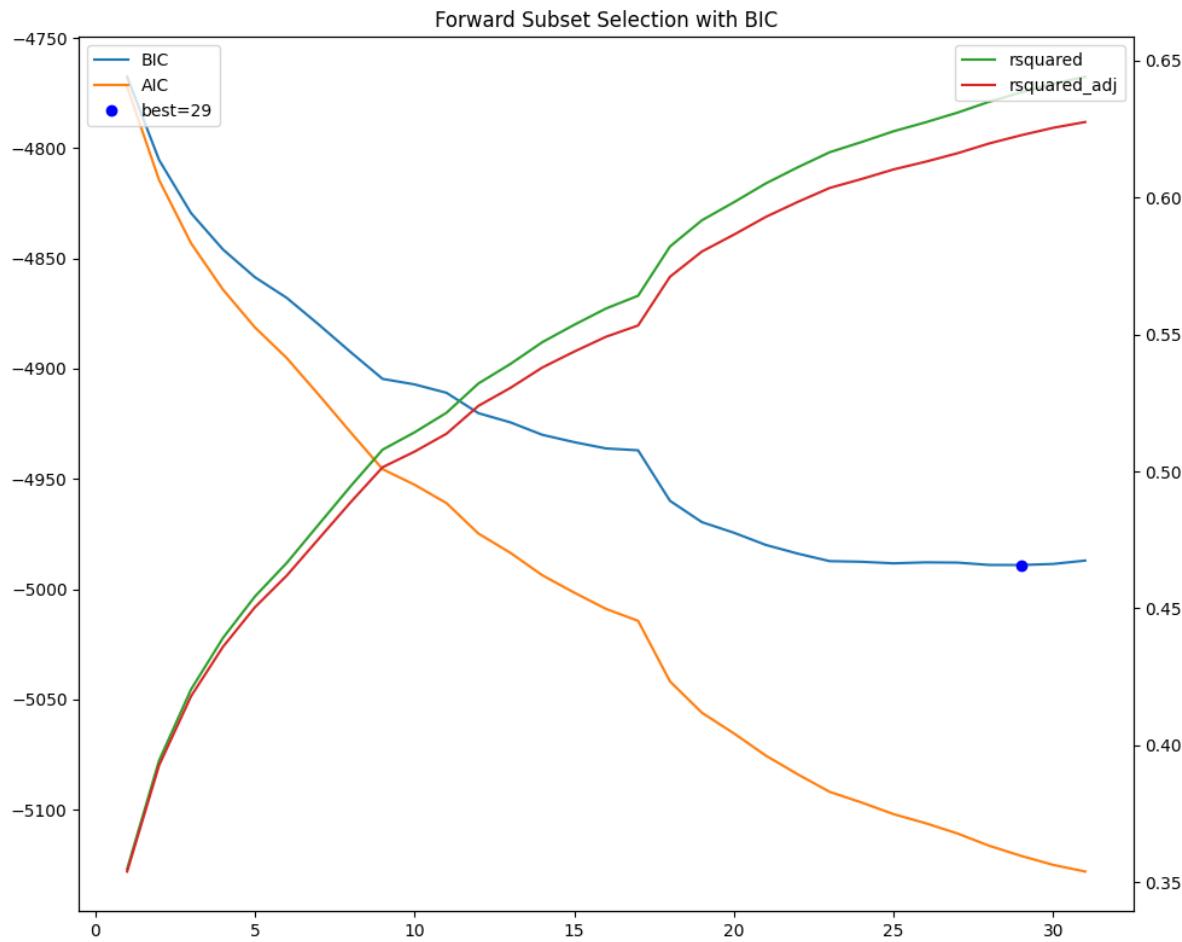
series_id	lag	description
CLAIMS	1	Initial Claims
M2REAL	2	Real M2 Money Stock
USWTRADE	2	All Employees, Wholesale Trade
SRVPRD	2	All Employees, Service-Providing
T5YFFM	3	5-Year Treasury Constant Maturity Minus Federa...
ISRATIO	2	Total Business: Inventories to Sales Ratio
SRVPRD	1	All Employees, Service-Providing
USFIRE	2	All Employees, Financial Activities
USTPU	1	All Employees, Trade, Transportation, and Util...
DTCOLNVHFNM	3	Consumer Motor Vehicle Loans Owned by Finance ...
IPB51222S	3	Industrial Production: Nondurable Energy Consu...
IPNMAT	3	Industrial Production: Nondurable Goods Materials
EXCAUS	3	Canadian Dollars to U.S. Dollar Spot Exchange ...
M1SL	3	M1
CONSPI	2	Nonrevolving consumer credit to Personal Income
GS10	1	Market Yield on U.S. Treasury Securities at 10...
CUMFNS	2	Capacity Utilization: Manufacturing (SIC)
IPMANSICS	2	Industrial Production: Manufacturing (SIC)
NDMANEMP	1	All Employees, Nondurable Goods
CES2000000008	1	Average Hourly Earnings of Production and Nons...
AWOTMAN	2	Average Weekly Overtime Hours of Production an...
IPBUSEQ	2	Industrial Production: Equipment: Business Equ...
INDPRO	2	Industrial Production: Total Index
REALLN	1	Real Estate Loans, All Commercial Banks
CES1021000001	2	All Employees, Mining, Quarrying, and Oil and ...
BAA	1	Moody's Seasoned Baa Corporate Bond Yield
AAA	1	Moody's Seasoned Aaa Corporate Bond Yield
CES1021000001	1	All Employees, Mining, Quarrying, and Oil and ...
IPMAT	1	Industrial Production: Materials

```
# Plot BIC vs number selected
fig, ax = plt.subplots(num=1, figsize=(10, 8))
selected['bic'].plot(ax=ax, c='C0')
selected['aic'].plot(ax=ax, c='C1')
ax.plot(best.name, float(best.iloc[0]), "ob")
ax.legend(['BIC', 'AIC', f"best={best.name}"], loc='upper left')
```

(continues on next page)

(continued from previous page)

```
ax.set_title(f"Forward Subset Selection with {ic.upper()}")
bx = ax.twinx()
selected['rsquared'].plot(ax=bx, c='C2')
selected['rsquared_adj'].plot(ax=bx, c='C3')
bx.legend(['rsquared', 'rsquared_adj'], loc='upper right')
bx.set_xlabel('# Predictors')
plt.tight_layout()
```



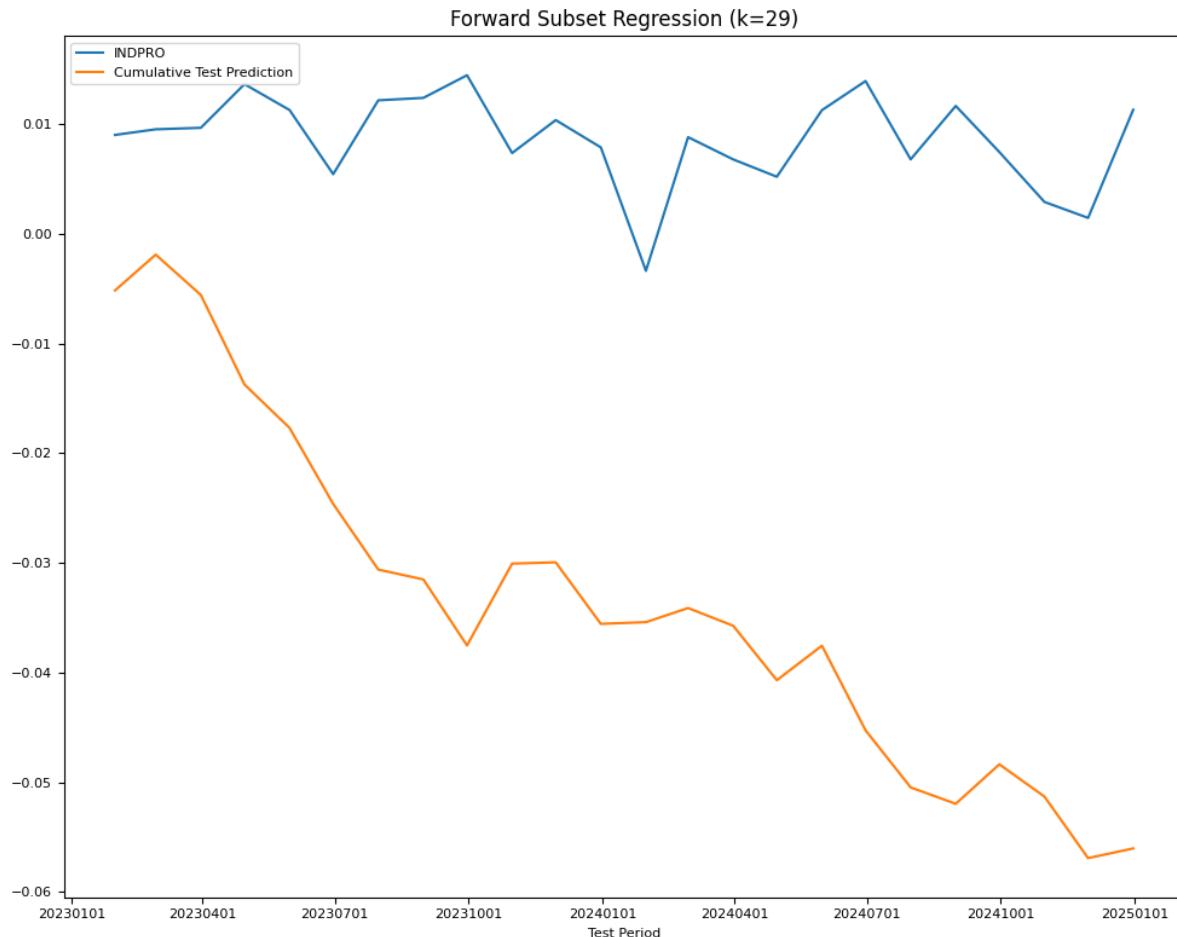
```
# evaluate train and test mse
X_subset = X_train[subset['select']]
model = sm.OLS(Y_train, X_subset).fit()
name = f"Forward Subset Regression (k={len(subset)})"
Y_pred = model.predict(X_test[subset['select']])
test[name] = mean_squared_error(Y_test, Y_pred)
train[name] = mean_squared_error(Y_train, model.predict(X_subset))
final_models[name] = model
```

```
fig, ax = plt.subplots(figsize=(10, 8))
plot_date(pd.concat([Y_test.cumsum(), Y_pred.cumsum()], axis=1),
          legend1=[target_id, 'Cumulative Test Prediction'],
          title=name,
          xlabel='Test Period',
```

(continues on next page)

(continued from previous page)

```
    fontsize=8,
    ax=ax)
plt.tight_layout()
```



```
DataFrame({ 'name': name,
            'train': np.sqrt(train[name]),
            'test': np.sqrt(test[name]) }, index=['RMSE'])
```

	name	train	test
RMSE	Forward Subset Regression (k=29)	0.005955	0.007674

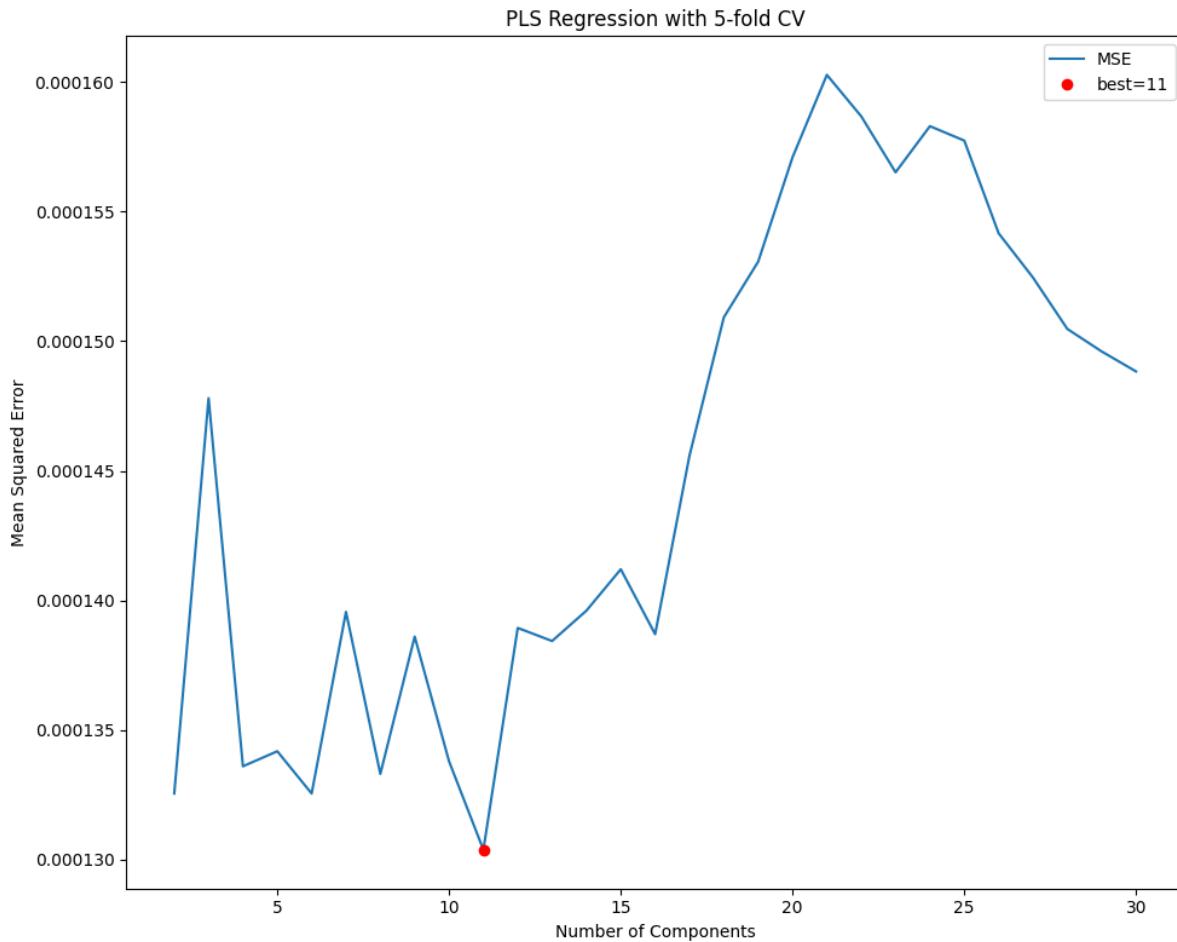
27.2.2 Partial Least Squares Regression

Partial Least Squares (PLS) Regression combines features of principal component analysis (PCA) and multiple linear regression. PLS constructs new latent variables (components) as linear combinations of the original predictors that maximize covariance with the response variable. The number of components (`n_components`) parameter controls the dimensionality reduction and must be chosen carefully to balance bias-variance tradeoff. This method is particularly useful when datasets have more predictors than observations and strong multicollinearity.

```
# split train and test, fit standard scaling using train set
from sklearn.preprocessing import StandardScaler
from sklearn.cross_decomposition import PLSRegression
from sklearn.model_selection import cross_val_score, KFold
X_train, X_test, Y_train, Y_test = ts_split(X, Y)
scale = StandardScaler().fit(X_train)
X_train = scale.transform(X_train)
X_test = scale.transform(X_test)
```

```
# fit with 5-fold CV to choose n_components
n_splits=5
kf = KFold(n_splits=n_splits, shuffle=True, random_state=0)
mse = Series(dtype=float)
for i in np.arange(2, 31):
    pls = PLSRegression(n_components=i)
    score = cross_val_score(estimator=pls,
                            X=X_train,
                            y=Y_train,
                            n_jobs=5,
                            verbose=VERBOSE,
                            cv=kf,
                            scoring='neg_mean_squared_error').mean()
    mse.loc[i] = -score
```

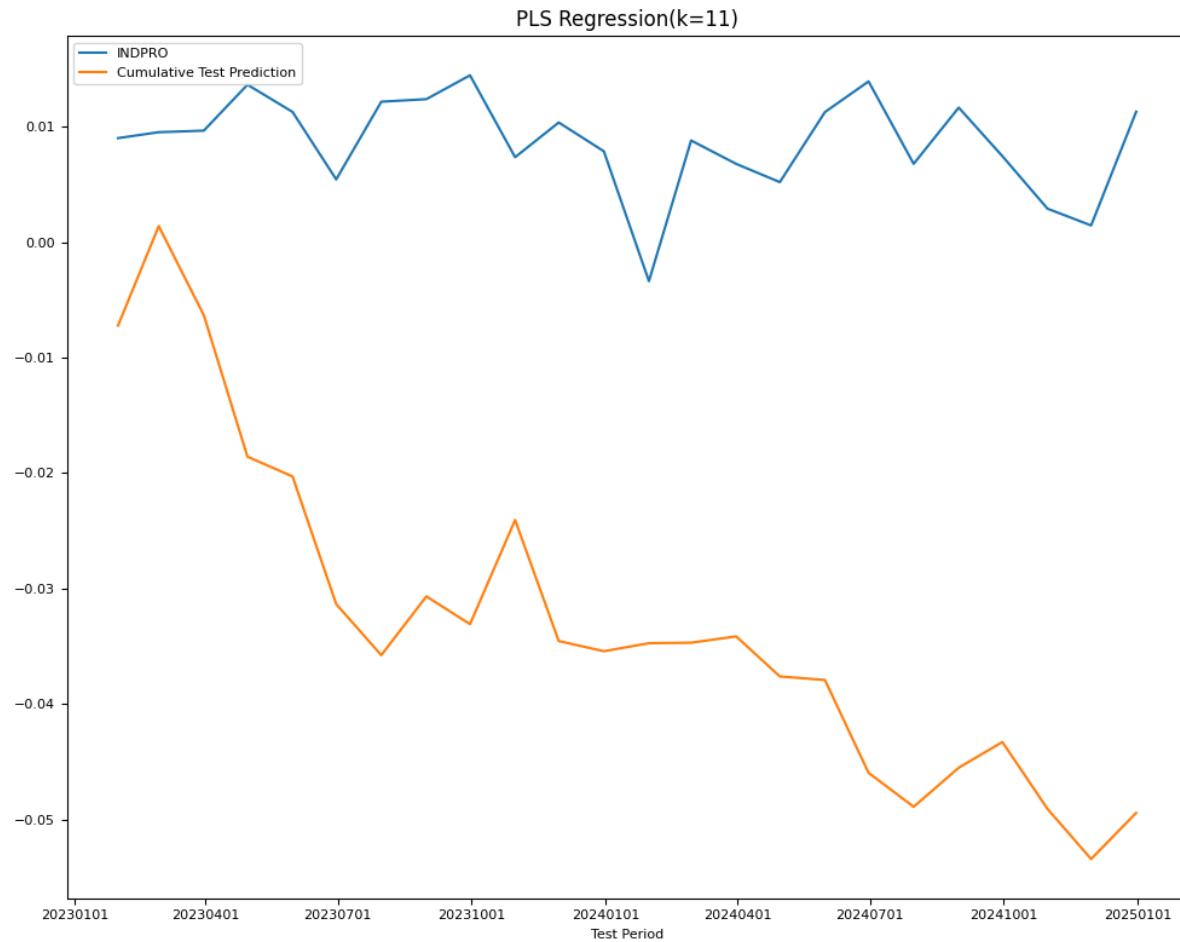
```
# show cross-validation results and best model
fig, ax = plt.subplots(figsize=(10, 8))
mse.plot(ylabel='Mean Squared Error',
          xlabel='Number of Components',
          title=f"PLS Regression with {n_splits}-fold CV",
          ax=ax)
best = mse.index[mse.argmin()]
ax.plot(best, mse.loc[best], "or")
ax.legend(['MSE', f"best={best}"])
plt.tight_layout()
```



```
### evaluate train and test mse
model = PLSRegression(n_components=best).fit(X_train, Y_train)
name = f"PLS Regression(k={best})"
Y_pred = Series(index=Y_test.index,
                data=model.predict(X_test).reshape((-1,)))
```

```
test[name] = mean_squared_error(Y_test, Y_pred)
train[name] = mean_squared_error(Y_train, model.predict(X_train))
final_models[name] = model
```

```
fig, ax = plt.subplots(figsize=(10, 8))
plot_date(pd.concat([Y_test.cumsum(), Y_pred.cumsum()], axis=1),
          legend1=[target_id, 'Cumulative Test Prediction'],
          title=name,
          xlabel='Test Period',
          fontsize=8,
          ax=ax)
plt.tight_layout()
```



```
DataFrame({ 'name': name,
            'train': np.sqrt(train[name]),
            'test': np.sqrt(test[name]) }, index=['RMSE'])
```

	name	train	test
RMSE	PLS Regression(k=11)	0.005306	0.008791

27.2.3 Ridge Regression

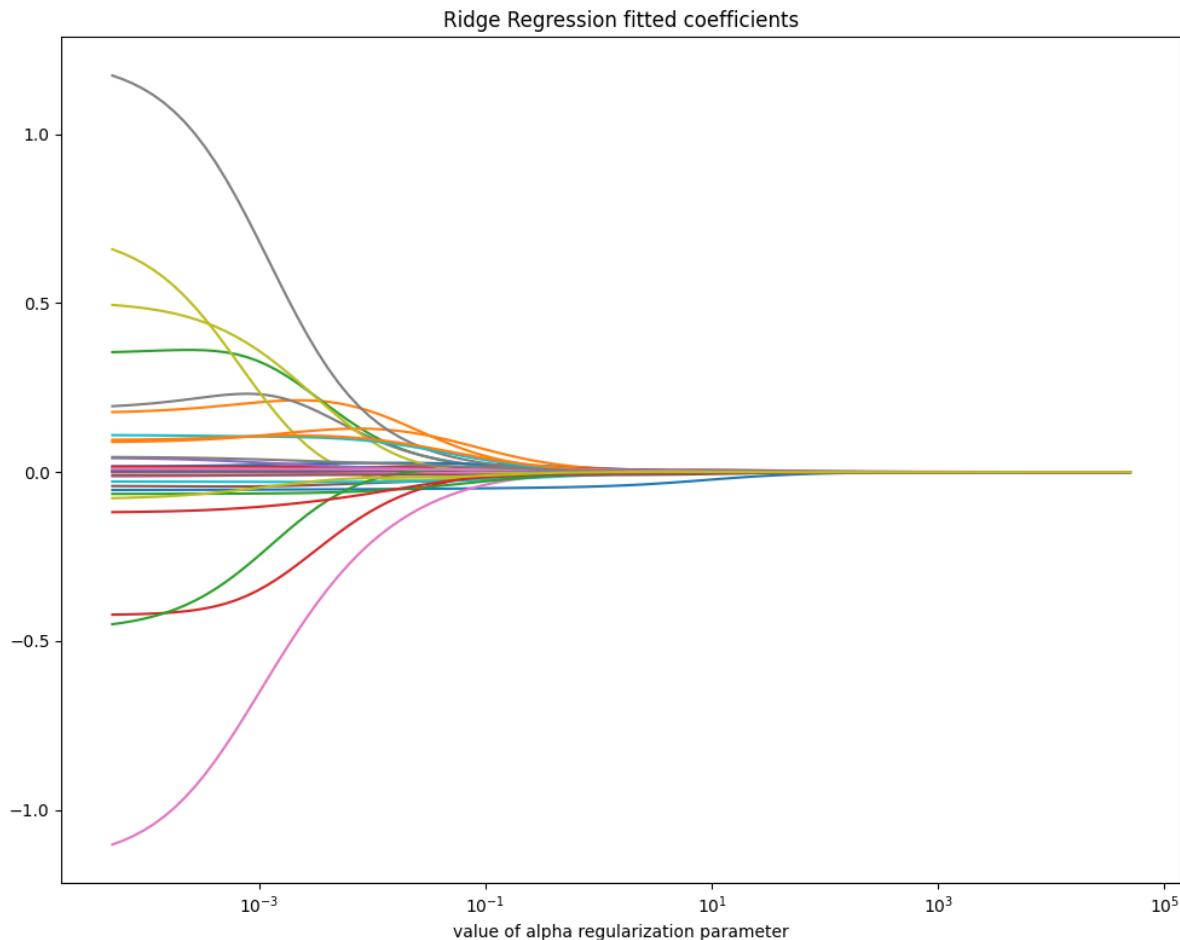
Ridge Regression is a linear regression model that includes L2 regularization, which helps prevent overfitting by adding a penalty term to the loss function. The objective function is modified as follows:

$$\min_w ||y - Xw||^2 + \lambda ||w||^2$$

where $||y - Xw||^2$ is the standard least squares error, and $\lambda ||w||^2$ is the penalty term that shrinks the model coefficients w towards zero. The λ parameter (alpha in sklearn's Ridge) controls the strength of regularization: larger values reduce model complexity by forcing coefficients to be smaller, while smaller values make Ridge behave like standard linear regression. Ridge retains all features but reduces their impact, which is useful when dealing with multicollinearity, without completely eliminating any predictor.

```
from sklearn.linear_model import Ridge, RidgeCV
alphas = 10**np.linspace(5, -4, 100)*0.5 # for parameter tuning
X_train, X_test, Y_train, Y_test = ts_split(X, Y)
scale = StandardScaler().fit(X_train)
X_train = scale.transform(X_train)
X_test = scale.transform(X_test)
np.random.seed(42)
```

```
# Plot fitted coefficients vs regularization alpha
coefs = [Ridge(alpha, fit_intercept=False) \
          .fit(X_subset, Y_train).coef_ for alpha in alphas]
fig, ax = plt.subplots(figsize=(10, 8))
ax.plot(alphas, coefs)
ax.set_xscale('log')
ax.set_xlabel('value of alpha regularization parameter')
ax.set_title('Ridge Regression fitted coefficients')
plt.tight_layout()
```



```
# RidgeCV LOOCV
model = RidgeCV(alphas=alphas,
                 scoring='neg_mean_squared_error',
                 cv=None, # to use Leave-One-Out cross validation
                 store_cv_values=True).fit(X_train, Y_train)
```

```

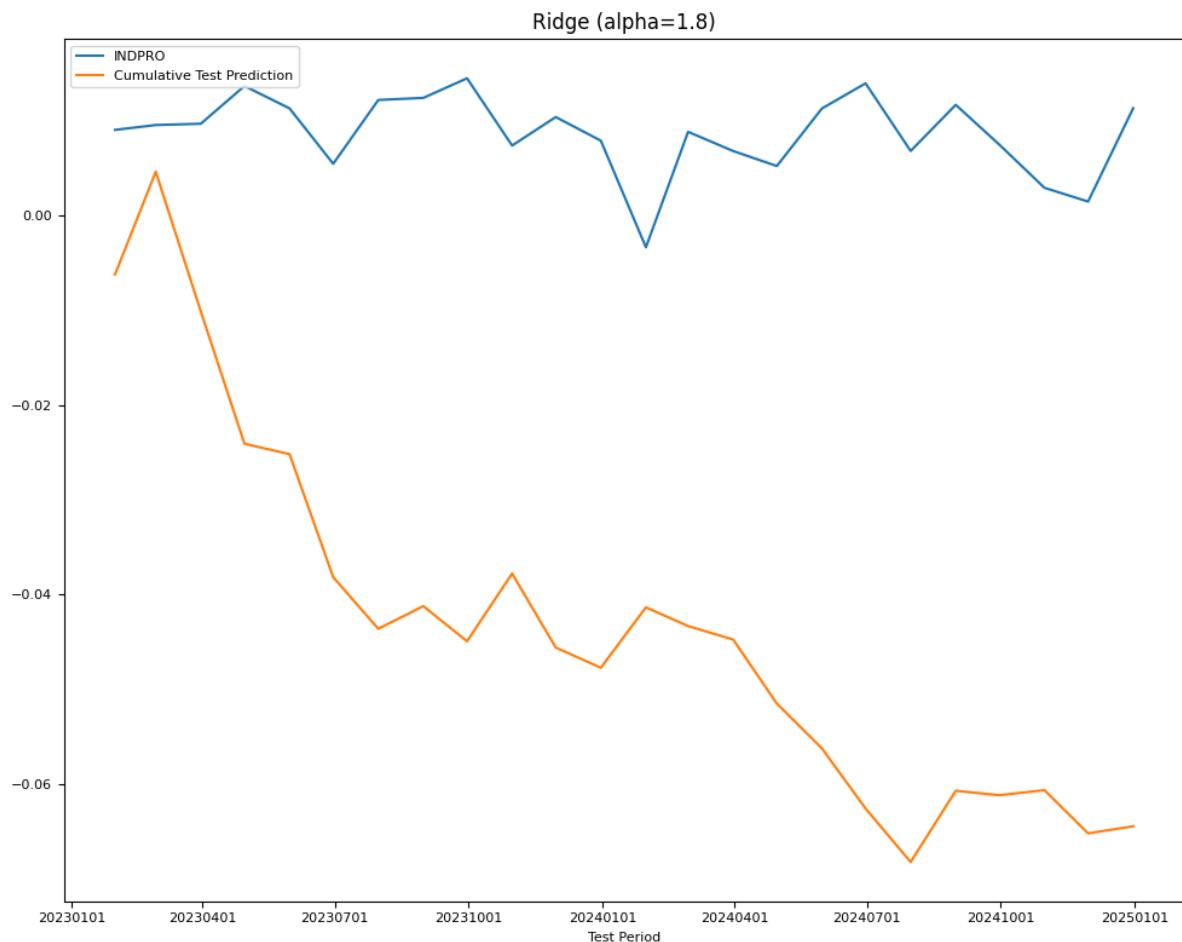
name = f"Ridge (alpha={model.alpha_:.1f})"
Y_pred = Series(index=Y_test.index,
                 data=model.predict(X_test).reshape((-1,)))
test[name] = mean_squared_error(Y_test, Y_pred)
train[name] = mean_squared_error(Y_train, model.predict(X_train))
final_models[name] = model

```

```

fig, ax = plt.subplots(figsize=(10, 8))
plot_date(pd.concat([Y_test.cumsum(), Y_pred.cumsum()], axis=1),
          legend1=[target_id, 'Cumulative Test Prediction'],
          title=name,
          xlabel='Test Period',
          fontsize=8,
          ax=ax)
plt.tight_layout()

```



```

DataFrame({ 'name': name,
            'train': np.sqrt(train[name]),
            'test': np.sqrt(test[name]) }, index=['RMSE'])

```

	name	train	test
RMSE	Ridge (alpha=1.8)	0.004305	0.009825

27.2.4 Lasso Regression

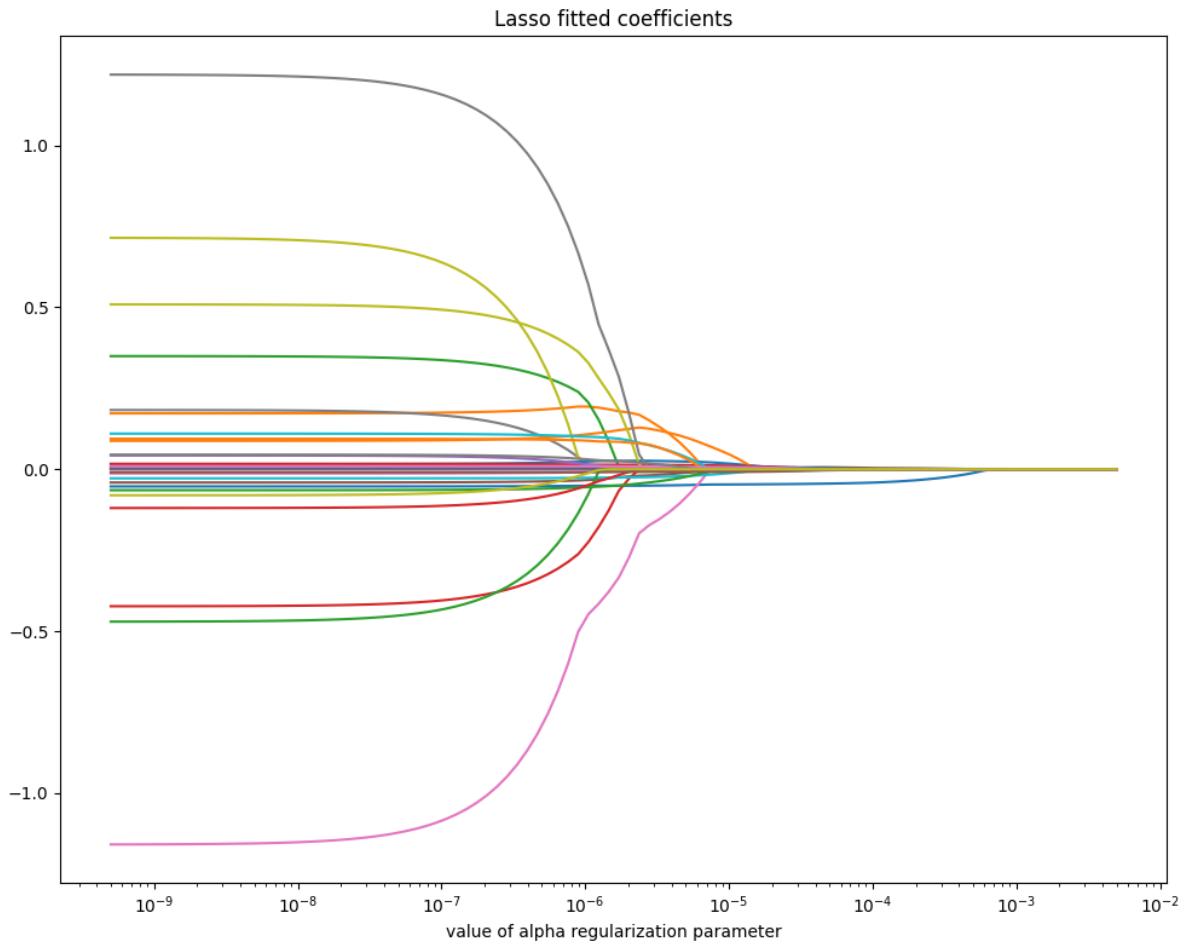
Lasso Regression (Least Absolute Shrinkage and Selection Operator) is a linear regression model that includes **L1 regularization**, which adds the sum of the absolute values of the coefficients as a penalty term to the loss function. This encourages sparsity, meaning it shrinks some coefficients to exactly zero, effectively performing feature selection. The objective function is:

$$\min_w \|y - Xw\|_2^2 + \alpha \|w\|_1$$

where α is a hyperparameter controlling the strength of regularization: a higher α increases shrinkage, leading to more coefficients being set to zero. Lasso regression is particularly useful when dealing with high-dimensional data where many features may be irrelevant, by reducing model complexity and improving interpretability. However, with correlated features, it tends to arbitrarily select one and ignore the others.

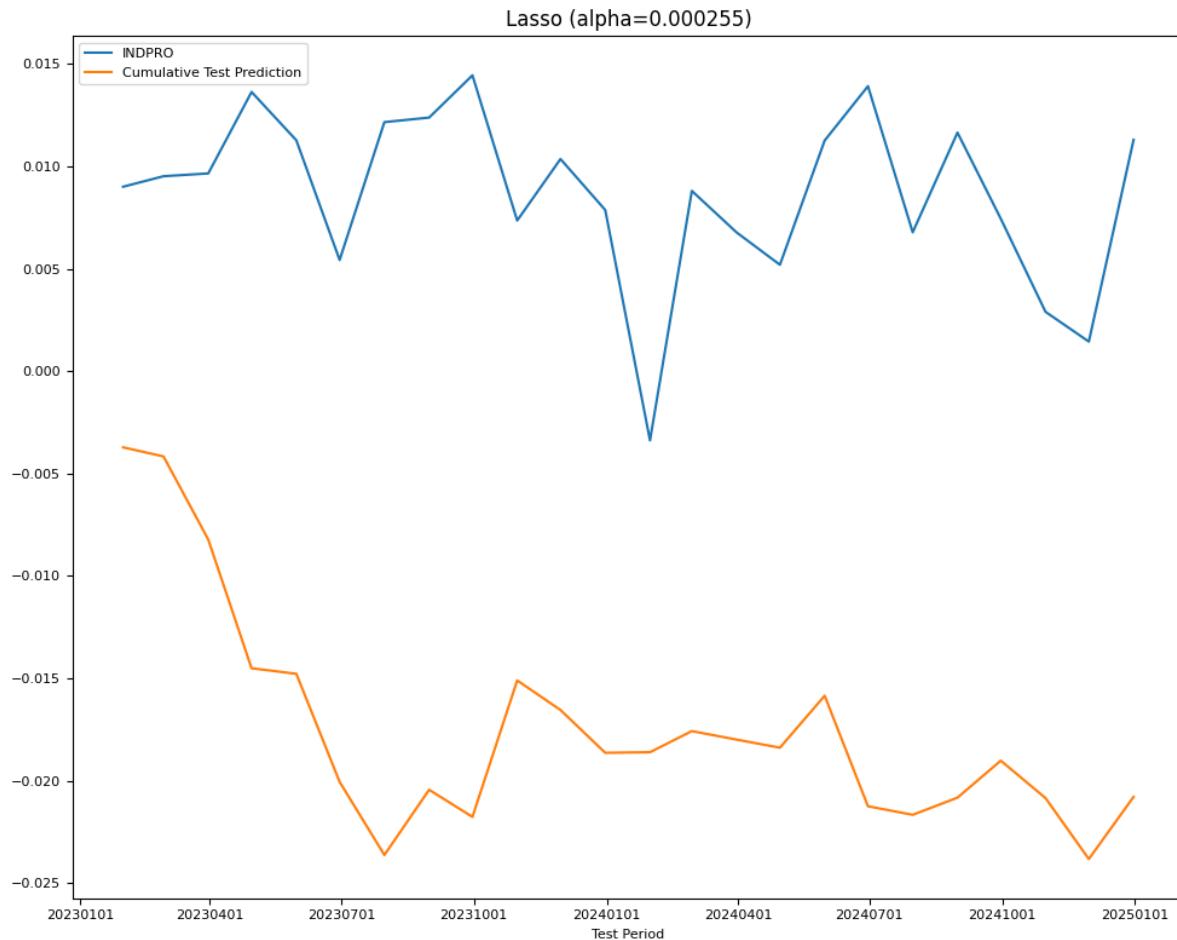
```
from sklearn.linear_model import Lasso, LassoCV
alphas = 10**np.linspace(-2, -9, 100)*0.5 # for parameter tuning
X_train, X_test, Y_train, Y_test = ts_split(X, Y)
scale = StandardScaler().fit(X_train)
X_train = scale.transform(X_train)
X_test = scale.transform(X_test)
```

```
# Plot fitted coefficients vs regularization
coefs = [Lasso(max_iter=10000, alpha=alpha) \
          .fit(X_subset, Y_train).coef_ for alpha in alphas]
fig, ax = plt.subplots(figsize=(10, 8))
ax.plot(alphas, coefs)
ax.set_xscale('log')
ax.set_xlabel('value of alpha regularization parameter')
ax.set_title('Lasso fitted coefficients')
plt.tight_layout()
```



```
# LassoCV 10-Fold CV
model = LassoCV(alphas=None,
                  cv=5,
                  n_jobs=5,
                  verbose=VERBOSE,
                  max_iter=20000).fit(X_train, Y_train)
name = f'Lasso (alpha={model.alpha_:.3g})'
Y_pred = Series(index=Y_test.index,
                 data=model.predict(X_test).reshape((-1,)))
test[name] = mean_squared_error(Y_test, Y_pred)
train[name] = mean_squared_error(Y_train, model.predict(X_train))
final_models[name] = model
```

```
fig, ax = plt.subplots(figsize=(10, 8))
plot_date(pd.concat([Y_test.cumsum(), Y_pred.cumsum()], axis=1),
          legend1=[target_id, 'Cumulative Test Prediction'],
          title=name,
          xlabel='Test Period',
          fontsize=8,
          ax=ax)
plt.tight_layout()
```



```
DataFrame({'name': name,
           'train': np.sqrt(train[name]),
           'test': np.sqrt(test[name])}, index=['RMSE'])
```

	name	train	test
RMSE	Lasso (alpha=0.000255)	0.006279	0.006808

```
# Display nonzero coeffs
nonzero = np.sum(np.abs(model.coef_) > 0)
argsort = np.flip(np.argsort(np.abs(model.coef_)))[:nonzero]
df = DataFrame({'series_id': columns_unmap(X.columns[argsort])[0],
                'lags': columns_unmap(X.columns[argsort])[1],
                'desc': [alf.header(s)
                         for s in columns_unmap(X.columns[argsort])[0]],
                'coef': model.coef_[argsort]}).round(6).set_index('series_id')
print("Lasso: Nonzero Coefficients")
df
```

Lasso: Nonzero Coefficients

series_id	lags	desc	coef
CLAIMS	1	Initial Claims	-0.005356
SRVPRD	1	All Employees, Service-Providing	-0.001268
M2REAL	2	Real M2 Money Stock	0.000893
IPNMAT	3	Industrial Production: Nondurable Goods Materials	0.000751
USGOVT	1	All Employees, Government	-0.000716
...
M2SL	3	M2	0.000020
COMPAPFF	3	3-Month Commercial Paper Minus FEDFUNDS	-0.000020
UEMPLT5	3	Number Unemployed for Less Than 5 Weeks	0.000019
CUSR0000SAS	3	Consumer Price Index for All Urban Consumers: ...	0.000009
RPI	3	Real Personal Income	0.000002

[82 rows x 3 columns]

27.2.5 Decision Tree

Decision trees partition the predictor space into regions, making predictions based on mean values (for regression) or mode (for classification).

- A **decision tree** consists of a sequence of splitting rules that divide observations into different regions. These trees are typically drawn upside down, with the leaves at the bottom.
- **Terminal nodes** or **leaves** represent the final partitions of the predictor space where observations are grouped.
- **Internal nodes** are points in the tree where the predictor space is split.
- **Branches** connect the nodes and represent different decision paths.
- A **stump** refers to a decision tree with only a single split (one internal node).

Decision trees are constructed using **recursive binary splitting**, a top-down, greedy algorithm. The process begins with all observations in a single region, and at each step, the predictor space is split into two new branches based on the best possible split at that moment (without considering future steps). The best split is determined by selecting the predictor and cutpoint that minimize a chosen cost function, such as the **Residual Sum of Squares (RSS)** in regression. The process continues recursively until a stopping criterion is met, such as a minimum number of observations per node.

Growing a tree until all leaves are pure (containing only one class) often leads to overfitting. Tree complexity is measured by the number of nodes, and controlling this complexity is crucial. If trees are allowed to grow too large, they may fit the training data well but perform poorly on unseen data. A balance must be found where accuracy on the test set is maximized before further tree growth begins to decrease performance.

Overfitting can be mitigated by pruning the tree using cost complexity pruning. A fully grown tree is simplified by removing less important splits to minimize the following function:

$$\sum_{m=1}^{|T|} \sum_{x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|$$

where $|T|$ is the number of terminal nodes, R_m represents a region in the predictor space, and \hat{y}_{R_m} is the predicted response for that region. Pruning is controlled by a tuning parameter α , where higher values result in smaller trees that generalize better.

For classification tasks, different impurity measures determine the best binary splits:

- **Classification error rate:** Measures the proportion of incorrectly classified observations in a node. While intuitive, it is not sensitive enough for effective tree growth.

$$\hat{\rho}_{m,c} = \frac{n_{m,c}}{n_m}$$

where n_m is the number of observations in node m and $n_{m,c}$ is the number of observations in node m belonging to class c .

- **Gini index:** Measures total variance across classes. A lower Gini index indicates that most observations in a node belong to a single class.

$$G = \sum_{k=1}^K \hat{\rho}_{mk} (1 - \hat{\rho}_{mk})$$

where $\hat{\rho}_{mk}$ represents the proportion of training observations in node m that belong to class k .

- **Entropy:** Measures node impurity. Lower entropy values indicate purer nodes.

$$D = - \sum_{k=1}^K \hat{\rho}_{mk} \log \hat{\rho}_{mk}$$

Deviance is a statistical measure used to assess model performance in classification problems. It is derived from the likelihood function and is a measure of how well the predicted probabilities match the actual class labels. It is calculated as:

$$\text{Deviance} = -2 \sum_{m=1}^g \sum_{c=1}^w n_{m,c} \ln \hat{p}_{m,c}$$

where:

- g is the number of terminal nodes (leaf nodes) in the forest.
- w is the number of classes.
- $n_{m,c}$ is the number of observations in node m belonging to class c .
- $\hat{p}_{m,c}$ is the predicted probability of class c in node m .

with the **residual mean deviance** given by: $\frac{\text{deviance}}{n-1}$

A lower deviance indicates better model performance, meaning the predicted class probabilities align more closely with the true class labels.

Advantages:

- Easy to interpret and visualize, even for non-experts.
- Often reflect human decision-making processes.
- Can handle qualitative predictors without requiring dummy variables.

Disadvantages:

- Less accurate than some other regression and classification models.
- Highly sensitive to small changes in data, making them non-robust.
- Tend to overfit categorical variables.

Decision tree performance can be significantly improved using **ensemble methods** such as:

- **Bagging (Bootstrap Aggregating):** Reduces variance by averaging predictions from multiple decision trees trained on different bootstrap samples.
- **Random Forests:** A variant of bagging that introduces additional randomness by selecting a random subset of features at each split.
- **Boosting:** Sequentially builds trees where each new tree corrects the errors of the previous ones, reducing bias and improving predictive performance.

By aggregating multiple decision trees, these ensemble methods enhance generalization and reduce overfitting, making decision trees more robust and effective for complex predictive modeling.

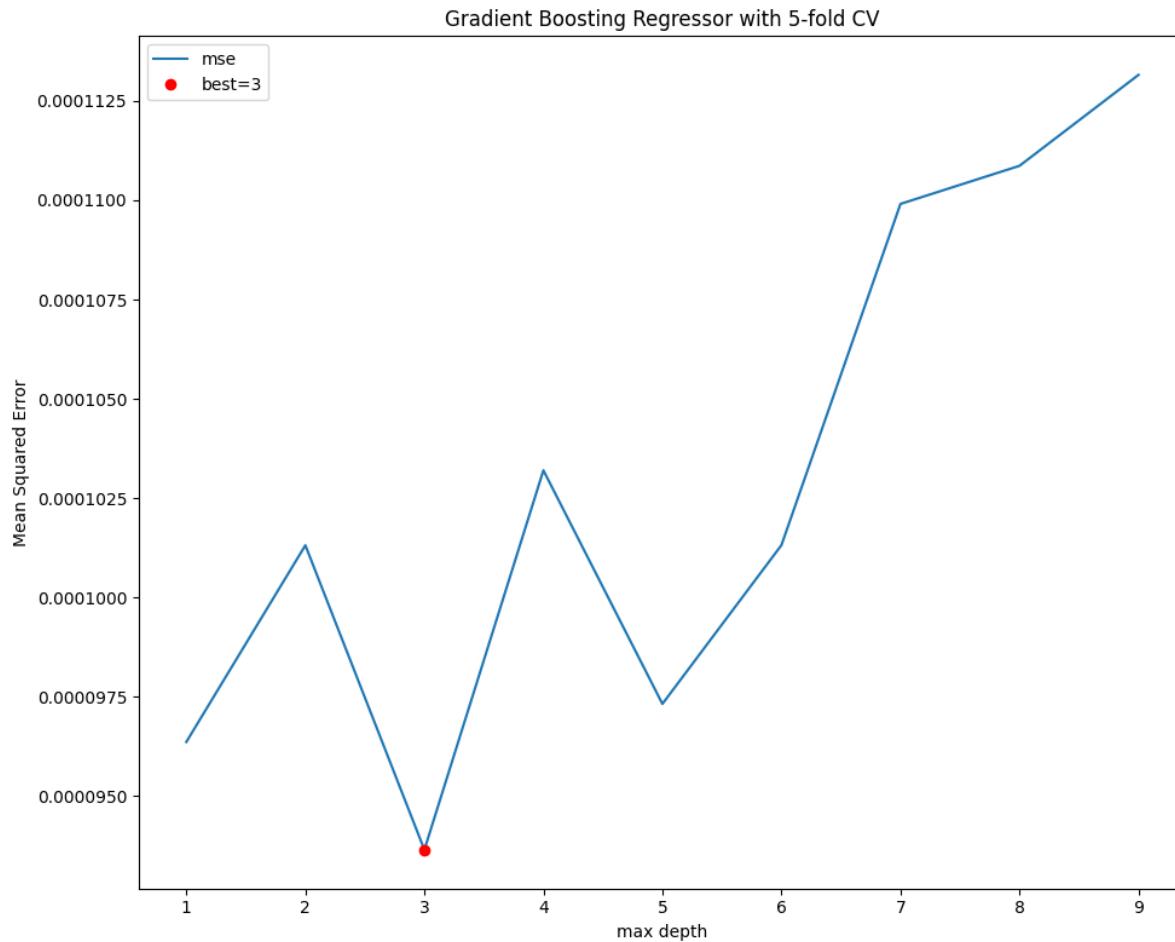
27.2.6 Gradient boosting

Gradient Boosting is an ensemble learning technique that builds models sequentially, where each new model corrects the errors of the previous ones. It uses decision trees as weak learners and minimizes the loss function by optimizing the model in the direction of the gradient of the loss. Key parameters include `n_estimators` (number of trees), `learning_rate` (step size for updates, controlling how much each tree contributes), and `max_depth` (tree depth, preventing overfitting). Gradient Boosting can be computationally expensive and prone to overfitting if not regularized using techniques like early stopping or shrinkage (learning_rate tuning).

```
from sklearn.ensemble import GradientBoostingRegressor
X_train, X_test, Y_train, Y_test = ts_split(X, Y)
scale = StandardScaler().fit(X_train)
X_train = scale.transform(X_train)
X_test = scale.transform(X_test)

# tune max_depth with 5-fold CV
n_splits=5
kf = KFold(n_splits=n_splits, shuffle=True, random_state=0)
mse = Series(dtype=float)
for i in range(1, 10): # tune max_depth for best performance
    boosted = GradientBoostingRegressor(max_depth=i, random_state=0)
    score = cross_val_score(boosted,
                           X_train,
                           Y_train,
                           cv=kf,
                           n_jobs=5,
                           verbose=VERBOSE,
                           scoring='neg_mean_squared_error').mean()
    mse.loc[i] = -score
```

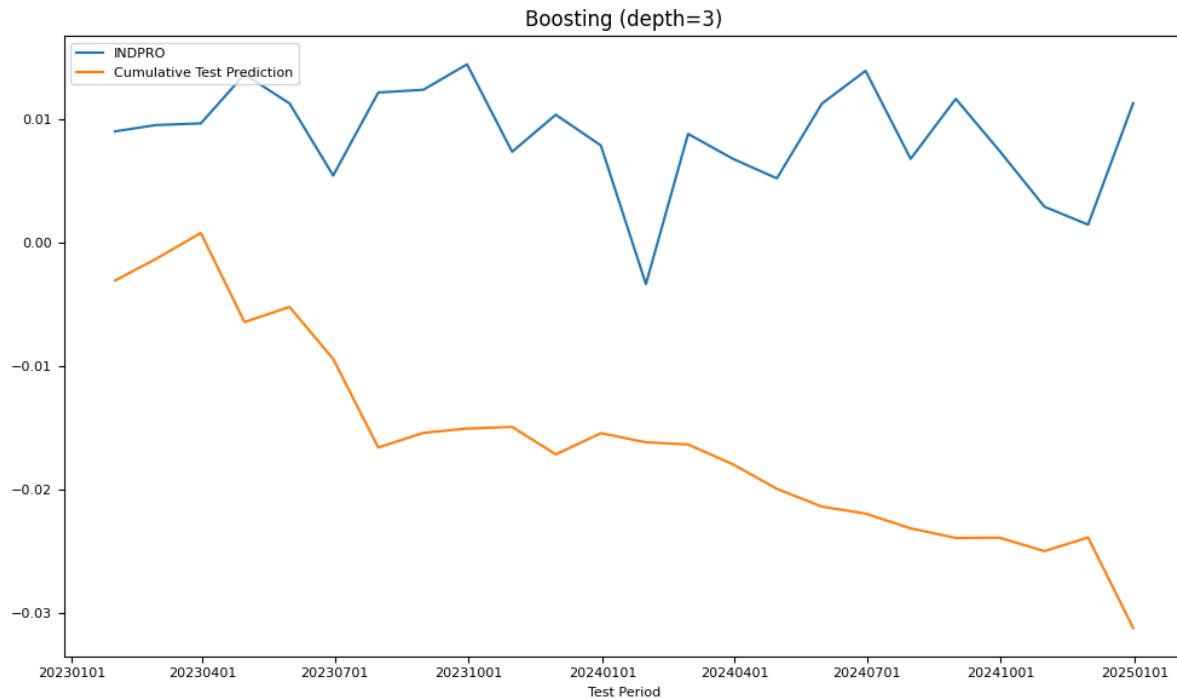
```
fig, ax = plt.subplots(figsize=(10, 8))
mse.plot(ax=ax, ylabel='Mean Squared Error', xlabel='max depth',
         title=f"Gradient Boosting Regressor with {n_splits}-fold CV")
best = mse.index[mse.argmin()]
ax.plot(best, mse.loc[best], "or")
ax.legend(['mse', f"best={best}"])
plt.tight_layout()
```



```
# evaluate train and test MSE
name = f"Boosting (depth={best})"
model = GradientBoostingRegressor(max_depth=best,
                                    random_state=0).fit(X_train, Y_train)
```

```
Y_pred = Series(index=Y_test.index,
                 data=model.predict(X_test).reshape((-1,)))
test[name] = mean_squared_error(Y_test, Y_pred)
train[name] = mean_squared_error(Y_train, model.predict(X_train))
final_models[name] = model
```

```
fig, ax = plt.subplots(figsize=(10, 6))
plot_date(pd.concat([Y_test.cumsum(), Y_pred.cumsum()], axis=1),
          legend1=[target_id, 'Cumulative Test Prediction'],
          title=name,
          xlabel='Test Period',
          fontsize=8,
          ax=ax)
plt.tight_layout()
```



```
DataFrame({'name': name,
           'train': np.sqrt(train[name]),
           'test': np.sqrt(test[name])}, index=['RMSE'])
```

	name	train	test
RMSE	Boosting (depth=3)	0.002704	0.007407

Feature importances:

```
# Show feature importance
top_n = 10
imp = Series(model.feature_importances_, index=X.columns).sort_values()
print(f"Gradient Boosting: Top {top_n} Feature Importances")
DataFrame.from_dict({i+1: {'importance': imp[s],
                            'series_id': s.split('_')[0],
                            'lags': s.split('_')[1],
                            'description': alf.header(s.split('_')[0])}
                     for i, s in enumerate(np.flip(imp.index[-top_n:]))},
                     orient='index')
```

Gradient Boosting: Top 10 Feature Importances

	importance	series_id	lags
1	0.140886	BUSLOANS	1
2	0.136621	CLAIMS	1
3	0.031817	UEMPLT5	3
4	0.030436	M2REAL	3
5	0.029806	USGOOD	1
6	0.027816	M2SL	1
7	0.022606	IPNMAT	1

(continues on next page)

(continued from previous page)

```

8      0.020286    DMANEMP      1
9      0.019784    EXCAUS      1
10     0.018134   UNRATE      1

                           description
1  Commercial and Industrial Loans, All Commercia...
2                      Initial Claims
3  Number Unemployed for Less Than 5 Weeks
4                      Real M2 Money Stock
5  All Employees, Goods-Producing
6                           M2
7  Industrial Production: Nondurable Goods Materials
8  All Employees, Durable Goods
9  Canadian Dollars to U.S. Dollar Spot Exchange ...
10     Unemployment Rate

```

27.2.7 Random Forest

Random Forest is an ensemble learning method that builds multiple decision trees and combines their predictions to improve accuracy and reduce overfitting. It operates by randomly selecting subsets of the training data and features (using bootstrap sampling and feature bagging) to create diverse trees. The final prediction is determined by majority voting (for classification) or averaging (for regression). Key parameters include `n_estimators` (number of trees), `max_depth` (maximum depth of each tree), `max_features` (number of features considered for splitting), and `min_samples_split` (minimum samples needed to split a node). Feature importance scores can be extracted to interpret which variables influence predictions most. Random Forest handles missing data well and mitigates overfitting through averaging multiple models.

```

from sklearn.ensemble import RandomForestRegressor
X_train, X_test, Y_train, Y_test = ts_split(X, Y)

```

```

# tune max_depth with 5-fold CV
n_splits=5
kf = KFold(n_splits=n_splits,
            shuffle=True,
            random_state=0)
mse = Series(dtype=float)
for i in range(3, 20): #tune for best performance
    model = RandomForestRegressor(max_depth=i, random_state=0)
    score = cross_val_score(model,
                            X_train,
                            Y_train,
                            cv=kf,
                            n_jobs=5,
                            verbose=VERBOSE,
                            scoring='neg_mean_squared_error').mean()
    mse.loc[i] = -score
    #print(i, np.sqrt(abs(score)))

```

```

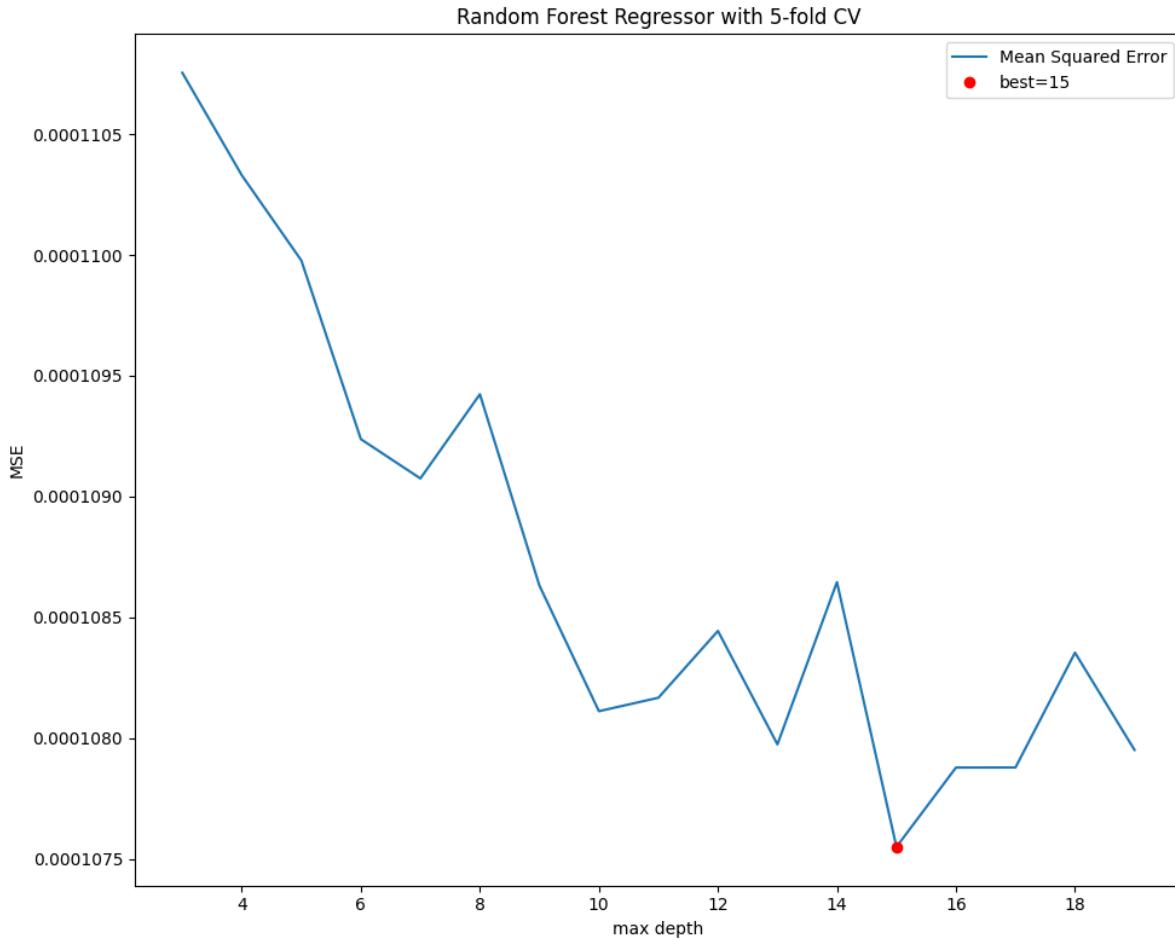
fig, ax = plt.subplots(figsize=(10, 8))
mse.plot(ax=ax, ylabel='MSE', xlabel='max depth',
         title=f"Random Forest Regressor with {n_splits}-fold CV")
best = mse.index[mse.argmin()]

```

(continues on next page)

(continued from previous page)

```
ax.plot(best, mse.loc[best], "or")
ax.legend(['Mean Squared Error', f"best={best}"])
plt.tight_layout()
```



```
name = f"RandomForest (depth={best})"
model = RandomForestRegressor(max_depth=best,
                             random_state=0).fit(X_train, Y_train)
```

```
Y_pred = Series(index=Y_test.index,
                 data=model.predict(X_test).reshape((-1,)))
test[name] = mean_squared_error(Y_test, Y_pred)
train[name] = mean_squared_error(Y_train, model.predict(X_train))
final_models[name] = model
```

```
fig, ax = plt.subplots(figsize=(10, 8))
plot_date(pd.concat([Y_test.cumsum(), Y_pred.cumsum()], axis=1),
          legend1=[target_id, 'Cumulative Test Prediction'],
          title=name,
          xlabel='Test Period',
          fontsize=8,
          ax=ax)
```

(continues on next page)

(continued from previous page)

```
plt.tight_layout()
```



```
DataFrame({ 'name': name,
            'train': np.sqrt(train[name]),
            'test': np.sqrt(test[name])}, index=['RMSE'])
```

	name	train	test
RMSE	RandomForest (depth=15)	0.00376	0.005875

Feature importance scores:

```
# Feature importance
top_n = 10
imp = Series(model.feature_importances_, index=X.columns).sort_values()
print(f"Random Forest: Top {top_n} Feature Importances")
DataFrame.from_dict({i+1: { 'importance': imp[s],
                            'series_id': s.split('_')[0],
                            'lags': s.split('_')[1],
                            'description': alf.header(s.split('_')[0])}
                     for i, s in enumerate(np.flip(imp.index[-top_n:]))},
                     orient='index')
```

Random Forest: Top 10 Feature Importances

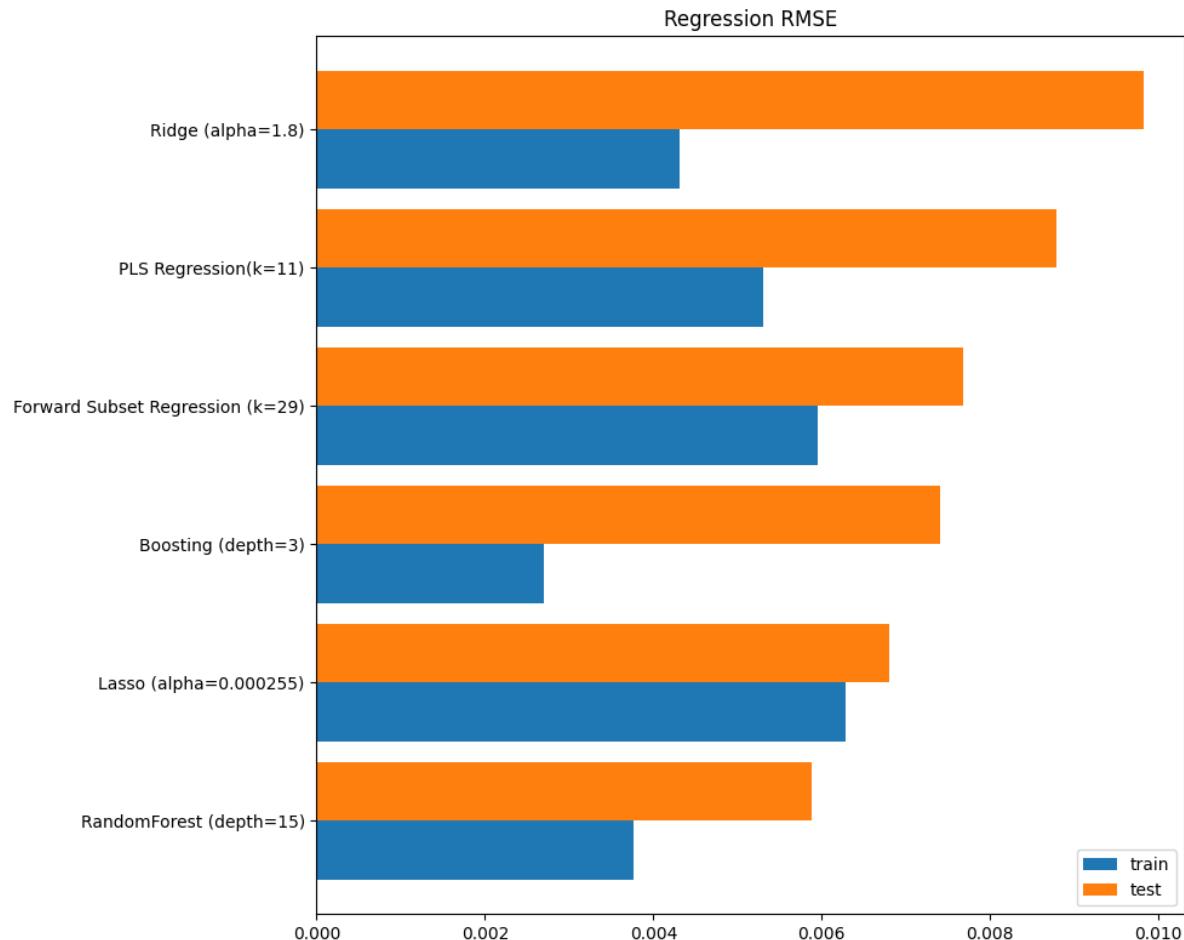
```
importance series_id lags \
1 0.051226 CLAIMS 1
2 0.046733 BUSLOANS 1
3 0.044055 M2SL 1
4 0.015825 USGOOD 1
5 0.015268 UNRATE 1
6 0.014617 UEMPLT5 3
7 0.014576 IPCONGD 1
8 0.014529 UEMPLT5 1
9 0.014496 M1SL 1
10 0.013855 DMANEMP 1

description
1 Initial Claims
2 Commercial and Industrial Loans, All Commercia...
3 M2
4 All Employees, Goods-Producing
5 Unemployment Rate
6 Number Unemployed for Less Than 5 Weeks
7 Industrial Production: Consumer Goods
8 Number Unemployed for Less Than 5 Weeks
9 M1
10 All Employees, Durable Goods
```

Summary:

The **root mean squared error (RMSE)** performance of all the models, based on train and test samples, is summarized in the following chart:

```
fig, ax = plt.subplots(figsize=(10, 8))
pd.concat([np.sqrt(r.to_frame()) for r in [train, test]], axis=1) \
    .sort_values('test') \
    .plot.barh(ax=ax, width=0.85)
ax.yaxis.set_tick_params(labelsize=10)
ax.set_title('Regression RMSE')
ax.figure.subplots_adjust(left=0.35)
plt.tight_layout()
```

**References:**

Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani. "An Introduction to Statistical Learning with Applications in R". New York, Springer, 2013.

