
Financial Data Science Python Notebooks

Terence Lim

Jun 07, 2024

CONTENTS

| | |
|---|------------|
| 1 Stock Prices | 3 |
| 1.1 Universe selection | 4 |
| 1.2 Measuring Stock Returns | 6 |
| 1.3 SQL | 11 |
| 1.4 Properties of stock returns | 13 |
| 1.5 Stock Delistings | 19 |
| 2 Jegadeesh-Titman Rolling Portfolios | 21 |
| 2.1 Price Momentum | 22 |
| 2.2 Hypothesis Testing | 25 |
| 3 Fama-French Portfolio Sorts | 33 |
| 3.1 Value effect | 34 |
| 3.2 Bivariate sorts | 35 |
| 3.3 Linear regression | 38 |
| 3.4 Structural change test | 41 |
| 3.5 Small size effect | 43 |
| 3.6 Price momentum effect | 46 |
| 3.7 Price reversal effect | 49 |
| 4 Fama-Macbeth Cross-sectional Regressions | 53 |
| 4.1 Mean variance optimization | 54 |
| 4.2 Implied alphas | 59 |
| 4.3 Cross-sectional regressions | 63 |
| 4.4 Nonlinear regression | 66 |
| 4.5 CSR with individual stocks | 67 |
| 5 Contrarian Trading | 71 |
| 5.1 Mean reversion | 72 |
| 5.2 Implementation shortfall | 75 |
| 5.3 Structural break with unknown changepoint | 76 |
| 5.4 Information coefficient | 78 |
| 5.5 Performance evaluation | 79 |
| 6 Quant Factors | 83 |
| 6.1 Factor investing | 84 |
| 6.2 Cluster analysis | 137 |
| 7 Event Study | 145 |
| 7.1 Abnormal returns | 146 |
| 7.2 Cross correlation | 158 |

| | | |
|-----------|---|------------|
| 7.3 | Multiple testing problem | 160 |
| 8 | Economic Data Revisions | 173 |
| 8.1 | Retrieving data from the web | 174 |
| 8.2 | FRED | 178 |
| 8.3 | FRED-MD and FRED-QD | 186 |
| 8.4 | Outliers | 189 |
| 9 | Linear Regression Diagnostics | 191 |
| 9.1 | Model assumptions | 192 |
| 9.2 | Residual plots | 195 |
| 10 | Econometric Forecasts | 201 |
| 10.1 | Seasonality | 202 |
| 10.2 | Stationarity | 204 |
| 10.3 | Autocorrelation | 205 |
| 10.4 | Time Series | 206 |
| 10.5 | Forecasting | 209 |
| 11 | Approximate Factor Models | 217 |
| 11.1 | Integration order | 218 |
| 11.2 | Approximate factor model | 221 |
| 12 | State Space Models | 229 |
| 12.1 | Hidden Markov Model | 230 |
| 12.2 | Gaussian Mixture Model | 234 |
| 13 | Term Structure of Interest Rates | 237 |
| 13.1 | Interest Rates | 238 |
| 13.2 | Yield Curve | 241 |
| 13.3 | Bootstrap | 243 |
| 13.4 | Principal Component Analysis | 248 |
| 13.5 | Singular Value Decomposition | 250 |
| 14 | Bond Returns and Risks | 253 |
| 14.1 | Term structure changes | 253 |
| 14.2 | Bond risk factors | 258 |
| 15 | Conditional Volatility and VaR | 269 |
| 15.1 | Value at Risk | 270 |
| 15.2 | Conditional volatility models | 279 |
| 15.3 | Backtesting VaR | 283 |
| 16 | Covariance Matrix | 285 |
| 16.1 | Portfolio risk analysis | 286 |
| 16.2 | Covariance matrix estimation | 291 |
| 17 | Options Pricing | 299 |
| 17.1 | Options | 300 |
| 17.2 | Binomial Tree | 304 |
| 17.3 | Black-Scholes-Merton | 310 |
| 17.4 | Monte Carlo Simulation | 318 |
| 18 | Market Microstructure | 323 |
| 18.1 | TAQ tick data | 324 |
| 18.2 | Liquidity by firm size | 327 |

| | |
|--|------------|
| 18.3 Sampling frequency | 329 |
| 18.4 Volatility from tick data | 335 |
| 18.5 Intraday liquidity | 338 |
| 19 Event Risk | 343 |
| 19.1 Earnings misses | 343 |
| 19.2 Poisson regression | 346 |
| 19.3 Credit losses | 348 |
| 19.4 Actuarial losses | 349 |
| 20 Principal Customers Graph | 351 |
| 20.1 Supply chain | 351 |
| 20.2 Graph properties | 352 |
| 21 Community Detection | 359 |
| 21.1 Industry taxonomy | 360 |
| 21.2 Community structure | 363 |
| 22 Graph Centrality | 369 |
| 22.1 Centrality measures | 370 |
| 22.2 BEA Input-Output Use Tables | 371 |
| 23 Link Prediction | 387 |
| 23.1 Link prediction algorithms | 390 |
| 23.2 Accuracy metrics | 391 |
| 24 Spatial Regression | 401 |
| 24.1 Earnings surprise | 402 |
| 24.2 Spatial dependence models | 403 |
| 25 FOMC Topic Modeling | 409 |
| 25.1 FOMC minutes | 410 |
| 25.2 Tokenization | 411 |
| 25.3 Topic models | 413 |
| 26 Management Sentiment Analysis | 421 |
| 26.1 Sentiment analysis | 422 |
| 26.2 SEC Edgar company filings | 423 |
| 27 Business Text Analysis | 433 |
| 27.1 Syntactic analysis | 434 |
| 27.2 Word2Vec | 437 |
| 27.3 Density-based clustering | 439 |
| 27.4 Exploring Spacy | 443 |
| 28 Classification | 445 |
| 28.1 Text classification | 446 |
| 28.2 Classification models | 448 |
| 28.3 Evaluation | 452 |
| 29 Regression | 461 |
| 29.1 Regression models | 464 |
| 29.2 Evaluation | 485 |
| 30 Deep Learning | 487 |
| 30.1 Text classification | 488 |

| | |
|--|------------|
| 30.2 Word Embeddings | 490 |
| 30.3 FeedForward Neural Network | 491 |
| 31 Recurrent Neural Networks | 499 |
| 31.1 Recurrent Neural Network | 501 |
| 31.2 Dynamic Factor Models | 507 |
| 32 Convolutional Neural Networks | 511 |
| 32.1 Temporal Convolutional Net (TCN) | 513 |
| 32.2 Vector Autoregression | 523 |
| 33 Reinforcement Learning | 531 |
| 33.1 Retirement spending policy | 533 |
| 33.2 Deep reinforcement learning | 535 |
| 34 FedSpeak Language Modeling | 547 |
| 34.1 Transformers | 549 |
| 34.2 Language modeling | 551 |
| 35 LLM Prompting | 557 |
| 35.1 Llama-3 LLM | 557 |
| 35.2 Ollama server | 557 |
| 35.3 Kaggle platform | 558 |
| 35.4 Prompting techniques | 560 |
| 36 LLM Text Summarization | 567 |
| 36.1 HuggingFace APIs | 568 |
| 36.2 ChatGPT LLM | 578 |
| 36.3 Text summarization and evaluation | 578 |
| 37 LLM Finetuning | 579 |
| 37.1 Unslot APIs | 581 |
| 37.2 Low-rank adaptation | 582 |
| 37.3 HuggingFace datasets | 583 |
| 37.4 Supervised fine-tuning | 584 |
| 38 LLM RAG, Chatbots and Agents | 589 |
| 38.1 Retrieval Augmented Generation | 590 |
| 38.2 Chatbot | 593 |
| 38.3 Agents | 596 |

Financial Data Science with **FinDS** Python package in Jupyter-notebooks:

- use database engines SQL, MongoDB, Redis
- interfaces for
 - structured data from CRSP, Compustat, IBES, TAQ
 - APIs from ALFRED, BEA
 - unstructured data from SEC Edgar, Federal Reserve websites
 - academic websites by Ken French, Loughran and MacDonald, Hoberg and Phillips
- recipes for econometrics, finance, graphs, event studies, backtesting
- applications of statistics, machine learning, neural networks and large language models

Topics

| notebook | Financial | Data | Science |
|------------------------|--|-------------------------------|--|
| stock_prices | Stock distributions, delistings | CRSP stocks | Statistical moments |
| jegadeesh_titman | Overlapping portfolios; Momentum effect | CRSP stocks | Hypothesis testing; Newey-West estimator |
| fama_french | Portfolio sorts; Value effect | CRSP stocks; Compustat | Linear regression; |
| fama_macbeth | Cross-sectional Regressions; CAPM | Ken French research library | Non-linear regression; Quadratic optimization |
| weekly_reversals | Mean reversion; Implementation shortfall | CRSP stocks | Structural breaks; Performance evaluation |
| quant_factors | Factor investing; Backtests | CRSP stocks; Compustat; IBES | Cluster analysis |
| event_study | Event studies | S&P key developments | Multiple testing; FFT |
| economic_releases | Economic data revisions; Employment payrolls | ALFRED | Outliers |
| regression_diagnostics | Consumer and producer prices | FRED | Linear regression diagnostics; Residual analysis |
| econometric_forecast | Production and Inflation | FRED | Time series analysis |
| approximate_factors | Approximate factor models | FRED-MD | Unit root test |
| economic_states | State space models | FRED-MD | Gaussian Mixture; HMM |
| term_structure | Interest rates | FRED yield curve | SVD |
| bond_returns | Bond risk factors | FRED bond returns | PCA |
| option_pricing | Binomial tree; Black-Scholes-Merton and the Greeks | Simulations | Monte Carlo simulation |
| conditional_volatility | Value at risk | FRED crypto-currencies | EWMA; GARCH |
| covariance_matrix | Portfolio risk | Fama-French industries | Covariance matrix estimation |
| market_microstructure | Market impact; Liquidity risk | TAQ tick data | High frequency volatility |
| event_risk | Earnings misses | IBES | Poisson regression; GLM |
| customer_ego | Supply chain | Compustat principal customers | Graph networks |

continues on next page

Table 1 – continued from previous page

| notebook | Financial | Data | Science |
|------------------------|--------------------------|--|---------------------------------------|
| industry_community | Industry sectors | Hoberg and Phillips research library | Community detection |
| bea_centrality | Input-output tables | Bureau of Economic Analysis | Graph centrality |
| link_prediction | Product markets | Hoberg and Phillips | Link prediction |
| spatial_regression | Earnings surprises | IBES Hoberg and Phillips | Spatial regression |
| fomc_topics | FOMC meetings | Federal Reserve website | Topic models |
| mdu_sentiment | 10-K filings | SEC Edgar; Loughran and Macdonald research library | Sentiment analysis |
| business_description | 10-K filings | SEC Edgar | POS tagging; Density-based clustering |
| classification_models | Industry classification | SEC Edgar | Classification |
| regression_models | Macroeconomic forecasts | FRED-MD | Regression |
| deep_classifier | Industry classification | SEC Edgar | Neural networks; Word embeddings |
| recurrent_net | Macroeconomic models | FRED-MD | RNN; Dynamic factor models |
| convolutional_net | Macroeconomic forecasts | FRED-MD | CNN; Vector autoregression |
| reinforcement_learning | Spending policy | SBBI | Reinforcement learning |
| fomc_language | Fedspeak | FOMC meetings minutes | Language modelling; Transformers |
| sentiment_llm | Financial news sentiment | Kaggle | LLM prompt engineering |
| summarization_llm | 10-K filings | SEC Edgar | LLM text summarization |
| finetune_llm | Industry classification | SEC Edgar | LLM fine-tuning |
| rag_agent | Corporate philanthropy | textbooks | LLM RAG, chatbots, agents |

Resources

1. Online Jupyter-book, or download pdf
2. FinDS API reference
3. FinDS repo
4. Jupyter notebooks repo

Contact

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

CHAPTER
ONE

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

Concepts:

- Fama-French universe and size deciles
- Stock splits, dividends and delistings
- Properties of stock returns

References:

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

```
import numpy as np
import scipy
import pandas as pd
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
from tqdm import tqdm
from finds.database import SQL, RedisDB
from finds.structured import BusDay, CRSP, CRSPBuffer, Benchmarks
from finds.utils import Finder, plot_date
from secret import credentials, CRSP_DATE, paths
VERBOSE = 0

# %matplotlib qt
```

```
# 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)
```

(continues on next page)

(continued from previous page)

```
find = Finder(sql)
outdir = paths['scratch']
```

1.1 Universe selection

The Center for Research in Security Prices (CRSP) provides the most widely used data for academic research into US stocks. Following Fama and French (1992), the CRSP universe is trimmed to only include US-domiciled, exchange-listed stocks:

- shrcd in [10, 11]
- exchcd in [1, 2, 3]

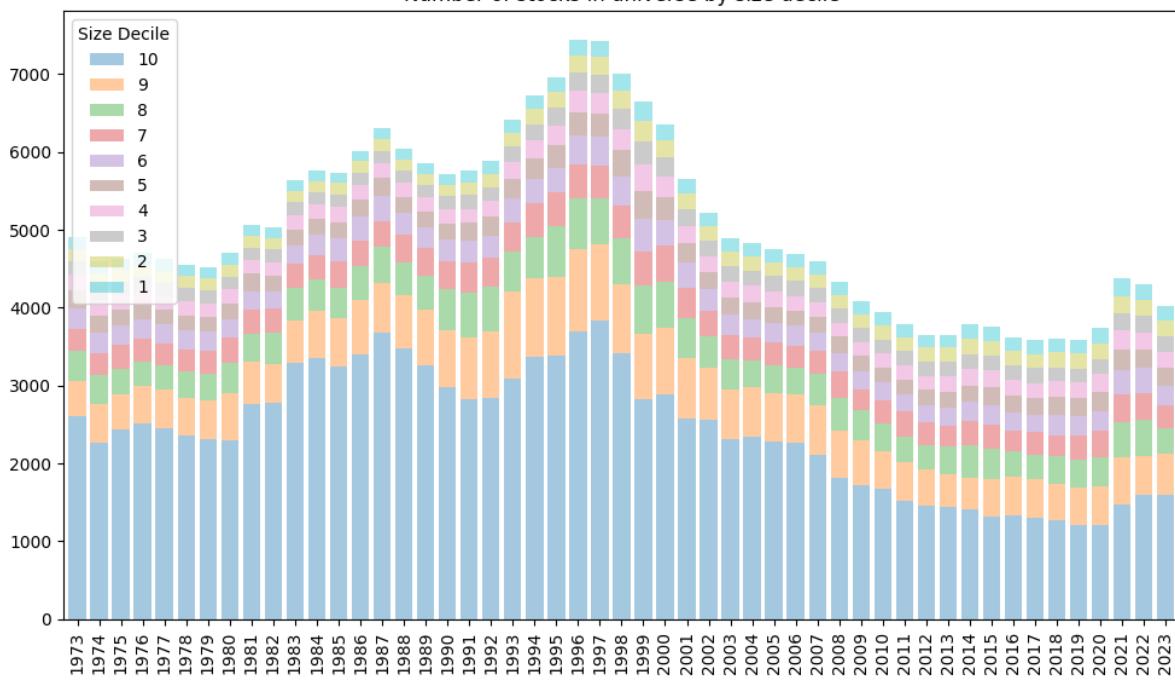
The universe is typically divided into 10 market capitalization-based deciles (with the 10th decile comprising the smallest stocks) whose breakpoints are determined only from those stocks listed on the NYSE. For a company with multiple classes securities, the combined market value of the company is used for all classes of its securities. The plots below of the market cap breakpoints of and the number of stocks in each decile by year are consistent with the observations by Stulz (2018) and others of the shrinking universe of public US firms and increased concentration of mega-cap stocks in recent years.

The shrcd (shares outstanding) field in CRSP is in units of thousands of shares. Hence all calculations and derivations (such as market capitalization) need to be multiplied by 1000 to get the true value.

```
# retrieve universe of stocks annually from 1981
start = bd.endyr(19731231)
rebals = bd.date_range(start, CRSP_DATE, freq='12M')
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)
                             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



```
# For assigning into size deciles, sum up the company's total market
# capitalization instead of separate capital values each security if multiple classes
names = find('ALPHABET', 'comnam').groupby('permno').tail(1).set_index('permno')
find(names['permco'].iloc[0], identifier='permco')
```

| | date | comnam | ncusip | shrccls | ticker | permno | nameendt | shrcd | \ |
|----|----------|--------------|----------|---------|----------|---------|----------|--------|---|
| 0 | 20040819 | GOOGLE INC | 38259P50 | A | GOOG | 90319 | 20050818 | 11 | |
| 1 | 20050819 | GOOGLE INC | 38259P50 | A | GOOG | 90319 | 20060720 | 11 | |
| 2 | 20060721 | GOOGLE INC | 38259P50 | A | GOOG | 90319 | 20080131 | 11 | |
| 3 | 20080201 | GOOGLE INC | 38259P50 | A | GOOG | 90319 | 20140402 | 11 | |
| 4 | 20140403 | GOOGLE INC | 38259P70 | C | GOOG | 14542 | 20150423 | 11 | |
| 5 | 20140403 | GOOGLE INC | 38259P50 | A | GOOGL | 90319 | 20150423 | 11 | |
| 6 | 20150424 | GOOGLE INC | 38259P70 | C | GOOG | 14542 | 20151004 | 11 | |
| 7 | 20150424 | GOOGLE INC | 38259P50 | A | GOOGL | 90319 | 20151004 | 11 | |
| 8 | 20151005 | ALPHABET INC | 02079K10 | C | GOOG | 14542 | 20180201 | 11 | |
| 9 | 20151005 | ALPHABET INC | 02079K30 | A | GOOGL | 90319 | 20180201 | 11 | |
| 10 | 20180202 | ALPHABET INC | 02079K10 | C | GOOG | 14542 | 20231012 | 11 | |
| 11 | 20180202 | ALPHABET INC | 02079K30 | A | GOOGL | 90319 | 20231012 | 11 | |
| 12 | 20231013 | ALPHABET INC | 02079K10 | C | GOOG | 14542 | 20231229 | 11 | |
| 13 | 20231013 | ALPHABET INC | 02079K30 | A | GOOGL | 90319 | 20231229 | 11 | |
| | | | | | | | | | |
| | exchcd | siccd | tsymbol | naics | primexch | trdstat | secstat | permco | |
| 0 | 3 | 7375 | GOOG | 514191 | Q | A | R | 45483 | |
| 1 | 3 | 7375 | GOOG | 518111 | Q | A | R | 45483 | |
| 2 | 3 | 7375 | GOOG | 518112 | Q | A | R | 45483 | |
| 3 | 3 | 7375 | GOOG | 519130 | Q | A | R | 45483 | |
| 4 | 3 | 7375 | GOOG | 519130 | Q | A | R | 45483 | |
| 5 | 3 | 7375 | GOOGL | 519130 | Q | A | R | 45483 | |
| 6 | 3 | 7375 | GOOG | 519190 | Q | A | R | 45483 | |
| 7 | 3 | 7375 | GOOGL | 519190 | Q | A | R | 45483 | |
| 8 | 3 | 7375 | GOOG | 519190 | Q | A | R | 45483 | |

(continues on next page)

(continued from previous page)

| | | | | | | | | |
|----|---|------|-------|--------|---|---|---|-------|
| 9 | 3 | 7375 | GOOGL | 519190 | Q | A | R | 45483 |
| 10 | 3 | 7375 | GOOG | 334419 | Q | A | R | 45483 |
| 11 | 3 | 7375 | GOOGL | 334419 | Q | A | R | 45483 |
| 12 | 3 | 7375 | GOOG | 541511 | Q | A | R | 45483 |
| 13 | 3 | 7375 | GOOGL | 541511 | Q | A | R | 45483 |

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

| | permno | cap | capco | decile | nyse | siccd | prc | naics |
|-------|-------------|--------------|-------|--------|-------|-------|--------|--------|
| 14542 | 806824250.0 | 1.633510e+09 | | 1 | False | 7375 | 140.93 | 541511 |
| 90319 | 826685420.0 | 1.633510e+09 | | 1 | False | 7375 | 139.69 | 541511 |

1.2 Measuring Stock Returns

In addition to gains from appreciated stock prices, adjusted for stock splits (facpr in CRSP), investors' total holding returns also include ordinary cash dividends (divamt in CRSP).

The total holding return on an asset from $t-1$ to t includes both price appreciation and any dividends paid on record date t : $R_t = \frac{P_t + D_t - P_{t-1}}{P_{t-1}}$. Ex-dividend means a company's dividend allocations have been specified. The ex-dividend date or "ex-date" is usually one business day before the record date.

The return of an asset over multiple periods is calculated using the product of the simple returns in each period: $1 + R_T = \prod_{t=1}^T (1 + R_t)$. This implicitly assumes that dividends have been reinvested in the same asset.

There are also continuously compounded returns, or log returns. These are computed as the difference of the natural logarithm of the price: $\ln P_t - \ln P_{t-1}$

1.2.1 Retrieve Yahoo Finance prices

- The historical "close" from Yahoo Finance is the split-adjusted price
- The historical "adjclose" from Yahoo Finance is equal to the dividend- and split-adjusted price: Daily changes of "adjclose" reflects the total holdings returns which includes the effect of both stock splits and cash dividends received.

```
# Prices from Yahoo Finance
from yahoo import get_price
ticker = 'AAPL'
date = 20200101
df = get_price(ticker, start_date=date, verbose=VERBOSE)
df.rename_axis(ticker)
prices_columns = df.columns
df.round(2)
```

| | open | high | low | close | adjClose | volume |
|----------|-------|-------|-------|-------|----------|-------------|
| 20200103 | 74.29 | 75.14 | 74.12 | 74.36 | 72.35 | 146322800.0 |
| 20200106 | 73.45 | 74.99 | 73.19 | 74.95 | 72.93 | 118387200.0 |
| 20200107 | 74.96 | 75.22 | 74.37 | 74.60 | 72.58 | 108872000.0 |

(continues on next page)

(continued from previous page)

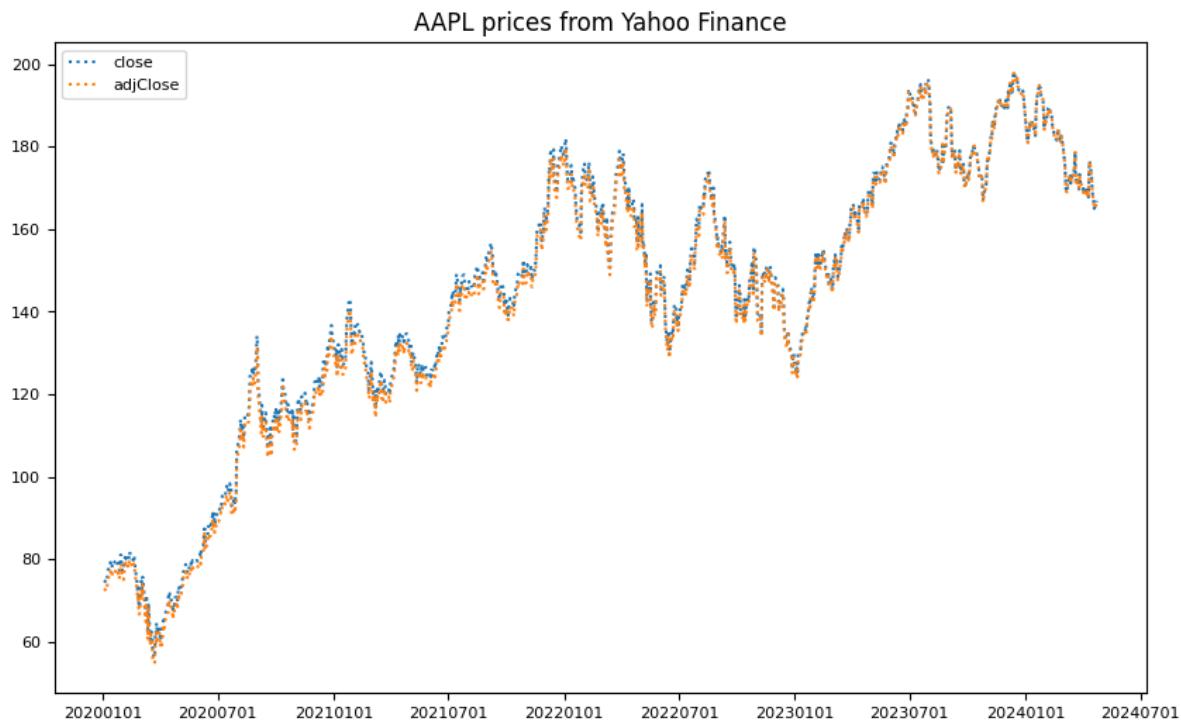
| | | | | | | |
|----------|--------|--------|--------|--------|--------|-------------|
| 20200108 | 74.29 | 76.11 | 74.29 | 75.80 | 73.75 | 132079200.0 |
| 20200109 | 76.81 | 77.61 | 76.55 | 77.41 | 75.32 | 170108400.0 |
| ... | ... | ... | ... | ... | ... | ... |
| 20240417 | 169.61 | 170.65 | 168.00 | 168.00 | 168.00 | 50901200.0 |
| 20240418 | 168.03 | 168.64 | 166.55 | 167.04 | 167.04 | 43122900.0 |
| 20240419 | 166.21 | 166.40 | 164.08 | 165.00 | 165.00 | 67772100.0 |
| 20240422 | 165.52 | 167.26 | 164.77 | 165.84 | 165.84 | 48116400.0 |
| 20240423 | 165.35 | 167.05 | 164.92 | 166.90 | 166.90 | 48917700.0 |

[1083 rows x 6 columns]

```
df = get_price(ticker, start_date=date, verbose=VERBOSE)
```

```
fig, ax = plt.subplots(figsize=(10, 6))
plot_date(df[['close', 'adjClose']], legend1=['close', 'adjClose'],
           title=f'{ticker} prices from Yahoo Finance',
           ls=':', fontsize=8, rotation=0, cn=0, ax=ax)
```

<Axes: title={'center': 'AAPL prices from Yahoo Finance'}>



Estimate dividend amount

Differences between daily changes in adjClose and close are estimates of dividend amounts (scaled by prior adjClose) on ex-dates.

```
# Lookup AAPL's permno identifier
find = Finder(sql)
permno = find('AAPL').tail(1)['permno'].iloc[0]
```

```
# dividend yield is excess of change in adjClose over change in close
df['divyld'] = ((df['adjClose'] / df['adjClose'].shift())
                 / (df['close'] / df['close'].shift())) - 1
# ex-dates inferred to be when amount is larger than rounding error
exdt = df.index[abs(df['divyld'])] > .0001
yahoo_div = df.loc[exdt, 'divyld']
```

1.2.2 Stock Splits and Dividends

Retrieve CRSP distributions

Dividend amount per share

```
crsp_div = DataFrame(**sql.run(f"""
SELECT divamt, exdt, retx, prc FROM dist INNER JOIN daily
ON dist.permno = daily.permno AND dist.exdt = daily.date AND dist.exdt <= {CRSP_DATE}
WHERE dist.permno={permno} AND dist.exdt>{date} AND dist.divamt > 0
""").strip()))
crsp_div = crsp_div.set_index('exdt').sort_index()
crsp_div
```

| | divamt | retx | prc |
|----------|--------|-----------|--------|
| exdt | | | |
| 20200207 | 0.770 | -0.015928 | 320.03 |
| 20200508 | 0.820 | 0.021038 | 310.13 |
| 20200807 | 0.820 | -0.024495 | 444.45 |
| 20201106 | 0.205 | -0.002856 | 118.69 |
| 20210205 | 0.205 | -0.004586 | 136.76 |
| 20210507 | 0.220 | 0.003623 | 130.21 |
| 20210806 | 0.220 | -0.006256 | 146.14 |
| 20211105 | 0.220 | 0.002120 | 151.28 |
| 20220204 | 0.220 | -0.002950 | 172.39 |
| 20220506 | 0.230 | 0.003253 | 157.28 |
| 20220805 | 0.230 | -0.002774 | 165.35 |
| 20221104 | 0.230 | -0.003600 | 138.38 |
| 20230210 | 0.230 | 0.000928 | 151.01 |
| 20230512 | 0.240 | -0.006791 | 172.57 |
| 20230811 | 0.240 | -0.001011 | 177.79 |
| 20231110 | 0.240 | 0.021874 | 186.40 |

```
# Compare CRSP actual and Yahoo Finance estimated dividend amounts
div = crsp_div.join(yahoo_div)
div['yahoo_div'] = div['divyld'] * (div['prc'] / (1 + div['retx']))
div[['divamt', 'yahoo_div', 'prc']].round(3)
```

| | divamt | yahoo_div | prc |
|----------|--------|-----------|--------|
| exdt | | | |
| 20200207 | 0.770 | 0.772 | 320.03 |
| 20200508 | 0.820 | 0.822 | 310.13 |
| 20200807 | 0.820 | 0.822 | 444.45 |
| 20201106 | 0.205 | 0.205 | 118.69 |
| 20210205 | 0.205 | 0.205 | 136.76 |
| 20210507 | 0.220 | 0.220 | 130.21 |

(continues on next page)

(continued from previous page)

| | | | |
|----------|-------|-------|--------|
| 20210806 | 0.220 | 0.220 | 146.14 |
| 20211105 | 0.220 | 0.220 | 151.28 |
| 20220204 | 0.220 | 0.220 | 172.39 |
| 20220506 | 0.230 | 0.230 | 157.28 |
| 20220805 | 0.230 | 0.230 | 165.35 |
| 20221104 | 0.230 | 0.230 | 138.38 |
| 20230210 | 0.230 | 0.230 | 151.01 |
| 20230512 | 0.240 | 0.240 | 172.57 |
| 20230811 | 0.240 | 0.240 | 177.79 |
| 20231110 | 0.240 | 0.240 | 186.40 |

Split adjustment factors

```
crsp_fac = DataFrame(**sql.run(f"""
SELECT 1 + facpr as facpr, exdt FROM dist
WHERE permno={permno} AND exdt>{date} AND facpr > 0.0
""".strip())).set_index('exdt').sort_index(ascending=False).cumprod()
```

```
# Retrieve CRSP raw prices around ex-dates
crsp_prc = DataFrame(**sql.run(f"""
SELECT date, prc from daily
WHERE permno={permno} AND date>{bd.offset(crsp_fac.index[0],-5)}
AND date<{bd.offset(crsp_fac.index[-1],5)}
""".strip())).set_index('date')
```

The Factor to Adjust Price, facpr, can be used to adjust prices for distributions (usually stock dividends and splits) so that stock prices before and after the distribution are comparable.

The 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

```
# Compare split-adjusted CRSP price to Yahoo Finance close price
crsp_fac = crsp_fac.reindex(crsp_prc.index) \
    .shift(-1) \
    .bfill() \
    .fillna(1) \
    .join(crsp_prc)
crsp_fac['prc_adjusted'] = crsp_fac['prc'] / crsp_fac['facpr']
crsp_fac[['prc', 'prc_adjusted']].join(df['close'], how='left').round(3)
```

| date | prc | prc_adjusted | close |
|----------|--------|--------------|---------|
| 20200825 | 499.30 | 124.825 | 124.825 |
| 20200826 | 506.09 | 126.522 | 126.522 |
| 20200827 | 500.04 | 125.010 | 125.010 |
| 20200828 | 499.23 | 124.808 | 124.808 |
| 20200831 | 129.04 | 129.040 | 129.040 |
| 20200901 | 134.18 | 134.180 | 134.180 |

(continues on next page)

(continued from previous page)

| | | | |
|----------|--------|---------|---------|
| 20200902 | 131.40 | 131.400 | 131.400 |
| 20200903 | 120.88 | 120.880 | 120.880 |
| 20200904 | 120.96 | 120.960 | 120.960 |

1.2.3 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, 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)
```

| permno | date | prc | original_ret | retx | dlstcd | dlret | ret |
|--------|----------|--------|--------------|---------|--------|---------|---------|
| 10275 | 20041130 | NaN | NaN | NaN | 584 | -0.2703 | -0.2703 |
| 10418 | 20041130 | NaN | NaN | NaN | 520 | 0.0509 | 0.0509 |
| 11194 | 20041130 | NaN | NaN | NaN | 233 | 0.0066 | 0.0066 |
| 12010 | 20041130 | NaN | NaN | NaN | 587 | 0.0490 | 0.0490 |
| 20459 | 20041130 | NaN | NaN | NaN | 231 | 0.0724 | 0.0724 |
| 32897 | 20041130 | NaN | NaN | NaN | 584 | -0.2357 | -0.2357 |
| 55589 | 20041130 | NaN | NaN | NaN | 233 | -0.0114 | -0.0114 |
| 64290 | 20041130 | 13.70 | 0.0178 | 0.0178 | 233 | 0.0000 | 0.0178 |
| 67708 | 20041130 | NaN | NaN | NaN | 233 | 0.0164 | 0.0164 |
| 69593 | 20041130 | NaN | NaN | NaN | 331 | 0.0998 | 0.0998 |
| 69681 | 20041130 | NaN | NaN | NaN | 584 | -0.4853 | -0.4853 |
| 70447 | 20041130 | 5.80 | -0.2246 | -0.2246 | 570 | NaN | -0.4572 |
| 75606 | 20041130 | NaN | NaN | NaN | 520 | -0.2466 | -0.2466 |
| 75684 | 20041130 | NaN | NaN | NaN | 233 | 0.0094 | 0.0094 |
| 76306 | 20041130 | NaN | NaN | NaN | 241 | -0.0329 | -0.0329 |
| 76691 | 20041130 | NaN | NaN | NaN | 582 | 0.0127 | 0.0127 |
| 77838 | 20041130 | NaN | NaN | NaN | 520 | -0.0041 | -0.0041 |
| 79149 | 20041130 | NaN | NaN | NaN | 574 | -0.2088 | -0.2088 |
| 79523 | 20041130 | NaN | NaN | NaN | 233 | 0.0043 | 0.0043 |
| 80211 | 20041130 | NaN | NaN | NaN | 470 | 0.0186 | 0.0186 |
| 80714 | 20041130 | NaN | NaN | NaN | 332 | 0.0337 | 0.0337 |
| 82491 | 20041130 | 4.21 | -0.0644 | -0.0644 | 551 | 0.0689 | 0.0000 |
| 83583 | 20041130 | 125.10 | 0.2810 | 0.2810 | 241 | 0.0624 | 0.3608 |

(continues on next page)

(continued from previous page)

| | | | | | | | |
|-------|----------|-------|--------|--------|-----|---------|---------|
| 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 SQL

- SQL: Structured query language (SQL) is a standard language for database creation and manipulation.
- MySQL: A database management system, not a programming language, that uses structured query language (SQL) to manage data inside.
- SQLAlchemy: A toolkit of Python SQL for using an SQL database.
- `pandas.read_sql(str, connection)`: Read SQL query or database table into a DataFrame.

SQL Cheatsheet:

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.

```
# inner join of identifier tables

# double up the % in a pandas string
pd.read_sql("select * from benchident where name like '%%momentum%%'", con=sql.engine)
```

| | permno | name | item |
|---|----------|---------------------------|------|
| 0 | Mom | F-F_Momentum_Factor_daily | 0 |
| 1 | Mom (mo) | F-F_Momentum_Factor | 0 |

```
cmd = """
select * from benchident inner join benchmarks
on benchident.permno = benchmarks.permno
where benchident.name like '%%momentum%%'
    and benchident.permno like '%%(mo)%%'
limit 5
"""

pd.read_sql(cmd, con=sql.engine)
```

| | permno | name | item | permno | date | ret |
|---|----------|---------------------|------|----------|----------|---------|
| 0 | Mom (mo) | F-F_Momentum_Factor | 0 | Mom (mo) | 19270131 | 0.0036 |
| 1 | Mom (mo) | F-F_Momentum_Factor | 0 | Mom (mo) | 19270228 | -0.0214 |
| 2 | Mom (mo) | F-F_Momentum_Factor | 0 | Mom (mo) | 19270331 | 0.0361 |
| 3 | Mom (mo) | F-F_Momentum_Factor | 0 | Mom (mo) | 19270430 | 0.0430 |
| 4 | Mom (mo) | F-F_Momentum_Factor | 0 | Mom (mo) | 19270531 | 0.0300 |

1.4 Properties of stock returns

1.4.1 Long-run Market Averages

- total stock returns
- dividend yield
- shares trading turnover

Equal-weighted time-series means of annual cap-weighted cross-sectional averages of universe stocks

```
# Loop over the 20-year eras, and compute means of annual cap-weighted averages
years = [2005, 1985, 1965, 1945, 1925]
results = DataFrame(columns=['divyld', 'turnover', 'means'])
for era in tqdm(years):
    label = f'{era+1}-{min(era+20, CRSP_DATE // 10000)}'
    divyld = {}
    means = {}
    turnover = {}
    for year in bd.date_range(era, min(CRSP_DATE // 10000, era+20), freq=12):
        univ = crsp.get_universe(bd.endyr(year)) # screen for universe stocks

        # Retrieve and compute cap-weighted average of annual 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 and compute cap-weighted average of annualized turnover
        cmd = f"""
        select permno, 252*AVG(vol/(shrout*1000)) AS turnover 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()
        turnover[year] = df['turnover'].dot(df['cap']) / df['cap'].sum()

        # Retrieve and compute 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())))

```

(continues on next page)

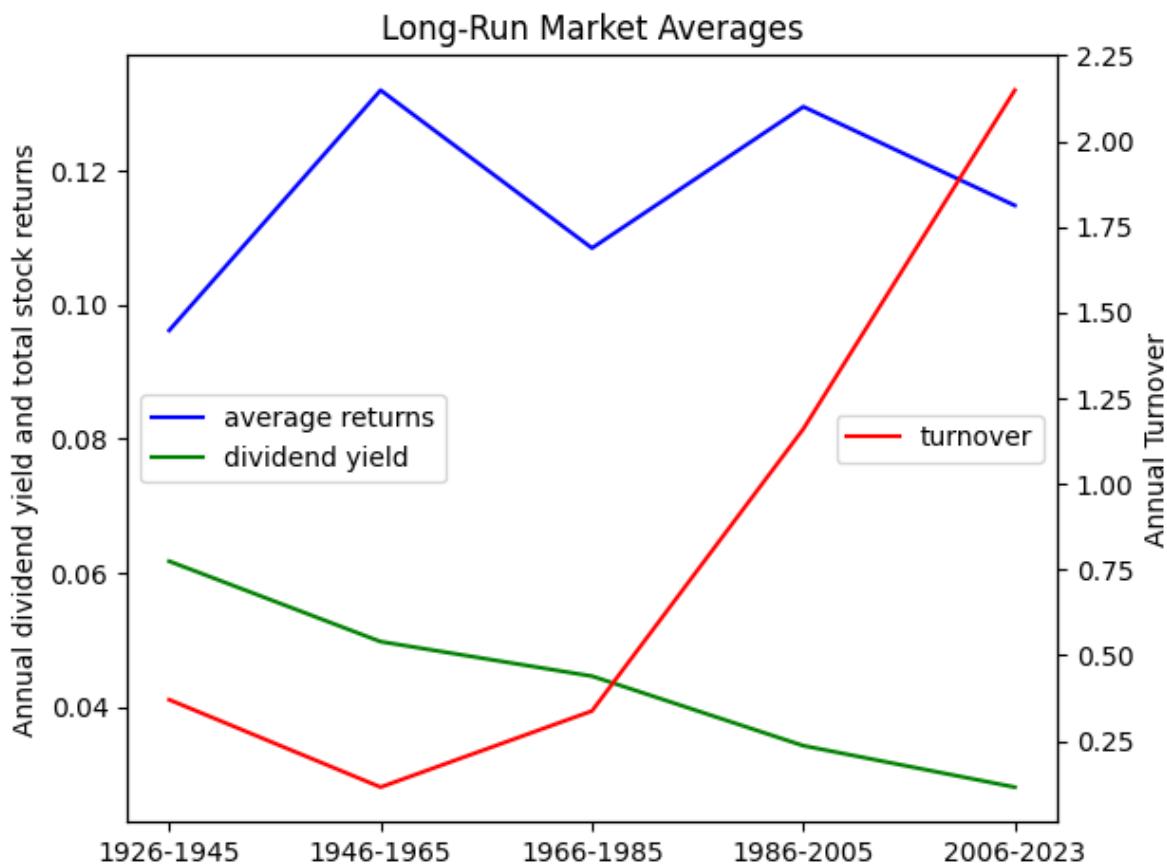
(continued from previous page)

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

0% | 0/5 [00:00<?, ?it/s]

100% [██████] 5/5 [1:07:12<00:00, 806.52s/it]

```
# Plot mean returns, dividend yield and turnover using both y-axes
results = results.iloc[::-1]
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()
```



1.4.2 Moments of individual stock returns

- 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 $\sqrt{12}$
- Means are computed from log returns, and annualized by multiplying by the respective number of periods in a year. We use log returns $\log 1 + R_t$ instead of simple returns R_t to skirt around some of the issues related to translating between arithmetic and geometric averages. It has been shown that:
 - By Jensen's inequality, the arithmetic mean is greater than the geometric mean
 - Under the assumption of log-normality, the arithmetic mean exceeds the geometric means of returns by half the volatility.
- 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. Left skew is also referred to as negative skew.
 - Right or positive skew has the opposite properties.

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

```
# helper function
def get_moments(freq, annualize, years, stride=20):
    """Helper to summarize moments of individual stock returns"""
    out = {}
    years = sorted(years)
    max_era = min(CRSP_DATE, bd.endyr(max(years) + stride)) # end year of last era

    # Use StocksBuffer to cache monthly returns for entire era
    if freq not in ['d', 'w']:
        monthly = CRSPBuffer(stocks=crsp, dataset='monthly', fields=['ret'],
                             beg=bd.endyr(years[0]), end=max_era)
    for era, next_era in zip(years, years[1:] + [max_era]):
        allstocks = DataFrame()
        anystocks = DataFrame()
        univ_year = bd.endyr(era) # universe as of end of previous calendar year

        # Loop over calendar years in era
        endyrs = bd.date_range(bd.endyr(era), bd.endyr(next_era), freq=freq)
        for beg, end in tqdm(bd.date_tuples(endyrs)):

            # 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 calendar year
            if freq == 'd':
                daily = CRSPBuffer(stocks=crsp, dataset='daily', fields=['ret'],
                                    beg=bd.offset(beg, -1), end=bd.endyr(end))
```

(continues on next page)

(continued from previous page)

```

# retrieve returns
if freq == 'd':
    ret = daily.get_ret(beg=beg, end=end).reindex(univ.index)
elif freq == 'w':
    ret = crsp.get_ret(beg=beg, end=end).reindex(univ.index)
else:    # monthly or coarser
    ret = monthly.get_ret(beg=beg, end=end).reindex(univ.index)

# horizontally stack returns
if allstocks.empty:
    allstocks = DataFrame(ret.rename(end))
else:    # every year
    allstocks = allstocks.join(ret.rename(end), how='inner')
anystocks = anystocks.join(ret.rename(end), how='outer')    # any year

anystocks = anystocks.dropna(axis=0, how='all')      # drop empty rows
out[f"{str(era) [:4]}-{int(str(next_era) [:4]) - 1}"] = {
    f"VolAnnualized": np.nanmedian(
        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),
    f"MeanAnnualized": np.nanmedian(
        np.nanmean(np.log(1 + allstocks), axis=1)) * annualize,
    f"Mean (all stocks)": np.nanmedian(
        np.nanmean(np.log(1 + anystocks), axis=1)) * annualize,
    f"Count (all stocks)": len(anystocks),
}
return out

```

```

# Collect results for different intervals and eras
intervals = [('d', 'daily', 252), ('w', 'weekly', 52), ('e', 'monthly', 12),
             (12, 'annual', 1)]      # different returns sampling frequencies
years = [2005, 1985, 1965, 1945, 1925]    # 20-year periods
results = DataFrame()
#results = pd.read_json(outdir / 'moments.json')
#results = results.set_index(results.columns[0])
for order, (freq, interval, annualize) in enumerate(intervals):
    df = DataFrame(get_moments(freq, annualize, years, stride=20)).reset_index()
    df['interval'] = interval

    # set index so that rows can be ordered by moment then freq
    df.index = [f"{moment}{order}" for moment in range(len(df))]
    results = pd.concat((results, df))    # accumulate df to results
    results.reset_index().to_json(outdir / 'moments.json')

```

```
out = results.sort_index().rename(columns={'index':'moment'})
```

```

# Show medians of moments of individual stock returns
out[~out['moment'].str.startswith("Count") &
    ~out['moment'].str.startswith("Mean")].set_index(['interval', 'moment']).round(3)

```

| | | 1925–1944 | 1945–1964 | 1965–1984 | 1985–2004 | 2005–2022 |
|----------|----------------|-----------|-----------|-----------|-----------|-----------|
| interval | moment | | | | | |
| daily | VolAnnualized | 0.543 | 0.267 | 0.337 | 0.392 | 0.437 |
| weekly | VolAnnualized | 0.540 | 0.254 | 0.347 | 0.366 | 0.407 |
| monthly | VolAnnualized | 0.553 | 0.244 | 0.340 | 0.355 | 0.381 |
| annual | VolAnnualized | 0.553 | 0.296 | 0.382 | 0.379 | 0.360 |
| daily | Skewness | 0.909 | 0.315 | 0.528 | 0.339 | 0.395 |
| weekly | Skewness | 0.932 | 0.528 | 0.627 | 0.343 | 0.281 |
| monthly | Skewness | 1.257 | 0.484 | 0.597 | 0.381 | 0.307 |
| annual | Skewness | 0.751 | 0.582 | 0.628 | 0.684 | 0.423 |
| daily | ExcessKurtosis | 14.887 | 5.446 | 4.926 | 8.749 | 13.268 |
| weekly | ExcessKurtosis | 7.535 | 2.645 | 2.916 | 4.384 | 6.589 |
| monthly | ExcessKurtosis | 6.073 | 1.074 | 1.648 | 2.080 | 2.306 |
| annual | ExcessKurtosis | 0.220 | -0.115 | 0.105 | 0.132 | -0.051 |

```
# Show counts of stocks
out[out['moment'].str.startswith("Count")].set_index(['interval', 'moment']).\
    astype(int)
```

| | | 1925–1944 | 1945–1964 | 1965–1984 | 1985–2004 | \ |
|----------|--------------------|-----------|-----------|-----------|-----------|-----------|
| interval | moment | | | | | |
| daily | Count | 305 | 606 | 877 | 1219 | |
| weekly | Count | 305 | 606 | 877 | 1219 | |
| monthly | Count | 305 | 606 | 877 | 1219 | |
| annual | Count | 305 | 606 | 877 | 1219 | |
| daily | Count (all stocks) | 1082 | 2448 | 10089 | 16273 | |
| weekly | Count (all stocks) | 1082 | 2447 | 10084 | 16271 | |
| monthly | Count (all stocks) | 1082 | 2448 | 10088 | 16273 | |
| annual | Count (all stocks) | 1082 | 2448 | 10088 | 16273 | |
| | | | | | | 2005–2022 |
| interval | moment | | | | | |
| daily | Count | 1579 | | | | |
| weekly | Count | 1579 | | | | |
| monthly | Count | 1579 | | | | |
| annual | Count | 1579 | | | | |
| daily | Count (all stocks) | 9050 | | | | |
| weekly | Count (all stocks) | 9042 | | | | |
| monthly | Count (all stocks) | 9049 | | | | |
| annual | Count (all stocks) | 9049 | | | | |

```
# Show medians of mean individual stock returns
out[out['moment'].str.startswith("Mean")].set_index(['interval', 'moment']).round(3)
```

| | | 1925–1944 | 1945–1964 | 1965–1984 | 1985–2004 | \ |
|----------|-------------------|-----------|-----------|-----------|-----------|---|
| interval | moment | | | | | |
| daily | MeanAnnualized | 0.040 | 0.104 | 0.089 | 0.099 | |
| weekly | MeanAnnualized | 0.046 | 0.109 | 0.087 | 0.100 | |
| monthly | MeanAnnualized | 0.047 | 0.108 | 0.088 | 0.099 | |
| annual | MeanAnnualized | 0.047 | 0.108 | 0.088 | 0.099 | |
| daily | Mean (all stocks) | 0.052 | 0.114 | 0.059 | 0.009 | |
| weekly | Mean (all stocks) | 0.061 | 0.118 | 0.057 | 0.010 | |
| monthly | Mean (all stocks) | 0.063 | 0.118 | 0.059 | 0.009 | |
| annual | Mean (all stocks) | 0.061 | 0.116 | 0.056 | 0.008 | |

(continues on next page)

(continued from previous page)

| 2005-2022 | | |
|-----------|-------------------|-------|
| interval | moment | |
| daily | MeanAnnualized | 0.059 |
| weekly | MeanAnnualized | 0.057 |
| monthly | MeanAnnualized | 0.059 |
| annual | MeanAnnualized | 0.059 |
| daily | Mean (all stocks) | 0.006 |
| weekly | Mean (all stocks) | 0.004 |
| monthly | Mean (all stocks) | 0.007 |
| annual | Mean (all stocks) | 0.006 |

1.4.3 Correlations

The linear correlation estimator (also known as Pearson's correlation) measures the linear relationship between two variables. Alternative measures can help capture nonlinear dependence:

- rank correlation (or Spearman's correlation) is the linear correlation estimator applied to the ranks of the observations. It measures monotonic correlation and is less sensitive to outliers.
- Kendall's τ measures the relative frequency of concordant and discordant pairs of two random variables X and Y. Concordant pairs agree about the relative position of X and Y, while pairs that disagree about the order are discordant. This statistic is robust to outliers and works well with skewed or non-normally distributed data.

Computing these measures between the value-weighted and equal-weighted total daily market returns suggests that the high estimated linear correlation may be sensitive to outliers, compared to estimators that are less.

```
# retrieve CRSP value- and equal-weighted total market returns
retd = bench.get_series(['vwretd', 'ewretd'], field='ret').dropna()
retd
```

| permno | vwretd | ewretd |
|----------|-----------|-----------|
| date | | |
| 19260102 | 0.004297 | 0.002941 |
| 19260104 | -0.001357 | 0.001036 |
| 19260105 | -0.004603 | -0.005856 |
| 19260106 | 0.000537 | 0.000888 |
| 19260107 | 0.003907 | 0.004258 |
| ... | ... | ... |
| 20231222 | 0.001708 | 0.003380 |
| 20231226 | 0.004208 | 0.005969 |
| 20231227 | 0.001604 | 0.001316 |
| 20231228 | 0.000415 | 0.001506 |
| 20231229 | -0.002798 | -0.003538 |

[25798 rows x 2 columns]

```
# Compute alternative measures of correlation
DataFrame(dict(kendall=scipy.stats.kendalltau(retd['vwretd'], retd['ewretd'])[0],
               spearman=scipy.stats.spearmanr(retd['vwretd'], retd['ewretd'])[0],
               pearson=scipy.stats.pearsonr(retd['vwretd'], retd['ewretd'])[0]),
               index=['Alternative measures of correlations']).round(4)
```

| | kendall | spearman | pearson |
|--------------------------------------|---------|----------|---------|
| Alternative measures of correlations | 0.8147 | 0.9462 | 0.9628 |

1.5 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.

JEGADEESH-TITMAN ROLLING PORTFOLIOS

The future is simply the past that's waiting to happen - Unknown

Concepts:

- Price momentum
- Univariate spread portfolios
- Overlapping returns, Newey-West correction
- Hypothesis testing, size and power

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.
- FRM Exam Book Part I Quantitative Analysis Chapter 6

Copyright 2024, Terence Lim

MIT License

```
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

At the end of each month t , we construct the sorting variable as the investment universe stocks' past 6 months returns. We determine the 20th and 80th percentile of NYSE-listed stocks, then buy all stocks in top fractile and short all stocks in the lowest fractile. The stocks are cap-weighted within each fractile, while the spread portfolio is the equal-weighted difference of the two sub-portfolios' returns. The spread portfolios' returns over the next six months, expressed on a monthly basis by dividiving by six, are recorded.

To be considered, a stock must be satisfy the usual investment universe criteria at the end of the rabalance month, and have non-missing month-end price six months ago.

```
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 momemtum 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
```

(continues on next page)

(continued from previous page)

```

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)

```

100%|██████████| 1170/1170 [02:12<00:00, 8.84it/s]

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

| | mean | std |
|---------------------|----------|----------|
| Overlapping Returns | 0.004303 | 0.024709 |

2.1.2 Non-overlapping portfolio returns

At the end of each month t , a spread portfolio is constructed in the same way. However, the return recorded is the equal-weighted average of the following month's return on six portfolios constructed between t and $t-5$ (inclusive). After each month, the stock weights on the spread portfolios are adjusted for their stocks' (split-adjusted) price change over the month, i.e. "buy-and-hold" 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)

```

(continues on next page)

(continued from previous page)

```

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
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% | 0/1170 [00:00<?, ?it/s]

100% |██████████| 1170/1170 [03:07<00:00, 6.25it/s]

| | mean | std |
|-------------------------|----------|----------|
| Non-overlapping Returns | 0.004399 | 0.051421 |

Correlations with portfolio lagged returns

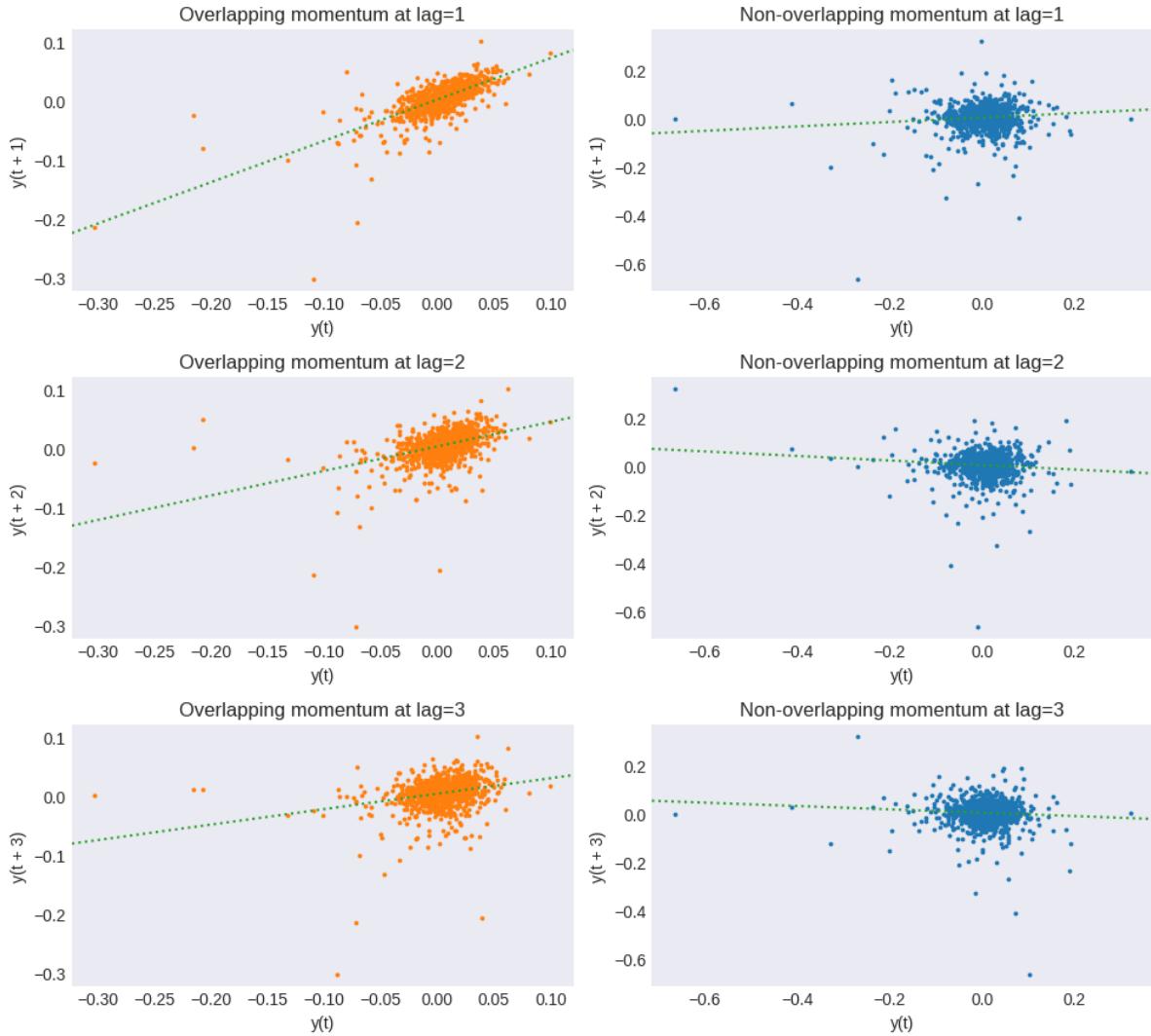
For the overlapping portfolio returns, each month's recorded return is actually a six-month return, hence up to 5/6 of adjacments months' returns could reflect the same month's stock returns. The Jegadeesh-Titman approach avoids such overlap.

```

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 momentum 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 momentum 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()

```



2.2 Hypothesis Testing

A hypothesis makes a precise statement about some population parameters; hypothesis testing can be reduced to asking how likely is the observed data if the hypothesis is true.

- 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

Raw standard errors are understated since the standard assumption of independent observations does not apply. The Newey-West (1987) estimator requires a “maximum lag considered for the control of autocorrelation”: a common choice for L is the fourth root of the number of observations, e.g. Greene (Econometric Analysis, 7th edition, section 20.5.2, p. 960).

The Newey-West correction almost doubles the estimate of the standard error for the overlapping case, but a tiny adjustment for 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

    # 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')
```

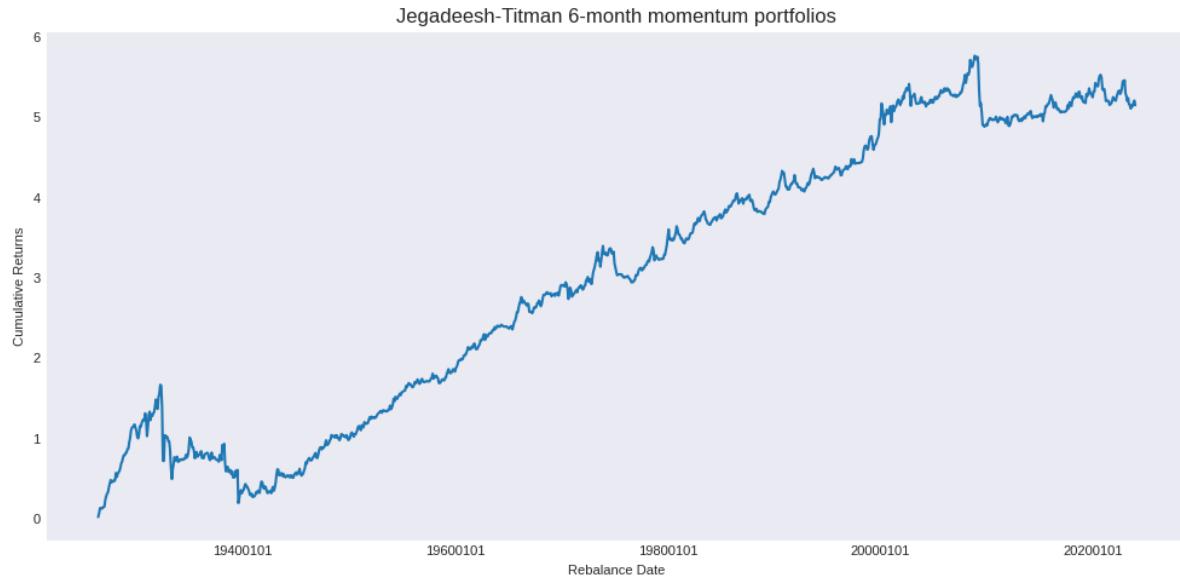
n = 1170 L = 6

| | Overlapping | | Non-overlapping | |
|------------------------|-------------|-----------|-----------------|-----------|
| | uncorrected | NeweyWest | uncorrected | NeweyWest |
| Standard Errors | | | | |
| params | 0.004303 | 0.004303 | 0.004399 | 0.004399 |
| bse | 0.000723 | 0.001298 | 0.001504 | 0.001474 |
| tvalues | 5.953723 | 3.314786 | 2.924986 | 2.984104 |
| pvalues | 0.000000 | 0.000917 | 0.003511 | 0.002844 |

Plot cumulative monthly average returns

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

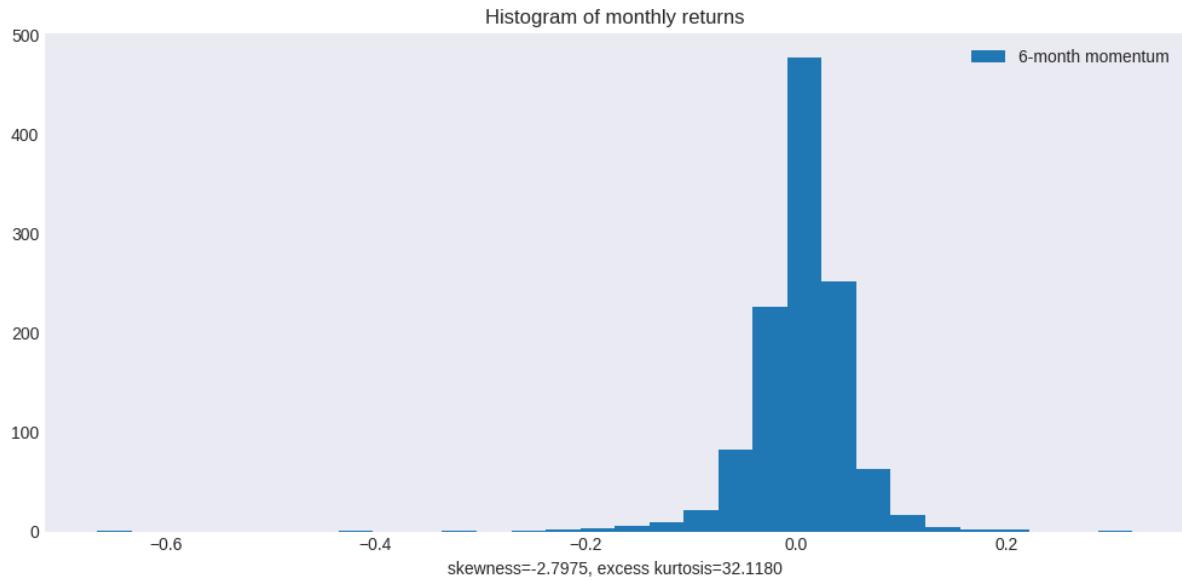
```
fig, ax = plt.subplots(figsize=(10, 5), clear=True)
plot_date(DataFrame(index=rebaldates, data=np.cumsum(jt), columns=['momentum']),
           ax=ax, fontsize=8, rotation=0,
           ylabel1='Cumulative Returns', xlabel='Rebalance Date',
           title='Jegadeesh-Titman 6-month momentum portfolios')
plt.tight_layout()
#plt.savefig(imgdir / 'jegadeesh_titman.jpg')
```



Plot histogram of monthly returns

- 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()
#plt.savefig(imgdir / 'jegadeesh_titman_hist.jpg')
```



2.2.3 Power of Test

A *Type II* error occurs when the alternative is true (i.e., the null is wrong), but the null is not rejected. The probability of a Type II is denoted by β , while the *power* of a test $1 - \beta$, specifies the probability that a false null is rejected.

Unlike the size of the test, the power of a test cannot be set and depends on the sample size, the size of the test and the distance between the true and assumed value of the parameters under the null, e.g. $1 - \beta(\alpha) = \Phi(C_\alpha \frac{\sigma^2}{\sqrt{n}} | \mu_a, \frac{\sigma^2}{\sqrt{n}})$ for a one-sided test $H_a : \hat{\mu} > \mu_0$.

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

| | True Null | False Null |
|-------------|-------------------------------|----------------------------|
| Decision | | |
| Accept Null | correct (1 - alpha) | Type II Error (beta) |
| Reject Null | Type I Error Size: (alpha) | correct 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))
```

(continues on next page)

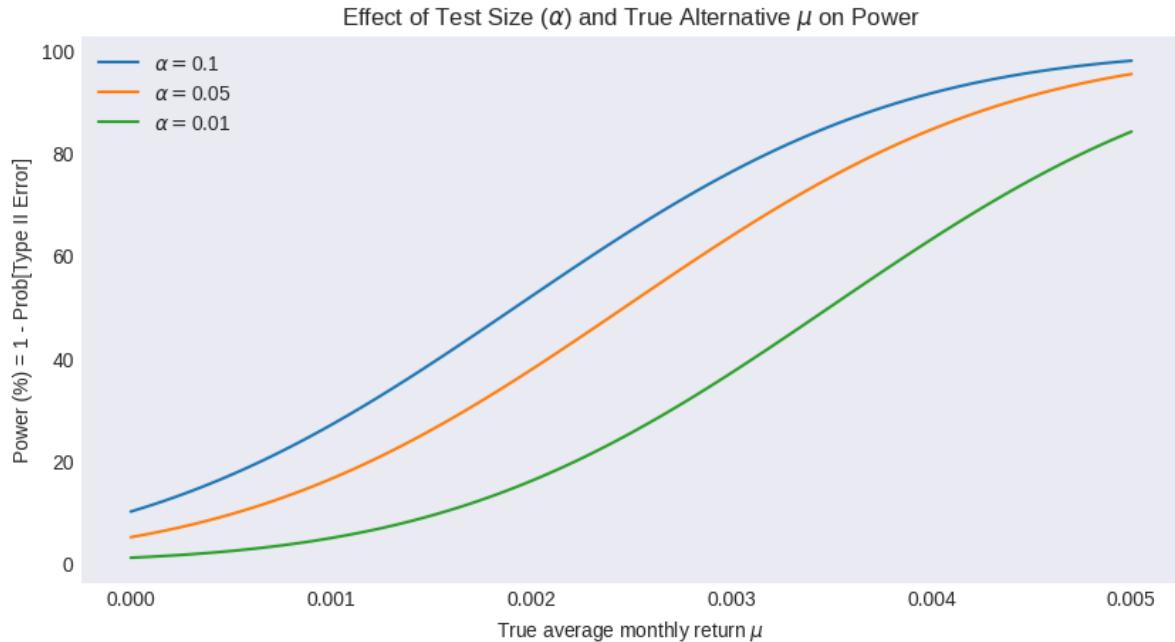
(continued from previous page)

```

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 0x7f1585217e10>



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]

```

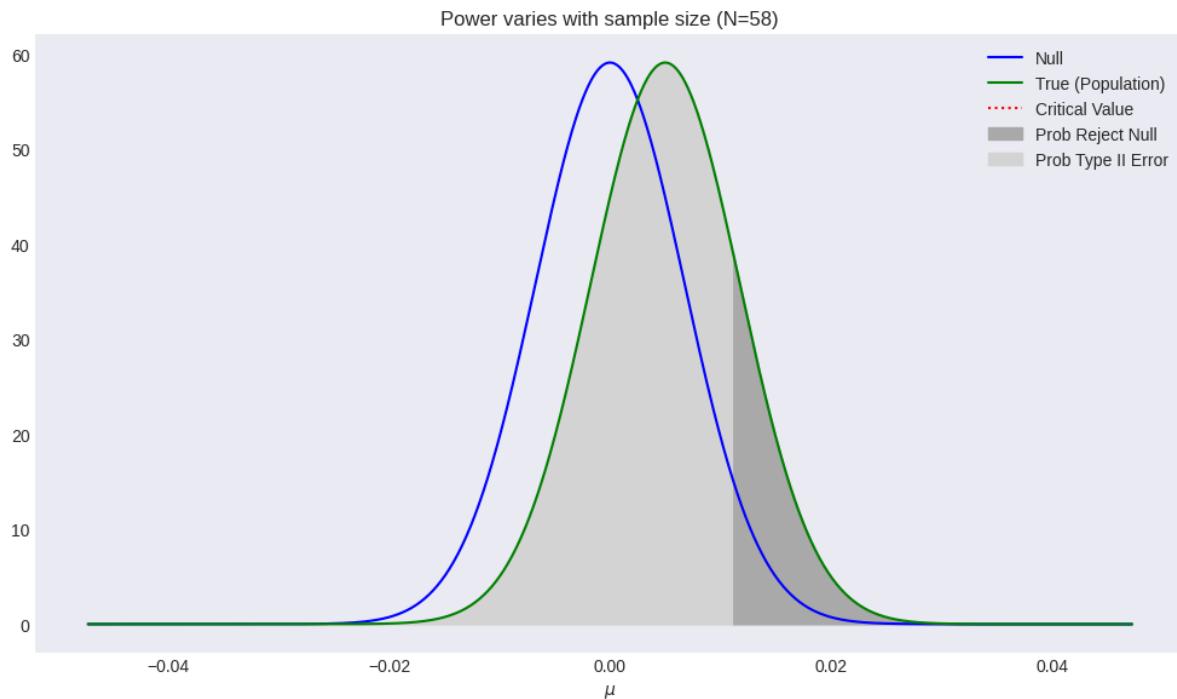
(continues on next page)

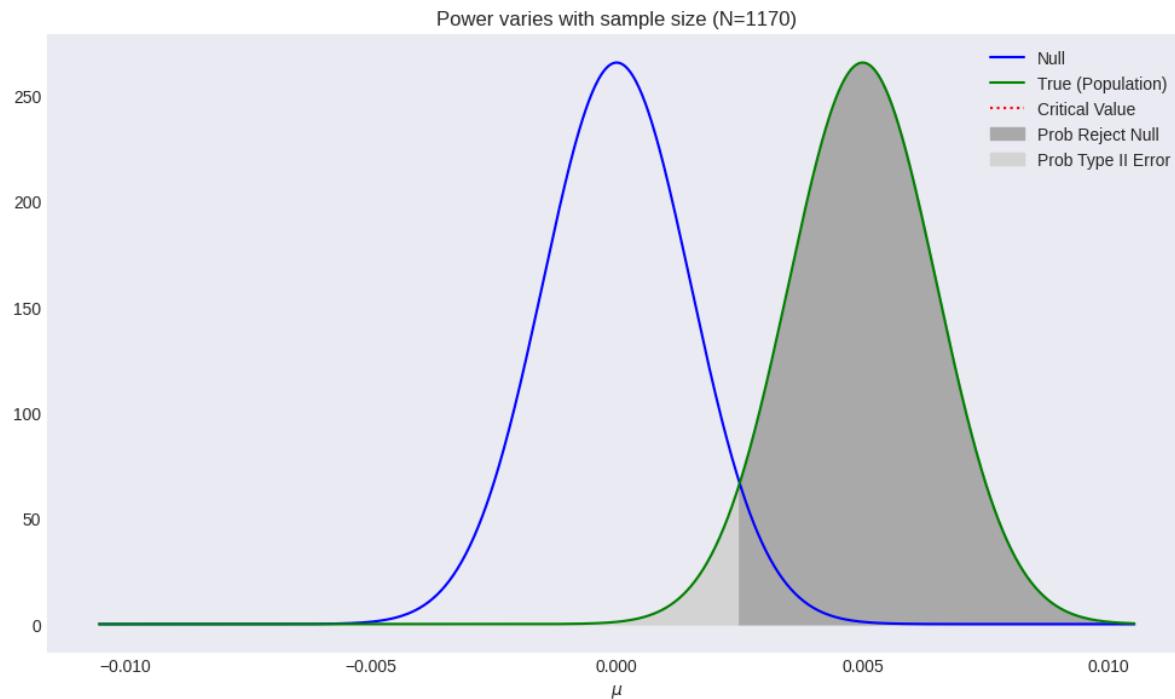
(continued from previous page)

```

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()

```





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

Concepts:

- Value and Fama-French research factors
- Bivariate portfolio sorts
- Linear Regression
- Structural Change Test

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

```
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)
```

(continues on next page)

(continued from previous page)

```
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 Value effect

Calculate Book-to-price ratios

- load items from Compustat Annual
- Construct HML as shareholders equity plus investment tax credits, less preferred stock, divided by December market cap.
- Require 6 month reporting lag and at least two years history in Compustat
- Do not add Deferred Taxes and Investment Tax Credit to BE for fiscal years ending in 1993 or later (FASB 109, which was issued in 1993, improves the accounting 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', 'txditc'],
                      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['txditc'].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['rebalance'] = 0
for datadate in tqdm(sorted(df['datadate'].unique())):
```

(continues on next page)

(continued from previous page)

```

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
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% |██████████| 747/747 [00:02<00:00, 331.67it/s]

223346

3.2 Bivariate sorts

Independent bivariate sorts create portfolios sorted by the two variables: book-to-price ratio and market capitalization size. The portfolios, which are constructed at the end of each June, are the intersections of 2 portfolios formed on size (market equity, ME) and 3 portfolios formed on the ratio of book equity to market equity (BE/ME). The size breakpoint for year t is the median NYSE market equity at the end of June of year t . Stocks are market cap-weighted within each of the 6 portfolios.

HML (High Minus Low) is the equal-weighted average return on the two value portfolios minus the average return on the two growth portfolios. SMB (Small Minus Big) is the equal-weighted average return on the three small portfolios minus the average return on the three big portfolios.

Causal Analysis

This procedure can be appreciated through the perspective of the field of causal analysis, where a submodel of propensity scores is often used to reduce the influence of confounding variables by creating groups with similar probabilities of receiving a treatment. The values of company size variable directly enters into the calculation of book-to-market ratio (as its denominator). Propensity scores are typically calculated using statistical methods such as logistic regression to predict the probability of receiving the treatment. Once propensity scores are obtained, researchers can use them in different ways:

- Stratification: Grouping subjects into strata based on propensity scores and analyzing each stratum separately.
- Matching: Pairing treated and control subjects with similar propensity scores.
- Regression Adjustment: Including the propensity score as a covariate in regression models to adjust for imbalances in baseline characteristics.

Just as propensity score analysis helps to mitigate the effects of confounding variables and improve the validity of causal inference in observational studies, the matching provided by bivariate sorting can more accurately estimate the returns to the Value outcome while controlling for any Small Size effect.

```

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)
```

```
holdings, 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

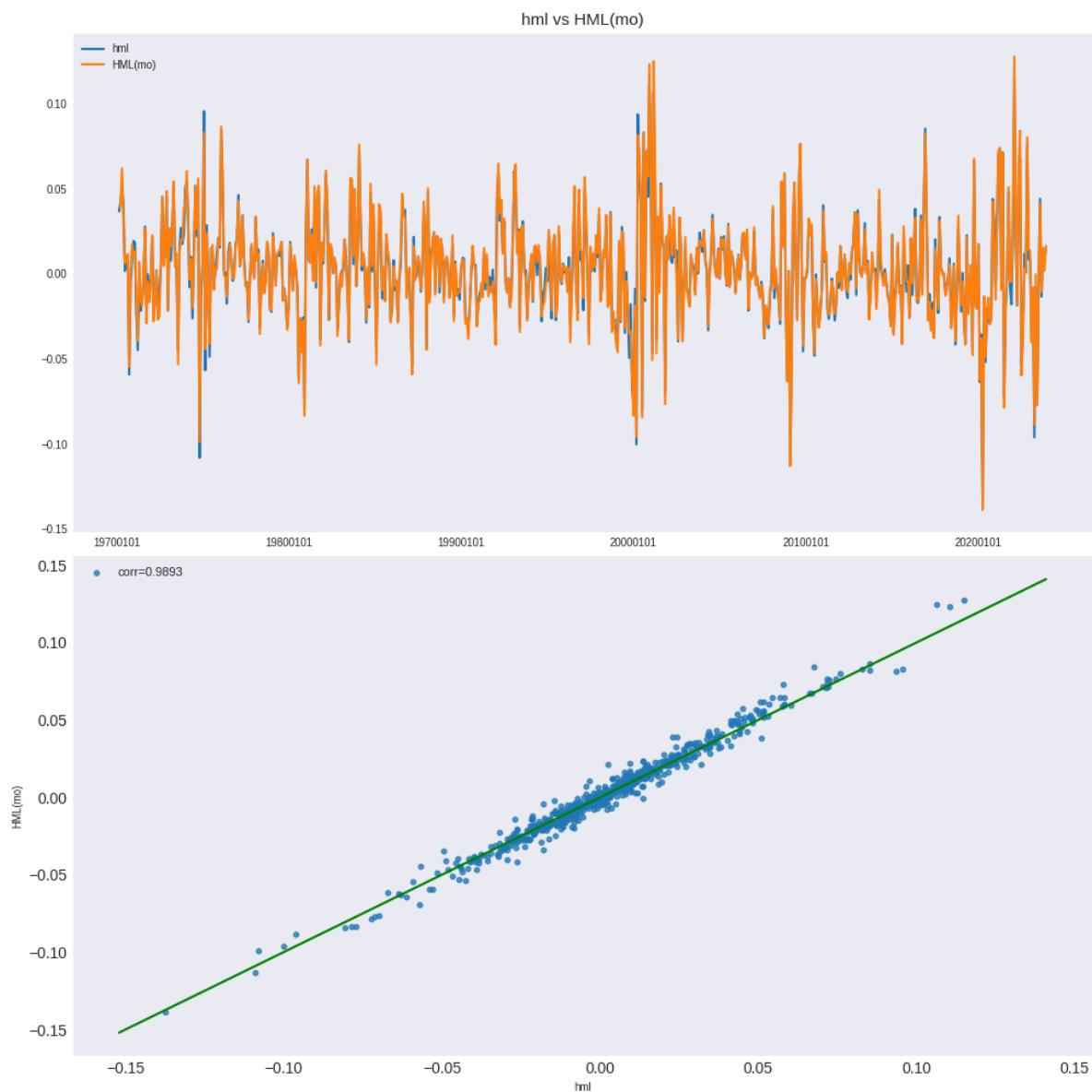
```
def plot_ff(y, label):
    """helper to scatter plot and compare portfolio returns"""
    y = y.rename(columns={'excess': label})
    corr = np.corrcoef(y, rowvar=False)[0, 1]
    fig, (ax1, ax2) = plt.subplots(2, 1, clear=True, figsize=(10, 10))
    plot_date(y, ax=ax1, title=f" vs ".join(y.columns), 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}")
```

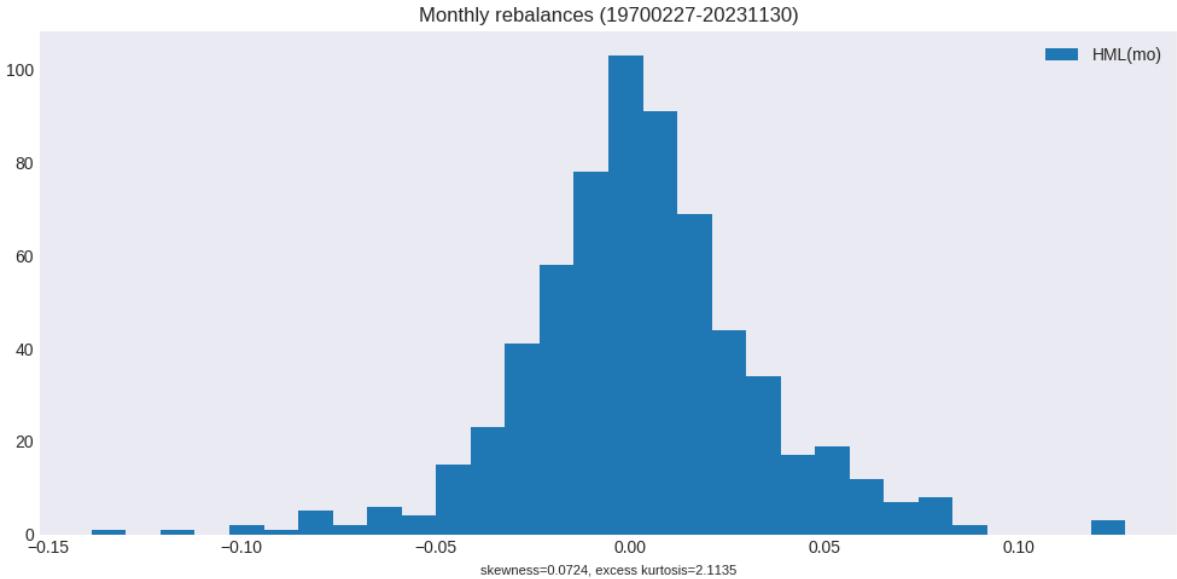
```
def plot_summary(y, label):
    """helper to plot histogram and statistics of portfolio returns"""
    y = y[label]
    kurt = kurtosis(y, bias=True, fisher=True) # excess kurtosis
    skewness = skew(y, bias=True)
    fig, ax = plt.subplots(1, 1, clear=True, figsize=(10, 5))
    ax.hist(y, bins=30)
    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
result = backtest(monthly, holdings, label)
y = backtest.fit([benchname], rebalbeg, LAST_DATE)
plot_ff(y, label)
plot_summary(y, benchname)
```

```
/home/terence/env3.11/lib/python3.11/site-packages/matplotlib/lines.py:1204:_
  FutureWarning: elementwise comparison failed; returning scalar instead, but in_
  the future will perform elementwise comparison
  neq = current != val
```

```
<R-squared of hml vs HML(mo) (19700227 - 20231130): 0.9786
```





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:

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

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

- 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 $\hat{s} \equiv RSE = \sqrt{RSS/(n-2)}$.

The estimators of the coefficients are normally distributed 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.

- 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.

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:

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

```
# 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: 19620731-20240229
OLS Regression Results
=====
Dep. Variable: Q("HML(mo)") R-squared: 0.042
Model: OLS Adj. R-squared: 0.041
Method: Least Squares F-statistic: 8.815
Date: Thu, 04 Apr 2024 Prob (F-statistic): 0.00308
Time: 19:30:34 Log-Likelihood: 1566.5
No. Observations: 740 AIC: -3129.
Df Residuals: 738 BIC: -3120.
Df Model: 1
Covariance Type: HAC
=====
      coef    std err      z   P>|z|    [0.025    0.975]
-----
Intercept  0.0037    0.001   2.628    0.009    0.001    0.007
Q("Mkt-RF(mo)") -0.1360   0.046  -2.969    0.003   -0.226   -0.046
=====
Omnibus: 54.654 Durbin-Watson: 1.653
Prob(Omnibus): 0.000 Jarque-Bera (JB): 249.432
Skew: -0.046 Prob(JB): 6.86e-55
Kurtosis: 5.843 Cond. No. 22.3
=====
Notes:
[1] Standard Errors are heteroscedasticity and autocorrelation robust (HAC) using
    6 lags and without small sample correction
```

3.4 Structural change test

The Chow F-test is normally used in the analysis of time series to test the presence of a structural change. We fit separate models before and after a posited breakpoint – in this case that the mean returns to HML are different before and after its 1993 publication. The null hypothesis of the Chow test is that estimated coefficients of the two submodels are equal, which would be equal a single restricted model estimated from the full data sample.

$\text{chow} = \frac{(RSS - (RSS_1 + RSS_2))/K}{(RSS_1 + RSS_2)/(N - 2K)}$ follows a $F(K, N - 2K)$ distribution

```
# Run restricted and unregression models
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: Thu, 04 Apr 2024 Prob (F-statistic): 1.59e-12
Time: 19:30:34 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: 29.359 Durbin-Watson: 1.600
Prob(Omnibus): 0.000 Jarque-Bera (JB): 55.200
Skew: 0.462 Prob(JB): 1.03e-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 (19931231-20240229):
OLS Regression Results
=====
```

```
Dep. Variable: Q("HML(mo)") R-squared: 0.008
Model: OLS Adj. R-squared: 0.005
```

(continues on next page)

(continued from previous page)

```

Method: Least Squares F-statistic: 2.916
Date: Thu, 04 Apr 2024 Prob (F-statistic): 0.0886
Time: 19:30:34 Log-Likelihood: 717.13
No. Observations: 362 AIC: -1430.
Df Residuals: 360 BIC: -1422.
Df Model: 1
Covariance Type: nonrobust
=====

            coef  std err      t  P>|t|  [0.025  0.975]
-----
Intercept  0.0016  0.002   0.892  0.373  -0.002  0.005
Q("Mkt-RF (mo)") -0.0665  0.039  -1.708  0.089  -0.143  0.010
=====

Omnibus: 24.683 Durbin-Watson: 1.696
Prob(Omnibus): 0.000 Jarque-Bera (JB): 87.223
Skew: 0.034 Prob(JB): 1.15e-19
Kurtosis: 5.404 Cond. No. 22.2
=====

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: 0.042
Model: OLS Adj. R-squared: 0.041
Method: Least Squares F-statistic: 32.43
Date: Thu, 04 Apr 2024 Prob (F-statistic): 1.79e-08
Time: 19:30:34 Log-Likelihood: 1566.5
No. Observations: 740 AIC: -3129.
Df Residuals: 738 BIC: -3120.
Df Model: 1
Covariance Type: nonrobust
=====

            coef  std err      t  P>|t|  [0.025  0.975]
-----
Intercept  0.0037  0.001   3.450  0.001  0.002  0.006
Q("Mkt-RF (mo)") -0.1360  0.024  -5.694  0.000  -0.183  -0.089
=====

Omnibus: 54.654 Durbin-Watson: 1.653
Prob(Omnibus): 0.000 Jarque-Bera (JB): 249.432
Skew: -0.046 Prob(JB): 6.86e-55
Kurtosis: 5.843 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)

```

(continues on next page)

(continued from previous page)

```
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
```

```
(5.195416274472406, 740, 2)
```

p-value of Chow test statistic

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

```
0.0057469646515541095
```

5% critical value to reject null

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

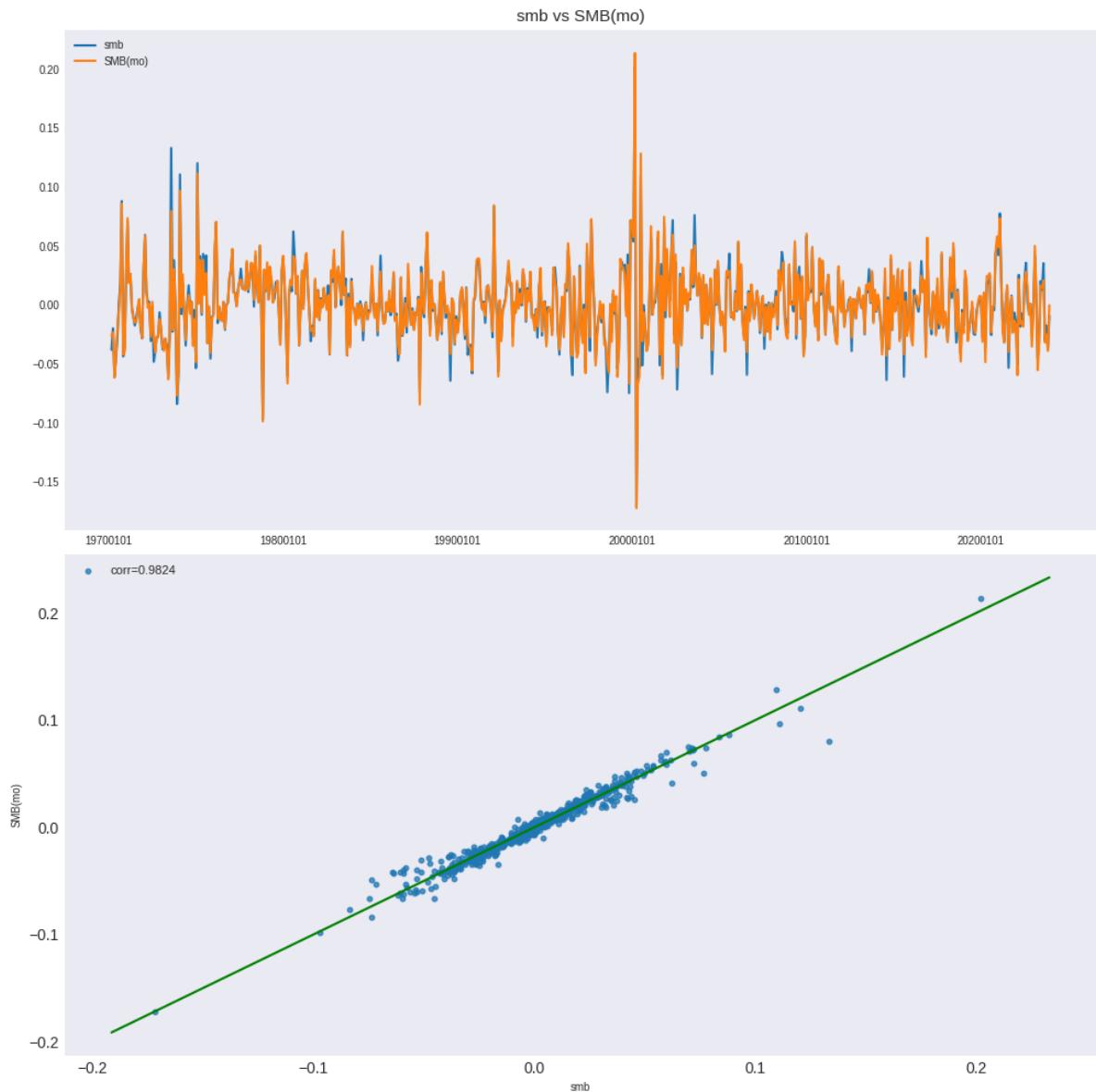
```
3.007958922728564
```

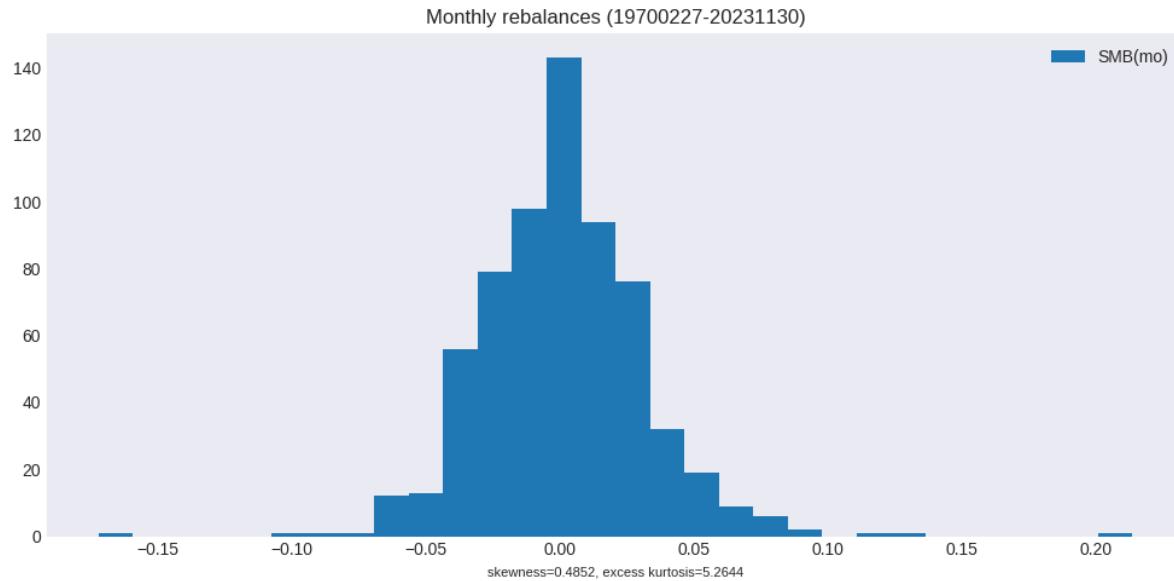
3.5 Small size effect

```
# 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)
```

```
/home/terence/env3.11/lib/python3.11/site-packages/matplotlib/lines.py:1204:_
  FutureWarning: elementwise comparison failed; returning scalar instead, but in_
  the future will perform elementwise comparison
    neq = current != val
```

```
<R-squared of smb vs SMB(mo) (19700227 - 20231130): 0.9652
```





```
# 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: 19620731-20240229

OLS Regression Results

| Dep. Variable: | Q("SMB(mo)") | R-squared: | 0.098 | | | |
|-------------------|------------------|---------------------|-----------|-------|--------|--------|
| Model: | OLS | Adj. R-squared: | 0.096 | | | |
| Method: | Least Squares | F-statistic: | 27.92 | | | |
| Date: | Thu, 04 Apr 2024 | Prob (F-statistic): | 2.05e-12 | | | |
| Time: | 19:31:42 | Log-Likelihood: | 1573.6 | | | |
| No. Observations: | 740 | AIC: | -3141. | | | |
| Df Residuals: | 737 | BIC: | -3127. | | | |
| Df Model: | 2 | | | | | |
| Covariance Type: | HAC | | | | | |
| | coef | std err | z | P> z | [0.025 | 0.975] |
| Intercept | 0.0008 | 0.001 | 0.701 | 0.483 | -0.001 | 0.003 |
| Q("HML(mo)") | -0.1032 | 0.088 | -1.168 | 0.243 | -0.277 | 0.070 |
| Q("Mkt-RF(mo)") | 0.1874 | 0.029 | 6.509 | 0.000 | 0.131 | 0.244 |
| Omnibus: | 103.616 | Durbin-Watson: | 2.039 | | | |
| Prob(Omnibus): | 0.000 | Jarque-Bera (JB): | 718.621 | | | |
| Skew: | 0.391 | Prob(JB): | 8.98e-157 | | | |
| Kurtosis: | 7.764 | Cond. No. | 34.9 | | | |

Notes:

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

3.6 Price momentum effect

```
# Signal is stocks' total return from 12 months ago, skipping most recent month
label, benchname, past, leverage = 'mom', 'Mom(mo)', (2,12), 1
rebalbeg, rebalend = 19261201, LAST_DATE
```

```
df = DataFrame()      # collect each month's momentum signal values
for rebaldate in tqdm(bd.date_range(rebalbeg, rebalend, 'endmo')):
    beg = bd.endmo(rebaldate, -past[1])  # require price at this date
    start = bd.offset(beg, 1)            # start date, inclusive, of signal
    end = bd.endmo(rebaldate, 1-past[0]) # end date of signal
    p = [monthly.get_universe(rebaldate), # retrieve prices and construct signal
          monthly.get_ret(start, end).rename(label),
          monthly.get_section(fields=['prc'], date=beg)['prc'].rename('beg'),
          monthly.get_section(fields=['prc'], date=end)['prc'].rename('end')]
    q = pd.concat(p, axis=1, join='inner').reset_index().dropna()
    q['rebaldate'] = rebaldate
    df = pd.concat([df, q[['permno', 'rebaldate', label]]], axis=0)
signals.write(df, label, overwrite=True)
```

100% |██████████| 1164/1164 [01:42<00:00, 11.38it/s]

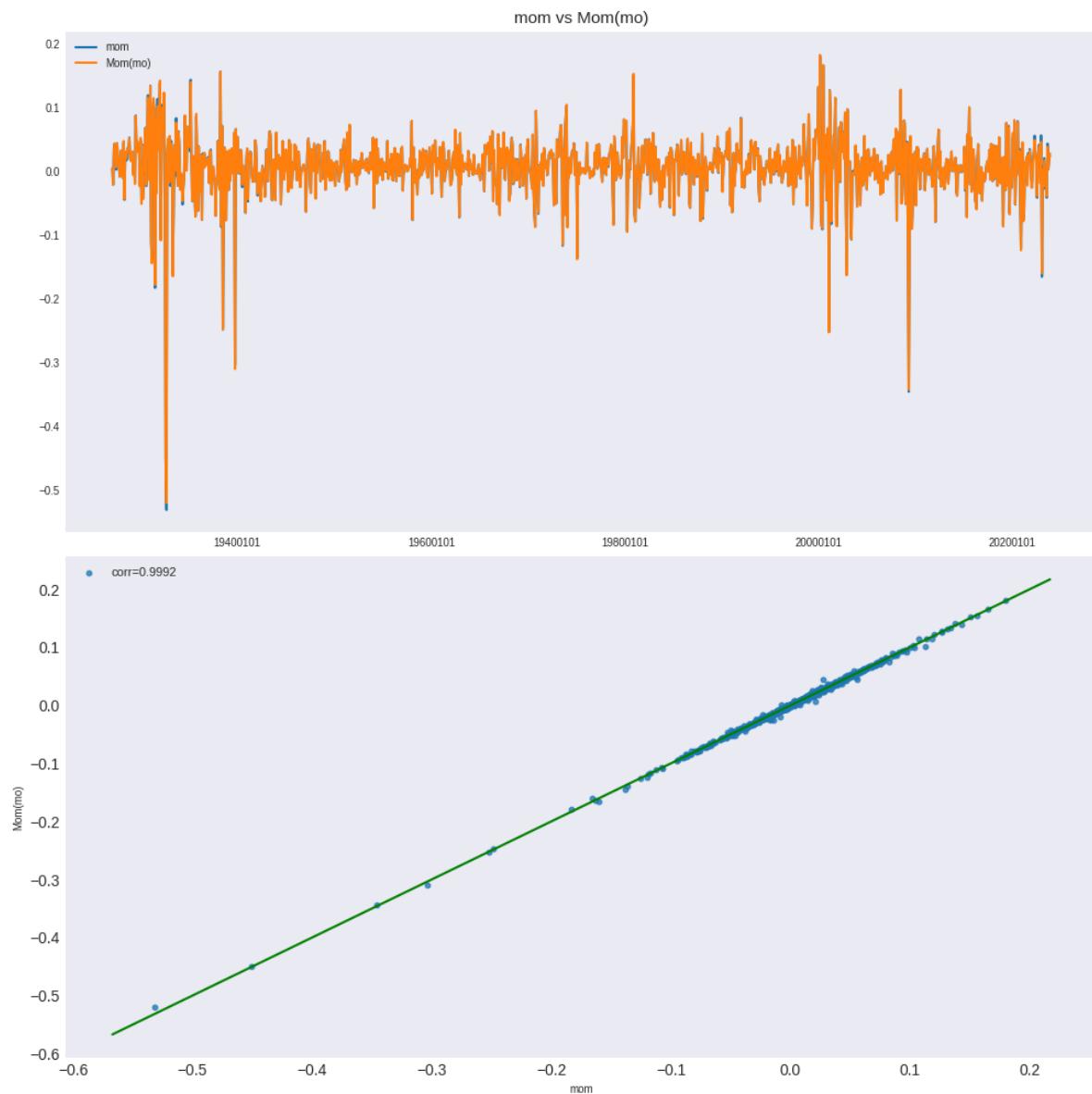
3406671

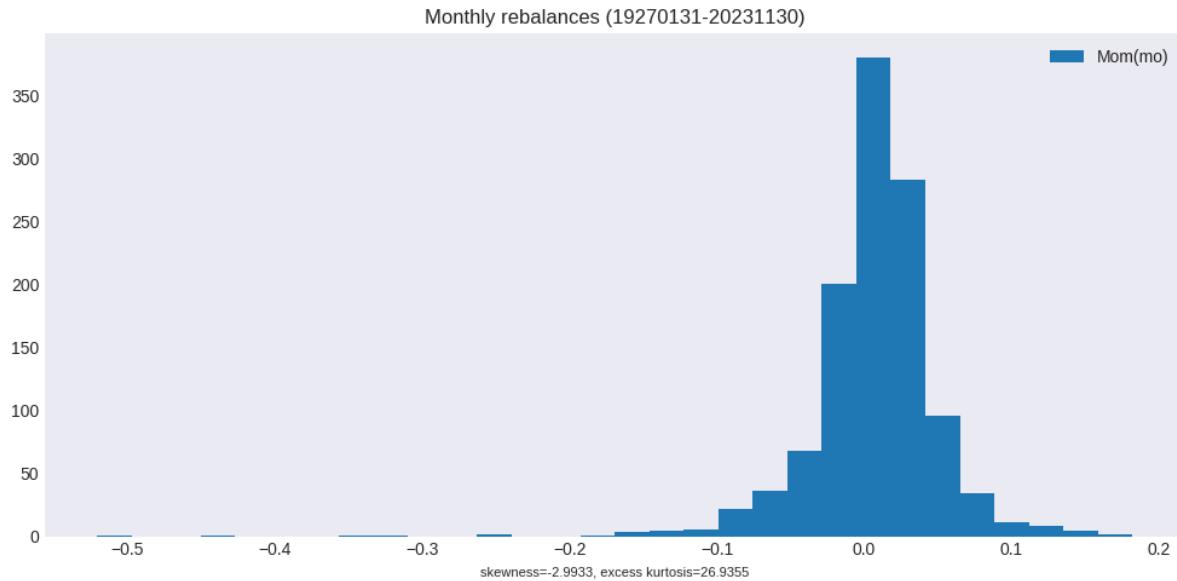
```
## Construct MOM bivariate portfolio sorts
holdings, smb = bivariate_sorts(stocks=monthly,
                                 label=label,
                                 signals=signals,
                                 rebalbeg=rebalbeg,
                                 rebalend=rebalend,
                                 window=0,
                                 months=[],
                                 leverage=leverage)
result = backtest(monthly, holdings, label)
y = backtest.fit([benchname])
```

```
# plot histogram and comparison of returns
plot_ff(y, label)
plot_summary(y, benchname)
```

```
/home/terence/env3.11/lib/python3.11/site-packages/matplotlib/lines.py:1204:_
  FutureWarning: elementwise comparison failed; returning scalar instead, but in_
  the future will perform elementwise comparison
  neq = current != val
```

<R-squared of mom vs Mom(mo) (19270131 - 20231130): 0.9985





```
# Linear regression on 3-factor model
x = [benchmark, "HML(mo)", "SMB(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=19261201, 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: 19261231-20240229

OLS Regression Results

| Dep. Variable: | Q("Mom(mo)") | R-squared: | 0.234 |
|-------------------|------------------|---------------------|----------|
| Model: | OLS | Adj. R-squared: | 0.232 |
| Method: | Least Squares | F-statistic: | 7.262 |
| Date: | Thu, 04 Apr 2024 | Prob (F-statistic): | 7.92e-05 |
| Time: | 19:38:33 | Log-Likelihood: | 2065.8 |
| No. Observations: | 1166 | AIC: | -4124. |
| Df Residuals: | 1162 | BIC: | -4103. |
| Df Model: | 3 | | |
| Covariance Type: | HAC | | |

| | coef | std err | z | P> z | [0.025 | 0.975] |
|-----------------|---------|---------|--------|-------|--------|--------|
| Intercept | 0.0095 | 0.001 | 8.238 | 0.000 | 0.007 | 0.012 |
| Q("HML(mo)") | -0.4526 | 0.121 | -3.725 | 0.000 | -0.691 | -0.214 |
| Q("SMB(mo)") | -0.0545 | 0.087 | -0.625 | 0.532 | -0.225 | 0.116 |
| Q("Mkt-RF(mo)") | -0.2238 | 0.061 | -3.657 | 0.000 | -0.344 | -0.104 |

| Omnibus: | 401.885 | Durbin-Watson: | 1.955 |
|----------------|---------|-------------------|----------|
| Prob(Omnibus): | 0.000 | Jarque-Bera (JB): | 3413.064 |
| Skew: | -1.347 | Prob(JB): | 0.00 |
| Kurtosis: | 10.937 | Cond. No. | 34.0 |

Notes:

[1] Standard Errors are heteroscedasticity and autocorrelation robust (HAC) using

(continues on next page)

(continued from previous page)

↳ 6 lags and without small sample correction

3.7 Price reversal effect

```
# Signal value is recent month's stock returns, sign flipped
label, benchname, past, leverage = 'strev', 'ST_Rev(mo)', (1,1), -1
rebalbeg, rebalend = 19260101, LAST_DATE    #rebalbeg = 20100101
```

```
# loop over each rebalance date to construct and collect signals values
df = DataFrame()
for rebaldate in tqdm(bd.date_range(rebalbeg, rebalend, 'endmo')):
    beg = bd.endmo(rebaldate, -past[1])    # beg price date of signal
    end = bd.endmo(rebaldate, 1-past[0])    # end price date of signal

    # Retrieve universe, require have prices at beg and end dates,
    # and construct signal as returns compounded between start and end dates
    p = [monthly.get_universe(rebaldate),
         monthly.get_section(fields=['prc'], date=beg) ['prc'].rename('beg'),
         monthly.get_section(fields=['prc'], date=end) ['prc'].rename('end'),
         monthly.get_ret(bd.offset(beg, 1), end).rename(label)]
    q = pd.concat(p, axis=1, join='inner').reset_index().dropna()
    q['rebaldate'] = rebaldate
    df = pd.concat((df, q[['permno','rebaldate', label]]), axis=0)

# Save signals values
signals.write(df, label, overwrite=True)
```

100% |██████████| 1175/1175 [01:18<00:00, 14.97it/s]

3706656

```
# Construct bivariate portfolios sort
holdings, smb = bivariate_sorts(stocks=monthly,
                                 label=label,
                                 signals=signals,
                                 rebalbeg=rebalbeg,
                                 rebalend=rebalend,
                                 window=0,
                                 months=[],
                                 leverage=leverage)
result = backtest(monthly, holdings, label)
y = backtest.fit([benchname])
```

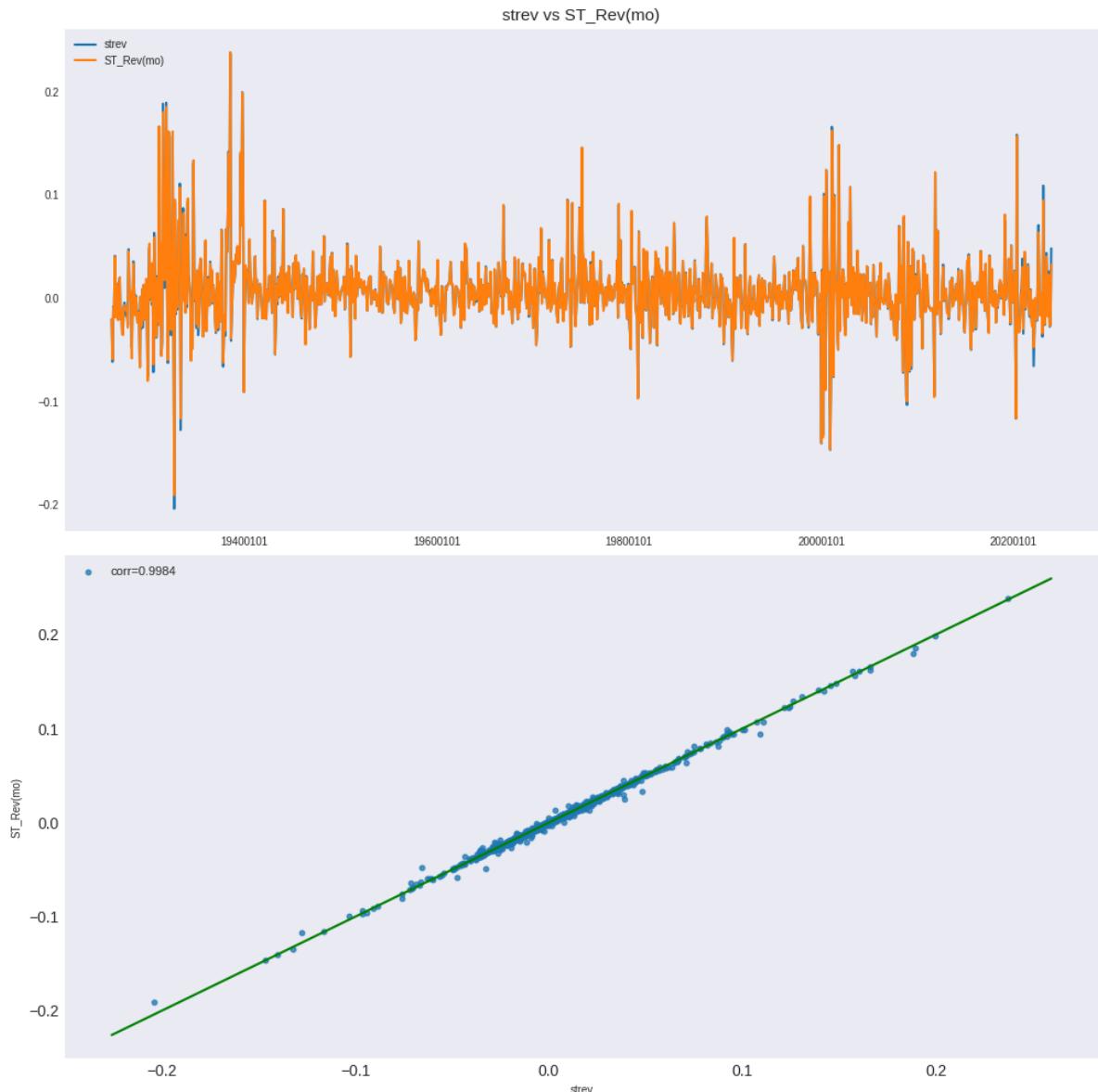
```
# plot histogram and comparison of returns
plot_ff(y, label)
plot_summary(y, benchname)
```

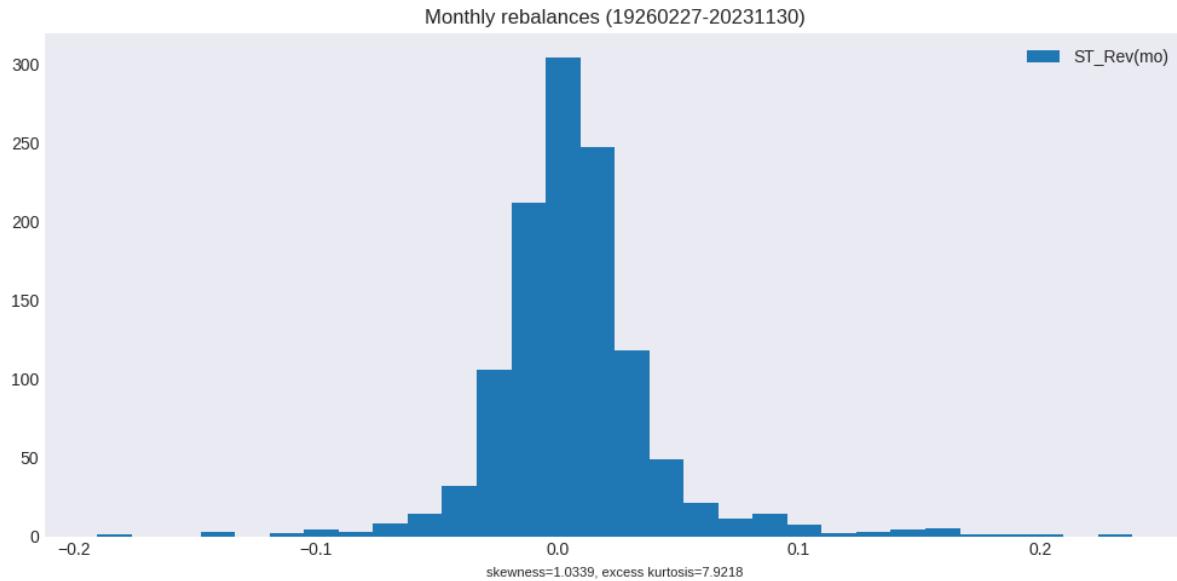
```
/home/terence/env3.11/lib/python3.11/site-packages/matplotlib/lines.py:1204:_
FutureWarning: elementwise comparison failed; returning scalar instead, but in_
(continues on next page)
```

(continued from previous page)

```
the future will perform elementwise comparison
neq = current != val
```

```
<R-squared of strev vs ST_Rev(mo) (19260227 - 20231130): 0.9968
```





```
# Linear regression on 3-factor model
x = [benchmark, "HML(mo)", "SMB(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=19260101, 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: 19260227-20240229

OLS Regression Results

| Dep. Variable: | Q("ST_Rev(mo)") | R-squared: | 0.061 | | | |
|-------------------|------------------|---------------------|----------|-------|--------|--------|
| Model: | OLS | Adj. R-squared: | 0.059 | | | |
| Method: | Least Squares | F-statistic: | 4.973 | | | |
| Date: | Thu, 04 Apr 2024 | Prob (F-statistic): | 0.00197 | | | |
| Time: | 19:45:05 | Log-Likelihood: | 2323.3 | | | |
| No. Observations: | 1172 | AIC: | -4639. | | | |
| Df Residuals: | 1168 | BIC: | -4618. | | | |
| Df Model: | 3 | | | | | |
| Covariance Type: | HAC | | | | | |
| | coef | std err | z | P> z | [0.025 | 0.975] |
| Intercept | 0.0058 | 0.001 | 5.404 | 0.000 | 0.004 | 0.008 |
| Q("HML(mo)") | -0.0145 | 0.079 | -0.185 | 0.853 | -0.169 | 0.140 |
| Q("SMB(mo)") | 0.1204 | 0.070 | 1.713 | 0.087 | -0.017 | 0.258 |
| Q("Mkt-RF(mo)") | 0.1234 | 0.040 | 3.062 | 0.002 | 0.044 | 0.202 |
| Omnibus: | 253.147 | Durbin-Watson: | 1.932 | | | |
| Prob(Omnibus): | 0.000 | Jarque-Bera (JB): | 4129.550 | | | |
| Skew: | 0.523 | Prob(JB): | 0.00 | | | |
| Kurtosis: | 12.136 | Cond. No. | 34.1 | | | |

Notes:

[1] Standard Errors are heteroscedasticity and autocorrelation robust (HAC) using

(continues on next page)

(continued from previous page)

↳ 6 lags and without small sample correction

FAMA-MACBETH CROSS-SECTIONAL REGRESSIONS

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

Concepts

- Portfolio Optimization, CAPM, Black-Litterman
- Tests of the CAPM, cross-sectional regression
- Polynomial regression, feature transformations, kernel trick
- Kernel regression

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.
- FRM Part I Exam Book Foundations of Risk Management Ch. 5.

```
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
```

(continues on next page)

(continued from previous page)

```
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

Given two investments with the same expected return (as measured by the mean of the returns), Markowitz demonstrated that a risk-averse investor will select the one with the lowest risk (as measured by the variance). Markowitz's theory relies on several assumption including the absence of market frictions (such as no taxes or transaction costs, perfect market competition), as well as normally distributed returns.

The assumption of normally distributed returns implies that a “rational investor” should evaluate potential portfolio allocations based entirely upon the associated means and variances of the return distributions. With all else being equal, investors prefer a higher mean return and a lower variance, and seek to reduce the variance of their portfolio returns by diversifying their investments with assets that whose prices do not move in lockstep with one another.

A major implementation issue with this framework comes when estimating the parameters required to apply the model (i.e., the mean and the variance of returns, along with correlations between each asset in the portfolio). These parameters are typically estimated using historical data over a certain period of time, but the resulting allocation can differ greatly depending on which historical or forecast data are used. There are several methodologies that have been used to deal with the problem of the uncertainty about these parameters, such as robust portfolio optimization and the Black-Litterman model.

```
# 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']\n    →')}),\n           name='Mkt')], axis=1)
```

| | SMALL LoBM | BIG BM2 | SMALL HiBM | BIG LoBM | SMALL BM2 | BIG HiBM | \ |
|------------|------------|----------|------------|----------|-----------|----------|---|
| mean | 0.007114 | 0.009685 | 0.011547 | 0.006786 | 0.006918 | 0.009200 | |
| volatility | 0.074835 | 0.069750 | 0.081174 | 0.053068 | 0.056324 | 0.071366 | |
| | Mkt | | | | | | |
| mean | 0.007077 | | | | | | |
| volatility | 0.053851 | | | | | | |

4.1.1 Global minimum variance portfolio

Given estimates of assets' variances and their correlations with each other, the Global Minimum Variance (GMV) portfolio identifies the allocation that achieves the lowest risk (and ignores mean returns). It is a convex (quadratic) optimization problem, with constraint that the portfolio weights must sum to 1. The Python package conveniently solves such problems numerically: $\min_w w^T \Sigma w$, such that $w^T 1 = 1$.

```
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 in closed form solution by differentiating the (convex) objective function and setting the FOC to zero: $GMV = \frac{\Sigma^{-1}1}{1^T \Sigma^{-1}1}$

```
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)\n           .set_index(assets.index).T
```

| | SMALL LoBM | BIG BM2 | SMALL HiBM | BIG LoBM | SMALL BM2 | BIG HiBM |
|-----------|------------|----------|------------|----------|-----------|----------|
| numerical | -0.486252 | 0.674582 | -0.334163 | 0.787655 | 0.831288 | -0.47311 |
| formula | -0.486252 | 0.674582 | -0.334163 | 0.787655 | 0.831288 | -0.47311 |

4.1.2 Efficient frontier

Each point on the efficient frontier curve represents the portfolio of risky assets that is expected to offer the highest return for the given level of risk as measured by the standard deviation of returns σ . A line is drawn from the risk-free rate becomes tangent to the efficient frontier at the point called the **tangency portfolio**. Portfolios that lie on this line, called the **capital market line**, given by $E(R_p) = r_f + \frac{E[R_M] - r_f}{\sigma_M} \sigma_p$, dominate all portfolios on the efficient frontier. In other words, a tangency portfolio is a portfolio that lies at the point where the efficient frontier is tangent to the highest possible capital market line (CML) in the risk-return space. The implication of the Capital Market Line is that all investors should allocate to the risk-free asset and the tangency market portfolio. This is called the **two fund separation theorem**.

```
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 also be solved with closed form solutions.

- 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 = $\frac{\Sigma^{-1} \mu}{1^T \Sigma^{-1} \mu}$

Furthermore, any portfolio on the efficient frontier is a linear combination of any 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'])})
```

```

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)

```

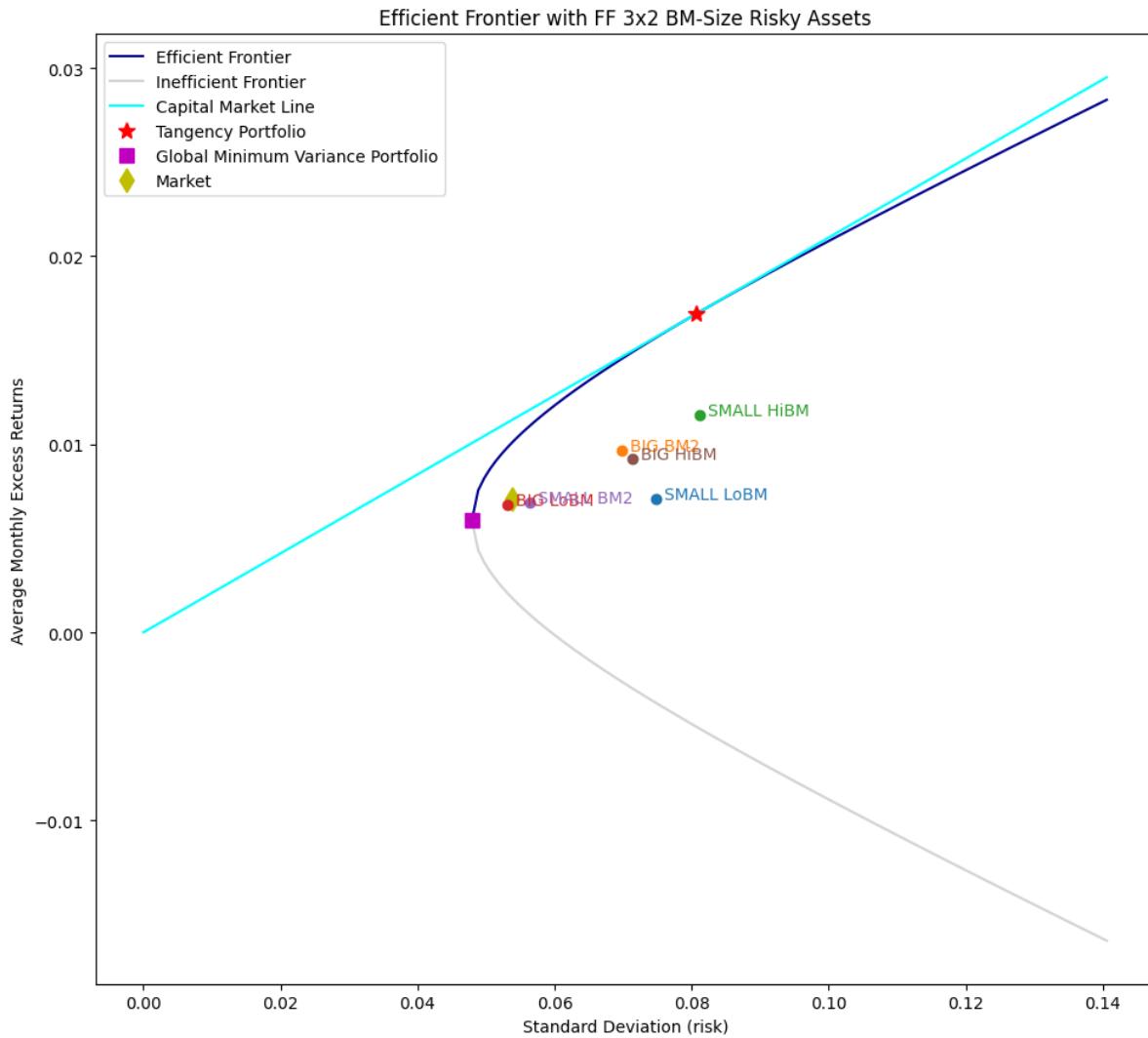
| | tangency portfolio | tangency formula | random efficient portfolio | \ |
|------------|--------------------|------------------|----------------------------|---|
| volatility | 0.080694 | 0.080934 | 0.077362 | |
| mean | 0.016937 | 0.016987 | 0.016228 | |
| | by formula | | | |
| volatility | 0.077362 | | | |
| mean | 0.016228 | | | |

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 derived an equilibrium model showing the relationship between the risk and expected return of a risky asset.

The derivation of CAPM includes several crucial assumptions, some of which are the same as those used by Markowitz. The CAPM model shows that market equilibrium is achieved when all investors hold portfolios consisting of the riskless asset and the market portfolio described earlier. Each investor's portfolio is just a combination of these two, with the proportional allocation between them being a function of the individual investor's risk appetite. Accordingly, the expected return on a risky asset is determined by that asset's relative contribution to the market portfolio's total risk. In this case, the relevant measure of risk is the risk that cannot be diversified away (i.e., non-diversifiable risk or systematic risk). This means that investors should only be compensated for the risk that cannot be eliminated by diversification.

Specifically, the total risk of a risky asset is decomposed into two components – a systematic component proxied by the asset's beta $\beta_i = \frac{\text{cov}(R_i, R_M)}{\text{var}(R_M)}$, and a idiosyncratic component that can be diversified away.

The **security market line** gives the relationship between the expected return for individual assets and risk as proxied by beta $E(R_i) = r_f + \beta_i(E[R_M] - r_f)$.

4.2 Implied alphas

If some portfolio allocation W is known to be solution of some mean-variance objective above, then we can infer the mean returns input if given the covariance matrix. In other words, these **implied alphas**, which are proportional to $w^T \Sigma$, when input as the mean returns vector along with the same covariance matrix, delivers W as the mean-variance optimization solution.

```
# 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 | 0.000737 | 0.001825 | 0.003137 | 0.000283 | |
| Market weights | 0.010605 | 0.018481 | 0.013655 | 0.661900 | |
| CAPM equilibrium returns | 0.007058 | 0.006734 | 0.007456 | 0.005633 | |
| historical mu | 0.007114 | 0.009685 | 0.011547 | 0.006786 | |
| implied/capm | 0.104370 | 0.271056 | 0.420693 | 0.050287 | |
| | SMALL BM2 | BIG HiBM | | | |
| HML weights | 0.000000 | 0.500000 | | | |
| HML implied-alpha | 0.001636 | 0.002976 | | | |
| Market weights | 0.238224 | 0.057136 | | | |
| CAPM equilibrium returns | 0.005778 | 0.006897 | | | |
| historical mu | 0.006918 | 0.009200 | | | |
| implied/capm | 0.283177 | 0.431512 | | | |

```
# 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 | 0.836663 | 0.622195 |

```
# 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 | 0.010605 | 0.018481 | | | 0.013655 | | 0.6619 | | 0.238224 | | |
| mean-variance weights | 0.010605 | 0.018481 | | | 0.013655 | | 0.6619 | | 0.238224 | | |
| formula | 0.010605 | 0.018481 | | | 0.013655 | | 0.6619 | | 0.238224 | | |
| | | | BIG | HiBM | | | | | | | |
| Market weights | | | | 0.057136 | | | | | | | |
| mean-variance weights | | | | 0.057136 | | | | | | | |
| formula | | | | 0.057136 | | | | | | | |

4.2.1 Black-Litterman Model

Mean-variance efficient portfolios are highly sensitive to the inputs. Errors in estimating expected returns are observed to many times more important than errors in estimating variances and covariances. Black and Litterman (1992) suggest that “shrinking” investors’ views with the expected returns implied by the market portfolio can deliver portfolio allocations that are less sensitive to parameter uncertainty.

```
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 | SMALL | BM2 | \ |
|----------------------------|-----------|------|-----------|-----|----------|------|----------|------|-------|-----|---|
| Tangency Portfolio Weights | -2.855105 | | 2.777550 | | 0.979435 | | 2.044196 | | | | |
| Market Weights | 0.010605 | | 0.018481 | | 0.013655 | | 0.661900 | | | | |
| Active Weights | -2.865709 | | 2.759070 | | 0.965780 | | 1.382296 | | | | |
| | | | SMALL | BM2 | BIG | HiBM | | | | | |
| Tangency Portfolio Weights | -1.344998 | | -0.601078 | | | | | | | | |

(continues on next page)

(continued from previous page)

| | | |
|----------------|-----------|-----------|
| Market Weights | 0.238224 | 0.057136 |
| Active Weights | -1.583222 | -0.658214 |

Black-Litterman alphas are computed as:

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

- τ is a non-negative scalar that reflects an overall level of confidence in the active views versus the equilibrium expected returns. It effectively determines the overall weight placed on the active views relative to the equilibrium expected returns
- In Bayesian terms, τ measures the subjective degree of uncertainty as to how precisely the equilibrium returns (the expected return priors) have been estimated

```
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: 0.00253498561210212

| | SMALL LoBM | BIG BM2 | SMALL HiBM | BIG LoBM | \ |
|-------------------------|------------|-----------|------------|----------|---|
| Black-Litterman weights | -0.104873 | 0.129661 | 0.052573 | 0.717601 | |
| Market weights | 0.010605 | 0.018481 | 0.013655 | 0.661900 | |
| Active weights | -0.115477 | 0.111180 | 0.038917 | 0.055701 | |
| | SMALL BM2 | BIG HiBM | | | |
| Black-Litterman weights | 0.174426 | 0.030612 | | | |
| Market weights | 0.238224 | 0.057136 | | | |
| Active weights | -0.063798 | -0.026524 | | | |

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

```
array([0.04029624, 0.04029624, 0.04029624, 0.04029624, 0.04029624,
       0.04029624])
```

It is also easy, with numerical solvers such as cvxpy, to impose constraints such as no short-sales, and to iteratively add more constraints such that optimization solution “look” more reasonable.

```
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: 0.004142093780605273
```

| | SMALL LoBM | BIG BM2 | SMALL HiBM | BIG LoBM | SMALL BM2 | \ |
|---------------------|------------|-----------|------------|----------|-----------|---|
| Constrained weights | 0.000000 | 0.009459 | 0.136044 | 0.751352 | 0.103146 | |
| Market weights | 0.010605 | 0.018481 | 0.013655 | 0.661900 | 0.238224 | |
| Active weights | -0.010605 | -0.009022 | 0.122388 | 0.089452 | -0.135078 | |
| | BIG HiBM | | | | | |
| Constrained weights | -0.000000 | | | | | |
| Market weights | 0.057136 | | | | | |
| Active weights | -0.057136 | | | | | |

```
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): 0.008062613779445192
```

| | SMALL LoBM | BIG BM2 | SMALL HiBM | BIG LoBM | SMALL BM2 | \ |
|---------------------|------------|----------|------------|----------|-----------|---|
| Constrained weights | 0.000000 | 0.038421 | 0.046917 | 0.732865 | 0.181798 | |
| Market weights | 0.010605 | 0.018481 | 0.013655 | 0.661900 | 0.238224 | |
| Active weights | -0.010605 | 0.019940 | 0.033261 | 0.070965 | -0.056426 | |

(continues on next page)

(continued from previous page)

| | BIG HiBM |
|---------------------|-----------|
| Constrained weights | 0.000000 |
| Market weights | 0.057136 |
| Active weights | -0.057136 |

4.3 Cross-sectional regressions

The Fama-MacBeth (1973) method estimates the betas and risk premia for any risk factors that are expected to determine asset prices. The parameters are estimated in two steps:

- First regress each time series of asset returns against proposed risk factor returns to determine each asset's beta exposures.
- Then regress all asset returns for each of T time periods against the previously estimated betas to determine the average risk premium for each factor.

This provides standard errors that are corrected for cross-sectional correlation effects.

4.3.1 Testing the CAPM

Retrieve test asset returns and risk-free rate. In addition to testing whether the “beta” is priced (i.e. that higher beta assets achieve linearly larger risk premiums than lower beta assets), we also test for non-linearity affects by including a beta-squared factor, and (supposedly) non-priced factors such as the residual riskiness of the assets.

```

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'])

/tmpp/ipykernel_2439586/3663017786.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 2024-02

| | _intercept | BETA | BETA2 | RES |
|--------|------------|----------|-----------|----------|
| mean | 0.007915 | 0.000055 | -0.008738 | 0.125885 |
| stderr | 0.001547 | 0.002278 | 0.002575 | 0.061047 |
| tstat | 5.115902 | 0.023976 | -3.393737 | 2.062093 |

Clustered standard errors

Alternative methods of correcting standard errors for time series and cross-sectional correlation in the error term may look into 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: | 2.709 | | |
| Date: | | Sun, 14 Apr 2024 | Prob (F-statistic): | 0.0435 | | |
| Time: | | 14:35:06 | Log-Likelihood: | 25361. | | |
| No. Observations: | | 18200 | AIC: | -5.071e+04 | | |
| Df Residuals: | | 18196 | BIC: | -5.068e+04 | | |
| Df Model: | | 3 | | | | |
| Covariance Type: | | nonrobust | | | | |
| | coef | std err | t | P> t | [0.025 | 0.975] |
| Intercept | 0.0079 | 0.002 | 4.199 | 0.000 | 0.004 | 0.012 |
| BETA | 5.462e-05 | 0.002 | 0.033 | 0.974 | -0.003 | 0.003 |

(continues on next page)

(continued from previous page)

| | | | | | | |
|----------------|----------|-------------------|----------|-------|--------|-------|
| BETA2 | -0.0087 | 0.006 | -1.432 | 0.152 | -0.021 | 0.003 |
| RES | 0.1259 | 0.051 | 2.465 | 0.014 | 0.026 | 0.226 |
| <hr/> | | | | | | |
| Omnibus: | 1538.581 | Durbin-Watson: | 1.765 | | | |
| Prob(Omnibus): | 0.000 | Jarque-Bera (JB): | 9938.436 | | | |
| Skew: | -0.056 | Prob(JB): | 0.00 | | | |
| Kurtosis: | 6.618 | Cond. No. | 173. | | | |
| <hr/> | | | | | | |

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: | 1.709 | | | |
| Date: | Sun, 14 Apr 2024 | Prob (F-statistic): | 0.192 | | | |
| Time: | 14:35:06 | Log-Likelihood: | 25361. | | | |
| No. Observations: | 18200 | AIC: | -5.071e+04 | | | |
| Df Residuals: | 18196 | BIC: | -5.068e+04 | | | |
| Df Model: | 3 | | | | | |
| Covariance Type: | hac-panel | | | | | |
| | coef | std err | t | P> t | [0.025 | 0.975] |
| Intercept | 0.0079 | 0.002 | 4.010 | 0.001 | 0.004 | 0.012 |
| BETA | 5.462e-05 | 0.002 | 0.029 | 0.977 | -0.004 | 0.004 |
| BETA2 | -0.0087 | 0.006 | -1.363 | 0.185 | -0.022 | 0.004 |
| RES | 0.1259 | 0.067 | 1.888 | 0.071 | -0.012 | 0.263 |
| <hr/> | | | | | | |
| Omnibus: | 1538.581 | Durbin-Watson: | 1.765 | | | |
| Prob(Omnibus): | 0.000 | Jarque-Bera (JB): | 9938.436 | | | |
| Skew: | -0.056 | Prob(JB): | 0.00 | | | |
| Kurtosis: | 6.618 | Cond. No. | 173. | | | |
| <hr/> | | | | | | |

Notes:

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

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 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**.

The slight concavity of the fitted curve is consistent with the negative coefficient on observed for squared-beta factor in the Fama-MacBeth regression. The lack of monotonicity would be consistent with an insignificant coefficient for the beta factor.

```
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]:
```

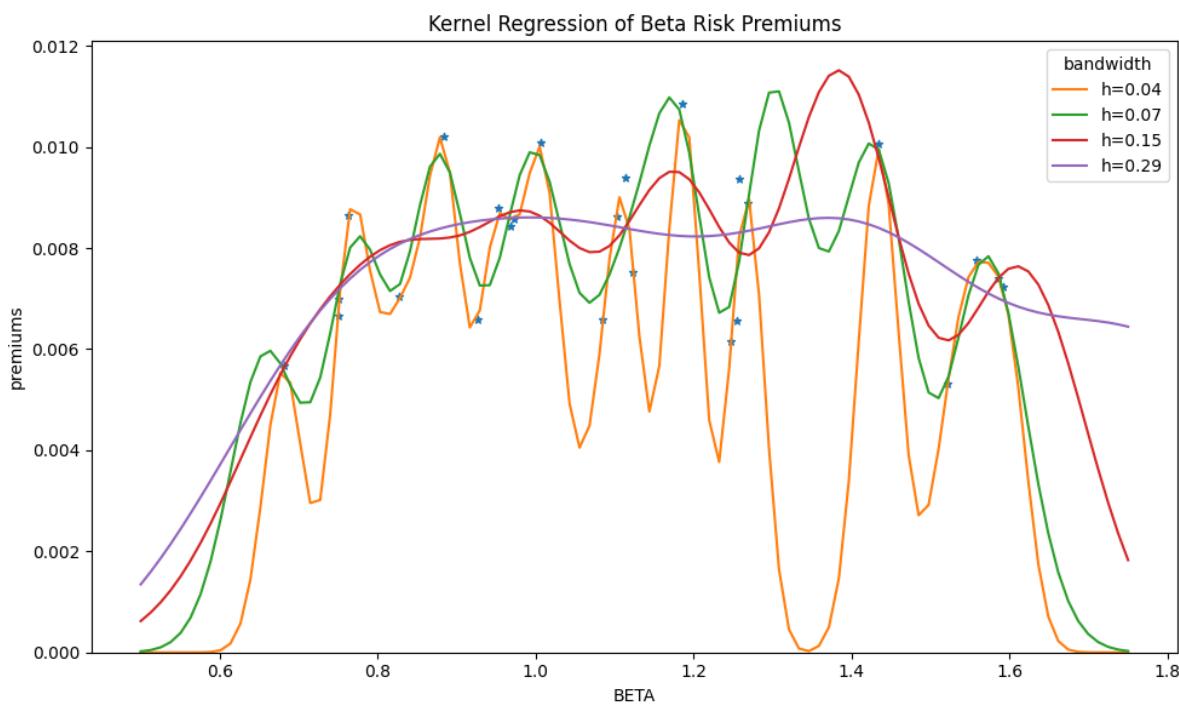
(continues on next page)

(continued from previous page)

```

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_ylim(bottom=0)
plt.legend(legend, loc='best', title='bandwidth')
plt.title('Kernel Regression of Beta Risk Premiums')
plt.tight_layout()

```



4.5 CSR with individual stocks

Beginning with Barra in the mid-1970's, industry practitioners have incorporated cross-sectional models using individual stock returns and characteristics, for the purpose of both forecasting risk premiums and measuring risk factors.

We run monthly cross-sectional regressions on the following individual stock characteristics (where each are winsored at 5% tail):

- size: rank of company market cap, standardized
- value: book-to-market ratio, standardized
- momentum: 12-month skip past month momentum, standardized
- reversal: 1-month reversal, standardized

The monthly cross-sectional risk premiums can be interpreted as returns to monthly-rebalanced dollar-neutral portfolios, each with unit exposure to a characteristic but zero net exposure to the other three characteristics. We compare these to the corresponding returns of the Fama-French factors, which are spreads of cap-weighted subportfolios.

```

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')
    #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])

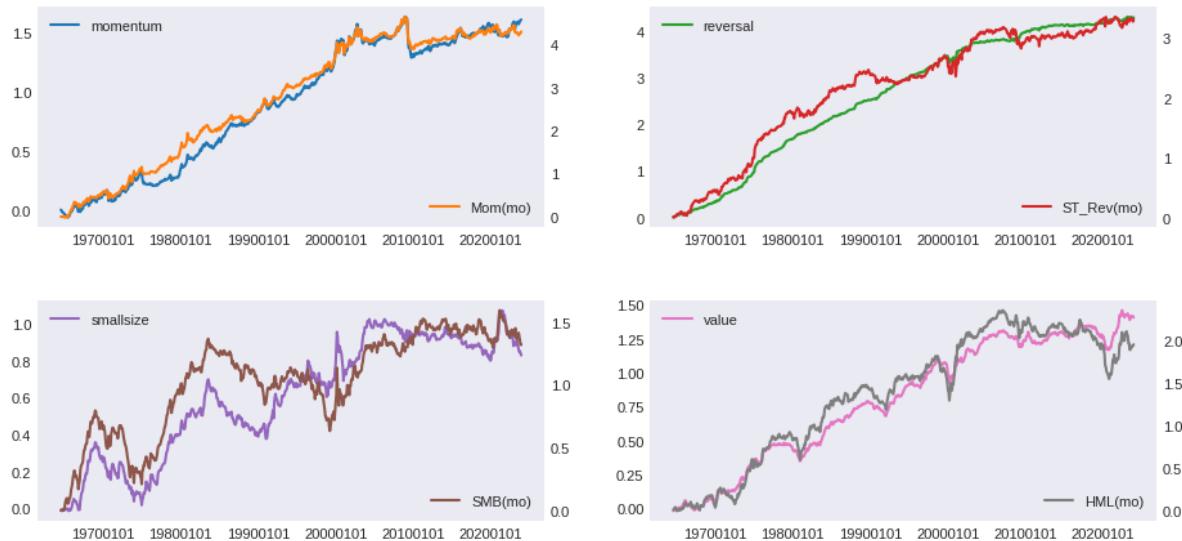
# Compute coefficients from FM cross-sectional regressions
monthly = CRSPBuffer(stocks=crsp, dataset='monthly', fields=['ret'],
                      beg=bd.begmo(rebalbeg, -13), end=bd.endmo(rebalend, 1))

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 returns
benchnames = {'momentum': 'Mom(mo)', 'reversal': 'ST_Rev(mo)', 'smallsize': 'SMB(mo)', 'value': 'HML(mo)'}
out = riskpremium.fit(benchnames.values())    # to compare portfolio-returns
riskpremium.plot(benchnames)

```

100% |██████████| 714/714 [00:32<00:00, 22.17it/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 | 0.0060 | 0.0005 | 0.0143 | 713 | 11.2067 | 1.4539 |
| value | 0.0020 | 0.0004 | 0.0112 | 713 | 4.7194 | 0.6123 |
| smallsize | 0.0012 | 0.0007 | 0.0179 | 713 | 1.7383 | 0.2255 |
| momentum | 0.0023 | 0.0007 | 0.0176 | 713 | 3.4282 | 0.4447 |

```
# 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) | 0.0060 | 0.0016 | 0.0424 | 713 | 3.8023 | 0.4933 |
| ST_Rev(mo) | 0.0046 | 0.0012 | 0.0316 | 713 | 3.9208 | 0.5086 |
| SMB (mo) | 0.0018 | 0.0011 | 0.0306 | 713 | 1.6112 | 0.2090 |
| HML (mo) | 0.0027 | 0.0011 | 0.0301 | 713 | 2.4228 | 0.3143 |

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

Correlation of FM Risk Premiums and FF Portfolio-Sort Returns

| | reversal | value | smallsize | momentum | Mom (mo) | ST_Rev (mo) | \ |
|-------------|----------|--------|-----------|----------|----------|-------------|---|
| reversal | 1.000 | 0.008 | 0.078 | -0.448 | -0.395 | 0.795 | |
| value | 0.008 | 1.000 | -0.224 | -0.189 | -0.166 | -0.021 | |
| smallsize | 0.078 | -0.224 | 1.000 | -0.004 | 0.132 | 0.006 | |
| momentum | -0.448 | -0.189 | -0.004 | 1.000 | 0.885 | -0.284 | |
| Mom (mo) | -0.395 | -0.166 | 0.132 | 0.885 | 1.000 | -0.313 | |
| ST_Rev (mo) | 0.795 | -0.021 | 0.006 | -0.284 | -0.313 | 1.000 | |
| SMB (mo) | 0.148 | -0.212 | 0.524 | -0.059 | -0.038 | 0.184 | |
| HML (mo) | 0.052 | 0.821 | -0.152 | -0.205 | -0.189 | 0.012 | |
| | | | SMB (mo) | HML (mo) | | | |
| reversal | | | 0.148 | 0.052 | | | |
| value | | | -0.212 | 0.821 | | | |
| smallsize | | | 0.524 | -0.152 | | | |
| momentum | | | -0.059 | -0.205 | | | |
| Mom (mo) | | | -0.038 | -0.189 | | | |
| ST_Rev (mo) | | | 0.184 | 0.012 | | | |
| SMB (mo) | | | 1.000 | -0.166 | | | |
| HML (mo) | | | -0.166 | 1.000 | | | |

CONTRARIAN TRADING

Fortune befriends the bold - Emily Dickinson

Concepts:

- Mean reversion
- Implementation shortfall
- Structural break with unknown changepoints
- Performance evaluation, information coefficient

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

```
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
```

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

(continues on next page)

(continued from previous page)

```
crsp = CRSP(sql, bd, rdb=rdb, verbose=VERBOSE)
bench = Benchmarks(sql, bd)
```

5.1 Mean reversion

The assumption of **mean reversion** underlies many investment and trading strategies. Similar assets should be priced identically, according to the **law of one price**, otherwise the opportunity for guaranteed **arbitrage** profits incentivize traders to buy the undervalued asset while short-selling the overvalued asset hence driving their prices closer together again. Deviations from long-run expectations, such as yield spreads in fixed income assets, should not persist. **Pairs trading** involves identifying two related securities, often based on their historical price relationships, and trading them simultaneously to profit from the temporary divergences in price between the two securities. **Statistical arbitrage**, also referred to as stat arb, models statistical relationships between securities and uses computationally intensive trading algorithms to simultaneously buy and sell portfolios of securities in financial market assets such as equities and commodities. Since such relationships are not risk-free, stat arb strategies seek to diversify as much as possible across a broad set of imperfectly correlated positions (unfortunately, correlations between strategies and asset classes seem to increase inconveniently around financial shocks, perhaps due liquidity constraints and deleveraging effects).

We construct and evaluate a **contrarian strategy** based on the weekly returns of US stocks and motivated by stock mispricing due to investor over-reaction, see Lo and Mackinlay (1990) and others. The **alpha** of this strategy can be approximated by its **information coefficient** and **volatility** components, resembling Grinold (1994)'s decomposition. We apply tests of structural breaks with unknown changepoints to examine the decline of its profitability over time. Trading strategies can experience large differences between the paper return of a portfolio where all trades are assumed to have transacted at the decision price and the actual return of the portfolio using actual transaction prices and amounts executed. The concept of **implementation shortfall**, introduced by Perold (1988), measures the total cost of implementing an investment decision by capturing both explicit and implicit trading costs, such as market impact, delay, as well as 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 $\sum w_t = \sum X_t/n = 0$.

As an aside, 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)

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']) / 
               (past_week['ret'].std(ddof=0) * len(past_week)))

    # 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

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

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

```

(continues on next page)

(continued from previous page)

```

# 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') ==
               bd.offset(rebaldate, 1)]\ 
               .reset_index()\ 
               .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

```

```
Last FamaFrench Date 2024-02-29 00:00:00
```

```
100%|██████████| 2607/2608 [01:17<00:00, 33.74it/s]
```

```
# Combine accumulated computations and report
df = pd.concat(results, axis=0)
dates = df.index
df.index = pd.DatetimeIndex(df.index.astype(str))
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]}'')
pd.concat([df[cols].mean(axis=0).rename('mean'),
           df[cols].std(axis=0).rename('std')], axis=1) \
.set_index(pd.Index(indexes)).round(4)
```

```
Summary of Weekly Mean Reversion Strategy 19740109-20231220
```

| | mean | std |
|----------------------------|---------|--------|
| Information coefficient | 0.0379 | 0.1027 |
| Cross-sectional Volatility | 0.0535 | 0.0192 |
| Gross return (alpha) | 0.0021 | 0.0080 |
| Slippage cost | 0.0017 | 0.0048 |
| 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 | 0.0042 |
| Spread cost | 0.0013 | 0.0023 |
| Portfolio turnover | 0.7362 | 0.0408 |

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.

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, since the stock market exchanges experience the most liquidity during the trading day at the close. 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 the CRSP database, closing bid and ask quotes are recorded so that the closing price, when not available, is set to the negative of the bid-ask average. Transactions costs are challenging to model and cannot be observed from historical prices in backtest simulations, so this measurement approach feasibly assumes trades are executed at the close a full day after their decision prices are considered.

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 the detection of structural change involving an unknown number of breaks in mean with unknown break points.

The R library `strucchange` implements tests for structural breaks with unknown breakpoints. The external Python package `rpy2` provides an interface to R libraries and programming language. 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
importr('strucchange')    # R package to use

# 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()

(continues on next page)
```

(continued from previous page)

```
output['mean(post)'] = Y[df.index > output['breakpoints']].mean()

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

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

Structural break test with unknown changepoint

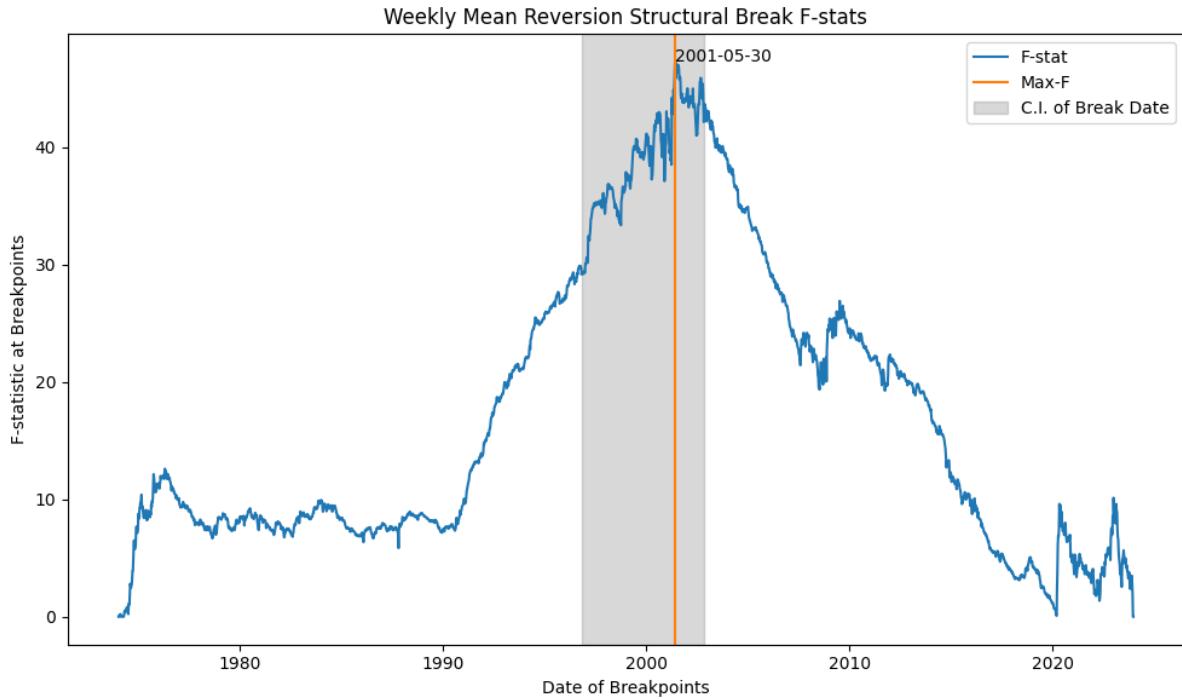
| | 2.5 % breakpoints | 97.5 % | statistic | p.value | method | \ |
|--------|-------------------|------------|------------|-----------|--------|-----------|
| sctest | 1996-10-23 | 2001-05-30 | 2002-11-06 | 17.226016 | 0.0 | aveF test |
| | mean(pre) | mean(post) | | | | |
| sctest | 0.003112 | 0.000957 | | | | |

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_1727373/3930510136.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 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 is simply measured by the (negative) cross-sectional correlation of stock returns from one week to the next.

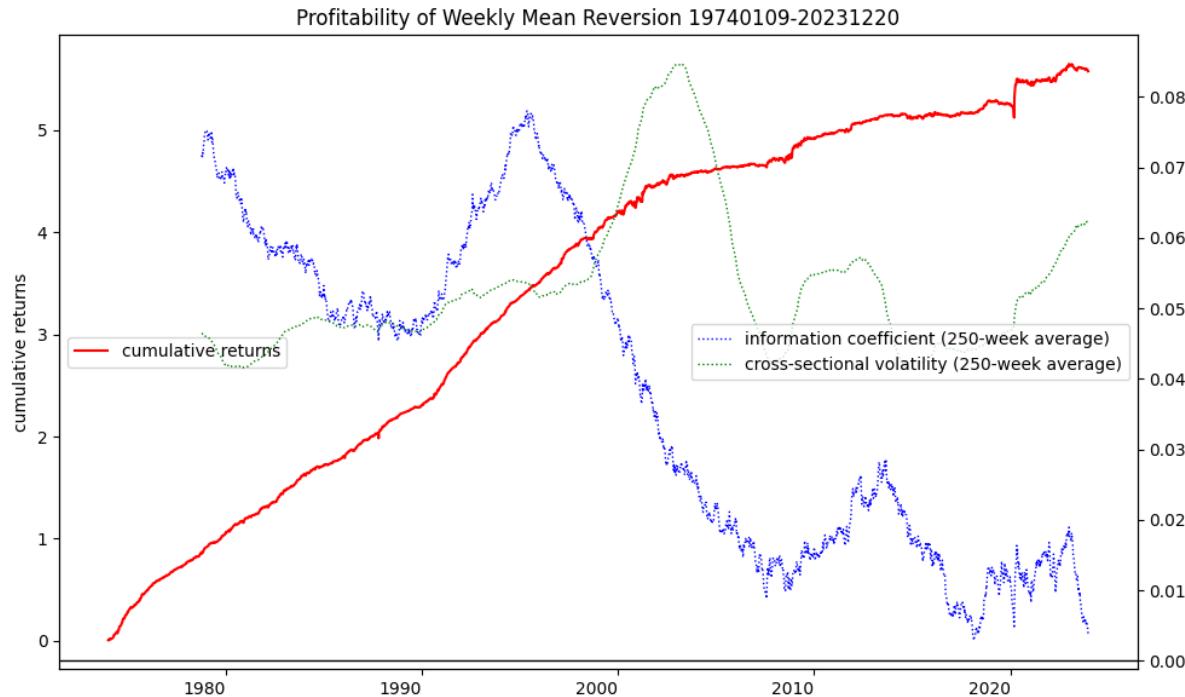
We plot the 5-year rolling average alpha and its information coefficient and volatility components. Whereas volatility exhibits cyclical changes, the estimated IC showed a sharp downward trend from a peak in the mid 1990's down to 2010.

```
## 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
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')
```

(continues on next page)

(continued from previous page)

```
#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()
```



5.5 Performance evaluation

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, the return with fair compensation for their risk exposure: portfolio managers rely on market benchmarks to measure the performance of a given investment relative to the CAPM equilibrium risk-return relationship.

- 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 | 20231220 | 20010530 | 20231220 |
| Sharpe Ratio | 1.9195 | 3.6667 | 0.7078 |
| Average Gross Return | 0.0021 | 0.0031 | 0.0010 |
| Std Dev Returns | 0.0080 | 0.0061 | 0.0097 |
| Market Beta | 0.0905 | 0.0686 | 0.1127 |
| Jensen Alpha (annualized) | 0.1041 | 0.1566 | 0.0405 |
| Appraisal Ratio | 1.8591 | 3.6606 | 0.6001 |
| Information Coefficient | 0.0379 | 0.0578 | 0.0138 |
| Cross-sectional Vol | 0.0535 | 0.0542 | 0.0526 |
| Total Slippage Cost | 0.0017 | 0.0028 | 0.0003 |
| Spread Cost | 0.0013 | 0.0023 | 0.0002 |
| Delay Cost | 0.0004 | 0.0006 | 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 | 0.7362 | 0.7393 | 0.7325 |
| Average Num Stocks | 1936 | 1981 | 1882 |

The structural break test suggested a statistically significant changepoint in mid-2001 in the time series of the strategy's weekly returns. The performance of the strategy in the periods before and after that dates shows a noticeable decline in annualized Sharpe ratio and average weekly returns, as well as an increase in riskiness. This coincides approximately with the New York and American Stock Exchanges switching to decimalization on Jan 29, 2001, which has led to tighter spreads. Previously, markets in the US utilized fractions in price quotes, where one-sixteenth (1/16) of one dollar was the smallest price movement that could be presented in a price quote.

QUANT FACTORS

Quants do it with models - Anonymous

Concepts

- Factor investing
- Performance Evaluation
- Cluster analysis

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”.
- FRM Part II Exam Book Investment Manager Ch. 1-3.
- Andrew Ang, 2014, “Asset Management: A Systematic Approach to Factor Investing”

```
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,
```

(continues on next page)

(continued from previous page)

```

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)
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

Under this view of investing, factor risks are the driving force behind assets' risk premiums. The market is an example of a tradeable, investment factor – the CAPM states that this is the only factor driving all asset returns. Other examples include interest rates, value-growth investing, low volatility investing, and momentum portfolios. Factors can also be fundamental macroeconomic factors, like inflation and economic growth. All assets have different exposures to risk factors and the greater the exposure, the higher the risk premium. Hence assets are merely bundles of factors.

Early multifactor models included the arbitrage pricing theory (APT) by Stephen Ross (1976), which uses the word “arbitrage” because the factors cannot be arbitrated or diversified away; and the intertemporal capital asset pricing model (ICAPM) by Robert C. Merton, based on the assumption of investors hedging risky positions over multiperiods. Behavioral theories, centering around investor under- or overreaction to recent news, other psychological biases, or bounded rationality, can also produce factor premiums.

6.1.1 Adaptive Markets Hypothesis

Lo's (2004) “Adaptive Markets Hypothesis” describes how financial markets are governed more by the laws of evolutionary biology than by the laws of physics. He suggests that the performance of investments would vary over time as the financial ecosystem and market conditions evolve. The study of financial markets should start by tracking the different species, that is the collections of individuals and institutions that share common traits, and their size, growth rates, interactions and other characteristics.

6.1.2 Factor Zoo

Cochrane (2011) coined the term “Factor Zoo” in response to the rate of factor production in academic research. Green, Hand and Zhang (2017) replicated the studies of almost 100 firm characteristic factors, avoiding the overweighting of microcap stocks and data snooping bias, to examine their predictability as a whole as well as during recent time periods. We attempt to reproduce many of these below.

```
# 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

(continues on next page)

(continued from previous page)

```

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())
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}}\n".format("Annualized"))
        f.write(f"{{:12s}}\n".format("".join(f"{{k:>10s}}" for k in sub) + "\n"))
        f.write(f"{{:12s}}\n".format(label + ":"))
        f.write(f"{{:12s}}\n".format("".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 divyld from CRSP monthly

```

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)
        df['start'] = monthly.get_section(dataset='monthly',

```

(continues on next page)

(continued from previous page)

```

        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)
    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['rebaldate'] = rebaldate
    out.append(df.dropna(subset=['start', 'end2']) \
        [['rebaldate', '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 [3:09:36<00:00, 2844.01s/it]

```

benchnames = ['Mkt-RF(mo)']
rebalbeg, rebalend = 19260101, LAST_DATE
columns = ['mom12m', 'mom6m', 'chmom', 'indmom', 'divyld', 'mom1m', 'mom36m']

```

(continues on next page)

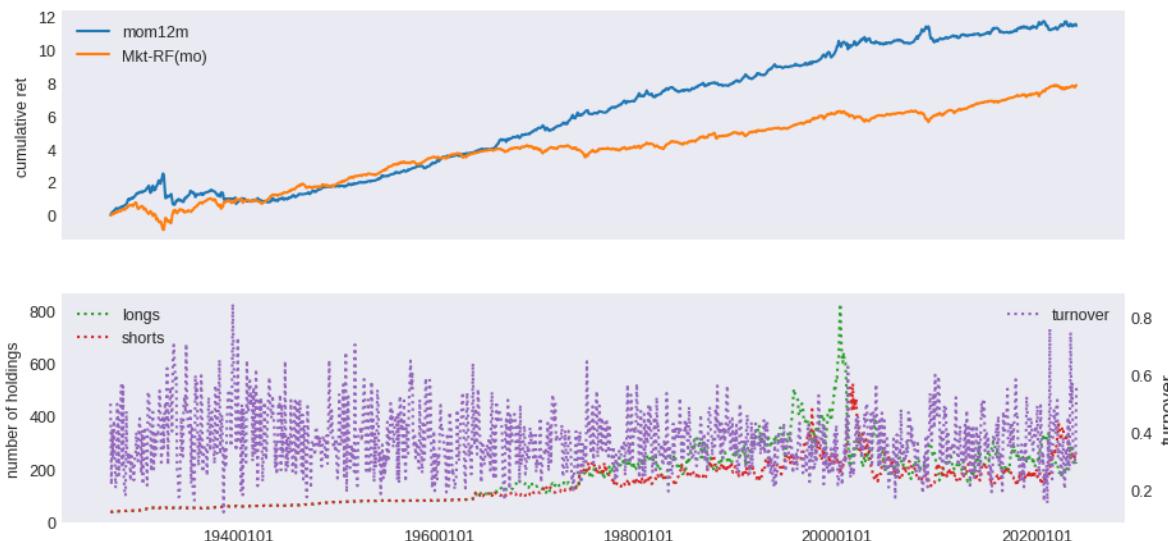
(continued from previous page)

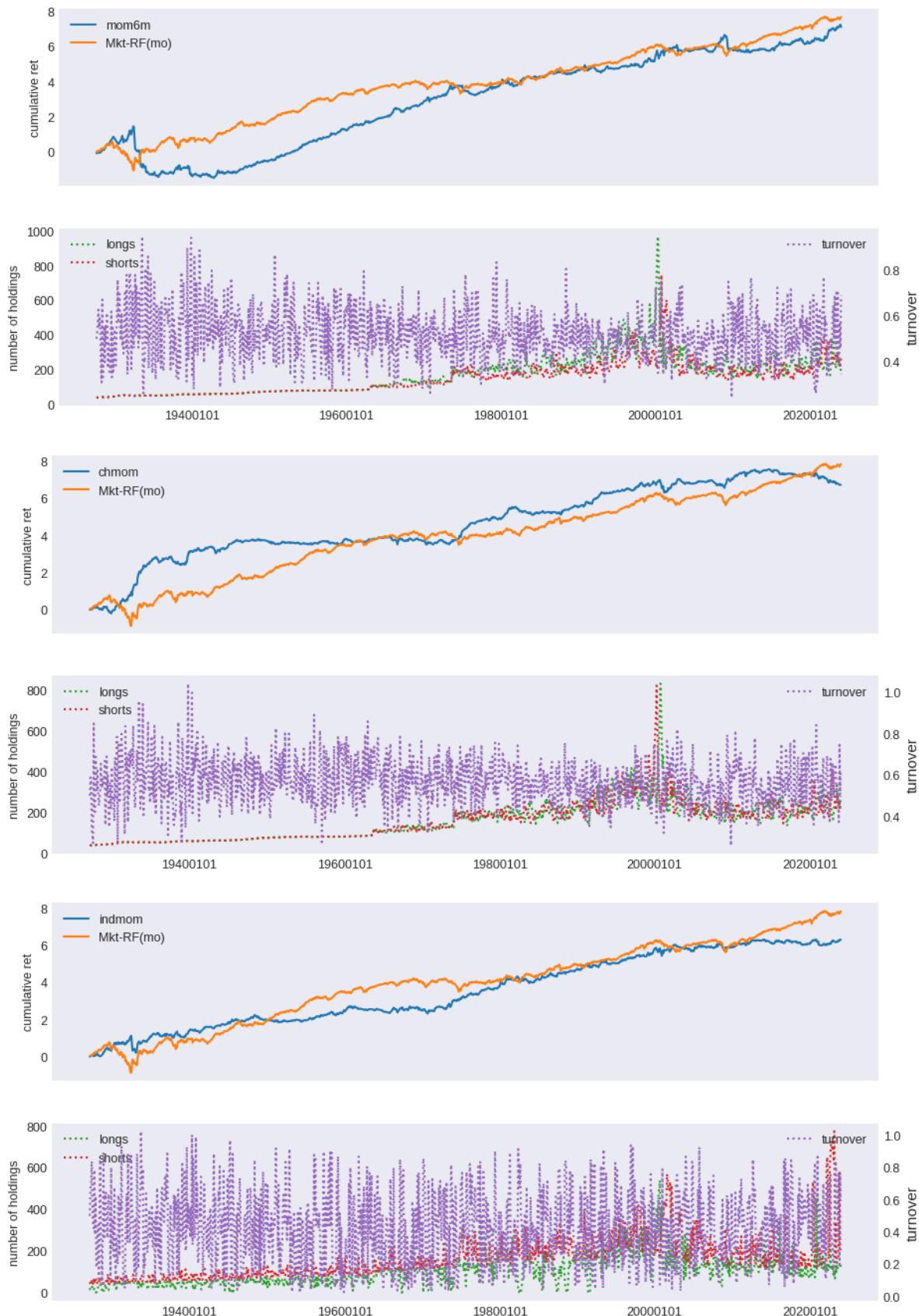
```

for label in tqdm(columns):
    holdings = univariate_sorts(monthly,
                                 label,
                                 SignalsFrame(signals.read(label)),
                                 rebalbeg,
                                 rebalend,
                                 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 [55:16<00:00, 473.81s/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 R2 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
    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

```

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

```

Last FamaFrench Date 2024-02-29 00:00:00

```

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) >= minvalid # 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)

```

0% | 0/5113 [00:00<?, ?it/s]

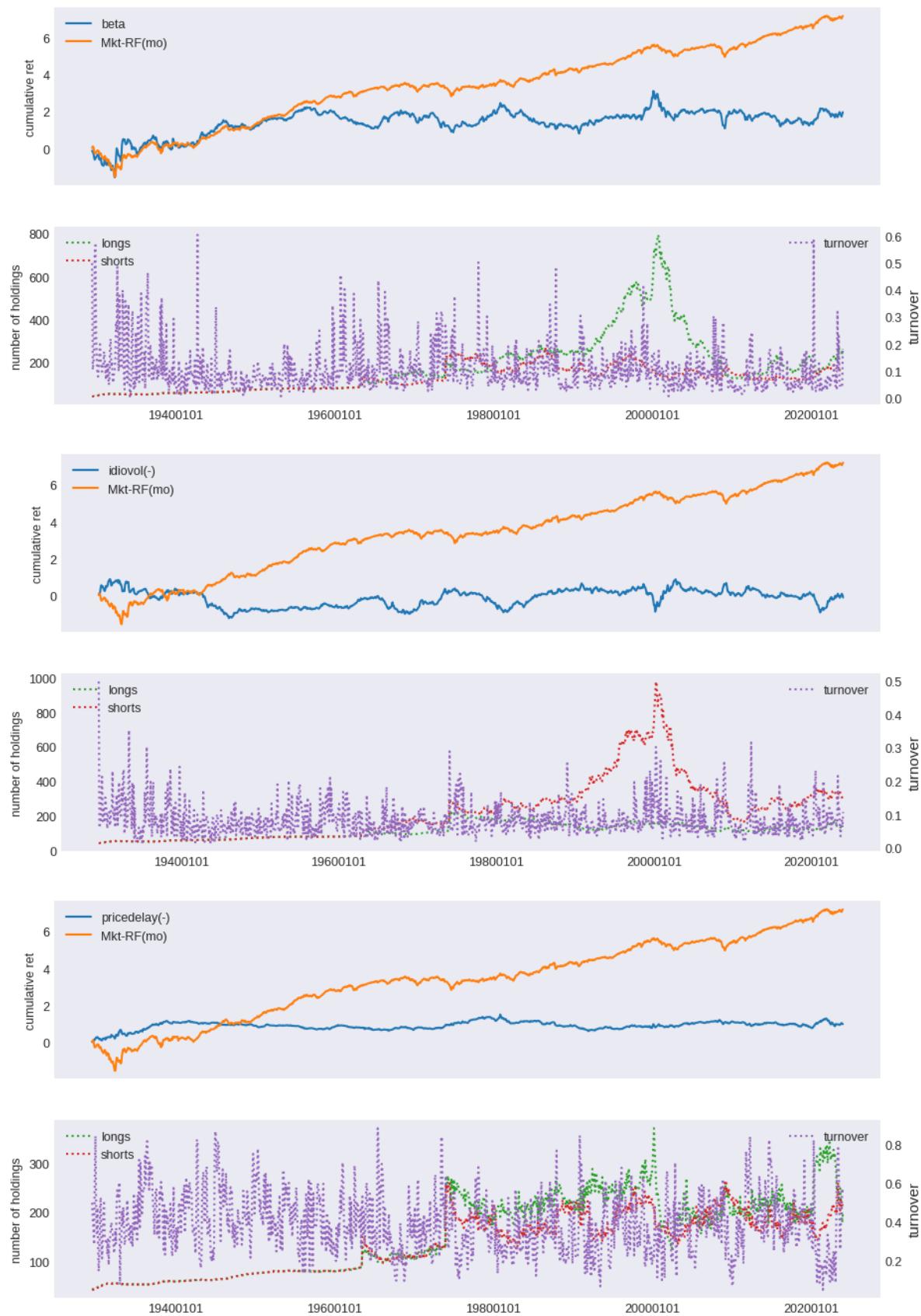
100% |██████████| 5113/5113 [33:48<00:00, 2.52it/s]

```

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 [16:04<00:00, 321.61s/it]



6.1.4 Liquidity

Liquidity signals from daily stock returns

```

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, shroutr "
          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['shroutr']
    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)

```

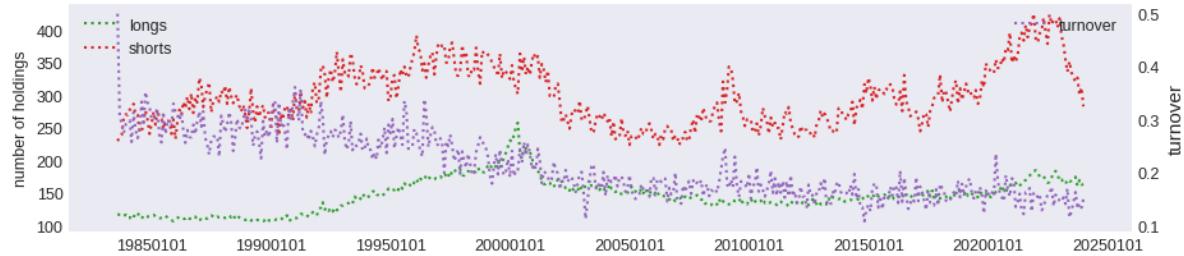
0% | 0/41 [00:00<?, ?it/s]

100% | ██████████ | 41/41 [26:09<00:00, 38.28s/it]

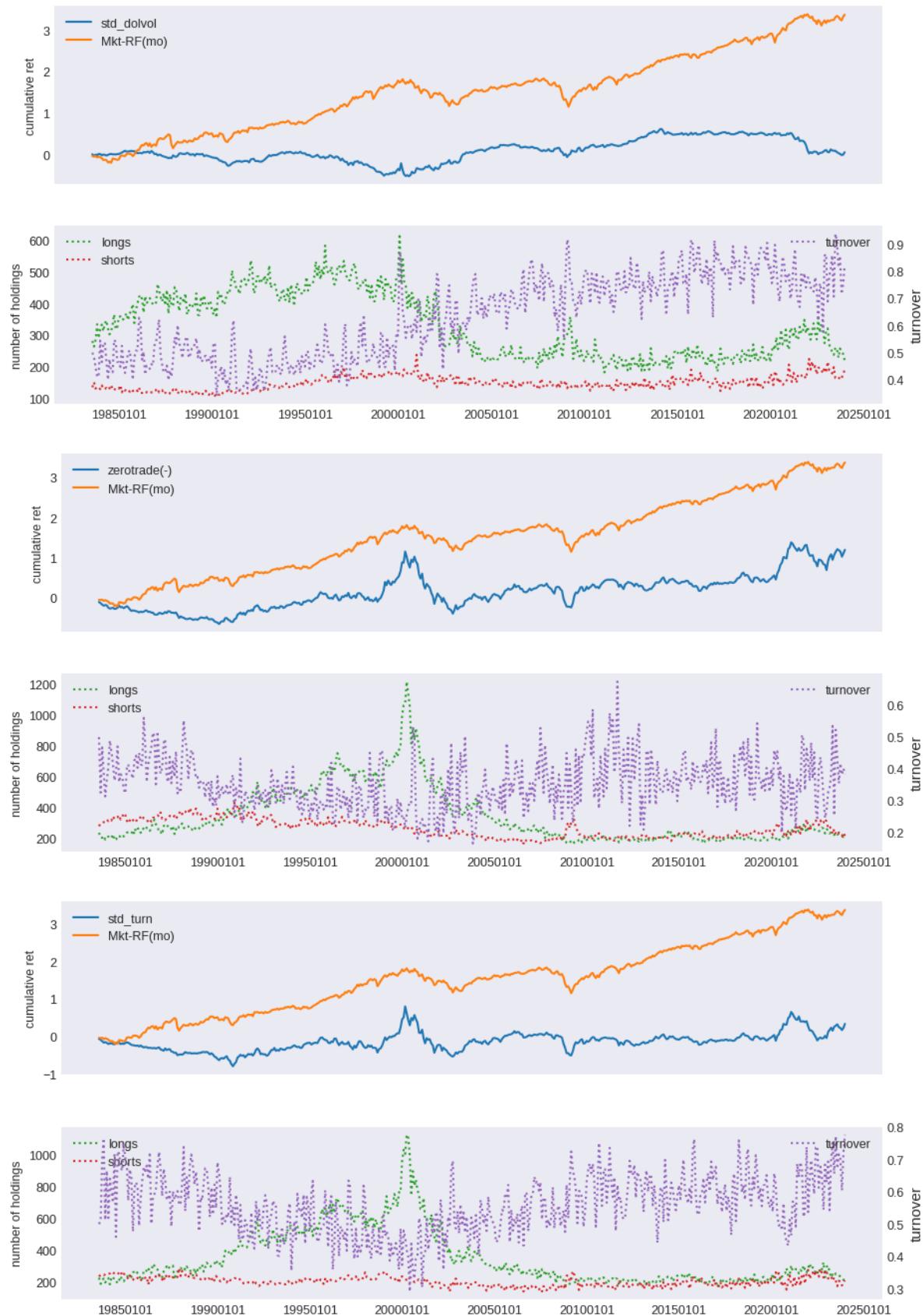
```
for label in columns:
    n = signals.write(out, label, overwrite=True)
    n = signals.write(dolvol, 'dolvol', overwrite=True)
```

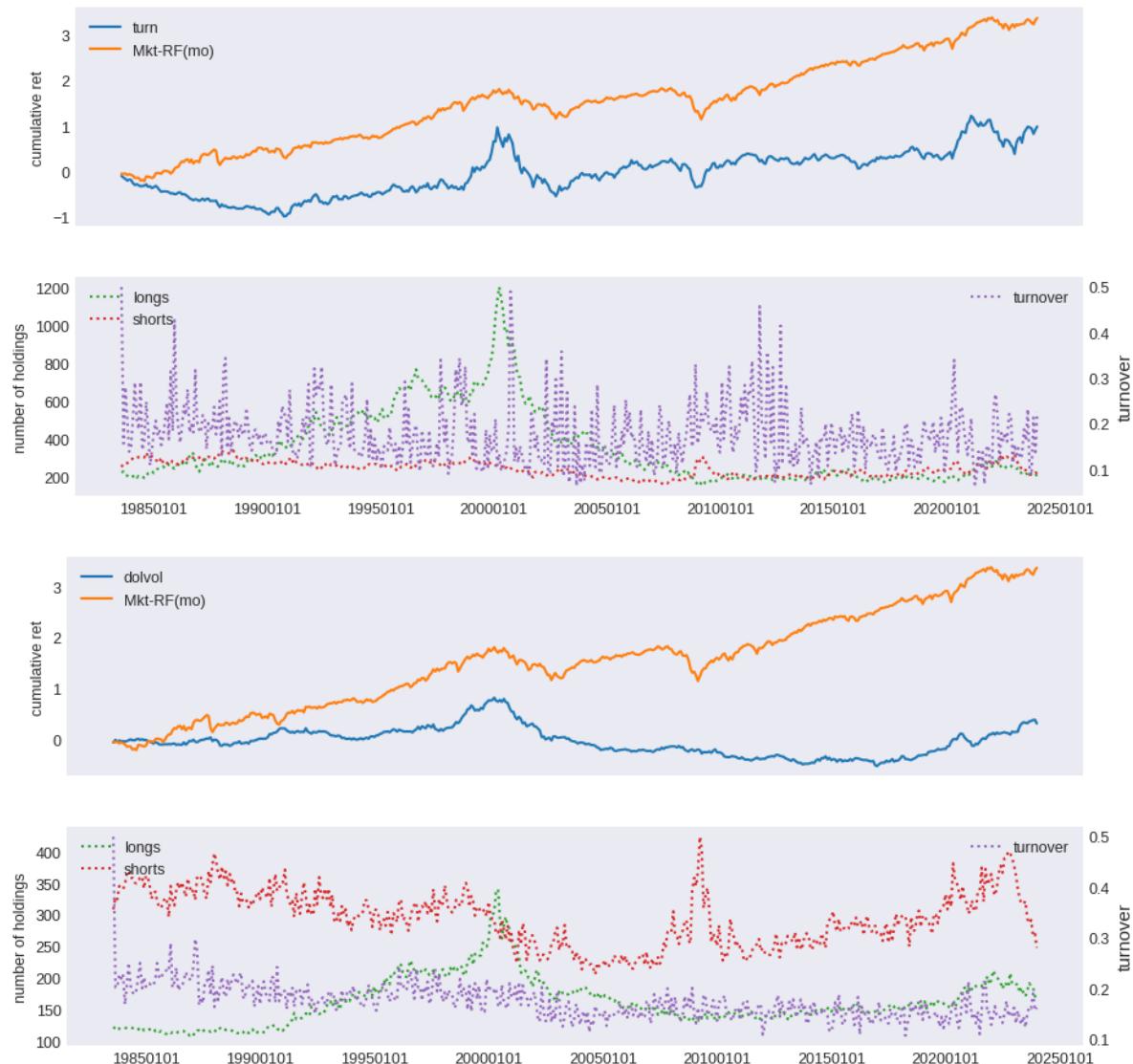
```
rebalbeg, rebalend = 19830601, LAST_DATE
benchnames = ['Mkt-RF(mo)']
for label in tqdm(columns + ['dolvol']):
    holdings = univariate_sorts(monthly,
                                 label,
                                 SignalsFrame(signals.read(label)),
                                 rebalbeg,
                                 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 [59:11<00:00, 394.66s/it]









6.1.5 Fundamentals

Fundamental signals from Compustat Annual

```
columns = ['absacc', 'acc', 'agr', 'bm', 'cashpr', 'cfp', 'chcsho',
           'chinv', 'depr', 'dy', 'egr', 'ep', 'gma', 'grcapx',
           'grltnoa', 'hire', 'invest', 'lev', 'lgr',
           'pchldepr', '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['rebalddate'] = 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['ppegt'].notna(),
                               fund['ppegt'], 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']
                         + 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['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['pchgmpchsale'] = (((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['pchsaledpchinv'] = ((fund['sale'] / lag['sale']) - (fund['invt'] / lag['invt']))
fund['pchsaledpchrect'] = ((fund['sale'] / lag['sale']) - (fund['rect'] / lag['rect']))
fund['pchsaledpchxsga'] = ((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'])
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,

```

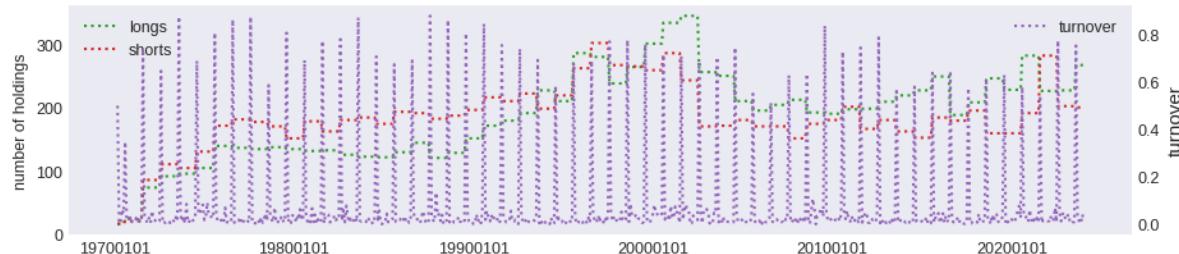
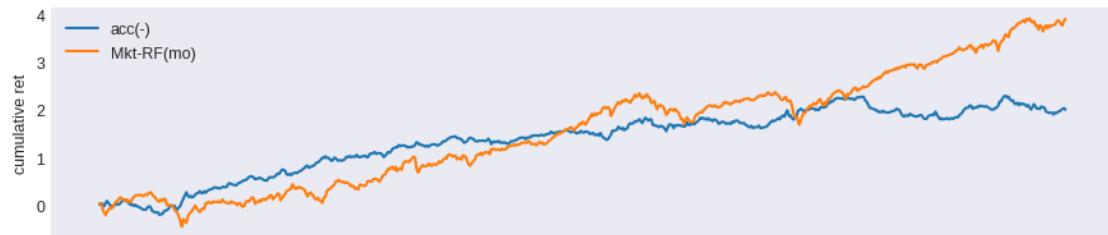
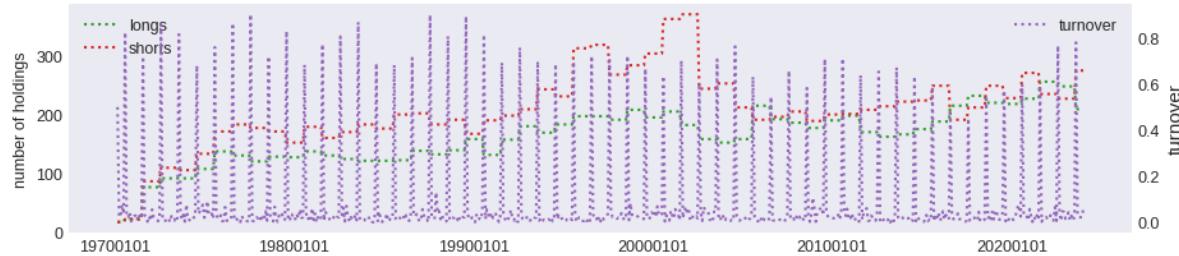
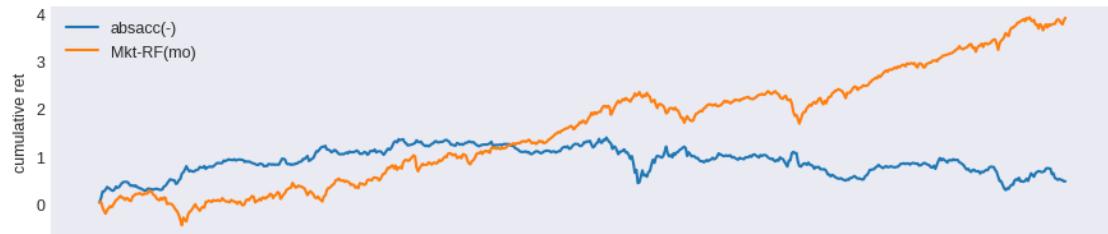
(continues on next page)

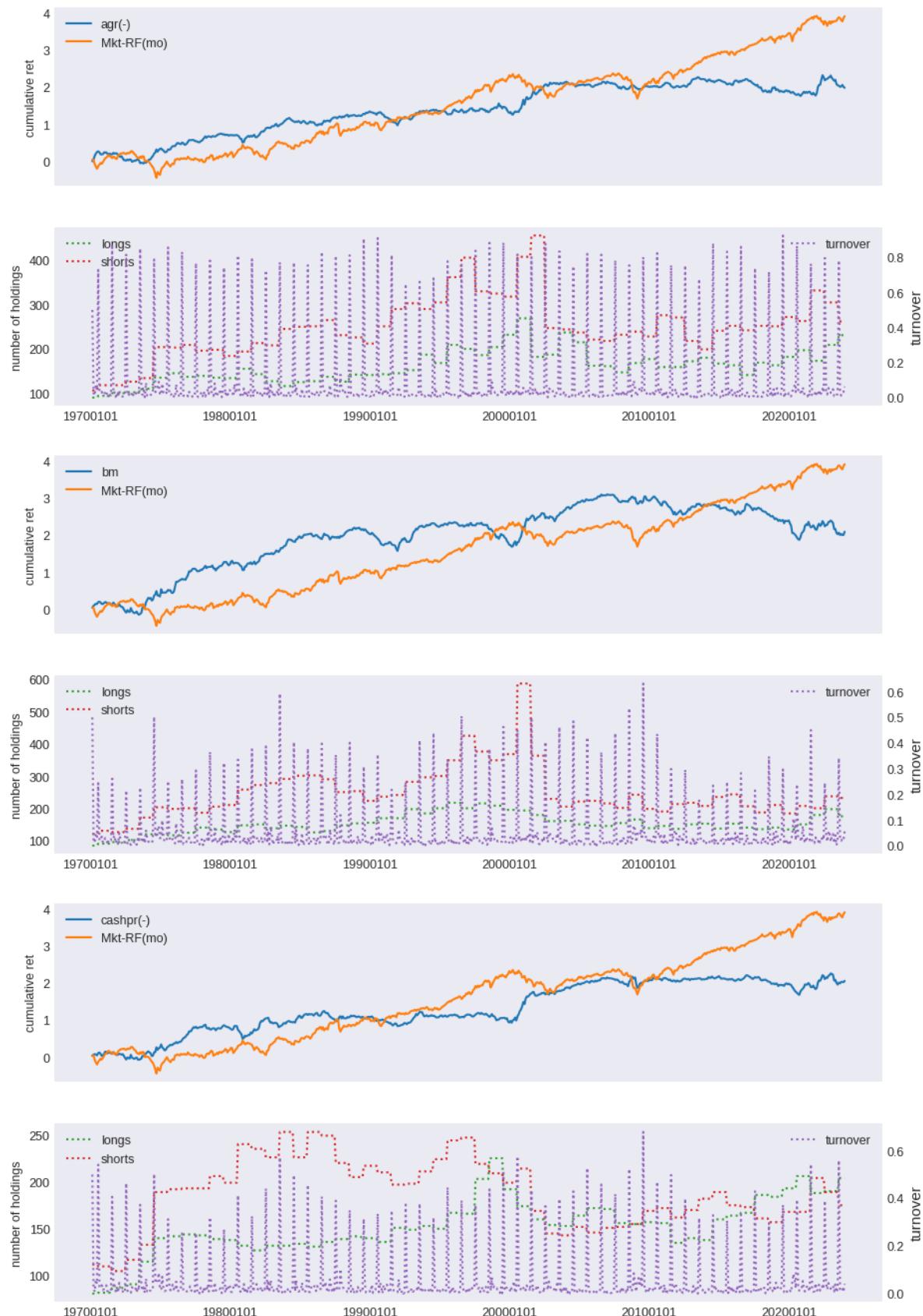
(continued from previous page)

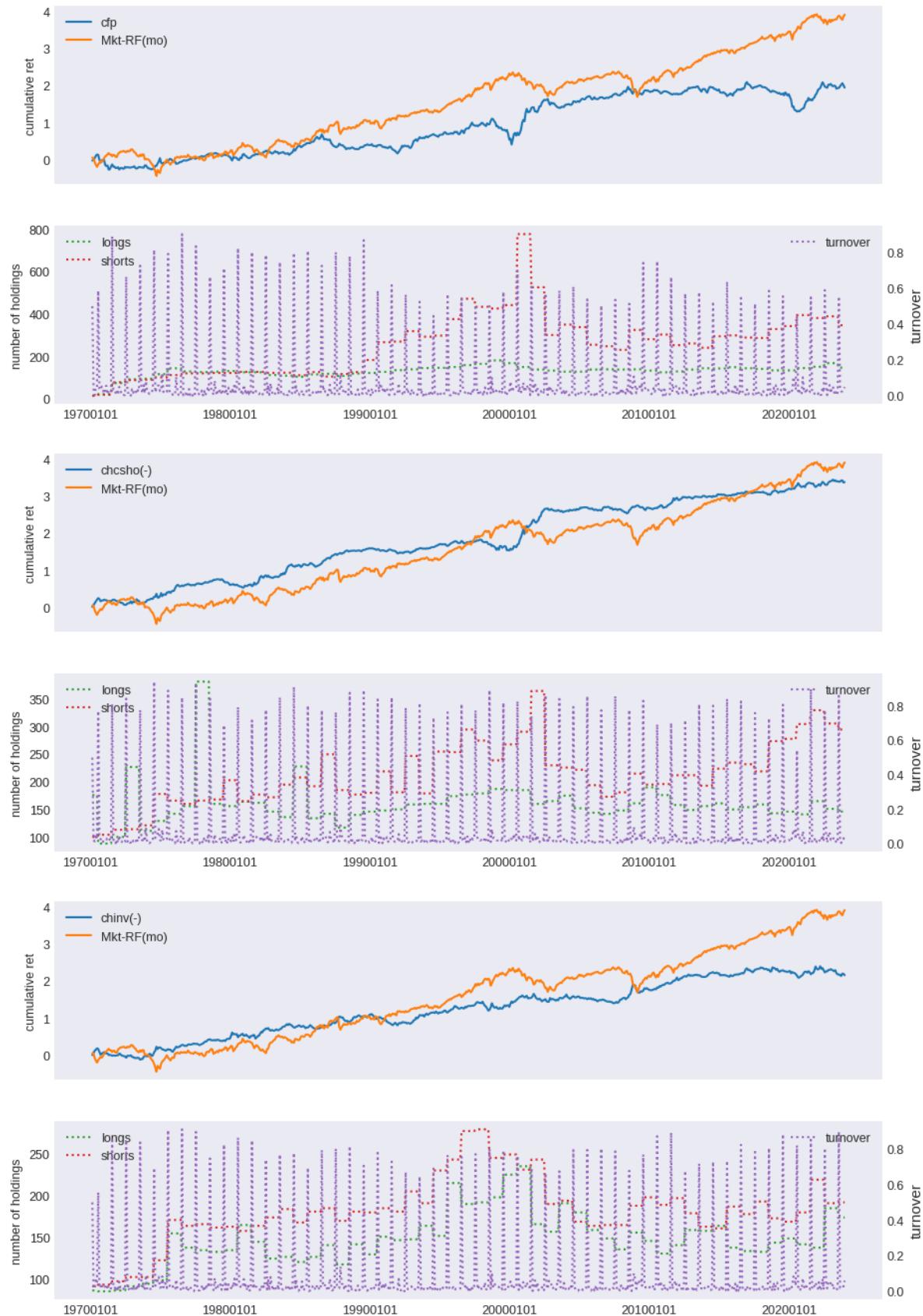
```
suffix=(leverage.get(label, 1) < 0) * '(-)'
```

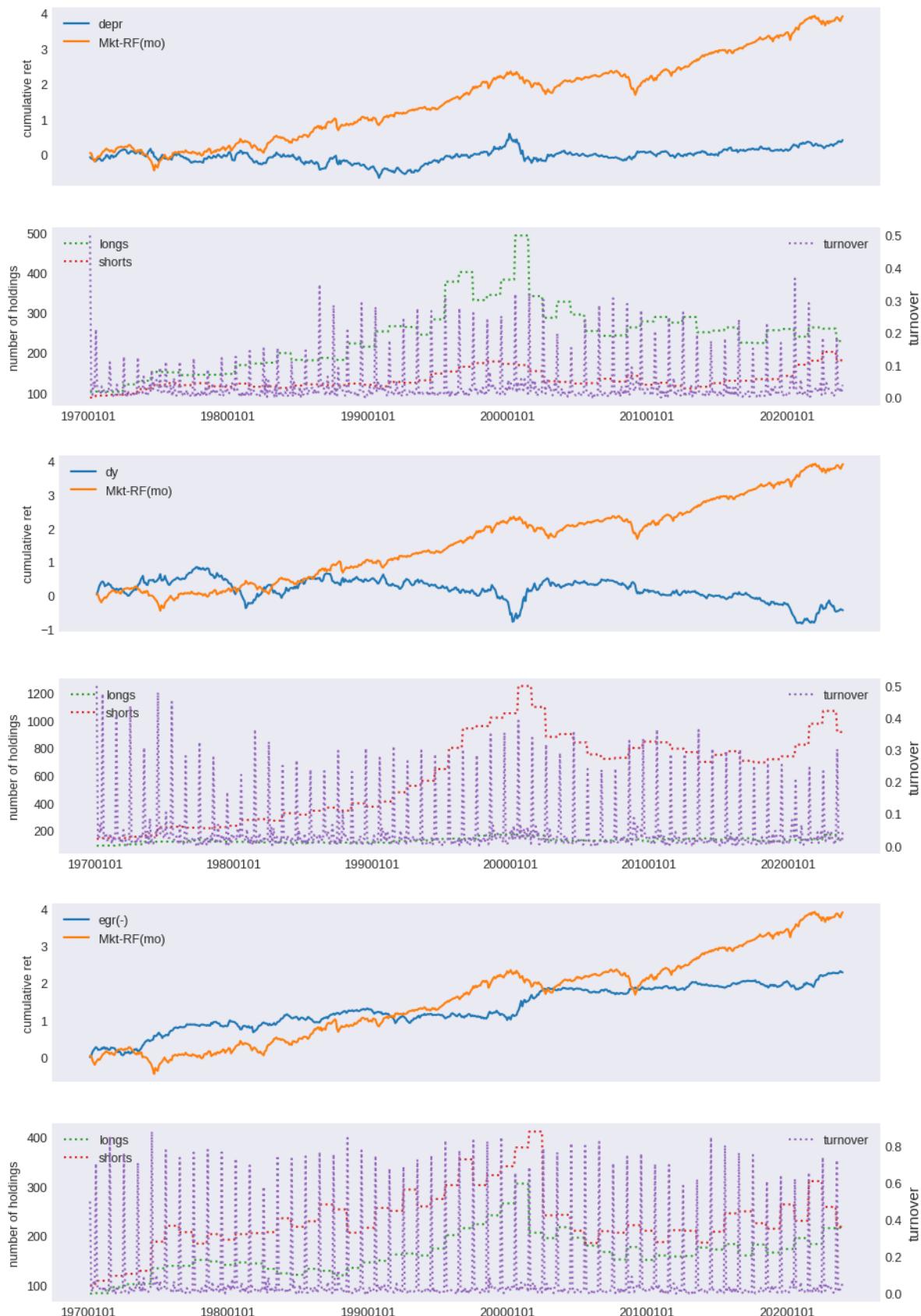
```
44%|██████████| 20/45 [21:57<27:24, 65.77s/it]/home/terence/Dropbox/github/data-
science-notebooks/finds/backtesting/backtest.py:310: RuntimeWarning: More than
20 figures have been opened. Figures created through the pyplot interface
(`matplotlib.pyplot.figure`) are retained until explicitly closed and may
consume too much memory. (To control this warning, see the rcParam `figure.max_
open_warning`). Consider using `matplotlib.pyplot.close()`.

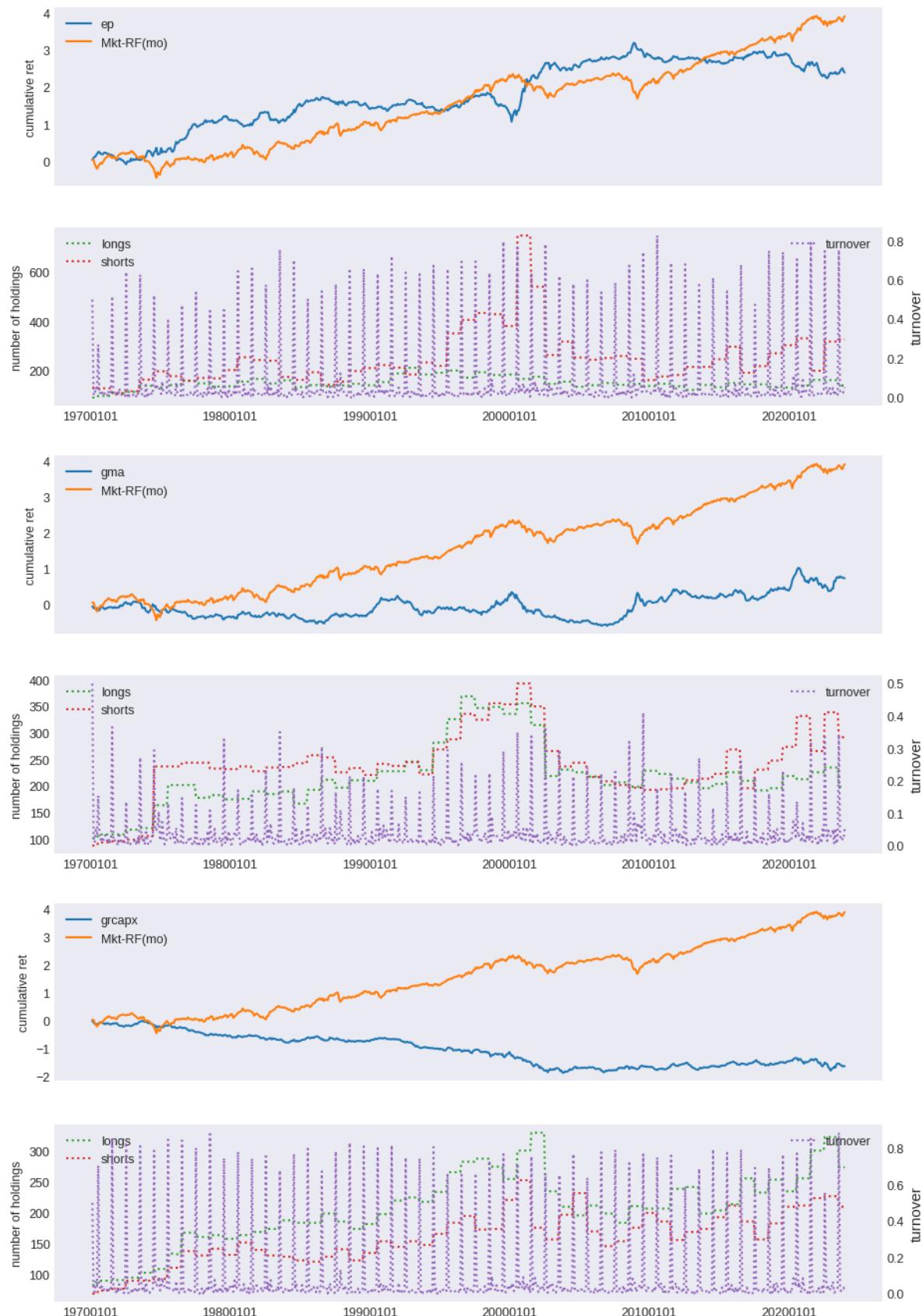
  fig, (ax1, ax2) = plt.subplots(2, 1, sharex=True, clear=True,
100%|██████████| 45/45 [48:40<00:00, 64.89s/it]
```

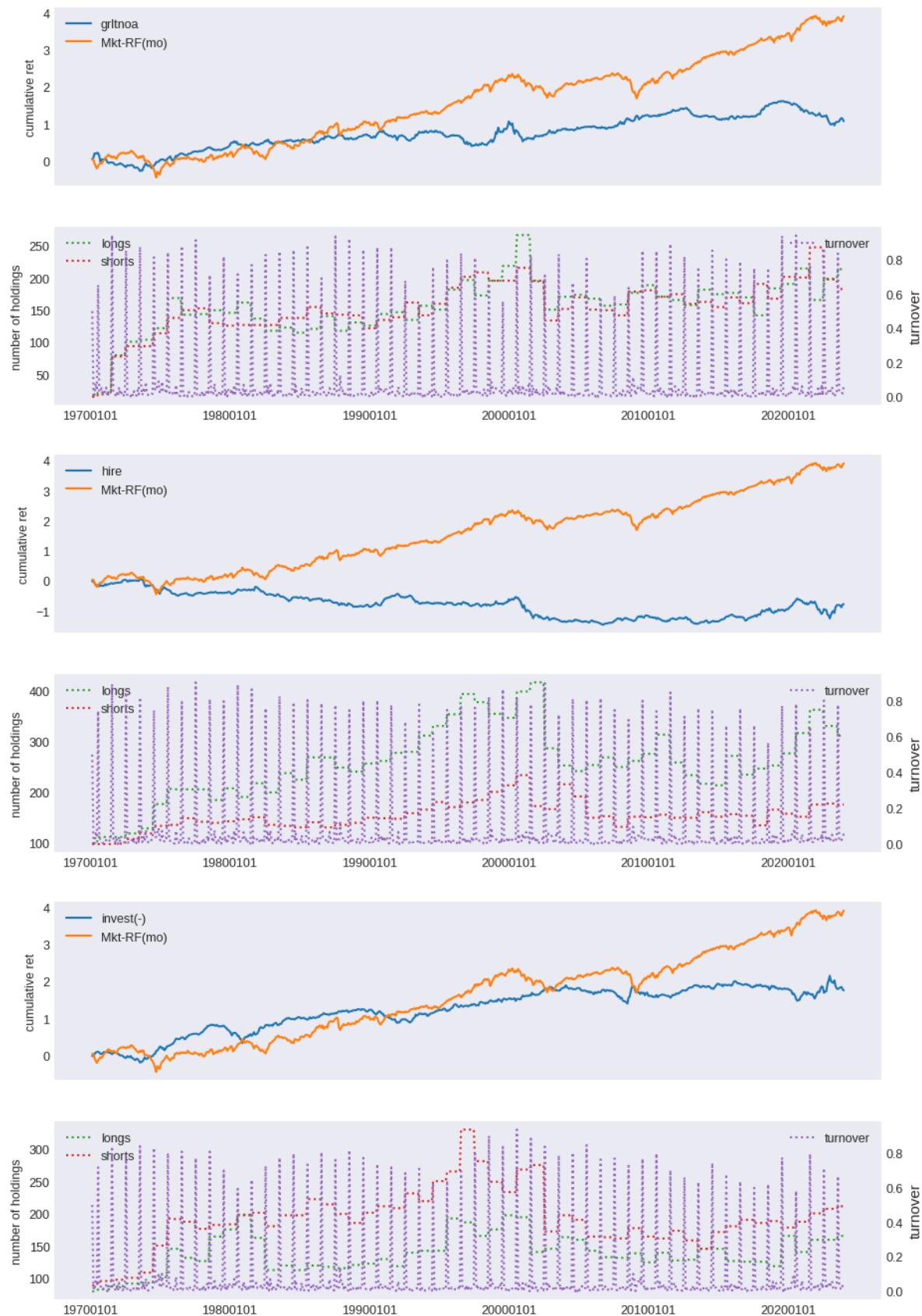


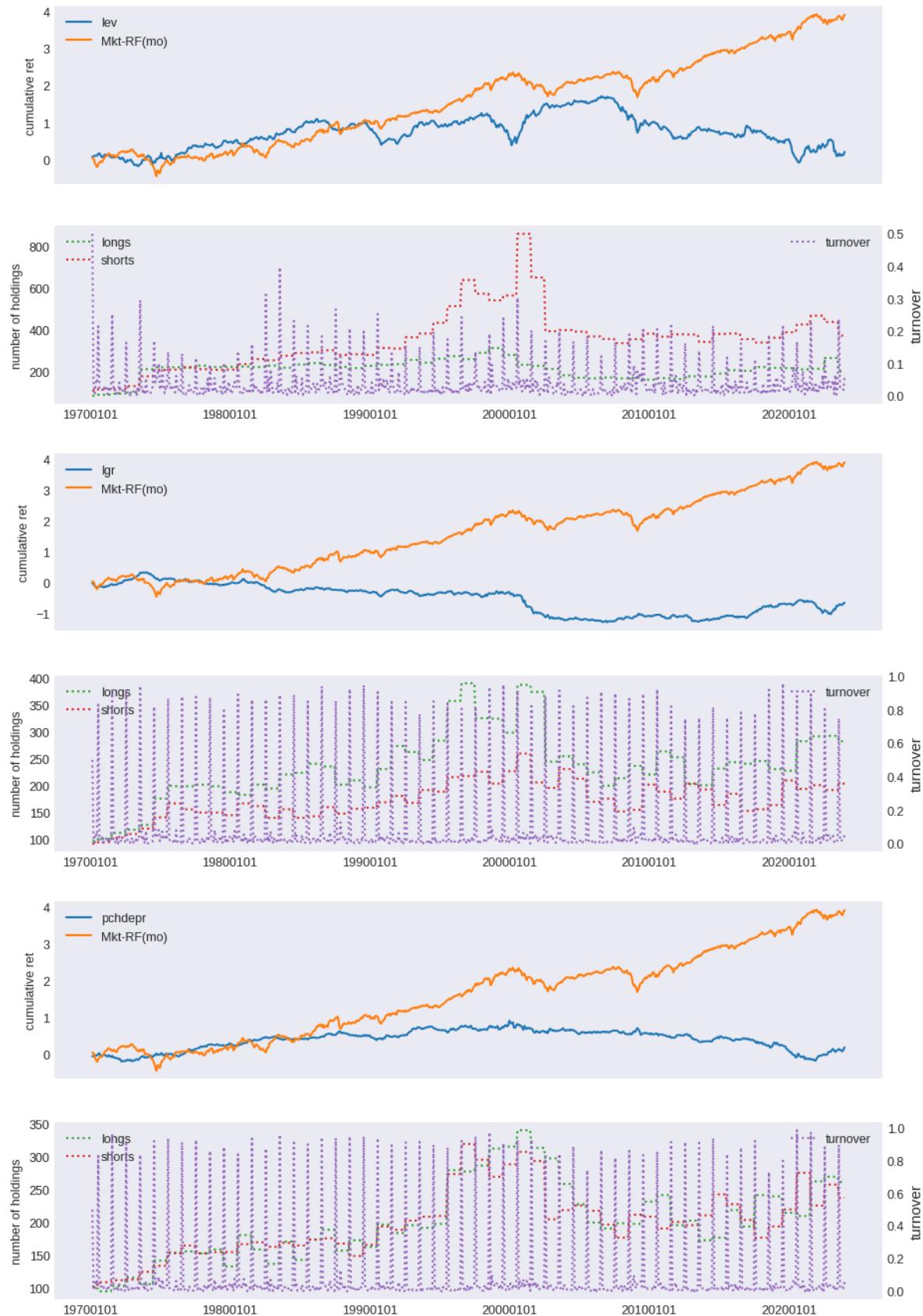






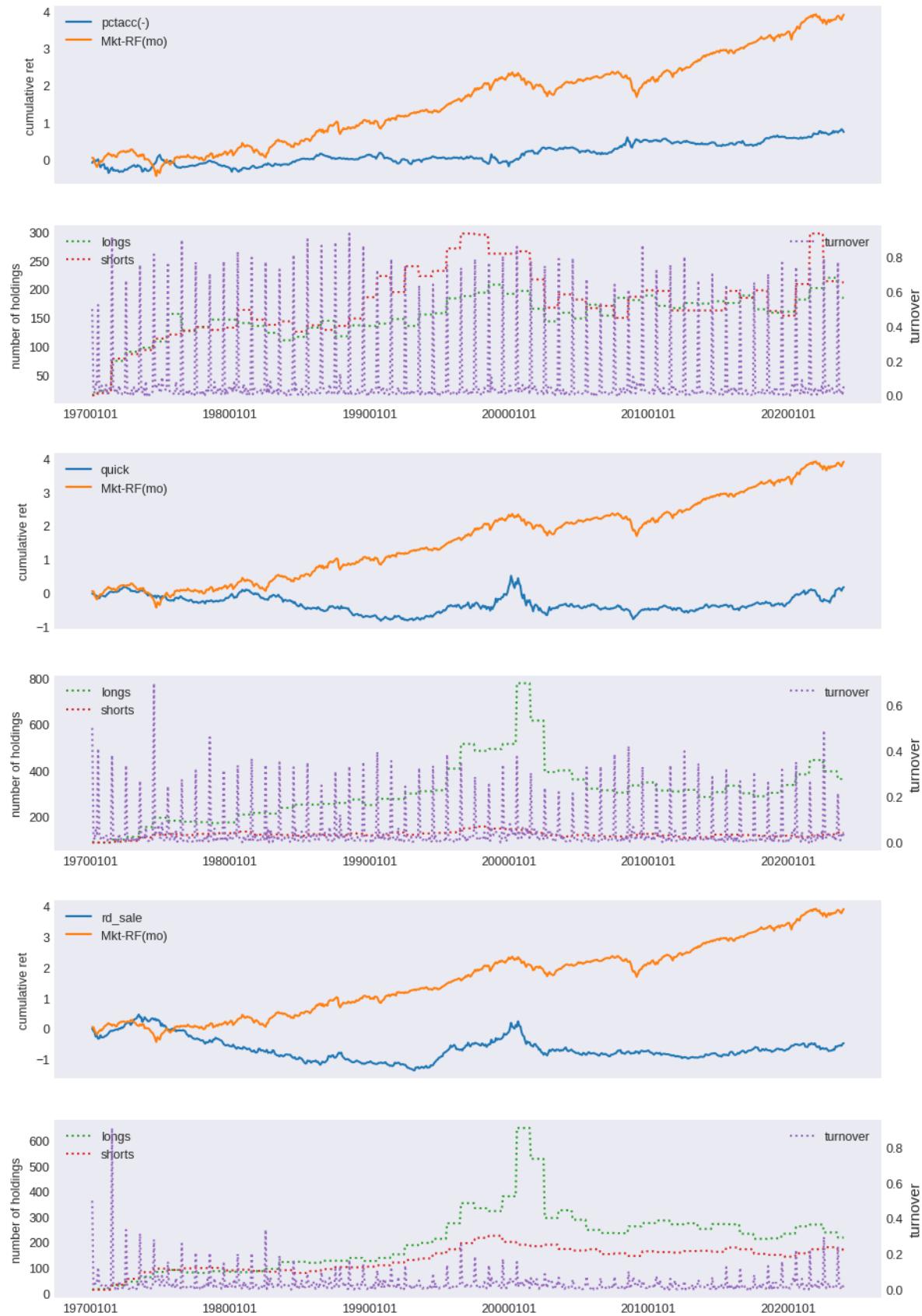


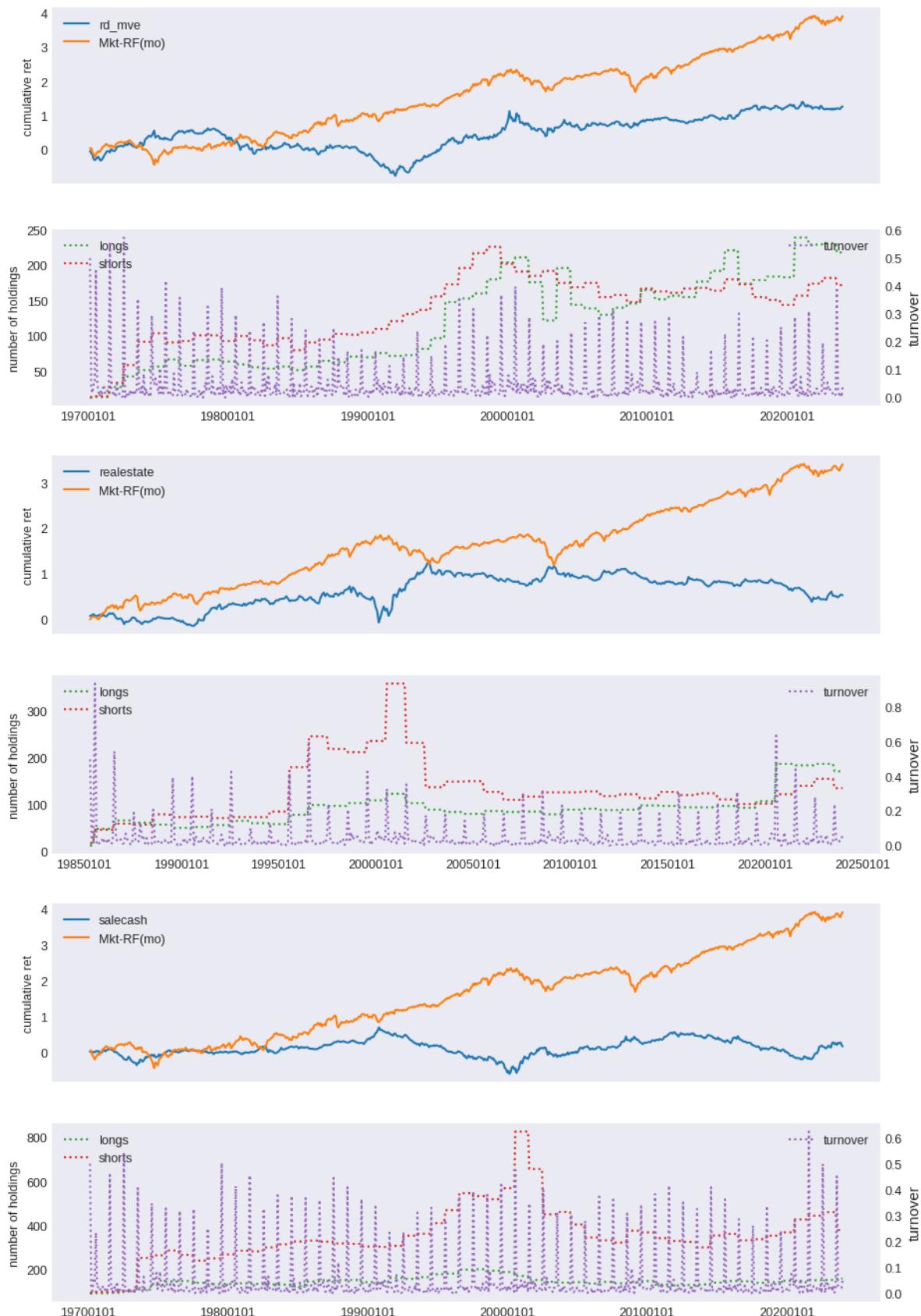


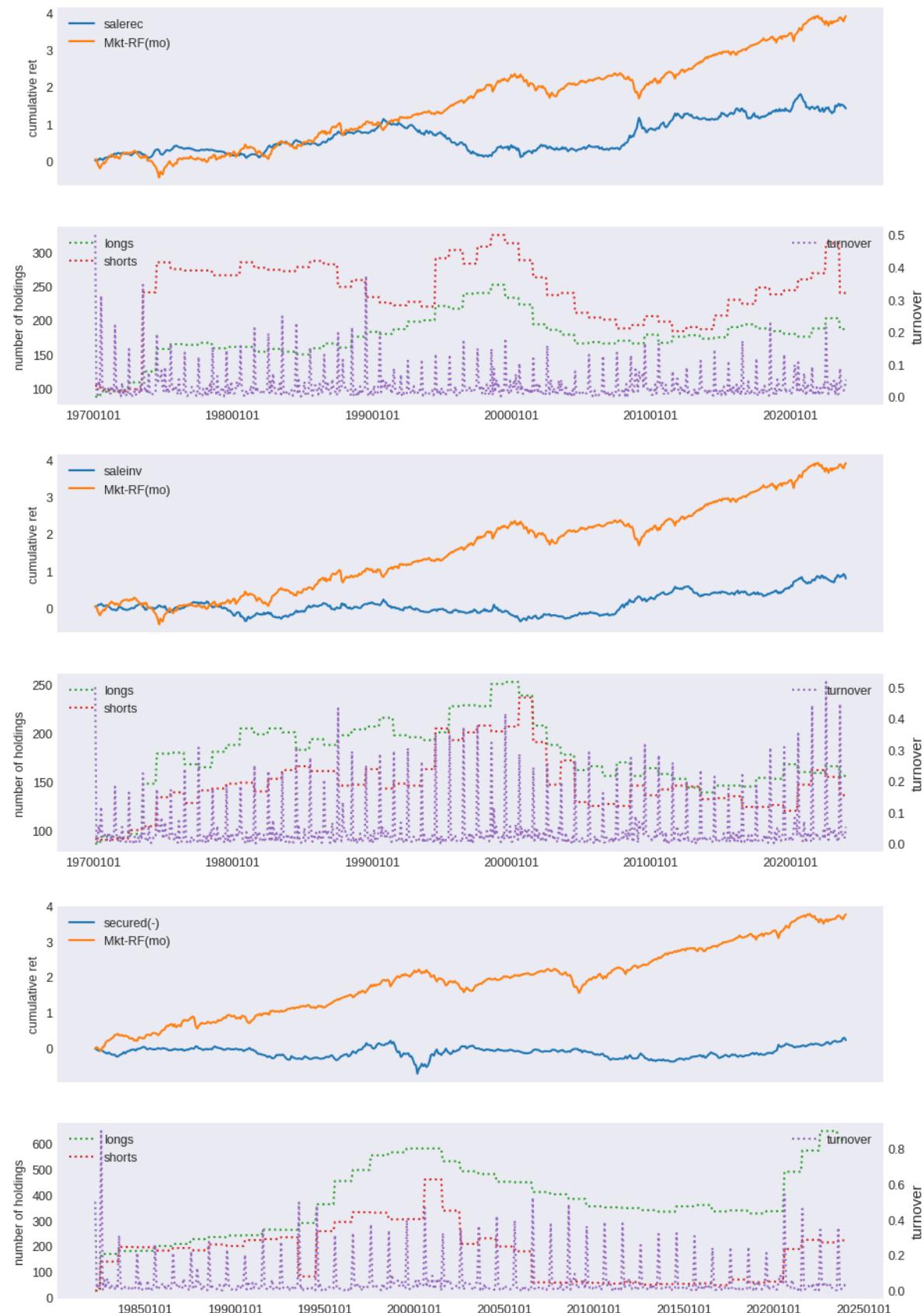


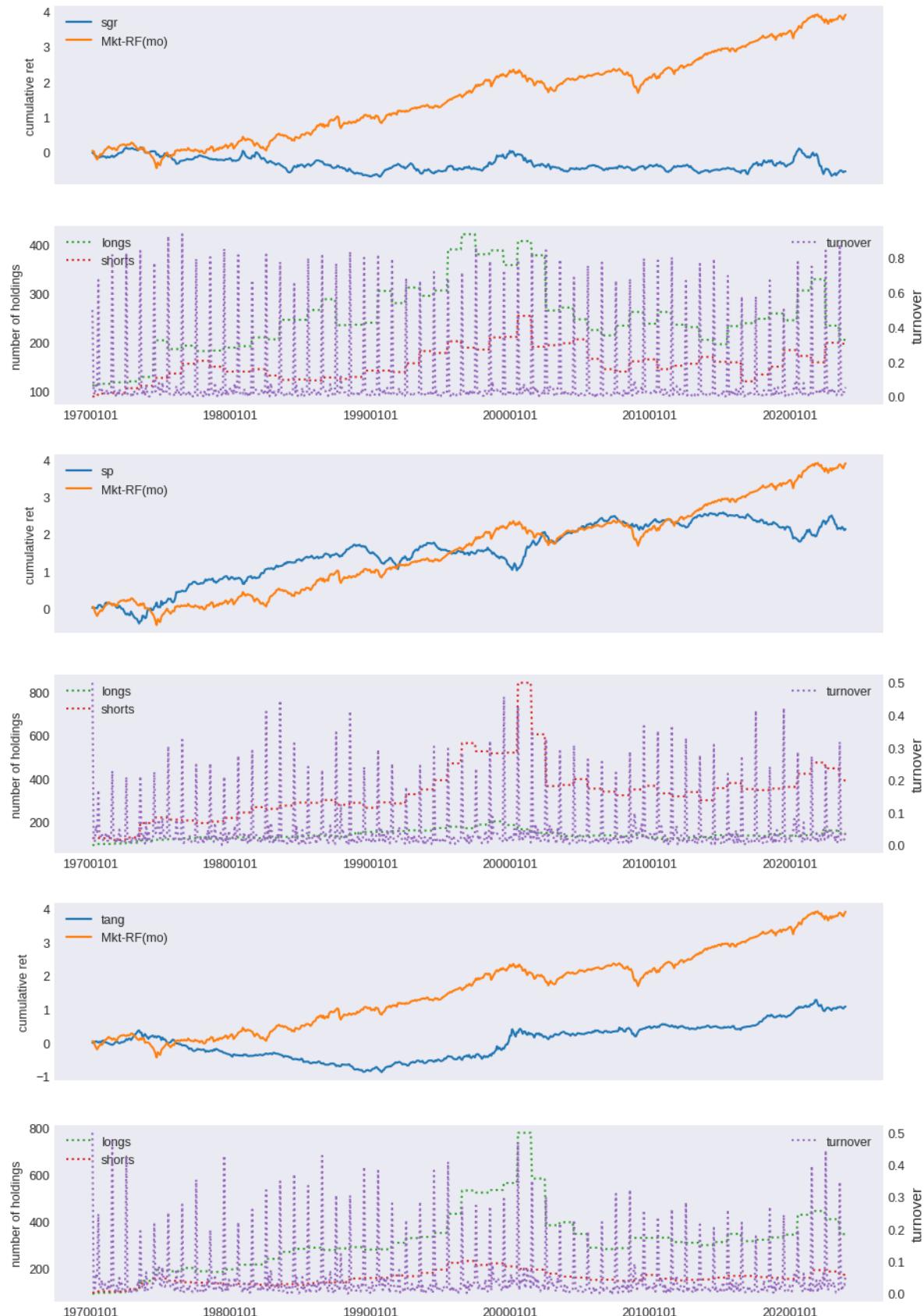




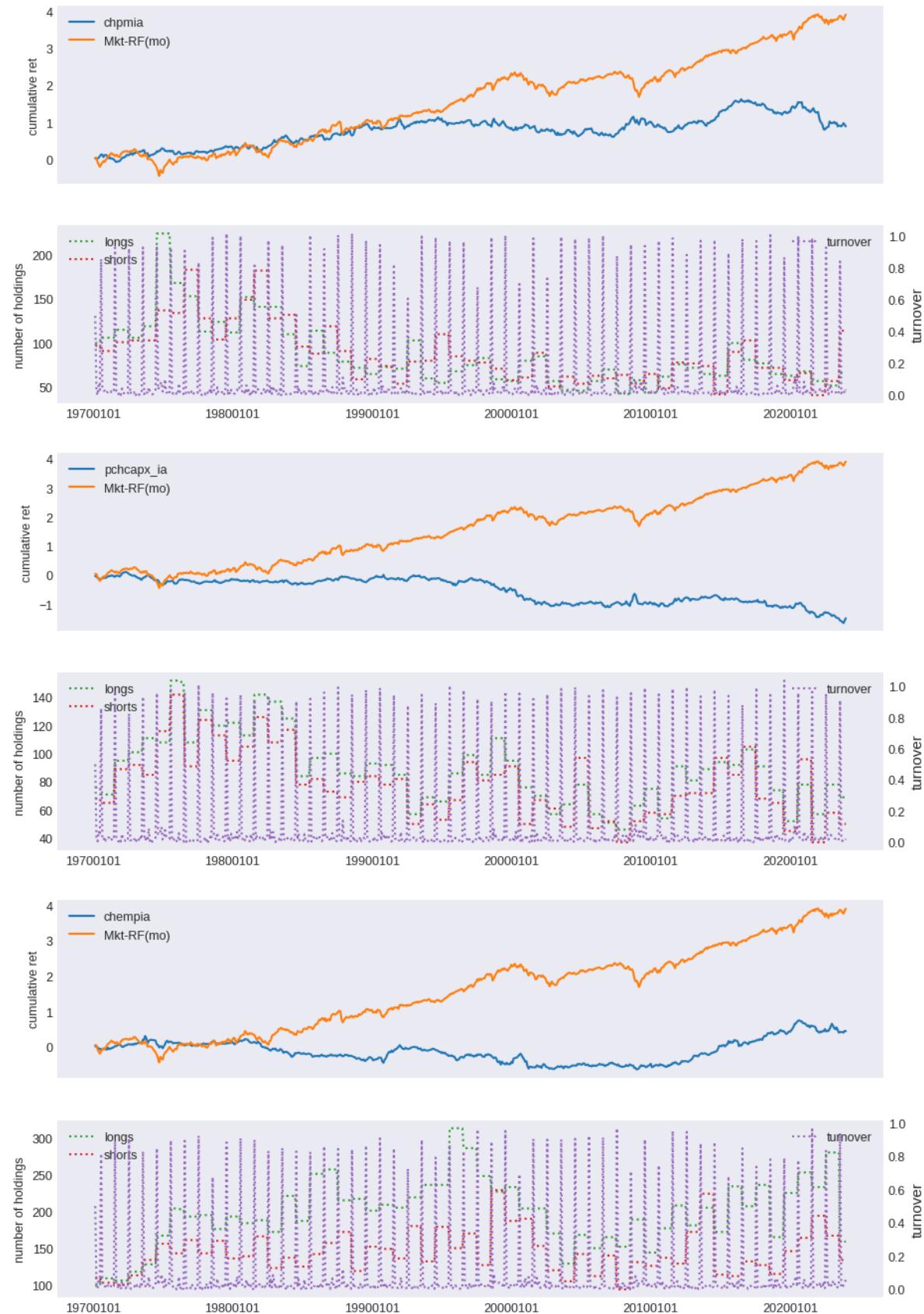














Fundamental signals from Compustat Quarterly

```
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'])\n    .drop_duplicates(['permno', 'datadate'])\n    .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

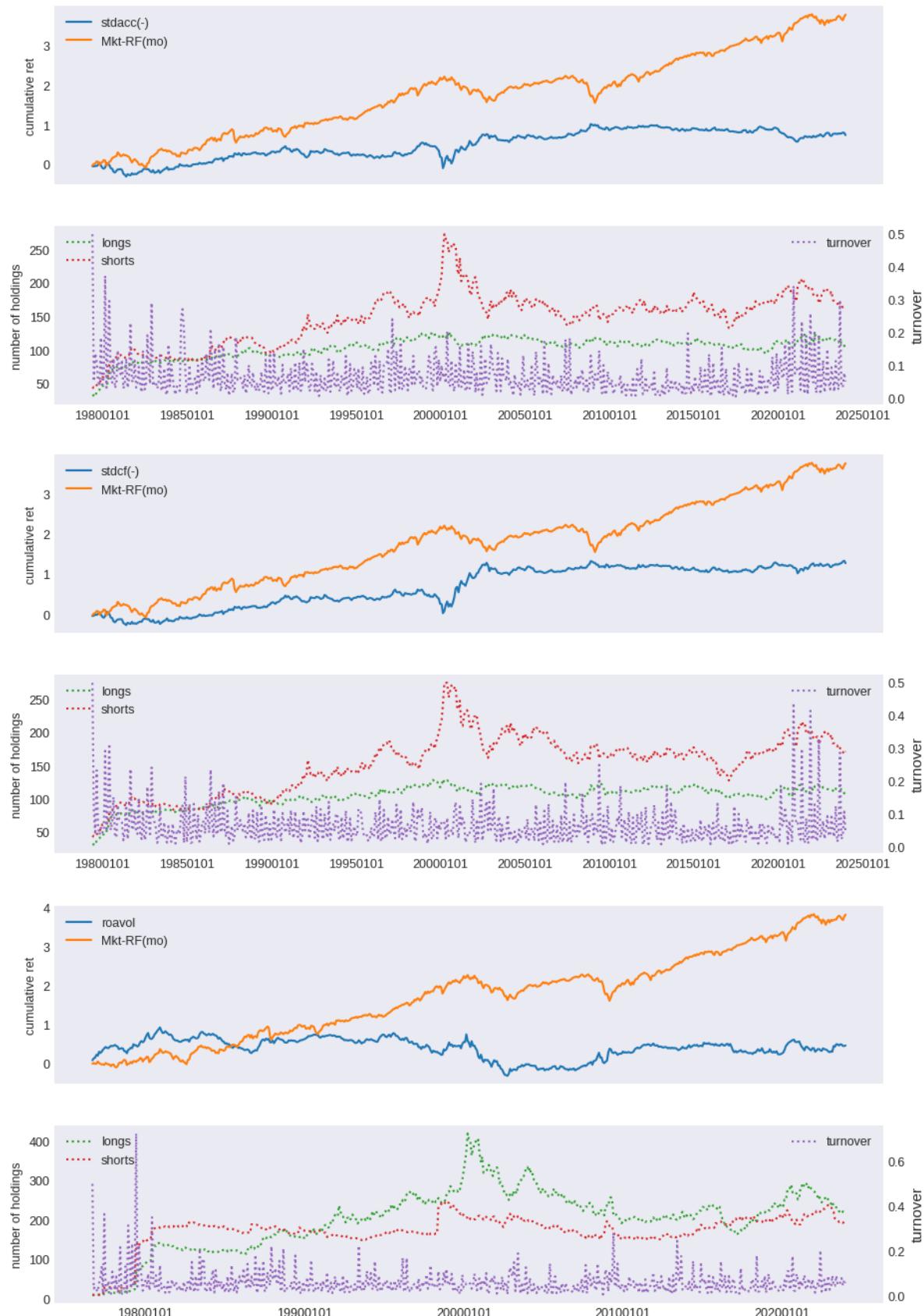
# compute signals from the 15 lags
fund['rebalddate'] = rebalddate
fund['stdacc'] = lags['sacc'].std(axis=1, skipna=False)
fund['stdcf'] = lags['scf'].std(axis=1, skipna=False)
fund['roavol'] = lags['roaq'].std(axis=1, skipna=False)
fund['sgrvol'] = lags['rsup'].std(axis=1, skipna=False)
fund['cinvest'] = (fund['cinvest'] -
                    lags['cinvest'][[1, 2, 3, 4]].mean(axis=1, skipna=False))

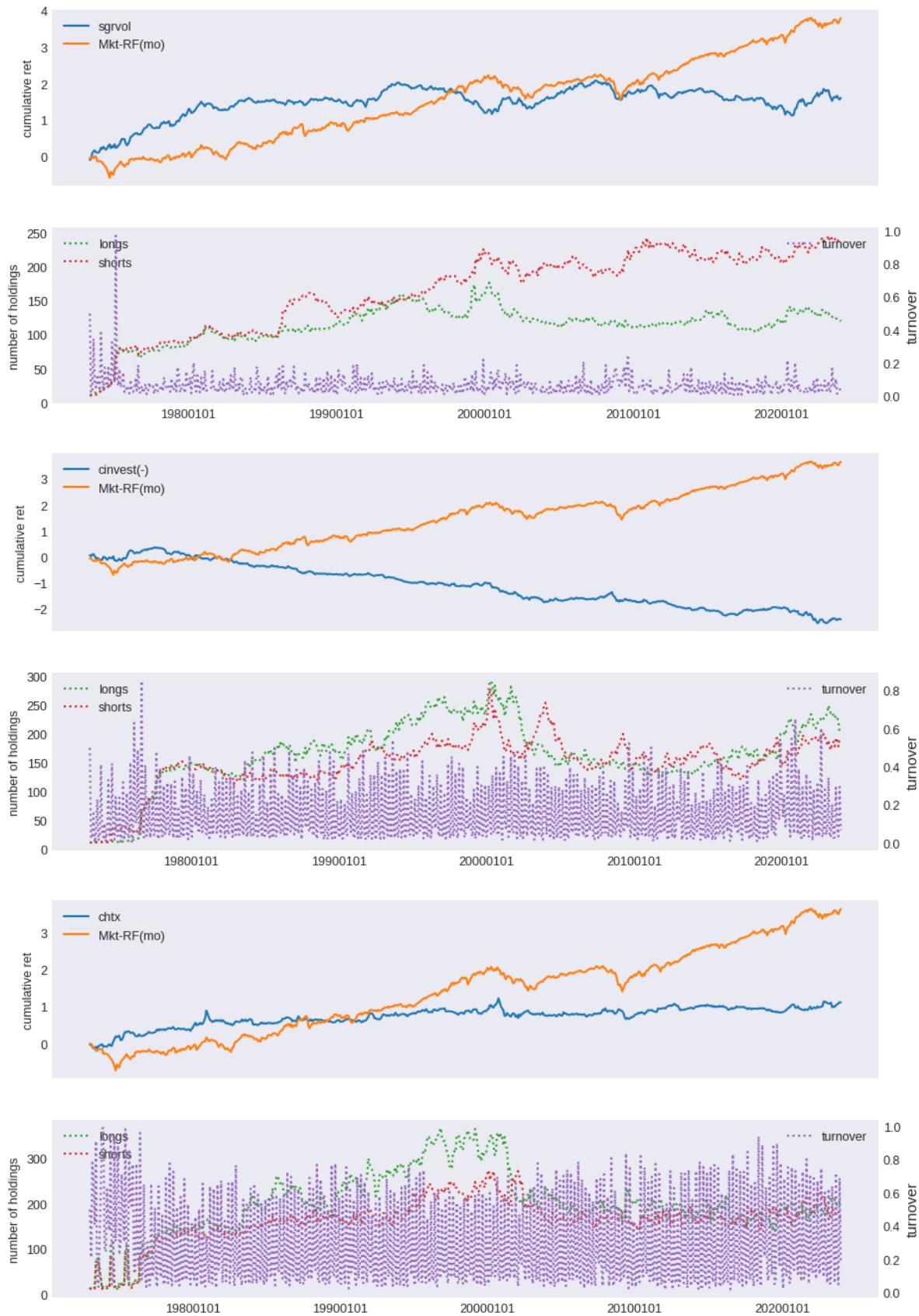
# count number of consecutive increasing quarters
fund['nincr'] = lags['nincr'][np.arange(8)].sum(axis=1)
```

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

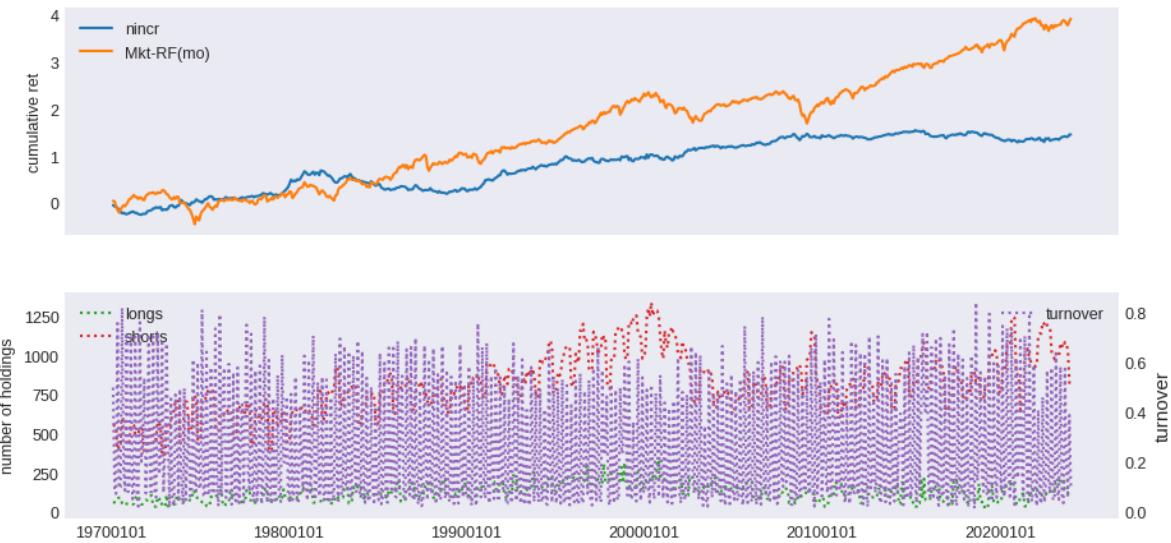
```
rebalbeg, rebalend = 19700101, LAST_DATE
benchnames = ['Mkt-RF (mo)']
for label in tqdm(columns):
    holdings = univariate_sorts(monthly,
                                 label,
                                 SignalsFrame(signals.read(label)),
                                 rebalbeg,
                                 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 [08:33<00:00, 51.36s/it]









6.1.6 Earnings Estimates

IBES Fiscal Year 1 signals

```
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['rebalance'] = 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)
```

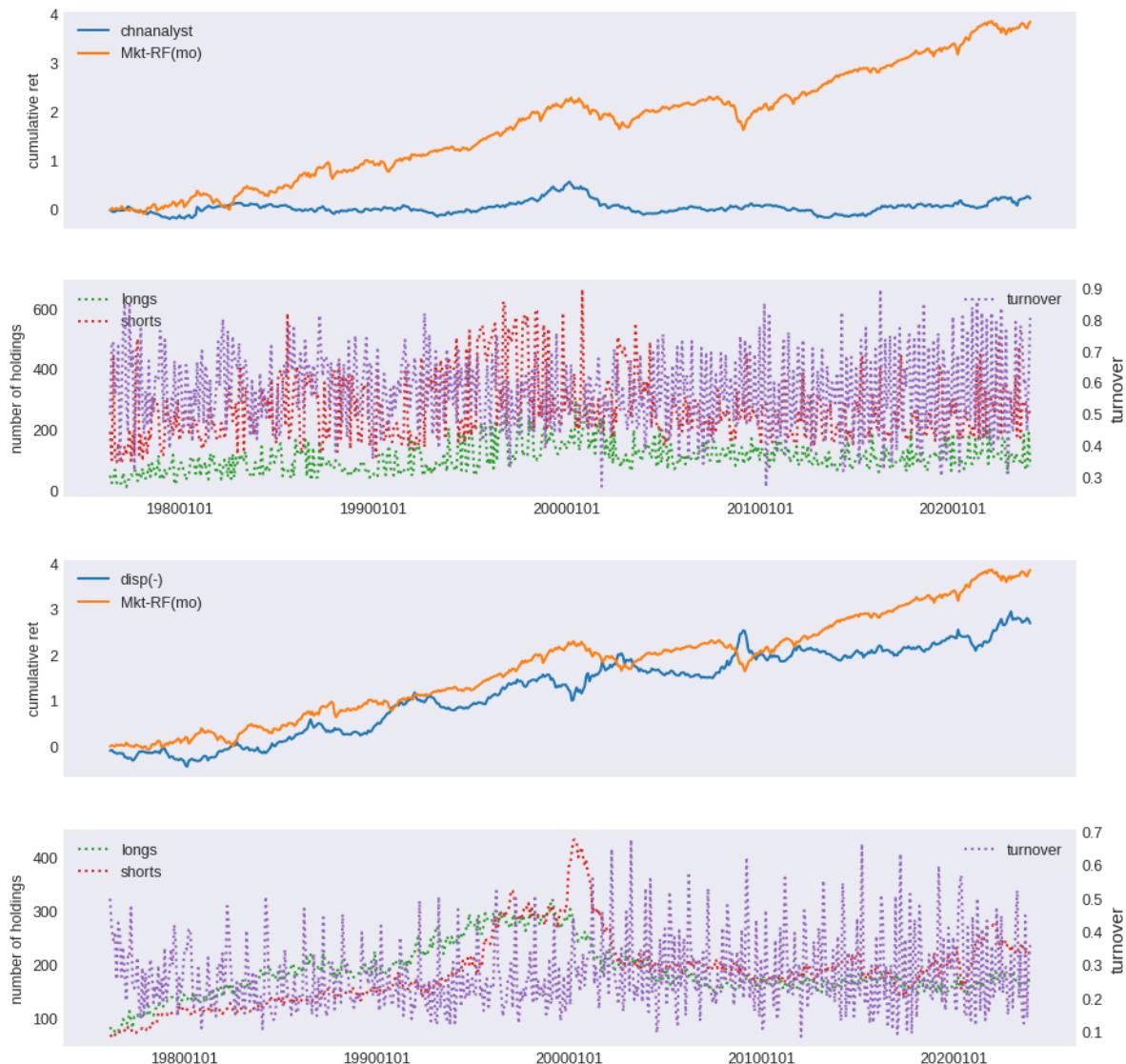
```

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 [02:45<00:00, 55.09s/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)
```

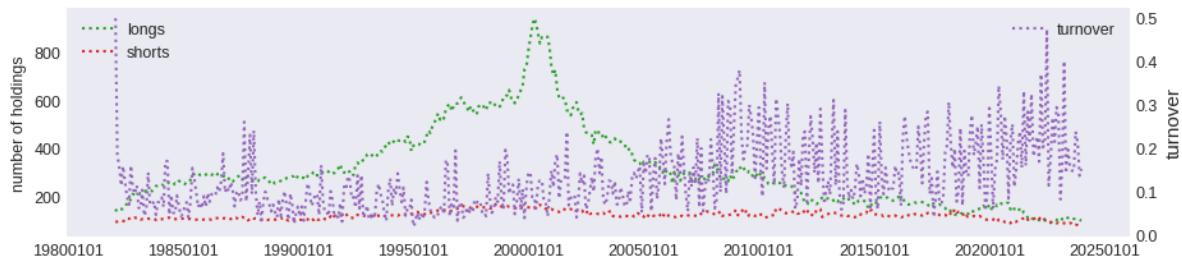
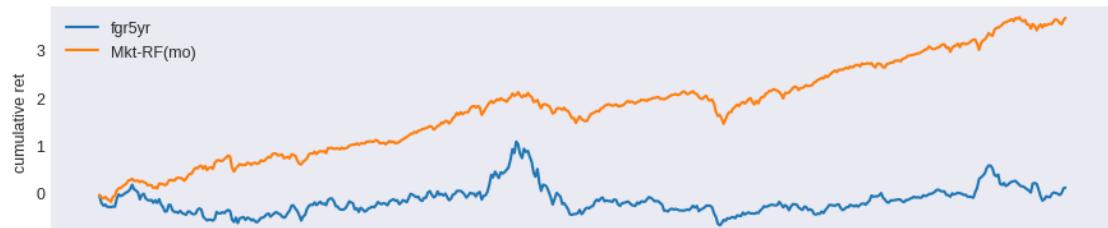
1311079

```

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 [00:46<00:00, 46.86s/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)
```

949558

```
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)

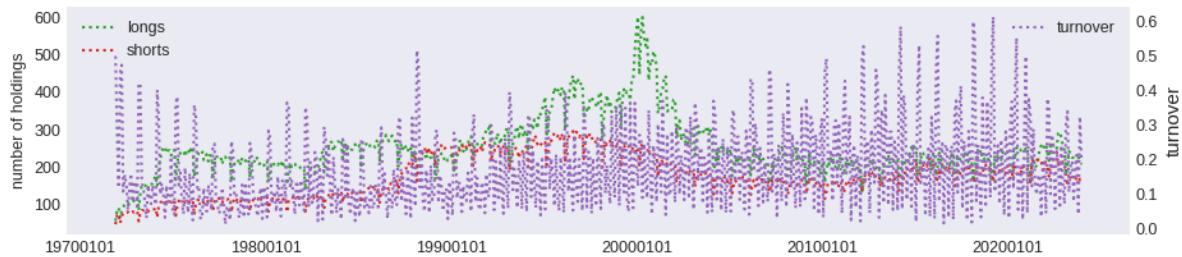
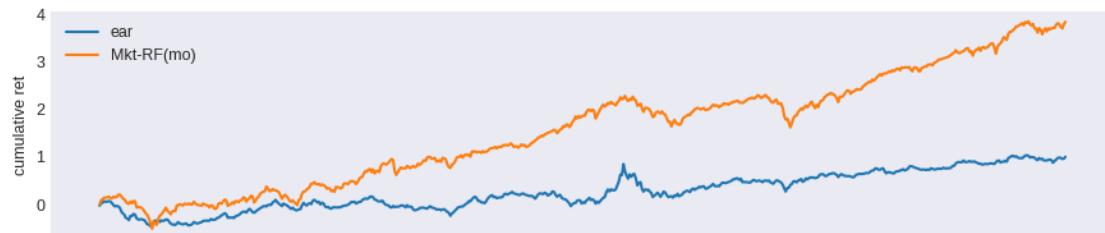
```
holdings,
label,
benchnames,
overlap=0,
outdir=outdir,
suffix=(leverage.get(label, 1) < 0) * '(-)'
```

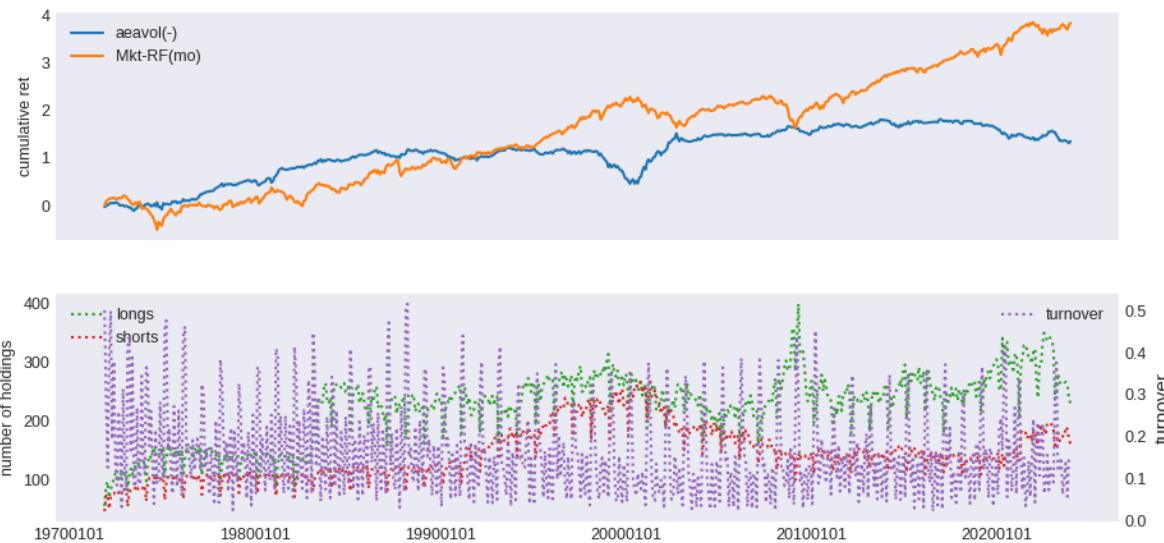
50% |██████████| 1/2 [02:36<02:36, 156.72s/it]

| | excess | sharpe | alpha | appraisal | welch-t | welch-p | turnover | longs | shorts |
|---------|--------|--------|--------|-----------|---------|---------|----------|---------|--------|
| ↳ buys | 0.019 | 0.167 | -0.004 | -0.038 | 0.866 | 0.387 | 2.098 | 249.304 | 176.47 |
| ↳ sells | 2.09 | 2.107 | | | | | | | |
| ear | 0.019 | 0.167 | -0.004 | -0.038 | 0.866 | 0.387 | 2.098 | 249.304 | 176.47 |
| ↳ | 2.09 | 2.107 | | | | | | | |

100% |██████████| 2/2 [05:49<00:00, 174.92s/it]

| | excess | sharpe | alpha | appraisal | welch-t | welch-p | turnover | longs | shorts |
|-----------|--------|--------|-------|-----------|---------|---------|----------|---------|--------|
| ↳ shorts | 0.026 | 0.235 | 0.04 | 0.38 | -0.876 | 0.381 | 1.943 | 222.169 | |
| ↳ buys | 1.935 | 1.951 | | | | | | | |
| aeavol(-) | 0.026 | 0.235 | 0.04 | 0.38 | -0.876 | 0.381 | 1.943 | 222.169 | |
| ↳ | 1.935 | 1.951 | | | | | | | |





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',
                      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['rebalance'] = 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, rebaldate]
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) * '(-)')
```

| | excess | sharpe | alpha | appraisal | welch-t | welch-p | turnover | longs | ... |
|----------|--------|--------|-------|-----------|---------|---------|----------|---------|------|
| ↳ shorts | | | | | | | | | ... |
| ↳ buys | | | | | | | | | ... |
| ↳ sells | | | | | | | | | ... |
| sfe | 0.052 | 0.285 | 0.081 | 0.462 | -0.034 | 0.973 | 2.126 | 148.671 | 297. |
| ↳ 957 | 2.12 | 2.132 | | | | | | | |



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'])
```

(continues on next page)

(continued from previous page)

```

summ = summ.join(hist[['price']], how='left')
summ = summ.reset_index() \
    .set_index(['permno', 'rebaldate']) \
    .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)

```

1069502

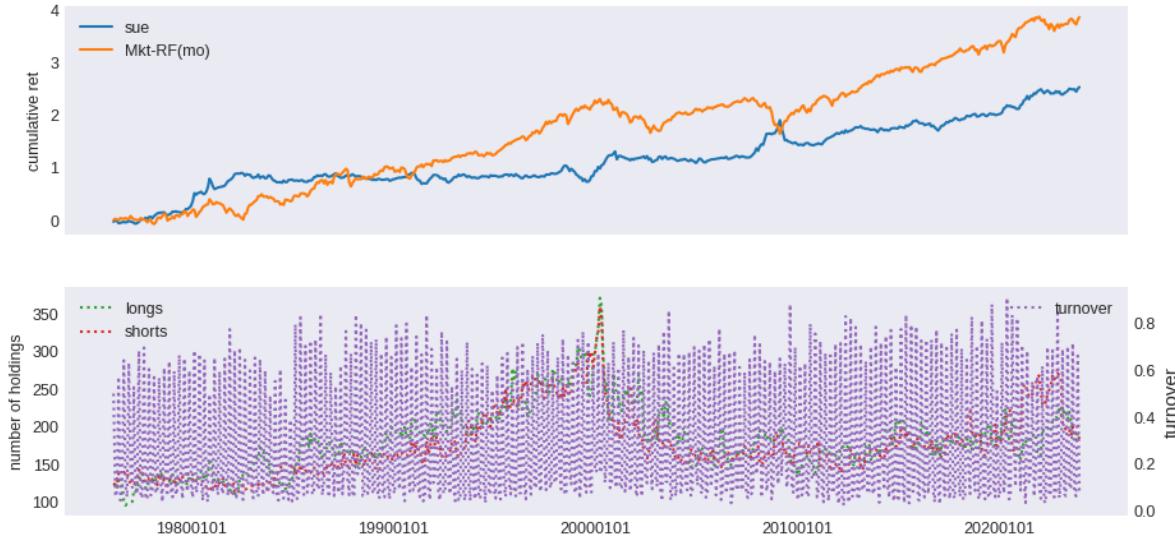
```

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 [04:15<00:00, 255.90s/it]

| | excess | sharpe | alpha | appraisal | welch-t | welch-p | turnover | longs | shorts |
|---------|--------|--------|-------|-----------|---------|---------|----------|--------|---------|
| ↳ buys | 0.053 | 0.479 | 0.057 | 0.518 | 0.512 | 0.609 | 3.436 | 183.05 | 177.005 |
| ↳ sells | 3.422 | 3.449 | | | | | | | |



6.1.7 Backtests

The backtests are all of univariate dollar-neutral spreads between the top and bottom deciles of the characteristic. Stocks are value-weighted within each decile, and exclude firms in the smallest quintile of NYSE market capitalization. Spread portfolios are rebalanced monthly, with fundamental data lagged in the usual manner (six months for annual, 4 months for quarterly). In addition to the time series of cumulative spread portfolio and market index returns, the monthly turnover rate and number of long and short positions are also graphed.

The summary of backtest results is sorted by Welch's t-value testing for difference in pre-2002 and post-2003 mean returns, as Green et al (2017) and others have also found a sharp drop in return predictability of many these factors then.

The **maximum drawdown** is also computed for each strategy – this is defined as the maximum loss from a peak to a trough of the cumulative returns of a strategy, before a new peak is attained. It is a historical measure of a worse case analysis.

```
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
```

(continues on next page)

(continued from previous page)

```

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')

```

| | Sharpe Ratio | Alpha | Appraisal Ratio | Avg Ret | Vol | Welch-t | \ |
|-------------|---------------|---------|-----------------|---------|--------|---------|---|
| chcsho (-) | 0.6159 | 0.0786 | 0.8231 | 0.0052 | 0.0294 | -0.9003 | |
| sue | 0.4788 | 0.0570 | 0.5179 | 0.0044 | 0.0318 | 0.5115 | |
| mom12m | 0.4557 | 0.1493 | 0.6395 | 0.0098 | 0.0732 | -1.4424 | |
| chfeps | 0.4380 | 0.0619 | 0.5166 | 0.0044 | 0.0350 | -0.7879 | |
| mom1m (-) | 0.4002 | 0.0597 | 0.3306 | 0.0064 | 0.0541 | -2.4864 | |
| ... | ... | ... | ... | ... | ... | ... | |
| pchquick | -0.1594 | -0.0149 | -0.1689 | -0.0012 | 0.0254 | -0.0714 | |
| cfp_ia | -0.1921 | -0.0353 | -0.2600 | -0.0022 | 0.0396 | -0.3738 | |
| pchcapx_ia | -0.2251 | -0.0287 | -0.2356 | -0.0023 | 0.0352 | 0.1500 | |
| grcapx | -0.2910 | -0.0412 | -0.4101 | -0.0025 | 0.0299 | 1.5998 | |
| cinvest (-) | -0.4001 | -0.0445 | -0.3788 | -0.0039 | 0.0340 | 0.0619 | |
| | Appraisal2002 | Ret2002 | Drawdown | | | | |
| chcsho (-) | 0.7323 | 0.0040 | -0.2720 | | | | |
| sue | 0.7110 | 0.0051 | -0.3999 | | | | |
| mom12m | 0.4489 | 0.0043 | -0.9670 | | | | |
| chfeps | 0.5138 | 0.0032 | -0.3299 | | | | |
| mom1m (-) | -0.2796 | -0.0009 | -0.7172 | | | | |
| ... | ... | ... | ... | | | | |
| pchquick | -0.2631 | -0.0013 | -0.7598 | | | | |
| cfp_ia | -0.4263 | -0.0029 | -0.8612 | | | | |
| pchcapx_ia | -0.1287 | -0.0020 | -0.8845 | | | | |
| grcapx | -0.1046 | -0.0002 | -0.8722 | | | | |
| cinvest (-) | -0.3585 | -0.0038 | -0.9590 | | | | |

[82 rows x 9 columns]

```

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 | \ |
|------------|--------------|--------|-----------------|---------|--------|---|
| chcsho (-) | 0.6159 | 0.0786 | 0.8231 | 0.0052 | 0.0294 | |
| sue | 0.4788 | 0.0570 | 0.5179 | 0.0044 | 0.0318 | |
| mom12m | 0.4557 | 0.1493 | 0.6395 | 0.0098 | 0.0732 | |

(continues on next page)

(continued from previous page)

| | | | | | |
|-----------------|--------|---------|---------|--------|--------|
| chfeps | 0.4380 | 0.0619 | 0.5166 | 0.0044 | 0.0350 |
| mom1m(-) | 0.4002 | 0.0597 | 0.3306 | 0.0064 | 0.0541 |
| chmom | 0.3990 | 0.0504 | 0.3097 | 0.0058 | 0.0492 |
| egr(-) | 0.3805 | 0.0599 | 0.5707 | 0.0035 | 0.0322 |
| chinv(-) | 0.3699 | 0.0513 | 0.4865 | 0.0033 | 0.0313 |
| indmom | 0.3590 | 0.0657 | 0.3781 | 0.0054 | 0.0513 |
| acc(-) | 0.3479 | 0.0386 | 0.3612 | 0.0031 | 0.0309 |
| bm_ia | 0.3231 | 0.0317 | 0.2402 | 0.0036 | 0.0389 |
| disp(-) | 0.3164 | 0.1002 | 0.6424 | 0.0047 | 0.0513 |
| ill(-) | 0.3151 | 0.0471 | 0.4007 | 0.0031 | 0.0343 |
| agr(-) | 0.3147 | 0.0546 | 0.4967 | 0.0031 | 0.0337 |
| nincr | 0.3075 | 0.0209 | 0.2409 | 0.0023 | 0.0254 |
| mom6m | 0.3058 | 0.1056 | 0.4872 | 0.0062 | 0.0683 |
| cashpr(-) | 0.3037 | 0.0484 | 0.3915 | 0.0032 | 0.0363 |
| roaq | 0.2944 | 0.0619 | 0.4443 | 0.0036 | 0.0421 |
| sfe | 0.2846 | 0.0808 | 0.4621 | 0.0044 | 0.0529 |
| ep | 0.2821 | 0.0633 | 0.4181 | 0.0037 | 0.0454 |
| sp | 0.2693 | 0.0386 | 0.2617 | 0.0033 | 0.0426 |
| invest(-) | 0.2677 | 0.0466 | 0.3954 | 0.0027 | 0.0352 |
| cfp | 0.2519 | 0.0516 | 0.3719 | 0.0030 | 0.0413 |
| pchsale_pchinvt | 0.2508 | 0.0237 | 0.2483 | 0.0020 | 0.0276 |
| pchsaleinv | 0.2475 | 0.0202 | 0.2302 | 0.0018 | 0.0253 |
| bm | 0.2459 | 0.0374 | 0.2371 | 0.0032 | 0.0456 |
| cash | 0.2429 | 0.0101 | 0.0724 | 0.0031 | 0.0436 |
| aeavol(-) | 0.2353 | 0.0400 | 0.3799 | 0.0022 | 0.0316 |
| maxret(-) | 0.2224 | 0.1211 | 0.6779 | 0.0041 | 0.0645 |
| chatoia | 0.2183 | 0.0240 | 0.2564 | 0.0017 | 0.0272 |
| stdcf(-) | 0.2100 | 0.0601 | 0.4901 | 0.0024 | 0.0392 |
| sgrvol | 0.2094 | 0.0114 | 0.0786 | 0.0027 | 0.0438 |
| salerec | 0.2025 | 0.0403 | 0.3190 | 0.0022 | 0.0376 |
| mom36m(-) | 0.1857 | 0.0211 | 0.1054 | 0.0033 | 0.0599 |
| chtx | 0.1811 | 0.0166 | 0.1379 | 0.0018 | 0.0350 |
| grltnoa | 0.1718 | 0.0221 | 0.1898 | 0.0017 | 0.0336 |
| tang | 0.1679 | 0.0076 | 0.0653 | 0.0017 | 0.0346 |
| ear | 0.1671 | -0.0039 | -0.0380 | 0.0016 | 0.0330 |
| mve_ia(-) | 0.1580 | 0.0056 | 0.0545 | 0.0014 | 0.0303 |
| rsup | 0.1577 | 0.0227 | 0.1763 | 0.0017 | 0.0372 |
| retvol(-) | 0.1532 | 0.1307 | 0.6220 | 0.0034 | 0.0776 |
| rd_mve | 0.1462 | 0.0125 | 0.0784 | 0.0020 | 0.0464 |
| saleinv | 0.1383 | 0.0356 | 0.3665 | 0.0012 | 0.0310 |
| stdacc(-) | 0.1344 | 0.0419 | 0.3665 | 0.0014 | 0.0357 |
| zerotrade(-) | 0.1340 | -0.0399 | -0.2231 | 0.0025 | 0.0639 |
| pctacc(-) | 0.1274 | 0.0256 | 0.2437 | 0.0012 | 0.0313 |
| chpmia | 0.1241 | 0.0174 | 0.1297 | 0.0014 | 0.0387 |
| turn | 0.1069 | -0.0467 | -0.2504 | 0.0020 | 0.0663 |
| pricedelay(-) | 0.1028 | 0.0051 | 0.0522 | 0.0009 | 0.0292 |
| gma | 0.0923 | 0.0171 | 0.1177 | 0.0011 | 0.0422 |
| realestate | 0.0845 | 0.0461 | 0.3114 | 0.0011 | 0.0459 |
| dolvol | 0.0737 | -0.0028 | -0.0271 | 0.0007 | 0.0309 |
| chempia | 0.0717 | -0.0055 | -0.0495 | 0.0007 | 0.0331 |
| beta | 0.0655 | -0.0764 | -0.3973 | 0.0017 | 0.0906 |
| roavol | 0.0646 | -0.0077 | -0.0520 | 0.0008 | 0.0438 |
| absacc(-) | 0.0628 | 0.0242 | 0.1743 | 0.0007 | 0.0413 |
| chnanalyst | 0.0550 | -0.0006 | -0.0068 | 0.0004 | 0.0252 |
| depr | 0.0526 | -0.0221 | -0.1692 | 0.0006 | 0.0423 |
| pchgm_pchsale | 0.0480 | 0.0003 | 0.0027 | 0.0004 | 0.0296 |

(continues on next page)

(continued from previous page)

| | | | | | |
|-----------------|---------|---------|---------|---------|--------|
| std_turn | 0.0408 | -0.0501 | -0.2939 | 0.0007 | 0.0585 |
| pchdepr | 0.0390 | 0.0031 | 0.0335 | 0.0003 | 0.0266 |
| secured(-) | 0.0389 | 0.0334 | 0.2773 | 0.0004 | 0.0375 |
| salecash | 0.0254 | 0.0206 | 0.1831 | 0.0003 | 0.0343 |
| lev | 0.0237 | -0.0003 | -0.0019 | 0.0003 | 0.0487 |
| quick | 0.0197 | -0.0297 | -0.2071 | 0.0003 | 0.0464 |
| fgr5yr | 0.0125 | -0.0565 | -0.2931 | 0.0002 | 0.0635 |
| std_dolvol | 0.0120 | 0.0040 | 0.0366 | 0.0001 | 0.0317 |
| idiovol(-) | -0.0038 | 0.0525 | 0.2693 | -0.0001 | 0.0695 |
| pchsale_pchrect | -0.0393 | -0.0056 | -0.0657 | -0.0003 | 0.0246 |
| dy | -0.0400 | 0.0386 | 0.2284 | -0.0007 | 0.0571 |
| divyld | -0.0430 | 0.0379 | 0.2468 | -0.0007 | 0.0552 |
| rd_sale | -0.0575 | -0.0152 | -0.0986 | -0.0007 | 0.0447 |
| sgr | -0.0791 | -0.0255 | -0.2109 | -0.0008 | 0.0362 |
| baspread | -0.1047 | -0.1260 | -0.5936 | -0.0024 | 0.0806 |
| pchsale_pchxsga | -0.1133 | -0.0125 | -0.1141 | -0.0010 | 0.0317 |
| hire | -0.1167 | -0.0314 | -0.2724 | -0.0012 | 0.0350 |
| lgr | -0.1272 | -0.0219 | -0.2415 | -0.0010 | 0.0270 |
| pchquick | -0.1594 | -0.0149 | -0.1689 | -0.0012 | 0.0254 |
| cfp_ia | -0.1921 | -0.0353 | -0.2600 | -0.0022 | 0.0396 |
| pchcapx_ia | -0.2251 | -0.0287 | -0.2356 | -0.0023 | 0.0352 |
| grcapx | -0.2910 | -0.0412 | -0.4101 | -0.0025 | 0.0299 |
| cinvest(-) | -0.4001 | -0.0445 | -0.3788 | -0.0039 | 0.0340 |

| | Welch-t | Appraisal2002 | Ret2002 | Drawdown |
|----------------|---------|---------------|---------|----------|
| chcsho(-) | -0.9003 | 0.7323 | 0.0040 | -0.2720 |
| sue | 0.5115 | 0.7110 | 0.0051 | -0.3999 |
| mom12m | -1.4424 | 0.4489 | 0.0043 | -0.9670 |
| chfeps | -0.7879 | 0.5138 | 0.0032 | -0.3299 |
| mom1m(-) | -2.4864 | -0.2796 | -0.0009 | -0.7172 |
| chmom | -2.0332 | -0.0982 | 0.0006 | -0.6133 |
| egr(-) | -0.6843 | 0.4828 | 0.0026 | -0.3384 |
| chinv(-) | -0.6077 | 0.3094 | 0.0024 | -0.2747 |
| indmom | -1.2340 | 0.2213 | 0.0023 | -0.6892 |
| acc(-) | -1.2949 | 0.0954 | 0.0012 | -0.3966 |
| bm_ia | -1.5394 | -0.0016 | 0.0009 | -0.6322 |
| disp(-) | -0.3845 | 0.7475 | 0.0038 | -0.5288 |
| ill(-) | -0.9953 | 0.3609 | 0.0017 | -0.6150 |
| agr(-) | -1.6115 | 0.1517 | 0.0005 | -0.4195 |
| nincr | -0.4708 | 0.2820 | 0.0017 | -0.4062 |
| mom6m | -0.3593 | 0.4688 | 0.0049 | -0.9885 |
| cashpr(-) | -0.9925 | 0.1124 | 0.0015 | -0.4252 |
| roaq | 1.0678 | 0.7910 | 0.0057 | -0.4532 |
| sfe | -0.0340 | 0.3968 | 0.0043 | -0.7412 |
| ep | -1.4876 | 0.2047 | 0.0006 | -0.6625 |
| sp | -0.9629 | 0.0534 | 0.0014 | -0.5642 |
| invest(-) | -1.2757 | 0.1567 | 0.0004 | -0.4267 |
| cfp | -0.2572 | 0.2313 | 0.0025 | -0.5696 |
| pchsale_pchinv | -2.0639 | -0.1278 | -0.0008 | -0.4948 |
| pchsaleinv | -1.7397 | -0.1256 | -0.0003 | -0.4893 |
| bm | -2.1594 | -0.2596 | -0.0014 | -0.7508 |
| cash | -0.2386 | 0.0028 | 0.0026 | -0.7074 |
| aeavol(-) | -0.8760 | 0.2552 | 0.0009 | -0.5583 |
| maxret(-) | -0.3880 | 0.6524 | 0.0031 | -0.7173 |
| chatoia | -1.2026 | 0.0484 | 0.0002 | -0.5343 |
| stdcf(-) | -0.5745 | 0.4072 | 0.0014 | -0.4787 |

(continues on next page)

(continued from previous page)

| | | | | |
|-----------------|---------|---------|---------|---------|
| sgrvol | -1.0245 | -0.2327 | 0.0005 | -0.6922 |
| salerec | 0.9534 | 0.5308 | 0.0040 | -0.6761 |
| mom36m(-) | -0.9490 | -0.0294 | 0.0004 | -0.7287 |
| chtx | -0.3338 | 0.2143 | 0.0013 | -0.4785 |
| grltnoa | -0.0995 | 0.1927 | 0.0015 | -0.4941 |
| tang | 0.9173 | 0.1566 | 0.0031 | -0.7361 |
| ear | 0.8660 | 0.2133 | 0.0029 | -0.5425 |
| mve_ia(-) | -0.3058 | 0.0120 | 0.0010 | -0.5842 |
| rsup | 0.8337 | 0.4375 | 0.0032 | -0.5391 |
| retvol(-) | -0.5153 | 0.5356 | 0.0018 | -0.7990 |
| rd_mve | 0.3087 | 0.1094 | 0.0026 | -0.7868 |
| saleinv | 1.9240 | 0.6342 | 0.0041 | -0.4900 |
| stdacc(-) | -0.2995 | 0.3049 | 0.0009 | -0.5056 |
| zerotrade(-) | 0.7004 | -0.1248 | 0.0043 | -0.8344 |
| pctacc(-) | 0.7131 | 0.2858 | 0.0022 | -0.3803 |
| chpmia | -0.4053 | 0.0081 | 0.0006 | -0.5937 |
| turn | 0.8012 | -0.1345 | 0.0043 | -0.8291 |
| pricedelay(-) | -0.2216 | -0.0522 | 0.0005 | -0.6159 |
| gma | 1.1049 | 0.4025 | 0.0034 | -0.6364 |
| realestate | -1.2871 | 0.1600 | -0.0014 | -0.6503 |
| dolvol | -0.3467 | 0.0515 | 0.0002 | -0.7621 |
| chempia | 1.9964 | 0.3559 | 0.0038 | -0.6884 |
| beta | -0.4256 | -0.5622 | -0.0001 | -0.9588 |
| roavol | 0.3854 | 0.0791 | 0.0016 | -0.7821 |
| absacc(-) | -1.4650 | -0.1357 | -0.0021 | -0.7729 |
| chnanalyst | 0.0380 | 0.0748 | 0.0004 | -0.5443 |
| depr | 0.7273 | 0.0109 | 0.0020 | -0.6522 |
| pchgm_pchsale | 0.1187 | 0.0961 | 0.0006 | -0.5638 |
| std_turn | 0.5724 | -0.2378 | 0.0021 | -0.8064 |
| pchdepr | -1.5397 | -0.2029 | -0.0016 | -0.6991 |
| secured(-) | 0.6495 | 0.4016 | 0.0015 | -0.6406 |
| salecash | 0.2888 | 0.2496 | 0.0007 | -0.7463 |
| lev | -1.7371 | -0.3459 | -0.0038 | -0.8697 |
| quick | 0.8571 | -0.1036 | 0.0021 | -0.7923 |
| fgr5yr | 0.1193 | -0.0965 | 0.0006 | -0.8761 |
| std_dolvol | 0.9647 | 0.1411 | 0.0014 | -0.5107 |
| idiovol(-) | -0.3467 | 0.3555 | -0.0014 | -0.9876 |
| pchsale_pchrect | -2.6701 | -0.5153 | -0.0034 | -0.7873 |
| dy | -0.6572 | -0.0302 | -0.0023 | -0.9186 |
| divyld | -0.9423 | -0.1059 | -0.0031 | -0.9645 |
| rd_sale | 0.5829 | -0.0405 | 0.0004 | -0.8706 |
| sgr | 0.0733 | -0.1395 | -0.0007 | -0.7032 |
| baspread | 0.3600 | -0.5421 | -0.0012 | -0.9522 |
| pchsale_pchxsga | -0.8570 | -0.2770 | -0.0023 | -0.7742 |
| hire | 1.5073 | 0.0525 | 0.0014 | -0.8262 |
| lgr | 1.7493 | 0.1136 | 0.0013 | -0.8286 |
| pchquick | -0.0714 | -0.2631 | -0.0013 | -0.7598 |
| cfp_ia | -0.3738 | -0.4263 | -0.0029 | -0.8612 |
| pchcapx_ia | 0.1500 | -0.1287 | -0.0020 | -0.8845 |
| grcapx | 1.5998 | -0.1046 | -0.0002 | -0.8722 |
| cinvest(-) | 0.0619 | -0.3585 | -0.0038 | -0.9590 |

```

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

The historical backtests of portfolio returns are considered as the features for cluster analysis, which can be used to identify and construct peer benchmarks for each strategy. Strategies whose returns move up or down together tend to load on similar “style” factors, and should be evaluated against each other.

The correlation between two series of returns is closely related to the Euclidean distance between the series. When the returns are standardized to have unit variance, the two metrics can be derived exactly from each other. Recall that the squared Euclidean norm between two standardized series x and y is:

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

since $\sum_i x_i^2 = \sum_i y_i^2 = n$, and $\rho = \sum_i x_i y_i / n$.

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

6.2.1 Hierarchical clustering

This builds a hierarchy of clusters by iteratively merging clusters based on their linkage similarity. Linkage methods determine how the distance between clusters is measured and how clusters are merged. Linkage methods include:

- Single Linkage: Measures the distance between the closest points of two clusters and merges the clusters with the smallest distance. It tends to produce elongated clusters.
- Complete Linkage: Measures the distance between the farthest points of two clusters and merges the clusters with the smallest maximum distance. It tends to produce compact, spherical clusters.
- Average Linkage: Measures the average distance between all pairs of points in two clusters and merges the clusters with the smallest average distance. It balances between single and complete linkage, often producing balanced clusters.
- Ward’s Method: Minimizes the within-cluster variance when merging clusters. It tends to produce clusters with similar sizes and shapes.

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

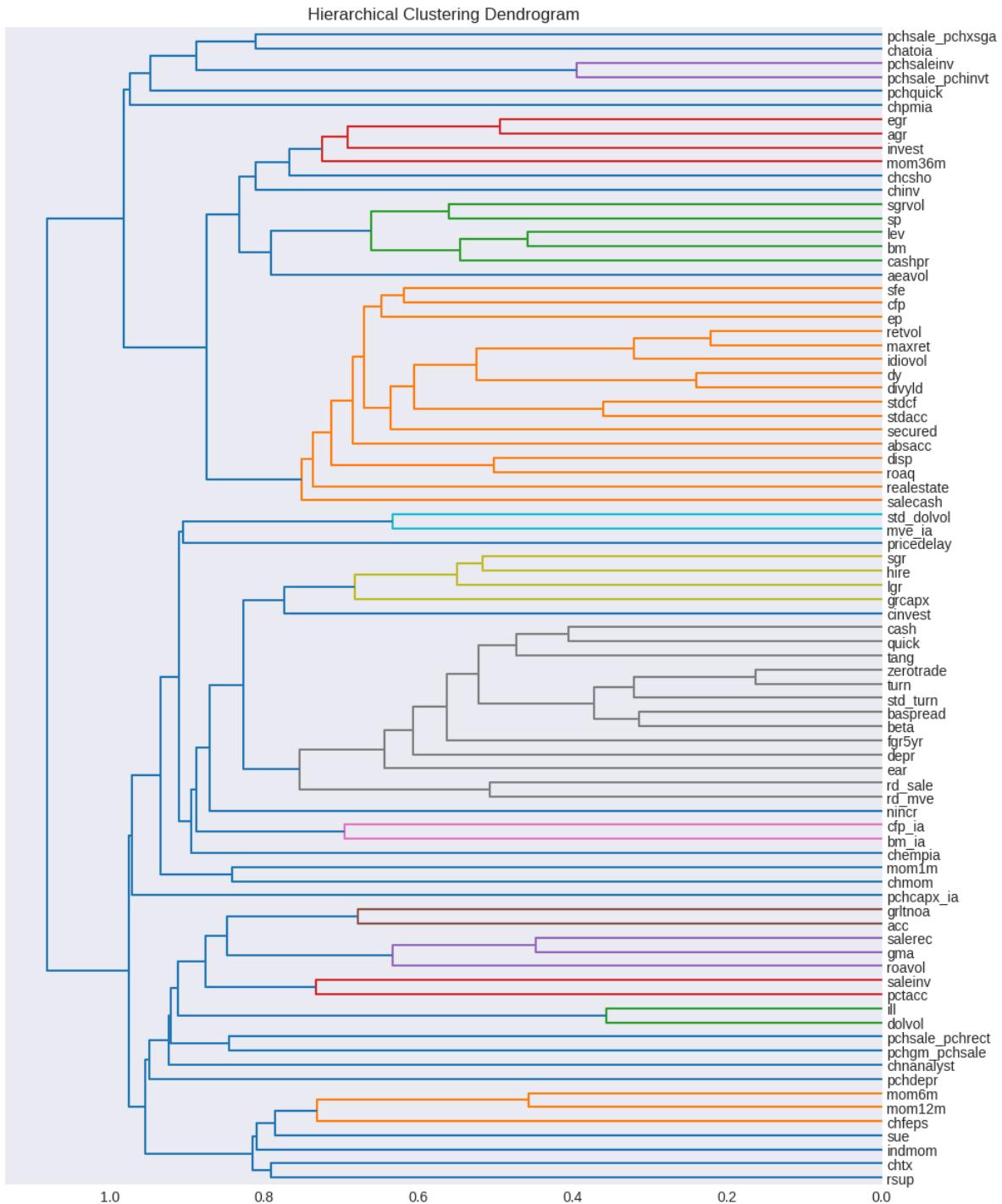
```
fig, ax = plt.subplots(figsize=(10, 12))
plt.title("Hierarchical Clustering Dendrogram")
# Create linkage matrix and then plot the dendrogram
# scikit-learn: "Plot Hierarchical Clustering Dendrogram"
counts = np.zeros(model.children_.shape[0])
n_samples = len(model.labels_)
# create the counts of 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)
```

(continues on next page)

(continued from previous page)

```
# Plot the corresponding dendrogram
dendrogram(linkage_matrix, orientation='left', labels=dist.columns, leaf_font_size=10)
plt.tight_layout()
```

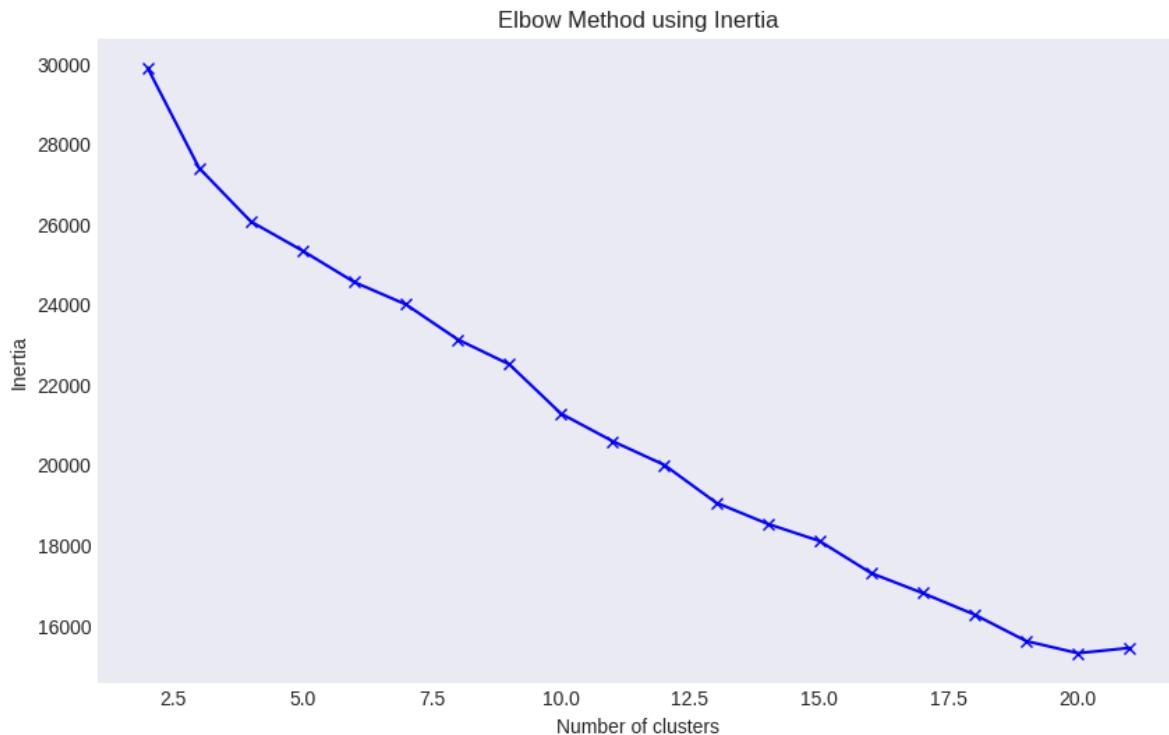


6.2.2 K-means clustering

K-means clustering is a popular unsupervised machine learning algorithm used for partitioning a dataset into K distinct, non-overlapping clusters. It starts by specifying the number of clusters, and choosing K initial cluster centroids randomly. Then it iteratively assigns data points to the nearest centroid, recalculating the centroids based on the means of data points in each new cluster, and repeating until convergence. K-means clustering aims to minimize the within-cluster sum of squared distances, making it suitable for datasets where clusters are spherical and have similar sizes.

The elbow method is a graphical method for selecting the “best” number of clusters in a k-means clustering algorithm. The graph shows the within-cluster sum-of-square values corresponding to the different values of K. The optimal K value is the point at which the graph forms an elbow.