CHAPTER
TWENTYEIGHT

DEEP LEARNING

May your choices reflect your hopes, not your fears – Nelson Mandela

We explore the application of deep learning techniques for text classification, specifically focusing on categorizing US companies based on their industry sectors. Using business description texts extracted from SEC 10-K filings, we apply natural language processing (NLP) methods and deep averaging networks (DAN) to classify firms according to the Fama-French 10-sector scheme. The analysis includes preprocessing textual data, leveraging pre-trained word embeddings for semantic representation, and evaluating various training strategies to optimize predictive accuracy and generalization performance.

```
# By: Terence Lim, 2020-2025 (terence-lim.github.io)
import numpy as np
import random
import time
import pandas as pd
from pandas import DataFrame, Series
from collections import Counter
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
import seaborn as sns
from tqdm import tqdm
import torch
from torch import nn
import torchinfo
from textblob import TextBlob
from tqdm import tqdm
from finds.database import SQL, RedisDB
from finds.unstructured import Edgar, Vocab
from finds.structured import BusDay, CRSP, PSTAT
from finds.readers import Sectoring
from finds.utils import Store
from secret import credentials, paths, CRSP_DATE
VERBOSE = 0
outdir = paths['scratch']
store = Store(outdir, ext='pkl')
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"device={device}")
print(f"torch.cuda.is_available()={torch.cuda.is_available()}"") # Should return True
print(f"torch.cuda.device_count()={torch.cuda.device_count()}"") # Number of available GPUs
print(f"torch.cuda.current_device()={torch.cuda.current_device()}"") # Current GPU index
print(f"torch.cuda.get_device_name(0)={torch.cuda.get_device_name(0)}"") # Name of the GPU
```

```
device=device(type='cuda')
torch.cuda.is_available()=True
torch.cuda.device_count()=1
torch.cuda.current_device()=0
torch.cuda.get_device_name(0)='NVIDIA GeForce RTX 3080 Laptop GPU'
```

```
sql = SQL(**credentials['sql'], verbose=VERBOSE)
user = SQL(**credentials['user'], verbose=VERBOSE)
bd = BusDay(sql)
rdb = RedisDB(**credentials['redis'])
crsp = CRSP(sql, bd, rdb, verbose=VERBOSE)
pstat = PSTAT(sql, bd, verbose=VERBOSE)
ed = Edgar(paths['10X'], zipped=True, verbose=VERBOSE)
```

28.1 Industry text classification

We begin by extracting a universe of US-domiciled common stocks at the start of the most recent year, along with their corresponding 10-K business descriptions from SEC filings. The target categories for our text classification task are drawn from the Fama-French 10-sector classification scheme.

```
# Retrieve universe of stocks as of start of latest year
univ = crsp.get_universe(bd.endmo(CRSP_DATE-10000))
CRSP_DATE
```

20241231

```
# lookup company names
comnam = crsp.build_lookup(source='permno', target='comnam', fillna="")
univ['comnam'] = comnam(univ.index)
```

```
# lookup ticker symbols
ticker = crsp.build_lookup(source='permno', target='ticker', fillna="")
univ['ticker'] = ticker(univ.index)
```

```
# lookup sic codes from Compustat, and map to FF 10-sector code
sic = pstat.build_lookup(source='lpermno', target='sic', fillna=0)
industry = Series(sic[univ.index], index=univ.index)
industry = industry.where(industry > 0, univ['siccd'])
sectors = Sectoring(sql, scheme='codes10', fillna='') # supplement from crosswalk
univ['sector'] = sectors[industry]
```

```
# retrieve 2023 10K business descriptions text
item, form = 'bus10K', '10-K'
rows = DataFrame(ed.open(form=form, item=item))
found = rows[rows['date'].between(bd.begyr(CRSP_DATE), bd.endyr(CRSP_DATE))]\n    .drop_duplicates(subset=['permno'], keep='last')\n    .set_index('permno')
```

28.1.1 Textblob

The `TextBlob` library simplifies common NLP tasks such as part-of-speech tagging, lemmatization, noun phrase extraction, sentiment analysis, and spelling correction. It provides friendly access to functionalities derived from NLTK and integrates with WordNet.

- <https://textblob.readthedocs.io/en/dev/quickstart.html>

For our task, `TextBlob` is employed to tokenize business descriptions and extract nouns. We filter the documents to retain only those containing at least 100 valid nouns to ensure robust semantic representation.

```
bus = {}
for permno in tqdm(found.index):
    if permno not in univ.index:
        continue
    doc = TextBlob(ed[found.loc[permno, 'pathname']].lower()) # tokenize and tag
    nouns = [word for word, tag in doc.tags
             if tag in ['NN', 'NNS'] and word.isalpha() and len(word) > 2]
    if len(nouns) > 100:
        bus[permno] = nouns
permnos = list(bus.keys())
```

100% |██████████| 4488/4488 [17:23<00:00, 4.30it/s]

28.1.2 Word Embeddings

Word embeddings are dense, numerical vector representations capturing semantic and syntactic meanings of words. These embeddings place words into a continuous vector space, positioning semantically similar or related words closely together. Word embeddings can be generated through neural network-based approaches or matrix factorization methods.

1. **Word2Vec** (Mikolov et al., 2013): Word2Vec utilizes shallow neural networks, usually comprising a single hidden layer, to learn embeddings from textual data. It has two primary training approaches:
 - **Skip-gram**: Predicts context words given a center word, effectively capturing representations of rare words.
 - **Continuous Bag of Words (CBOW)**: Predicts a center word from surrounding context words, typically faster and better for frequent words.
2. **GloVe (Global Vectors for Word Representation)** (Pennington et al., 2014): GloVe generates embeddings based on matrix factorization of global word-word co-occurrence statistics. Unlike Word2Vec, which relies on local context predictions, GloVe considers overall word pair co-occurrences, resulting in globally consistent embeddings.

Pre-trained GloVe vectors (300-dimensional) are utilized to represent the extracted words as embeddings.

```
# Load GloVe embeddings, source: "https://nlp.stanford.edu/data/glove.6B.zip"
embeddings_dim = 300 # dimension of GloVe embeddings vector

filename = paths['scratch'] / f"glove.6B.{embeddings_dim}d.txt.zip"
embeddings = pd.read_csv(filename, sep=" ", quoting=3,
                        header=None, index_col=0, low_memory=True)
embeddings.index = embeddings.index.astype(str).str.lower()
print(embeddings.shape)
```

(400000, 300)

28.1.3 Word vector arithmetic

Word embeddings reflect linguistic relationships through geometric relationships in vector space. Embeddings can be arithmetically combined and manipulated to uncover analogies and semantic similarities. However, these mathematical relationships are generally approximate and can highlight potential biases inherent in training data, such as implicit gender biases.

```
from sklearn.neighbors import NearestNeighbors
analogies = ["man king woman", "paris france tokyo", "big bigger cold"]
for analogy in analogies:
    words = analogy.lower().split()
    vectors = {word: embeddings.loc[word].values for word in words}
    vec = vectors[words[1]] - vectors[words[0]] + vectors[words[2]]

    sim = NearestNeighbors(n_neighbors=1).fit(embeddings)
    neighbors = sim.kneighbors(vec.reshape((1, -1)), n_neighbors=2,
                                return_distance=False).flatten().tolist()
    neighbors = [k for k in neighbors if embeddings.index[k] not in words]
    print(f"{words[1]} - {words[0]} + {words[2]} =",
          [embeddings.index[k] for k in neighbors])
```

```
king - man + woman = ['queen']
france - paris + tokyo = ['japan']
bigger - big + cold = ['colder']
```

28.1.4 Data preparation

We construct a custom vocabulary (Vocab) mapping each word to an index, encoding each document as a list of these indices. The pre-trained GloVe embedding matrix is adapted to include only words present in our corpus-specific vocabulary. Sector labels are converted into numerical values using LabelEncoder. The dataset is then stratified and split into training and testing subsets to maintain balanced class distributions.

```
words = Counter()
for nouns in bus.values():
    words.update(list(nouns))
vocab = Vocab(words.keys())
print('vocab len:', len(vocab))
```

```
vocab len: 85891
```

```
labels = []
x_all = []
for permno, nouns in bus.items():
    x = vocab.get_index([noun for noun in nouns])
    if sum(x):
        labels.append(univ.loc[permno, 'sector'])
        x_all.append(x)
class_encoder = LabelEncoder().fit(labels)      # .inverse_transform()
y_all = class_encoder.transform(labels)
```

```
store['dan'] = dict(y_all=y_all, x_all=x_all)
```

```

# retrieve from previously stored
y_all, x_all = store['dan'].values()

# relativize embeddings to words in vocab
vocab.set_embeddings(embeddings)
print(vocab.embeddings.shape)

(85891, 300)

vocab.dump(outdir / f"dan{embeddings_dim}.pkl")

# load vocab
vocab.load(outdir / f"dan{embeddings_dim}.pkl")

```

28.2 Feedforward neural networks

Neural networks are computational models inspired by the human brain. They are built from layers of simple computational units that transform input data to output predictions. Deep neural networks alternate between linear layers and non-linear activations, and can approximate any continuous function (Universal Approximation Theorem).

- **Neurons** are the basic computational units or nodes of a neural network. Each neuron receives input, processes it using a weighted sum and a bias term, and then applies an activation function to produce an output, which is then passed to the neurons in the next layer.
- **Activation functions** are the nonlinear mathematical functions applied to neurons in a neural network. They introduce non-linearity into the model, enabling it to learn and represent complex patterns in the data. Common activation functions include ReLU (Rectified Linear Unit), sigmoid, and tanh.
- **Input Layer** is the first layer of a neural network which directly receives the input data. Each neuron in the input layer represents one feature of the input.
- **Hidden Layers**, between the input layer and the output layer, take input from the previous layer of neurons, apply weights, biases, and activation functions, and pass the output to the next layer.
- **Output Layer** is the final layer of the neural network and it produces the network's output. Its neurons represent the predictions or classifications made by the network. The number of neurons in the output layer corresponds to the number of output classes or the dimensionality of the output. For classification tasks, softmax or sigmoid functions are often used in the output layer to provide probability distributions of the class predictions.

Feedforward neural networks (FFNNs) are the simplest form of neural networks, where the data flows in one direction (a forward pass) and the connections do not form a cycle. A **Multilayer Perceptron (MLP)** is a type of FFNN which must have at least one hidden layer: MLPs are composed of an input layer, one or more hidden layers, and an output layer, with non-linear activation functions applied between layers.

Optimization is the process of adjusting model parameters to align its predictions with true targets.

- **Loss function** measures how well a neural network's output matches the true label or target. During training, the goal is to minimize this loss. Common loss functions L1 (Mean Absolute Error) and L2 (Mean Squared Error) for regression tasks, and Cross-Entropy for classification tasks.
- **Stochastic Gradient Descent (SGD)** is an optimization method used to train neural networks by updating parameters using gradients from a single (or small batch of) data point(s) at each step. It allows efficient updates even on massive datasets, with the ability to escape local minima due to its noise.

- **Backpropagation** is used for training neural networks by updating the weights of neurons based on the error (loss) of the network's predictions: it involves calculating the gradient of the loss function with respect to each weight by using the chain rule of calculus, and propagating these gradients backward from the output layer to the input layer.
- **Computation Graph** is a graphical representation of the sequence of operations used to compute the forward pass and the backward pass for backpropagation. PyTorch's modules automatically constructs the computation graph and computes gradients, hence simplifying the implementation of neural networks.
- **Initialization** refers to the process of setting the initial values of the weights in a neural network before training begins. Poor initialization can lead to slow convergence or getting stuck in local minima. Common initialization methods include Xavier (Glorot) and He initialization

Training deep neural networks involves carefully tuning several key components to ensure effective learning and generalization.

- **Learning Rate**: If too low, training is slow; too high and loss spikes. A learning rate schedules (e.g. cosine annealing) is more efficient than a fixed learning rate.
- **Adam** (Adaptive Moment Estimation) is an optimization algorithm for training neural networks which improves on stochastic gradient descent and achieves good performance on problems with large, high-dimensional data sets. It adapts the learning rate for each parameter by computing adaptive learning rates from estimates of first and second moments of the gradients. **AdamW** improves the performance of Adam in deep networks by applying weight decay directly to model parameters separately from gradient-based updates.
- **Hyperparameters** are parameters not learned by the neural network during training. They are set before training and control how the network learns. Examples include: Learning rate (size of each update step in gradient descent); Number of epochs (how many times the model sees the entire dataset); and Model size (number of layers, units per layer).
- **Batching** divides the training data into smaller subsets called batches, rather than training the model on the entire dataset at once, which can be computationally intensive and inefficient. It also gives speedup compared to training the network one sample at a time due to more efficient matrix operations.
- **Dropout** is a regularization technique during training, where a random subset of neurons is “dropped out” or set to zero at each iteration. This reduces overfitting by ensuring that the model does not rely too heavily on any particular subset of neurons. Geoffrey Hinton, et al. in their 2012 paper that first introduced dropout. They found that using a simple method of 50% dropout for all hidden units and 20% dropout for input units achieve improved results with a range of neural networks on different problem types. It is not used on the output layer.

28.2.1 Deep Averaging Networks

Deep Averaging Network (DAN) is a straightforward feedforward neural network architecture used for text classification. It averages embeddings of document words and feeds this representation through multiple hidden layers to predict class labels. Key properties include:

- **Embedding Layer**: Uses pre-trained GloVe vectors (frozen or fine-tuned).
- **Fully Connected Layers**: Transform embeddings into classification scores.
- **Nonlinear Activations**: Employ ReLU for non-linearity.
- **Output Layer**: Applies LogSoftmax for multi-class predictions.
- **Dropout Layers**: Prevent overfitting.
- **Xavier Initialization**: Stabilizes training.

We investigate training strategies, such as frozen embeddings (fast, prevents overfitting on small data), fine-tuned embeddings (task-specific optimization but resource-intensive), and dropout regularization (enhances generalization).

```

class DAN(nn.Module):
    """Deep Averaging Network for classification"""
    def __init__(self,
                 vocab_dim,
                 num_classes,
                 hidden,
                 embedding,
                 freeze=True):
        super().__init__()
        self.embedding = nn.EmbeddingBag.from_pretrained(embedding)
        self.embedding.weight.requires_grad = not freeze
        D = nn.Dropout(0.0)
        V = nn.Linear(vocab_dim, hidden[0])
        nn.init.xavier_uniform_(V.weight)
        L = [D, V]
        self.drops = [D]
        for in_dim, out_dim in zip(hidden, hidden[1:] + [num_classes]):
            L.append(nn.ReLU())    # nonlinearity layer
            D = nn.Dropout(0.0)
            self.drops.append(D)
            L.append(D)            # dropout layer
            W = nn.Linear(in_dim, out_dim)    # dense linear layer
            nn.init.xavier_uniform_(W.weight)
            L.append(W)
        self.network = nn.Sequential(*L)
        self.classifier = nn.LogSoftmax(dim=-1)    # output is (N, C) logits

    def set_dropout(self, dropout):
        if dropout:
            self.drops[0].p = 0.2    # input layer
            for i in range(1, len(self.drops)):    # hidden layers
                self.drops[i].p = 0.5
        else:
            for i in range(len(self.drops)):
                self.drops[i].p = 0.0

    def set_freeze(self, freeze):
        """To freeze part of the model (embedding layer)"""
        self.embedding.weight.requires_grad = not freeze

    def forward(self, x):
        """Return tensor of log probabilities"""
        return self.classifier(self.network(self.embedding(x)))

    def predict(self, x):
        """Return predicted int class of input tensor vector"""
        return torch.argmax(self(x), dim=1).int().tolist()

    def save(self, filename):
        """Save model state to filename"""
        return torch.save(self.state_dict(), filename)

    def load(self, filename):
        """Load model name from filename"""
        self.load_state_dict(torch.load(filename, map_location='cpu'))
        return self

```

Split the data into stratified (i.e. equal class proportions) train and test set

```
# Stratified train-test split
num_classes = len(np.unique(labels))
train_index, test_index = train_test_split(
    np.arange(len(y_all)), stratify=y_all, random_state=42, test_size=0.2)
print(len(x_all), len(y_all), len(train_index), len(test_index), num_classes)
#Series(labels).value_counts().rename('count').to_frame()
pd.concat([Series(np.array(labels)[train_index]).value_counts().rename('Train'),
           Series(np.array(labels)[test_index]).value_counts().rename('Test')],
           axis=1)
```

3474 3474 2779 695 10

	Train	Test
Hlth	657	164
Other	612	153
HiTec	554	139
Manuf	275	69
Shops	246	62
Durbl	131	33
NoDur	114	28
Enrgy	81	20
Utils	72	18
Telcm	37	9

```
# Specify model and training parameters
layers = 2
hidden_size = 32
model = DAN(embeddings_dim,
            num_classes,
            hidden=[hidden_size] * layers,
            embedding=torch.FloatTensor(vocab.embeddings)).to(device)
torchinfo.summary(model)
```

```
=====
Layer (type:depth-idx)           Param #
=====
DAN
├─EmbeddingBag: 1-1           (25,767,300)
├─Sequential: 1-2
|  ├─Dropout: 2-1
|  ├─Linear: 2-2             9,632
|  ├─ReLU: 2-3
|  ├─Dropout: 2-4
|  ├─Linear: 2-5             1,056
|  ├─ReLU: 2-6
|  ├─Dropout: 2-7
|  └─Linear: 2-8             330
└─LogSoftmax: 1-3
=====
Total params: 25,778,318
Trainable params: 11,018
Non-trainable params: 25,767,300
=====
```

28.2.2 Training

Training employs

- **Adam optimizer** for adaptive learning rates.
- **Negative Log Likelihood (NLLLoss)** for multi-class classification.
- Batch training with shuffled data to improve generalization.
- Padding of variable-length word index lists to form uniform-length input tensors.
- Evaluation of both training and test performance per epoch.

```
batch_sz = 16
lr = 0.001
num_epochs = 50
optimizer = torch.optim.Adam(model.parameters(), lr=lr)
loss_function = nn.NLLLoss()
```

Helper function to batch and form an input for neural network. Pads each sample to have lengths equal to the max, and convert to Long tensor type.

```
def form_input(docs):
    """Pad lists of index lists to form batch of equal lengths"""
    lengths = [len(doc) for doc in docs]    # length of each doc
    max_length = max(1, max(lengths))        # to pad so all lengths equal max
    out = [doc + ([0] * (max_length-n)) for doc, n in zip(docs, lengths)]
    return torch.LongTensor(out)
```

```
accuracy = []
for imodel, (freeze, dropout) in enumerate([(True, False), (True, True), (False, True)]):
    model.set_freeze(freeze)
    model.set_dropout(dropout)
    accuracy.append(dict())

    # Loop over epochs
    for epoch in tqdm(range(num_epochs)):
        tic = time.time()

        # Form batches
        random.shuffle(train_index)
        batches = [train_index[i:(i+batch_sz)] for i in range(0, len(train_index), batch_sz)]

        # Train in batches
        total_loss = 0.0
        model.train()
        for batch in batches: # train by batch
            x = form_input([x_all[idx] for idx in batch]).to(device)
            y = torch.LongTensor([y_all[idx] for idx in batch]).to(device)
            model.zero_grad() # reset model gradient
            log_probs = model(x) # run model
            loss = loss_function(log_probs, y) # compute loss
            total_loss += float(loss)
            loss.backward() # loss step
            optimizer.step() # optimizer step
```

(continues on next page)

(continued from previous page)

```

model.eval()
model.save(outdir / f"dan{embeddings_dim}.pt")

if VERBOSE:
    print(f"Loss {epoch}/{num_epochs} {(freeze, dropout)}: " +
          f" {total_loss:.1f}")

with torch.no_grad():    # evaluate test error
    test_pred = [model.predict(form_input([x_all[i]]).to(device))[0]
                 for i in test_index]
    test_gold = [y_all[idx] for idx in test_index]
    test_correct = (np.array(test_pred) == np.array(test_gold)).sum()
    train_pred = [model.predict(form_input([x_all[i]]).to(device))[0]
                  for i in train_index]
    train_gold = [y_all[idx] for idx in train_index]
    train_correct = (np.array(train_pred) == np.array(train_gold)).sum()
    accuracy[imodel][epoch] = {
        'loss': total_loss,
        'train': train_correct/len(train_gold),
        'test': test_correct/len(test_gold)}

if VERBOSE:
    print(freeze,
          dropout,
          epoch,
          int(time.time() - tic),
          optimizer.param_groups[0]['lr'],
          train_correct/len(train_gold),
          test_correct/len(test_gold))

```

0% | 0/50 [00:00<?, ?it/s]

100% | ██████████ | 50/50 [03:51<00:00, 4.62s/it]
100% | ██████████ | 50/50 [03:51<00:00, 4.64s/it]
100% | ██████████ | 50/50 [04:02<00:00, 4.85s/it]

28.2.3 Evaluation

Evaluation includes computing confusion matrices of prediction errors for both training and testing data.

```

classes = class_encoder.classes_
cf_train = DataFrame(confusion_matrix(train_gold, train_pred),
                     index=pd.MultiIndex.from_product([['Actual'], classes]),
                     columns=pd.MultiIndex.from_product([['Predicted'], classes]))
cf_test = DataFrame(confusion_matrix(test_gold, test_pred),
                     index=pd.MultiIndex.from_product([['Actual'], classes]),
                     columns=pd.MultiIndex.from_product([['Predicted'], classes]))

```

```

for num, (title, cf) in enumerate({'Training': cf_train,
                                         'Test': cf_test}.items()):
    fig, ax = plt.subplots(num=1+num, clear=True, figsize=(8, 6))
    sns.heatmap(cf, ax=ax, annot=False, fmt='d', cmap='viridis', robust=True,

```

(continues on next page)

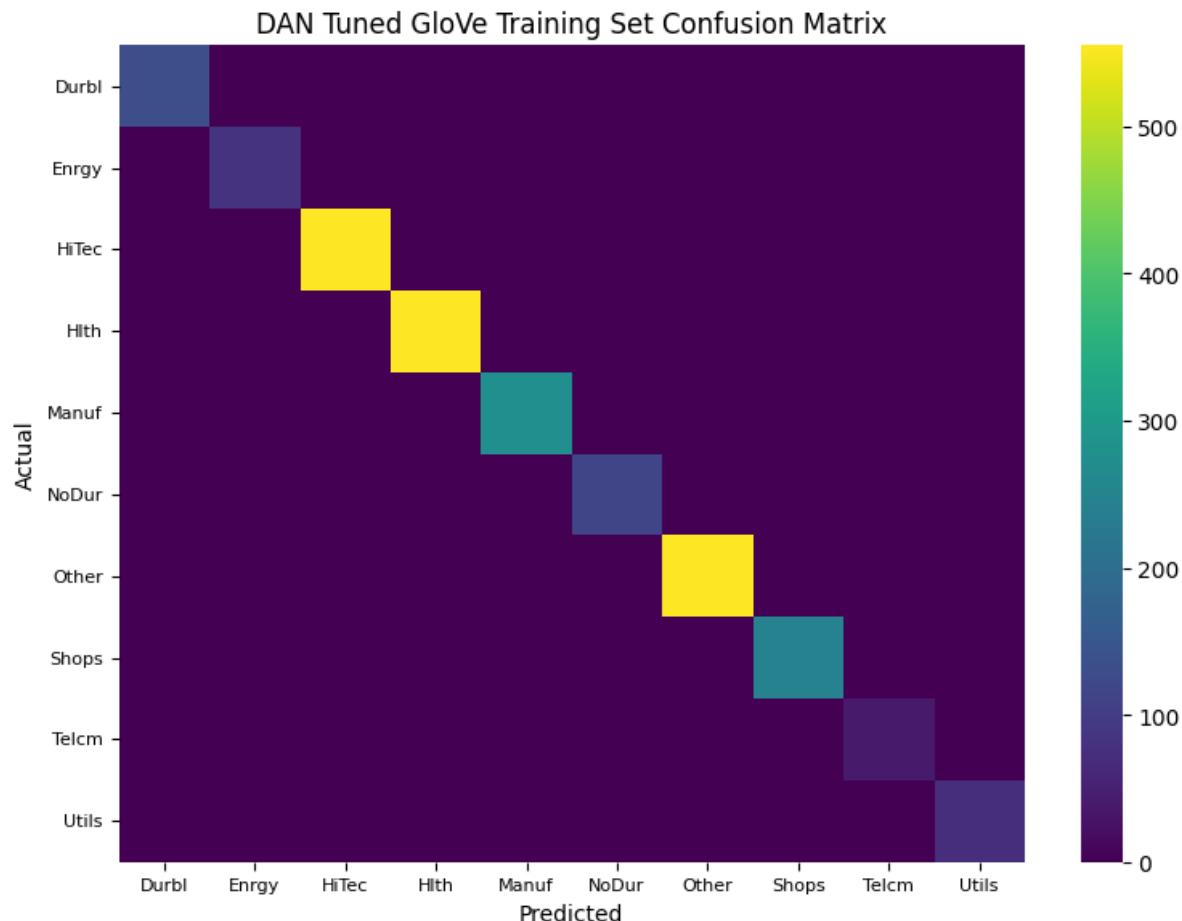
(continued from previous page)

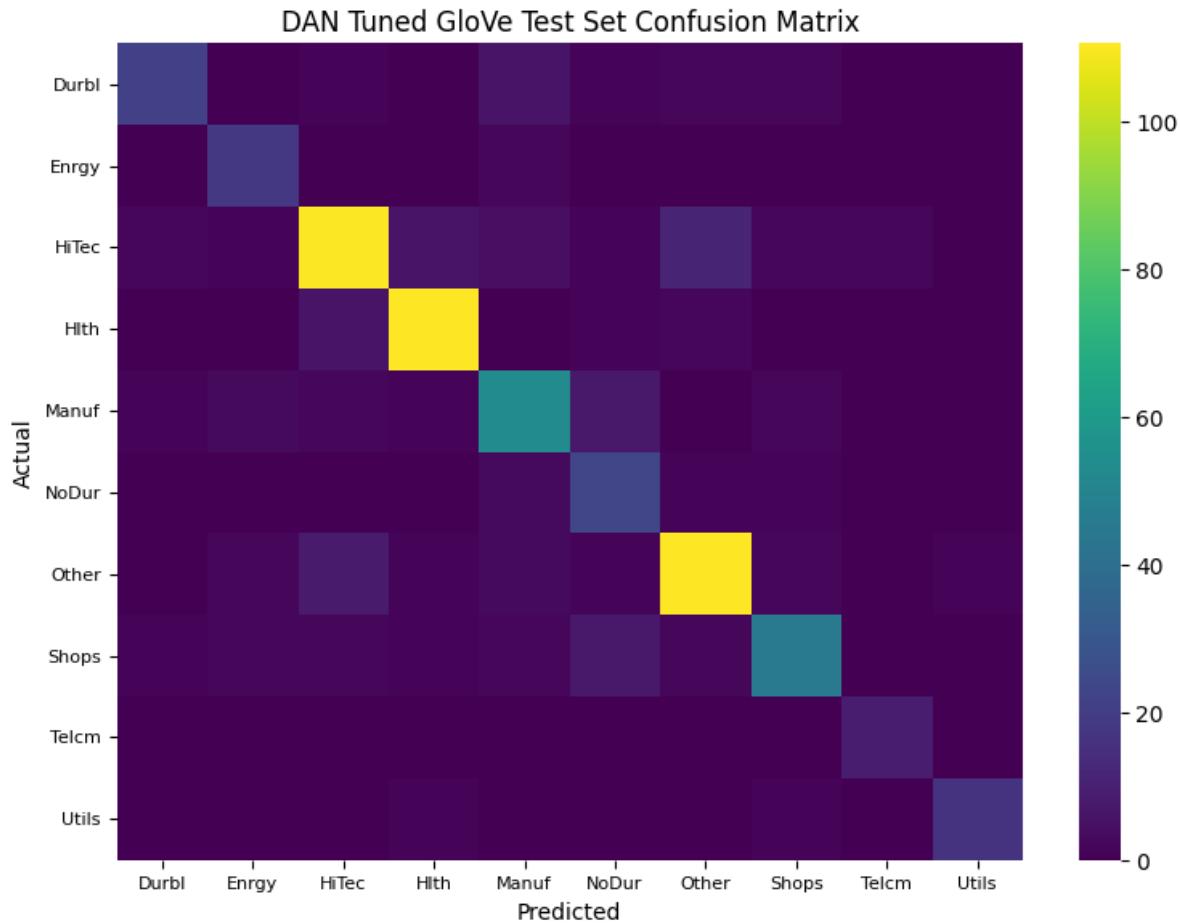
```

        yticklabels=class_encoder.classes_,
        xticklabels=class_encoder.classes_)

ax.set_title(f'DAN Tuned GloVe {title} Set Confusion Matrix')
ax.set_xlabel('Predicted')
ax.set_ylabel('Actual')
ax.yaxis.set_tick_params(labelsize=8, rotation=0)
ax.xaxis.set_tick_params(labelsize=8, rotation=0)
plt.subplots_adjust(left=0.35, bottom=0.25)
plt.tight_layout()

```

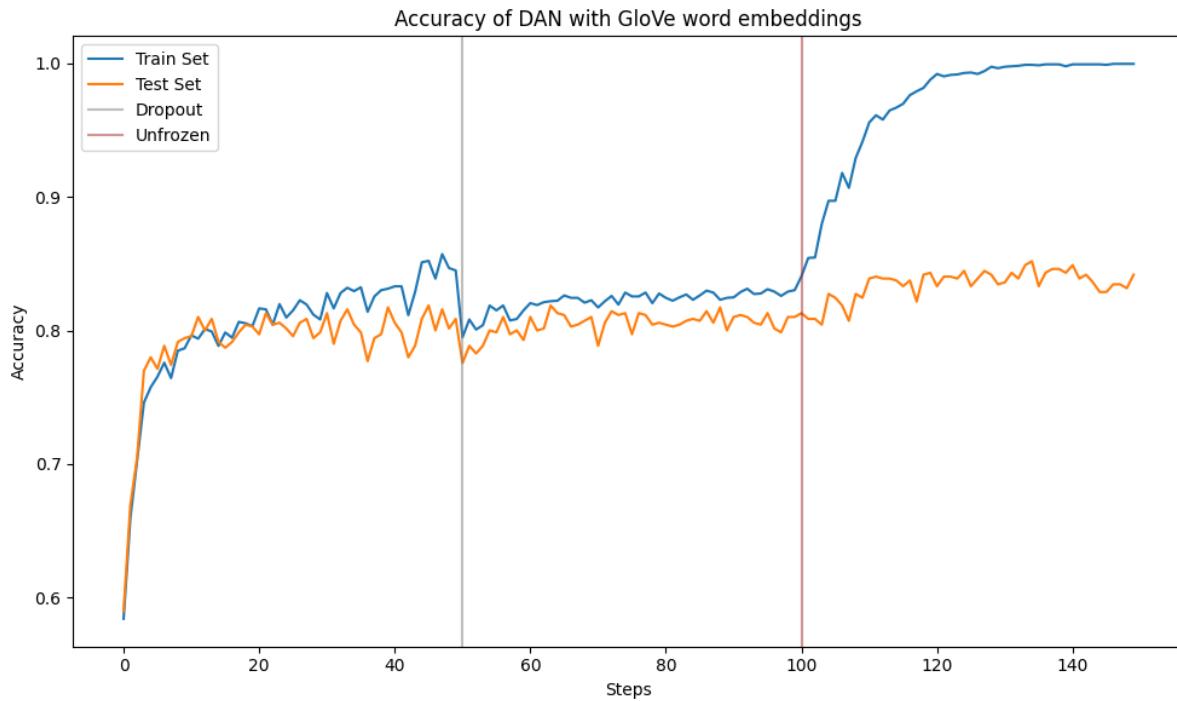




Initially, embeddings are frozen, then fine-tuned, with dropout introduced last, to highlight generalization improvements and the challenge of overfitting.

```
train_accuracy = pd.concat([Series([epoch['train']] for epoch in acc.values()),
                           for acc in accuracy],
                           ignore_index=True)
test_accuracy = pd.concat([Series([epoch['test']] for epoch in acc.values())
                           for acc in accuracy],
                           ignore_index=True)

fig, ax = plt.subplots(num=1, clear=True, figsize=(10, 6))
train_accuracy.plot(ax=ax)
test_accuracy.plot(ax=ax)
ax.axvline(len(accuracy[0]), c='grey', alpha=0.5)
ax.axvline(len(accuracy[0]) + len(accuracy[1]), c='brown', alpha=0.5)
ax.set_title(f'Accuracy of DAN with GloVe word embeddings')
ax.set_xlabel('Steps')
ax.set_ylabel('Accuracy')
ax.legend(['Train Set', 'Test Set', 'Dropout', 'Unfrozen'], loc='upper left')
plt.tight_layout()
```



When embeddings are frozen, the model overfits the training data, achieving 100% training accuracy. When dropout regularization is enabled, the test set accuracy slightly improves.

```
# Accuracy when frozen embeddings, unfrozen and with dropouts
p = (len(accuracy[0]) - 1, len(accuracy[0]) + len(accuracy[1]) - 1, -1)
print("Accuracy")
DataFrame({'frozen': [train_accuracy.iloc[p[0]], test_accuracy.iloc[p[0]]],
            'dropout': [train_accuracy.iloc[p[1]], test_accuracy.iloc[p[1]]],
            'unfrozen': [train_accuracy.iloc[p[2]], test_accuracy.iloc[p[2]]]},
            index=['train', 'test'])
```

Accuracy

	frozen	dropout	unfrozen
train	0.844908	0.830155	0.999640
test	0.808633	0.810072	0.841727

References:

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, Ruslan R. Salakhutdinov, July 2012, “Improving neural networks by preventing co-adaptation of feature detectors”

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation.

Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean, 2013, “Efficient Estimation of Word Representations in Vector Space”

Greg Durrett, 2021-2024, “CS388 Natural Language Processing course materials”, retrieved from <https://www.cs.utexas.edu/~gdurrett/courses/online-course/materials.html>

Philipp Krähenbühl, 2020-2024, “AI394T Deep Learning course materials”, retrieved from <https://www.philkr.net/dl-class/material> and <https://ut.philkr.net/deeplearning/>

Philipp Krähenbühl, 2025, “AI395T Advances in Deep Learning course materials”, retrieved from https://ut.philkr.net/advances_in_deeplearning/

CONVOLUTIONAL NEURAL NETWORKS

Life can only be understood backwards; but it must be lived forwards. - Søren Kierkegaard

Convolutional Neural Networks (CNNs) are particularly effective for analyzing structured data like images and sequences. By leveraging convolutional layers, CNNs extract hierarchical patterns from raw inputs for complex tasks such as image classification and time series prediction. We explore the application of **Temporal Convolutional Networks (TCNs)** for capturing dependencies in economic time series. The results from modeling multiple time series data such as CPI components are compared with classical models like Vector Autoregression (VAR).

```
# By: Terence Lim, 2020-2025 (terence-lim.github.io)
import numpy as np
import pandas as pd
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
import random
import torch
import torch.nn as nn
import torchinfo
from statsmodels.tsa.api import VAR
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from finds.structured import BusDay
from finds.readers import Alfred
from secret import credentials
# %matplotlib qt
VERBOSE = 0
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# train-test split date
split_date = '2021-12-01' # training period up to this date
```

29.1 Convolutions

Convolutional layers are memory-efficient neural network components designed to process spatially structured data, such as images. Unlike fully connected (linear) layers, which require vast numbers of parameters, convolutional layers use localized, shared filters called **ernels** to capture patterns in input data. Mathematically, a convolution operates as follows:

$$y_{i,j,k} = \sum_{l=1}^{C_1} \sum_{m=0}^{h-1} \sum_{n=0}^{w-1} x_{l,j+m,k+n} \cdot \omega_{i,l,m,n}$$

Here, ω is a small kernel (e.g., 3x3) that slides over the image, performing element-wise multiplications and summing the results.

29.1.1 Image Filters

Convolutions can be interpreted as applying image processing filters, such as:

- Box filter (averaging): $\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$
- Edge detection filter: $\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$

Thus, convolutional layers serve as feature extractors, enabling models to detect edges, textures, and patterns crucial for tasks like image classification and segmentation.

29.1.2 Receptive field and output size

Convolutions operate on local patches of an image, but by stacking multiple layers, they can expand their **receptive field**, allowing the network to capture broader context. Several parameters affect the output size and receptive field:

- Padding: Adds extra pixels around the input to control output size and preserve borders.
- Stride: Controls how far the filter moves across the input, affecting downsampling.
- Dilation: Inserts zeros between kernel elements, expanding the receptive field without increasing parameters.
- Transposed convolution (up-convolution): A learnable upsampling method used to increase output size, often employed in image segmentation models like **U-Net**.

29.1.3 Computer vision

CNNs process images based on :

- Recurring Patterns: They detect similar structures that appear across different images.
- Multi-Scale Patterns: They recognize features ranging from small edges to large object shapes.
- Local Invariance: They take advantage of the fact that neighboring pixels often have similar values.
- Semantic Grouping: They group together pixels that belong to the same object based on shared patterns.

They excel at computer vision tasks that involve recognizing structured patterns at multiple levels of abstraction. These tasks include:

- **Image Classification:** CNNs are highly effective at identifying what object is present in an image by detecting low-level patterns (like edges and textures) and gradually building up to high-level semantic features (like object categories).
- **Object Detection:** By capturing patterns at various scales and identifying object parts, CNNs can localize and label multiple objects within an image, even if they vary in size or position.
- **Semantic Segmentation:** CNNs perform well at assigning a class label to each pixel in an image by grouping together pixels that form the same object or region.

AlexNet (2012) was the first deep network to outperform non-deep vision systems, winning the ImageNet challenge competition and kicking off the Deep Learning revolution. Winning the 2015 competition, ResNet introduced shortcut “residual connections” for gradients in convolutional architectures. The U-Net (2016) model designed a symmetric hourglass-shaped architecture for semantic segmentation, combining down-sampling to capture context with up-sampling to produce higher output resolution. More recently, Vision Transformer (ViT) models have incorporated transformer encoders by dividing images into patches, treating each patch as a token.

29.2 Temporal convolutional networks (TCN)

Temporal Convolutional Networks (TCNs) are deep convolutional architectures designed for sequence data. They utilize causal and dilated convolutions along with residual connections to model long-range dependencies efficiently. Unlike CNNs for images, TCNs handle sequential data such as time series, text, and audio.

- Causal Convolutions: Ensure that each output at time t only depends on current and past inputs — preserving the temporal order:

$$y_t = \sum_{i=0}^{k-1} w_i x_{t-i}$$

- Dilated Convolutions: Introduce gaps between filter (kernel) elements to capture long-range dependencies without expanding filter size:

$$y_t = \sum_{i=0}^{k-1} w_i x_{t-d \cdot i}$$

- Residual Connections: Allow gradients to flow efficiently through deep networks, mitigating vanishing gradient issues:

$$\text{Output} = x + F(x)$$

Key hyperparameters of TCNs include:

- `kernel_size`: Size of convolutional filter.
- `dropout`: Regularization to prevent overfitting.
- `blocks`: Number of stacked convolutional layers.
- `dilation`: Grows exponentially (e.g., 1, 2, 4, ...).
- `activation`: Non-linear activation, typically ReLU.

```
class TCN(torch.nn.Module):
    class CausalConv1dBlock(torch.nn.Module):
        """Conv1d block with ReLU, skip, dropout, dilation and padding"""

        def __init__(self, in_channels, out_channels, kernel_size, dilation,
                     dropout):
            super().__init__()

            # print('kernel', kernel_size, 'dilation', dilation)
            self.network = torch.nn.Sequential(
                torch.nn.ConstantPad1d(((kernel_size-1)*dilation, 0), 0),
                torch.nn.Conv1d(in_channels, out_channels, kernel_size,
                               dilation=dilation),
                torch.nn.ReLU(),
                torch.nn.ConstantPad1d(((kernel_size-1)*dilation, 0), 0),
                torch.nn.Conv1d(out_channels, out_channels, kernel_size,
```

(continues on next page)

(continued from previous page)

```

        dilation=dilation),
        torch.nn.ReLU(),
        torch.nn.Dropout(dropout))
self.skip = lambda x: x
if in_channels != out_channels: # downsample for skip if necessary
    self.skip = torch.nn.Conv1d(in_channels, out_channels, 1)

def forward(self, x):
    return self.network(x) + self.skip(x) # with skip connection

def __init__(self, n_features, blocks, kernel_size, dropout):
    """TCN model by connecting multiple convolution layers"""
    super().__init__()
    in_channels = n_features
    L = []
    for dilation, hidden in enumerate(blocks):
        L.append(self.CausalConv1dBlock(in_channels=in_channels,
                                         out_channels=hidden,
                                         kernel_size=kernel_size,
                                         dilation=2**dilation,
                                         dropout=dropout))

        in_channels = hidden
    self.network = torch.nn.Sequential(*L) if L else lambda x: x
    if L:
        self.classifier = torch.nn.Conv1d(in_channels, n_features, 1)
    else:
        self.classifier = torch.nn.Sequential(
            torch.nn.ConstantPad1d((kernel_size-1, 0), 0),
            torch.nn.Conv1d(in_channels, n_features, kernel_size))

def forward(self, x):
    """input is (B, n_features, L)), linear expects (B, * n_features)"""
    return self.classifier(self.network(x))

def save(self, filename):
    """save model state to filename"""
    return torch.save(self.state_dict(), filename)

def load(self, filename):
    """load model name from filename"""
    self.load_state_dict(torch.load(filename, map_location='cpu'))
    return self

```

29.2.1 Data preparation

The dataset comprises economic time series of CPI components obtained from FRED (Federal Reserve Economic Data). The data, covering various CPI categories (e.g., food, housing, transportation), is log-transformed and differenced for stationarity and standardized using StandardScaler to mean 0 and variance 1.

```

alf = Alfred(api_key=credentials['fred']['api_key'], verbose=-1)
vspan = alf.date_spans('USREC') # recession periods for plots

```

```
# CPI for U.S. City Average: Monthly, Seasonally Adjusted
# https://fred.stlouisfed.org/release/tables?rid=10&eid=34483
# 'CUSR0000SEEA'
series_ids = ['CPIFABSL', 'CPIHOSSL', 'CPIAPPSL', 'CPITRNSL', 'CPIMEDSL', 'CPIOGSSL']
df = pd.concat([alf(s, log=1, diff=1) for s in series_ids], axis=1) \
    .dropna() \
    .sort_index()
df.index = BusDay.to_datetime(df.index)
df.index.freq = 'M'      # set index to datetime type and freq = 'M'
```

```
/tmp/ipykernel_1025513/4272565358.py:9: FutureWarning: 'M' is deprecated and will be removed in a future version, please use 'ME' instead.
df.index.freq = 'M'      # set index to datetime type and freq = 'M'
```

```
names = [s[s.find(':')+2:s.find(' in ')] for s in alf.header(series_ids)]
names
```

```
['Food and Beverages',
 'Housing',
 'Apparel',
 'Transportation',
 'Medical Care',
 'Other Goods and Services']
```

```
# Standardize the data data
scaler = StandardScaler().fit(df)
scaled_data = DataFrame(scaler.transform(df), columns=names, index=df.index)
scaled_data
```

	Food and Beverages	Housing	Apparel	Transportation	...
date					
1967-02-28	-1.520716	-1.129800	0.533990	0.266586	
1967-03-31	-0.805655	-1.129800	0.124553	-0.267488	
1967-04-30	-1.522780	-0.063784	0.529140	0.263339	
1967-05-31	-0.805655	-0.067262	0.122142	-0.003280	
1967-06-30	1.339539	-1.129800	0.524347	-0.267488	
...
2024-10-31	-0.252031	0.063171	-2.110286	-0.203592	
2024-11-30	-0.053379	-0.037266	-0.114479	0.027062	
2024-12-31	-0.180743	-0.209529	-0.015224	0.767819	
2025-01-31	0.090331	-0.037301	-3.137367	0.794814	
2025-02-28	-0.350673	0.133190	0.934107	-0.605692	
	Medical Care	Other Goods and Services			
date					
1967-02-28	-0.248585		-1.017195		
1967-03-31	-0.253160		-0.284868		
1967-04-30	0.988995		-1.017195		
1967-05-31	-0.266687		-0.286982		
1967-06-30	0.962232		-0.289083		
...		
2024-10-31	-0.692758		-0.172972		
2024-11-30	-0.702581		0.085335		

(continues on next page)

(continued from previous page)

```
2024-12-31      -1.058184      -1.113129
2025-01-31      -0.701262      -1.822241
2025-02-28      -0.563596      0.499572

[697 rows x 6 columns]
```

```
ntrain = sum(scaled_data.index < split_date)
M = scaled_data.shape[1]      # M is number of time series
```

29.2.2 Training

The TCN is trained to predict the next time step of the CPI components using past observations. Training involves splitting data into train and test sets, and using the Adam optimizer to minimize mean squared error (MSE) between predictions and actual values.

```
# Model training parameters
seq_len = 8          # length of each input sequence for TCN
batch_size = 16
step_size = 30         # learning rate scheduler step size
lr = 0.01            # initial learning rate
num_lr = 3
num_epochs = step_size * num_lr
results = {}          # to collect evaluate results
train_loss = {}
test_loss = {}
```

```
# Form input data from training set
n_features = scaled_data.shape[1]      # number of input planes
train_exs = [scaled_data.iloc[(i - seq_len):(i + 1)].values
            for i in range(seq_len, ntrain)]
```

```
# train_ex should have dimension (batch size, channels, sequence length+1)
train_ex = torch.tensor(scaled_data.values[:ntrain].T)[None,:,:].float().to(device)
```

First, a baseline model comprising just a single **1D Conv layer** is trained to predict next time step

```
model = torch.nn.Conv1d(n_features, n_features, kernel_size=1).to(device)
print(model)
print(torchinfo.summary(model))
modelname = "1D-Convolution"
train_loss[modelname] = []
test_loss[modelname] = []
optimizer = torch.optim.Adam(model.parameters())
loss_function = nn.MSELoss()
```

```
Conv1d(6, 6, kernel_size=(1,), stride=(1,))
=====
Layer (type:depth-idx)           Param #
=====
Convid                           42
```

(continues on next page)

(continued from previous page)

```
=====
Total params: 42
Trainable params: 42
Non-trainable params: 0
=====
```

```

for epoch in range(num_epochs):
    for _ in range(batch_size):
        total_loss = 0.0
        model.train()
        model.zero_grad()
        X = train_ex[:, :, :-1]
        Y = train_ex[:, :, 1:]
        output = model(X)
        loss = loss_function(output, Y) # calculated over all outputs
        total_loss += float(loss)
        loss.backward()
        optimizer.step()

    model.eval()
    train_loss[modelname].append(total_loss)

    X = torch.tensor(scaled_data.values.T)[None, :, :].float().to(device)
    pred = model(X).cpu().detach().numpy()[:, :, :].T
    test_loss[modelname] = mean_squared_error(scaled_data.values[ntrain:],
                                              pred[ntrain-1:-1])
    results[modelname] = {
        'Train Error': mean_squared_error(scaled_data.values[1:ntrain],
                                           pred[:ntrain-1]),
        'Test Error': mean_squared_error(scaled_data.values[ntrain:],
                                         pred[ntrain-1:-1])
    }
}
```

Save the fitted weights to compare with classical Vector Autoregression (VAR) models.

```
conv1d_weights = np.vstack([model.bias.cpu().detach().numpy(),
                           model.weight.cpu().detach().numpy()[:, :, 0].T])
```

Next, we train various TCN configurations, utilizing StepLR learning rate scheduler and shuffled batches, while varying:

- number of layers (blocks): 1, 2
- different kernel sizes: 1, 2
- dropout rates: 0, 0.5.

```
# [1,1,0], [1,2,0], [2,1,0], [2,2,0], [1,1,0.5], [1,2,0.5], [2,1,0.5], [2,2,0.5]
for block, kernel_size, dropout in [[1,1,0], [1,2,0], [2,1,0], [2,2,0], [2,2,0.3]]:
    modelname = f"TCN(b={block},k={kernel_size},d={dropout:.1f})"
    train_loss[modelname] = []
    test_loss[modelname] = []

    # Set model, optimizer, loss function and learning rate scheduler
    model = TCN(n_features=n_features,
                blocks=[n_features]*block,
                kernel_size=kernel_size,
```

(continues on next page)

(continued from previous page)

```

        dropout=dropout).to(device)
print()
print('*****', modelname, '*****')
print(model)
print(torchinfo.summary(model))

optimizer = torch.optim.Adam(model.parameters(), lr=lr)
scheduler = torch.optim.lr_scheduler.StepLR(
    optimizer, gamma=0.1, step_size=step_size
)
loss_function = nn.MSELoss()

# Run training loop over num_epochs with batch_size
num_epochs = step_size * num_lr
for epoch in range(num_epochs):

    # shuffle idxs into batches
    idxs = np.arange(len(train_exs))
    random.shuffle(idxs)
    batches = [idxs[i:min(len(idxs), i + batch_size)]
               for i in range(0, len(idxs), batch_size)]

    # train by batch
    total_loss = 0.0
    model.train()
    for batch in batches:
        # input has shape (batch_size, n_features, seq_len)
        # Creating a tensor from a list of numpy.ndarrays is extremely slow.
        nparray = np.array([[train_exs[idx][seq] for idx in batch]
                           for seq in range(seq_len+1)])
        train_ex = torch.tensor(nparray).permute(1, 2, 0).float().to(device)
        model.zero_grad()
        X = train_ex[:, :, :-1]
        Y = train_ex[:, :, 1:]
        output = model(X)
        loss = loss_function(output, Y) # calculated over all outputs
        total_loss += float(loss) / len(batches)
        loss.backward()
        optimizer.step()
        scheduler.step()

    model.eval()
    train_loss[modelname].append(total_loss)
    if VERBOSE and (epoch % (step_size//2)) == 0:
        print(epoch, num_epochs, optimizer.param_groups[0]['lr'], total_loss)

    # Compute MSE of one-period ahead forecast error in train and test sets
    X = torch.tensor(scaled_data.values.T)[None, :, :].float().to(device)
    pred = model(X).cpu().detach().numpy()[0, :, :].T
    test_loss[modelname].append(mean_squared_error(scaled_data.values[ntrain:], pred[ntrain-1:-1]))

results[modelname] = {
    'Train Error': mean_squared_error(scaled_data.values[1:ntrain],
                                       pred[:ntrain-1]),
    'Test Error': mean_squared_error(scaled_data.values[ntrain:], pred[ntrain-1:-1]))

```

(continues on next page)

(continued from previous page)

```

        pred[ntrain-1:-1])
    }
#print('Blocks:', block, 'Kernel size:', kernel_size, results[modelname])
#print(pd.concat(res, axis=1).T)

```

```

***** TCN(b=1,k=1,d=0.0) *****
TCN(
    (network): Sequential(
        (0): CausalConv1dBlock(
            (network): Sequential(
                (0): ConstantPad1d(padding=(0, 0), value=0)
                (1): Conv1d(6, 6, kernel_size=(1,), stride=(1,))
                (2): ReLU()
                (3): ConstantPad1d(padding=(0, 0), value=0)
                (4): Conv1d(6, 6, kernel_size=(1,), stride=(1,))
                (5): ReLU()
                (6): Dropout(p=0, inplace=False)
            )
        )
    )
    (classifier): Conv1d(6, 6, kernel_size=(1,), stride=(1,))
)
=====
Layer (type:depth-idx)           Param #
=====
TCN
|---Sequential: 1-1
|   |---CausalConv1dBlock: 2-1
|   |   |---Sequential: 3-1           84
|   |---Conv1d: 1-2                 42
=====
Total params: 126
Trainable params: 126
Non-trainable params: 0
=====

***** TCN(b=1,k=2,d=0.0) *****
TCN(
    (network): Sequential(
        (0): CausalConv1dBlock(
            (network): Sequential(
                (0): ConstantPad1d(padding=(1, 0), value=0)
                (1): Conv1d(6, 6, kernel_size=(2,), stride=(1,))
                (2): ReLU()
                (3): ConstantPad1d(padding=(1, 0), value=0)
                (4): Conv1d(6, 6, kernel_size=(2,), stride=(1,))
                (5): ReLU()
                (6): Dropout(p=0, inplace=False)
            )
        )
    )
    (classifier): Conv1d(6, 6, kernel_size=(1,), stride=(1,))
)
=====

Layer (type:depth-idx)           Param #
=====
```

(continues on next page)

(continued from previous page)

```

TCN
|---Sequential: 1-1
|   |---CausalConv1dBlock: 2-1
|   |   |---Sequential: 3-1      156
|   |---Conv1d: 1-2           42
=====
Total params: 198
Trainable params: 198
Non-trainable params: 0
=====

***** TCN(b=2, k=1, d=0.0) *****

TCN(
    (network): Sequential(
        (0): CausalConv1dBlock(
            (network): Sequential(
                (0): ConstantPad1d(padding=(0, 0), value=0)
                (1): Conv1d(6, 6, kernel_size=(1,), stride=(1,))
                (2): ReLU()
                (3): ConstantPad1d(padding=(0, 0), value=0)
                (4): Conv1d(6, 6, kernel_size=(1,), stride=(1,))
                (5): ReLU()
                (6): Dropout(p=0, inplace=False)
            )
        )
        (1): CausalConv1dBlock(
            (network): Sequential(
                (0): ConstantPad1d(padding=(0, 0), value=0)
                (1): Conv1d(6, 6, kernel_size=(1,), stride=(1,), dilation=(2,))
                (2): ReLU()
                (3): ConstantPad1d(padding=(0, 0), value=0)
                (4): Conv1d(6, 6, kernel_size=(1,), stride=(1,), dilation=(2,))
                (5): ReLU()
                (6): Dropout(p=0, inplace=False)
            )
        )
    )
    (classifier): Conv1d(6, 6, kernel_size=(1,), stride=(1,))
)
=====

Layer (type:depth-idx)          Param #
=====
TCN
|---Sequential: 1-1
|   |---CausalConv1dBlock: 2-1
|   |   |---Sequential: 3-1      84
|   |---CausalConv1dBlock: 2-2
|   |   |---Sequential: 3-2      84
|---Conv1d: 1-2                 42
=====
Total params: 210
Trainable params: 210
Non-trainable params: 0
=====

***** TCN(b=2, k=2, d=0.0) *****

```

(continues on next page)

(continued from previous page)

```

TCN(
    (network): Sequential(
        (0): CausalConv1dBlock(
            (network): Sequential(
                (0): ConstantPad1d(padding=(1, 0), value=0)
                (1): Conv1d(6, 6, kernel_size=(2,), stride=(1,))
                (2): ReLU()
                (3): ConstantPad1d(padding=(1, 0), value=0)
                (4): Conv1d(6, 6, kernel_size=(2,), stride=(1,))
                (5): ReLU()
                (6): Dropout (p=0, inplace=False)
            )
        )
        (1): CausalConv1dBlock(
            (network): Sequential(
                (0): ConstantPad1d(padding=(2, 0), value=0)
                (1): Conv1d(6, 6, kernel_size=(2,), stride=(1,), dilation=(2,))
                (2): ReLU()
                (3): ConstantPad1d(padding=(2, 0), value=0)
                (4): Conv1d(6, 6, kernel_size=(2,), stride=(1,), dilation=(2,))
                (5): ReLU()
                (6): Dropout (p=0, inplace=False)
            )
        )
    )
    (classifier): Conv1d(6, 6, kernel_size=(1,), stride=(1,))
)
=====
Layer (type:depth-idx)           Param #
=====
TCN
|---Sequential: 1-1           --
|   |---CausalConv1dBlock: 2-1   --
|   |   |---Sequential: 3-1     156
|   |   |---CausalConv1dBlock: 2-2   --
|   |   |---Sequential: 3-2     156
|---Conv1d: 1-2               42
=====
Total params: 354
Trainable params: 354
Non-trainable params: 0
=====
***** TCN(b=2,k=2,d=0.3) *****
TCN(
    (network): Sequential(
        (0): CausalConv1dBlock(
            (network): Sequential(
                (0): ConstantPad1d(padding=(1, 0), value=0)
                (1): Conv1d(6, 6, kernel_size=(2,), stride=(1,))
                (2): ReLU()
                (3): ConstantPad1d(padding=(1, 0), value=0)
                (4): Conv1d(6, 6, kernel_size=(2,), stride=(1,))
                (5): ReLU()
                (6): Dropout (p=0.3, inplace=False)
            )
        )
)

```

(continues on next page)

(continued from previous page)

```

        )
        (1): CausalConv1dBlock(
            (network): Sequential(
                (0): ConstantPad1d(padding=(2, 0), value=0)
                (1): Conv1d(6, 6, kernel_size=(2,), stride=(1,), dilation=(2,))
                (2): ReLU()
                (3): ConstantPad1d(padding=(2, 0), value=0)
                (4): Conv1d(6, 6, kernel_size=(2,), stride=(1,), dilation=(2,))
                (5): ReLU()
                (6): Dropout (p=0.3, inplace=False)
            )
        )
    )
    (classifier): Conv1d(6, 6, kernel_size=(1,), stride=(1,))
)
=====
Layer (type:depth-idx)           Param #
=====
TCN
├ Sequential: 1-1               --
|  └ CausalConv1dBlock: 2-1     --
|  |  └ Sequential: 3-1       156
|  |  └ CausalConv1dBlock: 2-2   --
|  |  └ Sequential: 3-2       156
└ Conv1d: 1-2                  42
=====
Total params: 354
Trainable params: 354
Non-trainable params: 0
=====

```

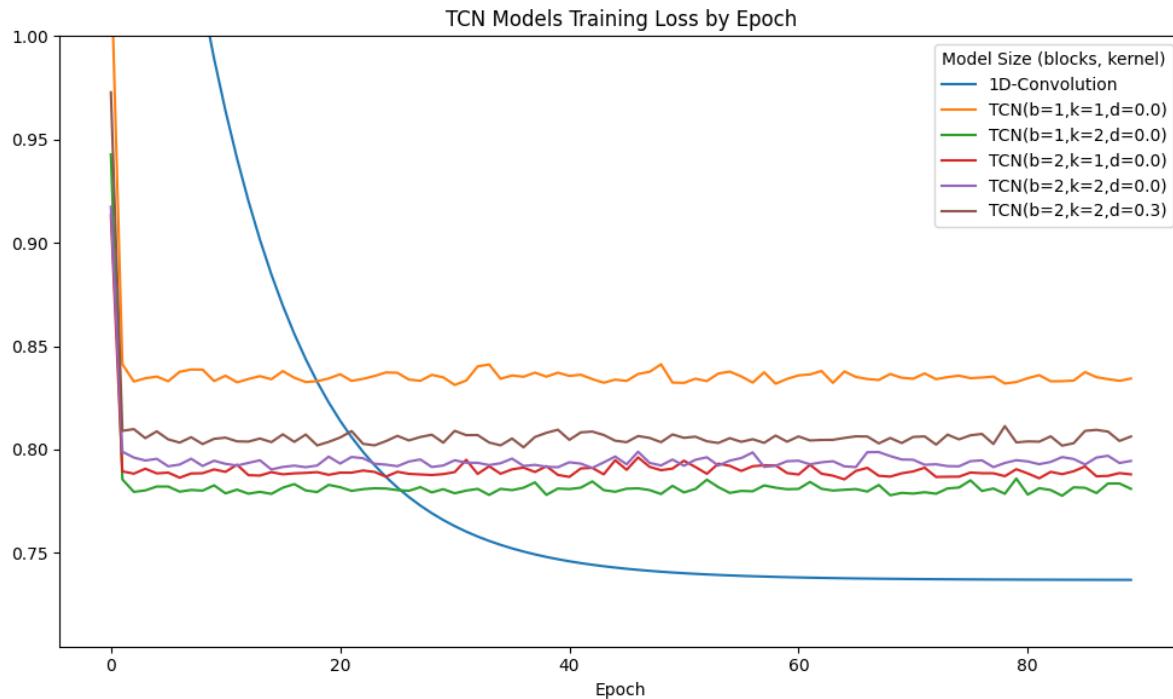
29.2.3 Evaluation

Training and test errors (MSE) from one-step-ahead forecasting are collected across models. Additionally, training and testing loss curves are plotted to analyze convergence and overfitting tendencies of different configurations.

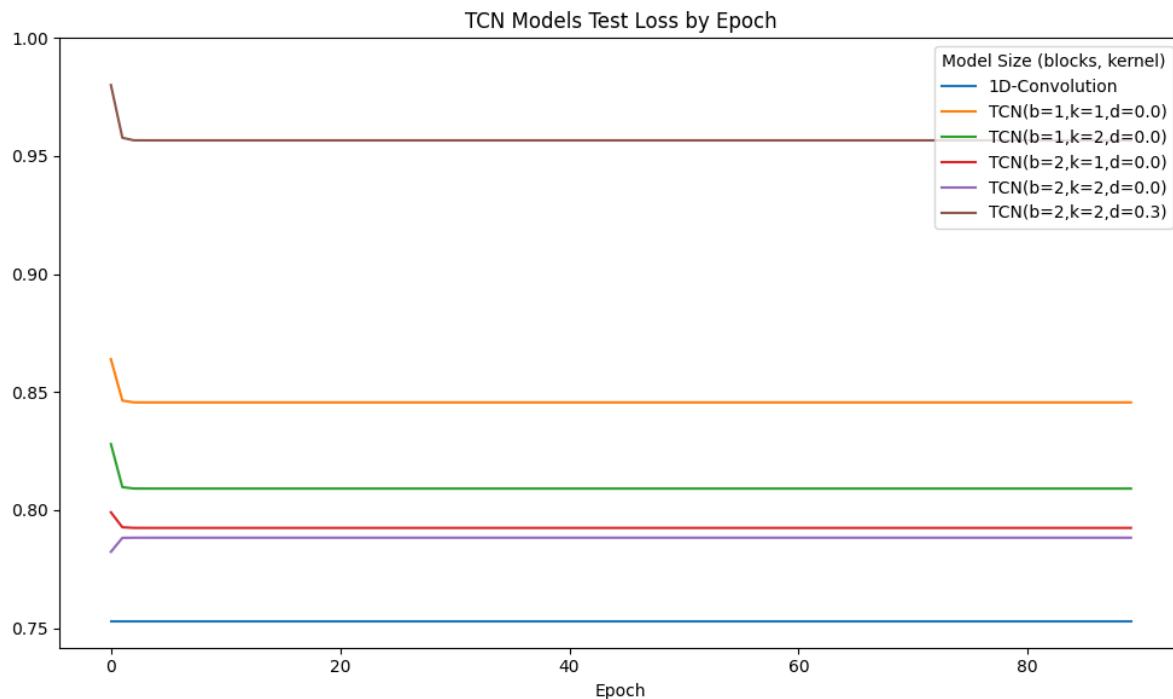
```

fig, ax = plt.subplots(figsize=(10, 6))
DataFrame(train_loss).plot(ax=ax)
ax.set_ylim(top=1.0)
ax.set_title(f"TCN Models Training Loss by Epoch")
ax.set_xlabel('Epoch')
ax.legend(title='Model Size (blocks, kernel)')
plt.tight_layout()

```



```
fig, ax = plt.subplots(figsize=(10, 6))
DataFrame(test_loss).plot(ax=ax)
ax.set_ylim(top=1.0)
ax.set_title(f"TCN Models Test Loss by Epoch")
ax.set_xlabel('Epoch')
ax.legend(title='Model Size (blocks, kernel)')
plt.tight_layout()
```



```
print('Sorted by Test Error')
DataFrame(results).T.sort_values('Test Error')
```

Sorted by Test Error

	Train Error	Test Error
1D-Convolution	0.736907	0.753075
TCN (b=2, k=2, d=0.0)	0.786185	0.788295
TCN (b=2, k=1, d=0.0)	0.788457	0.792443
TCN (b=1, k=2, d=0.0)	0.775419	0.809113
TCN (b=1, k=1, d=0.0)	0.832984	0.845615
TCN (b=2, k=2, d=0.3)	0.790285	0.956657

29.3 Vector Autoregression

Vector Autoregression (VAR) is a statistical time series model that captures linear interdependencies across multiple time series.

```
var_model = VAR(scaled_data.iloc[:ntrain], freq='ME')
```

29.3.1 Lag order

The lagged coefficients estimated from the Vector Autoregression help predict multi-step future outcomes. The optimal lag order (p) can be selected using information criteria such as AIC, BIC, HQIC, or FPE.

```
# up to max number p of VAR(p) lags
maxlags = 6

print("Optimal number of VAR(p) lags selected by various IC")
DataFrame({ic: var_model.fit(maxlags=maxlags, ic=ic).k_ar
           for ic in ['aic', 'fpe', 'hqic', 'bic']},
           index=['optimal p:'])\
           .rename_axis(columns='IC:')
```

Optimal number of VAR(p) lags selected by various IC

IC:	aic	fpe	hqic	bic
optimal p:	3	3	2	2

```
# Fit VAR(p) models
var_models = {p: var_model.fit(p) for p in range(1, maxlags+1)} # fit models
```

```
# Show model summary for VAR(1)
print(var_models[1].summary())
```

Summary of Regression Results

```
=====
Model:           VAR
Method:          OLS
Date:           Sat, 15, Mar, 2025
Time:           04:58:13
-----
No. of Equations:    6.00000  BIC:           -1.67401
Nobs:              657.000  HQIC:          -1.84967
Log likelihood:    -4907.30  FPE:            0.140733
AIC:              -1.96089  Det(Omega_mle):  0.132063
=====
```

Results for equation Food and Beverages

	coefficient	std. error	t-stat	
↳ prob				
const	-0.000685	0.035822	-0.019	
↳ 0.985				
L1.Food and Beverages	0.309512	0.037889	8.169	
↳ 0.000				
L1.Housing	0.212849	0.043404	4.904	
↳ 0.000				
L1.Apparel	-0.002978	0.038364	-0.078	
↳ 0.938				
L1.Transportation	-0.006074	0.038082	-0.159	
↳ 0.873				
L1.Medical Care	-0.007239	0.042771	-0.169	
↳ 0.866				
L1.Other Goods and Services	0.010009	0.037379	0.268	
↳ 0.789				

=====

Results for equation Housing

	coefficient	std. error	t-stat	
↳ prob				
const	-0.016126	0.027926	-0.577	
↳ 0.564				
L1.Food and Beverages	0.163358	0.029537	5.531	
↳ 0.000				
L1.Housing	0.494378	0.033836	14.611	
↳ 0.000				
L1.Apparel	0.038625	0.029907	1.291	
↳ 0.197				
L1.Transportation	0.074603	0.029687	2.513	
↳ 0.012				
L1.Medical Care	0.211794	0.033342	6.352	
↳ 0.000				
L1.Other Goods and Services	-0.022649	0.029139	-0.777	
↳ 0.437				

=====

Results for equation Apparel

(continues on next page)

(continued from previous page)

	prob	coefficient	std. error	t-stat	
<hr/>					
const		-0.004137	0.036418	-0.114	
↳ 0.910					
L1.Food and Beverages		0.032202	0.038519	0.836	
↳ 0.403					
L1.Housing		0.108030	0.044125	2.448	
↳ 0.014					
L1.Apparel		0.137027	0.039002	3.513	
↳ 0.000					
L1.Transportation		0.187727	0.038715	4.849	
↳ 0.000					
L1.Medical Care		0.116103	0.043482	2.670	
↳ 0.008					
L1.Other Goods and Services		0.007090	0.038000	0.187	
↳ 0.852					

Results for equation Transportation

	prob	coefficient	std. error	t-stat	
<hr/>					
const		0.001347	0.034599	0.039	
↳ 0.969					
L1.Food and Beverages		0.021051	0.036595	0.575	
↳ 0.565					
L1.Housing		0.070974	0.041921	1.693	
↳ 0.090					
L1.Apparel		0.032549	0.037054	0.878	
↳ 0.380					
L1.Transportation		0.425719	0.036782	11.574	
↳ 0.000					
L1.Medical Care		0.050082	0.041310	1.212	
↳ 0.225					
L1.Other Goods and Services		-0.035207	0.036103	-0.975	
↳ 0.329					

Results for equation Medical Care

	prob	coefficient	std. error	t-stat	
<hr/>					
const		0.033077	0.029060	1.138	
↳ 0.255					
L1.Food and Beverages		0.024634	0.030737	0.801	
↳ 0.423					
L1.Housing		0.232885	0.035211	6.614	
↳ 0.000					
L1.Apparel		0.044440	0.031122	1.428	

(continues on next page)

(continued from previous page)

↳ 0.153				
L1.Transportation	0.005782	0.030893	0.187	↳
↳ 0.852				
L1.Medical Care	0.431033	0.034697	12.423	↳
↳ 0.000				
L1.Other Goods and Services	0.114476	0.030323	3.775	↳
↳ 0.000				

Results for equation Other Goods and Services

↳ prob	coefficient	std. error	t-stat	↳
↳ -----				
const	-0.009490	0.037496	-0.253	↳
↳ 0.800				
L1.Food and Beverages	0.026291	0.039660	0.663	↳
↳ 0.507				
L1.Housing	0.076689	0.045432	1.688	↳
↳ 0.091				
L1.Apparel	0.094758	0.040157	2.360	↳
↳ 0.018				
L1.Transportation	0.018115	0.039861	0.454	↳
↳ 0.649				
L1.Medical Care	0.260564	0.044769	5.820	↳
↳ 0.000				
L1.Other Goods and Services	0.009675	0.039126	0.247	↳
↳ 0.805				

Correlation matrix of residuals

	Food and Beverages	Housing	Apparel	Transportation	↳
↳ Medical Care	0.014162	0.147358	0.085820	-0.000563	↳
Food and Beverages	1.000000	0.147358	0.085820	-0.000563	↳
↳ 0.014162	-0.010059				
Housing	0.147358	1.000000	0.093314	0.132546	↳
↳ 0.094535	0.019075				
Apparel	0.085820	0.093314	1.000000	0.126653	↳
↳ 0.037713	0.025531				
Transportation	-0.000563	0.132546	0.126653	1.000000	↳
↳ -0.054420	-0.012936				
Medical Care	0.014162	0.094535	0.037713	-0.054420	↳
↳ 1.000000	0.153701				
Other Goods and Services	-0.010059	0.019075	0.025531	-0.012936	↳
↳ 0.153701	1.000000				

29.3.2 Var(1) and Conv1d

The coefficients learned by VAR(1) models are compared to the learned weights of Conv1D layers, illustrating how classical linear models relate to convolution-based neural approaches in time series forecasting.

```
print('Coefficients of VAR(1) model')
DataFrame(np.vstack([var_models[1].intercept, var_models[1].coefs[0].T]),
          columns=var_models[1].names, index=var_models[1].exog_names).round(4)
```

Coefficients of VAR(1) model

	Food and Beverages	Housing	Apparel	\
const	-0.0007	-0.0161	-0.0041	
L1.Food and Beverages	0.3095	0.1634	0.0322	
L1.Housing	0.2128	0.4944	0.1080	
L1.Apparel	-0.0030	0.0386	0.1370	
L1.Transportation	-0.0061	0.0746	0.1877	
L1.Medical Care	-0.0072	0.2118	0.1161	
L1.Other Goods and Services	0.0100	-0.0226	0.0071	
	Transportation	Medical Care	\	
const	0.0013	0.0331		
L1.Food and Beverages	0.0211	0.0246		
L1.Housing	0.0710	0.2329		
L1.Apparel	0.0325	0.0444		
L1.Transportation	0.4257	0.0058		
L1.Medical Care	0.0501	0.4310		
L1.Other Goods and Services	-0.0352	0.1145		
	Other Goods and Services			
const	-0.0095			
L1.Food and Beverages	0.0263			
L1.Housing	0.0767			
L1.Apparel	0.0948			
L1.Transportation	0.0181			
L1.Medical Care	0.2606			
L1.Other Goods and Services	0.0097			

```
print('Tensor weights of Conv1D')
DataFrame(conv1d_weights, columns=names, index=['bias'] + names).round(4)
```

Tensor weights of Conv1D

	Food and Beverages	Housing	Apparel	\
bias	-0.0011	-0.0167	-0.0041	
Food and Beverages	0.3154	0.1677	0.0322	
Housing	0.1997	0.4808	0.1080	
Apparel	-0.0014	0.0396	0.1370	
Transportation	-0.0037	0.0773	0.1877	
Medical Care	-0.0018	0.2197	0.1160	
Other Goods and Services	0.0105	-0.0227	0.0072	
	Transportation	Medical Care	\	
bias	0.0014	0.0330		

(continues on next page)

(continued from previous page)

Food and Beverages	0.0210	0.0253
Housing	0.0713	0.2302
Apparel	0.0326	0.0447
Transportation	0.4256	0.0063
Medical Care	0.0493	0.4323
Other Goods and Services	-0.0349	0.1146
Other Goods and Services		
bias	-0.0086	
Food and Beverages	0.0233	
Housing	0.0861	
Apparel	0.0936	
Transportation	0.0166	
Medical Care	0.2514	
Other Goods and Services	0.0115	

29.3.3 Evaluation

The forecasting accuracy of VAR models, measured by train and test MSE, is compared across different lag orders.

```
# Calculate forecast errors for each observation and model
test_errors = {p: list() for p in range(maxlags+1)}
train_errors = {p: list() for p in range(maxlags+1)}

for i in range(maxlags, len(scaled_data)-1):
    data = scaled_data.iloc[i].values

    # test or train sample
    var_errors = train_errors if i < ntrain else test_errors

    # error of unconditional mean forecast
    var_errors[0].append(mean_squared_error(data, scaled_data.iloc[:ntrain].mean()))

    # accumulate to error of VAR(p) model forecasts
    for p in range(1, maxlags+1):
        pred = var_models[p].forecast(scaled_data.iloc[:i].values, 1)
        var_errors[p].append(mean_squared_error(data.reshape(1, -1), pred))
```

```
# Collect mean test and train set errors of all VAR(p) models
print('VAR models train and test set errors')
out = DataFrame({'Train Error': {f"VAR({p})": np.mean(errors)
                                 for p, errors in train_errors.items()},
                  'Test Error': {f"VAR({p})": np.mean(errors)
                                for p, errors in test_errors.items()}})

out
```

VAR models train and test set errors

	Train Error	Test Error
VAR(0)	1.009966	0.936672
VAR(1)	0.739496	0.757588

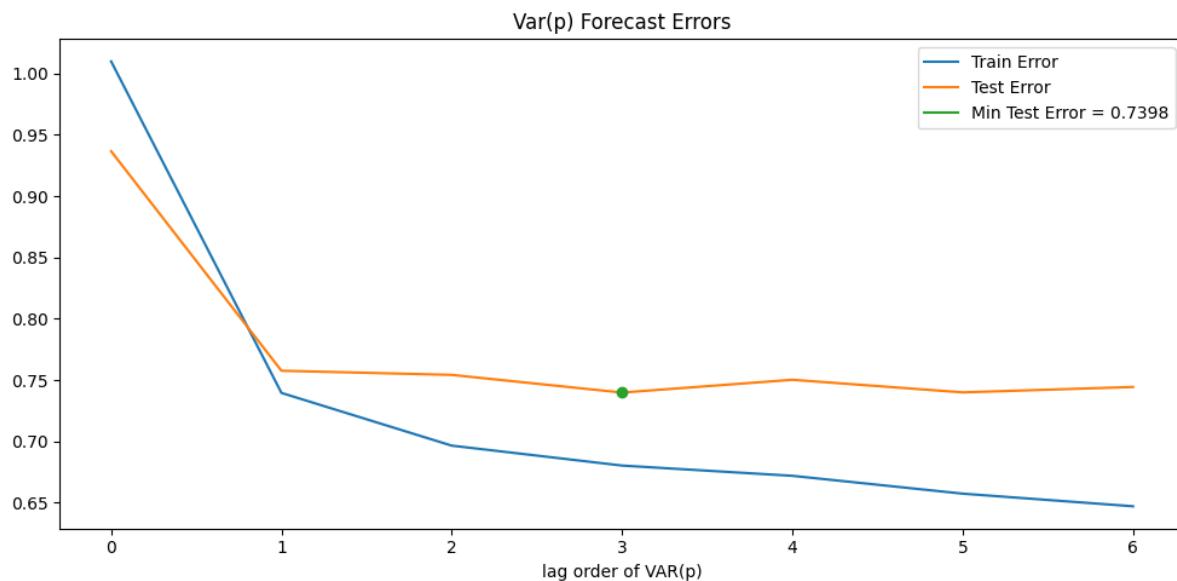
(continues on next page)

(continued from previous page)

VAR (2)	0.696483	0.754189
VAR (3)	0.680224	0.739767
VAR (4)	0.671862	0.750212
VAR (5)	0.657304	0.739927
VAR (6)	0.647010	0.744329

Error plots are generated to determine the optimal lag order for best predictive performance.

```
# Plot Errors
fig, ax = plt.subplots(1, 1, figsize=(10, 5))
ax.plot(np.arange(len(out)), out['Train Error'], color="C0")
ax.plot(np.arange(len(out)), out['Test Error'], color="C1")
ax.plot([], [], color="C2") # dummy for legend labels
argmin = out['Test Error'].argmin()
ax.plot(argmin, out.iloc[argmin]['Test Error'], 'o', color="C2")
ax.set_title(f'Var(p) Forecast Errors')
ax.set_xlabel('lag order of VAR(p)')
ax.legend(['Train Error', 'Test Error',
           f'Min Test Error = {out.iloc[argmin]["Test Error"]:.4f}'],
           loc='upper right')
plt.tight_layout()
```



References:

Philipp Krähenbühl, 2020-2024, “AI394T Deep Learning course materials”, retrieved from https://www.philkr.net/dl_class/material and <https://ut.philkr.net/deeplearning/>

Philipp Krähenbühl, 2025, “AI395T Advances in Deep Learning course materials”, retrieved from https://ut.philkr.net/advances_in_deeplearning/

RECURRENT NEURAL NETWORKS

History doesn't repeat itself, but it often rhymes - Mark Twain

We analyze the application of **Recurrent Neural Networks (RNNs)** for forecasting multivariate time series data, focusing on U.S. Consumer Price Index (CPI) components. RNN models are trained under different configurations to compare their predictive performance and investigate the behavior of their hidden states. Additionally, we examine how the temporal patterns learned by RNNs relate to the latent factors uncovered by Dynamic Factor Models (DFMs), a classical econometric approach for modeling co-movements in time series data.

```
# By: Terence Lim, 2020-2025 (terence-lim.github.io)
import numpy as np
import pandas as pd
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
import gc
import statsmodels.api as sm
import torch
import torch.nn as nn
import torchinfo
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from tqdm import tqdm
from finds.structured import BusDay
from finds.readers import Alfred
from secret import credentials
import warnings
```

```
# %matplotlib qt
VERBOSE = 0
if not VERBOSE: # Suppress FutureWarning messages
    warnings.simplefilter(action='ignore', category=FutureWarning)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# train-test split date
split_date = '2021-12-01'
```

30.1 Sequence modeling

Recurrent Neural Networks (RNNs) are powerful models for learning from sequential data, enabling tasks such as language modeling, translation, and time series forecasting by maintaining a memory of past information. Unlike feed-forward networks, RNNs process sequences one step at a time, using a hidden state that carries information across time steps to capture temporal dependencies. Variants like Elman and Jordan networks provide basic feedback mechanisms, while more advanced models such as LSTMs and GRUs address training challenges like vanishing gradients. However, RNNs remain inherently sequential, making them slower to train and less parallelizable compared to modern architectures like convolutional or transformer models.

30.1.1 Recurrent units

RNNs retain temporal information through recurrent connections, applying the same computation repeatedly at each time step while maintaining a **hidden state (memory)**. The hidden state h_t is updated based on the current input x_t and the previous hidden state h_{t-1} :

$$h_t = f_h(x_t, h_{t-1}, \theta_h)$$

$$h_t = f_h(x_t, h_{t-1}, \theta_h)$$

where the initial hidden state h_0 is typically initialized to zero, θ_h are learnable parameters, and f_h is the recurrent function (e.g., tanh, ReLU, GRU, LSTM).

Training RNNs is more complex than training feedforward neural networks due to dependencies across time steps.

RNNs are unfolded through time, applying the same parameters (shared weights) at each step, and trained using standard backpropagation, known as **Backpropagation Through Time (BPTT)** when applied over multiple time steps. This makes training computationally expensive, and prone to **vanishing gradients** (gradients shrink) and **exploding gradients** (gradients blow up), which respectively limit the model's ability to learn long-range dependencies and cause unstable updates.

30.1.2 Long Short-Term Memory (LSTM) networks

LSTMs mitigate the vanishing gradient problem by introducing a **cell state** and multiple **gating mechanisms** that regulate information flow. Key components include:

- Cell state c_t : A “memory” that runs through the network, modified by gates.
- Hidden state h_t : Output of the LSTM at each time step.
- Input x_t : Current input at time t .
- Previous state h_{t-1}, c_{t-1} : From the last time step.
- Gates:
 - Forget gate f_t : Decides what information to discard from the cell state.
 - Input gate i_t : Controls how much new information flows into the cell state.
 - Output gate o_t : Determines what part of the cell state should be output as h_t .

A cell update combines the past cell state and new input to update c_t . The h_t output is updated from c_t and previous its previous state h_{t-1} .

30.1.3 Gated Recurrent Unit (GRU) networks

GRUs are a simplified version of LSTMs that combine cell state and hidden state into a single vector, using fewer gates:

- Update gate (similar to combined forget/input gate): Controls what part of the past state to keep.
- Reset gate: Controls how to combine new input with the previous memory.
- Single hidden state h_t : Serves as both memory and output.

GRUs are computationally more efficient than LSTM, and perform comparably on many tasks.

30.2 Elman network

An **Elman network** is a basic RNN where recurrence occurs within the hidden layer:

$$h_t = f(x_t, h_{t-1})$$

Multiple Elman layers can be stacked to capture longer-term dependencies and more complex temporal patterns. For every element in the input sequence, each layer computes the following function:

$$h_t = \tanh(x_t W_{ih}^T + b_{ih} + h_{t-1} W_{hh}^T + b_{hh})$$

where

- h_t is the hidden state at time t ,
- x_t is the input at time t ,
- h_{t-1} is the hidden state of the previous layer at time $t - 1$ or the initial hidden state at time 0, and
- b 's and W 's are the learnable bias and weights

We implement a single-layer Elman RNN using PyTorch's standard RNN module to process sequential data, with Dropout regularization to reduce overfitting.

```
class Elman(nn.Module):
    def __init__(self, n_features, hidden_size, dropout, num_layers=1):
        super().__init__()
        self.hidden_size = hidden_size
        self.output_size = n_features
        self.num_layers = num_layers
        self.dropout = nn.Dropout(dropout)
        self.rnn = nn.RNN(input_size=n_features,
                          hidden_size=hidden_size,
                          num_layers=num_layers)
        self.o2o = nn.Linear(hidden_size, n_features)

    def forward(self, x, hidden):
        x = self.dropout(x)      # drop out input layer
        output, hidden = self.rnn(x, hidden)
        output = self.o2o(output[-1:, :])
        return output, hidden

    def init_hidden(self):
        return torch.zeros(self.num_layers, self.hidden_size)
```

30.2.1 Data preparation

CPI time series data for multiple components (e.g., food, housing) are collected from FRED. The data are log-transformed and differenced to ensure stationarity, then standardized (using `StandardScaler`) to have mean 0 and variance 1. The time series data are split into training followed by testing sets using a defined cutoff date.

```
# Max number of hidden states (RNN) or factors (Dynamic Model)
K = 2

# number of out-of-sample forecasts to predict
nforecast = 3

# Load time series from FRED
alf = Alfred(api_key=credentials['fred']['api_key'], verbose=VERBOSE)
vspan = alf.date_spans('USREC') # recession periods

# CPI for U.S. City Average: Monthly, Seasonally Adjusted
# https://fred.stlouisfed.org/release/tables?rid=10&eid=34483
# 'CUSR0000SEEA'
series_ids = ['CPIFABSL', 'CPIHOSSL', 'CPIAPPSL', 'CPITRNSL', 'CPIMEDSL', 'CPIOGSSL']
df = pd.concat([alf(s, log=1, diff=1) for s in series_ids], axis=1) \
    .dropna() \
    .sort_index()
df.index = BusDay.to_datetime(df.index)
df.index.freq = 'ME'      # set index to datetime type and freq = 'M'

names = [s[s.find(':')+2:s.find(' in ')] for s in alf.header(series_ids)]
names

['Food and Beverages',
 'Housing',
 'Apparel',
 'Transportation',
 'Medical Care',
 'Other Goods and Services']

# Standardize the data data
scaler = StandardScaler().fit(df)
scaled_data = DataFrame(scaler.transform(df), columns=names, index=df.index)
scaled_data
```

	Food and Beverages	Housing	Apparel	Transportation	\\
date					
1967-02-28	-1.520716	-1.129800	0.533990	0.266586	
1967-03-31	-0.805655	-1.129800	0.124553	-0.267488	
1967-04-30	-1.522780	-0.063784	0.529140	0.263339	
1967-05-31	-0.805655	-0.067262	0.122142	-0.003280	
1967-06-30	1.339539	-1.129800	0.524347	-0.267488	
...	
2024-10-31	-0.252031	0.063171	-2.110286	-0.203592	
2024-11-30	-0.053379	-0.037266	-0.114479	0.027062	
2024-12-31	-0.180743	-0.209529	-0.015224	0.767819	

(continues on next page)

(continued from previous page)

```

2025-01-31      0.090331 -0.037301 -3.137367      0.794814
2025-02-28      -0.350673  0.133190  0.934107     -0.605692

      Medical Care  Other Goods and Services
date
1967-02-28      -0.248585      -1.017195
1967-03-31      -0.253160      -0.284868
1967-04-30      0.988995      -1.017195
1967-05-31      -0.266687      -0.286982
1967-06-30      0.962232      -0.289083
...
      ...
2024-10-31      -0.692758      -0.172972
2024-11-30      -0.702581      0.085335
2024-12-31      -1.058184      -1.113129
2025-01-31      -0.701262      -1.822241
2025-02-28      -0.563596      0.499572

[697 rows x 6 columns]

```

```

# Create input data for RNN
ntrain = sum(scaled_data.index < split_date)
ntest = len(scaled_data.index) - ntrain - 1
n_features = scaled_data.shape[1]
data = scaled_data.values

```

30.2.2 Training

RNN models with different hidden sizes ($K = 1, 2$) are trained using an Adam optimizer and a StepLR learning rate scheduler, which reduces the learning rate during training for better convergence.

```

# Train model
num_layers = 1
dropout = 0.0
lr = 0.01          # starting learning rate
step_size = 100    # number of steps per learning rate
num_lr = 3         # number of learning rate periods
num_epochs = step_size * num_lr

train_loss = {}
hidden_states = {}
for hidden_size in range(1, K+1):
    torch.manual_seed(0)
    model = Elman(n_features=n_features,
                  hidden_size=hidden_size,
                  dropout=dropout,
                  num_layers=num_layers).to(device)
    print(model)
    torchinfo.summary(model)

    # Set optimizer and learning rate scheduler
    optimizer = torch.optim.Adam(model.parameters(), lr=lr)
    scheduler = torch.optim.lr_scheduler.StepLR(optimizer,
                                                step_size=step_size,

```

(continues on next page)

(continued from previous page)

```

gamma=0.1)

loss_function = nn.MSELoss()
train_loss[hidden_size] = []
for epoch in tqdm(range(num_epochs)):    # Run training loop per epoch
    model.train()
    model.zero_grad()
    hidden = model.init_hidden().to(device)
    loss = torch.FloatTensor([0]).to(device)
    for i in range(ntrain):
        x = torch.FloatTensor(data[[i], :]).to(device)
        y = torch.FloatTensor(data[[i+1], :]).to(device)
        output, hidden = model(x, hidden)
        l = loss_function(output, y)
        loss += l
    loss.backward()
    optimizer.step()
    scheduler.step()

    model.eval()
    train_loss[hidden_size].append(float(loss)/ntrain)
    #if VERBOSE:
    #    print(epoch, train_loss[hidden_size][-1], scheduler.get_last_lr())

# collect predictions and hidden states, and compute mse
with torch.no_grad():    # reduce memory consumption for eval
    loss_function = nn.MSELoss()
    hidden = model.init_hidden().to(device)
    hidden_states[hidden_size] = [hidden.cpu().numpy().flatten()]
    y_pred = [np.zeros(n_features)]
    for i in range(ntrain + ntest):
        x = torch.FloatTensor(data[[i], :]).to(device)
        y = torch.FloatTensor(data[[i+1], :]).to(device)
        output, hidden = model(x, hidden)
        hidden_states[hidden_size].append(hidden.cpu().numpy().flatten())
        y_pred.append(output.cpu().numpy().flatten())

    # k-step ahead forecast at end of period
    for i in range(nforecast):
        x = y
        y, hidden = model(x, hidden)
        y_pred.append(y.cpu().numpy().flatten())
    print(f"train MSE (hidden={hidden_size}):",
          mean_squared_error(data[1:ntrain+1, :], y_pred[1:ntrain+1]))
    print(f"test MSE (hidden={hidden_size}):",
          mean_squared_error(data[ntrain+1:ntrain+ntest+1, :],
                             y_pred[ntrain+1:ntrain+ntest+1]))

```

```

Elman(
    (dropout): Dropout(p=0.0, inplace=False)
    (rnn): RNN(6, 1)
    (o2o): Linear(in_features=1, out_features=6, bias=True)
)

```

100% |██████████| 300/300 [01:42<00:00, 2.93it/s]

```

train MSE (hidden=1): 0.8150888820868842
test MSE (hidden=1): 0.7845472493024133
Elman(
    (dropout): Dropout (p=0.0, inplace=False)
    (rnn): RNN(6, 2)
    (o2o): Linear(in_features=2, out_features=6, bias=True)
)

```

```
100%|██████████| 300/300 [01:38<00:00, 3.04it/s]
```

```

train MSE (hidden=2): 0.757587880335504
test MSE (hidden=2): 0.7211778265327311

```

30.2.3 Evaluation

The training and testing MSE for each RNN model are calculated for performance comparison. Training loss curves across epochs are plotted to assess convergence, and out-of-sample forecasts for 3 time steps ahead, after the end of the sample period, are visualized.

```

fig, ax = plt.subplots(figsize=(10, 6))
DataFrame(train_loss).plot(ax=ax)
ax.set_title(f"Elman Models Training Loss by Epoch")
ax.set_xlabel('Epoch')
ax.legend(title='Model Size')
plt.tight_layout()

```

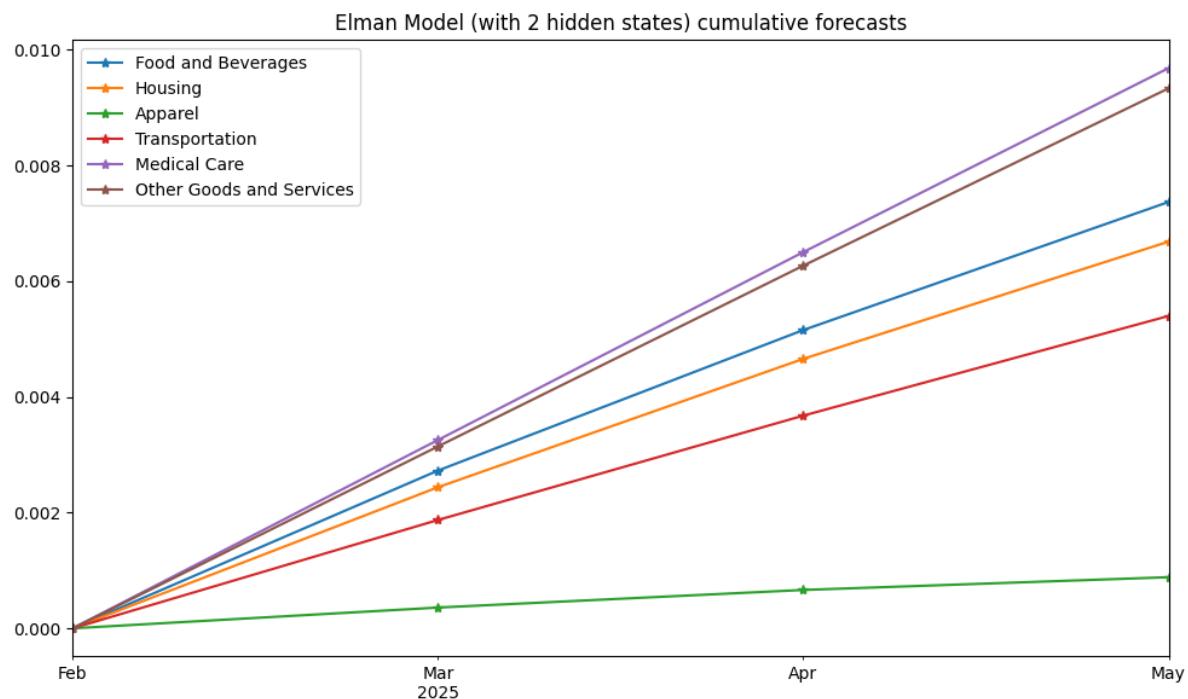


```
# Show forecasts of last model
pred = scaler.inverse_transform(y_pred)          # undo standardization
t = [scaled_data.index[-1] + pd.DateOffset(months=i) for i in range(nforecast + 1)]
forecasts = DataFrame(np.vstack((np.zeros(n_features), pred[-nforecast:])),
                      index=pd.PeriodIndex(t, freq='M'), columns=scaled_data.columns)
print("Monthly forecasts from RNN Model")
forecasts
```

Monthly forecasts from RNN Model

	Food and Beverages	Housing	Apparel	Transportation	Medical Care	\
2025-02	0.000000	0.000000	0.000000	0.000000	0.000000	
2025-03	0.002727	0.002439	0.000359	0.001873	0.003255	
2025-04	0.002429	0.002219	0.000304	0.001803	0.003247	
2025-05	0.002214	0.002026	0.000220	0.001723	0.003180	
	Other Goods and Services					
2025-02	0.000000					
2025-03	0.003144					
2025-04	0.003125					
2025-05	0.003065					

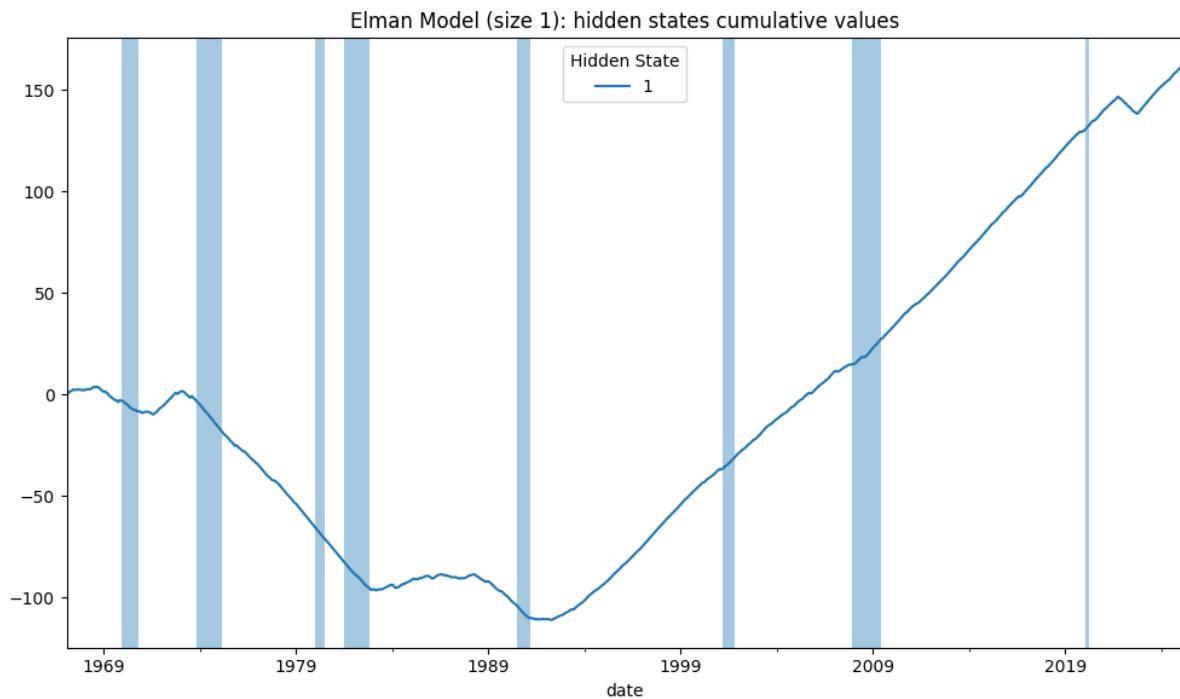
```
# Plot forecasts
fig, ax = plt.subplots(figsize=(10, 6))
forecasts.cumsum().plot(ax=ax, marker='*')
ax.set_title(f"Elman Model (with {K} hidden states) cumulative forecasts")
plt.tight_layout()
```

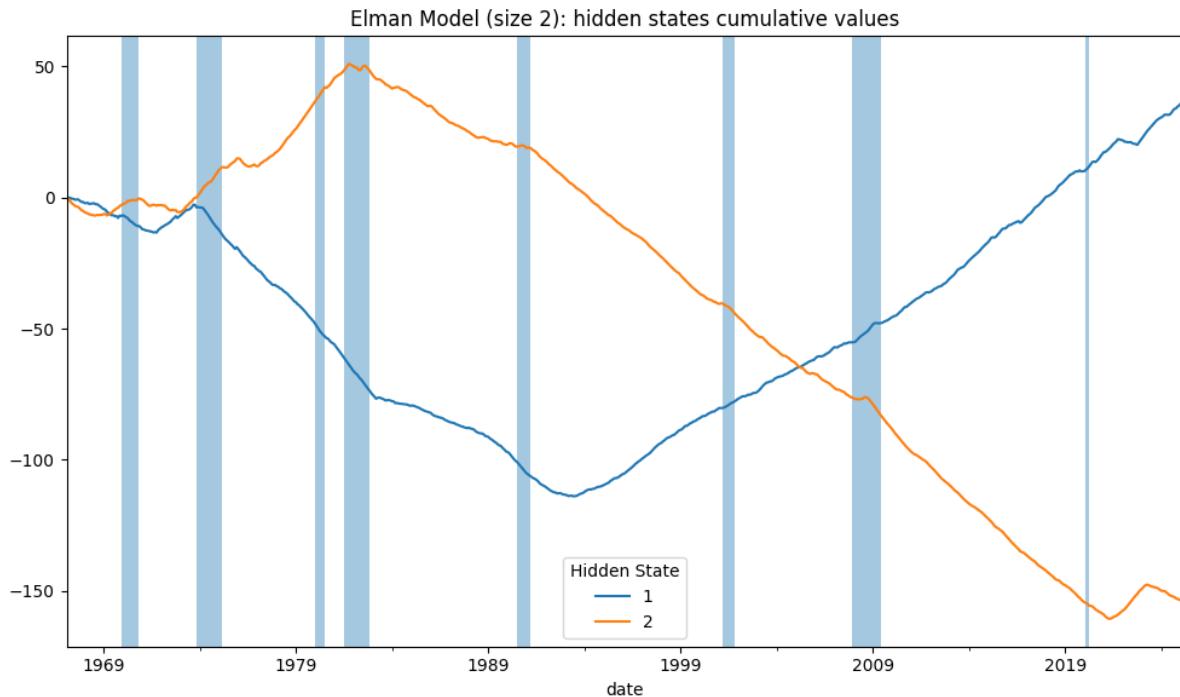


30.2.4 Hidden states

Hidden states from the last training epoch are collected, and their cumulative sums are plotted to illustrate how the internal memory of the Elman RNN evolves over time.

```
# Plot RNN hidden states values
for k, hidden in hidden_states.items():
    fig, ax = plt.subplots(figsize=(10, 6))
    hidden = DataFrame(np.array(hidden), index=scaled_data.index,
                       columns=[f"i+1" for i in range(k)])
    hidden.cumsum().plot(ax=ax, style='--')
    for a,b in vspans:
        if a >= min(hidden.index):
            ax.axvspan(a, min(b, max(hidden.index)), alpha=0.4)
    ax.legend(title='Hidden State')
    ax.set_title(f"Elman Model (size {k}): hidden states cumulative values")
    plt.tight_layout()
```





30.3 Dynamic Factor Models

This statistical model captures co-movements in time series using latent factors:

The basic model is:

$$y_t = \Lambda f_t + \epsilon_t$$

$$f_t = A_1 f_{t-1} + \dots + A_2 f_{t-2} + u_t$$

where:

- y_t is observed data at time t
- ϵ_t is idiosyncratic disturbance at time t
- f_t is the unobserved factor at time t
- $u_t \sim N(0, Q)$ is the factor disturbance at time t
- Λ is referred to as the matrix of factor loadings
- A_i are matrices of autoregression coefficients

We use `DynamicFactorMQ` from `statsmodels`, which employs an **Expectation-Maximization (EM)** algorithm for fitting, and so can accommodate a large number of observed variables. This can handle any collection of blocks of factors, including different factor autoregression orders, and AR(1) processes for idiosyncratic disturbances. The model allows incorporate monthly/quarterly mixed frequency data, making it suitable for **nowcasting**.

- https://www.statsmodels.org/dev/examples/notebooks/generated/statespace_dfm_coincident.html

We fit models with varying lag orders (p) to find an optimal autoregressive structure.

```

# Fit ar lags with best BIC
dynamic_factors = dict()
models = {}
K = 2
for ar in range(1, 5):
    mod = sm.tsa.DynamicFactorMQ(endog=scaled_data,
                                    factors=1,                      # num factor blocks
                                    factor_multiplicities=K,          # num factors in block
                                    factor_orders=ar,                # order of factor VAR
                                    idiosyncratic_ar1=True)
    fitted = mod.fit(disp=20 * bool(VERBOSE),
                      maxiter=1000,
                      full_output=True)
    models[ar] = dict(bic=fitted.bic,
                      mse=fitted.mse,
                      summary=fitted.summary().tables[0],
                      predict=fitted.predict(),
                      forecast=fitted.forecast(nforecast),
                      params=len(fitted.param_names))
    dynamic_factors[ar] = DataFrame(fitted.factors.filtered)
    dynamic_factors[ar].columns = np.arange(1, K+1)
    print(DataFrame(dict(bic=fitted.bic,
                          mse=fitted.mse,
                          parameters=len(fitted.param_names)),
                     index=[ar]))
del fitted
del mod
gc.collect()

```

	bic	mse	parameters
1	10478.491794	4.275815	31
	bic	mse	parameters
2	10499.374971	4.272672	35
	bic	mse	parameters
3	10506.462669	4.243688	39
	bic	mse	parameters
4	10517.472768	4.231597	43

30.3.1 Lag order

Bayesian Information Criterion (BIC) is used to select the optimal lag order

```

# dynamic model with best bic
best, model = min(models.items(), key=lambda item: item[1]['bic'])
mse = mean_squared_error(scaled_data, model['predict'])
print('Best lag:', best, ' bic:', model['bic'])
print(model['summary'])

```

```

Best lag: 1    bic: 10478.49179401509
                                         Dynamic Factor Results
-----
Dep. Variable:      "Food and Beverages", and 5 more    No. Observations: 697

```

(continues on next page)

(continued from previous page)

```

Model:                               Dynamic Factor Model      Log Likelihood
                                     ↵-5137.771
                                     + 2 factors in 1 blocks   AIC
                                     ↵10337.541
                                     + AR(1) idiosyncratic    BIC
                                     ↵10478.492
Date:                                Sat, 15 Mar 2025    HQIC
                                     ↵10392.038
Time:                                05:18:21      EM Iterations
                                     ↵     182
Sample:                               02-28-1967
                                     - 02-28-2025
Covariance Type:                    Not computed
=====
    
```

30.3.2 Evaluation

The models are evaluated based on **train/test MSE**. The 3-step-ahead forecasts, as of the end of the sample period, are also plotted.

```

# Show prediction errors
print('Dynamic Factor Model Train MSE:',
      mean_squared_error(scaled_data.iloc[1:ntrain+1],
                          model['predict'].iloc[1:ntrain+1]))
print('Dynamic Factor Model Test MSE:',
      mean_squared_error(scaled_data.iloc[ntrain+1:],
                          model['predict'].iloc[ntrain+1:]))
print('Number of parameters:', model['params'])
    
```

```

Dynamic Factor Model Train MSE: 0.7131933696697343
Dynamic Factor Model Test MSE: 0.7405157478690289
Number of parameters: 31
    
```

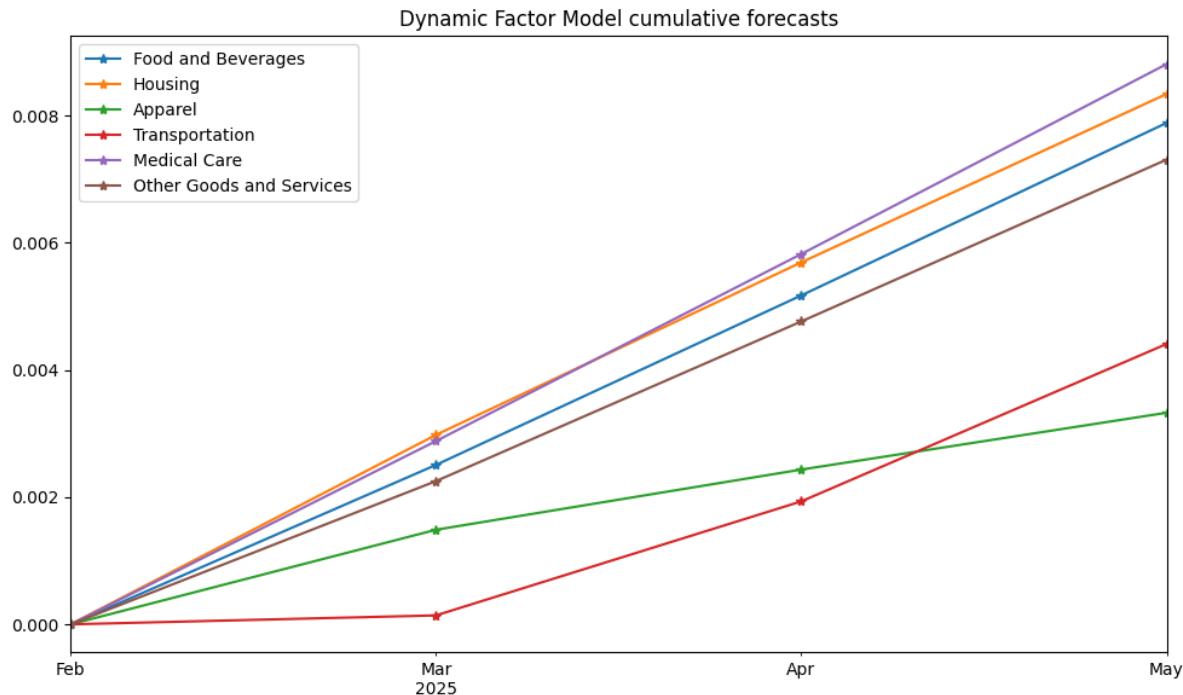
```

model_out = scaler.inverse_transform(model['forecast'].iloc[:nforecast])
model_out = DataFrame(np.vstack((np.zeros(n_features), model_out)),
                      index=pd.PeriodIndex(t, freq='M'), columns=scaled_data.columns)
print("Monthly forecasts from Dynamic Factor Model")
model_out
    
```

Monthly forecasts from Dynamic Factor Model

	Food and Beverages	Housing	Apparel	Transportation	Medical Care	\
2025-02	0.000000	0.000000	0.000000	0.000000	0.000000	
2025-03	0.002508	0.002980	0.001488	0.000141	0.002883	
2025-04	0.002666	0.002714	0.000947	0.001794	0.002942	
2025-05	0.002714	0.002649	0.000894	0.002477	0.002987	
	Other Goods and Services					
2025-02	0.000000					
2025-03	0.002253					
2025-04	0.002512					
2025-05	0.002546					

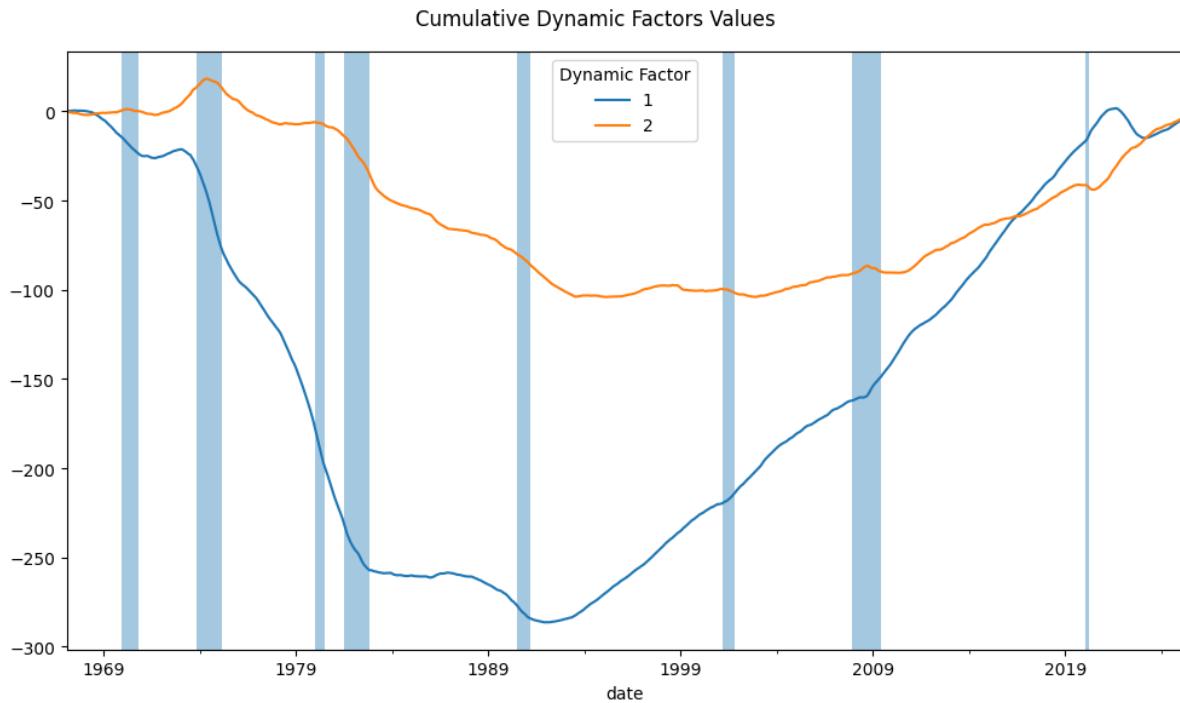
```
# Plot forecasts
fig, ax = plt.subplots(figsize=(10, 6))
model_out.cumsum().plot(ax=ax, marker='*')
ax.set_title(f"Dynamic Factor Model cumulative forecasts")
plt.tight_layout()
```



30.3.3 Dynamic factors

Dynamic factors represent unobserved common drivers of the time series, and their cumulative sums highlight underlying trends or cycles. In macroeconomic datasets, these latent factors often correspond to broad economic forces or business cycles. These cumulative factors are plotted to visualize their temporal patterns.

```
# Plot dynamic factors
dynamic_factor = dynamic_factors[best]
fig, ax = plt.subplots(figsize=(10, 6))
dynamic_factor.cumsum().plot(ax=ax, style='--')
ax.legend(title='Dynamic Factor')
for a,b in vspans:
    if a >= min(dynamic_factor.index):
        ax.axvspan(a, min(b, max(dynamic_factor.index)), alpha=0.4)
plt.suptitle(f"Cumulative Dynamic Factors Values")
plt.tight_layout()
```



The hidden states from the simple Elman RNN are compared to DFM latent factors, and with R-squared (R^2) statistics reported to measure the degree of overlap.

```
# RNN hidden state values explained by dynamic factors
rsq = dict()
for k, hidden_state in enumerate(np.array(hidden_states[2]).T):
    rsq[k+1] = sm.OLS(hidden_state, sm.add_constant(dynamic_factor).fillna(0))\
        .fit()
print('Proportion of variance of RNN hidden state values '
      'explained by dynamic factors:',
      np.mean([r.rsquared for r in rsq.values()]).round(4))
DataFrame({k: [r.rsquared, r.f_pvalue] for k, r in rsq.items()},
          index=['R-square', 'pvalue'])\
    .rename_axis(columns='Hidden State')\
    .round(4)
```

Proportion of variance of RNN hidden state values explained by dynamic factors: 0.7471

Hidden State	1	2
R-square	0.7494	0.7447
pvalue	0.0000	0.0000

References:

Philipp Krähenbühl, 2020-2024, “AI394T Deep Learning course materials”, retrieved from https://www.philkr.net/dl_class/material

REINFORCEMENT LEARNING

I have not failed. I've just found 10,000 ways that won't work - Thomas A. Edison

We combine financial modeling with reinforcement learning (RL) to evaluate static and adaptive retirement spending strategies. Traditional approaches, such as the **4% rule**, assume fixed withdrawal rates and asset allocations, that may not adapt well to changing market conditions. Leveraging historical economic data, simulations, and deep learning techniques, we apply RL to learn dynamic strategies which adjust asset allocations based on financial conditions, minimizing the risk of retirees outliving their savings.

```
# By: Terence Lim, 2020-2025 (terence-lim.github.io)
from pandas import DataFrame, Series
import pandas as pd
import numpy as np
import math
import random
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm
import numpy as np
import gymnasium as gym
from gymnasium import spaces
from stable_baselines3 import DQN
from typing import List, Tuple
from finds.database import SQL
from finds.structured import BusDay
from finds.utils import Store, subplots, set_xticks
import torch
from secret import credentials, paths
store = Store(paths['scratch'])
#pd.set_option('display.max_rows', None)
VERBOSE = 0
gym.logger.min_level = gym.logger.ERROR # Suppress warnings
```

```
# open connections
sql = SQL(**credentials['sql'], verbose=VERBOSE)
bd = BusDay(sql, verbose=VERBOSE)
outdir = paths['scratch'] / 'RL'
```

31.1 Retirement spending policy

A **retirement spending policy** guides how retirees withdraw funds from their savings to sustain their lifestyle while minimizing the risk of depleting assets. The key elements include the **withdrawal strategy**, which defines the initial withdrawal rate (e.g., the 4% rule) and its adjustments over time (e.g., inflation-linked or dynamic withdrawals); **asset allocation**, which balances stocks, bonds, and other investments to optimize growth and risk; **time horizon**, representing the expected duration of withdrawals, typically spanning 20-30 years or more; and **market conditions**, including interest rates, inflation, and asset returns, which influence portfolio sustainability. Effective policies seek to balance spending needs, longevity risk, and market fluctuations to minimize the risk of outliving their assets.

Benz, Ptak and Rekenthaler (2022) found that “For retirees who seek a fixed real withdrawal from their portfolio in retirement, a starting withdrawal rate of 3.8% is safe in Morningstar’s model over a 30-year time horizon, assuming a 90% success rate (defined here as a 90% likelihood of not running out of funds) and a balanced portfolio.”

31.1.1 SBBI data

The **Stocks, Bonds, Bills, and Inflation (SBBI)** dataset provides historical monthly, quarterly, and yearly total returns and yields for major U.S. asset classes, including large-cap stocks, small-cap stocks, corporate bonds, government bonds, and inflation. This data, which dates back to 1926, is commonly used for retirement portfolio simulations.

```
# Read sbbi data
sbbi_file = paths['data'] / 'SBBI/stocks-bonds-bills-and-inflation-data.xlsx'
df = pd.read_excel(sbbi_file,
                   sheet_name=0,
                   skiprows=list(range(9)) + [10],
                   header=0,
                   usecols='A,B,G,P',
                   index_col=0)
columns_official = df.columns.tolist()
df.columns = ['stocks', 'bonds', 'inflation']
df.index = bd.to_date(df.index)
df
```

	stocks	bonds	inflation
19260131	0.000000	0.013756	0.000000
19260228	-0.038462	0.006313	0.000000
19260331	-0.057471	0.004129	-0.005587
19260430	0.025305	0.007589	0.005618
19260531	0.017918	0.001412	-0.005587
...
20240831	0.024257	NaN	0.000814
20240930	0.021357	0.100107	0.001604
20241031	-0.009069	-0.046509	0.001151
20241130	0.058701	0.068325	-0.000542
20241231	-0.023838	-0.044901	0.000355

[1188 rows x 3 columns]

31.1.2 Scenario generator

To create realistic economic simulations, we extract different 30-year retirement periods from the 100-year span of historical data. These scenarios help model how varying economic conditions impact retirement outcomes.

```
# Scenario generator: episode, backtest a sample path
class Episodes:
    def __init__(self, data: DataFrame, T: int, num_loops: int = 1):
        self.data = np.log(1 + data)
        self.high = list(self.data.max() + self.data.std() * .1)
        self.low = list(self.data.min() - self.data.std() * .1)
        self.T = T                      # number of years per episode
        self.M = (T + 1) * 12            # number of monthly observations per episode
        self.num_loops = num_loops

    def __len__(self):
        return (len(self.data) - self.M + 1) * self.num_loops

    def __iter__(self):
        rows = []
        for i in range(self.num_loops):
            rows += np.random.permutation(len(self.data) - self.M + 1).tolist()
        for t in rows:
            df = self.data.iloc[t:(t + self.M), :].reset_index(drop=True)
            yield df.groupby(df.index // 12) \
                .sum() \
                .set_index(self.data.index[(t + 11):(t + self.M):12])
```

Summary statistics of all 30-year episodes:

```
T = 30
means = {s: [] for s in df.columns}
episodes = Episodes(df, T=T)
for episode in iter(episodes):
    for s in df.columns:
        means[s].append(episode[s].mean())
print('30-year sample periods:')
DataFrame({s: np.mean(x) for s, x in means.items()} | {'N': len(episodes)},
          index=['annualized mean'])
```

30-year sample periods:

	stocks	bonds	inflation	N
annualized mean	0.105185	0.05502	0.03604	817

We assess the effectiveness of a fixed annual withdrawal strategy by analyzing its performance under different conditions. Key inputs for such strategies include: portfolio asset allocation (e.g., stock/bond mix); market environment (e.g., past returns and inflation rates); and expected duration of withdrawals.

For example, a typical rule of a fixed 4% withdrawal rate (adjusted for inflation) for a 50% stock / 50% bond portfolio can be evaluated across rolling 30-year periods to estimate the likelihood that a retiree's savings will last the full duration.