```
# Selecting number of centers with elbow method
inertias = []
n_clusters = list(range(2, 22))
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 can also be used to evaluate the appropriateness of the number of clusters in K-means clustering. The silhouette score measures how similar a data point is to its own cluster compared to other clusters. It is the difference between cohesion subscore (i.e. the average distance between the data point and all other points within the same cluster) and separation subscore (the average distance between the data point and all points in each neighboring cluster). It is scaled by the maximum of those two subscores, hence ranges from -1 to 1, with higher values indicating better cluster separation. The number of clusters that maximizes the average silhouette score across all data points is chosen, which

provides the best balance between cohesion within clusters and separation between clusters.

```

# scikit-learn: "Selecting the number of clusters with silhouette analysis
# on KMeans clustering"
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)

    # 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")

    # 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])

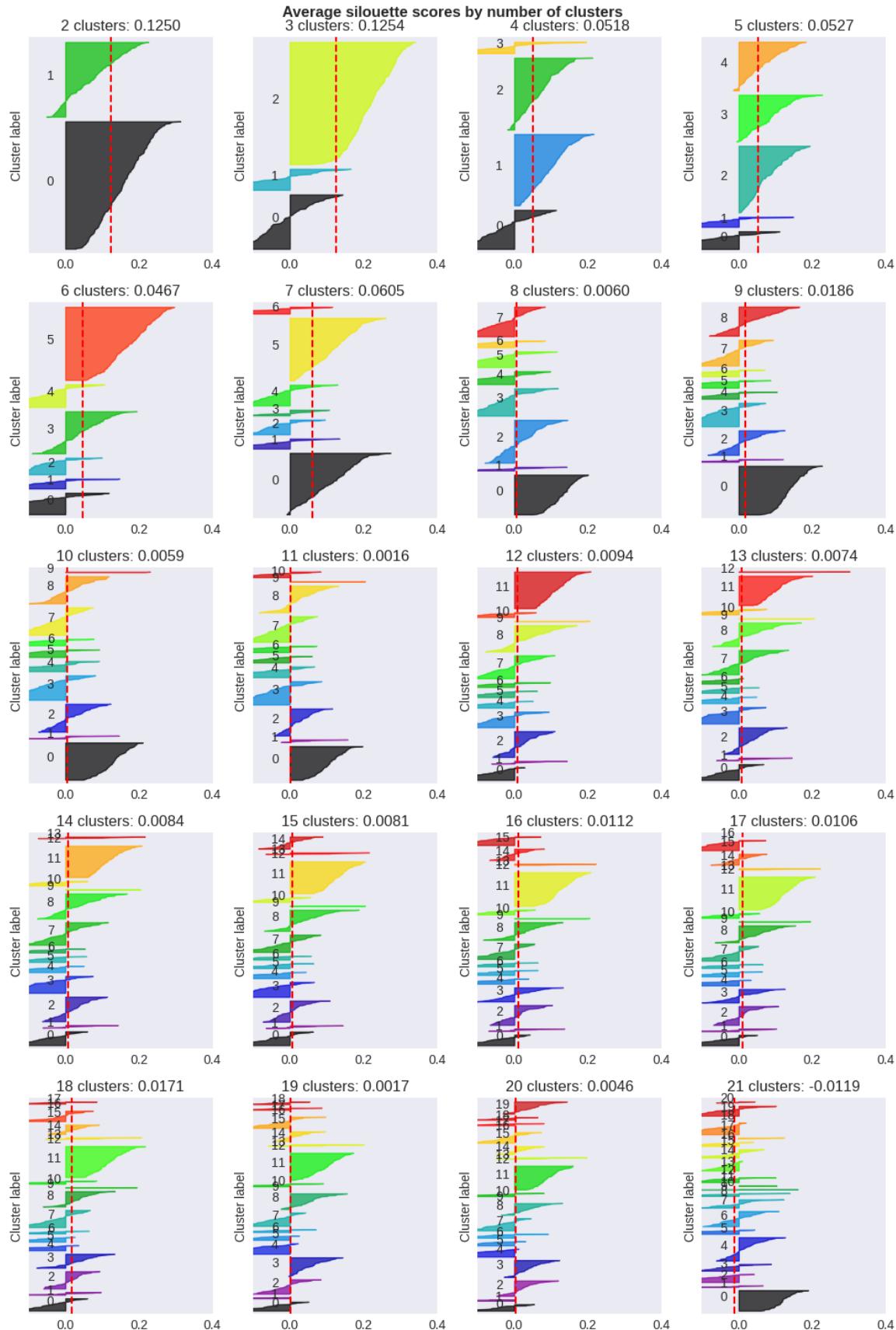
```

(continues on next page)

(continued from previous page)

```
plt.suptitle("Average silhouette scores by number of clusters", fontweight="bold")
plt.tight_layout()
plt.show()
```

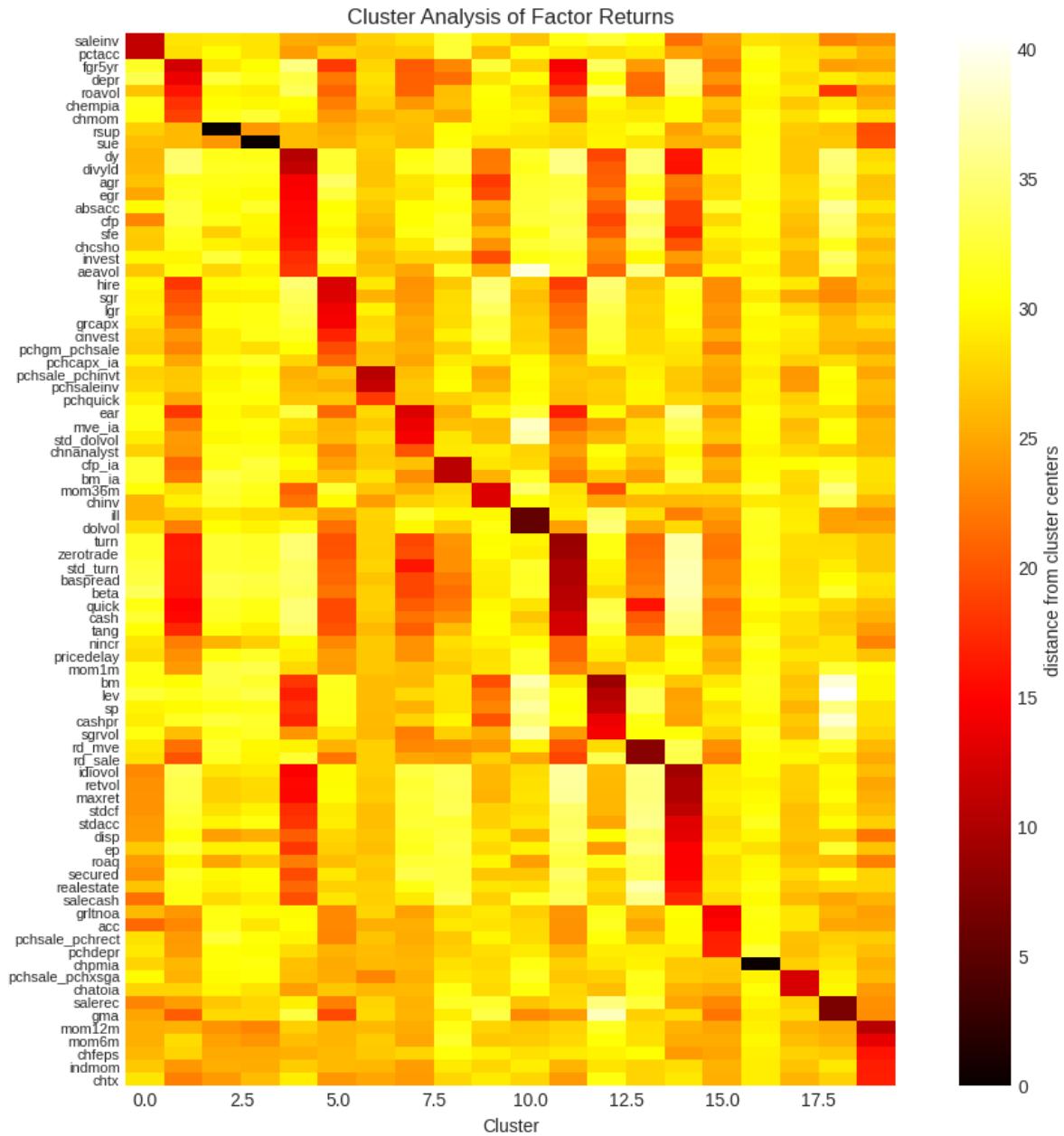
```
For n_clusters = 2 The average silhouette_score is : 0.12501306263608986
For n_clusters = 3 The average silhouette_score is : 0.1253520808683055
For n_clusters = 4 The average silhouette_score is : 0.051754214259910544
For n_clusters = 5 The average silhouette_score is : 0.05274630449356275
For n_clusters = 6 The average silhouette_score is : 0.04667415065092019
For n_clusters = 7 The average silhouette_score is : 0.060547187594243324
For n_clusters = 8 The average silhouette_score is : 0.006033319118781096
For n_clusters = 9 The average silhouette_score is : 0.018578386636466958
For n_clusters = 10 The average silhouette_score is : 0.005929767389313637
For n_clusters = 11 The average silhouette_score is : 0.0016142827400304784
For n_clusters = 12 The average silhouette_score is : 0.009433277087319107
For n_clusters = 13 The average silhouette_score is : 0.0074058888550970915
For n_clusters = 14 The average silhouette_score is : 0.008390097852726994
For n_clusters = 15 The average silhouette_score is : 0.008083959337648997
For n_clusters = 16 The average silhouette_score is : 0.011199855687941018
For n_clusters = 17 The average silhouette_score is : 0.010635928990571105
For n_clusters = 18 The average silhouette_score is : 0.017145225203944906
For n_clusters = 19 The average silhouette_score is : 0.0016945731172339912
For n_clusters = 20 The average silhouette_score is : 0.004636587977095786
For n_clusters = 21 The average silhouette_score is : -0.01191784490776267
```



Heatmap of correlations of strategy returns, grouped together in clusters identified by the K-Means algorithm with K=20.

```
kmeans = KMeans(n_clusters=20, 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('Cluster Analysis of Factor Returns')
plt.imshow(Z.iloc[:, :kmeans.n_clusters], cmap='hot', aspect=1/3)
plt.yticks(range(len(Z)), labels=Z.index, fontsize=8)
plt.xlabel('Cluster')
plt.colorbar(label='distance from cluster centers')
plt.tight_layout()
plt.show()
```



EVENT STUDY

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

Concepts

- Abnormal returns
- Convolution Theorem
- Multiple testing

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

MIT License. Copyright 2021-2024 Terence Lim

```
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 finds.backtesting import EventStudy
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)
```

(continues on next page)

(continued from previous page)

```
eventstudy = EventStudy(user, bench=bench, stocks=crsp, max_date=CRSP_DATE)
imgdir = paths['images'] / 'events'
```

Last FamaFrench Date 2024-02-29 00:00:00

7.1 Abnormal returns

Event studies assess the impact on stock returns of a company-specific event such as a merger or company announcement. They investigate deviations of stock returns from expected returns, called abnormal returns, on days around the disclosure of news.

Define the date of the announcement or event $t = 0$, and the daily stock return for each firm i is $R_{i,t}$. Returns of a market- or sector-reference group $R_{m,t}$ is computed as comparison. Then:

- Abnormal returns (AR) for each stock i at time t are defined as $AR_{i,t} = R_{i,t} - R_{m,t}$
- Average abnormal returns (AAR) across firms are defined as $AAR_t = \frac{1}{N} \sum_i^N AR_{i,t}$
- Cumulative average abnormal returns (CAR) are defined over time as $CAR_T = \sum_t AAR_t$
- Buy-and-hold average abnormal returns (BHAR) are defined across firms' buy-and-hold returns as $BHAR_T = \frac{1}{N} \sum_i^N [\Pi_t(1 + AR_{i,t}) - \Pi_t(1 + R_{m,t})]$

7.1.1 Announcement effect

An announcement effect is the immediate spike in stock price upon the announcement or event. An efficient reaction is one where the stock price maintains its new level after the event. An under-reaction occurs when the stock price continues to trend in the same direction after the announcement. An over-reaction occurs when stock prices drift in the opposite direction after the initial spike.

Often, the exact date and time cannot be pinpointed in historical data. An announcement window around the recorded announcement date, often encompassing one day before and after, accounts for this uncertainty.

A **pre-announcement drift** of stock prices could be consistent with insider information being leaked prior to the announcement, or other explanations. A **post-announcement drift** may possibly due to under- or over-reaction of investors to the news released.

When measuring announcement effects from many stocks and dates, the returns of stocks with the same announcement date are typically averaged together, so the dataset is reduced to unique announcement dates. Even after this transformation, Kolari et al (2010) demonstrate that even small cross-sectional correlation of abnormal returns can cause severe over-rejection of the null hypothesis of zero abnormal returns. In the presence of such cross-sectional correlations, an unbiased estimate of the mean abnormal return σ^2 is given by:

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

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

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)
```

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

Helper functions to retrieve size decile of universe stocks and run event study pipeline

```
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)

# 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',
```

(continues on next page)

(continued from previous page)

```

    '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 retrieve 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

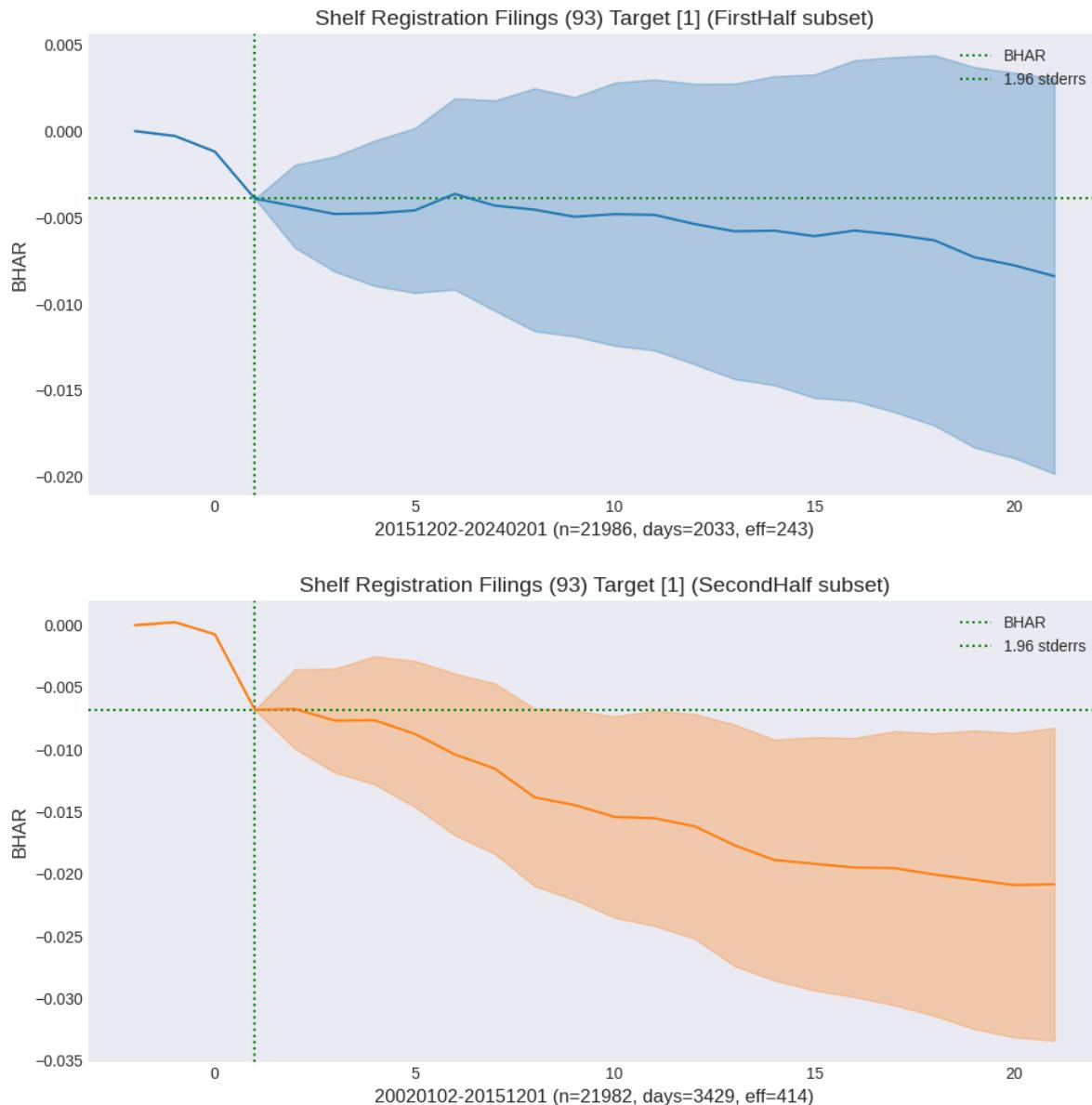
```

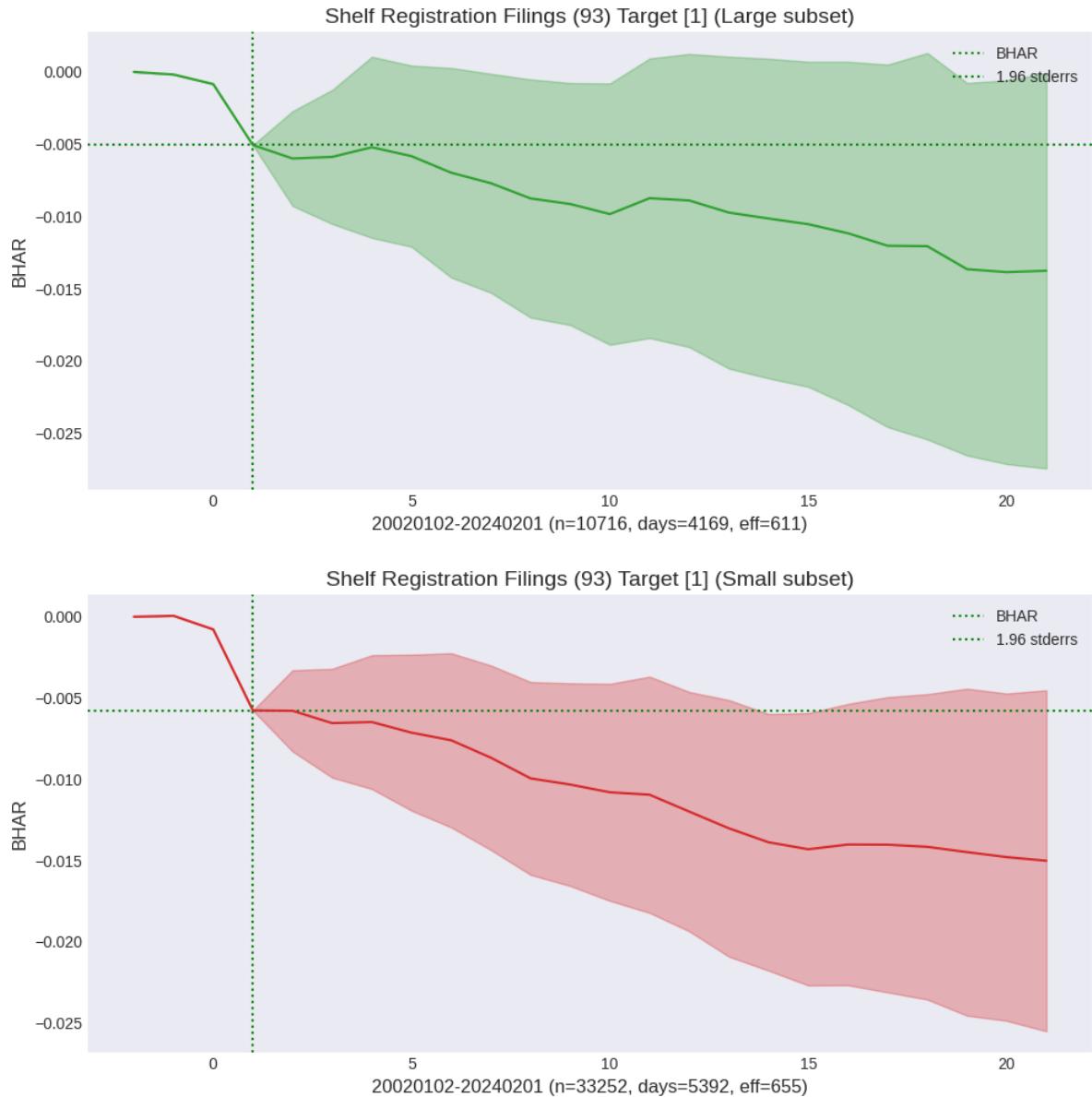
Show subsample plots for events with large post-announcement drift

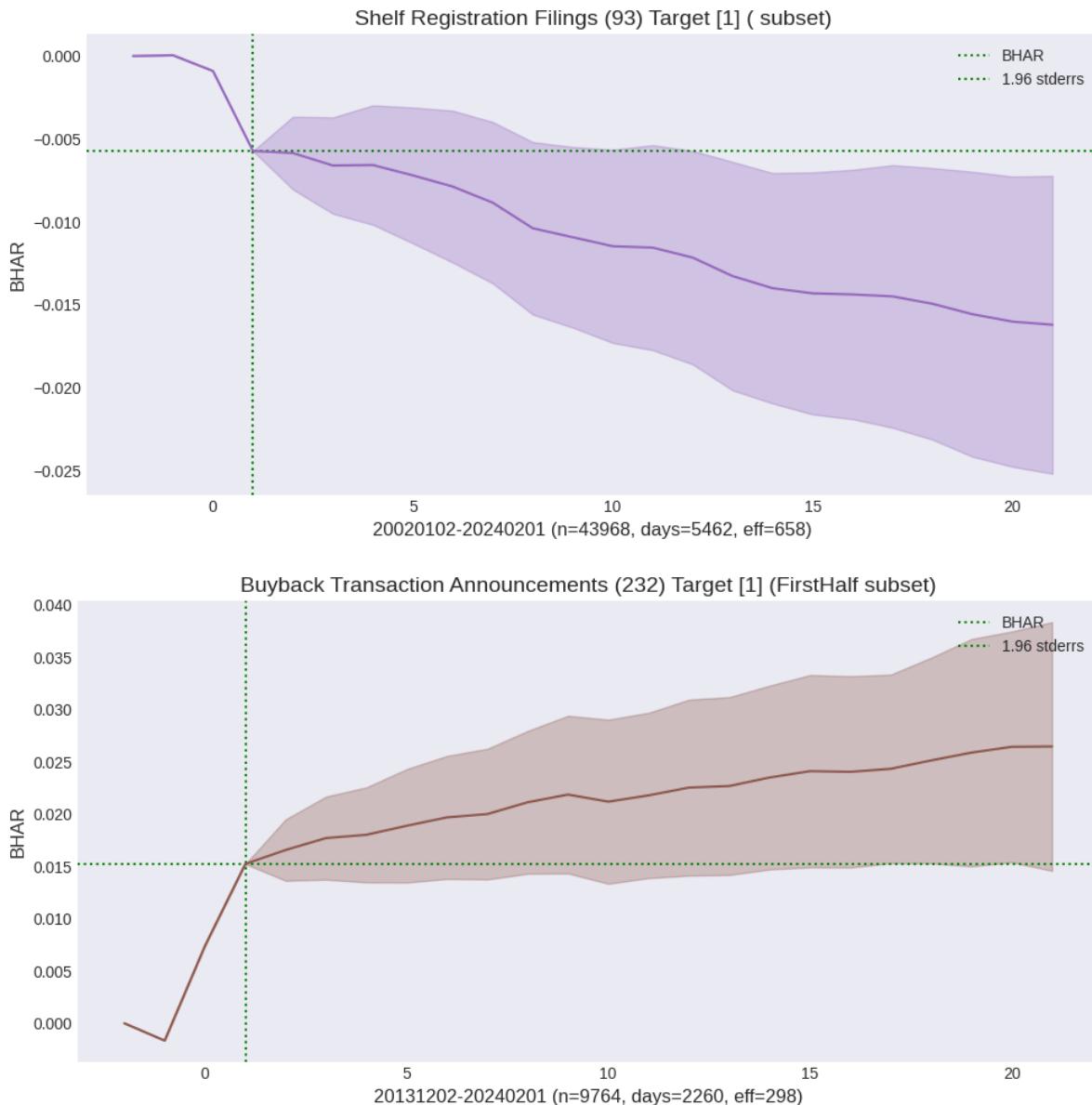
```

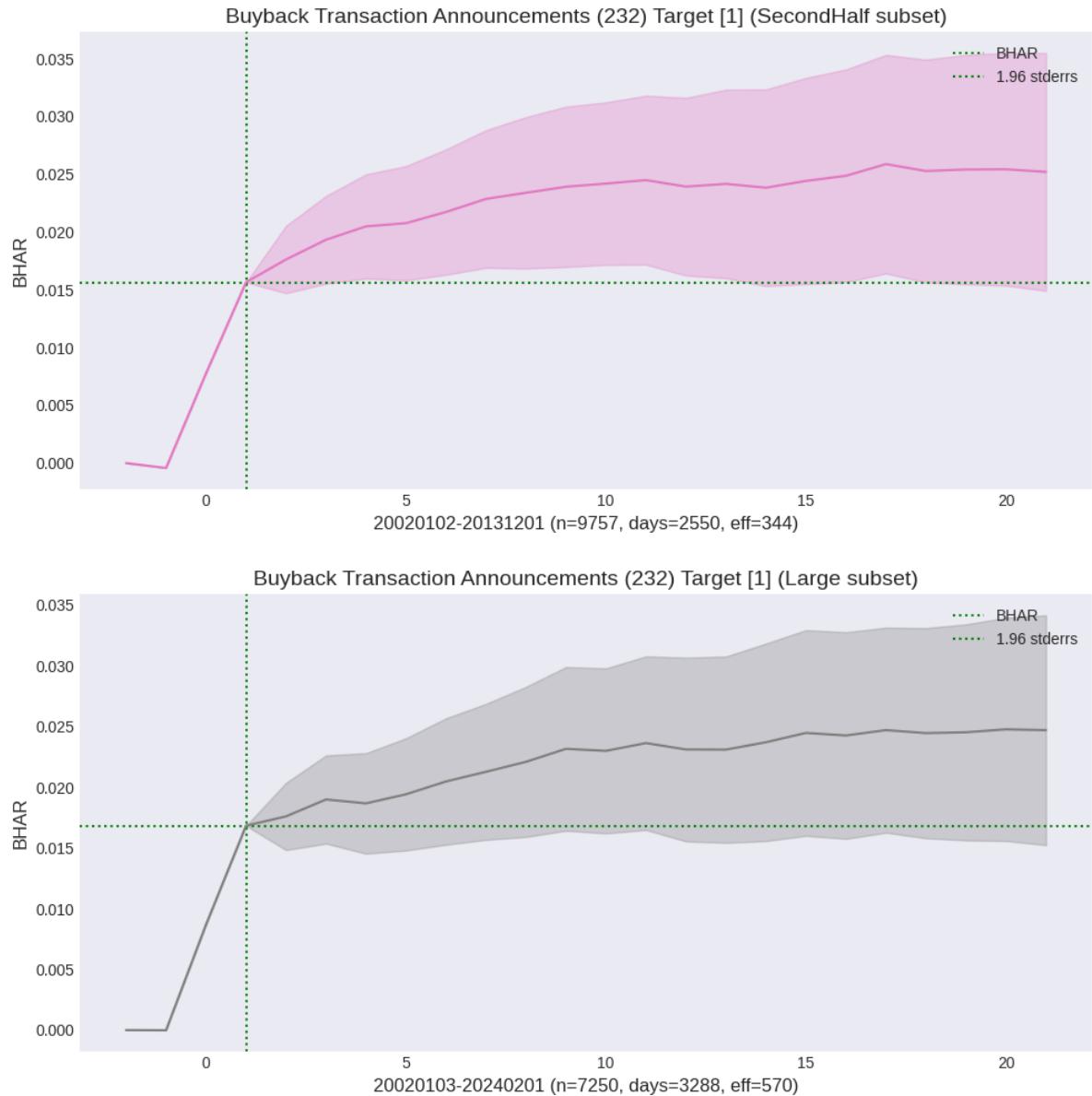
events_list = [[93, 1], [232, 1]] # largest drift returns
for i, (eventid, roleid) in enumerate(events_list):
    df = event_pipeline(eventstudy, eventid=eventid,
                        roleid=roleid,
                        beg=beg,
                        end=end,
                        left=left,
                        right=right,
                        post=post)
    halfperiod = np.median(df['announcedate'])
    sample = {'FirstHalf': df['announcedate'].ge(halfperiod).values,
              'SecondHalf': df['announcedate'].lt(halfperiod).values,
              'Large': df['size'].le(5).values,
              'Small': df['size'].gt(5).values,
              '' : []}
    for ifig, (label, rows) in enumerate(sample.items()):
        fig, ax = plt.subplots(clear=True, figsize=(10, 5))
        bhar = eventstudy.fit(model='sbhar', rows=rows)
        eventstudy.plot(model='sbhar',
                        title=eventformat(eventid, roleid) + f" ({label} subset)",
                        drift=True,
                        ax=ax,
                        c=f"C{i*5+ifig}")
    plt.tight_layout()
    plt.savefig(imgdir / (label + f"{eventid}_{roleid}.jpg"))

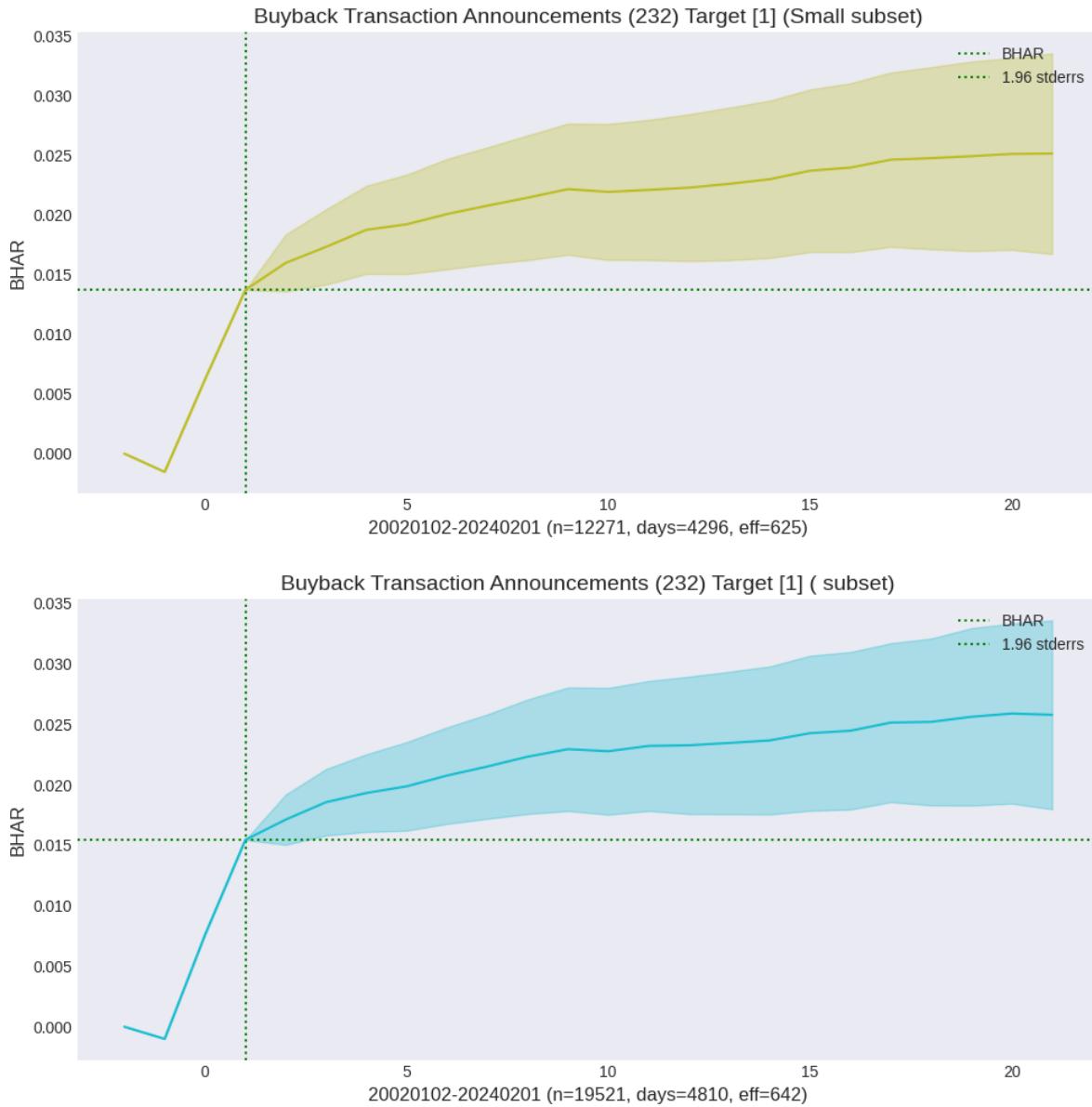
```











Show subsample plots for events with 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'].ge(halfperiod).values,
```

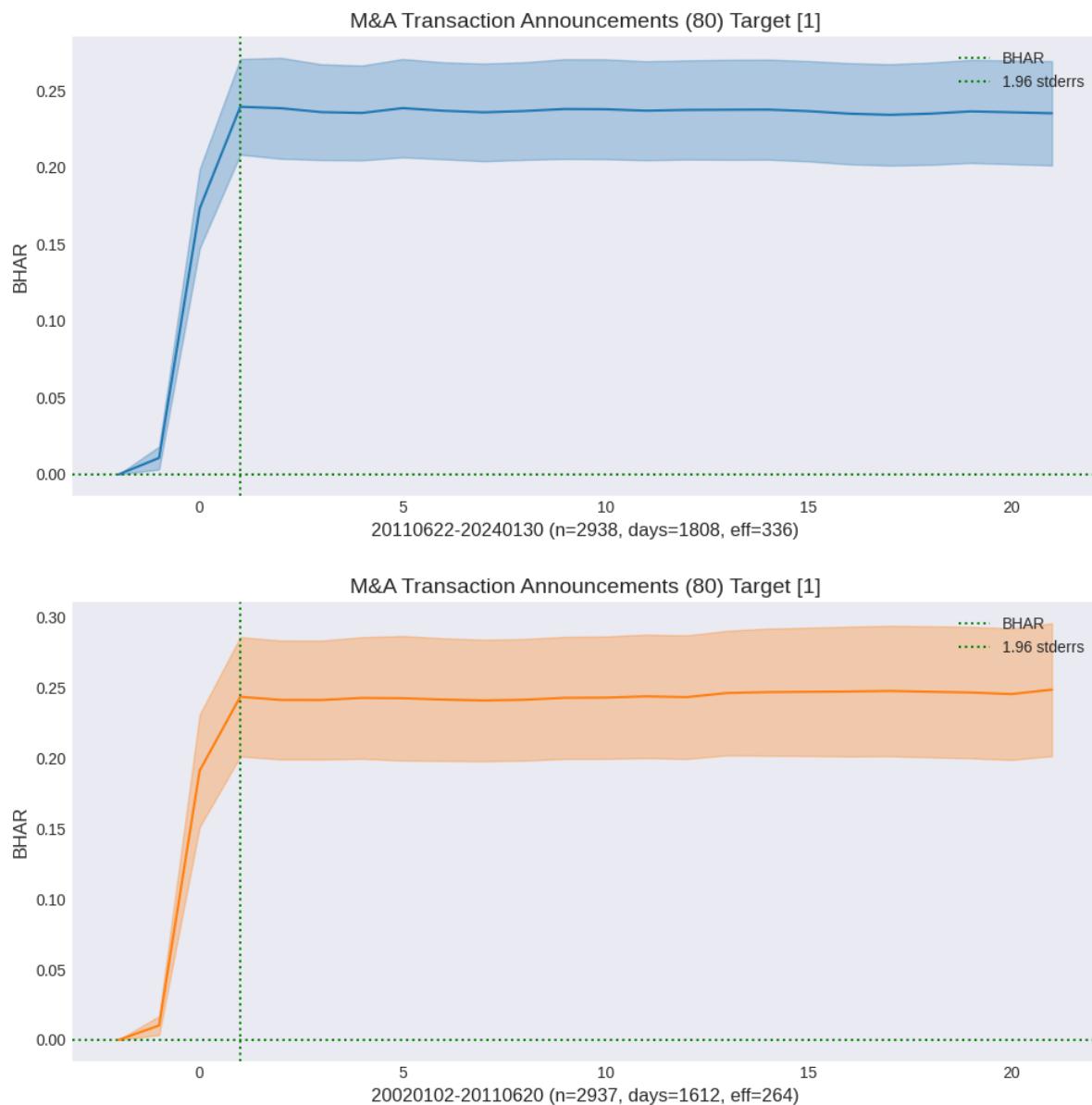
(continues on next page)

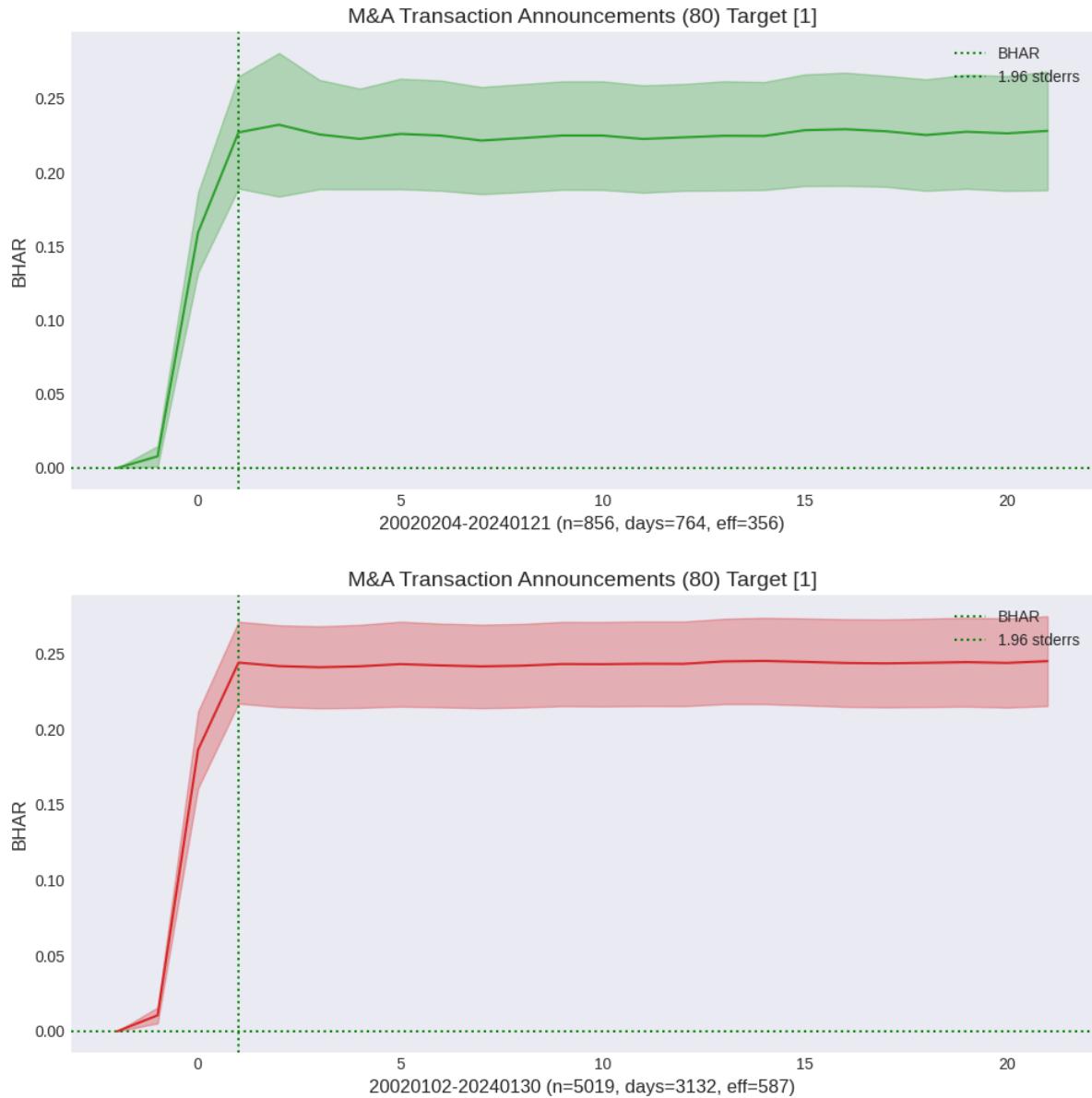
(continued from previous page)

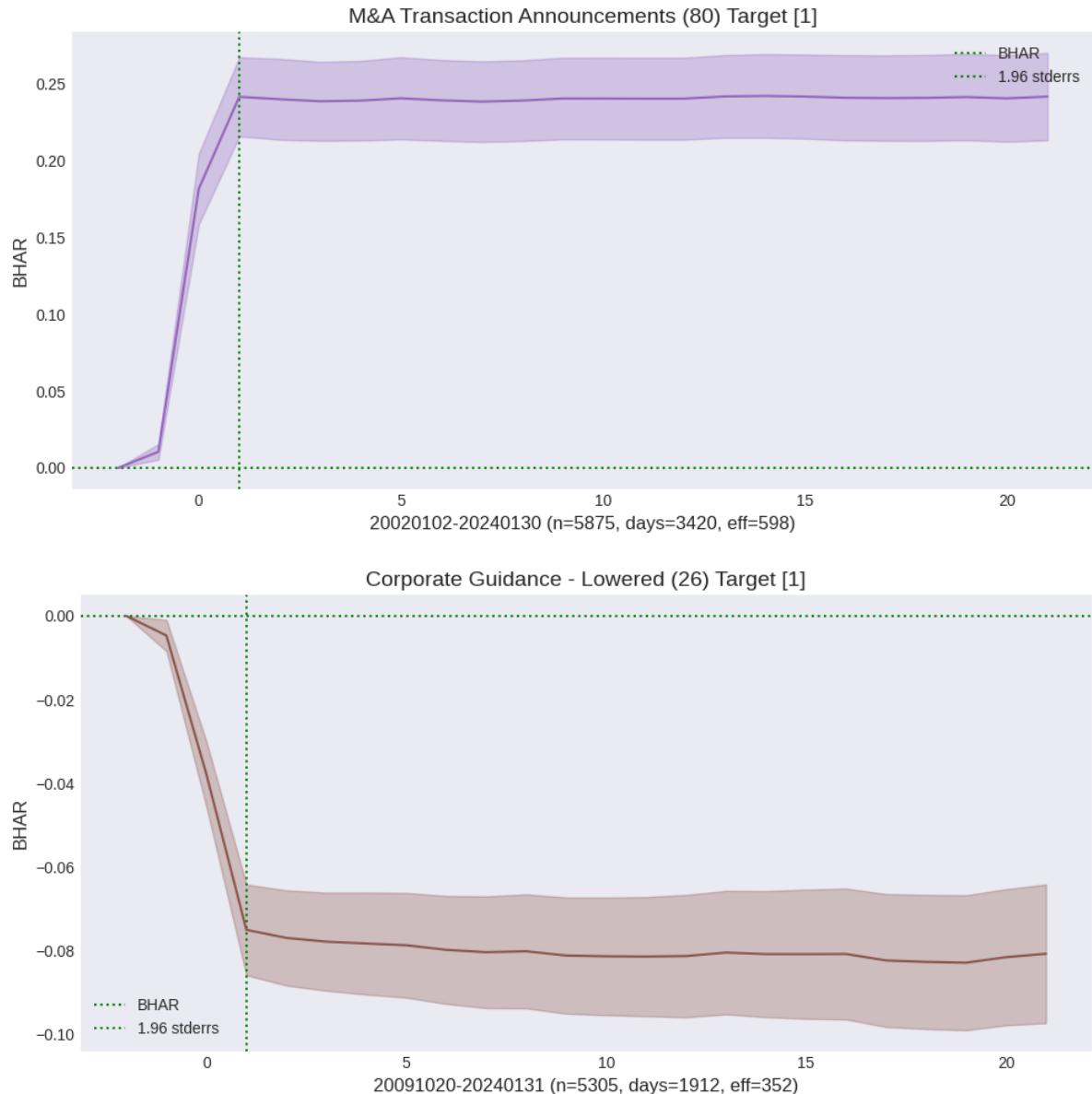
```

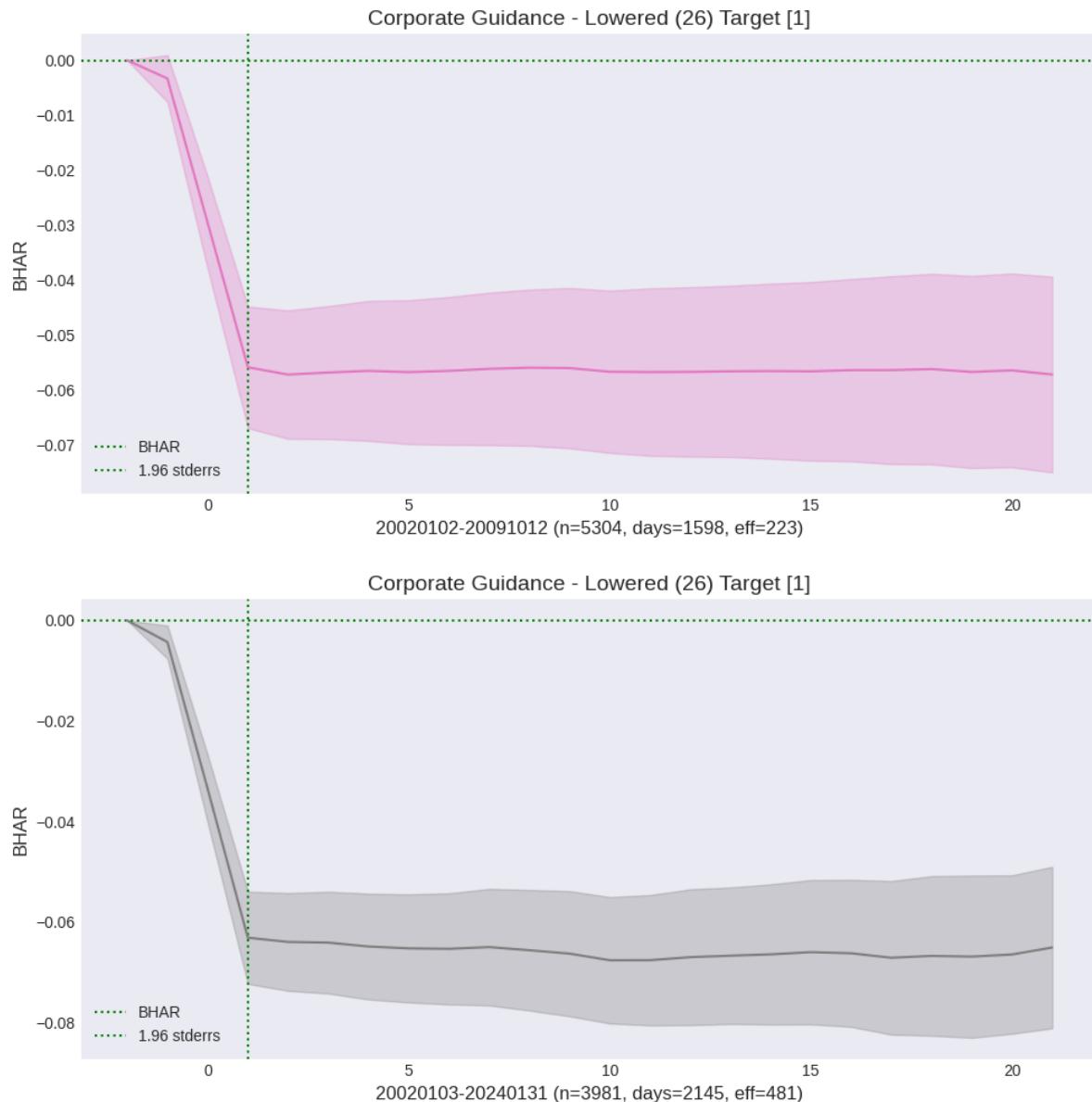
'SecondHalf': df['announcedate'].lt(halfperiod).values,
'Large': df['size'].le(5).values,
'Small': df['size'].gt(5).values,
':': []}
for ifig, (label, rows) in enumerate(sample.items()):
    fig, ax = plt.subplots(clear=True, figsize=(10, 5))
    bhar = eventstudy.fit(model='sbhar', rows=rows)
    eventstudy.plot(model='sbhar',
                     title=eventformat(eventid, roleid),
                     drift=False,
                     ax=ax,
                     c=f"C{i*5+ifig}")
plt.tight_layout()
plt.savefig(imgdir / (label + f"_{eventid}_{roleid}.jpg"))

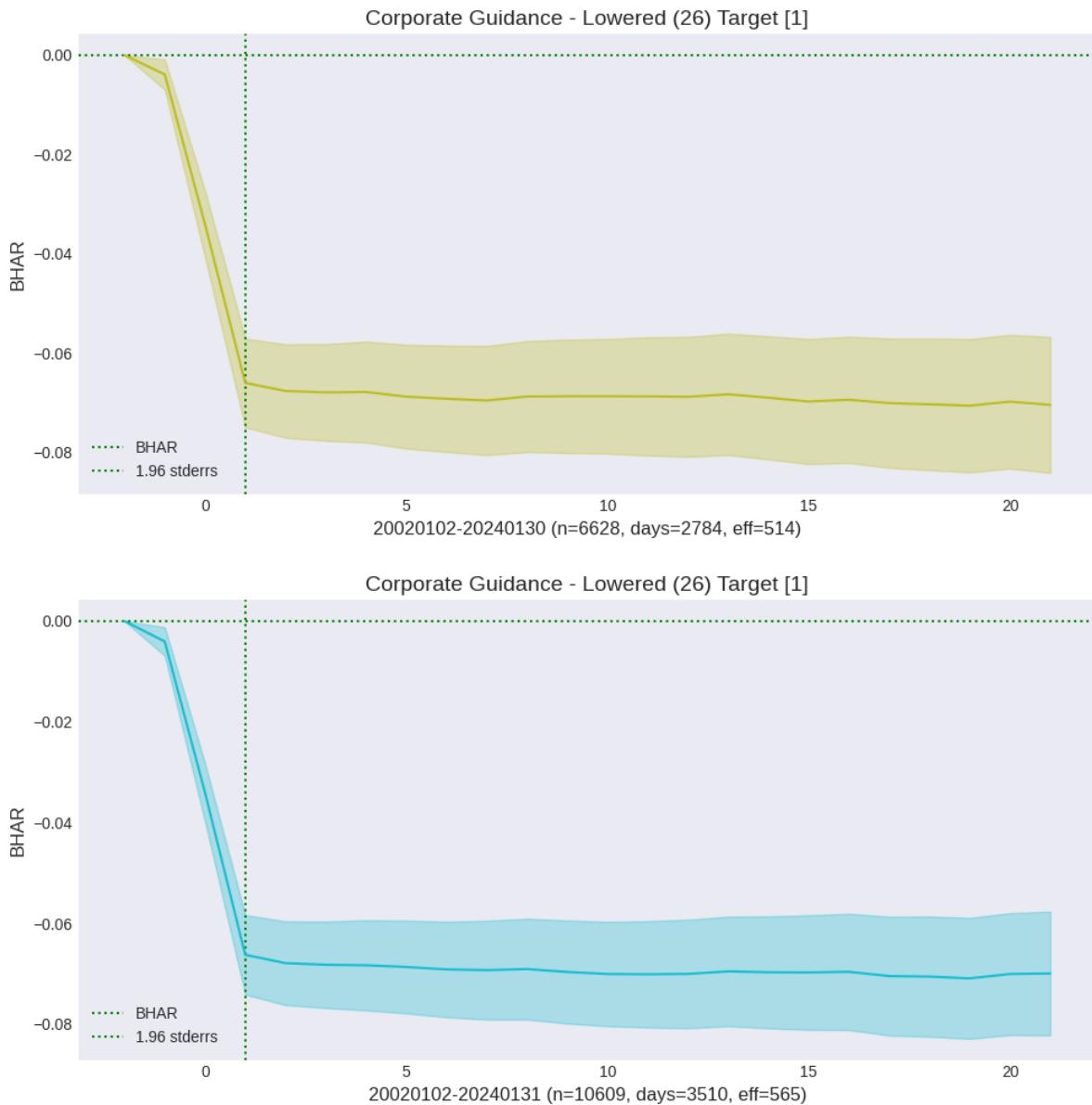
```











7.2 Cross correlation

Kolari et al (2018) consider event windows that partially overlap in calendar time, and modify standard errors and test statistics to adjust for the average percentage of overlaps in event windows and account for cross-sectional correlation effects.

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

where m is the length of the event window, τ is the ratio of number of overlapping calendar days of the event windows to the length of the event windows, and $\rho = \frac{\nu}{s^2}$ is the ratio of the average covariance between abnormal returns to their average variance. If the calendar time of the event day is the same for all firms such that all event periods are completely overlapping, and the residual series are standardized by their respective standard deviations, then ρ is just the average cross-sectional correlation of the estimation window residuals.

To approximate ρ : we assume that cross-sectional correlations between any two series are primarily due to their misalignment, where the number of time periods shifted is unknown. We first find the best alignment of event window residual returns between each pair of series that maximizes their pair-wise correlation, and then compute the average of these best-aligned correlations over all pairs.

7.2.1 Convolution theorem

From the field of signal processing, frequency domain techniques are well suited for computing the cross-correlations at all lags between each pair of time series, and terming every pair-specific best alignment with maximum correlation. Directly computing the correlations for every possible lag between two series may seem computationally expensive over all possible pairs. Such an operation between two series, known as a convolution, can be computed as a product of the Fourier transforms of the two series. The relevant definitions and theorems are summarized below:

A convolution is a mathematical operation on two functions (f and g) that produces a third function ($f*g$) which expresses how the shape of one is modified by the other. It is defined as the integral of the product of the two functions after one is reflected about the y-axis and shifted.

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

The Convolution Theorem states that under suitable conditions the Fourier transform of a convolution of two signals is the pointwise product of their Fourier transforms. More generally, convolution in the time domain equals point-wise multiplication in the frequency domain. Hence a convolution of two sequences can be obtained as the inverse transform of the product of the individual transforms. Let $*$ denote the convolution of two series:

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

Cross-correlation measures the similarity of two series as a function of the displacement of one relative to the other. This is also known as a sliding dot product or sliding inner-product. Let \star denote the cross-correlation of two series, and \bar{f} denote the complex conjugate:

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

The Cross-correlation Theorem states that the cross-correlation integral is equivalent to the convolution integral if one of the input signals is conjugated and time-reversed.

$$[f(t) \star g(t)](\tau) = [\overline{f(-t)} * g(t)](\tau)$$

where τ is called the displacement or lag. For highly-correlated f and g which have a maximum cross-correlation at a particular τ , a feature in f at t also occurs later in g at $t + \tau$, hence g could be described to lag f by τ .

```
# 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
```

(continues on next page)

(continued from previous page)

```

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.extend(np.max(conv, axis=0))
    shift = (N//2) + 1 # displacement location relative to center
    disp.extend(((np.argmax(conv, axis=0) + shift) % N) - shift + 1)
    cols.extend([(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:])))

```

```

([0.9998344824121533], [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:])))

```

```

([0.9996045585524033, 0.9993741961936712, 0.9997696328132938],
 [5, 7, 2],
 [(0, 1), (0, 2), (1, 2)])

```

7.3 Multiple testing problem

When testing a huge number of null hypotheses, we are bound to get some very testing small p-values by chance. If we make a decision about whether to reject each null hypothesis without accounting for the fact that we have performed a very large number of tests, then we may end up rejecting a great number of true null hypotheses – that is, making a large number of Type I errors.

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. 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.

The false discovery rate (FDR) is the ratio of false discoveries to number of positive tests, which is the sum of false discoveries (i.e. false positives) and true positives. To calculate the false discovery rate, we must take a guess at the fraction of tests that we do in which there is a real difference, and the probability that the test will give the right result when there is a real effect (power). The FDR views unacceptable in terms of a proportion. For example, if one false discovery were unacceptable for 100 tests, then 10 are unacceptable for 1,000 tests.

Family-wise error rate (FWER) generalizes this notion to the setting of m null hypotheses, H_1, \dots, H_m , and is defined as the probability error rate of making at least one Type I error. If we make the additional rather strong assumptions that

the m tests are independent and that all m null hypotheses are true, then

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

In the family-wise error rate, it is unacceptable to make a single false discovery. This is a very severe rule.

Given the p-values, the `multipletests` function from the `statsmodels` module outputs adjusted p-values, which can be thought of as a new set of p-values that have been corrected for multiple testing. If the adjusted p-value for a given hypothesis is less than or equal to alpha, then that hypothesis can be rejected while maintaining a FWER of no more than alpha. In other words, for such methods, the adjusted p-values resulting from the `multipletests` function can simply be compared to the desired FWER in order to determine whether or not to reject each hypothesis. We can use the same function to control FDR as well.

Compute BHAR and CAR of all events

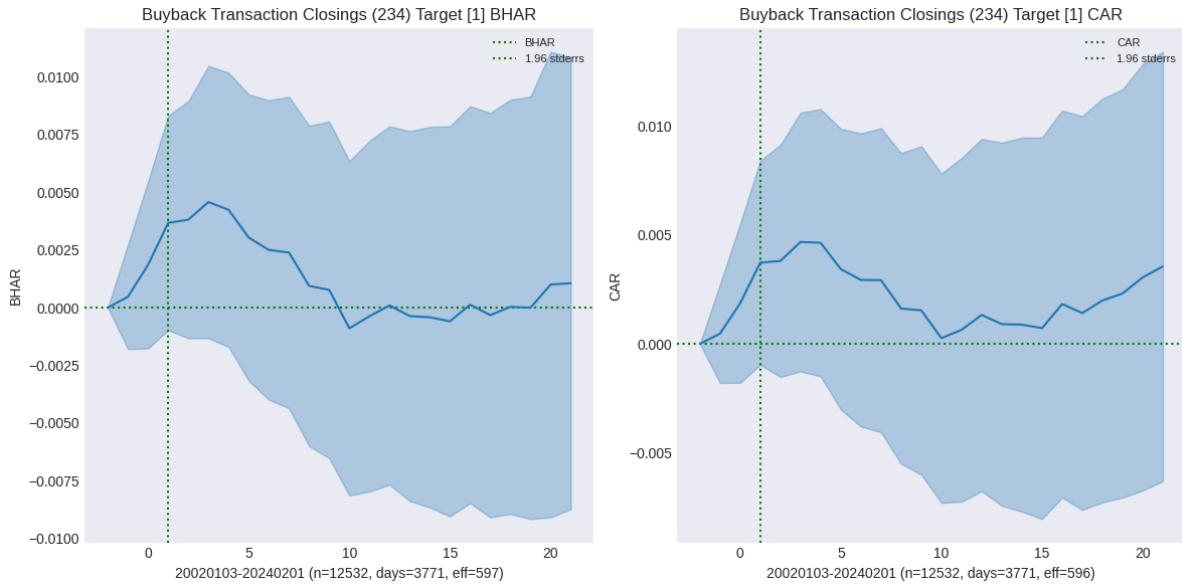
```
restart_event = 0 # 75,1
for i, eventid in tqdm(enumerate(events), total=len(events)):
    if eventid <= restart_event: # kludge to resume loop
        continue
    for roleid in roles:
        # retrieve all returns observations of this eventid, roleid
        df = event_pipeline(eventstudy,
                            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, 6), 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")
```

0% | 0/116 [00:00<?, ?it/s]

100% [██████████] 116/116 [26:18:33<00:00, 816.50s/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 | 20240201 | |
| Changes in Company Bylaws/Rules | Target | 20020430 | 20240201 | |
| Auditor Changes | Target | 20020103 | 20240131 | |
| Name Changes | Target | 20020430 | 20240130 | |
| Executive Changes - CFO | Target | 20020102 | 20240201 | |
| Special/Extraordinary Shareholders Meeting | Target | 20040106 | 20240131 | |
| Product-Related Announcements | Target | 20020102 | 20240201 | |
| Investor Activism - Target Communication | Target | 20020402 | 20240201 | |
| Executive/Board Changes - Other | Target | 20020101 | 20240201 | |
| Executive Changes - CEO | Target | 20020102 | 20240201 | |
| Index Constituent Adds | Target | 20020102 | 20240123 | |
| M&A Transaction Cancellations | Buyer | 20020107 | 20240128 | |
| M&A Transaction Closings | Target | 20020109 | 20240126 | |
| Business Expansions | Target | 20020102 | 20240201 | |
| Private Placements | Buyer | 20020103 | 20240131 | |
| Earnings Release Date | Target | 20020411 | 20240201 | |

(continues on next page)

(continued from previous page)

| | | | |
|---|-------------|----------|----------|
| Corporate Guidance - Lowered | Target | 20020102 | 20240131 |
| Client Announcements | Target | 20020102 | 20240201 |
| Conferences | Participant | 20070125 | 20240201 |
| Investor Activism - Activist Communication | Target | 20020102 | 20240201 |
| Special Calls | Target | 20030211 | 20240201 |
| Strategic Alliances | Target | 20020102 | 20240201 |
| Buyback Transaction Closings | Target | 20020103 | 20240201 |
| M&A Transaction Announcements | Buyer | 20020101 | 20240201 |
| M&A Transaction Closings | Buyer | 20020101 | 20240201 |
| M&A Transaction Announcements | Seller | 20020101 | 20240201 |
| M&A Calls | Target | 20031027 | 20240130 |
| Shareholder/Analyst Calls | Target | 20020108 | 20240201 |
| M&A Transaction Closings | Seller | 20020101 | 20240201 |
| Follow-on Equity Offerings | Target | 20020102 | 20240201 |
| M&A Rumors and Discussions | Target | 20020918 | 20240201 |
| Business Reorganizations | Target | 20020102 | 20240129 |
| Discontinued Operations/Downsizings | Target | 20020102 | 20240201 |
| Fixed Income Offerings | Target | 20020101 | 20240201 |
| Private Placements | Target | 20020103 | 20240201 |
| Delayed SEC Filings | Target | 20020102 | 20240130 |
| Labor-related Announcements | Target | 20020111 | 20240125 |
| M&A Transaction Announcements | Target | 20020102 | 20240130 |
| Earnings Calls | Target | 20020108 | 20240201 |
| Lawsuits & Legal Issues | Target | 20020101 | 20240201 |
| Board Meeting | Target | 20020204 | 20240126 |
| Corporate Guidance - New/Confirmed | Target | 20020101 | 20240201 |
| Seeking Acquisitions/Investments | Target | 20020103 | 20240201 |
| Impairments/Write Offs | Target | 20020408 | 20240201 |
| Company Conference Presentations | Target | 20041101 | 20240201 |
| Announcements of Sales/Trading Statement | Target | 20020102 | 20240130 |
| Considering Multiple Strategic Alternatives | Target | 20040324 | 20240131 |
| Debt Financing Related | Target | 20020102 | 20240201 |
| Buyback Tranche Update | Target | 20020107 | 20240201 |
| Seeking to Sell/Divest | Target | 20020110 | 20240117 |
| End of Lock-Up Period | Target | 20020115 | 20240130 |
| Corporate Guidance - Raised | Target | 20020103 | 20240201 |
| Announcements of Earnings | Target | 20020101 | 20240201 |
| Delistings | Target | 20020107 | 20240201 |
| Analyst/Investor Day | Target | 20040204 | 20240201 |
| Investor Activism - Proxy/Voting Related | Target | 20020107 | 20240201 |
| Annual General Meeting | Target | 20020122 | 20240129 |
| Buyback - Change in Plan Terms | Target | 20020102 | 20240201 |
| Buyback Transaction Announcements | Target | 20020102 | 20240201 |

| event | role | rows | days | \ |
|--|--------|--------|------|---|
| Shelf Registration Filings | Target | 43968 | 5462 | |
| Changes in Company Bylaws/Rules | Target | 26080 | 4614 | |
| Auditor Changes | Target | 9232 | 3738 | |
| Name Changes | Target | 1516 | 1282 | |
| Executive Changes - CFO | Target | 21873 | 5337 | |
| Special/Extraordinary Shareholders Meeting | Target | 5710 | 3176 | |
| Product-Related Announcements | Target | 203597 | 5559 | |
| Investor Activism - Target Communication | Target | 4995 | 2914 | |
| Executive/Board Changes - Other | Target | 215926 | 5558 | |
| Executive Changes - CEO | Target | 16560 | 5080 | |

(continues on next page)

(continued from previous page)

| | | Target | 27762 | 2158 | |
|---|--|-------------|--------|-----------|----------|
| Index Constituent Adds | | Buyer | 1481 | 1233 | |
| M&A Transaction Cancellations | | Target | 1453 | 1154 | |
| M&A Transaction Closings | | Target | 57173 | 5510 | |
| Business Expansions | | Buyer | 10955 | 4168 | |
| Private Placements | | Target | 218101 | 5086 | |
| Earnings Release Date | | Target | 10609 | 3510 | |
| Corporate Guidance - Lowered | | Target | 208109 | 5555 | |
| Client Announcements | | Participant | 280404 | 3467 | |
| Conferences | | Target | 8126 | 3707 | |
| Investor Activism - Activist Communication | | Target | 15526 | 4299 | |
| Special Calls | | Target | 28757 | 5281 | |
| Strategic Alliances | | Target | 12532 | 3771 | |
| Buyback Transaction Closings | | Buyer | 24750 | 5309 | |
| M&A Transaction Announcements | | Buyer | 42052 | 5496 | |
| M&A Transaction Closings | | Seller | 12010 | 4648 | |
| M&A Transaction Announcements | | Target | 7029 | 3193 | |
| M&A Calls | | Target | 21606 | 4209 | |
| Shareholder/Analyst Calls | | Seller | 18006 | 4962 | |
| M&A Transaction Closings | | Target | 20686 | 5015 | |
| Follow-on Equity Offerings | | Target | 21229 | 4421 | |
| M&A Rumors and Discussions | | Target | 4374 | 2743 | |
| Business Reorganizations | | Target | 18697 | 4379 | |
| Discontinued Operations/Downsizings | | Target | 26916 | 5203 | |
| Fixed Income Offerings | | Target | 14052 | 4838 | |
| Private Placements | | Target | 12480 | 1999 | |
| Delayed SEC Filings | | Target | 3941 | 2460 | |
| Labor-related Announcements | | Target | 5875 | 3420 | |
| M&A Transaction Announcements | | Target | 227032 | 5234 | |
| Earnings Calls | | Target | 40786 | 5420 | |
| Lawsuits & Legal Issues | | Target | 6482 | 2646 | |
| Board Meeting | | Target | 143964 | 5449 | |
| Corporate Guidance - New/Confirmed | | Target | 37127 | 5089 | |
| Seeking Acquisitions/Investments | | Target | 22850 | 3407 | |
| Impairments/Write Offs | | Target | 324553 | 4767 | |
| Company Conference Presentations | | Target | 12285 | 3231 | |
| Announcements of Sales/Trading Statement | | Target | 3914 | 2410 | |
| Considering Multiple Strategic Alternatives | | Target | 40154 | 5508 | |
| Debt Financing Related | | Target | 106352 | 4870 | |
| Buyback Tranche Update | | Target | 5230 | 2908 | |
| Seeking to Sell/Divest | | Target | 6822 | 3151 | |
| End of Lock-Up Period | | Target | 19376 | 4041 | |
| Corporate Guidance - Raised | | Target | 331260 | 5502 | |
| Announcements of Earnings | | Target | 16431 | 4736 | |
| Delistings | | Target | 6407 | 3211 | |
| Analyst/Investor Day | | Target | 6993 | 3020 | |
| Investor Activism - Proxy/Voting Related | | Target | 68147 | 4887 | |
| Annual General Meeting | | Target | 6627 | 3281 | |
| Buyback - Change in Plan Terms | | Target | 19521 | 4810 | |
| Buyback Transaction Announcements | | role | | effective | window \ |
| event | | Target | 658.0 | -0.005723 | |
| Shelf Registration Filings | | Target | 567.0 | 0.000111 | |
| Changes in Company Bylaws/Rules | | Target | 587.0 | -0.003299 | |
| Auditor Changes | | Target | 434.0 | 0.012904 | |

(continues on next page)

(continued from previous page)

| | | | |
|---|-------------|-------|-----------|
| Executive Changes - CFO | Target | 652.0 | -0.007732 |
| Special/Extraordinary Shareholders Meeting | Target | 513.0 | -0.000755 |
| Product-Related Announcements | Target | 662.0 | 0.005881 |
| Investor Activism - Target Communication | Target | 504.0 | 0.008170 |
| Executive/Board Changes - Other | Target | 661.0 | -0.000451 |
| Executive Changes - CEO | Target | 648.0 | -0.004515 |
| Index Constituent Adds | Target | 496.0 | -0.001032 |
| M&A Transaction Cancellations | Buyer | 444.0 | -0.000830 |
| M&A Transaction Closings | Target | 406.0 | 0.011893 |
| Business Expansions | Target | 663.0 | 0.001807 |
| Private Placements | Buyer | 623.0 | 0.000871 |
| Earnings Release Date | Target | 608.0 | 0.000167 |
| Corporate Guidance - Lowered | Target | 565.0 | -0.066260 |
| Client Announcements | Target | 663.0 | 0.008302 |
| Conferences | Participant | 426.0 | -0.000576 |
| Investor Activism - Activist Communication | Target | 553.0 | 0.015004 |
| Special Calls | Target | 574.0 | 0.019873 |
| Strategic Alliances | Target | 655.0 | 0.010172 |
| Buyback Transaction Closings | Target | 597.0 | 0.003669 |
| M&A Transaction Announcements | Buyer | 658.0 | 0.010916 |
| M&A Transaction Closings | Buyer | 663.0 | 0.004906 |
| M&A Transaction Announcements | Seller | 639.0 | 0.010365 |
| M&A Calls | Target | 543.0 | 0.060313 |
| Shareholder/Analyst Calls | Target | 562.0 | -0.000081 |
| M&A Transaction Closings | Seller | 649.0 | 0.004011 |
| Follow-on Equity Offerings | Target | 642.0 | -0.031380 |
| M&A Rumors and Discussions | Target | 561.0 | 0.013601 |
| Business Reorganizations | Target | 512.0 | -0.002682 |
| Discontinued Operations/Downsizings | Target | 581.0 | -0.008746 |
| Fixed Income Offerings | Target | 652.0 | -0.005389 |
| Private Placements | Target | 634.0 | 0.022058 |
| Delayed SEC Filings | Target | 498.0 | -0.014976 |
| Labor-related Announcements | Target | 476.0 | 0.002075 |
| M&A Transaction Announcements | Target | 598.0 | 0.241380 |
| Earnings Calls | Target | 630.0 | 0.000473 |
| Lawsuits & Legal Issues | Target | 658.0 | -0.000651 |
| Board Meeting | Target | 458.0 | 0.001948 |
| Corporate Guidance - New/Confirmed | Target | 661.0 | -0.002079 |
| Seeking Acquisitions/Investments | Target | 647.0 | -0.000001 |
| Impairments/Write Offs | Target | 502.0 | -0.002974 |
| Company Conference Presentations | Target | 568.0 | 0.001339 |
| Announcements of Sales/Trading Statement | Target | 570.0 | 0.002500 |
| Considering Multiple Strategic Alternatives | Target | 461.0 | -0.007378 |
| Debt Financing Related | Target | 660.0 | 0.005354 |
| Buyback Tranche Update | Target | 621.0 | 0.000841 |
| Seeking to Sell/Divest | Target | 499.0 | -0.002341 |
| End of Lock-Up Period | Target | 541.0 | -0.058891 |
| Corporate Guidance - Raised | Target | 611.0 | 0.032878 |
| Announcements of Earnings | Target | 660.0 | 0.000328 |
| Delistings | Target | 603.0 | -0.022604 |
| Analyst/Investor Day | Target | 524.0 | 0.006024 |
| Investor Activism - Proxy/Voting Related | Target | 507.0 | -0.001108 |
| Annual General Meeting | Target | 595.0 | 0.001885 |
| Buyback - Change in Plan Terms | Target | 583.0 | 0.008746 |
| Buyback Transaction Announcements | Target | 642.0 | 0.015452 |

(continues on next page)

(continued from previous page)

| event | role | window_t | post | \ |
|---|-------------|------------|-----------|---|
| Shelf Registration Filings | Target | -2.953200 | -0.010470 | |
| Changes in Company Bylaws/Rules | Target | 0.042656 | -0.010652 | |
| Auditor Changes | Target | -1.106310 | -0.011627 | |
| Name Changes | Target | 2.072100 | -0.021342 | |
| Executive Changes - CFO | Target | -3.119900 | -0.007747 | |
| Special/Extraordinary Shareholders Meeting | Target | -0.203839 | -0.013482 | |
| Product-Related Announcements | Target | 4.758170 | -0.003877 | |
| Investor Activism - Target Communication | Target | 1.945040 | -0.007546 | |
| Executive/Board Changes - Other | Target | -0.442385 | -0.002750 | |
| Executive Changes - CEO | Target | -1.210740 | -0.007229 | |
| Index Constituent Adds | Target | -0.394831 | -0.006559 | |
| M&A Transaction Cancellations | Buyer | -0.153281 | -0.011231 | |
| M&A Transaction Closings | Target | 1.369390 | -0.013536 | |
| Business Expansions | Target | 1.465250 | -0.002680 | |
| Private Placements | Buyer | 0.522021 | -0.003573 | |
| Earnings Release Date | Target | 0.158605 | -0.002156 | |
| Corporate Guidance - Lowered | Target | -16.435100 | -0.003715 | |
| Client Announcements | Target | 7.351210 | -0.002044 | |
| Conferences | Participant | -0.458456 | -0.002489 | |
| Investor Activism - Activist Communication | Target | 4.258800 | -0.004300 | |
| Special Calls | Target | 0.789582 | -0.013420 | |
| Strategic Alliances | Target | 3.713280 | -0.002782 | |
| Buyback Transaction Closings | Target | 1.549030 | -0.002613 | |
| M&A Transaction Announcements | Buyer | 3.012090 | -0.002259 | |
| M&A Transaction Closings | Buyer | 3.378730 | -0.001616 | |
| M&A Transaction Announcements | Seller | 2.572670 | -0.002726 | |
| M&A Calls | Target | 8.263100 | -0.002316 | |
| Shareholder/Analyst Calls | Target | -0.028738 | -0.002662 | |
| M&A Transaction Closings | Seller | 1.823550 | -0.001731 | |
| Follow-on Equity Offerings | Target | -5.485400 | -0.003701 | |
| M&A Rumors and Discussions | Target | 4.939740 | -0.001830 | |
| Business Reorganizations | Target | -0.770601 | -0.002331 | |
| Discontinued Operations/Downsizing | Target | -2.599930 | -0.001346 | |
| Fixed Income Offerings | Target | -3.096800 | -0.000678 | |
| Private Placements | Target | 3.816930 | -0.001219 | |
| Delayed SEC Filings | Target | -2.507320 | -0.001142 | |
| Labor-related Announcements | Target | 0.657079 | -0.000550 | |
| M&A Transaction Announcements | Target | 18.389100 | 0.000139 | |
| Earnings Calls | Target | 0.434752 | 0.000274 | |
| Lawsuits & Legal Issues | Target | -0.309857 | 0.000435 | |
| Board Meeting | Target | 0.686676 | 0.001542 | |
| Corporate Guidance - New/Confirmed | Target | -1.229350 | 0.000875 | |
| Seeking Acquisitions/Investments | Target | -0.000492 | 0.001187 | |
| Impairments/Write Offs | Target | -0.905683 | 0.002642 | |
| Company Conference Presentations | Target | 1.353990 | 0.001249 | |
| Announcements of Sales/Trading Statement | Target | 0.831303 | 0.003360 | |
| Considering Multiple Strategic Alternatives | Target | -1.002320 | 0.012381 | |
| Debt Financing Related | Target | 2.456220 | 0.003117 | |
| Buyback Tranche Update | Target | 0.491636 | 0.002297 | |
| Seeking to Sell/Divest | Target | -0.456121 | 0.007285 | |
| End of Lock-Up Period | Target | -9.594920 | 0.007141 | |
| Corporate Guidance - Raised | Target | 10.723600 | 0.005254 | |
| Announcements of Earnings | Target | 0.205462 | 0.003517 | |
| Delistings | Target | -4.433630 | 0.015234 | |

(continues on next page)

(continued from previous page)

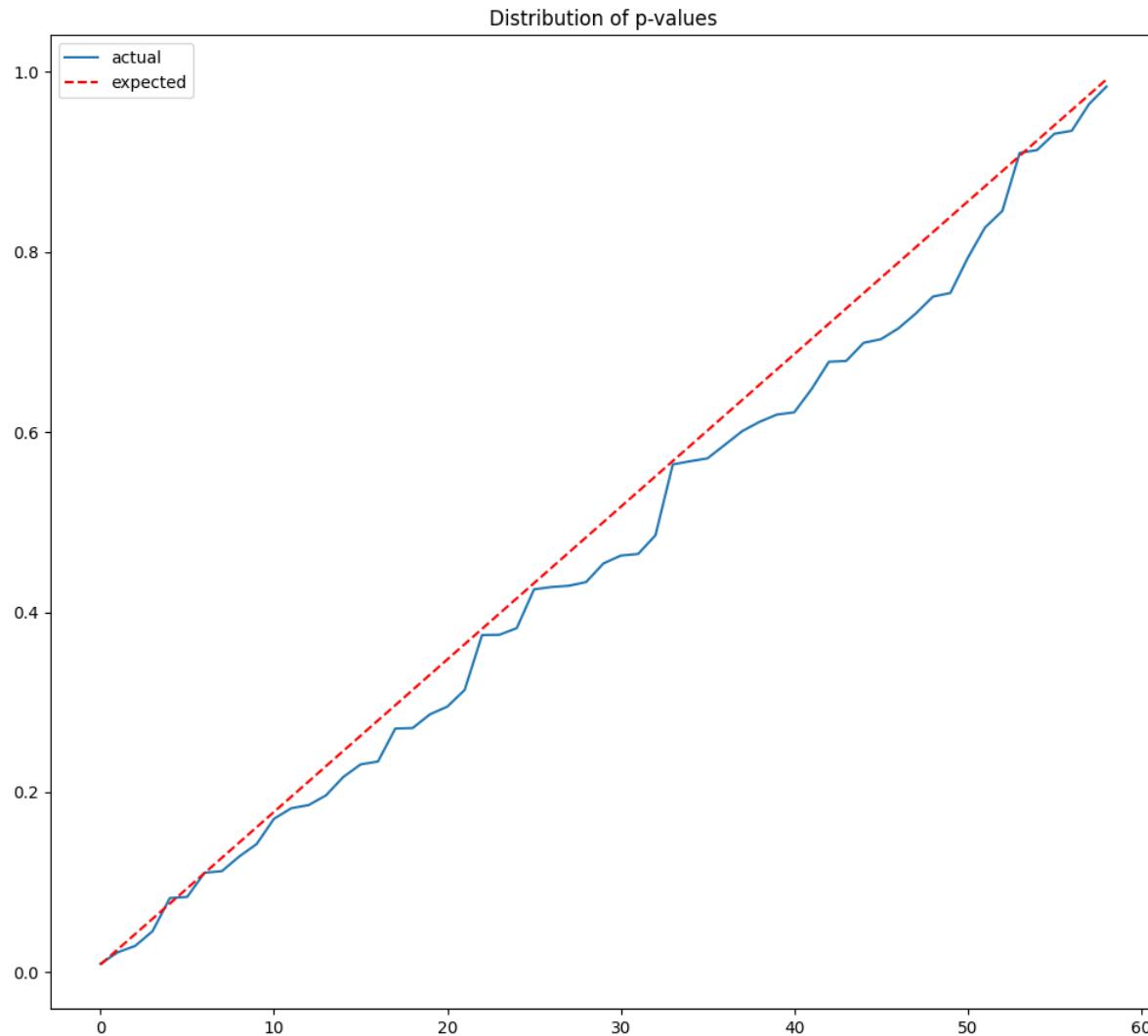
| | | | |
|---|-------------|-----------|----------|
| Analyst/Investor Day | Target | 2.396800 | 0.007048 |
| Investor Activism - Proxy/Voting Related | Target | -0.509462 | 0.010531 |
| Annual General Meeting | Target | 1.224010 | 0.006747 |
| Buyback - Change in Plan Terms | Target | 3.685190 | 0.009255 |
| Buyback Transaction Announcements | Target | 6.943710 | 0.010344 |
| | | | post_t |
| event | role | | |
| Shelf Registration Filings | Target | -2.287190 | |
| Changes in Company Bylaws/Rules | Target | -2.000120 | |
| Auditor Changes | Target | -1.737210 | |
| Name Changes | Target | -1.729880 | |
| Executive Changes - CFO | Target | -1.588070 | |
| Special/Extraordinary Shareholders Meeting | Target | -1.519900 | |
| Product-Related Announcements | Target | -1.467370 | |
| Investor Activism - Target Communication | Target | -1.198240 | |
| Executive/Board Changes - Other | Target | -1.189980 | |
| Executive Changes - CEO | Target | -1.100320 | |
| Index Constituent Adds | Target | -1.065910 | |
| M&A Transaction Cancellations | Buyer | -1.047170 | |
| M&A Transaction Closings | Target | -1.007740 | |
| Business Expansions | Target | -0.888018 | |
| Private Placements | Buyer | -0.887581 | |
| Earnings Release Date | Target | -0.797304 | |
| Corporate Guidance - Lowered | Target | -0.790521 | |
| Client Announcements | Target | -0.783472 | |
| Conferences | Participant | -0.734339 | |
| Investor Activism - Activist Communication | Target | -0.731366 | |
| Special Calls | Target | -0.697699 | |
| Strategic Alliances | Target | -0.576902 | |
| Buyback Transaction Closings | Target | -0.571668 | |
| M&A Transaction Announcements | Buyer | -0.566940 | |
| M&A Transaction Closings | Buyer | -0.544916 | |
| M&A Transaction Announcements | Seller | -0.456992 | |
| M&A Calls | Target | -0.415104 | |
| Shareholder/Analyst Calls | Target | -0.413827 | |
| M&A Transaction Closings | Seller | -0.386578 | |
| Follow-on Equity Offerings | Target | -0.381144 | |
| M&A Rumors and Discussions | Target | -0.365101 | |
| Business Reorganizations | Target | -0.312863 | |
| Discontinued Operations/Downsizing | Target | -0.218363 | |
| Fixed Income Offerings | Target | -0.194733 | |
| Private Placements | Target | -0.113048 | |
| Delayed SEC Filings | Target | -0.086236 | |
| Labor-related Announcements | Target | -0.082097 | |
| M&A Transaction Announcements | Target | 0.020541 | |
| Earnings Calls | Target | 0.044777 | |
| Lawsuits & Legal Issues | Target | 0.109043 | |
| Board Meeting | Target | 0.262207 | |
| Corporate Guidance - New/Confirmed | Target | 0.318072 | |
| Seeking Acquisitions/Investments | Target | 0.343154 | |
| Impairments/Write Offs | Target | 0.493208 | |
| Company Conference Presentations | Target | 0.496738 | |
| Announcements of Sales/Trading Statement | Target | 0.508218 | |
| Considering Multiple Strategic Alternatives | Target | 0.522977 | |
| Debt Financing Related | Target | 0.748678 | |

(continues on next page)

(continued from previous page)

| | | |
|--|--------|----------|
| Buyback Tranche Update | Target | 0.792872 |
| Seeking to Sell/Divest | Target | 0.873905 |
| End of Lock-Up Period | Target | 1.101470 |
| Corporate Guidance - Raised | Target | 1.234920 |
| Announcements of Earnings | Target | 1.291870 |
| Delistings | Target | 1.323590 |
| Analyst/Investor Day | Target | 1.334210 |
| Investor Activism - Proxy/Voting Related | Target | 1.371040 |
| Annual General Meeting | Target | 1.595650 |
| Buyback - Change in Plan Terms | Target | 2.180510 |
| Buyback Transaction Announcements | Target | 2.596150 |

```
# 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.argmin(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 | 0.0094 |

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 | 0.0094 |
| bonferroni | 0 | 0.5562 |

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 | 0.0094 |

(continues on next page)

(continued from previous page)

| | | |
|------------|---|--------|
| bonferroni | 0 | 0.5562 |
| holm | 0 | 0.5562 |

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.

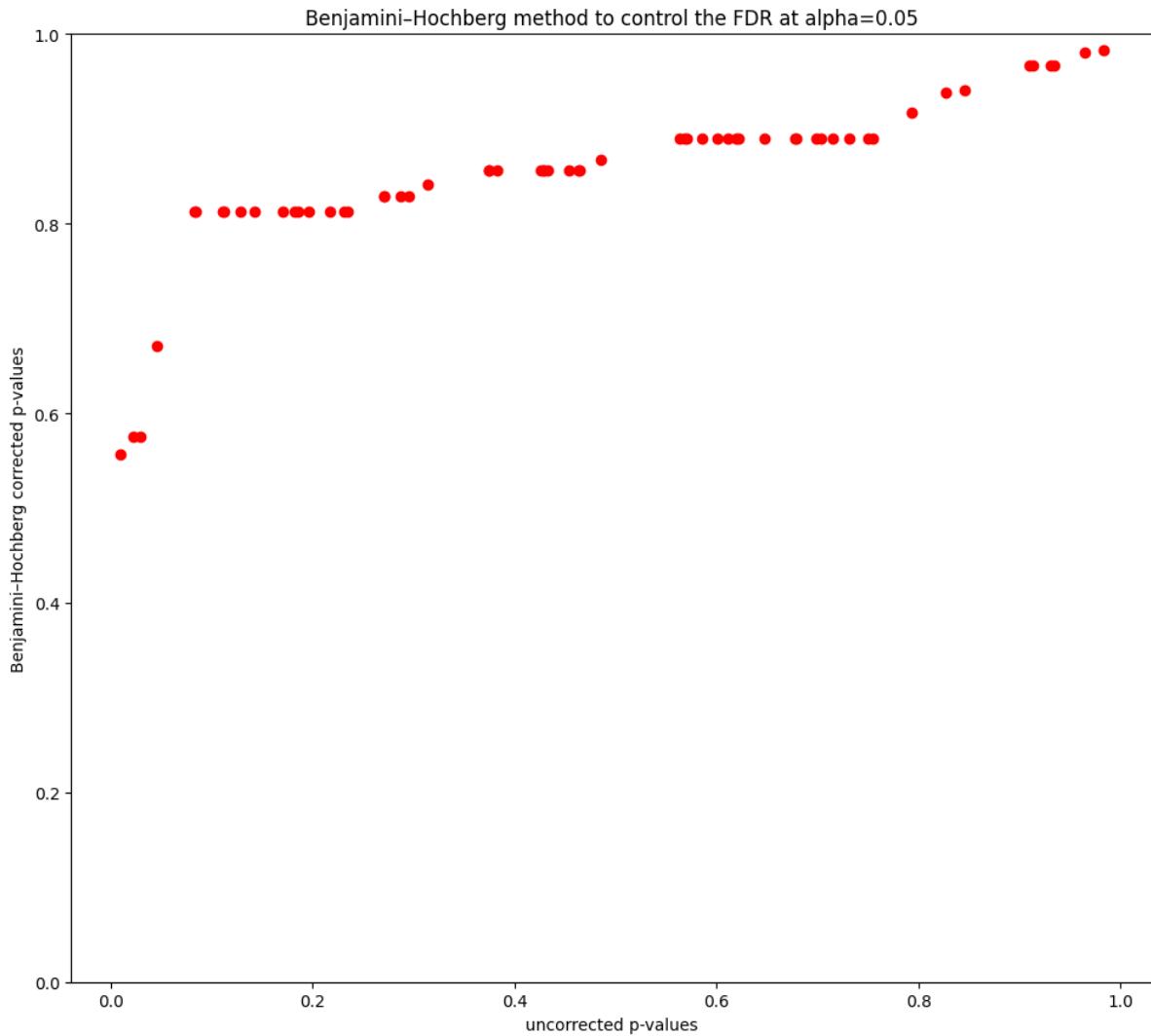
Similar to the Holm test, Benjamin-Hochberg also relies on the distribution of test statistics. However, in contrast to the Holm test that begins with the most significant test, the Benjamin-Hochberg approach starts with the least significant. We sort the tests from the lowest p-value (most significant) 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 | 0.0094 |
| bonferroni | 0 | 0.5562 |
| holm | 0 | 0.5562 |
| fdr_bh | 0 | 0.5562 |

```
# 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()
```



ECONOMIC DATA REVISIONS

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

Concepts:

- Retrieving data from the web
- Vintage economic data and revisions
- Payrolls employment indices
- St Louis Fed FRED, Archival FRED, and FRED-MD
- Outliers

References:

- <https://fred.stlouisfed.org/>
- <https://research.stlouisfed.org/econ/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>

```
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 secret import credentials
VERBOSE = 0
# %matplotlib qt
```

8.1 Retrieving data from the web

Data may be retrieved from the web by

1. Retrieving a structured file in csv, excel, json or any other format
2. Scraping a web page and searching for information near particular html tags or textual patterns
3. Calling API's provided to developers – these are a set of protocols with which you can communicate with the web server

8.1.1 Retrieve a formatted file

```
# This URL is the location of the FRED-MD csv file, maintained by researchers at the
# St Louis FRED
url = 'https://files.stlouisfed.org/files/htdocs/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 | DPCEA3M086SBEA | 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 | |
| .. | ... | ... | ... | ... | ... | ... |
| 779 | 11/1/2023 | 19225.440 | 15860.3 | 117.258 | 1.514120e+06 | |
| 780 | 12/1/2023 | 19265.016 | 15893.6 | 117.796 | 1.523993e+06 | |
| 781 | 1/1/2024 | 19382.171 | 15935.5 | 117.425 | 1.504096e+06 | |
| 782 | 2/1/2024 | 19370.105 | 15913.7 | 117.991 | 1.511532e+06 | |
| 783 | 3/1/2024 | 19407.154 | 15950.6 | 118.597 | NaN | |
| | | | | | | |
| | RETAILx | INDPRO | IPFPNSS | IPFINAL | IPCONGD | ... |
| 0 | 5.00000 | 5.0000 | 5.0000 | 5.0000 | 5.0000 | ... |
| 1 | 18235.77392 | 21.9665 | 23.3891 | 22.2688 | 31.7011 | ... |
| 2 | 18369.56308 | 22.3966 | 23.7048 | 22.4617 | 31.9337 | ... |
| 3 | 18523.05762 | 22.7193 | 23.8483 | 22.5719 | 31.9337 | ... |
| 4 | 18534.46600 | 23.2032 | 24.1927 | 22.9026 | 32.4374 | ... |
| .. | ... | ... | ... | ... | ... | ... |
| 779 | 700707.00000 | 102.9382 | 101.2601 | 101.7148 | 102.3259 | ... |
| 780 | 703256.00000 | 102.6149 | 100.8179 | 101.2544 | 101.7841 | ... |
| 781 | 695631.00000 | 101.8110 | 100.3736 | 100.9190 | 101.6965 | ... |
| 782 | 702712.00000 | 102.2599 | 100.5583 | 100.7451 | 100.8224 | ... |
| 783 | 707682.00000 | 102.6577 | 101.3064 | 101.7222 | 102.0617 | ... |
| | | | | | | |
| | DNDGRG3M086SBEA | DSERRG3M086SBEA | CES0600000008 | CES2000000008 | \ | |
| 0 | 6.000 | 6.000 | 6.00 | 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 | | |

(continues on next page)

(continued from previous page)

```

...
779      119.324      124.533      30.26      34.96
780      119.192      124.912      30.42      34.98
781      118.755      125.799      30.57      35.30
782      119.537      126.130      30.68      35.24
783      119.761      126.646      30.84      35.42

CES3000000008  UMCSENTx  DTCOLNVHFNM  DTCTHFNM  INVEST  VIXCL$x
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
...
779      26.89      61.3      517477.03  918934.92  5008.1757  13.8563
780      27.13      69.7      521938.91  921959.34  5085.0478  12.6960
781      27.21      79.0      524940.39  924448.27  5112.8331  13.3453
782      27.36      76.9      527544.90  925641.71  5111.0308  13.8808
783      27.48      79.4          NaN      NaN    5207.6646  13.7658

[784 rows x 127 columns]

```

8.1.2 Scrape a web page

```
# 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
# methods
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>\n        Economic Data Series by Tag | FRED | St. Louis Fed</title>
```

```
# 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])
```

```
# 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
```

[10-Year Treasury Constant Maturity Minus 2-Year Treasury Constant Maturity](/series/T10Y2Y)

```
# 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'

8.1.3 Call an API

```
# 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": "2024-05-03", "realtime_end": "2024-05-03", "serieses": [{"id": "T10Y2Y", "realtime_start": "2024-05-03", "realtime_end": "2024-05-03", "title": "10-Year Treasury Constant Maturity Minus 2-Year Treasury Constant Maturity", "observation_start": "1976-06-01", "observation_end": "2024-05-02", "frequency": "Daily", "frequency_short": "D", "units": "Percent", "units_short": "%", "seasonal_adjustment": "Not Seasonally Adjusted", "seasonal_adjustment_short": "NSA", "last_updated": "2024-05-02 16:01:10-05", "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': '2024-05-03',
'realtime_end': '2024-05-03',
'serieses': [{"id': 'T10Y2Y',
'realtime_start': '2024-05-03',
'realtime_end': '2024-05-03',
'title': '10-Year Treasury Constant Maturity Minus 2-Year Treasury Constant Maturity',
'observation_start': '1976-06-01',
'observation_end': '2024-05-02',
'frequency': 'Daily',
'frequency_short': 'D',
'units': 'Percent',
'units_short': '%',
'seasonal_adjustment': 'Not Seasonally Adjusted',
'seasonal_adjustment_short': 'NSA',
'last_updated': '2024-05-02 16:01:10-05',
'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['serieses'])
df
```

```

      id realtime_start realtime_end \
0  T10Y2Y      2024-05-03  2024-05-03

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

      observation_end frequency frequency_short      units units_short \
0      2024-05-02      Daily             D    Percent      %

      seasonal_adjustment seasonal_adjustment_short      last_updated \
0  Not Seasonally Adjusted                           NSA  2024-05-02 16:01:10-05

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

```

8.2 FRED

Short for Federal Reserve Economic Data, FRED is an online database consisting of hundreds of thousands of economic data time series from national, international, public, and private sources. Created and maintained by the Research Department at the Federal Reserve Bank of St. Louis, FRED data can be downloaded from their website, an Excel add-in, or through api calls.

Archival FRED, or ALFRED, is a superset of FRED that allows you to retrieve vintage versions of economic data that were available on specific dates in history.

```

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

```

8.2.1 Popular FRED series

In addition to being tagged by categories, frequencies, seasonal adjustments, sources and numerous concepts which can be filtered on, FRED series titles can also be sorted by update recency and popularity. The most popular series currently are:

- <https://fred.stlouisfed.org/tags/series?ob=pv&pageID=1>

```

# scrape FRED most popular page
popular = {}
titles = Alfred.popular(1)
for title in titles:
    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 20240503

| | title | popularity |
|--------|---|------------|
| T10Y2Y | 10-Year Treasury Constant Maturity Minus 2-Yea... | 100 |

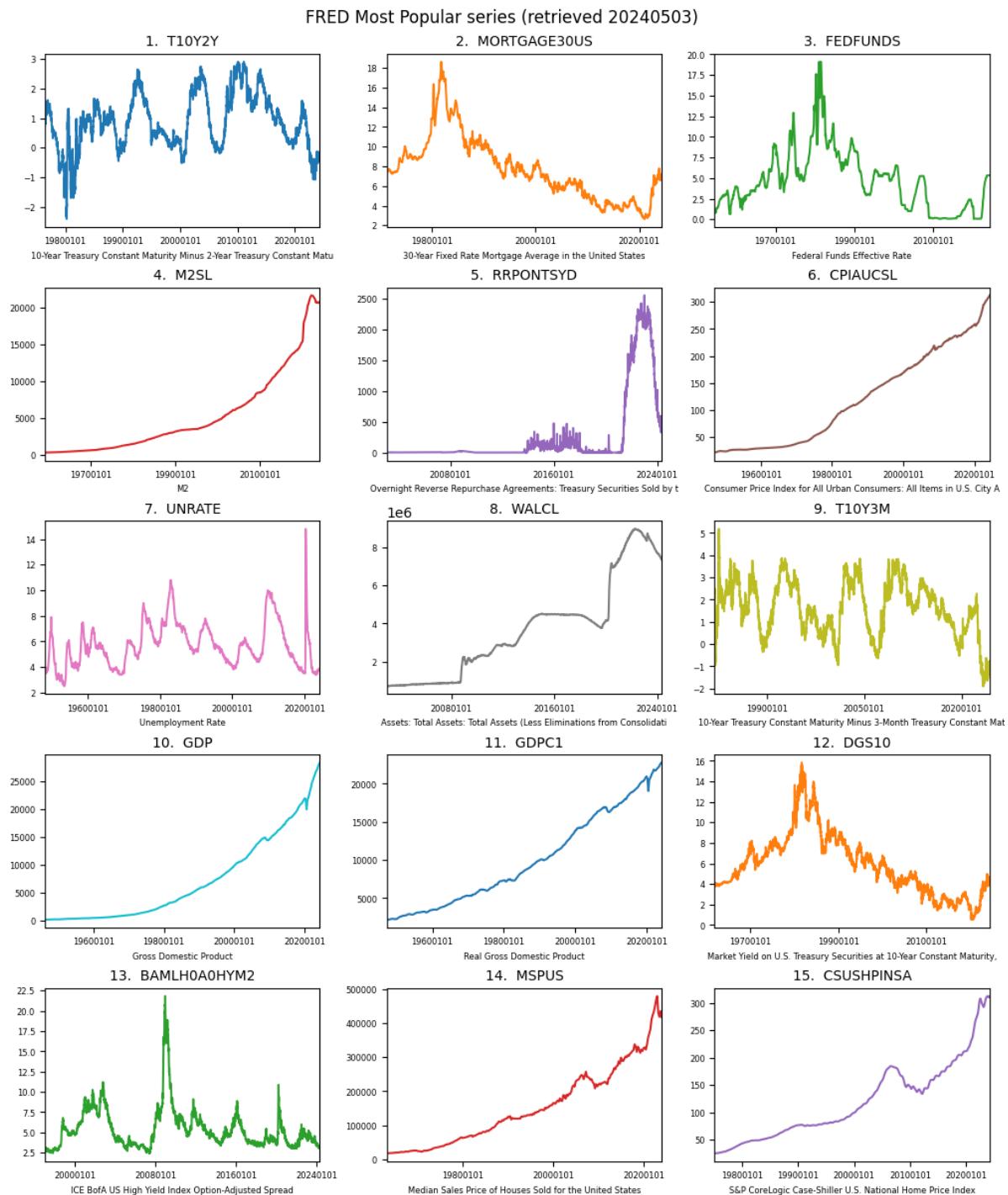
(continues on next page)

(continued from previous page)

| | | |
|---------------|---|----|
| 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 20240503)')
```



8.2.2 FRED series categories

Retrieve Total Non-Farm Payroll (PAYEMS) and its parent 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/empsit.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) |
| 32312: | Durable Goods | (children=0, series=63) |

(continues on next page)

(continued from previous page)

```
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 (1939-01-01-2024-03-01)

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 closing of schools. The Bureau of Labor Statistics (BLS) adjusts the

(continues on next page)

(continued from previous page)

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

8.2.3 Revisions and vintage dates

Economic data for past observation periods are revised as more accurate estimates become available. As a result, previous vintages of data can be superseded. ALFRED, the archival FRED tool, captures these data revisions.

On the first Friday of every month, the Bureau of Labor Statistics (BLS) publishes its Employment Situation Summary, which contains the highly-anticipated Total Nonfarm Payroll Employment based on firms' reports of the number of people employed. However, it is a very rough estimate and is revised the following month, and then again the month after as more information arrives at the BLS. These revisions can be large and may materially change the picture of the economy.

```
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 20240503):

| | PAYEMS | realtime_start | realtime_end |
|----------|--------|----------------|--------------|
| date | | | |
| 20230131 | 154773 | 20240202 | 99991231 |
| 20230228 | 155060 | 20240202 | 99991231 |
| 20230331 | 155206 | 20240202 | 99991231 |
| 20230430 | 155484 | 20240202 | 99991231 |
| 20230531 | 155787 | 20240202 | 99991231 |
| 20230630 | 156027 | 20240202 | 99991231 |
| 20230731 | 156211 | 20240202 | 99991231 |
| 20230831 | 156421 | 20240202 | 99991231 |
| 20230930 | 156667 | 20240202 | 99991231 |
| 20231031 | 156832 | 20240202 | 99991231 |
| 20231130 | 157014 | 20240202 | 99991231 |
| 20231231 | 157304 | 20240308 | 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 | 99991231 |
| 20231231 | 157304 | 20240308 | 99991231 |

```
print("Fourth Release:")
data[3] = alf(series_id, release=4, start=start, end=end, freq=freq, realtime=True)
data[3]
```

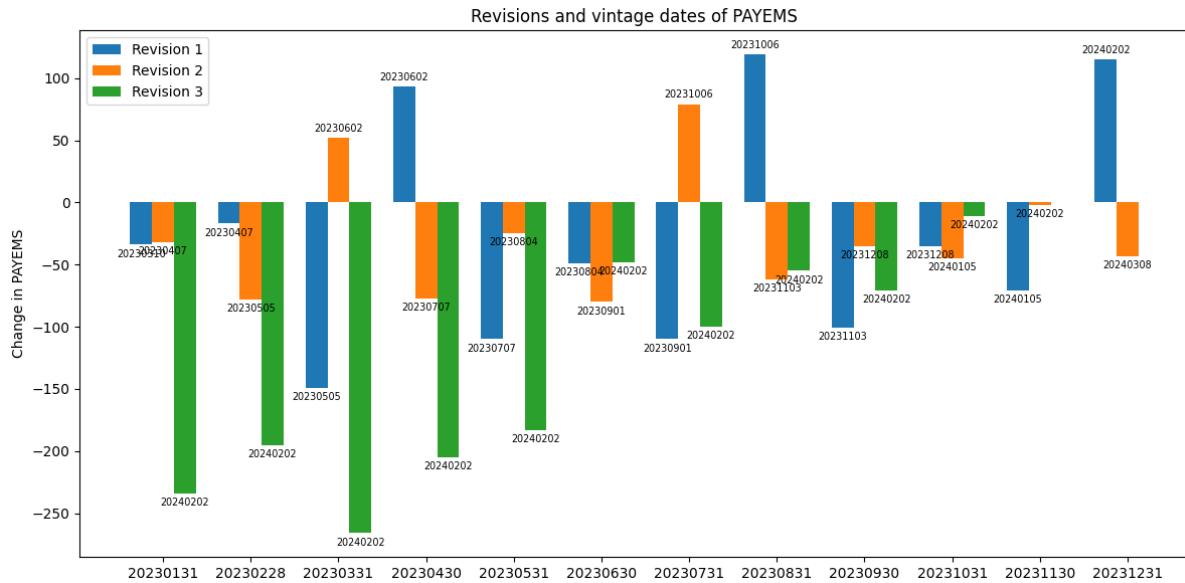
Fourth Release:

| | PAYEMS | realtime_start | realtime_end |
|----------|--------|----------------|--------------|
| date | | | |
| 20230131 | 154773 | 20240202 | 99991231 |
| 20230228 | 155060 | 20240202 | 99991231 |
| 20230331 | 155206 | 20240202 | 99991231 |
| 20230430 | 155484 | 20240202 | 99991231 |
| 20230531 | 155787 | 20240202 | 99991231 |
| 20230630 | 156027 | 20240202 | 99991231 |
| 20230731 | 156211 | 20240202 | 99991231 |
| 20230831 | 156421 | 20240202 | 99991231 |
| 20230930 | 156667 | 20240202 | 99991231 |
| 20231031 | 156832 | 20240202 | 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 | -349.0 |
| Revision 2 | -348.0 |
| Revision 3 | -1368.0 |

```
#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()
```



TODO: Explain, and initial and final ADP and JOLTS cumulative graphs over past year

8.3 FRED-MD and FRED-QD

FRED-MD and FRED-QD are datasets of monthly and quarterly observations mimic the coverage of macroeconomic datasets already used in the literature. They are updated in real-time through the FRED database, and relieve the researcher of the task of incorporating data changes and revisions (a task accomplished by the data desk at the Federal Reserve Bank of St. Louis).

8.3.1 Release dates

```
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,
                                     release=1,
                                     start=end-4, # within 4 days of monthend
                                     end=end,
                                     realtime=True)
                               for sub in Alfred.splice_[title]])
# This block handles the quarterly data
for title in md_df.columns:
    if title in alf.splice_:
        if isinstance(Alfred.splice_[title], str):
            out[title] = alf(series_id=Alfred.splice_[title],
                              release=1,
                              start=end-4, # within 4 days of monthend
                              end=end,
                              realtime=True)
        else:
            for sub in Alfred.splice_[title]:
                out[title] = pd.concat([out[title],
                                      alf(series_id=sub,
                                          release=1,
                                          start=end-4, # within 4 days of monthend
                                          end=end,
                                          realtime=True)])
    else:
        if title in md_df.index:
            out[title] = alf(series_id=title,
                              release=1,
                              start=end-4, # within 4 days of monthend
                              end=end,
                              realtime=True)
# This block handles the monthly data
for title in md_df.columns:
    if title in alf.splice_:
        if isinstance(Alfred.splice_[title], str):
            out[title] = alf(series_id=Alfred.splice_[title],
                              release=1,
                              start=end-4, # within 4 days of monthend
                              end=end,
                              realtime=True)
        else:
            for sub in Alfred.splice_[title]:
                out[title] = pd.concat([out[title],
                                      alf(series_id=sub,
                                          release=1,
                                          start=end-4, # within 4 days of monthend
                                          end=end,
                                          realtime=True)])
    else:
        if title in md_df.index:
            out[title] = alf(series_id=title,
                              release=1,
                              start=end-4, # within 4 days of monthend
                              end=end,
                              realtime=True)
```

(continues on next page)

(continued from previous page)

```
start=end-4,  # within 4 days of monthend
end=end,
realtime=True)
for sub in Alfred.splice_[title][1:]]
```

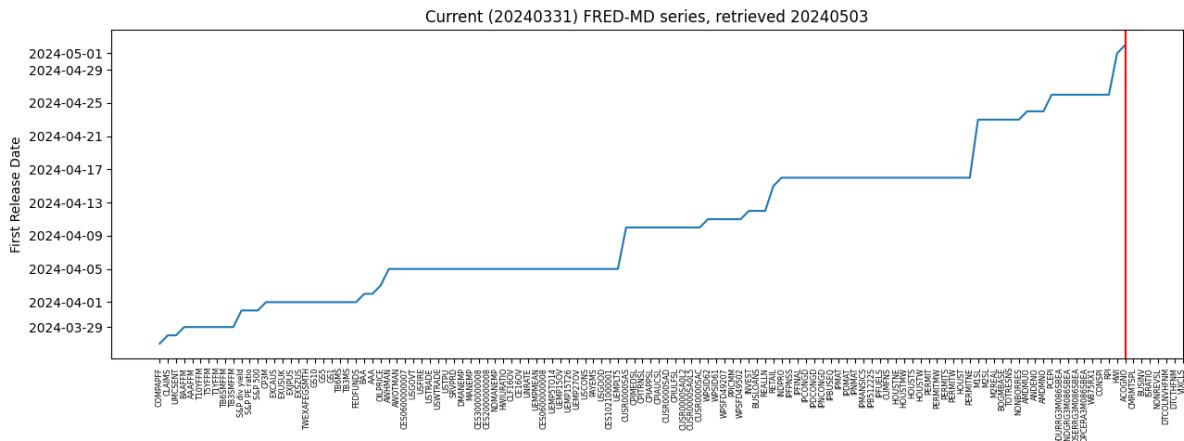
FRED-MD vintage: monthly/current.csv

```
# 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
20231231    1523993.0
20240131    1504096.0
20240229    1511532.0
Name: CMRMTSPL, dtype: float64
date
20240131    243376.0
20240229    249075.0
20240331    251771.0
Name: ACOGNO, dtype: float64
date
20231231    2555879.0
20240131    2556109.0
20240229    2567540.0
Name: BUSINV, dtype: float64
date
20231231    1.38
20240131    1.39
20240229    1.38
Name: ISRATIO, dtype: float64
date
20231231    3700.45907
20240131    3709.54790
20240229    3712.40931
Name: NONREVSL, dtype: float64
date
20240131    0.156926
20240229    0.156615
20240331    NaN
Name: CONSPI, dtype: float64
date
20240331    34.37
20240430    34.66
20240531    33.62
Name: S&P PE ratio, dtype: float64
date
20231231    521938.91
20240131    524940.39
20240229    527544.90
Name: DTCOLNVHFNM, dtype: float64
date
20231231    921959.34
20240131    924448.27
20240229    925641.71
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))
```

```
(HWI, Help Wanted Index for United States)
  20240501
(ACOGNO, Manufacturers' New Orders: Consumer Goods)
  20240502
```

(continues on next page)

(continued from previous page)

```
(CMRMTSPL, Real Manufacturing and Trade Industries Sales)      ↴
    ↵      None
(BUSINV, Total Business Inventories)      ↴
    ↵      None
(ISRATIO, Total Business: Inventories to Sales Ratio)      ↴
    ↵      None
(NONREVSL, Nonrevolving Consumer Credit Owned and Securitized)      ↴
    ↵      None
(DTCOLNVHFN, Consumer Motor Vehicle Loans Owned by Finance Companies, Level)      ↴
    ↵      None
(DTCTHFN, Total Consumer Loans and Leases Owned and Securitized by Finance Companies, Level)      ↴
    ↵      None
(VIXCLS, CBOE Volatility Index: VIX)      ↴
    ↵      None
dtype: object
```

8.4 Outliers

- John Tukey proposed this criterian, where $k = 1.5$ indicates an “outlier”, and $k = 3$ indicates data that is “far out”.
- iq10: require within median \pm [10 times interquartile range $Q3-Q1$]
- tukey: $[Q1 - 1.5(Q3-Q1), Q3 + 1.5(Q3-Q1)]$
- farout : tukey with $3IQ$ instead of $1.5IQ$

TODO:

- separate graphs by method (too busy) – number of outliers not fraction
- list of months with most combined outliers, show number of each

```
outliers = dict()
for method in ['iq10', 'tukey', 'farout']:
    outliers[method] = np.sum(is_outlier(data, method=method), axis=1)
fig, ax = plt.subplots(figsize=(10, 6))
plt.step(data.index[12:], DataFrame(outliers)[12:] / data.shape[1], where='mid')
plt.title('Fraction of outlier observations by month')
plt.legend(outliers.keys())
plt.tight_layout()
```

```
print('Frequency of outliers by method')
for method, num in outliers.items():
    print(f" {method:6s}: {np.mean(num):5.1f}/{data.shape[1]} outliers per month")
```


LINEAR REGRESSION DIAGNOSTICS

The only thing we know about the future is that it will be different - Peter Drucker

Concepts:

- Linear regression diagnostics
- HAC robust standard errors
- Multicollinearity
- Residual analysis
- Consumer and Producer Prices Indices

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

```
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'])
```

Retrieve monthly Consumer Price Index as the endogenous variable, and Producer Price Index as the exogenous. We use the monthly differences of logs of both series.

```
# 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

1. $E[y_i] = b_0 + b_1 x_{i1} + \dots + b_k x_{ik}$. &
2. $\{x_{i1}, \dots, x_{ik}\}$ are non-stochastic variables. &
3. $Var(y_i) = \sigma^2$. &
4. $\{y_i\}$ are independent random variables. &
5. $\{y_i\}$ are normally distributed. &

Under assumptions 1-4, the least squares regression estimator $b = (X'X)^{-1}X'y$ is an unbiased estimator of the parameter vector b ;

- has variance-covariance matrix $Var(b) = \sigma^2(X'X)^{-1}$,
- and standard error $b_j \text{ se}(b_j) = \sigma \sqrt{(X'X)^{-1}_{[j+1,j+1]}}$.
- Under assumptions 1-5, the least squares estimator is normally distributed.

```
# 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: | 0.455 | | |
| Model: | OLS | Adj. R-squared: | 0.452 | | |
| Method: | Least Squares | F-statistic: | 165.4 | | |
| Date: | Tue, 09 Apr 2024 | Prob (F-statistic): | 5.95e-78 | | |
| Time: | 08:59:07 | Log-Likelihood: | 2761.8 | | |
| No. Observations: | 599 | AIC: | -5516. | | |
| Df Residuals: | 595 | BIC: | -5498. | | |
| Df Model: | 3 | | | | |
| Covariance Type: | nonrobust | | | | |
| ===== | | | | | |
| | coef | std err | t | P> t | [0.025 |
| 975] | | | | | 0. |

(continues on next page)

(continued from previous page)

| ----- | | | | | | |
|--------------------|----------------|-------------------|----------------|-------|-----------|----|
| Intercept | 0.0009 | 0.000 | 6.203 | 0.000 | 0.001 | 0. |
| ⁰⁰¹ | ⁶⁶³ | ⁰³⁵ | ²⁷¹ | | | |
| CPIAUCSL.shift(1) | 0.5815 | 0.041 | 14.049 | 0.000 | 0.500 | 0. |
| ⁶⁶³ | | | | | | |
| CPIAUCSL.shift(2) | -0.0454 | 0.041 | -1.103 | 0.271 | -0.126 | 0. |
| ⁰³⁵ | | | | | | |
| WPSFD4131.shift(1) | 0.2001 | 0.036 | 5.549 | 0.000 | 0.129 | 0. |
| ²⁷¹ | | | | | | |
| ===== | | | | | | |
| Omnibus: | 113.351 | Durbin-Watson: | | | 2.052 | |
| Prob(Omnibus): | 0.000 | Jarque-Bera (JB): | | | 548.449 | |
| Skew: | -0.750 | Prob(JB): | | | 8.05e-120 | |
| Kurtosis: | 7.442 | Cond. No. | | | 520. | |
| ===== | | | | | | |

Notes:

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

9.1.1 Heteroskedasity and HAC robust errors

When the variances of residuals are not constant, than the variance of the coefficients is not given by the standard OLS estimate $\sigma^2(X'X)^{-1}$. The coefficient estimates are consistent, but their covariance matrix is of the form $(X'X)^{-1}(X'\Omega X)(X'X)^{-1}$ where a number of choices have been proposed for the choice Ω . When estimated as the diagonal of squared residuals, this provides White's (1980) estimator, often referred to as the heteroskedasticity-consistent or sandwich estimator. Other estimators take into account the effect of leverage points in the design matrix.

If the error terms are serially correlated, then statistical inference using the usual heteroskedasticity-robust standard errors can be misleading. Newy and West (1987) proposed the Heteroskedasticity and Autocorrelation Consistent (HAC) estimator which adjusts for serially correlated errors by applying a weighting scheme to estimates of $m - 1$ autocorrelation coefficients. A rule of thumb for choosing the truncation parameter is $m = 0.75T^{1/3}$, where the autocorrelation coefficients are weighted with $1 + 2 \sum_{j=1}^{m-1} \frac{m-j}{m} \hat{\rho}_j$

```
robust = model.get_robustcov_results(cov_type='HAC', use_t=None, maxlags=0)
print(robust.summary())
```

| OLS Regression Results | | | | | | |
|------------------------|------------------|---------------------|---|------|----------|----|
| Dep. Variable: | CPIAUCSL | R-squared: | | | 0.455 | |
| Model: | OLS | Adj. R-squared: | | | 0.452 | |
| Method: | Least Squares | F-statistic: | | | 111.5 | |
| Date: | Tue, 09 Apr 2024 | Prob (F-statistic): | | | 2.75e-57 | |
| Time: | 08:59:07 | Log-Likelihood: | | | 2761.8 | |
| No. Observations: | 599 | AIC: | | | -5516. | |
| Df Residuals: | 595 | BIC: | | | -5498. | |
| Df Model: | 3 | | | | | |
| Covariance Type: | HAC | | | | | |
| ===== | | | | | | |
| ^{975]} | coef | std err | t | P> t | [0.025 | 0. |

(continues on next page)

(continued from previous page)

```

-----
 ↵
Intercept      0.0009    0.000    5.739    0.000    0.001    0.
 ↵001
CPIAUCSL.shift(1) 0.5815    0.074    7.876    0.000    0.437    0.
 ↵727
CPIAUCSL.shift(2) -0.0454    0.053   -0.849    0.396   -0.150    0.
 ↵060
WPSFD4131.shift(1) 0.2001    0.055    3.628    0.000    0.092    0.
 ↵308
=====
Omnibus:           113.351   Durbin-Watson:          2.052
Prob(Omnibus):    0.000    Jarque-Bera (JB):      548.449
Skew:              -0.750    Prob(JB):            8.05e-120
Kurtosis:          7.442    Cond. No.           520.
=====
```

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 or numerical scaling issues may cause the design matrix to have a high condition number.

The Variance Inflation Factor (VIF) measures the extent to which the variance of an estimated regression coefficient is increased due to multicollinearity in a multiple regression analysis. The VIF for each predictor variable X_i is derived from the R^2 value obtained by regressing X_i against all other predictor variables.

$$VIF_i = \frac{1}{1 - R_i^2}$$

One recommendation is that if VIF is greater than 5 or 10, then the explanatory variable is highly collinear with the other explanatory variables, and the parameter estimates will have large standard errors because of this.

```

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) | 3.424302 |
| CPIAUCSL.shift(2) | 3.408784 |
| WPSFD4131.shift(1) | 2.283663 |

9.1.3 Omitted variables

Omitting a variable with a non-zero coefficient in the true model generating the data has two effects.

1. The remaining variables absorb the effects of the omitted variable attributable to common variation. This changes the regression coefficients on the included variables so that they do not consistently estimate the effect of a change in the explanatory variable on the dependent variable. If the variables are highly correlated, then the bias is large. But if the omitted variable is not correlated to the other variables, then those other coefficients can be estimated consistently.
2. The estimated residuals are larger in magnitude than the true shocks. This is because the residuals contain both the true shock and any effect of the omitted variable that cannot be captured by the included variable.

An extraneous (also known as a superfluous or irrelevant) variable is one that is included in the model but is not needed. This type of variable has a true coefficient of 0 and is consistently estimated to be 0 in large samples. However, standard errors of the relevant variables grows as their correlation with the extraneous variables increases.

9.2 Residual plots

Residual plots are standard methods used to detect deficiencies in a model specification. 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. Both outliers and model specification problems can be detected in residual plots.

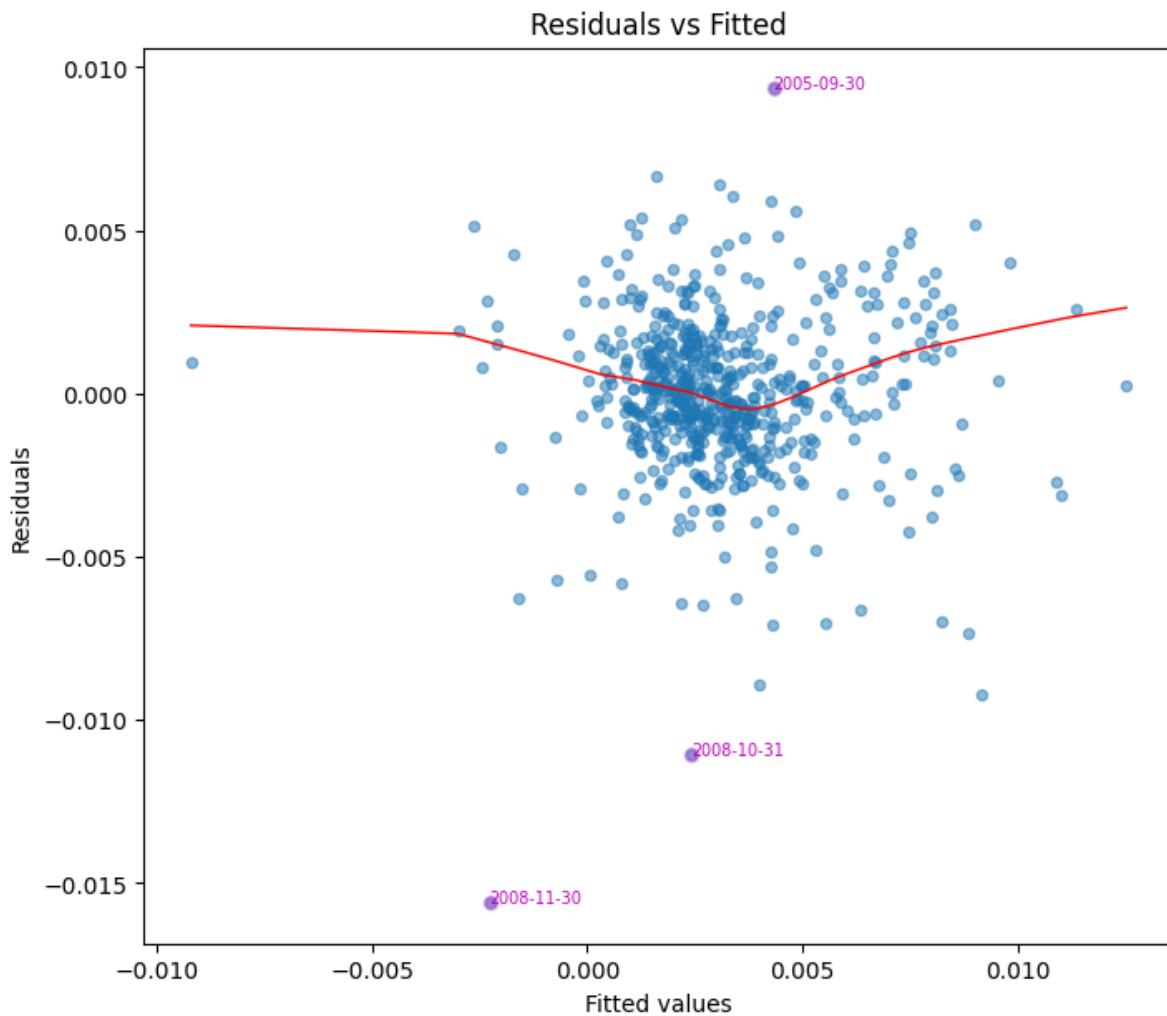
9.2.1 Residuals vs fitted plot

The Residuals vs Fitted plot shows if residuals have non-linear patterns: the residual data points should be evenly spread around a flat fitted line.

```
# 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 | 2005-09-30 | 2008-10-31 | 2008-11-30 |
|----------|------------|------------|------------|
| outliers | 0.00935 | -0.011058 | -0.015598 |



9.2.2 QQ plot of residuals

The Normal Q-Q plot shows if residuals are normally distributed: the standardized data points should lie along the 45-degree line representing the theoretical quantiles of the standard normal distribution.

Outliers, or data points that differ significantly from the others, may indicate some measurement error or anomaly, or that the population has a non-normal heavy-tailed distribution.

```
fig, ax = plt.subplots(clear=True, figsize=(8, 7))
plot_qq(model.resid, ax=ax)
```

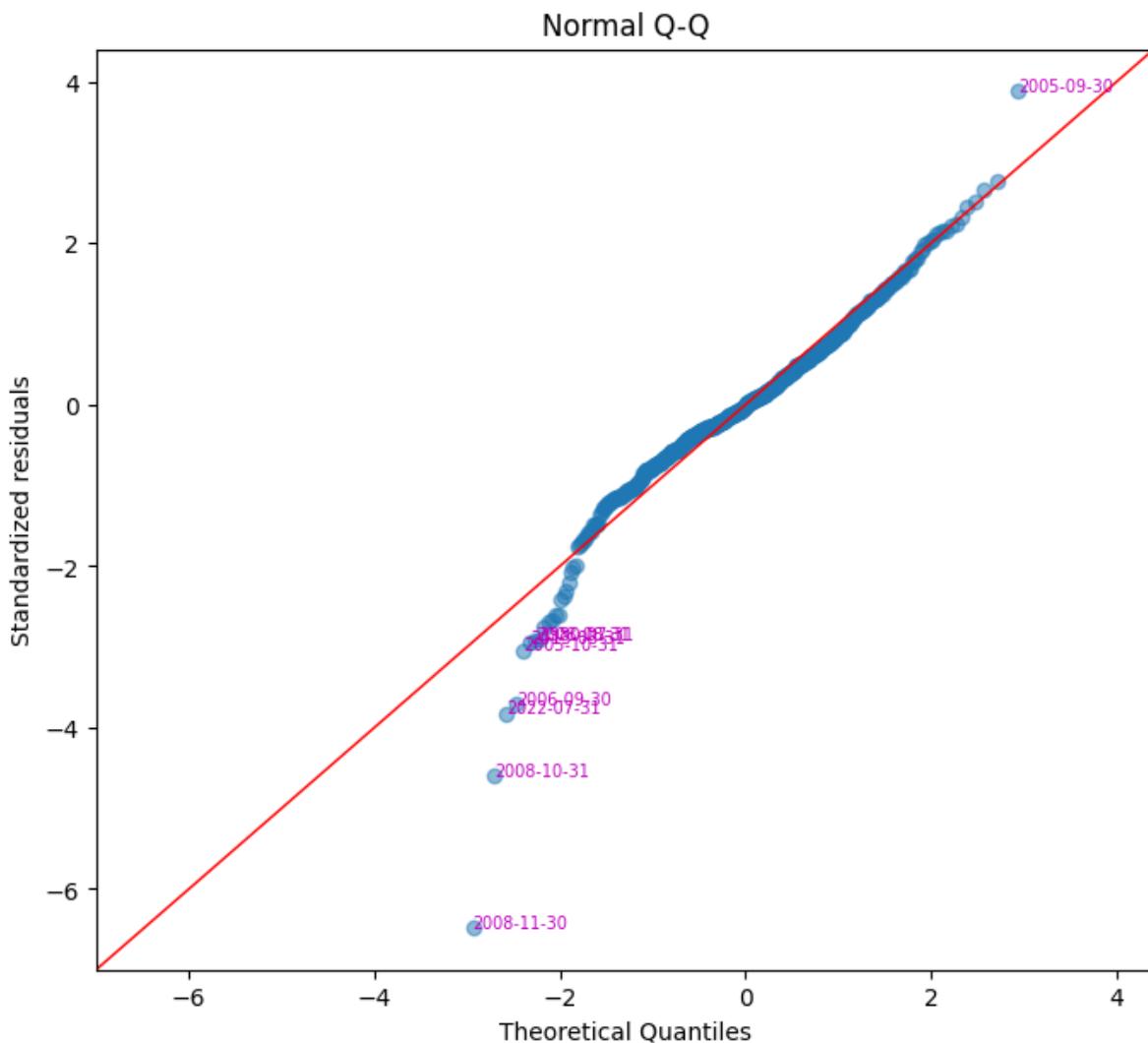
```
/home/terence/env3.11/lib/python3.11/site-packages/statsmodels/graphics/gofplots.
  ↪py:1045: UserWarning: color is redundantly defined by the 'color' keyword
  ↪argument and the fmt string "b" (-> color=(0.0, 0.0, 1.0, 1)). The keyword
  ↪argument will take precedence.
  ax.plot(x, y, fmt, **plot_style)
```

| | residuals | standardized |
|------|-----------|--------------|
| date | | |

(continues on next page)

(continued from previous page)

| | | |
|------------|-----------|-----------|
| 2008-11-30 | -0.015598 | -6.482161 |
| 2008-10-31 | -0.011058 | -4.595524 |
| 2022-07-31 | -0.009212 | -3.828462 |
| 2006-09-30 | -0.008912 | -3.703545 |
| 2005-10-31 | -0.007328 | -3.045586 |
| 2013-03-31 | -0.007109 | -2.954440 |
| 2008-08-31 | -0.007021 | -2.917872 |
| 1980-07-31 | -0.007007 | -2.911938 |
| 2005-09-30 | 0.009350 | 3.885717 |

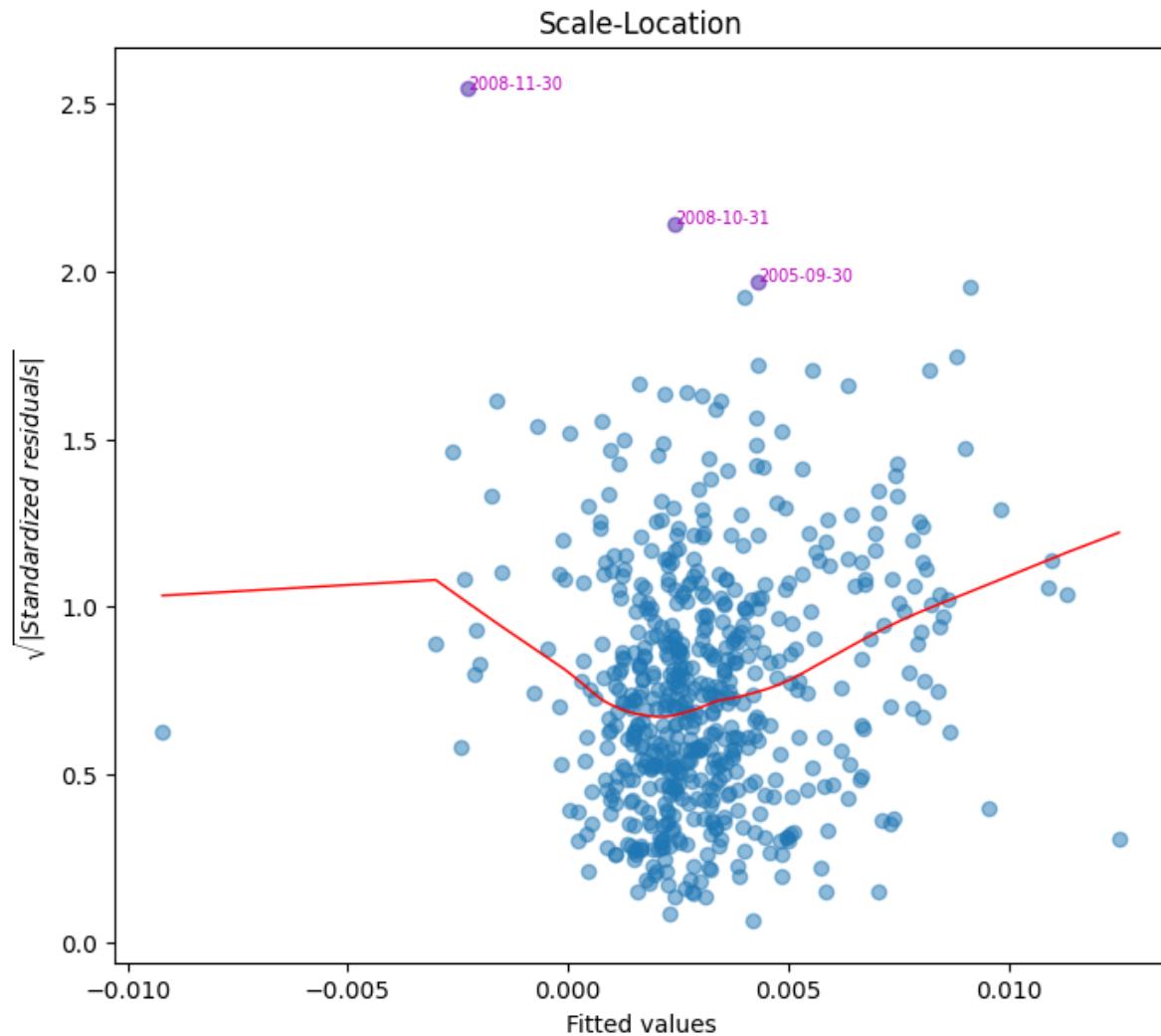


9.2.3 Scale of residuals

The Scale-Location plot shows if residuals are spread equally along the range of predictors, around a flat fitted line: it checks the assumption of equal variance or homoscedasticity.