The **Base** model is a simple **buy-and-hold** strategy where funds are invested in a fixed allocation at the start of retirement and are not rebalanced, even if allocation weights drift over time. Withdrawals remain fixed as a percentage of initial wealth, adjusted for inflation.

```

class BaseModel:
    """Buy-and-hold allocation model where assets weights drift from initial"""

    name = 'Buy-and-Hold'
    def __init__(self, T: int, W: List[float]):
        assert W[-1] > 0           # spend must be positive
        self.initial = dict(T=int(T), W=np.array(W))

    def reset(self, market_changes: List[float]):
        self.T = int(self.initial['T'])
        self.W = np.array(self.initial['W'])
        return list(market_changes)

    def step(self, action: List, market_changes: List) -> Tuple:
        assert self.T > 0
        self.W[:-1] = np.array(action).flatten() * self.W[:-1].sum() # rebalance
        self.W = self.W * np.exp(np.array(market_changes))           # price changes
        self.T = self.T - 1
        wealth = sum(self.W[:-1]) # remaining wealth
        if wealth < self.W[-1]:
            truncated = True           # is not enough for spend
            self.W[:-1] = 0.0
        else:
            truncated = False          # is sufficient for spend
            spend = self.W[-1] * self.W[:-1] / wealth # allocate and deduct spending
            self.W[:-1] = self.W[:-1] - spend
        terminated = not self.T or truncated
        return terminated, truncated

    def predict(self, obs: List) -> List:
        """Allow initial asset allocation to drift"""
        return self.W[:-1] / sum(self.W[:-1])

```

The Fixed model adds annual rebalancing, ensuring that the portfolio maintains a constant stock/bond allocation throughout retirement.

```

class FixedModel(BaseModel):
    """Allocation model where assets are rebalanced to fixed weights"""

    name = 'Annual-Rebalance'
    def predict(self, obs: List) -> List:
        """Action to rebalance asset allocation to fixed initial weight"""
        return self.initial['W'][:-1] / sum(self.initial['W'][:-1])

```

To assess the risk of retirees running out of money, we simulate rolling 30-year periods and track shortfalls. Our starting scenarios assume a 50/50 stock-bond allocation and a 4% inflation-adjusted withdrawal rule. The probability of shortfall measures the fraction of simulations where assets were depleted before reaching 30 years.

```

alloc = 50    # 50-50 stocks/bonds initial allocation
rule = 4.0    # 4 percent spending policy
model = BaseModel(T=T, W=[alloc, 100-alloc, rule])

```

```

result = {}
for n, episode in enumerate(iter(episodes)):
    obs = model.reset(episode.iloc[0])

```

(continues on next page)

(continued from previous page)

```

for year in episode.index[1:]:
    obs = episode.loc[year].to_list()
    action = model.predict(obs)
    terminated, truncated = model.step(action, obs)
    if truncated:
        break
    result[episode.index[0]] = model.T
prob = np.mean(np.array(list(result.values())) != 0)
print('Number of 30-year scenerios:', len(episodes))
print('Probability of shortfall: ', round(prob, 4))

```

```

Number of 30-year scenerios: 817
Probability of shortfall: 0.0747

```

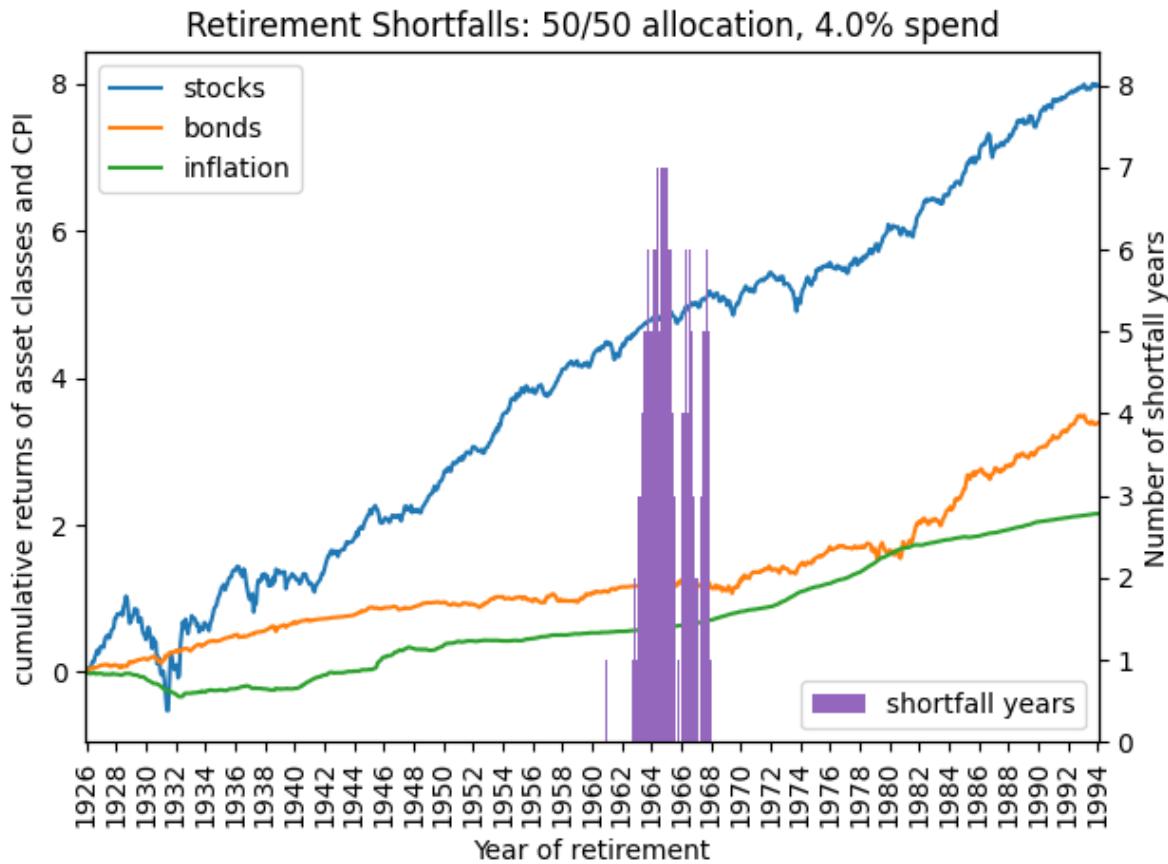
The graph illustrates the years in which retirees ran out of money, with bar heights representing the number of years their assets fell short of the goal. Additionally, plots of compounded asset returns and inflation highlight how periods of lower investment returns and higher inflation increased the risk of shortfall.

```

fails = Series(result).sort_index().rename('shortfall_years')
market = df.reindex(fails.index).cumsum()
fails.index = fails.index.astype(str).str.slice(0,4)
market.index = fails.index
ax = market.plot()
ax.set_title(f"Retirement Shortfalls: {alloc}/{100-alloc} allocation, {rule}% spend")
ax.set_ylabel('cumulative returns of asset classes and CPI')
ax.set_xlabel('Year of retirement')
ax.legend(loc='upper left')
##
bx = ax.twinx()
fails.plot(kind='bar', width=1.0, color='C4', ax=bx)
bx.set_ylabel('Number of shortfall years')
bx.legend(['shortfall years'], loc='lower right')
set_xticks(ax=ax, nskip=23, rotation=90)

plt.tight_layout()

```



31.1.3 Historical simulations

Next, we expanded the analysis to include a broader range of asset allocations (0% to 100% in stocks) and withdrawal rates (3% to 5%) to evaluate their impact on portfolio longevity.

```
# range of spending policy rules
rules = np.arange(3, 5.1, 0.1)

# simulate fixed and initial equity allocations from 0 to 100%
alocs = np.arange(0, 105, 5)
```

```
TAIL = 0.95
tail = int(100 * (1 - TAIL))
def compute_shortfall(x, q):
    """Compute average number of years of shortfall given tail probability level, q"""
    x = sorted(x)  # each simulation's results (years of shortfall)
    q = int(q * len(x))
    return x[q], np.mean(x[q:])
```

```
for num, Model in enumerate([BaseModel, FixedModel]):
    fail = DataFrame(columns=rules, index=alocs, dtype=float)
    shortfall = DataFrame(columns=rules, index=alocs, dtype=float)
    quantile = DataFrame(columns=rules, index=alocs, dtype=float)
```

(continues on next page)

(continued from previous page)

```

for alloc in tqdm(allocs):
    for rule in rules:

        # Evaluate for this allocation strategy and spending policy
        model = Model(T=T, W=[alloc, 100-alloc, rule])
        result = []
        for n, episode in enumerate(iter(episodes)): # for every 30-year sample
            obs = model.reset(episode.iloc[0])
            for year in episode.index[1:]:
                obs = episode.loc[year].to_list()
                action = model.predict(obs)
                terminated, truncated = model.step(action, obs)
                if truncated:
                    break
            result.append(model.T)
        fail.loc[alloc, rule] = np.mean(np.array(result) != 0)
        quantile.loc[alloc, rule], shortfall.loc[alloc, rule] = compute_
        shortfall(result, TAIL)
        store[model.name] = dict(fail=fail, shortfall=shortfall)
    
```

100%|██████████| 21/21 [07:33<00:00, 21.57s/it]
100%|██████████| 21/21 [07:32<00:00, 21.57s/it]

We compare the proportion of scenarios where the Base buy-and-hold strategy performed better or worse than the Fixed strategy with annual rebalancing.

```

#for model in [BaseModel, FixedModel]:
#    fail = store[model.name]['fail']
#    print(f"Probability of Shortfall: with {model.name} allocation")
#    print(fail.iloc[::-1, :].round(2).to_string())
print('Buy-and-hold outperformed Annual-Rebalance:',
      round(np.mean(store[FixedModel.name]['fail'] > store[BaseModel.name]['fail']),_
            3))
print('Annual-Rebalance outperformed Buy-and-Hold:',
      round(np.mean(store[FixedModel.name]['fail'] < store[BaseModel.name]['fail']),_
            3))
    
```

Buy-and-hold outperformed Annual-Rebalance: 0.374
Annual-Rebalance outperformed Buy-and-Hold: 0.297

31.1.4 Risk of spending shortfall

To measure the likelihood and severity of depleting funds before the end of retirement, we use two key metrics:

- **Probability of shortfall:** The percentage of simulations where retirees outlived their assets.
- **Expected shortfall period:** This metric quantifies the severity of shortfalls by estimating how many years retirees would be without funds in the worst-case scenarios (e.g., the worst 5% of simulations).

```

for model in [BaseModel, FixedModel]:
    fail = store[model.name]['fail'].iloc[::-1, :]
    plt.figure(figsize=(8, 6))
    
```

(continues on next page)

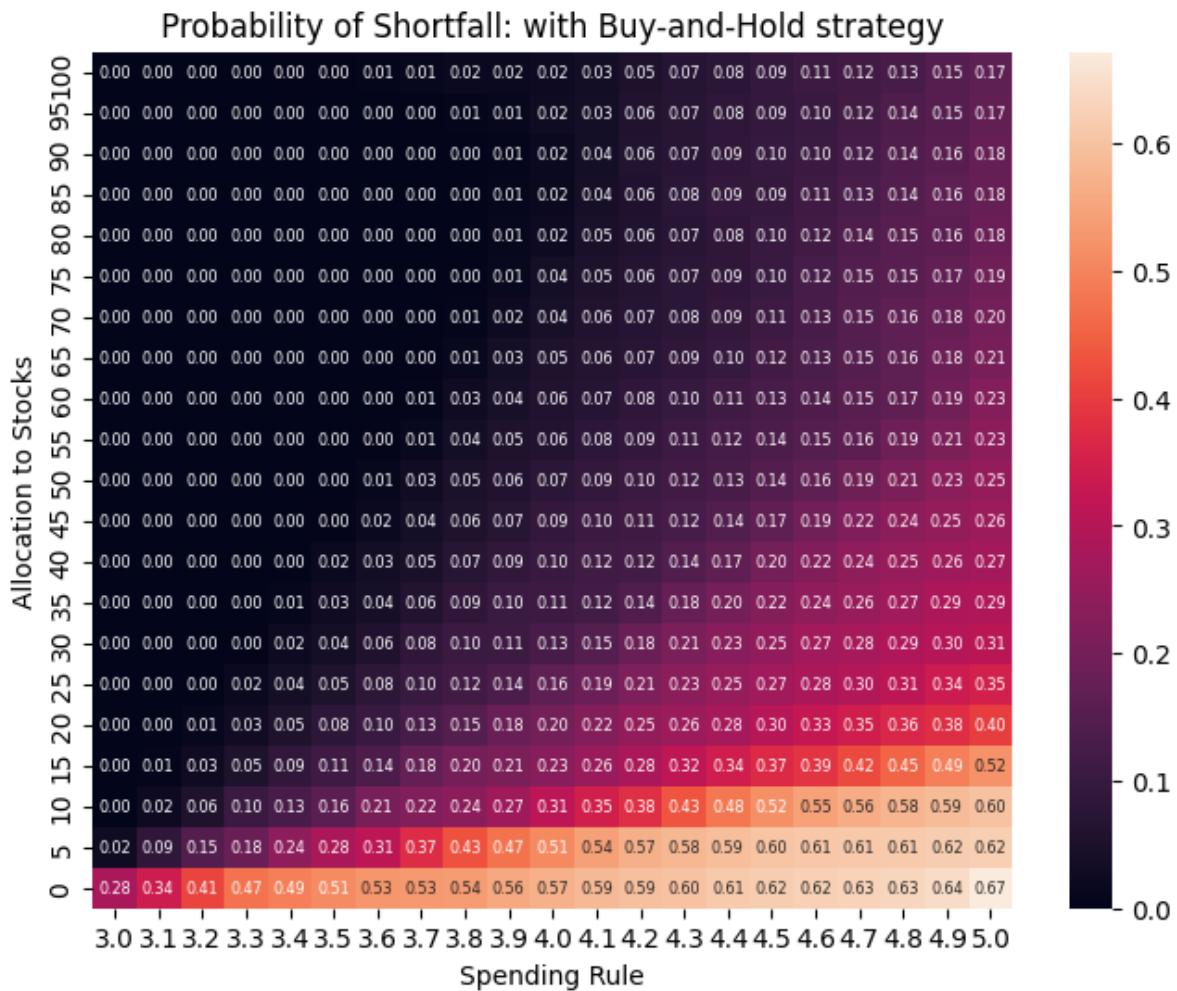
(continued from previous page)

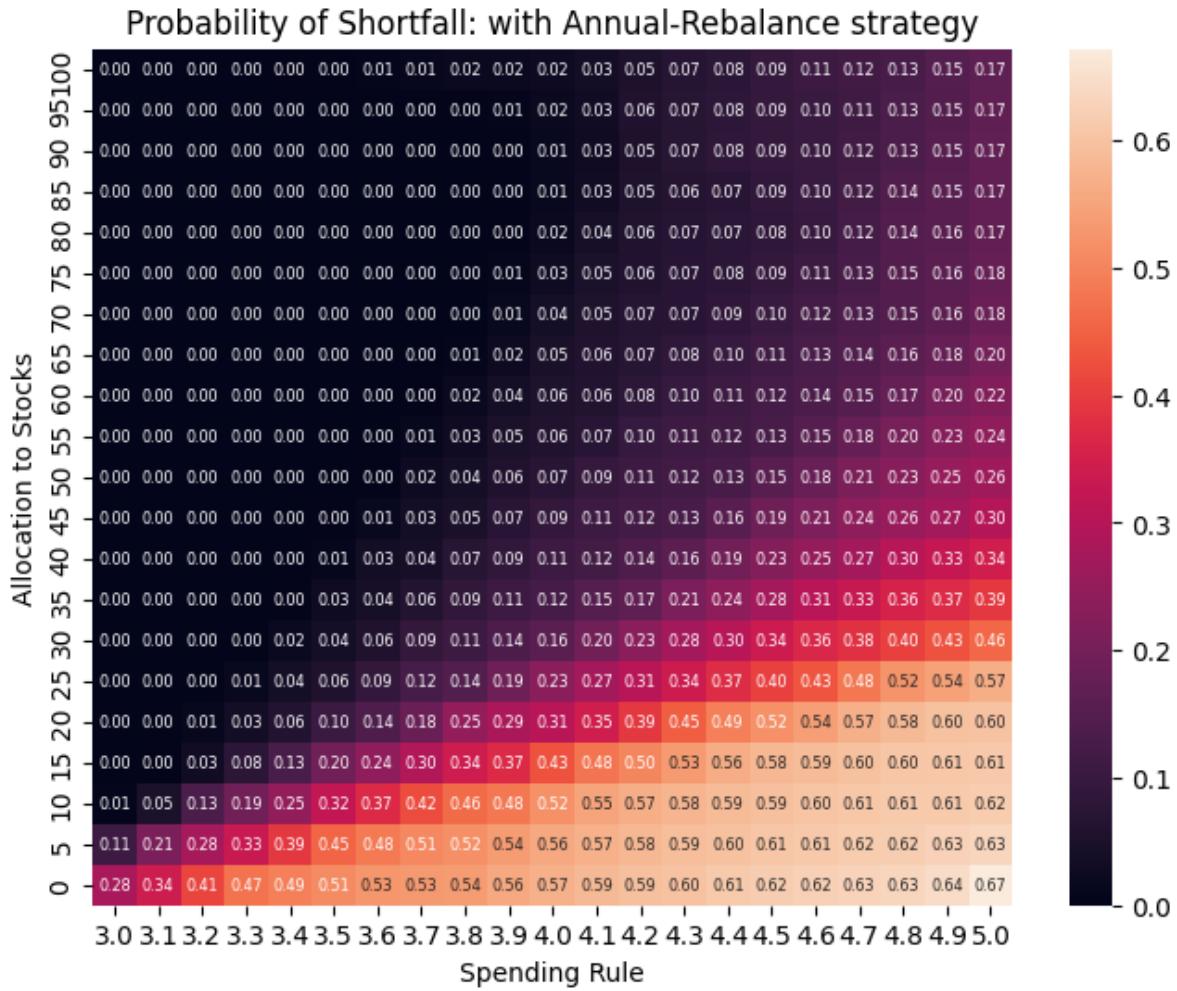
```

sns.heatmap(fail, annot=True, cbar=True, fmt='%.2f', annot_kws=dict(fontsize='xx-
➥small'),
            xticklabels=np.round(fail.columns, 1), yticklabels=fail.index)

# Labels and title
plt.xlabel("Spending Rule")
plt.ylabel("Allocation to Stocks")
plt.title(f"Probability of Shortfall: with {model.name} strategy")
plt.show()

```





Probability of shortfall:

Contour lines in the probability of shortfall graph show the combinations of asset allocation and withdrawal rates that result in the same likelihood of running out of money.

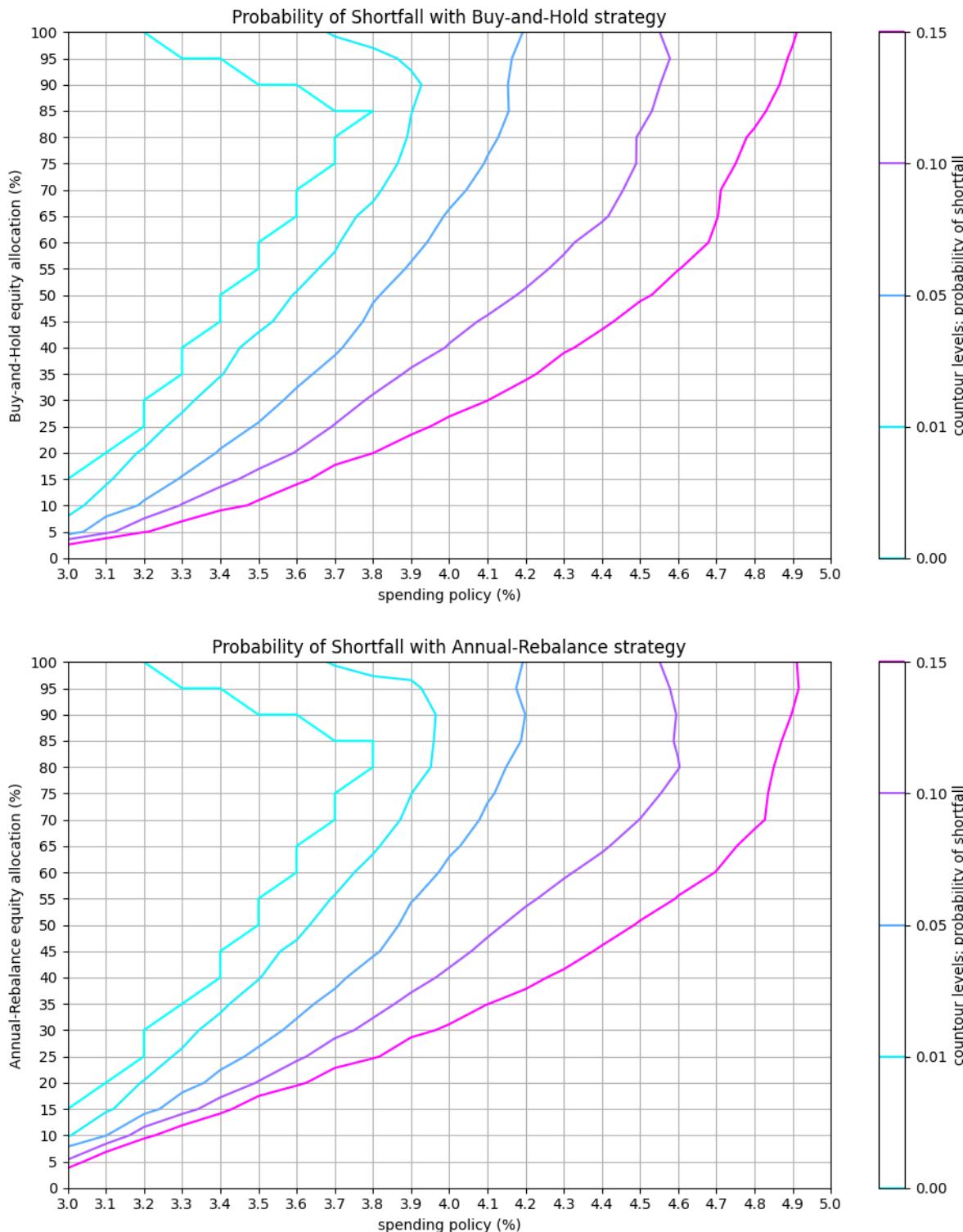
```
def plot_contour(Z, levels, title, label):
    """Helper to plot contour lines at given levels"""
    X, Y = np.meshgrid(rules, allocs)
    fig, ax = plt.subplots(figsize=(10, 6))
    cp = ax.contour(X, Y, Z, levels=levels, cmap='cool')
    ax.set_title(f"{title} with {model.name} strategy")
    ax.set_xticks(rules)
    ax.set_xlabel('spending policy (%)')
    ax.set_yticks(allocs)
    ax.set_ylabel(f'{model.name} equity allocation (%)')
    ax.grid(which='both')
    fig.colorbar(cp, label=label)
    plt.tight_layout()
```

```
for model in [BaseModel, FixedModel]:
    fail = store[model.name]['fail']
    plot_contour(fail, levels=[0.0, .01, .05, .1, .15],
```

(continues on next page)

(continued from previous page)

```
title="Probability of Shortfall",
label="contour levels: probability of shortfall")
```



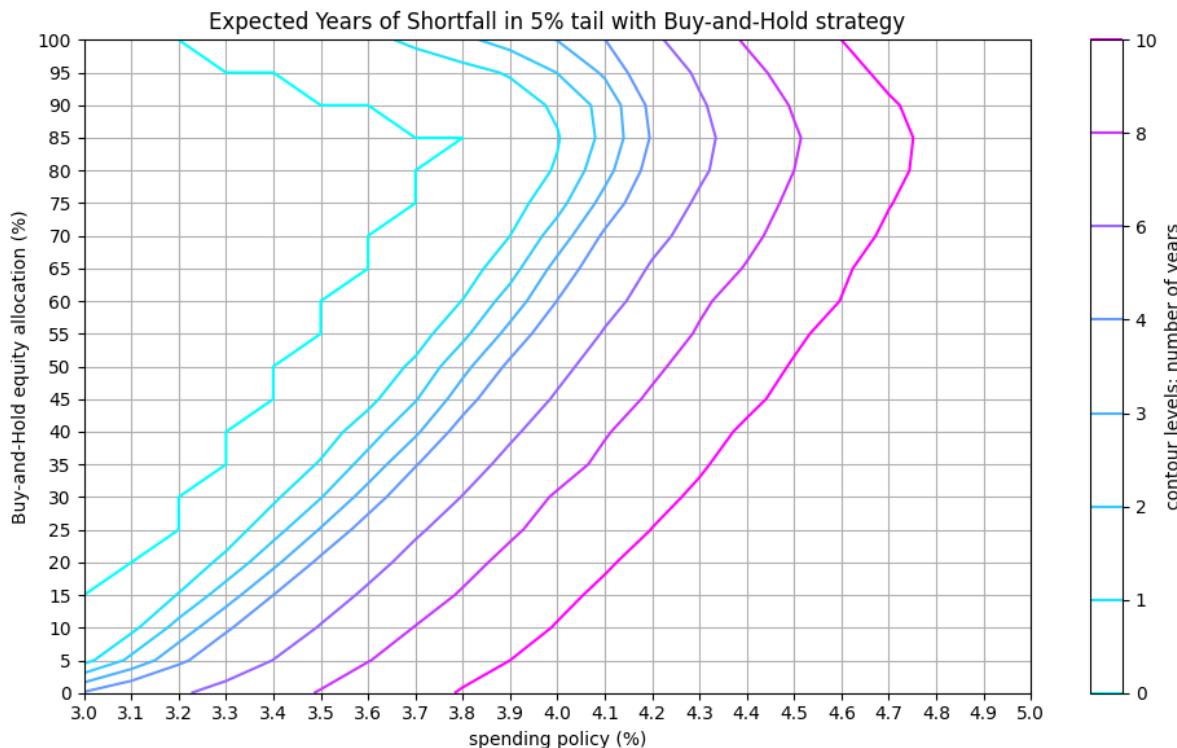
Expected shortfall period:

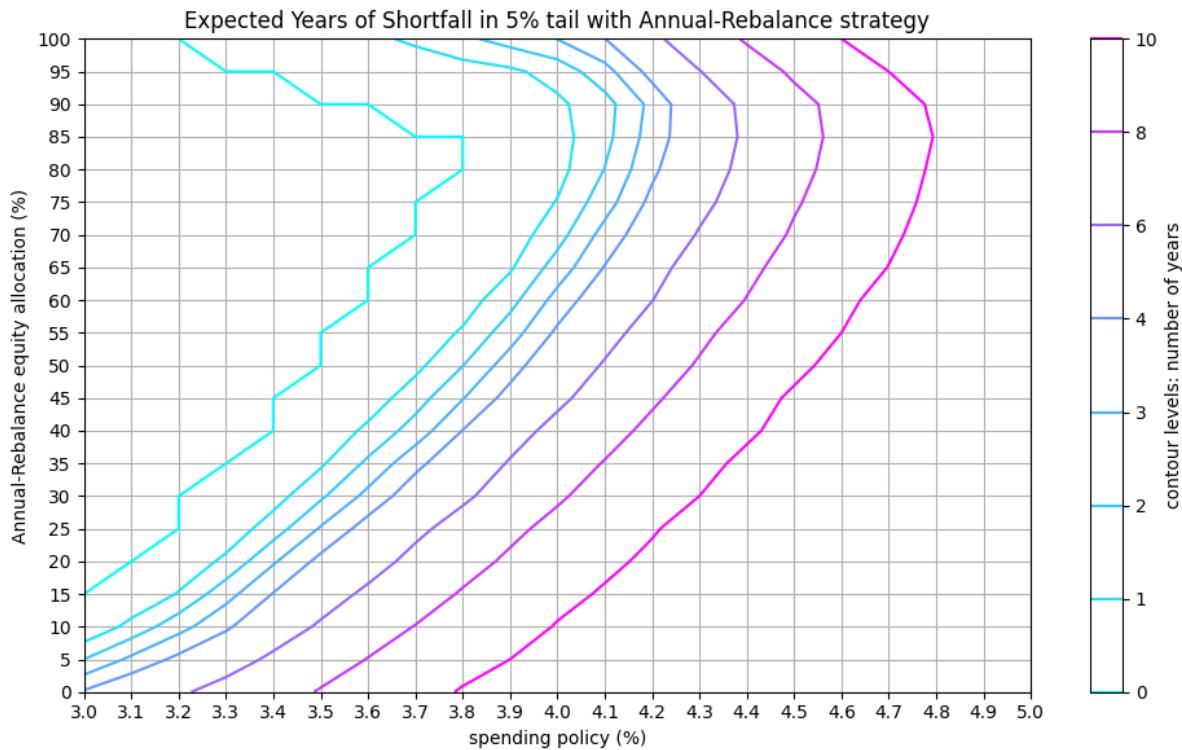
This metric estimates the potential duration of financial shortfalls, which helps retirees plan for worst-case scenarios.

```
#for model in [BaseModel, FixedModel]:
#    shortfall = store[model.name]['shortfall']
#    print(f"Expected Years of Shortfall in {tail}% tail: {model.name} allocation")
#    print(shortfall.iloc[::-1, :].round(1).to_string())
print('Buy-and-hold outperformed Annual-Rebalance:',
      round(np.mean(store[FixedModel.name]['shortfall']) > store[BaseModel.name][
        'shortfall']), 3))
print('Annual-Rebalance outperformed Buy-and-Hold:',
      round(np.mean(store[FixedModel.name]['shortfall']) < store[BaseModel.name][
        'shortfall']), 3))
```

```
Buy-and-hold outperformed Annual-Rebalance: 0.059
Annual-Rebalance outperformed Buy-and-Hold: 0.61
```

```
for model in [BaseModel, FixedModel]:
    shortfall = store[model.name]['shortfall']
    plot_contour(shortfall, levels=[0, 1, 2, 3, 4, 6, 8, 10],
                 title=f"Expected Years of Shortfall in {tail}% tail",
                 label="contour levels: number of years")
```





31.2 Deep reinforcement learning

Unlike static asset allocation models, **reinforcement learning (RL)** can learn optimal strategies that adapt to market conditions and remaining wealth.

31.2.1 Gymnasium environment

The Gymnasium (formerly OpenAI Gym) library provides a Pythonic interface for reinforcement learning problems. Stable Baselines3 (SB3) implements a set of RL algorithms in PyTorchf OpenAI's Gym library.

<https://gymnasium.farama.org/index.html>

<https://github.com/DLR-RM/stable-baselines3>

31.2.2 State space

During training, the RL model learns to predict optimal spending actions based on the current financial state. The **state space** includes:

- current wealth and allocation,
- recent market (equity and bonds) and inflation changes
- spending amount
- years since retirement

31.2.3 Actions

Exploitation: Selects the action with the highest expected value, based on past training data (e.g., Q-learning, SARSA).

Exploration: Tries alternative strategies to discover better long-term policies.

31.2.4 Reward function

- If assets are depleted, the model applies a severe penalty of -100 times the square of remaining years: $-(100T^2)$
- If wealth is positive, the reward is proportional to the wealth-to-spending coverage ratio: $\sqrt{\frac{W}{S(T+1)}}$

```

FAIL = 100      # failure reward factor
class CustomEnv(gym.Env):
    """Custom gymnasium environment, using Episodes scenario generator"""
    def __init__(self, model: BaseModel, episodes: Episodes):
        super().__init__()
        self.model = model          # for stepping through a 30-year episode
        self.episodes = episodes    # for generating a sample 30-year episode
        self.iterator = iter(self.episodes) # iterator to reset a 30-year sample
        T = self.model.initial['T']
        W = self.model.initial['W']
        low = np.array([0] + episodes.low + [0]*len(W))
        high = np.array([T] + episodes.high + [T]*len(W))
        self.observation_space = spaces.Box(low=low, high=high)
        self.action_space = spaces.Discrete(21)
        #self.action_space = spaces.Box(low=0.0, high=1.0)

    def reset(self, seed=0):
        super().reset(seed=seed)

        # generate a fresh 30-year episode
        self.episode = next(self.iterator, None)
        if self.episode is None:
            self.iterator = iter(self.episodes)
            self.episode = next(self.iterator)
        self.n = 0

        # return initial observations
        deltas = self.model.reset(self.episode.iloc[self.n])
        obs = [self.model.T] + list(deltas) + self.model.W.tolist()
        return obs, {}

    def step(self, action):
        S = self.model.W[-1]           # amount to spend at t-1
        W = np.sum(self.model.W[:-1])  # wealth at t-1
        T = self.model.T               # year remaining till termination

        # Convert action to asset allocation weights
        action = action * 0.05
        action = np.array([action, 1-action])

        # Grab next market move at time t
        self.n = self.n + 1
        deltas = self.episode.iloc[self.n].tolist()

```

(continues on next page)

(continued from previous page)

```

# Apply rebalance wealth and market move (t=1)
terminated, truncated = self.model.step(action, deltas)

# Calculate reward (t=1)
reward = -(T*T*FAIL) if truncated else math.sqrt(W / (S*T))

# Return as next observation
obs = [T] + list(deltas) + self.model.W.tolist()
return obs, reward, terminated, truncated, {}

```

Helpers to evaluate trained model

```

def evaluate(env, model, episodes):
    """Return success likelihood, shortfalls and asset allocation actions"""
    result = []
    actions = {t: [] for t in range(episodes.T + 1)} # to store predicted actions
    for n, episode in enumerate(iter(episodes)):
        obs, info = env.reset()
        terminated = False
        while not terminated:
            action, _states = model.predict(np.array(obs))
            actions[env.model.T].append(float(action))
            obs, rewards, terminated, truncated, info = env.step(action)
            result.append(env.model.T if truncated else 0)
        #print(n, truncated, action, rewards, env.model.W)
    return np.mean(np.array(result) != 0), *compute_shortfall(result, TAIL), actions

```

31.2.5 Deep Q-Network (DQN)

We use **Deep Q-Networks (DQN)** from Stable Baselines3, which is designed for discrete action spaces. This approach employs **deep reinforcement learning** to maximize wealth longevity while adapting asset allocation strategies dynamically.

```

TIMESTEPS = int(5e5)
initial_alloc = 50
fail, actions, shortfall, quantile = {}, {}, {}, {}
for rule in tqdm(rules): # train a model for each spending rule

    # define and train model for this spending rule
    name = str(round(rule, 1))
    W = [initial_alloc, 100-initial_alloc, rule]
    env = CustomEnv(model=BaseModel(T=T, W=W), episodes=episodes)
    clf = DQN('MlpPolicy', env, verbose=VERBOSE)
    clf.learn(total_timesteps=TIMESTEPS)
    clf.save(f"{outdir}/{name}")

    # evaluate model
    test_clf = clf.load(f"{outdir}/{name}", env=None)
    fail[name], quantile[name], shortfall[name], actions[name] =\
        evaluate(env, test_clf, episodes)

    store['dqn'] = dict(fail=fail, shortfall=shortfall,
                        quantile=quantile, actions=actions)

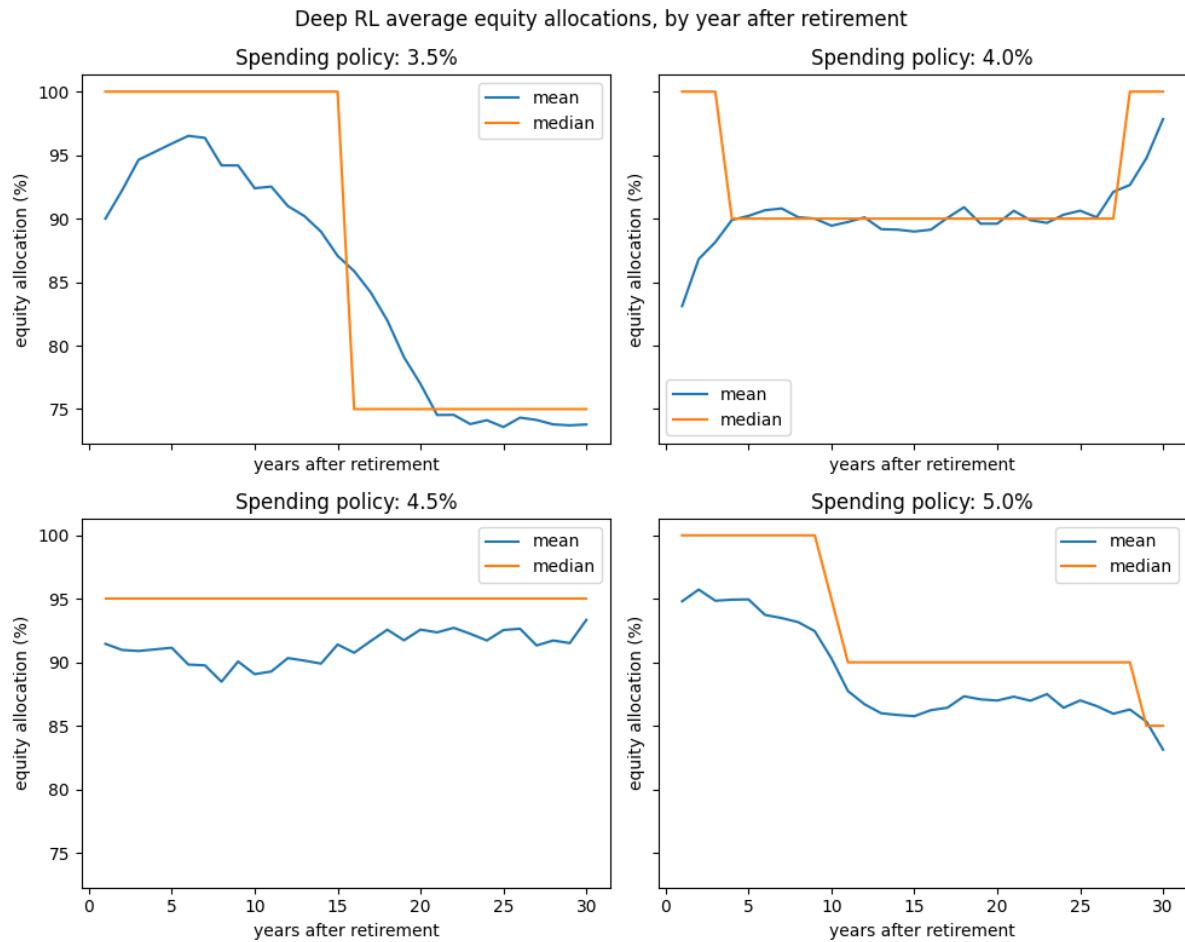
```

100% |██████████| 21/21 [2:00:19<00:00, 343.79s/it]

```
print(DataFrame(fail, index=["Deep RL"]).round(2).to_string())
```

	3.0	3.1	3.2	3.3	3.4	3.5	3.6	3.7	3.8	3.9	4.0	4.1	4.2	4.3	4.4	4.5	4.6	4.7	4.8	4.9	5.0	
Deep RL	0.0	0.0	0.0	0.0	0.01	0.0	0.0	0.01	0.02	0.02	0.03	0.06	0.06	0.06	0.08	0.1	0.11	0.14	0.12	0.16	0.21	0.19

```
# Plot average allocations over time
labels = list(np.arange(T) + 1)
fig, axs = subplots(nrows=2, ncols=2, figsize=(10, 8), sharex=True, sharey=True)
plt.suptitle('Deep RL average equity allocations, by year after retirement')
for ax, rule in zip(axs, ["3.5", "4.0", "4.5", "5.0"]):
    y = [[a*5 for a in actions[rule][i]] for i in labels]
    mean = [np.mean(a) for a in y]
    median = [np.median(a) for a in y]
    ax.plot(labels, mean, label='mean')
    ax.plot(labels, median, label='median')
    ax.set_title(f"Spending policy: {rule}%")
    ax.set_xlabel('years after retirement')
    ax.set_ylabel('equity allocation (%)')
    ax.legend()
plt.tight_layout()
```



References:

Richard S. Sutton and Andrew G. Barto, 2018, “Reinforcement Learning: An Introduction”, MIT Press.

Christine Benz, Jeffrey Ptak John Rekenthaler, Dec. 12, 2022, “The State of Retirement Income: 2022. A look at how higher bond yields, lower equity valuations, and inflation affect starting safe withdrawal rates”, Morningstar Portfolio and Planning Research.

LANGUAGE MODELING

Attention is all you need - Vaswani et al

Transformers, built on the **attention** mechanism, are neural network models designed to process variable-length sequences and capture complex dependencies in language without relying on recurrence or convolution. By leveraging self-attention, multi-head attention, and positional encodings, transformers can model long-range relationships between words for tasks like text generation, translation, and summarization. We apply transformer-based models to language modeling of Federal Reserve meeting minutes, introducing perplexity as a key evaluation metric and exploring decoding strategies such as nucleus sampling to generate coherent and diverse text.

```
# By: Terence Lim, 2020-2025 (terence-lim.github.io)
from typing import Callable, List
import math
import numpy as np
import pandas as pd
from pandas import Series, DataFrame
import bisect
import matplotlib.pyplot as plt
from nltk.tokenize import wordpunct_tokenize as tokenize
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.optim import Adam
from torch.optim.lr_scheduler import StepLR
from torch.utils.data import Dataset, DataLoader
import torchinfo
from tqdm import tqdm
from finds.database.mongodb import MongoDB
from finds.unstructured import Unstructured, Vocab
from secret import credentials, paths
# %matplotlib qt
VERBOSE = 0
```

```
mongodb = MongoDB(**credentials['mongodb'], verbose=VERBOSE)
fomc = Unstructured(mongodb, 'FOMC')
outdir = paths['scratch']
device = 'cuda' if torch.cuda.is_available() else 'cpu'
print('Device:', device)
```

Device: cuda

32.1 Transformers

Transformers are a neural network architecture built entirely on attention mechanisms, designed to process variable-length sequential data without relying on recurrence (as in RNNs) or convolution (as in CNNs). They are especially effective for natural language processing (NLP) tasks such as language modeling, translation, and text generation.

Traditional models like RNNs and CNNs face limitations when processing language data. RNNs are sequential, making them slow to train and difficult to parallelize, and they suffer from vanishing gradients. CNNs, while powerful for structured patterns like images, are less suited for variable-length, syntactically complex language sequences.

Because language involves variable lengths, hierarchical syntax, and long-range dependencies, attention allows the model to focus on relevant parts of the input when generating outputs. To introduce positional information, since Transformers lack an inherent sense of sequential order, positional encodings to provide information about word positions are added to the token embeddings. In autoregressive tasks, **causal masks** are applied to ensure that each token only attends to previous tokens, not future ones.

32.1.1 Attention mechanism

Attention is a mechanism that enables models to reason about a **set of elements and their relationships**, dynamically weighting the importance of different parts of the input.

Attention is essentially a set operator designed to reason about a set of elements and their relationships, dynamically weighting the importance of different parts of the input when producing each element of the output sequence. Unlike RNNs, attention does not require sequential processing, making it highly parallelizable and efficient for sequence modeling.

The inputs to the attention operator are called:

- Queries (Q): What we're looking for.
- Keys (K): Labels that help identify relevant content.
- Values (V): The actual content or information to aggregate.

In **self-attention**, these inputs are all derived from the same input, allowing each input element to attend to every other element. Each input token is represented by three vectors: *query*, *key*, and *value*. Attention scores are computed as a scaled dot product between queries and keys, and these scores weight the values to produce a contextualized representation:

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^\top}{\sqrt{C}}\right)V$$

Learnable weight matrices W_Q, W_K, W_V enable queries, keys, and values, respectively, to adapt to different input patterns:

$$\text{Attention}(X; W_Q, W_K, W_V) = \text{Attention}(XW_Q, XW_K, XW_V)$$

where X is the embedded input.

Cross-attention is a mechanism used in encoder-decoder transformer architectures for tasks such as machine translation, where queries (Q) come from one source (typically the decoder) and keys (K) and values (V) come from another source (typically the encoder output)..

Multi-head attention runs multiple independent attention layers (heads) in parallel. Each head learns different ways of attending to the input, capturing different aspects of the relationships. Heads are concatenated and linearly projected back to the output dimension.

32.1.2 Masked attention

Auto-regressive prediction is used for sequence generation tasks such as text completion and translation. To ensure predictions are causal (based only on past and current information), causal masks hide future tokens when computing attention. The mask is just a upper-triangular matrix applied to the attention scores to block future tokens, ensuring that each word can only attend to itself and earlier words. This prevents the model from “cheating” by looking at future tokens when generating or predicting a sequence, preserving the causal structure required for tasks like language modeling and text completion.

32.1.3 Positional encoding

Since Transformers treat inputs as sets of tokens without inherent order, **positional encodings** provide sequence information. These are added to token embeddings to enable the model to understand word positions.

Types of positional encodings include:

- Absolute Positional Embeddings: Add a fixed position index to each input token. Although simple and straightforward, this method is not generalizable to longer sequences than seen during training.
- Relative Positional Embeddings: encode pairwise distances between tokens. These relative distances are bounded and reusable, hence independent of the total length of the sequence.
- Sinusoidal Positional Embeddings: Predefined using sine and cosine functions of different frequencies. This can capture relative position information which generalizes to sequences longer than trained on. $PE(n, 2i) = \sin\left(\frac{n}{10000^{2i/C}}\right)$, $PE(n, 2i + 1) = \cos\left(\frac{n}{10000^{2i/C}}\right)$
- Rotary Positional Embeddings (RoPE): Combines both absolute and relative positional information through a rotation operation, which extrapolates well to longer contexts and is widely adopted in large language models (LLMs).
- Learnable Positional Embeddings: Initialized randomly and learned during training like token embeddings. This is fully flexible, but performance may degrade if sequence length varies significantly between training and testing.

```
class PositionalEncoding(nn.Module):
    """Positional encoder, learned with an embeddings layer"""
    def __init__(self, d_model: int, max_len: int, dropout: float= 0.0):
        super().__init__()
        self.dropout = nn.Dropout(dropout)
        self.emb = nn.Embedding(num_embeddings=max_len, embedding_dim=d_model)

    def forward(self, x):
        """
        Args:
            x: Tensor, shape [seq_len, batch_size, embedding_dim]
        """
        to_embed = torch.LongTensor(np.asarray(range(0, x.size(1)))) \
            .to(x.device)
        embedded = self.emb(to_embed)
        embedded = self.dropout(embedded)
        return x + embedded.unsqueeze(0)

    """ Alternate positional encodings with sine function
    def __init__(self, d_model: int, max_len: int, dropout: float = 0.1):
        super().__init__()
        self.dropout = nn.Dropout(p=dropout)
        position = torch.arange(0, max_len, dtype=torch.float).unsqueeze(1)
        div_term = torch.exp(torch.arange(0, d_model, 2) \
            * (-math.log(10000.0) / d_model))
        pe = torch.zeros(max_len, d_model)
```

(continues on next page)

(continued from previous page)

```

pe[:, 0::2] = torch.sin(position * div_term)
pe[:, 1::2] = torch.cos(position * div_term)
pe = pe[:, None, :]
self.register_buffer('pe', pe)

def forward(self, x):
    x = x + self.pe[:x.size(1), 0, :]
    return self.dropout(x)
"""

```

32.1.4 Transformer layers

A transformer-based neural network is built from repeated blocks of Transformer **layers**, each consisting of:

- Multi-Head Attention: Combines several attention “heads” that learn different relationships between tokens. Each head performs: $\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{C}}\right)V$ and the final output is concatenated and linearly projected back to match input dimensions.
- Feedforward Neural Network: Applies a two-layer fully connected network (MLP) with non-linearity (typically ReLU) in between: $\text{MLP}(x) = \text{ReLU}(\text{Linear}_1(x)) \rightarrow \text{Linear}_2$
- Residual Connections: Directly connect input and output of each sub-layer.
- Layer Normalization: Normalizes inputs within each layer.

The input sentence is split into parts (characters, words, or “tokens”). The model takes token embeddings with positional encodings, applies layers of attention and MLPs, and outputs contextualized representations of each token.

```

class Transformer(nn.Module):
    """Transformer neural network"""
    def __init__(self, seq_len: int, vocab_size: int, d_model: int, nhead: int,
                 num_layers: int, dim_feedforward: int, dropout: float):
        super().__init__()

        # model dimensions
        self.seq_len = seq_len
        self.vocab_size = vocab_size
        self.d_model = d_model

        # define layers
        self.embedding = nn.Embedding(num_embeddings=vocab_size,
                                      embedding_dim=d_model)
        self.positional = PositionalEncoding(max_len=seq_len,
                                             d_model=d_model,
                                             dropout=dropout)

        layer = nn.TransformerEncoderLayer(d_model=d_model,
                                           nhead=nhead,
                                           dim_feedforward=dim_feedforward,
                                           dropout=dropout,
                                           batch_first=True)
        self.encoder = nn.TransformerEncoder(encoder_layer=layer,
                                             num_layers=num_layers)
        self.decoder = nn.Linear(in_features=d_model,
                               out_features=vocab_size)

```

(continues on next page)

(continued from previous page)

```

# initialize weights
self.embedding.weight.data.uniform_(-0.1, 0.1)
self.decoder.weight.data.uniform_(-0.1, 0.1)
self.decoder.bias.data.zero_()

def causal_mask(self, sz: int, device: str = 'cpu'):
    """returns upper triu set to -inf"""
    return nn.Transformer.generate_square_subsequent_mask(sz=self.seq_len,
                                                          device=device)

def forward(self, x):
    if len(x.shape) == 1:
        x = x[None, :]
    assert x.size(-1) == self.seq_len
    x = self.embedding(x) * math.sqrt(self.d_model)    # embedding
    x = self.positional(x)                            # position encoding
    x = self.encoder(x, mask=self.causal_mask(sz=len(x), device=x.device))  # encoder
    x = self.decoder(x)                                # linear layer
    x = F.log_softmax(x, dim=-1)                      # classify
    return x

def save(self, filename):
    """save model state to filename"""
    return torch.save(self.state_dict(), filename)

def load(self, filename):
    """load model name from filename"""
    self.load_state_dict(torch.load(filename, map_location='cpu'))
    return self

```

32.2 Language modeling

Language modeling is the task of estimating the probability distribution over word sequences. By learning this distribution from large text corpora, models capture linguistic structure, enabling downstream tasks like translation and text generation.

32.2.1 Perplexity

Accuracy (measuring whether the predicted word is exactly correct) is not a meaningful metric for language models. Predicting the exact next word in a sequence is highly uncertain and difficult, so accuracy would be very low even for strong models. Instead, we care about how well the language model assigns probability distributions over possible next words. Perplexity quantifies how well the it predicts the test set, calculated as the exponential of the average negative log likelihood over the test set:

$$\text{Perplexity} = \exp \left(-\frac{1}{N} \sum_{i=1}^N \log P(w_i | w_{i-1}, \dots, w_{i-n+1}) \right)$$

- N is the total number of words in the test set.
- $P(w_i | w_{i-1}, \dots, w_{i-n+1})$ is the probability assigned by the language model to the word w_i given its context $w_{i-1}, \dots, w_{i-n+1}$.

Intuitively, perplexity measures how surprised the model is by the text. It can be interpreted as the geometric mean of the inverse probabilities assigned by the model, hence lower perplexity indicates better model performance and generalization.

```
def get_next_log_probs(model, context: List[str], unk=UNK):
    """log P(word | context) where word ranges over the vocab"""
    # pad to length seq_len
    if len(context) > model.seq_len:
        context = context[-model.seq_len:]
    elif len(context) < model.seq_len:
        context = ([unk] * (model.seq_len - len(context))) + context
    context = torch.LongTensor(vocab.get_index(context)) \
        .to(device) \
        .unsqueeze(0)
    output = model(context)
    logits = output[0, -1, :]
    return logits.cpu().detach().numpy()
```

```
def get_perplexity(model, sentence: List[str]) -> float:
    """Compute perplexity score"""

    context = [UNK] + sentence
    log_probs = 0
    for i in range(len(context) - len(sentence), len(context)):
        log_probs += get_next_log_probs(model=model, context=context[:i]) [vocab.get_
        ↪index(context[i])]
    return np.exp(-log_probs / len(sentence))
```

32.2.2 Fedspeak

The Fed has a jargon all of its own, with Alan Binder coining the term *Fedspeak* to describe the “turgid dialect of English” used by Federal Reserve Board chairs. We explore language modeling of minutes text from all FOMC meetings since 1993.

The text data are tokenized and converted into indices in a Vocab object. PyTorch Dataset and DataLoader tools simplify the processing of chunks and batches of the data. The most recent document is held-out from training to serve as the test set for perplexity evaluation.

```
# Retrieve and preprocess FOMC minutes text
dates = fomc['minutes'].distinct('date')           # check dates stored in MongoDB
docs = Series({doc['date']: [w.lower() for w in tokenize(doc['text'])]}
             for doc in fomc.select('minutes'))
name='minutes').sort_index()
UNK = " "
vocab = Vocab(set().union(*docs.tolist()), unk=UNK)
print(f"len(vocab)={len(vocab)}, len(docs)={len(docs)}: {min(dates)}-{max(dates)}")
```

```
len(vocab)=8675, len(docs)=256: 19930203-20250129
```

```
# Pytorch Dataset and DataLoader
class FOMCdataset(Dataset):
    """Subclass of torch Dataset
```

Notes:

(continues on next page)

(continued from previous page)

```

All subclasses should overwrite __getitem__(),
supporting fetching a data sample for a given key. Subclasses
could also optionally overwrite __len__(), which is expected to
return the size of the dataset

"""

def __init__(self, text: Series, seq_len: int, get_index: Callable[[str], int]):
    self.text = text
    self.seq_len = seq_len
    self.get_index = get_index
    self.counts = np.cumsum([len(s) // seq_len for s in text])

def __len__(self):
    return self.counts[-1]

def __getitem__(self, idx):
    assert 0 <= idx < len(self), "idx out of range"
    doc = bisect.bisect_right(self.counts, idx)

    start = (idx - (self.counts[doc-1] if doc > 0 else 0)) * self.seq_len
    end = start + self.seq_len
    chunk = self.text.iloc[doc][start:end]
    return (torch.LongTensor([0] + self.get_index(chunk[:-1])),
            torch.LongTensor(self.get_index(chunk)))

# length of input sequence
seq_len = 30

# split last document to be test set
test_len = 1
test_set = docs.iloc[-test_len: ].tolist()
train_set = FOMCDataset(docs.iloc[: -test_len], seq_len, vocab.get_index)
dataloader = DataLoader(train_set, batch_size=32, shuffle=True)
DataFrame({'docs': len(docs) - test_len, 'chunks': len(train_set)}), index=['Train'])

```

	docs	chunks
Train	255	54877

Create the model:

```

# Create the model
lr = 0.0001
step_size = 30
num_epochs = 100 #step_size * 1

d_model = 512 #512
nhead = 4 # 4
num_layers = 3 # 2
dim_feedforward = 2048 # 512 #1024
dropout = 0.3 # 0.3 # 0.2

model = Transformer(seq_len=seq_len,
                    vocab_size=len(vocab),
                    d_model=d_model,
                    nhead=nhead,

```

(continues on next page)

(continued from previous page)

```
        num_layers=num_layers,
        dim_feedforward=dim_feedforward,
        dropout=dropout).to(device)
torchinfo.summary(model)
```

```
=====
Layer (type:depth-idx)                                     Param #
=====
Transformer
|─Embedding: 1-1                                         4,441,600
|─PositionalEncoding: 1-2                                 --
|  └Dropout: 2-1                                         --
|  └Embedding: 2-2                                       15,360
|─TransformerEncoder: 1-3                                --
|  └ModuleList: 2-3                                      --
|    └TransformerEncoderLayer: 3-1                         3,152,384
|    └TransformerEncoderLayer: 3-2                         3,152,384
|    └TransformerEncoderLayer: 3-3                         3,152,384
|─Linear: 1-4                                           4,450,275
=====
Total params: 18,364,387
Trainable params: 18,364,387
Non-trainable params: 0
=====
```

Train the model:

```
# Specify training parameters
criterion = nn.NLLLoss().to(device)
optimizer = Adam(model.parameters(), lr=lr)
scheduler = StepLR(optimizer, step_size=step_size, gamma=0.1)
```

```
perplexity = []
losses = []
for epoch in tqdm(range(num_epochs)):
    model.train()
    for train_ex, target_ex in dataloader:
        optimizer.zero_grad()
        train_ex, target_ex = train_ex.to(device), target_ex.to(device)
        output = model(train_ex)
        loss = criterion(output.view(-1, len(vocab)), target_ex.view(-1))
        loss.backward()
        optimizer.step()
    scheduler.step()

    # Evaluate perplexity on test set
    model.eval()
    perplexity.append(np.mean([get_perplexity(model, s) for s in test_set]))
    losses.append(loss.item())
    if VERBOSE:
        print(f"Epoch: {epoch}, Loss: {loss.item()}, Perplexity: {perplexity[-1]}")
model.save(outdir / f"transformer{nhead}_{dim_feedforward}.pt")
```

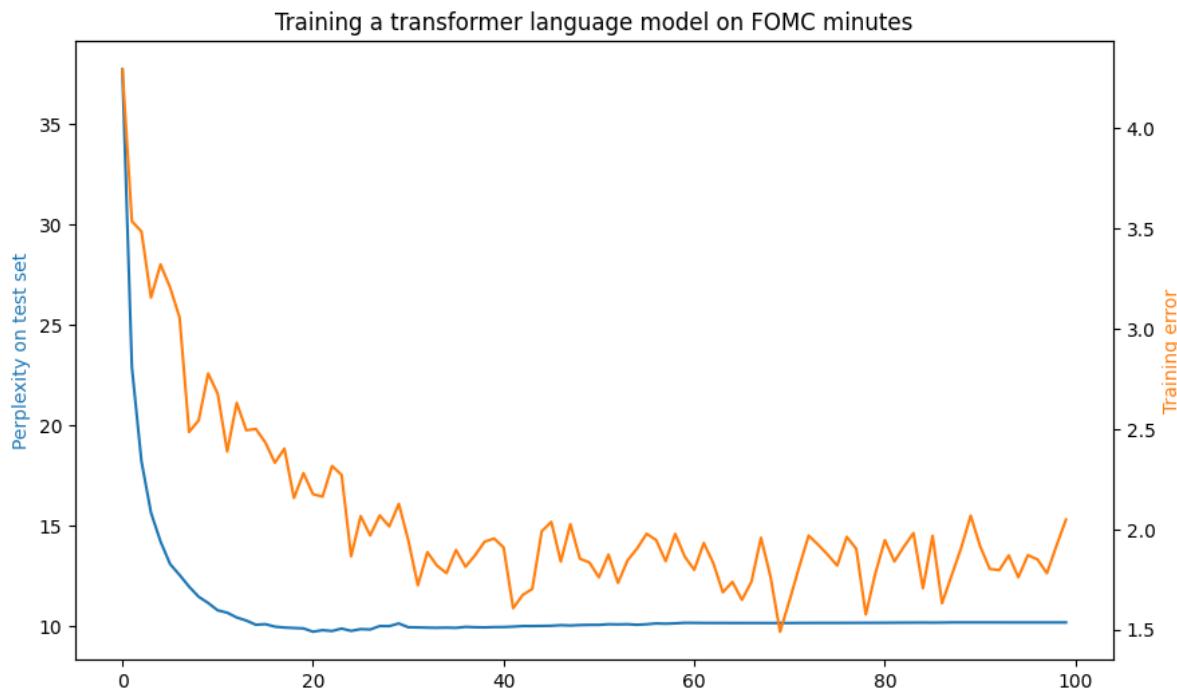
100% |██████████| 100/100 [1:03:53<00:00, 38.33s/it]

```
# save model checkpoint
import warnings
with warnings.catch_warnings():
    warnings.filterwarnings('ignore')  ## ignore the weights_only=True future warning
model.load(outdir / f"transformer{nhead}_{dim_feedforward}.pt")
```

Evaluate the model:

```
# Plot perplexity
fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(perplexity, color="C0")
ax.set_ylabel('Perplexity on test set', color="C0")
bx = ax.twinx()
bx.plot(losses, color="C1")
bx.set_ylabel('Training error', color="C1")
plt.title('Training a transformer language model on FOMC minutes')
print('Perplexity:', perplexity[-1], ' Loss:', losses[-1])
```

Perplexity: 10.185370762878241 Loss: 2.0497496128082275



32.2.3 Decoding

Decoding refers to the process of generating a sequence of words based on learned probabilities. Language models generate text by sampling from a probability distribution over the next word $P(y_i|y_1, \dots, y_{i-1})$, given previous words:

- **Greedy** approach: At each step of generation, the word with the highest probability according to the model is selected as the next word. While simple and computationally efficient, this results in repetitive or less diverse outputs.
- **Beam search** maintains a fixed number (beam width) of partial candidate sequences of words. At each step, it expands all possible next words for each candidate, keeping the top k based on their joint probabilities. This allows exploration of multiple promising paths, but can be computationally expensive, and may still produce suboptimal outputs due to early pruning.
- **Nucleus Sampling** samples from the smallest set of k words whose cumulative probability mass exceeds a pre-defined threshold p . This approach promotes diversity in generated text by allowing for the possibility of sampling from a larger set of words.

```
def get_nucleus_sequence(model, n: int, p: float, context: List[str] = []):
    """Sample sequence of words given context using nucleus sampling"""
    if not context:
        context = [UNK]

    for i in range(n):
        probs = np.exp(get_next_log_probs(model, context))
        probs_sorted = sorted(probs, reverse=True)
        probs_cum = np.cumsum(probs_sorted)
        num_drop = sum(probs_cum > p)
        threshold = probs_sorted[-num_drop]
        probs[probs < threshold] = 0.
        probs /= sum(probs)
        choice = vocab.get_word(np.random.choice(len(probs), p=probs))
        context.append(choice)
        #print(i, drop, len(probs), len(probs_sorted))
    return context
```

```
import textwrap
wrapper = textwrap.TextWrapper(width=80, fix_sentence_endings=True)
```

Finally, nucleus sampling with $p = 0.95$ is used to generate new text conditioned on starting contexts, balancing diversity and coherence.

```
n, p = seq_len * 4, 0.95
for context in ['the financial markets', 'participants noted that']:

    # generate from context with nuclear sampling
    words = get_nucleus_sequence(model, n=n, p=p, context=context.split())

    # pretty-print the output
    out = ''
    is_end = True
    is_space = ''
    for w in words:
        if not w.isalnum():
            out += w
        else:
            if is_end:
                out += '\n'
            is_end = False
            is_space = w
    print(wrapper.fill(out))
```

(continues on next page)

(continued from previous page)

```

w = w.capitalize()
out += is_space + w
is_end = w in ['!', '?', '.']
is_space = ' '*bool(w not in ['"', "'", '-'])
print(f"{{context.upper()}}...")
print(wrapper.fill(out))
print()

```

THE FINANCIAL MARKETS...

The financial markets. In addition, the tga and the resulting decline in the soma portfolio would result in a combination of shifts in the composition of reserve liabilities, and a waning volume of credit allocation liquidity. In that regard, the appropriate course of monetary policy, a number of participants noted that purchases of longer-term securities were faced by the likely onset of the financial crisis in mid-december. Labor market conditions improved further in january but expanded modestly on balance over the intermeeting period. Consumer price inflation— as measured by the 12-month percentage change in the price index for personal consumption expenditures(pce)— was elevated in march

PARTICIPANTS NOTED THAT...

Participants noted that recent indicators and orders pointed to somewhat more moderate expansion of spending for equipment and software. The nominal deficit on u. S. Trade in goods and services was significantly larger in the third quarter than in the previous quarter. The value of exports of goods and services also increased considerably in july, with increases widespread by categories. Imports of services rose more than exports. The increase in imports was concentrated in consumer goods, however, consumer goods, and services, which decreased exports of capital goods. Imports of services in july and august were expanding briskly; the gains were concentrated in industrial supplies, semiconductors, and services.

References:

<https://pytorch.org/tutorials/beginner/transformer Tutorial.html>

Jay Alammar, The Illustrated Transformer, retrieved from <https://jalammar.github.io/illustrated-transformer/>

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. <https://arxiv.org/abs/1706.03762>

Greg Durrett, 2021-2024, “CS388 Natural Language Processing course materials”, retrieved from <https://www.cs.utexas.edu/~gdurrett/courses/online-course/materials.html>

Philipp Krähenbühl, 2020-2024, “AI394T Deep Learning course materials”, retrieved from https://www.philkr.net/dl_class/material and <https://ut.philkr.net/deeplearning/>

CHAPTER
THIRTYTHREE

LARGE LANGUAGE MODELS

I didn't have time to write a short letter, so I wrote a long one instead - Mark Twain

We introduce large language models (LLMs) through a financial natural language processing (NLP) task: summarizing the *Quantitative and Qualitative Disclosures About Market Risk* sections of 10-K reports. To assess performance, we compare the overlap and readability of summaries generated by GPT-4o-mini, a proprietary closed-source model, and DeepSeek-R1-14B, an open-source model that can be downloaded and run locally. Small language models, particularly those trained using techniques like **distillation**, can closely approximate the performance of larger models while offering lower latency and reduced memory requirements.

```
# By: Terence Lim, 2020-2025 (terence-lim.github.io)
import numpy as np
import pandas as pd
from pandas import DataFrame, Series
import textwrap
from pprint import pprint
from rouge_score import rouge_scorer
from tqdm import tqdm
from finds.database import SQL, RedisDB
from finds.unstructured import Edgar
from finds.structured import BusDay, CRSP, PSTAT
from finds.readers import Sectoring
from secret import paths, credentials
VERBOSE = 0
```

```
sql = SQL(**credentials['sql'], verbose=VERBOSE)
user = SQL(**credentials['user'], verbose=VERBOSE)
bd = BusDay(sql)
rdb = RedisDB(**credentials['redis'])
crsp = CRSP(sql, bd, rdb, verbose=VERBOSE)
pstat = PSTAT(sql, bd, verbose=VERBOSE)
ed = Edgar(paths['10X'], zipped=True, verbose=VERBOSE)
```

33.1 OpenAI GPT models

Large language models are built using transformer-based deep learning architectures and pre-trained on massive text corpora. GPT models, short for Generative Pre-trained Transformers, use an autoregressive approach to learn the structure of language by predicting the next token given the previous ones. The transformer architectures allows these models to capture long-range dependencies in text, making them particularly powerful for understanding context and generating fluent text. Modern LLMs extend this base with techniques like instruction tuning and reinforcement learning from human feedback (RLHF), which improve their usability and alignment with human intent.

- **Pre-training** teaches the model general language patterns from large amounts of raw text data. This process builds a foundational base that can be fine-tuned for specific tasks later.
- **Instruction tuning** guides the model to follow specific types of tasks or instructions.
- **RLHF** improves output quality by training the model to reflect human preferences.

OpenAI's GPT series, from GPT-2 to GPT-03, has demonstrated increasingly powerful capabilities due to the scale of their parameters and training data, containing millions or billions, and now trillions of adjustable weights in their deep neural networks. GPT-3 represented a fundamental shift in AI, demonstrating how scaling models alone could achieve generalization. It also introduced *In-Context Learning*, allowing models to learn from examples in the prompt without fine-tuning. GPT-4 expanded the context length to 128K **tokens** (which are how LLMs represent the fundamental units of text, which can be as small as single characters or as large as whole words), significantly improving its ability to understand and summarize long documents. These models, however, are only available through proprietary APIs.

```
gpt_name = "gpt-4o-mini"
```

33.1.1 LangChain framework

A modular framework for building applications with language models, such as **LangChain** simplifies the process of integrating language models with external data sources and other AI tools. It abstracts over the underlying LLM API (OpenAI, Ollama, etc.) and allows users to create chains of prompts, tools, and logic for custom NLP workflows.

```
# Initializes an OpenAI model using LangChain. temperature=0 ensures deterministic
→outputs
from langchain_openai import ChatOpenAI
gpt_model = ChatOpenAI(model_name=gpt_name, temperature=0, **credentials['openai'])
pprint(gpt_model.to_json())
```

```
{'id': ['langchain', 'chat_models', 'openai', 'ChatOpenAI'],
 'kwargs': {'model_name': 'gpt-4o-mini',
            'openai_api_key': {'id': ['OPENAI_API_KEY'],
                               'lc': 1,
                               'type': 'secret'},
            'temperature': 0.0},
 'lc': 1,
 'name': 'ChatOpenAI',
 'type': 'constructor'}
```

Temperature controls randomness in generation: lower values yield more deterministic responses, while higher values lead to more creative or diverse outputs.

33.1.2 Open-source models

A large language model consists of three key components:

- Architecture: The structure of the model (e.g., Transformer-based).
- Weights: The learned parameters that define the model's behavior.
- Training code & data: The scripts and datasets used to train the model.

LLMs are categorized as:

- **Closed models:** API-only, no access to weights or training data (e.g., GPT-4).
- **Open models:** Model weights are available, but full training details are not (e.g., LLaMA, Qwen).
- **Open-source models:** Full transparency including architecture, code, data, and weights (e.g., DeepSeek-R1).

BERT (Bidirectional Encoder Representations from Transformers), released by Google in October 2018 not long after the seminal “Attention is All You Need” paper, pioneered transformers-based models for NLP tasks. Llama-2 and Llama-3 from Meta continued this trend with larger and more efficient architectures. Instruction-tuned models like Alpaca was built Meta’s LLaMA model to improve task performance with minimal resources.

LLM	Number of Parameters	Context Length
BERT-Base	110 million	512
BERT-Large	340 million	512
Llama-2-7B	7 billion	4K
Alpaca	7 billion	4K
Llama-3-8B	8 billion	8K
Llama-3-70B	70 billion	8K
GPT-2	1.5 billion	1K
GPT-3.5	175 billion	4K
GPT-4	~1 trillion	128K

33.2 DeepSeek-R1 model

DeepSeek-R1 is a powerful open-weight language model released by DeepSeek in January 2025, with size ranging from 1.3B to 236B parameters across different variants. Supporting a context length up to 128K tokens with a GPT-style transformer decoder-only architecture, it was trained with 6-10T tokens from multilingual internet sources. Furthermore, DeepSeek-R1 was fine-tuned to implement chain-of-thought reasoning without explicit prompting. Its training process included:

- synthetic dataset of thousands of long-form CoT examples
- group relative policy optimization, a reinforcement learning that improved its ability to solve challenging problems
- fine-tuning using a final round of reinforcement learning to boost its reasoning accuracy, helpfulness and harmlessness.

The model exposes its reasoning during inference, a departure from the typical black-box approach of other models, allowing users to witness the model’s “thinking process” as it works through problems.

33.2.1 Distilled models

Distillation compresses LLMs by transferring knowledge from a large *teacher* model to a smaller *student* model.

- Knowledge Distillation (KD): Student learns from the teacher's output probabilities (soft targets) in addition to true labels (hard targets).
- Intermediate Layer Distillation: Transfers information from internal layers.
- Data Augmentation: Uses teacher-generated samples to expand the training set.

LLM distillation is expected to become an even more important practice in the AI world. Examples include GPT-4o distilled into GPT-4o-mini, or DeepSeek-R1 variants trained on Llama and Qwen to preserve reasoning capabilities with fewer parameters.

Distilled versions of DeepSeek-R1 are available in various sizes, including 1.5B, 7B, 14B, 32B, and 70B parameters. These models used DPO (Direct Preference Optimization) or supervised fine-tuning on synthetic highly-curated datasets generated by the larger R1 models, retaining 90–95% of teacher model performance with lower latency.

<https://ollama.com/library/deepseek-r1>

```
# model name in Ollama
model_name = "deepseek-r1:14b"
```

33.2.2 Small language models

Small language models (SLMs) are smaller in scale and scope than large language models (LLMs), with number of parameters ranging from a few million to a few billion. Requiring less memory and computational power, they can be deployed in resource-constrained environments such as edge devices, mobile apps and off-line situations where AI inferencing (when a model generates a response to a user's query) must be done without a data network.

33.2.3 Ollama server

Ollama simplifies running open-source LLMs locally. After installing the Ollama runtime and pulling a model (e.g., `deepseek-r1:14b`), it can serve requests on localhost. It provides a simple API for creating, running, and managing models, as well as a library of pre-built models. This allows experimentation with high-performance LLMs, improving accessibility, privacy, and latency.

<https://github.com/ollama/ollama>

1. Install Ollama (<https://ollama.com/>)
 - curl <https://ollama.ai/install.sh> | sh
 - ls -ltra which ollama``
 - ollama --version
2. Pull a model (stored in `/usr/share/ollama/.ollama/models/`)
 - ollama pull deepseek-r1:14b
 - ollama list
3. Serve an LLM
 - ollama run deepseek-r1:14b # uses GPU
 - ollama ps

4. or Linux service

- sudo systemctl status ollama # service status
- sudo systemctl disable ollama # disable so it does not start up again upon reboot
- sudo systemctl stop ollama # stop service
- sudo systemctl restart ollama # restart service
- sudo rm /etc/systemd/system/ollama.service # delete service file
- sudo rm \$(which ollama) # remove ollama binary

5. Endpoint

- curl http://localhost:11434/api/generate -d '{"model": "deepseek-r1:14b", "prompt": "Why is the sky blue?"}'

```
# Initializes a local LLM (DeepSeek-R1) using Ollama
from langchain_ollama.llms import OllamaLLM
model = OllamaLLM(model=model_name, temperature=0)
pprint(model.to_json())
```

```
{'id': ['langchain_ollama', 'llms', 'OllamaLLM'],
 'lc': 1,
 'name': 'OllamaLLM',
 'repr': "OllamaLLM(model='deepseek-r1:14b', temperature=0.0)",
 'type': 'not_implemented'}
```

33.3 Text summarization

Summarization condenses lengthy documents into concise outputs. LLMs can perform abstractive summarization, generating summaries in their own words rather than extracting sentences. Summarization is a core NLP benchmark, critical for a wide variety of applications.

33.3.1 Natural language processing (NLP) tasks

These tasks play a crucial role in the field of **natural language processing**, challenging research and applications that have enhanced how machines understand and interact with human language. The performance of LLMs on these tasks are commonly evaluated using large benchmark datasets, such as MMLU (undergraduate level knowledge), GSM-8K (grade-school math), HumanEval (coding), GPQA (graduate-level questions), and MATH (math word problems). However, the interpretation of these results should be tempered by the inadvertent risk that some benchmark examples found their way in the data set used for training models.

- Natural Language Inference (NLI), also known as textual entailment, is the task of determining the relationship between two sentences, i.e. predict whether one sentence (the hypothesis) logically follows from another sentence (the premise).
- Named Entity Recognition (NER) involves identifying and classifying named entities within a text into predefined categories such as person names, organizations, locations, dates, etc.
- Text Generation is the process of generating coherent and contextually relevant text given a certain input or prompt.
- Machine Translation (MT) is the task of automatically translating text from one language to another.

- Text Summarization involves creating a concise summary of a longer text while preserving its key information and meaning.
- Reading comprehension requires models to read a passage of text and answer questions about it, demonstrating understanding of the text. Some challenges when developing and evaluating reading comprehension models include:
 - Artifacts, which refer to incorrect or misleading information generated by models that do not reflect the true content of the text but rather exploit patterns in the training data
 - Adversarial attacks, which are instances where models fail due to intentional manipulation or perturbation of the input, aiming to mislead or deceive the model.
 - Multihop reasoning, which refers to the ability of a model to connect multiple pieces of information or “hops” across the text to arrive at an answer.
- Question-Answering (QA) systems that automatically answer questions posed by humans in natural language, either based on a given context or dataset (known as closed-QA) or diverse topics from any domain (open-QA).
- Sentiment Analysis is the task of determining the sentiment or emotional tone expressed in a piece of text, such as positive, negative, or neutral.

33.3.2 10-K Market risk disclosures

We focus on Item 7A of the 10-K reports: *Quantitative and Qualitative Disclosures About Market Risk*. After retrieving and filtering disclosures from the SEC’s EDGAR database, only the largest firms with sufficiently long reports are retained. One representative document per sector is selected for summarization.

```
# Retrieve universe of stocks
beg, end = 20240101, 20240331
univ = crsp.get_universe(bd.endmo(beg, -1))

# lookup company names
comnam = crsp.build_lookup(source='permno', target='comnam', fillna="")
univ['comnam'] = comnam(univ.index)

# lookup sic codes from Compustat, and map to FF 10-sector code
sic = pstat.build_lookup(source='lpermno', target='sic', fillna=0)
industry = Series(sic[univ.index], index=univ.index)
industry = industry.where(industry > 0, univ['siccd'])
sectors = Sectoring(sql, scheme='codes10', fillna='') # supplement from crosswalk
univ['sector'] = sectors[industry]

# Load Disclosure about Market Risk text from 10-K's
item, form = 'qqr10K', '10-K'
rows = DataFrame(ed.open(form=form, item=item))
found = rows[rows['date'].between(beg, end)] \
    .drop_duplicates(subset=['permno'], keep='last') \
    .set_index('permno')

# Keep largest decile of stocks
found = found.loc[found.index.intersection(univ.index[univ['decile'] == 1])]

# Require minimum length of text
docs = {permno: ed[found.loc[permno, 'pathname']].lower()}
```

(continues on next page)

(continued from previous page)

```

for permno in found.index)
permnos = [permno for permno, doc in docs.items() if len(doc)>2000]
found = found.join(Series(docs, name='item').reindex(permnos), how='inner')
docs = univ.loc[found.index].groupby('sector').sample(1)

```

33.3.3 Generation

A LangChain pipeline is used to apply two models (DeepSeek-R1 via Ollama and GPT-4o-mini via OpenAI) to generate summaries. Model endpoints are configured with deterministic settings (temperature = 0). A prompt template and output parser are defined to extract core content, looping through each 10-K document. Summaries are generated and collected for analysis.

```
summary = {} # to collect generated summaries
```

Define Langchain input prompt template

```

from langchain_core.prompts import ChatPromptTemplate
prompt_template = """
{role}.
Please summarize this risk report in about 300 words in prose form:

{text}
"""
prompt = ChatPromptTemplate.from_template(prompt_template)

```

Select Langchain output parser

```

from langchain_core.output_parsers import StrOutputParser
parser = StrOutputParser()

```

```

def collect_summaries(model, role="You are a helpful AI assistant."):
    """Helper to iterate over companies and generate summaries of risk reports"""
    summ = {}
    for i, permno in enumerate(docs.index):
        print(f'==== {i+1}/{len(docs)} .', univ.loc[permno, 'comnam'], '====')
        chain = prompt | model | parser
        response = chain.invoke({"role": role, "text": found.loc[permno, 'item']})
        print("\n".join([textwrap.fill(s, width=80) for s in response.split("\n"))))
        print()
        summ[permno] = response.split('</think>')[-1] # remove model's "thinking"
    return summ

```

Generate summaries with DeepSeek-R1-14b model

```
summary[model_name] = collect_summaries(model)
```

```

===== 1/10. PACCAR INC =====
<think>
Okay, so I need to summarize this risk report into about 300 words. Let me read
through it carefully first.

```

(continues on next page)

(continued from previous page)

The report is about market risks and derivative instruments, focusing on interest rates, currencies, and commodities. It mentions that the figures are in millions. The company uses hedging programs to manage these risks, as described in Note P.

Starting with interest-rate risk: They measure this by estimating how a 100 basis point increase would affect fair values. In 2023, assets like cash equivalents and fixed rate loans show potential losses, while liabilities such as fixed rate term debt and swaps show gains. The total for 2023 is a loss of \$17.7 million, which is better than the previous year's \$1.1 million loss.

Next, currency risk: They hedge against several currencies like CAD, EUR, GBP, etc. A 10% unfavorable change in exchange rates would cause losses of \$259.7 million in 2023 and \$216.6 million in 2022. But these are offset by changes in the underlying exposures.

Then, commodity price risk: They use forward contracts to hedge against price fluctuations for production inputs. A 10% unfavorable change would lead to losses of \$3.3 million in 2023 and \$2.5 million in 2022, which are also offset by hedged exposures.

I should structure the summary to cover each risk category with their respective figures and explanations. Make sure it's concise, around 300 words, and flows well.

</think>

The risk report outlines a company's strategies and exposures related to market risks, including interest rates, currencies, and commodities, as of December 31, 2023, and 2022. The company employs hedging programs to mitigate these risks.

****Interest-Rate Risk:****

The company assesses the impact of a 100 basis point increase in interest rates on the fair value of its assets and liabilities. In 2023, cash equivalents and marketable debt securities showed potential losses of \$29.2 million, while fixed-rate loans indicated a \$146.5 million loss. On the liability side, fixed rate term debt and swaps resulted in gains of \$156.8 million and \$1.2 million, respectively. The net interest-rate risk for 2023 was a manageable \$17.7 million loss, an improvement from the previous year's \$1.1 million loss.

****Currency Risk:****

To manage exchange rate fluctuations, the company hedges against major currencies like the Canadian dollar and Euro. A 10% unfavorable change in these rates could lead to losses of \$259.7 million in 2023 and \$216.6 million in 2022. These potential losses are largely offset by corresponding changes in the value of hedged exposures, indicating effective risk management.

****Commodity Price Risk:****

The company uses forward contracts to stabilize earnings against adverse commodity price movements, particularly for truck production inputs. A 10% unfavorable change in commodity prices could result in \$3.3 million and \$2.5 million losses in 2023 and 2022, respectively. These figures reflect the company's ability to mitigate risks through hedging.

In summary, the company demonstrates a structured approach to managing market risks, with effective use of derivatives and hedging strategies to minimize potential financial impacts across interest rates, currencies, and commodities.

(continues on next page)

(continued from previous page)

===== 2/10. PHILLIPS 66 =====

<think>

Okay, so I'm trying to understand all these risks mentioned in the document. Let me start by reading through them carefully.

First, there are market conditions like fluctuations in prices and margins for NGLs, crude oil, natural gas, and refined products. That makes sense because energy prices can be really volatile due to things like supply and demand changes or geopolitical events.

Then there's government policies affecting pricing, regulation, taxation, especially exports. I know that export policies can have a big impact on supply and demand, so this is an important factor.

Capacity constraints in pipelines, storage, and fractionation facilities are another risk. If these infrastructure issues arise, it could limit how much product they can transport, leading to bottlenecks or higher costs.

OPEC and non-OPEC actions influence supply and demand, which affects prices. I remember that OPEC's decisions can cause significant shifts in the market.

The success of DCP LP integration is mentioned, including achieving synergies. This probably refers to a business strategy where they're combining operations, so if this doesn't go as planned, it could hurt their performance.

Unexpected technical difficulties or cost increases during construction or operation are risks too. Construction delays can be costly and disrupt production.

Drilling and production volumes around midstream assets are another point. If the wells aren't producing as expected, it affects the company's revenue from those assets.

Permits and regulations compliance are also risks. They need to get permits for projects, and if they can't or if regulations change, it could delay things or require more spending.

Savings and cost reductions from business transformation initiatives might not happen as planned. If they don't achieve these savings, their financial goals could be at risk.

Renewable fuels policies and climate change regulations are factors too. Changes in these areas could affect demand for traditional fuels or require new investments.

Economic and political developments like the Russia-Ukraine war can impact markets. Also, things like inflation, interest rates, and expropriation of assets pose risks.

Public health crises, like pandemics, can disrupt operations and reduce demand for their products. The recovery after such events is also uncertain.

Capital projects might not be completed on time or within budget. Delays here can lead to cost overruns and project failures.

(continues on next page)

(continued from previous page)

Asset dispositions or acquisitions could face challenges if they don't complete them successfully. This includes both the sale of assets and buying new ones.

Litigation or government actions could disrupt operations, leading to legal fees or operational changes.

Accidents, weather events, civil unrest, etc., can damage facilities and interrupt operations, causing financial losses.

Meeting sustainability goals is another risk. If they don't reduce GHG emissions as planned or develop new technologies, it could affect their reputation and operations.

New products might not be accepted by the market, leading to wasted investments.

Monetary conditions and exchange controls can impact international trade and profitability.

Environmental regulations requiring significant investments or reducing demand for their products are risks. They might have to spend a lot on compliance or face reduced sales.

Liability from environmental issues like cleanup costs is another concern.

Changes in laws and regulations, including alternative energy mandates, could force them to adapt their business models.

Climate change concerns might lead to changes in how they operate or increase expenses through litigation or other costs.

Impairment charges could occur if the value of assets decreases due to market conditions or strategic decisions.

Access to capital is crucial; if it's limited or more expensive, they might not be able to fund their operations effectively.

Counterparties going bankrupt could lead to financial losses from unpaid debts or obligations.

Cybersecurity incidents can compromise sensitive data and lead to legal liabilities or operational disruptions.

Joint ventures that the company doesn't control could make decisions that negatively impact their own business.

Lastly, all these factors are generally described in Item 1A of the risk factors in the report.

I think I've covered all the points. Now, how do I summarize this? Maybe by categorizing them into market risks, regulatory risks, operational risks, financial risks, and environmental/sustainability risks. That way, it's easier to understand and address each category separately.

</think>

The document outlines a comprehensive list of risks that could impact the company's performance and operations. These risks can be categorized as follows:

(continues on next page)

(continued from previous page)

1. **Market Risks**:**
 - Volatility in energy prices (NGLs, crude oil, natural gas, refined products).
 - Fluctuations in margins and demand for refining and petrochemical products.
 - Impact of OPEC and non-OPEC actions on supply and demand.
2. **Regulatory and Policy Risks**:**
 - Changes in government policies affecting pricing, regulation, taxation, and exports.
 - Compliance with environmental regulations (GHG emissions, renewable fuels).
 - Influence of international monetary conditions and exchange controls.
3. **Operational Risks**:**
 - Capacity constraints in transportation infrastructure (pipelines, storage).
 - Delays or cost overruns in construction and capital projects.
 - Challenges in permit acquisition and regulatory compliance.
 - Risks from joint ventures not under the company's control.
4. **Financial Risks**:**
 - Failure to achieve expected savings from business transformation initiatives.
 - Impairment charges due to asset value declines.
 - Limitations in accessing capital or increased funding costs.
 - Counterparty risks (e.g., bankruptcy leading to financial losses).
5. **Environmental and Sustainability Risks**:**
 - Meeting sustainability goals and adapting to climate change regulations.
 - Development of new technologies and products that may not gain market acceptance.
6. **Geopolitical and Public Health Risks**:**
 - Impact of geopolitical events (e.g., Russia-Ukraine war) on markets.
 - Disruptions from public health crises (e.g., pandemics).
7. **Legal and Reputational Risks**:**
 - Litigation risks and potential liabilities from environmental issues.
 - Cybersecurity threats leading to data breaches or operational disruptions.

By categorizing these risks, the company can better manage and mitigate potential challenges across different areas of its operations.

===== 3/10. MASTERCARD INC =====

<think>

Okay, so I need to summarize this risk report from MasterCard's 2023 Form 10-K about market risks, specifically items related to quantitative and qualitative disclosures. The user wants a summary in around 300 words, written in prose form.

First, I'll read through the provided text carefully to understand the key points. It seems like the report is divided into sections on foreign exchange risk and interest rate risk.

Under foreign exchange risk, MasterCard uses derivative contracts to manage currency exposures from transactions and asset/liability fluctuations. They mention a hypothetical 10% adverse change leading to potential losses of around

(continues on next page)

(continued from previous page)

\$414 million in 2023 and \$94 million in 2022. There's also something about short-term settlement activities limiting their exposure to a few days, so the impact isn't material even with a 10% change. Additionally, they talk about net investment hedges for foreign subsidiaries but didn't have any as of December 31, 2023.

For interest rate risk, MasterCard has available-for-sale debt investments and fixed-rate debt. They mention that a 100 basis point adverse change wouldn't cause material losses in either year. They use derivative contracts to hedge some of their fixed-rate debt exposure.

I need to structure this summary clearly, probably starting with an overview of market risk, then breaking down into foreign exchange and interest rate risks separately. I should highlight the strategies they use, like derivatives for hedging, and mention the hypothetical losses but also note that these are before considering offsets from hedged activities. It's important to convey that their exposure is managed effectively through these measures.

I'll make sure to keep it concise, around 300 words, using clear and professional language without getting too technical. I should avoid jargon where possible and ensure each section flows logically into the next.
</think>

The risk report outlines Mastercard's approach to managing market risks, including foreign exchange and interest rate risks, which can lead to financial losses due to adverse changes in currency values or interest rates. The company actively monitors these risks and employs strategies such as derivative financial instruments to mitigate potential impacts.

Foreign Exchange Risk:

Mastercard manages foreign exchange risk through derivative contracts aimed at hedging currency exposures from transactions and asset/liability fluctuations. These contracts help reduce transaction gains or losses resulting from currency fluctuations against the U.S. dollar and euro. A hypothetical 10% adverse change in currency values could result in fair value losses of approximately \$414 million (2023) and \$94 million (2022). However, Mastercard's short-term settlement activities typically limit exposure to a few days between transactions and settlements, minimizing material impacts from such changes. Additionally, the company may use derivatives to hedge its net investment in foreign subsidiaries, though no such hedges were designated as of December 31, 2023.

Interest Rate Risk:

The company's available-for-sale debt investments include fixed and variable rate securities sensitive to interest rate fluctuations. A hypothetical 100 basis point adverse change in interest rates would not result in material losses for its investments or derivative contracts used to hedge fixed-rate debt as of December 31, 2023 and 2022.

Overall, Mastercard employs robust risk management strategies, including the use of derivatives, to effectively mitigate market risks and safeguard against potential financial losses.

===== 4/10. BRISTOL MYERS SQUIBB CO =====

<think>

Okay, so I need to summarize this risk report into about 300 words. Let me read

(continues on next page)

(continued from previous page)

through it carefully first.

The report is about market risks the company faces, specifically from currency exchange rates and interest rates. They use derivatives like forwards and swaps to hedge these risks but don't trade with them.

Looking at foreign exchange risk: a lot of their revenue and cash flows are affected by changes in Euro and Japanese Yen. They use forward contracts and put options to manage this. There's also transaction risk from non-functional currencies, which they offset with forwards not designated as hedges. They hedge net investments with affiliates using these contracts too.

They estimate that a 10% appreciation in the hedged currencies would decrease the fair value of their FX contracts by certain amounts, reducing earnings. Cross-currency swaps are used for long-term debt and foreign subsidiaries; here, a 10% change affects the swap values differently.

For interest rate risk: they use fixed-to-floating swaps to balance their debt. They also analyze how changes in rates affect their marketable securities and long-term debt. A 1% increase in rates doesn't have a big impact on earnings but does decrease the fair value of long-term debt.

Credit risk is managed by investing only with high-quality institutions, diversifying derivatives across counterparties to reduce default risks.

I need to condense all this into a coherent summary without getting too technical. Make sure to mention each type of risk (FX and interest rate), how they hedge, the impact of currency appreciation/depreciation, and their credit risk management strategies.

</think>

The company faces market risks primarily from fluctuations in foreign exchange rates and interest rates. To mitigate these risks, it employs derivative financial instruments such as forwards, swaps, and put options, which are used for hedging rather than trading.

****Foreign Exchange Risk:****

A significant portion of the company's revenue, earnings, and cash flows are exposed to changes in Euro and Japanese Yen exchange rates. The company uses foreign currency forward contracts and purchased local currency put option contracts to manage risks from intercompany transactions. Additionally, these forwards are used to hedge net investment exposures in international affiliates. Cross-currency swaps are employed to manage long-term debt denominated in Euros.

The impact of a 10% appreciation in hedged currencies is estimated to decrease the fair value of foreign exchange contracts and increase/decrease cross-currency swap values, affecting earnings over time.

****Interest Rate Risk:****

The company uses fixed-to-floating interest rate swaps to balance its debt structure. Sensitivity analysis shows that a 1% increase in short-term or long-term interest rates would not significantly impact earnings but could decrease the fair value of long-term debt.

****Credit Risk Management:****

Investments are limited to high-credit-quality institutions, and derivatives are

(continues on next page)

(continued from previous page)

diversified across counterparties to mitigate default risks.

In summary, the company effectively manages market risks through strategic use of financial instruments while adhering to conservative credit policies.

===== 5/10. CARRIER GLOBAL CORP =====

<think>

Okay, so I need to summarize this risk report about market risks for a company. The user provided the entire section under item 7a, which includes foreign currency exposures, commodity price exposures, and interest rate exposures. They also mentioned that there's been no significant change in their market risk exposure for the year ended December 31, 2023.

First, I should read through each part carefully to understand what each section is talking about.

Starting with foreign currency exposures: The company operates internationally, so they're exposed to exchange rate fluctuations. Their non-U.S. operations have different functional currencies, which affects their reported results when the U.S. dollar strengthens or weakens against those currencies. They manage material currency exposures at the legal entity level but don't hedge translation risk. However, for specific acquisitions like VCS and TCC, they used financial instruments like window forward contracts and cross-currency swaps to mitigate risks related to paying in foreign currencies.

Next, commodity price exposures: The company is affected by volatile commodity prices and uses fixed-price contracts when appropriate. They also face fuel costs for shipping but don't have any hedges in place as of December 31, 2023.

Interest rate exposures: Most of their long-term debt has fixed rates, so they don't expect much impact from interest rate fluctuations.

I need to condense all this into about 300 words. I should structure it by first introducing the overall market risks (currency, commodities, interest rates), then go into each category with key points, and conclude without significant changes in exposure.

I should make sure to mention their strategies for managing these risks, like using swaps and contracts, but also note where they don't hedge, such as translation risk. I'll need to keep the language clear and concise, avoiding too much jargon so it's accessible.

Let me outline the summary:

1. Introduction: Briefly state that the company faces market risks from currency, commodities, and interest rates, with no significant change in 2023.
2. Foreign Currency Exposures: Explain their international operations leading to exchange rate risks. Mention active management at entity level but no hedging of translation risk. Highlight specific actions taken for VCS and TCC acquisitions using financial instruments.
3. Commodity Price Exposures: Discuss volatility in commodity prices, use of fixed-price contracts, and absence of hedges for fuel costs.
4. Interest Rate Exposures: Note that most debt is fixed-rate, so minimal expected impact from rate changes.
5. Conclusion: Reiterate no significant change in overall market risk exposure.

(continues on next page)

(continued from previous page)

I should ensure each section flows logically into the next, providing a comprehensive yet concise overview without getting bogged down in details. Also, I need to watch the word count to stay around 300 words, so I'll aim for about 6-7 sentences per main point.

</think>

The company faces market risks primarily from fluctuations in foreign currency exchange rates, commodity prices, and interest rates, with no significant changes in exposure during 2023.

****Foreign Currency Exposures:**** The company's global operations expose it to exchange rate risks, particularly affecting its U.S. dollar reporting. While they manage material currency exposures at the entity level without hedging translation risk, specific strategies were employed for acquisitions. For instance, window forward contracts were used for the VCS acquisition to mitigate euro-denominated risks, and cross-currency swaps were utilized for the TCC acquisition to hedge yen-denominated investments.

****Commodity Price Exposures:**** The company deals with volatile commodity prices by using fixed-price contracts where appropriate. However, they lack hedges against fuel cost fluctuations impacting shipping expenses.

****Interest Rate Exposures:**** With most long-term debt having fixed rates, the company anticipates minimal impact from interest rate changes.

In conclusion, the company's market risk management remains consistent, focusing on strategic hedging for specific risks while acknowledging areas where exposure is accepted without mitigation.

===== 6/10. LULULEMON ATHLETICA INC =====

<think>

Okay, so I need to summarize this risk report into about 300 words. Let me read through it carefully first.

The report starts with market risks, specifically foreign currency exchange and translation risks. It mentions that their international subsidiaries use local currencies as functional currencies, which are then converted to USD for consolidated financial statements. Exchange rate fluctuations affect reported revenues, expenses, assets, and liabilities. In 2023, revenue was lower by \$89.8 million compared to 2022 due to exchange rates.

They also talk about translation risk where foreign currency differences are recorded in other comprehensive income (OCI). They have a Canadian subsidiary with significant net assets, so they use forward contracts to hedge part of this exposure. The impact on OCI was an increased loss of \$9 million, including hedge gains.

Then there's transaction risk from intercompany transactions and inventory purchases in non-functional currencies. They hold cash and other monetary assets in different currencies and have forward contracts to hedge these. The sensitivity analysis shows that a 10% depreciation in the USD would affect derivative fair values by \$29.8 million, but this is offset by changes in hedged items.

They plan to use more derivatives in the future for additional currency pairs but won't trade them for profit. There's also mention of interest rate risk from

(continues on next page)

(continued from previous page)

their revolving credit facility, which has a variable rate. As of now, there are no borrowings except for \$6.3 million in letters of credit. They don't do any hedging yet but might in the future.

Cash and cash equivalents are held in short-term deposits and money market funds, totaling \$2.2 billion. Interest rates affect the interest generated on these balances. Credit risk is managed by dealing with reputable institutions, though some deposits exceed insured limits. They monitor counterparties' credit standing to minimize risks.

Inflation is another concern, particularly wage increases affecting operating margins in 2022 and 2023. Higher transportation costs and raw material prices could hurt margins if selling prices don't rise or cost efficiencies aren't found.

I need to condense all this into a coherent summary without exceeding 300 words. I'll structure it by main risk categories: foreign currency, interest rate, credit, and inflation risks. Highlight the key points like revenue impact, hedging strategies, sensitivity analysis, current financial status, and future plans regarding derivatives. Also, mention the effects of inflation on margins.

I should ensure that each section is concise, using bullet points or clear paragraphs to separate the different types of risk. Make sure to include the main figures like \$89.8 million revenue drop, \$2.2 billion cash, and sensitivity results. Avoid jargon where possible and keep it in prose form as requested.

Let me start drafting:

- Foreign currency exchange and translation risks: Explain how exchange rates affect financial statements, mention the 2023 revenue drop, their use of forward contracts to hedge, and the impact on OCI.
- Transaction risk: Discuss intercompany transactions and inventory purchases in non-functional currencies, their hedging strategies with forward contracts, and sensitivity analysis results.
- Interest rate risk: Describe the credit facility's variable rates, current borrowing status, and future plans for hedging if needed.
- Credit risk: Talk about holding cash in reputable institutions, monitoring counterparties, and minimal losses experienced.
- Inflation: Highlight past impacts on margins due to wages and freight costs, potential future effects, and reliance on price increases or cost efficiencies.

I need to make sure each section flows logically and stays within the word limit. Let me check for any redundancies or areas that can be more concise.
</think>

The risk report outlines several key market risks faced by the company, primarily focusing on foreign currency exchange, translation, transaction, interest rate, credit, and inflationary risks.

****Foreign Currency and Translation Risks:**** The company's international subsidiaries use local currencies, which are converted to USD for consolidated financial statements. Exchange rate fluctuations significantly impact reported

(continues on next page)

(continued from previous page)

revenues, expenses, assets, and liabilities. In 2023, revenue decreased by \$89.8 million due to unfavorable exchange rates. To mitigate translation risk, the company records foreign currency differences in other comprehensive income (OCI). They also use forward contracts to hedge risks associated with their significant Canadian subsidiary, resulting in a \$9 million increase in OCI losses, including hedge gains.

****Transaction Risks:**** The company faces transaction risks from intercompany transactions and inventory purchases denominated in non-functional currencies. To manage these risks, they hold cash and other monetary assets in various currencies and employ forward contracts for hedging. Sensitivity analysis indicates that a 10% USD depreciation would affect derivative fair values by \$29.8 million, offset by changes in hedged items.

****Interest Rate Risks:**** The company's revolving credit facility exposes it to variable interest rates. Currently, there are no borrowings except for \$6.3 million in letters of credit. While they do not engage in hedging activities at present, they plan to use derivatives for future currency pairs without trading for profit.

****Credit Risks:**** The company manages credit risks by holding cash in reputable financial institutions and monitoring counterparty credit standing. Although some deposits exceed insured limits, they have experienced minimal losses and maintain investment-grade relationships to mitigate risks.

****Inflation Risks:**** Inflationary pressures, such as rising wages and transportation costs, have impacted operating margins in recent years. Sustained increases could adversely affect margins if selling prices do not rise or cost efficiencies are not achieved. The company remains vigilant about these factors to ensure margin stability.

This structured approach ensures clarity and conciseness, effectively conveying the key risks and strategies outlined in the report within 300 words.

===== 7/10. AIRBNB INC =====

<think>
Okay, so I need to summarize this risk report about market risks that Airbnb is facing. The user provided the entire section from item 7a, which talks about both quantitative and qualitative disclosures regarding market risks. They specifically mentioned foreign currency risk and investment risk.

First, I should read through the entire text carefully to understand all the points made. It starts by saying that their global operations expose them to various market risks, mainly foreign currency and investment risks.

Under foreign currency exchange risk, they mention that they support over 40 currencies, with some key ones like euro, British pound, etc. Their revenue and costs are in these currencies, which can fluctuate against the US dollar, affecting their financial results. They have several exposures: revenue and cost of revenue, funds held, unbilled amounts, and intercompany balances.

They use foreign currency derivatives to hedge these risks but don't eliminate them entirely. There's a mention that if there was an adverse 10% change in exchange rates as of Dec 31, 2023, they would lose about \$20 million.

Then, investment and interest rate risk: they have a large cash and short-term

(continues on next page)

(continued from previous page)

investments portfolio, mainly in debt securities. They don't invest for trading or speculation, so their exposure to interest rate changes is limited because their investments are short-term. A 100 basis point increase would decrease their portfolio by \$20 million.

I need to condense all this into about 300 words. I should structure it with an introduction, then sections on foreign currency risk and investment risk, highlighting the key points like hedging strategies, potential losses from exchange rate changes, and their approach to managing interest rates through short-term investments.

I also need to make sure the summary is in prose form, clear and concise, without using bullet points or technical jargon. I should mention the specific currencies, the use of derivatives, the amounts at risk, and their investment strategies.

Wait, let me check if there are any other important details. They talk about how guests can pay in different currencies, which introduces more currency risks because hosts might choose a different payment currency. Also, they have separate funds for bookings, which adds to their foreign exchange exposures.

In the investment section, it's important to note that they focus on preserving capital and liquidity, so they stick to safe, short-term investments. This minimizes their exposure to interest rate fluctuations but still, a significant change could impact them.

I should also include the specific figures: \$20 million loss from a 10% exchange rate change and the same figure for a 100 basis point increase in interest rates. These numbers show the potential financial impact.

Putting it all together, I'll start with an overview of their market risks, then break down each type, explain the exposures, their hedging strategies, and the potential losses. Conclude by summarizing their approach to managing these risks.

</think>

Airbnb faces significant market risks primarily due to its global operations, which expose it to foreign currency exchange risk and investment risk.

Foreign Currency Exchange Risk:

Airbnb operates in over 40 currencies, with key exposures including the euro, British pound, Canadian dollar, Australian dollar, Brazilian real, and Mexican peso. Fluctuations in these currencies against the US dollar can impact its financial results through various channels: revenue and costs denominated in foreign currencies, funds held on behalf of customers, unbilled amounts from its "Pay Less Upfront" program, and intercompany balances. To mitigate this risk, Airbnb uses foreign currency derivative contracts to hedge forecasted exposures, though it does not eliminate all risks. A 10% adverse exchange rate change as of December 31, 2023, could result in a \$20 million loss.

Investment and Interest Rate Risk:

Airbnb's investment portfolio, primarily consisting of short-term, highly-liquid debt securities, is managed to preserve capital and meet liquidity needs. As of December 31, 2023, it held \$6.9 billion in cash and cash equivalents, \$3.2 billion in short-term investments, and \$5.9 billion in funds receivable. The portfolio's short maturity minimizes sensitivity to interest rate changes. A

(continues on next page)

(continued from previous page)

hypothetical 100 basis point increase in rates could decrease the investment portfolio's fair value by \$20 million. Airbnb avoids speculative investments and trading activities, focusing on stability rather than risk.

In summary, while Airbnb employs hedging strategies and conservative investment practices to manage risks, significant currency fluctuations or interest rate changes could still impact its financial performance.

===== 8/10. MERCADOLIBRE INC =====

<think>

Okay, so I'm trying to understand this document about MercadoLibre's risks and sensitivities. It seems like it's part of their financial disclosures, maybe in their annual report or something similar. Let me break down what each section is saying.

First, the foreign exchange risk. They have a significant amount of revenue from Brazil, Argentina, and other Latin American countries. Since they're dealing with multiple currencies, especially the Brazilian real and Argentine peso, which can be volatile, this could affect their financials. If the real or peso weakens against the dollar, their reported revenue in dollars might decrease because when converted, those currencies would buy fewer dollars. That makes sense because if you have a lot of revenue in a currency that's dropping, your overall revenue in USD terms goes down.

Next is interest rate risk. They have long-term debt, which means they're exposed to changes in interest rates. If rates go up, the value of their existing debt might decrease, and their borrowing costs could increase if they need to refinance or take on more debt. I'm not entirely sure how sensitive they are to rate changes, but it's something to watch, especially with global interest rates potentially rising.

Then there's commodity price risk. They're exposed to fuel and electricity prices because these are operational costs. If oil prices go up, their delivery and logistics costs would increase, affecting their margins. Similarly, higher electricity prices could impact their operations, especially in countries where energy is a significant cost. This seems like a manageable risk but could have noticeable effects if prices spike.

Equity price risk is tied to their LTRP plans. These are long-term retention programs that pay out based on stock performance. The value of these obligations is sensitive to the company's stock price. If the stock goes up, the payouts increase, which could strain their finances. Conversely, a drop in stock price reduces the payout liability. They've done a sensitivity analysis showing how a 40% change in stock price affects the obligation. That's pretty significant; a 40% drop would reduce the liability from \$418 million to about \$251 million, which is a substantial difference.

Inflation risk is another factor, especially in countries like Argentina where inflation has been high. High inflation can erode purchasing power and affect consumer behavior. If people have less money due to inflation, they might spend less online, impacting MercadoLibre's sales. Additionally, rising costs for goods and services could squeeze margins if the company can't pass these costs onto customers.

I'm a bit confused about how exactly the equity price affects their obligations. It seems like it's based on the stock price relative to a fixed average from

(continues on next page)

(continued from previous page)

previous years. So each payment is tied to whether the current stock price is higher or lower than that average. If the stock does well, they have to pay more; if not, less.

I'm also thinking about how these risks interact. For example, inflation could lead to higher interest rates, which would then affect their debt costs. Or a weaker currency might make imports more expensive, contributing to inflation. It's like a web of interconnected risks that could amplify each other in certain economic conditions.

Another point is the sensitivity analysis for equity price risk. They show a static present value and how it changes with different stock prices. This helps investors understand how exposed they are to stock price fluctuations. A 40% change seems extreme, but it's useful to see the range of possible outcomes.

I wonder if there are any mitigating factors or strategies MercadoLibre uses to manage these risks. For example, do they hedge their currency exposure? How flexible are their costs in the face of inflation? And how do they structure their LTRP plans to balance retention with financial risk?

Overall, this section is highlighting several key areas where MercadoLibre faces potential volatility and challenges. It's important for investors to consider these risks when evaluating the company's long-term prospects.

</think>

MercadoLibre Risk Analysis Summary

1. **Foreign Exchange Risk:**

- **Exposure:** Significant revenue from Brazil, Argentina, and other Latin American countries exposed to currency fluctuations, particularly the Brazilian real and Argentine peso.
- **Impact:** Currency devaluation can reduce reported USD revenue when converted.

2. **Interest Rate Risk:**

- **Exposure:** Long-term debt subject to interest rate changes.
- **Impact:** Rising rates may decrease debt value and increase borrowing costs.

3. **Commodity Price Risk:**

- **Exposure:** Operational costs tied to fuel and electricity prices.
- **Impact:** Price spikes can increase operational expenses and reduce margins.

4. **Equity Price Risk:**

- **Exposure:** Long-term retention programs (LTRP) linked to stock performance.
- **Impact:** Stock price fluctuations affect payout obligations; a 40% drop reduces liability from \$418 million to \$251 million.

5. **Inflation Risk:**

- **Exposure:** High inflation in countries like Argentina affecting purchasing power and operational costs.
- **Impact:** Could reduce consumer spending and squeeze margins if cost increases aren't offset.

(continues on next page)

(continued from previous page)

6. ****Interconnected Risks:****

- Inflation may lead to higher interest rates, affecting debt costs and potentially exacerbating currency issues.

7. ****Mitigation Strategies:****

- Consider hedging strategies for currency exposure, flexible cost structures, and balanced LTRP designs to manage financial risks.

****Conclusion:**** MercadoLibre faces multiple interconnected risks that could impact its financial health. Investors should assess these factors alongside the company's risk management strategies when evaluating long-term prospects.

===== 9/10. A T & T INC =====

<think>

Okay, so I'm trying to understand this document about AT&T's financial report. It looks like it's an audit report from their independent registered public accounting firm. Let me go through it step by step.

First, the report is divided into two main sections: one on the internal control over financial reporting and another on the consolidated financial statements. Both have an opinion section, a basis for opinion, and some additional details.

Starting with the opinion on internal control over financial reporting. The auditors state that they've audited AT&T's internal controls as of December 31, 2023, based on the COSO criteria. They conclude that the company maintained effective internal control over financial reporting in all material respects. That sounds positive; it means the auditors think AT&T has good systems in place to manage their finances properly.

Next, looking at the basis for this opinion. It mentions that management is responsible for maintaining these controls and assessing their effectiveness. The auditors' role was to express an opinion based on their audit, which they did by following PCAOB standards. They obtained an understanding of the internal controls, assessed risks, tested the design and operation of the controls, and performed other necessary procedures. Their conclusion is that AT&T's controls are effective.

Moving on to the report on the consolidated financial statements. The auditors also express an unqualified opinion here, meaning they didn't find any material issues with the financial statements. They were prepared in accordance with GAAP, which is standard for public companies in the U.S.

The basis for this opinion section explains that the auditors are responsible for expressing opinions on whether the financial statements present fairly, in all material respects, the company's financial position and results of operations. They did this by auditing according to PCAOB standards, assessing risks, testing accounting policies, significant estimates, transactions, and disclosures.

There are also two specific sections discussing areas where the audit was particularly complex: internal control over financial reporting and the impairment evaluations for certain reporting units. The auditors explain that these areas required a deeper dive because they involve subjective assumptions and complex methodologies. They tested controls, assessed assumptions against industry trends, performed sensitivity analyses, and involved specialists to ensure everything was in order.

(continues on next page)

(continued from previous page)

The report ends with a note that the auditing firm has been serving AT&T since 1999, which shows a long-standing relationship, likely indicating consistency and familiarity with the company's operations.

I'm trying to think if there's anything I might be missing. The document seems comprehensive, covering both the internal controls and financial statements. It mentions specific areas like impairment evaluations and valuation models, which are known for being complex and subject to judgment. The auditors' involvement of specialists in these areas makes sense because it adds depth to their analysis.

I wonder if there were any material weaknesses or significant deficiencies mentioned, but from what I can see, the opinions are unqualified, so no major issues were found. It's also worth noting that the report is dated February 23, 2024, which would typically be around the time when earnings are released, so it's timely.

Overall, this audit report seems to provide assurance to stakeholders that AT&T's financial reporting is reliable and their internal controls are effective. The detailed explanation of the auditing process gives confidence in the independence and thoroughness of the audit.

</think>

The audit report on AT&T provides a comprehensive assessment of both its internal control over financial reporting and its consolidated financial statements, concluding with an unqualified opinion in both areas. Here's a structured summary:

Key Findings:

1. **Internal Control Over Financial Reporting:**

- The auditors found that AT&T maintained effective internal controls as of December 31, 2023, based on the COSO framework.
- The company's systems provide reasonable assurance regarding the reliability of financial reporting and compliance with GAAP.

2. **Consolidated Financial Statements:**

- The financial statements were prepared in accordance with GAAP and present AT&T's financial position accurately.
- No material issues were identified, leading to an unqualified opinion.

Methodology:

- Auditors followed PCAOB standards, assessing risks, testing controls, and involving specialists for complex areas like impairment evaluations.
- Detailed procedures included testing assumptions against industry trends and performing sensitivity analyses.

Areas of Complexity:

- **Impairment Evaluations:** Involved subjective assumptions and complex valuation methods. The auditors ensured these were sound through thorough testing and specialist involvement.
- **Internal Controls:** The effectiveness was validated through a comprehensive audit process, confirming the controls' operational efficiency.

Conclusion:

The report assures stakeholders of AT&T's financial integrity and control

(continues on next page)

(continued from previous page)

systems. The unqualified opinions indicate no significant issues, reflecting well on the company's financial governance. The long-standing relationship with the auditing firm underscores consistency and familiarity in their work.

===== 10/10. REPUBLIC SERVICES INC =====

<think>

Okay, so I need to summarize this risk report into about 300 words. Let me read through it carefully first.

The report starts with Item 7A, which is about quantitative and qualitative disclosures on market risks. The main areas discussed are interest rate risk, fuel price risk, and commodities price risk.

Starting with interest rate risk: The company's major exposure is to changes in U.S. interest rates and fluctuations in SOFR (which I think stands for Secured Overnight Financing Rate). They manage this by using a mix of fixed and floating rate debt. Their variable rate debt is valued at fair value because the rates are current, but this can change over time. They've used swap agreements as hedges before, but it looks like they might not be doing that anymore or maybe have changed their strategy.

Looking at the table provided, it shows fixed and variable rate debts across different years up to 2028. The total fair value is around \$9.8 billion. They also mention having \$2.23 billion in floating rate debt and swap contracts with a notional value of \$350 million. If interest rates go up or down by 100 basis points, their annual interest expense would change by about \$20 million. But this doesn't account for new borrowings or economic impacts.

Next is fuel price risk: Fuel costs are a big part of their operations. They try to hedge against fuel price changes when it's economically practical, but as of December 31, 2023, they had no hedges in place. They charge fuel recovery fees to most customers, but not all. A 20-cent per gallon change in diesel would affect their fuel costs by about \$27 million and the fees by \$36 million. They also mention other petrochemical-based products used in operations, which could increase costs if prices go up.

Lastly, commodities price risk: They deal with recycled materials like old corrugated containers and newsprint. Market volatility affects these prices. In the past, they've used swaps and collars as hedges, but none are currently in place. A \$10 per ton change would impact both revenue and operating income by about \$10 million. Their recycling revenue dropped from \$359.1 million in 2022 to \$312.3 million in 2023.

Putting this all together, the summary should cover each risk area with key points: how they're managed, current positions (like hedges), and potential financial impacts. I need to make sure it's concise, around 300 words, and flows well.

</think>

The risk report outlines the company's exposure to various market risks, including interest rates, fuel prices, and commodities.

Interest Rate Risk: The company manages this through a mix of fixed and floating rate debt, with variable rate debt valued at fair value. As of December 31, 2023, their total debt was approximately \$9.8 billion, with \$2.23 billion in floating rate debt and swap contracts. A 100 basis point interest rate change

(continues on next page)

(continued from previous page)

could alter annual interest expenses by about \$20 million.

****Fuel Price Risk:**** Fuel costs are significant, and while the company charges recovery fees to most customers, not all are covered. A 20-cent per gallon price change in diesel would affect fuel costs by \$27 million and fees by \$36 million. Additionally, petrochemical-based product costs may rise with fuel prices.

****Commodities Price Risk:**** The company markets recycled materials, which face market volatility. Despite past hedging strategies, no hedges were active as of the report date. A \$10 per ton price change could impact revenue and operating income by \$10 million each. Recycling revenue decreased from \$359.1 million in 2022 to \$312.3 million in 2023.

In summary, the company faces notable risks from fluctuating interest rates, fuel prices, and commodities markets, with strategies in place to mitigate some of these impacts.

Show ollama processes

```
!ollama ps
```

NAME	ID	SIZE	PROCESSOR	UNTIL
deepseek-r1:14b	ea35dfe18182	11 GB	100% GPU	4 minutes from now

Generate summaries with OpenAI GPT-4o-mini model

```
summary[gpt_name] = collect_summaries(gpt_model)
```

===== 1/10. PACCAR INC =====

The risk report outlines the company's exposure to market risks, specifically focusing on interest rate, currency, and commodity price risks, with figures presented in millions.

In terms of interest rate risks, the company employs hedging programs to mitigate exposure to fluctuations. The report quantifies the potential impact of a 100 basis point increase in interest rates on the fair value of interest-sensitive assets and liabilities. For 2023, the fair value losses for cash equivalents and marketable debt securities amounted to \$29.2 million, while fixed-rate loans reflected a loss of \$146.5 million. Conversely, fixed-rate term debt showed gains of \$156.8 million, and interest-rate swaps contributed a gain of \$1.2 million, resulting in a total net loss of \$17.7 million for the year, compared to a loss of \$1.1 million in 2022.

Regarding currency risks, the company utilizes foreign currency exchange contracts to hedge against fluctuations in various currencies, including the Canadian dollar and the euro. The sensitivity analysis indicates a potential loss of \$259.7 million from a 10% unfavorable change in exchange rates as of December 31, 2023, an increase from a loss of \$216.6 million in 2022. However, these losses are expected to be offset by corresponding changes in the values of the underlying hedged exposures.

Lastly, the report addresses commodity price risks, where the company engages in commodity forward contracts to stabilize earnings and cash flows against adverse price movements. The potential loss in fair value from a 10% unfavorable change

(continues on next page)

(continued from previous page)

in commodity prices is estimated at \$3.3 million for 2023, up from \$2.5 million in 2022, with similar offsetting effects from the underlying hedged exposures. Overall, the report highlights the company's proactive approach to managing market risks through various hedging strategies.

===== 2/10. PHILLIPS 66 =====

The risk report outlines the market risks faced by the company and its subsidiaries, primarily stemming from fluctuations in commodity prices, interest rates, and foreign currency exchange rates. The company is particularly exposed to the prices of crude oil, refined petroleum products, natural gas liquids (NGL), natural gas, renewable feedstock, and electric power. To manage these risks, the company employs derivative contracts, including futures, forwards, swaps, and options, which help convert fixed-price contracts to floating market prices and optimize supply chain value.

The report emphasizes the company's policy of remaining exposed to market prices while using derivatives to balance physical systems, meet refinery requirements, and manage cash flow risks associated with price fluctuations. A Value at Risk (VaR) model is utilized to estimate potential losses from adverse market changes, indicating that the VaR for derivative instruments as of December 31, 2023, was immaterial to cash flows and operations.

Interest rate risk is another significant concern, as the company holds both fixed-rate and variable-rate debt. Fixed-rate debt can lead to changes in fair value due to market interest rate fluctuations, while variable-rate debt exposes the company to short-term interest expense changes. The report provides detailed tables of the company's debt instruments, highlighting their sensitivity to interest rate changes.

Additionally, the company faces foreign currency risk from its international operations but generally does not hedge this exposure. Risk monitoring is overseen by the CEO and CFO, ensuring that risks related to commodity prices, interest rates, and foreign exchange rates are effectively managed. The report concludes with a cautionary note regarding forward-looking statements, emphasizing the uncertainties and risks that could impact future performance, including market conditions, regulatory changes, and geopolitical events.

===== 3/10. MASTERCARD INC =====

The risk report outlines the company's exposure to market risk, specifically focusing on interest rate and foreign currency exchange rate fluctuations. Market risk refers to potential economic losses from adverse changes in these factors. The company has limited exposure to such risks, and management actively monitors and implements policies to govern funding, investments, and the use of derivative financial instruments to mitigate these risks.

To manage foreign currency risk, the company utilizes foreign exchange derivative contracts to hedge against anticipated receipts and disbursements in currencies other than its functional currency. This strategy aims to minimize transaction gains and losses due to currency fluctuations, particularly against the U.S. dollar and euro. A hypothetical 10% adverse change in the value of functional currencies could lead to significant fair value losses on outstanding foreign exchange derivatives, estimated at approximately \$414 million and \$94 million for the years ending December 31, 2023, and 2022, respectively, before considering any offsetting effects.

Additionally, the company faces foreign exchange risk from daily settlement

(continues on next page)

(continued from previous page)

activities, which it manages through short-duration derivative contracts. However, a similar hypothetical 10% adverse change would not materially impact the fair value of these contracts. The company also has exposure related to the translation of net investments in foreign subsidiaries, although as of December 31, 2023, it had no designated net investment hedges.

Regarding interest rate risk, the company holds both fixed and variable-rate securities. It maintains a policy of investing in high-quality securities while ensuring liquidity and diversification. A hypothetical adverse change of 100 basis points in interest rates would not significantly affect the fair value of these investments or the company's interest rate derivative contracts related to fixed-rate debt. Overall, the company employs various strategies to manage its market risk effectively.

===== 4/10. BRISTOL MYERS SQUIBB CO =====

The risk report outlines the company's exposure to market risks, particularly from fluctuations in currency exchange rates and interest rates. To mitigate these risks, the company employs various derivative financial instruments, although these are not used for trading purposes. The report highlights significant foreign exchange risks, particularly related to the euro and Japanese yen, which affect the company's revenues, earnings, and cash flow. To manage these risks, the company utilizes foreign currency forward contracts and purchased local currency put options, primarily for intercompany transactions. Additionally, these contracts help hedge against foreign currency exposures related to net investments in international affiliates.

The report estimates that a 10% appreciation in the currencies being hedged against the U.S. dollar would lead to a decrease in the fair value of foreign exchange contracts by \$409 million and \$782 million as of December 31, 2023, and 2022, respectively. Conversely, cross-currency swap contracts, which are used to manage risks from long-term debt in euros, would see an increase in fair value by \$46 million in 2023, while decreasing by \$73 million in 2022 under similar currency appreciation scenarios.

Regarding interest rate risk, the company employs fixed-to-floating interest rate swap contracts to balance its debt portfolio. A sensitivity analysis indicates that a 1% increase in interest rates would not materially impact earnings. However, it is estimated that such an increase would decrease the fair value of long-term debt by \$3.0 billion in 2023 and \$2.6 billion in 2022.

Lastly, the report addresses credit risk associated with counterparties in derivative transactions. The company maintains a strict investment policy to minimize credit risk, ensuring that investments are made only with high-quality institutions and diversifying counterparties to mitigate potential defaults. For further details, the report refers to additional financial statements and supplementary data.

===== 5/10. CARRIER GLOBAL CORP =====

The risk report outlines the company's exposure to market risks, including fluctuations in foreign currency exchange rates, interest rates, and commodity prices, which could affect its financial performance. As of December 31, 2023, there has been no significant change in the company's exposure to these market risks.

In terms of foreign currency exposure, the company operates globally, which subjects it to exchange rate fluctuations relative to its reporting currency,

(continues on next page)

(continued from previous page)

the U.S. dollar. Many of its international operations use currencies other than the U.S. dollar, meaning that the company's reported results can vary based on the strength or weakness of the dollar against these currencies. While the company actively manages material currency exposures related to transactions at the legal entity level, it does not hedge against currency translation risk.

The report highlights specific transactions, such as the acquisition of the VCS business, where 80% of the euro-denominated purchase price was paid in cash, exposing the company to exchange rate risks. To mitigate this risk, the company utilized window forward contracts, with changes in their fair value reflected in other income or expense. Similarly, for the TCC acquisition, the company employed cross currency swaps and a Japanese term loan facility to hedge against foreign currency translation risks associated with its investments in subsidiaries operating in yen.

Regarding commodity price exposure, the company faces volatility in the prices of certain commodities and shipping fuel costs. While it uses fixed-price contracts to manage some of this exposure, it currently does not have any commodity hedge contracts in place. Lastly, the report notes that most of the company's long-term debt carries fixed interest rates, insulating it from significant impacts due to fluctuations in market interest rates.

===== 6/10. LULULEMON ATHLETICA INC =====

The risk report outlines various market risks faced by the company, focusing on foreign currency exchange risk, interest rate risk, credit risk, and inflation.