```
fig, ax = plt.subplots(clear=True, figsize=(8, 7))
plot_scale(model.fittedvalues, model.resid, ax=ax)
```

```
array([377, 414, 415])
```



9.2.4 Leverage and influential points

The projection matrix from the least squares estimator $H = X(X^T)^{-1}X^T$ is also called the **hat matrix** in statistics, and the i -th diagonal element of H , given by $h_{ii} = x_i^T(X^T)^{-1}x_i$, is the **leverage** of the i -th observation.

Even though the data may have extreme leverage or outlier values, they might not be influential on the regression fit. **Cook's Distance** can be expressed using the leverage and the squared internally studentized residual $D_i = \frac{1}{p}t_i^2 \frac{h_{ii}}{1-h_{ii}}$

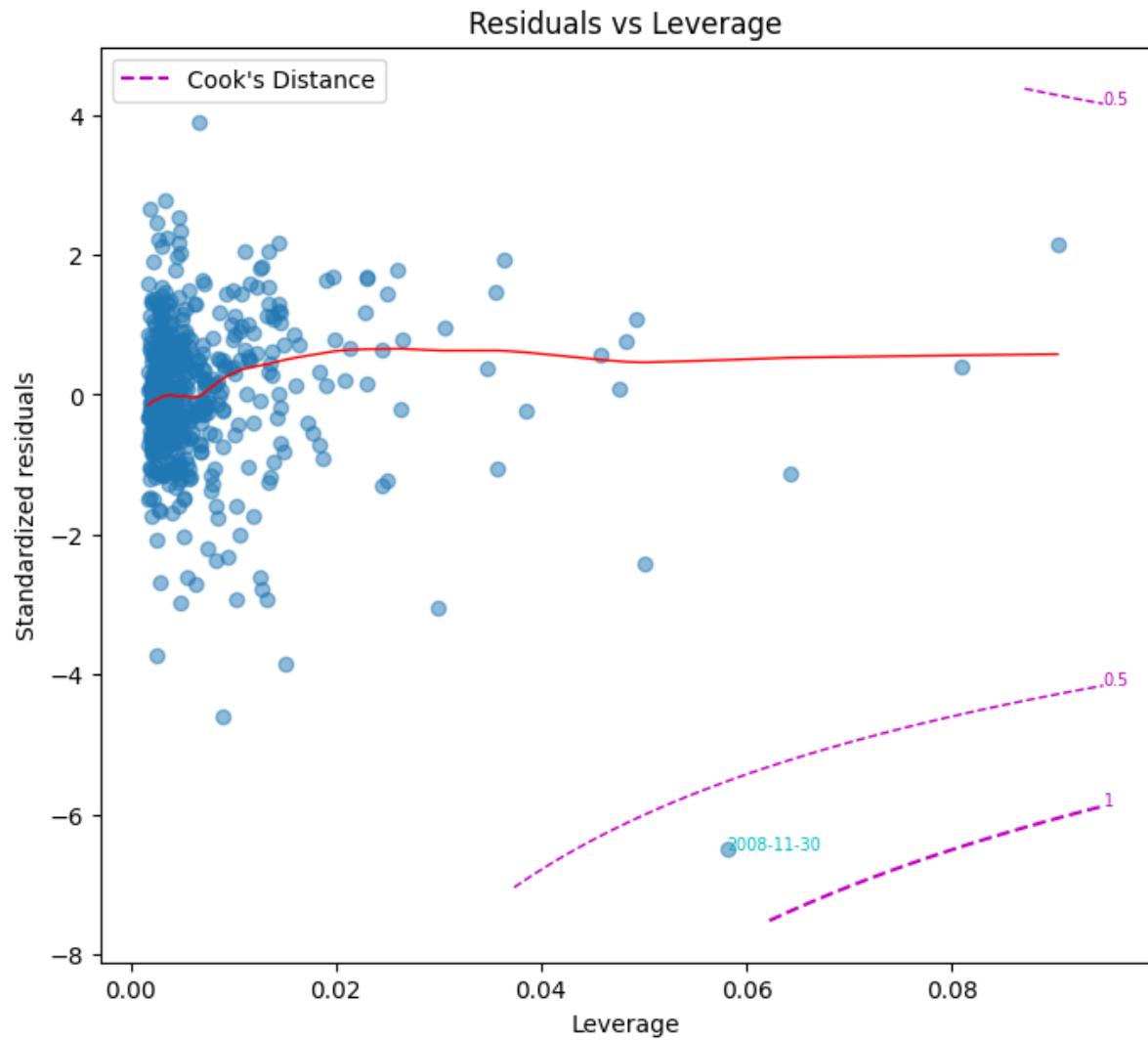
where

- p is the number of regression model parameters,
- $t_i = \frac{\hat{\epsilon}_i}{\hat{\sigma}(1-h_{ii})}$, also known as the **studentized residual**, which accounts for the estimated residuals having unequal variances (even though the true errors have equal variance).
- and $\hat{\sigma} = \sqrt{\sum_{j=1}^n \hat{\epsilon}_j^2/n}$.

A threshold of $D_i > 1$ is often suggested to identify influential observations. The Residuals vs Leverage plot helps us to find influential outliers.

```
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: []
```



ECONOMETRIC FORECASTS

In economics, the majority is always wrong - John Kenneth Galbraith

- Trends: seasonality, random walk, unit root
- Autocovariance: stationarity, AR, MA, SARIMAX
- Forecasting: single-step, multi-step
- Granger causality, impulse response function, Vector Autoregression
- Industrial Production

```
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
```

```
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()
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)
```

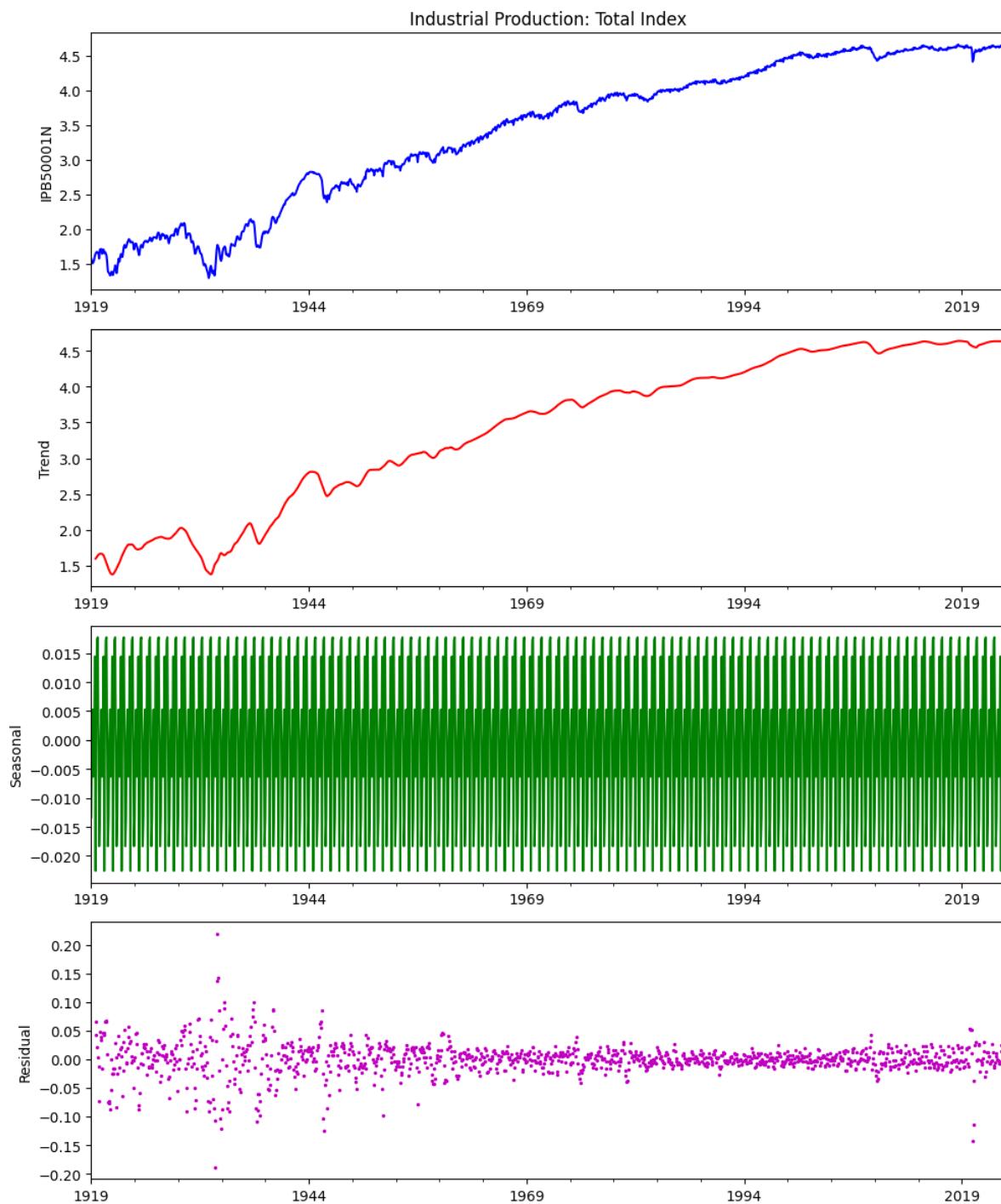
'Industrial Production: Total Index'

10.1 Seasonality

A time series can be decomposed into three distinct components: the trend, which captures the changes in the level of the time series over time; the seasonal component, which captures predictable changes in the time series according to the time of year; and the cyclical component, which captures the cycles in the data.

Suppose that Y_t is a seasonal time series that has a different mean in each period. The seasonality repeats each s periods (e.g. every 3 in a quarterly series or 12 in a monthly series). Such deterministic seasonalities produce differences in the mean of a time series that are simple to model 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),
                     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

Covariance stationarity depends on the first two moments of a time series: the mean and the autocovariances.

Autocovariance is defined as the covariance between a time series observations at different points in time. Its definition is the time-series analog of the covariance between two random variables. The h th autocovariance is defined as: $\gamma_{t,h} = E[(Y_t - E[Y_t])(Y_{t-h} - E[Y_{t-h}])]$

A time series is covariance-stationary if its first two moments satisfy three key properties.

1. The mean is constant and does not change over time (i.e., $E[Y_t] = m$ for all t).
2. The variance is finite and does not change over time (i.e., $V[Y_t] = \gamma_0 < \infty$ for all t).
3. The autocovariance is finite, does not change over time, and only depends on the distance between observation h (i.e., $Cov[Y_t, Y_{t-h}] = \gamma_h$ for all t).

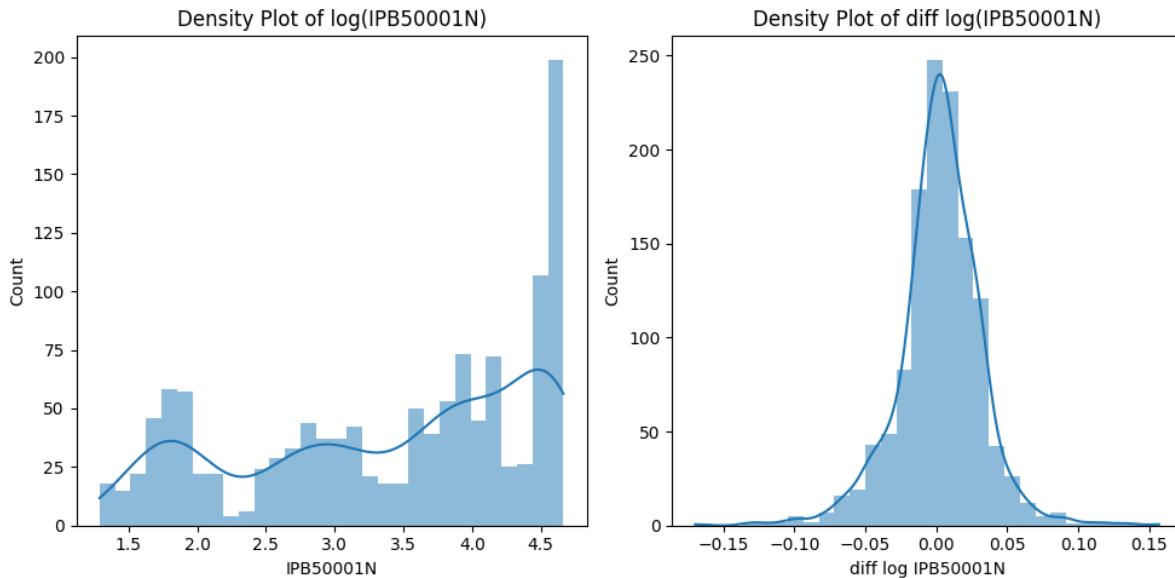
Covariance stationarity is important when modeling and forecasting time series. A covariance-stationary time series has constant relationships over time.

The null hypothesis of the Augmented Dickey-Fuller is that there is a unit root, with the alternative that there is no unit root. If the pvalue is less than a critical size, then we can reject that there is a unit root.

```
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) | -7.235 | 0.0 | 23 | 1237 | -3.436 | |
| | critical(5%) | critical(10%) | | | | |
| I(1) | -2.864 | | -2.568 | | | |

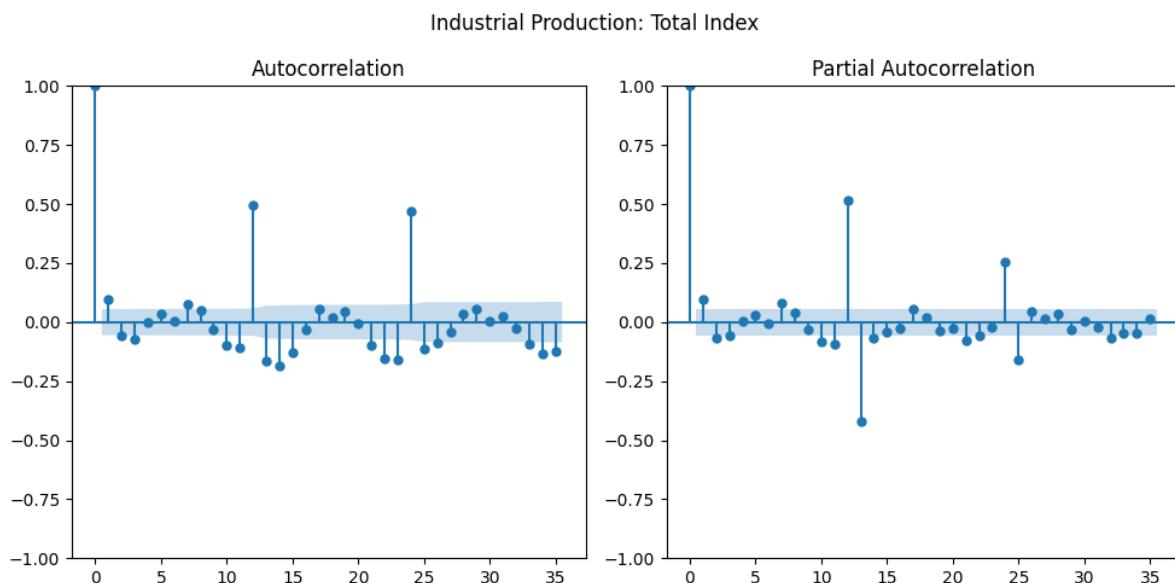
```
# Histogram Plot and Kernel Density Estimate
fig, axes = plt.subplots(1, 2, clear=True, figsize=(10,5))
sns.histplot(df.dropna(), bins=30, lw=0, kde=True, ax=axes[0])
axes[0].set_title(f"Density Plot of log({series_id})")
sns.histplot(df.diff().dropna().rename(f"diff log {series_id}"),
             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

The autocorrelation at lag h is defined as the ratio: $\rho_h = \frac{\gamma_h}{\gamma_0}$. The partial autocorrelation measures the strength of the correlation between these two time-series observations after controlling for the observations between them (i.e., the intermediate lags $Y_{t-1}, Y_{t-2}, \dots, Y_{t-h+1}$).

```
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))
plt.tight_layout()
```



10.4 Time Series

10.4.1 AR models

Autoregressive models relate the current value of a time series Y_t to its previous values. A first order AR process, which can be denoted by AR(1), evolves according to: $Y_t = \delta + \phi Y_{t-1} + \epsilon_t$, where δ is called the intercept, ϕ is the AR parameter, and the shock ϵ is white noise with constant and finite variance σ^2 . White noise processes have zero autocorrelation and autocovariance

Its partial autocorrelation function (**PACF**) is non-zero only for the first lag, while its autocorrelation function (**ACF**) exhibits a slow decay $\rho(h) = \phi^{|h|}$

The p th order AR process generalizes the first-order process to include p lags of Y in the model. The AR(p) model is: $Y_t = \mu + \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \dots + \phi_p Y_{t-p} + \epsilon_t$

10.4.2 MA models

A first-order **moving average** model, denoted MA(1), is defined as: $Y_t = \mu + \theta \epsilon_{t-1} + \epsilon_t$

The observed value of Y_t depends on both the contemporaneous and previous shock. Moving averages are always covariance-stationary. An MA(1) has a limited memory, because only the shock in the previous period impacts the current value.

The MA(1) generalizes to an MA(q), which includes q lags of the shock: $Y_t = \mu + \epsilon_t + \phi_1 \epsilon_{t-1} + \dots + \theta_q \epsilon_{t-q}$

Its ACF is always zero for lags larger than q , but the PACF is more complex.

10.4.3 ARMA models

Autoregressive Moving Average (ARMA) processes combine AR and MA processes. 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 $|\phi_1| < 1$. The MA coefficient plays no role in determining whether the process is covariance-stationary, because any MA is covariance-stationary and 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 are the orders of the seasonal lag polynomials, and f is 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.

A pair of time series have a spurious relationship when there are no fundamental links between them, but a regression of one on the other produces a coefficient estimate that is large and seemingly statistically different from zero when using conventional statistical distributions to obtain the critical values. When a time series contains a unit root, it is common to find spurious relationships with other time series that have a time trend or a unit root.

The solution is to difference a time series that contains a unit root. If Y_t has a unit root (i.e. has integration order 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 specified $(p, d, q) \times (P, D, Q)_s$, with

- the (p,d,q) order of the model for the number of AR parameters, differences (indicating the integration order of the process), and MA parameters,
- the (P,D,Q,s) order of the seasonal component for the AR parameters, differences, MA parameters, 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
```

(continues on next page)

(continued from previous page)

```

At iterate    0      f= -2.09203D+00      |proj g|=  3.50751D-01
At iterate    5      f= -2.09212D+00      |proj g|=  2.65613D-02
At iterate   10      f= -2.09214D+00      |proj g|=  1.27872D-01
* * *

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  8.002D-04  -2.092D+00
F = -2.0921435906737225

CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH

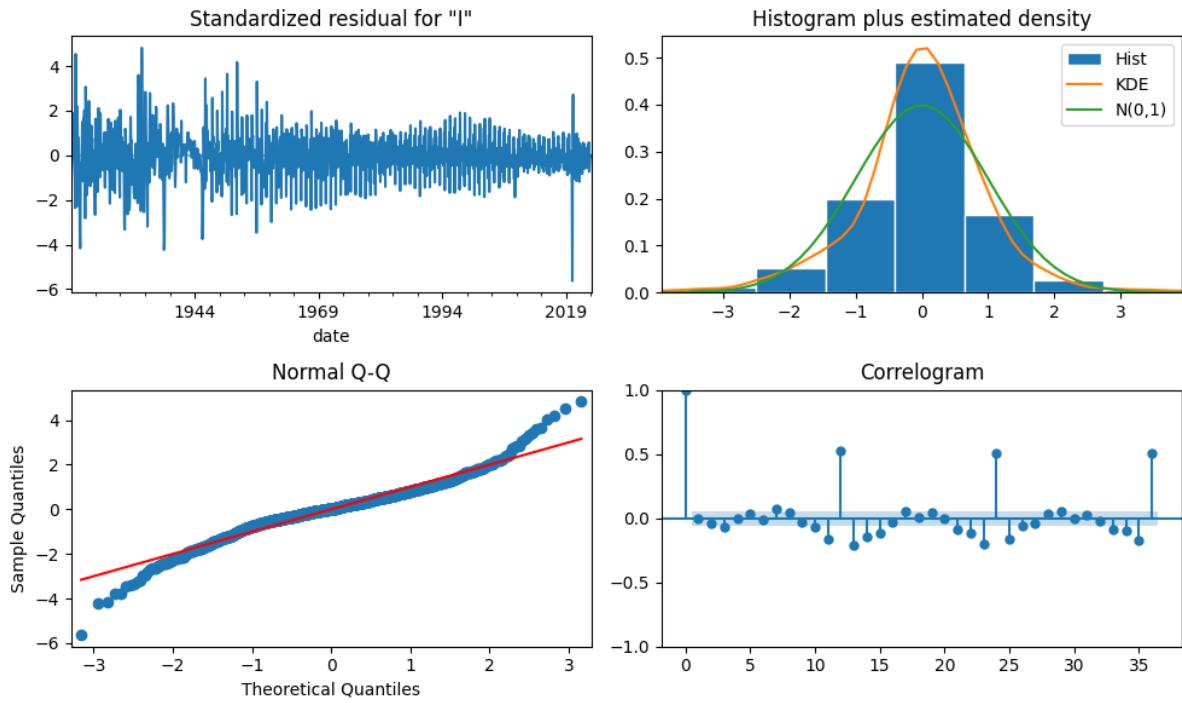
```

This problem is unconstrained.

| Dep. Variable: | IPB50001N | No. Observations: | 1262 | | | |
|--------------------------------|----------------------------|--------------------------|--------------------------|--------|--------|--------|
| Model: | SARIMAX(1, 1, 1) | Log Likelihood | 2640.285 | | | |
| Date: | Tue, 09 Apr 2024 | AIC | -5272.570 | | | |
| Time: | 11:16:42 | BIC | -5252.012 | | | |
| Sample: | 01-31-1919 - 02-29-2024 | HQIC | -5264.845 | | | |
| Covariance Type: opg | | | | | | |
| | coef | std err | z | P> z | [0.025 | 0.975] |
| intercept | 0.0027 | 0.001 | 2.393 | 0.017 | 0.000 | 0.005 |
| ar.L1 | -0.1199 | 0.222 | -0.541 | 0.588 | -0.554 | 0.314 |
| ma.L1 | 0.2237 | 0.221 | 1.011 | 0.312 | -0.210 | 0.657 |
| sigma2 | 0.0009 | 2.24e-05 | 39.741 | 0.000 | 0.001 | 0.001 |
| Ljung-Box (L1) (Q): | | 0.02 | Jarque-Bera (JB): | 554.43 | | |
| Prob(Q): | | 0.88 | Prob(JB): | 0.00 | | |
| Heteroskedasticity (H): | | 0.32 | Skew: | -0.20 | | |
| Prob(H) (two-sided): | | 0.00 | Kurtosis: | 6.22 | | |

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

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.

- AIC is defined as: $AIC = T \ln \hat{\sigma}^2 + 2k$, where T is the sample size and k is the number of parameters.
- BIC alters the penalty and is defined as: $BIC = T \ln \hat{\sigma}^2 + k \ln T$

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:             1250
Model:                 AutoReg(1)    Log Likelihood:          3331.030
Method:                Conditional MLE  S.D. of innovations:   0.017
Date:                  Tue, 09 Apr 2024  AIC:                  -6656.061
Time:                  11:16:45      BIC:                  -6640.670
Sample:                03-31-1919  HQIC:                  -6650.275
                       - 03-31-2023
=====
            coef    std err      z   P>|z|    [ 0.025   0.975]
-----
const      0.0013      0.000    2.697    0.007      0.000    0.002
INDPRO.L1  0.4854      0.025   19.685    0.000      0.437    0.534
            Roots
=====
          Real      Imaginary      Modulus      Frequency
-----
AR.1      2.0601      +0.0000j      2.0601      0.0000
```

10.5.2 One-step forecast

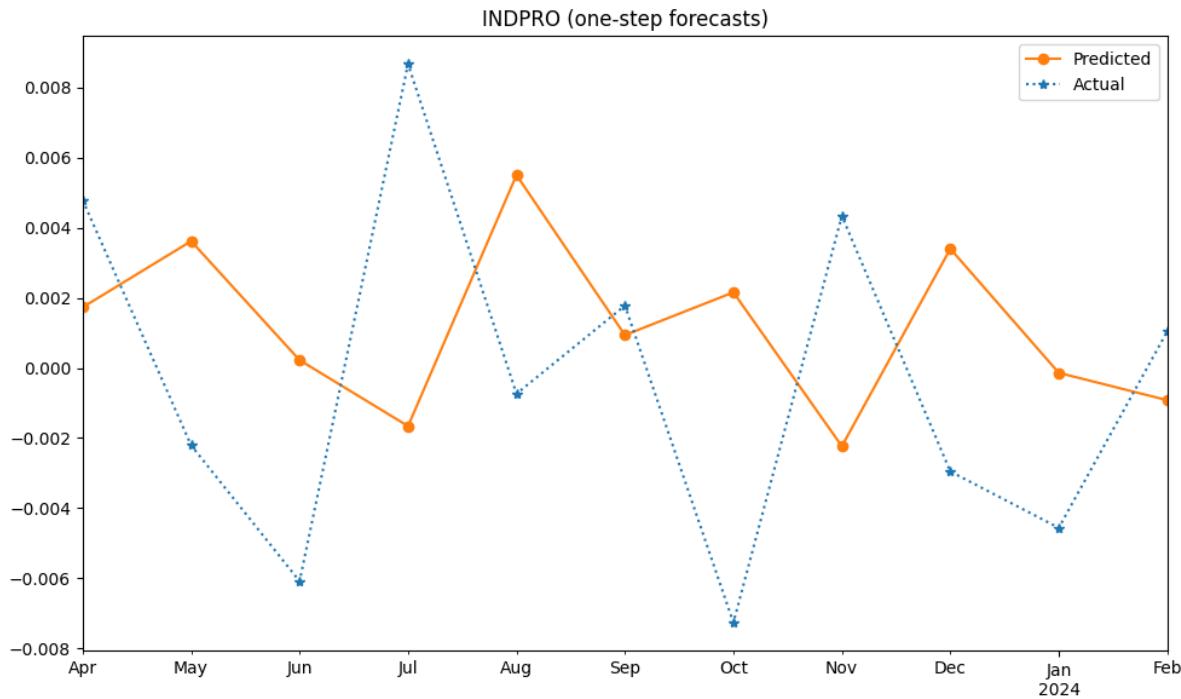
The one-step forecast is the expectation of Y_{T+1} conditional the time T information set, which contains all values that are known at time T, including the entire prior history of Y (Y_T, Y_{T-1}, \dots), as well as all values of any other variable that occurred at time T or earlier.

```
# 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=0.006210397949646384
```

```
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()
```

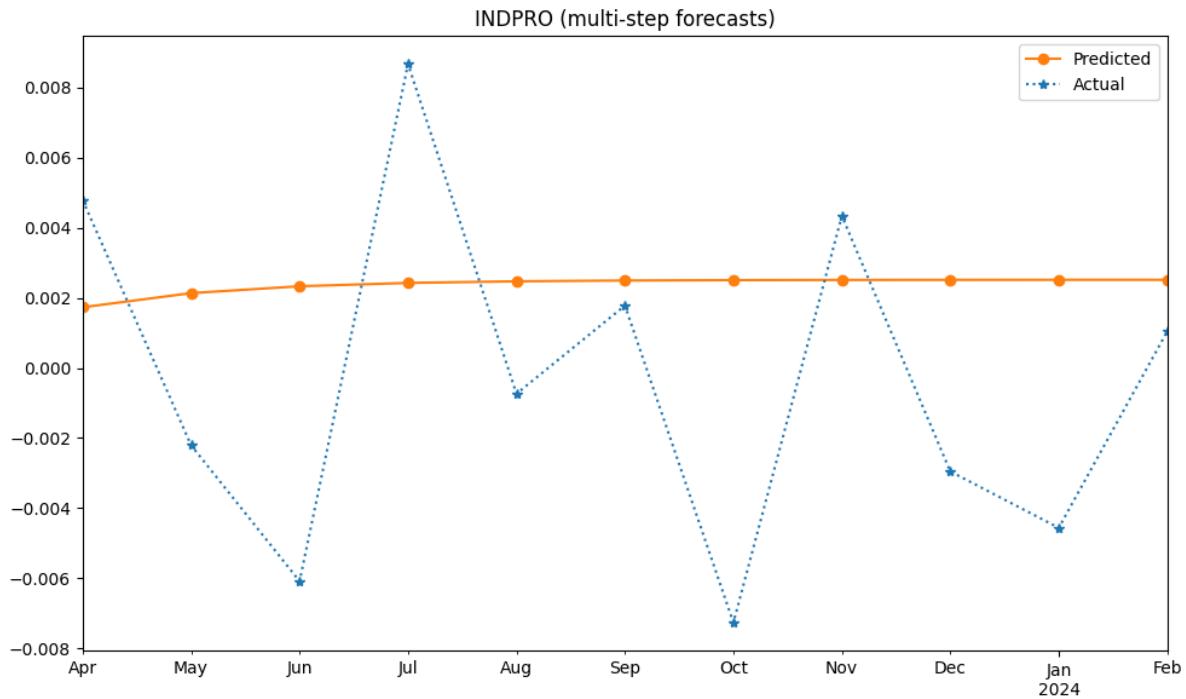


10.5.3 Multi-step forecast

Forecasts are generated recursively 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=0.005469



10.5.4 Granger Causality

```
# 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
 ssr based F test: F=0.3694 , p=0.5435 , df_denom=741, df_num=1
 ssr based chi2 test: chi2=0.3709 , p=0.5425 , df=1

(continues on next page)

(continued from previous page)

```
likelihood ratio test: chi2=0.3708 , p=0.5426 , df=1
parameter F test: F=0.3694 , p=0.5435 , df_denom=741, df_num=1
```

Granger Causality

number of lags (no zero) 2

```
ssr based F test: F=7.5604 , p=0.0006 , df_denom=738, df_num=2
```

```
ssr based chi2 test: chi2=15.2233 , p=0.0005 , df=2
```

```
likelihood ratio test: chi2=15.0694 , p=0.0005 , df=2
```

```
parameter F test: F=7.5604 , p=0.0006 , df_denom=738, df_num=2
```

Granger Causality

number of lags (no zero) 3

```
ssr based F test: F=5.3906 , p=0.0011 , df_denom=735, df_num=3
```

```
ssr based chi2 test: chi2=16.3259 , p=0.0010 , df=3
```

```
likelihood ratio test: chi2=16.1489 , p=0.0011 , df=3
```

```
parameter F test: F=5.3906 , p=0.0011 , df_denom=735, df_num=3
```

OLS Regression Results

```
=====
Dep. Variable: INDPRO      R-squared: 0.089
Model: OLS                  Adj. R-squared: 0.084
Method: Least Squares      F-statistic: 2.668
Date: Tue, 09 Apr 2024     Prob (F-statistic): 0.0313
Time: 11:16:47              Log-Likelihood: 2430.5
No. Observations: 742       AIC: -4851.
Df Residuals: 737          BIC: -4828.
Df Model: 4
Covariance Type: HAC
=====
```

| | coef | std err | t | P> t | [0.025 | 0. |
|-------------------|---------|-------------------|--------|-------|------------|-----|
| 975] | | | | | | |
| Intercept | 0.0019 | 0.001 | 1.954 | 0.051 | -8.95e-06 | 0. |
| INDPRO.shift(1) | 0.2548 | 0.143 | 1.778 | 0.076 | -0.027 | 0. |
| CPIAUCSL.shift(1) | 0.4099 | 0.213 | 1.921 | 0.055 | -0.009 | 0. |
| CPIAUCSL.shift(2) | -0.4703 | 0.214 | -2.202 | 0.028 | -0.890 | -0. |
| CPIAUCSL.shift(3) | -0.1124 | 0.144 | -0.779 | 0.436 | -0.395 | 0. |
| Omnibus: | 727.398 | Durbin-Watson: | | | 1.983 | |
| Prob(Omnibus): | 0.000 | Jarque-Bera (JB): | | | 126368.644 | |
| Skew: | -3.898 | Prob(JB): | | | 0.00 | |
| Kurtosis: | 66.456 | Cond. No. | | | 563. | |

Notes:

[1] Standard Errors are heteroscedasticity and autocorrelation robust (HAC) using 0 lags and without small sample correction

Null Hypothesis: INDPRO granger-causes CPIAUCSL

Granger Causality

(continues on next page)

(continued from previous page)

```
number of lags (no zero) 1
ssr based F test:      F=0.0173 , p=0.8955 , df_denom=741, df_num=1
ssr based chi2 test:  chi2=0.0173 , p=0.8952 , df=1
likelihood ratio test: chi2=0.0173 , p=0.8952 , df=1
parameter F test:      F=0.0173 , p=0.8955 , df_denom=741, df_num=1
```

Granger Causality

```
number of lags (no zero) 2
ssr based F test:      F=0.0053 , p=0.9947 , df_denom=738, df_num=2
ssr based chi2 test:  chi2=0.0107 , p=0.9947 , df=2
likelihood ratio test: chi2=0.0107 , p=0.9947 , df=2
parameter F test:      F=0.0053 , p=0.9947 , df_denom=738, df_num=2
```

Granger Causality

```
number of lags (no zero) 3
ssr based F test:      F=0.1255 , p=0.9450 , df_denom=735, df_num=3
ssr based chi2 test:  chi2=0.3802 , p=0.9443 , df=3
likelihood ratio test: chi2=0.3801 , p=0.9443 , df=3
parameter F test:      F=0.1255 , p=0.9450 , df_denom=735, df_num=3
```

OLS Regression Results

```
=====
Dep. Variable:          CPIAUCSL      R-squared:                   0.387
Model:                 OLS            Adj. R-squared:             0.384
Method:                Least Squares  F-statistic:                  52.69
Date:                  Tue, 09 Apr 2024  Prob (F-statistic):        4.63e-39
Time:                  11:16:47        Log-Likelihood:             3399.4
No. Observations:      742            AIC:                      -6789.
Df Residuals:          737            BIC:                      -6766.
Df Model:                  4
Covariance Type:        HAC
=====
            coef    std err          t      P>|t|      [0.025      0.
    975]
    --
Intercept      0.0012      0.000      6.816      0.000      0.001      0.
    002
CPIAUCSL.shift(1)  0.6216      0.045     13.719      0.000      0.533      0.
    711
INDPRO.shift(1)   -0.0012      0.014     -0.086      0.931     -0.028      0.
    026
INDPRO.shift(2)   7.455e-05     0.016      0.005      0.996     -0.032      0.
    032
INDPRO.shift(3)    0.0058      0.010      0.608      0.543     -0.013      0.
    025
=====
Omnibus:            86.562      Durbin-Watson:            2.123
Prob(Omnibus):      0.000      Jarque-Bera (JB):        713.052
Skew:                  0.030      Prob(JB):                  1.45e-155
Kurtosis:                 7.802      Cond. No.                   317.
=====
```

Notes:

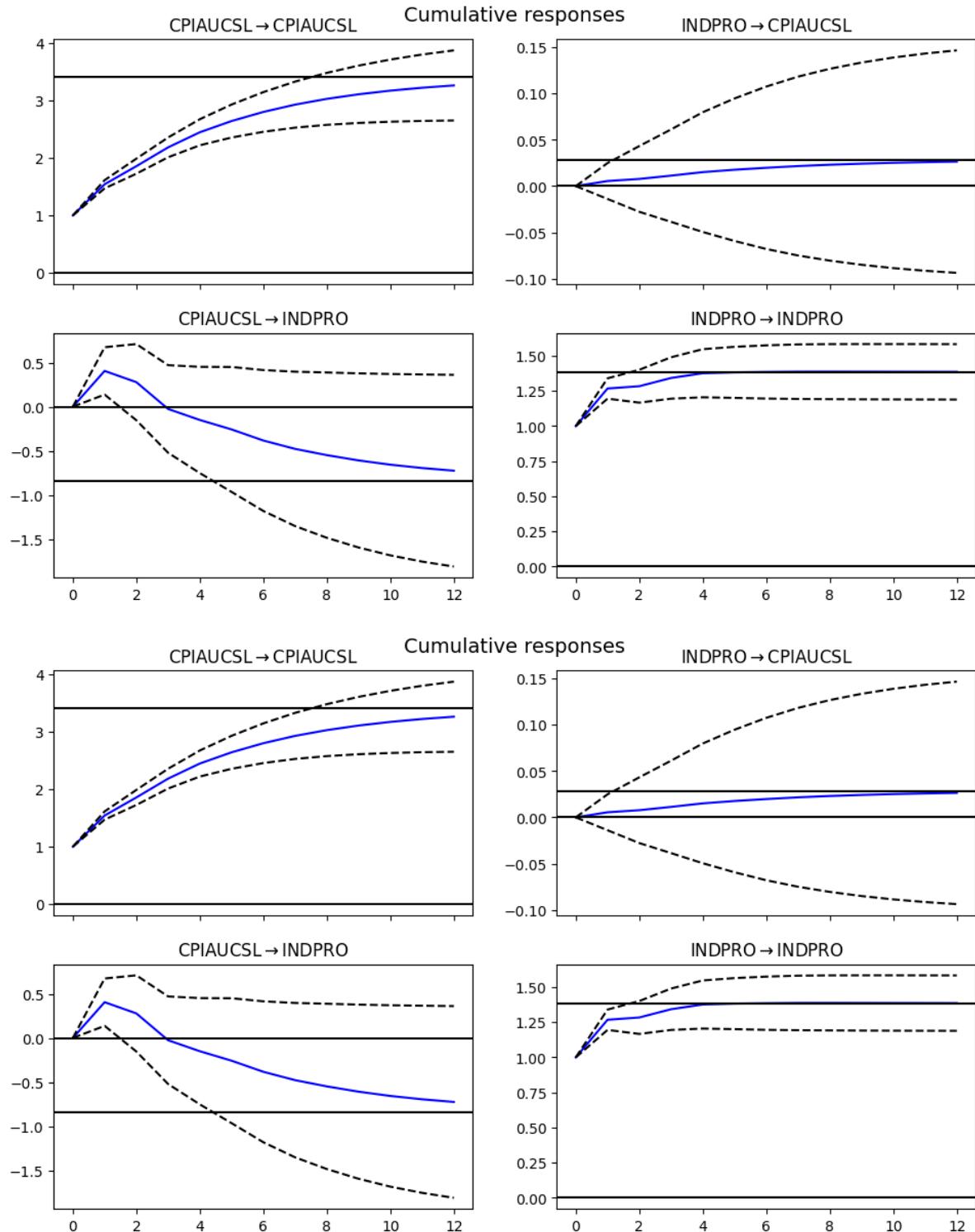
[1] Standard Errors are heteroscedasticity and autocorrelation robust (HAC) using 0 lags and without small sample correction

10.5.5 Impulse Response Function

- Vector Autoregression (VAR)

```
# 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: Tue, 09, Apr, 2024
Time: 11:16:47
-----
No. of Equations: 2.00000 BIC: -21.3130
Nobs: 742.000 HQIC: -21.3664
Log likelihood: 5847.68 FPE: 5.08298e-10
AIC: -21.4000 Det(Omega_mle): 4.98842e-10
-----
Results for equation CPIAUCSL
=====
      coefficient      std. error      t-stat      prob
-----
const      0.000906      0.000141      6.439      0.000
L1.CPIAUCSL  0.542635      0.036649     14.806      0.000
L1.INDPRO   0.005496      0.009885      0.556      0.578
L2.CPIAUCSL  0.018542      0.041788      0.444      0.657
L2.INDPRO   -0.002211      0.010121     -0.218      0.827
L3.CPIAUCSL  0.147812      0.037002      3.995      0.000
L3.INDPRO   0.002745      0.009758      0.281      0.778
-----
Results for equation INDPRO
=====
      coefficient      std. error      t-stat      prob
-----
const      0.001908      0.000526      3.628      0.000
L1.CPIAUCSL  0.405543      0.136956      2.961      0.003
L1.INDPRO   0.267880      0.036938      7.252      0.000
L2.CPIAUCSL  -0.456232      0.156160     -2.922      0.003
L2.INDPRO   -0.057815      0.037823     -1.529      0.126
L3.CPIAUCSL  -0.126543      0.138275     -0.915      0.360
L3.INDPRO   0.071427      0.036463      1.959      0.050
-----
Correlation matrix of residuals
      CPIAUCSL      INDPRO
CPIAUCSL  1.000000  0.098825
INDPRO    0.098825  1.000000
```



APPROXIMATE FACTOR MODELS

It is better to be vaguely right than precisely wrong - Carveth Read

- Integration Order
- Approximate factor model

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.

```
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 |

(continues on next page)

(continued from previous page)

```

7 1887-03-31 1888-04-30
8 1890-07-31 1891-05-31
9 1893-01-31 1894-06-30
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) specification is the most widely used unit root test. An ADF test is implemented using an OLS regression where the difference of a series is regressed on its lagged level, relevant deterministic terms, and lagged differences. The general form of an ADF regression is

$$\Delta Y_t = \gamma Y_{t-1} + (\delta_o + \delta_1 t) + \lambda_1 \Delta Y_{t-1} + \dots + \lambda_p \Delta Y_{t-p}$$

The value of γ is 0 when Y_t is a random walk, and so is non-stationary. The alternative is $\gamma < 0$ which corresponds to the case that Y_t is covariance-stationary. Note that the alternative is one-sided, since positive values of γ correspond to an AR coefficient that is larger than 1.

When the null of a unit root cannot be rejected, the series should be differenced. The best practice is to repeat the ADF test on the iteratively-differenced data to ensure that it is stationary.

```

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

FRED-MD and FRED-QD series are tagged by one of seven transformation codes, which suggest transformations, such as taking logs or differences, to apply to the underlying series. The number of times differences are taken of a series should remove unit roots and transform the series to be covariance stationary.

```
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 (20240331):

| 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 | 50 | 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 | 18 | 4 | 0 | 11 | 11 | 0 | |
| 2 | 1 | 0 | 0 | 32 | 11 | 19 | 2 | 19 | 4 | 15 | |
| 3 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 4 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 10 | 6 | 4 | |
| 5 | 1 | 1 | 0 | 140 | 30 | 106 | 4 | 52 | 14 | 38 | |
| 6 | 2 | 1 | 0 | 50 | 0 | 30 | 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 |
| PERMITNE | 4 | 1 | 1 |
| HOUSTMW | 4 | 1 | 1 |
| HOUSTNE | 4 | 1 | 1 |

| | title |
|----------|---|
| PERMITMW | New Privately-Owned Housing Units Authorized i... |
| PERMITNE | New Privately-Owned Housing Units Authorized i... |

(continues on next page)

(continued from previous page)

```
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 | \ | | title |
|---------------|-------|------|-----------|---|--|---|
| NWPI | 1 | 1 | 1 | | | *** NWPI *** |
| TLBSNNBBDI | 1 | 1 | 1 | | | *** TLBSNNBBDI *** |
| TLBSNNCBBDI | 1 | 1 | 1 | | | *** TLBSNNCBBDI *** |
| HWI | 1 | 1 | 1 | | | Help Wanted Index for United States |
| GFDEBTN | 2 | 2 | 1 | | | Federal Debt: Total Public Debt |
| S&P div yield | 2 | 2 | 1 | | | S&P's Composite Common Stock: Dividend Yield |
| CES2000000008 | 5 | 2 | 1 | | | Average Hourly Earnings of Production and Nons... |
| TLBSHNO | 5 | 2 | 1 | | | Households and Nonprofit Organizations; Total ... |
| OPHMFG | 5 | 2 | 1 | | | Manufacturing Sector: Labor Productivity (Outp... |
| SPCS20RSA | 5 | 2 | 1 | | | S&P CoreLogic Case-Shiller 20-City Composite H... |

11.2 Approximate factor model

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 of the dimensions found by PCA is a linear combination of the features. The first principal component of a set of features X_1, X_2, \dots, X_p is the normalized linear combination of the features $Z_1 = \phi_{11}X_1 + \phi_{21}X_2 + \dots + \phi_{p1}X_p$ that has the largest variance. We can then iteratively find the second principal component Z_2 (the linear combination of features that has maximal variance out of all linear combinations that are uncorrelated with previously found components), third component Z_3 , and so on.

11.2.1 BIC criterion

Select number of components

11.2.2 Data imputation

EM algorithm

```
# Verify BaiNg implementation on original FRED-MD and FRED-QD
qd_df, qd_codes = fred_qd(202004)
md_df, md_codes = fred_md(201505)
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: quarterly/2020-04.csv
 FRED-MD vintage: monthly/2015-05.csv

(continues on next page)

(continued from previous page)

FRED-MD monthly series:

| | selected | variance explained | start | end | obs | series |
|----------|---------------------------|---|--------------------------------|----------|-----|-----------------|
| factors | 8 | 0.484228 | 19600331 | 20141231 | 658 | 133 |
| Factor:1 | Variance Explained=0.1625 | | | | | |
| series | | | | | | description |
| 0.7426 | USGOOD | | All Employees, Goods-Producing | | | |
| 0.7236 | PAYEMS | | All Employees, Total Nonfarm | | | |
| 0.6999 | MANEMP | | All Employees, Manufacturing | | | |
| 0.6563 | NAPM | | | | | *** NAPM *** |
| 0.6538 | IPMANSICS | Industrial Production: Manufacturing (SIC) | | | | |
| 0.6510 | DMANEMP | All Employees, Durable Goods | | | | |
| 0.6312 | INDPRO | Industrial Production: Total Index | | | | |
| 0.6036 | NAPMNOI | | | | | *** NAPMNOI *** |
| 0.6024 | NAPMPI | | | | | *** NAPMPI *** |
| 0.5601 | CUMFNS | Capacity Utilization: Manufacturing (SIC) | | | | |
| Factor:2 | Variance Explained=0.0705 | | | | | |
| series | | | | | | description |
| 0.5960 | T10YFFM | 10-Year Treasury Constant Maturity Minus Feder... | | | | |
| 0.5881 | AAAFFM | Moody's Seasoned Aaa Corporate Bond Minus Fede... | | | | |
| 0.5576 | BAAFFM | Moody's Seasoned Baa Corporate Bond Minus Fede... | | | | |
| 0.5543 | T5YFFM | 5-Year Treasury Constant Maturity Minus Feder... | | | | |
| 0.4591 | TB6SMFFM | 6-Month Treasury Bill Minus Federal Funds Rate | | | | |
| 0.4569 | TB3SMFFM | 3-Month Treasury Bill Minus Federal Funds Rate | | | | |
| 0.4137 | T1YFFM | 1-Year Treasury Constant Maturity Minus Feder... | | | | |
| 0.2385 | COMPAPFF | 3-Month Commercial Paper Minus FEDFUNDS | | | | |
| 0.2098 | BUSINV | Total Business Inventories | | | | |
| 0.1803 | M2REAL | Real M2 Money Stock | | | | |
| Factor:3 | Variance Explained=0.0652 | | | | | |
| series | | | | | | description |
| 0.6899 | CUSR0000SAC | Consumer Price Index for All Urban Consumers: ... | | | | |
| 0.6791 | DNDGRG3M086SBEA | Personal consumption expenditures: Nondurable ... | | | | |
| 0.6448 | CPIAUCSL | Consumer Price Index for All Urban Consumers: ... | | | | |
| 0.6157 | CUSR0000SA0L5 | Consumer Price Index for All Urban Consumers: ... | | | | |
| 0.5871 | CUUR0000SA0L2 | Consumer Price Index for All Urban Consumers: ... | | | | |
| 0.5680 | PCEPI | Personal Consumption Expenditures: Chain-type ... | | | | |
| 0.5635 | CPITRNSL | Consumer Price Index for All Urban Consumers: ... | | | | |
| 0.5241 | CPIULFSL | Consumer Price Index for All Urban Consumers: ... | | | | |
| 0.4626 | PPIFCG | Producer Price Index by Commodity for Finished... | | | | |
| 0.4539 | PPIITM | Producer Price Index by Commodity Intermediate... | | | | |
| Factor:4 | Variance Explained=0.0543 | | | | | |
| series | | | | | | description |
| 0.4458 | GS1 | Market Yield on U.S. Treasury Securities at 1-... | | | | |
| 0.4383 | GS5 | Market Yield on U.S. Treasury Securities at 5-... | | | | |
| 0.4236 | AAA | Moody's Seasoned Aaa Corporate Bond Yield | | | | |
| 0.4163 | TB6MS | 6-Month Treasury Bill Secondary Market Rate, D... | | | | |
| 0.4028 | GS10 | Market Yield on U.S. Treasury Securities at 10... | | | | |
| 0.3767 | BAA | Moody's Seasoned Baa Corporate Bond Yield | | | | |
| 0.3340 | TB3MS | 3-Month Treasury Bill Secondary Market Rate, D... | | | | |
| 0.3257 | CP3M | 3-Month AA Financial Commercial Paper Rates | | | | |
| 0.2521 | HOUST | New Privately-Owned Housing Units Started: Tot... | | | | |
| 0.2448 | HOUSTW | New Privately-Owned Housing Units Started: Tot... | | | | |
| Factor:5 | Variance Explained=0.0428 | | | | | |
| series | | | | | | description |
| 0.2489 | PERMITW | New Privately-Owned Housing Units Authorized i... | | | | |
| 0.2380 | PERMIT | New Privately-Owned Housing Units Authorized i... | | | | |
| 0.2325 | HOUSTW | New Privately-Owned Housing Units Started: Tot... | | | | |

(continues on next page)

(continued from previous page)

```

0.2199      GS5  Market Yield on U.S. Treasury Securities at 5-...
0.2193      GS1  Market Yield on U.S. Treasury Securities at 1-...
0.2155      HOUST New Privately-Owned Housing Units Started: Tot...
0.2005      GS10 Market Yield on U.S. Treasury Securities at 10...
0.1971      T1YFFM 1-Year Treasury Constant Maturity Minus Federa...
0.1963      PERMITMW New Privately-Owned Housing Units Authorized i...
0.1904      TB6MS 6-Month Treasury Bill Secondary Market Rate, D...
Factor:6 Variance Explained=0.0364
    series                               description
0.2481      IPCONGD      Industrial Production: Consumer Goods
0.1904      NAPMEI       *** NAPMEI ***
0.1831      IPDCONGD     Industrial Production: Durable Consumer Goods
0.1820      ISRATIO      Total Business: Inventories to Sales Ratio
0.1784      IPFINAL      Industrial Production: Final Products
0.1547      IPFPNSS      Industrial Production: Final Products and Noni...
0.1538      NAPMII       *** NAPMII ***
0.1526      NAPM         *** NAPM ***
0.1436      TB6SMFFM     6-Month Treasury Bill Minus Federal Funds Rate
0.1419      AWHMAN      Average Weekly Hours of Production and Nonsupe...
Factor:7 Variance Explained=0.0266
    series                               description
0.4509      TWEXMMTH    Nominal Major Currencies U.S. Dollar Index (Go...
0.2293      EXSZUS      Swiss Francs to U.S. Dollar Spot Exchange Rate
0.1660      EXUSUK      U.S. Dollars to U.K. Pound Sterling Spot Excha...
0.1656      EXJPUS      Japanese Yen to U.S. Dollar Spot Exchange Rate
0.1482      SRVPRD      All Employees, Service-Providing
0.1286      CES3000000008 Average Hourly Earnings of Production and Nons...
0.1228      CES0600000008 Average Hourly Earnings of Production and Nons...
0.1199      USTRADE      All Employees, Retail Trade
0.1042      DPCERA3M086SBEA Real personal consumption expenditures (chain-...
0.1026      UMCSENT     University of Michigan: Consumer Sentiment
Factor:8 Variance Explained=0.0259
    series                               description
0.3414      S&P 500      S&P's Common Stock Price Index: Composite
0.2753      S&P div yield S&P's Composite Common Stock: Dividend Yield
0.2028      S&P PE ratio  S&P's Composite Common Stock: Price-Earnings R...
0.1817      EXCAUS      Canadian Dollars to U.S. Dollar Spot Exchange ...
0.1222      UMCSENT     University of Michigan: Consumer Sentiment
0.1175      TWEXMMTH    Nominal Major Currencies U.S. Dollar Index (Go...
0.1042      NONBORRES   Reserves of Depository Institutions, Nonborrowed
0.0986      IPNCONGD    Industrial Production: Non-Durable Consumer Goods
0.0950      IPCONGD     Industrial Production: Consumer Goods
0.0828      AMBSL       St. Louis Adjusted Monetary Base (DISCONTINUED)
FRED-QD quarterly series:
    selected  variance explained      start      end  obs  series
factors        7          0.497416  19600331  20191231  240    247
Factor:1 Variance Explained=0.2018
    series                               description
0.8403      USPRIV      All Employees, Total Private
0.8208      USGOOD      All Employees, Goods-Producing
0.8142      PAYEMS      All Employees, Total Nonfarm
0.8122      IPMANSICS   Industrial Production: Manufacturing (SIC)
0.8109      OUTMS      Manufacturing Sector: Real Sectoral Output for...
0.8058      INDPROM      Industrial Production: Total Index
0.7781      MANEMP      All Employees, Manufacturing
0.7722      HOANBS      Nonfarm Business Sector: Hours Worked for All ...

```

(continues on next page)

(continued from previous page)

| | | |
|------------------------------------|-----------------|---|
| 0.7693 | DMANEMP | All Employees, Durable Goods |
| 0.7661 | UNRATE | Unemployment Rate |
| Factor:2 Variance Explained=0.0818 | | |
| | series | description |
| 0.5122 | AAAFFM | Moody's Seasoned Aaa Corporate Bond Minus Fede... |
| 0.4907 | T5YFFM | 5-Year Treasury Constant Maturity Minus Federa... |
| 0.4545 | PERMIT | New Privately-Owned Housing Units Authorized i... |
| 0.4419 | BUSINV | Total Business Inventories |
| 0.4180 | HOUST | New Privately-Owned Housing Units Started: Tot... |
| 0.4001 | PERMITS | New Privately-Owned Housing Units Authorized i... |
| 0.3814 | TCU | Capacity Utilization: Total Index |
| 0.3755 | MZMREAL | Real MZM Money Stock (DISCONTINUED) |
| 0.3733 | TB3SMFFM | 3-Month Treasury Bill Minus Federal Funds Rate |
| 0.3719 | GS10TB3M | *** GS10TB3M *** |
| Factor:3 Variance Explained=0.0730 | | |
| | series | description |
| 0.7561 | CUSR0000SA0L2 | Consumer Price Index for All Urban Consumers: ... |
| 0.7420 | CUSR0000SAC | Consumer Price Index for All Urban Consumers: ... |
| 0.7370 | DGDSRG3Q086SBEA | Personal consumption expenditures: Goods (chai... |
| 0.7165 | PCECTPI | Personal Consumption Expenditures: Chain-type ... |
| 0.7092 | CPITRNSL | Consumer Price Index for All Urban Consumers: ... |
| 0.6989 | DNDGRG3Q086SBEA | Personal consumption expenditures: Nondurable ... |
| 0.6733 | CUSR0000SA0L5 | Consumer Price Index for All Urban Consumers: ... |
| 0.6636 | CPIAUCSL | Consumer Price Index for All Urban Consumers: ... |
| 0.6523 | WPSID61 | Producer Price Index by Commodity: Intermediat... |
| 0.6285 | PPIIDC | Producer Price Index by Commodity: Industrial ... |
| Factor:4 Variance Explained=0.0468 | | |
| | series | description |
| 0.3966 | IMFSL | Institutional Money Market Funds (DISCONTINUED) |
| 0.3490 | CES9093000001 | All Employees, Local Government |
| 0.3283 | CES9092000001 | All Employees, State Government |
| 0.2499 | USGOVT | All Employees, Government |
| 0.2313 | GFDEBTN | Federal Debt: Total Public Debt |
| 0.2276 | REVOLSL | Revolving Consumer Credit Owned and Securitized |
| 0.2274 | EXUSEU | U.S. Dollars to Euro Spot Exchange Rate |
| 0.2194 | COMPRMS | Manufacturing Sector: Real Hourly Compensation... |
| 0.2173 | USSERV | All Employees, Other Services |
| 0.2145 | NWPI | *** NWPI *** |
| Factor:5 Variance Explained=0.0376 | | |
| | series | description |
| 0.3689 | OPHMFG | Manufacturing Sector: Labor Productivity (Outp... |
| 0.3003 | HWI | Help Wanted Index for United States |
| 0.2973 | NWPI | *** NWPI *** |
| 0.2960 | AWHMAN | Average Weekly Hours of Production and Nonsupe... |
| 0.2725 | OPHPBS | Business Sector: Labor Productivity (Output pe... |
| 0.2355 | OPHNFB | Nonfarm Business Sector: Labor Productivity (O... |
| 0.2347 | UNRATELT | *** UNRATELT *** |
| 0.2281 | UNLPNBS | Nonfarm Business Sector: Unit Nonlabor Payment... |
| 0.2158 | ULCMFG | Manufacturing Sector: Unit Labor Costs for All... |
| 0.1965 | TLBSNNCBBDI | *** TLBSNNCBBDI *** |
| Factor:6 Variance Explained=0.0301 | | |
| | series | description |
| 0.2739 | ULCBS | Business Sector: Unit Labor Costs for All Workers |
| 0.2704 | ULCNFB | Nonfarm Business Sector: Unit Labor Costs for ... |
| 0.2515 | CONSPI | Nonrevolving consumer credit to Personal Income |
| 0.2465 | EXUSEU | U.S. Dollars to Euro Spot Exchange Rate |

(continues on next page)

(continued from previous page)

```

0.2169      AHETPI  Average Hourly Earnings of Production and Nons...
0.1946      CONSUMER  Consumer Loans, All Commercial Banks
0.1775      NONREVSL  Nonrevolving Consumer Credit Owned and Securit...
0.1480      TOTALSL   Total Consumer Credit Owned and Securitized
0.1417      UMCSENT   University of Michigan: Consumer Sentiment
0.1399  B021RE1Q156NBEA  Shares of gross domestic product: Imports of g...
Factor:7 Variance Explained=0.0264
                                series                                description
0.2817  USEPUINDEXM  Economic Policy Uncertainty Index for United S...
0.2304  TNWBSHNO   Households and Nonprofit Organizations; Net Wo...
0.2199  TABSHNO    Households and Nonprofit Organizations; Total ...
0.2087  TFAABSHNO  Households and Nonprofit Organizations; Total ...
0.2081  TARESA     *** TARESA ***
0.2028  S&P 500     S&P's Common Stock Price Index: Composite
0.1928  NASDAQCOM  NASDAQ Composite Index
0.1498  GS10TB3M   *** GS10TB3M ***
0.1492  TB6M3M    *** TB6M3M ***
0.1331  PCEPILFE   Personal Consumption Expenditures Excluding Fo...

```

```

# 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)

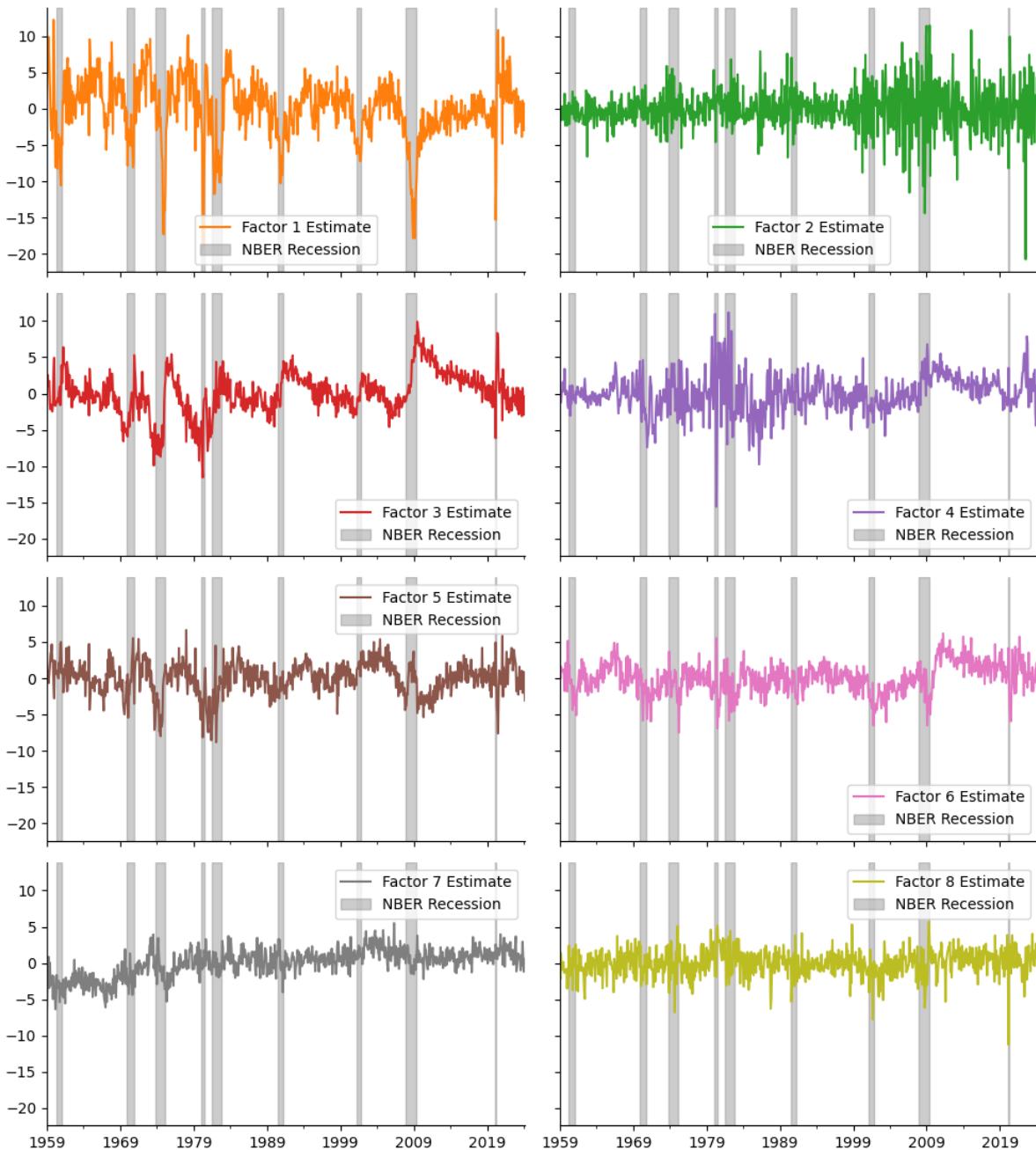
(continued from previous page)

```

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:Mar-2024



STATE SPACE MODELS

If you want to understand today, you have to search yesterday - Pearl Buck

- Hidden Markov Models
 - Viterbi algorithm
- Gaussian Mixture Model

```
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
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

In HMM, a multivariate time series is modeled by a system of unobserved (hidden) states that follow a Markov process. There are **transition probabilities** associated with going from one state to another, or back to itself, represented by a transition matrix A . To move from one state to another, we are given information about the previous state and the current **observation** X : an observation is associated with an **emission probability** for each state θ . Finally we may have prior knowledge of the initial state of the system, that represented by a probability vector π . Hence a HMM model is defined by the parameters $\lambda = (A, \theta, \pi)$; we observe a sequence of values X , which are generated by unobserved state sequence Z .

We use the Python package `hmmlearn` to solve the three fundamental tasks:

- Evaluation refers to computing the likelihood of a particular observation, given a model, i.e. what is $P(X|\lambda)$. This problem is solved by the Foward-Backward algorithm, with its `score` method
- Decoding seeks to predict the sequence of hidden states in Z from the sequence of observations X , and is solved by a dynamic programming algorithm known as Viterbi algorithm, with the `predict` method.
- Training to learn the model parameters $\lambda = (A, \theta, \pi)$ by maximizing the likelihood, is solved by the Baum-Welch algorithm, a special instance of the Expectation-Maximization (EM) algorithm, with the `fit` method

Its `GaussianHMM` model assumes a Gaussian distribution for emissions, where `covariance_type` may be, in increasing order of the number of parameters to be estimated:

- “spherical”: each state uses a single variance value that applies to all features
- “diag”: each state uses a diagonal covariance matrix.
- “tied”: all states use the same full covariance matrix.
- “full”: each state uses a full (unrestricted) covariance matrix.

An information criterion such as `bic` can be used to select the best model balancing the goodness of fit and the number of parameters.

```
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)
        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,
```

(continues on next page)

(continued from previous page)

```

    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=42,
                                  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: -53166.15571297737 is not greater than -53166.
-155712466214. Delta is -5.111578502692282e-07
Model is not converging. Current: -6163.604416561489 is not greater than -6163.
-6044165126905. Delta is -4.8798938223626465e-08
Model is not converging. Current: 33662.62951552664 is not greater than 34171.
-8083827127. Delta is -509.17886718605587
Model is not converging. Current: 38239.33317549184 is not greater than 39427.
-428995004455. Delta is -1188.095819512615
Model is not converging. Current: -128529.4491603166 is not greater than -128529.
-44915884043. Delta is -1.4761608326807618e-06
Model is not converging. Current: -123452.93136393784 is not greater than -123452.
-93136010495. Delta is -3.8328871596604586e-06
Model is not converging. Current: -120079.61301174758 is not greater than -120079.
-61300834864. Delta is -3.398949047550559e-06
Model is not converging. Current: -130787.20497186083 is not greater than -130787.
-2049689297. Delta is -2.9311340767890215e-06
Model is not converging. Current: -125163.69776382159 is not greater than -125163.
-69776326745. Delta is -5.541369318962097e-07
Model is not converging. Current: -124252.97721651862 is not greater than -124252.
-97721368226. Delta is -2.836357452906668e-06

```

```
# 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 | 197612200.0 | 197649489.0 | 8001 | 98798099.0 | full | 1 |
| 12 | 11467703.0 | 11492530.0 | 5327 | 5728524.0 | diag | 6 |
| 14 | 197612200.0 | 197649489.0 | 8001 | 98798099.0 | tied | 1 |
| 24 | 7466415.0 | 7468909.0 | 535 | 3732673.0 | spherical | 4 |

```
# 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']
```

12.1.1 Markov Chains

- geometric run
- irreducibility

```
# 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 states and average INDPRO")
matrix.join(indpro).round(4)
```

HMM states and average INDPRO

| | 0 | 1 | 2 | 3 | Stationary | INDPRO |
|---|--------|--------|--------|--------|------------|---------|
| 0 | 0.6158 | 0.0000 | 0.0000 | 0.3842 | 0.0442 | -0.0076 |
| 1 | 0.0059 | 0.8874 | 0.0196 | 0.0870 | 0.2185 | 0.0010 |

(continues on next page)

(continued from previous page)

| | | | | | | |
|---|--------|--------|--------|--------|--------|---------|
| 2 | 0.0000 | 0.0081 | 0.8655 | 0.1264 | 0.4647 | 0.0049 |
| 3 | 0.0576 | 0.0765 | 0.2136 | 0.6523 | 0.2726 | -0.0007 |

Plot predicted states

```
def plot_states(modelname, labels, num=1, series_id='INDPRO', freq='M'):
    """helper to plot predicted states 'IPMANSICS'"""

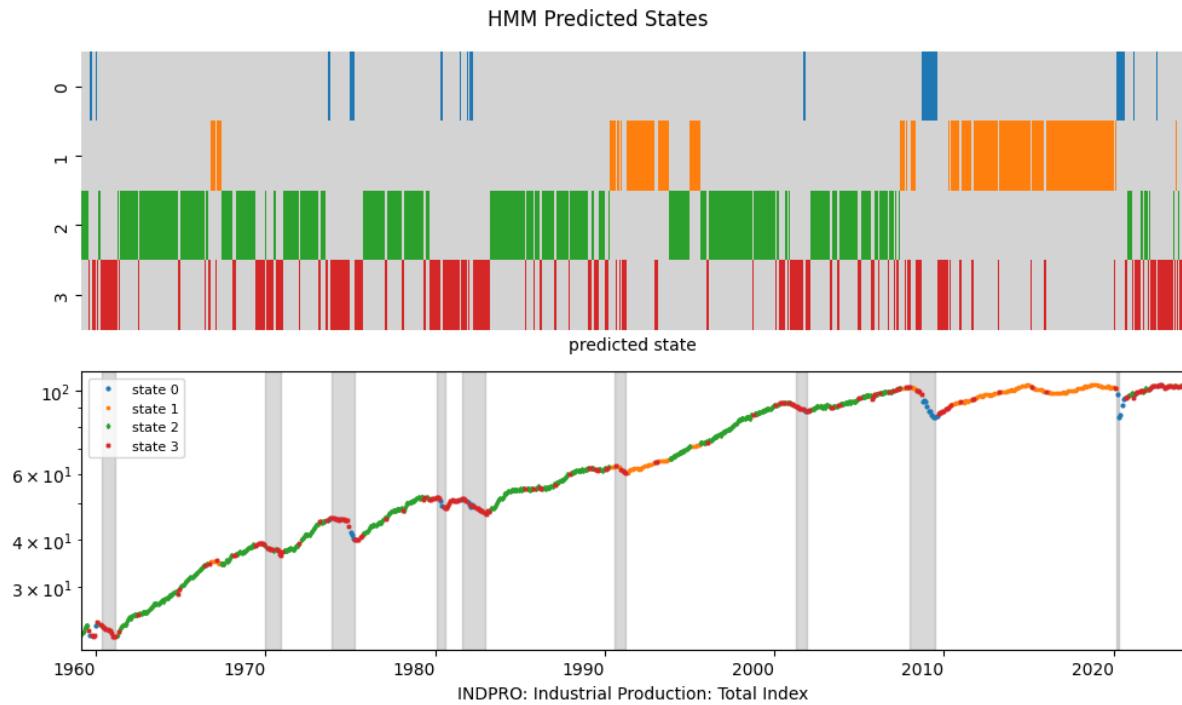
    # n_components markers
    n_components = len(np.unique(labels))
    markers = ["o", "s", "d", "X", "P", "8", "H", "*", "x", "+"][:n_components]

    fig, (bx, ax) = plt.subplots(nrows=2, ncols=1, figsize=(10, 6))
    fig.suptitle(f"{modelname.upper()} Predicted States")

    # 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')
    plt.tight_layout()
```

```
plot_states('HMM', pred_markov, num=1, freq=freq)
```

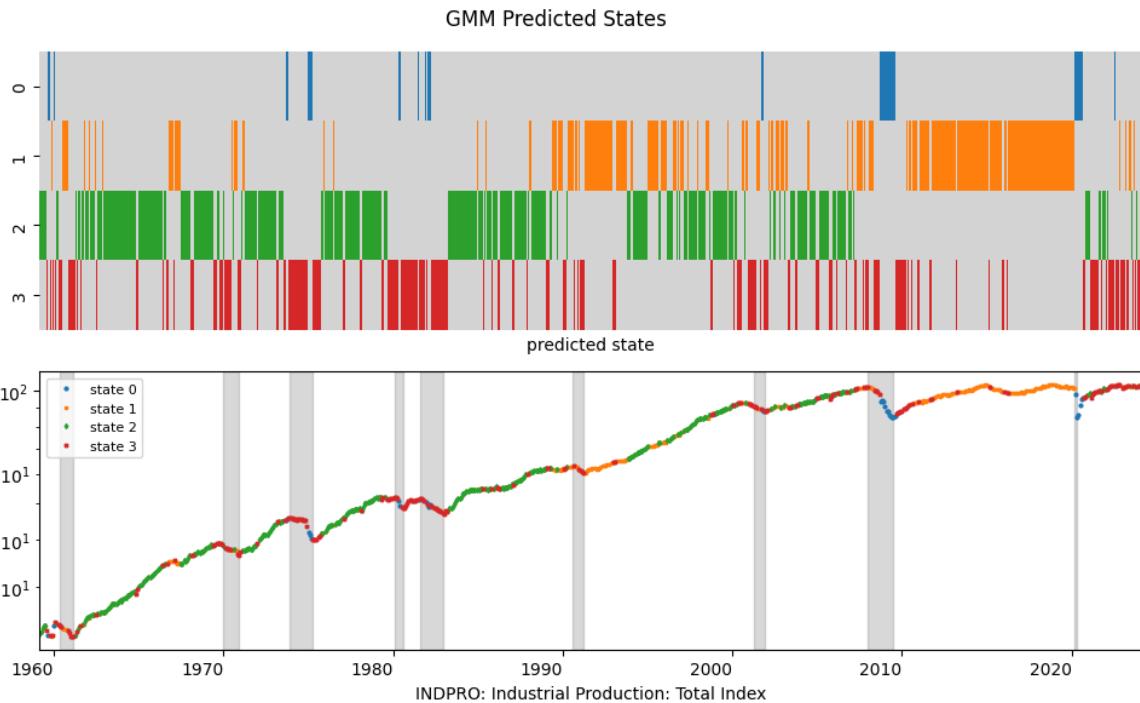


12.2 Gaussian Mixture Model

```

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)

```



```
# 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 | 0.800000 | 0.719231 |
| Stationary prob of state 0 | 0.044240 | 0.042254 |
| Stationary prob of state 1 | 0.218489 | 0.297055 |
| Stationary prob of state 2 | 0.464711 | 0.408451 |
| Stationary prob of state 3 | 0.272560 | 0.252241 |

```
# 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)
```

(continues on next page)

(continued from previous page)

```
return f"#{scale(r):02x}{scale(g):02x}{scale(b):02x}"
```

```
dot = graphviz.Digraph(engine='circo', graph_attr={'splines': 'true'})
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
    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')
```

```
dot
# dot.format = 'png'
# dot.view(filename='digraph') # Visualize the graph
```

```
<graphviz.graphs.Digraph at 0x7f97d2c7a3d0>
```

CHAPTER
THIRTEEN

TERM STRUCTURE OF INTEREST RATES

The best thing about the future is that it comes one day at a time - Abraham Lincoln

Concepts

- Interest Rates
- Yield curve
- Splines
- Bootstrapping
- PCA and SVD

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>

```
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 compounding frequency used for an interest rate defines the unit of measurement. If an interest rate is measured with annual compounding, the quoted interest rate is assumed to be compounded once a year; if measured with semi-annual compounding, it is assumed to be compounded twice a year; and so on. To convert a rate expressed with compounding frequency m times per year to an equivalent continuously-compounded rate, we equate the values at the end of a year:

$$e^{rt} = \left(1 + \frac{R_m}{m}\right)^{mt}$$

The annualized percentage yield (**APY**), or annualized effective yield, includes the effects of compounding. To convert a simple yield to an APY you need to apply the standard financial formula: $APY = (1 + R_m/m)^m - 1$, where m is the stated number of compounding periods per year for the quoted rate.

The **spot rate** for a certain maturity is the zero-coupon interest rate for that maturity. It is also referred to as the zero-coupon interest rate, or just the “zero.” It is directly related to the discount factor for that maturity: Discount factor $= (1 + \frac{R_m}{m})^{-t}$

The **forward rate** for a future period is the spot rate for that period implicit in the spot rates observed in the market today. Current longer-term spot rates can be calculated by compounding forward rates.

A bond's **yield to maturity** is the single spot rate, which if applied to all the bond's cash flows, would make the cash flows' present value equal to the bond's market price.

The **par rate** is the coupon rate on a bond that causes the bond price to equal par. A bond that is issued at par has a yield to maturity that is equal to the coupon rate.

The **term structure** of interest rates is the relationship between interest rates or bond yields and different terms or maturities. When graphed, the term structure of interest rates is known as a **yield curve**. When the term structure is upward-sloping, the forward rate for a period beginning at time T is greater than the T -year spot rate which in turn is greater than the T -year par rate. When the term structure is downward-sloping, the forward rate for a period beginning at time T is less than the T -year spot rate, which in turn is less than the T -year par rate.

Swap rates are the fixed rates exchanged for floating rates in a swap agreement. Swap rates are 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 |

(continues on next page)

(continued from previous page)

| | | | |
|--------|------------|---------|---------------------------|
| 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 |
| 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 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 2024-04-05 | 5.47 | 5.43 | 5.34 | 5.05 | 4.73 | 4.54 | 4.38 | 4.39 | 4.39 | 4.65 | 4.54 |
| 2024-04-08 | 5.48 | 5.43 | 5.35 | 5.07 | 4.78 | 4.60 | 4.43 | 4.43 | 4.42 | 4.65 | 4.55 |
| 2024-04-09 | 5.48 | 5.43 | 5.34 | 5.03 | 4.74 | 4.52 | 4.37 | 4.38 | 4.36 | 4.60 | 4.50 |
| 2024-04-10 | 5.49 | 5.45 | 5.40 | 5.19 | 4.97 | 4.77 | 4.61 | 4.59 | 4.55 | 4.76 | 4.64 |
| 2024-04-11 | 5.48 | 5.45 | 5.38 | 5.17 | 4.93 | 4.77 | 4.61 | 4.60 | 4.56 | 4.77 | 4.65 |

[15555 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
```

(continues on next page)

(continued from previous page)

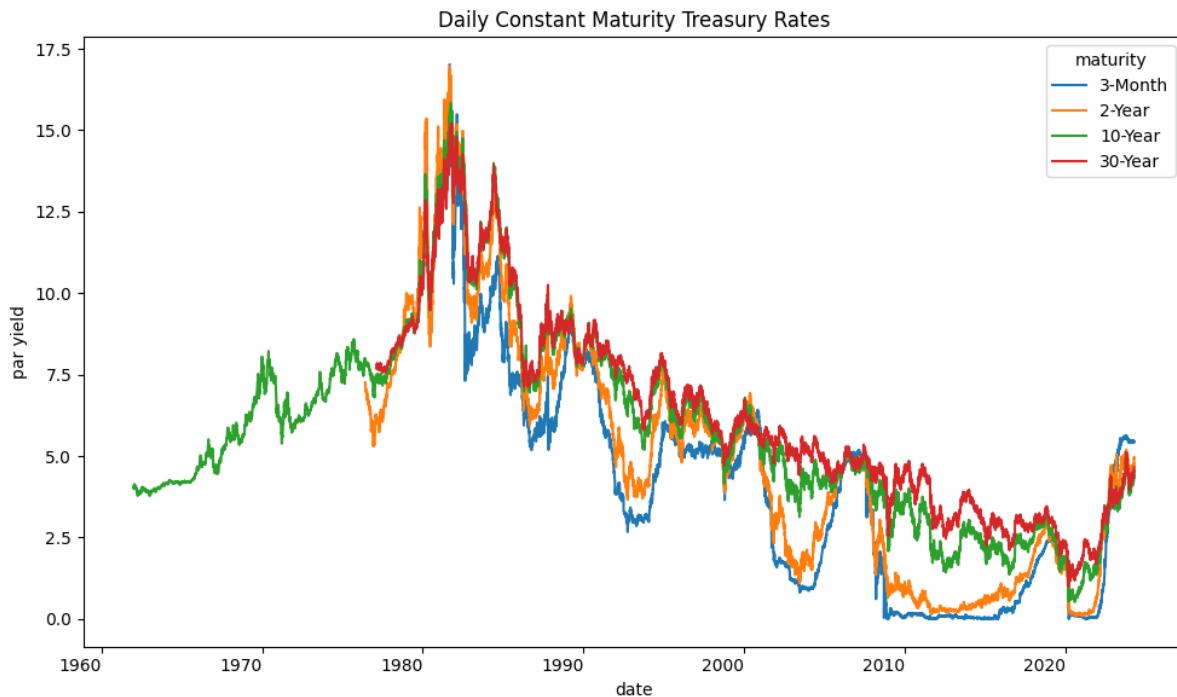
```
monthly = monthly.rename_axis(columns='maturity').sort_index(axis=1)
monthly
```

| 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 |
| 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 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 2023-11-30 | 5.53 | 5.52 | 5.44 | 5.28 | 4.88 | 4.64 | 4.49 | 4.53 | 4.50 | 4.84 | 4.66 |
| 2023-12-31 | 5.54 | 5.44 | 5.34 | 4.96 | 4.46 | 4.19 | 4.00 | 4.04 | 4.02 | 4.32 | 4.14 |
| 2024-01-31 | 5.54 | 5.45 | 5.21 | 4.79 | 4.32 | 4.11 | 3.98 | 4.03 | 4.06 | 4.39 | 4.26 |
| 2024-02-29 | 5.49 | 5.44 | 5.28 | 4.92 | 4.54 | 4.33 | 4.19 | 4.21 | 4.21 | 4.49 | 4.38 |
| 2024-03-31 | 5.51 | 5.47 | 5.36 | 4.99 | 4.59 | 4.38 | 4.20 | 4.21 | 4.21 | 4.46 | 4.36 |

[852 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"
```

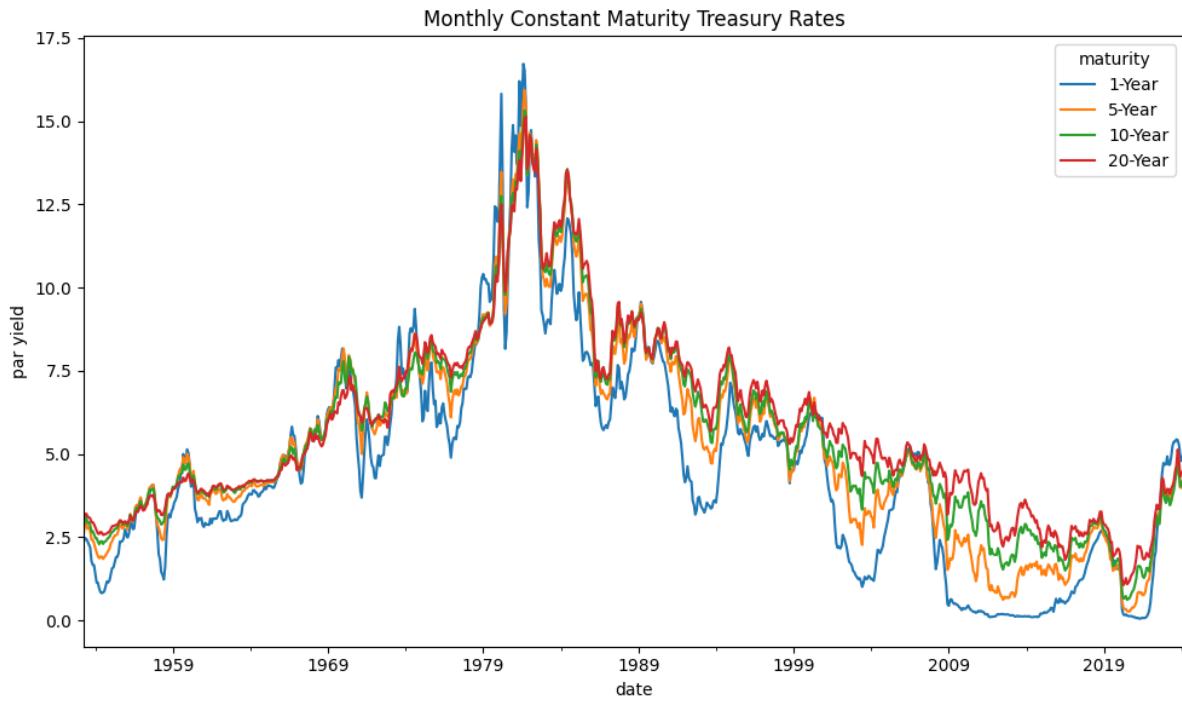
```
cols = [3, 24, 120, 360]
fig, ax = plt.subplots(figsize=(10, 6))
daily[cols].rename(columns=mapper).plot(ax=ax, rot=0)
plt.title('Daily Constant Maturity Treasury Rates')
plt.ylabel('par yield')
plt.tight_layout()
```



```

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 Treasury's official yield curve is a par yield curve derived using a monotone convex method. The inputs are indicative, bid-side market price quotations (not actual transactions) for the most recently auctioned securities obtained by the Federal Reserve Bank of New York at or near 3:30 PM each trading day.

Historically, Treasury has used a quasi-cubic hermite spline (HS) method for yield curve construction. The HS method directly uses secondary market yields as inputs to create a yield curve that is assumed to be a par curve. The monotone convex method for deriving the official Treasury yield curve replaced the previous quasi-cubic hermite spline method as of December 6, 2021.

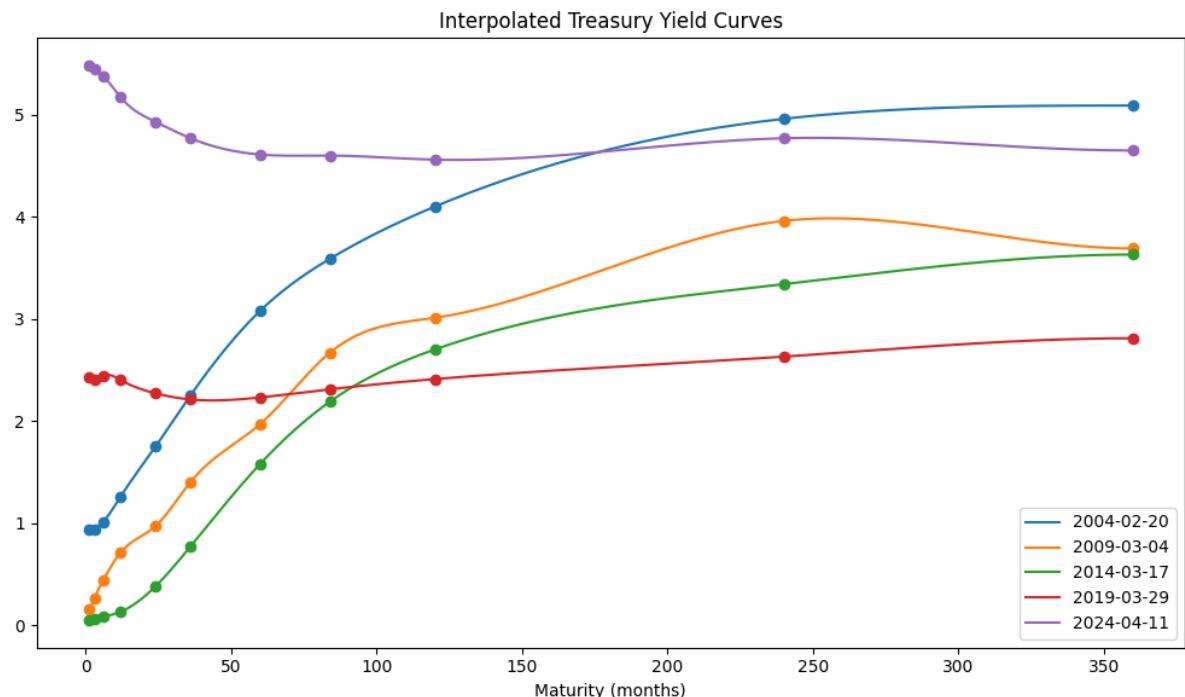
CMT yields are read directly from the Treasury's daily par yield curve and represent "bond equivalent yields" for securities that pay semiannual interest, which are expressed on a simple annualized basis. These yields are not effective annualized yields or Annualized Percentage Yields (APY), which include the effect of compounding. To convert a CMT yield to an APY you need to apply the standard financial formula: $APY = (1 + y/2)^2 - 1$

13.2.1 Splines

Interpolate yields at other maturities with a piecewise cubic polynomial which is twice continuously differentiable

```
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()
```



13.3 Bootstrap

This is the process of backing out spot rates by working forward and fitting zero-coupon rates to progressively longer maturity instruments

```
# 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

    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()
```

| | |
|----|-----------|
| | 2024-0411 |
| 6 | 0.053800 |
| 12 | 0.051673 |

(continues on next page)

(continued from previous page)

```
18 0.050150
24 0.049225
30 0.048370
```

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

    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

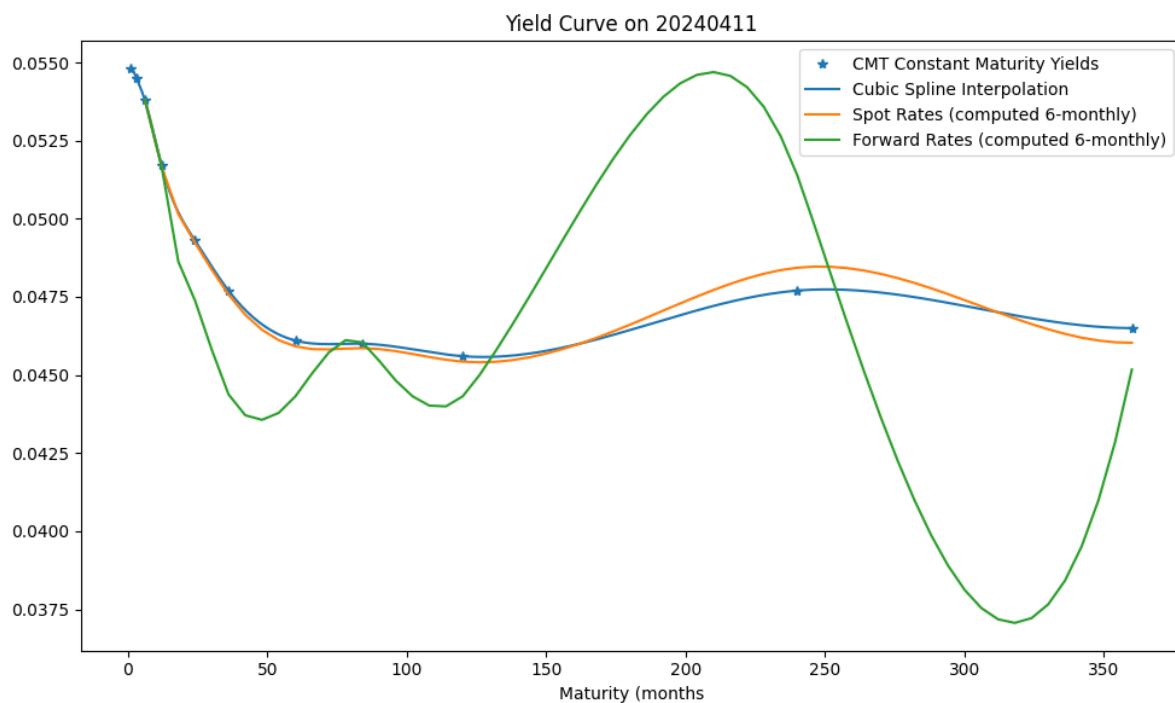
    Args:
        spots : Sequence of simple annual spot rates
        m : Number of compounding periods per year
        skip: Number of initial periods skipped

    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)')
plt.tight_layout()
```



There are more sophisticated methods for fitting yield curves to treasury prices

```
# Download reconstructed yield curve data by Liu and Wu (2021)
# file_id from https://sites.google.com/view/jingcynthiawu/yield-data
file_id = '15uGZet8bS5rEHy9nbGmgCHXyMGUSoHx3' # monthly
#file_id = '15o5ahwIs-Pbaq79H4Sm0LUHGAEbCFXMs' # 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")))
```

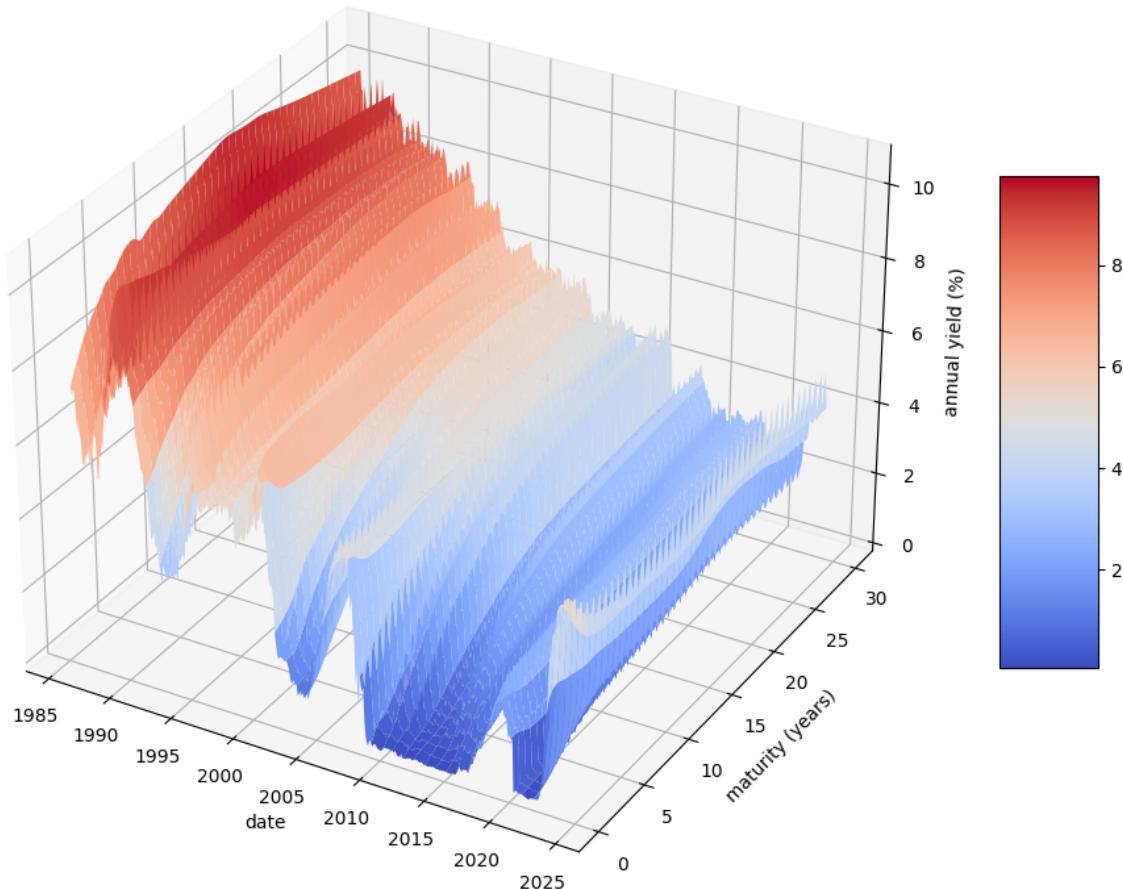
(continues on next page)

(continued from previous page)

```
+ 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')
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)]



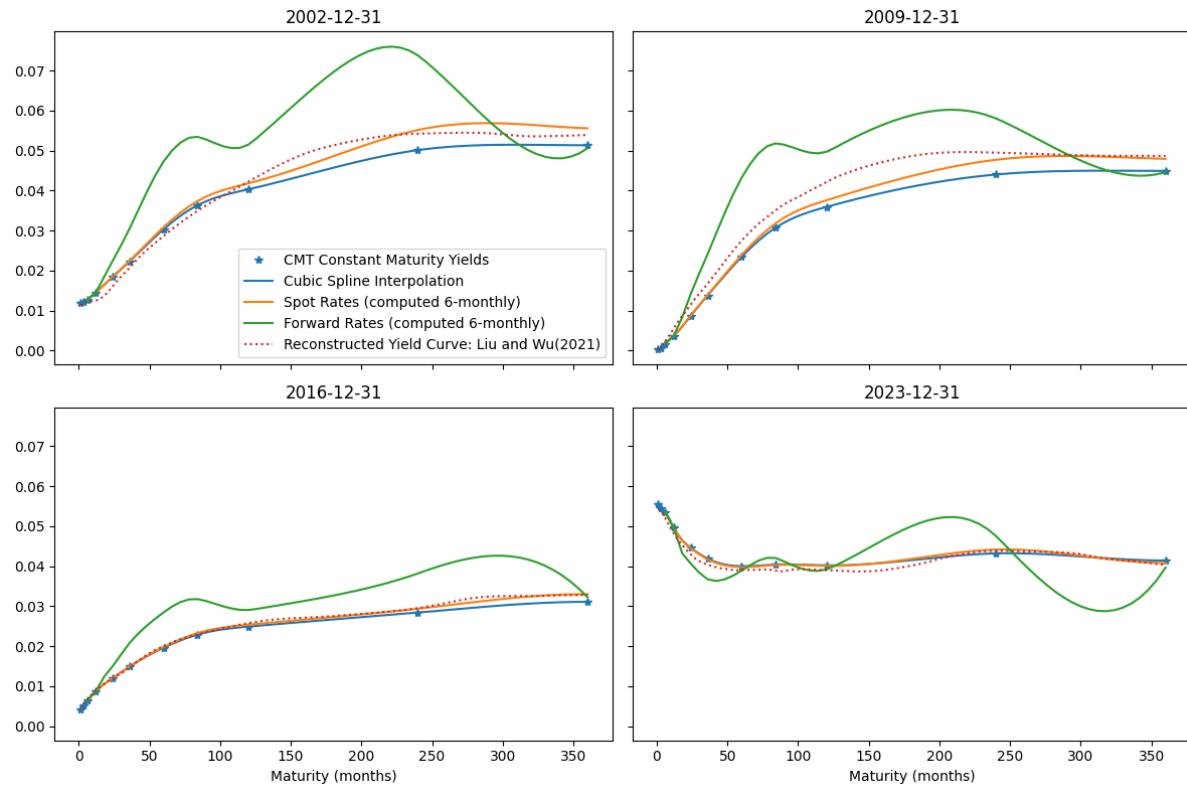
```
# Plot historical Yield Curves
curve_dates = sorted(liuwu.index[-1:0:-(7*12)])[-4:]
for date in curve_dates:
    yield_curve[date] = CubicSpline(
        x=monthly.columns.to_list(), y=monthly.loc[date].values, bc_type='clamped')
```

```
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]

    # 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()
```



13.4 Principal Component Analysis

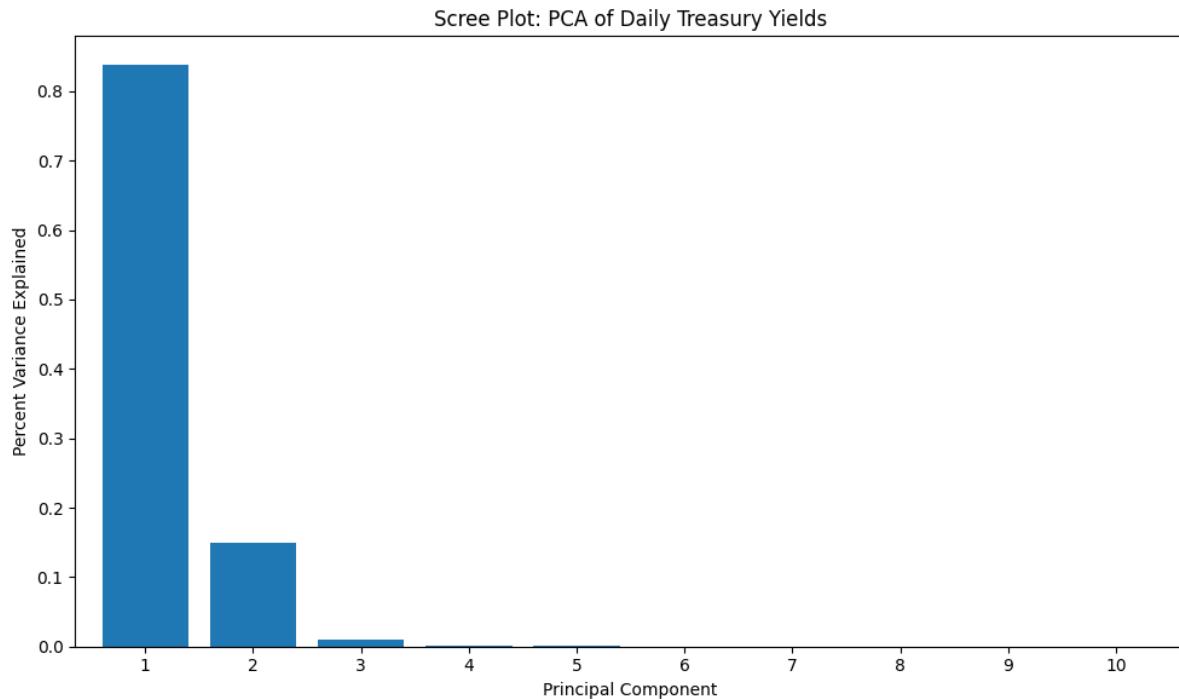
A principal components analysis finds multiple factors and estimates their relative importance in describing movements in the term structure. We carry out a principal components analysis on the daily changes in the Constant Maturities Treasury rates. The number of factors equals the number of rates.

The first factor accounts for 83.7% of the variance of all daily rate movements, while the first two account for 98.7%. The first three factors account for 99.8% of the variance, or most of the uncertainty in interest rate movements.

```
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_)]})
```

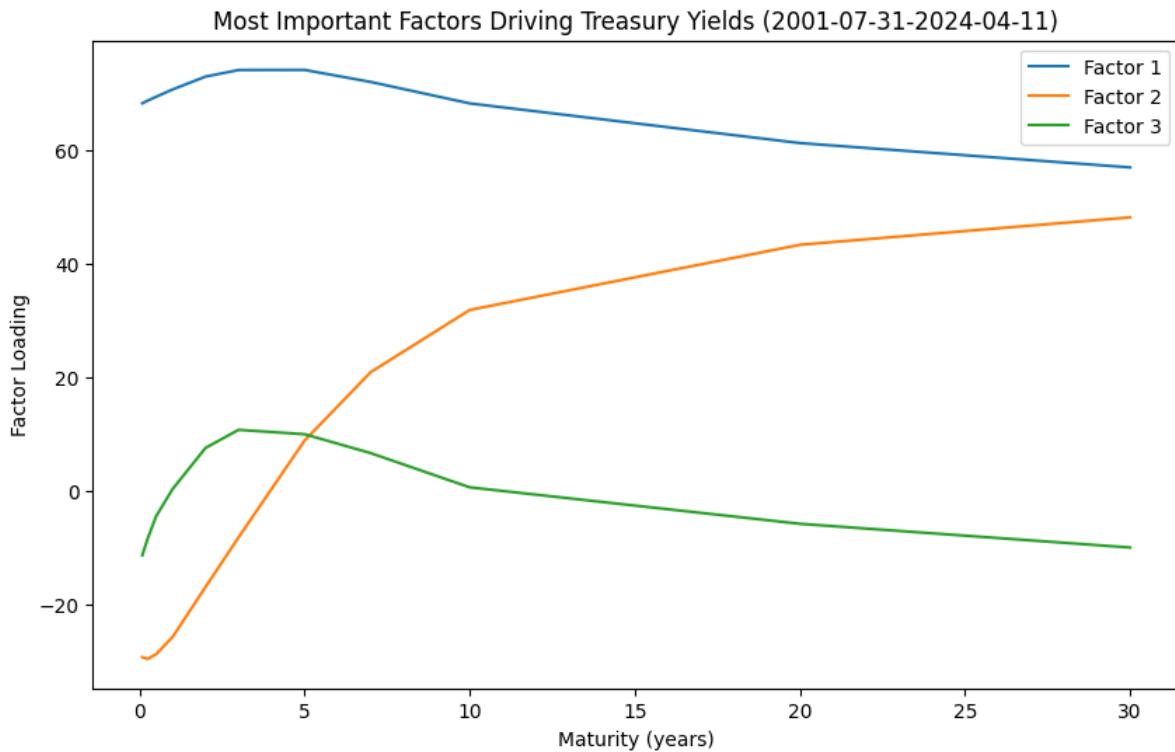
| Cumulative Variance Explained | |
|-------------------------------|----------|
| PC 1 | 0.837657 |
| PC 2 | 0.986801 |
| PC 3 | 0.997540 |
| PC 4 | 0.998884 |
| PC 5 | 0.999374 |
| PC 6 | 0.999713 |
| PC 7 | 0.999861 |
| PC 8 | 0.999925 |
| PC 9 | 0.999962 |
| PC 10 | 0.999985 |
| PC 11 | 1.000000 |

```
# 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_.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])[:10]}-{str(X.index[-1])[:10]})")
ax.set_xlabel('Maturity (years)')
ax.set_ylabel('Factor Loading')
```

Text(0, 0.5, 'Factor Loading')



Most important factors driving treasury yields:

- Factor 1 is a shift in the “level” of the term structure where all rates move in the same direction by approximately (but not exactly) the same amount.
- Factor 2 is a shift in the “slope” where short-term rates move in one direction and long-term rates move in the other direction. It corresponds to steepening or flattening of the term structure.
- Factor 3 is a shift in the “twist” of the term structure, where relatively short-term and relatively long-term rates move in one direction while intermediate rates move in the other direction

13.5 Singular Value Decomposition

Intuitively, PCA (also called eigendecomposition) is a rotation of the cloud of data points in Euclidean space where the co-ordinate axes are chosen such that each successive axis captures as much variance as possible.

Singular Value Decomposition (SVD) of a matrix is a factorization of that matrix into a product three matrices, which can be interpreted as a rotation, followed by a rescaling, followed by another rotation: $A = USV^T$.

The singular value decomposition of a data matrix A is a generalization of an eigendecomposition of the square symmetric matrix: $A^T A = V \Lambda V^T$.

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.

```
# 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
```

(5677, 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])
```

Low-rank approximation by PCA: $A' A \approx V D V'$

```
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:
```

```
    Frobenius Norm
K
1      0.175740
2      0.012739
3      0.001739
4      0.000728
```

Low-rank approximation by SVD: $A \approx USV'$

```
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      0.402918
2      0.114886
3      0.049601
4      0.033414
```

BOND RETURNS AND RISKS

The essence of investment management is the management of risks, not the management of returns - Benjamin Graham

Concepts

- Term structure changes
- Bond risk factors

References:

- FRM Part I Exam Book Valuation and Risk Models Ch12-13

```
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 Term structure changes

14.1.1 Duration

Duration, or Macaulay Duration, measures the change in bond price P with a instantaneous change in its yield y .

Consider a bond with price P and yield y . Because it leads to a simpler duration formula, we first measure the yield with continuous compounding. The present value relationship between P and y is $P = \sum_{t=1}^n c_t e^{-yt}$. Then after differentiating: $\Delta P = -\sum_{t=1}^n c_t t e^{-yt} \Delta y$

We define the yield-based duration as the proportional change in the bond price for a small change in the yield:

$$D = \frac{\Delta P}{P \Delta y} = \sum_{t=1}^n t \frac{c_t e^{-yt}}{P}$$

This formulation gives us another interpretation of duration. It can be calculated by taking an average of the times when cash flows are received weighted by the proportion of the bond's value received at each time. Duration is therefore a measure of how long an investor has to wait to receive cash flows. (This explains why the word “duration” was chosen to describe the sensitivity of proportional changes to yield).

Modified duration: If the yield on the bond is measured with semi-annual compounding rather than continuous compounding, the expression for (Macaulay) duration must be divided by $(1 + y_2/2)$ (or $1 + y_m/m$ for m'thly compounding).

Effective duration describes the percentage change in the price of a bond with embedded options due to a small change in all rates.

14.1.2 DV01

DV01 describes the impact of a one-basis-point change in interest rates on the value of a portfolio.

$$DV01 = -\frac{\Delta P}{\Delta r}$$

where Δr is the size of a small parallel shift in the interest rate term structure (measured in basis points).

14.1.3 Convexity

Bond duration and convexity are in the first two terms of the derivative of a bond's price with respect to interest rates in a Taylor expansion. Convexity measures the curvature of the bond price-interest rate relationship. It indicates how much the bond's duration changes as interest rates change. While duration helps estimate bond price changes for small interest rate movements, convexity accounts for larger changes, providing a more accurate prediction of bond price behavior

Yield-based convexity when yields are expressed with continuous compounding is

$$C = \frac{1}{P} \frac{1}{(1+r/m)^2} \sum_{t=1}^T (t/2m + (t/m)^2) \frac{C_t}{(1+r/m)^t}$$

It is 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 $(1 + y_m/2)^2$ with m'thly compounding) and the result is referred to as modified convexity. Effective convexity measures the sensitivity of the duration measure to changes in interest rates.

$$C = \frac{1}{P} \left[\frac{P^+ + P^- - 2P}{(\Delta r)^2} \right]$$

```
# 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
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

(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) * (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

```

```

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

```
# 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']) /
            (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 |

A **positive convexity** improves the bond holder's position when there is a parallel shift in interest rates. While both strategies provide a yield of 4% and a duration of 8.1758, the barbell strategy always produces a better result when there is a parallel shift in the yield curve. An opportunity for arbitrageurs:

- Invest a certain USD amount in the barbell, and
- Short the same USD amount of the bullet

This would be profitable if shifts in the term structure were always parallel.

14.1.5 Key rate shifts

Suppose we consider three spot rates: the two-year rate, the five-year rate, and the ten-year rate. Each of these can affect spot rates in their neighborhoods, and the combined effect of all three key rate shifts is a one-basis-point shift in all rates. Shifts in each of those spot rates can be defined as shown below. These shifts are sometimes referred to as **key rate shifts**, and provide a way of splitting the DV01 measure into three separate measures.

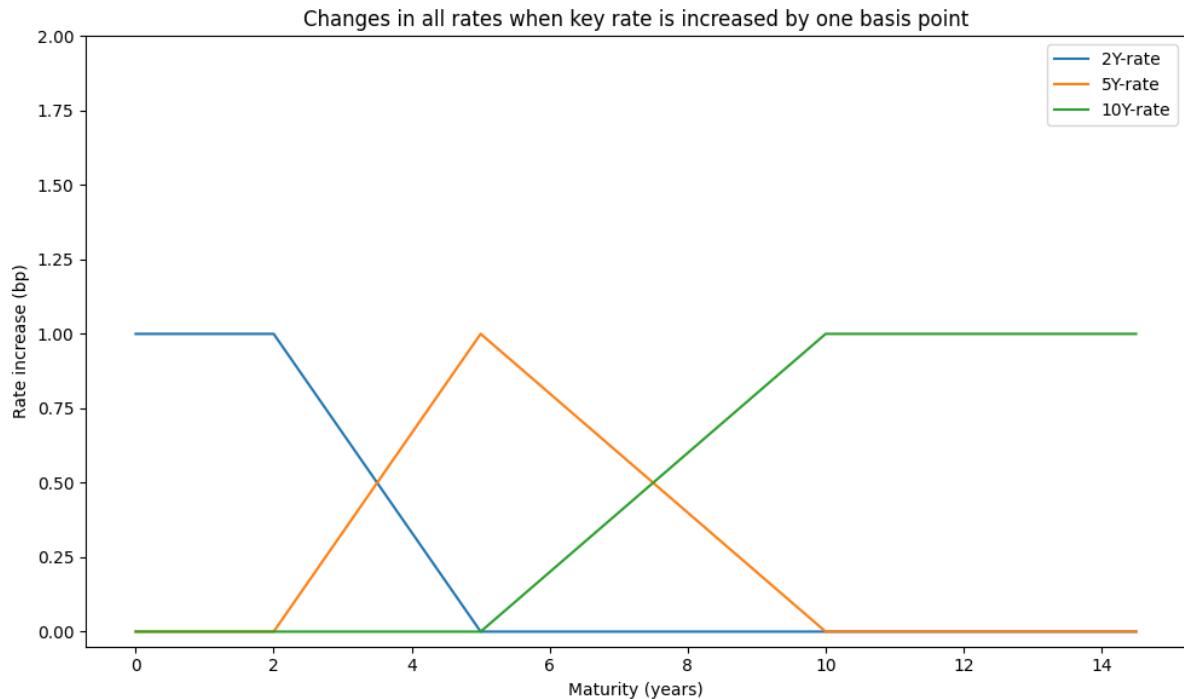
The impact of the shifts are called **partial 01s** or **key rate 01s** (KR01s), e.g.:

- KR01₁: The reduction in a portfolio's value from a one-basis-point increase in the two-year spot rate
- KR01₂: The reduction in a portfolio's value from a one-basis-point increase in the five-year spot rate
- KR01₃: The reduction in a portfolio's value from a one-basis-point increase in the ten-year spot rate

It follows that: $DV01 = KR01_1 + KR01_2 + KR01_3$. A bond investor can estimate portfolio values under, or hedge against, a wider range of (small) term structure movements.

```
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 Bond risk factors

```
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']:
    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 | 20240418 |

```
# 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()
rets
```

| | BAMLCC0A0CMTRIV | BAMLCC0A1AAATTRIV | BAMLCC0A2AAATTRIV | BAMLCC0A3ATTRIV | \ |
|----------|------------------|-------------------|--------------------|-----------------|-----|
| 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 | |
| ... | ... | ... | ... | ... | ... |
| 20240411 | -0.001330 | -0.001659 | -0.001234 | -0.001240 | |
| 20240412 | 0.001673 | 0.001828 | 0.001822 | 0.001739 | |
| 20240415 | -0.006152 | -0.008150 | -0.006125 | -0.006174 | |
| 20240416 | -0.004341 | -0.004776 | -0.003966 | -0.004142 | |
| 20240417 | 0.004802 | 0.006168 | 0.004689 | 0.004873 | |
| | 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 | | |
| ... | ... | ... | ... | ... | |
| 20240411 | -0.001418 | 0.000582 | -0.000040 | | |
| 20240412 | 0.001592 | 0.000727 | 0.001509 | | |
| 20240415 | -0.006110 | -0.000410 | -0.002345 | | |
| 20240416 | -0.004571 | -0.001117 | -0.002740 | | |
| 20240417 | 0.004725 | 0.001034 | 0.002748 | | |
| | 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 | ... | |
| ... | ... | ... | ... | ... | |
| 20240411 | -0.000588 | -0.001859 | -0.001963 | ... | |
| 20240412 | 0.002151 | 0.001789 | 0.002657 | ... | |
| 20240415 | -0.004446 | -0.006793 | -0.008662 | ... | |

(continues on next page)

(continued from previous page)

| | | | | |
|----------------------|-----------|-----------|-----------|-----|
| 20240416 | -0.003974 | -0.004730 | -0.005714 | ... |
| 20240417 | 0.004275 | 0.005568 | 0.006826 | ... |
| \ | | | | |
| BAMLEMPTPRVICRPITRIV | | | | |
| 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 | |
| ... | ... | ... | ... | ... |
| 20240411 | -0.001561 | -0.000714 | -0.003072 | |
| 20240412 | 0.001016 | 0.001856 | 0.000311 | |
| 20240415 | -0.002430 | -0.002237 | -0.003116 | |
| 20240416 | -0.002959 | -0.002457 | -0.002725 | |
| 20240417 | 0.001567 | 0.002147 | 0.001095 | |
| \ | | | | |
| BAMLEMRLCRPILATRIV | | | | |
| 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 | |
| ... | ... | ... | ... | ... |
| 20240411 | -0.003734 | -0.002164 | -0.001999 | |
| 20240412 | -0.000522 | 0.000781 | -0.000229 | |
| 20240415 | -0.004952 | -0.003309 | -0.000858 | |
| 20240416 | -0.004515 | -0.003117 | -0.004533 | |
| 20240417 | 0.001602 | 0.001795 | 0.001092 | |
| \ | | | | |
| BAMLHYH0A0HYM2TRIV | | | | |
| 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 | |
| ... | ... | ... | ... | ... |
| 20240411 | -0.001882 | -0.001767 | -0.001925 | |
| 20240412 | -0.001196 | -0.000885 | -0.001148 | |
| 20240415 | -0.002717 | -0.002836 | -0.002138 | |
| 20240416 | -0.004080 | -0.003503 | -0.004105 | |
| 20240417 | 0.000397 | 0.000712 | 0.000162 | |
| \ | | | | |
| BAMLHYH0A3CMTRIV | | | | |
| date | | | | |
| 19990104 | 0.003234 | | | |
| 19990105 | 0.005428 | | | |
| 19990106 | 0.003480 | | | |
| 19990107 | 0.001005 | | | |
| 19990108 | 0.007643 | | | |
| ... | ... | | | |
| 20240411 | -0.002184 | | | |
| 20240412 | -0.002668 | | | |
| 20240415 | -0.003969 | | | |

(continues on next page)

(continued from previous page)

```
20240416      -0.006513
20240417      -0.000130
```

```
[6603 rows x 33 columns]
```

```
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
BAMLCC0A2AAATRIV           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
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
BAMLEMFSFCRPITRIV          ICE BofA Private Sector Financial Emerging Markets
  ↵Corporate Plus Index Total Return Index Value
BAMLEMHBHYCRPITRIV          ICE BofA High Yield Emerging Markets
```

(continues on next page)

(continued from previous page)

| | |
|--|--|
| ↳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 | |
| BAMLEMPTPRVICRPITRIV | ICE BofA Private Sector Issuers Emerging Markets |
| ↳Corporate Plus Index Total Return Index Value | |
| BAMLEMRCRPIASIASTRIV | 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 | |

```
# 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]
```

```
[5.90633015e-01 1.84670310e-01 7.66846462e-02 2.31249861e-02
 2.08709778e-02 1.73997768e-02 1.59048070e-02 1.11428201e-02
 1.08065948e-02 9.53333795e-03 7.17195056e-03 5.80339671e-03
 5.13102872e-03 4.61504627e-03 3.95216602e-03 2.55170452e-03
 2.42215195e-03 2.22998793e-03 1.31893510e-03 9.51987980e-04
 7.84773755e-04 6.15732469e-04 3.99797350e-04 3.69901192e-04
 3.07591420e-04 2.74143218e-04 9.27920937e-05 8.50594135e-05
 5.40645899e-05 3.96089404e-05 3.55645916e-05 1.55810718e-05
 5.76269423e-06]
```

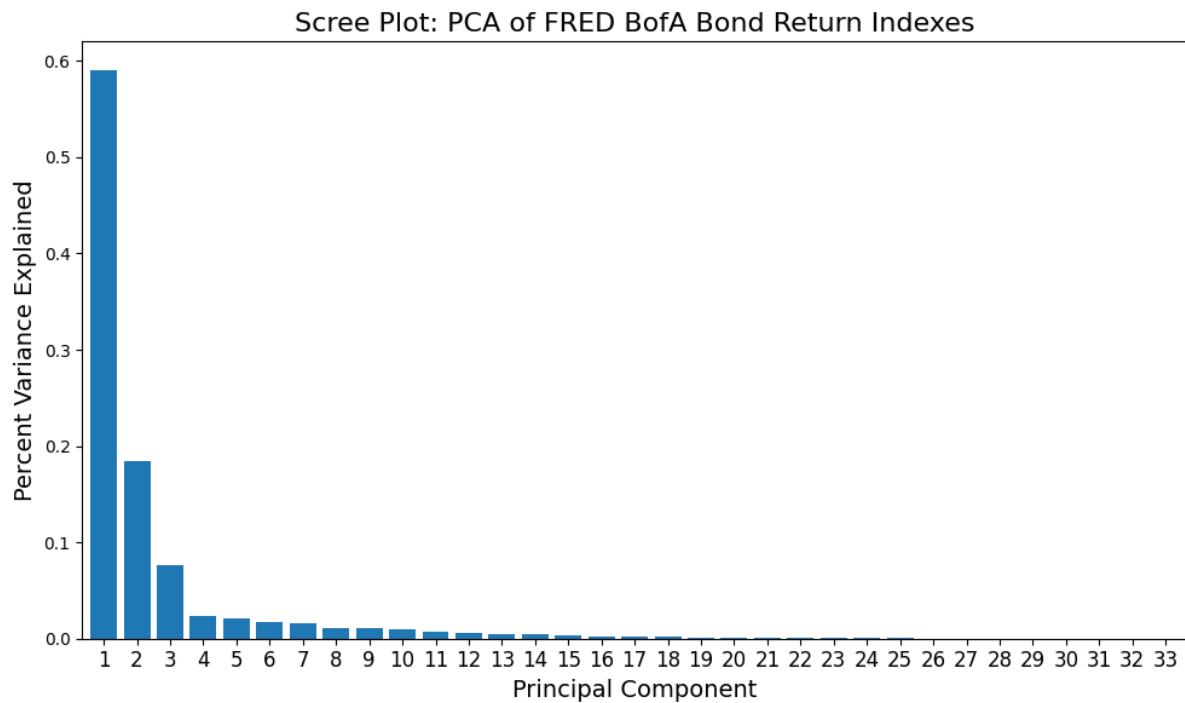
| Cumulative Variance Ratio Explained | |
|-------------------------------------|----------|
| 1 | 0.590633 |
| 2 | 0.775303 |
| 3 | 0.851988 |
| 4 | 0.875113 |
| 5 | 0.895984 |
| 6 | 0.913384 |
| 7 | 0.929289 |

(continues on next page)

(continued from previous page)

| | |
|----|----------|
| 8 | 0.940431 |
| 9 | 0.951238 |
| 10 | 0.960771 |

```
# 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', 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()
```



14.2.1 Explainability of Statistical 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 changes in interest rate spreads:

- level: average of 2-year and 10-year treasury rates
- slope: 10-year minus 2-year rates
- twist: (10-year minus 5-year) minus (5-year minus 2-year)
- credit: BAA minus 10-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
```

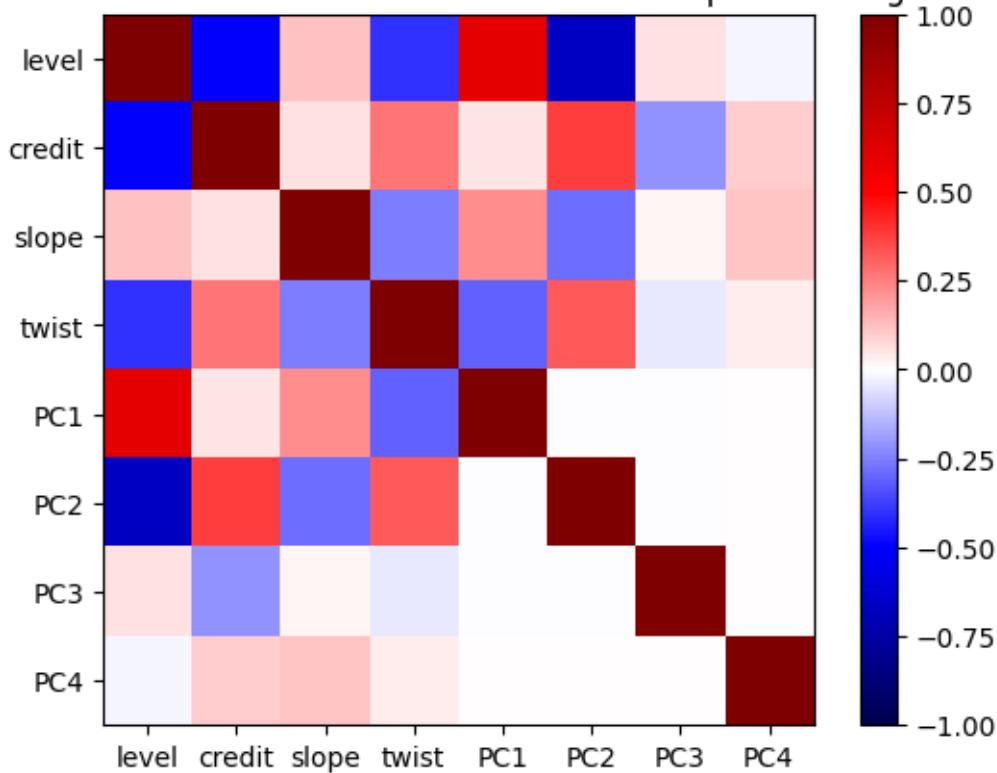
| | level | credit | slope | twist |
|------------|--------|--------|---------------|---------------|
| date | | | | |
| 1986-01-03 | 0.010 | -0.04 | 1.776357e-15 | 1.776357e-15 |
| 1986-01-06 | 0.015 | -0.01 | 1.000000e-02 | -1.000000e-02 |
| 1986-01-07 | -0.100 | 0.06 | -6.000000e-02 | -8.881784e-16 |
| 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 |
| ... | ... | ... | ... | ... |
| 2024-04-12 | -0.055 | 0.04 | -1.000000e-02 | 3.000000e-02 |
| 2024-04-15 | 0.090 | 0.01 | 8.000000e-02 | -4.000000e-02 |
| 2024-04-16 | 0.040 | 0.00 | 0.000000e+00 | 0.000000e+00 |
| 2024-04-17 | -0.060 | 0.02 | -4.000000e-02 | 2.000000e-02 |
| 2024-04-18 | 0.050 | -0.01 | -8.881784e-16 | -2.000000e-02 |

[9580 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)
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

| | PC1 | R-squared: | 0.540 | | | |
|-------------------|------------------|---------------------|-----------|-------|---------|--------|
| Dep. Variable: | | Model: | OLS | | | |
| Method: | Least Squares | F-statistic: | 422.9 | | | |
| Date: | Sun, 21 Apr 2024 | Prob (F-statistic): | 2.47e-323 | | | |
| Time: | 13:15:12 | Log-Likelihood: | -16027. | | | |
| No. Observations: | 6318 | AIC: | 3.206e+04 | | | |
| Df Residuals: | 6313 | BIC: | 3.210e+04 | | | |
| Df Model: | 4 | | | | | |
| Covariance Type: | HAC | | | | | |
| | coef | std err | z | P> z | [0.025 | 0.975] |
| Intercept | 0.0222 | 0.091 | 0.245 | 0.807 | -0.156 | 0.200 |
| credit | 66.9094 | 10.707 | 6.249 | 0.000 | 45.923 | 87.895 |
| level | 66.9367 | 3.158 | 21.195 | 0.000 | 60.747 | 73.127 |
| slope | 9.1005 | 2.491 | 3.653 | 0.000 | 4.218 | 13.983 |
| twist | -13.9458 | 2.207 | -6.319 | 0.000 | -18.271 | -9.620 |

Omnibus: 4119.817 Durbin-Watson: 1.162

(continues on next page)

(continued from previous page)

| | | | |
|----------------|--------|-------------------|------------|
| Prob(Omnibus): | 0.000 | Jarque-Bera (JB): | 216965.568 |
| Skew: | 2.469 | Prob(JB): | 0.00 |
| Kurtosis: | 31.281 | Cond. No. | 40.5 |

Notes:

[1] Standard Errors are heteroscedasticity and autocorrelation robust (HAC) using ≈ 63 lags and without small sample correction

OLS Regression Results

| | | | |
|-------------------|------------------|---------------------|-----------|
| Dep. Variable: | PC2 | R-squared: | 0.492 |
| Model: | OLS | Adj. R-squared: | 0.492 |
| Method: | Least Squares | F-statistic: | 273.9 |
| Date: | Sun, 21 Apr 2024 | Prob (F-statistic): | 1.99e-217 |
| Time: | 13:15:12 | Log-Likelihood: | -12670. |
| No. Observations: | 6318 | AIC: | 2.535e+04 |
| Df Residuals: | 6313 | BIC: | 2.538e+04 |
| Df Model: | 4 | | |
| Covariance Type: | HAC | | |

| | coef | std err | z | P> z | [0.025 | 0.975] |
|-----------|----------|---------|---------|-------|---------|---------|
| Intercept | 0.0038 | 0.049 | 0.079 | 0.937 | -0.092 | 0.099 |
| credit | 7.9067 | 3.362 | 2.352 | 0.019 | 1.318 | 14.496 |
| level | -28.0466 | 1.680 | -16.695 | 0.000 | -31.339 | -24.754 |
| slope | -14.4437 | 1.191 | -12.131 | 0.000 | -16.777 | -12.110 |
| twist | 0.1590 | 1.291 | 0.123 | 0.902 | -2.372 | 2.690 |

| | | | |
|----------------|----------|-------------------|------------|
| Omnibus: | 4046.973 | Durbin-Watson: | 1.259 |
| Prob(Omnibus): | 0.000 | Jarque-Bera (JB): | 201753.832 |
| Skew: | 2.419 | Prob(JB): | 0.00 |
| Kurtosis: | 30.258 | Cond. No. | 40.5 |

Notes:

[1] Standard Errors are heteroscedasticity and autocorrelation robust (HAC) using ≈ 63 lags and without small sample correction

OLS Regression Results

| | | | |
|-------------------|------------------|---------------------|-----------|
| Dep. Variable: | PC3 | R-squared: | 0.049 |
| Model: | OLS | Adj. R-squared: | 0.048 |
| Method: | Least Squares | F-statistic: | 9.627 |
| Date: | Sun, 21 Apr 2024 | Prob (F-statistic): | 9.28e-08 |
| Time: | 13:15:12 | Log-Likelihood: | -11869. |
| No. Observations: | 6318 | AIC: | 2.375e+04 |
| Df Residuals: | 6313 | BIC: | 2.378e+04 |
| Df Model: | 4 | | |
| Covariance Type: | HAC | | |

| | coef | std err | z | P> z | [0.025 | 0.975] |
|-----------|----------|---------|--------|-------|---------|--------|
| Intercept | -0.0046 | 0.025 | -0.181 | 0.857 | -0.054 | 0.045 |
| credit | -12.9168 | 2.346 | -5.507 | 0.000 | -17.514 | -8.319 |
| level | -2.0945 | 1.339 | -1.564 | 0.118 | -4.719 | 0.530 |
| slope | 1.7102 | 1.241 | 1.378 | 0.168 | -0.722 | 4.143 |
| twist | 0.3878 | 1.103 | 0.352 | 0.725 | -1.773 | 2.549 |

(continues on next page)

(continued from previous page)

```
=====
Omnibus:                 3422.015   Durbin-Watson:           1.565
Prob(Omnibus):           0.000     Jarque-Bera (JB):      498792.470
Skew:                   1.565     Prob(JB):                  0.00
Kurtosis:                46.416   Cond. No.                 40.5
=====
```

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:            0.023
Method:                         Least Squares   F-statistic:            12.35
Date:                          Sun, 21 Apr 2024   Prob (F-statistic):      5.27e-10
Time:                           13:15:12     Log-Likelihood:          -8150.9
No. Observations:                6318    AIC:                  1.631e+04
Df Residuals:                   6313    BIC:                  1.635e+04
Df Model:                        4
Covariance Type:                 HAC
=====
```

| | coef | std err | z | P> z | [0.025 | 0.975] |
|-----------|---------|---------|--------|-------|--------|--------|
| Intercept | -0.0015 | 0.013 | -0.111 | 0.911 | -0.027 | 0.024 |
| credit | 2.6211 | 1.654 | 1.585 | 0.113 | -0.620 | 5.862 |
| level | 0.6178 | 0.766 | 0.807 | 0.420 | -0.883 | 2.119 |
| slope | 2.7265 | 0.431 | 6.329 | 0.000 | 1.882 | 3.571 |
| twist | 1.6352 | 0.454 | 3.606 | 0.000 | 0.746 | 2.524 |

```
=====
Omnibus:                 2609.560   Durbin-Watson:           1.953
Prob(Omnibus):           0.000     Jarque-Bera (JB):      152469.248
Skew:                   1.171     Prob(JB):                  0.00
Kurtosis:                26.952   Cond. No.                 40.5
=====
```

Notes:

[1] Standard Errors are heteroscedasticity and autocorrelation robust (HAC) using
 ↵63 lags and without small sample correction

CHAPTER
FIFTEEN

CONDITIONAL VOLATILITY AND VAR

The only constant in life is change - Heraclitus

Concepts:

- Value at Risk
- Coherence and Expected Shortfall
- Cryptocurrency Indices
- Historical Simulation and Stressed VaR
- Bootstrapping
- EWMA and GARCH
- Backtesting VaR

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

```
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
```

15.1 Value at Risk

Value at Risk (VaR) is an important risk measure that focuses on adverse events and their probability. A VaR with a confidence level of $\alpha\%$ is found by searching for the loss that has an $\alpha\%$ chance of being exceeded.

15.1.1 Coherence

One problem with VaR is that it does not say anything about how bad losses might be when they exceed the VaR level. Furthermore, Artzner et al. proposed four properties a risk measure should have.

1. Monotonicity: If (regardless of what happens) a portfolio always produces a worse result than another portfolio, it should have a higher risk measure.
2. Translation Invariance: If an amount of cash K is added to a portfolio, its risk measure should decrease by K .
3. Homogeneity: Changing the size of a portfolio by multiplying the amounts of all the components by X results in the risk measure being multiplied by X .
4. Subadditivity: For any two portfolios, the risk measure for the portfolio formed by merging them should be no greater than the sum of the risk measures for the portfolios.

A risk measure that satisfies all four conditions is termed coherent. **Expected shortfall (ES)** has all four properties, while VaR has only the first three properties. ES is computed as the probability-weighted average of tail losses. Other coherent risk measures can be derived by applying a risk aversion function to weight the quantiles. ES is a special case where the risk aversion weights are equal for all tail-loss quantiles.

```
# 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()
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.tail()
```

| | CBBCHUSD | CBBTCUSD | CBETHUSD | CBLTCUSD |
|------------|-----------|-----------|-----------|-----------|
| date | | | | |
| 2024-04-11 | -0.020373 | -0.007875 | -0.010817 | 0.024184 |
| 2024-04-12 | -0.141308 | -0.042222 | -0.080178 | -0.138679 |

(continues on next page)

(continued from previous page)

```
2024-04-13 -0.104275 -0.046878 -0.070173 -0.104611
2024-04-14  0.084578  0.024820  0.043828  0.025499
2024-04-15 -0.031220 -0.035412 -0.016538 -0.018678
```

15.1.2 Parametric method

Under the assumption that returns normally distributed, VaR and ES at the confidence level α is:

$$VaR = -\mu + \sigma z_{1-\alpha}$$

$$ES = -\mu + \sigma \frac{\exp(-z_{1-\alpha}^2/2)}{(1-\alpha)\sqrt{2\pi}}$$

where z_α is the standard normal variate corresponding to α .

For example, with $\alpha = 0.95$, $z_\alpha = -1.645$ which is the value whose cdf is the bottom 5% tail. In practice, μ and σ are not known and must be estimated. Since means are measured less precisely than volatility, and the returns are observed at short intervals, μ is often assumed to be 0.

The normal probability distribution has skewness (third central moment) equal to zero and kurtosis (fourth central moment) equal to 3. Under the assumption that geometric returns are normally distributed, then arithmetic returns have a **lognormal distributed**. The skewness of the lognormal is $(\exp(\sigma^2) + 2)\sqrt{(\exp(\sigma^2) - 1)}$, and 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 varies from a minimum of over 3, and hence exhibits fatter tails than the normal.

```
# 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)

| | CCBCHUSD | CCBTCUSD | CBETHUSD | CBLTCUSD |
|--------------------|----------|----------|----------|----------|
| volatility | 0.0597 | 0.0407 | 0.0520 | 0.0557 |
| value_at_risk | 0.0982 | 0.0669 | 0.0855 | 0.0916 |
| expected_shortfall | 0.1232 | 0.0839 | 0.1072 | 0.1149 |

15.1.3 Delta-Normal method

This model is used to provide an approximate results for non-linear portfolios. The underlying assets or risk factors are modeled with a normal distribution. The risk of a portfolio, which may exhibit non-linear payoffs with options securities, is modeled by its delta, or the instantaneous rate of change in its value with respect to the rate of change in the underlying. A better approximation includes the second derivative of the portfolio value with the underlying assets or risk factors, called the delta-gamma method.