Foreign currency exchange risk is primarily associated with the translation of financial statements from local currencies of international subsidiaries into U.S. dollars. In 2023, fluctuations in exchange rates resulted in a revenue decrease of \$89.8 million compared to 2022. The company records foreign currency translation differences as other comprehensive income (loss) within stockholders' equity. A significant portion of net assets is held in Canadian dollars, and the company uses forward currency contracts to hedge against translation exposure. The translation of Canadian subsidiaries contributed to an increase in other comprehensive loss of \$9 million, despite net investment hedge gains. Additionally, transaction risk arises from intercompany transactions and inventory purchases in currencies other than the subsidiaries' functional currencies. As of January 28, 2024, the company had forward currency contracts to hedge against foreign currency revaluation gains and losses.

Interest rate risk is linked to the company's revolving credit facility, which has a variable interest rate. As of January 28, 2024, there were no borrowings under this facility, but the company may consider using derivative financial instruments in the future to mitigate potential losses if a significant balance arises.

Credit risk is minimal, as the company holds cash with reputable financial institutions and invests in AAA-rated money market funds. The company actively monitors the creditworthiness of its counterparties to limit exposure.

Lastly, inflation poses a risk to operating results, particularly due to rising costs in wages, transportation, and raw materials. Increased costs may adversely affect operating margins if selling prices do not adjust accordingly. The report emphasizes the importance of managing these risks to maintain financial stability.

(continues on next page)

(continued from previous page)

===== 7/10. AIRBNB INC =====

The risk report outlines the market risks faced by the company, primarily focusing on foreign currency risk and investment risk due to its extensive global operations. In 2023, the company conducted transactions in over 40 currencies, with significant exposure to the euro, British pound, Canadian dollar, Australian dollar, Brazilian real, and Mexican peso. This exposure arises from international revenue and expenses, which are subject to fluctuations in foreign currency exchange rates against the U.S. dollar. Consequently, a strengthening U.S. dollar can negatively impact financial results, while a weakening dollar can be beneficial.

The company faces foreign currency risks related to various aspects, including revenue from bookings in foreign currencies, funds receivable and payable, and intercompany balances. To mitigate these risks, the company employs foreign currency derivative contracts aimed at managing forecasted foreign-denominated revenue and other related balances. However, these hedges do not completely eliminate the impact of currency fluctuations, and the company may opt not to hedge certain exposures due to economic or accounting considerations. A hypothetical adverse change of 10% in foreign currency exchange rates could lead to a loss of approximately \$20 million.

Additionally, the report addresses investment and interest rate risk, particularly concerning the company's investment portfolio. As of December 31, 2023, the company held \$6.9 billion in cash and cash equivalents and \$3.2 billion in short-term investments, primarily in high-quality debt securities. The company aims to preserve capital and maintain liquidity without significantly increasing risk, avoiding speculative investments. Due to the short maturities of its investments, the portfolio is relatively insensitive to interest rate changes, with a potential \$20 million decrease in value anticipated from a hypothetical 100 basis point increase in interest rates.

===== 8/10. MERCADOLIBRE INC =====

The risk report outlines the market risks faced by the company, primarily stemming from macroeconomic instability and fluctuations in interest rates and foreign currency exchange rates, particularly with the Brazilian real, Argentine peso, and Mexican peso. These factors can significantly impact the value of the company's financial assets and liabilities. The company also faces risks related to its long-term retention programs (LTRPs), which involve cash payments to employees that vary based on the market price of its stock.

With substantial international operations, the company is exposed to foreign currency risks that can adversely affect its financial results. It engages in transactions in various foreign currencies and charges its international subsidiaries for the use of intellectual property and corporate services. To mitigate these risks, the company employs foreign currency exchange forward contracts and currency swaps, although these hedges do not completely eliminate the impact of currency fluctuations.

As of December 31, 2023, the company reported significant cash and cash equivalents, receivables, and investments in foreign currencies, totaling over \$12 billion. The report highlights a consolidated loss of \$615 million due to foreign currency fluctuations, particularly in Argentina, where government restrictions on accessing U.S. dollars have exacerbated losses.

Interest rate changes also pose risks to the company's earnings and cash flows, affecting the cost of financing and the returns on investments. The report notes

(continues on next page)

(continued from previous page)

that a hypothetical increase in interest rates could lead to increased financial liabilities.

Additionally, the company's LTRPs expose it to equity price risk, with a total contractual obligation fair value of \$418 million as of December 31, 2023. The report includes sensitivity analyses showing how changes in equity prices could impact the company's financial obligations related to these programs. Overall, the report emphasizes the complexities and potential financial impacts of market risks on the company's operations.

===== 9/10. A T & T INC =====

The risk report outlines AT&T Inc.'s exposure to market risks, primarily from fluctuations in interest rates and foreign currency exchange rates, which affect its cost of capital. The company employs a strategic approach to manage these risks, utilizing derivatives such as interest rate swaps, locks, and cross-currency swaps, strictly for hedging purposes rather than speculative trading. The report indicates that there are no anticipated changes to these risk management strategies in the near future.

A significant factor in estimating postretirement benefit obligations is the weighted-average discount rate, which has seen increased volatility and is currently lower than historical averages. This results in higher obligations for the company, although future increases in discount rates could lead to lower obligations and improved funded status.

Interest rate risk is managed through a mix of fixed- and floating-rate debt, with the majority of financial instruments being medium- to long-term fixed-rate notes. The company has established limits on interest rate risk and closely monitors its debt and derivatives portfolios. As of December 31, 2023, AT&T had no interest rate locks but utilized cross-currency swaps to mitigate risks associated with foreign-denominated debt.

Foreign exchange risk is addressed through contracts that hedge costs and debt in foreign currencies. The report notes that AT&T's foreign-denominated debt has been converted to fixed-rate U.S. dollars, effectively eliminating associated risks. A sensitivity analysis is employed to assess the impact of market risk exposures on the fair value of financial instruments.

Overall, the report emphasizes AT&T's commitment to maintaining financial flexibility and managing risks effectively through established policies and procedures, ensuring the integrity of its financial reporting and internal controls.

===== 10/10. REPUBLIC SERVICES INC =====

The risk report outlines the company's exposure to market risks, particularly focusing on interest rate risk, fuel price risk, and commodities price risk.

In terms of interest rate risk, the company is primarily affected by fluctuations in U.S. interest rates and the Secured Overnight Financing Rate (SOFR). To manage this risk, the company employs a mix of fixed and floating rate debt. As of December 31, 2023, the carrying value of its variable rate debt is close to its fair value, reflecting current market conditions. The company has also utilized interest rate swap agreements as cash flow hedges to mitigate the impact of interest rate fluctuations on its variable rate debt. The report indicates that a 100 basis point change in interest rates could alter annual interest expenses by approximately \$20 million.

(continues on next page)

(continued from previous page)

Regarding fuel price risk, fuel costs are a significant operational expense for the company. Although it charges fuel recovery fees to most customers, it cannot do so universally. As of the end of 2023, the company had no fuel hedges in place. A 20-cent per gallon change in diesel fuel prices could affect fuel costs by about \$27 million annually, while the corresponding change in fuel recovery fees could be around \$36 million.

Lastly, the report addresses commodities price risk, particularly concerning the marketing of recycled materials. The company has experienced volatility in commodity prices due to market supply and demand fluctuations. As of December 31, 2023, it had no hedges in place for recycling commodities. A \$10 per ton change in recycled commodity prices could impact annual revenue and operating income by approximately \$10 million. Revenue from recycling activities decreased from \$359.1 million in 2022 to \$312.3 million in 2023. Overall, the report highlights the company's proactive approach to managing these market risks while acknowledging the inherent uncertainties.

33.3.4 Evaluation

ROUGE

Recall-Oriented Understudy for Gisting Evaluation (ROUGE) is a set of metrics used to evaluate the quality of summaries by comparing them to reference summaries or human-generated summaries.

- ROUGE-N measures the overlap of n-grams (contiguous sequences of n words) between the system-generated and the reference summaries
- ROUGE-L measures the longest common subsequence (LCS).

BLEU Bilingual Evaluation Understudy (BLEU) evaluates n-gram precision with a brevity penalty to discourage overly short outputs. Originally for machine translation, it is also used for summarization.

- N-gram Precision measures the overlap of n-grams (typically up to 4-grams) between the system-generated summary and the reference summary.
- Brevity Penalty penalizes overly short summaries that do not capture enough information from the reference summaries.
- Cumulative BLEU calculates the geometric mean of BLEU scores for 1-gram to n-gram, rewarding systems that produce more accurate translations across longer phrases.

```
# These metrics compare overlapping n-grams to measure content similarity.
from rouge_score import rouge_scorer
```

```
# computes ROUGE-1 and ROUGE-2 scores between model-generated summaries
def collect_rouge(target, prediction):
    """Helper to loop over companies to compute rouge scores of two risk summaries"""
    scores = {'rouge1': [], 'rouge2': []}
    scorer = rouge_scorer.RougeScorer(scores.keys(), use_stemmer=True)
    for permno in docs.index:
        score = scorer.score(target=target[permno], prediction=prediction[permno])
        for rouge_type in scores.keys():
            scores[rouge_type].append(Series(score[rouge_type]._asdict(),
                                              name=univ.loc[permno, 'comnam']))
    return scores
```

```
# Display and compare rouge metric
def display_rouge(rouge_type, scores):
    """Helper to display rouge scores over the companies"""
    df = pd.concat(scores[rouge_type], axis=1)
    print(f"{'rouge_type.upper()':>15} metric:")
    return pd.concat([df, df.T.mean().rename('average')], axis=1).T # display
```

```
# Compute rouge-1 and rouge-2 scores between gpt- and llama-generated summaries
scores = collect_rouge(target=summary[gpt_name], prediction=summary[model_name])
```

```
display_rouge("rouge1", scores)
```

ROUGE1 metric:

	precision	recall	fmeasure
PACCAR INC	0.787234	0.742475	0.764200
PHILLIPS 66	0.366142	0.324042	0.343808
MASTERCARD INC	0.759184	0.628378	0.687616
BRISTOL MYERS SQUIBB CO	0.773913	0.585526	0.666667
CARRIER GLOBAL CORP	0.761628	0.451724	0.567100
LULULEMON ATHLETICA INC	0.560241	0.636986	0.596154
AIRBNB INC	0.720472	0.639860	0.677778
MERCADOLIBRE INC	0.467593	0.331148	0.387716
A T & T INC	0.393805	0.312281	0.348337
REPUBLIC SERVICES INC	0.783410	0.553746	0.648855
average	0.637362	0.520617	0.568823

```
display_rouge("rouge2", scores)
```

ROUGE2 metric:

	precision	recall	fmeasure
PACCAR INC	0.483986	0.456376	0.469775
PHILLIPS 66	0.079051	0.069930	0.074212
MASTERCARD INC	0.381148	0.315254	0.345083
BRISTOL MYERS SQUIBB CO	0.441048	0.333333	0.379699
CARRIER GLOBAL CORP	0.350877	0.207612	0.260870
LULULEMON ATHLETICA INC	0.244713	0.278351	0.260450
AIRBNB INC	0.367589	0.326316	0.345725
MERCADOLIBRE INC	0.116279	0.082237	0.096339
A T & T INC	0.097778	0.077465	0.086444
REPUBLIC SERVICES INC	0.462963	0.326797	0.383142
average	0.302543	0.247367	0.270174

33.3.5 Role prompting

By adjusting the system prompt (e.g., “You are a patient teacher”), LLMs can be guided to produce more accessible summaries. This technique, known as **role prompting**, is helpful for tailoring the tone and persona of responses for specific audiences.

```
# generates simplified summaries for readability
summary['simple_deepseek'] = collect_summaries(
    model,
    role="You are a patient lower-school teacher, using simple words to explain to
    ↪your students in the fifth grade.")
```

```
===== 1/10. PACCAR INC =====
<think>
Okay, so I need to summarize this risk report for fifth graders. Let me read
through it carefully first.
```

The report talks about market risks and derivative instruments. It mentions interest-rate risks, currency risks, and commodity price risks. Each section has some numbers and explanations.

Starting with interest-rate risks: The company uses hedging programs to manage how changes in interest rates affect them. They estimate the impact if there's a 100 basis point increase across all yield curves. There are tables showing potential losses or gains for assets and liabilities in 2023 and 2022.

Next, currency risks: The company hedges against exchange rate fluctuations for several currencies like Canadian dollar, euro, etc. They mention potential losses from unfavorable changes in foreign exchange rates, with numbers for 2023 and 2022.

Then, commodity price risks: They use forward contracts to hedge prices of commodities used in truck production. The report states the potential loss from a 10% unfavorable change in commodity prices, again giving figures for both years.

I need to simplify this into prose that fifth graders can understand. I should avoid jargon and explain terms like "hedging" as strategies to reduce risk. Maybe use analogies they can relate to, like protecting something with insurance.

I'll structure it by introducing each type of risk, explain the method used (hedging), mention the potential losses in simple terms, and note that these are offset by other changes. I should keep sentences short and clear, avoiding complex structures.

Let me make sure I cover all three risks: interest rates, currencies, commodities. For each, a brief explanation of what it is, how they manage it, and the associated numbers without getting too bogged down in specifics.

I think that's a good plan. Now, let me put it all together in a summary.
</think>

Understanding Market Risks for Kids

Imagine you have a big lemonade stand, and you want to protect yourself from things that might make your profits go down. There are three main risks we talk

(continues on next page)

(continued from previous page)

about here: interest rates, currencies, and commodity prices.

1. **Interest Rates**: Just like how the bank gives you different rates if you save or borrow money, companies have to deal with changing interest rates too. If rates go up suddenly, it can affect how much money they make or owe. The company uses something called hedging to protect against these changes. They look at how much their assets and debts might change if interest rates went up by a full percentage point. In 2023, they saw a small loss, but in 2022, it was almost no loss.

2. **Currencies**: When you sell lemonade to people from other countries who use different money, the exchange rate can affect how much you make. The company uses contracts to protect against changes in these rates for currencies like the Canadian dollar and euro. If these rates go 10% against them, they might lose some money, but this was \$259.7 million in 2023 and \$216.6 million in 2022. However, other factors balance this out.

3. **Commodity Prices**: Making lemonade needs lemons, right? If the price of lemons goes up or down a lot, it affects your costs. The company uses forward contracts to lock in prices for materials they need, like metals and plastics for making trucks. A 10% bad change in these prices could cost them \$3.3 million in 2023 and \$2.5 million in 2022, but again, other changes help reduce this impact.

So, just like you might save money or buy insurance to protect your lemonade stand, companies use strategies like hedging to manage these risks and keep their profits safe.

===== 2/10. PHILLIPS 66 =====

<think>

Okay, so I'm trying to understand all these risks mentioned in the document. Let me start by reading through them carefully.

First, there are market conditions like fluctuations in prices and margins for NGLs, crude oil, natural gas, and refined products. That makes sense because energy prices can be really volatile due to things like supply and demand changes or geopolitical events.

Then there's government policies affecting pricing, regulation, taxation, especially exports. I know that export policies can have a big impact on supply and demand, so this is an important factor.

Capacity constraints in pipelines, storage, and fractionation facilities are another risk. If these infrastructure issues arise, it could limit how much product they can transport, leading to bottlenecks or higher costs.

OPEC and non-OPEC actions influence supply and demand, which affects prices. I remember that OPEC's decisions can cause significant shifts in the market.

The success of DCP LP integration is mentioned, including achieving synergies. This probably refers to a business strategy where they're combining operations, so if this doesn't go as planned, it could hurt their performance.

Unexpected technical difficulties or cost increases during construction or operation are risks too. Construction delays can be costly and disrupt production.

(continues on next page)

(continued from previous page)

Drilling and production volumes around midstream assets are another point. If the wells aren't producing as expected, it affects the company's revenue from those assets.

Permits and regulations compliance are also risks. They need to get permits for projects, and if they can't or if regulations change, it could delay things or require more spending.

Savings and cost reductions from business transformation initiatives might not happen as planned. If they don't achieve these savings, their financial goals could be at risk.

Renewable fuels policies and climate change regulations are factors too. Changes in these areas could affect demand for traditional fuels or require new investments.

Economic and political developments like the Russia-Ukraine war can impact markets. Also, things like inflation, interest rates, and expropriation of assets pose risks.

Public health crises, like pandemics, can disrupt operations and reduce demand for their products. The recovery after such events is also uncertain.

Capital projects might not be completed on time or within budget. Delays here can lead to cost overruns and project failures.

Asset dispositions or acquisitions could face challenges if they don't complete them successfully. This includes both the sale of assets and buying new ones.

Litigation or government actions could disrupt operations, leading to legal fees or operational changes.

Accidents, weather events, civil unrest, etc., can damage facilities and interrupt operations, causing financial losses.

Meeting sustainability goals is another risk. If they don't reduce GHG emissions as planned or develop new technologies, it could affect their reputation and operations.

New products might not be accepted by the market, leading to wasted investments.

Monetary conditions and exchange controls can impact international trade and profitability.

Environmental regulations requiring significant investments or reducing demand for their products are risks. They might have to spend a lot on compliance or face reduced sales.

Liability from environmental issues like cleanup costs is another concern.

Changes in laws and regulations, including alternative energy mandates, could force them to adapt their business models.

Climate change concerns might lead to changes in how they operate or increase expenses through litigation or other costs.

(continues on next page)

(continued from previous page)

Impairment charges could occur if the value of assets decreases due to market conditions or strategic decisions.

Access to capital is crucial; if it's limited or more expensive, they might not be able to fund their operations effectively.

Counterparties going bankrupt could lead to financial losses from unpaid debts or obligations.

Cybersecurity incidents can compromise sensitive data and lead to legal liabilities or operational disruptions.

Joint ventures that the company doesn't control could make decisions that negatively impact their own business.

Lastly, all these factors are generally described in Item 1A of the risk factors in the report.

I think I've covered all the points. Now, how do I summarize this? Maybe by categorizing them into market risks, regulatory risks, operational risks, financial risks, and environmental/sustainability risks. That way, it's easier to understand and address each category separately.

</think>

The document outlines a comprehensive list of risks that could impact the company's performance and operations. These risks can be categorized as follows:

1. **Market Risks**:**

- Volatility in energy prices (NGLs, crude oil, natural gas, refined products).
- Fluctuations in margins and demand for refining and petrochemical products.
- Impact of OPEC and non-OPEC actions on supply and demand.

2. **Regulatory and Policy Risks**:**

- Changes in government policies affecting pricing, regulation, taxation, and exports.
- Compliance with environmental regulations (GHG emissions, renewable fuels).
- Influence of international monetary conditions and exchange controls.

3. **Operational Risks**:**

- Capacity constraints in transportation infrastructure (pipelines, storage).
- Delays or cost overruns in construction and capital projects.
- Challenges in permit acquisition and regulatory compliance.
- Risks from joint ventures not under the company's control.

4. **Financial Risks**:**

- Failure to achieve expected savings from business transformation initiatives.
- Impairment charges due to asset value declines.
- Limitations in accessing capital or increased funding costs.
- Counterparty risks (e.g., bankruptcy leading to financial losses).

5. **Environmental and Sustainability Risks**:**

- Meeting sustainability goals and adapting to climate change regulations.
- Development of new technologies and products that may not gain market

(continues on next page)

(continued from previous page)

acceptance.

6. **Geopolitical and Public Health Risks**:**

- Impact of geopolitical events (e.g., Russia-Ukraine war) on markets.
- Disruptions from public health crises (e.g., pandemics).

7. **Legal and Reputational Risks**:**

- Litigation risks and potential liabilities from environmental issues.
- Cybersecurity threats leading to data breaches or operational disruptions.

By categorizing these risks, the company can better manage and mitigate potential challenges across different areas of its operations.

===== 3/10. MASTERCARD INC =====

<think>

Okay, so I need to explain this risk report to fifth graders. Let me read through it again and try to understand what it's saying.

The report is about market risks that a company faces, specifically MasterCard, I think. Market risk here refers to the possibility of losing money because of changes in things like interest rates or foreign exchange rates. The company talks about how they manage these risks by using derivatives and other financial instruments.

First, there's something about foreign currency risk. They use derivative contracts to handle the exposure from transactions that happen in different currencies. These contracts help them reduce losses if the exchange rate changes. For example, a 10% change could lead to a loss of around \$414 million in 2023 and \$94 million in 2022. But they also mention that these numbers don't account for the actual transactions they're hedging against.

Then there's interest rate risk. They have investments that are sensitive to changes in interest rates, but a 1% change wouldn't cause significant losses because their portfolio is diversified and they invest in high-quality securities. They also use derivative contracts to hedge some of their fixed-rate debt.

I need to simplify all this into about 300 words using simple language. I should avoid jargon and explain concepts like derivatives, exchange rates, and interest rates in a way that fifth graders can understand. Maybe compare it to something they know, like saving money or exchanging currencies when traveling.

I'll start by introducing what market risk is, then talk about how the company manages foreign currency risks using contracts, and then move on to interest rate risks. I should also mention that these risks are managed through policies and tools, but even with all that, there's still some potential for loss if things change unexpectedly.

Let me structure it step by step: introduce market risk, explain foreign exchange risk with an example, talk about how they manage it, then move to interest rates, and finally wrap up by saying that while they take steps to minimize these risks, there are still possible losses.

</think>

****Understanding Market Risks in Simple Terms****

(continues on next page)

(continued from previous page)

Imagine you have a piggy bank where you save your money. Sometimes, the value of your savings can change based on things like how much interest you earn or changes in exchange rates if you're saving different currencies.

Market risk is like the possibility that your piggy bank's value might go down because of these changes. Companies, just like you, want to protect their money from losing value.

One way they do this is by using something called derivatives, which are like special contracts. For example, if a company does business in different countries and gets money in other currencies, they can use these contracts to protect against the risk that those currencies might lose value compared to their home currency, like the US dollar or euro.

They also watch interest rates because if the bank where they keep their money changes the interest rate, it affects how much they earn. By using more of these special contracts, they can reduce the chance that their savings will be affected by these changes.

Even though companies use these tools to protect themselves, there's still a chance that unexpected changes could cause losses. So, while they take steps to keep their money safe, they know there are no guarantees in the ever-changing world of money and exchange rates.

===== 4/10. BRISTOL MYERS SQUIBB CO =====

<think>

Okay, so I need to explain this risk report to fifth graders. Let me read through it again to make sure I understand the main points.

The report talks about market risks related to currency exchange rates and interest rates. The company uses derivatives like forwards and swaps to hedge these risks when it's cost-effective. They mention that all financial instruments, including derivatives, are subject to counterparty credit risk, which is part of their fair value measurement. Importantly, they don't use derivatives for trading purposes.

Looking at foreign exchange risk, a big part of their revenue, earnings, and cash flow are affected by changes in currency rates, mainly the euro and Japanese yen. They use forward contracts and put options to manage risks from intercompany transactions. There's also transaction risk from non-functional currencies, which they handle with forwards but aren't designated as hedges. Additionally, cross-currency swaps help manage long-term debt in euros and hedge net investments.

They estimate that a 10% appreciation in the hedged currencies would decrease the fair value of their foreign exchange contracts by certain amounts, reducing earnings. Similarly, cross-currency swaps would have different impacts on their fair values.

For interest rate risk, they use fixed-to-floating swaps to balance their debt and analyze sensitivity to rate changes. A 1% increase in rates wouldn't harm earnings much, but a larger increase in long-term rates would decrease the fair value of their long-term debt.

Credit risk is managed by investing only with high-quality institutions and diversifying derivatives across counterparties to reduce default risks.

(continues on next page)

(continued from previous page)

Now, I need to simplify this into about 300 words. I'll start by introducing market risks, then break it down into foreign exchange and interest rate risks, explaining how they use derivatives to hedge. Then touch on credit risk management. I should avoid jargon and make each part clear for kids.

</think>

Imagine you have a big company that sells things around the world. Sometimes, when they sell or buy stuff in other countries, the money they get can change value because of something called exchange rates. It's like if you have dollars and someone else has euros, and one day those euros might be worth more or less compared to your dollars.

The company uses special tools called derivatives to protect themselves from these changes. These tools are kind of like insurance policies that help them guess what the money will look like in the future so they don't lose too much if things change.

They also have to worry about interest rates, which are like the cost of borrowing money. If the interest rates go up, it might make their debts more expensive. But again, they use these derivative tools to balance things out and not get hurt too badly.

Lastly, they make sure that the people they deal with (their counterparties) are trustworthy so that if something goes wrong, they don't lose money because someone else didn't pay them back.

So, in short, the company is careful about how it handles money changes and interest rates using special tools to stay safe financially.

===== 5/10. CARRIER GLOBAL CORP =====

</think>

Okay, so I need to explain this risk report to fifth graders. Let me read through it again and try to understand what it's saying.

The report talks about market risks like foreign currency exchange rates, interest rates, and commodity prices affecting the company's operations and financial condition. It says there haven't been significant changes in their exposure to these risks in 2023.

First part is about foreign currency exposures. They have operations all over the world, so when currencies change against the US dollar, it affects their reported results. They manage some currency risks with things like window forward contracts and cross-currency swaps, especially for big acquisitions. But they don't hedge translation risk entirely.

Then there's commodity price exposure. They use fixed-price contracts sometimes but didn't have any hedges at the end of 2023. Fuel costs for shipping are another concern.

Interest rate exposure is low because most debt has fixed rates, so changes in market rates won't affect them much.

I need to simplify all this into a summary that's easy for fifth graders to understand. Maybe use examples they can relate to, like how exchanging money when traveling can change how much you have, or how prices of things like oil

(continues on next page)

(continued from previous page)

can go up and down affecting costs.

I should avoid jargon and keep sentences short. Make sure each point is clear and relatable.

</think>

The company faces risks from changes in foreign exchange rates, interest rates, and commodity prices, which can affect their profits and financial health. These risks haven't changed much over the past year.

****Foreign Currencies:****

They operate worldwide, so when other countries' currencies go up or down against the US dollar, it affects how much money they make. They use strategies like window forward contracts to manage some of these risks, especially for big purchases. For example, if they buy something in euros and the euro goes up, it might cost them more in dollars.

****Commodity Prices:****

They rely on materials whose prices can swing, like oil used for shipping. Sometimes they lock in fixed prices to avoid sudden jumps, but at the end of 2023, they didn't have any such contracts.

****Interest Rates:****

Most of their loans have fixed rates, so even if market rates change, it won't affect them much. It's like having a set monthly payment on a loan, no surprises.

In summary, they manage some risks with smart financial tools but still face uncertainties from global markets and material costs.

===== 6/10. LULULEMON ATHLETICA INC =====

</think>

Okay, so I need to explain this risk report to fifth graders. Let me read through it again and try to understand what each part is saying.

First, the report talks about market risks, specifically foreign currency exchange risk and translation risk. It mentions that their international subsidiaries use local currencies as functional currencies, but the financial statements are in US dollars. So when they translate from other currencies to USD, changes in exchange rates affect the reported amounts. In 2023, their revenue was lower by \$89.8 million compared to 2022 because of this.

They also mention translation risk where differences from translating balance sheets into USD are recorded as part of comprehensive income. They have a Canadian subsidiary with significant net assets and use forward contracts to hedge some of this exposure. The impact on their comprehensive loss was an increase due to these hedges.

Then there's transaction risk, which comes from transactions in currencies other than the functional currency—like intercompany transactions or inventory purchases. They hold cash in different currencies too. As of a certain date, they had forward contracts to hedge revaluation gains and losses on monetary assets and liabilities of their subsidiaries.

They did a sensitivity analysis showing that a 10% depreciation in USD would affect the fair value of derivatives by \$29.8 million. But this is offset by

(continues on next page)

(continued from previous page)

changes in the underlying hedged items. They plan to use more derivatives in the future for other currency pairs but don't trade them for profit.

Next, interest rate risk: their credit facility has variable rates, so if they borrow money, changing rates could affect them. As of the date, they didn't have any borrowings except for some letters of credit. They don't do any hedging now but might in the future if needed.

They hold cash and equivalents in safe places like money market funds, which are AAA-rated. Even though some deposits exceed insured limits, they haven't had any losses. They monitor the financial institutions they deal with to minimize credit risk.

Lastly, inflation could affect their costs—like wages, freight, and raw materials—which might hurt their margins if they can't pass these costs onto customers or find efficiencies.

So, putting this all together, I need to explain it in simple terms for fifth graders. Maybe start by talking about exchanging money between countries and how the value of money changes over time. Then explain that when companies do business internationally, they have to convert currencies, which can lead to gains or losses based on exchange rates. Use an example with a pretend company to make it relatable.

I should also mention hedges as a way to protect against these risks, maybe using the analogy of insurance. Keep each section short and use everyday language so it's easy to understand.

</think>

Explaining Market Risks in Simple Terms for Fifth Graders

Imagine you're a kid who loves to travel and exchange money between countries. When you go on vacation, you might take some dollars and exchange them into another country's currency, like Canadian dollars or Euros. Now, companies that do business all over the world also have to deal with exchanging money, but it can be tricky because the value of these currencies changes over time.

1. Foreign Currency Exchange Risk:

Just like how you might get more or fewer toys when you exchange your money, companies can end up with more or less "toys" (money) based on how the exchange rate changes. For example, if a company earns money in Canada but reports it in US dollars, a change in the value of the Canadian dollar against the US dollar can make their earnings go up or down. In 2023, this change made their revenue \$89.8 million less than in 2022.

2. Translation Risk:

When companies translate their money from one country's currency to another, like converting Canadian dollars back to US dollars for reports, the value can change. This is called translation risk. Sometimes, these changes can affect how much profit or loss a company shows on its financial statements.

3. Transaction Risk:

Companies also face risks when they do transactions in different currencies. For example, if they buy something from another country and pay in that country's currency, the value of that payment can change by the time it's converted back to their home currency. To protect against this, companies might use "hedges,"

(continues on next page)

(continued from previous page)

which are like insurance policies to lock in exchange rates.

****4. Interest Rate Risk:****

Borrowing money is another area where companies face risks. If they take a loan with an interest rate that changes over time, and if those rates go up, it could cost them more money to pay back the loan. Companies keep an eye on this so they can plan for any increases.

****5. Inflation:****

Inflation is when things like food, clothes, and other goods get more expensive over time. For companies, this means their costs might go up too—like paying more for materials or worker wages. If a company can't increase its prices enough to cover these higher costs, it could affect their profits.

****6. Credit Risk:****

When companies deal with banks or other financial institutions, there's always a chance that the other side might not be able to pay back what they owe. To minimize this risk, companies work with reliable institutions and keep track of how much money they're dealing with from each partner.

In summary, just like you have to be careful when exchanging money while traveling, companies must manage these risks carefully to ensure their profits aren't affected by changes in currency values, interest rates, or inflation. Using tools like hedges and working with trustworthy institutions helps them navigate these challenges smoothly.

===== 7/10. AIRBNB INC =====

<think>

Okay, so I need to summarize this risk report for fifth graders. Let me read through it carefully first.

The report talks about market risks that Airbnb faces because they operate all over the world. The main risks are foreign currency risk and investment risk.

Foreign currency risk is when changes in exchange rates affect their financial results. They accept payments in over 40 currencies, so if the US dollar strengthens or weakens, it can impact how much money they make or spend. For example, if the US dollar gets stronger, their revenue from other countries might decrease because those dollars are worth less.

They also mention that they use foreign currency derivatives to hedge against some of these risks, but they don't eliminate all the effects. There's a 20 million loss mentioned if exchange rates were to change unfavorably.

Investment risk is related to their investment portfolio. They have a lot of cash and short-term investments in things like debt securities. If interest rates go up or down, it can affect how much they earn on these investments. A 100 basis point increase would decrease their portfolio by 20 million. But since most of their investments are short-term, the risk from interest rate changes isn't too bad.

I need to explain all this in simple terms for kids. Maybe use analogies they can understand, like comparing currencies to different types of money and how exchanging them can be tricky. Also, simplify investment risks by talking about saving money in a piggy bank or something safe where small changes in interest rates don't matter much.

(continues on next page)

(continued from previous page)

I should structure it into two main parts: foreign currency risk and investment risk. Use everyday language and avoid jargon. Maybe include examples they can relate to, like traveling abroad and exchanging money.

</think>

Understanding Airbnb's Market Risks in Simple Terms

Airbnb faces some financial risks because they operate all around the world, dealing with many different currencies and investments. Let's break down these risks into two main parts: foreign currency risk and investment risk.

1. **Foreign Currency Risk:**

- Imagine you have money in one country's currency, like euros from France or pounds from England. If the value of your home currency (like the US dollar) changes, it can affect how much money you actually have when you exchange it.
- Airbnb deals with over 40 currencies, which means they are exposed to changes in exchange rates. For example, if the US dollar becomes stronger, the money they make from other countries might decrease because those dollars are worth less.
- To protect against these changes, Airbnb uses something called foreign currency derivatives, which are like insurance policies to reduce the impact of bad exchange rate changes. However, even with this protection, there's still some risk. If exchange rates were to change unfavorably, they could lose about \$20 million.

2. **Investment Risk:**

- Airbnb keeps a lot of money in safe investments, like short-term debt securities and government bonds. These are similar to putting money in a savings account where it's secure but doesn't earn a lot.
- If interest rates change, the value of these investments can go up or down. For instance, if interest rates were to increase by 1%, their investment portfolio could decrease by about \$20 million.
- However, since most of their investments are short-term, they aren't too affected by small changes in interest rates. It's like having money in a piggy bank where you don't need it right away, so minor fluctuations won't hurt much.

In summary, Airbnb manages risks from exchanging currencies and investing by using protective measures, but there are still some uncertainties due to the global nature of their business.

===== 8/10. MERCADOLIBRE INC =====

</think>

Okay, so I'm trying to understand this document about MercadoLibre's risks and sensitivities. It seems like it's part of their financial disclosures, maybe in their annual report or something similar. Let me break down what each section is saying.

First, the foreign exchange risk. They have a significant amount of revenue from Brazil, Argentina, and other Latin American countries. Since they're dealing with multiple currencies, especially the Brazilian real and Argentine peso, which can be volatile, this could affect their financials. If the real or peso weakens against the dollar, their reported revenue in dollars might decrease because when converted, those currencies would buy fewer dollars. That makes sense because if you have a lot of revenue in a currency that's dropping, your overall revenue in USD terms goes down.

(continues on next page)

(continued from previous page)

Next is interest rate risk. They have long-term debt, which means they're exposed to changes in interest rates. If rates go up, the value of their existing debt might decrease, and their borrowing costs could increase if they need to refinance or take on more debt. I'm not entirely sure how sensitive they are to rate changes, but it's something to watch, especially with global interest rates potentially rising.

Then there's commodity price risk. They're exposed to fuel and electricity prices because these are operational costs. If oil prices go up, their delivery and logistics costs would increase, affecting their margins. Similarly, higher electricity prices could impact their operations, especially in countries where energy is a significant cost. This seems like a manageable risk but could have noticeable effects if prices spike.

Equity price risk is tied to their LTRP plans. These are long-term retention programs that pay out based on stock performance. The value of these obligations is sensitive to the company's stock price. If the stock goes up, the payouts increase, which could strain their finances. Conversely, a drop in stock price reduces the payout liability. They've done a sensitivity analysis showing how a 40% change in stock price affects the obligation. That's pretty significant; a 40% drop would reduce the liability from \$418 million to about \$251 million, which is a substantial difference.

Inflation risk is another factor, especially in countries like Argentina where inflation has been high. High inflation can erode purchasing power and affect consumer behavior. If people have less money due to inflation, they might spend less online, impacting MercadoLibre's sales. Additionally, rising costs for goods and services could squeeze margins if the company can't pass these costs onto customers.

I'm a bit confused about how exactly the equity price affects their obligations. It seems like it's based on the stock price relative to a fixed average from previous years. So each payment is tied to whether the current stock price is higher or lower than that average. If the stock does well, they have to pay more; if not, less.

I'm also thinking about how these risks interact. For example, inflation could lead to higher interest rates, which would then affect their debt costs. Or a weaker currency might make imports more expensive, contributing to inflation. It's like a web of interconnected risks that could amplify each other in certain economic conditions.

Another point is the sensitivity analysis for equity price risk. They show a static present value and how it changes with different stock prices. This helps investors understand how exposed they are to stock price fluctuations. A 40% change seems extreme, but it's useful to see the range of possible outcomes.

I wonder if there are any mitigating factors or strategies MercadoLibre uses to manage these risks. For example, do they hedge their currency exposure? How flexible are their costs in the face of inflation? And how do they structure their LTRP plans to balance retention with financial risk?

Overall, this section is highlighting several key areas where MercadoLibre faces potential volatility and challenges. It's important for investors to consider these risks when evaluating the company's long-term prospects.

(continues on next page)

(continued from previous page)

```
</think>

**MercadoLibre Risk Analysis Summary**

1. **Foreign Exchange Risk:**
   - **Exposure:** Significant revenue from Brazil, Argentina, and other Latin American countries exposed to currency fluctuations, particularly the Brazilian real and Argentine peso.
   - **Impact:** Currency devaluation can reduce reported USD revenue when converted.

2. **Interest Rate Risk:**
   - **Exposure:** Long-term debt subject to interest rate changes.
   - **Impact:** Rising rates may decrease debt value and increase borrowing costs.

3. **Commodity Price Risk:**
   - **Exposure:** Operational costs tied to fuel and electricity prices.
   - **Impact:** Price spikes can increase operational expenses and reduce margins.

4. **Equity Price Risk:**
   - **Exposure:** Long-term retention programs (LTRP) linked to stock performance.
   - **Impact:** Stock price fluctuations affect payout obligations; a 40% drop reduces liability from $418 million to $251 million.

5. **Inflation Risk:**
   - **Exposure:** High inflation in countries like Argentina affecting purchasing power and operational costs.
   - **Impact:** Could reduce consumer spending and squeeze margins if cost increases aren't offset.

6. **Interconnected Risks:**
   - Inflation may lead to higher interest rates, affecting debt costs and potentially exacerbating currency issues.

7. **Mitigation Strategies:**
   - Consider hedging strategies for currency exposure, flexible cost structures, and balanced LTRP designs to manage financial risks.
```

Conclusion: MercadoLibre faces multiple interconnected risks that could impact its financial health. Investors should assess these factors alongside the company's risk management strategies when evaluating long-term prospects.

===== 9/10. A T & T INC =====

```
<think>
Okay, so I'm trying to understand this document about AT&T's financial report. It looks like it's an audit report from their independent registered public accounting firm. Let me go through it step by step.
```

First, the report is divided into two main sections: one on the internal control over financial reporting and another on the consolidated financial statements. Both have an opinion section, a basis for opinion, and some additional details.

Starting with the opinion on internal control over financial reporting. The

(continues on next page)

(continued from previous page)

auditors state that they've audited AT&T's internal controls as of December 31, 2023, based on the COSO criteria. They conclude that the company maintained effective internal control over financial reporting in all material respects. That sounds positive; it means the auditors think AT&T has good systems in place to manage their finances properly.

Next, looking at the basis for this opinion. It mentions that management is responsible for maintaining these controls and assessing their effectiveness. The auditors' role was to express an opinion based on their audit, which they did by following PCAOB standards. They obtained an understanding of the internal controls, assessed risks, tested the design and operation of the controls, and performed other necessary procedures. Their conclusion is that AT&T's controls are effective.

Moving on to the report on the consolidated financial statements. The auditors also express an unqualified opinion here, meaning they didn't find any material issues with the financial statements. They were prepared in accordance with GAAP, which is standard for public companies in the U.S.

The basis for this opinion section explains that the auditors are responsible for expressing opinions on whether the financial statements present fairly, in all material respects, the company's financial position and results of operations. They did this by auditing according to PCAOB standards, assessing risks, testing accounting policies, significant estimates, transactions, and disclosures.

There are also two specific sections discussing areas where the audit was particularly complex: internal control over financial reporting and the impairment evaluations for certain reporting units. The auditors explain that these areas required a deeper dive because they involve subjective assumptions and complex methodologies. They tested controls, assessed assumptions against industry trends, performed sensitivity analyses, and involved specialists to ensure everything was in order.

The report ends with a note that the auditing firm has been serving AT&T since 1999, which shows a long-standing relationship, likely indicating consistency and familiarity with the company's operations.

I'm trying to think if there's anything I might be missing. The document seems comprehensive, covering both the internal controls and financial statements. It mentions specific areas like impairment evaluations and valuation models, which are known for being complex and subject to judgment. The auditors' involvement of specialists in these areas makes sense because it adds depth to their analysis.

I wonder if there were any material weaknesses or significant deficiencies mentioned, but from what I can see, the opinions are unqualified, so no major issues were found. It's also worth noting that the report is dated February 23, 2024, which would typically be around the time when earnings are released, so it's timely.

Overall, this audit report seems to provide assurance to stakeholders that AT&T's financial reporting is reliable and their internal controls are effective. The detailed explanation of the auditing process gives confidence in the independence and thoroughness of the audit.

</think>

(continues on next page)

(continued from previous page)

The audit report on AT&T provides a comprehensive assessment of both its internal control over financial reporting and its consolidated financial statements, concluding with an unqualified opinion in both areas. Here's a structured summary:

Key Findings:

1. **Internal Control Over Financial Reporting:**

- The auditors found that AT&T maintained effective internal controls as of December 31, 2023, based on the COSO framework.
- The company's systems provide reasonable assurance regarding the reliability of financial reporting and compliance with GAAP.

2. **Consolidated Financial Statements:**

- The financial statements were prepared in accordance with GAAP and present AT&T's financial position accurately.
- No material issues were identified, leading to an unqualified opinion.

Methodology:

- Auditors followed PCAOB standards, assessing risks, testing controls, and involving specialists for complex areas like impairment evaluations.
- Detailed procedures included testing assumptions against industry trends and performing sensitivity analyses.

Areas of Complexity:

- **Impairment Evaluations:** Involved subjective assumptions and complex valuation methods. The auditors ensured these were sound through thorough testing and specialist involvement.
- **Internal Controls:** The effectiveness was validated through a comprehensive audit process, confirming the controls' operational efficiency.

Conclusion:

The report assures stakeholders of AT&T's financial integrity and control systems. The unqualified opinions indicate no significant issues, reflecting well on the company's financial governance. The long-standing relationship with the auditing firm underscores consistency and familiarity in their work.

===== 10/10. REPUBLIC SERVICES INC =====

<think>

Okay, so I need to summarize this risk report for fifth graders. Let me read through it carefully first.

The report talks about market risks the company faces, specifically interest rate risk, fuel price risk, and commodities price risk. Hmm, I should break each section down into simpler terms.

Starting with interest rate risk: The company has both fixed and variable debt. Fixed rates don't change, while variable ones do based on market rates. They use swaps to manage this risk. If rates go up or down by 100 basis points, their interest expense changes by about \$20 million. I should explain what a swap is in simple terms, maybe like an agreement to balance out the risks.

Next, fuel price risk: Fuel costs are a big part of their expenses. They try to hedge against price changes but currently have no hedges in place. A 20-cent change per gallon affects both their costs and what they charge customers. Petrochemical products used in operations also cost money, so higher prices

(continues on next page)

(continued from previous page)

there would increase their costs too.

Lastly, commodities price risk: They sell recycled materials, and market fluctuations affect their revenue. Previously, they used derivatives like swaps to hedge this, but none are active now. A \$10 per ton change affects both revenue and operating income by about \$10 million.

I need to make sure I explain terms like "hedging" in a way fifth graders can understand, maybe as strategies to protect against price changes. Also, I should avoid jargon and keep sentences short and clear.

Let me structure the summary with each risk as a separate point, using simple language and examples they can relate to, like how changing fuel prices affect their daily lives or school budget.

</think>

Summary of Market Risks in Simple Terms

The company faces several risks related to market changes that could impact its financial health. Here's a breakdown of these risks explained in a way fifth graders can understand:

1. **Interest Rate Risk:**

- The company borrows money through loans with both fixed and variable interest rates.
- Fixed rates stay the same, while variable rates change based on market conditions.
- To manage these changes, they use something called "swaps," which are agreements to balance out potential losses or gains from rate changes.
- If interest rates go up or down by a certain amount (100 basis points), their yearly interest expense would change by about \$20 million. This is like how a school might adjust its budget if the cost of supplies goes up or down.

2. **Fuel Price Risk:**

- Fuel costs are a major expense for the company, similar to how a school might spend money on transportation.
- They try to protect against fuel price changes by using strategies called "hedging," but currently, they don't have any such strategies in place.
- A 20-cent change in diesel price per gallon would affect both their costs and what they charge customers. For example, if diesel goes up by 20 cents, their expenses increase, and they might need to charge more to cover this cost.

3. **Commodities Price Risk:**

- The company sells recycled materials like cardboard and newspaper.
- Market changes can cause prices of these materials to go up or down, affecting both revenue and how much money they make.
- They used to use tools like "swaps" and "collars" to manage these price changes, but currently, they aren't using any such tools.
- A \$10 per ton change in recycled material prices would impact their yearly revenue and profits by about \$10 million. This is similar to how a lemonade stand might adjust its prices if the cost of lemons fluctuates.

In summary, the company deals with risks from changing interest rates, fuel prices, and recycled materials markets. They use strategies like swaps to manage some of these risks but are currently exposed to others. Understanding these risks helps them make better financial decisions, much like how planning for price changes helps a school manage its budget effectively.

```
scores = collect_rouge(target=summary[gpt_name], prediction=summary['simple_deeplearn']  
→'])
```

```
display_rouge("rouge1", scores)
```

ROUGE1 metric:

	precision	recall	fmeasure
PACCAR INC	0.397394	0.408027	0.402640
PHILLIPS 66	0.366142	0.324042	0.343808
MASTERCARD INC	0.371681	0.283784	0.321839
BRISTOL MYERS SQUIBB CO	0.331731	0.226974	0.269531
CARRIER GLOBAL CORP	0.502674	0.324138	0.394130
LULULEMON ATHLETICA INC	0.317172	0.537671	0.398983
AIRBNB INC	0.391304	0.440559	0.414474
MERCADOLIBRE INC	0.467593	0.331148	0.387716
A T & T INC	0.393805	0.312281	0.348337
REPUBLIC SERVICES INC	0.414439	0.504886	0.455213
average	0.395393	0.369351	0.373667

```
display_rouge("rouge2", scores)
```

ROUGE2 metric:

	precision	recall	fmeasure
PACCAR INC	0.133987	0.137584	0.135762
PHILLIPS 66	0.079051	0.069930	0.074212
MASTERCARD INC	0.066667	0.050847	0.057692
BRISTOL MYERS SQUIBB CO	0.053140	0.036304	0.043137
CARRIER GLOBAL CORP	0.134409	0.086505	0.105263
LULULEMON ATHLETICA INC	0.052632	0.089347	0.066242
AIRBNB INC	0.093458	0.105263	0.099010
MERCADOLIBRE INC	0.116279	0.082237	0.096339
A T & T INC	0.097778	0.077465	0.086444
REPUBLIC SERVICES INC	0.150134	0.183007	0.164948
average	0.097753	0.091849	0.092905

Generate simple summaries with GPT-4o-mini

```
summary['simple_gpt-4o'] = collect_summaries(  
    gpt_model,  
    role="You are a patient lower-school teacher, using simple words to explain to  
→your students in the fifth grade.")
```

===== 1/10. PACCAR INC =====

The risk report talks about how a company manages different types of financial risks, especially related to market changes. It focuses on three main areas: interest rates, currency exchange rates, and commodity prices.

First, for interest-rate risks, the company looks at how changes in interest rates can affect the value of its assets and debts. They estimate what would

(continues on next page)

(continued from previous page)

happen if interest rates suddenly went up by 1%. In 2023, the company faced losses of \$29.2 million from cash and marketable securities and \$146.5 million from fixed-rate loans. However, they also had gains from fixed-rate debts and interest-rate swaps, leading to a total loss of \$17.7 million, which was worse than the previous year's loss of \$1.1 million.

Next, the report discusses currency risks. The company uses contracts to protect itself from changes in foreign currency values, especially with currencies like the Canadian dollar and the euro. If the value of these currencies drops by 10%, the company could lose \$259.7 million in 2023, compared to \$216.6 million in 2022. However, these losses would be balanced out by changes in the value of the things they are protecting.

Lastly, the company also deals with commodity price risks. They use contracts to lock in prices for materials needed to make trucks. If commodity prices drop by 10%, they could lose \$3.3 million in 2023, up from \$2.5 million in 2022. Again, these losses would be offset by changes in the value of the materials they are buying.

Overall, the report shows how the company tries to manage risks to protect its finances from sudden changes in the market.

===== 2/10. PHILLIPS 66 =====

The risk report discusses the various market risks that our company and some of its subsidiaries face. These risks come from changes in prices of crude oil, natural gas, and other energy products, as well as fluctuations in interest rates and foreign currency exchange rates. To manage these risks, we use special financial contracts called derivatives.

Our main strategy is to stay connected to the market prices of these commodities. We use derivatives to switch from fixed prices in our contracts back to floating market prices. This helps us balance our supply and demand needs and take advantage of market opportunities, like moving products to where they can be sold for more money or storing them for better prices later. We also use derivatives to protect our cash flow from price changes.

The use of these financial tools is carefully controlled by rules set by our board of directors. We monitor our risks daily to ensure we stay within safe limits. We use a method called Value at Risk (VaR) to estimate potential losses from market changes, and as of the end of 2023, these risks were not significant to our cash flow.

Additionally, we face interest rate risks from our debts. Fixed-rate debts can lose value if interest rates rise, while variable-rate debts can lead to higher interest costs. We provide detailed information about our debts and their interest rates.

Lastly, we are also exposed to foreign currency risks due to our international operations, but we generally do not take steps to protect against these risks. Our CEO and CFO keep a close eye on all these risks to ensure the company remains stable and successful.

===== 3/10. MASTERCARD INC =====

The risk report discusses market risk, which is the chance of losing money due to changes in things like interest rates and foreign currency exchange rates. The company, Mastercard, has a limited exposure to these risks and keeps a close

(continues on next page)

(continued from previous page)

watch on them. They have rules in place to manage their funding, investments, and the use of special financial tools called derivatives to help reduce these risks.

One type of risk they face is foreign exchange risk, which happens when they deal with money in different currencies. To manage this risk, Mastercard uses foreign exchange derivative contracts. These contracts help them handle expected payments and receipts in currencies that are not their main currency, which is usually the U.S. dollar or the euro. If the value of these currencies changes by 10%, it could lead to a loss of about \$414 million on their contracts as of December 31, 2023. However, they also have activities that can offset these losses.

Mastercard also deals with foreign exchange risk when they have investments in other countries. They may use contracts to protect these investments, but as of December 31, 2023, they did not have any contracts specifically for this purpose. In 2022, a similar 10% change in the U.S. dollar could have led to a loss of about \$203 million.

Another risk is interest rate risk, which affects their investments in bonds. Mastercard aims to invest in high-quality bonds and manage their risks carefully. A change in interest rates by 1% would not significantly impact the value of their investments. They also use interest rate derivatives to protect against changes in the value of their fixed-rate debt, and similar to foreign exchange, a 1% change would not have a major effect on their contracts. Overall, Mastercard actively manages these risks to protect their financial health.

===== 4/10. BRISTOL MYERS SQUIBB CO =====

The risk report discusses how a company faces market risks related to changes in currency exchange rates and interest rates. To protect itself from these risks, the company uses certain financial tools called derivatives, but only when they are cost-effective. These derivatives are not used for trading but to manage risks.

One major risk is foreign exchange risk, which happens when the value of money in different countries changes. The company earns a lot of its money in euros and Japanese yen, so it uses contracts called foreign currency forwards to help manage this risk. These contracts help protect the company from losing money when it buys or sells things in different currencies. However, these contracts are not always perfect hedges, meaning they don't cover all risks completely.

The report estimates that if the euro and yen become stronger against the U.S. dollar by 10%, the company could lose a significant amount of money on its foreign exchange contracts. Additionally, the company uses cross-currency swaps to manage risks from long-term debts in euros and to protect its investments in foreign businesses.

The report also talks about interest rate risk, which is the risk of losing money when interest rates change. The company uses interest rate swaps to balance its debts. If interest rates go up by 1%, the impact on the company's earnings would not be very large, but it could still affect the value of its long-term debt.

Lastly, the company is careful about credit risk, which is the risk of losing money if someone it does business with doesn't pay. It only invests in high-quality institutions and spreads its investments to reduce risk. Overall, the

(continues on next page)

(continued from previous page)

company takes steps to manage these financial risks to protect its earnings and investments.

===== 5/10. CARRIER GLOBAL CORP =====

The risk report discusses the market risks that our company faces, which include changes in foreign currency exchange rates, interest rates, and commodity prices. These risks can affect how well we perform financially. For the year ending December 31, 2023, there have not been any major changes in our exposure to these market risks.

One important area is foreign currency exposure. Since we operate in many countries, we deal with different currencies. Our main reporting currency is the U.S. dollar, and when other currencies change in value compared to the dollar, it can affect our financial results. We manage some of these currency risks related to our sales and purchases, but we do not protect against all risks from currency changes.

Recently, we made an acquisition and paid a large part of the purchase price in euros. To protect ourselves from changes in the euro's value compared to the U.S. dollar, we used special contracts called window forward contracts. These contracts help us manage the risk of losing money due to currency fluctuations.

For another acquisition, we used cross currency swaps and a loan in Japanese yen to help manage the risks related to the yen. These financial tools help us keep track of changes in currency values and their impact on our investments.

We also face risks from the prices of commodities, which are materials we use to make our products. We sometimes use fixed price contracts to help manage these costs, but as of December 31, 2023, we do not have any contracts in place to protect against changes in commodity prices.

Lastly, most of our long-term debt has fixed interest rates, so changes in interest rates are not expected to significantly affect our financial results. Overall, we are actively managing these risks to protect our company.

===== 6/10. LULULEMON ATHLETICA INC =====

The risk report discusses various financial risks that the company faces, particularly related to foreign currency exchange, interest rates, credit, and inflation.

****Foreign Currency Exchange Risk**:** The company has international subsidiaries that operate in local currencies, but its financial statements are in U.S. dollars. This means that when the value of the U.S. dollar changes, it affects how much revenue and expenses are reported. In 2023, the company reported \$89.8 million less in revenue compared to 2022 due to these currency fluctuations. The company uses forward currency contracts to protect itself from these risks, especially with its Canadian subsidiary, which saw a loss of \$9 million from currency translation.

****Transaction Risk**:** The company also faces risks when its subsidiaries conduct transactions in currencies different from their main currency. They hold cash and other assets in various currencies and use forward contracts to manage these risks. As of January 28, 2024, the company had a liability of \$2.2 million from these contracts, and a 10% drop in the U.S. dollar could lead to a \$29.8 million decrease in value.

(continues on next page)

(continued from previous page)

****Interest Rate Risk**:** The company has a credit facility that allows it to borrow up to \$400 million at variable interest rates. If it borrows a significant amount, it could be affected by changes in interest rates. Currently, they have no borrowings but may consider hedging against interest rate risks in the future.

****Credit Risk**:** The company keeps cash with reputable banks and invests in high-rated money market funds. They are cautious about credit risk, which is the risk of losing money if a bank fails to meet its obligations. They have not faced any losses and believe this risk is low.

****Inflation**:** Rising costs for products, wages, and transportation can hurt the company's profits. In recent years, increased wages and shipping costs have affected their operating margins. If these costs continue to rise without a corresponding increase in product prices, it could impact their financial health.

===== 7/10. AIRBNB INC =====

The risk report discusses the market risks faced by a company with global operations, focusing on foreign currency risk and investment risk.

****Foreign Currency Risk**:** The company operates in over 40 currencies, with key currencies including the euro, British pound, Canadian dollar, Australian dollar, Brazilian real, and Mexican peso. This means that when the value of these currencies changes compared to the U.S. dollar, it can affect the company's financial results. For example, if the U.S. dollar strengthens, the company could lose money on its international sales. The company has various sources of foreign currency risk, such as revenue from bookings made in other currencies, funds held for customers, and payments to hosts. To manage this risk, the company uses foreign currency derivative contracts, which help protect against changes in exchange rates. However, these contracts do not completely eliminate the risk, and the company may choose not to hedge certain exposures due to costs or accounting reasons. If there were a significant change in exchange rates, it could lead to substantial losses.

****Investment and Interest Rate Risk**:** The company also faces risks related to interest rates, which can affect the earnings from its investments. As of December 31, 2023, the company had significant cash and short-term investments, primarily in safe, liquid assets like government bonds and corporate debt. The goal of these investments is to keep the money safe while ensuring it is available when needed. Because the investments are mostly short-term, they are not very sensitive to changes in interest rates. A hypothetical increase in interest rates could lead to a small decrease in the value of the investment portfolio, but the company does not expect to face major risks from interest rate changes.

===== 8/10. MERCADOLIBRE INC =====

The risk report discusses the various market risks that our company faces due to its business operations. These risks mainly come from changes in the economy, interest rates, and currency exchange rates, especially with currencies like the Brazilian real, Argentine peso, and Mexican peso. These changes can affect the value of our financial assets and liabilities.

We also have long-term retention programs for employees, which involve cash payments that depend on our stock price. This means that if our stock price changes, the amount we pay to employees can also change.

(continues on next page)

(continued from previous page)

Since we operate in many countries, we deal with different currencies, which exposes us to foreign currency risk. This can impact our financial results because we earn money and spend money in various currencies. To manage this risk, we use contracts that help protect us from unfavorable changes in currency exchange rates. However, these contracts do not completely eliminate the risk.

As of December 31, 2023, we had significant amounts of cash and investments in local currencies. Our subsidiaries mostly earn and spend money in their local currencies, except for our Argentine subsidiaries, which use the U.S. dollar due to high inflation. We also experienced a loss of \$615 million in foreign currency due to changes in the Argentine market.

Interest rates also affect our earnings and cash flow. Changes in interest rates can impact the costs of loans and the income we earn from our investments. As of December 31, 2023, we had various loans and investments that are sensitive to interest rate changes.

Lastly, our long-term retention programs for employees are linked to our stock price, meaning that if our stock price goes up or down, it affects how much we owe to employees. Overall, these market risks can significantly impact our financial performance.

===== 9/10. A T & T INC =====

The risk report discusses how AT&T Inc. manages market risks, particularly those related to interest rates and foreign currency exchange rates. These risks can affect the company's costs and financial stability. To handle these risks, AT&T uses various financial tools called derivatives, such as interest rate swaps and foreign currency contracts. These tools help the company control its financial risks and maintain flexibility without engaging in risky trading practices.

One important aspect of the report is the company's approach to estimating its future obligations for employee benefits, which relies on a discount rate. This rate is influenced by the returns on high-quality corporate bonds. Recently, these rates have been lower and more unstable than in the past, which means that the company's obligations could be higher. If rates increase in the future, it could lower these obligations and improve the company's financial health.

The report also highlights interest rate risk, as most of AT&T's financial instruments are fixed-rate notes. Changes in interest rates can significantly affect their value. To manage this risk, AT&T monitors its debt structure and uses interest rate swaps to balance fixed and floating rates.

Additionally, AT&T addresses foreign exchange risk by converting foreign debt into U.S. dollars through cross-currency swaps. This helps eliminate risks related to currency fluctuations. The report includes details about the company's financial instruments and their fair values as of December 31, 2023.

Overall, AT&T is committed to managing its market risks carefully to ensure long-term financial stability and flexibility. The company regularly assesses its financial strategies and maintains effective internal controls to support accurate financial reporting.

===== 10/10. REPUBLIC SERVICES INC =====

The risk report discusses the financial risks related to interest rates, fuel prices, and commodity prices that the company faces.

(continues on next page)

(continued from previous page)

****Interest Rate Risk:**** The company is mainly affected by changes in interest rates in the United States. To manage this risk, they use a mix of fixed and variable interest rate debts. As of December 31, 2023, the company had a total of about \$10.4 billion in fixed-rate debt and around \$2.6 billion in variable-rate debt. They also use financial agreements called swap contracts to help protect against changes in interest rates. If interest rates go up or down by 1%, the company's interest expenses could change by about \$20 million each year.

****Fuel Price Risk:**** Fuel costs are a significant expense for the company. They may use fuel hedges, which are contracts to lock in fuel prices, but as of the end of 2023, they had no such contracts in place. A change of 20 cents per gallon in diesel fuel prices could affect their fuel costs by about \$27 million annually. They charge some customers for fuel recovery fees, but not all, which can impact their revenue. In 2023, their fuel costs were \$541.6 million, which was lower than in 2022.

****Commodity Price Risk:**** The company also deals with the prices of recycled materials, like old cardboard and newspapers. Changes in supply and demand can cause these prices to fluctuate. They have previously used financial tools to manage this risk but had no hedges in place as of December 31, 2023. A \$10 change in the price of recycled materials could affect their revenue and operating income by about \$10 million. In 2023, they earned \$312.3 million from recycling, down from \$359.1 million in 2022.

Overall, the report highlights how the company is working to manage these financial risks to protect its operations and profitability.

```
scores = collect_rouge(target=summary[gpt_name], prediction=summary['simple_gpt-4o'])
```

Display rouge-1 scores for simple GPT-4o-mini summary

```
display_rouge("rouge1", scores)
```

ROUGE1 metric:

	precision	recall	fmeasure
PACCAR INC	0.653571	0.612040	0.632124
PHILLIPS 66	0.551971	0.536585	0.544170
MASTERCARD INC	0.560261	0.581081	0.570481
BRISTOL MYERS SQUIBB CO	0.550000	0.542763	0.546358
CARRIER GLOBAL CORP	0.551155	0.575862	0.563238
LULULEMON ATHLETICA INC	0.576577	0.657534	0.614400
AIRBNB INC	0.658621	0.667832	0.663194
MERCADOLIBRE INC	0.553333	0.544262	0.548760
A T & T INC	0.665480	0.656140	0.660777
REPUBLIC SERVICES INC	0.650794	0.667752	0.659164
average	0.597176	0.604185	0.600267

Display rouge-2 scores for simple GPT-4o-mini summary

```
display_rouge("rouge2", scores)
```

ROUGE2 metric:

	precision	recall	fmeasure
PACCAR INC	0.336918	0.315436	0.325823
PHILLIPS 66	0.244604	0.237762	0.241135
MASTERCARD INC	0.241830	0.250847	0.246256
BRISTOL MYERS SQUIBB CO	0.257525	0.254125	0.255814
CARRIER GLOBAL CORP	0.231788	0.242215	0.236887
LULULEMON ATHLETICA INC	0.243976	0.278351	0.260032
AIRBNB INC	0.359862	0.364912	0.362369
MERCADOLIBRE INC	0.254181	0.250000	0.252073
A T & T INC	0.257143	0.253521	0.255319
REPUBLIC SERVICES INC	0.382166	0.392157	0.387097
average	0.280999	0.283933	0.282281

33.3.6 Readability

Readability scores such as **Flesch-Kincaid** and **Gunning-Fog** assess how easy a summary is to read. These metrics are calculated based on sentence length, word complexity, and syllable count, and correspond to U.S. grade levels. Simpler summaries which score lower are suitable for broader audiences.

```
# applies Flesch-Kincaid and Gunning-Fog readability indexes to measure how complex
# each summary is
from readability import Readability
fog = {permno: {name: Readability(summary[name][permno]).flesch_kincaid().grade_level
               for name in ['simple_deepseek', 'simple_gpt-4o', model_name, gpt_
               name]}
      for permno in summary[gpt_name].keys()}
DataFrame(fog).T # display grade-level
```

	simple_deepseek	simple_gpt-4o	deepseek-r1:14b	gpt-4o-mini
60506	7	9	13	15
13356	12	10	12	17
91233	10	10	16	16
19393	9	10	15	15
19285	8	10	17	16
92203	9	11	15	15
20190	9	12	15	17
92221	12	10	12	16
66093	16	12	16	16
86228	9	8	10	12

```
fog = {permno: {name: Readability(summary[name][permno]).gunning_fog().grade_level
               for name in ['simple_deepseek', 'simple_gpt-4o', model_name, gpt_
               name]}
      for permno in summary[gpt_name].keys()}
DataFrame(fog).T # display grade-level
```

	simple_deepseek	simple_gpt-4o	deepseek-r1:14b	gpt-4o-mini
60506	9	12	college_graduate	college_graduate
13356	college	college	college	college_graduate
91233	12	12	college_graduate	college_graduate

(continues on next page)

(continued from previous page)

19393	12	college	college_graduate	college_graduate
19285	11	college	college_graduate	college_graduate
92203	12	college	college_graduate	college_graduate
20190	12	college	college_graduate	college_graduate
92221	12	college		12 college_graduate
66093	college_graduate	college	college_graduate	college_graduate
86228	12		12 college	college

References:

Greg Durrett, 2021-2024, “CS388 Natural Language Processing course materials”, retrieved from <https://www.cs.utexas.edu/~gdurrett/courses/online-course/materials.html>

Philipp Krähenbühl, 2020-2024, “AI394T Deep Learning course materials”, retrieved from https://www.philkr.net/dl_class/material and <https://ut.philkr.net/deeplearning/>

Philipp Krähenbühl, 2025, “AI395T Advances in Deep Learning course materials”, retrieved from https://ut.philkr.net/advances_in_deeplearning/

CHAPTER
THIRTYFOUR

FINE-TUNING

To improve is to change; to be perfect is to change often - Winston Churchill

Large language models (LLMs) have demonstrated remarkable general capabilities, but tailoring them to specific tasks or domains may require fine-tuning – adjusting model weights by further training on task-specific data. We examine the fine-tuning of Meta’s **Llama-3.1** model using tools from the Hugging Face ecosystem, applying efficient techniques such as quantization and low-rank adaptation (LoRA) to an industry text classification task using firm-level 10-K filings.

```
# By: Terence Lim, 2020-2025 (terence-lim.github.io)
import numpy as np
import pandas as pd
from pandas import DataFrame, Series
import os
from tqdm import tqdm
from pathlib import Path
from pprint import pprint
import textwrap
import warnings
import bitsandbytes as bnb
import torch
from datasets import Dataset
from peft import LoraConfig, PeftConfig
from trl import SFTTrainer
from transformers import (AutoModelForCausalLM,
                          AutoTokenizer,
                          BitsAndBytesConfig,
                          pipeline,
                          logging)
import matplotlib.pyplot as plt
from sklearn.metrics import (accuracy_score,
                             classification_report,
                             confusion_matrix)
from sklearn.model_selection import train_test_split
from finds.database import SQL, RedisDB
from finds.unstructured import Edgar
from finds.structured import BusDay, CRSP, PSTAT
from finds.readers import Sectoring
from finds.utils import Store
from secret import paths, CRSP_DATE, credentials
logging.set_verbosity_error()
```

```
NUM_TRAIN_EPOCHS = 2    # 0 # 1
RESUME_FROM_CHECKPOINT = False    # False # True
```

(continues on next page)

(continued from previous page)

```
MAX_SEQ_LENGTH = 1024 #512 #2048
LOGGING_STEPS = 200
```

```
VERBOSE = 0
sql = SQL(**credentials['sql'], verbose=VERBOSE)
bd = BusDay(sql)
rdb = RedisDB(**credentials['redis'])
crsp = CRSP(sql, bd, rdb, verbose=VERBOSE)
pstat = PSTAT(sql, bd, verbose=VERBOSE)
ed = Edgar(paths['10X'], zipped=True, verbose=0)
store = Store('assets', ext='pkl')
permnos = list(store.load('nouns').keys())
print(f"len(permnos)={len(permnos)}") # comparable sample
```

```
len(permnos)=3474
```

34.1 Meta Llama-3.1 model

Meta's **Llama 3.1** is an open-source large language model released in July 2024 under the Llama 3.1 Community License, permitting broad use, including commercial applications. Key highlights include:

- Model variants:
 - 8B: 8 billion parameters.
 - 70B: 70 billion parameters.
 - 405B: 405 billion parameters.
- Context length of up to 128,000 tokens.
- Pre-trained on over 15 trillion tokens sourced from publicly available datasets.
- Fine-tuned using supervised fine-tuning (SFT) and reinforcement learning with human feedback (RLHF).
- Multilingual support, including English, French, German, Hindi, Italian, Portuguese, Spanish, and Thai.

<https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct>

```
base_model = 'meta-llama/Llama-3.1-8B-Instruct'
```

```
# Show current memory stats
gpu_stats = torch.cuda.get_device_properties(0)
max_memory = round(gpu_stats.total_memory / (1024**3), 3)
print(f"GPU = {gpu_stats.name}. Max memory = {max_memory} GB.")

def cuda_memory(title, trainer_stats=None):
    """Show final memory and optional trainer stats"""
    if torch.cuda.is_available():
        device = torch.device('cuda')
        total_memory = torch.cuda.get_device_properties(device).total_memory
        reserved_memory = torch.cuda.memory_reserved(device)
        allocated_memory = torch.cuda.memory_allocated(device)
        free_memory = total_memory - reserved_memory
```

(continues on next page)

(continued from previous page)

```

print(f'----- {title.upper()} -----')
if trainer_stats:
    print(f"trainer_stats.metrics['train_runtime'] } seconds used for"
        ↪training.")
    print(f"Total memory: {total_memory / (1024**3):.2f} GB")
    print(f"Reserved memory: {reserved_memory / (1024**3):.2f} GB")
    print(f"Allocated memory: {allocated_memory / (1024**3):.2f} GB")
    print(f"Free memory: {free_memory / (1024**3):.2f} GB")

```

GPU = NVIDIA GeForce RTX 3080 Laptop GPU. Max memory = 15.739 GB.

34.2 Supervised fine-tuning (SFT)

Supervised Fine-Tuning is the process of enhancing a pre-trained language model by fine-tuning it on labeled input–output pairs using standard supervised learning. Common use cases include:

- Instruction tuning: The model learns to follow new instructions
- Chatbot fine-tuning (e.g., with help-desk data)
- Domain adaptation (e.g., legal, medical)

34.2.1 Huggingface framework

Several ecosystems support fine-tuning and training of LLMs. The Hugging Face Ecosystem includes:

- `transformers`: Model architectures and training components.
- `Transformers Reinforcement Learning (trl)`: Training large language models (LLMs) with reinforcement learning techniques, especially for alignment tasks like RLHF (Reinforcement Learning with Human Feedback) and DPO (Direct Preference Optimization).
- `bitsandbytes`: Enables efficient low-bit model quantization, allowing large language models to run on limited GPU memory without much loss in performance.
- `Parameter-Efficient Fine-Tuning (peft)`: Tools to fine-tune large language models by training only a small number of additional parameters.
- `Accelerate`: Distributed training optimization.
- `datasets`: For loading, processing, and managing datasets

It provides access to 100k+ pre-trained transformer models, and tools for efficient-tuning of these models using low memory and quantized weights.

If you encounter a gated model repository on Hugging Face, it means the model requires manual access approval from the authors before you can use or download it. You should log in to your `huggingface.ro` account, go to the Model Page, and click on the “Request Access” button – approval may take up to a few days. When authorized, make sure you have set your Hugging Face token in your environment (e.g. `huggingface-cli login`), see <https://huggingface.co/settings/tokens>

```

# Locations to save fine-tuned model weights
output_dir = str(Path(paths['scratch'], "fine-tuned-model"))    # training checkpoints
model_dir = str(Path(paths['scratch'], "Llama-3.1-8B-Instruct-FF-Sector"))  # final
    ↪model

```

```

from trl import SFTConfig
args = SFTConfig(
    output_dir=output_dir,                                     # directory to save and repository id
    num_train_epochs=NUM_TRAIN_EPOCHS, #####1                 # number of training epochs
    per_device_train_batch_size=2, #####1                   # batch size per device during training
    gradient_accumulation_steps=4, #####8                  # before performing a backward/update_
    pass
    gradient_checkpointing=True,                            # use gradient checkpointing to save_
    memory
    optim="paged_adamw_32bit",                            # or "steps" or "no" or "epoch"
    logging_strategy="steps",                            # learning rate, based on QLoRA paper
    logging_steps=LOGGING_STEPS, ##### 1,
    learning_rate=2e-4,                                    # learning rate, based on QLoRA paper
    weight_decay=0.001,
    fp16=True,
    bf16=False,
    max_grad_norm=0.3,                                    # max gradient norm based on QLoRA paper
    max_steps=-1,                                         # warmup ratio based on QLoRA paper
    warmup_ratio=0.03,
    group_by_length=False,                                # use cosine learning rate scheduler
    lr_scheduler_type="cosine",
    report_to="tensorboard",
    max_seq_length=MAX_SEQ_LENGTH, #512, ##### should be 1024? or MAX_CHARS // 4
    packing=False,
    dataset_kwarg={
        "add_special_tokens": False,
        "append_concat_token": False,
    }
)

```

34.2.2 Tokenizer

The AutoTokenizer in Hugging Face is a smart utility that automatically loads the correct tokenizer for a given pretrained model.

```

# Load the tokenizer and set the pad token id.
tokenizer = AutoTokenizer.from_pretrained(base_model)
tokenizer.pad_token_id = tokenizer.eos_token_id

```

34.2.3 Quantization

Quantization converts high-precision data to lower-precision data, for instance, by representing model weights and activation values as 4-bit or 8-bit integers instead of 32-bit floating point numbers. The bitsandbytes library for efficient low-bit model quantization is integrated with Hugging Face and works seamlessly with parameter-efficient fine-tuning like QLora.

```

# Load the Llama-3.1-8b-instruct model in 4-bit quantization to save GPU memory
bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_use_double_quant=False,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype="float16",
)

```

34.2.4 AutoModel

The `AutoModel` class in Hugging Face is a convenient interface that automatically loads the correct model architecture based on the model name or path. Its variants automatically load the correct model head (e.g., classification layer, decoder head) based on your specific task, e.g.

Class	Task	Output
<code>AutoModel</code>	Base model (no head)	Hidden states
<code>AutoModelForSequenceClassification</code>	Text classification (e.g. sentiment)	Class logits
<code>AutoModelForTokenClassification</code>	Token labeling (e.g. NER, POS)	Token-level logits
<code>AutoModelForQuestionAnswering</code>	Extractive QA	Start/end logits for answer spans
<code>AutoModelForCausalLM</code>	Text generation (GPT-style)	Next-token logits
<code>AutoModelForMaskedLM</code>	Mask filling (BERT-style)	Predictions for masked tokens
<code>AutoModelForSeq2SeqLM</code>	Translation, summarization (T5, BART)	Generated sequences
<code>AutoModelForMultipleChoice</code>	Multiple-choice QA (e.g. SWAG)	Choice logits
<code>AutoModelForVision2Seq</code>	Image captioning	Generated text
<code>AutoModelForImageClassification</code>	Vision tasks	Class logits
<code>AutoModelForSpeechSeq2Seq</code>	Speech translation	Generated text from audio

```
model = AutoModelForCausalLM.from_pretrained(
    base_model,
    device_map="auto",
    torch_dtype="float16",
    quantization_config=bnb_config,
)
model.config.use_cache = False
model.config.pretraining_tp = 1
```

34.2.5 Parameter-efficient fine-tuning

Parameter-Efficient Fine-Tuning (PEFT) is both a technique and a Hugging Face library for adapting large language models (LLMs) to new tasks by training only a small subset of parameters. Instead of updating the entire model, the base (pretrained) model is kept frozen, and lightweight, trainable components called **adapters** are added. These adapters typically involve only a few million parameters, making fine-tuning faster and more memory-efficient.

- **Low-rank factorization:** This is a compression technique which decomposes a large matrix of weights into a smaller, lower-rank matrix, resulting in a more compact approximation that requires fewer parameters and computations.
- **LoRA:** A small number of trainable low-rank matrices are added to the model's attention layers. The original weights are frozen and just these adapters are fine-tuned.
- **QLora:** Combines LoRA with Quantization: The base model is converted to 4-bit precision, reducing memory usage dramatically without losing much performance.

```
# Extract the linear module names from the model using the bits and bytes library.
def find_all_linear_names(model):
    cls = bnb.nn.Linear4bit
    lora_module_names = set()
    for name, module in model.named_modules():
        if isinstance(module, cls):
            names = name.split('.')
            lora_module_names.add(names[0] if len(names) == 1 else names[-1])
    if 'lm_head' in lora_module_names: # needed for 16 bit
        lora_module_names.remove('lm_head')
    return list(lora_module_names)
modules = find_all_linear_names(model)
modules
```

```
['q_proj', 'down_proj', 'v_proj', 'gate_proj', 'o_proj', 'up_proj', 'k_proj']
```

```
# Configure LoRA for the target modules, task type, and other training arguments
peft_config = LoraConfig(
    lora_alpha=16,
    lora_dropout=0,
    r=64,
    bias="none",
    task_type="CAUSAL_LM",
    target_modules=modules,
)
```

34.3 Industry text classification

We fine-tune the model for classifying firms into ten Fama-French sector categories based on their business descriptions in 10-K filings. The text data for each U.S.-domiciled common stock is drawn from the most recent year's Business Description section of their 10-K filings.

Load 10-K business description text for industry classification task

```
# Retrieve universe of stocks
beg, end = bd.begyr(CRSP_DATE), bd.endyr(CRSP_DATE)
print(f" {beg=}, {end=}")
univ = crsp.get_universe(bd.endyr(CRSP_DATE, -1))

# lookup company names
comnam = crsp.build_lookup(source='permno', target='comnam', fillna="")
univ['comnam'] = comnam(univ.index)

# lookup company names
comnam = crsp.build_lookup(source='permno', target='comnam', fillna="")
univ['comnam'] = comnam(univ.index)

# lookup ticker symbols
ticker = crsp.build_lookup(source='permno', target='ticker', fillna="")
univ['ticker'] = ticker(univ.index)

# lookup sic codes from Compustat, and map to FF 10-sector code
sic = pstat.build_lookup(source='lpermno', target='sic', fillna=0)
```

(continues on next page)

(continued from previous page)

```

industry = Series(sic[univ.index], index=univ.index)
industry = industry.where(industry > 0, univ['siccd'])
sectors = Sectoring(sql, scheme='codes10', fillna='')    # supplement from crosswalk
univ['sector'] = sectors[industry]

# retrieve latest year's bus10K's
item, form = 'bus10K', '10-K'
rows = DataFrame(ed.open(form=form, item=item))
rows = rows[rows['date'].between(beg, end)]\ 
    .drop_duplicates(subset=['permno'], keep='last')\ 
    .set_index('permno')\ 
    .reindex(permnos)

# split documents into train/test sets
labels = univ.loc[permnos, 'sector']
class_labels = np.unique(labels)
print(f" {class_labels=}")

train_index, test_index = train_test_split(permnos,
                                            stratify=labels,
                                            random_state=42,
                                            test_size=0.2)

```

```

beg=20240102, end=20241231
class_labels=array(['Durbl', 'Enrgy', 'HiTec', 'Hlth', 'Manuf', 'NoDur', 'Other',
                   'Shops', 'Telcm', 'Utils'], dtype=object)

```

34.3.1 HuggingFace dataset module

The training data are converted to LLM instruction statements, and implemented as a HuggingFace Dataset class. This class can be conveniently created from many different sources, including data files of various formats or from a generator function.

```

# Create LLM instruction statement
MAX_CHARS = MAX_SEQ_LENGTH * 2
class_text = """ + "' or '".join(class_labels) + """
def generate_prompt(permno, test=False):
    text = ed[rows.loc[permno, 'pathname']].replace('\n', '') [:MAX_CHARS]
    return f"""
Classify the text into one of these {len(class_labels)} classification labels:
{class_text}
and return the answer as the label.
text: {text}
label: {'' if test else univ.loc[permno, 'sector']}""".strip()

```

```
cuda_memory('before dataset')
```

```

----- BEFORE DATASET -----
Total memory: 15.74 GB
Reserved memory: 6.83 GB
Allocated memory: 5.63 GB
Free memory: 8.91 GB

```

```

X_train = DataFrame(columns=['text'], index=train_index,
                     data=[generate_prompt(permno, test=False) for permno in train_
                           →index])
X_test = DataFrame(columns=['text'], index=test_index,
                     data=[generate_prompt(permno, test=True) for permno in test_index])
y_test = [univ.loc[permno, 'sector'] for permno in test_index]

train_data = Dataset.from_pandas(X_train[["text"]])
test_data = Dataset.from_pandas(X_test[["text"]])
print(textwrap.fill(train_data['text'][3]))

```

Classify the text into one of these 10 classification labels: 'Durbl' or 'Energy' or 'HiTec' or 'Hlth' or 'Manuf' or 'NoDur' or 'Other' or 'Shops' or 'Telcm' or 'Utils' and return the answer as the label.

text: ITEM 1. BUSINESS OVERVIEW B. RILEY FINANCIAL, INC. (NASDAQ: RILY) (THE COMPANY IS A DIVERSIFIED FINANCIAL SERVICES PLATFORM THAT DELIVERS TAILORED SOLUTIONS TO MEET THE STRATEGIC, OPERATIONAL, AND CAPITAL NEEDS OF ITS CLIENTS AND PARTNERS. WE OPERATE THROUGH SEVERAL CONSOLIDATED SUBSIDIARIES (COLLECTIVELY, B. RILEY THAT PROVIDE INVESTMENT BANKING, BROKERAGE, WEALTH MANAGEMENT, ASSET MANAGEMENT, DIRECT LENDING, BUSINESS ADVISORY, VALUATION, AND ASSET DISPOSITION SERVICES TO A BROAD CLIENT BASE SPANNING PUBLIC AND PRIVATE COMPANIES, FINANCIAL SPONSORS, INVESTORS, FINANCIAL INSTITUTIONS, LEGAL AND PROFESSIONAL SERVICES FIRMS, AND INDIVIDUALS. THE COMPANY OPPORTUNISTICALLY INVESTS IN AND ACQUIRES COMPANIES OR ASSETS WITH ATTRACTIVE RISK-ADJUSTED RETURN PROFILES TO BENEFIT OUR SHAREHOLDERS. WE OWN AND OPERATE SEVERAL UNCORRELATED CONSUMER BUSINESSES AND INVEST IN BRANDS ON A PRINCIPAL BASIS. OUR APPROACH IS FOCUSED ON HIGH QUALITY COMPANIES AND ASSETS IN INDUSTRIES IN WHICH WE HAVE EXTENSIVE KNOWLEDGE AND CAN BENEFIT FROM OUR EXPERIENCE TO MAKE OPERATIONAL IMPROVEMENTS AND MAXIMIZE FREE CASH FLOW. OUR PRINCIPAL INVESTMENTS OFTEN LEVERAGE THE FINANCIAL, RESTRUCTURING, AND OPERATIONAL EXPERTISE OF OUR PROFESSIONALS WHO WORK COLLABORATIVELY ACROSS DISCIPLINES. WE REFER TO B. RILEY AS A PLATFORM BECAUSE OF THE UNIQUE COMPOSITION OF OUR BUSINESS. OUR PLATFORM HAS GROWN CONSIDERABLY AND BECOME MORE DIVERSIFIED OVER THE PAST SEVERAL YEARS. WE HAVE INCREASED OUR MARKET SHARE AND EXPANDED THE DEPTH AND BREADTH OF OUR BUSINESSES BOTH ORGANICALLY AND THROUGH OPPORTUNISTIC ACQUISITIONS. OUR INCREASINGLY DIVERSIFIED PLATFORM ENABLES US TO INVEST OPPORTUNISTICALLY AND TO DELIVER STRONG LONG-TERM INVESTMENT PERFORMANCE THROUGHOUT A RANGE OF ECONOMIC CYCLES. OUR PLATFORM IS COMPRISED OF MORE THAN 2,700 AFFILIATED PROFESSIONALS, INCLUDING EMPLOYEES AND INDEPENDENT CONTRACTORS. WE ARE HEADQUARTERED IN LOS ANGELES, CALIFORNIA AND MAINTAIN OFFICES THROUGHOUT THE U.S., INCLUDING IN NEW YORK, CHICAGO, METRO DISTRICT OF COLUMBIA, AT label: Other

```

# verify max_seq_length sufficient
curr_max = 0
for row, data in enumerate(train_data):
    tokenized = tokenizer.tokenize(data['text'])
    curr_max = max(curr_max, len(tokenized))
#    print(f"row={row}, {len(tokenized)}")
assert curr_max < args.max_seq_length
print(curr_max, f"MAX_SEQ_LENGTH={args.max_seq_length}")

```

```
820 MAX_SEQ_LENGTH=1024
```

```
cuda_memory('after dataset')
```

```
----- AFTER DATASET -----
Total memory: 15.74 GB
Reserved memory: 6.83 GB
Allocated memory: 5.63 GB
Free memory: 8.91 GB
```

34.3.2 Pipeline

Hugging Face's `pipeline` function enables one-line use for easy inference, by simply specifying the model, tokenizer, generation parameters (e.g. sampling methodology, maximum new tokens), and task, e.g.:

- “text-classification”: Sentiment analysis, topic labeling
- “token-classification”: Named Entity Recognition (NER), POS tagging
- “question-answering”: Extractive QA from context
- “text-generation”: Generate text (GPT-style)
- “summarization”: Generate summaries from long text

```
# Use the text generation pipeline to predict labels from the "text"
def generate(prompt, model=model, tokenizer=tokenizer, verbose=False):
    """Generate a response"""
    pipe = pipeline(task="text-generation",
                    model=model,
                    tokenizer=tokenizer,
                    do_sample=False,
                    top_p=None,
                    top_k=None,
                    return_full_text=False,
                    max_new_tokens=4,    # 2
                    temperature=None)    # 0.1
    result = pipe(prompt)
    answer = result[0]['generated_text'].split("label:")[-1].strip()
    if verbose:
        print(f"len(prompt)={}, result={}, answer={}")
    return answer

def predict(test, model, tokenizer, verbose=False):
    """Predict test set"""
    y_pred = []
    for i in tqdm(range(len(test))):
        prompt = test.iloc[i]["text"]
        answer = generate(prompt, model, tokenizer, verbose=verbose)
        # Determine the predicted category
        for category in class_labels:
            if category.lower() in answer.lower():
                y_pred.append(category)
                break
        else:
```

(continues on next page)

(continued from previous page)

```
    y_pred.append("none")
return y_pred
```

Create function that will use the predicted labels and true labels to compute the overall accuracy, classification report, and confusion matrix.

```
def evaluate(y_true, y_pred):
    mapping = {label: idx for idx, label in enumerate(class_labels)}

    def map_func(x):
        return mapping.get(x, -1) # Map to -1 if not found, should not occur with
    ↵correct data

    y_true_mapped = np.vectorize(map_func)(y_true)
    y_pred_mapped = np.vectorize(map_func)(y_pred)
    labels = list(mapping.values())
    target_names = list(mapping.keys())
    if -1 in y_pred_mapped:
        labels += [-1]
        target_names += ['none']

    # Calculate accuracy
    accuracy = accuracy_score(y_true=y_true_mapped, y_pred=y_pred_mapped)
    print(f'Accuracy: {accuracy:.3f}')

    # Generate classification report
    class_report = classification_report(y_true=y_true_mapped, y_pred=y_pred_mapped,
                                           target_names=target_names,
                                           labels=labels, zero_division=0.0)
    print('\nClassification Report:')
    print(class_report)

    # Generate confusion matrix
    conf_matrix = confusion_matrix(y_true=y_true_mapped, y_pred=y_pred_mapped,
                                   labels=labels)
    print('\nConfusion Matrix:')
    print(conf_matrix)
```

Evaluate accuracy before fine-tuning the model

```
y_pred = predict(X_test, model, tokenizer)
Series(y_pred).value_counts()
```

100% |██████████| 695/695 [05:45<00:00, 2.01it/s]

Manuf	217
NoDur	184
HiTec	109
Other	65
none	54
Hlth	24
Utils	15
Telcm	14
Shops	8

(continues on next page)

(continued from previous page)

```
Enrgy      4
Durbl      1
Name: count, dtype: int64
```

```
evaluate(y_test, y_pred)
```

Accuracy: 0.203

Classification Report:

	precision	recall	f1-score	support
Durbl	0.00	0.00	0.00	33
Enrgy	0.50	0.10	0.17	20
HiTec	0.25	0.19	0.22	139
Hlth	0.88	0.13	0.22	164
Manuf	0.22	0.70	0.34	69
NoDur	0.03	0.21	0.06	28
Other	0.25	0.10	0.15	153
Shops	0.75	0.10	0.17	62
Telcm	0.36	0.56	0.43	9
Utils	0.67	0.56	0.61	18
none	0.00	0.00	0.00	0
accuracy			0.20	695
macro avg	0.35	0.24	0.21	695
weighted avg	0.44	0.20	0.21	695

Confusion Matrix:

```
[[ 0  0  2  0 18 10  2  1  0  0  0  0]
 [ 0  2  0  0  8  8  2  0  0  0  0  0]
 [ 0  0 27  0 43 38 18  0  8  4  1]
 [ 0  0 73 21 15 22 11  0  1  1 20]
 [ 0  0  3  0 48 12  5  1  0  0  0  0]
 [ 1  0  0  0 17  6  4  0  0  0  0  0]
 [ 0  0  4  1 41 64 16  0  0  0 27]
 [ 0  0  0  1 26 16  7  6  0  0  6]
 [ 0  0  0  0  0  4  0  0  5  0  0]
 [ 0  2  0  1  1  4  0  0  0 10  0]
 [ 0  0  0  0  0  0  0  0  0  0  0]]
```

34.3.3 Trainer

Create the model trainer using training arguments, a LoRA configuration, and a dataset.

```
trainer = SFTTrainer(
    model=model,
    args=args,
    train_dataset=train_data,
    peft_config=peft_config,
    #   dataset_text_field="text",
    processing_class=tokenizer
)
```

```
# Initiate model training
cuda_memory('before training')
trainer_stats = trainer.train(resume_from_checkpoint=RESUME_FROM_CHECKPOINT)

----- BEFORE TRAINING -----
Total memory: 15.74 GB
Reserved memory: 11.04 GB
Allocated memory: 8.22 GB
Free memory: 4.70 GB
{'loss': 1.1984, 'grad_norm': 0.1371612697839737, 'learning_rate': 0.
 ↪0001670747898848231, 'num_tokens': 1091299.0, 'mean_token_accuracy': 0.
 ↪7146163220703602, 'epoch': 0.5755395683453237}
{'loss': 1.1205, 'grad_norm': 0.16719305515289307, 'learning_rate': 8.
 ↪029070592154895e-05, 'num_tokens': 2179799.0, 'mean_token_accuracy': 0.
 ↪7273549642927366, 'epoch': 1.1496402877697842}
{'loss': 1.034, 'grad_norm': 0.19266854226589203, 'learning_rate': 9.
 ↪47361624665869e-06, 'num_tokens': 3270551.0, 'mean_token_accuracy': 0.
 ↪7437317748367787, 'epoch': 1.725179856115108}
{'train_runtime': 9602.662, 'train_samples_per_second': 0.579, 'train_steps_per_
↪second': 0.072, 'train_loss': 1.103452600044888, 'num_tokens': 3784895.0, 'mean_
↪token_accuracy': 0.7479746815689067, 'epoch': 1.99568345323741}
```

```
# Save trained model and tokenizer
model.config.use_cache = True
trainer.save_model(output_dir)
tokenizer.save_pretrained(output_dir)
cuda_memory('after training', trainer_stats=trainer_stats)
```

```
----- AFTER TRAINING -----
9602.662 seconds used for training.
Total memory: 15.74 GB
Reserved memory: 14.43 GB
Allocated memory: 8.26 GB
Free memory: 1.31 GB
```

34.3.4 Evaluation

```
y_pred = predict(X_test, model, tokenizer, verbose=False)
Series(y_pred).value_counts()
```

```
0% | 0/695 [00:00<?, ?it/s]/home/terence/env3.11/lib/python3.11/site-
↪packages/torch/utils/checkpoint.py:87: UserWarning: None of the inputs have_
↪requires_grad=True. Gradients will be None
warnings.warn(
100%|██████████| 695/695 [08:21<00:00, 1.39it/s]
```

Hlth	168
Other	156
HiTec	140
Manuf	59
Shops	59

(continues on next page)

(continued from previous page)

```
NoDur      34
Durbl     29
Enrgy     21
Utils     19
Telcm     10
Name: count, dtype: int64
```

```
evaluate(y_test, y_pred)
```

Accuracy: 0.829

Classification Report:

	precision	recall	f1-score	support
Durbl	0.83	0.73	0.77	33
Enrgy	0.90	0.95	0.93	20
HiTec	0.79	0.80	0.80	139
Hlth	0.89	0.91	0.90	164
Manuf	0.80	0.68	0.73	69
NoDur	0.59	0.71	0.65	28
Other	0.85	0.86	0.85	153
Shops	0.83	0.79	0.81	62
Telcm	0.90	1.00	0.95	9
Utils	0.84	0.89	0.86	18
accuracy			0.83	695
macro avg	0.82	0.83	0.83	695
weighted avg	0.83	0.83	0.83	695

Confusion Matrix:

```
[[ 24   0   6   0   1   1   0   1   0   0]
 [  0  19   0   0   1   0   0   0   0   0]
 [  0   2 111   7   2   2  12   2   1   0]
 [  0   0   9 149   1   1   3   1   0   0]
 [  5   0   4   2  47   6   2   2   0   1]
 [  0   0   0   1   2  20   2   3   0   0]
 [  0   0  10   4   5   1 132   1   0   0]
 [  0   0   0   4   0   3   4  49   0   2]
 [  0   0   0   0   0   0   0   0   9   0]
 [  0   0   0   1   0   0   1   0   0  16]]
```

```
# merge and save model
from transformers import AutoModelForCausalLM, AutoTokenizer
from peft import PeftModel

del model
del trainer
torch.cuda.empty_cache()
cuda_memory('after empty')
```

```
# Reload base model and tokenizer to cpu
device_map = "cpu"
```

(continues on next page)

(continued from previous page)

```
tokenizer = AutoTokenizer.from_pretrained(base_model)
base_model_reload = AutoModelForCausalLM.from_pretrained(
    base_model,
    return_dict=True,
    low_cpu_mem_usage=True,
    torch_dtype=torch.float16,
    device_map=device_map, # "cpu", # "auto",
    trust_remote_code=True,
)
```

```
# Merge adapter with base model
from peft import PeftModel
model = PeftModel.from_pretrained(base_model_reload, output_dir, device_map=device_map)
model = model.merge_and_unload()
```

```
# Save the merged model
model.save_pretrained(model_dir)
tokenizer.save_pretrained(model_dir)
```

```
# Reload merged model and tokenizer
tokenizer = AutoTokenizer.from_pretrained(model_dir)
base_model_reload = AutoModelForCausalLM.from_pretrained(
    model_dir,
    return_dict=True,
    low_cpu_mem_usage=True,
    torch_dtype=torch.float16,
    device_map="auto", # 'cpu',
    trust_remote_code=True,
)
```

```
# Check it is working
y_pred = predict(X_test, model, tokenizer)
evaluate(y_test, y_pred)
```

References:

Philipp Krähenbühl, 2025, “AI395T Advances in Deep Learning course materials”, retrieved from https://ut.philkr.net/advances_in_deeplearning/

Tim Dettmers, “Bitsandbytes: 8-bit Optimizers and Quantization for PyTorch”, 2022. GitHub repository: <https://github.com/TimDettmers/bitsandbytes>

<https://www.datacamp.com/tutorial/fine-tuning-llama-3-1>

PROMPTING

The important thing is not to stop questioning. Curiosity has its own reason for existing - Albert Einstein

We explore how different prompting strategies influence the performance of large language models (LLMs) on the task of financial sentiment classification. We examine **zero-shot**, **few-shot**, and **chain-of-thought (CoT)** prompting techniques using Google's open-source Gemma-3 model deployed locally with Ollama. Careful design of prompts and enforcement of specific output formats improve the interpretability and reliability of LLM outputs. We also probe the model's vision and multi-lingual capabilities, by prompting it to analyze a chart image and respond in different languages.

```
# By: Terence Lim, 2020-2025 (terence-lim.github.io)
import pandas as pd
from pandas import DataFrame, Series
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
import re
import json
from pprint import pprint
import textwrap
from tqdm import tqdm
import ollama
from secret import paths
```

35.1 Sentiment analysis

Sentiment analysis has evolved significantly from its early reliance on lexicon-based methods, which used hand-crafted dictionaries to score sentiment. Supervised machine learning models brought improvements by learning from labeled examples. With the rise of deep learning, especially recurrent neural networks (RNNs), models gained the ability to capture contextual information and temporal dependencies in text. Transformer-based models, particularly large pre-trained language models, now dominate the field. These models can be fine-tuned or prompted to perform complex and domain-specific tasks with high accuracy and fluency.

35.1.1 Financial news sentiment

We use a labeled dataset compiled by Malo et al. (2014) containing financial news headlines annotated for sentiment from a retail investor's perspective. This dataset is hosted on **Kaggle**, a popular platform for sharing machine learning challenges and datasets.

<https://www.kaggle.com/datasets/ankurzing/sentiment-analysis-for-financial-news>

```
# Loads financial news headline data labeled with sentiment (positive, neutral, -negative)
news = pd.read_csv(paths['data'] / 'all-data.csv',
                    names=["sentiment", "text"],
                    encoding="utf-8",
                    encoding_errors="replace")
news
```

	sentiment	text
0	neutral	According to Gran , the company has no plans t...
1	neutral	Technopolis plans to develop in stages an area...
2	negative	The international electronic industry company ...
3	positive	With the new production plant the company woul...
4	positive	According to the company 's updated strategy f...
...
4841	negative	LONDON MarketWatch -- Share prices ended lower...
4842	neutral	Rinkuskiai 's beer sales fell by 6.5 per cent ...
4843	negative	Operating profit fell to EUR 35.4 mn from EUR ...
4844	negative	Net sales of the Paper segment decreased to EU...
4845	negative	Sales in Finland decreased by 10.5 % in Januar...

[4846 rows x 2 columns]

```
# Hold out 15 examples in the "training set" that are not used for evaluation
X_train, X_test, y_train, y_test = train_test_split(news['text'], news['sentiment'],
                                                random_state=0, train_size=15,
                                                stratify=news['sentiment'])
```

```
pd.concat([y_train, X_train], axis=1)
```

	sentiment	text
2044	neutral	The estimated turnover of the new company is L...
3066	neutral	On 25 August 2009 , Sampo 's stake in Nordea w...
331	positive	Finnish investment group Panostaja Oyj said it...
1228	positive	Furthermore , our fully electrically driven cr...
3049	neutral	No decision on such sale of the now issued or ...
165	positive	Both operating profit and net sales for the ni...
4803	negative	UPM-Kymmene Corp. , the world 's largest maker...
3381	neutral	The total value of the order , placed by Aspo ...
1450	neutral	Uponor maintains its full-year guidance for 20...
3484	neutral	`` These developments partly reflect the gover...
1588	positive	Renzo Piano 's building design will be a wond...
2330	neutral	Finnish L&T Recoil , a company specialising in...
2572	neutral	Stora Enso , a global paper , packaging and wo...
3887	neutral	The plant is scheduled for completion in late ...
1714	negative	TeliaSonera 's underlying results however incl...

```
targets = ['negative', 'neutral', 'positive']
```

35.2 Google Gemma 3 model

Google's **Gemma 3** is a collection of lightweight, open models designed for deployment across platforms from mobile devices to workstations. Key technical features include:

- Available in four configurations: 1B, 4B, 12B, and 27B parameters
- Multimodal capabilities: The 4B, 12B, and 27B models support both text and image inputs, while the 1B model is optimized for text-only applications.
- The 1B model supports a 32K-token context window, while the larger models (4B and above) provide a 128K-token context window.
- Multilingual support out-of-the-box for over 35 languages.

```
# Model name for Gemma-3-4B model in Ollama
model_name = "gemma3:12b"
```

35.3 Structured dialogue

35.3.1 Prompt engineering

Instead of explicit training, the model relies on its pre-trained knowledge to interpret and respond to a task based purely on how the prompt is phrased. Rather than modifying the model itself, prompt engineering uses careful wording to guide the model's behavior. Prompts should clearly explain the task and may specify the expected output format (e.g. JSON) for easier parsing. A well-crafted prompt can significantly improve performance, especially in zero- or few-shot settings.

To improve prompt results, start by assigning an identity to the LLM and sharing relevant background: this helps tailor the tone, detail, and relevance of the response. Be clear about the format you want, such as prose instead of bullet points, tables for comparisons, or timelines when useful. Mention how you'll use the output (e.g., for a class, presentation, or blog post) to influence tone and content. Specify the style or tone you prefer, like persuasive or direct, and clarify whether you're aiming for an essay or a casual post. When asking for rewrites, use specific instructions like "streamline" or "embellish" to guide the changes. Use the LLM to check for missing ideas in your writing, and be iterative with your questions to get more focused answers. For long articles, ask for summaries with a word or time limit. If you're editing your own work, ask to preserve your tone. In technical fields, instruct the model not to simplify or substitute specialized terms. You can also request focused insights or key takeaways. Lastly, to guard against hallucinations, ask for the model's confidence or sources.

While earlier models required extensive trial and error to find effective prompts, modern instruction-tuned LLMs like DeepSeek-R1 and GPT-4o understand natural language instructions more reliably, reducing the need for manual prompt tuning.

35.3.2 Structured outputs

LLMs naturally generate free-form text, which can be unpredictable and hard to parse. Structured outputs, such as JSON, enable downstream integration and ensure consistency. Methods for achieving structured output include:

- In-context demonstrations of expected format.
- Function calling / tool use, where the LLM generates function calls.
- Constrained decoding, which enforces valid output formats but requires custom decoders or grammars.

```
# Extracts a structured JSON response from the LLM's output
def parse_json(s):
    def padding(t):
        if t.count('}') < t.count('{'):
            t += '}'
        if s.count(']') < s.count('['):
            t += ']'
        return t
    s = s.replace("```json", "```")
    s = s.strip()
    s = padding(s)
    try:
        out = json.loads(s[s.index("{"):s.index("}")+1])
        assert "sentiment" in out
    except:
        out = {"sentiment": "neutral", "reasoning": ""}
    return out
```

35.3.3 Zero-shot prompt

Zero-shot prompting allows an LLM to perform a task without any task-specific tuning or examples: just a single instruction and input. The model relies entirely on its pretraining to infer the desired output.

```
# Simple, instruction-only prompt asking for a single-word sentiment label in JSON
def generate_prompt(text):
    return f"""
In one word only, provide the sentiment of the following text as either "positive" or
"neutral" or "negative".
Do not provide any other answer.
Provide your output in json format.
Text: '''{text}'''
sentiment:""".strip()
print(generate_prompt(news['text'].iloc[0]))
```

```
In one word only, provide the sentiment of the following text as either "positive" or
"neutral" or "negative".
Do not provide any other answer.
Provide your output in json format.
Text: '''According to Gran , the company has no plans to move all production to
Russia , although that is where the company is growing .'''
```

```
# Sends prompt to LLaMA3 via Ollama, with temperature=0 for deterministic output
pred0 = []
for i, (text, sentiment) in tqdm(enumerate(zip(X_test, y_test)), total=len(y_test)):
    s = generate_prompt(text)
    output = ollama.generate(model=model_name, prompt=s, options={"temperature":0})
    if i < 5:
        print(f"sentiment={sentiment}: {output['response']}")
    print()
    pred0.append(parse_json(output.response)['sentiment'].lower())
```

0% | 1/4831 [00:04<6:00:00, 4.47s/it]

```
sentiment='neutral': ````json
{
    "sentiment": "positive"
}
````
```

0% | 2/4831 [00:04<2:52:27, 2.14s/it]

```
sentiment='neutral': ````json
{
 "sentiment": "neutral"
}
````
```

0% | 4/4831 [00:05<1:11:40, 1.12it/s]

```
sentiment='negative': ````json
{
    "sentiment": "negative"
}
````
```

```
sentiment='positive': neutral
```

0% | 5/4831 [00:06<1:00:23, 1.33it/s]

```
sentiment='positive': ````json
{
 "sentiment": "positive"
}
````
```

100% | 4831/4831 [33:40<00:00, 2.39it/s]

```
# Evaluate predictions
print(f"Accuracy: {accuracy_score(y_true=y_test[:len(pred0)], y_pred=pred0):.3f}")
print(classification_report(y_true=y_test[:len(pred0)], y_pred=pred0))
```

(continues on next page)

(continued from previous page)

```
c = confusion_matrix(y_true=y_test[:len(pred0)], y_pred=pred0, labels=targets)
DataFrame(columns=pd.MultiIndex.from_product([['Predicted'], targets]),
          index=targets, data=c)      # display confusion matrix
```

| Accuracy: 0.799 | | | | |
|-----------------|-----------|--------|----------|---------|
| | precision | recall | f1-score | support |
| negative | 0.77 | 0.82 | 0.79 | 602 |
| neutral | 0.84 | 0.82 | 0.83 | 2870 |
| positive | 0.73 | 0.73 | 0.73 | 1359 |
| accuracy | | | 0.80 | 4831 |
| macro avg | 0.78 | 0.79 | 0.79 | 4831 |
| weighted avg | 0.80 | 0.80 | 0.80 | 4831 |

| Predicted | | | |
|-----------|----------|---------|----------|
| | negative | neutral | positive |
| negative | 493 | 108 | 1 |
| neutral | 136 | 2367 | 367 |
| positive | 12 | 349 | 998 |

35.3.4 In-context learning

Few-shot prompting, or **In-Context Learning (ICL)**, enhances performance by including labeled examples directly in the prompt. Instead of fine-tuning (and altering the pretrained neural network weights of) the model with new training examples, the model figures out how to perform well on that task simply by taking a few task-specific examples as input.

Even when examples are randomly selected, few-shot prompts often outperform zero-shot, as the examples provide additional context. However, the exact choice and order of examples can affect accuracy and variance in responses.

```
# Creates 15 prompt examples from training set
few_shots = [f"Text: '{text}'\nSentiment: {sentiment}"
             for text, sentiment in zip(X_train, y_train)]

# Concatenates few-shot examples before posing a new query
def generate_prompt(text, few_shots=few_shots):
    examples = "\n\n".join(few_shots)
    return f"""

Here are {len(few_shots)} examples of providing the sentiment based on the given text.

{examples}

In one word only, provide the sentiment of the following text as either "positive" or
→"neutral" or "negative".
Do not provide any other answer.
Text: '{text}'"
sentiment: """.strip()
print(generate_prompt(news['text'].iloc[0]))
```

Here are 15 examples of providing the sentiment based on the given text.

Text: '''The estimated turnover of the new company is LVL 2,5 million EEK 40_

(continues on next page)

(continued from previous page)

```

'million .'''  

Sentiment: neutral

Text: '''On 25 August 2009 , Sampo 's stake in Nordea was 19.45 % .'''  

Sentiment: neutral

Text: '''Finnish investment group Panostaja Oyj said its net profit went up to 8.6  

  ↵mln euro $ 11.4 mln in fiscal 2005-06 , ended October 31 , 2006 , from 2.8 mln  

  ↵euro $ 3.7 mln in the same period of fiscal 2004-05 .'''  

Sentiment: positive

Text: '''Furthermore , our fully electrically driven cranes are environmentally  

  ↵friendly .'''  

Sentiment: positive

Text: '''No decision on such sale of the now issued or existing treasury shares to  

  ↵YA Global has been made yet .'''  

Sentiment: neutral

Text: '''Both operating profit and net sales for the nine-month period increased ,  

  ↵respectively by 26.6 % and 3.4 % , as compared to the corresponding period in  

  ↵2006 .'''  

Sentiment: positive

Text: '''UPM-Kymmene Corp. , the world 's largest maker of magazine paper , on  

  ↵Tuesday reported a 19-percent profit drop as lower paper prices , higher costs  

  ↵and a strong euro hurt revenue .'''  

Sentiment: negative

Text: '''The total value of the order , placed by Aspo ' marine transportation  

  ↵subsidiary ESL Shipping Oy , is EUR 60 million ( USD 77.5 m ) .'''  

Sentiment: neutral

Text: '''Uponor maintains its full-year guidance for 2010 .'''  

Sentiment: neutral

Text: ''' These developments partly reflect the government 's higher activity in  

  ↵the field of dividend policy . '''  

Sentiment: neutral

Text: '''Renzo Piano 's building design will be a wonderful addition to London 's  

  ↵skyline , '' says Noud Veeger , EVP and Area Director for Central and North  

  ↵Europe at KONE .'''  

Sentiment: positive

Text: '''Finnish L&T Recoil , a company specialising in used oil regeneration , is  

  ↵building a facility in Hamina in Finland in 2008 .'''  

Sentiment: neutral

Text: '''Stora Enso , a global paper , packaging and wood products company , and  

  ↵Neste Oil , a Finnish company engaged in the refining and marketing of oil ,  

  ↵have inaugurated the demonstration plant at Varkaus , Finland for biomass to  

  ↵liquids production utilizing forestry residues .'''  

Sentiment: neutral

Text: '''The plant is scheduled for completion in late February 2007 with hand

```

(continues on next page)

(continued from previous page)

over of some areas in January Two other suppliers of Nokia - Aspocomp Group Oyj and Perlos - have announced their plans to establish plants within the Nokia complex Together , they will invest Rs 365 crore .'''
Sentiment: neutral

Text: '''TeliaSonera 's underlying results however included 457 mln skr in positive one-offs , hence the adjusted underlying EBITDA actually amounts to 7. 309 bln skr , clearly below expectations , analysts said .'''
Sentiment: negative

In one word only, provide the sentiment of the following text as either "positive" or "neutral" or "negative".

Do not provide any other answer.

Text: '''According to Gran , the company has no plans to move all production to Russia , although that is where the company is growing .'''
sentiment:

```
# Sends prompt demonstrating the desired response
pred1 = []
for i, (text, sentiment) in tqdm(enumerate(zip(X_test, y_test)), total=len(y_test)):
    s = generate_prompt(text)
    output = ollama.generate(model=model_name, prompt=s, options={"temperature":0})
    if i < 5:
        print(f"{sentiment}: {output['response']}")  
        print()
    pred1.append(output.response.strip().split('\n')[-1].lower())
```

0% | 0/4831 [00:00<?, ?it/s]

0% | 2/4831 [00:00<27:25, 2.93it/s]

sentiment='neutral': positive

sentiment='neutral': neutral

0% | 4/4831 [00:01<15:20, 5.25it/s]

sentiment='negative': negative

sentiment='positive': neutral

0% | 6/4831 [00:01<11:30, 6.99it/s]

sentiment='positive': neutral

100% | 4831/4831 [09:20<00:00, 8.61it/s]

```
# Evaluate predictions
print(f"Accuracy: {accuracy_score(y_true=y_test[:len(pred1)], y_pred=pred1):.3f}")
```

(continues on next page)

(continued from previous page)

```
print(classification_report(y_true=y_test[:len(pred1)], y_pred=pred1))
c = confusion_matrix(y_true=y_test[:len(pred1)], y_pred=pred1, labels=targets)
DataFrame(columns=pd.MultiIndex.from_product([['Predicted'], targets]),
          index=targets, data=c) # display confusion matrix
```

| Accuracy: 0.814 | | | | |
|-----------------|-----------|--------|----------|---------|
| | precision | recall | f1-score | support |
| negative | 0.80 | 0.94 | 0.86 | 602 |
| neutral | 0.84 | 0.85 | 0.85 | 2870 |
| positive | 0.76 | 0.67 | 0.71 | 1359 |
| accuracy | | | 0.81 | 4831 |
| macro avg | 0.80 | 0.82 | 0.81 | 4831 |
| weighted avg | 0.81 | 0.81 | 0.81 | 4831 |

| Predicted | | | |
|-----------|----------|---------|----------|
| | negative | neutral | positive |
| negative | 564 | 37 | 1 |
| neutral | 126 | 2451 | 293 |
| positive | 14 | 428 | 917 |

35.3.5 Chain-of-thought (CoT) prompting

Chain-of-thought (CoT) prompting encourages LLMs to reason step-by-step before making a final prediction. Instead of jumping to answers, LLMs can be prompted to explain their reasoning step by step. Introduced by Wei et al. (2022), CoT can be paired with few-shot prompts to improve performance on tasks requiring logical inference. By generating intermediate reasoning, the model delays its decision-making, reducing errors and hallucinations.

Extensions include:

- Self-Consistency: Voting over multiple CoT outputs.
- Tree of Thoughts (ToT): Exploring multiple reasoning paths.
- ReAct: Combining reasoning with tool use.
- Reflexion: Prompting the model to critique and revise its own answers.

```
# Expands prompt to include step-by-step instructions: extract terms, identify
#sentiment cues, provide reasoning
def generate_prompt(text, few_shots=few_shots):
    '''Evaluate the financial news by identifying the cause (e.g., earnings report,_
    economic policy,
market reaction) and its effect on investor sentiment. If the cause indicates_
    positive impact
on stock prices or economy, classify it as "positive". If it suggests decline or_
    uncertainty,
classify it as "negative". '''
    examples = "\n\n".join(few_shots)
    return f"""
Here are {len(few_shots)} examples of providing the sentiment based on the given text.

{examples}.
```

(continues on next page)

(continued from previous page)

Analyze the financial news headline in a step-by-step manner.
 First, identify key financial terms (e.g., profit, loss, growth, decline).
 Second, extract phrases indicating sentiment (e.g., 'strong earnings,' 'market turmoil ↘').
 Finally, provide a reasoned conclusion and assess whether the sentiment is "positive", "negative", or "neutral" from the perspective of retail investors.
 If you cannot assess the sentiment, then classify it "neutral".
 Provide your sentiment label and reasoning in json format.
 Do not provide any other answer.
 Text: `{text}""".strip()`

```
# CoT prompt to induce intermediate reasoning steps
pred2 = []
for i, (text, sentiment) in tqdm(enumerate(zip(X_test, y_test)), total=len(y_test)):
    s = generate_prompt(text)
    output = ollama.generate(model=model_name, prompt=s, options={"temperature":0})
    if i < 5:
        print(f"Labeled {sentiment=}. RESPONSE={}")
        pprint(f"{output.response}")
    pred2.append(parse_json(output.response) ['sentiment'].lower())
```

100%|██████████| 4831/4831 [25:46<00:00, 3.12it/s]

```
# Evaluate prediction
print(f"Accuracy: {accuracy_score(y_true=y_test[:len(pred2)], y_pred=pred2):.3f}")
print(classification_report(y_true=y_test[:len(pred2)], y_pred=pred2))
c = confusion_matrix(y_true=y_test[:len(pred2)], y_pred=pred2, labels=targets)
DataFrame(columns=pd.MultiIndex.from_product([['Predicted'], targets]),
          index=targets, data=c) # display confusion matrix
```

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| negative | 0.11 | 0.11 | 0.11 | 73 |
| neutral | 0.55 | 0.60 | 0.57 | 355 |
| positive | 0.19 | 0.15 | 0.17 | 168 |
| accuracy | | | 0.41 | 596 |
| macro avg | 0.28 | 0.29 | 0.28 | 596 |
| weighted avg | 0.39 | 0.41 | 0.40 | 596 |

| Predicted | | | | |
|-----------|----------|---------|----------|--|
| | negative | neutral | positive | |
| negative | 8 | 52 | 13 | |
| neutral | 48 | 213 | 94 | |
| positive | 19 | 124 | 25 | |

35.4 Vision language

Analyze a chart image, specifically Figure 4 from Malo et al (2014)

```
# Load and display a test image
from PIL import Image
import matplotlib.pyplot as plt
image_filename = "assets/MSDTRPL_EC013-H-2020-1602198302.webp"
image_filename = "assets/malo-fig4.png"
image = Image.open(image_filename)
plt.imshow(image)
plt.axis('off')
```

```
(np.float64(-0.5), np.float64(497.5), np.float64(486.5), np.float64(-0.5))
```

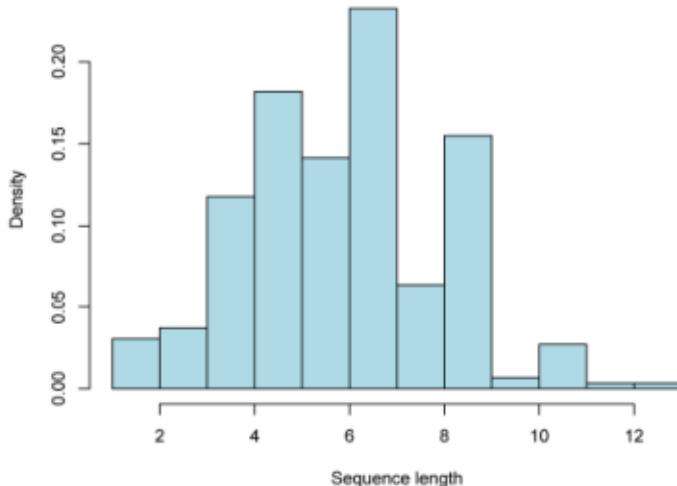


Figure 4. : The figure shows the distribution of entity-sequence lengths (before pruning) in the financial phrasebank, which consists of 5000 sentences. An entity-sequence is a linear representation of the phrase structure produced by the entity-extractor.

```
response = ollama.chat(
    model=model_name,
    messages=[
        {
            'role': 'user',
            'content': 'Describe this image:',
            'images': [image_filename]
        }
    ]
)
```

```
print("\n".join(textwrap.fill(s) for s in response['message']['content'].split('\n')))
```

Here's a description of the image you sent:

****Type of Chart:****

The image displays a histogram.

****Data Represented:****

The histogram shows the distribution of "sequence lengths" (likely referring to the length of entity sequences) from a financial phrase bank consisting of 5000 sentences. The sequence lengths are represented before pruning.

****Axes:****

- * ****X-axis (Horizontal):**** Represents "Sequence Length," ranging from 2 to 12, with integer values.
- * ****Y-axis (Vertical):**** Represents "Density," ranging from 0 to 0.20.

****Key Observations:****

- * The distribution is somewhat skewed, with a peak around a sequence length of 6 or 7.
- * There's a noticeable number of sequences with lengths between 6 and 10.
- * The density decreases as the sequence length moves away from the peak.

****Overall Impression:****

The histogram provides a visual summary of how long the entity sequences are, before any pruning or filtering is applied. It suggests that most sequences are relatively short (between 2 and 10 units), but there's also a presence of longer sequences.

35.5 Multi-lingual

Translate to Mandarin and french

```
mandarin = ollama.chat(
    model=model_name,
    messages=[
        {
            'role': 'user',
            'content': 'Please translate to mandarin: ' + response[
                'message'] ['content'],
        }
    ]
)
```

```
print("\n".join(textwrap.fill(s) for s in mandarin['message'] ['content'].split('\n')))
```

Okay, here's a Mandarin translation, aiming for accuracy and clarity. I've included explanations after each section to clarify choices and offer alternatives. I've also broken it down into sections mirroring the original. **Please read the notes at the very end - they are crucial for understanding the nuances of the translation.**

1. Type of Chart:

- * **Mandarin:** 图片“展示了” (Zhè zhāng túpiàn xiǎnshì) (Zhè zhāng túpiàn xiǎnshì de shì yī gè zhífāngtú.)
- * **Explanation:**
 - * `图片 (zhāng túpiàn)` : "This image" - standard way to refer to an image.
 - * `展示了 (xiǎnshì)` : "Displays/Shows" - a common and neutral verb.
 - * `直方图 (zhífāngtú)` : "Histogram" - the standard term.

2. Data Represented:

- * **Mandarin:** 5000 份“展示了” 5000 份 (Zhège zhífāngtú zhǎnshì le láizì yī gè bāohán 5000 gè jùzi de jīnróng duānyǔ kù zhōng de “xùliè chángdù” de fēnbù qíngkuàng. Xùliè chángdù shì zài xiūjiǎn zhīqián de shùjù.)
- * **Explanation:**
 - * `展示了 (zhǎnshì)` : "Shows/Displays" - used again for consistency.
 - * `份 (láizì)` : "From" - indicates the source of the data.
 - * `份 (bāohán)` : "Containing/Consisting of"
 - * `份“展示了” (jīnróng duānyǔ kù)` : "Financial phrase bank" - a direct translation.
 - * `份 (jùzi)` : "Sentences"
 - * `“展示了” (“xùliè chángdù”)` : "Sequence Length" - The quotation marks are important to indicate it's a specific term. `展示了 (xùliè)` means "sequence" and `份 (chángdù)` means "length."
 - * `份“展示了” (fēnbù qíngkuàng)` : "Distribution situation/pattern" - a common way to describe a distribution.
 - * `份 (xiūjiǎn)` : "Pruning" - a good translation for the technical term.
 - * `份 (zhīqián)` : "Before"

3. Axes:

- * **Mandarin:**
 - * **X轴 (轴):** 轴“展示了” 2 到 12 (X zhóu (shuǐpíng) : Dàibiǎo “xùliè chángdù”, fànwéi cóng 2 dào 12, zhíwèi zhěngshù.)
 - * **Y轴 (轴):** 轴“展示了” 0 到 0.20 (Y zhóu (chōngcù) : Dàibiǎo “midù”, fànwéi cóng 0 dào 0.20.)
- * **Explanation:**
 - * `X轴 (shuǐpíng)` : "X-axis (Horizontal)" - `轴 (zhóu)` means "axis," and `展示了 (shuǐpíng)` means "horizontal."
 - * `Y轴 (chōngcù)` : "Y-axis (Vertical)" - `轴 (chōngcù)` means "vertical."
 - * `轴 (dàibiǎo)` : "Represents"
 - * `轴 (fànwéi)` : "Range"
 - * `展示了 (zhíwèi zhěngshù)` : "Values are integers"

4. Key Observations:

(continues on next page)

(continued from previous page)

* **Mandarin:**

* 6 7 (Fēnbù qíngkuàng yǒuxiē qīngxiá, fēngzhí chūxiàn zài xùliè chángdù suōyú 6 huò 7 zuōyōu.)

* 6 10 (Kěyǐ kànjiàn, zài 6 dào 10 zhījiān de xùliè chángdù shùliàng bǐjiào duō.)

* (Suízhe xùliè chángdù yuǎnlí fēngzhí, mìdù zhújiàn xiàjiàng.)

* **Explanation:**

* ` (yǒuxiē qīngxiá)` : "Somewhat skewed" - a good translation for "skewed."

* ` (fēngzhí)` : "Peak"

* ` (chūxiàn zài)` : "Appears at"

* ` (suōyú)` : "Approximately"

* ` (zuōyōu)` : "Around/Approximately" - softens the number.

* ` (kěyǐ kànjiàn)` : "It can be seen that"

* ` (shùliàng bǐjiào duō)` : "The number is relatively large"

* ` (suízhe)` : "As/With"

* ` (yuǎnlí)` : "Away from"

* ` (zhújiàn xiàjiàng)` : "Gradually decreases"

5. Overall Impression:

* **Mandarin:** 10 (Zhège zhífāngtú tígōng le guānyú shítǐ xùliè chángdù de shijué zōngjié, zài yìngyòng rènhé xiūjiǎn huò guòlǜ zhíqíán. Tā biāomíng dàdū zhī xùliè xiāngduì jiào duǎn (zài 2 dào 10 gè dānwèi zhījīān), yě yǒng yǒu jiào cháng de xùliè.)

* **Explanation:**

* ` (tígōng le)` : "Provides"

* ` (shijué zōngjié)` : "Visual summary"

* ` (yìngyòng)` : "Apply"

* ` (guòlǜ)` : "Filtering"

* ` (xiāngduì jiào duǎn)` : "Relatively short"

* ` (dānwèi)` : "Units"

* ` (yě yǒng yǒu)` : "But there also exists"

IMPORTANT NOTES:

* **Technical Jargon:** I'm assuming a reasonably technical audience. If the audience is less familiar with data analysis terms, some phrases might need simplification.

* **"Entity Sequences":** The translation of "entity sequences" is a bit tricky. ` (shítǐ xùliè)` is a literal translation, but depending on the context, a more descriptive phrase might be better.