15.1.4 Monte Carlo Simulation method

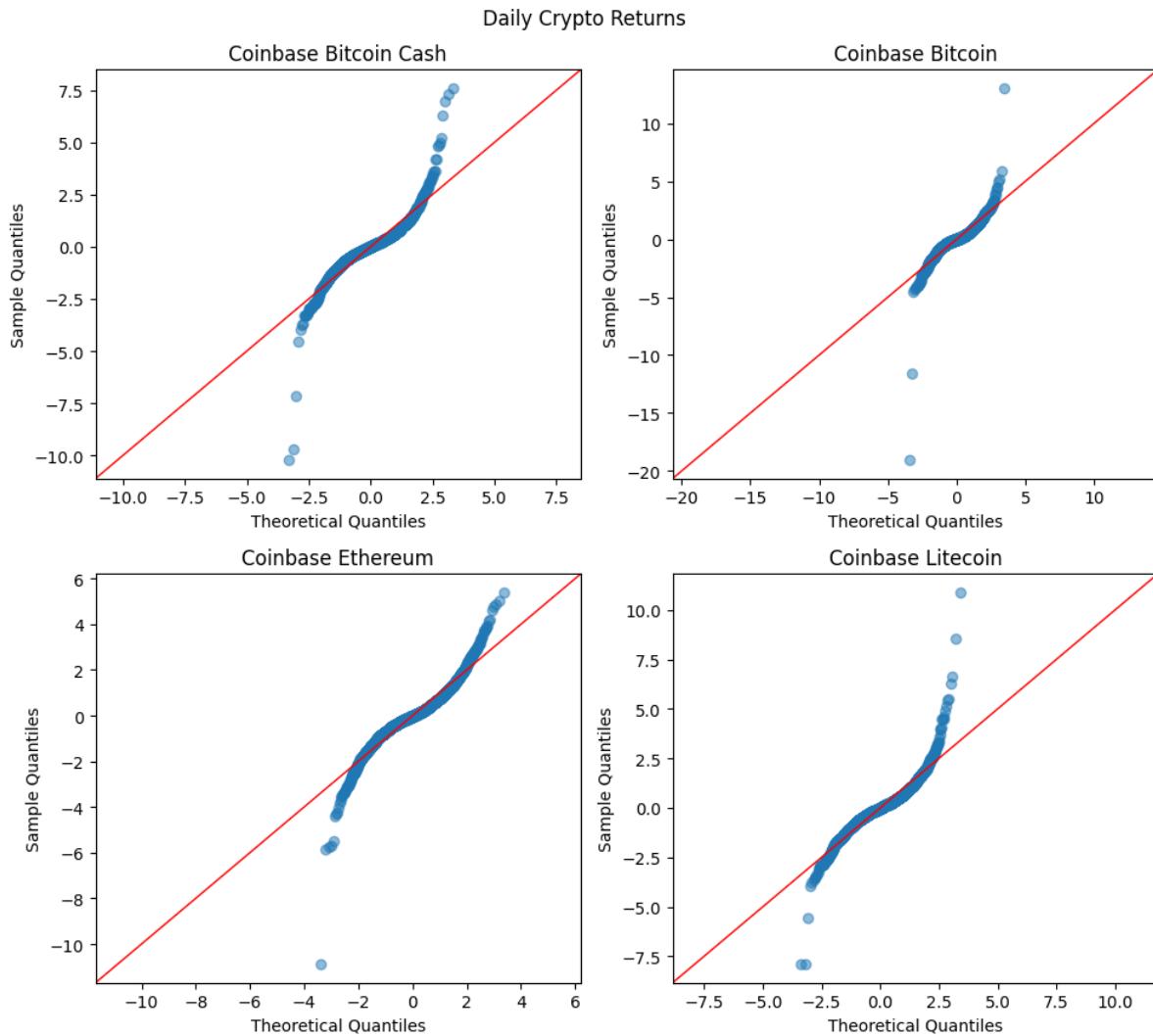
Monte Carlo simulation works for both linear and non-linear portfolios. This method generates scenarios by taking random samples from the normal (or other theoretical) distribution assumed for the underlying assets or risk factors. Suppose 1000 MC scenarios are simulated: the VaR with $\alpha = 95\%$ confidence will be the 50th (i.e. 5th percentile) worst loss and expected shortfall is the average of the 49 losses worse than this.

However, simple models based on Gaussian and independence assumptions may not be appropriate for the data sample, requiring the consideration of **non-parametric** approaches.

The QQ plot compare the distribution of the data sample to a reference normal distribution. The plot would appear linear when the observed quantiles are close to the theoretical quantiles. If the empirical distribution has heavier tails than the reference distribution, the QQ plot will have steeper slopes at its tails.

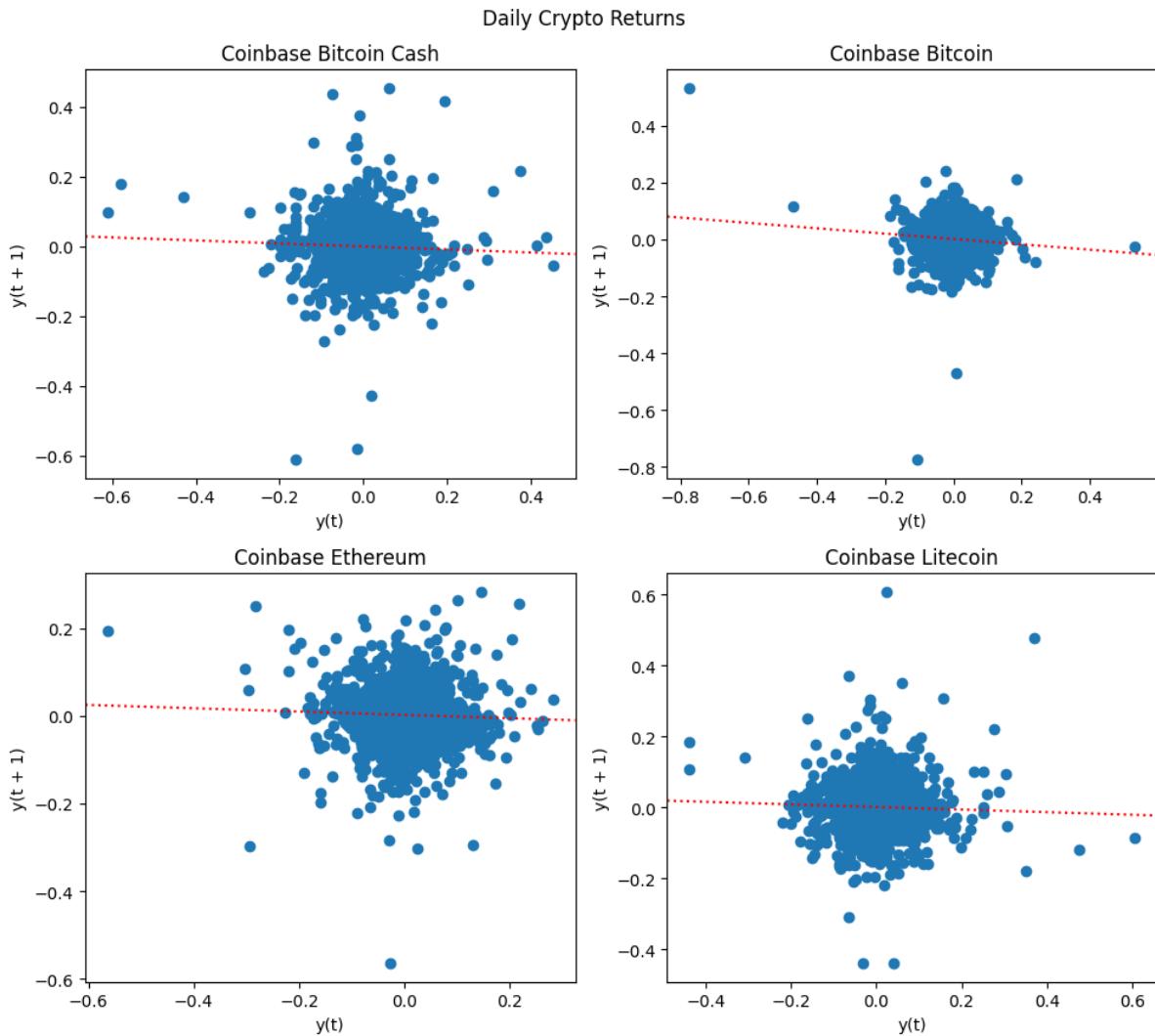
```
# QQ Plot for Gaussian assumption
from statsmodels.graphics.gofplots import ProbPlot
fig, axes = plt.subplots(2, 2, figsize=(10, 9))
for label, ax in zip(cryptos, axes.flatten()):
    pp = ProbPlot(cryptos[label].dropna(), fit=True)
    pp.qqplot(ax=ax, color='C0', alpha=.5)
    sm.qqline(ax=ax, fmt='r--', line='45', lw=1)
    ax.set_title(f"\"{titles[label]}\"")
plt.suptitle("Daily Crypto Returns")
plt.tight_layout()
```

```
/home/terence/env3.11/lib/python3.11/site-packages/statsmodels/graphics/gofplots.
  ↵py:1045: 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)
```



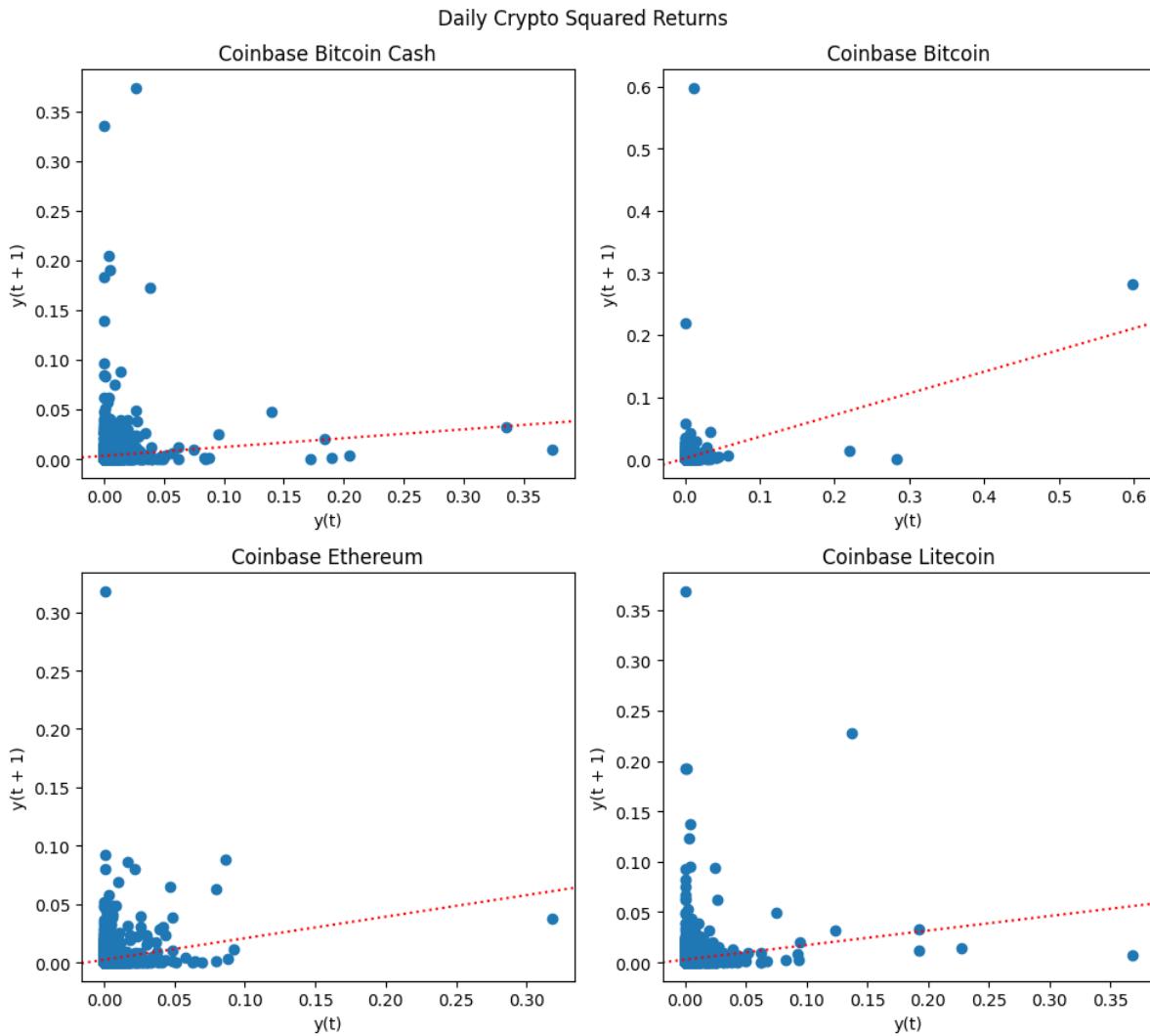
Lagplot of daily returns

```
# 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()
```



Lagplot 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()
```



15.1.5 Historical Simulation method

The simplest non-parametric approach to estimate VaR is by means of historical simulation (HS). The HS approach estimates VaR by ordering the loss observations and reading off the required VaR quantile. Expected shortfall is computed by averaging the observations smaller than that quantile. Since non-parametric approaches do not depend on assumptions about the model distribution, they can accommodate fat tails, skewness, and any other non-normal features that can cause problems for parametric approaches. However, non-parametric approaches are affected by how representative of data sample is of the period to be measured.

```
# 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)
```

(continues on next page)

(continued from previous page)

```

kurt = stats.kurtosis(X)
jb = stats.jarque_bera(X)[0]
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 | 2302 | 3388 | 2886 | 2796 |
| value_at_risk | 0.0844 | 0.0606 | 0.0764 | 0.0836 |
| expected_shortfall | 0.0082 | 0.0031 | 0.0054 | 0.0064 |
| volatility | 0.0597 | 0.0407 | 0.052 | 0.0557 |
| skewness | -0.2437 | -1.8572 | -0.4497 | 0.7674 |
| excess_kurtosis | 12.8465 | 51.3866 | 5.8203 | 10.2764 |
| jb_statistic | 24108.4383 | 419504.1796 | 9452.3917 | 20809.0086 |
| jb_pvalue | 0.0 | 0.0 | 0.0 | 0.0 |

15.1.6 Stressed VaR method

Volatilities increase during stressed market conditions, and correlations also generally increase. It is sometimes stated that “in stressed markets all correlations go to one.” In periods of heightened volatility such as the Great Financial Crisis of 2008, correlations can be quite different from those in normal market conditions. This is sometimes referred to as a correlation break-down. When concerned with estimating what will happen in extreme market conditions, VaR or ES should be estimated based what correlations will be in such conditions rather than in normal market conditions.

Stress testing is designed to identify vulnerabilities, particularly those involving high volatility. **Stressed VaR** is computed based on a “stressed” historical period in the markets, as opposed to simply the most recent number of years, to get an idea of possible losses given worse market conditions. For example, the period from Nov 2021 through November 21, 2022 has been dubbed the “crypto winter”, culminating with collapse of the FTX crypto exchange.

```

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 |

(continues on next page)

(continued from previous page)

| | | | | |
|-----------------|----------|----------|----------|---------|
| 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 |

15.1.7 Bootstrap method

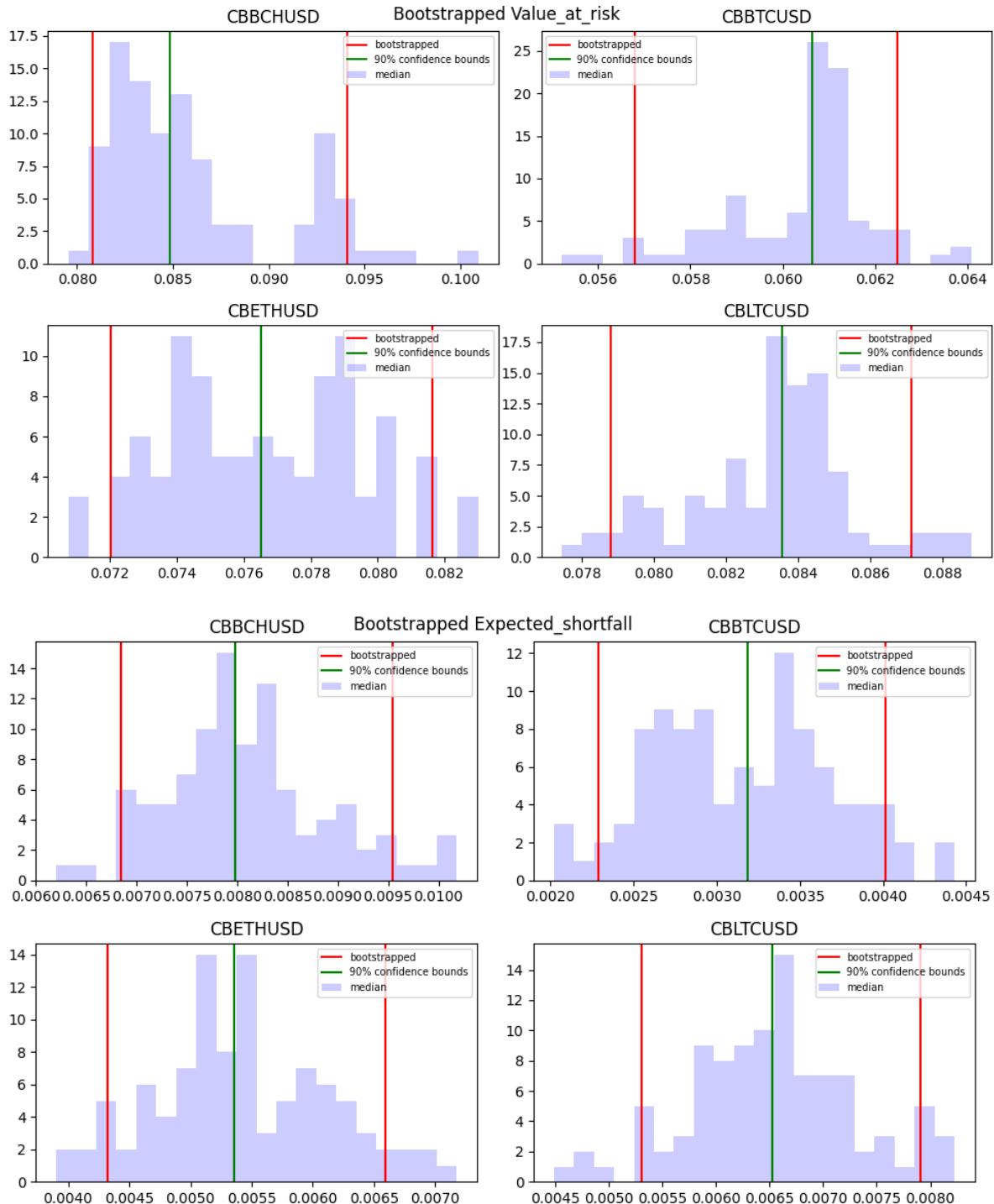
One simple but powerful improvement over basic HS is to estimate VaR and ES from bootstrapped data. The bootstrap procedure involves resampling from the existing data set with replacement. Each new ‘resampled’ sample gives us a new VaR estimate, and their mean can be taken to be the best estimate, with standard error and confidence interval computed from the distribution of resample-based estimates. We can also use the bootstrap to estimate ES’s in much the same way.

Perhaps the main limitation of the bootstrap is that standard bootstrap procedures presuppose that observations are independent over time. There are various way to modify bootstraps to allow for such dependence, such as the block approach: Divide sample data into non-overlapping blocks of equal length, and select a block at random.

```
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']) \
    .rename_axis(index='Value at Risk')
```

| | | | | |
|---------------|----------|----------|----------|----------|
| | CBBCHUSD | CBBTCUSD | CBETHUSD | CBLTCUSD |
| Value at Risk | | | | |

(continues on next page)

(continued from previous page)

| | | | | |
|--------|----------|----------|----------|----------|
| lower | 0.080831 | 0.056794 | 0.072047 | 0.078821 |
| median | 0.084858 | 0.060644 | 0.076511 | 0.083562 |
| upper | 0.094109 | 0.062483 | 0.081622 | 0.087124 |

```
# display confidence intervals of VaR
DataFrame(intervals['expected_shortfall'], index=['lower', 'median', 'upper'])\
    .rename_axis(index='Expected Shortfall')
```

| | CBBCHUSD | CCBTCUSD | CBETHUSD | CBLTCUSD |
|--------------------|----------|----------|----------|----------|
| Expected Shortfall | | | | |
| lower | 0.006844 | 0.002288 | 0.004324 | 0.005304 |
| median | 0.007976 | 0.003185 | 0.005357 | 0.006525 |
| upper | 0.009545 | 0.004016 | 0.006594 | 0.007911 |

15.2 Conditional volatility models

A return distribution may have parameters that vary through time. For ample, a simple mixture model of two normal distributions with different volatilities exhibits more peakedness and fatter tails than the normal. It is important to distinguish between a model where returns are unconditionally normal and a model where they are conditionally normal. In the latter, the return distribution is normal each day, while the standard deviation of the return varies throughout time (in other words, volatility is stochastic). During some periods it is high; during other periods it is low. This leads to an unconditional distribution with fat tails.

15.2.1 EWMA model

Current volatility can be estimated as an equal-weighted average of recent squared returns. This makes a simplifying assumption that the expected return is 0, which is reasonable over short periods such as daily where the standard deviation of the return is much more important than the mean. One problem with using an equal average is that a large (squared) return entering or dropping out of the sample is that the volatility will experience a large one-day increase or drop.

One way of overcoming these problems is to use exponential smoothing, which is also referred to as the exponentially weighted moving average (EWMA). In EWMA, the weight applied to the squared return from k days ago is λ multiplied by the weight applied to the squared return $k - 1$ days ago (where λ is a positive constant that is less than one). In the 1990s, the JP Morgan RiskMetrics group used this approach and found that setting $\lambda = 0.94$ proved a good choice for estimating volatilities of a wide range of market variables. The EWMA estimate can be conveniently expressed as a recursive equation:

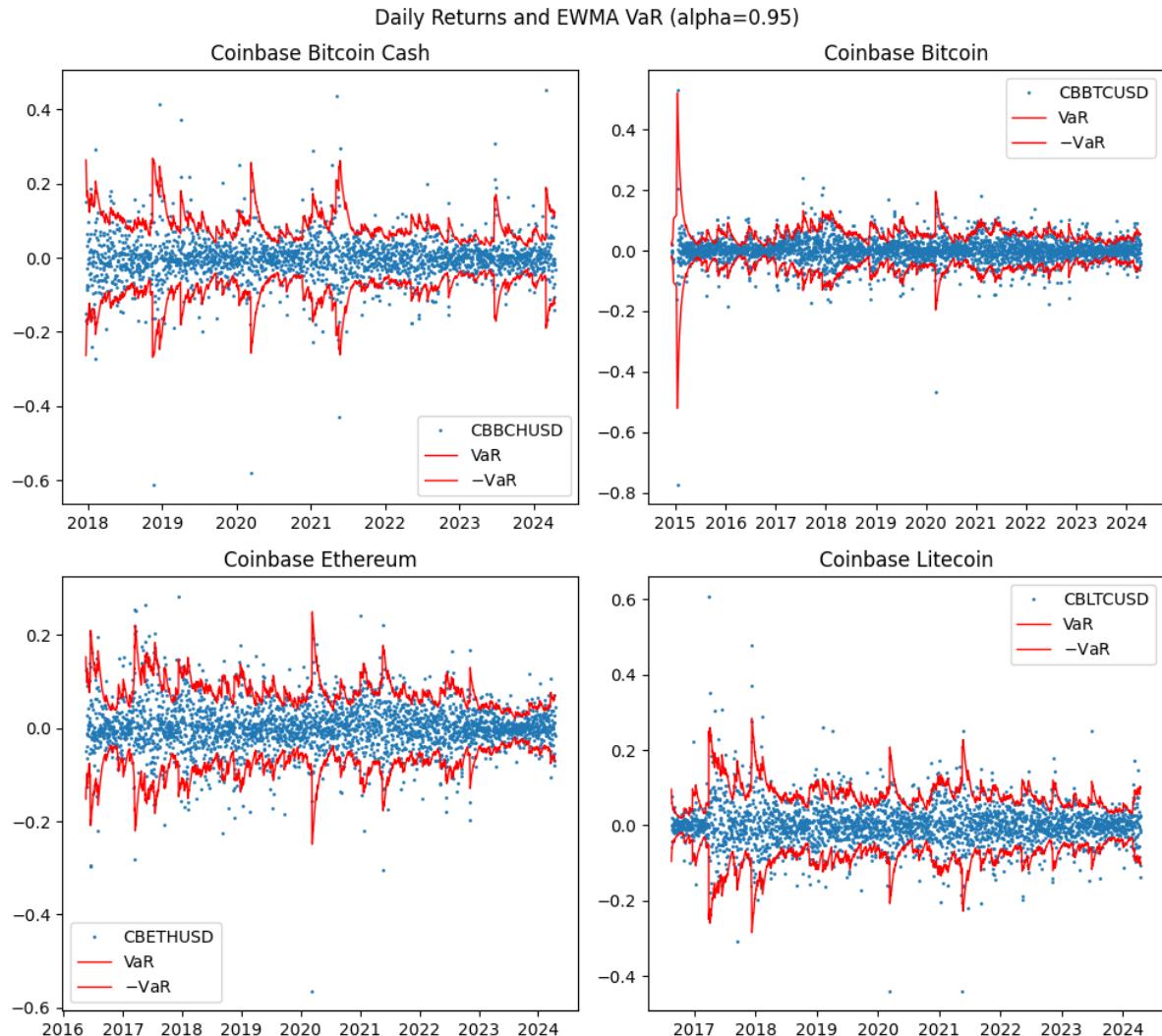
$$\sigma_t^2 = (1 - \lambda)r_{t-1}^2 + \lambda\sigma_{t-1}^2$$

This formula provides a very simple way of implementing EWMA. The new estimate of the variance rate on day n is a weighted average of the estimate of the variance rate made for the previous day ($n - 1$), and the most recent observation of the squared return on day ($n - 1$).

```
# 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')],
                  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 | CCBTCUSD | CBETHUSD | CBLTCUSD |
|--------------------|-----------|-----------|-----------|-----------|
| N | 2301.0000 | 3387.0000 | 2885.0000 | 2795.0000 |
| value_at_risk | 1.5056 | 1.4689 | 1.5831 | 1.5682 |
| expected_shortfall | 0.1249 | 0.0673 | 0.0778 | 0.0922 |
| volatility | 0.9260 | 0.9220 | 0.9417 | 0.9416 |
| skewness | 0.1289 | -0.0729 | -0.0307 | 0.0187 |
| excess_kurtosis | -1.3967 | -1.4662 | -2.1265 | -1.7600 |
| jb_statistic | 252.8316 | 334.9892 | 92.1766 | 179.2390 |
| jb_pvalue | 0.0000 | 0.0000 | 0.0000 | 0.0000 |

15.2.2 GARCH model

The generalized autoregressive conditional heteroskedasticity or **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 rate $\hat{\sigma}$. The updated formula for the variance rate is:

$$\sigma_t^2 = \alpha r_{t-1}^2 + \beta \sigma_{t-1}^2 + \gamma \hat{\sigma}$$

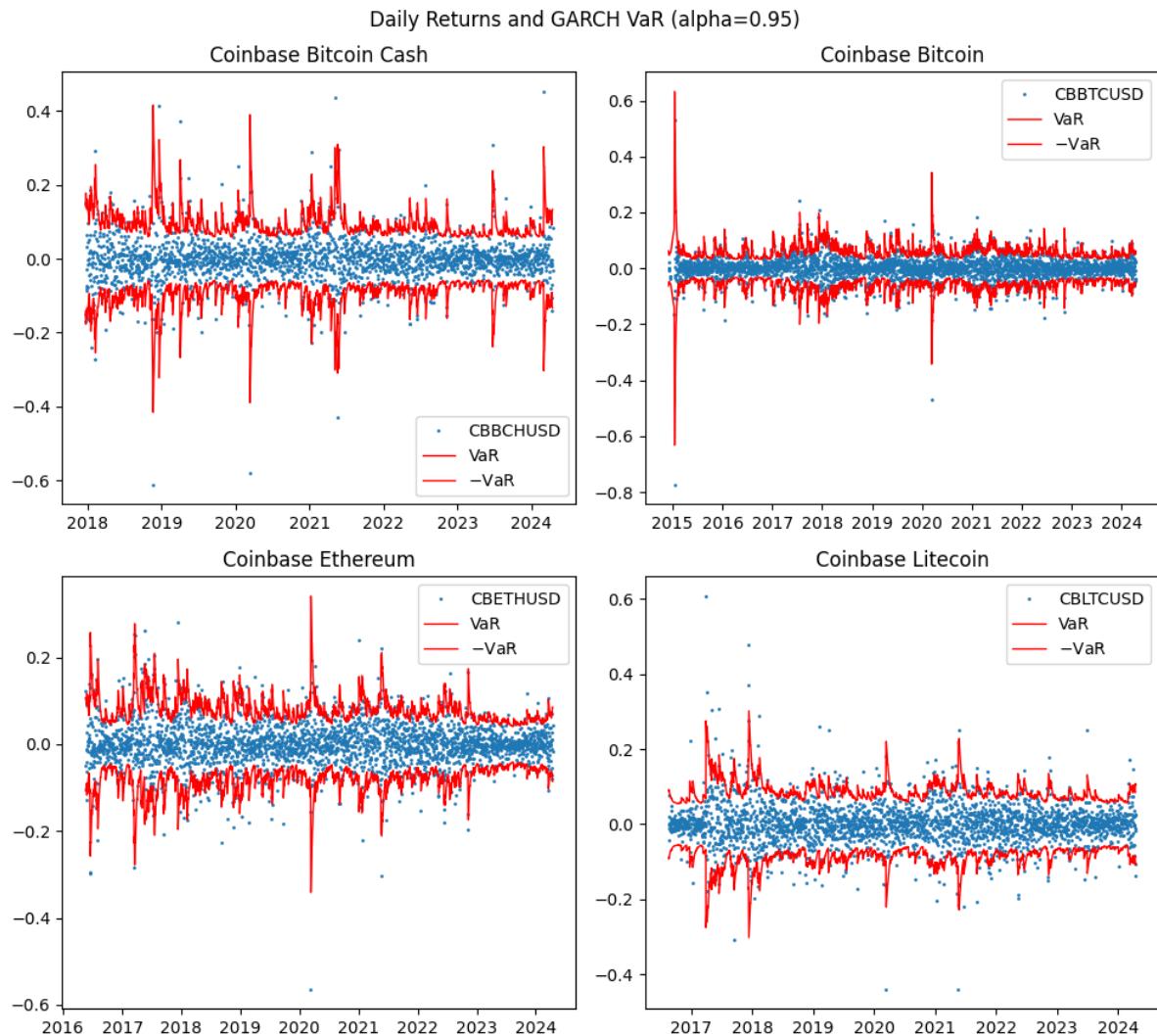
The weights must be sum to one, hence $\alpha + \beta \leq 1$ and $\gamma = 1 - \alpha - \beta$.

GARCH (1,1), compared to EWMA, incorporate mean-reversion where the $\hat{\sigma}$ term provides a “pull” toward the long-run average mean. And just as with ARMA specifications of time series models, GARCH(p,q) can incorporate additional lags of q squared returns and p previous variance rate estimates to generate better fitting models.

```
# Estimate GARCH(1, 1) by calling R's rugarch library
import rpy2.robjects as ro
from rpy2.robjects.packages import importr
from finds.utils import PyR
def rugarch(X: Series, savefig: str = '', verbose=VERBOSE) -> Series:
    """GARCH(1,1) wrapper around 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))
for label, ax in zip(cryptos, axes.flatten()):
    Z = pd.concat([cryptos[label],
                   parametric_risk(garch[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 GARCH VaR (alpha={alpha})")
plt.tight_layout()
```



```
# Properties of GARCH normalized returns
garch_hist = dict()
```

(continues on next page)

(continued from previous page)

```

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 | 2301.0000 | 3387.0000 | 2885.0000 | 2795.0000 |
| value_at_risk | 1.3172 | 1.2899 | 1.4004 | 1.4004 |
| expected_shortfall | 0.1053 | 0.0467 | 0.0755 | 0.0844 |
| volatility | 0.7728 | 0.7654 | 0.8372 | 0.8437 |
| skewness | 0.0286 | -0.0493 | -0.0064 | 0.0512 |
| excess_kurtosis | -2.6270 | -2.8977 | -2.8001 | -1.8373 |
| jb_statistic | 13.6517 | 2.8470 | 4.8221 | 158.6495 |
| jb_pvalue | 0.0011 | 0.2409 | 0.0897 | 0.0000 |

15.3 Backtesting VaR

The simplest method to verify the accuracy of the model is to record the failure rate, which gives the proportion of times VaR is exceeded in a given sample. The number of violations or exceptions should follow a binomial distribution with probability of failure $p = 1 - \alpha$, 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.

15.3.1 Kupiec Likelihood Ratio test

Kupiec (1995) develops approximate 95 percent confidence regions for such a test, based on by the (two-sided) tail points of the log-likelihood ratio:

$$LR = -2[(N - S) \ln(1 - p) + S \ln(p)] + 2[(N - S) \ln(1 - S/N) + S \ln(S/N)]$$

```

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

```

(continues on next page)

(continued from previous page)

```
>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}'})
```

| Kupiec LR Test: CBBCHUSD CBBTCUSD CBETHUSD CBLTCUSD | | | | |
|---|--------|--------|--------|--------|
| EWMA(0.94) | | | | |
| lr | 1.6293 | 3.2575 | 1.3298 | 2.7826 |
| violations | 102 | 147 | 131 | 121 |
| N | 2302 | 3388 | 2886 | 2796 |
| pvalue | 0.2018 | 0.0711 | 0.2488 | 0.0953 |

```
# 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}'})
```

| Kupiec LR Test: CBBCHUSD CBBTCUSD CBETHUSD CBLTCUSD | | | | |
|---|--------|--------|--------|---------|
| GARCH(1,1) | | | | |
| lr | 5.2248 | 5.302 | 3.8198 | 10.5494 |
| violations | 92 | 141 | 122 | 104 |
| N | 2302 | 3388 | 2886 | 2796 |
| pvalue | 0.0223 | 0.0213 | 0.0507 | 0.0012 |

15.3.2 Conditional coverage tests

This framework focuses on unconditional coverage because it ignores time variation in the data. In theory, the exceptions should be evenly spread over time. The observations could cluster or “bunch” closely in time, which also may invalidate the model. The LR statistic can be extended to specify that the deviations must be serially independent.

```
# TODO: conditional likelihood test
```

COVARIANCE MATRIX

Alone, we can do so little; together, we can do so much - Helen Keller

Concepts

- Portfolio Risk Analysis
- Covariance Matrix Estimation

References

- FRM Part II Exam Book Investment Management Ch. 5
- Chen et al., “Shrinkage Algorithms for MMSE Covariance Estimation” IEEE Trans. # on Sign. Proc., Volume 58, Issue 10, October 2010.
- Olivier Ledoit and Michael Wolf, “Honey, I Shrunk the Sample Covariance Matrix”, July 2003, The Journal of Portfolio Management 30(4)
- Florin Spinu, “An algorithm for computing risk parity weights,” SSRN, 2013.

```
import numpy as np
from sklearn.decomposition import PCA
from pandas import DataFrame
import cvxpy as cp
from tqdm import tqdm
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
from sklearn.covariance import LedoitWolf, OAS, EmpiricalCovariance
from sklearn import cluster
from finds.utils import ColorMap
from finds.readers import FFReader
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

# %matplotlib qt
VERBOSE = 0
```

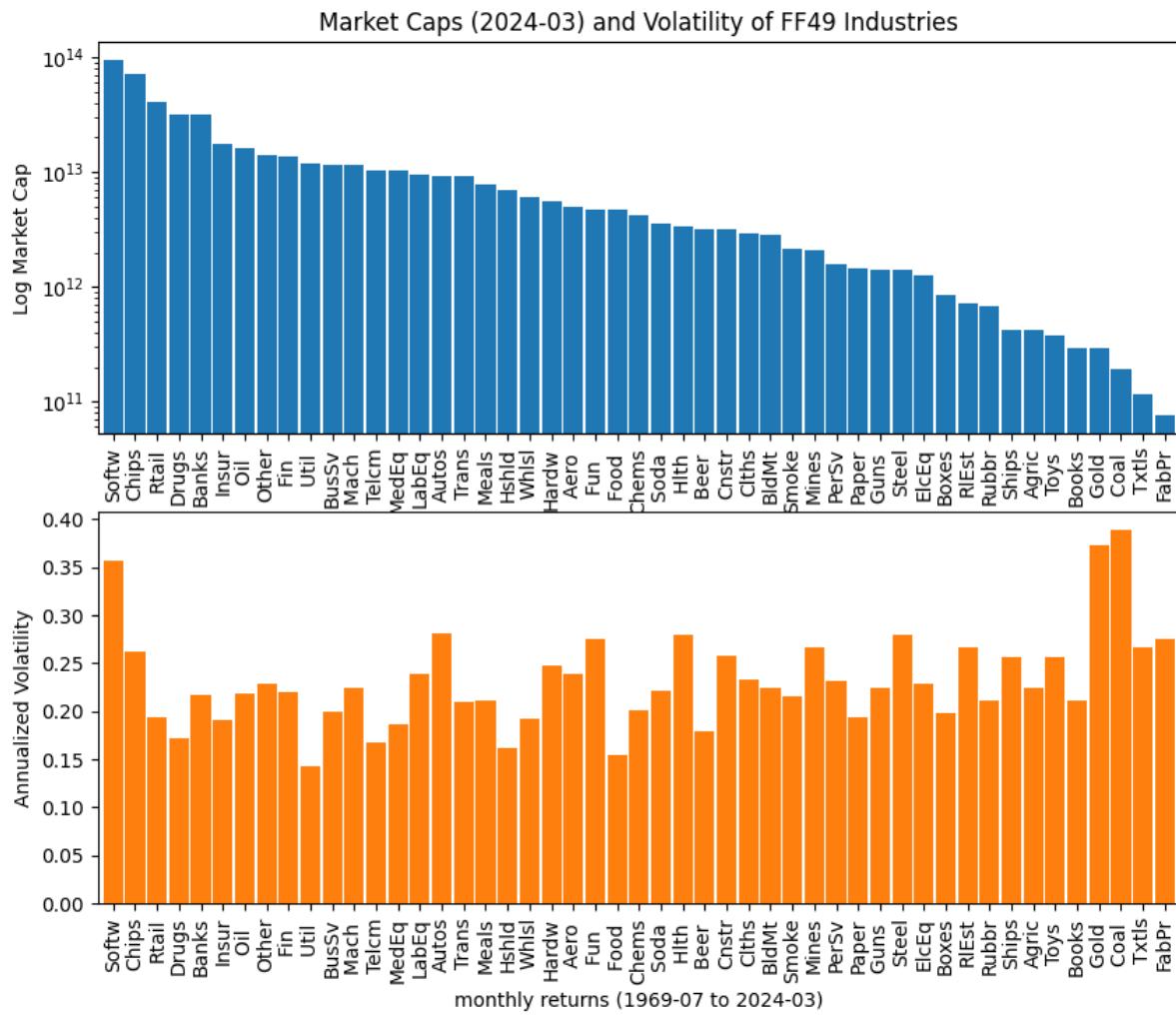
16.1 Portfolio risk analysis

```
# Retrieve industry returns from Ken French Data Library website
symbol = '49_Industry_Portfolios'
ff = FFReader(symbol)
keep = ff[0].index[(ff[0] > -99).all(axis=1)]
rets = (ff[0] / 100).reindex(keep)
caps = (ff[4] * ff[5] * 10e6).reindex(keep)    # number of firms x avg market cap
weights = caps.iloc[-1] / caps.iloc[-1].sum()
```

```
X = (rets - rets.mean(axis=0))      # demean by industry
sigma = 12 * X.T @ X / len(X)      # annualized covariance matrix
Y = caps.iloc[-1] \
    .rename('cap') \
    .to_frame() \
    .join(DataFrame({'vol': np.sqrt(np.diag(sigma))}, index=X.columns)) \
    .sort_values('cap', ascending=False)
```

```
# Plot annualized volatility and market caps of industries
fig, ax = plt.subplots(nrows=2, figsize=(10, 8))
Y['cap'].plot.bar(ax=ax[0], color="C0", width=0.9)
ax[0].set_yscale('log')
ax[0].set_title(f"Market Caps ({X.index[-1]}) and Volatility of FF49 Industries")
ax[0].set_ylabel(f"Log Market Cap")
Y['vol'].plot.bar(ax=ax[1], color="C1", width=0.9)
ax[1].set_ylabel(f"Annualized Volatility")
ax[1].set_xlabel(f"monthly returns ({X.index[0]} to {X.index[-1]})")
```

```
Text(0.5, 0, 'monthly returns (1969-07 to 2024-03)')
```



16.1.1 Risk budgetting

- Portfolio Risk - volatility of (market) portfolio is $\sigma_P = W^T \Sigma W$, where W is the vector of securities' weights in the portfolio and Σ is the covariance matrix of their returns
- Covariance: $\sigma_{iP} = \rho_{iP} \cdot \sigma_i \cdot \sigma_P$, where ρ_{iP} is the correlation of the returns of security i w.r.t portfolio P
- Beta - the systematic risk risk of security vs portfolio, or regression slope of the security's returns on portfolio returns: $\beta_i = \frac{\sigma_{iP}}{\sigma_P^2}$
- Marginal Contribution to Risk - sensitivity of portfolio volatility to small change in weight: $\frac{\partial \sigma_P}{\partial w_i} = \frac{\sigma_{iP}}{\sigma_P}$
- Percent Contribution to Risk - what fraction of portfolio risk would change approximately if this security was deleted from the portfolio: $\beta_i \cdot w_i$. This risk decomposition that sums to 1.
- Contribution to risk - this decomposition of portfolio risk sums to portfolio volatility. It can also be expressed as asset weight times asset volatility times correlation with portfolio: $\beta_i \cdot w_i \cdot \sigma_P = w_i \cdot \sigma_i \cdot \rho_{iP}$.

```
# Helper to compute portfolio risk budget
def risk_budget(w, sigma, labels):
```

(continues on next page)

(continued from previous page)

```
"""Compute portfolio risk analytics"""
sigma_ = np.array(sigma) * 100 * 100    # express as percent returns
w_ = np.array(w)

# Portfolio volatility (percent)
vol = np.sqrt(w_.T @ sigma_ @ w_)

# Covariance of each security wrt market portfolio
cov = sigma_ @ w_

# Beta of each security wrt market portfolio
beta = cov / (w_.T @ sigma_ @ w_)

# Marginal Contribution to Risk of each security
marginal = beta * vol

# Percent Contribution to Risk
percent = beta * w_ * 100

# Contribution to Risk
contrib = marginal * w_

return DataFrame({'weight': list(w),
                  'Beta': list(beta),
                  'MCR(%)': list(marginal),
                  'PCR(%)': list(percent),
                  'CR': list(contrib)},
                  index=labels)
```

```
vol = np.sqrt(weights.T @ sigma @ weights) # market portfolio risk
print(f"Market Portfolio Risk Budget (vol={vol*100:.2f}%)")
risk_budget(weights, sigma, weights.index).sort_values('CR', ascending=False)
```

Market Portfolio Risk Budget (vol=19.35%)

| | weight | Beta | MCR (%) | PCR (%) | CR |
|-------|----------|----------|-----------|-----------|----------|
| Softw | 0.192584 | 1.574845 | 30.475899 | 30.328962 | 5.869164 |
| Chips | 0.142872 | 1.179219 | 22.819870 | 16.847733 | 3.260319 |
| Rtail | 0.081477 | 0.827907 | 16.021394 | 6.745528 | 1.305373 |
| Banks | 0.064025 | 0.845579 | 16.363371 | 5.413810 | 1.047663 |
| Drugs | 0.064479 | 0.589820 | 11.414007 | 3.803093 | 0.735962 |
| Fin | 0.027743 | 0.969658 | 18.764520 | 2.690145 | 0.520588 |
| Insur | 0.035976 | 0.728126 | 14.090454 | 2.619527 | 0.506922 |
| Other | 0.028330 | 0.910110 | 17.612159 | 2.578348 | 0.498954 |
| Mach | 0.023061 | 0.972623 | 18.821886 | 2.242987 | 0.434056 |
| BusSv | 0.023067 | 0.955382 | 18.488250 | 2.203781 | 0.426469 |
| LabEq | 0.019119 | 1.052405 | 20.365795 | 2.012080 | 0.389371 |
| Oil | 0.032719 | 0.596297 | 11.539353 | 1.951038 | 0.377559 |
| Autos | 0.018619 | 0.986945 | 19.099042 | 1.837576 | 0.355602 |
| Trans | 0.018589 | 0.883918 | 17.105289 | 1.643083 | 0.317964 |
| MedEq | 0.020948 | 0.724425 | 14.018833 | 1.517501 | 0.293662 |
| Meals | 0.015693 | 0.869115 | 16.818833 | 1.363934 | 0.263944 |
| Telcm | 0.020987 | 0.594172 | 11.498234 | 1.247000 | 0.241315 |
| Hardw | 0.011335 | 0.986408 | 19.088649 | 1.118097 | 0.216371 |

(continues on next page)

(continued from previous page)

| | | | | | |
|-------|----------|----------|-----------|----------|----------|
| Fun | 0.009615 | 1.139940 | 22.059751 | 1.096003 | 0.212095 |
| Whlsl | 0.012280 | 0.871606 | 16.867039 | 1.070371 | 0.207135 |
| Aero | 0.009961 | 0.932730 | 18.049895 | 0.929067 | 0.179790 |
| Util | 0.024195 | 0.375009 | 7.257048 | 0.907339 | 0.175585 |
| Hshld | 0.014130 | 0.602340 | 11.656284 | 0.851110 | 0.164704 |
| Chems | 0.008469 | 0.810134 | 15.677445 | 0.686088 | 0.132770 |
| Cnstr | 0.006447 | 1.048531 | 20.290834 | 0.676030 | 0.130823 |
| Hlth | 0.006709 | 0.976104 | 18.889260 | 0.654847 | 0.126724 |
| Clths | 0.005866 | 0.942774 | 18.244268 | 0.553030 | 0.107021 |
| BldMt | 0.005701 | 0.956470 | 18.509299 | 0.545237 | 0.105512 |
| Food | 0.009570 | 0.504737 | 9.767512 | 0.483043 | 0.093477 |
| Soda | 0.007138 | 0.616253 | 11.925540 | 0.439867 | 0.085122 |
| Beer | 0.006491 | 0.575478 | 11.136461 | 0.373514 | 0.072281 |
| Mines | 0.004227 | 0.874984 | 16.932405 | 0.369892 | 0.071580 |
| Steel | 0.002806 | 1.047448 | 20.269883 | 0.293932 | 0.056881 |
| PerSv | 0.003182 | 0.921152 | 17.825841 | 0.293143 | 0.056728 |
| ElcEq | 0.002571 | 0.980634 | 18.976924 | 0.252161 | 0.048797 |
| Paper | 0.002955 | 0.740217 | 14.324451 | 0.218709 | 0.042324 |
| Smoke | 0.004310 | 0.482936 | 9.345629 | 0.208167 | 0.040284 |
| Guns | 0.002814 | 0.648680 | 12.553038 | 0.182568 | 0.035330 |
| RlEst | 0.001445 | 1.037531 | 20.077966 | 0.149957 | 0.029019 |
| Boxes | 0.001692 | 0.731979 | 14.165026 | 0.123842 | 0.023966 |
| Rubbr | 0.001372 | 0.874000 | 16.913362 | 0.119923 | 0.023207 |
| Toys | 0.000754 | 0.995149 | 19.257797 | 0.074994 | 0.014513 |
| Ships | 0.000858 | 0.863957 | 16.719015 | 0.074134 | 0.014346 |
| Agric | 0.000856 | 0.708072 | 13.702388 | 0.060618 | 0.011731 |
| Books | 0.000598 | 0.875354 | 16.939570 | 0.052356 | 0.010132 |
| Coal | 0.000392 | 0.881060 | 17.049990 | 0.034549 | 0.006686 |
| Gold | 0.000586 | 0.435823 | 8.433900 | 0.025519 | 0.004938 |
| Txtls | 0.000235 | 0.941579 | 18.221139 | 0.022085 | 0.004274 |
| FabPr | 0.000152 | 0.901686 | 17.449141 | 0.013682 | 0.002648 |

16.1.2 Risk parity portfolios

To construct a simple risk-parity portfolio (RPP): first equalize the risk contributions of the asset classes and then apply leverage to achieve a target volatility

A desired risk budget can be approximated as a convex optimization problem, see Spinu (2013): $\min_w \frac{1}{2} w^T \Sigma w - \sum_i b_i \log w_i$ subject to $0 \leq w_i \leq 1/N$ and $\sum_i w_i = 1$, where b_i is the desired RPP risk budget

```
# Set up variables and constraints
N = len(sigma)
b = np.ones(N) / N # risk parity
W = cp.Variable(N) # portfolio weights to solve for
constraints = [W >= 0] # non-negative weights constraint
```

```
# Solve objective
obj = 0.5 * cp.quad_form(W, sigma) - cp.sum(cp.multiply(b, cp.log(W)))
prob = cp.Problem(cp.Minimize(obj), constraints)
prob.solve()
```

2.6150587857892416

```
# normalize solution weights
rpp = (W/cp.sum(W)).value
```

```
vol = np.sqrt(rpp.T @ sigma @ rpp)
print(f"Risk Parity Portfolio (vol={vol*100:.2f}%)")
risk_budget(rpp, sigma, weights.index).sort_values('weight', ascending=False)
```

Risk Parity Portfolio (vol=16.49%)

| | weight | Beta | MCR (%) | PCR (%) | CR |
|-------|----------|----------|-----------|----------|----------|
| Util | 0.038752 | 0.526629 | 8.682950 | 2.040816 | 0.336486 |
| Food | 0.030451 | 0.670192 | 11.049988 | 2.040816 | 0.336486 |
| Smoke | 0.029572 | 0.690115 | 11.378474 | 2.040816 | 0.336486 |
| Telcm | 0.028807 | 0.708446 | 11.680724 | 2.040816 | 0.336486 |
| Drugs | 0.028718 | 0.710630 | 11.716722 | 2.040816 | 0.336486 |
| Gold | 0.028147 | 0.725044 | 11.954388 | 2.040816 | 0.336486 |
| Beer | 0.027761 | 0.735127 | 12.120636 | 2.040817 | 0.336486 |
| Hshld | 0.026958 | 0.757049 | 12.482075 | 2.040816 | 0.336486 |
| Soda | 0.024502 | 0.832909 | 13.732845 | 2.040816 | 0.336486 |
| Oil | 0.024255 | 0.841397 | 13.872790 | 2.040816 | 0.336486 |
| Guns | 0.023502 | 0.868368 | 14.317474 | 2.040816 | 0.336486 |
| MedEq | 0.023254 | 0.877618 | 14.469991 | 2.040816 | 0.336486 |
| Agric | 0.022965 | 0.888652 | 14.651916 | 2.040816 | 0.336486 |
| Insur | 0.021931 | 0.930560 | 15.342893 | 2.040816 | 0.336486 |
| Boxes | 0.021501 | 0.949152 | 15.649437 | 2.040816 | 0.336486 |
| Rtail | 0.021334 | 0.956624 | 15.772626 | 2.040816 | 0.336486 |
| Paper | 0.020921 | 0.975495 | 16.083771 | 2.040816 | 0.336486 |
| Hardw | 0.019528 | 1.045064 | 17.230802 | 2.040816 | 0.336486 |
| Banks | 0.019314 | 1.056656 | 17.421937 | 2.040816 | 0.336486 |
| Whlsl | 0.019224 | 1.061622 | 17.503814 | 2.040816 | 0.336486 |
| Chems | 0.019213 | 1.062229 | 17.513827 | 2.040816 | 0.336486 |
| Meals | 0.019213 | 1.062229 | 17.513831 | 2.040816 | 0.336486 |
| Books | 0.018773 | 1.087080 | 17.923556 | 2.040816 | 0.336486 |
| Other | 0.018731 | 1.089539 | 17.964098 | 2.040816 | 0.336486 |
| Rubbr | 0.018719 | 1.090222 | 17.975366 | 2.040816 | 0.336486 |
| Trans | 0.018624 | 1.095792 | 18.067196 | 2.040816 | 0.336486 |
| PerSv | 0.018267 | 1.117226 | 18.420601 | 2.040816 | 0.336486 |
| BusSv | 0.018203 | 1.121150 | 18.485298 | 2.040816 | 0.336486 |
| Fin | 0.017891 | 1.140717 | 18.807923 | 2.040817 | 0.336486 |
| Clths | 0.017758 | 1.149216 | 18.948046 | 2.040816 | 0.336486 |
| Ships | 0.017606 | 1.159151 | 19.111846 | 2.040816 | 0.336486 |
| FabPr | 0.017405 | 1.172564 | 19.333011 | 2.040816 | 0.336486 |
| Aero | 0.017372 | 1.174790 | 19.369715 | 2.040816 | 0.336486 |
| Mines | 0.017281 | 1.180988 | 19.471892 | 2.040816 | 0.336486 |
| ElcEq | 0.017153 | 1.189796 | 19.617125 | 2.040816 | 0.336486 |
| LabEq | 0.017010 | 1.199755 | 19.781322 | 2.040816 | 0.336486 |
| Mach | 0.016924 | 1.205842 | 19.881692 | 2.040816 | 0.336486 |
| Autos | 0.016799 | 1.214823 | 20.029764 | 2.040816 | 0.336486 |
| Toys | 0.016731 | 1.219815 | 20.112076 | 2.040816 | 0.336486 |
| BldMt | 0.016702 | 1.221907 | 20.146569 | 2.040816 | 0.336486 |
| Hlth | 0.016620 | 1.227922 | 20.245745 | 2.040816 | 0.336486 |
| Chips | 0.016573 | 1.231395 | 20.302992 | 2.040816 | 0.336486 |
| Coal | 0.016564 | 1.232074 | 20.314192 | 2.040816 | 0.336486 |
| Txtls | 0.016560 | 1.232393 | 20.319461 | 2.040816 | 0.336486 |
| RlEst | 0.015650 | 1.304027 | 21.500541 | 2.040817 | 0.336486 |

(continues on next page)

(continued from previous page)

| | | | | | |
|-------|----------|----------|-----------|----------|----------|
| Cnstr | 0.015634 | 1.305394 | 21.523079 | 2.040816 | 0.336486 |
| Steel | 0.015361 | 1.328586 | 21.905464 | 2.040816 | 0.336486 |
| Fun | 0.015278 | 1.335796 | 22.024346 | 2.040816 | 0.336486 |
| Softw | 0.013988 | 1.458990 | 24.055548 | 2.040816 | 0.336486 |

16.2 Covariance matrix estimation

16.2.1 Principal components

Each component can be interpreted as weights in a portfolio holdings the industries as assets. The projection of industry returns on a component simply computes the returns of the corresponding portfolio.

```
# Fit PCA
pca = PCA().fit(X)

# Retrieve components, and sign flip if necessary
loadings = (np.diag(pca.singular_values_) @ pca.components_).T # compute loadings by
#column
components = DataFrame.from_records(pca.components_.T, index=X.columns)
components *= np.sign(components.sum(axis=0))

# Compute projections, can be interpreted as portfolio returns
proj = pca.transform(X)
proj *= np.sign(np.sum(pca.components_, axis=1)) # flip signs if necessary

# Equal weighted market average return
avg = X.mean(axis=1)
```

The first component resembles a portfolio with long positions invested across all assets, and explains more than half the variance. The other components resemble long-short spread portfolios that explain incremental amounts of variance.

```
K = 20
print(f"Top {K} principal components")
DataFrame({
    'frac weights +ve': np.mean(components.iloc[:, :K].values >= 0, axis=0),
    'sum weights': np.sum(components.iloc[:, :K].values, axis=0),
    'sum sqr weights': np.sum((components.iloc[:, :K].values)**2, axis=0),
    'sum abs weights': np.sum(np.abs(components.iloc[:, :K].values), axis=0),
    'corr avg ret': [np.corrcoef(avg, proj[:, i])[0, 1] for i in range(K)],
    'cumulative expl ratio': np.cumsum(pca.explained_variance_ratio_[:K]),
    index=[f"PC{i+1}" for i in range(K)])
    .round(4)
```

Top 20 principal components

| | frac weights +ve | sum weights | sum sqr weights | sum abs weights |
|-----|------------------|-------------|-----------------|-----------------|
| PC1 | 1.0000 | 6.8347 | 1.0 | 6.8347 |
| PC2 | 0.2449 | 0.0869 | 1.0 | 4.1490 |
| PC3 | 0.5102 | 0.2746 | 1.0 | 3.5353 |
| PC4 | 0.6735 | 0.9467 | 1.0 | 5.1509 |
| PC5 | 0.4898 | 0.5331 | 1.0 | 5.2453 |
| PC6 | 0.5510 | 0.7318 | 1.0 | 5.5496 |

(continues on next page)

(continued from previous page)

| | | | | |
|------------------------------------|--------|--------|--------|--------|
| PC7 | 0.5714 | 0.0774 | 1.0 | 5.5749 |
| PC8 | 0.5102 | 0.0148 | 1.0 | 4.9275 |
| PC9 | 0.4898 | 0.1839 | 1.0 | 5.1437 |
| PC10 | 0.4694 | 0.0692 | 1.0 | 5.6381 |
| PC11 | 0.5510 | 0.2505 | 1.0 | 4.7172 |
| PC12 | 0.5306 | 0.1901 | 1.0 | 5.2332 |
| PC13 | 0.6735 | 0.1061 | 1.0 | 4.7941 |
| PC14 | 0.4490 | 0.0851 | 1.0 | 4.8345 |
| PC15 | 0.4286 | 0.1587 | 1.0 | 4.9447 |
| PC16 | 0.4286 | 0.0192 | 1.0 | 4.8334 |
| PC17 | 0.4694 | 0.0584 | 1.0 | 5.5921 |
| PC18 | 0.6122 | 0.1920 | 1.0 | 5.1250 |
| PC19 | 0.5306 | 0.1623 | 1.0 | 5.1083 |
| PC20 | 0.4898 | 0.1706 | 1.0 | 5.3034 |
| corr avg ret cumulative expl ratio | | | | |
| PC1 | 0.9988 | | 0.5604 | |
| PC2 | 0.0043 | | 0.6235 | |
| PC3 | 0.0107 | | 0.6636 | |
| PC4 | 0.0356 | | 0.7006 | |
| PC5 | 0.0178 | | 0.7299 | |
| PC6 | 0.0213 | | 0.7521 | |
| PC7 | 0.0020 | | 0.7701 | |
| PC8 | 0.0003 | | 0.7843 | |
| PC9 | 0.0043 | | 0.7984 | |
| PC10 | 0.0015 | | 0.8104 | |
| PC11 | 0.0052 | | 0.8218 | |
| PC12 | 0.0039 | | 0.8328 | |
| PC13 | 0.0021 | | 0.8430 | |
| PC14 | 0.0016 | | 0.8527 | |
| PC15 | 0.0030 | | 0.8621 | |
| PC16 | 0.0004 | | 0.8710 | |
| PC17 | 0.0010 | | 0.8794 | |
| PC18 | 0.0033 | | 0.8870 | |
| PC19 | 0.0027 | | 0.8941 | |
| PC20 | 0.0028 | | 0.9011 | |

Spectral Clustering

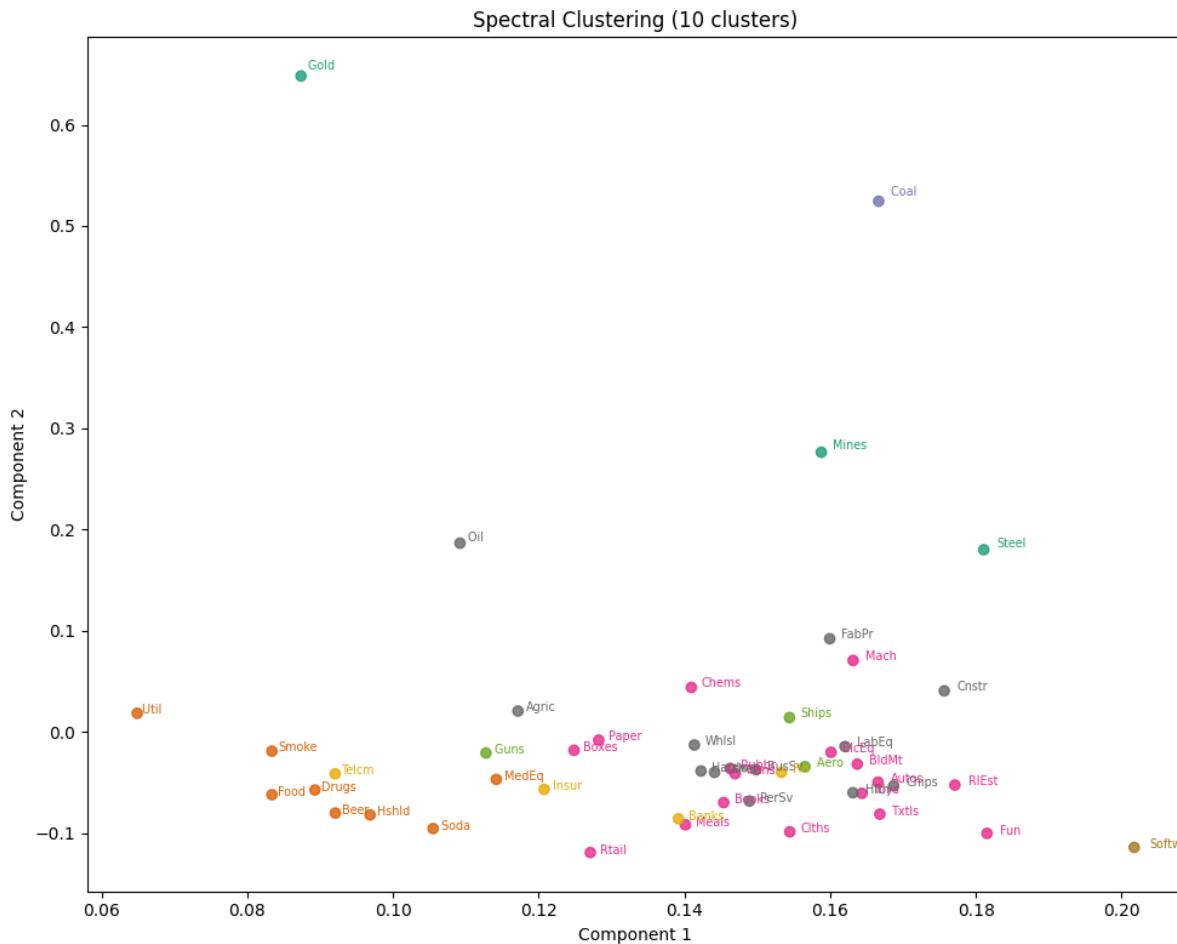
- Component 1 appears to load on the average market factor (so-called beta risk)
- Component 2 can be interpreted as loading on a commodities factor

```
# Spectral clustering
spectral = cluster.SpectralClustering(
    n_clusters=10,
    eigen_solver="arpack",
    #affinity="nearest_neighbors",
    random_state=42,
)
spectral.fit(X.T)    # number of features equals the number observations dates
cmap = ColorMap(spectral.n_clusters, colormap='Dark2')
fig, ax = plt.subplots(figsize=(10, 8))
ax.scatter(components.iloc[:, 0], components.iloc[:, 1],
           c=cmap[spectral.labels_], alpha=.8)
for t, c, xy in zip(components.index, spectral.labels_,
                     components.iloc[:, :2].values):
```

(continues on next page)

(continued from previous page)

```
    ax.annotate(text=t, xy=xy, xytext=xy * 1.01, color=cmap[c], fontsize='x-small')
ax.set_xlabel('Component 1')
ax.set_ylabel('Component 2')
ax.set_title(f"Spectral Clustering ({spectral.n_clusters} clusters}")
plt.tight_layout()
plt.show()
```



```
 DataFrame({'cluster': spectral.labels_}, index=components.index).sort_values('cluster  
↪')
```

| | cluster |
|-------|---------|
| Steel | 0 |
| Mines | 0 |
| Gold | 1 |
| Drugs | 2 |
| MedEq | 2 |
| Util | 2 |
| Hshld | 2 |
| Smoke | 2 |
| Beer | 2 |
| Soda | 2 |

(continues on next page)

(continued from previous page)

| | |
|-------|---|
| Food | 2 |
| Coal | 3 |
| Trans | 4 |
| ElcEq | 4 |
| Meals | 4 |
| Autos | 4 |
| Rtail | 4 |
| Mach | 4 |
| Paper | 4 |
| RlEst | 4 |
| Toys | 4 |
| BldMt | 4 |
| Txtls | 4 |
| Fun | 4 |
| Books | 4 |
| Clths | 4 |
| Rubbr | 4 |
| Chems | 4 |
| Boxes | 4 |
| Ships | 5 |
| Aero | 5 |
| Guns | 5 |
| Fin | 6 |
| Telcm | 6 |
| Insur | 6 |
| Banks | 6 |
| Softw | 7 |
| Whlsl | 8 |
| Agric | 8 |
| Bussv | 8 |
| PerSv | 8 |
| Oil | 8 |
| FabPr | 8 |
| Cnstr | 8 |
| Hlth | 8 |
| LabEq | 8 |
| Other | 8 |
| Chips | 9 |
| Hardw | 9 |

16.2.2 EWMA

The Exponentially-Weighted Moving Average (EWMA) method simply gives recent data, that may be more relevant for predicting risk, a heavier weight when computing the empirical covariance matrix. In 1996, JP Morgan Risk Metrics proposed a lambda parameter of 0.97, in other words, where weights placed on historical data decay at a monthly rate of 0.03.

The time it takes for the weight to decay by half is called the **half-life**.

```
# Compute half-life of risk metrics' lambda for monthly data
def halflife(decay, half=0.5):
    """Returns halflife (t) from its definition: 0.5 = (1-decay)^t"""
    return -np.log(1/half)/np.log(1 - decay)
```

```
risk_metrics_lambda = 0.97 # for monthly data
print('Half-life: ', halflife(decay=1-risk_metrics_lambda).round(1), 'months')
```

Half-life: 22.8 months

```
def ewma(X, decay):
    """Helper to compute EWMA covariance matrix estimate"""
    weights = (1 - decay)**np.arange(len(X))[:-1]
    return (weights.reshape((1, -1)) * X.T @ X / weights.sum())
```

16.2.3 Shrinkage methods

Covariance matrix estimates can be regularized using shrinkage. Ledoit and Wolf proposed a close formula to compute the asymptotically optimal shrinkage parameter (minimizing a MSE criterion). They shrink the covariance towards the identity matrix: $(1 - \beta)\Sigma + \beta \frac{tr(\Sigma)}{N} I_n$ where β is the shrinkage factor given by Ledoit and Wolf (1993)

Chen et al. proposed the Oracle Approximating Shrinkage (OAS) Estimator whose convergence is significantly better under the assumption that the data are Gaussian.

16.2.4 Global MVP volatility

The out-of-sample (OOS) realized volatility of the Global Minimum Variance (GMV) portfolio can serve as a practical test of the accuracy of covariance matrix estimation techniques. The GMV is constructed to minimize portfolio volatility, hence is very sensitive to small errors in the covariance matrix. It relies on efficient diversification across assets, and inaccurate covariance estimates may lead to higher realized volatility if the portfolio inadequately hedges its positions against risk.

Starting in Jan 2000, we update the covariance matrix estimate monthly, using the rolling past 30 years of data, and record the month-ahead return of the GMV portfolio. At the end of the test sample, we expect the best covariance matrix estimator to realize the minimum returns volatility.

```
# Helper method to compute Minimum Variance Portfolio and realized volatility
def gmv(cov, ret):
    """Compute minimum variance portfolio and return"""
    w = np.linalg.inv(cov) @ np.ones((cov.shape[1], 1))
    return float(np.array(ret) @ w/sum(w))
```

```
# Rolling monthly evaluation
decay = 0.03 # risk metrics monthly decay rate for EWMA
n_components = 10 # for PCA
split = '2000-01' # start rolling OOS prediction tests from this date
```

```
r = {} # collect realized returns
for date in tqdm(rets.index[rets.index >= split]):
    X_train = rets.iloc[rets.index < date, :][-30*12:] # 30 years of training data
    X_test = rets.iloc[rets.index == date, :] # predict one month ahead returns
    r[date] = {}

    # Empirical covariance
    cov = EmpiricalCovariance().fit(X_train).covariance_
```

(continues on next page)

(continued from previous page)

```
r[date]['Covariance'] = gmv(cov, X_test)
r[date]['Diagonal'] = gmv(np.diag(np.diag(cov)), X_test)
r[date]['EWMA'] = gmv(ewma(X_train, decay=decay), X_test)
r[date][f"PCA-{n_components}"] = gmv(PCA(n_components).fit(X_train).get_
covariance(), X_test)
r[date]['Identity'] = gmv(np.identity(len(cov)), X_test)
r[date]['LW'] = gmv(LedoitWolf().fit(X_train).covariance_, X_test)
r[date]['OAS'] = gmv(OAS().fit(X_train).covariance_, X_test)
```

0% | 0 / 291 [00:00<?, ?it/s]

100% |██████████| 291 / 291 [01:25<00:00, 3.41it/s]

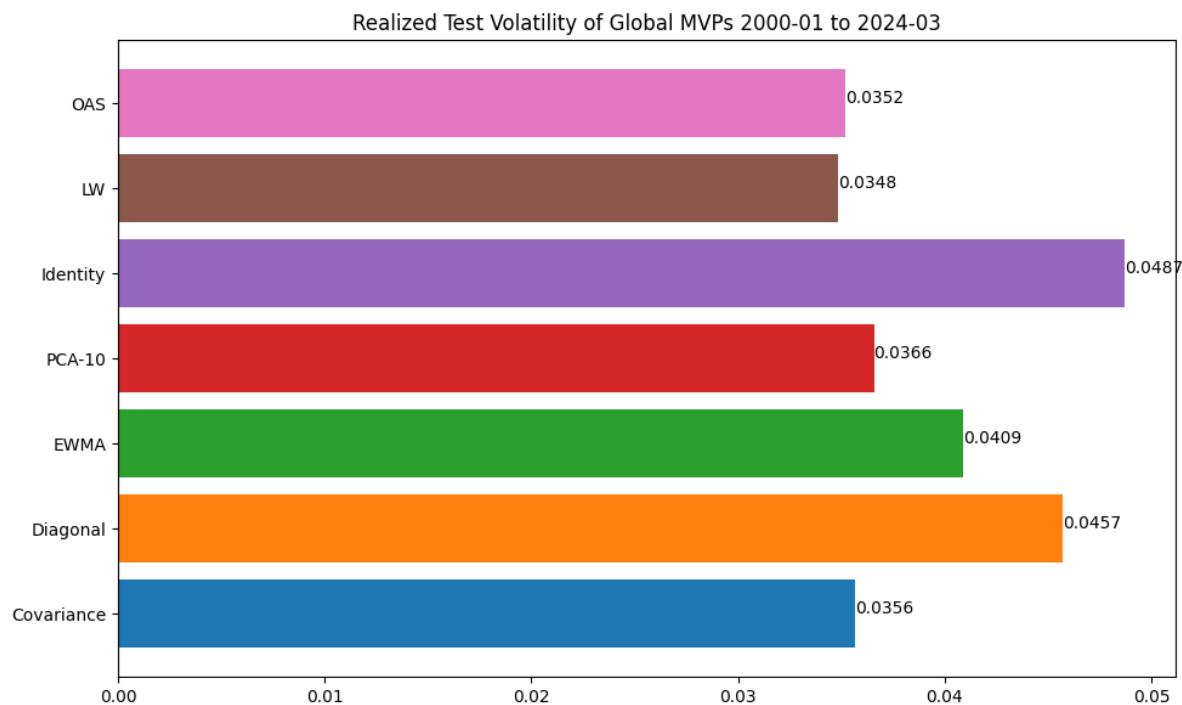
```
ts = DataFrame.from_dict(r, orient='index')
vol = ts.std().rename('realized volatility').to_frame()
print('Realized volatility of minimum variance portfolio')
vol.T
```

Realized volatility of minimum variance portfolio

| | Covariance | Diagonal | EWMA | PCA-10 | Identity | \ |
|---------------------|------------|----------|----------|----------|----------|---|
| realized volatility | 0.035649 | 0.045699 | 0.040867 | 0.036571 | 0.048712 | |
| | LW | OAS | | | | |
| realized volatility | 0.034823 | 0.035177 | | | | |

Plot evaluation period realized volatility of minimum variance portfolios

```
fig, ax = plt.subplots(figsize=(10, 6))
values = vol.values.flatten()
ax.barrh(vol.index, width=values, color=list(mcolors.TABLEAU_COLORS.values()))
for row, val in enumerate(values):
    ax.annotate(f'{val:.4f}', xy=(val, row))
ax.set_title(f'Realized Test Volatility of Global MVPs {split} to {rets.index[-1]}')
plt.tight_layout()
```



CHAPTER
SEVENTEEN

OPTIONS PRICING

The essence of investment management is the management of risks, not the management of returns - Benjamin Graham

Concepts

- Options
- Binomial Tree Pricing
- Black-Scholes-Merton
- VIX
- Monte Carlo Simulation

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, Ross, Rubinstein, 1979
- 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

```
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
```

17.1 Options

A **European** call (or put) option gives the buyer the right to buy (or sell) an asset at a certain price on a specific date. An **American** call (or put) option gives the buyer the right to buy (or sell) an asset at a certain price at any time before or on the specified date. The date specified in the option is known as the **expiration** date (also called the **maturity** date). The price at which an asset can be bought or sold using an option is referred to as the **strike** price (also called the **exercise** price).

Options can be in-the-money, at-the-money, or out-of-the-money. This is referred to as their moneyness. Under the simplest definition of these terms, it is imagined what would happen if the option could be exercised today. An option that would give a positive payoff if exercised today is referred to as in-the-money. If it would give a negative payoff, the option is referred to as out-of-the-money. An option that is at-the-money would give a payoff of zero.

The price of stock option depends on the:

- price of the underlying stock, S
- strike price, K
- risk-free rate, r
- volatility of the stock price, σ
- time to maturity, T ,
- dividends to be paid during the life of the option, q

Put-call parity describes the relationship between the price of a European call option and that of a European put option with the same strike price and time to maturity: $S - C = PV(K) - P$

17.1.1 Option Strategies

Trading strategies involve entering positions simultaneously in different options and, optionally (no put intended), the underlying asset.

- Buying a put option when the underlying asset is held is referred to as a **protective** put strategy. A put option combined with the asset gives a position equivalent to a call option combined with an amount of cash equal to $PV(K)$
- An asset plus a short call is known as a **covered call**. The call is usually out-of-the-money, allowing the asset owner to obtain a cash inflow equal to the call option premium in exchange for giving up some
- A **bull spread** is a position appropriate for an investor expecting an increase in the price of an asset. To create the spread, the trader buys a European call option with strike price $K1$ and sells a European call option with higher strike price $K2$.
- A **bear spread** is a position where the trader buys a European put option with strike price $K2$ and sells a European put option with lower strike price $K1$.
- A **box spread** is a portfolio created from a bull spread (using call options) and a bear spread (using put options). The strike prices and times to maturity used for the bull spread are the same as those used for the bear spread.
- A **butterfly spread** involves positions in three options. It can be created from either call or put options. If calls are used, a trader will take positions in one long European call with strike price $K1$, one long European call with higher strike price $K3$, and two short European calls with strike price $K2 = (K1 + K3)/2$.
- To create a **calendar spread**, a trader buys a call option maturing at time T^* and sells a call maturing at a nearer time T , both at the same strike price of K
- A **straddle** is created from a long call and a long put with the same strike price and time to maturity.

- A **strangle** position reduces the cost of a straddle by making the strike price of the call greater than the strike price of the put.

```
# 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)
    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')
```

(continues on next page)

(continued from previous page)

```

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)}

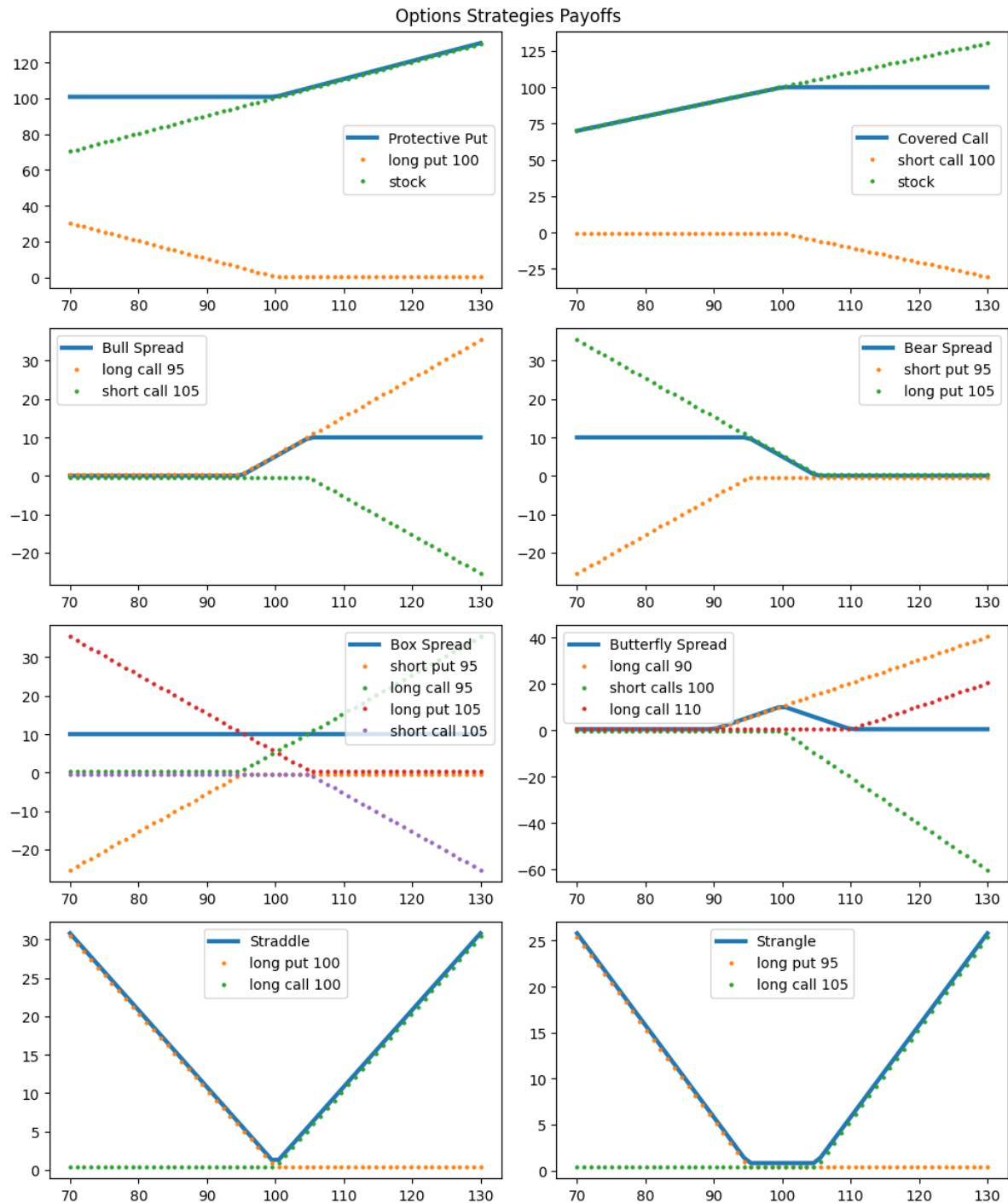
plot_payoff(butterfly_spread(K1=90, K3=110), ax=ax[5], label='Butterfly Spread')

def straddle(K):
    return lambda s: {f"long put {K}": put_payoff(K)(s),
                      f"long call {K}": call_payoff(K)(s)}
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()

```



17.1.2 Exotic Options

Standard European and American options are termed plain vanilla options. Options with non-standard properties are termed exotic options (or simply exotics).

- 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.

17.2 Binomial Tree

This valuation technique was proposed by Cox, Ross, and Rubinstein (1979) and continues to widely used for pricing American-style options and many other derivatives.

17.2.1 No-arbitrage

The no-arbitrage argument means that prices are calculated on the assumption that there are no arbitrage opportunities for market participants. The **law of one price** states that if two portfolios provide the same cash flows at the same times in the future, they should sell for the same price. Otherwise, a trader can short the more expensive portfolio and buy the cheaper portfolio to lock in a riskless profit. Binomial trees apply no-arbitrage arguments to pricing derivatives.

Suppose that the price of a non-dividend paying stock is currently S , and that during a time T it will either move up to S_u or down to S_d . We consider a derivative that provides a payoff of f_u if the stock price increases, and a payoff of f_d if the stock price decreases. We then form a portfolio consisting of:

- A short position in one unit of the derivative; and
- A position of Δ in the stock which we set to equal

$$\Delta = \frac{f_u - f_d}{S_u - S_d}$$

The value of the portfolio at time T is

- $Su\Delta - f_u$ if the stock price increases, and
- $Sd\Delta - f_d$ 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:

$$f = e^{-rT} [pf_u + (1 - p)f_d]$$

$$\text{where: } p = \frac{e^{rT} - d}{u - d}$$

17.2.2 Risk Neutral Pricing

We will also use binomial trees to introduce what is known as risk-neutral valuation. This is a result that allows derivatives to be valued by assuming that market participants require an expected return equal to the risk-free rate on all investments. We define a risk-neutral world as one where investors do not adjust their required expected returns for risk, so that the expected return on all assets is the risk-free rate. To put this 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. The risk-neutral valuation principle states that if we assume we are in a risk-neutral world, we get the correct price for a derivative. As it turns out, the price is correct in the real world (where investors do care about risk) as well as in a risk-neutral world.

First, we note that if 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 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.

We have therefore demonstrated the truth of the risk-neutral valuation result. If we assume a risk-neutral world, the probability of an upward movement is p , and the value of the derivative is its expected payoff discounted at the risk-free rate (i.e., it is the value that would apply if market participants were risk neutral).

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.

17.2.3 Binomial Trees for Valuing Options

In practice, we model stock price changes using a multi-step tree, and define

- 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

$$- u = e^{\sigma\sqrt{\Delta t}}$$

$$- u = e^{-\sigma\sqrt{\Delta t}}$$

where Δt is measured in years.

- hence $f = e^{-r\Delta t} [pf_u + (1 - p)f_d]$ where $p = \frac{e^{\sigma\sqrt{\Delta t}} - d}{u - d}$
- the delta, or position taken in the stock to hedge a short position in the derivative, is $\Delta = \frac{f_u - f_d}{Su - Sd}$

Dividends: Suppose a stock paying a continuous dividend yield at rate q . Then we adjust the formula for p , while everything else about the tree, including the calculation of u and d and the roll back procedure, is the same as before

$$p = \frac{e^{(r-q)\Delta t} - d}{u - d}$$

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
                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))
```

| | 0 | 1 | 2 |
|--------|---------|---------|---------|
| stock | 41.2995 | 29.0000 | 20.3635 |
| option | 11.2995 | 0.0000 | 0.0000 |
| delta | 0.0000 | 0.0000 | 0.0000 |

| | 0 | 1 |
|--------|---------|---------|
| stock | 34.6076 | 24.3010 |
| option | 5.5483 | 0.0000 |
| delta | 1.0963 | 0.0000 |

| | 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))
```

| | 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 | 5.2523 |
| 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)
```

| 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 |

| STEP 1 | 0 | 1 |
|--------|---------|---------|
| stock | 34.6076 | 24.3010 |
| option | 0.4941 | 5.6990 |
| delta | -0.0970 | -0.8380 |

| STEP 0 | 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)
```

| STEP 4 | 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 |

| STEP 3 | 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 |

| STEP 2 | 0 | 1 | 2 |
|--------|---------|---------|---------|
| stock | 37.2367 | 29.0000 | 22.5852 |
| option | 0.2474 | 2.4387 | 7.4148 |
| delta | -0.0684 | -0.5379 | -0.7788 |

| STEP 1 | 0 | 1 |
|--------|---------|---------|
| stock | 32.8613 | 25.5924 |
| option | 1.3356 | 4.8958 |
| delta | -0.3015 | -0.6846 |

| STEP 0 | 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 |
| STEP 2 | 0 | 1 | 2 | |
| stock | 2825.7257 | 2500.0000 | 2211.8212 | |
| option | 328.7911 | 78.2792 | 0.0000 | |
| delta | 1.1303 | 0.5153 | 0.0000 | |
| STEP 1 | 0 | 1 | | |
| stock | 2657.8778 | 2351.5002 | | |
| option | 202.0979 | 38.8125 | | |
| delta | 0.8177 | 0.2555 | | |
| STEP 0 | 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))
```

| | 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 | | | | |

(continues on next page)

(continued from previous page)

| | |
|--------|--------|
| 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 |

17.3 Black-Scholes-Merton

The famous Black-Scholes-Merton model was published in two papers in 1973 and has had a major influence on the way options are priced and hedged. 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 paper, Merton used no-arbitrage arguments like those used in connection with binomial trees in the previous chapter. The two papers derived the same option pricing formula.

Merton used assumptions and arguments (many of which are similar to those for valuaing options using binomial trees), including lognormal stock price distribution, continuous security trading and no riskless opportunities, to show that the price evolution of derivatives satisfies the Black-Scholes-Merton 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 use Merton's result to value a European option, we must 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:

$$C = Se^{-qT}\Phi(d_1) - Ke^{-rT}\Phi(d_2)$$

$$p = Ke^{-rT}\Phi(-d_2) - Se^{-qT}\Phi(-d_1)$$

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.

American call options on a non-dividend paying stock should never be exercised early, hence the same formula provides prices for American call options on non-dividend paying stocks as well as for European call options. 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 formulas

```
def _d1(S, K, sigma, r, 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
```

17.3.1 Implied Volatility

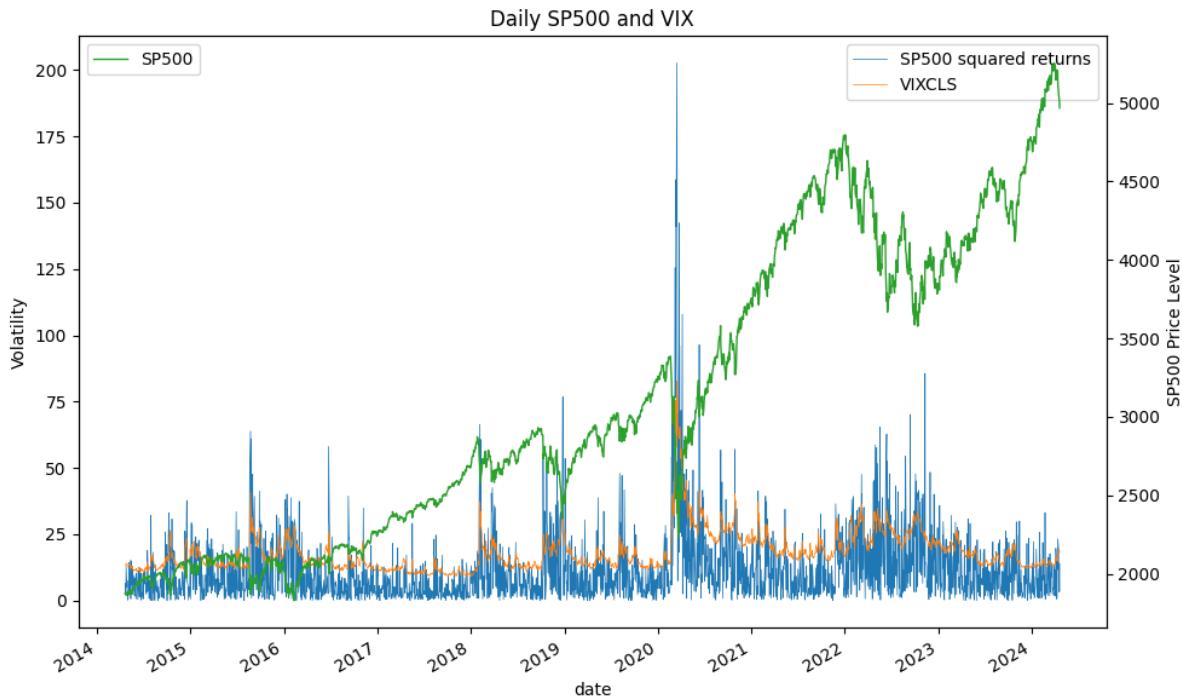
The implied volatility of an option is the volatility that gives the market price of the option when it is substituted into the Black-Scholes-Merton formula.

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. 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.

```
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()
```



17.3.2 Volatility Smile

In practice, implied volatilities vary with strike prices. This indicates that the market does not price options consistently with the Black-Scholes-Merton assumptions. The volatility smile defines the relationship between the implied volatility of an option and its strike price. 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.

For equity options, the volatility smile tends to be downward sloping. This means that out-of-the-money puts and in-the-money calls tend to have high implied volatilities whereas out-of-the-money calls and in-the-money puts tend to have low implied volatilities. This is sometimes referred to as a volatility skew. There is a negative correlation between equity prices and volatility, which means that stock price declines are accompanied by increases in volatility, and stock price increases are accompanied by decreases in volatility.

For foreign currency options, the volatility smile is U-shaped. Both out-of-the-money and in-the-money options have higher implied volatilities than at-the-money options. The volatility of an exchange rate is far from constant, and exchange rates frequently exhibit jumps, sometimes in response to the actions of central banks. Both a nonconstant volatility and jumps will have the effect of making extreme outcomes more likely.

Often traders also use a volatility term structure. The implied volatility of an option then depends on its life. When volatility smiles and volatility term structures are combined, they produce a volatility surface. This defines implied volatility as a function of both the strike price and the time to maturity. When quoting options prices, traders interpolate between the known implied volatilities to determine an implied volatility for the option under consideration. This is then substituted into the Black-Scholes-Merton equation to determine the option price. This procedure is a way of overcoming the fact that the market does not price options consistently with the Black-Scholes-Merton assumptions.

17.3.3 The “Greeks”

The Greek letters (or Greeks, as they are sometimes called) are designed to provide information about the risks of options positions. Some Greek letters are concerned with movements in the price of the underlying asset; some are concerned with volatility changes; and others involve interest rates and dividend yields. Typically, a trader is subject to limits on how large the Greek letters can be. If one of the Greek letters exceeds the applicable limit near the end of the trading day, a trader must either execute a trade that corrects the situation or seek special permission from the risk management function to maintain the existing position.

- 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 $\$ \Theta + (r - q)S\Delta + \frac{1}{2}\sigma^2S^2\Gamma = (r - q)C\$$

```
# 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)
```

(continues on next page)

(continued from previous page)

```

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)
    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 vega"""
    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.):

```

(continues on next page)

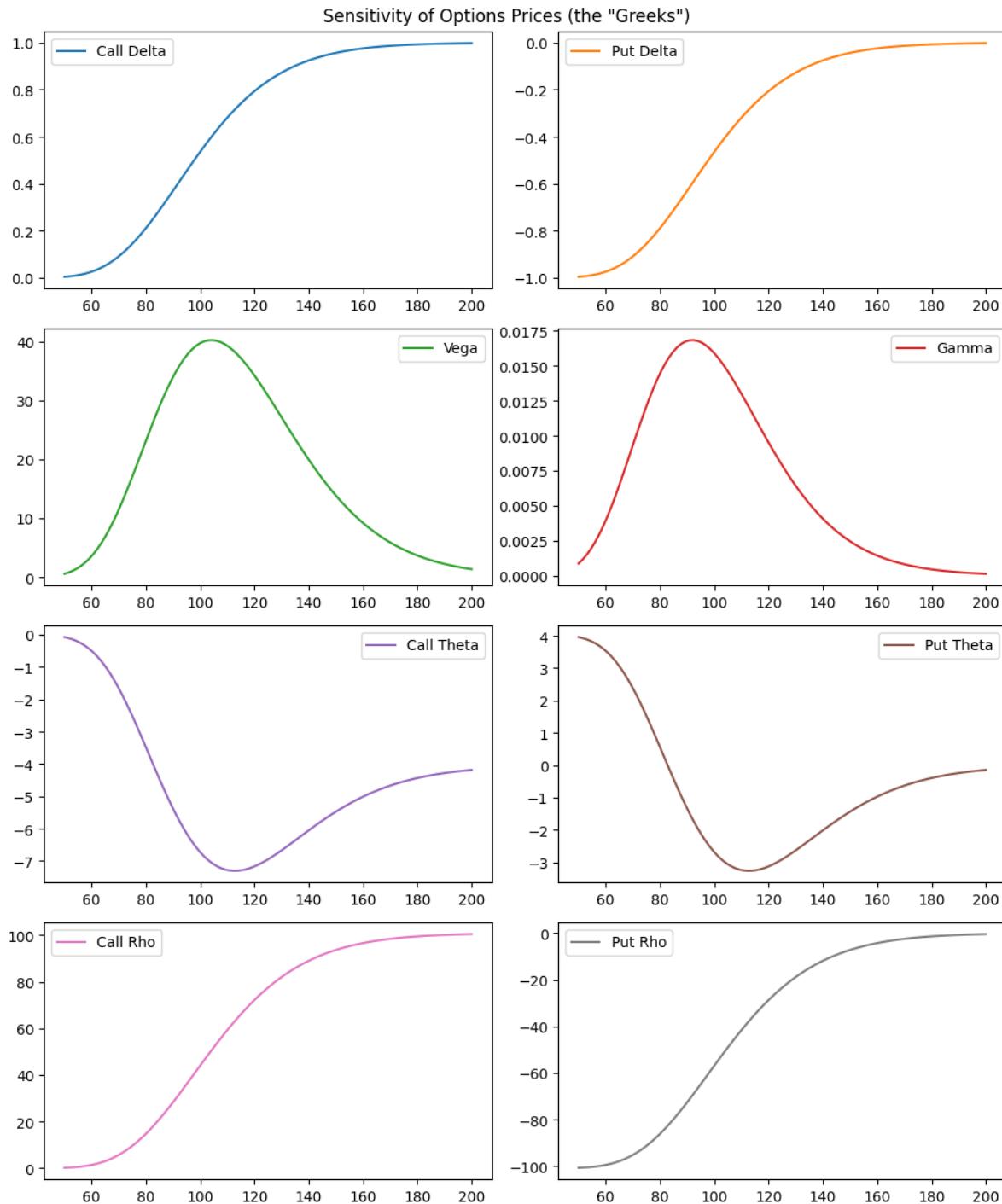
(continued from previous page)

```
"""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)
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
```

17.4 Monte Carlo Simulation

Simulation provides a simple method to approximate the analytically intractable expectation. Suppose X is a random variable that can be simulated (e.g., a normal) and g is a function that can be evaluated at realizations of X .

Simulated draws are iid by construction, and b draws are used to approximate the expected value as: $\hat{E}[g(X)] = \frac{1}{b} \sum_{i=1}^b g(X_i)$

The approximated expectation is an average of b iid random variables, and so the Law of Large Numbers (LLN) applies: $b \lim_{b \rightarrow \infty} \hat{E}[g(X)] = E[g(X)]$

Furthermore, the Central Limit Theorem (CLT) also applies to the average. The variance of the simulated expectation is estimated by: $V[\hat{E}[g(X)]] = \sigma_b^2/b$

This variance is estimated using: $\sigma_g^2 = \frac{1}{b} \sum_{i=1}^b (g(X_i) - E[g(X_i)])^2$ which is the standard variance estimator for iid data.

The standard error of the simulated expectation σ_g/\sqrt{b} measures the accuracy of the approximation, which allows b to be chosen to achieve any desired level of accuracy.

17.4.1 Antithetic Variates

Antithetic variates add a second set of random variables that are constructed to have a negative correlation with the iid variables used in the simulation. They are generated in pairs using a single uniform value. Only $b/2$ uniforms are required to produce b simulated values because each draw $U - i$ also generates the $1 - U_i$ which both map through the inverse CDF to generate random variables that are negatively correlated.

Antithetic pairs produced using this algorithm are only guaranteed to reduce the simulation error if the function $g(X)$ is monotonic to ensure negative correlation. The improvement in accuracy when using antithetic variables occurs due to the negative correlation between the paired random variables which decreases standard error.

17.4.2 Control Variates

A control variate is another derived random variable $h(X_i)$ that has mean 0 but is correlated with the error of the primary random variable $g(X_i)$. A good control variate should have two properties.

1. First, it should be inexpensive to construct from x_i , than the computation cost of simply increasing the number of simulations.
2. Second, a control variate should have a high correlation with $g(X)$. The optimal combination parameter β that minimizes the approximation error is estimated using the regression:
3. $g(x_i) = \alpha + \beta h(x_i)$

17.4.3 Simulating Option Prices

Assume that the log of the stock price is normally distributed. The time T log stock price s_T is equal to the sum of the initial stock price s_0 , a mean, and a normally distributed error:

$$s_T = s_0 + T(r_f - \sigma^2/2) + \sqrt{T}x_i$$

where x_i is a simulated value from a $N(0, \sigma^2)$, r_f is the risk-free rate, and σ^2 is the variance of the stock return.

The final stock price is $S_T = \exp(s_T)$, and the present value of the option's payoff is

$$C = \exp(-r_f T) \max(S_T - K, 0)$$

The expected price of a call option is estimated by the average of the simulated values:

$$E[C] = \bar{C} = \frac{1}{b} \sum_{i=1}^b C_i$$

where C_i are simulated values of the payoff of the call option.

```
# 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])
            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
        """

```

(continues on next page)

(continued from previous page)

```

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_
    ↪price
        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([[-1.49453301,  1.49453301,  0.80816507],
       [-0.80816507, -0.28928827,  0.28928827]])
```

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 | 0.0004 |
| 200 | 247.65 | 25.97 | 0.0002 |
| 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 | 0.0021 |
| 204800 | 278.93 | 0.93 | 0.0081 |
| 819200 | 278.31 | 0.46 | 0.0361 |
| 3276800 | 278.59 | 0.23 | 0.2964 |

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 | 0.0003 |
| 800 | 276.97 | 14.15 | 0.0007 |
| 3200 | 284.38 | 7.63 | 0.0006 |
| 12800 | 277.30 | 3.72 | 0.0019 |
| 51200 | 279.28 | 1.87 | 0.0070 |
| 204800 | 278.15 | 0.93 | 0.0298 |
| 819200 | 278.46 | 0.46 | 0.1209 |
| 3276800 | 278.48 | 0.23 | 0.4569 |

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 | 0.0005 |
| 200 | 251.85 | 22.76 | 0.0003 |
| 800 | 287.49 | 13.52 | 0.0005 |
| 3200 | 279.92 | 6.76 | 0.0004 |
| 12800 | 278.43 | 3.32 | 0.0007 |
| 51200 | 279.01 | 1.64 | 0.0027 |
| 204800 | 279.02 | 0.83 | 0.0120 |
| 819200 | 278.22 | 0.41 | 0.0453 |
| 3276800 | 278.64 | 0.21 | 0.3122 |

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 | 0.0005 |
| 200 | 227.29 | 22.03 | 0.0006 |
| 800 | 277.91 | 12.32 | 0.0004 |
| 3200 | 286.51 | 6.79 | 0.0006 |
| 12800 | 276.56 | 3.31 | 0.0020 |
| 51200 | 279.47 | 1.66 | 0.0063 |
| 204800 | 277.88 | 0.83 | 0.0273 |
| 819200 | 278.41 | 0.41 | 0.1039 |
| 3276800 | 278.41 | 0.21 | 0.5222 |

MARKET MICROSTRUCTURE

Beware of little expenses. A small leak will sink a great ship - Benjamin Franklin

- NYSE Daily TAQ tick data
- Liquidity by firm size
- Sampling frequency
- Volatility from tick data
- Intraday liquidity

```
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
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)
```

Last FamaFrench Date 2024-02-29 00:00:00

```
taqdir = paths['taq']
storedir = paths['scratch'] / 'ticks'
```

(continues on next page)

(continued from previous page)

```
open_t = pd.to_datetime('1900-01-01T9:30')      # exclude <=
close_t = pd.to_datetime('1900-01-01T16:00')    # exclude >
EPSILON = 1e-15
dates = [20191007, 20191008, 20180305, 20180306]
```

18.1 TAQ tick data

Extract tick data from TAQ daily and store by date and symbol

```
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 iterataq(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()
```

Compute intraday liquidity measures

```
intervals = ((v, 's') for v in [1, 2, 5, 15, 30]) + ((v, 'm') for v in [1, 2, 5])
max_num = 100000
```

(continues on next page)

(continued from previous page)

```

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'],
                                                   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))

```

(continues on next page)

(continued from previous page)

```

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}

```

Multiprocessing is a package that supports spawning processes. The Pool API parallelizes the execution of a function across multiple input values, distributing the input data across processes (data parallelism).

```

# Run each day as a parallel thread, must use as context manager or close
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']

```

18.2 Liquidity by firm size

VWAP execution benchmark The other issue with VWAP as well as TWAP, market open or close prices is that they are benchmarks that can be influenced by the trade itself. Zero slippage to VWAP but being 100 percent of market volume does not necessarily mean it is a good execution. Zero slippage to the closing price while being part of an adverse high / low on the close is also not a good execution.

Daily average of liquidity measures, by market cap

- depth
- quoted spread
- effective spread
- price impact
- realized spread
- Lee-Ready tick test

```
# 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',
                           'tvarOHL5m', 'qvarOHL5m']].mean())
result.columns = ['Volatility(trade price)', "Volatility(midquote)",
                  'Volatility(HL trade price)', "Volatility(HL midquote)",
                  'Volatility(OHL trade price)', "Volatility(OHL 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]
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 |

(continues on next page)

(continued from previous page)

| | | | | |
|------------------------------|--------|--------|--------|--------|
| 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 |
| 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 |

18.3 Sampling frequency

continuous trading

spurious autocorrelation

- Variance Ratio

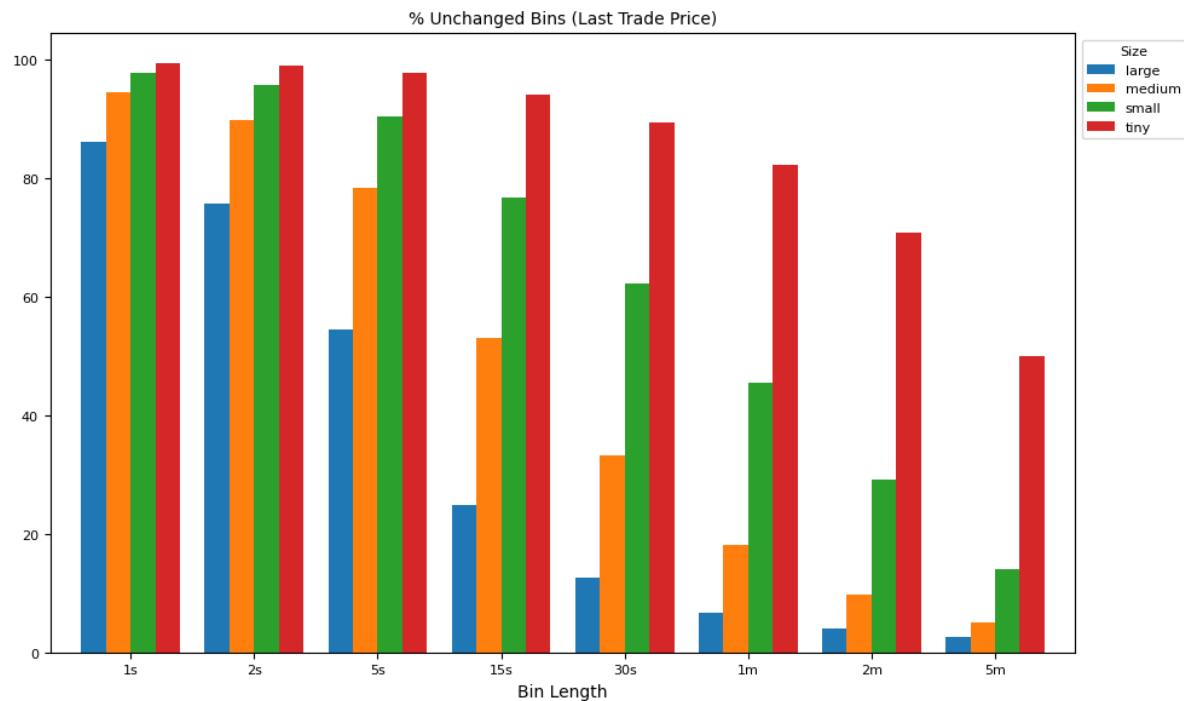
```
def plot_helper(result, xticks, keys, legend, xlabel, title, ylim=[], figsize=(10, 6), num=1, fontsize=8):
    """helper to plot bar graphs by sampling frequency"""
    fig, ax = plt.subplots(num=num, clear=True, figsize=figsize)
    result.plot(kind='bar',
                fontsize=fontsize,
                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{v}u{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)

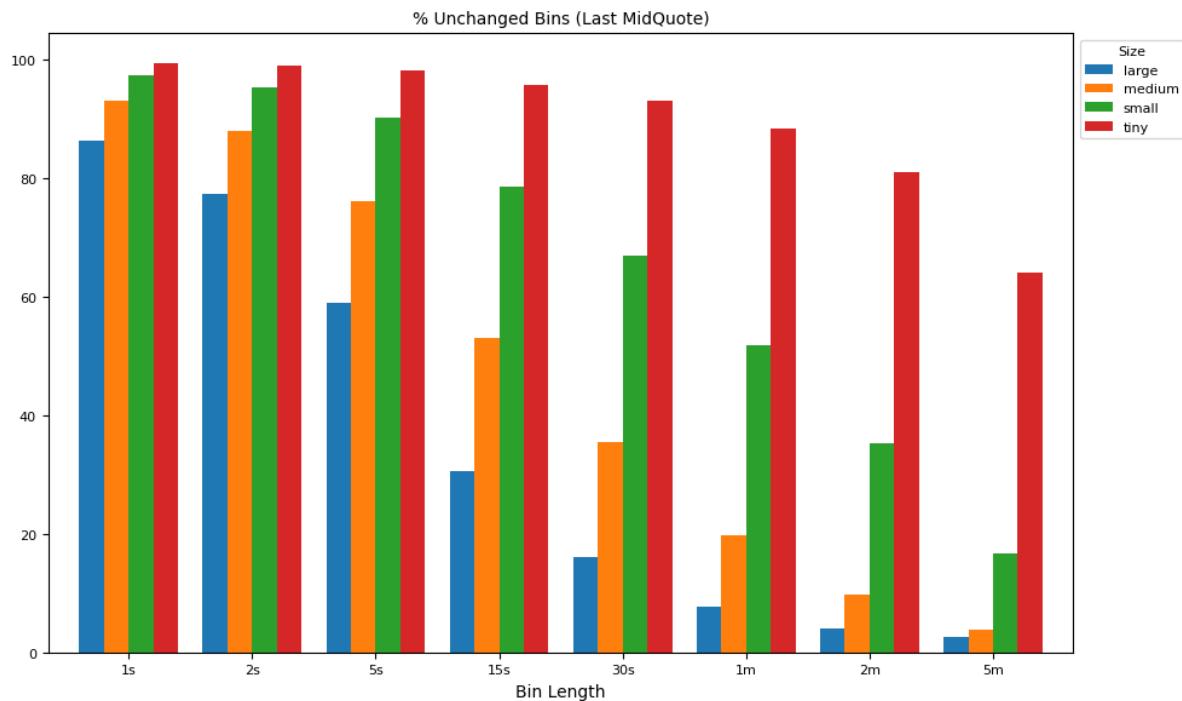
```



```

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)

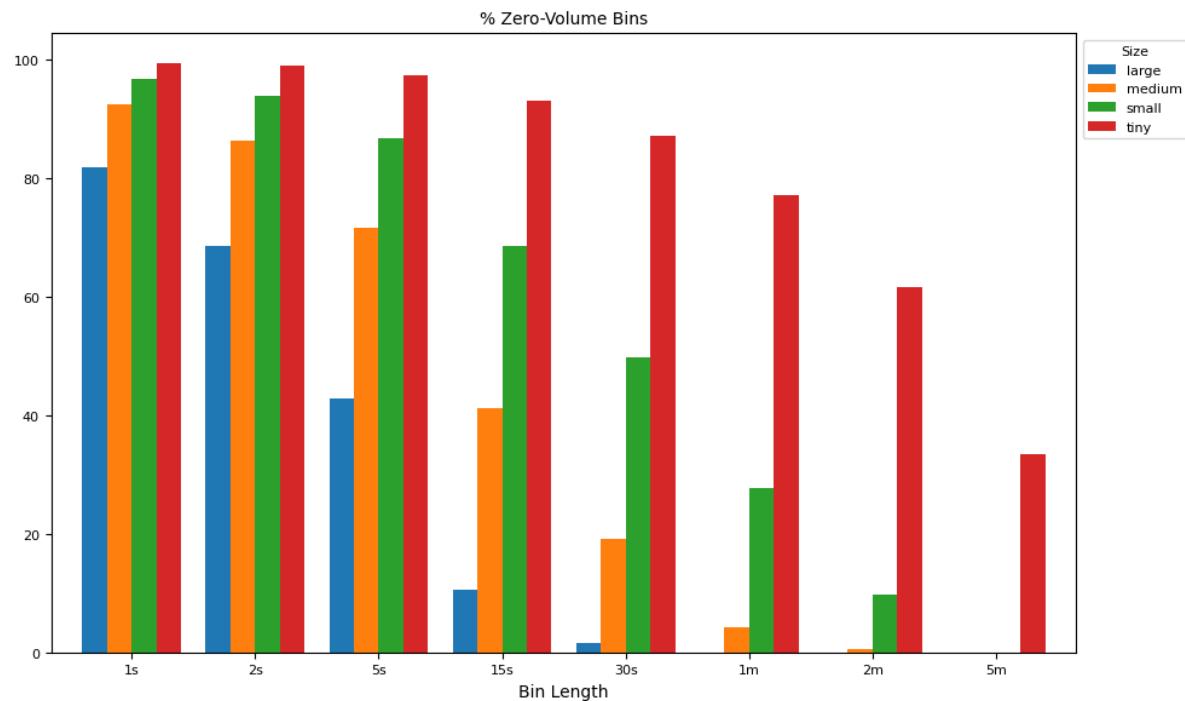
```



```

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)

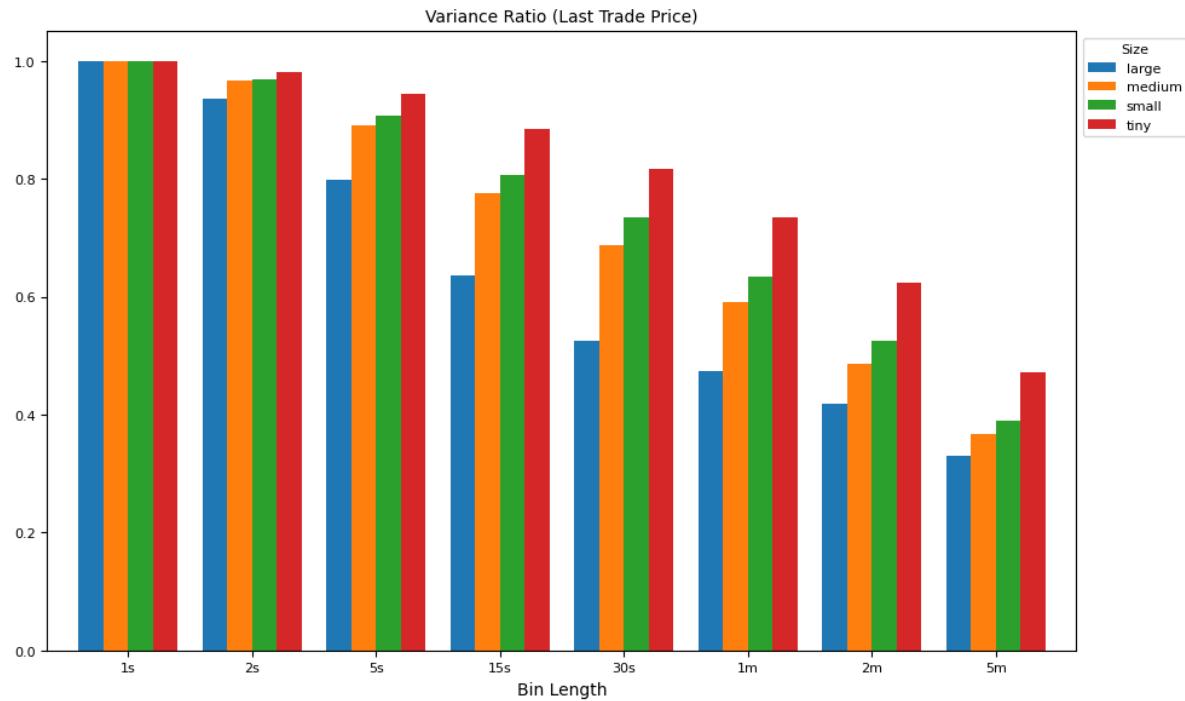
```



```

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)

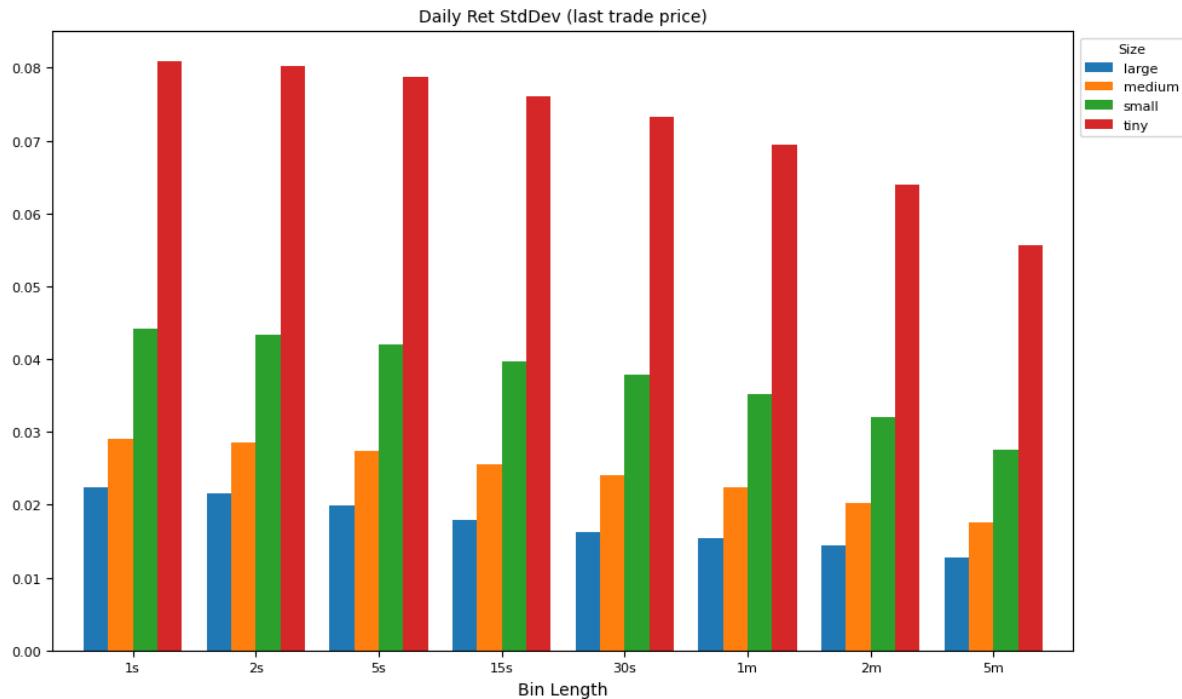
```



```

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)

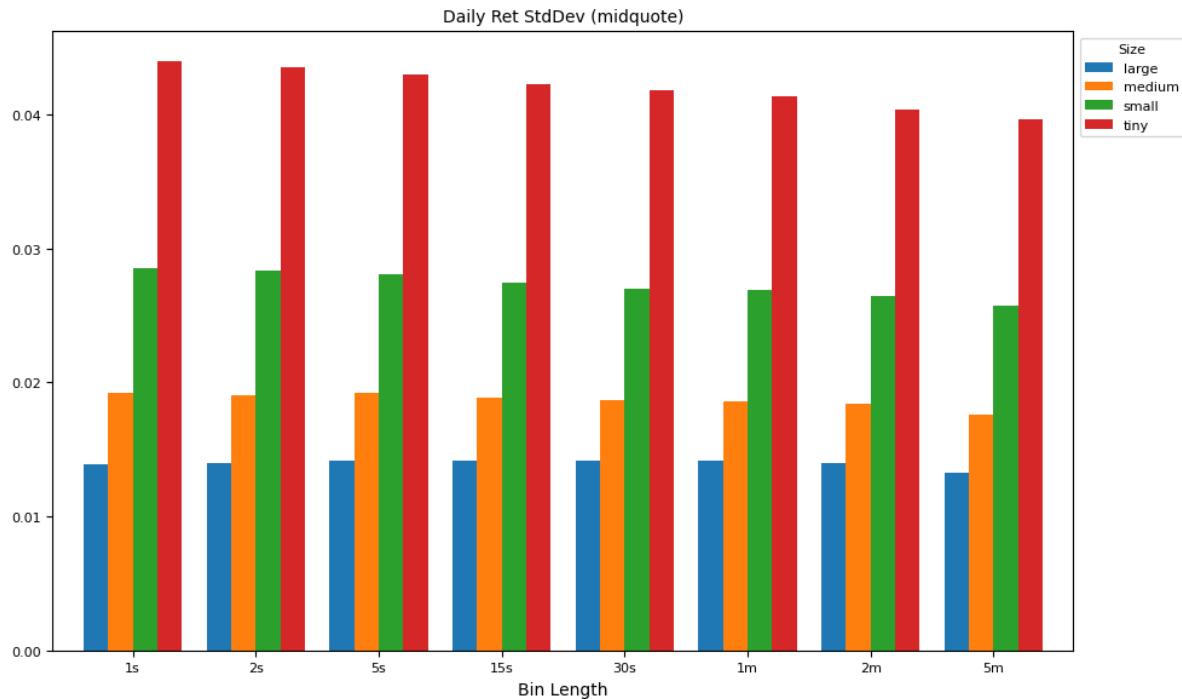
```



```

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)

```



18.4 Volatility from tick data

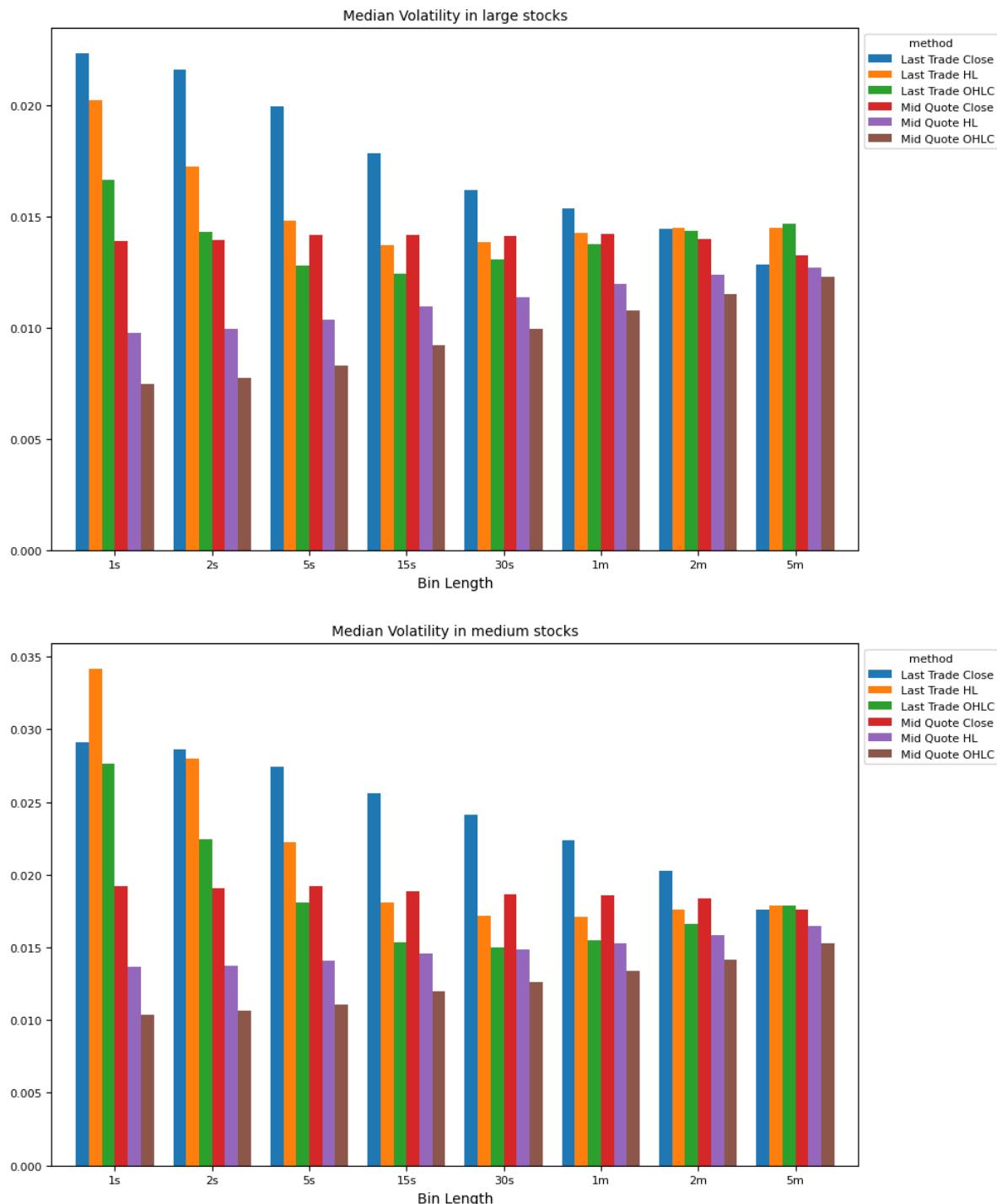
- Parkinsons (HL)
- Klass-Garman (OHLC) methods

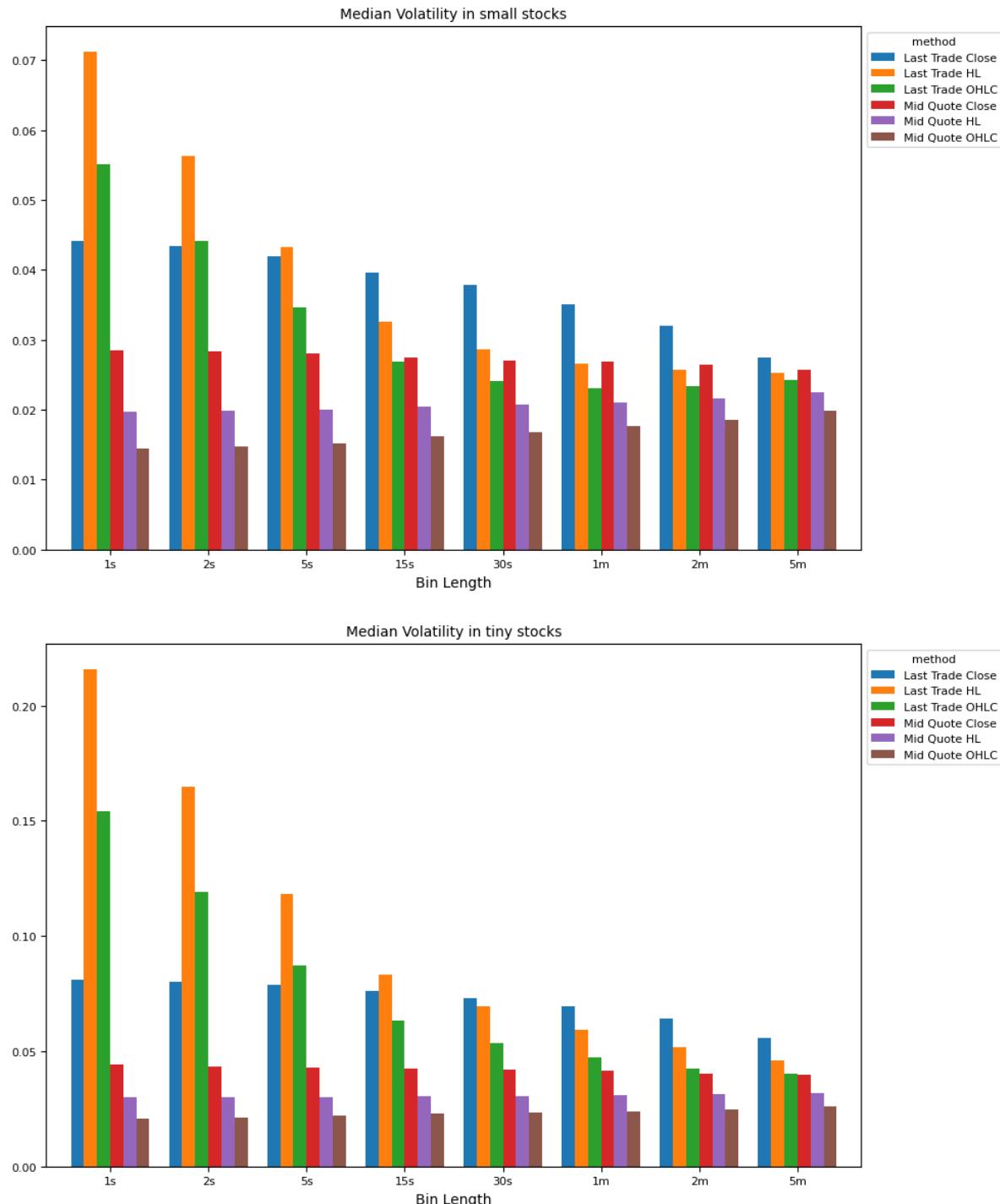
```
# Compare methods of volatility estimates, by interval and market cap
for ifig, (split_label, split_df) in enumerate(groupby):
    vol_df = np.sqrt(split_df[[c for c in daily_df.columns if "qvar" in c or "tvar" in c]])
    result = []
    for col in [c for c in vol_df.columns if "qvar" in c or "tvar" 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()})
    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,
```

(continues on next page)

(continued from previous page)

```
xlabel="Bin Length",
keys=result.columns,
num=ifig+1,
legend='method')
```





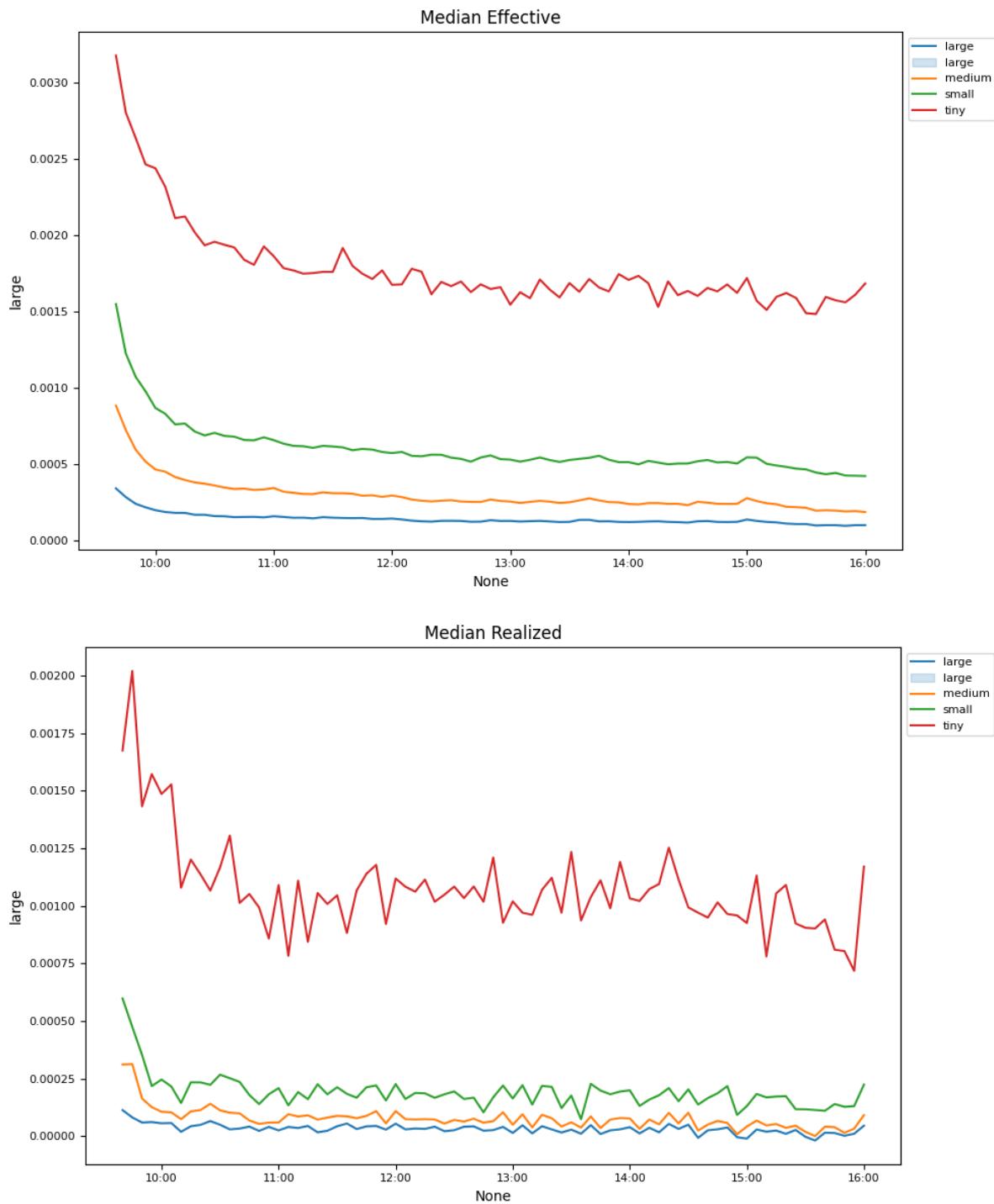
18.5 Intraday liquidity

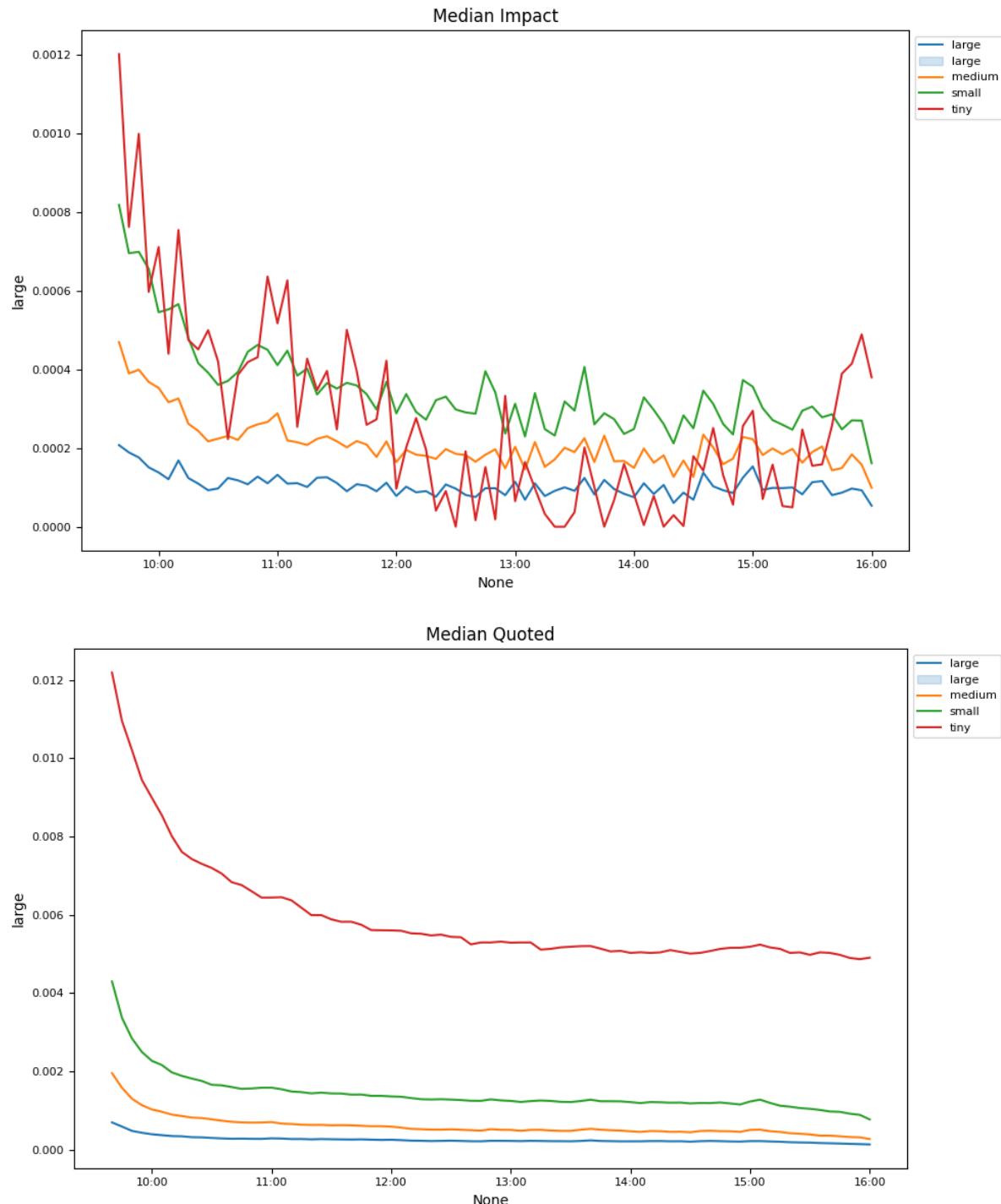
Order types (and intraday patterns)

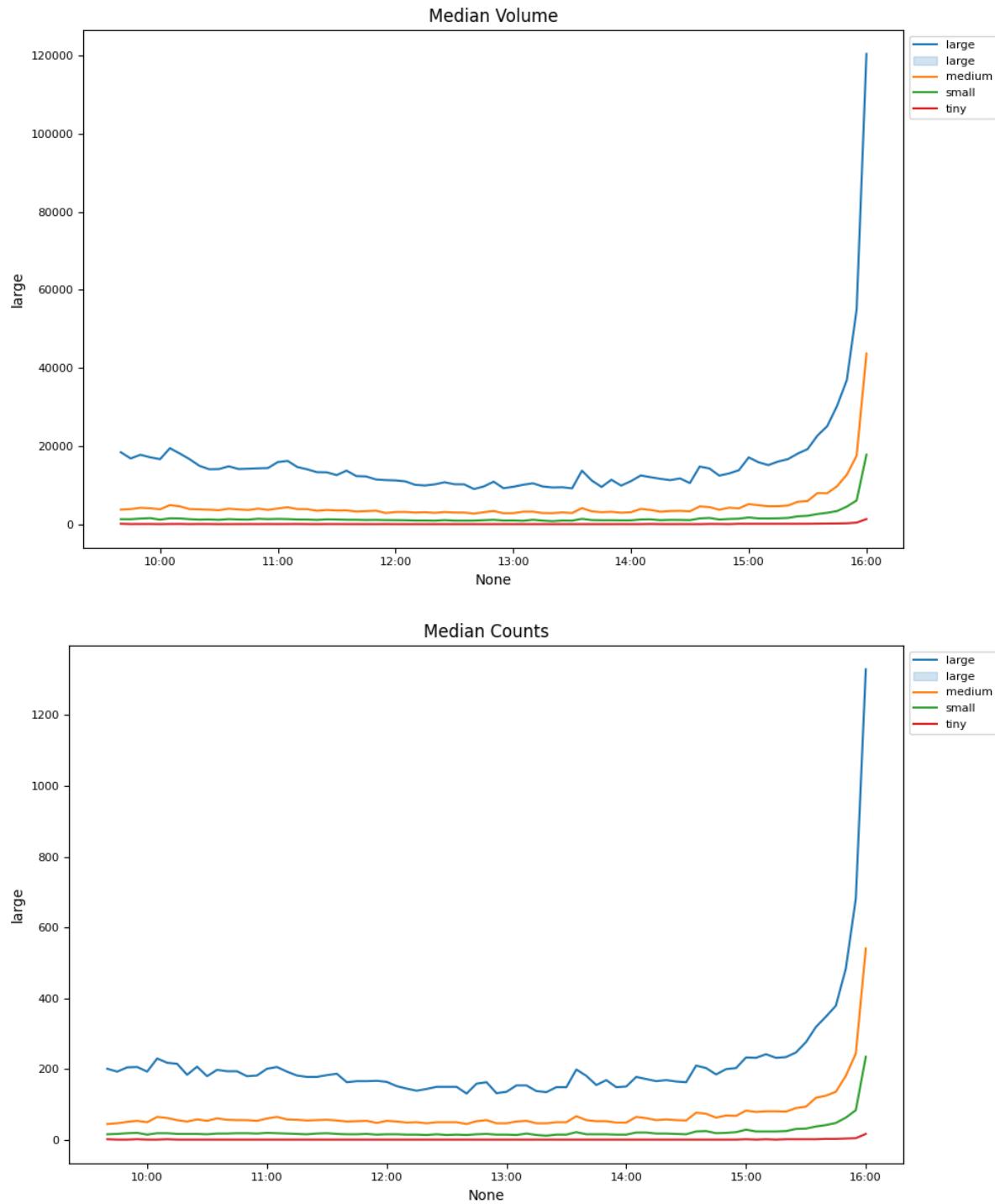
- market
- limit
- stop
- market-on-close

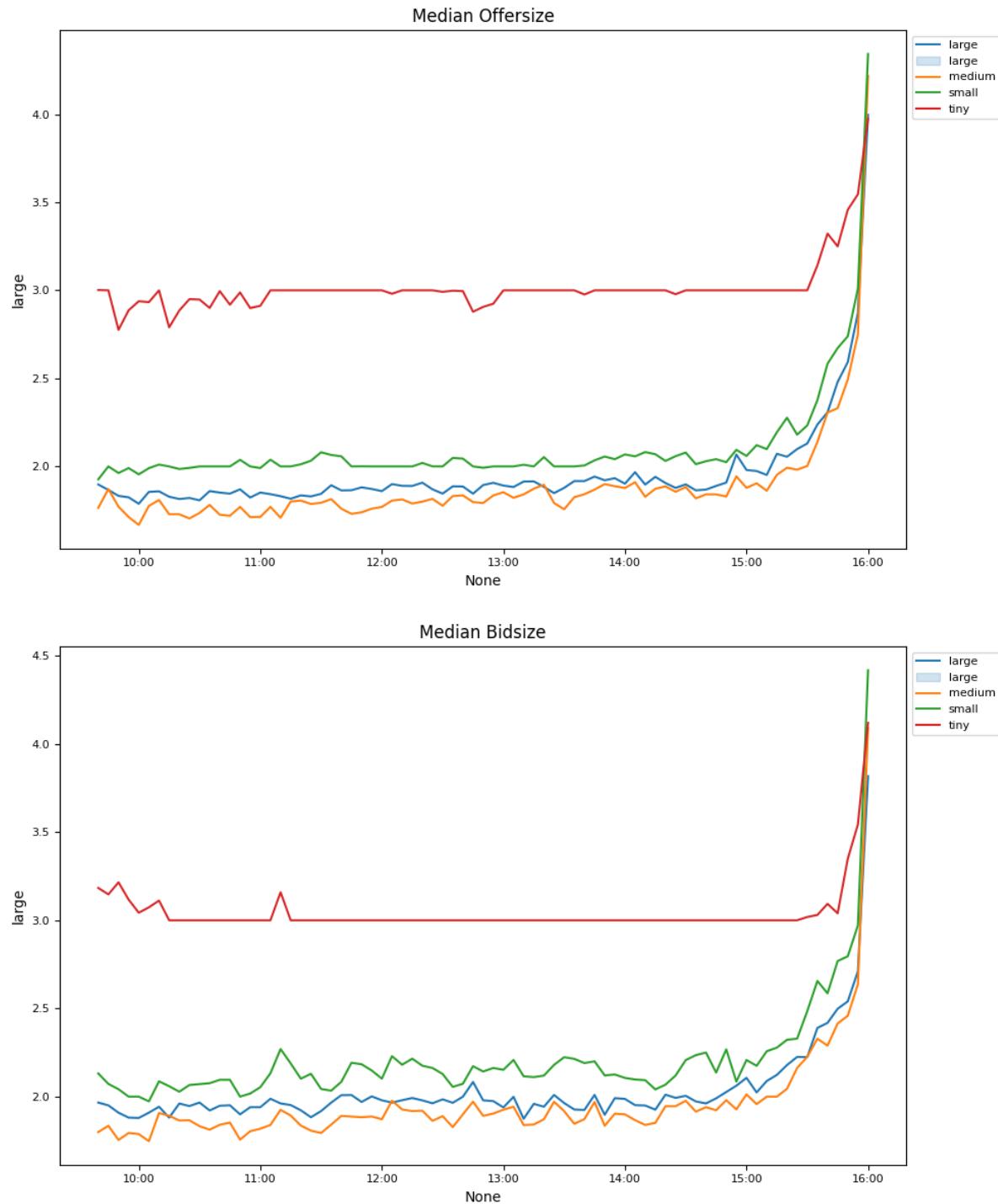
```
# Plot intraday spreads, depths and volumes
keys = ['effective', 'realized', 'impact', 'quoted',
        'volume', 'counts', 'offersize', 'bidsize']
for num, key in enumerate(keys):
    df = bins_df[key].drop(columns=['Round_Lot', 'Symbol'])
    df.index = list(zip(df['permno'], df['date']))

    # 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()
```









EVENT RISK

Don't worry about failure; you only have to be right once — Drew Houston

Concepts:

- Earnings surprises
- Poisson regression
- Generalized Linear Models
- Credit Losses

```
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)
store = Store(paths['scratch'])
LAST_DATE = CRSP_DATE
```

19.1 Earnings misses

The quarterly earnings expectations from all the analysts that cover a company are averaged out to find the consensus analyst estimate. The number of analysts covering a particular company is largely dependent on the size of the company and the trading volume of its stock. Large, well-known companies tend to be covered by many analysts and the consensus is therefore derived from many opinions. A small, lesser-known company may be covered by only one or a few research analysts, and sometimes none at all.

If the company reports earnings below what research analysts were expecting, this would be considered an **earnings miss**. If the company reports earnings greater than what analysts expected, it would be considered an **earnings beat**

1. Each quarter and stock in the universe: select latest IBES statistical period that is no later than than the fiscal quarter end date
2. SUE (standardized unexpected earnings) is computed as the fiscal quarter's reported earnings minus median Q1 estimate, scaled by 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 miss 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()
```



Median quarterly earnings surprise tended to be negative prior to the mid-90's (especially smaller stocks) but turned generally positive after that.

```
# 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

| | exposures | frac | count |
|-----------|-----------|----------|-------|
| rebaldate | | | |
| 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 |
| ... | ... | ... | ... |
| 202209 | 2080 | 0.009615 | 20 |
| 202212 | 2008 | 0.007968 | 16 |
| 202303 | 1910 | 0.005759 | 11 |
| 202306 | 1902 | 0.003680 | 7 |
| 202309 | 1842 | 0.008686 | 16 |

[156 rows x 3 columns]

19.2 Poisson regression

The Poisson distribution is typically used to model counts. A Poisson regression models the expected response as a function of the covariates, specifically: $\log(E[Y|X]) = \beta X$, or equivalently $E[Y|X] = e^{\beta X}$

- Interpretation of coefficients: an increase in X_i by one unit is associated with a change in $E[Y_i]$ by a factor of $\exp \beta_i$
- Under the Poisson model, $Var[Y_i] = E[Y_i]$
- No negative fitted values, because the Poisson model only allows for non-negative values.

Poisson regression may also be appropriate for *rate* data, where the count of events divided by some measure of that unit's **exposure**. When modelling the number of stocks with large earnings misses each quarter, the denominator of the rate would be the total number of stocks N in the universe that quarter. The log of the total number is called the **offset** variable, and enters on the right-hand side of the equation and a parameter estimate constrained to 1.

$$\log\left[\frac{E[Y|X]}{\text{exposure}}\right] = \log E[Y|X] - \log(\text{exposure}) = X\beta - \log(\text{exposure})$$

19.2.1 Generalized Linear Models

Linear and Poisson regression are generalized linear models (GLMs).

- Each approach models the mean of Y_i as a function of the predictors, specifically: the transformed mean $E[Y_i]$ is a linear function of the predictors. That transformation is called a **link function** ν , which provides the relationship between the linear predictor and the mean of the distribution function. The inverse of the link function is called the **mean function**. The link functions for linear and Poisson regression are the identity $\nu(E[Y]) = E[Y]$ and $\log \nu(E[Y]) = \log E[Y]$ respectively.
- Conditional on X , Y belongs to some family of distributions, typically Gaussian for linear regression and Poisson for Poisson regression. These distributions are members of the **exponential family**: other well-known members of this family are the Bernoulli, Multinomial, Exponential, Gamma, and the negative binomial distributions.

Any regression approach that models the response Y as coming from a particular member of the exponential family, and then transforming the mean of the response so that the transformed mean is a linear function of the predictors is known as a generalized linear model. 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 $E[Y] = e^{X\beta}$

```
# 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', 'WILL5000IND']],
              axis=1)\n
      .dropna()
X.index = (X.index // 100).astype(str)

# 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'])\n
      .fit()
y_pred = glm.predict(exog=Z[['const'] + list(X.columns)],
                      exposure=Z['exposures']).rename('predicted')
print(glm.summary())
```

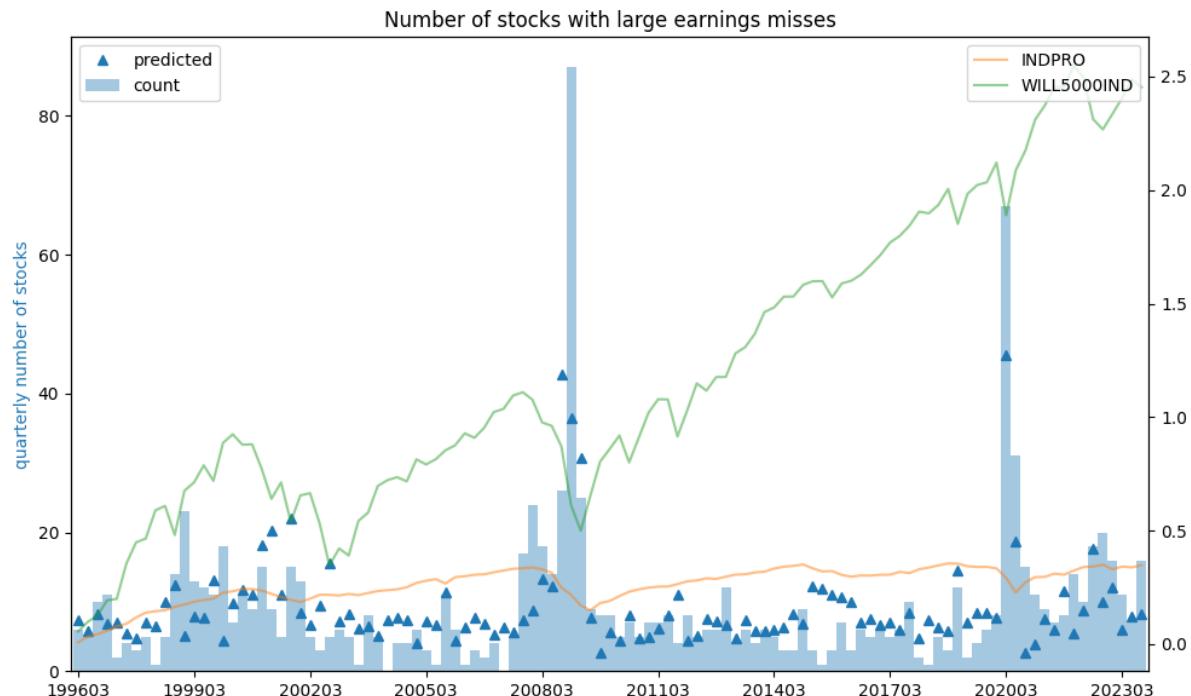
| Generalized Linear Model Regression Results | | | | | | |
|---|------------------|---------------------|----------|-------|--------|--------|
| ===== | | | | | | |
| Dep. Variable: | count | No. Observations: | 111 | | | |
| Model: | GLM | Df Residuals: | 108 | | | |
| Model Family: | Poisson | Df Model: | 2 | | | |
| Link Function: | Log | Scale: | 1.0000 | | | |
| Method: | IRLS | Log-Likelihood: | -431.05 | | | |
| Date: | Sat, 27 Apr 2024 | Deviance: | 451.81 | | | |
| Time: | 17:00:51 | Pearson chi2: | 523. | | | |
| No. Iterations: | 5 | Pseudo R-squ. (CS): | 0.9695 | | | |
| Covariance Type: | nonrobust | | | | | |
| ===== | | | | | | |
| | coef | std err | z | P> z | [0.025 | 0.975] |
| const | -5.3083 | 0.032 | -163.880 | 0.000 | -5.372 | -5.245 |

(continues on next page)

(continued from previous page)

| | | | | | | |
|-------------|----------|-------|---------|-------|---------|---------|
| INDPRO | -20.9026 | 1.478 | -14.143 | 0.000 | -23.799 | -18.006 |
| WILL5000IND | -3.1993 | 0.302 | -10.602 | 0.000 | -3.791 | -2.608 |
| ===== | | | | | | |

```
# 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['WILL5000IND'].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()
```



19.3 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.

19.4 Actuarial losses

Actuaries use the **frequency-severity** method for determining the expected number of claims that an insurer will receive during a given time period and how much the average claim will cost. Both are random (unlike the previous credit loss scenario where loss severity was assumed constant), and historical data may be used to model the expected number of claims and the expected cost of each claim, then multiplied together assuming they are independent.

<https://sites.google.com/site/zoubin019/teaching/math-5639-actuarial-loss-models>

19.4.1 Claim frequency

TODO

19.4.2 Claim severity

TODO

19.4.3 Aggregate loss

TODO

PRINCIPAL CUSTOMERS GRAPH

Forget about your competitors, just focus on your customers - Jack Ma

Concepts:

- Supply chain
- Graph properties

References;

- Ling Cen and Sudipto Dasgupta, 2021, The Economics and Finance of Customer-Supplier Relationships

```
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)
```

20.1 Supply chain

The interrelationships between upstream supplier firms and downstream customer firms, referred to as “supply-chain” relationships, constitute an important sources of linkages in the economy.

20.1.1 Principal customers

Regulation S-K and SFAS 131 require firms to disclose principal customers that account for more than 10% of a firm's total sales. An important feature of these disclosure, however, is that the suppliers are much smaller than their corporate principal customers. The primary reason is that customers are identified as "important" for the supplier in terms of the percentage of the supplier's sales.