* **Naturalness:** While I'm aiming for accuracy, a native speaker might phrase some sentences slightly differently for a more natural flow.

* **Context is Key:** The best translation always depends on the specific context and the intended audience. If you can provide more context, I can refine the translation further.

* **Simplified vs. Traditional Chinese:** This translation is in Simplified Chinese, which is most commonly used in mainland China. If you need Traditional Chinese, let me know.

```

french = ollama.chat(
    model=model_name,
    messages=[
        {
            'role': 'user',
            'content': 'Please translate to french: ' + response['message'
        } [ 'content' ],
        }
    ]
)

print("\n".join(textwrap.fill(s) for s in french['message'][ 'content' ].split('\n')))
```

Okay, here's a French translation of the description, aiming for accuracy and clarity. I've included a couple of options for certain phrases to give you some flexibility. I've also added notes after the translation to explain some choices.

Option 1 (More Formal) :

"Voici une description de l'image que vous avez envoyée :

Type de graphique :

L'image présente un histogramme.

Données représentées :

L'histogramme illustre la distribution des "longueurs de séquences" (probablement faisant référence à la longueur des séquences d'entités) provenant d'une banque de phrases financières composée de 5000 phrases. Les longueurs de séquences sont représentées avant élagage.

Axes :

- * **Axe des abscisses (horizontal) :** Représente la "Longueur de séquence", variant de 2 à 12, avec des valeurs entières.
- * **Axe des ordonnées (vertical) :** Représente la "Densité", variant de 0 à 0,20.

Observations principales :

- * La distribution est légèrement asymétrique, avec un pic autour d'une longueur de séquence de 6 ou 7.
- * On observe un nombre notable de séquences dont la longueur se situe entre 6 et 10.
- * La densité diminue à mesure que la longueur de la séquence s'éloigne du pic.

Impression générale :

L'histogramme fournit un résumé visuel de la longueur des séquences d'entités, avant toute élagage ou filtrage. Il suggère que la plupart des séquences sont relativement courtes (entre 2 et 10 unités), mais qu'il existe également des séquences plus longues."

(continues on next page)

(continued from previous page)

Option 2 (Slightly Less Formal):

"Voici une description de l'image que vous avez envoyée :

Type de graphique :

L'image montre un histogramme.

Données représentées :

L'histogramme montre la distribution des "longueurs de séquences" (probablement la longueur des séquences d'entités) provenant d'une banque de phrases financières contenant 5000 phrases. Les longueurs de séquences sont affichées avant l'élagage.

Axes :

- * **Axe horizontal (abscisses) :** Représente la "Longueur de séquence", allant de 2 à 12, avec des valeurs entières.
- * **Axe vertical (ordonnées) :** Représente la "Densité", allant de 0 à 0,20.

Observations principales :

- * La distribution est un peu asymétrique, avec un pic autour d'une longueur de séquence de 6 ou 7.
- * Il y a un nombre important de séquences dont la longueur se situe entre 6 et 10.
- * La densité diminue à mesure que la longueur de la séquence s'éloigne du pic.

Impression générale :

L'histogramme donne un aperçu visuel de la longueur des séquences d'entités, avant tout élagage ou filtrage. Il indique que la plupart des séquences sont relativement courtes (entre 2 et 10 unités), mais qu'il y a aussi des séquences plus longues."

Notes on Choices & Vocabulary:

- * ***"Histogramme"***: This is the standard French translation of "histogram."
- * ***"Longueurs de séquences"***: This is a direct translation of "sequence lengths."
- * ***"Banque de phrases financières"***: "Financial phrase bank" translates well to this.
- * ***"Élagage"***: This is the best translation for "pruning" in this context. It implies removing or shortening sequences.
- * ***"Asymétrique"***: This is the standard term for "skewed" in a statistical context.
- * ***"Axe des abscisses/ordonnées" vs. "Axe horizontal/vertical"***: The first is more technically correct, the second is more conversational. I've included both options.
- * ***"Fournit un résumé visuel" vs. "Donne un aperçu visuel"***: Both are good translations of "provides a visual summary."
- * ***"Indique" vs. "Suggère"***: Both are good translations of

(continues on next page)

(continued from previous page)

"suggests." "Indique" is a bit stronger.
* **"Contenant" vs. "composée de"**: Both are good translations of "consisting of". "Contenant" is slightly less formal.

Which option to choose?

* **Option 1** is more suitable for a formal report or technical document.
* **Option 2** is more appropriate for a more casual explanation.

I hope this helps! Let me know if you have any other questions.

References:

- Malo, P., Sinha, A., Takala, P., Korhonen, P. and Wallenius, J. (2014): "Good debt or bad debt: Detecting semantic orientations in economic texts." *Journal of the American Society for Information Science and Technology*.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le and Denny Zhou, 2023, Chain-of-Thought Prompting Elicits Reasoning in Large Language Models, <https://arxiv.org/abs/2201.11903>
- Greg Durrett, 2023, "CS388 Natural Language Processing course materials", retrieved from <https://www.cs.utexas.edu/~gdurrett/courses/online-course/materials.html>
- <https://llama.meta.com/docs/how-to-guides/prompting/>
- <https://www.promptingguide.ai/techniques/cot>
- <https://medium.com/age-of-awareness/chain-of-thought-in-ai-7f45c3d2c12a>

AGENTS

The true sign of intelligence is not knowledge but imagination - Albert Einstein

We introduce the capabilities of LLM agents using chatbots with memory, retrieval-augmented generation (RAG), and multi-agent collaboration. Employing Microsoft's Phi-4-mini model, a ChromaDB vector store and sentence embedding models via Ollama, we demonstrate how LLMs can be enhanced to improve factual accuracy and support long-context reasoning. The final application centers around measuring the value of corporate philanthropy, illustrating how agents can ground their reasoning in retrieved knowledge and engage in multi-role dialogue to develop actionable plans.

```
# By: Terence Lim, 2020-2025 (terence-lim.github.io)
import torch
from transformers import AutoModelForCausalLM, AutoProcessor, GenerationConfig,_
    _logging
import io
import soundfile
import matplotlib.pyplot as plt
from PIL import Image
from tqdm import tqdm
from pprint import pprint
import warnings
import textwrap
def wrap(text):
    """Helper to wrap text for pretty printing"""
    return "\n".join([textwrap.fill(s, width=80) for s in text.split('\n')])
torch.random.manual_seed(0)
logging.set_verbosity_error()
# display gpu and memory
gpu_stats = torch.cuda.get_device_properties(0)
max_memory = round(gpu_stats.total_memory / 1024 / 1024 / 1024, 3)
print(f"GPU = {gpu_stats.name}. Max memory = {max_memory} GB.")
```

GPU = NVIDIA GeForce RTX 3080 Laptop GPU. Max memory = 15.739 GB.

36.1 Microsoft Phi-4 model

The Microsoft Phi-4 family is designed for compactness and strong reasoning across modalities. Released in February 2024, Phi-4-multimodal is a 5.6B parameter model which processes audio, vision, and language simultaneously within the same representation space. Its capabilities include:

- Instruction following: Instruct-tuned for multi-turn conversation, summarization, Q&A
- Math and code reasoning: Fine-tuned on mathematical and coding data
- Multilingual: Supports multiple languages with cross-lingual reasoning
- Vision: integrates a visual encoder that converts image inputs into embeddings, supporting tasks such as
 - OCR (Optical Character Recognition)
 - chart and table understanding
 - image-based reasoning (e.g., figures, diagrams)
- Audio Capabilities: a specialized audio encoder for speech inputs, supporting:
 - Automatic Speech Recognition (ASR)
 - Multilingual speech translation

<https://azure.microsoft.com/en-us/blog/empowering-innovation-the-next-generation-of-the-phi-family/>

```
# load and configures Phi-4-mini-instruct using HuggingFace
model_path = "microsoft/Phi-4-mini-instruct"
model_path = "microsoft/Phi-4-multimodal-instruct"

with warnings.catch_warnings(): # ignore small sample warnings
    warnings.simplefilter("ignore")
    processor = AutoProcessor.from_pretrained(model_path, trust_remote_code=True)

    print(processor.tokenizer)

model = AutoModelForCausalLM.from_pretrained(
    model_path,
    trust_remote_code=True,
    torch_dtype='auto',
    _attn_implementation='flash_attention_2',
    # _attn_implementation='eager',      # 'flash_attention_2',
).cuda()
```

```
GPT2TokenizerFast(name_or_path='microsoft/Phi-4-multimodal-instruct', vocab_size=200019, model_max_length=131072, is_fast=True, padding_side='right', truncation_side='right', special_tokens={'bos_token': '<|endoftext|>', 'eos_token': '<|endoftext|>', 'unk_token': '<|endoftext|>', 'pad_token': '<|endoftext|>'}, clean_up_tokenization_spaces=False, added_tokens_decoder={199999: AddedToken("<|endoftext|>", rstrip=False, lstrip=False, single_word=False, normalized=False, special=True), 200010: AddedToken("<|endoftext10|>", rstrip=False, lstrip=False, single_word=False, normalized=False, special=True), 200011: AddedToken("<|endoftext11|>", rstrip=False, lstrip=False, single_word=False, normalized=False, special=True), 200018: AddedToken("<|endofprompt|>", rstrip=False, lstrip=False, single_word=False, normalized=False, special=True), 200019: AddedToken("<|assistant|>", rstrip=True, lstrip=False, single_
```

(continues on next page)

(continued from previous page)

```

word=False, normalized=False, special=True),
200020: AddedToken("<|end|>", rstrip=True, lstrip=False, single_word=False,
word=False, special=True),
200021: AddedToken("<|user|>", rstrip=True, lstrip=False, single_
word=False, normalized=False, special=True),
200022: AddedToken("<|system|>", rstrip=True, lstrip=False, single_
word=False, normalized=False, special=True),
200023: AddedToken("<|tool|>", rstrip=True, lstrip=False, single_
word=False, normalized=False, special=False),
200024: AddedToken("<|/tool|>", rstrip=True, lstrip=False, single_
word=False, normalized=False, special=False),
200025: AddedToken("<|tool_call|>", rstrip=True, lstrip=False, single_
word=False, normalized=False, special=False),
200026: AddedToken("<|/tool_call|>", rstrip=True, lstrip=False, single_
word=False, normalized=False, special=False),
200027: AddedToken("<|tool_response|>", rstrip=True, lstrip=False, single_
word=False, normalized=False, special=False),
200028: AddedToken("<|tag|>", rstrip=True, lstrip=False, single_word=False,
normalized=False, special=True),
}
)

```

Loading checkpoint shards: 0% | 0/3 [00:00<?, ?it/s]

Memory usage

```

def cuda_memory():
    """Show GPU memory free"""
    if torch.cuda.is_available():
        device = torch.device('cuda')
        total_memory = torch.cuda.get_device_properties(device).total_memory
        reserved_memory = torch.cuda.memory_reserved(device)
        allocated_memory = torch.cuda.memory_allocated(device)
        free_memory = total_memory - reserved_memory
        print(f"Total memory: {total_memory / (1024**3):.2f} GB")
        print(f"Reserved memory: {reserved_memory / (1024**3):.2f} GB")
        print(f"Allocated memory: {allocated_memory / (1024**3):.2f} GB")
        print(f"Free memory: {free_memory / (1024**3):.2f} GB")

cuda_memory()

```

Total memory: 15.74 GB
 Reserved memory: 10.45 GB
 Allocated memory: 10.42 GB
 Free memory: 5.29 GB

Generate model response

```

generation_args = {
    "max_new_tokens": 512,
    "#return_full_text": False,
    "temperature": 0.6,    # nonzero so that chat responses can vary slightly
    "do_sample": True,    # False,
}

```

(continues on next page)

(continued from previous page)

```
generation_config = GenerationConfig.from_pretrained(model_path, 'generation_config.  
↳ json')
```

Helpers to format content to prompt for response

```
def create_content(query, image=None, audio=None):  
    """Formats a message content string"""  
    image_prompt = '' if image is None else '<|image_1|>'  
    audio_prompt = '' if audio is None else '<|audio_1|>'  
    prompt = f'{image_prompt}{audio_prompt}{query}'  
    return prompt  
  
def create_prompt(query, image=None, audio=None):  
    """Generate a single prompt for inference response"""  
    user_prompt = '<|user|>'  
    assistant_prompt = '<|assistant|>'  
    prompt_suffix = '<|end|>'  
    content = create_content(query, image=image, audio=audio)  
    prompt = f'{user_prompt}{content}{prompt_suffix}{assistant_prompt}'  
    return prompt  
  
def pipe(query, image=None, audio=None, verbose=False, **kwargs):  
    """Pipeline to format and send query, and generate and decode response"""  
    if isinstance(query, list): # query input is given as a list of messages  
        prompt = processor.tokenizer.apply_chat_template(query,  
                                                       tokenize=False,  
                                                       add_generation_prompt=True)  
        # remove last </endoftext> if it is there, which is used for training, not  
        ↳ inference.  
        # For training, make sure to add </endoftext> in the end.  
        if prompt.endswith('<|endoftext|>'):  
            prompt = prompt.rstrip('<|endoftext|>')  
    elif isinstance(query, str): # query input is given as a string  
        prompt = create_prompt(query, image=image, audio=audio)  
    else:  
        raise Exception('Invalid prompt format, must be str or list of messages')  
    if verbose:  
        print(prompt)  
    inputs = processor(prompt, images=image, audios=audio, return_tensors='pt').to(  
        ↳ 'cuda:0')  
  
    with warnings.catch_warnings(): # ignore small sample warnings  
        ↳  
            warnings.simplefilter("ignore")  
  
        generate_ids = model.generate(  
            **inputs,  
            generation_config=generation_config,  
            # https://huggingface.co/microsoft/Phi-4-multimodal-instruct/discussions/  
            ↳ 46  
            num_logits_to_keep=1,  
            **kwargs,  
        )  
        generate_ids = generate_ids[:, inputs['input_ids'].shape[1] :]  
        response = processor.batch_decode(  
            generate_ids, skip_special_tokens=True, clean_up_tokenization_spaces=False)[0]
```

(continues on next page)

(continued from previous page)

```
return response
```

36.2 Chatbot

A chatbot simulates ongoing dialogue between a user and an LLM. By appending previous user inputs and model responses into the prompt, the chatbot maintains conversational memory. This enables it to handle follow-up questions, build on earlier answers, and maintain topic continuity over multiple turns. During a chat session, the prompt is iteratively expanded to include each previous exchange. This accumulated memory allows the model to generate responses that take prior context into account.

Simulate user queries

```
# Simulate a sequence of user queries to be processed
sequence_user_queries = [
    "What are the challenges to measuring the value of corporate philanthropy?",
    "Please suggest some solutions.",
    "Which of these are the most promising?",
    "Please concisely summarize into an action plan"
]
```

36.2.1 Memory

During a chat session, the prompt is iteratively expanded to include each previous exchange. This accumulated memory allows the model to generate responses that take prior context into account. The final response is displayed, along with the complete prompt history used in that turn—demonstrating how memory accumulation influences generation.

```
# Prompt grows over time with user and assistant turns.
memory = [] # memory of past AI-assistant turns
for turns in tqdm(range(len(sequence_user_queries))):

    # Add initial user query
    query = sequence_user_queries[0]
    messages = [{"role": "user", "content": query}]

    # 3. Loop through past turns of AI response and user prompts
    for response, query in zip(memory, sequence_user_queries[1:len(memory)+1]):

        # Add AI assistant's response
        messages.append({"role": "assistant", "content": response})

        # Add user's next query
        messages.append({"role": "user", "content": query})

    # 4. Generate response and save to memory
    output = pipe(messages, **generation_args)
    memory.append(output)
    print(f'----- {turns=} -----')
    print('QUERY:', query)
    print('RESPONSE:')
    print(wrap(memory[-1]))
```

25% | [] 1/4 [00:19<00:57, 19.22s/it]

----- turns=0 -----

QUERY: What are the challenges to measuring the value of corporate philanthropy?

RESPONSE:

Measuring the value of corporate philanthropy is challenging for several reasons:

1. **Subjectivity:** The value of corporate philanthropy is often subjective and difficult to quantify. Different stakeholders may have different opinions on the worthiness of a cause or the effectiveness of a charitable contribution. What one person views as valuable, another may not.

2. **Lack of standardized metrics:** There is no universally accepted standard for measuring the impact of corporate philanthropy. While some organizations may use metrics such as the percentage of profits donated or the number of people reached, these metrics may not accurately reflect the true value of the philanthropic efforts.

3. **Long-term impact:** Many corporate philanthropic initiatives have long-term goals, making it difficult to measure their impact in the short term. It can take years for the full impact of a philanthropic effort to become apparent, making it challenging to assess its value in the present.

4. **Conflicting objectives:** Corporate philanthropy often involves balancing the interests of various stakeholders, including shareholders, employees, customers, and the communities in which the company operates. These conflicting objectives can make it difficult to determine the true value of a philanthropic effort.

5. **Lack of transparency:** Many companies are not transparent about their philanthropic efforts, making it difficult to assess their impact. Companies may not disclose how much money they are donating, where the money is going, or how the money is being used.

6. **Complexity of measuring impact:** Measuring the impact of corporate philanthropy can be complex, as it involves assessing the social, economic, and environmental outcomes of a philanthropic effort. These outcomes can be difficult to measure and may not be directly linked to the philanthropic effort.

Overall, measuring the value of corporate philanthropy is a complex task that requires careful consideration of various factors and the use of multiple metrics. It is important for companies to be transparent about their philanthropic efforts and to use standardized metrics to measure their impact.

50% | [] 2/4 [00:39<00:39, 19.79s/it]

----- turns=1 -----

QUERY: Please suggest some solutions.

RESPONSE:

To tackle the challenges of measuring the value of corporate philanthropy, the following solutions can be implemented:

1. **Develop standardized metrics:** Establishing standardized metrics for measuring the impact of corporate philanthropy can help to create a common framework for evaluation. These metrics could include, for example, the number of people

(continues on next page)

(continued from previous page)

reached, the amount of money donated, and the percentage of profits donated.

2. Use a multi-dimensional approach: Measuring the impact of corporate philanthropy should be a multi-dimensional approach that takes into account social, economic, and environmental outcomes. This can be achieved by using a combination of quantitative and qualitative data to assess the impact of a philanthropic effort.

3. Increase transparency: Companies can increase transparency by disclosing their philanthropic efforts, including the amount of money donated, the causes supported, and the outcomes achieved. This can help stakeholders to better understand the value of a company's philanthropic efforts and to build trust.

4. Engage stakeholders: Engaging stakeholders, such as employees, customers, and community members, in the evaluation of corporate philanthropy can help to ensure that the efforts are aligned with their interests and values. This can be achieved through mechanisms such as surveys, focus groups, and community forums.

5. Use third-party evaluations: Third-party evaluations can provide an independent assessment of the impact of corporate philanthropy. These evaluations can help to ensure that the metrics used are valid and reliable and that the impact of the philanthropic effort is accurately measured.

6. Set clear goals and objectives: Setting clear goals and objectives for corporate philanthropy can help to ensure that the efforts are aligned with the company's overall mission and values. This can help to ensure that the impact of the philanthropic effort is meaningful and sustainable.

7. Monitor and evaluate progress: Regularly monitoring and evaluating the progress of corporate philanthropy can help to ensure that the efforts are having the desired impact. This can include tracking key performance indicators, conducting impact assessments, and adjusting the efforts as needed.

By implementing these solutions, companies can improve their ability to measure the value of corporate philanthropy and demonstrate the impact of their philanthropic efforts to stakeholders.

75% | [REDACTED] | 3/4 [00:56<00:18, 18.43s/it]

----- turns=2 -----

QUERY: Which of these are the most promising?

RESPONSE:

While all of the solutions mentioned are important in improving the measurement of corporate philanthropy, the following are among the most promising:

1. Develop standardized metrics: Establishing standardized metrics for measuring the impact of corporate philanthropy can help to create a common framework for evaluation. This can help companies better understand the value of their philanthropic efforts and communicate their impact to stakeholders.

2. Use a multi-dimensional approach: A multi-dimensional approach that takes into account social, economic, and environmental outcomes can provide a more comprehensive assessment of the impact of corporate philanthropy. This can help companies better understand the full impact of their efforts and demonstrate their commitment to creating positive change.

(continues on next page)

(continued from previous page)

3. Increase transparency: Disclosing philanthropic efforts, including the amount of money donated, the causes supported, and the outcomes achieved, can help to build trust among stakeholders. This can also help companies to better understand the value of their philanthropic efforts and demonstrate their commitment to creating positive change.

4. Engage stakeholders: Engaging stakeholders, such as employees, customers, and community members, in the evaluation of corporate philanthropy can help to ensure that the efforts are aligned with their interests and values. This can help companies to better understand the impact of their philanthropic efforts and demonstrate their commitment to creating positive change.

5. Use third-party evaluations: Third-party evaluations can provide an independent assessment of the impact of corporate philanthropy. This can help to ensure that the metrics used are valid and reliable and that the impact of the philanthropic effort is accurately measured.

These solutions can help companies to more accurately measure the value of their corporate philanthropy and demonstrate their commitment to creating positive change. By implementing these solutions, companies can better communicate their impact to stakeholders and build trust among stakeholders.

100% |██████████| 4/4 [01:07<00:00, 16.98s/it]

----- turns=3 -----

QUERY: Please concisely summarize into an action plan

RESPONSE:

1. Develop standardized metrics: Create a framework for evaluating corporate philanthropy, including metrics such as the number of people reached, the amount of money donated, and the percentage of profits donated.

2. Use a multi-dimensional approach: Assess the social, economic, and environmental outcomes of corporate philanthropy using both quantitative and qualitative data.

3. Increase transparency: Disclose philanthropic efforts, including the amount of money donated, the causes supported, and the outcomes achieved.

4. Engage stakeholders: Involve employees, customers, and community members in the evaluation of corporate philanthropy through surveys, focus groups, and community forums.

5. Use third-party evaluations: Hire independent evaluators to assess the impact of corporate philanthropy, ensuring that the metrics used are valid and reliable.

6. Set clear goals and objectives: Define clear goals and objectives for corporate philanthropy that align with the company's overall mission and values.

7. Monitor and evaluate progress: Track key performance indicators, conduct impact assessments, and adjust philanthropic efforts as needed to ensure they are having the desired impact.

By implementing this action plan, companies can improve their ability to measure

(continues on next page)

(continued from previous page)

the value of corporate philanthropy and demonstrate their commitment to creating positive change.

Show the final response from the last turn of the Chatbot

```
print('FINAL ANSWER: ')
print(wrap(output))
```

FINAL ANSWER:

1. Develop standardized metrics: Create a framework for evaluating corporate philanthropy, including metrics such as the number of people reached, the amount of money donated, and the percentage of profits donated.
2. Use a multi-dimensional approach: Assess the social, economic, and environmental outcomes of corporate philanthropy using both quantitative and qualitative data.
3. Increase transparency: Disclose philanthropic efforts, including the amount of money donated, the causes supported, and the outcomes achieved.
4. Engage stakeholders: Involve employees, customers, and community members in the evaluation of corporate philanthropy through surveys, focus groups, and community forums.
5. Use third-party evaluations: Hire independent evaluators to assess the impact of corporate philanthropy, ensuring that the metrics used are valid and reliable.
6. Set clear goals and objectives: Define clear goals and objectives for corporate philanthropy that align with the company's overall mission and values.
7. Monitor and evaluate progress: Track key performance indicators, conduct impact assessments, and adjust philanthropic efforts as needed to ensure they are having the desired impact.

By implementing this action plan, companies can improve their ability to measure the value of corporate philanthropy and demonstrate their commitment to creating positive change.

36.3 Retrieval-augmented generation (RAG)

One major challenge in standard LLMs is **hallucination**, where the model produces text that sounds believable but is actually false or unsupported by facts. Because LLMs are trained to model word sequences rather than factual accuracy, they can sometimes generate convincing yet misleading statements—especially when no clear answer exists in their training data. For example, Hicks et al. (2024) argue that LLMs are “bullshit machines” that prioritize fluent language over truth. Similarly, Mahowald et al. (2023) caution against assuming that skill with language equates to actual reasoning or understanding.

RAG addresses this issue by anchoring the model’s responses in retrieved documents, making the output more factually reliable. In a common implementation known as prompt-based in-context retrieval, the LLM is prompted with the relevant

document and instructed to answer solely based on that information, similar to reading a reference article before answering a question.

RAG enhances LLM performance by dynamically retrieving and integrating external knowledge, allowing for better factual accuracy and handling of long-context tasks. It is increasingly used in several practical domains:

- Search-Augmented LLMs improve accuracy in fact-checking and open-domain Q&A tasks.
- Enterprise Applications enable better handling of legal, financial, and medical documents.
- Personalized AI Assistants can use retrieval to support domain-specific queries using private or proprietary knowledge bases (e.g., for customer service).

Retrieval-Augmented Generation (RAG) is a two-step process: first, the system retrieves information from an external knowledge base, and then it generates a response based on that retrieved content.

The LangChain framework offers flexible tools for loading and preprocessing input documents of many formats. A textbook is read and divided into chunks of roughly 1,000 characters with some overlap. These chunks provide manageable input sizes for retrieval while maintaining continuity. The text itself outlines how companies can measure philanthropic value through social, business, and investor-oriented metrics, and forms the knowledge base for grounding agent responses.

```
# LangChain to loads and split documents into chunks for retrieval
from langchain_community.document_loaders import TextLoader,_
    UnstructuredMarkdownLoader
from langchain_text_splitters import RecursiveCharacterTextSplitter
```

```
# Loads markdown file (MVCP.md) and split into overlapping chunks
loader = TextLoader('assets/MVCP.md')
document = loader.load()
text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=200)
chunked_documents = text_splitter.split_documents(document)
```

```
print('Number of chunks:', len(chunked_documents))
print(wrap(str(chunked_documents[:2])))
```

```
Number of chunks: 196
[Document(metadata={'source': 'MVCP.md'}, page_content='MEASURING THE VALUE OF
CORPORATE PHILANTHROPY: SOCIAL IMPACT, BUSINESS BENEFITS, AND INVESTOR
RETURNS\nby\nTerence Lim, Ph.D.\nReport Author and Manager, Standards and
Measurement,\nCommittee Encouraging Corporate Philanthropy\nthrough the
2008-2009 Goldman Sachs Public Service Program)\n\nHow to measure the value and
results of corporate philanthropy remains\nnone of corporate giving
professionals' greatest challenges. Social and\nbusiness benefits are often
long-term or intangible, which make\nsystematic measurement complex. And yet:
Corporate philanthropy faces\nincreasing pressures to show it is as
strategic, cost-effective, and value-enhancing\nas possible. The industry faces a
critical need to assess current practices and\nmeasurement trends, clarify the
demands practitioners face for impact evidence,\nand identify the most promising
steps forward in order to make progress on these\nchallenges.'),
Document(metadata={'source': 'MVCP.md'}, page_content='This report aims to meet
that need, by providing the corporate\nphilanthropic community with a review of
recent measurement studies, models,\nand evidence drawn from complementary
business disciplines as well as the social\nsector. Rather than present an other
compendium of narrative accounts and case\nstudies, we endeavor to generalize
the most valuable concepts and to recognize\nthe strengths and limitations of
various measurement approaches. In conjunction\nwith the annotated references
that follow, the analysis herein should provide an\nexcellent starting point for
```

(continues on next page)

(continued from previous page)

companies wishing to adapt current methodologies in the field to their own corporate giving programs.\nTo realize meaningful benefits, philanthropy cannot be treated as just another "check in the box," but rather must be executed no less professionally,\nproactively, and strategically than other core business activities. Our hope is\nthat this work will enlighten giving professionals, CEOs, and the investor')]

36.3.1 ChromaDB vector database

ChromaDB is an open-source vector store used to store and retrieve text embeddings. It enables fast and efficient semantic search, allowing the system to find the most relevant document chunks based on a given query.

For compatibility, ChromaDB requires SQLite version 3.35 or higher. If the system uses an older version, apply this workaround patch: Install pysqlite3-binary, then enter the following three lines in sqlite_version.py to swap the packages.

```
__import__('pysqlite3')
import sys
sys.modules['sqlite3'] = sys.modules.pop('pysqlite3')
```

```
# ChromaDB vector store to store and retrieve document embeddings.
import sqlite_version # hack to use lower version of SQLite
import chromadb
client = chromadb.Client()
try:
    client.delete_collection(name="docs")
except:
    pass
#collection = client.get_or_create_collection(name="docs")
collection = client.create_collection(name="docs")
```

The sentence embeddings for both documents and queries are generated using the mxbai-embed-large model served via **Ollama**. This embedding model, which reached state-of-the-art performance in April 2024, was trained on over 700 million high-quality sentence pairs and fine-tuned on 30 million triplets. These embeddings are stored in ChromaDB to support rapid document retrieval.

```
# Ollama for generating sentence embeddings using mxbai-embed-large model
# !ollama pull mxbai-embed-large
import ollama

# store chunk embeddings in a vector database
for i, d in enumerate(chunked_documents):
    response = ollama.embeddings(model="mxbai-embed-large", prompt=d.page_content)
    embedding = response["embedding"]
    collection.add(
        ids=[str(i)],
        embeddings=[embedding],
        documents=[d.page_content],
    )
```

36.3.2 Retrieval

The retrieval pipeline consists of four main steps:

- A user query is written in natural language.
- The query is embedded using the same model as the document chunks.
- The top 5 most semantically similar chunks are retrieved from the ChromaDB vector store.
- A new prompt is constructed by combining the user's question with the retrieved content for generation.

```
# A sample question prompt
query = "What are the challenges to measuring the value of corporate philanthropy?"
```

Embed the prompt query

```
# generate an embedding for the query
response = ollama.embeddings(
    prompt=query,
    model="mxbai-embed-large"
)
len(response['embedding']) # vector length
```

1024

Retrieve the 5 most similar document chunks from the vector database

```
# ses ChromaDB to find top 5 most similar chunks
results = collection.query(
    query_embeddings=[response["embedding"]],
    n_results=5,
)
data = ". ".join(results['documents'][0])
print(textwrap.fill(data))
```

MEASURING THE VALUE OF CORPORATE PHILANTHROPY: SOCIAL IMPACT, BUSINESS BENEFITS, AND INVESTOR RETURNS by Terence Lim, Ph.D. Report Author and Manager, Standards and Measurement, Committee Encouraging Corporate Philanthropy (through the 2008-2009 Goldman Sachs Public Service Program) How to measure the value and results of corporate philanthropy remains one of corporate giving professionals' greatest challenges. Social and business benefits are often long-term or intangible, which make systematic measurement complex. And yet: Corporate philanthropy faces increasing pressures to show it is as strategic, cost-effective, and value-enhancing as possible. The industry faces a critical need to assess current practices and measurement trends, clarify the demands practitioners face for impact evidence, and identify the most promising steps forward in order to make progress on these challenges.. detailed insights into the related measurement process which can help demonstrate understanding of what drives long term business success quality of management and superior potential to create financial value. The value of corporate philanthropy is measurable; as with many elements of business, however, it cannot always be measured as precisely as we would like. What gets measured gets managed goes the old adage; indeed measurement plays a crucial role in enabling companies to reach their full

(continues on next page)

(continued from previous page)

potential both philanthropically and as more successful and sustainable enterprises overall.. This report aims to meet that need, by providing the corporate philanthropic community with a review of recent measurement studies, models, and evidence drawn from complementary business disciplines as well as the social sector. Rather than present an other compendium of narrative accounts and case studies, we endeavor to generalize the most valuable concepts and to recognize the strengths and limitations of various measurement approaches. In conjunction with the annotated references that follow, the analysis herein should provide an excellent starting point for companies wishing to adapt current methodologies in the field to their own corporate giving programs. To realize meaningful benefits, philanthropy cannot be treated as just another "check in the box," but rather must be executed no less professionally, proactively, and strategically than other core business activities. Our hope is that this work will enlighten giving professionals, CEOs, and the investor. Corporate philanthropy is as vital as ever to business and society but it faces steep pressures to demonstrate that it is also cost effective and aligned with corporate needs. Indeed many corporate giving professionals cite measurement as their primary management challenge. Social and business benefits are often long-term, intangible or both and a systematic measurement of these results can be complex. Social change takes time. The missions and intervention strategies involved are diverse. For these reasons, the field of corporate philanthropy has been unable to determine a shared definition or method of measurement for social impact. Similarly, the financial value of enhancing intangibles such as a company's reputational and human capital cannot be measured directly and may not be converted into tangible bottom line profits in the near term. Corporate givers and grant recipients often use formal anecdotal methods to convey impact. While stories. **Summary** The attractiveness of these ROI methods for calculating corporate philanthropy's social returns is in bringing businesslike quantitative frameworks to evaluating and comparing the effectiveness of diverse social programs, and aggregating their social impact. However these sophisticated methodologies place heavy demands on data collection assumptions and value judgments underlying the analysis. Funders must assemble data and calculations on the program's monetary benefits and make subjective judgments on the relative value of different types of social changes. Corporate funders need to be knowledgeable and thoughtful about these limitations and typically should not rely solely on ROI when evaluating grants. Proponents of these methods note that the benefits of ROI analysis lie more in encouraging funders to lay bare the assumptions and trade-offs that may already be involved in their grant making decisions.

36.3.3 Generation

Construct the prompt combining the query and supporting context retrieved. Then send to the LLM, which generates a grounded response.

```
# assembles relevant chunks into a prompt alongside the original question
prompt = f"""
Only use relevant information in the following text delimited by triple quotes:

'''{data}'''"

Respond to this prompt: {query}."""
```

```
# send a conversation-style chat-template prompt incorporating the retrieved text
messages = [
    {"role": "user", "content": prompt},
]
output = pipe(messages, **generation_args)
print('QUERY:', query)
print('RESPONSE:')
print(wrap(output))
```

QUERY: What are the challenges to measuring the value of corporate philanthropy?
RESPONSE:

The challenges to measuring the value of corporate philanthropy include:

1. Social and business benefits are often long-term or intangible, making systematic measurement complex.
2. Corporate philanthropy faces increasing pressures to show it is as strategic, cost-effective, and value-enhancing as possible.
3. The industry faces a critical need to assess current practices and measurement trends.
4. There is a need to clarify the demands practitioners face for impact evidence.
5. Identifying the most promising steps forward to make progress on these challenges.
6. Demonstrating understanding of what drives long-term business success and quality of management.
7. Measuring the financial value of enhancing intangibles such as a company's reputational and human capital can be complex and may not be directly convertible into tangible bottom-line profits in the near term.
8. The measurement of social impact lacks a shared definition or method, and social change takes time.
9. Missions and intervention strategies in corporate philanthropy are diverse.
10. The use of sophisticated ROI methodologies for evaluating and comparing the effectiveness of diverse social programs places heavy demands on data collection assumptions and value judgments underlying the analysis.

```
# show the supporting source document chunk ids
print(set(results['ids'][0]))
```

```
{'0', '1', '84', '195', '4'}
```

36.4 Multi-agents

In multi-agent workflows, multiple LLM agents play different roles and collaborate to accomplish a shared objective. In this demonstration, three role-playing agents work together to simulate human-like decision-making processes:

- A Chief Executive Officer (CEO) asks strategic questions about measuring philanthropic value.
- A Chief Giving Officer (CGO) synthesizes responses into an evolving plan.
- An AI Assistant (nicknamed CorGi) uses RAG to answer the CEO's questions using only retrieved text.

The agents interact with each other over several turns, collaboratively building and refining a plan grounded in the corporate philanthropy document.

36.4.1 Tool calling

Traditional LLMs are limited in that they cannot retrieve live data or execute real-world functions unless explicitly programmed. Without tools, they may also make computational errors or respond inconsistently to ambiguous prompts. Recent advances focus on enabling tool use, where LLMs can:

- Make function calls.
- Use APIs to gather real-time data.
- Perform calculations or database lookups.

The main approaches include:

- Allow the model to generate function calls instead of plain text.
- Zero-Shot Tool Use: Models learn to use tools on-the-fly via structured prompts.
- Fine-Tuned Tool Use: Models are trained to recognize and call specific functions during generation.

AI Assistant agent:

The AI Assistant agent, called CorGi, responds to queries by leveraging RAG. Its prompt instructs it to only use retrieved documents to answer and to reply "I don't know" if the information isn't found in the text. This strict constraint helps minimize hallucination and ensures that all responses are grounded in the knowledge base.

```
# AI assistant agent answers using only retrieved text chunks
def rag_agent(query, n_results=5):
    """Role of a RAG-based question-answering agent"""
    response = ollama.embeddings(prompt=query, model="mxbai-embed-large")

    results = collection.query(query_embeddings=[response["embedding"]],
                               n_results=n_results)
    text = "\n".join(results['documents'][0])

    rag_prompt=f"""
    You are a helpful AI corporate giving research assistant.
    Use only the information in the following text to succinctly answer the question.
    Reply "I don't know" if the information is not found in the text.
    text: {text}.
    question: {query}"""

    # generate a response combining the prompt and data
    output = pipe(rag_prompt, **generation_args)
    return output, results['ids'][0]
```

Query about a concept present in the document

```
pprint(rag_agent("What is corporate philanthropy?"))
```

```
'Corporate philanthropy is when companies engage in charitable activities and '
'support social causes, which can enhance their reputation, address employee '
'concerns, and potentially contribute to business growth through innovation '
'and understanding market shifts.',
['102', '1', '87', '140', '142'])
```

Query about a concept not present in the document

```
pprint(rag_agent("What is corporate greed?"))
```

```
("I don't know.", ['137', '140', '136', '130', '117'])
```

36.4.2 Role playing

Chief Giving Officer agent:

This agent summarizes CorGi's outputs into a coherent strategic plan that can be reviewed by leadership.

```
# CGO (Chief Giving Officer) agent summarizes answers into a growing plan
def cgo_agent(text):
    """Role of an information-summarizing Chief Giving Officer"""
    cgo_prompt = f"""
You are the chief giving officer of a company.
You want to describe a plan for measuring the value
of your company's corporate philanthropy program.
Summarize the plan in bullet points by only using
the most relevant and distinct information in the following text:
{text}"""
    output = pipe(cgo_prompt, **generation_args)
    return output
```

Chief Executive Officer agent:

This agent reads the current version of the plan and generates new questions to improve or expand it.

```
# CEO (Chief Executive Officer) agent asks new questions to improve the corporate
→giving plan
def ceo_agent(text, verbose=False):
    """Role as an inquisitive Chief Executive Officer"""
    ceo_prompt = f"""
You are the chief executive officer of a company.
You want to know about the plan for measuring the value
of your company's corporate philanthropy program.
Please ask a simple and general question for an idea to improve the your company's
→plan
that is different than the text:
{text}"""
    output = pipe(ceo_prompt, verbose=verbose, **generation_args)
    return output
```

Over multiple turns, this role-playing setup allows the system to iteratively build a more comprehensive and data-backed plan.

```
# iterate over 10 turns to co-develop a full plan
memory = '* Measure the amount of monetary donations.'
docs = []
for turn in range(10):
    print(f"Turn {turn+1}...")
    question = ceo_agent(memory)
    print('CEO >>>', wrap(question))
    print('\n-----\n')
    answer, doc = cgo_agent(question)
    docs.append(doc)    # for diagnostics: track the RAG chunks retrieved
    print('Corgi-AI >>>', wrap(answer))
    print('\n-----\n')
    memory = cgo_agent("\n".join([memory, answer]))
    print('CGO >>>', wrap(memory))
    print('\n-----\n')
```

Turn 1...

CEO >>> How can we better evaluate the impact of our employees' volunteer efforts and community engagement through our corporate philanthropy program?

Corgi-AI >>> To better evaluate the impact of employees' volunteer efforts and community engagement through corporate philanthropy, companies can use internal surveys to assess whether the philanthropic program is meeting employee needs and creating a greater sense of identity between employee and employer. They can also analyze complementary disciplines such as human resources, marketing, risk management, and capital budgeting to improve measurement methods and identify long-term financial benefits. Additionally, companies can assess whether approaches used in the nonprofit sector can be applied to corporate giving programs. By understanding the strengths and limitations of various measurement approaches, companies can adapt current methodologies to their own corporate giving programs, and ensure that philanthropy is executed professionally, proactively, and strategically.

CGO >>> - Measure the amount of monetary donations.
 - Use internal surveys to assess employee needs and sense of identity.
 - Analyze complementary disciplines: human resources, marketing, risk management, and capital budgeting.
 - Identify long-term financial benefits through these disciplines.
 - Assess applicability of nonprofit sector approaches to corporate giving.
 - Adapt measurement methodologies to ensure professional, proactive, and strategic philanthropy.

Turn 2...

CEO >>> What are some innovative methods or metrics we could consider to better understand and enhance the impact of our company's corporate philanthropy program?

(continues on next page)

(continued from previous page)

Corgi-AI >>> I don't know

CGO >>> - Track and report monetary donations.

- Conduct internal surveys to gauge employee needs, sense of identity, and satisfaction.
 - Analyze the impact of philanthropy on human resources, marketing, risk management, and capital budgeting.
 - Identify and evaluate long-term financial benefits from philanthropy across these disciplines.
 - Compare corporate giving approaches with nonprofit sector methods.
 - Adapt measurement methodologies to ensure strategic and proactive philanthropy.
-

Turn 3...

CEO >>> What alternative frameworks or metrics can we explore to better capture and communicate the multifaceted impact of our corporate philanthropy program beyond traditional financial metrics and internal surveys?

Corgi-AI >>> The text does not provide specific alternative frameworks or metrics
↳ to explore
beyond traditional financial metrics and internal surveys.

CGO >>> - Track and report monetary donations.

- Conduct internal surveys to gauge employee needs, sense of identity, and satisfaction.
 - Analyze the impact of philanthropy on human resources, marketing, risk management, and capital budgeting.
 - Identify and evaluate long-term financial benefits from philanthropy across these disciplines.
 - Compare corporate giving approaches with nonprofit sector methods.
 - Adapt measurement methodologies to ensure strategic and proactive philanthropy.
-

Turn 4...

CEO >>> How can we develop a more comprehensive and innovative approach to measure
↳ the
long-term impact of our company's corporate philanthropy on community development and social change?

Corgi-AI >>> To develop a more comprehensive and innovative approach to measure
↳ the long-term
impact of a company's corporate philanthropy on community development and social change, it is suggested to analyze complementary disciplines such as human resources, marketing, risk management, and capital budgeting. This would improve

(continues on next page)

(continued from previous page)

the measurement methods and help identify long-term financial benefits. Additionally, it would be beneficial to review recent measurement studies, models, and evidence from business disciplines and the social sector, as well as anecdotal methods, to understand the strengths and limitations of various measurement approaches. Furthermore, interviews with senior corporate management and giving professionals can provide insights into common questions they face and help clarify what is needed in terms of impact evidence. Finally, it is important to adopt a systematic and data-based evidence approach to quantify the positive effects of corporate philanthropy and make a more persuasive case for why companies should engage in philanthropic causes.

CGO >>> - Track and report monetary donations.

- Conduct internal surveys to gauge employee needs, sense of identity, and satisfaction.
- Analyze the impact of philanthropy on human resources, marketing, risk management, and capital budgeting.
- Identify and evaluate long-term financial benefits from philanthropy across these disciplines.
- Compare corporate giving approaches with nonprofit sector methods.
- Adapt measurement methodologies to ensure strategic and proactive philanthropy.

To measure the long-term impact of corporate philanthropy, the following steps can be taken:

1. Analyze the impact of philanthropy on human resources, marketing, risk management, and capital budgeting.
 2. Identify and evaluate long-term financial benefits from philanthropy across these disciplines.
 3. Review recent measurement studies, models, and evidence from business disciplines and the social sector, as well as anecdotal methods.
 4. Conduct interviews with senior corporate management and giving professionals to understand common questions and what impact evidence is needed.
 5. Adopt a systematic and data-based evidence approach to quantify the positive effects of corporate philanthropy and make a more persuasive case for why companies should engage in philanthropic causes.
-

Turn 5...

CEO >>> What are some innovative methods or metrics we can use to better understand and report the long-term societal and community impact of our company's corporate philanthropy program?

Corgi-AI >>> I don't know

CGO >>> - Track and report monetary donations.

- Conduct internal surveys to gauge employee needs, sense of identity, and satisfaction.
- Analyze the impact of philanthropy on human resources, marketing, risk

(continues on next page)

(continued from previous page)

management, and capital budgeting.

- Identify and evaluate long-term financial benefits from philanthropy across these disciplines.

- Review recent measurement studies, models, and evidence from business disciplines and the social sector, as well as anecdotal methods.

- Conduct interviews with senior corporate management and giving professionals to understand common questions and what impact evidence is needed.

- Adopt a systematic and data-based evidence approach to quantify the positive effects of corporate philanthropy and make a more persuasive case for why companies should engage in philanthropic causes.

Turn 6...

CEO >>> How can we integrate community feedback and local partnerships into our corporate philanthropy program to enhance its value and effectiveness?

Corgi-AI >>> I don't know.

CGO >>> - Track and report monetary donations.

- Conduct internal surveys to gauge employee needs, sense of identity, and satisfaction.

- Analyze the impact of philanthropy on human resources, marketing, risk management, and capital budgeting.

- Identify and evaluate long-term financial benefits from philanthropy across these disciplines.

- Review recent measurement studies, models, and evidence from business disciplines and the social sector, as well as anecdotal methods.

- Conduct interviews with senior corporate management and giving professionals to understand common questions and what impact evidence is needed.

- Adopt a systematic and data-based evidence approach to quantify the positive effects of corporate philanthropy and make a more persuasive case for why companies should engage in philanthropic causes.

Turn 7...

CEO >>> How can we incorporate external third-party evaluations and community
 ↴ feedback

into our current plan for measuring the value of our corporate philanthropy program to gain a more comprehensive understanding of its impact?

Corgi-AI >>> I don't know

CGO >>> - Track and report monetary donations.

- Conduct internal surveys for employee insight.

- Analyze philanthropy's impact on HR, marketing, risk, and budgeting.

- Identify long-term financial benefits of philanthropy.

- Review measurement studies, models, and evidence.

- Interview senior management and professionals.

(continues on next page)

(continued from previous page)

- Adopt a systematic, data-based approach to quantify impacts and advocate for philanthropy.
-

Turn 8...

CEO >>> What are some innovative methods we could use to evaluate the social and cultural impact of our corporate philanthropy initiatives?

Corgi-AI >>> To evaluate the social and cultural impact of corporate philanthropy initiatives, innovative methods could include the application of financial valuation methods, such as ROI analysis, which bring businesslike quantitative frameworks to evaluating and comparing the effectiveness of diverse social programs and aggregating their social impact. Additionally, systematic measurement that brings rigor and discipline to the field, such as data-based evidence quantifying the positive effects of corporate philanthropy, can be used. These methods should be complemented with a review of recent measurement studies, models, and evidence drawn from complementary business disciplines as well as the social sector. However, it is important to note that these sophisticated methodologies place heavy demands on data collection assumptions and value judgments underlying the analysis. Proponents of these methods note that the benefits of ROI analysis lie more in encouraging funders to lay bare the assumptions and trade-offs that may already be involved in their grant making decisions. It is also important to recognize the strengths and limitations of various measurement approaches and to adapt current methodologies to fit corporate giving programs.

CGO >>> - Track and report monetary donations.

- Conduct internal surveys for employee insight.
 - Analyze philanthropy's impact on HR, marketing, risk, and budgeting.
 - Identify long-term financial benefits of philanthropy.
 - Review measurement studies, models, and evidence.
 - Interview senior management and professionals.
 - Adopt a systematic, data-based approach to quantify impacts and advocate for philanthropy.
 - Apply financial valuation methods, such as ROI analysis, to evaluate and compare the effectiveness of diverse social programs.
 - Use systematic measurement to bring rigor and discipline, quantifying the positive effects of corporate philanthropy.
 - Complement methodologies with a review of recent measurement studies, models, and evidence from complementary business disciplines and the social sector.
 - Recognize the strengths and limitations of various measurement approaches and adapt current methodologies to fit corporate giving programs.
-

Turn 9...

CEO >>> How can we develop a more inclusive and engaging method to involve our stakeholders in the evaluation and improvement of our corporate philanthropy program?

Corgi-AI >>> The report suggests bringing businesslike quantitative frameworks to u

(continues on next page)

(continued from previous page)

↳evaluate

and compare the effectiveness of diverse social programs, and aggregating their social impact. It also emphasizes the importance of making subjective judgments on the relative value of different types of social changes. However, it also cautions that these sophisticated methodologies place heavy demands on data collection assumptions and value judgments underlying the analysis. Therefore, stakeholders should be knowledgeable and thoughtful about these limitations.

To develop a more inclusive and engaging method to involve stakeholders in the evaluation and improvement of the corporate philanthropy program, one could consider the following steps:

1. Use businesslike quantitative frameworks to evaluate and compare the effectiveness of diverse social programs.
2. Aggregate the social impact of these programs.
3. Make subjective judgments on the relative value of different types of social changes.
4. Ensure stakeholders are knowledgeable and thoughtful about these limitations.
5. Encourage stakeholders to lay bare the assumptions and trade-offs involved in their grant making decisions.

However, the report does not provide a specific method for involving stakeholders in the evaluation and improvement of the corporate philanthropy program. Therefore, it may be necessary to develop a tailored approach that takes into account the unique needs and motivations of your stakeholders.

CGO >>> - Track and report monetary donations.

- Conduct internal surveys for employee insight.
 - Analyze philanthropy's impact on HR, marketing, risk, and budgeting.
 - Identify long-term financial benefits of philanthropy.
 - Review measurement studies, models, and evidence.
 - Interview senior management and professionals.
 - Adopt a systematic, data-based approach to quantify impacts and advocate for philanthropy.
 - Apply financial valuation methods, such as ROI analysis, to evaluate and compare the effectiveness of diverse social programs.
 - Use systematic measurement to bring rigor and discipline, quantifying the positive effects of corporate philanthropy.
 - Complement methodologies with a review of recent measurement studies, models, and evidence from complementary business disciplines and the social sector.
 - Recognize the strengths and limitations of various measurement approaches and adapt current methodologies to fit corporate giving programs.
-

Turn 10...

CEO >>> How can we develop a more comprehensive and innovative approach to

↳evaluating

the impact and effectiveness of our company's philanthropic initiatives that goes beyond traditional metrics and incorporates emerging trends in social impact measurement?

Corgi-AI >>> To develop a more comprehensive and innovative approach to evaluating

(continues on next page)

(continued from previous page)

the impact and effectiveness of a company's philanthropic initiatives, you can consider the following strategies:

1. Integrate both qualitative and quantitative methods: Use a combination of narrative accounts, case studies, and quantitative data to capture the full range of impacts. This can help to capture the intangible and long-term benefits that traditional metrics may not fully capture.
2. Adopt a multi-stakeholder perspective: Involve a diverse range of stakeholders in the evaluation process, including employees, beneficiaries, community members, and investors. This can help to ensure that the evaluation captures a wide range of perspectives and experiences.
3. Leverage technology and data analytics: Utilize advanced data collection and analysis tools, such as big data, machine learning, and artificial intelligence, to gather and analyze data on the impact of philanthropic initiatives. This can help to identify patterns and trends that may not be apparent through traditional methods.
4. Focus on outcomes and impact: Shift the focus from outputs (the activities and services provided) to outcomes (the short-term and long-term effects) and impact (the broader societal and environmental changes) of philanthropic initiatives. This can help to capture the full range of benefits and impacts of philanthropy.
5. Incorporate theory of change and logic models: Develop a clear theory of change or logic model that outlines the desired outcomes and impact of philanthropic initiatives. This can help to guide the evaluation process and ensure that it is aligned with the company's goals and values.
6. Use benchmarking and best practices: Compare your philanthropic initiatives and outcomes with those of similar organizations and initiatives to identify areas for improvement and innovation. This can help to benchmark your performance and identify best practices in the field.
7. Engage in continuous learning and improvement: Use the evaluation process as an opportunity for learning and improvement. Share the results and insights with stakeholders, and use them to inform future philanthropic initiatives and strategies.

By adopting these strategies, companies can develop a more comprehensive and innovative approach to evaluating the impact and effectiveness of their philanthropic initiatives, capturing a wider range of benefits and impacts, and driving meaningful and sustainable change.

- CGO >>> - Track and report monetary donations.
- Conduct internal surveys for employee insight.
- Analyze philanthropy's impact on HR, marketing, risk, and budgeting.
- Identify long-term financial benefits of philanthropy.
- Review measurement studies, models, and evidence.
- Interview senior management and professionals.
- Adopt a systematic, data-based approach to quantify impacts and advocate for philanthropy.

(continues on next page)

(continued from previous page)

- Apply financial valuation methods, such as ROI analysis, to evaluate and compare the effectiveness of diverse social programs.
 - Use systematic measurement to bring rigor and discipline, quantifying the positive effects of corporate philanthropy.
 - Complement methodologies with a review of recent measurement studies, models, and evidence from complementary business disciplines and the social sector.
 - Recognize the strengths and limitations of various measurement approaches and adapt current methodologies to fit corporate giving programs.
-

```
# Display the final turn
print("FINAL ANSWER")
print(wrap(memory))
```

FINAL ANSWER

- Track and report monetary donations.
- Conduct internal surveys for employee insight.
- Analyze philanthropy's impact on HR, marketing, risk, and budgeting.
- Identify long-term financial benefits of philanthropy.
- Review measurement studies, models, and evidence.
- Interview senior management and professionals.
- Adopt a systematic, data-based approach to quantify impacts and advocate for philanthropy.
- Apply financial valuation methods, such as ROI analysis, to evaluate and compare the effectiveness of diverse social programs.
- Use systematic measurement to bring rigor and discipline, quantifying the positive effects of corporate philanthropy.
- Complement methodologies with a review of recent measurement studies, models, and evidence from complementary business disciplines and the social sector.
- Recognize the strengths and limitations of various measurement approaches and adapt current methodologies to fit corporate giving programs.

For attribution, show the source reference documents which supported the responses.

```
# source document IDs of text chunks retrieved are tracked for attribution
print("Source documents reference id's:")
print(set(docs))
```

```
Source documents reference id's:
{'12', '5', '97', '9', '0', '102', '192', '1', '142', '8', '84', '2', '4', '13',
 '87'}
```

36.5 Multi-lingual

The Phi-4-multimodal model handles multilingual text in over 20 languages, including English, French, German, Hindi, Italian, Portuguese, Spanish, and Thai; and is capable of mathematics and coding tasks.

```
# Translate english to french
french = pipe(f"Please translate to French: {memory}", max_new_tokens=512)
print(wrap(french))
```

- Suivre et rapporter les dons monétaires.
- Conduire des enquêtes internes pour obtenir des insights sur les employés.
- Analyser l'impact de la philanthropie sur le RH, la marketing, le risque et le budget.
- Identifier les avantages financiers à long terme de la philanthropie.
- Examiner les études de mesure, les modèles et les preuves.
- Interroger les dirigeants seniors et les professionnels.
- Adopter une approche systématique, basée sur les données, pour quantifier les impacts et plaider en faveur de la philanthropie.
- Appliquer des méthodes de valorisation financière, telles que l'analyse de retour sur investissement (ROI), pour évaluer et comparer l'efficacité des programmes sociaux diversifiés.
- Utiliser des mesures systématiques pour apporter rigueur et discipline, quantifier les effets positifs de la philanthropie d'entreprise.
- Compléter les méthodologies par une revue des études de mesure récentes, des modèles et des preuves provenant de disciplines commerciales complémentaires et du secteur social.
- Reconnaître les forces et les limitations des différentes approches de mesure et adapter les méthodologies actuelles aux programmes de dons d'entreprise.

```
# Translate back to english
print(wrap(pipe(f"Please translate to English: {french}", max_new_tokens=512)))
```

- Track and report monetary donations.
- Conduct internal surveys to gain insights about employees.
- Analyze the impact of philanthropy on human resources, marketing, risk, and budget.
- Identify the long-term financial benefits of philanthropy.
- Review measurement studies, models, and evidence.
- Interview senior executives and professionals.
- Adopt a systematic, data-driven approach to quantify impacts and advocate for philanthropy.
- Apply financial valuation methods, such as return on investment (ROI) analysis, to evaluate and compare the effectiveness of diverse social programs.
- Use systematic measures to bring rigor and discipline, quantify the positive effects of corporate philanthropy.
- Complete methodologies by reviewing recent measurement studies, models, and evidence from complementary commercial disciplines and the social sector.
- Recognize the strengths and limitations of different measurement approaches and adapt current methodologies to corporate donation programs.

36.6 Vision language

Optical character recognition (OCR) and chart and table interpretation: figure 6 from MVCP document

```
# Load and display a test image
image = Image.open("assets/lim-fig6.png")
plt.imshow(image)
plt.axis('off')
```

```
(np.float64(-0.5), np.float64(953.5), np.float64(785.5), np.float64(-0.5))
```

Figure 6: A Framework for Measuring Employee Engagement and Corporate Philanthropy



Source: Adapted from Bhattacharya, C. B., Sen, S., & Korschun, D. (2008) and Bartel, C. (2001).

```
print(wrap(pipe("Describe what is presented in this image", image=image, max_new_tokens=512)))
```

The image is a diagram titled "Figure 6: A Framework for Measuring Employee Engagement and Corporate Philanthropy." It is divided into several sections, each with a specific focus. The top left section is labeled "CORPORATE PHILANTHROPY ACTIVITIES" and includes examples such as grants and employee volunteer programs. The top right section is labeled "BUSINESS IMPACT" and lists outcomes like increased output, sales, and profitability. Below these sections, there are three main columns. The first column on the left is labeled "EMPLOYEE NEEDS FULFILLED" and includes points such as self-enhancement, work-life integration, reputational shield, bridge to company, and collective self-esteem. The middle column is labeled "INTERMEDIATE OUTCOME TO BE TARGETED AND MEASURED" and includes "EMPLOYEE ATTITUDES" and "Sense of organizational identification." The third column on the right is labeled "JOB-RELATED BEHAVIORS" and lists outcomes like reduced absenteeism, retention, efficiency, co-operative behaviors, work effort, and advocacy. At the bottom, there is a section labeled

(continues on next page)

(continued from previous page)

"OTHER MODERATING FACTORS" which includes extrinsic incentives, employee characteristics, and employee perception of HR practices, work environment, management, and company capabilities. The source of the diagram is cited as adapted from Bhattacharya, C. B., Sen, S., & Korschun, D. (2008) and Bartel, C. (2001) .

Probing the model's image reasoning capabilities:

```
# Load and display a test image
image = Image.open("assets/MSDTRPL_EC013-H-2020-1602198302.webp")
plt.imshow(image)
plt.axis('off')
```

```
(np.float64(-0.5), np.float64(999.5), np.float64(562.5), np.float64(-0.5))
```



```
print(wrap(pipe("What is shown in this image?", image=image, max_new_tokens=512)))
```

The image depicts a large crowd of people, many of whom are holding up their hands. The crowd appears to be in a state of excitement or celebration, with some individuals raising their hands in the air. The people are dressed in a variety of clothing, including suits and casual attire. The background is filled with more people, suggesting that this is a public event or gathering.

```
print(wrap(pipe("Identify the movie that this scene is taken from, and explain your reasoning",
               image=image, max_new_tokens=128)))
```

This scene is from the movie 'Forrest Gump.' The reasoning behind this identification includes the distinctive style of the crowd, the casual yet significant attire of the individuals, and the overall chaotic yet focused atmosphere typical of a pivotal moment in the film. The scene captures a moment of intense public reaction, which is a recurring theme in the movie.

```
print(wrap(pipe("Please give your best estimate of the number of people in this image  
→",  
    image=image, max_new_tokens=32)))
```

There are approximately 30 people visible in the image.

```
print(wrap(pipe("Briefly recount a specific instance of corporate philanthropy in the  
→plot of the movie Trading Places?",  
    max_new_tokens=512)))
```

In the movie "Trading Places," a notable instance of corporate philanthropy occurs when Louis Winthorpe III, played by Eddie Murphy, and Billy Ray Valentine, played by Dan Akroyd, are swapped due to a bet. Louis, who is a successful commodities trader, is given a chance to live a life of luxury and privilege, while Billy, a struggling street trader, is given Louis' life.

During this period, Louis, who is now living a life of luxury, decides to use his newfound wealth to help those in need. He donates a significant amount of money to a local homeless shelter, which is a clear act of corporate philanthropy. This act not only helps the shelter but also serves as a turning point in Louis' character development, as he begins to understand the value of giving back to the community.

This instance of corporate philanthropy in the movie highlights the importance of using wealth and resources to help those in need, and it also serves as a reminder that even those who are successful can make a positive impact on society.

36.7 Audio

Automatic speech recognition (ASR) and speech translation:

```
# Perform speech-to-text on an audio file  
url = "assets/pork-bellies!.mp3"  
audio, samplerate = soundfile.read(url)  
response = pipe("Transcribe the audio to text", audio=[(audio, samplerate)], max_new_  
→tokens=512)  
print(wrap(response))
```

Pork bellies! I have a hunch something very exciting is going to happen in the pork belly market this morning.

```
from playsound import playsound  
playsound(url)    # play the original sound file
```

References:

- Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal†, Heinrich Kütter, Mike Lewis†, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, Douwe Kiela, 2021, Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks
- Chen et al. (2017), “Reading Wikipedia to Answer Open-Domain Questions.”

- Ram et al. (2023), “In-Context Retrieval-Augmented Language Models.”
- Kyle Mahowald and Anna A. Ivanova and Idan A. Blank and Nancy Kanwisher and Joshua B. Tenenbaum and Evelina Fedorenko, 2024, Dissociating language and thought in large language models, <https://arxiv.org/abs/2301.06627>
- Hicks, Michael Townsen, Humphries, James and Slater, Joe, 2024, “ChatGPT is bullshit”, Ethics and Information Technology Volume 26, Issue 2, <https://doi.org/10.1007/s10676-024-09775-5>
- Lim, Terence, 2010, Measuring the value of corporate philanthropy: Social impact, business benefits, and investor returns. published by CECP.