```
# retrieve principal customers info
year = 2016
cust = sql.read_dataframe(f"select gvkey, cgvkey, stic, ctic, conm, cconnm "
                           f"  from customer "
                           f"  where srctype >= {year}0101 "
                           f"  and srctype <= {year}1231")
```

```
# construct Series to lookup company full name from ticker
lookup = pd.concat([Series(cust['conm'].values, cust['stic'].values),
                     Series(cust['cconnm'].values, cust['ctic'].values)]) \
    .drop_duplicates()
```

20.2 Graph properties

In a **graph**, **nodes** (also called vertices) represent entities or objects, while **edges** represent the connections or relationships between these entities. The **degree** of a graph is the number of edges that are directly connected to a particular node.

Density is the edges to nodes ratio: $\frac{2m}{n(n-1)}$ for undirected graphs, and $\frac{m}{n(n-1)}$ for directed graphs, where m is the number of edges and n is the number of nodes

A **simple** graph does not contain any self-loops (edges that connect a vertex to itself) and does not have multiple edges between the same pair of vertices.

A **subgraph** is a subset of nodes within a larger graph, and including only the edges that connect those selected nodes. A **component** of a graph is a subgraph where every node is connected to every other node in the subgraph by some path

Attributes of graphs:

- weighted versus unweighted: whether there are any numerical values associated with each relationship (edge)
- directed versus undirected: whether the relationships (edges) explicitly require a start and an end node
- cyclic versus acyclic: cyclic graphs contain at least one cycle, meaning there is a path that starts and ends at the same node. Acyclic graphs, on the other hand, do not have any cycles.
- connected versus disconnected: whether there is a path between any two nodes in graph, irrespective of distance
- Weakly connected components of a graph are sets of nodes where every pair of nodes has a path between them when ignoring the direction of edges. Strongly connected components are sets of nodes where every pair of nodes has a directed path between them.
- Weakly connected graphs have a path between every pair of nodes when considering the direction of edges, but ignoring their direction. Strongly connected graphs, on the other hand, have a directed path between every pair of nodes, considering the direction of edges.

```
# 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)
```

```
# 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 | 3052 |
| num_nodes | 1830 |
| num_selfloops | 11 |
| density | 0.000912 |
| is_weighted | False |
| is_weakly_connected | False |
| weakly_connected_components | 89 |
| size_largest_weak_component | 1628 |
| is_strongly_connected | False |
| strongly_connected_components | 1829 |
| size_largest_strong_component | 2 |

```
# remove self-loops, if any
DG.remove_edges_from(nx.selfloop_edges(DG))
```

20.2.1 Clustering coefficient

The clustering coefficient measures the degree to which nodes in a graph tend to cluster together.

- a **triplet** is three nodes that are connected by either two (open triplet) or three (closed triplet) undirected ties.
- a **triangle** comprises three closed triplets, one centered on each of the nodes
- the **clustering coefficient** of a node is the fraction of possible triangles through that node that exist $\frac{2\#\text{triangles}}{(\text{degree})(\text{degree}-1)}$
- **graph transitivity** is the fraction of all possible triangles present $3\frac{\#\text{triangles}}{\#\text{triads}}$ where possible triangles are identified by the number of **triads** (two edges with a shared vertex).

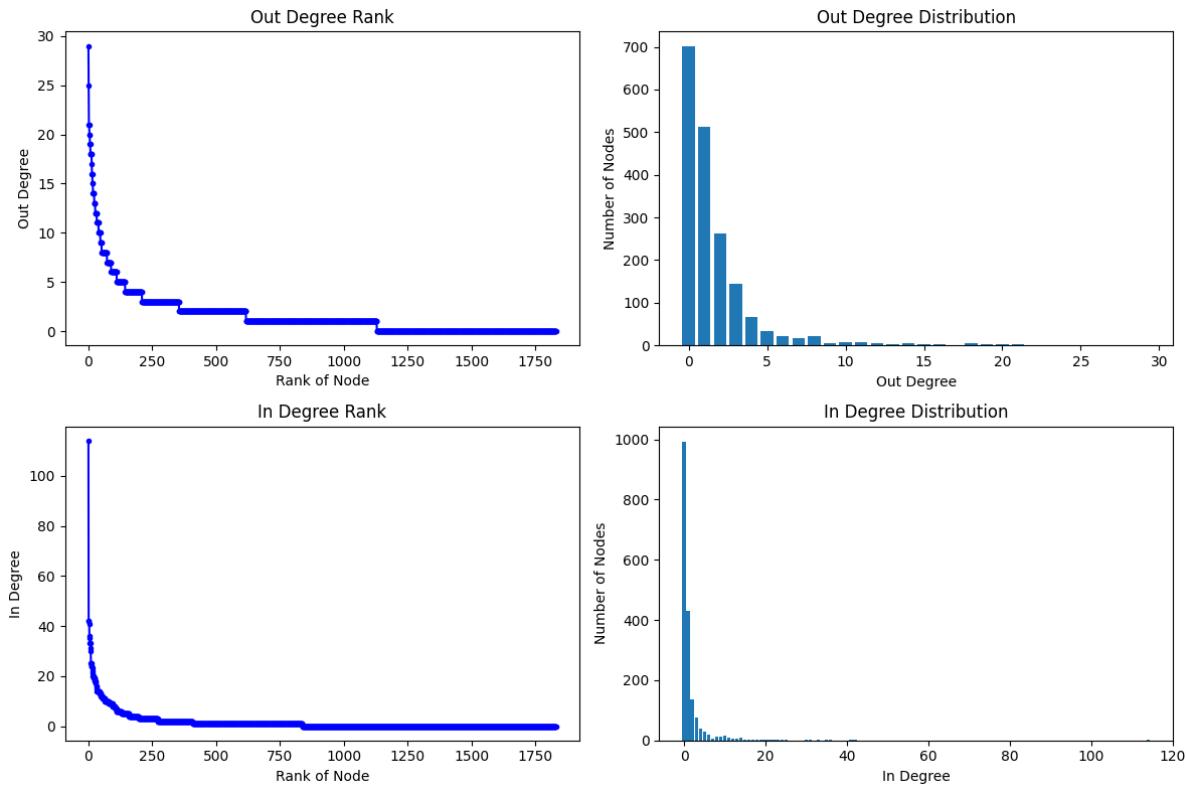
```
# as an undirected graph
G = nx.Graph(DG)
DataFrame({'transitivity': nx.transitivity(G),
           'average clustering': nx.average_clustering(G)},
           index=['clustering'])
```

| | transitivity | average clustering |
|------------|--------------|--------------------|
| clustering | 0.005751 | 0.009872 |

20.2.2 Degree analysis

In directed graphs, the **out degree** is the number of edges pointing out of the node, while the **in degree** is the number of edges pointing in.

```
# 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 ax, 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()
```



20.2.3 Ego network

Ego is some individual focal node. A network can have as many egos as it has nodes. **Neighborhood** is the collection of ego and all nodes to whom ego has a connection at some path length. The neighborhood is usually one-step; that is, it includes only ego and nodes that are directly adjacent. The neighborhood also includes all of the edges among all of the actors to whom ego has a direct connection.

```
# find node with greatest degree
ego, degree = max(G.degree(), key=lambda x: x[1])

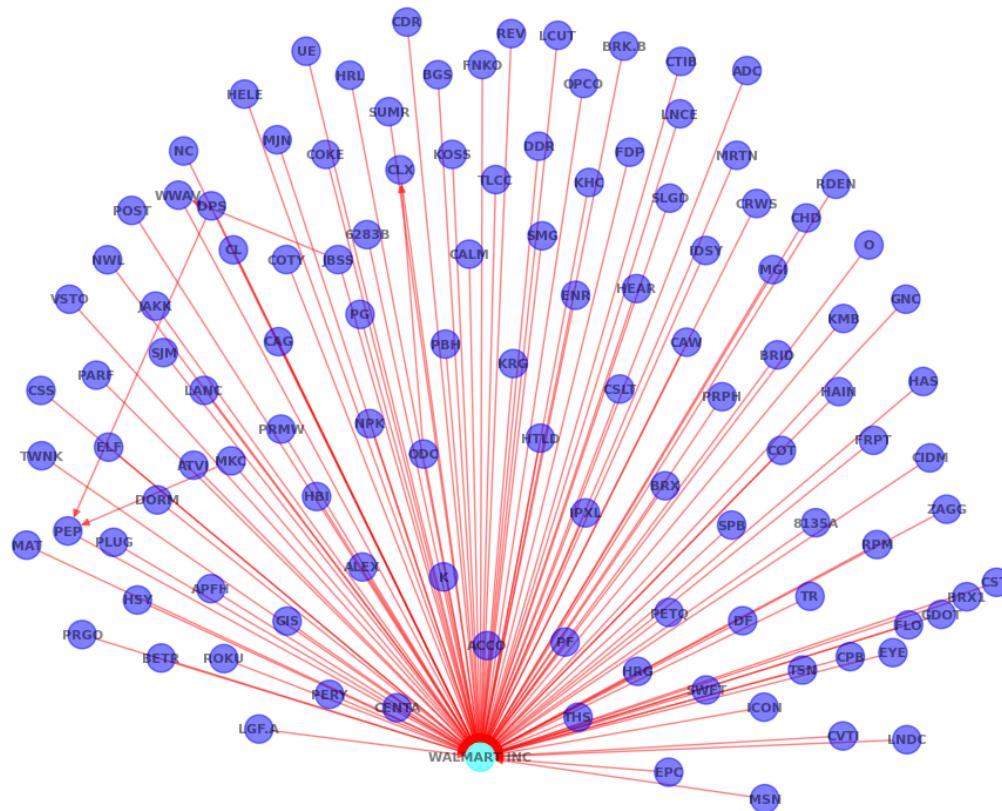
# build subgraph of ego and neighbors
all_neighbors = list(nx.all_neighbors(DG, ego)) # predecessors and successors
neighbors = list(nx.neighbors(DG, ego)) # successors only
ego_graph = DG.subgraph([ego] + all_neighbors).copy()
```

```
# Plot ego graph
node_color = (dict.fromkeys(all_neighbors, 'b')
              | dict.fromkeys(neighbors, 'g')
              | {ego: 'cyan'})

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,
           width=1, node_color=node_color, labels=labels, style='-' ,
           title=f"Ego network {ego}")
plt.show()
```

Ego network WMT



20.2.4 Path lengths

The **distance** or **path length** between two nodes is the shortest path (fewest number of edges) required to travel from one to the other.

The **eccentricity** of a node is the maximum distance from the to all other nodes in a graph. **Diameter** is the maximum eccentricity over all nodes in the graph.

```
# 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
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)}

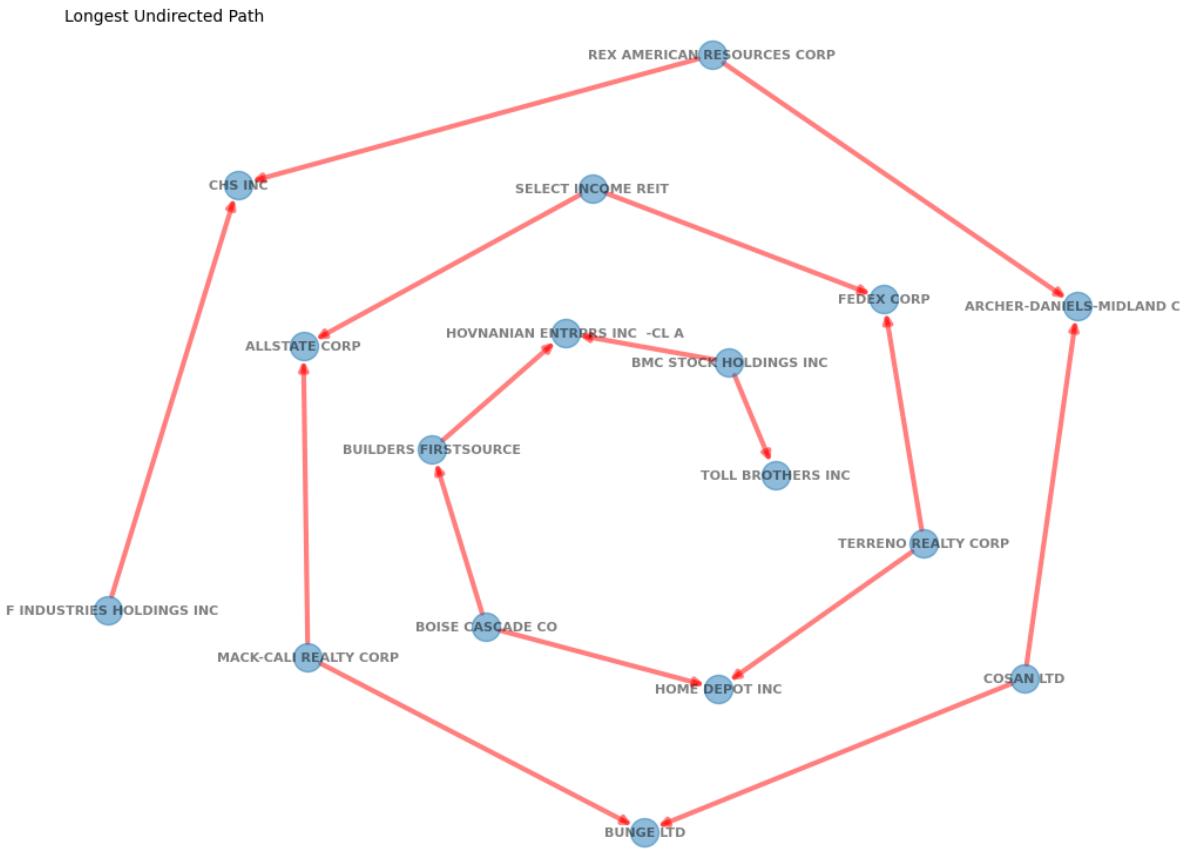
# Plot nodes in a spiral graph
labels = {ticker: lookup[ticker] for ticker in best_path}
graph_draw(DG.subgraph(best_path), figsize=(10, 8), node_size=300,
           width=3, labels=labels, style='-', pos=pos,
           title=f"Longest Undirected Path")

```

```

{'TOL': (5.0, 0.0),
 'BMCH': (3.2418138352088386, 5.048825908847379),
 'HOV': (-2.9130278558299967, 6.365081987779772),
 'BLDR': (-7.919939972803563, 1.1289600644789377),
 'BCC': (-5.882792587772507, -6.811222457771354),
 'HD': (2.8366218546322624, -9.589242746631385),
 'TRNO': (10.561873153154025, -3.0735704801881845),
 'FDX': (9.046827052119655, 7.883839184625469),
 'SIR': (-1.8915004395119759, 12.861657206103963),
 'ALL': (-12.755823666385478, 5.769658793384592),
 'CLI': (-12.586072936146786, -8.160316663340547),
 'BG': (0.07081116780881257, -15.999843304811256),
 'CZZ': (14.345517298452366, -9.121739606007393),
 'ADM': (16.33404206610353, 7.563006662879537),
 'REX': (2.5980071459488383, 18.821539758202537),
 '9919B': (-15.193758257176427, 13.005756803142337),
 'CF': (-20.110849086791077, -6.045969649966372) }

```



CHAPTER
TWENTYONE

COMMUNITY DETECTION

Realize that everything connects to everything else - Leonardo DaVinci

Concepts

- Industrial Taxonomy
- Community Detection

References

- Gerard Hoberg and Gordon Phillips, 2016, Text-Based Network Industries and Endogenous Product Differentiation. *Journal of Political Economy* 124 (5), 1423-1465.

```
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)
```

Last FamaFrench Date 2024-03-28 00:00:00

21.1 Industry taxonomy

Industry classification, or industry taxonomy, classifies companies into industrial groupings based on similar production processes, products, or behavior in financial markets.

21.1.1 TNIC

Hoberg and Phillips (2016) calculate firm-by-firm pairwise similarity scores by parsing the product descriptions from the firm 10Ks and forming word vectors for each firm, after basic screens to eliminate common words are applied. The words used only include nouns or proper nouns (see paper for details), and exclude geographic terms. The cosine similarity score is computed as a continuous measure of product similarity for every pair of firms in our sample in each year, producing a pairwise similarity matrix.

These product descriptions are legally required to be accurate, as Item 101 of Regulation S-K legally requires that firms describe the significant products they offer to the market, and these descriptions must also be updated and representative of the current fiscal year of the 10-K. Hence a a Text Based Industry Classification (TNIC) scheme, derived from the textual similarity of product descriptions, may better reflect firms' changes in the product market space over time.

Source: <https://hobergphillips.tuck.dartmouth.edu/industryclass.htm>

```
# 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 one 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 |
|----------|------|--------|--------|--------|
| 24633638 | 2021 | 1004 | 1210 | 0.0085 |
| 24633639 | 2021 | 1004 | 1823 | 0.0071 |
| 24633640 | 2021 | 1004 | 2285 | 0.0077 |
| 24633641 | 2021 | 1004 | 4091 | 0.0339 |
| 24633642 | 2021 | 1004 | 9698 | 0.0039 |
| ... | ... | ... | ... | ... |
| 25478201 | 2021 | 349972 | 322154 | 0.0444 |
| 25478202 | 2021 | 349972 | 331856 | 0.0169 |
| 25478203 | 2021 | 349972 | 332115 | 0.0214 |
| 25478204 | 2021 | 349972 | 345556 | 0.0781 |
| 25478205 | 2021 | 349972 | 347007 | 0.0711 |

(continues on next page)

(continued from previous page)

```
[844568 rows x 4 columns]
```

21.1.2 Industry classification

TODO

- SIC, 2-digit, 3-, 4-
- NAICS, 2, 3, 4, 5, 6

```
# 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  4218  4218  4216
```

```
# 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  4218  4218  4218
```

21.1.3 Sectors

An industry taxonomy may include detailed classifications, and groupings into broader sector

- Fama and French group 4-digit SIC codes into broader groups of 5, 10, 12, 17, 30, 38, 48, or 49 industry sectors.
- The Bureau of Economic Analysis groups 6-digit NAICS codes into a few dozen “summary” level groups, using definitions that were updated 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 ''
```

(continues on next page)

(continued from previous page)

```

fillna = 0 if scheme.startswith('sic') else ''

# load the sectoring class from SQL
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

```

```

4259 codes5
4259 codes10
4259 codes12
4259 codes17
4259 codes30
4259 codes38
4259 codes48
4259 codes49
4218 sic2
4218 sic3
3943 bea1947
3943 bea1963
3943 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 | |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| 345556 | 16069 | 2836 | 325414 | Hlth | Hlth | Hlth | Other | Hlth | Chems | |
| 345920 | 20194 | 3524 | 333112 | Manuf | Manuf | Manuf | Machn | FabPr | Machn | |
| 345980 | 20333 | 5961 | 455110 | Cnsmr | Shops | Shops | Rtail | Rtail | Rtail | |
| 347007 | 15533 | 2836 | 325414 | Hlth | Hlth | Hlth | Other | Hlth | Chems | |
| 349972 | 15642 | 2836 | 325414 | Hlth | Hlth | Hlth | Other | 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 | | | |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| 345556 | Drugs | Drugs | 28 | 283 | 325 | 325 | 325 | | | |
| 345920 | Mach | Mach | 35 | 352 | 333 | 333 | 333 | | | |
| 345980 | Rtail | Rtail | 59 | 596 | 44RT | 44RT | 4A0 | | | |
| 347007 | Drugs | Drugs | 28 | 283 | 325 | 325 | 325 | | | |
| 349972 | Drugs | Drugs | 28 | 283 | 325 | 325 | 325 | | | |

(continues on next page)

(continued from previous page)

```
[3943 rows x 16 columns]
```

```
# 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()
```

| | |
|------------------------|----------|
| | 2021 |
| transitivity | 0.874473 |
| average_clustering | 0.585098 |
| connected | False |
| connected_components | 8 |
| size_largest_component | 3908 |
| directed | False |
| weighted | True |
| negatively_weighted | False |
| edges | 405305 |
| nodes | 3925 |
| selfloops | 0 |
| density | 0.052631 |

21.2 Community structure

TODO

21.2.1 Measuring partitions

TODO

```
# 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)
```

(continues on next page)

(continued from previous page)

```
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 (2021)

| | communities | modularity | coverage | performance |
|---------|-------------|------------|----------|-------------|
| bea1947 | 40 | 0.337511 | 0.791471 | 0.928649 |
| bea1963 | 58 | 0.331096 | 0.748338 | 0.950073 |
| bea1997 | 61 | 0.331030 | 0.748153 | 0.950577 |
| codes10 | 10 | 0.340272 | 0.938264 | 0.850497 |
| codes12 | 12 | 0.340412 | 0.931380 | 0.879861 |
| codes17 | 17 | 0.278554 | 0.737459 | 0.788439 |
| codes30 | 30 | 0.339191 | 0.927052 | 0.900273 |
| codes38 | 36 | 0.339822 | 0.803928 | 0.893095 |
| codes48 | 48 | 0.337848 | 0.765192 | 0.946618 |
| codes49 | 49 | 0.337751 | 0.764032 | 0.952728 |
| codes5 | 5 | 0.341942 | 0.944133 | 0.819802 |
| sic2 | 69 | 0.333970 | 0.755767 | 0.944701 |
| sic3 | 236 | 0.294952 | 0.706283 | 0.959787 |

21.2.2 Detecting partitions

TODO

```
# 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 (2021)

| | communities | modularity | coverage | performance |
|---------|-------------|------------|----------|-------------|
| greedy | 26 | 0.347560 | 0.982830 | 0.507290 |
| label | 109 | 0.355460 | 0.990437 | 0.906120 |
| louvain | 19 | 0.363978 | 0.835415 | 0.823403 |

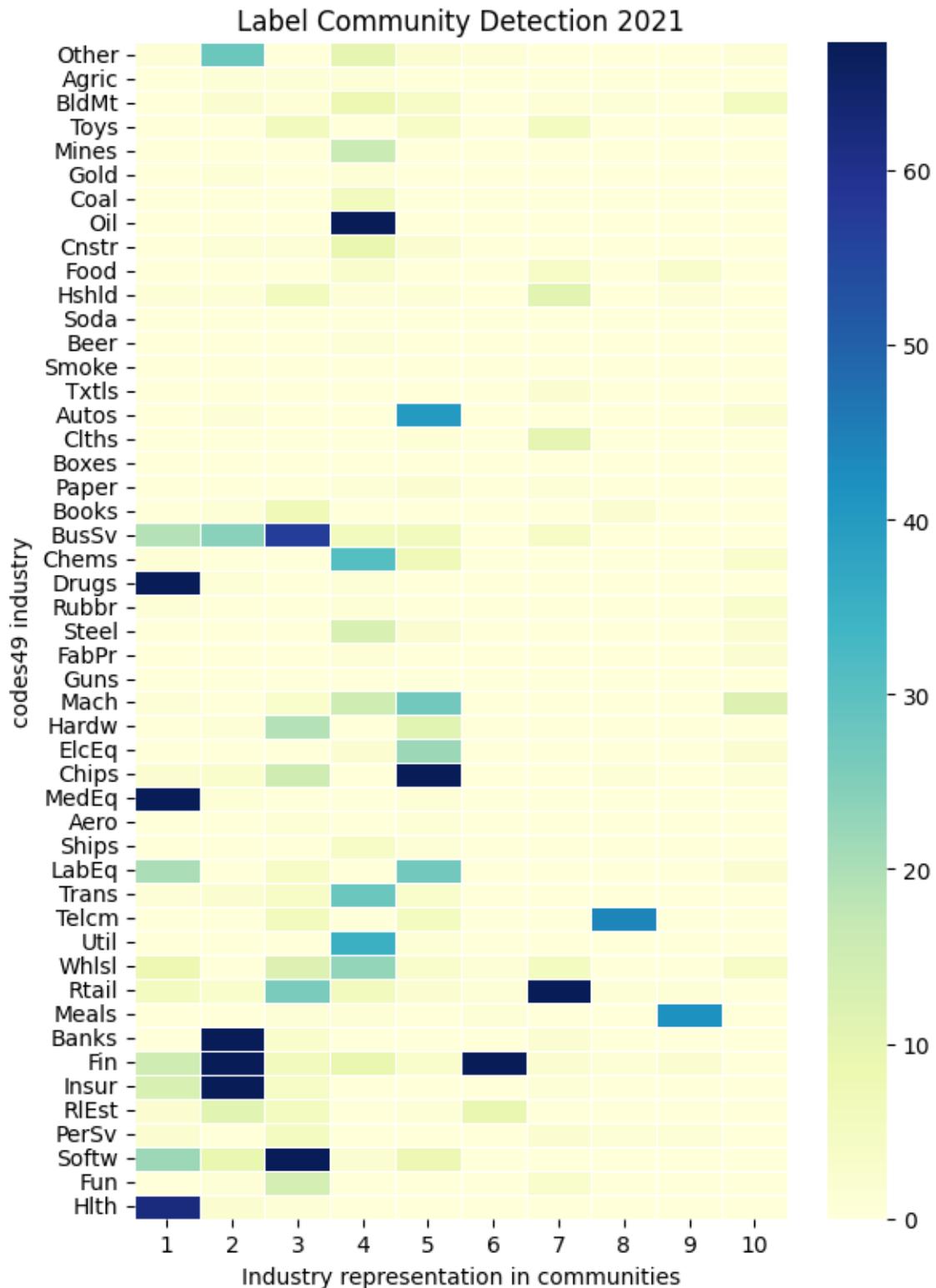
```

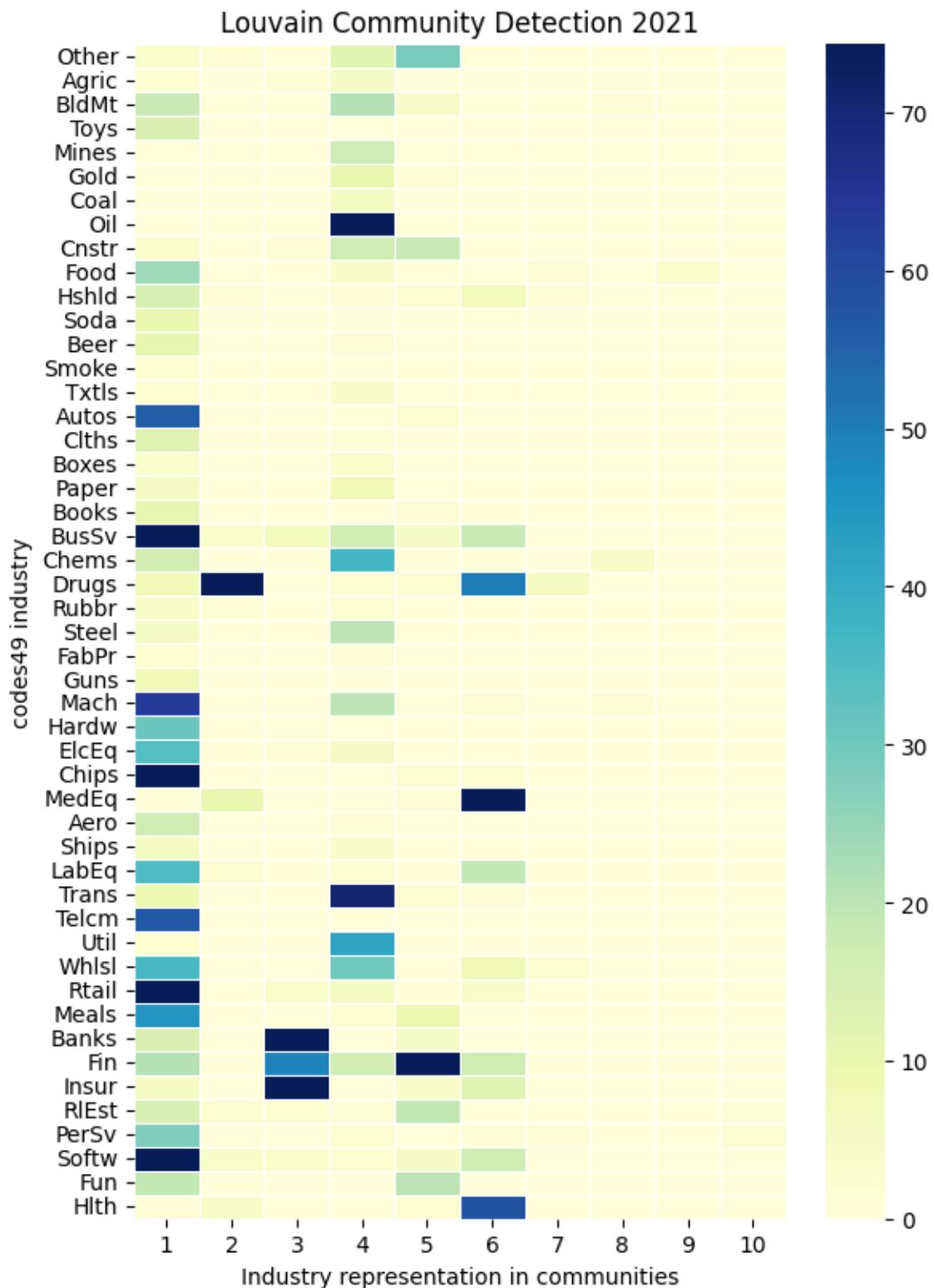
# Visualize Fama-French 49-industries in the detected communities
key = 'codes49'
for ifig, detection in enumerate(communitys.keys()):

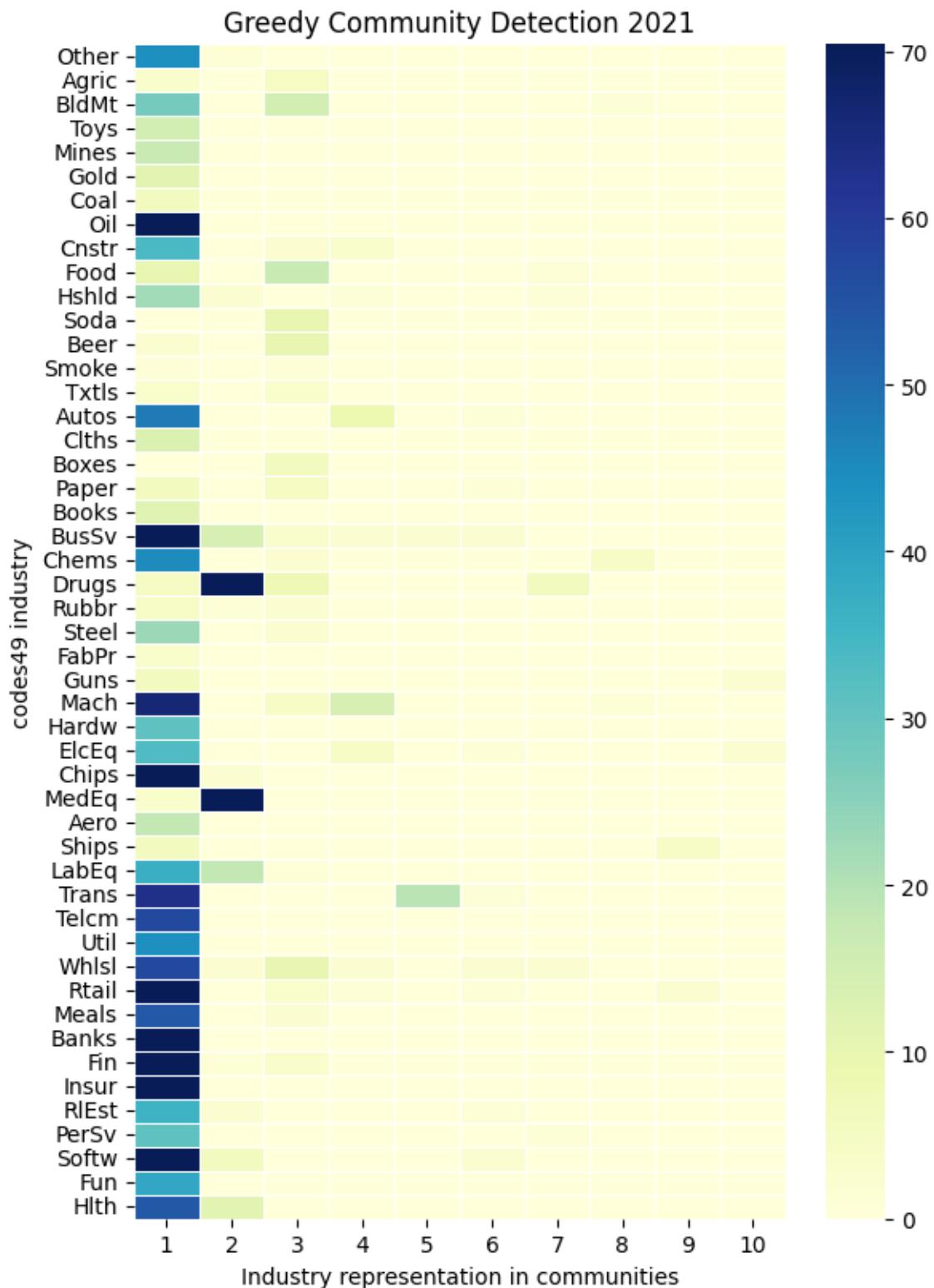
    # count industries represented in each partition
    industry = []
    communitys_sequence = sorted(communitys[detection], key=len, reverse=True)
    for i, community in enumerate(communitys_sequence):
        industry.append(nodes[key][list(community)].value_counts().rename(i+1))
    names = sectorings[key].sectors['name'].drop_duplicates(keep='first')
    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)

```







CHAPTER
TWENTYTWO

GRAPH CENTRALITY

And so we all matter - maybe less then a lot but always more than none - John Green

Concepts:

- BEA Input-Output Use Table
- Centrality

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>

```
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, 2022]
vintages = [1997, 1963, 1947]    # when sectoring schemes were revised
vintage = min(vintages)
```

22.1 Centrality measures

Measures of graph centrality quantify the importance of nodes within a graph.

- **Degree Centrality** measures the number of edges connected to a node.
- **Betweenness Centrality** quantifies the extent to which a node lies on the shortest paths between other nodes in the graph.
- **Closeness Centrality** measures how close a node is to all other nodes in the graph in terms of shortest path lengths.
- **Eigenvector Centrality** evaluates the centrality of a node based on the centrality of its neighbors. Nodes with higher eigenvector centrality are connected to other highly central nodes.
- **PageRank Centrality** was originally developed for ranking web pages by importance. It can be viewed as a random walk process on the graph, representing the long-term likelihood of visitation of each node by randomly following edges.
- A **Hub** is a node that has many out-edge to other nodes. It acts as a facilitator of flows other parts of the graph
- An **Authority** is a node that has many in-edges from other nodes. It can be regarded as highly influential in that many other nodes reference it.

```
# 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)
```

(continues on next page)

(continued from previous page)

```
out['closeness'] = nx.closeness_centrality(G)
return out
```

22.2 BEA Input-Output Use Tables

Input-output analysis is a type of economic model used to study the interdependencies between different sectors of an economy. It typically involves constructing an input-output table that records the flows of goods and services between sectors. This quantifies the relationships between inputs and outputs among various sectors, showing how changes in one sector can affect others.

```
# Read IOUse tables from BEA website
ioUses = {year: bea.read_iouUse(year=year, vintage=vintage) for year in years}
```

```
## Set directed edges with tail on user (table column) --> head on maker (row)
## 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
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')\
        .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
```

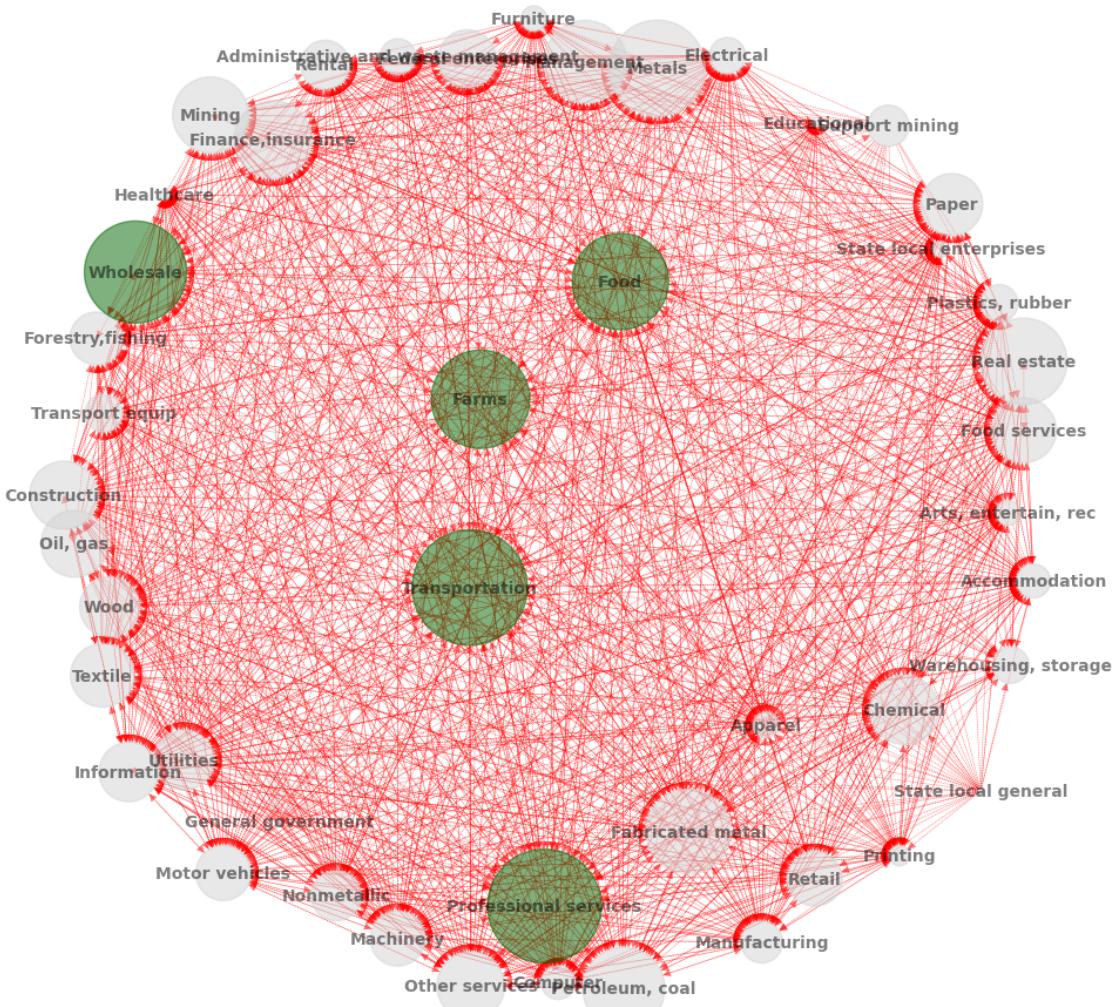
(continues on next page)

(continued from previous page)

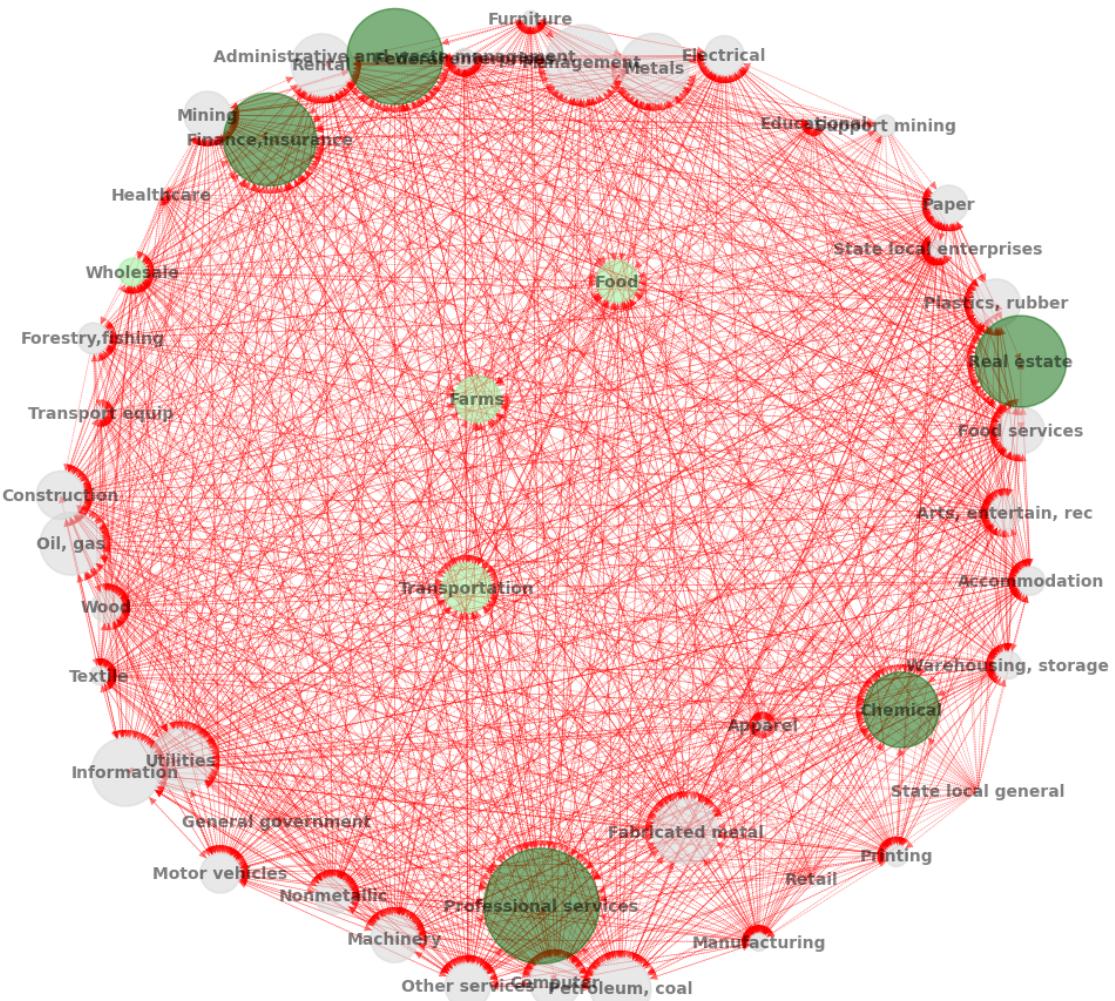
```
centrality = DataFrame(nodes_centrality(G))      # compute centrality metrics
master = master.join(centrality, how='left')
master['bea'] = BEA.short_desc[master.index].to_list()

# 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}-sectors)")
```

Top 5 Nodes By Pagerank in 1947 (1947-sectors)



Top 5 Nodes By Pagerank in 2022 (1947-sectors)



```
# Compare 1947 and 1997 sectors (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
```

(continues on next page)

(continued from previous page)

| | |
|----------------------------------|---------------------------------------|
| 444000 | Building material and garden |
| ↳ equipment and supplies dealers | |
| 445000 | |
| ↳ Food and beverage stores | |
| 446000 | Health |
| ↳ and personal care stores | |
| 447000 | |
| ↳ Gasoline stations | |
| 448000 | Clothing and |
| ↳ clothing accessories stores | |
| 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 | Transit and ground |
| ↳ passenger transportation | |
| 486000 | |
| ↳ Pipeline transportation | |
| 487000 | Scenic and sightseeing transportation |
| ↳ and support activities | |
| 488000 | Scenic and sightseeing transportation |
| ↳ and support activities | |
| 492000 | |
| ↳ Couriers and messengers | |
| 511000 | Publishing industries, except |
| ↳ internet (includes software) | |
| 511110 | |
| ↳ Newspaper publishers | |
| 511120 | |
| ↳ Periodical publishers | |
| 511130 | |
| ↳ Book publishers | |
| 511140 | Directory, mailing list, |
| ↳ and other publishers | |
| 511190 | Directory, mailing list, |
| ↳ and other publishers | |
| 511210 | |
| ↳ Software publishers | |
| 512000 | Motion picture and |
| ↳ sound recording industries | |
| 512100 | Motion |
| ↳ picture and video industries | |
| 512200 | |
| ↳ Sound recording industries | |

(continues on next page)

(continued from previous page)

| | |
|--------------------------------|--|
| 513000 | Broadcasting |
| ↳ 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 | |
| 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 | |

(continues on next page)

(continued from previous page)

| | |
|---------------------------------------|---|
| 541511 | Custom |
| ↳ 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 | |

(continues on next page)

(continued from previous page)

621400
↳ outpatient care centers
621500
↳ diagnostic laboratories
621600
↳ home health care services
621900
↳ ambulatory health care services
622000
↳ hospitals
623000
↳ residential care facilities
623100
↳ community care facilities
623200 Residential mental health, substance abuse, and other
↳ residential care facilities
623300
↳ 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
↳ and related activities
711100
↳ performing arts companies
711200
↳ spectator sports
711300 Promoters of performing arts and sports and
↳ agents for public figures
711500
↳ writers, and performers
712000 Museums, historical
↳ sites, zoos, and parks
713000
↳ 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

(continues on next page)

(continued from previous page)

| | |
|--------|-------------------------------------|
| 446000 | RETAIL TRADE |
| 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 |

(continues on next page)

(continued from previous page)

| | |
|--------|-------------------------------------|
| 541513 | PROFESSIONAL AND TECHNICAL SERVICES |
| 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 |

(continues on next page)

(continued from previous page)

| | |
|--------|--|
| 444000 | Other retail |
| 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 |

(continues on next page)

(continued from previous page)

| | |
|--------|--|
| 541511 | Computer systems design and related services |
| 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 |

```
# Display node centrality measures from latest year
ioUse = ioUses[max(years)]
data = ioUse[(~ioUse['rowcode']).str.startswith(drop) &
```

(continues on next page)

(continued from previous page)

```
~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 | |
|-------------------------------|----------|
| 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 | 1694 |
| nodes | 47 |
| selfloops | 0 |
| density | 0.783534 |

```
# 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 2022

| | user | maker | in_degree | out_degree | hub | authority | \ |
|-------|---------|--------|-----------|------------|-------|-----------|---|
| 111CA | 184284 | 451556 | 0.478 | 0.783 | 0.006 | 0.009 | |
| 113FF | 9386 | 91911 | 0.543 | 0.652 | 0.001 | 0.002 | |
| 211 | 249074 | 617352 | 0.674 | 0.674 | 0.012 | 0.010 | |
| 212 | 48300 | 122150 | 0.870 | 0.804 | 0.002 | 0.003 | |
| 213 | 26920 | 23522 | 0.087 | 0.652 | 0.002 | 0.000 | |
| 22 | 203222 | 473707 | 1.000 | 0.717 | 0.014 | 0.030 | |
| 23 | 1039108 | 367652 | 1.000 | 0.739 | 0.036 | 0.027 | |
| 311FT | 603053 | 372818 | 0.630 | 0.804 | 0.013 | 0.022 | |
| 313TT | 24087 | 60100 | 0.870 | 0.761 | 0.001 | 0.003 | |

(continues on next page)

(continued from previous page)

| | | | | | | |
|--------|---------|---------|-------|-------|-------|--------|
| 315AL | 6468 | 20431 | 0.543 | 0.674 | 0.000 | 0.001 |
| 321 | 61185 | 197254 | 0.957 | 0.804 | 0.002 | 0.011 |
| 322 | 75976 | 181304 | 0.978 | 0.739 | 0.003 | 0.008 |
| 323 | 43001 | 86246 | 0.739 | 0.804 | 0.002 | 0.008 |
| 324 | 675113 | 669702 | 1.000 | 0.761 | 0.012 | 0.037 |
| 325 | 206855 | 683053 | 1.000 | 0.804 | 0.009 | 0.033 |
| 326 | 183307 | 367073 | 1.000 | 0.783 | 0.007 | 0.019 |
| 327 | 60399 | 234086 | 0.957 | 0.761 | 0.002 | 0.011 |
| 331 | 71633 | 361595 | 0.891 | 0.717 | 0.003 | 0.005 |
| 332 | 210277 | 513487 | 1.000 | 0.761 | 0.006 | 0.019 |
| 333 | 213396 | 249624 | 1.000 | 0.761 | 0.006 | 0.009 |
| 334 | 58016 | 363419 | 0.978 | 0.717 | 0.004 | 0.021 |
| 335 | 76638 | 206688 | 1.000 | 0.717 | 0.002 | 0.008 |
| 3361MV | 336886 | 195265 | 0.978 | 0.783 | 0.009 | 0.010 |
| 3364OT | 101642 | 79386 | 0.391 | 0.717 | 0.004 | 0.003 |
| 337 | 46531 | 59315 | 0.565 | 0.717 | 0.001 | 0.004 |
| 339 | 70223 | 128923 | 0.935 | 0.804 | 0.003 | 0.012 |
| 42 | 1188638 | 71846 | 0.848 | 0.848 | 0.093 | 0.004 |
| 44RT | 1080120 | 0 | 0.000 | 0.913 | 0.087 | 0.000 |
| 48 | 553698 | 390771 | 1.000 | 0.848 | 0.033 | 0.033 |
| 493 | 62616 | 184456 | 0.913 | 0.696 | 0.004 | 0.017 |
| 51 | 734393 | 638594 | 1.000 | 0.848 | 0.067 | 0.044 |
| 52 | 526941 | 1102224 | 1.000 | 0.674 | 0.052 | 0.087 |
| 531 | 1067551 | 1352230 | 1.000 | 0.739 | 0.087 | 0.100 |
| 532RL | 239390 | 449819 | 1.000 | 0.717 | 0.019 | 0.028 |
| 54 | 785265 | 1958497 | 1.000 | 0.913 | 0.049 | 0.145 |
| 55 | 272563 | 726281 | 0.891 | 0.870 | 0.026 | 0.050 |
| 56 | 544761 | 1207073 | 1.000 | 0.913 | 0.039 | 0.096 |
| 61 | 113082 | 20555 | 0.348 | 0.848 | 0.009 | 0.001 |
| 62 | 1091749 | 14862 | 0.130 | 0.848 | 0.081 | 0.001 |
| 71 | 154598 | 132038 | 1.000 | 0.913 | 0.014 | 0.010 |
| 721 | 105815 | 76041 | 0.978 | 0.826 | 0.006 | 0.006 |
| 722 | 562141 | 284931 | 1.000 | 0.848 | 0.035 | 0.024 |
| 81 | 311845 | 304730 | 1.000 | 0.913 | 0.024 | 0.022 |
| GFE | 38866 | 76333 | 0.761 | 0.739 | 0.002 | 0.007 |
| GFG | 459174 | 0 | 0.000 | 0.848 | 0.030 | -0.000 |
| GSLE | 276831 | 37227 | 0.891 | 0.783 | 0.016 | 0.003 |
| GSLG | 1151110 | 0 | 0.000 | 0.870 | 0.067 | -0.000 |

| | eigenvector | pagerank | betweenness | closeness | \ |
|-------|-------------|----------|-------------|-----------|---|
| 111CA | 0.051 | 0.019 | 0.001 | 0.657 | |
| 113FF | 0.010 | 0.008 | 0.001 | 0.687 | |
| 211 | 0.124 | 0.034 | 0.004 | 0.754 | |
| 212 | 0.023 | 0.019 | 0.008 | 0.885 | |
| 213 | 0.005 | 0.004 | 0.000 | 0.523 | |
| 22 | 0.102 | 0.033 | 0.003 | 1.000 | |
| 23 | 0.117 | 0.020 | 0.003 | 1.000 | |
| 311FT | 0.046 | 0.015 | 0.001 | 0.730 | |
| 313TT | 0.005 | 0.003 | 0.002 | 0.885 | |
| 315AL | 0.002 | 0.001 | 0.000 | 0.687 | |
| 321 | 0.043 | 0.008 | 0.003 | 0.958 | |
| 322 | 0.034 | 0.013 | 0.002 | 0.979 | |
| 323 | 0.024 | 0.004 | 0.003 | 0.793 | |
| 324 | 0.104 | 0.032 | 0.002 | 1.000 | |
| 325 | 0.129 | 0.049 | 0.003 | 1.000 | |
| 326 | 0.066 | 0.020 | 0.003 | 1.000 | |

(continues on next page)

(continued from previous page)

| | | | | |
|--------|-------|-------|-------|-------|
| 327 | 0.055 | 0.012 | 0.002 | 0.958 |
| 331 | 0.060 | 0.042 | 0.001 | 0.902 |
| 332 | 0.110 | 0.036 | 0.002 | 1.000 |
| 333 | 0.058 | 0.019 | 0.003 | 1.000 |
| 334 | 0.088 | 0.022 | 0.001 | 0.979 |
| 335 | 0.052 | 0.013 | 0.002 | 1.000 |
| 3361MV | 0.044 | 0.014 | 0.003 | 0.979 |
| 3364OT | 0.004 | 0.001 | 0.000 | 0.622 |
| 337 | 0.017 | 0.002 | 0.001 | 0.697 |
| 339 | 0.012 | 0.003 | 0.003 | 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 | 0.006 | 1.000 |
| 493 | 0.025 | 0.007 | 0.001 | 0.920 |
| 51 | 0.222 | 0.040 | 0.005 | 1.000 |
| 52 | 0.397 | 0.074 | 0.002 | 1.000 |
| 531 | 0.345 | 0.072 | 0.003 | 1.000 |
| 532RL | 0.141 | 0.033 | 0.002 | 1.000 |
| 54 | 0.516 | 0.114 | 0.017 | 1.000 |
| 55 | 0.167 | 0.047 | 0.010 | 0.902 |
| 56 | 0.455 | 0.079 | 0.011 | 1.000 |
| 61 | 0.001 | 0.001 | 0.001 | 0.605 |
| 62 | 0.000 | 0.000 | 0.000 | 0.535 |
| 71 | 0.058 | 0.009 | 0.011 | 1.000 |
| 721 | 0.043 | 0.007 | 0.004 | 0.979 |
| 722 | 0.116 | 0.019 | 0.004 | 1.000 |
| 81 | 0.088 | 0.018 | 0.011 | 1.000 |
| GFE | 0.018 | 0.004 | 0.004 | 0.807 |
| GFG | 0.000 | 0.000 | 0.000 | 0.000 |
| GSLE | 0.010 | 0.002 | 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 |
| 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 |

(continues on next page)

(continued from previous page)

| | |
|--------|-------------------------------------|
| 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 |

CHAPTER
TWENTYTHREE

LINK PREDICTION

A hidden connection is stronger than an obvious one - Heraclitus

Concepts:

- Link prediction
- Accuracy

```
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)
```

Retrieve TNIC schemes from Hoberg and Phillips website

Source: <https://hobergphillips.tuck.dartmouth.edu/industryclass.htm>

```
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+')
```

(continues on next page)

(continued from previous page)

```
for k,v in tnic_data.items():
    print(k, v.shape)
```

```
tnic2 (50402140, 4)
tnic3 (25657918, 4)
```

```
# extract last year of both tnic schemes, merge in permno, and require in univ
year = 2021
capsize = 10 # large cap (large than NYSE median)
univ = crsp.get_universe(bd.endyr(year))
univ = univ[univ['decile'] <= capsizer]
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)]
```

```
# 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)
    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)
```

(continues on next page)

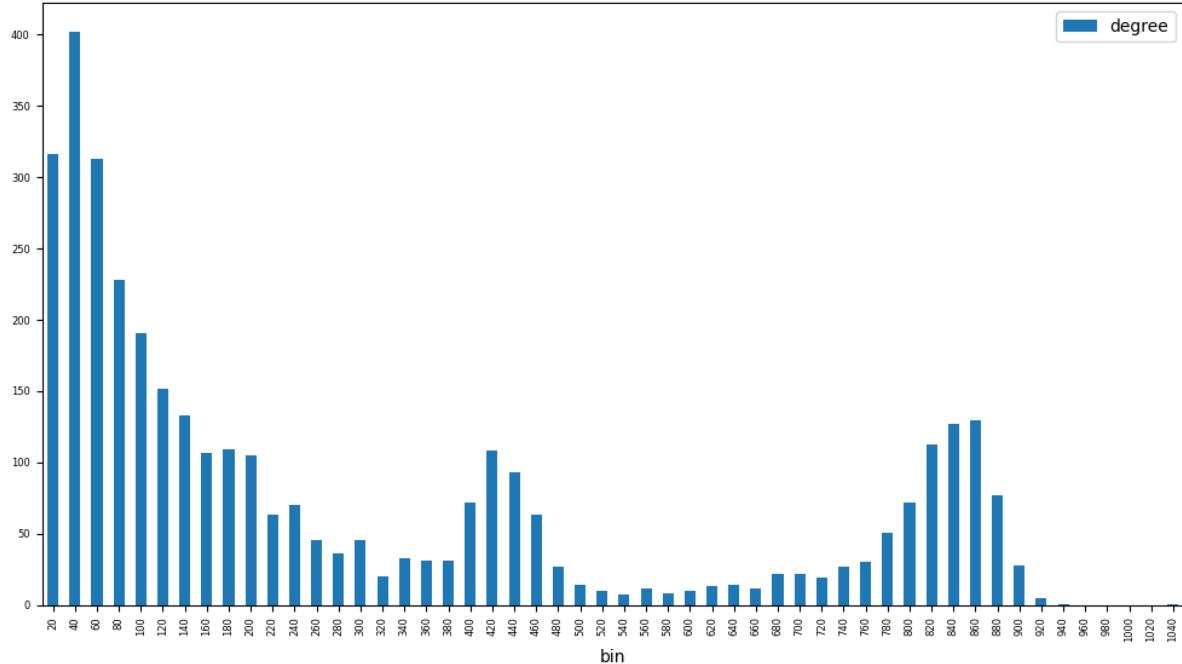
(continued from previous page)

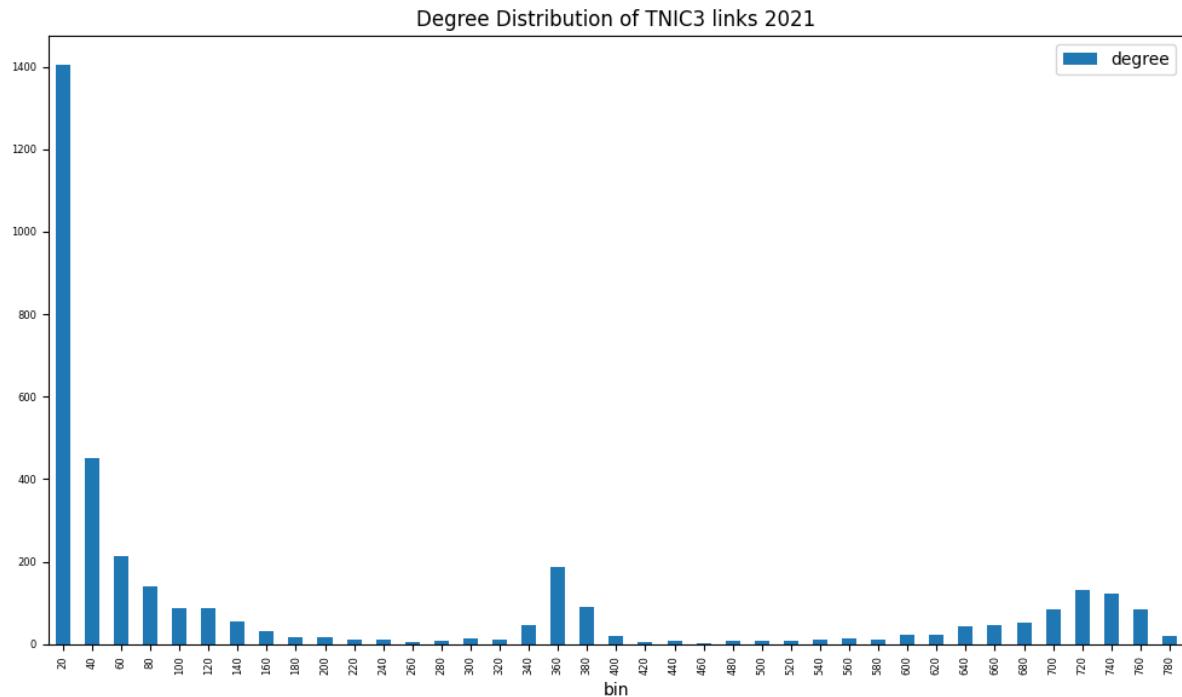
```
degree.groupby('bin').sum().plot(kind='bar', ax=ax, fontsize=6)
ax.set_title(f'Degree Distribution of {scheme.upper()} links {year}')
plt.tight_layout()

G[scheme] = g
```

```
tnic2 nodes = 3620 edges = 1051648
tnic3 nodes = 3620 edges = 690372
```

Degree Distribution of TNIC2 links 2021





```
print(f"Graph properties of TNIC schemes {year}")
DataFrame(results)
```

Graph properties of TNIC schemes 2021

| | tnic2 | tnic3 |
|------------------------|----------|----------|
| transitivity | 0.839705 | 0.880754 |
| average_clustering | 0.596602 | 0.581718 |
| connected | True | False |
| connected_components | 1 | 24 |
| size_largest_component | 3620 | 3587 |
| directed | False | False |
| weighted | True | True |
| negatively_weighted | False | False |
| edges | 525824 | 345186 |
| nodes | 3620 | 3620 |
| selfloops | 0 | 0 |
| density | 0.080274 | 0.052697 |

23.1 Link prediction algorithms

TODO

The goal of link prediction is to predict missing or future connections between nodes in a network. Given a partially observed network, link prediction infers which links are most likely to be added or missing based on the observed connections and the structure of the network.

- jaccard_coefficient
- resource_allocation

- adamic_adar
- preferential_attachment

```
# 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'])
```

23.2 Accuracy metrics

TODO

23.2.1 ROC Curve

- AUC

23.2.2 Precision Recall

FP, FN, TP, TN

- precision
- recall
- accuracy
- confusion matrix
- F1 score: when imbalanced class distribution, and looking for a balanced measure between precision and recall (Type I and Type II errors). Harmonic mean of the precision and recall

$$2 \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} = \frac{TP}{TP + (FP + FN)/2}$$

```
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)
```

(continues on next page)

(continued from previous page)

```
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)

    # 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 |
| False | 0.99 | 0.99 | 0.99 | 6024566 |
| True | 0.63 | 0.63 | 0.63 | 180638 |
| accuracy | | | 0.98 | 6205204 |
| macro avg | 0.81 | 0.81 | 0.81 | 6205204 |
| weighted avg | 0.98 | 0.98 | 0.98 | 6205204 |

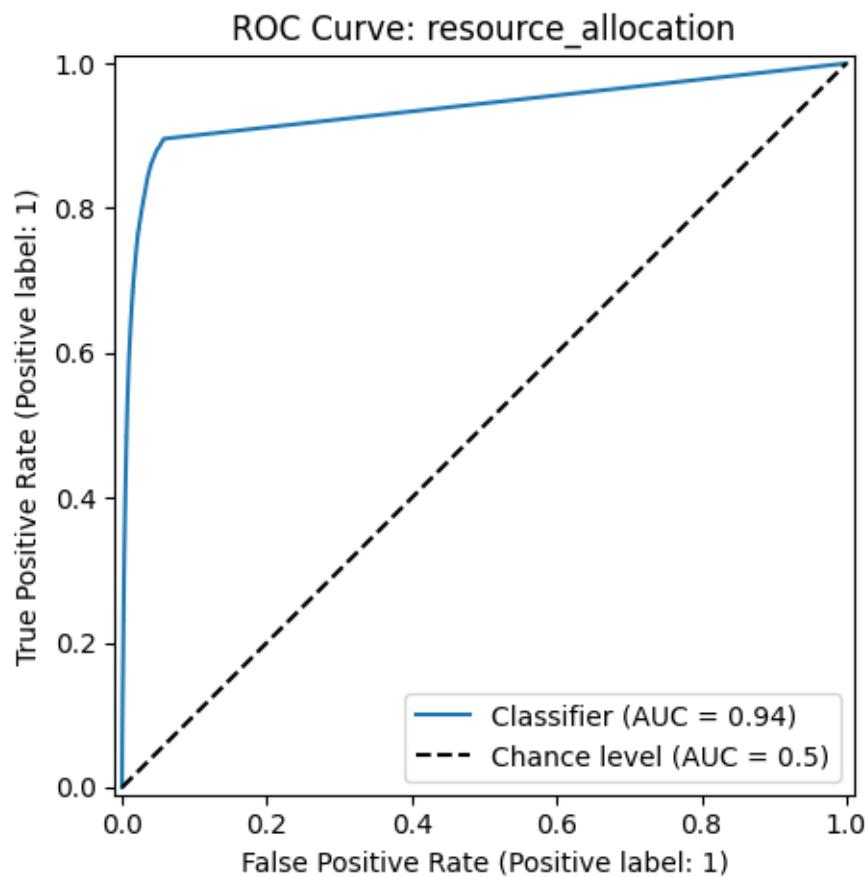
| Classification Report: jaccard_coefficient | | | | |
|--|-----------|--------|----------|---------|
| | precision | recall | f1-score | support |
| False | 0.99 | 0.99 | 0.99 | 6024566 |
| True | 0.60 | 0.60 | 0.60 | 180638 |
| accuracy | | | 0.98 | 6205204 |
| macro avg | 0.79 | 0.79 | 0.79 | 6205204 |
| weighted avg | 0.98 | 0.98 | 0.98 | 6205204 |

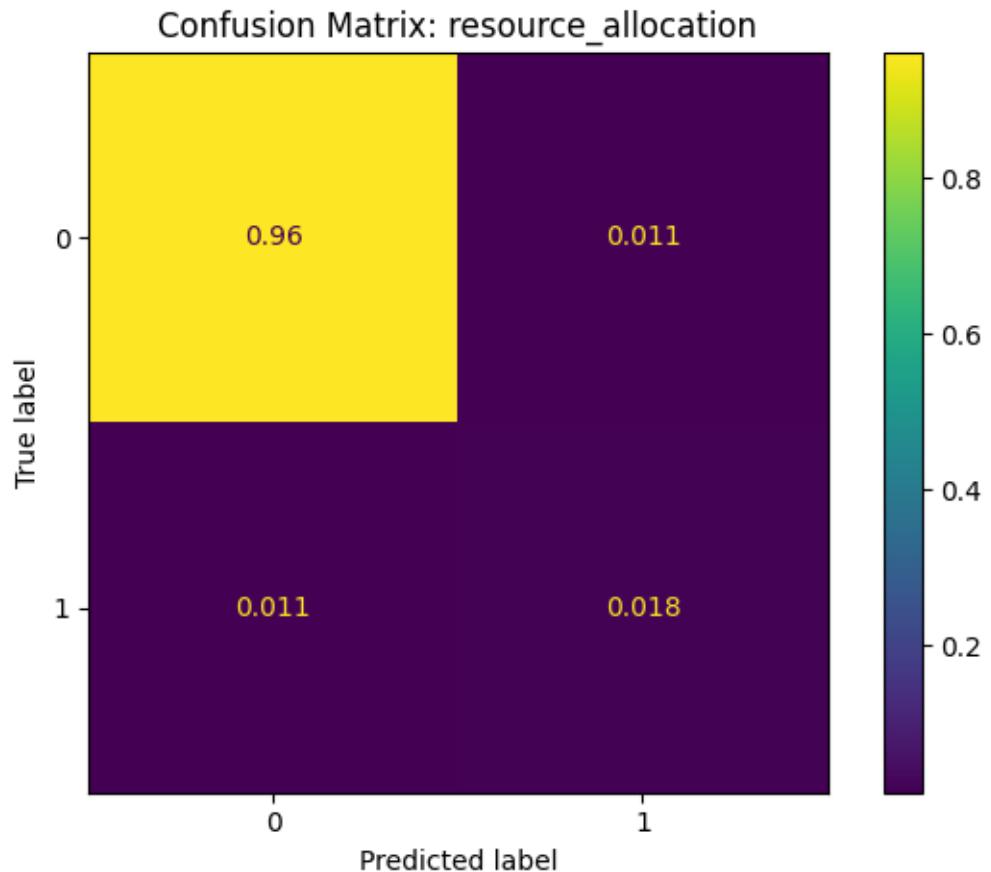
| Classification Report: adamic_adar | | | | |
|------------------------------------|-----------|--------|----------|---------|
| | precision | recall | f1-score | support |
| False | 0.99 | 0.99 | 0.99 | 6024566 |

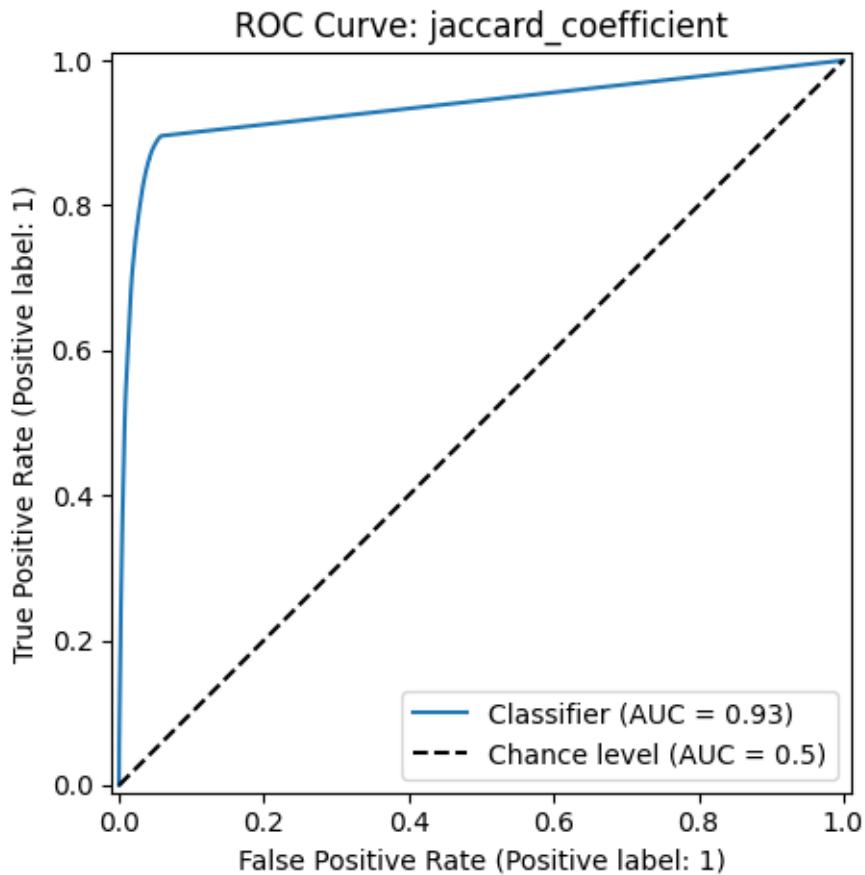
(continues on next page)

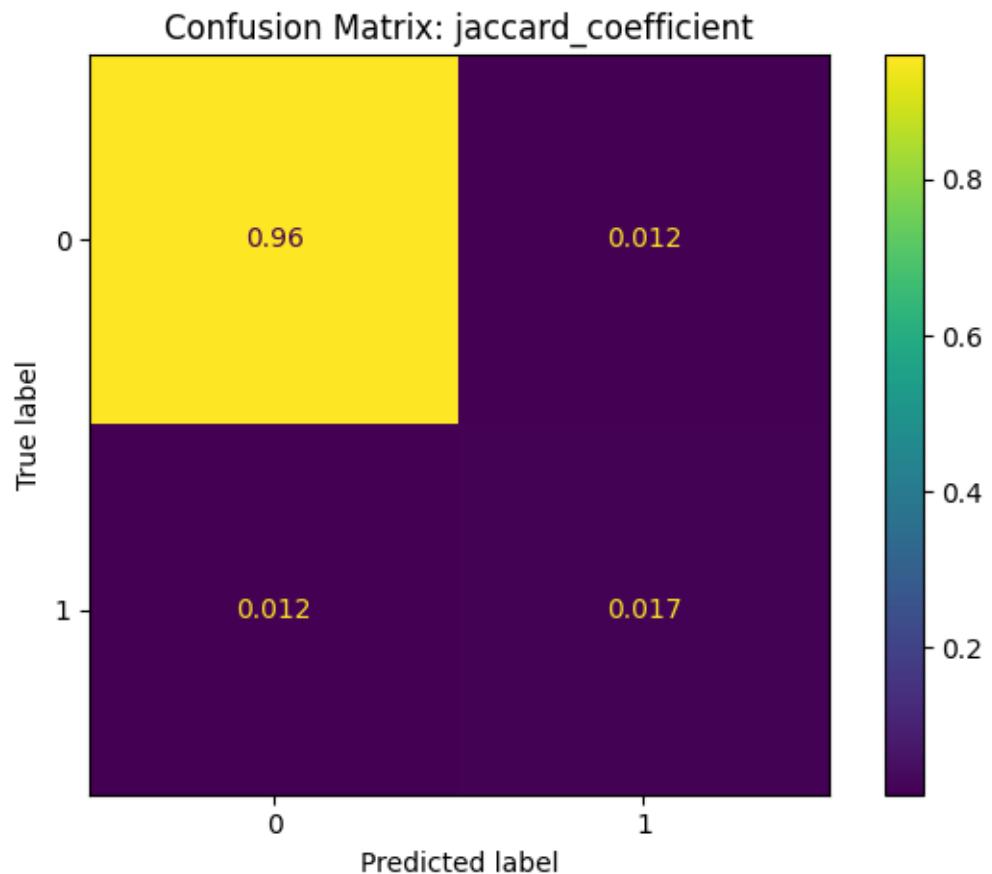
(continued from previous page)

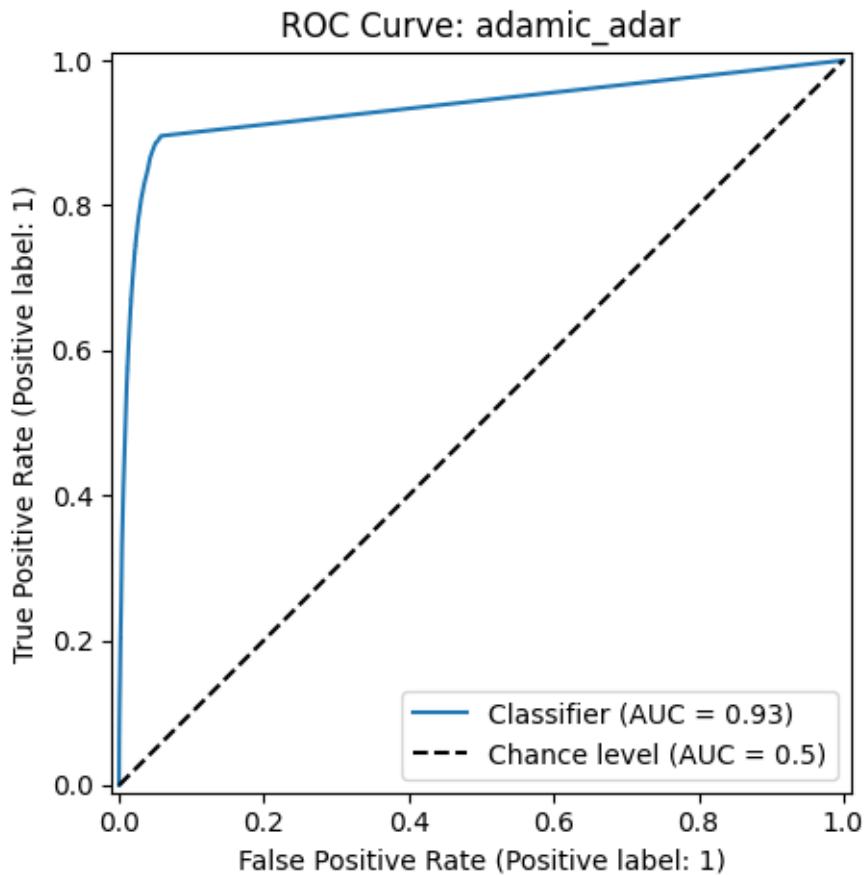
| | | | | |
|--|-----------|--------|----------|---------|
| True | 0.59 | 0.59 | 0.59 | 180638 |
| accuracy | | | 0.98 | 6205204 |
| macro avg | 0.79 | 0.79 | 0.79 | 6205204 |
| weighted avg | 0.98 | 0.98 | 0.98 | 6205204 |
| Classification Report: preferential_attachment | | | | |
| | precision | recall | f1-score | support |
| False | 0.97 | 0.97 | 0.97 | 6024566 |
| True | 0.10 | 0.10 | 0.10 | 180638 |
| accuracy | | | 0.95 | 6205204 |
| macro avg | 0.54 | 0.54 | 0.54 | 6205204 |
| weighted avg | 0.95 | 0.95 | 0.95 | 6205204 |

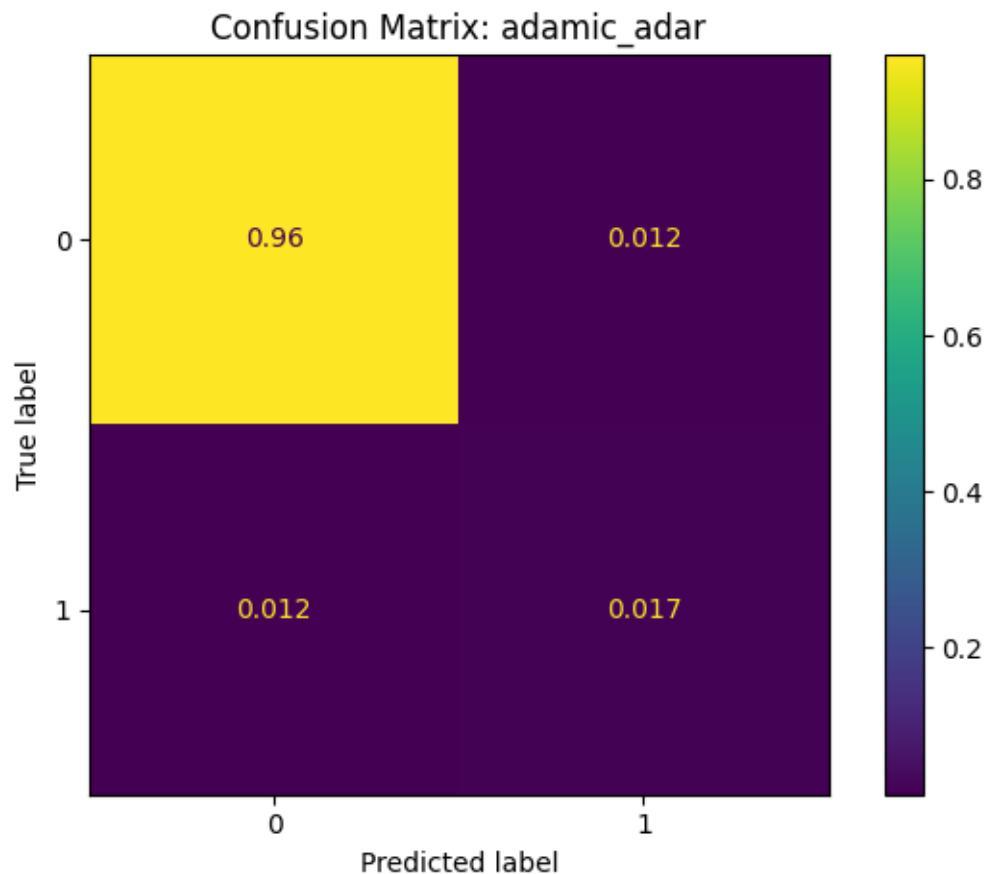


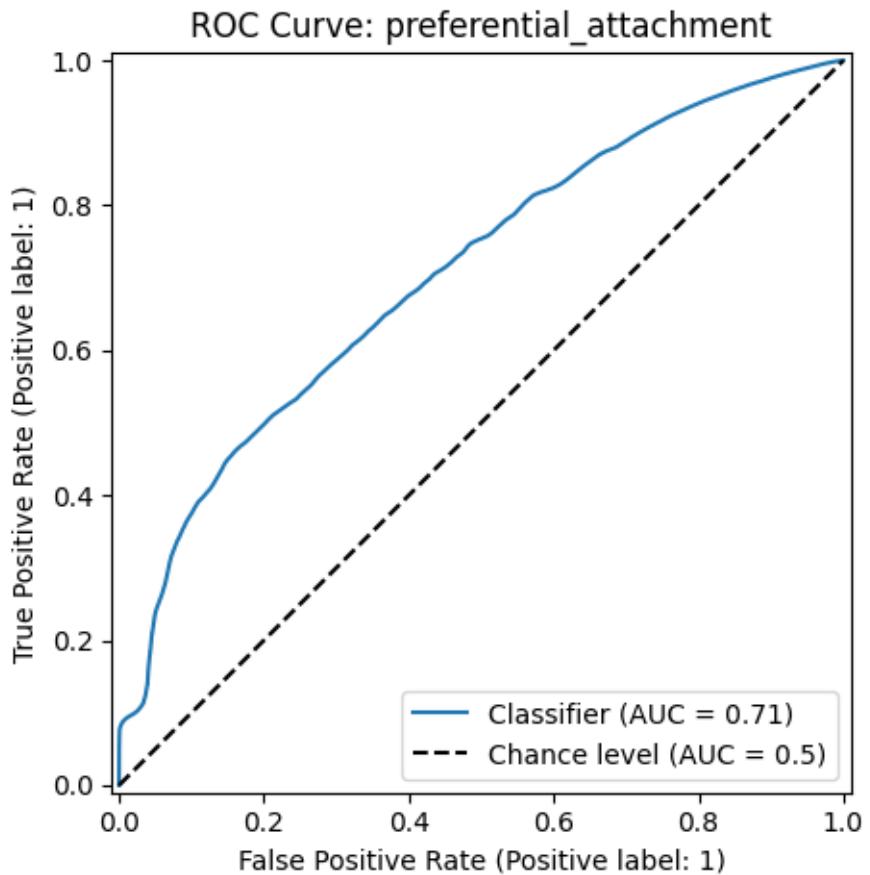


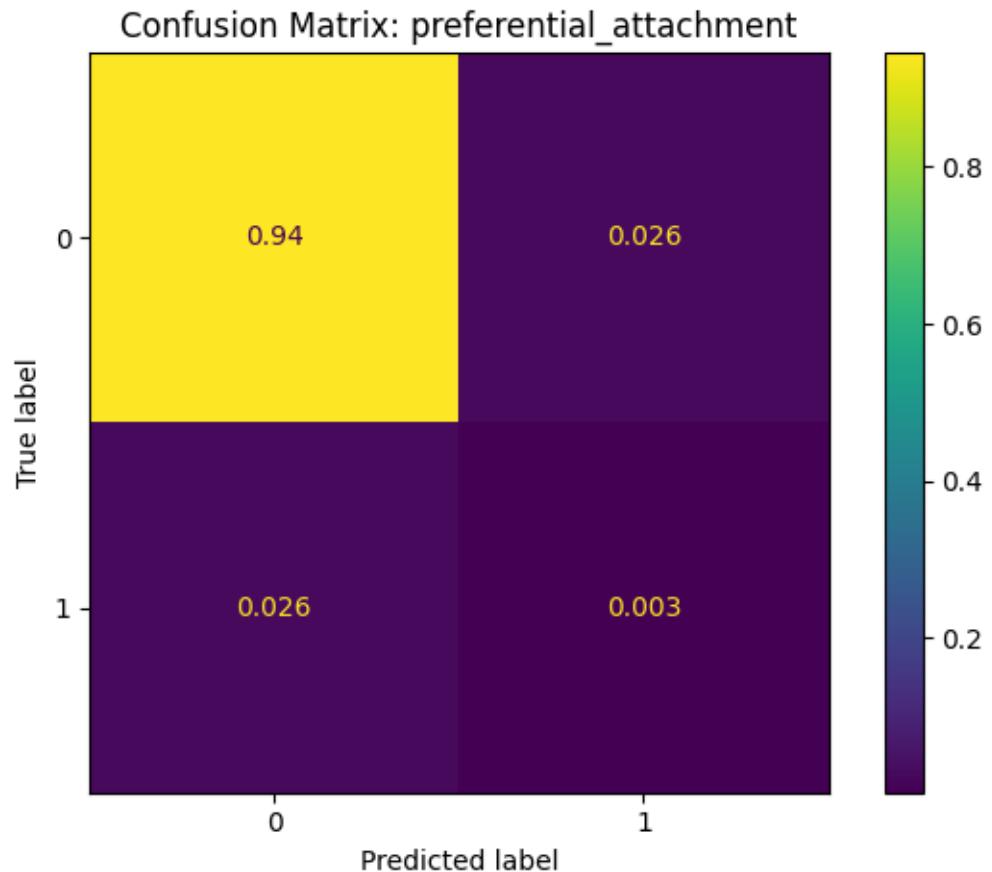












CHAPTER
TWENTYFOUR

SPATIAL REGRESSION

Everything is related to everything else. But near things are more related than distant things - Waldo Tobler

Concepts

- Earnings Surprises
- Spatial Dependence

```
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'
```

(continues on next page)

(continued from previous page)

```
# 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
```

24.1 Earnings surprise

An earnings surprise, or unexpected earnings, is the difference between the reported earnings and expected earnings.

```
# 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
```

```
df = store['fund']
```

```
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()))
```

```
# Retrieve TNIC schemes 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()
```

```
fund, tnic_df = store['spatial']
```

```
sue = fund.loc[~fund['sue'].isna(), ['permno', 'rebaldate', 'sue']] \
    .reset_index(drop=True)
sue['rebaldate'] // 100
```

```
lookup = pstat.build_lookup(source='lpermno', target='gvkey', fillna=0)
```

24.2 Spatial dependence models

24.2.1 Moran's I

$$E(I) = \frac{-1}{N-1}$$

summary results compare to null of zero so is more conservative (difference compared to true expectations is more positive than calculated)

https://en.wikipedia.org/wiki/Moran's_I

24.2.2 Spatial lag model

https://lost-stats.github.io/Geo-Spatial/spatial_lag_model.html

https://pysal.org/spreg/generated/spreg.ML_Lag.html

spatial lag model $Y_i = \beta X_i + \rho W Y_j + \epsilon_i$

where

- Y_j is the set of Y values from observations other than i ,
- W is a matrix of spatial weights, which are higher for j s that are spatially closer to i .

```
# Compute quarterly spatial regressions
out = dict(moran={}, rho={}, beta={})
years = range(1988, 2022)
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']]

        # 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
```

(continues on next page)

(continued from previous page)

```

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])

```

100% |██████████| 34/34 [10:56<00:00, 19.30s/it]

```

# Show latest quarter's results
print(spatial.summary)

```

```

REGRESSION
-----
SUMMARY OF OUTPUT: MAXIMUM LIKELIHOOD SPATIAL LAG (METHOD = FULL)
-----
Data set : 202209
Weights matrix : tnic3
Dependent Variable : sue
Number of Observations: 2464
Mean dependent var : -0.0009
Number of Variables : 3
S.D. dependent var : 0.0337
Degrees of Freedom : 2461
Pseudo R-squared : 0.0786
Spatial Pseudo R-squared: 0.0037
Sigma-square ML : 0.001
Log likelihood : 4937.
Akaike info criterion : -9869.
S.E. of regression : 0.032
Schwarz criterion : -9851.
362
933

-----
Variable Coefficient Std.Error z-Statistic
Probability
-----
CONSTANT -0.00011554 0.0006571 -1.7581916 0.
0787149

```

(continues on next page)

(continued from previous page)

```

    ret      0.0096268      0.0028302      3.4015254      0.
0006701
    W_sue    0.4260827      0.0388927     10.9553442      0.
0000000
-----
-----
===== END OF REPORT
=====

```

```

# Show spatial regression results
ols = spreg.OLS(y=y['sue'].values, x=x[['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      : 202209
Weights matrix : tnic3
Dependent Variable : sue
Number of Observations: 2464
Mean dependent var : -0.0009
Number of Variables : 2
S.D. dependent var : 0.0337
Degrees of Freedom : 2462
R-squared       : 0.0044
Adjusted R-squared : 0.0040
Sum squared residual: 2.786
F-statistic      : 10.9488
Sigma-square     : 0.001
Prob(F-statistic) : 0.
0009502
S.E. of regression : 0.034
Log likelihood    : 4862.
690
Sigma-square ML  : 0.001
Akaike info criterion : -9721.
379
S.E of regression ML: 0.0336
Schwarz criterion : -9709.
760

White Standard Errors
-----
-----
Variable      Coefficient      Std.Error      t-Statistic
Probability
-----
-----
CONSTANT      -0.0009294      0.0006788      -1.3691415      0.
1710799
ret           0.0097161      0.0034833      2.7893193      0.
0053225
-----
-----

```

(continues on next page)

(continued from previous page)

```

REGRESSION DIAGNOSTICS
MULTICOLLINEARITY CONDITION NUMBER           1.014

TEST ON NORMALITY OF ERRORS
TEST                         DF      VALUE      PROB
Jarque-Bera                  2       5577.863    0.0000

DIAGNOSTICS FOR HETROSKEDEASTICITY
RANDOM COEFFICIENTS
TEST                         DF      VALUE      PROB
Breusch-Pagan test           1       15.827    0.0001
Koenker-Bassett test         1       3.382     0.0659

DIAGNOSTICS FOR SPATIAL DEPENDENCE
TEST                         MI/DF    VALUE      PROB
Moran's I (error)            0.1373   15.494    0.0000
Lagrange Multiplier (lag)    1        234.778    0.0000
Robust LM (lag)              1        0.391     0.5320
Lagrange Multiplier (error)  1        237.113    0.0000
Robust LM (error)            1        2.726     0.0987
Lagrange Multiplier (SARMA)  2        237.504    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: 202209

| | Moran's I | Expected | p_norm |
|-------------------------------|-----------|-----------|--------------|
| under assumption of normality | 0.136383 | -0.000406 | 2.128943e-53 |

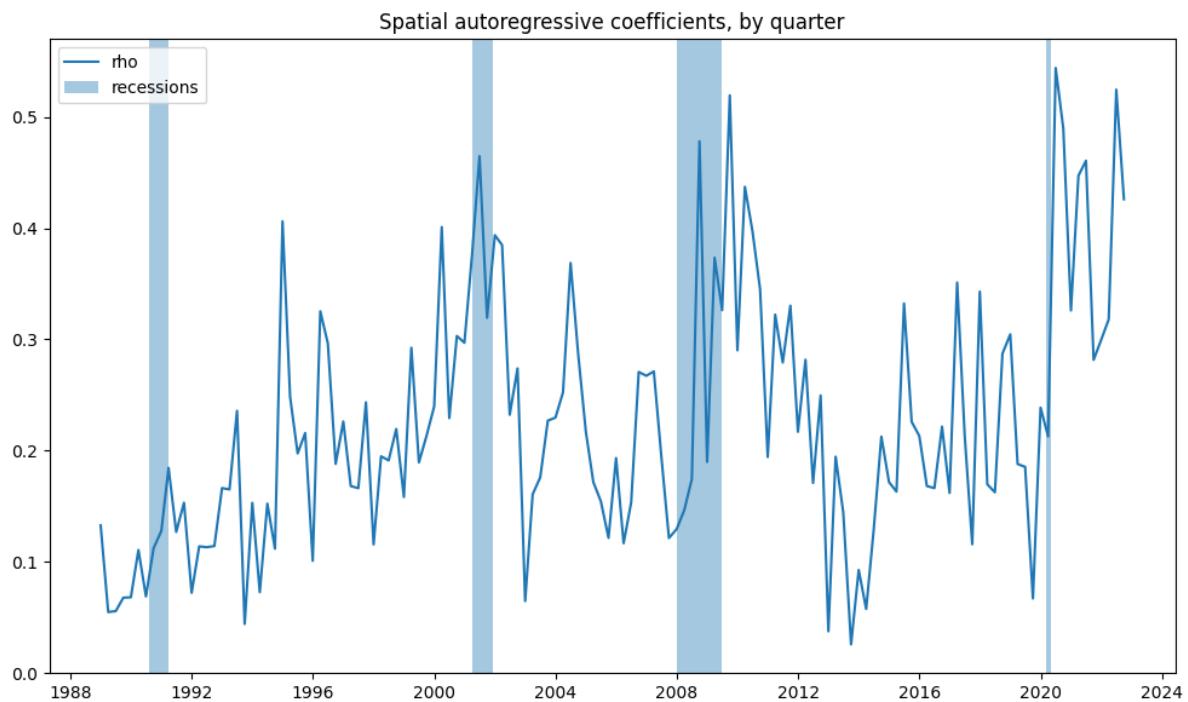
```

# 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})
    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 | 0.053145 | 0.224963 | 0.012003 |
| stderr | 0.002668 | 0.009765 | 0.000720 |
| t | 19.919201 | 23.036839 | 16.672970 |
| nw_stderr | 0.004736 | 0.018049 | 0.000979 |
| nw_t | 11.220779 | 12.463735 | 12.265549 |

```
# 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
TWENTYFIVE

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

- FOMC meeting minutes
- Tokenization
- Topic models

```
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 secret import credentials, paths
# %matplotlib qt
VERBOSE = 0
```

```
mongodb = MongoDB(**credentials['mongodb'], verbose=VERBOSE)
fomc = Unstructured(mongodb, 'FOMC')
```

```
## retrieve recessions dates for plotting
alf = Alfred(api_key=credentials['fred']['api_key'])
vspan = alf.date_spans(series_id='USREC')
```

25.1 FOMC minutes

The FOMC holds eight regularly scheduled meetings during the year and other meetings as needed. The minutes of regularly scheduled meetings are released three weeks after the date of the policy decision.

<https://www.federalreserve.gov/monetarypolicy/fomccalendars.htm>

- keep minutes text approximately from (and including):
 - “Developments in Financial Markets” or
 - “Discussion of Guideliness for Policy Normalization” or
 - “Financial Developments and Open Market Operations” or
 - “discussion of the economic outlook” or
 - “the information reviewed at this meeting” or
 - “the staff presented several briefings” (i.e. staff presentations on ad-hoc policy topics)
- skip earlier text which may be
 - re-affirmation of general policy statement
 - organizational items
- delete after adjourn line or date scheduled for next meeting, ignore:
 - notation votes, minutes approvals, signature, and footnotes
 - intermeeting conference calls

```
minutes = FOMCReader()
DataFrame({'dates': len(minutes), 'start': min(minutes), 'end': max(minutes)},
          index=['FOMC minutes'])
```

| | dates | start | end |
|--------------|-------|----------|----------|
| FOMC minutes | 249 | 19930203 | 20240320 |

```
# retrieve dates 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:', ", ".join([f'{k}: {len(v)} chars' for k,v in docs.items()]))
```

NEW:

```
# clean 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])
```

(continues on next page)

(continued from previous page)

```
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
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...
...
20230920 Developments in Financial Markets and Open Mar...
20231101 Developments in Financial Markets and Open Mar...
20231213 Developments in Financial Markets and Open Mar...
20240131 Developments in Financial Markets and Open Mar...
20240320 Developments in Financial Markets and Open Mar...
```

[249 rows x 1 columns]

25.2 Tokenization

Splitting text into smaller units called **tokens**, which are usually words, but also may be phrases or subwords

25.2.1 Regular Expressions

A regular expression is a special text string for describing a text search pattern to match character combinations in strings. They are used in tokenization, to find word boundaries and remove whitespace.

Basic regex.

| Expression | Description |
|------------|--|
| \d | digit from 0 to 9 |
| \w | word character: ASCII letter, digit or underscore |
| \s | whitespace character: space, tab, newline, carriage return, vertical tab |
| \D | character that is not a digit |
| \W | character that is not a word character |
| \S | character that is not a whitespace character |
| [...] | one of the characters in the bracket |
| [a-zA-Z] | one of the characters in the range |
| [^a] | one character that is not a |
| [^a-z] | one character that is not in the range |
| \b | word boundary |
| \B | not a word boundary |
| . | any character except line break |
| . | the period character |
| \ | escapes a special character: . * + ? \$ ^ / \ |
| ^ | start of string: position is an ASCII letter, digit or underscore |
| \$ | end of string: position is an ASCII letter, digit or underscore |
| + | quantifier: one or more |
| * | quantifier: zero or more |
| ? | quantifier: zero or one |
| | OR operand |
| (...) | capturing group |

25.2.2 Stopwords

```
# 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']
```

25.2.3 Vectorization

Converts text into numerical representations

```
# 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()
```

(continues on next page)

(continued from previous page)

```

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")

```

25.3 Topic models

TODO

Topic modeling is a statistical technique to discover latent topics that exist within a collection of documents

- LSA
- LDA
- PLSI
- NMF

```

# 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',

```

(continues on next page)

(continued from previous page)

```

        alpha_W=0.00005,
        alpha_H=0.00005,
        l1_ratio=0.5),
    tfidf_vectorizer) }

# 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()

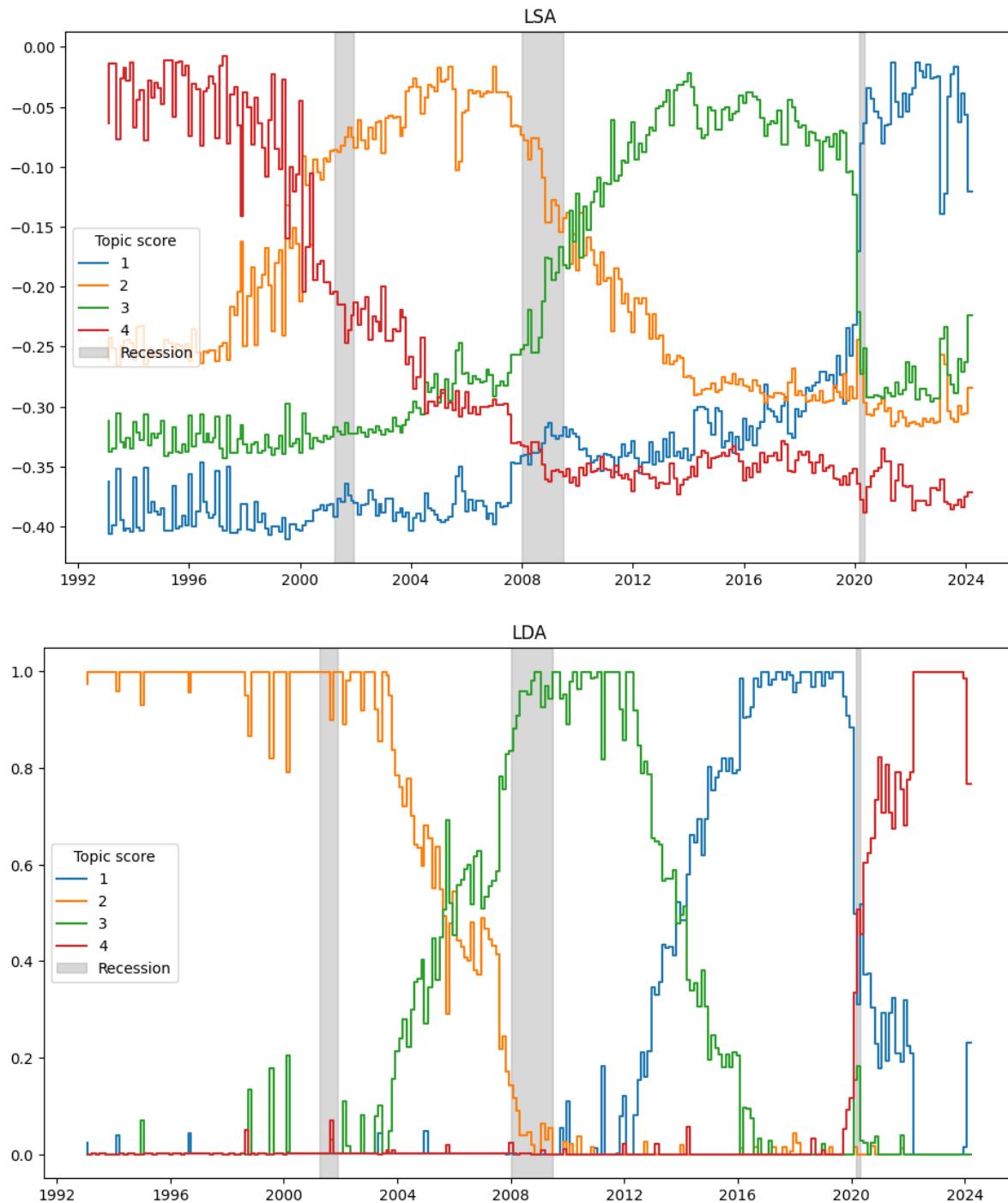
    # fit model and transform inputs
    model = base.fit(vectorized)
    transformed = model.transform(vectorized)

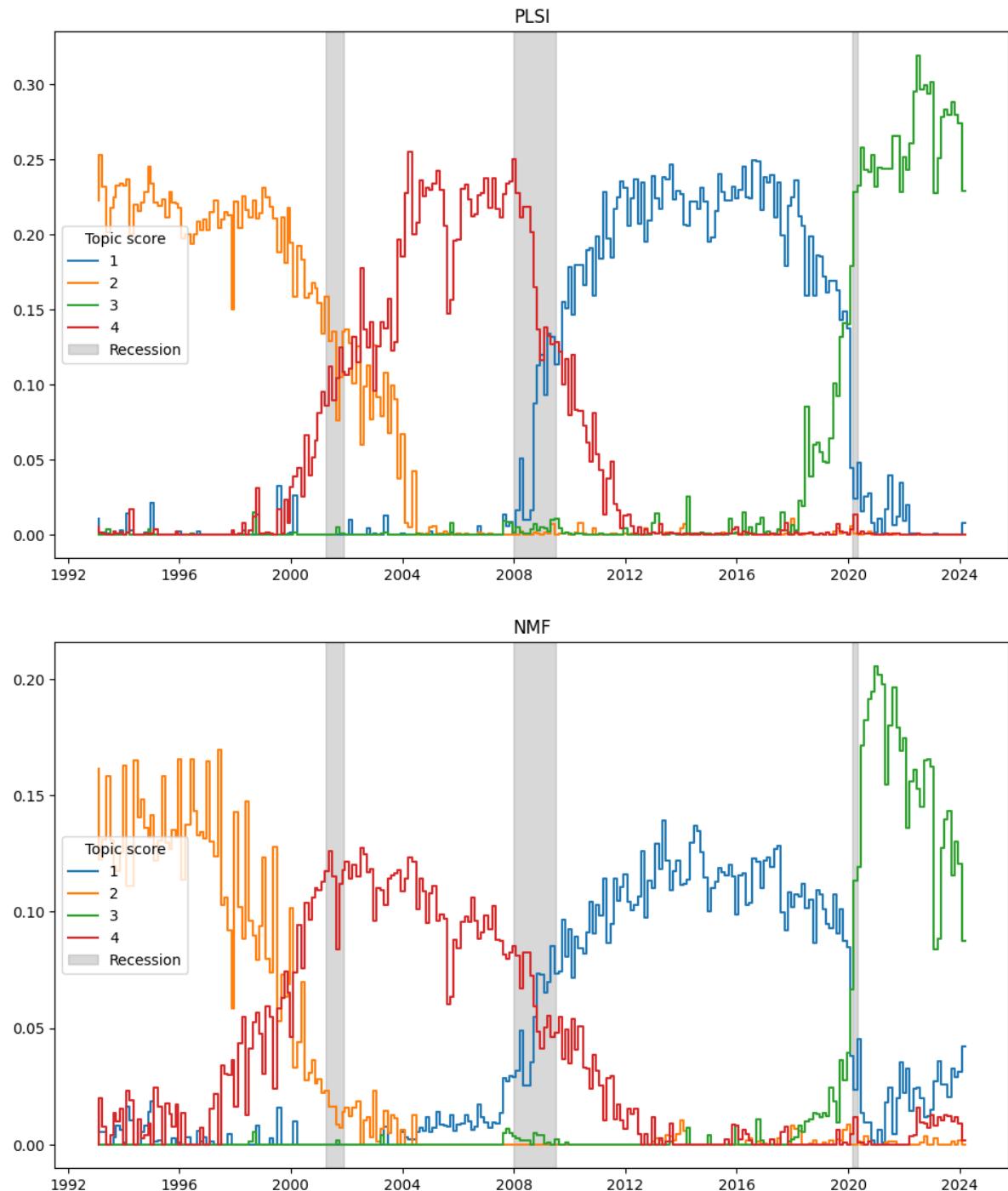
    # save the model fitted coefficients
    if name == 'LSA': # Additional step for LSA
        kmeans = KMeans(n_clusters=n_components, n_init=5, random_state=37) \
            .fit(transformed) # find centroids of the latent factors
        transformed = kmeans.transform(transformed) # distance to centroid
        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 score')
    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])}

```





```
# Display word cloud of top n features
figsize = (10, 8)
for ifig, (name, score) in enumerate(scores.items()):
    wc = WordCloud(height=300, width=500, colormap='cool')
    top_n = 20
    fig, axes = plt.subplots(2, 2, num=ifig+5, figsize=figsize, clear=True)
    for topic, components in enumerate(score):
        words = {feature_names[i].replace(' ', '_') : components[i]
```

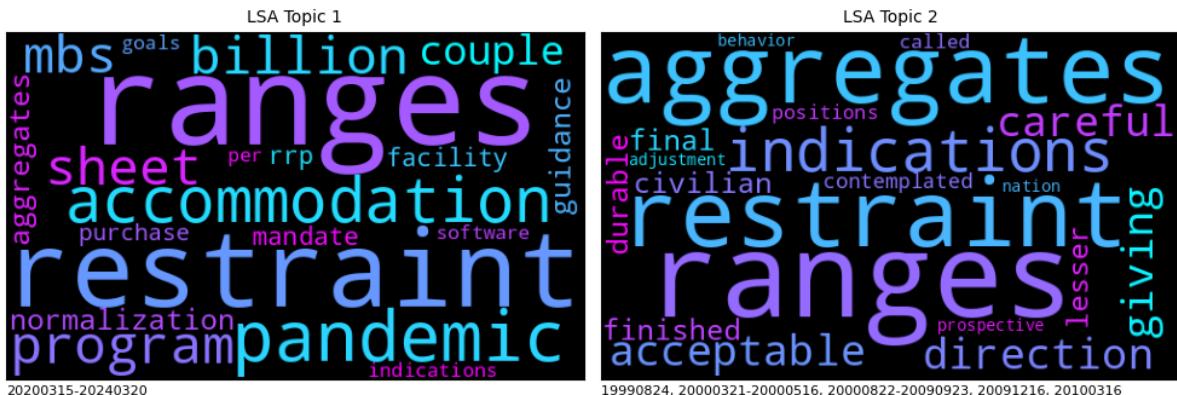
(continues on next page)

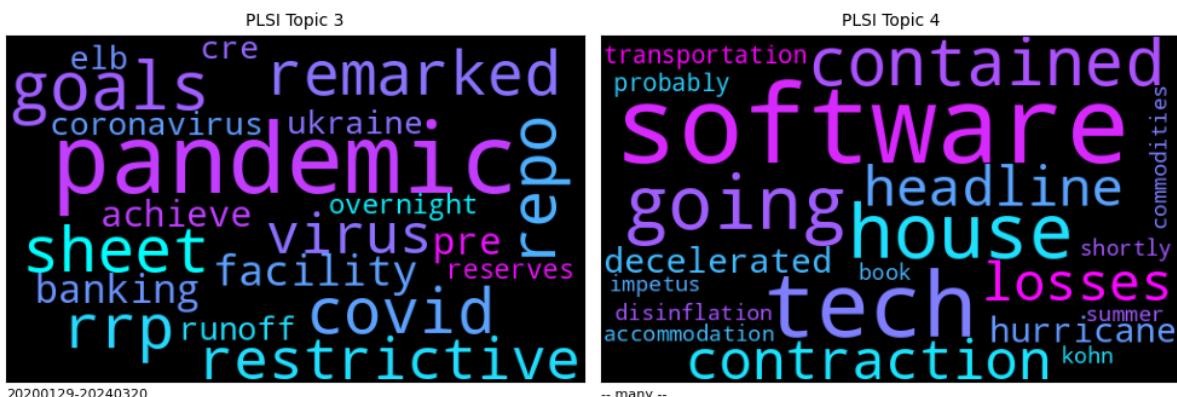
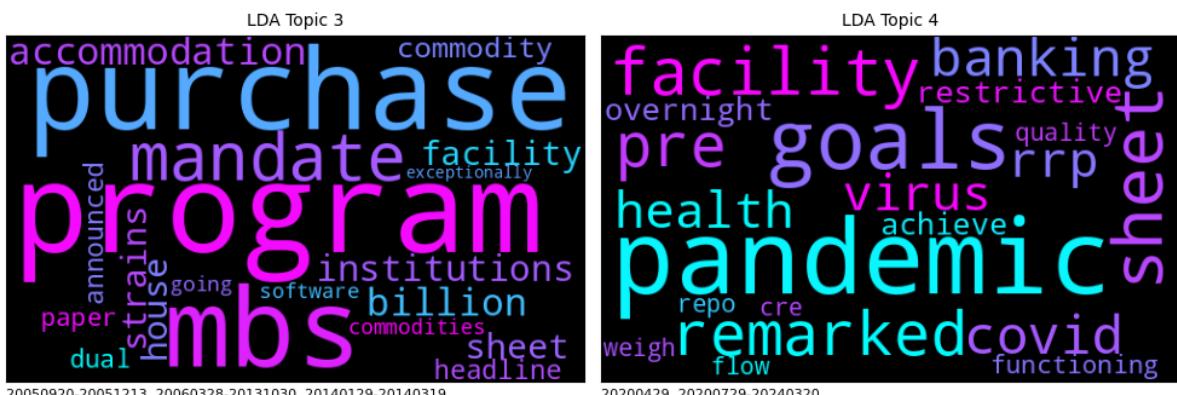
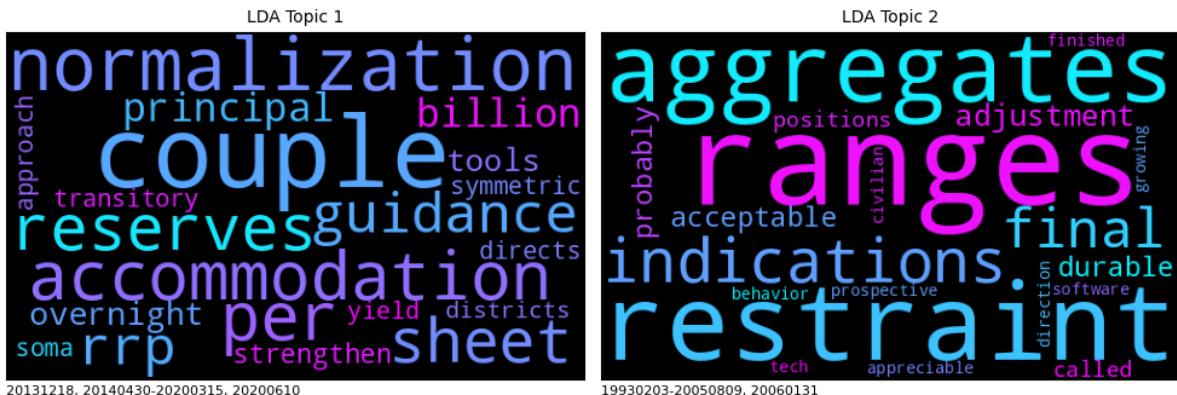
(continued from previous page)

```

    for i in components.argsort()[:-top_n - 1:-1]:
        #print("Topic", topic+1, topics[name])
        #print(list(words.keys()))
        ax = axes[topic//2, topic % 2]
        ax.imshow(wc.generate_from_frequencies(words))
        ax.axes.yaxis.set_visible(False)    # make axes ticks invisible
        ax.xaxis.set_ticks([])
        ax.xaxis.set_ticklabels([])
        ax.set_title(f"{name} Topic {topic+1}", fontsize=10)
        regime = ", ".join([f"{d[0]}-{d[1]}" if d[0] != d[1] else f"{d[0]}"
                            for d in topics[name][topic]])
        ax.set_xlabel(regime if len(regime) < 75 else '-- many --',
                      fontsize=8,
                      loc='left')
    plt.tight_layout()

```







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
Concepts:

- Sentiment Analysis
- SEC Edgar Company Filings

References:

- Cohen, Malloy and Nguyen (2020)
- 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.

```
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)
ed = Edgar(paths['10X'], zipped=True, verbose=VERBOSE)
signals = Signals(user)
store = Store(paths['scratch'], ext='pkl')
```

Last FamaFrench Date 2024-03-28 00:00:00

26.1 Sentiment analysis

Loughran and MacDonald sentiment word lists

- <https://sraf.nd.edu/loughranmcdonald-master-dictionary/>

```
# to download from google drive, need this prefix and the 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=',', encoding='utf-8')

# 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())}
```

```
DataFrame(words['positive'], columns=['Positive Words'])
```

```
Positive Words
0      attractive
1  collaborating
2  benefitting
3    achieving
4      progress
..      ...
349      worthy
350  tremendous
351      succeed
352  enhancements
353    exclusives
```

```
[354 rows x 1 columns]
```

```
DataFrame(words['negative'], columns=['Negative Words'])
```

```
Negative Words
0      arrears
1  unannounced
2    deceiving
3      harsh
4  disallowances
..      ...
2350  undercutting
2351  misrepresented
2352      bribed
2353    convicted
2354      fault
```

```
[2355 rows x 1 columns]
```

```
# stopwords
generic = ['ME', 'MY', 'MYSELF', 'WE', 'OUR', 'OURS', 'OURSELVES',
           'YOU', 'YOUR', 'YOURS', 'YOURSELF', 'YOURSELVES',
```

(continues on next page)

(continued from previous page)

```
'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
features = Series(sentiment_vectorizer.get_feature_names_out())
sentiment_points = (features.isin(words['positive']).astype(int).values
                    - features.isin(words['negative']).astype(int).values)
```

26.2 SEC Edgar company filings

TODO

26.2.1 10-K's

- management discussion and analysis section

```
# open mda10K archive
item, form = 'mda10K', '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
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
```

```
# 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 | 14499 | 133190 | 19931129 | 20231229 |

```
# 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:
```

(continues on next page)

(continued from previous page)

```

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']]
    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% | 0/14499 [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%|██████████| 14499/14499 [10:47<00:00, 22.39it/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))

```

```

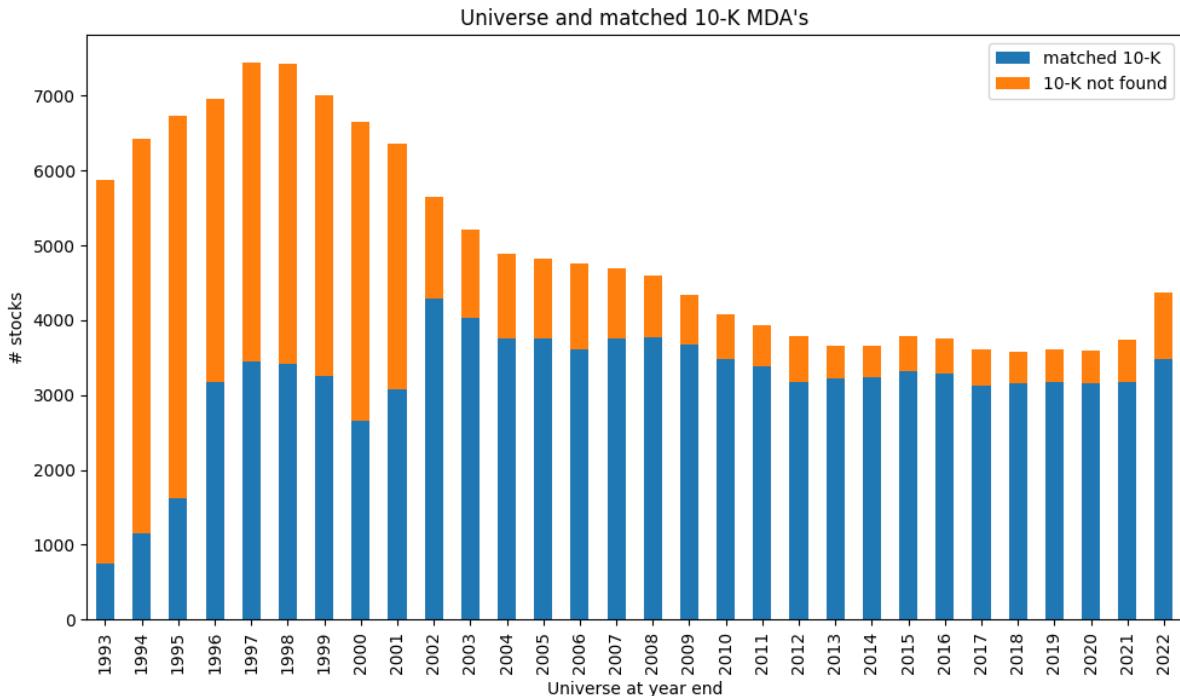
(signals_write) mdasent 126842
(signals_write) mdachg 103966
126842 103966

```

```
# right join data with univ, to identify universe companies with missing mda
data = pd.concat([data[data['year']==year]\n    .drop(columns=['year'])\\
    .set_index('permno')\\
    .join(univ[['year']], how='right')\\
    .reset_index()\n    for year, univ in univs.items() if year < LAST_YEAR],\n    ignore_index=True)\n\n# store temporary\nstore['sentiment'] = data
```

```
data = store['sentiment']
```

```
# Stacked Bar Plot of universe coverage by year\ny1 = data[data['mdasent'].notna()].groupby('year')['permno'].count()\ny0 = data[data['mdasent'].isna()].groupby('year')['permno']\\
    .count()\\
    .reindex(y1.index)\nfig, ax = plt.subplots(1, 1, clear=True, figsize=(10, 6))\ny1.plot(kind='bar', label='matched 10-K', color='C0', ax=ax, rot=90)\ny0.plot(kind='bar', label='10-K not found', color='C1', ax=ax, rot=90, bottom=y1)\nax.set_ylabel('# stocks')\nax.set_xlabel('Universe at year end')\nax.set_title("Universe and matched 10-K MDA's")\nax.legend()\nplt.tight_layout()
```



```
# Stacked Bar Plot of filings date, by month and day-of-week\ny = DataFrame.from_records([{'date': int(d),\n    'day': bd.datetime(int(d)).strftime('%A'),\n    'year': int(d)} for d in range(1993, 2023)])
```

(continues on next page)

(continued from previous page)

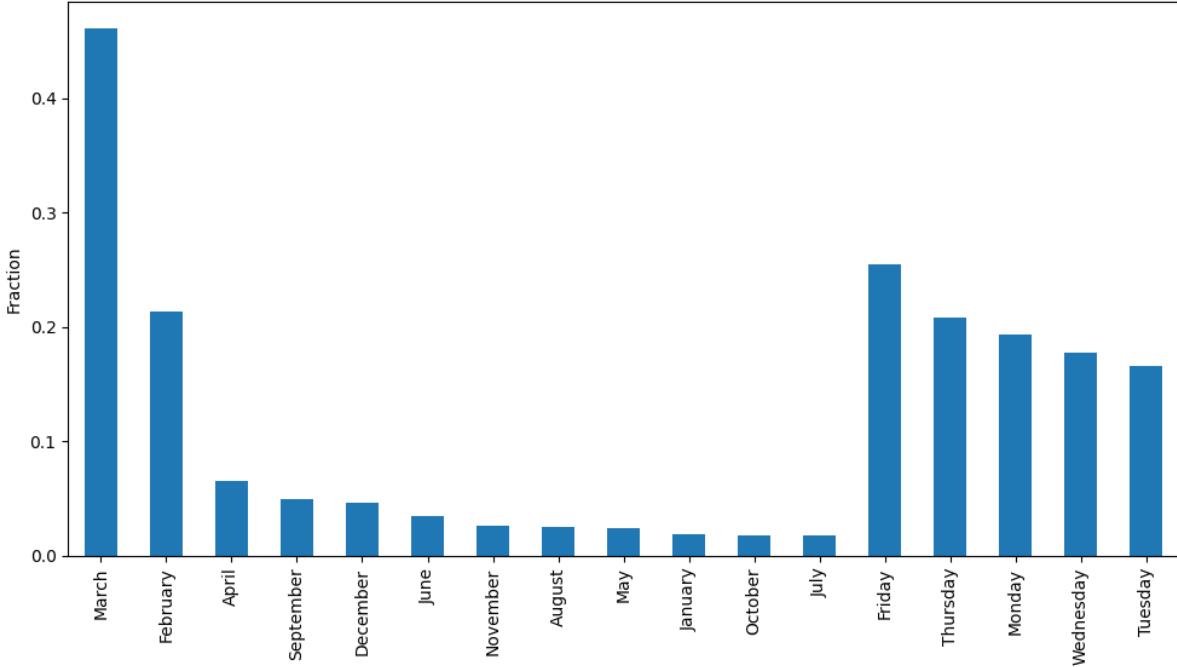
```

'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()

```

10-K Filings by Month and Day-of-Week



Plt median annual sentiment and change in sentiment vs total corporate profits

```

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['currln'].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"],
            fontsize='small', loc='upper left')

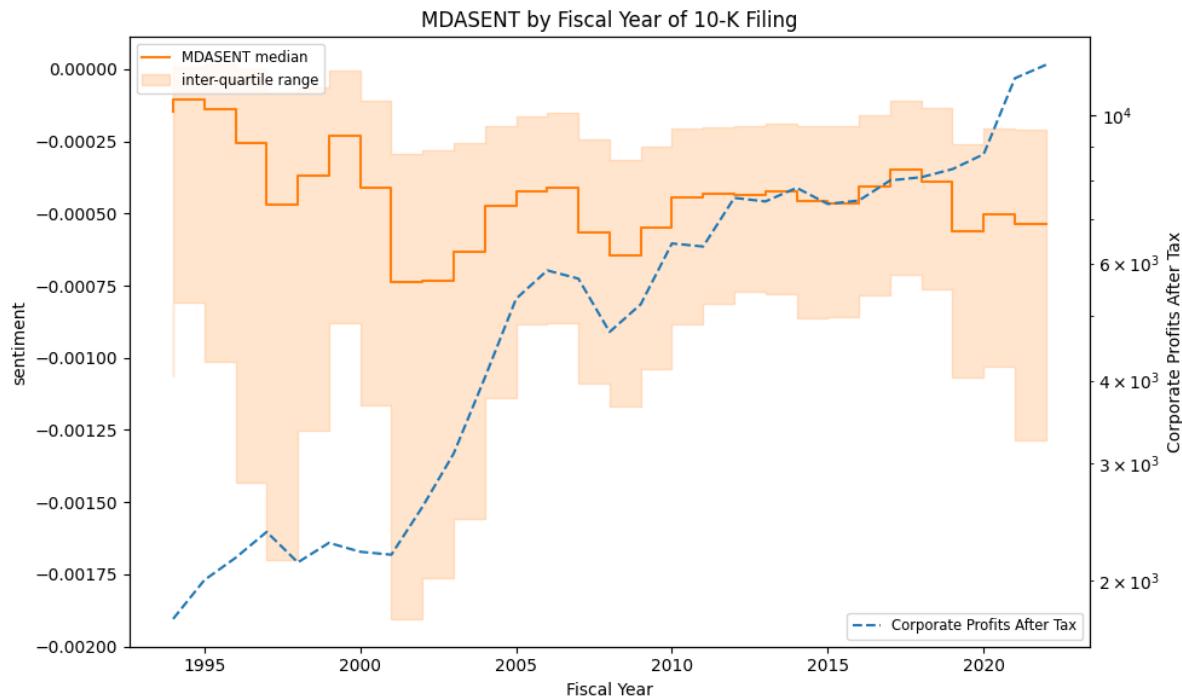
```

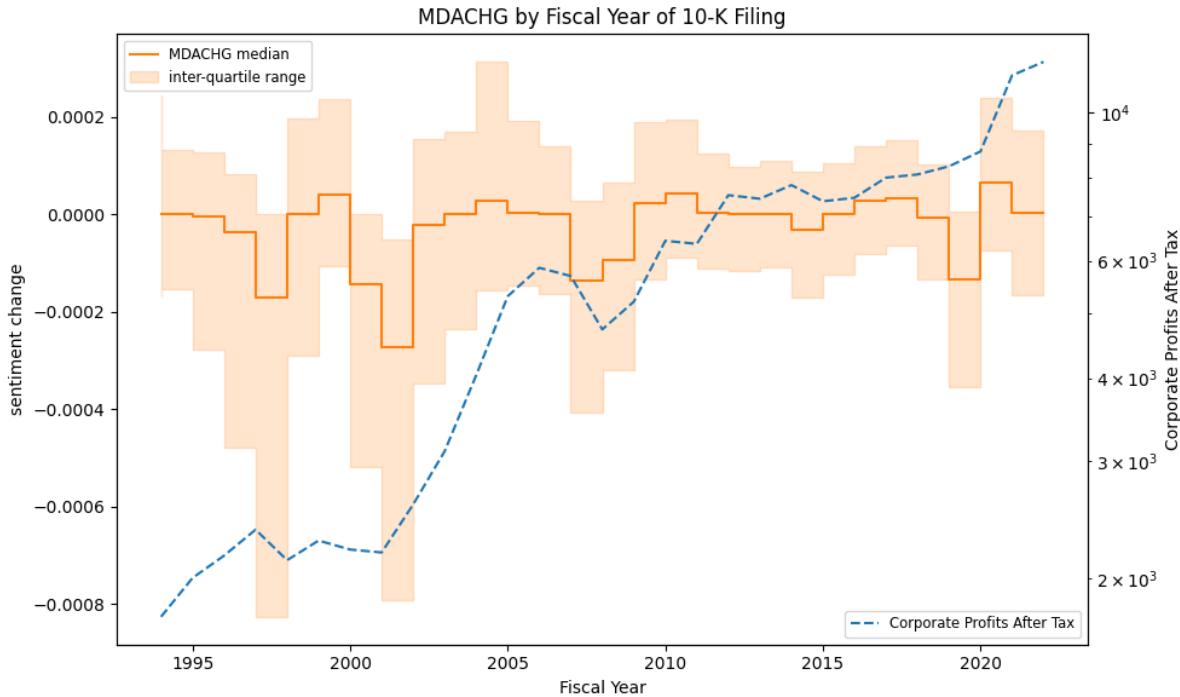
(continues on next page)

(continued from previous page)

```
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
```





Compute decile spread returns, in same year (Jan-Dec) and year ahead (Apr-Mar)

```

for ifig, key in enumerate(['mdasent', 'mdachg']):
    ret1 = {} # to collect year-ahead spread returns
    ret0 = {} # to collect current-year spread returns
    for year in range(1999, max(data['year'])+1): # loop over years

        # compute current year 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 spread returns
        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']

```

(continues on next page)

(continued from previous page)

```

ret1[end] = {'ret': pos-neg, 'npos': sum(sub==1), 'nneg': sum(sub==3) }

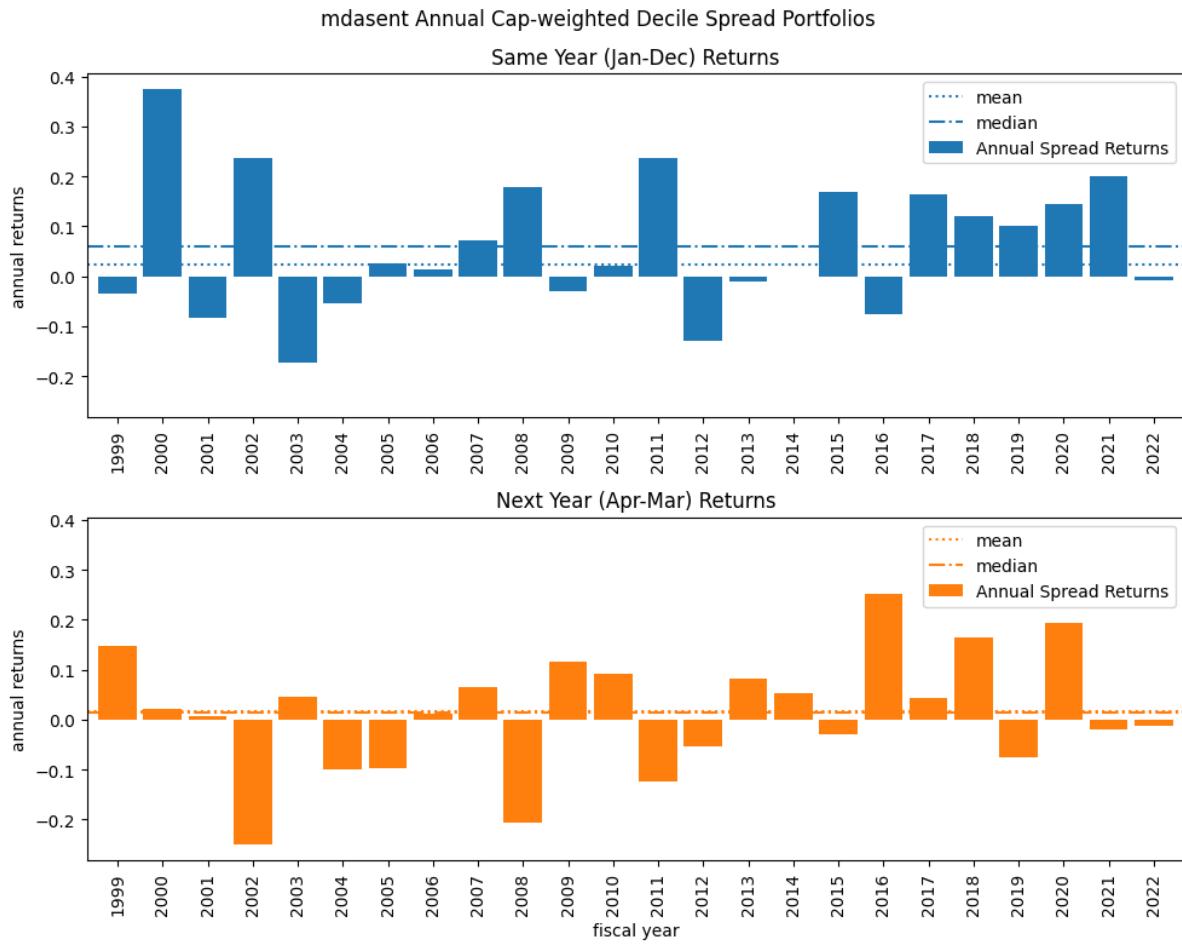
# collect same-year and year-ahead spread returns
r0 = DataFrame.from_dict(ret0, orient='index').sort_index()
r0.index = r0.index // 10000
r1 = DataFrame.from_dict(ret1, orient='index').sort_index()
r1.index = (r1.index // 10000) - 2

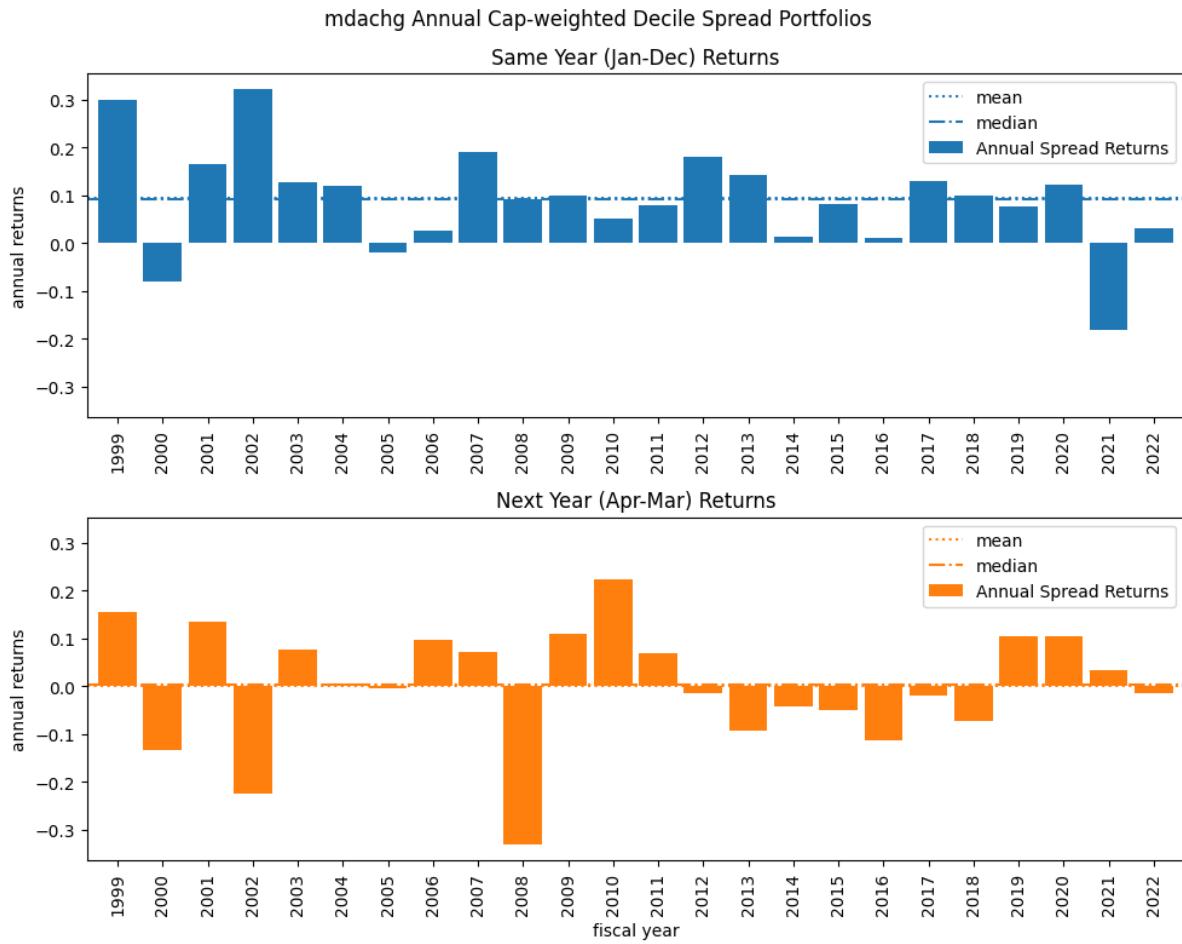
# plot same-year spread returns
fig, ax = plt.subplots(nrows=2, clear=True, figsize=(10, 8), sharey=True)
r0['ret'].plot(kind='bar', ax=ax[0], width=.85, color="C0")
ax[0].axhline(r0['ret'].median(), linestyle=':', color='C0')
ax[0].axhline(r0['ret'].mean(), linestyle='-.', color='C0')
ax[0].set_title(f"Same Year (Jan-Dec) Returns")
ax[0].set_ylabel('annual returns')
ax[0].legend(['mean', 'median', 'Annual Spread Returns'])

# plot year-ahead 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}} Annual Cap-weighted Decile Spread Portfolios")
plt.tight_layout()

```





CHAPTER
TWENTYSEVEN

BUSINESS TEXT ANALYSIS

You shall know a word by the company it keeps - J. R. Firth

- Syntactic analysis
- Word2Vec
- Density-based Clustering

```
import re
import numpy as np
from scipy import spatial
import pandas as pd
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
import spacy
from sklearn import cluster
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from tqdm import tqdm
from finds.database import SQL, RedisDB
from finds.structured import CRSP, BusDay
from finds.unstructured import Edgar
from finds.utils import Store, Finder, ColorMap
from secret import credentials, paths
# %matplotlib qt
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)
ed = Edgar(paths['10X'], zipped=True, verbose=VERBOSE)
store = Store(paths['scratch'])
find = Finder(sql)
```

Last FamaFrench Date 2024-03-28 00:00:00

27.1 Syntactic analysis

27.1.1 Spacy language pipeline

```
# ! python -m spacy download en_core_web_sm
nlp = spacy.load("en_core_web_lg")

# Retrieve universe of stocks, keep to decile
univ = crsp.get_universe(bd.endmo(20221231))
comnam = crsp.build_lookup('permno', 'comnam', fillna="") # company name
univ['comnam'] = comnam(univ.index)
ticker = crsp.build_lookup('permno', 'ticker', fillna="") # tickers
univ['ticker'] = ticker(univ.index)

# retrieve 2023 bus10K's
item, form = 'bus10K', '10-K'
rows = DataFrame(ed.open(form=form, item=item))
found = rows[rows['permno'].isin(univ.index[univ.decile <= 1])] # large decile
    & rows['date'].between(20230101, 20231231)]\n    .drop_duplicates(subset=['permno'], keep='last')\n    .set_index('permno')
```

27.1.2 Tokenization

```
# Example of NVIDIA's 10-K business description text
nvidia = find('NVIDIA')['permno'].iloc[0]
doc = nlp(ed[found.loc[nvidia, 'pathname']][:nlp.max_length].lower())
tokens = DataFrame.from_records([{'text': token.text,
                                'lemma': token.lemma_,
                                'alpha': token.is_alpha,
                                'stop': token.is_stop,
                                'punct': token.is_punct}
                                for token in doc], index=range(len(doc)))
tokens.head(30)
```

| | text | lemma | alpha | stop | punct |
|----|-------------|------------|-------|-------|-------|
| 0 | item | item | True | False | False |
| 1 | 1 | 1 | False | False | False |
| 2 | . | . | False | False | True |
| 3 | business | business | True | False | False |
| 4 | \n\n | \n\n | False | False | False |
| 5 | our | our | True | True | False |
| 6 | company | company | True | False | False |
| 7 | \n\n | \n\n | False | False | False |
| 8 | nvidia | nvidia | True | False | False |
| 9 | pioneered | pioneer | True | False | False |
| 10 | accelerated | accelerate | True | False | False |
| 11 | computing | computing | True | False | False |
| 12 | to | to | True | True | False |
| 13 | help | help | True | False | False |
| 14 | solve | solve | True | False | False |

(continues on next page)

(continued from previous page)

```

15      the          the  True  True  False
16      most         most  True  True  False
17  challenging  challenging  True  False  False
18  computational  computational  True  False  False
19      problems      problem  True  False  False
20      .             .  False  False  True
21      since         since  True  True  False
22      our            our  True  True  False
23      original      original  True  False  False
24      focus          focus  True  False  False
25      on              on  True  True  False
26      pc              pc  True  False  False
27  graphics        graphic  True  False  False
28      ,              ,  False  False  True
29      we              we  True  True  False

```

27.1.3 Part-of-speech tagging

```

tags = DataFrame.from_records([{'text': token.text,
                               'pos': token.pos_,
                               'tag': token.tag_,
                               'dep': token.dep_}
                               for token in doc], index=range(len(doc)))
tags.head(30)

```

| | text | pos | tag | dep |
|----|---------------|-------|-------|----------|
| 0 | item | NOUN | NN | ROOT |
| 1 | 1 | NUM | CD | nummod |
| 2 | . | PUNCT | . | punct |
| 3 | business | NOUN | NN | nsubj |
| 4 | \n\n | SPACE | _SP | dep |
| 5 | our | PRON | PRP\$ | poss |
| 6 | company | NOUN | NN | appos |
| 7 | \n\n | SPACE | _SP | dep |
| 8 | nvidia | PROPN | NNP | appos |
| 9 | pioneered | VERB | VBD | ROOT |
| 10 | accelerated | VERB | VBD | xcomp |
| 11 | computing | NOUN | NN | dobj |
| 12 | to | PART | TO | aux |
| 13 | help | VERB | VB | advcl |
| 14 | solve | VERB | VB | xcomp |
| 15 | the | DET | DT | det |
| 16 | most | ADV | RBS | advmod |
| 17 | challenging | ADJ | JJ | amod |
| 18 | computational | ADJ | JJ | amod |
| 19 | problems | NOUN | NNS | dobj |
| 20 | . | PUNCT | . | punct |
| 21 | since | SCONJ | IN | prep |
| 22 | our | PRON | PRP\$ | poss |
| 23 | original | ADJ | JJ | amod |
| 24 | focus | NOUN | NN | pobj |
| 25 | on | ADP | IN | prep |
| 26 | pc | NOUN | NN | compound |

(continues on next page)

(continued from previous page)

```

27     graphics    NOUN    NNS      pobj
28     ,    PUNCT    ,      punct
29     we    PRON    PRP      nsubj

```

27.1.4 Entity recognition

```

ents = DataFrame.from_records([{'text': ent.text,
                                'label': ent.label_,
                                'start': ent.start_char,
                                'end': ent.end_char}
                                for ent in doc.ents], index=range(len(doc.ents)))
ents.head(20)

```

| | text | label | start | end |
|----|----------------------|----------|-------|------|
| 0 | 1 | CARDINAL | 5 | 6 |
| 1 | nvidia | ORG | 357 | 363 |
| 2 | gpu | ORG | 382 | 385 |
| 3 | av | ORG | 515 | 517 |
| 4 | gpu | ORG | 576 | 579 |
| 5 | today | DATE | 684 | 689 |
| 6 | thousands | CARDINAL | 835 | 844 |
| 7 | gpu | ORG | 1115 | 1118 |
| 8 | thousands | CARDINAL | 1161 | 1170 |
| 9 | gpu | ORG | 1325 | 1328 |
| 10 | gpus | GPE | 2191 | 2195 |
| 11 | multi-billion-dollar | MONEY | 2290 | 2310 |
| 12 | third | ORDINAL | 2431 | 2436 |
| 13 | over 37 billion | MONEY | 2663 | 2678 |
| 14 | gpu | ORG | 2809 | 2812 |
| 15 | 1999 | DATE | 2816 | 2820 |
| 16 | 2006 | DATE | 2968 | 2972 |
| 17 | gpu | ORG | 3028 | 3031 |
| 18 | today | DATE | 3280 | 3285 |
| 19 | gpus | GPE | 3291 | 3295 |

```

# Entity Visualizer
from spacy import displacy
displacy.render(doc[:300], style="ent", jupyter=True)

```

```
<IPython.core.display.HTML object>
```

27.1.5 Dependency tree

```
sentence_spans = list(doc.sents)
displacy.render(sentence_spans[2:4], style="dep", jupyter=True)
```

<IPython.core.display.HTML object>

27.2 Word2Vec

Trained using the word2vec family of algorithms

```
# Extract nouns
bus = {}
for permno in tqdm(found.index):
    doc = nlp(ed[found.loc[permno, 'pathname']][:nlp.max_length].lower())
    nouns = " ".join([re.sub("[^a-zA-Z]+", "", token.lemma_) for token in doc
                      if token.pos_ in ['NOUN'] and len(token.lemma_) > 2])
    if len(nouns) > 100:
        bus[permno] = nouns
store['business'] = bus
```

```
bus = store.load('business')
permnos = list(bus.keys())
tickers = univ.loc[permnos, 'ticker'].to_list()
```

```
# example of word vector
vec1 = nlp(bus[nvidia]).vector
vec1
```

```
array([-0.5370803,  0.03360266, -0.7142777,  1.0528733,  2.59639,  ,
       0.29917082,  1.3443472,  3.7829578, -2.2698855, -1.2616774,  ,
       5.9859304,  2.0566463, -4.519684,  2.2885911, -1.1176012,  ,
       2.4155278,  3.2593274,  1.5960239, -2.3042047,  0.06216303,  ,
       0.21148767,  1.7382729, -2.3506453,  0.7941269, -1.2192215,  ,
      -1.8039206, -1.8674678, -1.6660213, -0.7127941,  1.0603017,  ,
       1.2367841,  1.2303644, -1.1252155, -0.44499,  0.34517637,  ,
      -0.38984352,  1.4724953,  0.7678367,  1.3465229,  0.35140815,  ,
       0.2309201, -0.14058447,  0.18483874,  0.99899405, -1.4584388,  ,
       1.5930446,  2.0110247, -1.9788889,  0.49480593, -1.2044214,  ,
       0.11300751,  2.2886782, -0.19940257, -3.7744892, -0.5143599,  ,
       0.5165942, -1.6783403,  1.471545,  0.44622797, -1.4958122,  ,
       2.2605124,  1.2073922, -2.2683856, -1.2163076,  2.4330237,  ,
       1.984873, -2.2740626, -3.403873,  0.47273394,  2.995668,  ,
      -0.9620564,  0.6815842, -1.2293031,  0.39141572, -0.38473308,  ,
       1.3203373, -1.4432802,  1.0591677, -1.8518488, -0.16182004,  ,
      -2.5851288, -0.59578013,  0.949991404,  1.3326784, -0.23613031,  ,
      -0.06382382, -1.2549951, -1.870869,  0.81396717, -0.22290176,  ,
      -1.1564466,  1.1623534,  1.5485297, -2.269457,  0.47645366,  ,
      -0.5349471,  0.52957475, -0.5513478,  0.898683,  1.1934973,  ,
       3.0264792,  0.34337226,  1.9315122,  1.8684549,  0.25543526,  ,
       4.1305084, -1.0766245, -1.783644, -0.16379517, -2.5999024,  ])
```

(continues on next page)

(continued from previous page)

```

2.416761 , -0.25838712, -1.1239128 , 0.0806675 , 0.83068776,
1.7715666 , -2.6283164 , -1.3049829 , -0.43024743, -2.4749107 ,
-1.9384258 , -2.6229663 , 0.4618342 , 1.057843 , -0.37454468,
-2.819155 , 0.37408748, -2.9028332 , 2.9274974 , -1.8085246 ,
-2.426946 , 0.35030136, 3.2538562 , 0.7459811 , -0.20202728,
-0.42221448, -1.0485231 , -0.4264125 , 1.9211813 , 0.34063482,
-0.6206286 , -0.8266343 , -0.15856954, 0.83639336, 1.7174845 ,
0.20949106, -3.530206 , -0.23094082, 0.10127478, 3.302818 ,
-0.27075577, 1.287823 , 0.2584661 , 1.1708083 , -0.6232318 ,
0.5208176 , 2.811072 , 1.6814557 , -0.9234696 , -2.6148355 ,
-0.9125833 , -1.2430634 , 0.69818443, 1.6250277 , -1.756842 ,
-1.1041273 , -1.9850543 , 0.5356962 , 0.68917316, -1.5397801 ,
-1.5052795 , -0.24432632, -0.2890989 , 0.9179231 , 2.1375725 ,
1.9237542 , 0.62545127, -0.32388556, -1.931465 , -1.3508677 ,
-1.7556167 , 1.6963013 , 1.1774518 , -1.8767457 , -1.0262736 ,
0.63852936, -0.9283388 , -1.4915254 , 1.5973182 , 1.6986073 ,
-0.03490533, -1.1837475 , 0.10344972, -1.8024594 , 1.811266 ,
0.66438913, -2.8240383 , -0.103202 , 0.4685329 , 0.0105569 ,
-0.765648 , -1.0457621 , -0.30014274, -1.0558511 , 3.54757 ,
0.75964266, -3.3850913 , 1.3490607 , -0.07135642, -1.585785 ,
1.0023903 , -0.07662493, -1.1104459 , 2.163392 , 0.80607325,
1.6615813 , 2.6736174 , -4.110095 , -0.5684414 , 0.33428678,
-1.7811066 , 1.2861276 , -1.5071455 , -0.9563607 , -0.65822935,
-2.6227672 , 0.4704638 , 2.0072794 , 1.157605 , -0.15971239,
2.342728 , -2.7271366 , -1.313225 , 1.6697329 , 2.773483 ,
0.526841 , -0.35118487, -0.24909748, 0.89834905, 0.9178003 ,
-2.0666726 , -1.1335946 , -0.11632897, 0.9670184 , -0.19256516,
0.3340973 , -1.7204974 , 1.2734429 , 0.32692093, 1.1701695 ,
0.5955371 , -1.8531798 , -3.5321872 , -2.0358229 , 0.09263046,
-1.9751374 , 0.93768364, -1.0953122 , 0.29624373, 0.86260045,
-1.508276 , 4.900358 , 1.68672 , 1.238053 , 1.8480684 ,
-0.7564861 , -0.241455 , 1.911321 , -0.9201174 , -0.78837055,
-0.23056957, -0.22257806, 0.19382319, -1.0487769 , 0.3879833 ,
-2.8108838 , 0.90410644, -1.8231851 , -1.0712295 , 1.0877283 ,
3.359274 , 0.3441454 , 1.5441068 , 0.03014776, 3.7927184 ,
0.24134938, 1.225233 , 1.8023691 , -2.3324652 , -0.01004765,
0.41854864, -0.47862425, -0.8102291 , 0.73383987, -1.5596336 ,
0.17658985, 1.2276987 , -2.1915882 , -0.8576518 , 2.1577606 ],
dtype=float32)

```

```

# Compute sentence vectors
vecs = np.array([nlp(bus[permno]).vector for permno in bus.keys()])
store['vectors'] = vecs

```

```
vecs = store['vectors']
```

```

# Distance matrix
n = len(bus)
distances = np.zeros((n, n))
for row in range(n):
    for col in range(row, n):
        distances[row, col] = spatial.distance.cosine(vecs[row], vecs[col])
        distances[col, row] = distances[row, col]

```

Show most similar

```

def most_similar(p):
    dist = distances[permnos.index(p)]
    dist[permnos.index(p)] = max(dist) # to ignore own distance
    return univ.loc[permnos[np.argmin(dist)]]
for name in ['NVIDIA', 'APPLE COMPUTER', 'JNJ', 'EXXON MOBIL', 'AMERICAN EXPRESS']:
    p = find(name) ['permno'].iloc[-1]
    print(f" {most_similar(p)} ['comnam'] )' is most similar to '{name}'")

```

```

QUALCOMM INC' is most similar to 'NVIDIA'
MICROSOFT CORP' is most similar to 'APPLE COMPUTER'
PFIZER INC' is most similar to 'JNJ'
PIONEER NATURAL RESOURCES CO' is most similar to 'EXXON MOBIL'
U S BANCORP DEL' is most similar to 'AMERICAN EXPRESS'

```

27.3 Density-based clustering

27.3.1 t-SNE visualization

T-distributed Stochastic Neighbor Embedding (t-SNE) is a tool to visualize high-dimensional data. It converts similarities between data points to joint probabilities and tries to minimize the Kullback-Leibler divergence between the joint probabilities of the low-dimensional embedding and the high-dimensional data.

<https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html>

```

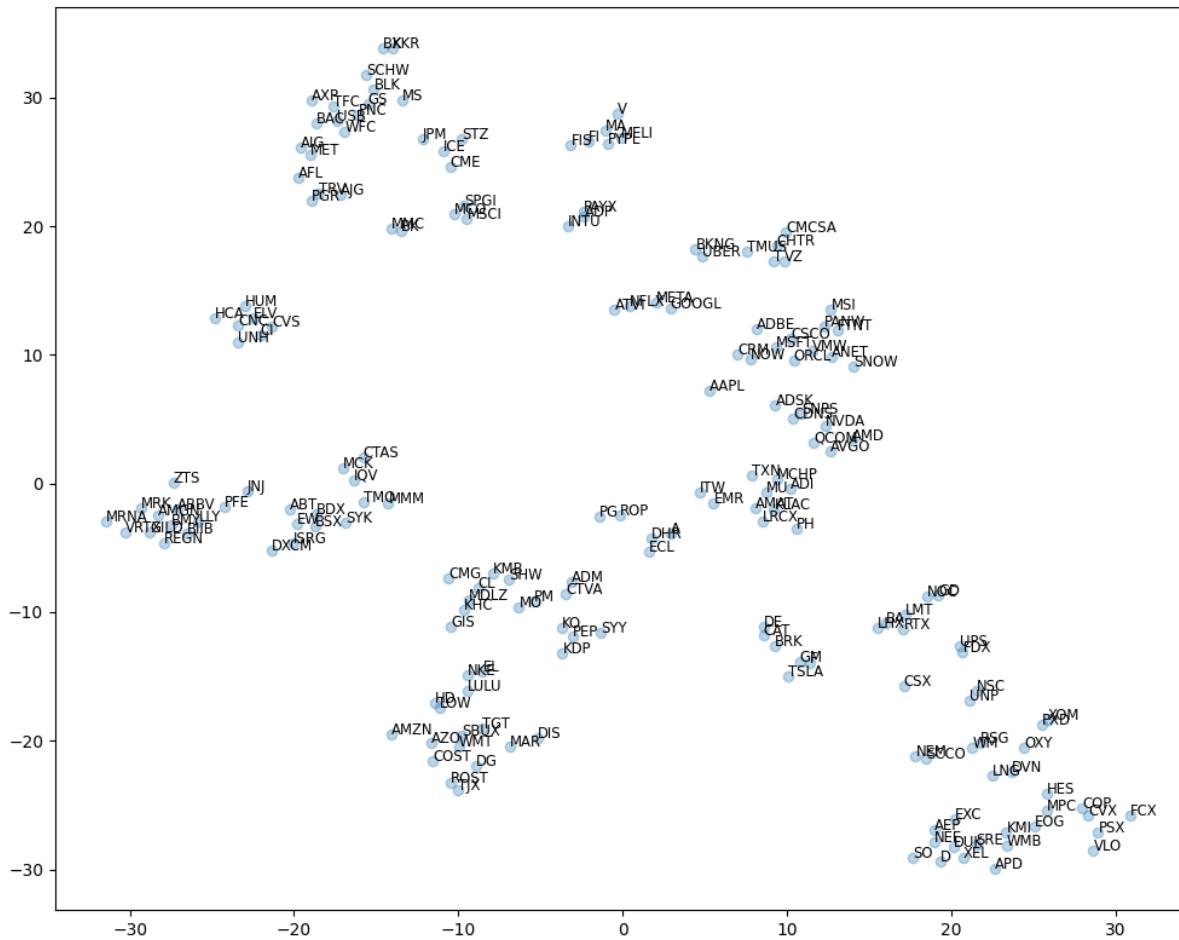
Z = TSNE(n_components=2, perplexity=10, random_state=42) \
    .fit_transform(vecs)

```

```

fig, ax = plt.subplots(figsize=(10, 8))
ax.scatter(Z[:, 0], Z[:, 1], color="C0", alpha=.3)
for text, x, y in zip(tickers, Z[:, 0], Z[:, 1]):
    ax.annotate(text=text, xy=(x, y), fontsize='small')
plt.tight_layout()

```



27.3.2 DBSCAN

Density-based spatial clustering of applications with noise

TODO: see wikipedia, most cited, advantages, how to choose eps

```
# eps is the most important parameter for DBSCAN
eps = 4
db = cluster.DBSCAN(eps=eps)      # default eps
db.fit(Z)
n_clusters = len(set(db.labels_).difference({-1}))
n_noise = np.sum(db.labels_ == -1)
DataFrame(dict(clusters=n_clusters, noise=n_noise, eps=eps), index=['DBSCAN'])
```

| | clusters | noise | eps |
|--------|----------|-------|-----|
| DBSCAN | 9 | 8 | 4 |

```
cmap = ColorMap(n_clusters)
fig, ax = plt.subplots(figsize=(10, 8))
# plot core samples with larger marker size
ax.scatter(Z[db.core_sample_indices_, 0],
```

(continues on next page)

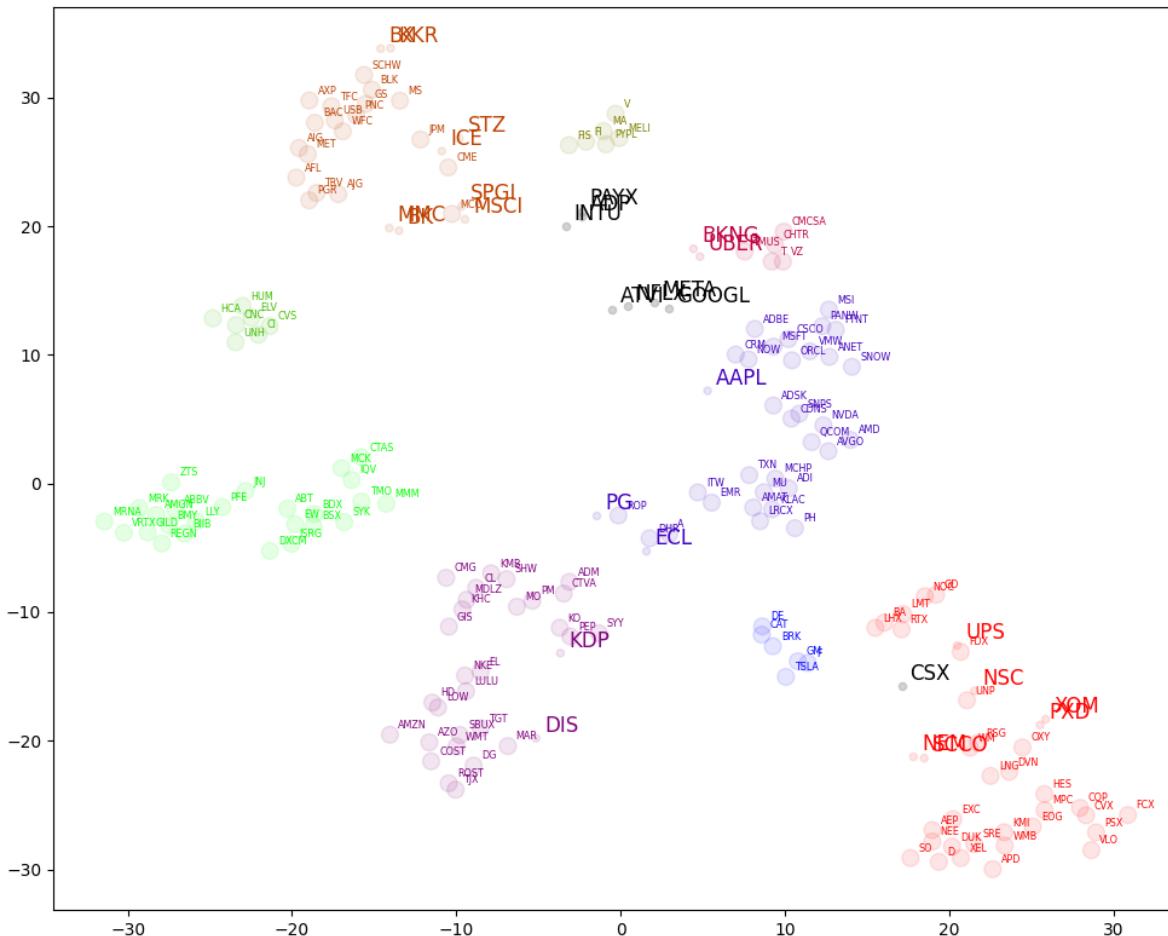
(continued from previous page)

```

Z[db.core_sample_indices_, 1],
c=cmap[db.labels_[db.core_sample_indices_]],
alpha=.1, s=100, edgecolors=None)
# plot non-core samples with smaller marker size
non_core = np.ones_like(db.labels_, dtype=bool)
non_core[db.core_sample_indices_] = False
non_core[db.labels_ < 0] = False
ax.scatter(Z[non_core, 0], Z[non_core, 1], c=cmap[db.labels_[non_core]],
alpha=.1, s=20, edgecolors=None)
# plot noise samples
ax.scatter(Z[db.labels_ < 0, 0], Z[db.labels_ < 0, 1], c="darkgrey",
alpha=.5, s=20, edgecolors=None)

# annotate with tickers not in core samples
for i, (t, c, xy) in enumerate(zip(tickers, db.labels_, Z)):
    if i in db.core_sample_indices_:
        ax.annotate(text=t, xy=xy+.5, color=cmap[c], fontsize='xx-small')
    elif c == -1:
        ax.annotate(text=t, xy=xy+.5, color='black', fontsize='large')
    else:
        ax.annotate(text=t, xy=xy+.5, color=cmap[c], fontsize='large')
plt.tight_layout()

```



```
print("Samples tagged as noise:")
univ.loc[np.array(permnos)[db.labels_ < 0]].sort_values('naics')
```

Samples tagged as noise:

| permno | cap | capco | decile | nyse | siccd | prc | naics | \ |
|--------|--------------|--------------|--------|------|-------|------|--------|--------|
| 90319 | 5.254979e+08 | 1.055039e+09 | | 1 | False | 7375 | 88.23 | 334419 |
| 62148 | 6.400468e+07 | 6.400468e+07 | | 1 | False | 4011 | 30.98 | 482111 |
| 79678 | 6.003189e+07 | 6.003189e+07 | | 1 | False | 7370 | 76.55 | 511210 |
| 78975 | 1.093416e+08 | 1.093416e+08 | | 1 | False | 7370 | 389.22 | 511210 |
| 44644 | 9.898358e+07 | 9.898358e+07 | | 1 | False | 7374 | 238.86 | 518210 |
| 13407 | 2.704040e+08 | 2.704040e+08 | | 1 | False | 7375 | 120.34 | 519190 |
| 89393 | 1.313239e+08 | 1.313239e+08 | | 1 | False | 7841 | 294.88 | 532282 |
| 61621 | 4.165557e+07 | 4.165557e+07 | | 1 | False | 8700 | 115.56 | 541219 |

| permno | comnam | ticker |
|--------|-------------------------------|--------|
| 90319 | ALPHABET INC | GOOGL |
| 62148 | C S X CORP | CSX |
| 79678 | ACTIVISION BLIZZARD INC | ATVI |
| 78975 | INTUIT INC | INTU |
| 44644 | AUTOMATIC DATA PROCESSING INC | ADP |
| 13407 | META PLATFORMS INC | META |
| 89393 | NETFLIX INC | NFLX |
| 61621 | PAYCHEX INC | PAYX |

Ignore the rest (may go to DAN)

```
vec = (nlp.vocab['king'].vector
       - nlp.vocab['man'].vector
       + nlp.vocab['woman'].vector)
print(vec.shape)
sim = nlp.vocab.vectors.most_similar(vec[None,:], n=10)
[nlp.vocab.strings[hashkey] for hashkey in sim[0][0]]
```

```
# Load pretrained embeddings
emb = nn.Embedding\
    .from_pretrained(torch.FloatTensor(nlp.vocab.vectors.data))
```

```
# test for Spacy.nlp and torch.embeddings
test_vocab = ['king', 'man', 'woman', 'queen', 'e9s82j']
for w in test_vocab:
    vocab_id = nlp.vocab.strings[w]
    spacy_vec = nlp.vocab[w].vector
    row = nlp.vocab.vectors.key2row.get(vocab_id, None) # dict
    if row is None:
        print('{} is oov'.format(w))
        continue
    vocab_row = torch.tensor(row, dtype=torch.long)
    embed_vec = emb(vocab_row)
    print(np.allclose(spacy_vec, embed_vec.detach().numpy()))
```

```

for key, row in nlp.vocab.vectors.key2row.items():
    if row == 0:
        print(nlp.vocab.strings[key])

```

27.4 Exploring Spacy

“”“ hashkey = nlp.vocab.strings[v]: general hash table between vocab strings and ids vec = nlp.vocab[v].vector: np array with word embedding vector from vocab string row = nlp.vocab.vectors.key2row: dict from word’s hashkey to int

emb = nn.Embedding.from_pretrained(torch.FloatTensor(nlp.vocab.vectors.data)) emb(row) == vec : equivalence of torch embedding and spacy vector

token = nlp('king man queen woman')[0] token.lower : hashkey token.lower_ : str token.lex_id : row of word vector token.has_vector : has word vector representation “”“

```

doc = nlp('king queen man woman a23kj4j')
line = [tok.lex_id for tok in doc
        if not(tok.is_stop or tok.is_punct or tok.is_oov or tok.is_space) ]

vec = (nlp.vocab['king'].vector
       - nlp.vocab['man'].vector
       + nlp.vocab['woman'].vector)
print(vec.shape)
sim = nlp.vocab.vectors.most_similar(vec[None,:], n=10)
[nlp.vocab.strings[hashkey] for hashkey in sim[0][0]]

# Load pretrained embeddings
emb = nn.Embedding\
    .from_pretrained(torch.FloatTensor(nlp.vocab.vectors.data))

# test for Spacy.nlp and torch.embeddings
test_vocab = ['king', 'man', 'woman', 'queen', 'e9s82j']
for w in test_vocab:
    vocab_id = nlp.vocab.strings[w]
    spacy_vec = nlp.vocab[w].vector
    row = nlp.vocab.vectors.key2row.get(vocab_id, None) # dict
    if row is None:
        print('{} is oov'.format(w))
        continue
    vocab_row = torch.tensor(row, dtype=torch.long)
    embed_vec = emb(vocab_row)
    print(np.allclose(spacy_vec, embed_vec.detach().numpy()))

for key, row in nlp.vocab.vectors.key2row.items():
    if row == 0:
        print(nlp.vocab.strings[key])

```

CHAPTER
TWENTYEIGHT

CLASSIFICATION

May your choices reflect your hopes, not your fears – Nelson Mandela

Concepts:

- Text classification
- Classification models
- Evaluation

References:

- ISLR

```
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
# %matplotlib qt
VERBOSE = 0
store = Store(paths['scratch'], ext='pkl')
```

```
sql = SQL(**credentials['sql'], verbose=VERBOSE)
user = SQL(**credentials['user'], verbose=VERBOSE)
bd = BusDay(sql)
```

(continues on next page)

(continued from previous page)

```
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)
```

Last FamaFrench Date 2024-03-28 00:00:00

28.1 Text classification

```
# Retrieve universe of stocks
univ = crsp.get_universe(bd.endmo(20221231))
```

```
# 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 bus10K's
item, form = 'bus10K', '10-K'
rows = DataFrame(ed.open(form=form, item=item))
found = rows[rows['date'].between(20230101, 20231231)] \
    .drop_duplicates(subset=['permno'], keep='last') \
    .set_index('permno')
```

28.1.1 Tagging and Lemmatizing

Lemmatize using WordNet's built-in `morphy` function.

```
# !nltk.download('averaged_perceptron_tagger')
lemmatizer = WordNetLemmatizer()
bus = {}
for permno in tqdm(found.index):
    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]
```

(continues on next page)

(continued from previous page)

```

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

```

100% |██████████| 4625/4625 [12:55<00:00, 5.97it/s]

```

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))

```

28.1.2 Vectorizer

- TfIdf

Train/test split of corpus

```

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 | 705 | 176 | 0.25 | 0.25 |
| Other | 609 | 153 | 0.21 | 0.21 |
| HiTec | 565 | 141 | 0.20 | 0.20 |
| Manuf | 275 | 69 | 0.10 | 0.10 |
| Shops | 257 | 64 | 0.09 | 0.09 |
| Durbl | 131 | 33 | 0.05 | 0.05 |
| NoDur | 116 | 29 | 0.04 | 0.04 |
| Enrgy | 75 | 19 | 0.03 | 0.03 |
| Utils | 74 | 18 | 0.03 | 0.03 |
| Telcm | 40 | 10 | 0.01 | 0.01 |

```

# 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,

```

(continues on next page)

(continued from previous page)

```

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: | (2847, 10181) | (712, 10181) |

28.2 Classification models

- `class_weight`: The “balanced” mode uses the values of `y` to automatically adjust weights inversely proportional to class frequencies in the input data as `n_samples / (n_classes * np.bincount(y))`.

```

results = dict()
# results = store['classification']

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:]

```

28.2.1 Naive Bayes

Generative model

For classification, there are generally two types of approaches:

- Generative: Estimate probability model, then define the classifier – e.g. Naive-Bayes
- Discriminative: Directly define the classifier – e.g. Logistic regression, Perceptron, Support vector machine

Binomial Naive Baye

Naive Bayes is a very simple classifier, yet it still takes all the feature evidence into account. It is very efficient in terms of storage space and computation time. Training consists only of storing counts of classes and feature occurrences as each example is seen. It performs surprisingly well for classification on many real-world tasks, because the violation of the independence assumption tends not to hurt classification performance.

However, the violation of the independence assumption makes probability estimates more extreme: the probability will be overestimated for the correct class and underestimated for the incorrect class(es). This does become a problem if we're going to be using the probability estimates themselves – so Naive Bayes should be used with caution for actual decision-making with costs and benefits.

Bag of words assumes position doesn't matter, hence the features only encode word identity and not position. Naive Bayes assumption is the conditional independence assumption that the probabilities $P(f_i|c)$ of each feature f_i are independent given the class c and hence can be 'naively' multiplied as follows: $P(f_1, f_2, \dots, f_n|c) = P(f_1|c) \cdot P(f_2|c) \cdot \dots \cdot P(f_n|c)$

Since naive Bayes naively multiplies all the feature likelihoods together, zero probabilities in the likelihood term for any class will cause the probability of the class to be zero, no matter the other evidence. The simplest solution is the add-one (Laplace) smoothing.

Let N_c be the number of documents in our training data with class c and N_{doc} be the total number of documents. Then: $P(c) = N_c/N_{doc}$ $\hat{c} = \operatorname{argmax}_{c \in C} \log P(c) \sum_{i \in n} \log P(w_i|c)$, where the maximum likelihood estimate of the probability of frequency of word w_i is $P(w_i|c) = \frac{\operatorname{count}(w_i, c)}{\sum_{w \in V} \operatorname{count}(w, c)}$

Multinomial Naive Bayes

Laplace smoothing

```
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 | 0.760801 | 0.713483 | 0.003135 | 0.016976 |

28.2.2 Perceptron

- 0-1 loss
- Elasticnet penalty
- OVR

The perceptron update is:

- $\hat{y} \leftarrow \text{sign}(wx)$
 - $w \leftarrow w + \alpha x$ if $y = +1$ and $y \neq \hat{y}$
 - $w \leftarrow w - \alpha x$ if $y = -1$ and $y \neq \hat{y}$
- $\Rightarrow w \leftarrow w + \alpha x (y - \hat{y})/2$ where $y \in \{-1, +1\}$ #

```
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 | 0.760801 | 0.713483 | 0.003135 | 0.016976 |
| perceptron | 0.949069 | 0.754213 | 0.003511 | 0.572992 |

28.2.3 Support Vector Machine

Margin

- Hinge loss

Kernel

- Linear SVC
- OVR

```
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 | 0.760801 | 0.713483 | 0.003135 | 0.016976 |
| perceptron | 0.949069 | 0.754213 | 0.003511 | 0.572992 |
| linearsvc | 0.987706 | 0.820225 | 0.005223 | 0.373052 |

28.2.4 Logistic Regression

Cross-entropy loss

- logistic loss and binomial
- Softmax
- Multinomial

The logistic regression update 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 $y = 0$
- $\Rightarrow w \leftarrow w + \alpha x (y - P(y = 1|x))$ where $y \in \{0, 1\}$

```
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 | 0.760801 | 0.713483 | 0.003135 | 0.016976 |
| perceptron | 0.949069 | 0.754213 | 0.003511 | 0.572992 |
| linearsvc | 0.987706 | 0.820225 | 0.005223 | 0.373052 |
| logistic | 0.897085 | 0.807584 | 0.004926 | 2.042012 |

28.3 Evaluation

Imbalanced data set

Precision

Recall

F1 score

The F1 score = $2(precision \times recall)/(precision + recall)$ can be interpreted as a harmonic mean of the precision and recall. In the multi-class and multi-label case, this is the weighted average of the F1 score of each class.

28.3.1 Precision, Recall and F1-score

```
# Plot precision, recall, f1

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.
  py:1509: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0
  in labels with no predicted samples. Use 'zero_division' parameter to control
  this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/terence/env3.11/lib/python3.11/site-packages/sklearn/metrics/_classification.
  py:1509: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0
  in labels with no predicted samples. Use 'zero_division' parameter to control
  this behavior.
  _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())
```

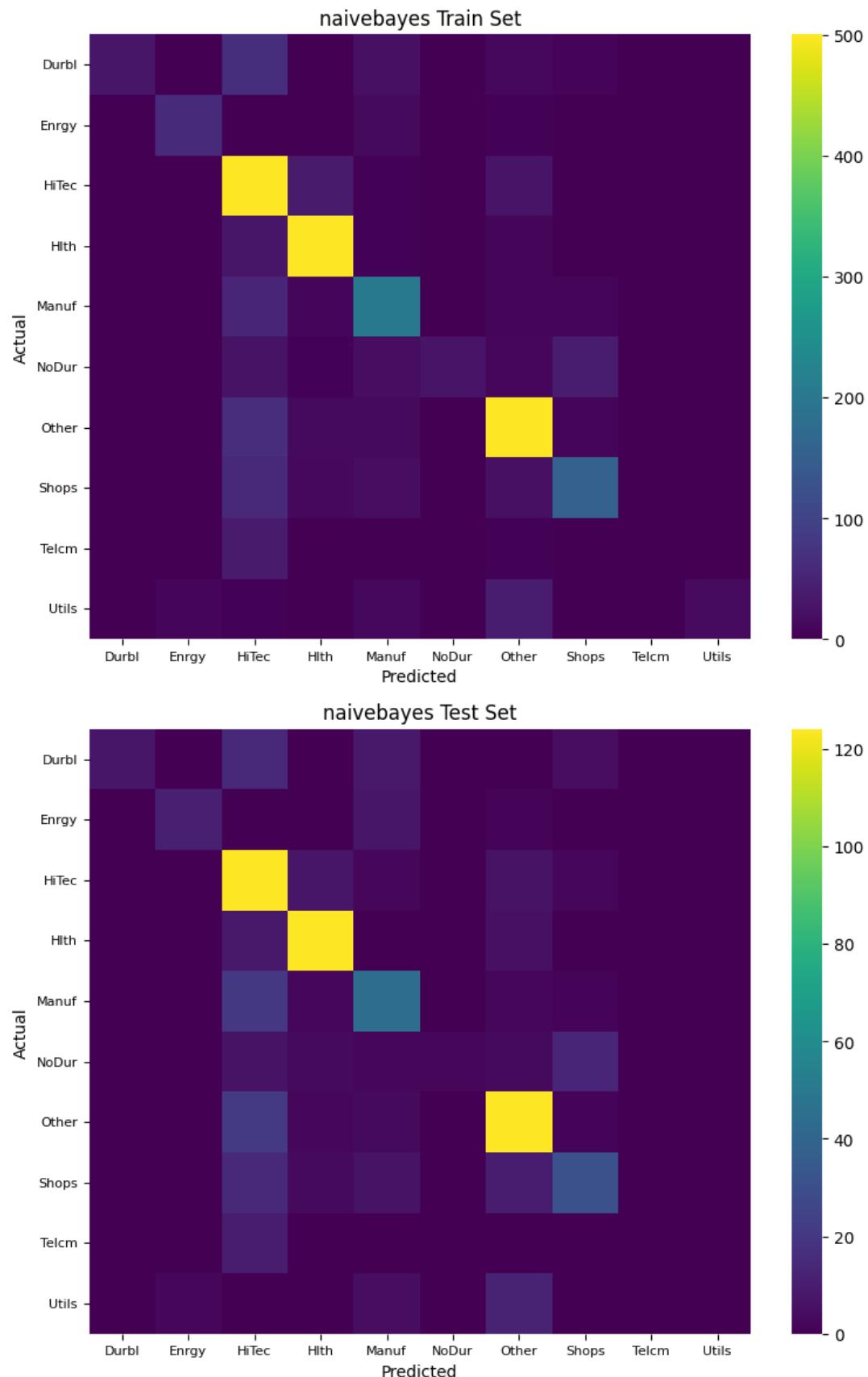
| | Precision | | Recall | | F1-score | |
|------------|-----------|----------|----------|----------|----------|----------|
| | train | test | train | test | train | test |
| naivebayes | 0.755631 | 0.626188 | 0.542914 | 0.461119 | 0.571358 | 0.470883 |
| perceptron | 0.948395 | 0.715983 | 0.926773 | 0.705942 | 0.936185 | 0.707481 |
| linearsvc | 0.983966 | 0.817232 | 0.987891 | 0.809486 | 0.985768 | 0.810462 |
| logistic | 0.893775 | 0.824410 | 0.849069 | 0.755437 | 0.868002 | 0.780065 |

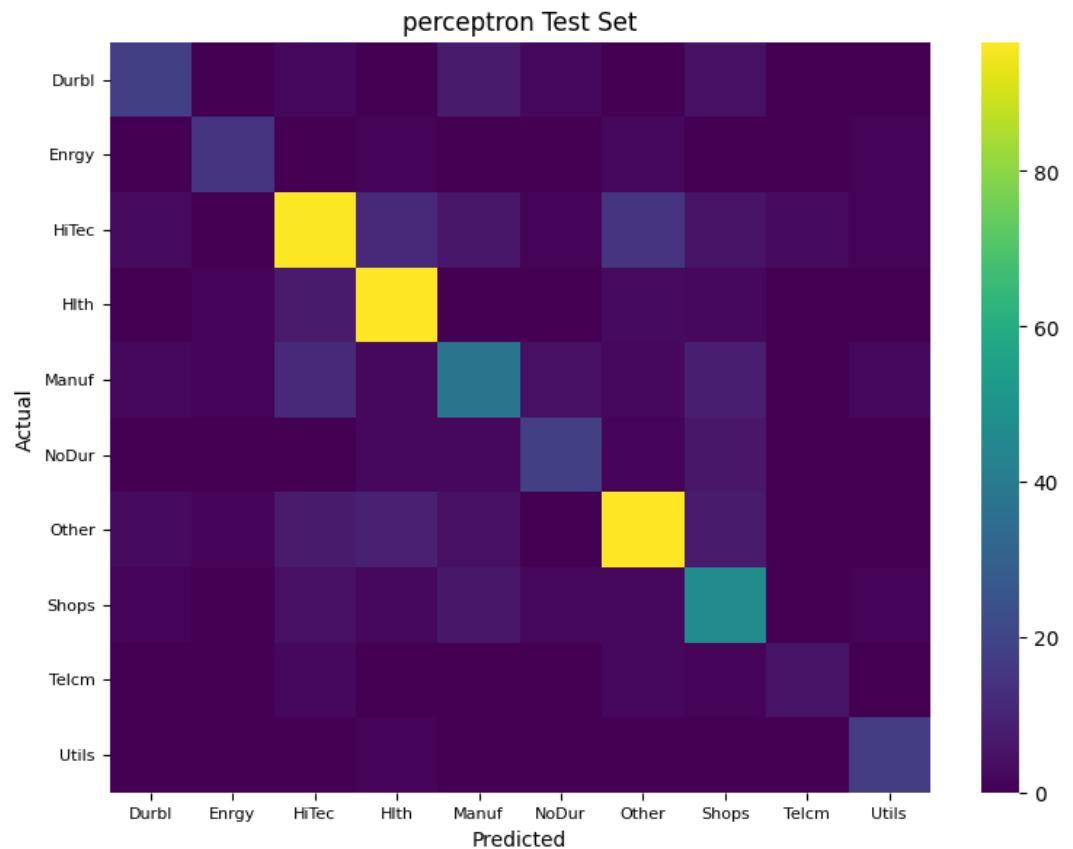
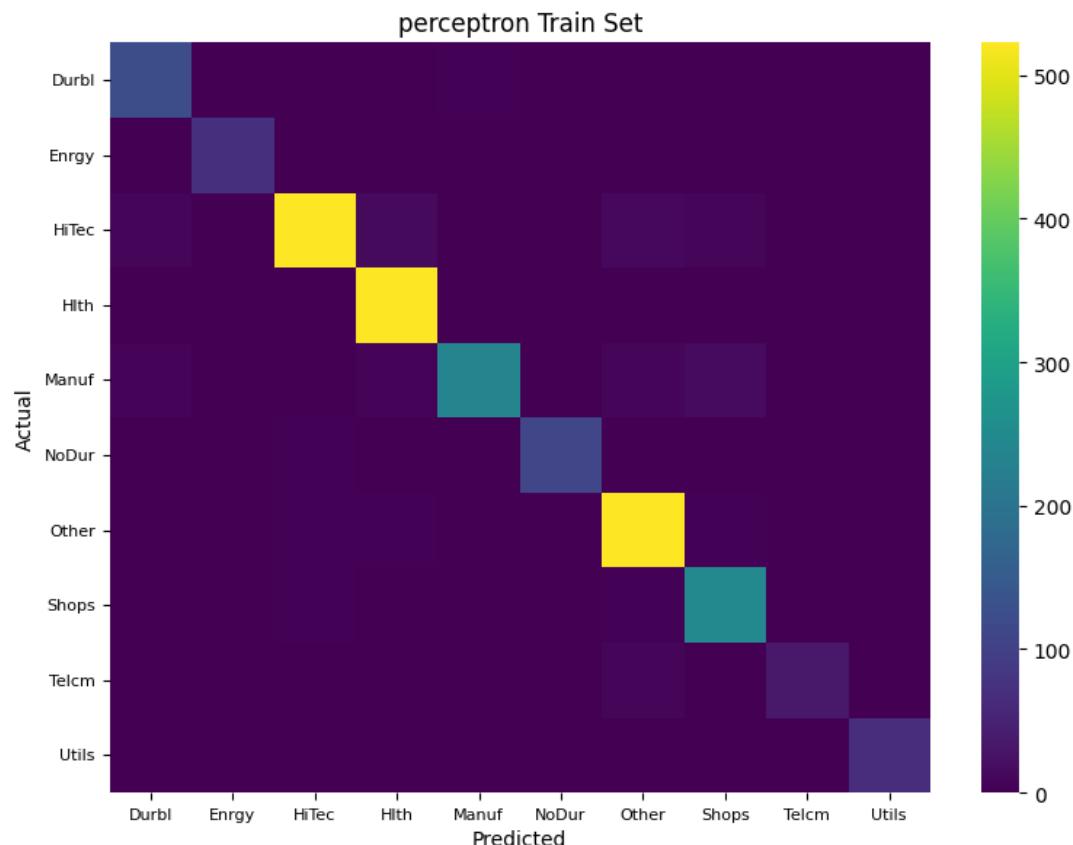
28.3.2 Confusion Matrix

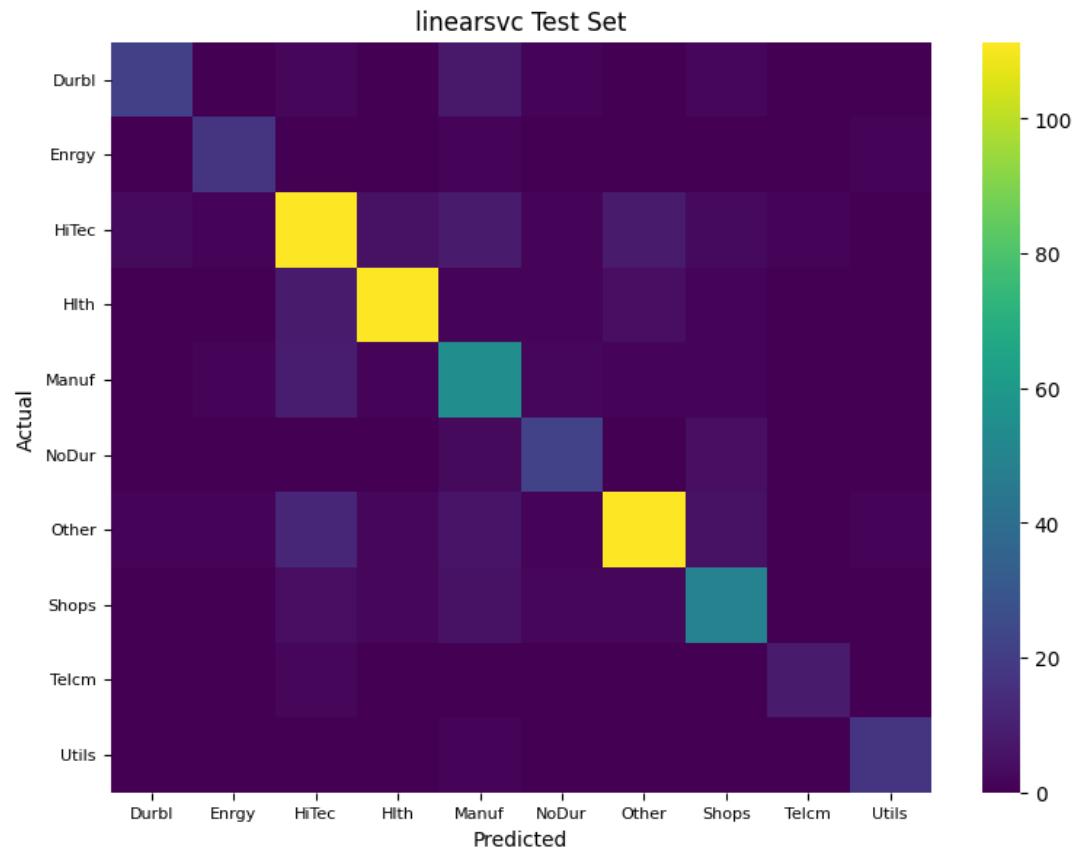
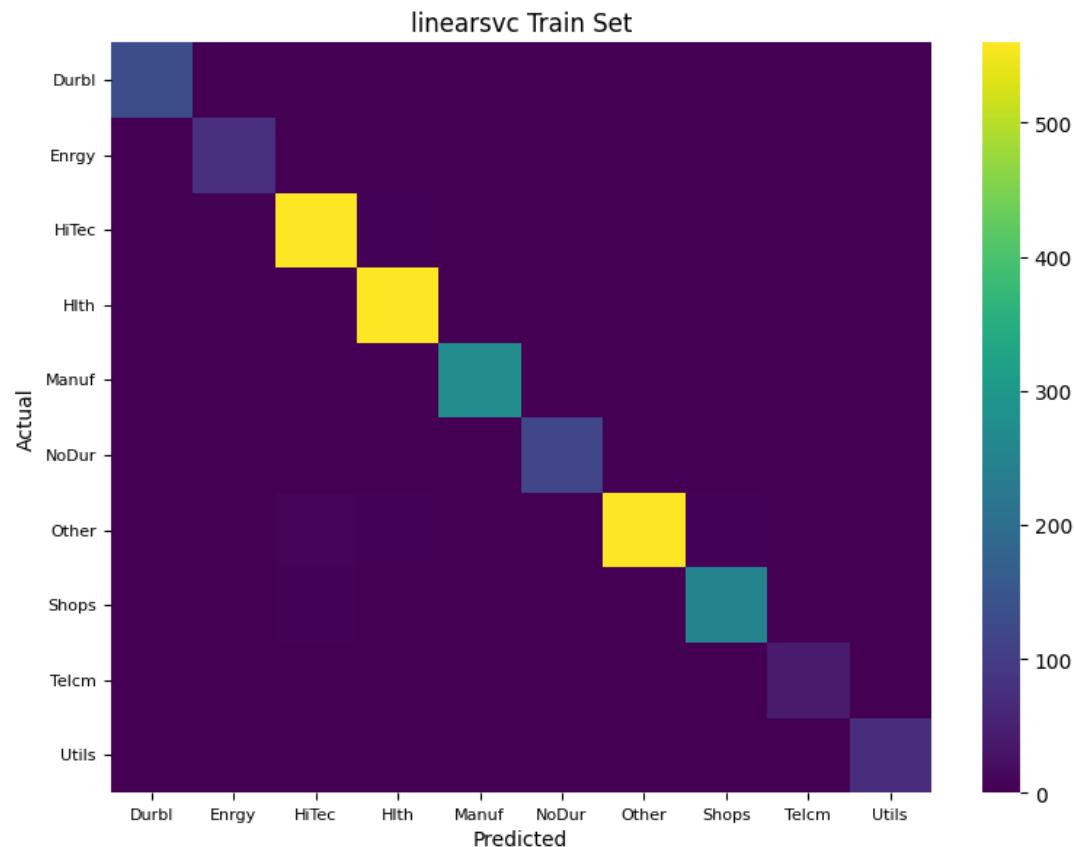
```

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()

```







28.3.3 Feature Importance

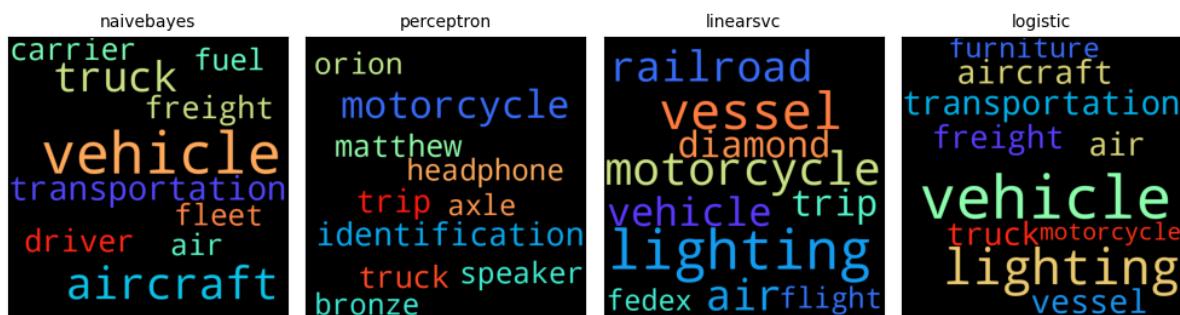
- Plot by event in each row, then by model in column
- exponentiate NAIVE BAYES to probabilities (generative)

WordCloud

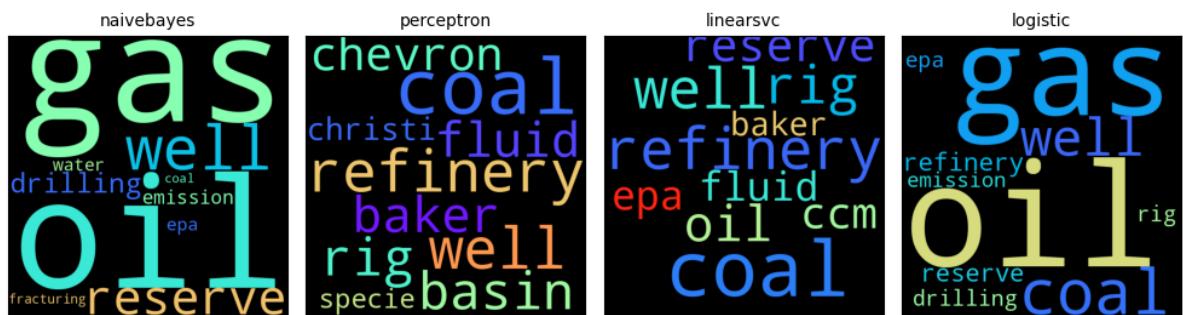
```
wc = WordCloud(height=500, width=500, prefer_horizontal=1.0, colormap='rainbow')

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.xaxis.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()
```

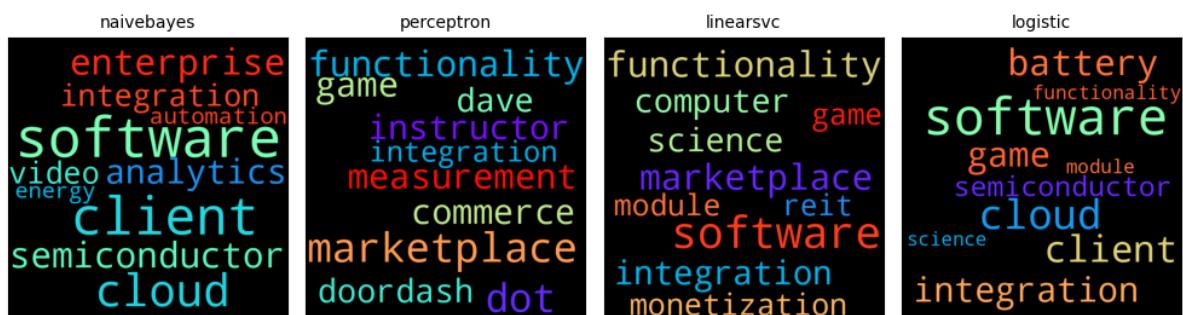
Durbl



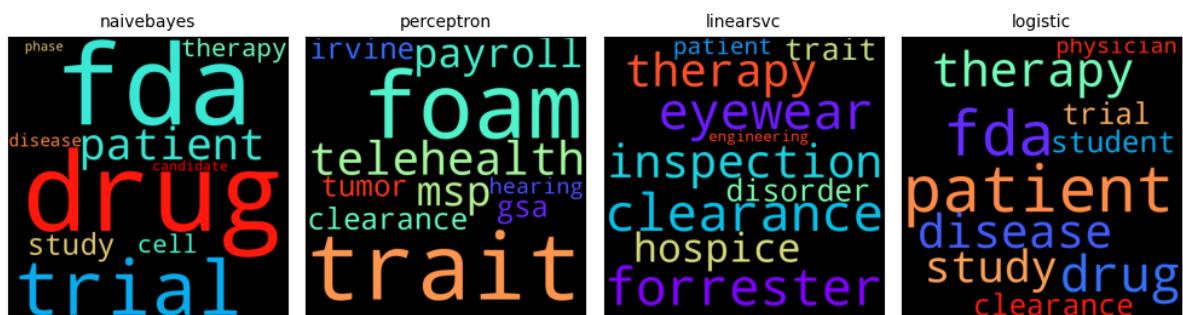
Enrgy



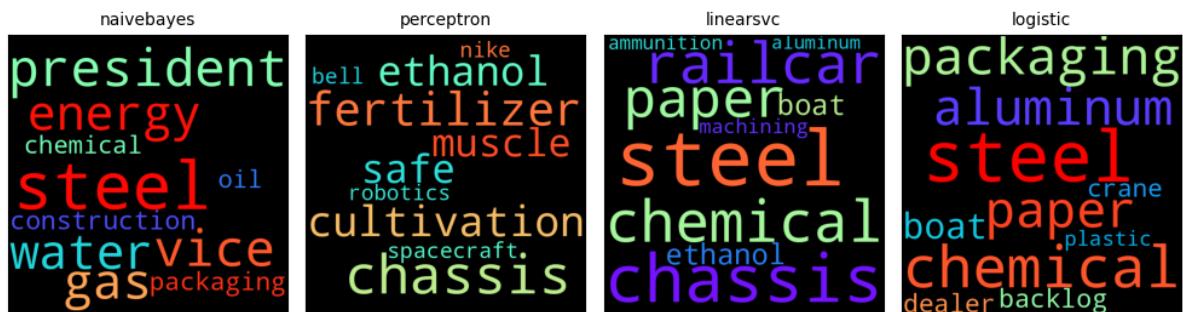
HiTec



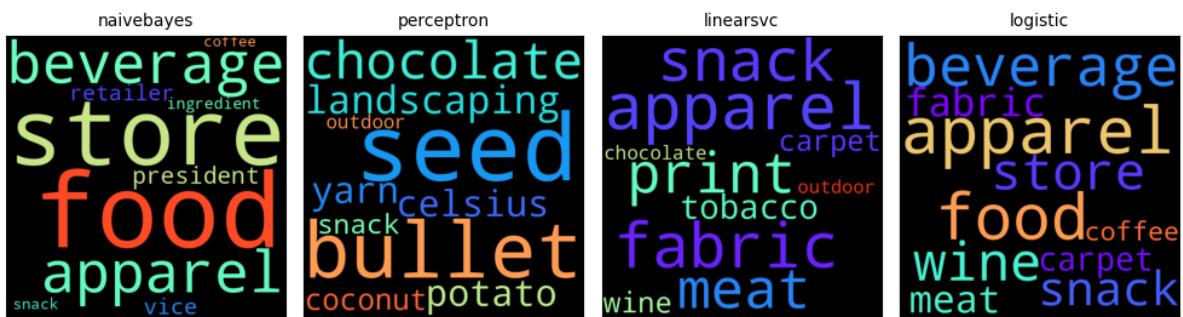
Hlth



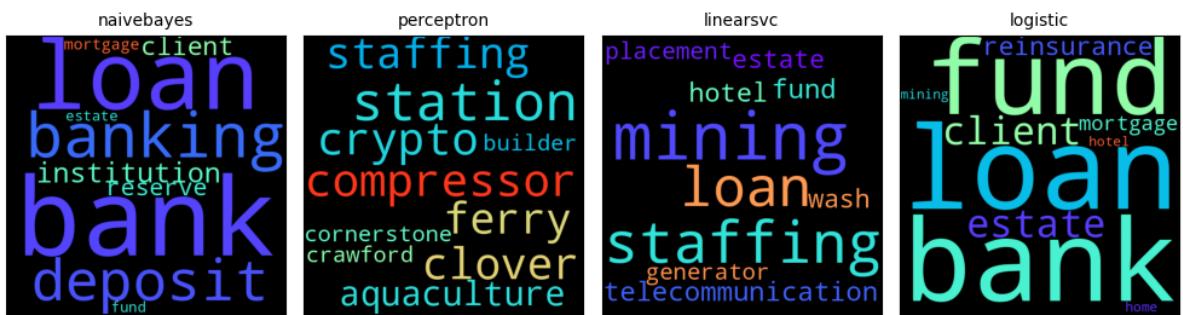
Manuf



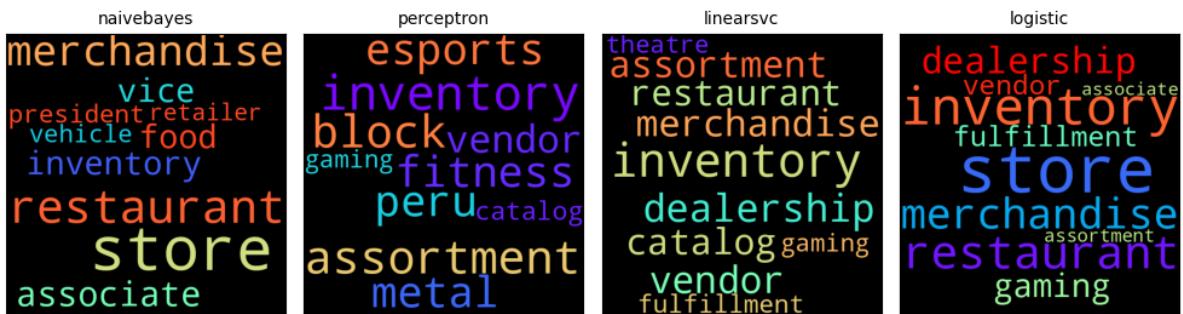
NoDur



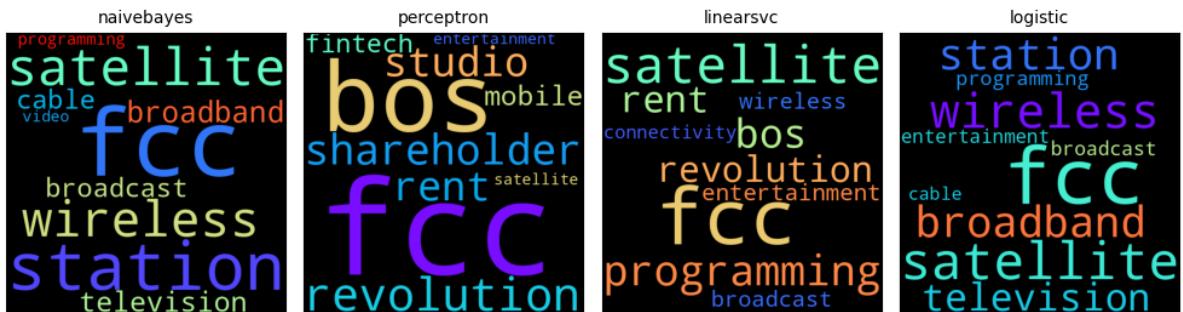
Other



Shops



Telcm



CHAPTER
TWENTYNINE

REGRESSION

Concepts:

- Regression Models
- Hyperparameter Tuning
- Information Criterion

```
import numpy as np
import pandas as pd
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
from pandas.api.types import is_list_like, is_numeric_dtype
import statsmodels.api as sm
from sklearn.metrics import mean_squared_error
import time
from finds.readers.alfred import Alfred, fred_qd, fred_md
from finds.utils import plot_date
from secret import credentials, paths
# %matplotlib qt
VERBOSE = 0
```

```
alf = Alfred(api_key=credentials['fred']['api_key'], verbose=-1)
```

```
# Get FRED-MD data
freq = 'M'
beg = 19640701 # 19620701
split_date = 20221231
df, t = fred_md() # transforms = t['transform']
end = df.index[-2]
```

FRED-MD vintage: monthly/current.csv

```
# 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 transformations
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: (716, 126) dropna: (716, 122)
series_id
ACOGNO      332
UMCSENT     163
TWEXAFEGSMTH 103
ANDENO      44
Name: count, dtype: int64
```

| | RPI | W875RX1 | DPCERA3M086SBEA | CMRMTSPL | RETAIL | INDPRO | \ |
|----------|-----------------|-----------------|-----------------|---------------|-----------|-----------------|---|
| 19640731 | 0.005422 | 0.005404 | 0.007343 | 0.019986 | 0.004947 | 0.006551 | |
| 19640831 | 0.005644 | 0.006061 | 0.005992 | -0.020139 | 0.013974 | 0.006508 | |
| 19640930 | 0.004123 | 0.004205 | -0.004164 | 0.027173 | 0.009372 | 0.003699 | |
| 19641031 | 0.000555 | 0.000650 | 0.006499 | -0.013866 | -0.039421 | -0.013947 | |
| 19641130 | 0.006703 | 0.007352 | -0.009111 | -0.009745 | 0.009335 | 0.030432 | |
| ... | ... | ... | ... | ... | ... | ... | \ |
| 20231031 | 0.002011 | 0.002671 | 0.001534 | -0.001822 | -0.001868 | -0.007083 | |
| 20231130 | 0.003920 | 0.004822 | 0.004145 | 0.006184 | 0.001042 | 0.003600 | |
| 20231231 | 0.002056 | 0.002097 | 0.004578 | 0.006499 | 0.003631 | -0.003146 | |
| 20240131 | 0.006063 | 0.002633 | -0.003154 | -0.013142 | -0.010902 | -0.007865 | |
| 20240229 | -0.000623 | -0.001369 | 0.004809 | 0.004932 | 0.010128 | 0.004399 | |
| | IPFPNSS | IPFINAL | IPCONGD | IPDCONGD | ... | DDURRG3M086SBEA | \ |
| 19640731 | 0.010344 | 0.011309 | 0.014078 | 0.016436 | ... | 0.000667 | |
| 19640831 | -0.000936 | -0.000939 | -0.001865 | 0.007223 | ... | -0.002031 | |
| 19640930 | -0.004690 | -0.003758 | -0.011266 | -0.023659 | ... | 0.000728 | |
| 19641031 | -0.010403 | -0.013272 | -0.020031 | -0.109873 | ... | -0.001712 | |
| 19641130 | 0.029040 | 0.031929 | 0.036883 | 0.128118 | ... | 0.004137 | |
| ... | ... | ... | ... | ... | ... | ... | \ |
| 20231031 | -0.003736 | -0.005295 | -0.006291 | -0.043572 | ... | -0.001469 | |
| 20231130 | 0.004855 | 0.008046 | 0.007774 | 0.040181 | ... | -0.002269 | |
| 20231231 | -0.004377 | -0.004537 | -0.005309 | 0.009311 | ... | 0.000147 | |
| 20240131 | -0.004417 | -0.003318 | -0.000861 | -0.024891 | ... | 0.006598 | |
| 20240229 | 0.001838 | -0.001725 | -0.008632 | 0.019836 | ... | -0.000164 | |
| | DNDGRG3M086SBEA | DSERRG3M086SBEA | CES0600000008 | CES2000000008 | ... | | \ |

(continues on next page)

(continued from previous page)

| | | | | | |
|--|-----------|-----------|-----------|-----------|---------|
| 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 | |
| ... | ... | ... | ... | ... | |
| 20231031 | -0.006139 | -0.002998 | -0.002355 | 0.001729 | |
| 20231130 | -0.003030 | 0.000679 | 0.007291 | 0.004863 | |
| 20231231 | 0.004876 | 0.000458 | -0.004356 | -0.007758 | |
| 20240131 | -0.002566 | 0.004037 | -0.000355 | 0.008535 | |
| 20240229 | 0.010236 | -0.004448 | -0.001327 | -0.010808 | |
| CES3000000008 DTCOLNVHFN M DTCTHFN M 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 |
| ... | ... | ... | ... | ... | ... |
| 20231031 | -0.004147 | 0.000003 | -0.000566 | -0.004573 | 19.0462 |
| 20231130 | 0.007839 | -0.000492 | 0.000069 | 0.008778 | 13.8563 |
| 20231231 | -0.000080 | 0.000385 | 0.000526 | 0.018190 | 12.6960 |
| 20240131 | -0.005941 | -0.002851 | -0.000590 | -0.009783 | 13.3453 |
| 20240229 | 0.002553 | -0.000785 | -0.001406 | -0.005802 | 13.8808 |

[716 rows x 122 columns]

Data after the split date (December 2022) are held out for testing, and are excluded from the training data set

```
# 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)
```

```
# collect final fitted models
```

(continues on next page)

(continued from previous page)

```
test = Series(name='test', dtype=float)      # collect test and train errors
train = Series(name='train', dtype=float)
final_models = {}
```

29.1 Regression models

29.1.1 Forward Selection

AIC

BIC

```
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,
                        '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

| select | aic | \ |
|------------|-----------|---|
| CLAIMS_1 | -4768.644 | |
| M2REAL_2 | -4810.935 | |
| USWTRADE_2 | -4840.491 | |
| SRVPRD_2 | -4861.002 | |

(continues on next page)

(continued from previous page)

| | | |
|--|-----------------|-----------|
| 5-Year Treasury Constant Maturity Minus Federal... | T5YFFM_3 | -4877.677 |
| Total Business: Inventories to Sales Ratio | ISRATIO_2 | -4891.445 |
| All Employees, Service-Providing | SRVPRD_1 | -4908.643 |
| All Employees, Wholesale Trade | USWTRADE_1 | -4924.049 |
| All Employees, Financial Activities | USFIRE_2 | -4935.217 |
| All Employees, Retail Trade | USTRADE_1 | -4947.335 |
| Industrial Production: Non-Durable Consumer Ene... | IPB51222S_3 | -4954.897 |
| Industrial Production: Non-Durable Goods Materials | IPNMAT_3 | -4966.446 |
| Canadian Dollars to U.S. Dollar Spot Exchange Rate | EXCAUS_3 | -4976.370 |
| M1 | M1SL_3 | -4986.332 |
| Consumer Motor Vehicle Loans Owned by Finance C... | DTCOLNVHFN_M_3 | -4996.521 |
| Nonrevolving consumer credit to Personal Income | CONSPI_2 | -5003.756 |
| Moody's Seasoned Aaa Corporate Bond Yield | AAA_1 | -5009.808 |
| Moody's Seasoned Baa Corporate Bond Yield | BAA_1 | -5017.239 |
| All Employees, Mining, Quarrying, and Oil and G... | CES1021000001_3 | -5022.081 |
| Manufacturers' Unfilled Orders: Durable Goods | AMDMUO_2 | -5028.370 |
| Capacity Utilization: Manufacturing (SIC) | CUMFNS_1 | -5033.466 |
| Industrial Production: Manufacturing (SIC) | IPMANSICS_1 | -5058.855 |
| All Employees, Nondurable Goods | NDMANEMP_1 | -5073.971 |
| Industrial Production: Materials | IPMAT_2 | -5081.590 |
| Industrial Production: Equipment: Business Equi... | IPBUSEQ_2 | -5088.804 |
| Average Hourly Earnings of Production and Nonsu... | CES2000000008_1 | -5094.599 |
| Average Weekly Overtime Hours of Production and... | AWOTMAN_2 | -5101.126 |
| All Employees, Mining, Quarrying, and Oil and G... | CES1021000001_1 | -5106.815 |
| Industrial Production: Consumer Goods | IPCONGD_2 | -5111.903 |
| Real Estate Loans, All Commercial Banks | REALLN_1 | -5116.565 |
| bic rsquared \ | | |
| Initial Claims | -4764.094 | 0.357 |
| Real M2 Money Stock | -4801.836 | 0.396 |
| All Employees, Wholesale Trade | -4826.842 | 0.423 |
| All Employees, Service-Providing | -4842.803 | 0.441 |
| 5-Year Treasury Constant Maturity Minus Federal... | -4854.929 | 0.456 |
| Total Business: Inventories to Sales Ratio | -4864.147 | 0.468 |
| All Employees, Service-Providing | -4876.795 | 0.482 |
| All Employees, Wholesale Trade | -4887.652 | 0.495 |
| All Employees, Financial Activities | -4894.270 | 0.504 |
| All Employees, Retail Trade | -4901.839 | 0.514 |
| Industrial Production: Non-Durable Consumer Ene... | -4904.851 | 0.521 |
| Industrial Production: Non-Durable Goods Materials | -4911.850 | 0.530 |
| Canadian Dollars to U.S. Dollar Spot Exchange Rate | -4917.225 | 0.538 |
| M1 | -4922.637 | 0.546 |
| Consumer Motor Vehicle Loans Owned by Finance C... | -4928.276 | 0.554 |
| Nonrevolving consumer credit to Personal Income | -4930.961 | 0.560 |
| Moody's Seasoned Aaa Corporate Bond Yield | -4932.464 | 0.565 |
| Moody's Seasoned Baa Corporate Bond Yield | -4935.345 | 0.571 |
| All Employees, Mining, Quarrying, and Oil and G... | -4935.637 | 0.575 |
| Manufacturers' Unfilled Orders: Durable Goods | -4937.377 | 0.580 |
| Capacity Utilization: Manufacturing (SIC) | -4937.923 | 0.584 |
| Industrial Production: Manufacturing (SIC) | -4958.763 | 0.600 |
| All Employees, Nondurable Goods | -4969.329 | 0.610 |
| Industrial Production: Materials | -4972.398 | 0.615 |
| Industrial Production: Equipment: Business Equi... | -4975.063 | 0.620 |
| Average Hourly Earnings of Production and Nonsu... | -4976.308 | 0.624 |
| Average Weekly Overtime Hours of Production and... | -4978.286 | 0.629 |
| All Employees, Mining, Quarrying, and Oil and G... | -4979.425 | 0.633 |

(continues on next page)

(continued from previous page)

| | | |
|--|--------------|-------|
| Industrial Production: Consumer Goods | -4979.963 | 0.637 |
| Real Estate Loans, All Commercial Banks | -4980.075 | 0.640 |
| | rsquared_adj | |
| Initial Claims | 0.356 | |
| Real M2 Money Stock | 0.394 | |
| All Employees, Wholesale Trade | 0.420 | |
| All Employees, Service-Providing | 0.438 | |
| 5-Year Treasury Constant Maturity Minus Federal... | 0.452 | |
| Total Business: Inventories to Sales Ratio | 0.463 | |
| All Employees, Service-Providing | 0.477 | |
| All Employees, Wholesale Trade | 0.489 | |
| All Employees, Financial Activities | 0.498 | |
| All Employees, Retail Trade | 0.507 | |
| Industrial Production: Non-Durable Consumer Ene... | 0.513 | |
| Industrial Production: Non-Durable Goods Materials | 0.522 | |
| Canadian Dollars to U.S. Dollar Spot Exchange Rate | 0.529 | |
| M1 | 0.537 | |
| Consumer Motor Vehicle Loans Owned by Finance C... | 0.544 | |
| Nonrevolving consumer credit to Personal Income | 0.549 | |
| Moody's Seasoned Aaa Corporate Bond Yield | 0.554 | |
| Moody's Seasoned Baa Corporate Bond Yield | 0.559 | |
| All Employees, Mining, Quarrying, and Oil and G... | 0.563 | |
| Manufacturers' Unfilled Orders: Durable Goods | 0.567 | |
| Capacity Utilization: Manufacturing (SIC) | 0.571 | |
| Industrial Production: Manufacturing (SIC) | 0.587 | |
| All Employees, Nondurable Goods | 0.596 | |
| Industrial Production: Materials | 0.601 | |
| Industrial Production: Equipment: Business Equi... | 0.606 | |
| Average Hourly Earnings of Production and Nonsu... | 0.610 | |
| Average Weekly Overtime Hours of Production and... | 0.614 | |
| All Employees, Mining, Quarrying, and Oil and G... | 0.618 | |
| Industrial Production: Consumer Goods | 0.621 | |
| Real Estate Loans, All Commercial Banks | 0.624 | |

```
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')
```

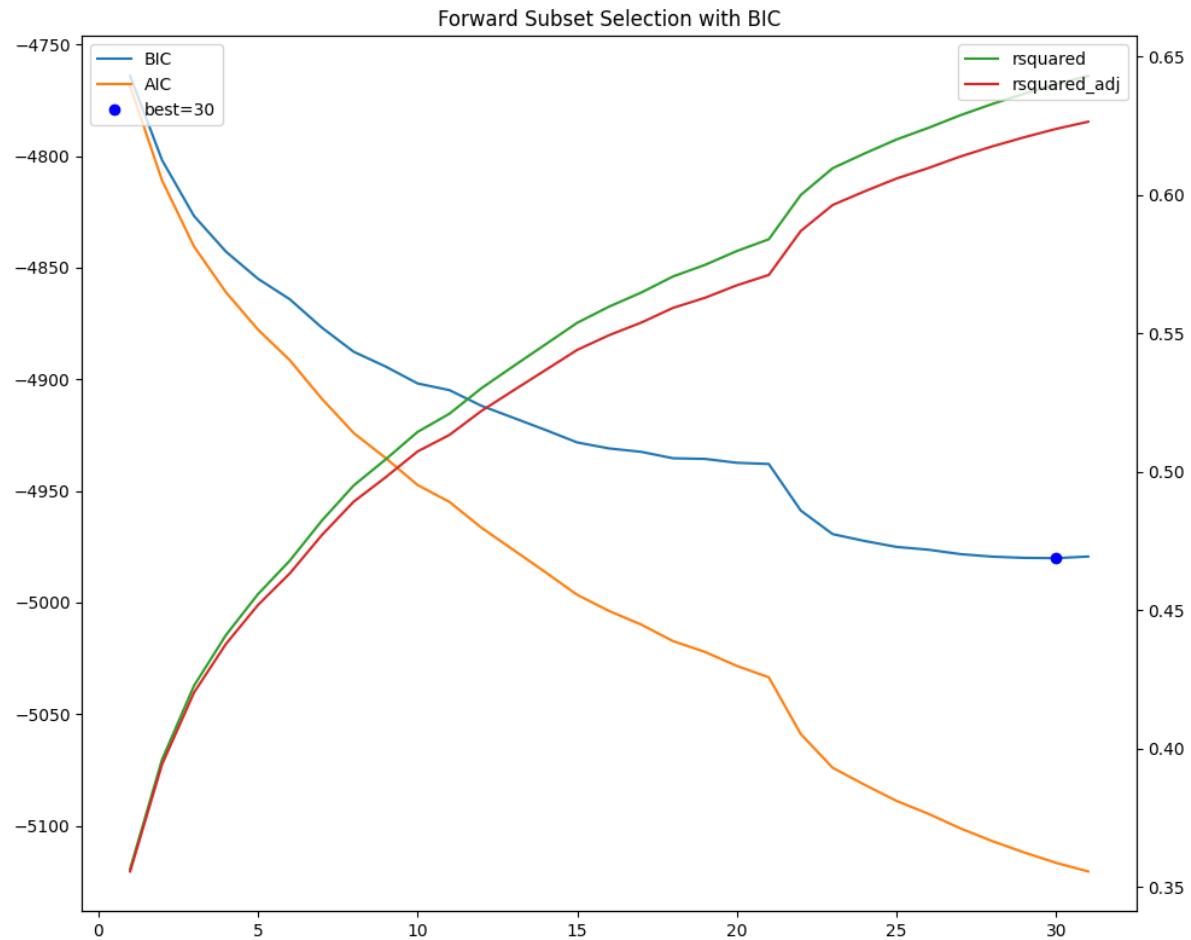
| series_id | lag | description |
|-----------|-----|---|
| CLAIMS | 1 | Initial Claims |
| M2REAL | 2 | Real M2 Money Stock |
| USWTRADE | 2 | All Employees, Wholesale Trade |
| SRVPRD | 2 | All Employees, Service-Providing |
| T5YFFM | 3 | 5-Year Treasury Constant Maturity Minus Federa... |
| ISRATIO | 2 | Total Business: Inventories to Sales Ratio |
| SRVPRD | 1 | All Employees, Service-Providing |
| USWTRADE | 1 | All Employees, Wholesale Trade |
| USFIRE | 2 | All Employees, Financial Activities |
| USTRADE | 1 | All Employees, Retail Trade |
| IPB51222S | 3 | Industrial Production: Non-Durable Consumer En... |
| IPNMAT | 3 | Industrial Production: Non-Durable Goods Mater... |

(continues on next page)

(continued from previous page)

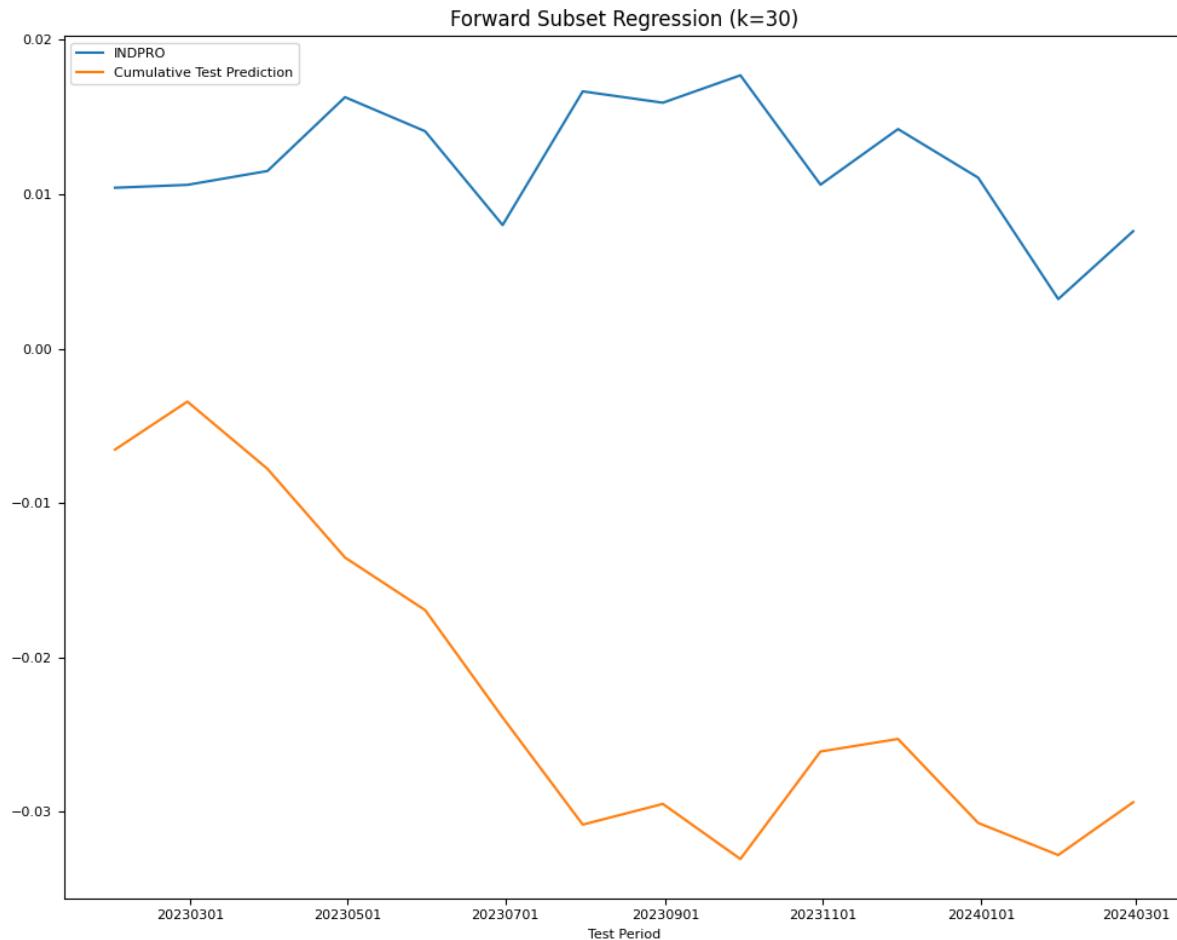
| | | |
|---------------|---|---|
| EXCAUS | 3 | Canadian Dollars to U.S. Dollar Spot Exchange ... |
| M1SL | 3 | M1 |
| DTCOLNVHFM | 3 | Consumer Motor Vehicle Loans Owned by Finance ... |
| CONSPI | 2 | Nonrevolving consumer credit to Personal Income |
| AAA | 1 | Moody's Seasoned Aaa Corporate Bond Yield |
| BAA | 1 | Moody's Seasoned Baa Corporate Bond Yield |
| CES1021000001 | 3 | All Employees, Mining, Quarrying, and Oil and ... |
| AMDMUO | 2 | Manufacturers' Unfilled Orders: Durable Goods |
| CUMFNS | 1 | Capacity Utilization: Manufacturing (SIC) |
| IPMANSICS | 1 | Industrial Production: Manufacturing (SIC) |
| NDMANEMP | 1 | All Employees, Nondurable Goods |
| IPMAT | 2 | Industrial Production: Materials |
| IPBUSEQ | 2 | Industrial Production: Equipment: Business Equ... |
| CES2000000008 | 1 | Average Hourly Earnings of Production and Nons... |
| AWOTMAN | 2 | Average Weekly Overtime Hours of Production an... |
| CES1021000001 | 1 | All Employees, Mining, Quarrying, and Oil and ... |
| IPCONGD | 2 | Industrial Production: Consumer Goods |
| REALLN | 1 | Real Estate Loans, All Commercial Banks |

```
# 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')
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',
          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=30) | 0.005965 | 0.008271 |

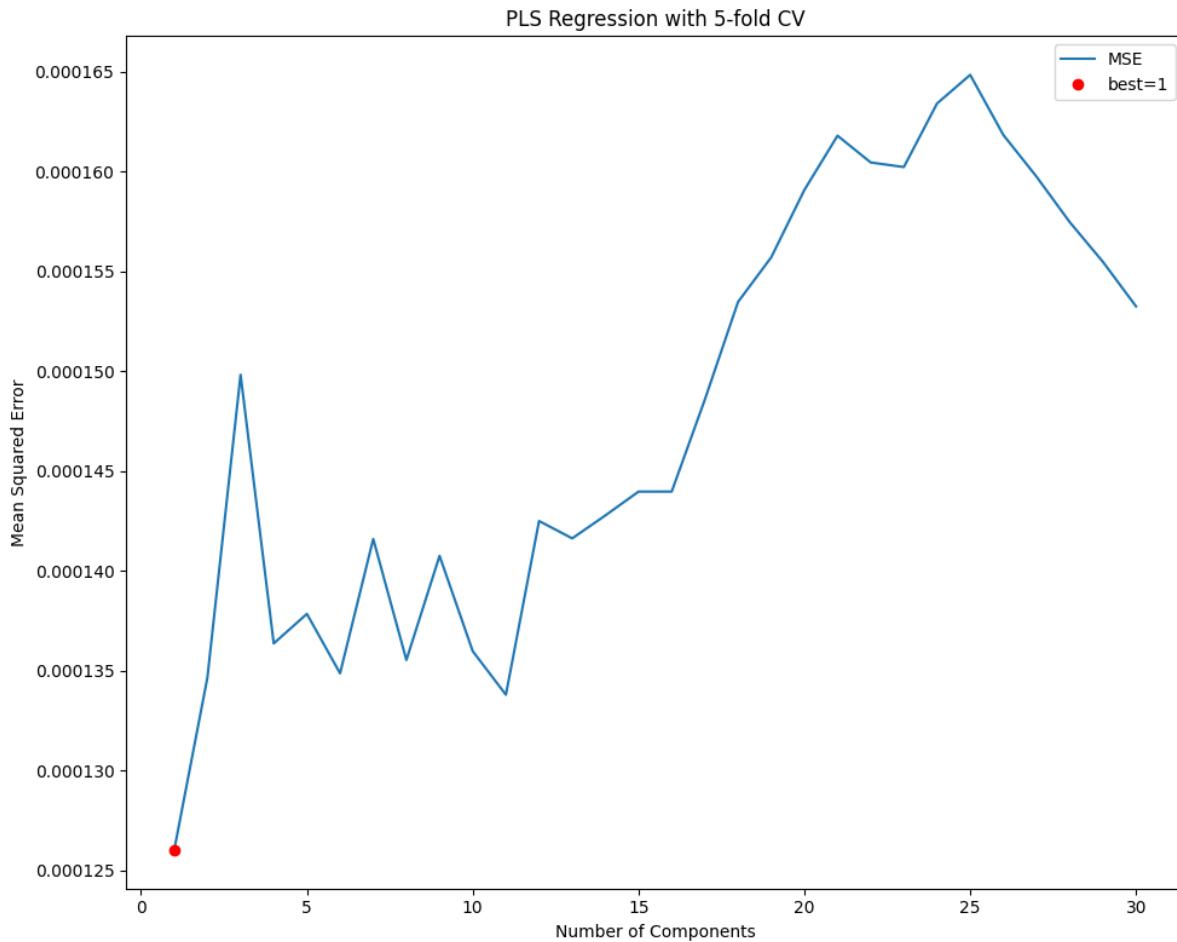
29.1.2 Partial Least Squares Regression

K-Fold cross validation

```
# 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(1, 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
```

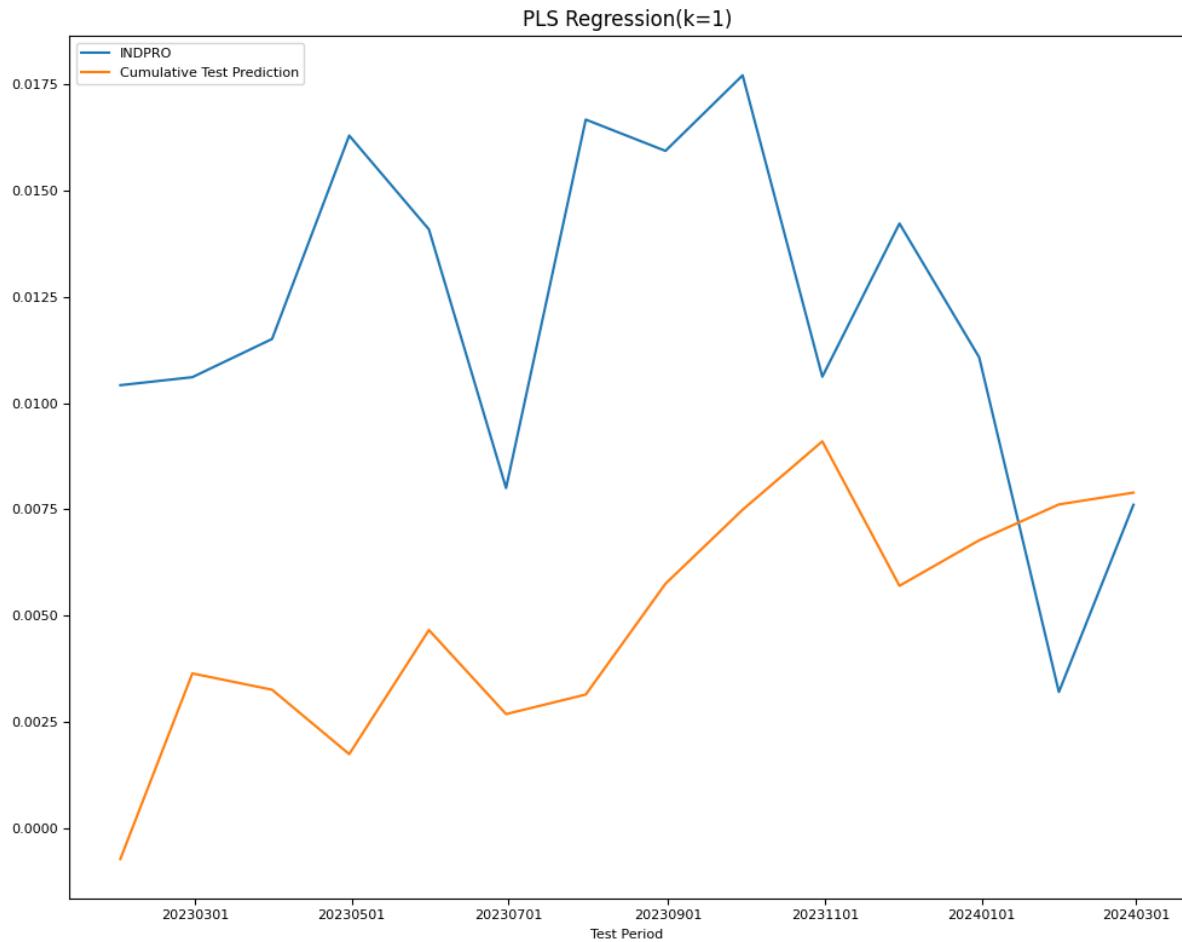
```
# show cross-validation results and best model
fig, ax = plt.subplots(figsize=(10, 8))
mse.plot(ylabel='Mean Squared Error',
          xlabel='Number of Components',
          title=f"PLS Regression with {n_splits}-fold CV",
          ax=ax)
best = mse.index[mse.argmin()]
ax.plot(best, mse.loc[best], "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'])
```

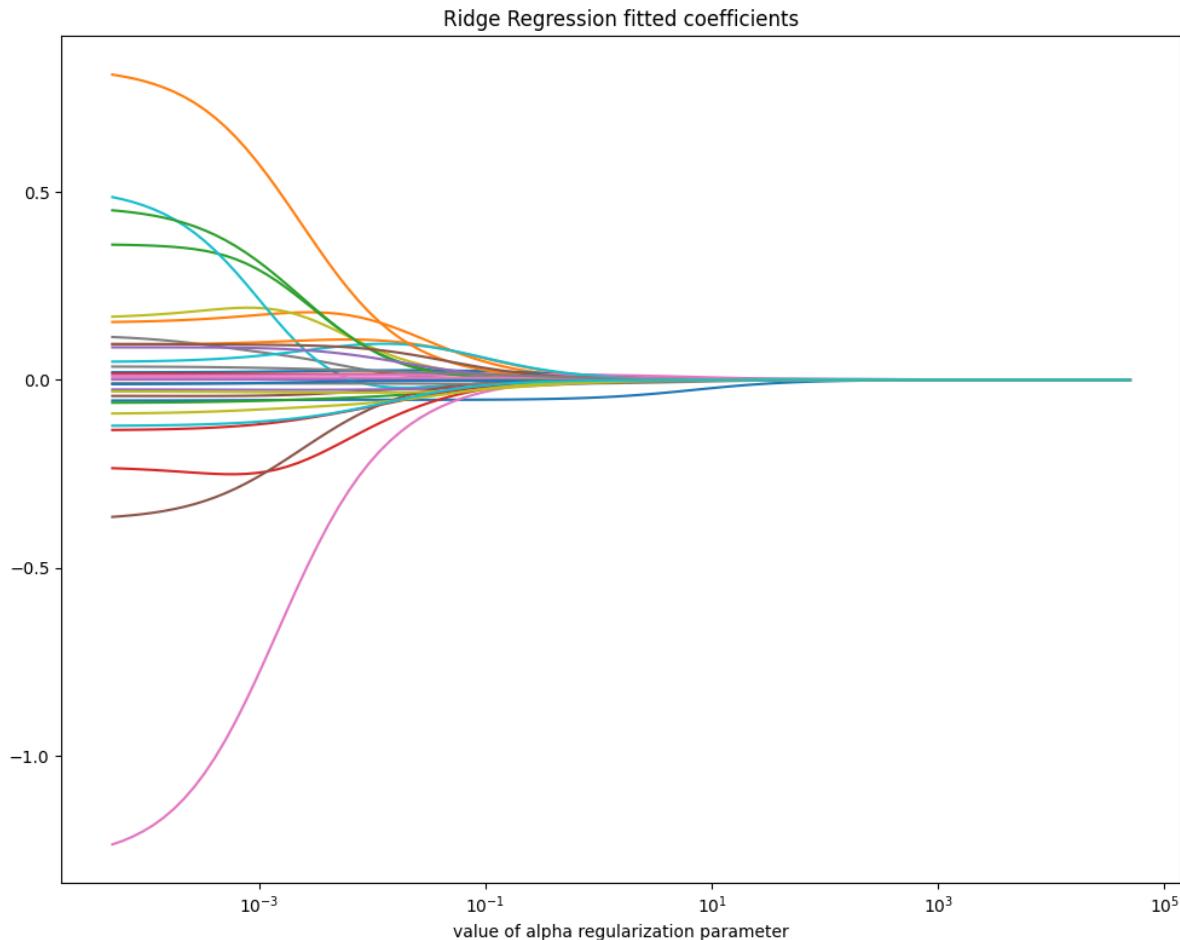
| | name | train | test |
|------|---------------------|---------|----------|
| RMSE | PLS Regression(k=1) | 0.00856 | 0.006206 |

29.1.3 Ridge Regression

Cross validation

```
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)
```

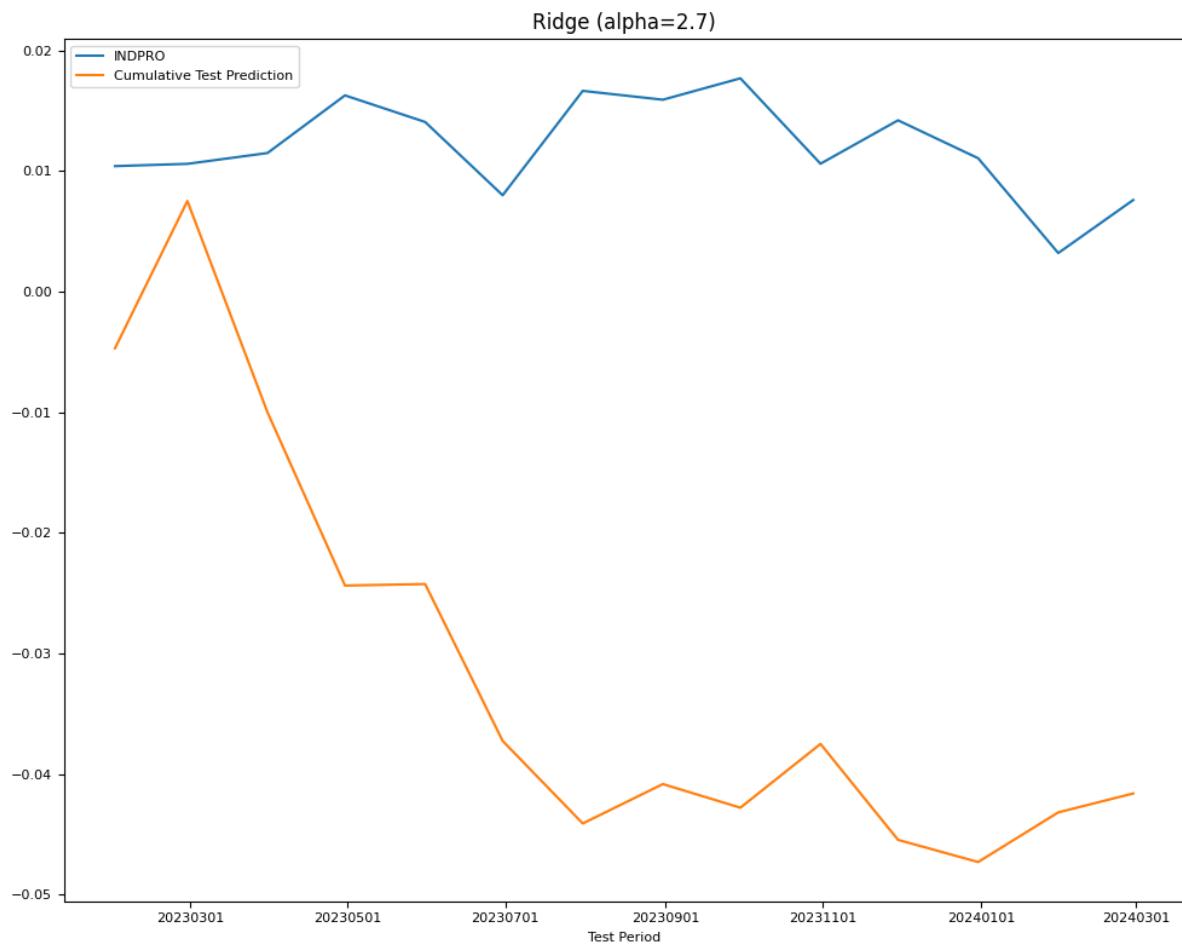
```
# Plot fitted coefficients vs regularization alpha
coefs = [Ridge(alpha, fit_intercept=False) \
          .fit(X_subset, Y_train).coef_ for alpha in alphas]
fig, ax = plt.subplots(figsize=(10, 8))
ax.plot(alphas, coefs)
ax.set_xscale('log')
ax.set_xlabel('value of alpha regularization parameter')
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'])
```

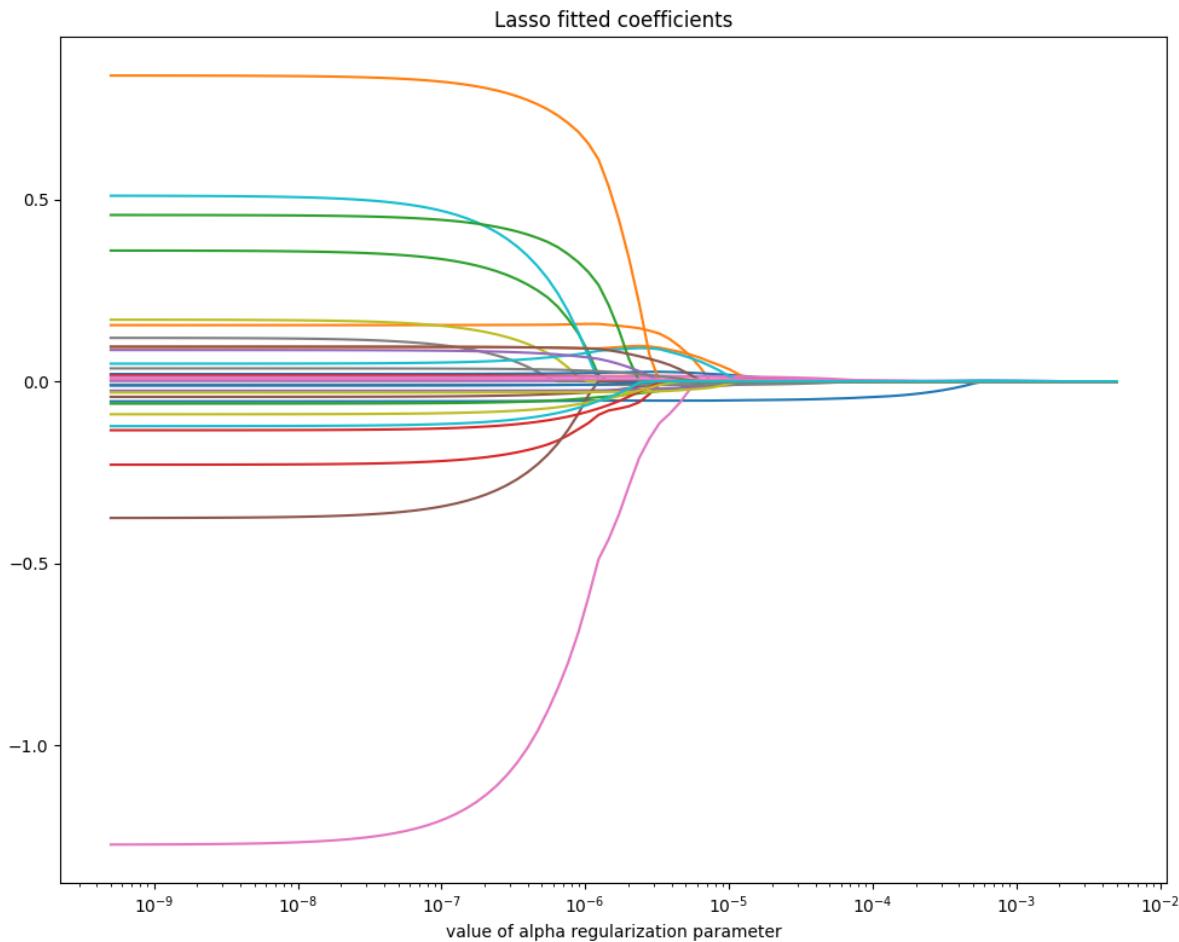
| | name | train | test |
|------|-------------------|----------|----------|
| RMSE | Ridge (alpha=2.7) | 0.004358 | 0.011474 |

29.1.4 Lasso Regression

Cross Validation

```
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,
```

(continues on next page)

(continued from previous page)

```

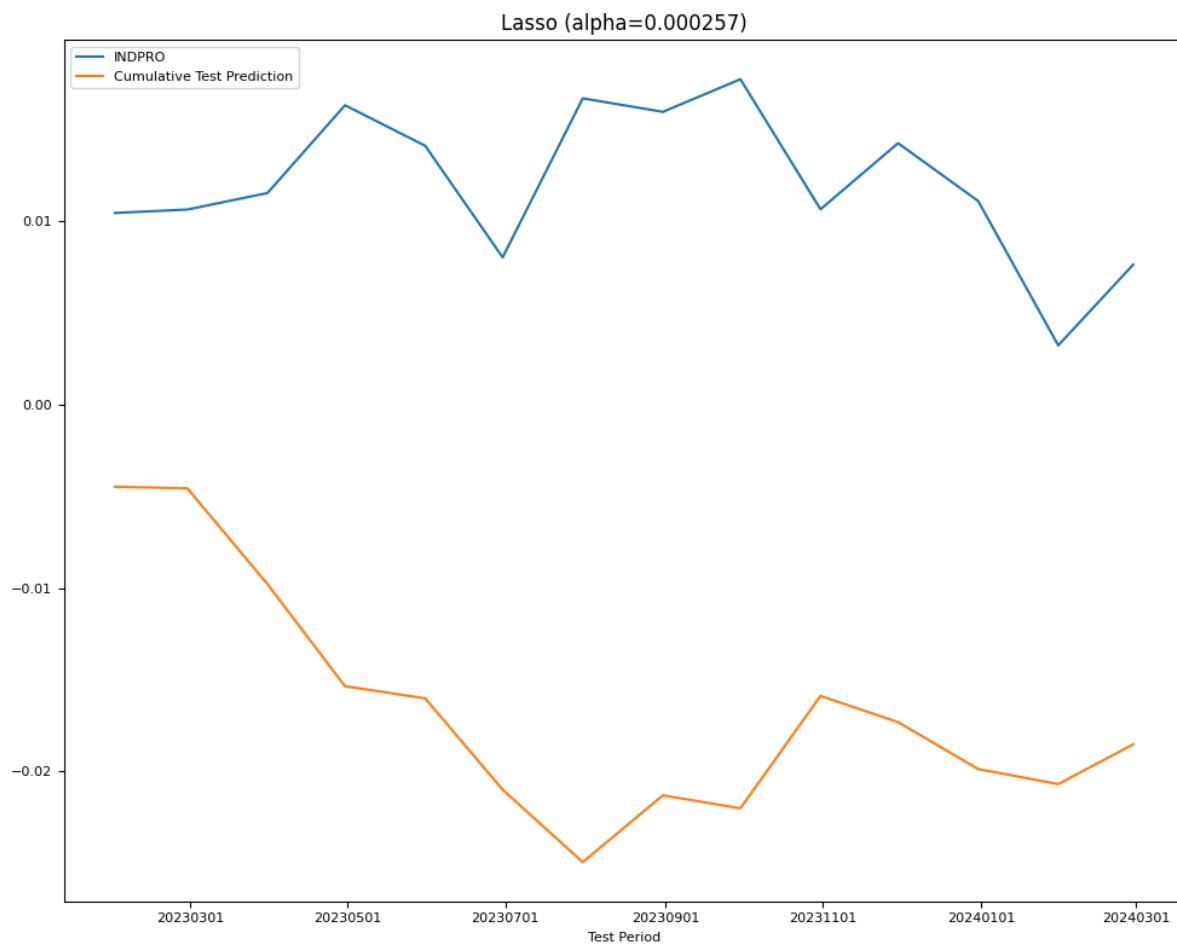
cv=5,
n_jobs=5,
verbose=VERBOSE,
max_iter=20000).fit(X_train, Y_train)
name = f'Lasso (alpha={model.alpha_:.3g})'
Y_pred = Series(index=Y_test.index,
                 data=model.predict(X_test).reshape((-1,)))
test[name] = mean_squared_error(Y_test, Y_pred)
train[name] = mean_squared_error(Y_train, model.predict(X_train))
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])},

```

(continues on next page)

(continued from previous page)

```
'test': np.sqrt(test[name])}, index=['RMSE'])
```

| | name | train | test |
|------|------------------------|----------|----------|
| RMSE | Lasso (alpha=0.000257) | 0.006304 | 0.007601 |

```
# 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 | -0.005390 |
| SRVPRD | 1 | All Employees, Service-Providing | -0.001311 |
| M2REAL | 2 | Real M2 Money Stock | 0.000903 |
| IPNMAT | 3 | Industrial Production: Non-Durable Goods Mater... | 0.000708 |
| VIXCLS | 1 | CBOE Volatility Index: VIX | -0.000705 |
| ... | ... | ... | ... |
| IPFUELS | 3 | Industrial Production: Non-Durable Consumer En... | -0.000033 |
| IPNCONGD | 1 | Industrial Production: Non-Durable Consumer Goods | -0.000023 |
| WPSFD49502 | 2 | Producer Price Index by Commodity: Final Deman... | 0.000017 |
| M2SL | 3 | M2 | 0.000015 |
| HWIURATIO | 2 | Ratio of Help Wanted/No. Unemployed | -0.000010 |

[78 rows x 3 columns]

29.1.5 Decision Tree

The tree-based methods involve stratifying or segmenting the predictor space into a number of simple regions. In order to make a prediction for a given observation, we typically use the mean or the mode of the training observations in the region to which it belongs. The set of splitting rules used to segment the predictor space can be summarized in a tree.

- A **Decision tree** consists of a series of splitting rules, that segments observations into regions of predictor space. Decision trees are typically drawn upside down, in the sense that the leaves are at the bottom of the tree.
- **Terminal nodes** or **leaves** are partitions of the predictor space that observations are segmented into.
- **Internal nodes** are points along the tree where the predictor space is split.
- **Branches** are those segments of the trees that connect the nodes.
- A **Stump** is a decision tree with only one internal node.

Recursive binary splitting is a top-down, greedy approach to construct decision trees. It begins at the top of the tree (at which point all observations belong to a single region) and then successively splits the predictor space; each split is indicated via two new branches further down on the tree. At each step of the tree-building process, the best split is made

at that particular step, rather than looking ahead and picking a split that will lead to a better tree in some future step. We first select that predictor and the cutpoint s such that splitting the predictor space into the regions leads to the greatest possible reduction in a cost function (RSS). Next, we repeat the process, looking for the best predictor and best cutpoint in order to split the data in one of the two previously identified regions further so as to minimize the cost within each of the resulting regions. The process continues until a stopping criterion is reached.

A procedure that grows trees until the leaves are pure tends to overfit. The complexity of the tree lies in the number of nodes. We artificially limit the maximum size of each tree, as measured by the number of nodes it's allowed to have, indicated on the horizontal axis of the fitting graph. For each tree size we create a new tree from scratch, using the training data. We measure two values: its accuracy on the training set and its accuracy on the holdout (test) set. As the trees are allowed to get larger, the training-set accuracy continues to increase, but the holdout accuracy eventually declines. The complexity at which holdout accuracy declines as the tree grows past its "sweet spot".

The **cost complexity pruning** strategy reduces the complexity of a tree (that is less likely to overfit the data) is to grow a very large tree, and then prune it back in order to obtain a subtree. However, estimating the test error from cross-validation error for every possible subtree is too cumbersome, since there is an extremely large number of possible subtrees. Cost complexity pruning (also known as weakest link pruning) considers a sequence of trees indexed by a nonnegative tuning parameter α , rather than considering every possible subtree. For each value of α there corresponds a subtree T such that $\sum_{m=1}^{|T|} \sum_{x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha|T|$ is as small as possible, where $|T|$ indicates the number of terminal nodes of the tree T , R_m is the subset of predictor space corresponding to the m th terminal node, and \hat{y}_{R_m} is the predicted response associated with R_m . The tuning parameter α controls a trade-off between the subtree's complexity (number of terminal nodes) and its fit to the training data.

Alternate criteria of impurity for making the binary splits in a classification tree:

- **Classification error rate** (i.e. the fraction of the training observations in that region that do not belong to the most common class) is not sufficiently sensitive for tree-growing. $\hat{p}_{m,c} \frac{n_{m,c}}{n_m}$, where n_m is the number of observations in node m , and $n_{m,c}$ is the number of observations in node m that are in category c .
- **Gini index** is a measure of total variance across the K classes, is defined by $G = \sum_{k=1}^K \hat{p}_{mk}(1 - \hat{p}_{mk})$ where \hat{p}_{mk} represent the proportion of training observations in the m th region that are from the k th class. It is close to zero or one. For this reason the Gini index is referred as a measure of node purity – a small value indicates that a node contains predominantly observations from a single class.
- **Entropy**, given by $D = \sum_{k=1}^K \hat{p}_{mk} \log \hat{p}_{mk}$. The entropy will take on a value near zero if the \hat{p}_{mk} 's are all near zero or one.

For classification,

$$\text{Deviance} = -2 \sum_{m=1}^g \sum_{c=1}^w n_{m,c} \ln \hat{p}_{m,c}$$

$$\text{Residual mean deviance} = \frac{\text{deviance}}{n-1}$$

If there is a highly non-linear and complex relationship between the features and the response, then decision trees (which assume a model of the form $f(x) = \sum_{m=1}^M c_m \cdot 1_{(X \in R_m)}$) may outperform linear regression models (which assume the form $f(x) = \beta_0 \sum_{j=1}^p X_j \beta_j$)

Advantages

- Trees are very easy to explain to people. In fact, they are even easier to explain than linear regression!
- Decision trees may more closely mirror human decision-making than do the regression and classification approaches.
- Trees can be displayed graphically, and are easily interpreted even by a non-expert (especially if the trees are small).

- Trees can easily handle qualitative predictors without the need to create dummy variables

Disadvantages:

- Trees generally do not have the same level of predictive accuracy as some of the other regression and classification
- Trees can be very non-robust. In other words, a small change in the data can cause a large change in the final estimated tree.
- Trees overfits categorical variables

By aggregating many decision trees, using ensemble methods like bagging, random forests, and boosting, the predictive performance of trees can be substantially improved.

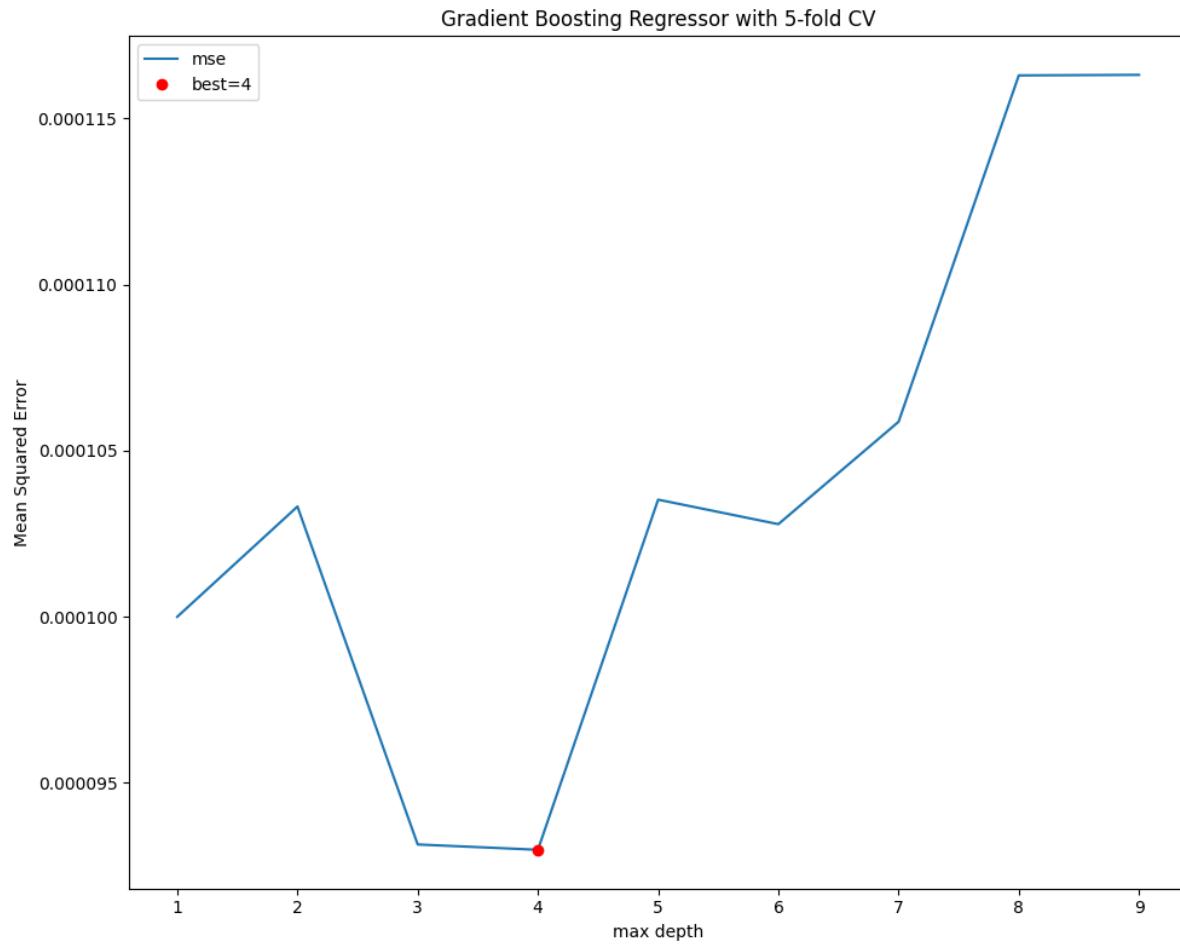
29.1.6 Gradient boosting

KFold cross validation

```
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
```

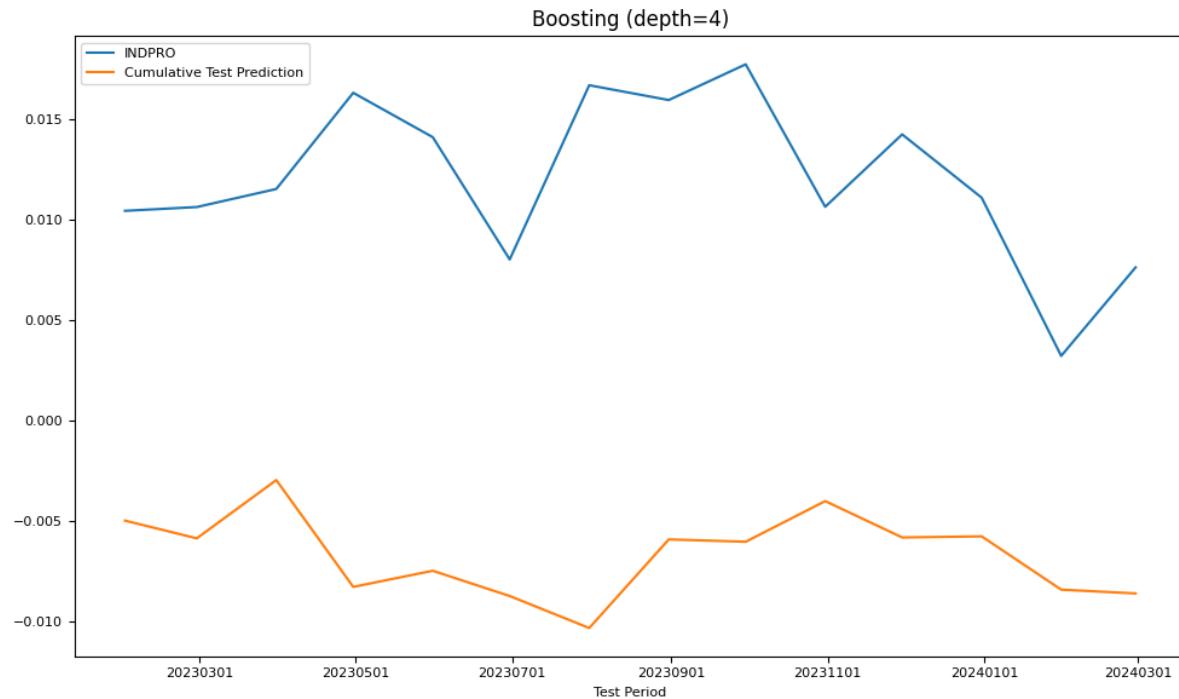
```
fig, ax = plt.subplots(figsize=(10, 8))
mse.plot(ax=ax, ylabel='Mean Squared Error', xlabel='max depth',
         title=f"Gradient Boosting Regressor with {n_splits}-fold CV")
best = mse.index[mse.argmin()]
ax.plot(best, mse.loc[best], "or")
ax.legend(['mse', f"best={best}"])
plt.tight_layout()
```



```
# 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'])
```

| RMSE | name | train | test |
|--------------------|----------|---------|------|
| Boosting (depth=4) | 0.001661 | 0.00698 | |

```
# 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 | 0.179785 | CLAIMS | 1 |
| 2 | 0.088659 | M2SL | 1 |
| 3 | 0.040664 | BUSLOANS | 1 |
| 4 | 0.030011 | HWIURATIO | 1 |
| 5 | 0.027526 | UEMPLT5 | 3 |
| 6 | 0.025408 | MANEMP | 1 |
| 7 | 0.018103 | IPNMAT | 1 |
| 8 | 0.017666 | PERMITW | 1 |

(continues on next page)

(continued from previous page)

```

9      0.016720    EXCAUS     1
10     0.016259    USGOOD     1

                                              description
1                               Initial Claims
2                               M2
3  Commercial and Industrial Loans, All Commercia...
4          Ratio of Help Wanted/No. Unemployed
5          Number Unemployed for Less Than 5 Weeks
6          All Employees, Manufacturing
7  Industrial Production: Non-Durable Goods Mater...
8  New Privately-Owned Housing Units Authorized i...
9  Canadian Dollars to U.S. Dollar Spot Exchange ...
10          All Employees, Goods-Producing

```

29.1.7 Random Forest

KFold cross validation

```

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)))

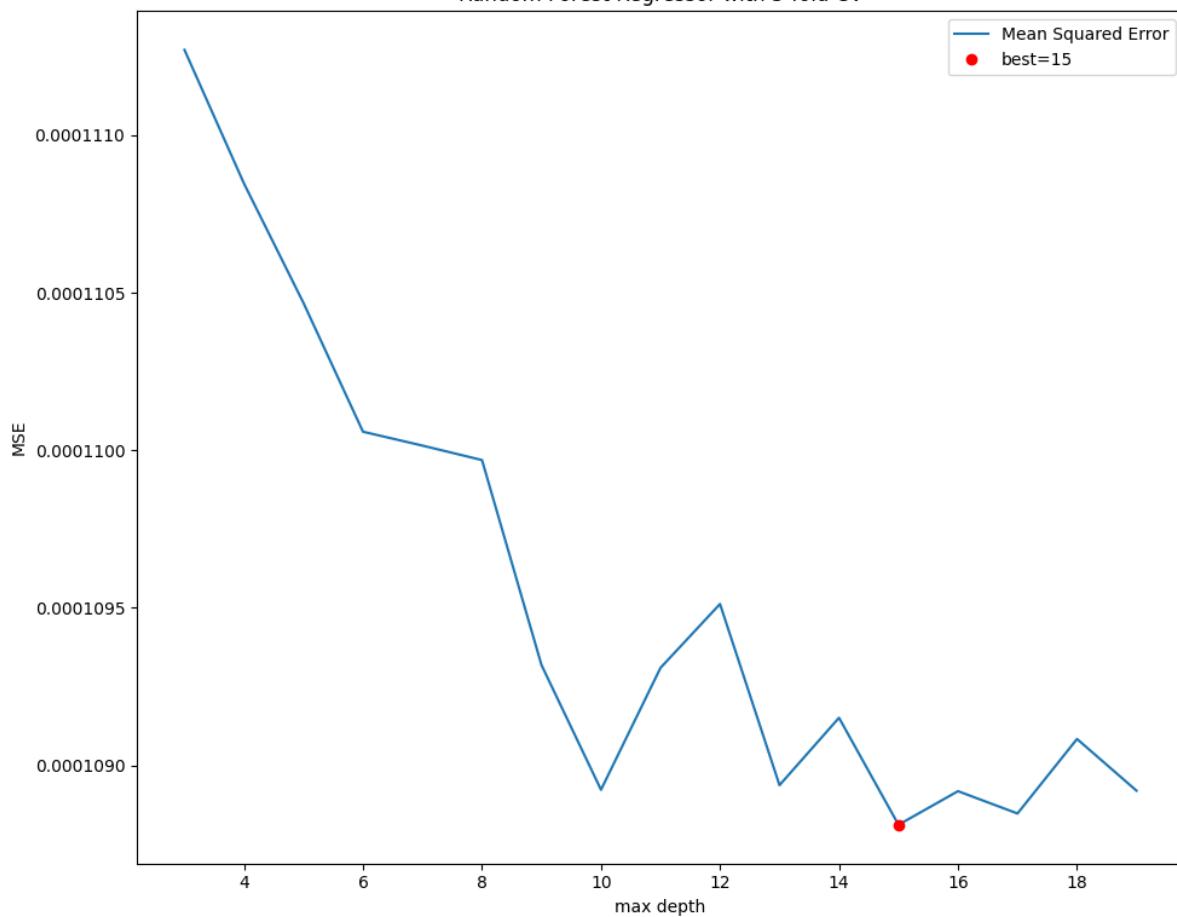
```

```

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()]
ax.plot(best, mse.loc[best], "or")
ax.legend(['Mean Squared Error', f"best={best}"])
plt.tight_layout()

```

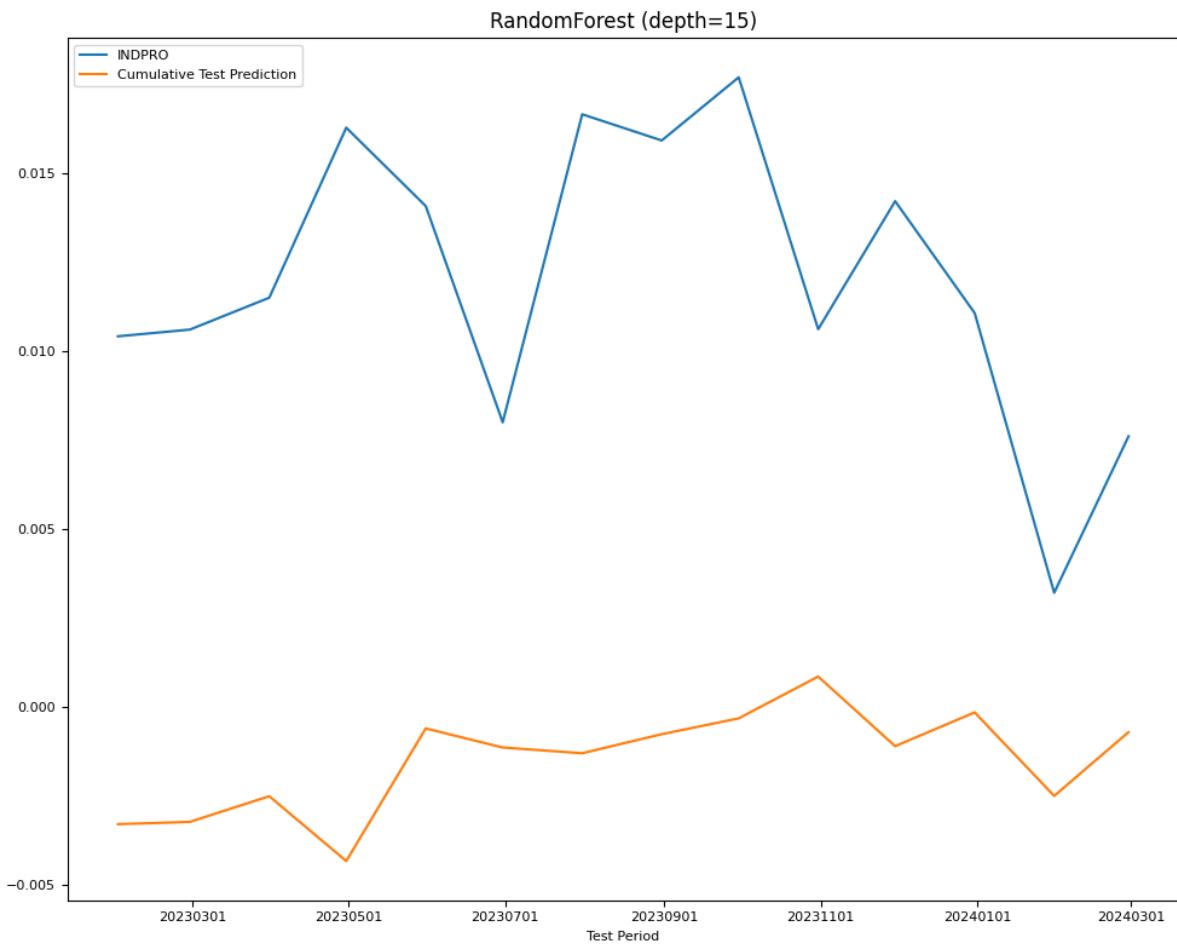
Random Forest Regressor with 5-fold CV



```
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)
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) | 0.003833 | 0.006165 |

```
# 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 0.051142 CLAIMS 1
2 0.047626 BUSLOANS 1
3 0.044440 M2SL 1
4 0.027707 USGOOD 1
5 0.016242 UEMPLT5 1
6 0.014992 UEMPLT5 3
7 0.014374 IPCONGD 1
8 0.014345 M1SL 1
9 0.012969 OILPRICE 1
10 0.012449 MANEMP 1

description
1 Initial Claims
2 Commercial and Industrial Loans, All Commercia...
3 M2
4 All Employees, Goods-Producing
5 Number Unemployed for Less Than 5 Weeks
6 Number Unemployed for Less Than 5 Weeks
7 Industrial Production: Consumer Goods
8 M1
9 Crude Oil, spliced WTI and Cushing
10 All Employees, Manufacturing

```

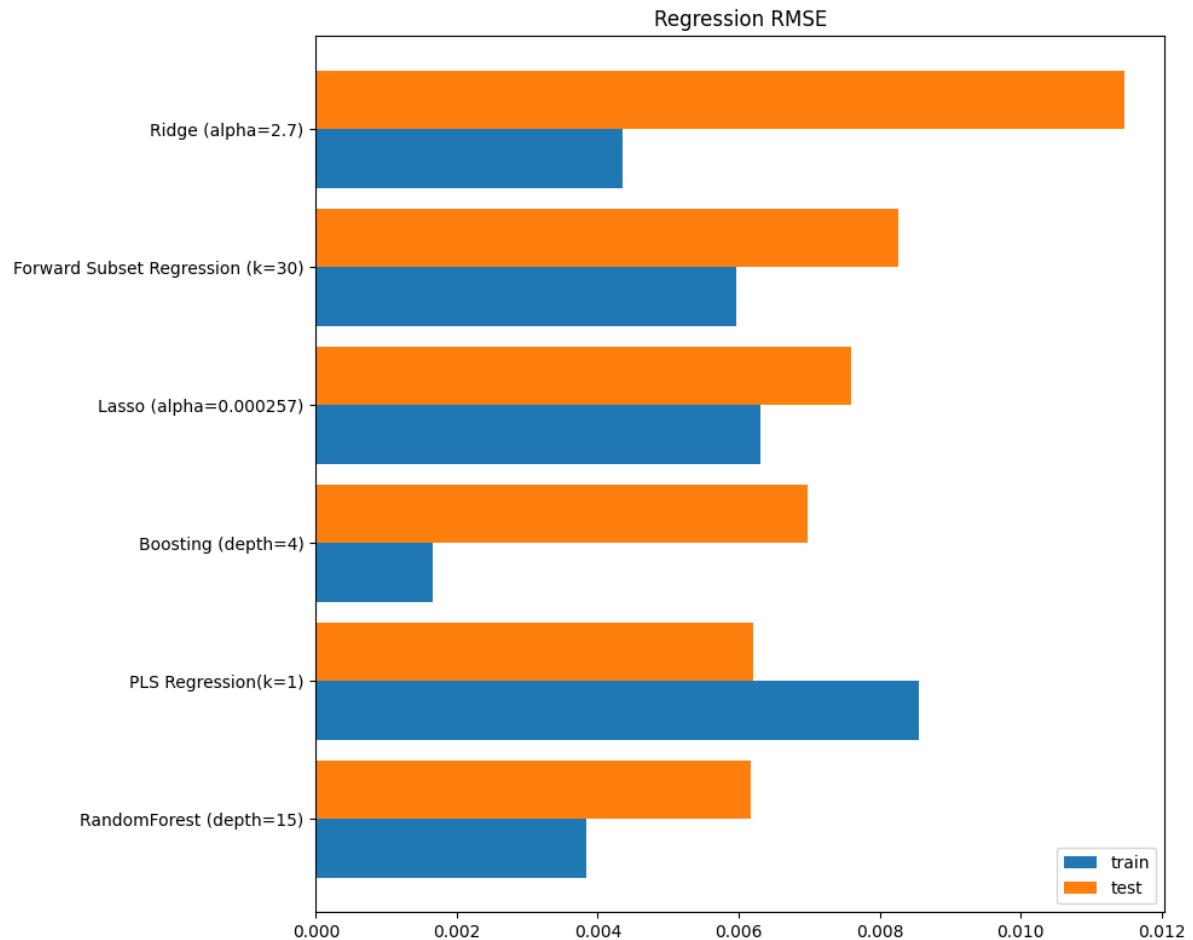
29.2 Evaluation

RMSE

```

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()

```



DEEP LEARNING

When you come to a fork in the road, take it - Yogi Berra

Concepts:

- Word Embeddings
- Feedforward Neural Networks

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, 2023, “CS388 Natural Language Processing course materials”, retrieved from <https://www.cs.utexas.edu/~gdurrett/courses/online-course/materials.html>
- Philipp Krahenbuhl, 2020, “CS395T Deep Learning course materials”, retrieved from http://www.philkr.net/dl_class/material

```
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
import nltk
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 MongoDB, SQL, RedisDB
```

(continues on next page)

(continued from previous page)

```
from finds.unstructured import Unstructured, Edgar, Vocab
from finds.structured import BusDay, CRSP, PSTAT
from finds.readers import Sectoring
from finds.utils import Store
from secret import credentials, paths
# %matplotlib qt
# jupyter-notebook --NotebookApp.iopub_data_rate_limit=1.0e12
VERBOSE = 0
outdir = paths['scratch']
store = Store(outdir, ext='pkl')
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print('device is', device)
```

device is cuda

```
# dimension of embeddings vector
embeddings_dim = 300
# max size of vocab
vocab_len = 15000
```

```
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)
```

Last FamaFrench Date 2024-03-28 00:00:00

30.1 Text classification

```
# Retrieve universe of stocks
univ = crsp.get_universe(bd.endmo(20221231))
```

```
# 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'])
```

(continues on next page)

(continued from previous page)

```
sectors = Sectoring(sql, scheme='codes10', fillna='')      # supplement from crosswalk
univ['sector'] = sectors[industry]
```

```
# retrieve 2023 bus10K's
item, form = 'bus10K', '10-K'
rows = DataFrame(ed.open(form=form, item=item))
found = rows[rows['date'].between(20230101, 20231231)]\n    .drop_duplicates(subset=['permno'], keep='last')\n    .set_index('permno')
```

30.1.1 Tagging and Lemmatizing

```
# !nltk.download('averaged_perceptron_tagger')
lemmatizer = WordNetLemmatizer()
bus = {}
for permno in tqdm(found.index):
    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())
```

```
# Create textual dataset and encode labels
words = Counter()
for nouns in bus.values():
    words.update(list(nouns))
vocab = Vocab(words.keys())
print('vocab len:', len(vocab))
```

```
vocab len: 79581
```

```
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()
```

30.2 Word Embeddings

- Continuous BOW, skip-gram, Word2Vec, GloVe matrix factorization

30.2.1 GloVe

```
# Retrieve GloVe embeddings
glove_file = f"glove.6B.{embeddings_dim}d.txt"
source = "https://nlp.stanford.edu/data/glove.6B.zip"
response = requests.get(source)
source = io.BytesIO(response.content)
source = paths['scratch'] / 'glove.6B.zip'
with zipfile.ZipFile(source).open(glove_file) as f:
    filename = paths['scratch'] / glove_file
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)

# Relativize embeddings to words in vocab
vocab.set_embeddings(embeddings)
print(vocab.embeddings.shape)
vocab.dump(outdir / f"dan{embeddings_dim}_{vocab_len}.pkl")
```

```
(400000, 300)
(79581, 300)
```

```
# load previously saved Vocab
vocab.load(outdir / f"dan{embeddings_dim}_{vocab_len}.pkl")
```

30.2.2 Word Vector Arithmetic

It has been noted that these examples are only suggestive of what the vectors may capture, and such arithmetic results are generally not as sharp; they may also show the potential for biases, such as gender roles, implicit in the training samples.

```
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()
```

(continues on next page)

(continued from previous page)

```
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']
```

30.3 FeedForward Neural Network

- nonlinearity
- hidden layers and size
- initialization
- optimization: adam
- dropout

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.

30.3.1 Deep Averaging Networks

- Frozen and fine-tuning word embedding vectors

```
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)
```

(continues on next page)

(continued from previous page)

```

        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] = 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

```

Train Models

```

# 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()

```

3559 3559 2847 712 10

| | count |
|-------|-------|
| Hlth | 881 |
| Other | 762 |
| HiTec | 706 |
| Manuf | 344 |
| Shops | 321 |
| Durbl | 164 |
| NoDur | 145 |
| Enrgy | 94 |

(continues on next page)

(continued from previous page)

| | |
|-------|----|
| Utils | 92 |
| Telcm | 50 |

```
# Model and training parameters
layers = 1
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           (23, 874, 300)
├─Sequential: 1-2
│  ├─Dropout: 2-1
│  ├─Linear: 2-2               9, 632
│  ├─ReLU: 2-3
│  ├─Dropout: 2-4
│  ├─Linear: 2-5               330
├─LogSoftmax: 1-3
=====
Total params: 23,884,262
Trainable params: 9,962
Non-trainable params: 23,874,300
=====
```

```
batch_sz = 16
lr = 0.001
num_epochs = 50
optimizer = torch.optim.Adam(model.parameters(), lr=lr)
loss_function = nn.NLLLoss()
```

```
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), (False, False), (False, True)]):
    model.set_freeze(freeze=freeze)
    model.set_dropout(dropout=dropout)
    accuracy.append(dict())

    # Loop over epochs
    for epoch in tqdm(range(num_epochs)):
        tic = time.time()
```

(continues on next page)

(continued from previous page)

```

# 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
model.eval()
model.save(outdir / f"dan{embeddings_dim}.pt")

if VERBOSE:
    print(f"Loss {epoch}/{num_epochs} ({freeze, dropout}):" +
          f"\n{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 [03:48<00:00, 4.56s/it]
100% | ██████████ | 50/50 [03:56<00:00, 4.72s/it]
100% | ██████████ | 50/50 [03:56<00:00, 4.73s/it]

Confusion matrix

```

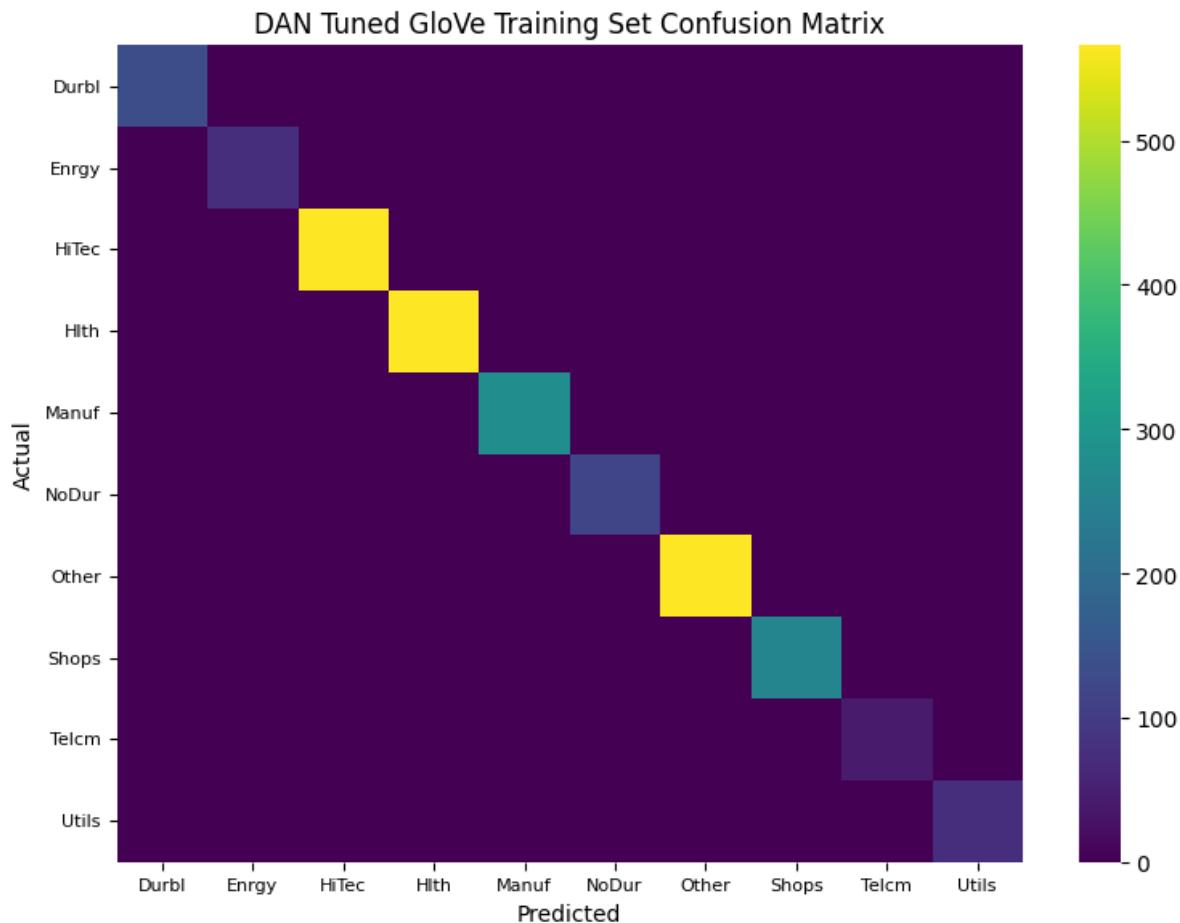
classes = class_encoder.classes_
cf_train = DataFrame(confusion_matrix(train_gold, train_pred),
                     index=pd.MultiIndex.from_product([['Actual'], classes]),
                     columns=pd.MultiIndex.from_product([['Predicted'], classes]))
cf_test = DataFrame(confusion_matrix(test_gold, test_pred),
                     index=pd.MultiIndex.from_product([['Actual'], classes]),
                     columns=pd.MultiIndex.from_product([['Predicted'], classes]))

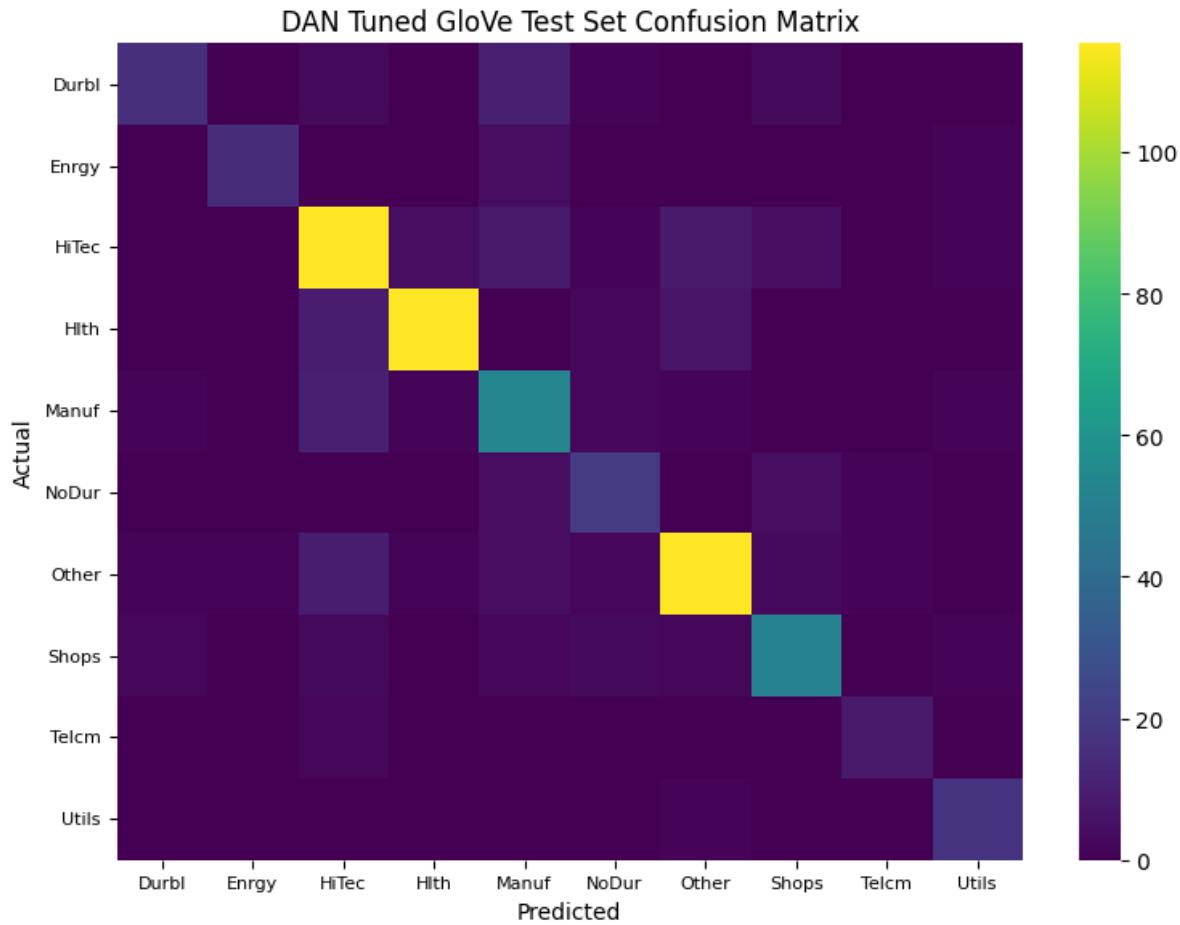
```

```

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,
                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()

```





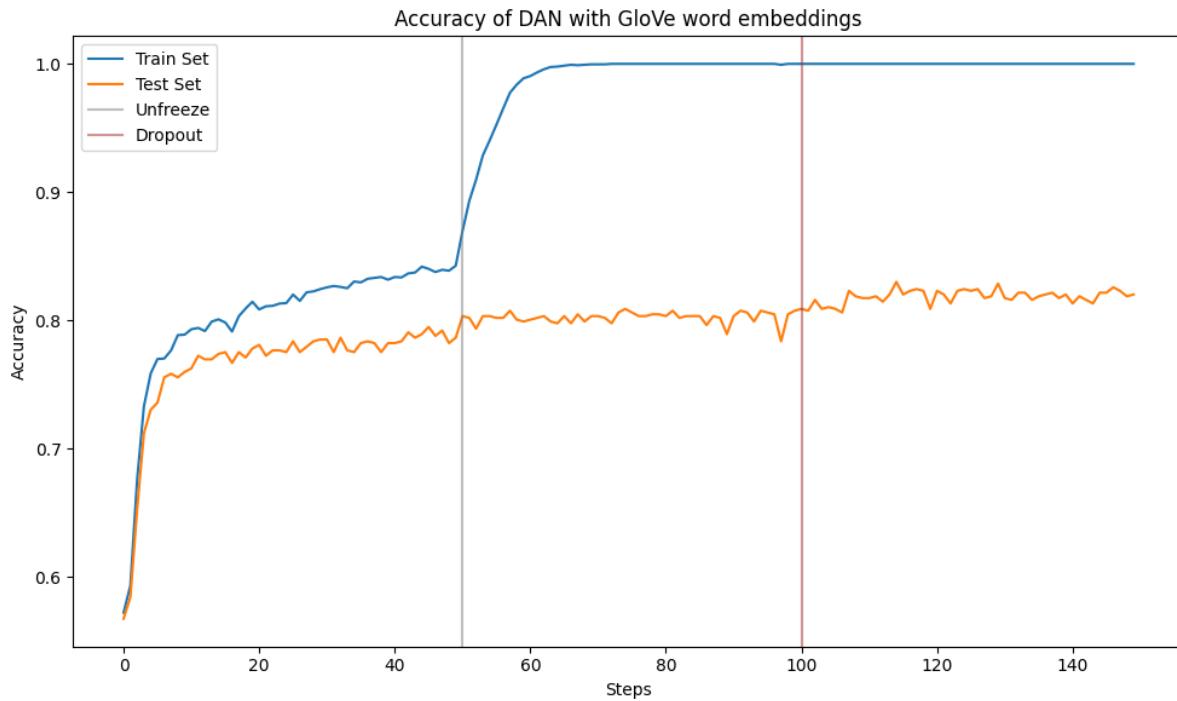
Accuracy

```

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', 'Unfreeze', 'Dropout'], loc='upper left')
plt.tight_layout()

```



```
# 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]]],
            'unfrozen': [train_accuracy.iloc[p[1]], test_accuracy.iloc[p[1]]],
            'dropout': [train_accuracy.iloc[p[2]], test_accuracy.iloc[p[2]]]},
            index=['train', 'test'])
```

Accuracy

| | frozen | unfrozen | dropout |
|-------|----------|----------|----------|
| train | 0.842641 | 1.000000 | 1.000000 |
| test | 0.786517 | 0.807584 | 0.820225 |

RECURRENT NEURAL NETWORKS

The only thing we know about the future is that it will be different - Peter Drucker

Concepts:

- Recurrent Neural Network
- Linear Dynamic Factor Models
- (?) Dynamic Factor Models: latent autocorrelated factors estimated by EM or Kalman Filter algorithm
 - model selection on factor AR lags
- (?) RNN: hidden states feedback

https://www.statsmodels.org/dev/examples/notebooks/generated/statespace_dfm_coincident.html

```
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")
```

```
# Max number of hidden states (RNN) or factors (Dynamic Model)
K = 2
```

```
# number of out-of-sample forecasts to predict
nforecast = 3
```

```
# train-test split date
split_date = '2021-12-01'

# 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 | -1.514294 | -1.123374 | 0.528419 | 0.262158 | |
| 1967-03-31 | -0.803761 | -1.123374 | 0.117546 | -0.270473 | |
| 1967-04-30 | -1.516345 | -0.064928 | 0.523552 | 0.258920 | |
| 1967-05-31 | -0.803761 | -0.068382 | 0.115127 | -0.006978 | |
| 1967-06-30 | 1.327849 | -1.123374 | 0.518742 | -0.270473 | |
| ... | ... | ... | ... | ... | ... |
| 2023-12-31 | -0.305673 | -0.150090 | -0.300364 | -0.226813 | |
| 2024-01-31 | 0.136279 | 0.874513 | -1.728819 | -0.823810 | |
| 2024-02-29 | -0.763532 | 0.154003 | 0.841913 | 0.932892 | |
| 2024-03-31 | -0.555699 | 0.155815 | 1.046454 | 0.409135 | |
| 2024-04-30 | -0.757514 | -0.408180 | 2.163913 | 0.318836 | |
| | Medical Care | Other Goods and Services | | | |
| date | | | | | |
| 1967-02-28 | -0.260945 | | -1.020565 | | |
| 1967-03-31 | -0.265522 | | -0.290026 | | |
| 1967-04-30 | 0.977131 | | -1.020565 | | |
| 1967-05-31 | -0.279054 | | -0.292134 | | |
| 1967-06-30 | 0.950357 | | -0.294231 | | |

(continues on next page)

(continued from previous page)

```

...
2023-12-31      -0.199913      -0.950240
2024-01-31      0.045640       0.340696
2024-02-29      -1.610667      -1.786460
2024-03-31      0.152838       -0.080117
2024-04-30      0.016674       0.048790

[687 rows x 6 columns]

```

31.1 Recurrent Neural Network

31.1.1 Elman Network

Applies a multi-layer Elman RNN with tanh (or ReLU) non-linearity to an input sequence.

For each 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

```

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)

```

```

# 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

```

31.1.2 Learning rate

learning rate schedulimg

```

# 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,
                                                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)
            output, hidden = model(x, hidden)
            y_pred.append(hidden.cpu().numpy().flatten())

```

(continues on next page)

(continued from previous page)

```

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)
)

```

```

/home/terence/env3.11/lib/python3.11/site-packages/tqdm/auto.py:21: TqdmWarning:-
  ↵IPProgress not found. Please update jupyter and ipywidgets. See https://
  ↵ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
100%|██████████| 300/300 [01:32<00:00, 3.23it/s]

```

```

train MSE (hidden=1): 0.8109380710349426
test MSE (hidden=1): 0.8741724611011269
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 [01:30<00:00, 3.31it/s]
```

```

train MSE (hidden=2): 0.753971276651208
test MSE (hidden=2): 0.7898786960734855

```

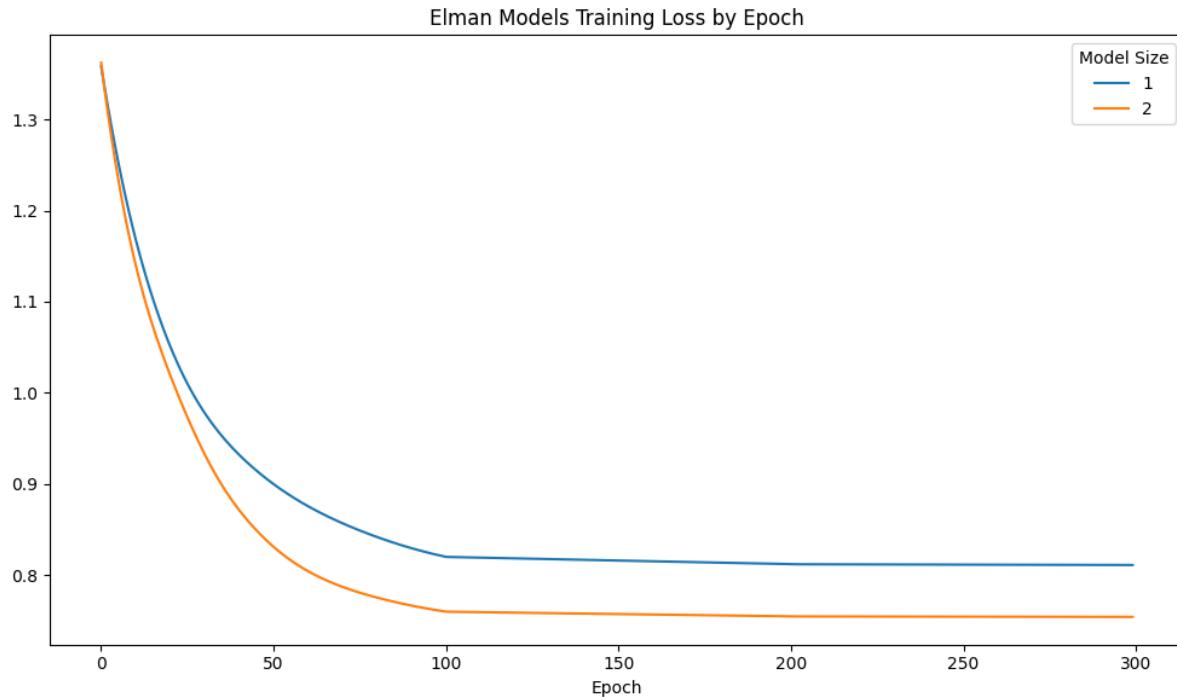
```

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()

```

(continues on next page)

(continued from previous page)



```
# 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

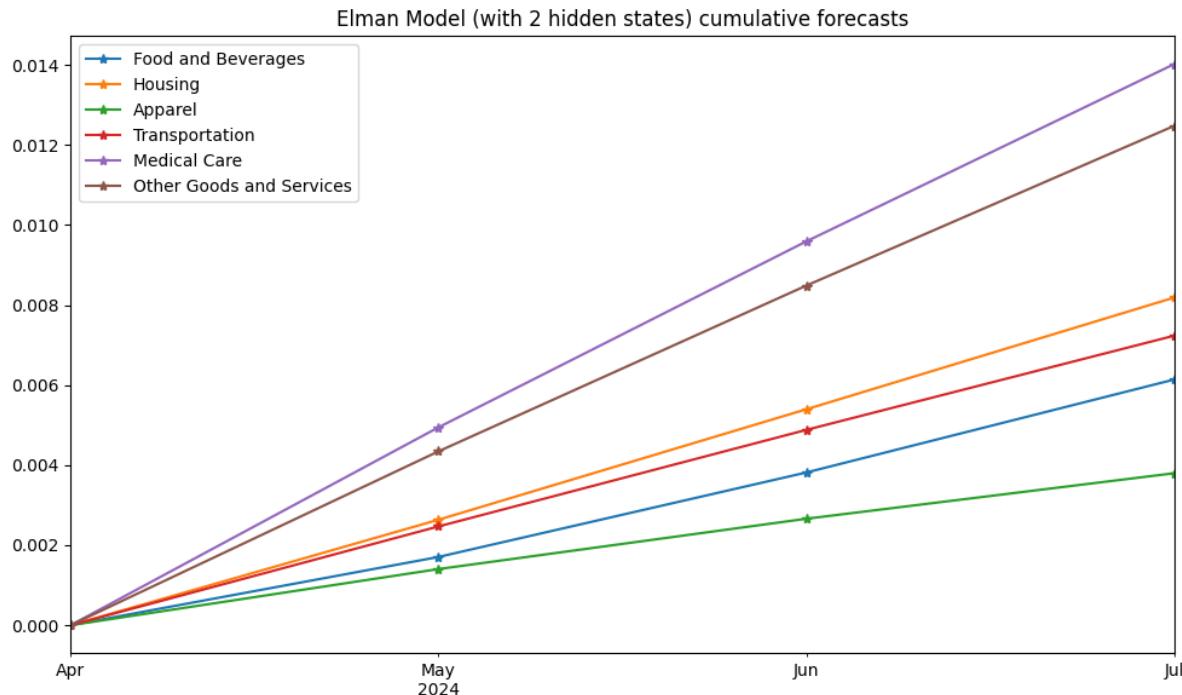
| | Food and Beverages | Housing | Apparel | Transportation | Medical Care | \ |
|--------------------------|--------------------|----------|----------|----------------|--------------|---|
| 2024-04 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | |
| 2024-05 | 0.001705 | 0.002635 | 0.001403 | 0.002468 | 0.004944 | |
| 2024-06 | 0.002111 | 0.002760 | 0.001257 | 0.002412 | 0.004646 | |
| 2024-07 | 0.002322 | 0.002791 | 0.001137 | 0.002353 | 0.004429 | |
| Other Goods and Services | | | | | | |
| 2024-04 | | 0.000000 | | | | |
| 2024-05 | | 0.004344 | | | | |
| 2024-06 | | 0.004142 | | | | |
| 2024-07 | | 0.003991 | | | | |

```
# Plot forecasts
fig, ax = plt.subplots(figsize=(10, 6))
```

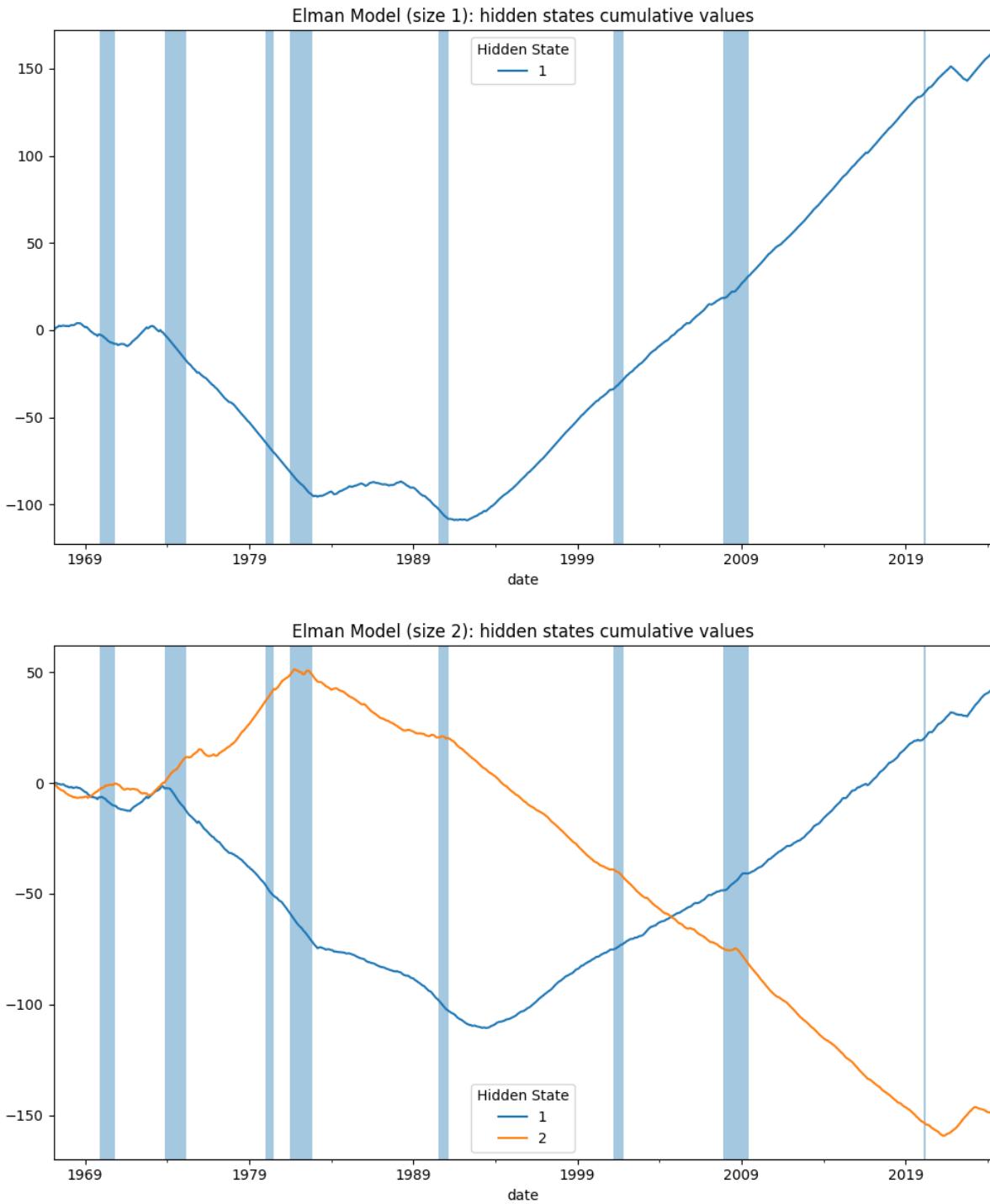
(continues on next page)

(continued from previous page)

```
forecasts.cumsum().plot(ax=ax, marker='*')
ax.set_title(f"Elman Model (with {K} hidden states) cumulative forecasts")
plt.tight_layout()
```



```
# 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()
```



31.2 Dynamic Factor Models

The basic model is:

$$y_t = \Lambda f_t + \epsilon_t$$

$$f_t = A_1 f_{t-1} + \cdots + A_2 f_{t-2} + u_t$$

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

The `DynamicFactorMQ` model in `statsmodels` package uses the EM algorithm for parameter fitting, and so can accommodate a large number of observed variables. Specifications of the model can include any collection of blocks of factors, including different factor autoregression orders, and AR(1) processes for idiosyncratic disturbances. It can incorporate monthly/quarterly mixed frequency data, which allows for Nowcasting models.

```
# 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  10336.076878  4.277298      31
          bic      mse  parameters
2  10357.094374  4.273679      35
          bic      mse  parameters
3  10360.490386  4.238931      39
          bic      mse  parameters
4  10377.211151  4.233325      43

```

```

# AR 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: 10336.076878452934
                                         Dynamic Factor Results
=====
Dep. Variable:      "Food and Beverages", and 5 more    No. Observations: 687
Model:                  Dynamic Factor Model    Log Likelihood -5066.787
                         + 2 factors in 1 blocks    AIC 10195.575
                                         + AR(1) idiosyncratic    BIC 10336.077
Date:                  Mon, 27 May 2024    HQIC 10249.934
Time:                      14:39:53    EM Iterations 221
Sample:                  02-28-1967
                         - 04-30-2024
Covariance Type:    Not computed
=====
```

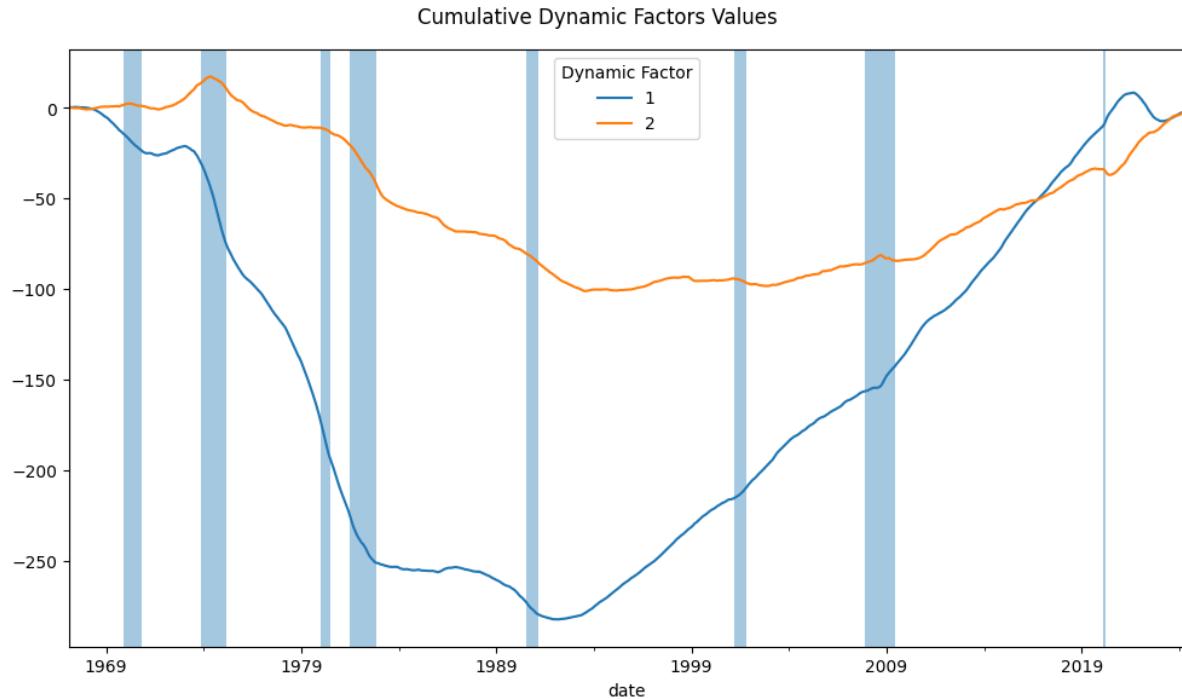
```

# 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()

# Show forecasts of selected series
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: 0.7115009239837685
Dynamic Factor Model Test MSE: 0.7963197085903823
Number of parameters: 31
```



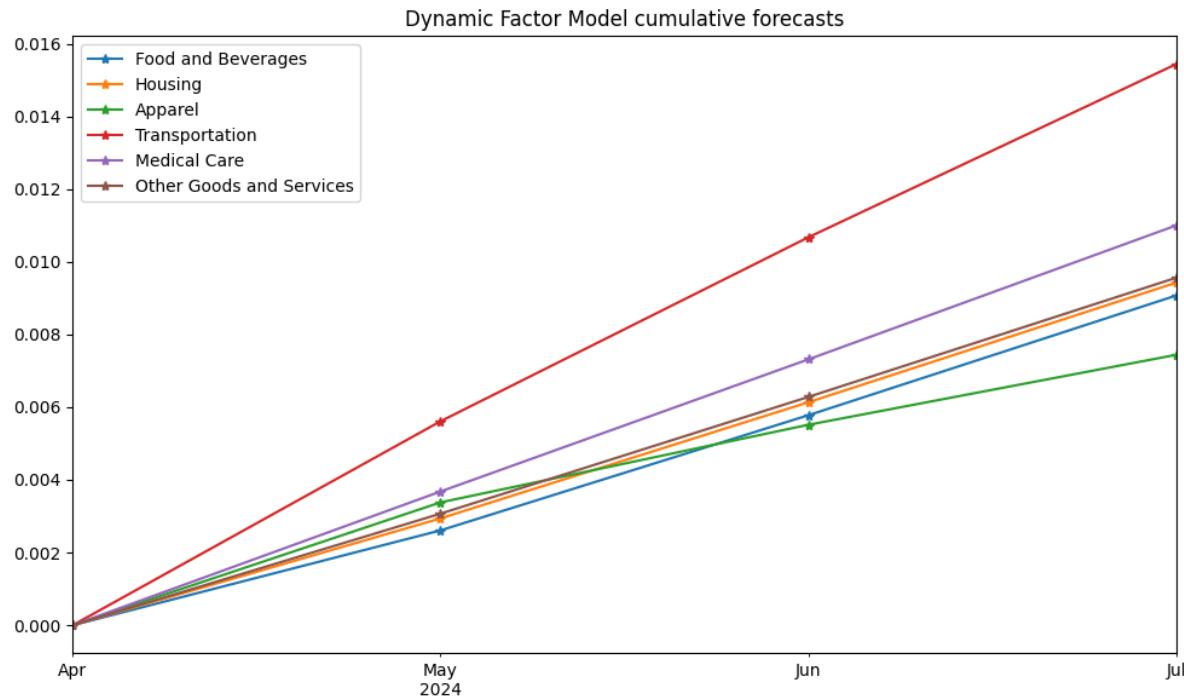
```
model_out = scaler.inverse_transform(model['forecast'].iloc[:nforecast])
model_out = DataFrame(np.vstack((np.zeros(n_features), model_out)),
                      index=pd.PeriodIndex(t, freq='M'), columns=scaled_data.columns)
print("Monthly forecasts from Dynamic Factor Model")
model_out
```

Monthly forecasts from Dynamic Factor Model

| | Food and Beverages | Housing | Apparel | Transportation | Medical Care | \ |
|---------|--------------------|----------|----------|----------------|--------------|---|
| 2024-04 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | |
| 2024-05 | 0.002610 | 0.002940 | 0.003380 | 0.005612 | 0.003679 | |
| 2024-06 | 0.003168 | 0.003195 | 0.002135 | 0.005062 | 0.003635 | |
| 2024-07 | 0.003294 | 0.003290 | 0.001925 | 0.004764 | 0.003685 | |

| | Other Goods and Services |
|---------|--------------------------|
| 2024-04 | 0.000000 |
| 2024-05 | 0.003070 |
| 2024-06 | 0.003211 |
| 2024-07 | 0.003279 |

```
# 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()
```



```
# 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.
7325

| Hidden State | 1 | 2 |
|--------------|--------|-------|
| R-square | 0.7409 | 0.724 |
| pvalue | 0.0000 | 0.000 |

CONVOLUTIONAL NEURAL NETWORKS

Life can only be understood backwards; but it must be lived forwards. - Søren Kierkegaard

Concepts

- Temporal Convolutional Net (TCN)
- Vector Autoregression

TODO

- either TCN/VAR of all the series, of the PCE components, or most popular or representative

```
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
```

```
# up to max number p of VAR(p) lags
maxlags = 6
```

```
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'
```

(continues on next page)

(continued from previous page)

```
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_602057/4272565358.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 | -1.514294 | -1.123374 | 0.528419 | 0.262158 | |
| 1967-03-31 | -0.803761 | -1.123374 | 0.117546 | -0.270473 | |
| 1967-04-30 | -1.516345 | -0.064928 | 0.523552 | 0.258920 | |
| 1967-05-31 | -0.803761 | -0.068382 | 0.115127 | -0.006978 | |
| 1967-06-30 | 1.327849 | -1.123374 | 0.518742 | -0.270473 | |
| ... | ... | ... | ... | ... | ... |
| 2023-12-31 | -0.305673 | -0.150090 | -0.300364 | -0.226813 | |
| 2024-01-31 | 0.136279 | 0.874513 | -1.728819 | -0.823810 | |
| 2024-02-29 | -0.763532 | 0.154003 | 0.841913 | 0.932892 | |
| 2024-03-31 | -0.555699 | 0.155815 | 1.046454 | 0.409135 | |
| 2024-04-30 | -0.757514 | -0.408180 | 2.163913 | 0.318836 | |
| | Medical Care | Other Goods and Services | | | |
| date | | | | | |
| 1967-02-28 | -0.260945 | | -1.020565 | | |
| 1967-03-31 | -0.265522 | | -0.290026 | | |
| 1967-04-30 | 0.977131 | | -1.020565 | | |
| 1967-05-31 | -0.279054 | | -0.292134 | | |
| 1967-06-30 | 0.950357 | | -0.294231 | | |
| ... | ... | | ... | | |
| 2023-12-31 | -0.199913 | | -0.950240 | | |
| 2024-01-31 | 0.045640 | | 0.340696 | | |
| 2024-02-29 | -1.610667 | | -1.786460 | | |
| 2024-03-31 | 0.152838 | | -0.080117 | | |
| 2024-04-30 | 0.016674 | | 0.048790 | | |

(continues on next page)

(continued from previous page)

[687 rows x 6 columns]

```
ntrain = sum(scaled_data.index < split_date)
M = scaled_data.shape[1]      # M is number of time series
```

32.1 Temporal Convolutional Net (TCN)

- convolution as a filter such as a weighted average
- hyperparameters:
 - dropout
 - hidden layers and size
 - batch size and shuffling

TODO:

- example of 1D 2-channel and weighted average

```
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,
                               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):
```

(continues on next page)

(continued from previous page)

```

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

```

```

# Model training parameters
seq_len = 8          # length of each input sequence for TCN
batch_size = 16
step_size = 100       # 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 = {}

```

```

# 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 a 1D Convolution Model

```

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] = []
optimizer = torch.optim.Adam(model.parameters())
loss_function = nn.MSELoss()

```

```

Conv1d(6, 6, kernel_size=(1,), stride=(1,))
=====

```

(continues on next page)

(continued from previous page)

| Layer (type:depth-idx) | Param # |
|------------------------|---------|
| Convid | 42 |
| Total params: | 42 |
| Trainable params: | 42 |
| Non-trainable params: | 0 |

```
# train_ex should have dimension (batch size, channels, sequence length+1)
train_ex = torch.tensor(scaled_data.values[:ntrain].T) [None,:,:,:].float().to(device)
```

```
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() [0,:,:,:].T
    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])
    }
```

```
conv1d_weights = np.vstack([model.bias.cpu().detach().numpy(),
                           model.weight.cpu().detach().numpy()[:, :, 0].T])
```

Fit TCN models

```
for block, kernel_size, dropout in [[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]]:
    modelname = f"TCN(b={block},k={kernel_size},d={dropout:.1f})"
    train_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,
                dropout=dropout).to(device)
    print()
    print('*****', modelname, '*****')
```

(continues on next page)

(continued from previous page)

```

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)[:, :, :].float().to(device)
pred = model(X).cpu().detach().numpy()[:, :, :].T
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])
}
#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 #
=====
TCN                                --
|---Sequential: 1-1                --
|   |---CausalConv1dBlock: 2-1    --
|   |   |---Sequential: 3-1      156
|   |---Conv1d: 1-2                42
=====
```

(continues on next page)

(continued from previous page)

```

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
|  └Convid: 1-2                   42
=====
Total params: 210
Trainable params: 210
Non-trainable params: 0
=====
***** TCN(b=2, 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,))
            )
        )
    )
)

```

(continues on next page)

(continued from previous page)

```

        (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=1,k=1,d=0.5) *****
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=0.5, 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
|   |   |   84
|---Conv1d: 1-2
|   42
=====
Total params: 126
Trainable params: 126
Non-trainable params: 0
=====

***** TCN(b=1,k=2,d=0.5) *****
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=0.5, 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
|---Conv1d: 1-2
|   42
=====
Total params: 198
Trainable params: 198
Non-trainable params: 0
=====

***** TCN(b=2,k=1,d=0.5) *****
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=0.5, inplace=False)
            )
        )
    )
)
=====
```

(continues on next page)

(continued from previous page)

```

        )
    (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=0.5, 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=0.5) *****
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=0.5, 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=0.5, inplace=False)
            )
        )
    )
)

```

(continues on next page)

(continued from previous page)

```

        )
        (classifier): Conv1d(6, 6, kernel_size=(1,), stride=(1,))
    )
=====

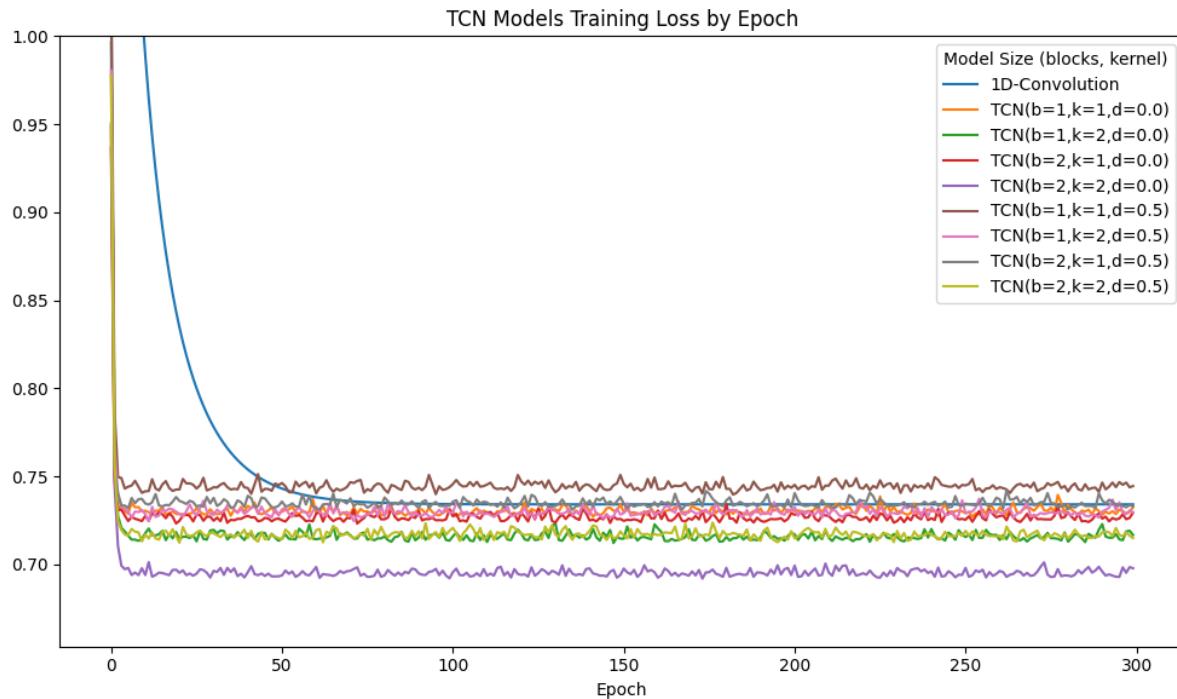
```

| 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 | |

```

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()

```



```

print('Sorted by Test Error')
DataFrame(results).T.sort_values('Test Error')

```

Sorted by Test Error

| | Train Error | Test Error |
|-----------------------|-------------|------------|
| TCN (b=2, k=1, d=0.5) | 0.730975 | 0.797471 |
| 1D-Convolution | 0.734049 | 0.804834 |
| TCN (b=1, k=1, d=0.0) | 0.730197 | 0.823324 |
| TCN (b=2, k=2, d=0.5) | 0.703123 | 0.825678 |
| TCN (b=1, k=2, d=0.5) | 0.716962 | 0.852393 |
| TCN (b=1, k=1, d=0.5) | 0.735571 | 0.865943 |
| TCN (b=1, k=2, d=0.0) | 0.708393 | 0.877891 |
| TCN (b=2, k=1, d=0.0) | 0.726959 | 0.881181 |
| TCN (b=2, k=2, d=0.0) | 0.706048 | 0.932908 |

32.2 Vector Autoregression

predict multiple time series (of the extracted approximate factors)

- vector autoregression as a weighted moving average

```
var_model = VAR(scaled_data.iloc[:ntrain], freq='ME')
```

32.2.1 Lag order

The lagged coefficients estimated from the Vector Autoregression produce a multi-period cumulative forecast

```
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: 2024-05-27 11:45:11
```

(continues on next page)

(continued from previous page)

Time: 14:50:03

| | | | |
|-------------------|----------|------------------|----------|
| No. of Equations: | 6.00000 | BIC: | -1.69952 |
| Nobs: | 657.000 | HQIC: | -1.87519 |
| Log likelihood: | -4898.92 | FPE: | 0.137188 |
| AIC: | -1.98641 | Det (Omega_mle): | 0.128736 |

Results for equation Food and Beverages

| prob | coefficient | std. error | t-stat |
|-----------------------------|-------------|------------|--------|
| const | -0.002494 | 0.035539 | -0.070 |
| L1.Food and Beverages | 0.310812 | 0.037875 | 8.206 |
| L1.Housing | 0.213124 | 0.043391 | 4.912 |
| L1.Apparel | -0.003392 | 0.037857 | -0.090 |
| L1.Transportation | -0.005201 | 0.037952 | -0.137 |
| L1.Medical Care | -0.007755 | 0.042442 | -0.183 |
| L1.Other Goods and Services | 0.009024 | 0.037218 | 0.242 |

Results for equation Housing

| prob | coefficient | std. error | t-stat |
|-----------------------------|-------------|------------|--------|
| const | -0.013239 | 0.027721 | -0.478 |
| L1.Food and Beverages | 0.163946 | 0.029543 | 5.549 |
| L1.Housing | 0.494461 | 0.033845 | 14.609 |
| L1.Apparel | 0.038379 | 0.029529 | 1.300 |
| L1.Transportation | 0.072678 | 0.029603 | 2.455 |
| L1.Medical Care | 0.209727 | 0.033105 | 6.335 |
| L1.Other Goods and Services | -0.022371 | 0.029031 | -0.771 |

Results for equation Apparel

| prob | coefficient | std. error | t-stat |
|------|-------------|------------|--------|
|------|-------------|------------|--------|

(continues on next page)

(continued from previous page)

| const | | -0.007941 | 0.036635 | -0.217 |
|-----------------------------|-------|-----------|----------|--------|
| L1.Food and Beverages | 0.828 | 0.031403 | 0.039043 | 0.804 |
| L1.Housing | 0.421 | 0.110336 | 0.044729 | 2.467 |
| L1.Apparel | 0.014 | 0.139551 | 0.039025 | 3.576 |
| L1.Transportation | 0.000 | 0.186002 | 0.039123 | 4.754 |
| L1.Medical Care | 0.007 | 0.117097 | 0.043751 | 2.676 |
| L1.Other Goods and Services | 0.851 | 0.007189 | 0.038366 | 0.187 |

Results for equation Transportation

| | prob | coefficient | std. error | t-stat |
|-----------------------------|-------|-------------|------------|--------|
| const | 0.996 | 0.000177 | 0.034475 | 0.005 |
| L1.Food and Beverages | 0.584 | 0.020122 | 0.036742 | 0.548 |
| L1.Housing | 0.085 | 0.072483 | 0.042092 | 1.722 |
| L1.Apparel | 0.324 | 0.036200 | 0.036724 | 0.986 |
| L1.Transportation | 0.000 | 0.422621 | 0.036816 | 11.479 |
| L1.Medical Care | 0.235 | 0.048914 | 0.041172 | 1.188 |
| L1.Other Goods and Services | 0.348 | -0.033877 | 0.036105 | -0.938 |

Results for equation Medical Care

| | prob | coefficient | std. error | t-stat |
|-----------------------|-------|-------------|------------|--------|
| const | 0.342 | 0.027613 | 0.029051 | 0.951 |
| L1.Food and Beverages | 0.421 | 0.024938 | 0.030960 | 0.805 |
| L1.Housing | 0.000 | 0.234873 | 0.035469 | 6.622 |
| L1.Apparel | 0.164 | 0.043036 | 0.030946 | 1.391 |
| L1.Transportation | 0.875 | 0.004867 | 0.031023 | 0.157 |
| L1.Medical Care | | 0.431491 | 0.034694 | 12.437 |

(continues on next page)

(continued from previous page)

| | | | | |
|-----------------------------|----------|----------|-------|---|
| ↳ 0.000 | | | | |
| L1.Other Goods and Services | 0.115051 | 0.030424 | 3.782 | ↳ |
| ↳ 0.000 | | | | |

Results for equation Other Goods and Services

| ↳ prob | coefficient | std. error | t-stat | ↳ |
|-----------------------------|-------------|------------|--------|---|
| const | -0.010534 | 0.037357 | -0.282 | ↳ |
| ↳ 0.778 | | | | |
| L1.Food and Beverages | 0.026800 | 0.039813 | 0.673 | ↳ |
| ↳ 0.501 | | | | |
| L1.Housing | 0.077605 | 0.045611 | 1.701 | ↳ |
| ↳ 0.089 | | | | |
| L1.Apparel | 0.095652 | 0.039795 | 2.404 | ↳ |
| ↳ 0.016 | | | | |
| L1.Transportation | 0.017545 | 0.039894 | 0.440 | ↳ |
| ↳ 0.660 | | | | |
| L1.Medical Care | 0.257502 | 0.044614 | 5.772 | ↳ |
| ↳ 0.000 | | | | |
| L1.Other Goods and Services | 0.011147 | 0.039123 | 0.285 | ↳ |
| ↳ 0.776 | | | | |

Correlation matrix of residuals

| | Food and Beverages | Housing | Apparel | Transportation | ↳ |
|--------------------------|--------------------|-----------|----------|----------------|-----------|
| ↳ Medical Care | 0.014996 | 1.000000 | 0.147115 | 0.084629 | -0.001656 |
| Food and Beverages | | -0.008149 | | | |
| ↳ 0.095471 | | 0.147115 | 1.000000 | 0.091234 | 0.132240 |
| Housing | | 0.019458 | | | |
| Apparel | | 0.084629 | 0.091234 | 1.000000 | 0.128269 |
| Transportation | | -0.001656 | 0.132240 | 0.128269 | 1.000000 |
| ↳ -0.053153 | | -0.013329 | | | |
| Medical Care | | 0.014996 | 0.095471 | 0.036681 | -0.053153 |
| ↳ 1.000000 | | 0.153789 | | | |
| Other Goods and Services | | -0.008149 | 0.019458 | 0.024285 | -0.013329 |
| ↳ 0.153789 | | 1.000000 | | | |

Compare VAR(1) model coefficients to fitted weights of 1D convolution

```
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 | -0.0025 | -0.0132 | -0.0079 | |
| L1.Food and Beverages | 0.3108 | 0.1639 | 0.0314 | |

(continues on next page)

(continued from previous page)

| | | | |
|-----------------------------|--------------------------|--------------|--------|
| L1.Housing | 0.2131 | 0.4945 | 0.1103 |
| L1.Apparel | -0.0034 | 0.0384 | 0.1396 |
| L1.Transportation | -0.0052 | 0.0727 | 0.1860 |
| L1.Medical Care | -0.0078 | 0.2097 | 0.1171 |
| L1.Other Goods and Services | 0.0090 | -0.0224 | 0.0072 |
| | Transportation | Medical Care | \ |
| const | 0.0002 | 0.0276 | |
| L1.Food and Beverages | 0.0201 | 0.0249 | |
| L1.Housing | 0.0725 | 0.2349 | |
| L1.Apparel | 0.0362 | 0.0430 | |
| L1.Transportation | 0.4226 | 0.0049 | |
| L1.Medical Care | 0.0489 | 0.4315 | |
| L1.Other Goods and Services | -0.0339 | 0.1151 | |
| | Other Goods and Services | | |
| const | -0.0105 | | |
| L1.Food and Beverages | 0.0268 | | |
| L1.Housing | 0.0776 | | |
| L1.Apparel | 0.0957 | | |
| L1.Transportation | 0.0175 | | |
| L1.Medical Care | 0.2575 | | |
| L1.Other Goods and Services | 0.0111 | | |

```
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 | -0.0025 | -0.0132 | -0.0079 | |
| Food and Beverages | 0.3108 | 0.1639 | 0.0314 | |
| Housing | 0.2131 | 0.4945 | 0.1103 | |
| Apparel | -0.0034 | 0.0384 | 0.1396 | |
| Transportation | -0.0052 | 0.0727 | 0.1860 | |
| Medical Care | -0.0078 | 0.2097 | 0.1171 | |
| Other Goods and Services | 0.0090 | -0.0224 | 0.0072 | |
| | Transportation | Medical Care | \ | |
| bias | 0.0002 | 0.0276 | | |
| Food and Beverages | 0.0201 | 0.0249 | | |
| Housing | 0.0725 | 0.2349 | | |
| Apparel | 0.0362 | 0.0430 | | |
| Transportation | 0.4226 | 0.0049 | | |
| Medical Care | 0.0489 | 0.4315 | | |
| Other Goods and Services | -0.0339 | 0.1151 | | |
| | Other Goods and Services | | | |
| bias | -0.0105 | | | |
| Food and Beverages | 0.0268 | | | |
| Housing | 0.0776 | | | |
| Apparel | 0.0957 | | | |
| Transportation | 0.0175 | | | |
| Medical Care | 0.2575 | | | |

(continues on next page)

(continued from previous page)

| | |
|--------------------------|--------|
| Other Goods and Services | 0.0111 |
|--------------------------|--------|

```
# 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()))

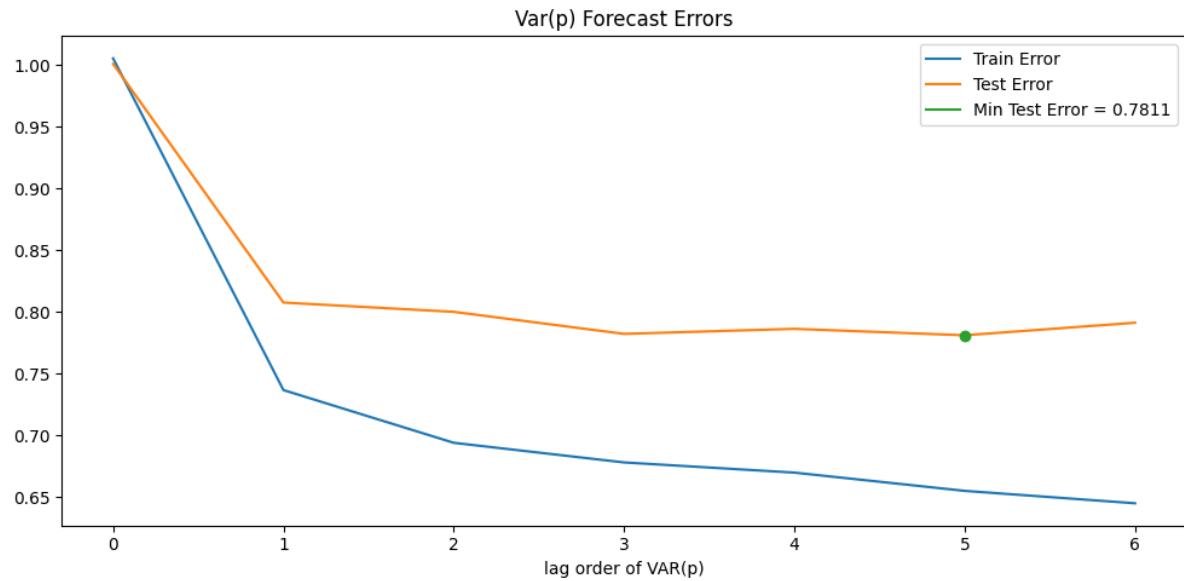
    # accumulate to error of VAR(p) model forecasts
    for p in range(1, maxlags+1):
        pred = var_models[p].forecast(scaled_data.iloc[:i].values, 1)
        var_errors[p].append(mean_squared_error(data.reshape(1, -1), pred))
```

```
# 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) | 1.005237 | 1.000794 |
| VAR(1) | 0.736693 | 0.807536 |
| VAR(2) | 0.694004 | 0.800034 |
| VAR(3) | 0.678115 | 0.782192 |
| VAR(4) | 0.669869 | 0.786281 |
| VAR(5) | 0.655099 | 0.781057 |
| VAR(6) | 0.645045 | 0.791213 |

```
# 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()
```



CHAPTER
THIRTYTHREE

REINFORCEMENT LEARNING

UNDER CONSTRUCTION

I have not failed. I've just found 10,000 ways that won't work - Thomas A. Edison

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.

TODO:

- shortfall tail probability: (1) probability of shortfall (2) expected years of shortfall in 5% tail
- for historical simulation, save matrices to compare: does one model's matrix dominate another

```
from pandas import DataFrame, Series
import pandas as pd
import numpy as np
import math
import random
import matplotlib.pyplot as plt
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
from secret import credentials, paths
store = Store(paths['scratch'])
#pd.set_option('display.max_rows', None)
VERBOSE = 0
```

```
# open connections
sql = SQL(**credentials['sql'], verbose=VERBOSE)
bd = BusDay(sql, verbose=VERBOSE)
outdir = paths['scratch'] / 'RL'
```

Stocks, Bonds, Bills and Inflation

Using data beginning in 1926, the SBBI dataset includes monthly, quarterly, and yearly total returns and yields of most of the major U.S asset classes: large-cap stocks, small-cap stocks, corporate bonds, government bonds of several maturities, and inflation.

```
# 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 |
| ... | ... | ... | ... |
| 20230228 | -0.024399 | -0.042743 | 0.005582 |
| 20230331 | 0.036714 | 0.042510 | 0.003311 |
| 20230430 | 0.015609 | 0.012058 | 0.005059 |
| 20230531 | 0.004347 | -0.030476 | 0.002518 |
| 20230630 | 0.066075 | -0.003346 | 0.003229 |

[1170 rows x 3 columns]

```
# 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 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 | 0.105343 | 0.055095 | 0.036288 | 799 |

33.1 Retirement spending policy

We test the sensitivity of the common heuristic for retirement spending, a fixed annual withdrawal rate, on models driven by three key variables: the asset allocation of the portfolio, the market environment, and the length of the retiree's drawdown period. For example, the typical rule of a fixed 4% withdrawal rate (with annual inflation adjustment) for 50% stock/50% bond portfolios can be evaluated over rolling historical 30-year periods, to estimate the likelihood that a retiree would not run out of money over a 30-year time horizon.

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.”

The default strategy is to rebalance each year to fixed initial allocation, and withdrawal is fixed at the given percent of initial wealth adjusted by annual inflation.

```

class BaseModel:
    """Buy-and-hold allocation model where assets weights drift from initial"""

    name = 'initial'
    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:

```

(continues on next page)

(continued from previous page)

```

        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])

```

```

class FixedModel(BaseModel):
    """Allocation model where assets are rebalanced to fixed weights"""

    name = 'fixed'
    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])

```

Base simulations of 50-50 4% spending policy over all 30-year sample periods

```

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])

    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: 799
Probability of shortfall: 0.0763

```

```

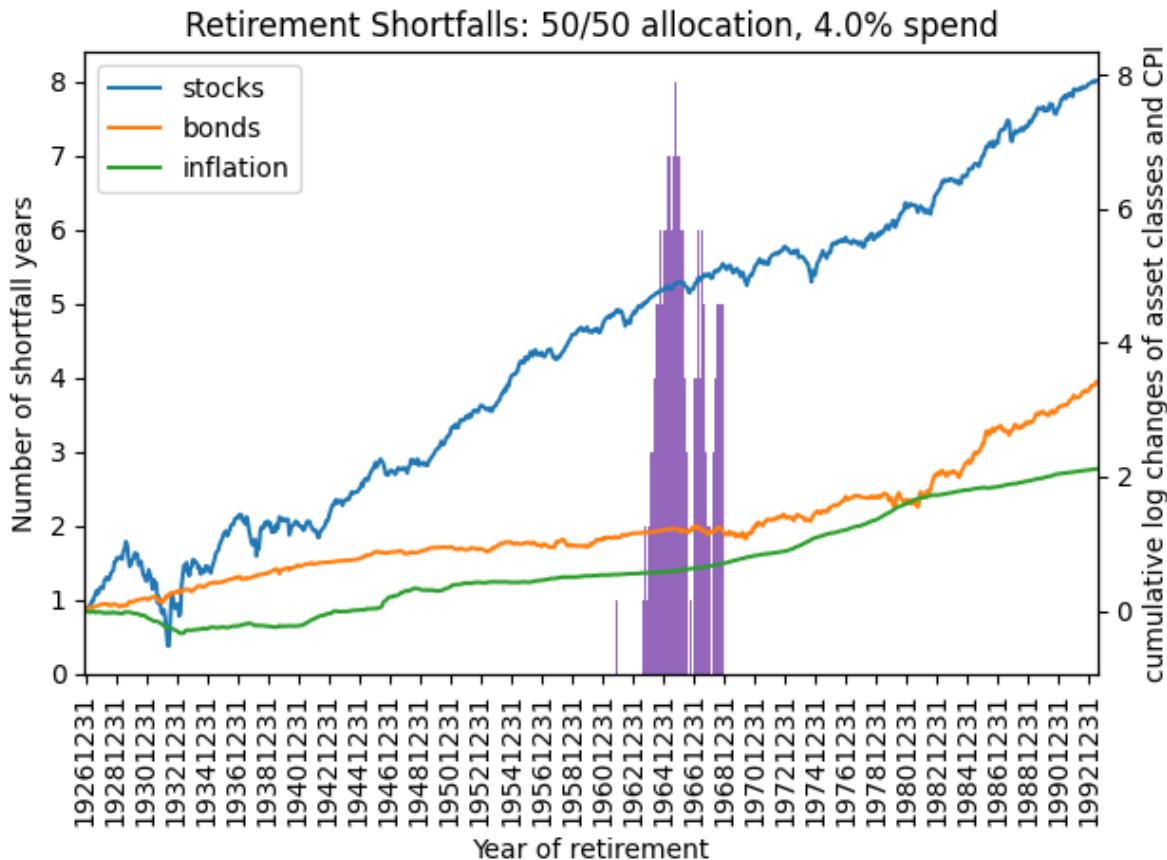
fails = Series(result).sort_index().rename('shortfall_years')
market = df.reindex(fails.index).cumsum()
fails.index = fails.index.astype(str)
ax = fails.plot(kind='bar', width=1.0, color='C4')
ax.set_title(f"Retirement Shortfalls: {alloc}/{100-alloc} allocation, {rule}% spend")
ax.set_ylabel('Number of shortfall years')
ax.set_xlabel('Year of retirement')
set_xticks(ax=ax, nskip=23, rotation=90)
bx = ax.twinx()
market.index = market.index.astype(str)
market.plot(ax=bx)

```

(continues on next page)

(continued from previous page)

```
bx.set_ylabel('cumulative log changes of asset classes and CPI')
bx.legend()
plt.tight_layout()
```



33.2 Deep reinforcement learning

Gymnasium (formerly OpenAI gym) custom environment **Gymnasium** provides a simply and pythonic interface for representing general Reinforcement Learning problems. It is a maintained fork of OpenAI's Gym library.

<https://gymnasium.farama.org/index.html>

pip install gymnasium

Stable Baselines3 (SB3) is a set of implementations of reinforcement learning algorithms in PyTorch.

<https://github.com/DLR-RM/stable-baselines3>

pip install stable-baselines3[extra]

Reward is $-(T\text{FAIL})$ if truncated else $\text{math.sqrt}(W / (S \cdot (T+1)))$

- if insufficient: reward is -100 times square of remaining years
- else reward is a concave (sqrt) function of wealth to spending coverage
- when positive, prefer that wealth to remaining spending ratio is higher

Learns to predict action from observing current state, and exploiting/exploring

- current wealth and allocation,
- recent market (equity and bonds) and inflation changes
- spending amount
- years since retirement

Exploit action with greatest expected value

- value is recursive expectation: many algorithms e.g. “Q-learning”, “SARSA”

Explore actions that are not necessarily best (at the time) during training

- to gather information for improving decision

```
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
```

(continues on next page)

(continued from previous page)

```

deltas = self.episode.iloc[self.n].tolist()

# 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, {}

```

Helpers to evaluate trained model

```

def compute_shortfall(x, q):
    """Compute expected years of shortfall given probability level"""
    x = sorted(x)
    q = int(q * len(x))
    return x[q], np.mean(x[q:])

```

```

TAIL = 0.95
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(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

```

Stable Baselines 3 with DQN algorithm for discrete action spaces

DQN (Deep Q Network) provides vanilla Deep Q-Learning

```

initial_alloc = 50
rules = np.arange(3, 5.1, 0.1)

```

```

TIMESTEPS = int(5e5)
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}")

```

(continues on next page)

(continued from previous page)

```
# evaluate model
test_clf = clf.load(f"{outdir}/{name}", env=None)
fail[name], quantile[name], shortfall[name], actions[name] = \
    evaluate(env, test_clf, episodes)

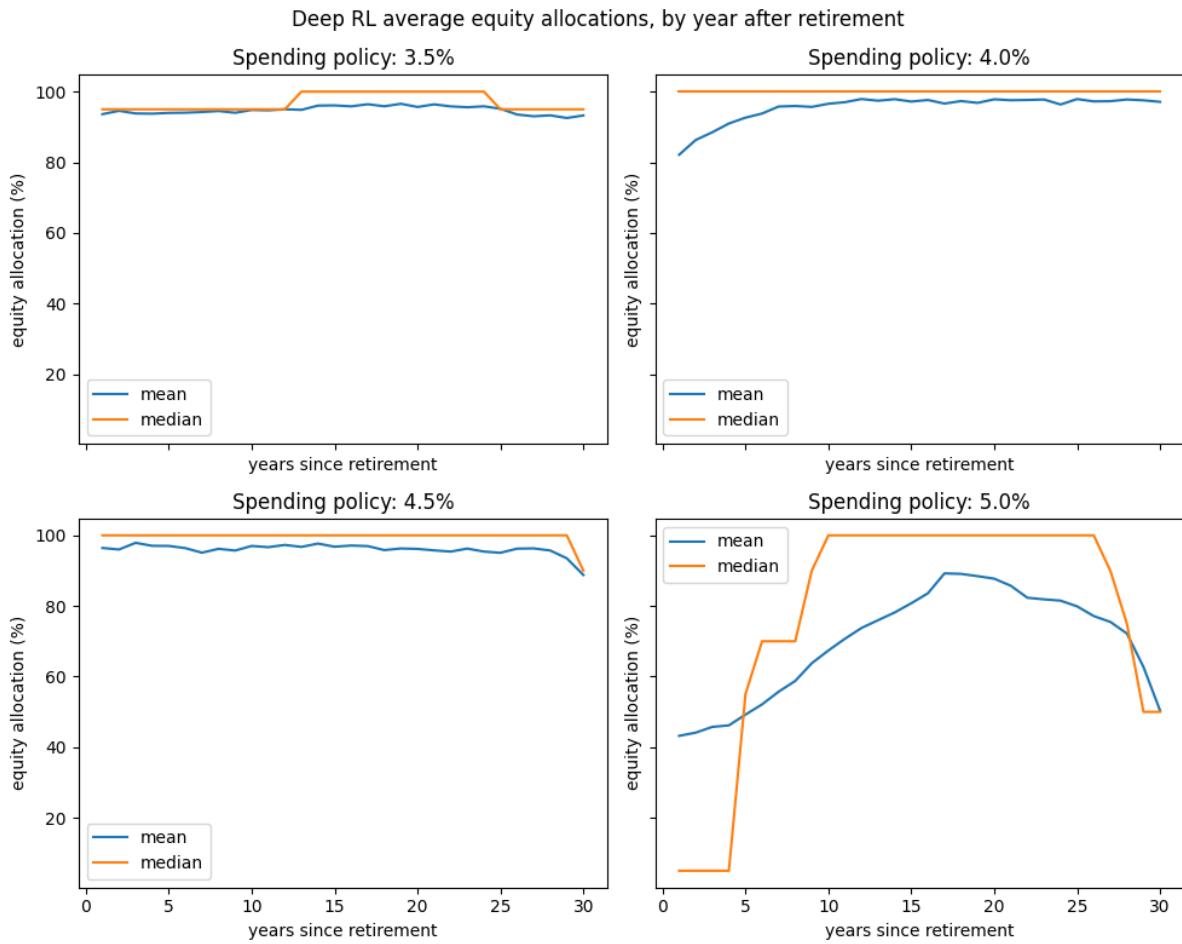
store['dqn'] = dict(fail=fail, shortfall=shortfall,
                    quantile=quantile, actions=actions)
```

```
0%|          | 0/21 [00:00<?, ?it/s]/home/terence/env3.11/lib/python3.11/site-
  ↵ packages/tqdm/auto.py:21: TqdmWarning: IPProgress not found. Please update
  ↵ jupyter and ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/user_
  ↵ install.html
  ↵ from .autonotebook import tqdm as notebook_tqdm
100%|██████████| 21/21 [1:42:19<00:00, 292.37s/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 | 0.02 | 0.0 | 0.0 | 0.01 | 0.01 | 0.01 | 0.02 | 0.01 | 0.03 | 0.03 | 0.05 |
| ↳ 0.07 | 0.08 | 0.12 | 0.13 | 0.14 | 0.14 | 0.18 | 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 since retirement')
    ax.set_ylabel('equity allocation (%)')
    ax.legend()
plt.tight_layout()
```



33.2.1 Historical simulations

Extended grid of simulations, by asset allocation target and spending withdrawal rate

```
# simulate fixed and initial equity allocations from 0 to 100%
allocs = np.arange(0, 105, 5)
for num, Model in enumerate([BaseModel, FixedModel]):
    fail = DataFrame(columns=rules, index=allocs, dtype=float)
    shortfall = DataFrame(columns=rules, index=allocs, dtype=float)
    quantile = DataFrame(columns=rules, index=allocs, dtype=float)

    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)
                    result.append(obs)
            fail[alloc] = result[0]
            shortfall[alloc] = result[1]
            quantile[alloc] = result[2]
```

(continues on next page)

(continued from previous page)

```

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 [06:53<00:00, 19.71s/it]
100%|██████████| 21/21 [06:54<00:00, 19.74s/it]

TODO: Heatmap with (numbers)

```

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} asset allocation')
    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()

```

```

tail = int(100 * (1 - TAIL))
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())

```

| Probability of Shortfall: with intial allocation | | | | | | | | | | | | | | | |
|--|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| 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 | |
| 100 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.01 | 0.01 | 0.01 | 0.02 | 0.02 | 0.02 | 0.03 | 0.05 | 0.07 | 0.08 |
| 95 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.01 | 0.01 | 0.01 | 0.03 | 0.04 | 0.06 | 0.07 | 0.08 |
| 90 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.01 | 0.02 | 0.04 | 0.06 | 0.07 | 0.08 | 0.09 |
| 85 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.01 | 0.02 | 0.04 | 0.06 | 0.07 | 0.08 | 0.09 |
| 80 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.01 | 0.02 | 0.05 | 0.06 | 0.07 | 0.07 | 0.08 |
| 75 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.01 | 0.04 | 0.05 | 0.07 | 0.08 | 0.09 | 0.10 |
| 70 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.01 | 0.02 | 0.04 | 0.06 | 0.07 | 0.08 | 0.09 | 0.11 |
| 65 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.02 | 0.04 | 0.05 | 0.07 | 0.07 | 0.09 | 0.10 | 0.12 |
| 60 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.01 | 0.03 | 0.04 | 0.06 | 0.07 | 0.09 | 0.10 | 0.13 |

(continues on next page)

(continued from previous page)

| | | | | | | | | | | | | | | |
|---|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| 55 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.02 | 0.04 | 0.05 | 0.06 | 0.08 | 0.09 | 0.11 |
| | 0.11 | 0.12 | 0.14 | 0.15 | 0.17 | 0.19 | 0.22 | 0.24 | | | | | | |
| 50 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.01 | 0.03 | 0.05 | 0.06 | 0.08 | 0.09 | 0.11 | 0.12 |
| | 0.12 | 0.13 | 0.15 | 0.17 | 0.19 | 0.22 | 0.24 | 0.26 | | | | | | |
| 45 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.01 | 0.02 | 0.04 | 0.06 | 0.07 | 0.09 | 0.11 | 0.12 | 0.13 |
| | 0.13 | 0.15 | 0.17 | 0.20 | 0.22 | 0.24 | 0.25 | 0.27 | | | | | | |
| 40 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.02 | 0.03 | 0.05 | 0.07 | 0.09 | 0.10 | 0.12 | 0.12 | 0.13 |
| | 0.15 | 0.17 | 0.20 | 0.22 | 0.25 | 0.26 | 0.27 | 0.28 | | | | | | |
| 35 | 0.00 | 0.00 | 0.00 | 0.00 | 0.01 | 0.03 | 0.04 | 0.06 | 0.09 | 0.11 | 0.11 | 0.12 | 0.14 | 0.15 |
| | 0.18 | 0.21 | 0.23 | 0.25 | 0.26 | 0.28 | 0.29 | 0.30 | | | | | | |
| 30 | 0.00 | 0.00 | 0.00 | 0.00 | 0.02 | 0.04 | 0.06 | 0.09 | 0.11 | 0.11 | 0.13 | 0.15 | 0.19 | 0.20 |
| | 0.21 | 0.23 | 0.25 | 0.28 | 0.28 | 0.30 | 0.31 | 0.31 | | | | | | |
| 25 | 0.00 | 0.00 | 0.00 | 0.02 | 0.04 | 0.05 | 0.08 | 0.10 | 0.12 | 0.14 | 0.17 | 0.19 | 0.22 | 0.23 |
| | 0.24 | 0.26 | 0.27 | 0.29 | 0.30 | 0.32 | 0.35 | 0.36 | | | | | | |
| 20 | 0.00 | 0.00 | 0.01 | 0.03 | 0.05 | 0.08 | 0.10 | 0.13 | 0.15 | 0.19 | 0.20 | 0.23 | 0.25 | 0.26 |
| | 0.27 | 0.28 | 0.31 | 0.33 | 0.35 | 0.37 | 0.38 | 0.41 | | | | | | |
| 15 | 0.00 | 0.01 | 0.03 | 0.05 | 0.09 | 0.12 | 0.14 | 0.18 | 0.20 | 0.22 | 0.23 | 0.26 | 0.29 | 0.30 |
| | 0.32 | 0.35 | 0.38 | 0.40 | 0.43 | 0.46 | 0.50 | 0.53 | | | | | | |
| 10 | 0.00 | 0.02 | 0.06 | 0.11 | 0.13 | 0.16 | 0.21 | 0.23 | 0.24 | 0.27 | 0.32 | 0.36 | 0.39 | 0.40 |
| | 0.44 | 0.49 | 0.53 | 0.56 | 0.58 | 0.59 | 0.60 | 0.61 | | | | | | |
| 5 | 0.03 | 0.09 | 0.15 | 0.19 | 0.25 | 0.29 | 0.32 | 0.38 | 0.44 | 0.48 | 0.52 | 0.55 | 0.58 | 0.59 |
| | 0.60 | 0.61 | 0.62 | 0.62 | 0.63 | 0.63 | 0.64 | 0.64 | | | | | | |
| 0 | 0.29 | 0.35 | 0.42 | 0.48 | 0.50 | 0.52 | 0.54 | 0.54 | 0.55 | 0.57 | 0.58 | 0.60 | 0.61 | 0.62 |
| | 0.61 | 0.63 | 0.63 | 0.64 | 0.64 | 0.65 | 0.66 | 0.69 | | | | | | |
| Probability of Shortfall: with fixed allocation | | | | | | | | | | | | | | |
| | 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 |
| | 4.3 | 4.4 | 4.5 | 4.6 | 4.7 | 4.8 | 4.9 | 5.0 | | | | | | |
| 100 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.01 | 0.01 | 0.01 | 0.02 | 0.02 | 0.02 | 0.03 | 0.05 | 0.07 |
| | 0.07 | 0.08 | 0.10 | 0.11 | 0.12 | 0.13 | 0.15 | 0.17 | | | | | | |
| 95 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.01 | 0.01 | 0.02 | 0.03 | 0.06 | 0.07 |
| | 0.07 | 0.08 | 0.10 | 0.10 | 0.11 | 0.14 | 0.15 | 0.17 | | | | | | |
| 90 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.01 | 0.03 | 0.05 | 0.07 |
| | 0.07 | 0.08 | 0.10 | 0.10 | 0.12 | 0.14 | 0.15 | 0.17 | | | | | | |
| 85 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.01 | 0.01 | 0.03 | 0.05 | 0.07 |
| | 0.07 | 0.08 | 0.09 | 0.10 | 0.12 | 0.14 | 0.16 | 0.17 | | | | | | |
| 80 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.02 | 0.04 | 0.06 | 0.07 |
| | 0.07 | 0.08 | 0.09 | 0.10 | 0.12 | 0.15 | 0.16 | 0.18 | | | | | | |
| 75 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.01 | 0.03 | 0.05 | 0.06 | 0.07 |
| | 0.07 | 0.08 | 0.09 | 0.11 | 0.13 | 0.15 | 0.16 | 0.18 | | | | | | |
| 70 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.01 | 0.04 | 0.06 | 0.07 | 0.08 |
| | 0.08 | 0.09 | 0.10 | 0.12 | 0.14 | 0.15 | 0.17 | 0.19 | | | | | | |
| 65 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.01 | 0.02 | 0.05 | 0.06 | 0.07 | 0.08 |
| | 0.08 | 0.10 | 0.12 | 0.13 | 0.14 | 0.16 | 0.18 | 0.20 | | | | | | |
| 60 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.02 | 0.04 | 0.06 | 0.06 | 0.08 | 0.09 |
| | 0.10 | 0.11 | 0.12 | 0.14 | 0.15 | 0.18 | 0.21 | 0.23 | | | | | | |
| 55 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.01 | 0.03 | 0.05 | 0.06 | 0.08 | 0.10 | 0.11 |
| | 0.11 | 0.12 | 0.13 | 0.16 | 0.18 | 0.21 | 0.23 | 0.25 | | | | | | |
| 50 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.02 | 0.04 | 0.06 | 0.07 | 0.10 | 0.11 | 0.12 |
| | 0.12 | 0.13 | 0.16 | 0.18 | 0.22 | 0.24 | 0.26 | 0.27 | | | | | | |
| 45 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.02 | 0.03 | 0.05 | 0.07 | 0.09 | 0.11 | 0.12 | 0.13 |
| | 0.13 | 0.16 | 0.19 | 0.22 | 0.25 | 0.26 | 0.28 | 0.31 | | | | | | |
| 40 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.01 | 0.03 | 0.04 | 0.07 | 0.09 | 0.11 | 0.12 | 0.14 | 0.15 |
| | 0.16 | 0.20 | 0.23 | 0.26 | 0.28 | 0.31 | 0.34 | 0.35 | | | | | | |
| 35 | 0.00 | 0.00 | 0.00 | 0.00 | 0.01 | 0.03 | 0.04 | 0.06 | 0.09 | 0.11 | 0.13 | 0.15 | 0.17 | 0.18 |
| | 0.21 | 0.25 | 0.28 | 0.31 | 0.34 | 0.36 | 0.38 | 0.40 | | | | | | |
| 30 | 0.00 | 0.00 | 0.00 | 0.00 | 0.02 | 0.04 | 0.06 | 0.09 | 0.11 | 0.14 | 0.16 | 0.20 | 0.24 | 0.25 |

(continues on next page)

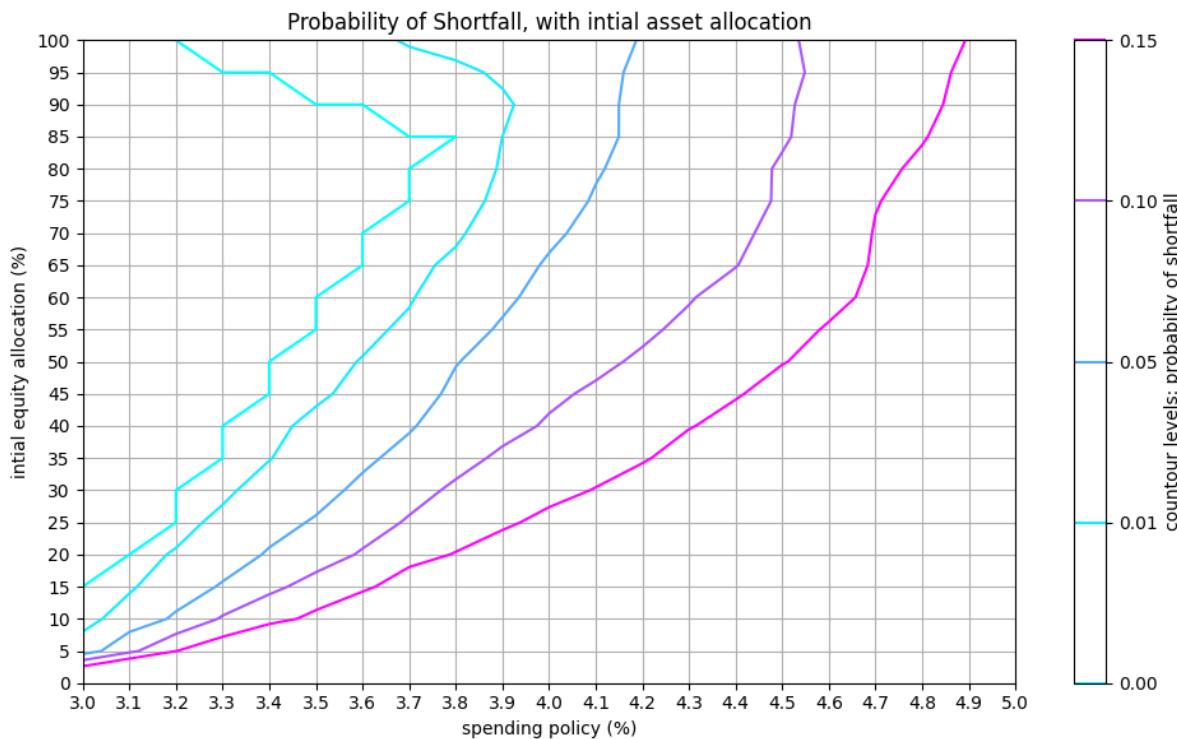
(continued from previous page)

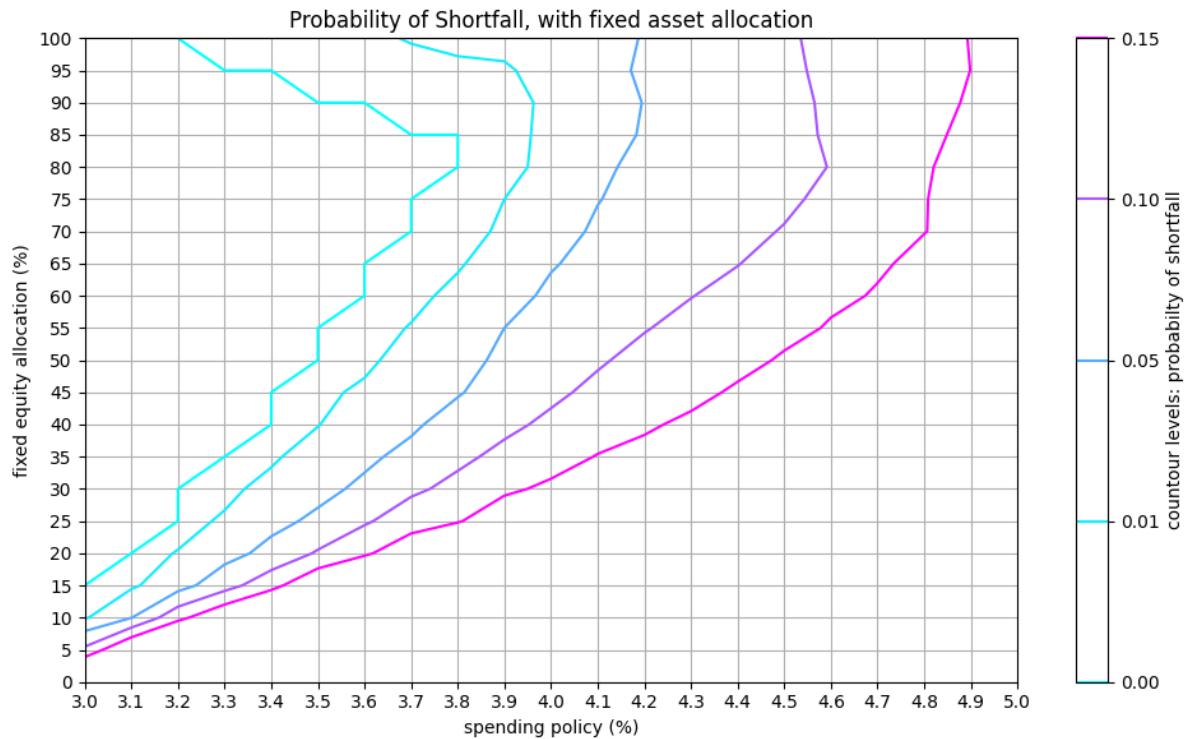
| | | | | | | | |
|--------|------|------|------|------|------|------|------|
| ↳ 0.28 | 0.31 | 0.34 | 0.37 | 0.39 | 0.41 | 0.44 | 0.47 |
| 25 | 0.00 | 0.00 | 0.00 | 0.01 | 0.04 | 0.06 | 0.09 |
| ↳ 0.34 | 0.38 | 0.41 | 0.44 | 0.49 | 0.54 | 0.56 | 0.58 |
| 20 | 0.00 | 0.00 | 0.01 | 0.03 | 0.06 | 0.11 | 0.14 |
| ↳ 0.46 | 0.50 | 0.53 | 0.55 | 0.58 | 0.60 | 0.61 | 0.61 |
| 15 | 0.00 | 0.01 | 0.03 | 0.08 | 0.13 | 0.20 | 0.25 |
| ↳ 0.55 | 0.58 | 0.60 | 0.60 | 0.61 | 0.62 | 0.62 | 0.62 |
| 10 | 0.01 | 0.05 | 0.14 | 0.20 | 0.25 | 0.33 | 0.38 |
| ↳ 0.59 | 0.60 | 0.61 | 0.62 | 0.62 | 0.63 | 0.63 | 0.63 |
| 5 | 0.11 | 0.21 | 0.28 | 0.34 | 0.40 | 0.46 | 0.49 |
| ↳ 0.61 | 0.61 | 0.62 | 0.62 | 0.63 | 0.63 | 0.64 | 0.65 |
| 0 | 0.29 | 0.35 | 0.42 | 0.48 | 0.50 | 0.52 | 0.54 |
| ↳ 0.61 | 0.63 | 0.63 | 0.64 | 0.64 | 0.65 | 0.66 | 0.69 |

```
compare = store[BaseModel.name]['fail'] < store[FixedModel.name]['fail']
print(compare.mean().mean())
```

```
0.3741496598639455
```

```
for model in [BaseModel, FixedModel]:
    fail = store[model.name]['fail']
    plot_contour(fail, levels=[0.0, .01,.05,.1,.15],
                 title="Probability of Shortfall",
                 label="contour levels: probability of shortfall")
```





```
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())
```

| Expected Years of Shortfall in 5% tail: (model.name) allocation | | | | | | | | | | | | | | | |
|---|------|------|------|------|------|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 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 |
| 4.4 | 4.5 | 4.6 | 4.7 | 4.8 | 4.9 | 5.0 | | | | | | | | | |
| 100 | 0.0 | 0.0 | 0.0 | 0.1 | 0.3 | 0.6 | 0.8 | 1.2 | 1.8 | 2.5 | 3.1 | 4.1 | 5.8 | 7.1 | 9.0 |
| 8.2 | 9.3 | 10.1 | 11.0 | 11.6 | 12.4 | 12.9 | | | | | | | | | |
| 95 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 | 0.2 | 0.4 | 0.7 | 1.1 | 2.1 | 3.2 | 5.0 | 6.3 | 8.0 |
| 7.6 | 8.7 | 9.6 | 10.4 | 11.0 | 11.6 | 12.2 | | | | | | | | | |
| 90 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 | 0.2 | 0.5 | 1.2 | 2.4 | 4.3 | 5.8 | 7.5 |
| 7.1 | 8.2 | 9.0 | 9.9 | 10.6 | 11.2 | 11.7 | | | | | | | | | |
| 85 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.3 | 1.0 | 2.3 | 4.2 | 5.6 | 7.3 |
| 6.9 | 7.9 | 8.9 | 9.6 | 10.4 | 11.0 | 11.7 | | | | | | | | | |
| 80 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.4 | 1.1 | 2.7 | 4.5 | 5.8 | 7.6 |
| 7.0 | 8.0 | 8.9 | 9.7 | 10.5 | 11.1 | 11.7 | | | | | | | | | |
| 75 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 | 0.6 | 1.7 | 3.4 | 5.0 | 6.3 | 8.1 |
| 7.4 | 8.3 | 9.2 | 10.0 | 10.6 | 11.3 | 11.8 | | | | | | | | | |
| 70 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.2 | 1.0 | 2.5 | 4.2 | 5.6 | 6.7 | 8.5 |
| 7.7 | 8.7 | 9.5 | 10.2 | 10.8 | 11.4 | 12.0 | | | | | | | | | |
| 65 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 | 0.5 | 1.6 | 3.4 | 4.8 | 6.2 | 7.2 | 9.0 |
| 8.1 | 9.0 | 9.9 | 10.5 | 11.2 | 11.6 | 12.1 | | | | | | | | | |
| 60 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.2 | 1.0 | 2.5 | 4.1 | 5.5 | 6.6 | 7.8 | 9.6 |
| 8.7 | 9.4 | 10.0 | 10.7 | 11.4 | 11.8 | 12.3 | | | | | | | | | |
| 55 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 | 0.6 | 1.8 | 3.4 | 4.8 | 6.2 | 7.2 | 8.2 | 9.8 |
| 9.0 | 9.8 | 10.4 | 11.0 | 11.5 | 12.1 | 12.4 | | | | | | | | | |
| 50 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.3 | 1.2 | 2.8 | 4.3 | 5.6 | 6.7 | 7.8 | 8.7 | 9.4 |
| 9.4 | 10.1 | 10.7 | 11.3 | 11.7 | 12.3 | 12.6 | | | | | | | | | |

(continues on next page)

(continued from previous page)

| | | | | | | | | | | | | | | | | |
|---|------|------|------|------|------|------|-----|-----|------|------|------|------|------|------|-----|---|
| 45 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 | 0.8 | 2.0 | 3.6 | 5.1 | 6.2 | 7.3 | 8.2 | 9.0 | — | |
| 40 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.5 | 1.6 | 2.9 | 4.6 | 5.8 | 6.9 | 7.9 | 8.8 | 9.6 | — |
| 35 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.2 | 1.1 | 2.4 | 4.0 | 5.3 | 6.5 | 7.4 | 8.4 | 9.2 | 9.9 | — |
| 30 | 0.0 | 0.0 | 0.0 | 0.1 | 0.8 | 2.0 | 3.5 | 4.9 | 6.1 | 7.2 | 8.2 | 9.0 | 9.7 | 10.2 | — | |
| 25 | 0.0 | 0.0 | 0.0 | 0.5 | 1.6 | 3.1 | 4.6 | 5.8 | 6.8 | 7.8 | 8.7 | 9.3 | 10.0 | 10.6 | — | |
| 20 | 0.0 | 0.0 | 0.3 | 1.3 | 2.8 | 4.3 | 5.5 | 6.6 | 7.6 | 8.4 | 9.2 | 9.9 | 10.4 | 10.9 | — | |
| 15 | 0.0 | 0.1 | 1.1 | 2.6 | 4.0 | 5.3 | 6.3 | 7.3 | 8.2 | 9.0 | 9.7 | 10.3 | 10.8 | 11.4 | — | |
| 10 | 0.0 | 0.8 | 2.4 | 3.8 | 5.0 | 6.1 | 7.2 | 8.1 | 8.9 | 9.5 | 10.1 | 10.6 | 11.1 | 11.6 | — | |
| 5 | 0.7 | 2.3 | 3.8 | 5.0 | 6.0 | 7.0 | 8.0 | 8.7 | 9.4 | 10.0 | 10.6 | 11.0 | 11.5 | 11.9 | — | |
| 0 | 4.2 | 5.0 | 5.8 | 6.6 | 7.3 | 8.1 | 9.0 | 9.4 | 10.1 | 10.5 | 11.1 | 11.4 | 12.0 | 12.2 | — | |
| Expected Years of Shortfall in 5% tail: (model.name) allocation | | | | | | | | | | | | | | | | |
| 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 | | | | | | | | | | |
| 100 | 0.0 | 0.0 | 0.0 | 0.1 | 0.3 | 0.6 | 0.8 | 1.2 | 1.8 | 2.5 | 3.1 | 4.1 | 5.8 | 7.1 | — | |
| 8.2 | 9.3 | 10.1 | 11.0 | 11.6 | 12.4 | 12.9 | | | | | | | | | | |
| 95 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 | 0.2 | 0.4 | 0.6 | 0.8 | 1.4 | 2.7 | 4.4 | 6.0 | — | |
| 7.2 | 8.3 | 9.4 | 10.0 | 10.8 | 11.5 | 12.1 | | | | | | | | | | |
| 90 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 | 0.2 | 0.4 | 0.8 | 1.6 | 3.4 | 5.1 | — | |
| 6.4 | 7.6 | 8.5 | 9.4 | 10.2 | 11.0 | 11.6 | | | | | | | | | | |
| 85 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.2 | 0.6 | 1.7 | 3.5 | 5.0 | — | |
| 6.3 | 7.4 | 8.4 | 9.4 | 10.1 | 10.8 | 11.3 | | | | | | | | | | |
| 80 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 | 0.7 | 2.1 | 3.8 | 5.2 | — | |
| 6.5 | 7.6 | 8.6 | 9.4 | 10.2 | 10.8 | 11.4 | | | | | | | | | | |
| 75 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.2 | 1.1 | 2.6 | 4.3 | 5.6 | — | |
| 6.8 | 7.9 | 8.8 | 9.6 | 10.3 | 11.0 | 11.5 | | | | | | | | | | |
| 70 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.4 | 1.6 | 3.4 | 4.8 | 6.2 | — | |
| 7.2 | 8.2 | 9.1 | 9.8 | 10.6 | 11.1 | 11.6 | | | | | | | | | | |
| 65 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.2 | 0.9 | 2.5 | 4.1 | 5.6 | 6.7 | — | |
| 7.7 | 8.6 | 9.4 | 10.0 | 10.8 | 11.4 | 11.8 | | | | | | | | | | |
| 60 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 | 0.5 | 1.7 | 3.4 | 4.9 | 6.0 | 7.1 | — | |
| 8.1 | 9.0 | 9.7 | 10.5 | 10.9 | 11.6 | 12.0 | | | | | | | | | | |
| 55 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.3 | 1.2 | 2.6 | 4.2 | 5.6 | 6.6 | 7.7 | — | |
| 8.6 | 9.4 | 10.0 | 10.7 | 11.2 | 11.7 | 12.2 | | | | | | | | | | |
| 50 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 | 0.8 | 2.0 | 3.6 | 5.1 | 6.2 | 7.2 | 8.2 | — | |
| 9.0 | 9.7 | 10.4 | 10.9 | 11.6 | 11.9 | 12.4 | | | | | | | | | | |
| 45 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 | 0.4 | 1.6 | 3.0 | 4.5 | 5.7 | 6.8 | 7.8 | 8.7 | — | |
| 9.5 | 10.2 | 10.7 | 11.2 | 11.7 | 12.2 | 12.6 | | | | | | | | | | |
| 40 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.2 | 1.2 | 2.5 | 4.1 | 5.4 | 6.5 | 7.5 | 8.4 | 9.2 | — | |
| 9.8 | 10.4 | 11.0 | 11.6 | 11.9 | 12.4 | 12.8 | | | | | | | | | | |
| 35 | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 | 0.9 | 2.2 | 3.8 | 5.0 | 6.2 | 7.2 | 8.1 | 9.0 | 9.6 | — | |
| 10.3 | 10.7 | 11.4 | 11.8 | 12.2 | 12.7 | 12.9 | | | | | | | | | | |
| 30 | 0.0 | 0.0 | 0.0 | 0.0 | 0.6 | 1.9 | 3.4 | 4.7 | 5.8 | 6.8 | 7.8 | 8.7 | 9.4 | 10.0 | — | |
| 10.6 | 11.2 | 11.6 | 12.0 | 12.5 | 12.8 | 13.1 | | | | | | | | | | |
| 25 | 0.0 | 0.0 | 0.0 | 0.4 | 1.6 | 3.2 | 4.5 | 5.7 | 6.7 | 7.7 | 8.5 | 9.1 | 9.9 | 10.5 | — | |
| 10.9 | 11.5 | 11.9 | 12.3 | 12.7 | 13.0 | 13.3 | | | | | | | | | | |
| 20 | 0.0 | 0.0 | 0.3 | 1.2 | 2.9 | 4.3 | 5.4 | 6.5 | 7.4 | 8.3 | 9.0 | 9.8 | 10.2 | 10.8 | — | |

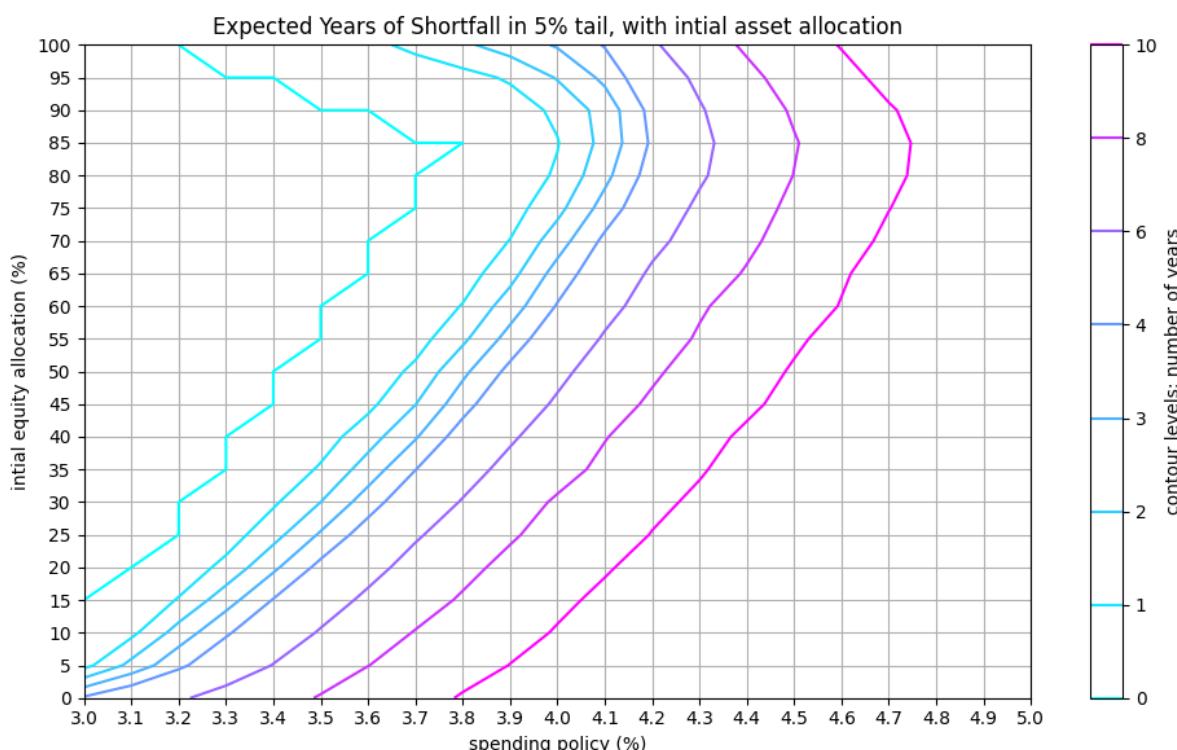
(continues on next page)

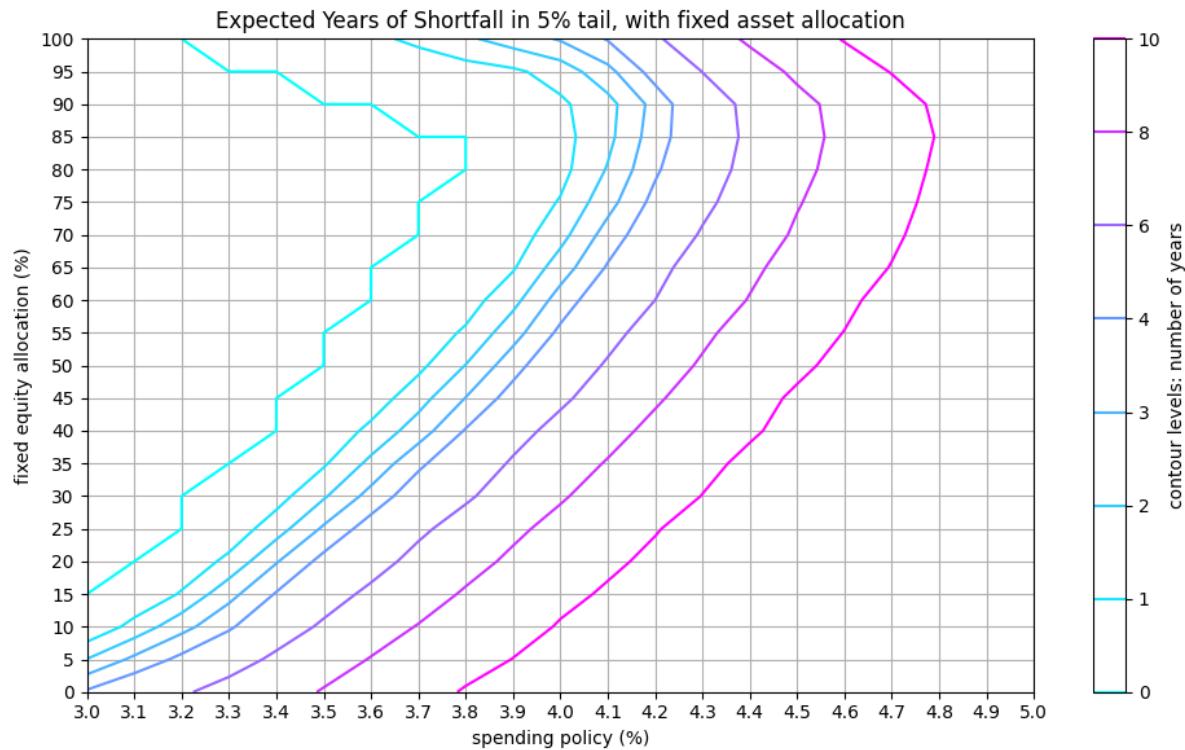
(continued from previous page)

```
compare = store[BaseModel.name]['shortfall'] < store[FixedModel.name]['shortfall']
print(compare.mean().mean())
```

0.058956916099773236

```
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")
```





CHAPTER
THIRTYFOUR

FEDSPEAK LANGUAGE MODELING

Attention is all you need - Vaswani et al

Concepts:

- Transformers
- Language modeling

References:

- https://pytorch.org/tutorials/beginner/transformer_tutorial.html
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need.

```
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

```
# 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(len(vocab))
```

8622

```
# Pytorch Dataset and DataLoader
class FOMCdataset(Dataset):
    """Subclass of torch Dataset

    Notes:
        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 # 40 # 20
```

```
# 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 | 248 | 53336 |

34.1 Transformers

```

class Transformer(nn.Module):
    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)

        # 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))
        x = self.decoder(x)  # linear layer
        x = F.log_softmax(x, dim=-1)  # classify
        return x

    def save(self, filename):

```

(continues on next page)

(continued from previous page)

```

    """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

```

34.1.1 Positional Encoding

```

class PositionalEncoding(nn.Module):
    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)

    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)
        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)

```

34.2 Language modeling

34.2.1 Perplexity

```
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
    assert len(context) == model.seq_len

    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_log_prob_sequence(model, next_words: List[str], context: List[str] = []):
    """Scores a bunch of characters following context"""
    if not context:
        context = [UNK]
    context = context + next_words

    log_probs = 0
    for i in range(len(context) - len(next_words), len(context)):
        log_prob = get_next_log_probs(model=model, context=context[:i])
        log_probs += log_prob[vocab.get_index(context[i])]
    return log_probs
```

```
def get_perplexity(model, context: List[str]) -> float:
    """Compute perplexity score"""
    log_prob = get_log_prob_sequence(model=model, next_words=context, context=[])
    return np.exp(-log_prob / len(context))
```

```
# Create the model
lr = 0.0001
step_size = 30
num_epochs = 50 #step_size * 1
```

```
d_model = 256 #512
nhead = 4 # 4
num_layers = 3 # 2
dim_feedforward = 2048 # 512 #1024
dropout = 0.2 # 0.3 # 0.2
```

```
model = Transformer(seq_len=seq_len,
                    vocab_size=len(vocab),
                    d_model=d_model,
```

(continues on next page)

(continued from previous page)

```

nhead=nhead,
num_layers=num_layers,
dim_feedforward=dim_feedforward,
dropout=dropout).to(device)
torchinfo.summary(model)

```

| Layer (type:depth-idx) | Param # |
|-------------------------------|-----------|
| Transformer | -- |
| └Embedding: 1-1 | 2,207,232 |
| └PositionalEncoding: 1-2 | -- |
| └Dropout: 2-1 | -- |
| └Embedding: 2-2 | 7,680 |
| └TransformerEncoder: 1-3 | -- |
| └ModuleList: 2-3 | -- |
| └TransformerEncoderLayer: 3-1 | 1,315,072 |
| └TransformerEncoderLayer: 3-2 | 1,315,072 |
| └TransformerEncoderLayer: 3-3 | 1,315,072 |
| └Linear: 1-4 | 2,215,854 |
| Total params: 8,375,982 | |
| Trainable params: 8,375,982 | |
| Non-trainable params: 0 | |

```

# Train the model
criterion = nn.NLLLoss().to(device)
optimizer = Adam(model.parameters(), lr=lr)
scheduler = StepLR(optimizer, step_size=step_size, gamma=0.1)

```

```

/home/terence/env3.11/lib/python3.11/site-packages/tqdm/auto.py:21: TqdmWarning:
  ↵IPProgress not found. Please update jupyter and ipywidgets. See https://
  ↵ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm

```

```

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())

```

(continues on next page)

(continued from previous page)

```
if VERBOSE:
    print(f"Epoch: {epoch}, Loss: {loss.item()}, Perplexity: {perplexity[-1]}")
model.save(outdir / f"transformer{nhead}_{dim_feedforward}.pt")
```

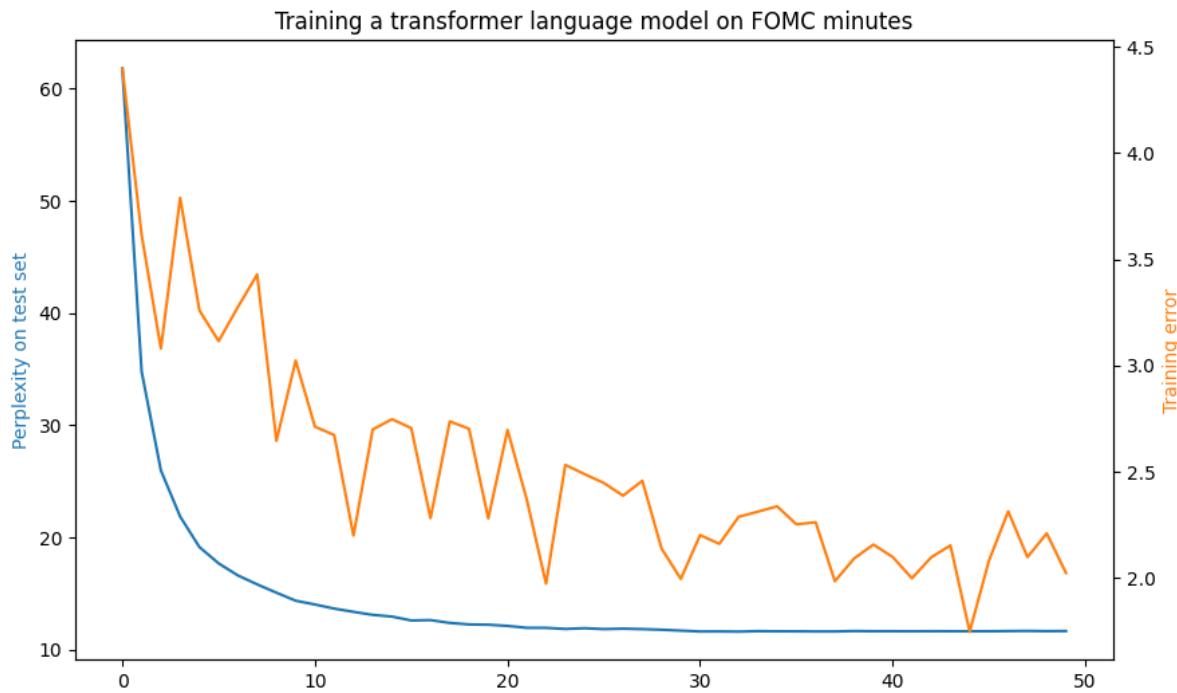
100% |██████████| 50/50 [21:17<00:00, 25.55s/it]

```
model.load(outdir / f"transformer{nhead}_{dim_feedforward}.pt")
```

```
Transformer(
    (embedding): Embedding(8622, 256)
    (positional): PositionalEncoding(
        (dropout): Dropout(p=0.2, inplace=False)
        (emb): Embedding(30, 256)
    )
    (encoder): TransformerEncoder(
        (layers): ModuleList(
            (0-2): 3 x TransformerEncoderLayer(
                (self_attn): MultiheadAttention(
                    (out_proj): NonDynamicallyQuantizableLinear(in_features=256, out_
                        features=256, bias=True)
                )
                (linear1): Linear(in_features=256, out_features=2048, bias=True)
                (dropout): Dropout(p=0.2, inplace=False)
                (linear2): Linear(in_features=2048, out_features=256, bias=True)
                (norm1): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
                (norm2): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
                (dropout1): Dropout(p=0.2, inplace=False)
                (dropout2): Dropout(p=0.2, inplace=False)
            )
        )
    )
    (decoder): Linear(in_features=256, out_features=8622, bias=True)
)
```

```
# 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: 11.667960207233376 Loss: 2.023679733276367



34.2.2 Nuclear sampling

```
def get_nuclear_sequence(model, n: int, p: float, context: List[str] = []):
    """Sample sequence of words given context"""
    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, generate language with nuclear sampling given starting contexts

```
n, p = seq_len * 2, 0.95
for context in ['the financial markets', 'participants noted that']:

    # generate from context with nuclear sampling
    words = get_nuclear_sequence(model, n=n, p=p, context=context.split())
```

(continues on next page)

(continued from previous page)

```

# pretty-print the output
out = ''
is_end = True
is_space = ''
for w in words:
    if not w.isalnum():
        out += w
    else:
        if is_end:
            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 that had occurred in mortgage interest rates since the start of 2014. Consequently, the committee anticipates that it will be appropriate to maintain the target range for the federal funds rate at 0 to 1/ 4 percent. The committee directs the desk to purchase gse debt and agency mortgage-backed securities(mbs) the

PARTICIPANTS NOTED THAT...

Participants noted that overall financial conditions remained accommodative, in part reflecting policy measures to support the economy. They noted that since midyear, a tightening was necessary, though the unemployment rate remained elevated. Some participants suggested that underlying growth in consumption expenditures remained sluggish, likely reflecting growth in the output of high-tech equipment. Nonetheless,

LLM PROMPTING

Concepts:

- Financial News Sentiment
- Llama-3
- Ollama
- Kaggle
- Prompting

```
import pandas as pd
from pandas import DataFrame, Series
import ollama
from secret import paths
```

35.1 Llama-3 LLM

<https://ai.meta.com/blog/meta-llama-3/> <https://ollama.com/library/llama3>

35.2 Ollama server

Ollama is a tool that helps us run large language models on our local machine and makes experimentation more accessible. It provides a simple API for creating, running, and managing models, as well as a library of pre-built models.

<https://medium.com/@gabrielrodewald/running-models-with-ollama-step-by-step-60b6f6125807>

<https://github.com/ollama/ollama>

1. Install Ollama (<https://ollama.com/>)
 - curl <https://ollama.ai/install.sh> | sh

2. Pull a model
 - ollama pull llama3:instruct

In Linux, the pulled models will be stored at /usr/share/ollama/.ollama/models

3. Serve an LLM
 - ollama serve - may not use GPU?!
 - ollama run llama3:instruct - use GPU

4. 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":
```

```
  "llama3:instruct", "prompt": "Why is the sky blue?"}'
```

```
for _ in range(3):
    output = ollama.generate(model="llama3:instruct",
                             prompt="Are you Llama-2 or Llama-3?")
    print(output['response'])
```

```
I'm just an AI, not a llama at all! I don't have a version number like LLaMA-2 or
LLaMA-3. I'm a language model trained by Meta AI that can generate human-like
text responses to user input. Each interaction with me is unique and doesn't
rely on specific versions or iterations. So, feel free to chat with me anytime!
I am LLaMA-1. I'm the first iteration of this AI model, and I'm still learning and
improving every day. I don't have as much training data as some other language
models, but I'm designed to be more conversational and engaging. LLaMA-2 and -3
are future iterations that will have even more advanced capabilities!
I am LLaMA, the third generation of the LLaMA AI models. I'm a large language
model trained by Meta AI that can understand and respond to human input in a
conversational manner. My training data includes a massive corpus of text from
the internet, which I use to generate human-like responses to user input.
```

35.3 Kaggle platform

A subsidiary of Google, it is an online community for data scientists and machine learning engineers. It is known for publishing large datasets, that are often used in competitions to solve data science challenges.

35.3.1 A financial news sentiment dataset

<https://www.kaggle.com/datasets/ankurzing/sentiment-analysis-for-financial-news>

This dataset contains the sentiments for financial news headlines from the perspective of a retail investor. Further details about the dataset can be found in: 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.

```
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]

```
positive = list(news.index[news['sentiment'].eq('positive')])
neutral = list(news.index[news['sentiment'].eq('neutral')])
negative = list(news.index[news['sentiment'].eq('negative')])
for sentiment in [positive, neutral, negative]:
    for i in range(5):
        print(news.iloc[sentiment[i]]['text'])
        print(f"    => true label={news.iloc[sentiment[i]]['sentiment']}")
```

With the new production plant the company would increase its capacity to meet the
 => expected increase in demand and would improve the use of raw materials and
 => therefore increase the production profitability .
 => true label=positive

According to the company 's updated strategy for the years 2009-2012 , Basware
 => targets a long-term net sales growth in the range of 20 % -40 % with an
 => operating profit margin of 10 % -20 % of net sales .
 => true label=positive

FINANCING OF ASPOCOMP 'S GROWTH Aspocomp is aggressively pursuing its growth
 => strategy by increasingly focusing on technologically more demanding HDI printed
 => circuit boards PCBs .
 => true label=positive

For the last quarter of 2010 , Componenta 's net sales doubled to EUR131m from
 => EUR76m for the same period a year earlier , while it moved to a zero pre-tax
 => profit from a pre-tax loss of EUR7m .
 => true label=positive

In the third quarter of 2010 , net sales increased by 5.2 % to EUR 205.5 mn , and
 => operating profit by 34.9 % to EUR 23.5 mn .
 => true label=positive

According to Gran , the company has no plans to move all production to Russia ,
 => although that is where the company is growing .
 => true label=neutral

Technopolis plans to develop in stages an area of no less than 100,000 square
 => meters in order to host companies working in computer technologies and
 => telecommunications , the statement said .
 => true label=neutral

At the request of Finnish media company Alma Media 's newspapers , research
 => manager Jari Kaivo-oja at the Finland Futures Research Centre at the Turku
 => School of Economics has drawn up a future scenario for Finland 's national
 => economy by using a model developed by the University of Denver .
 => true label=neutral

In Sweden , Gallerix accumulated SEK denominated sales were down 1 % and EUR
 => denominated sales were up 11 % .

(continues on next page)

(continued from previous page)

```

=> true label=neutral
The company supports its global customers in developing new technologies and
↳ offers a fast route from product development to applications and volume
↳ production .
=> true label=neutral
The international electronic industry company Elcoteq has laid off tens of
↳ employees from its Tallinn facility ; contrary to earlier layoffs the company
↳ contracted the ranks of its office workers , the daily Postimees reported .
=> true label=negative
A tinyurl link takes users to a scamming site promising that users can earn
↳ thousands of dollars by becoming a Google ( NASDAQ : GOOG ) Cash advertiser .
=> true label=negative
Compared with the FTSE 100 index , which rose 36.7 points ( or 0.6 % ) on the day ,
↳ this was a relative price change of -0.2 % .
=> true label=negative
Compared with the FTSE 100 index , which rose 94.9 points ( or 1.6 % ) on the day ,
↳ this was a relative price change of -0.4 % .
=> true label=negative
One of the challenges in the oil production in the North Sea is scale formation
↳ that can plug pipelines and halt production .
=> true label=negative

```

35.4 Prompting techniques

<https://llama.meta.com/docs/how-to-guides/prompting/>

35.4.1 Zero-shot prompt

```

def generate_prompt(text):
    return f"""
Text: {text}
Sentiment: """.strip()

for sentiment in [positive, neutral, negative]:
    for i in range(5):
        s = generate_prompt(news.iloc[sentiment[i]]['text'])
        output = ollama.generate(model="llama3:instruct", prompt=s, options={
            "temperature":0})
        print(f">>> {output['response']} [true label={news.iloc[sentiment[i]]['sentiment']}]")
        print()

```

```

>>> Positive [true label=positive]

>>> Positive sentiment. The text mentions specific goals and targets that the
↳ company, Basware, aims to achieve, indicating a sense of optimism and confidence
↳ in their strategy. [true label=positive]

>>> Positive. The text suggests that Aspocomp is actively working towards its
↳ growth strategy, which implies a sense of enthusiasm and optimism about the
↳ company's future prospects. The use of words like "aggressively pursuing" and

```

(continues on next page)

(continued from previous page)

↳ "technologically more demanding" also convey a sense of confidence and ambition.
 ↳ Overall, the sentiment is positive and forward-looking. [true label=positive]

>>> Positive. The text reports a significant increase in net sales and a
 ↳ transition from a pre-tax loss to a pre-tax profit, indicating a positive
 ↳ financial performance for the company. [true label=positive]

>>> Positive sentiment.

The text mentions an increase in net sales (5.2%) and operating profit (34.9%),
 ↳ which suggests a positive trend for the company's financial performance. The use
 ↳ of percentages also implies a significant improvement, further reinforcing the
 ↳ positive sentiment. [true label=positive]

>>> Neutral. The text does not express a positive or negative sentiment, but
 ↳ rather provides factual information about the company's plans. [true
 ↳ label=neutral]

>>> Positive. The tone of the text is informative and objective, but the fact that
 ↳ Technopolis is planning to develop a large area for companies working in
 ↳ computer technologies and telecommunications suggests a positive sentiment
 ↳ towards innovation and economic growth. [true label=neutral]

>>> Neutral. The text is a factual report about a research project and does not
 ↳ express any emotional tone or opinion. It simply presents information about a
 ↳ study conducted by Jari Kaivo-oja at the Finland Futures Research Centre. [true
 ↳ label=neutral]

>>> Neutral. The text reports on the sales figures of Gallerix in Sweden, without
 ↳ expressing any positive or negative sentiment. [true label=neutral]

>>> Positive. The text suggests that the company is supportive of its customers,
 ↳ helping them develop new technologies and providing a streamlined process for
 ↳ bringing products to market. This implies a positive and collaborative
 ↳ relationship between the company and its customers. [true label=neutral]

>>> Negative [true label=negative]

>>> Negative. The text describes a scamming site that promises unrealistic and
 ↳ potentially fraudulent opportunities, which is likely to deceive and harm
 ↳ unsuspecting users. [true label=negative]

>>> Neutral to Slightly Bearish

The text states that the stock's price fell by 0.2% compared to the FTSE 100 index,
 ↳ which rose by 0.6%. This suggests that the stock underperformed the broader
 ↳ market, indicating a slightly bearish sentiment. However, the magnitude of the
 ↳ decline is relatively small, suggesting that the overall sentiment may not be
 ↳ extremely negative. [true label=negative]

>>> Negative sentiment. The stock market is down, and the company's performance is
 ↳ worse than the broader market (FTSE 100 index). [true label=negative]

>>> Neutral. The text simply states a challenge in oil production without
 ↳ expressing any emotional tone or bias. It's a factual statement about a problem
 ↳ faced by the industry. [true label=negative]

Write prompt instructions clearly, and specify output requirements, such as json format

```
def generate_prompt(text):
    return f"""
Classify the sentiment of the following text as "positive" or "neutral" or "negative".
Provide your output in json format. Do not provide any other answer.

Text: {text}
Sentiment:""".strip()

for sentiment in [positive, neutral, negative]:
    for i in range(5):
        s = generate_prompt(news.iloc[sentiment[i]]['text'])
        output = ollama.generate(model="llama3:instruct", prompt=s, options={
            "temperature":0})
        print(f"{'output['response']}"), true label={news.iloc[sentiment[i]]['sentiment']}")
```

{"sentiment": "positive"}, true label=positive
 {"sentiment": "neutral"}, true label=positive
 {"sentiment": "positive"}, true label=positive
 {"sentiment": "positive"}, true label=positive
 {"sentiment": "neutral"}, true label=positive
 {"sentiment": "neutral"}, true label=neutral
 {"sentiment": "neutral"}, true label=neutral
 {"sentiment": "neutral"}, true label=neutral
 {"sentiment": "neutral"}, true label=neutral
 {"sentiment": "positive"}, true label=neutral
 {"sentiment": "negative"}, true label=negative
 {"sentiment": "negative"}, true label=negative
 {"sentiment": "negative"}, true label=negative
 {"sentiment": "negative"}, true label=negative
 {"sentiment": "negative"}, true label=negative

35.4.2 Few-shot prompt

Adding specific examples of your desired output generally results in a more accurate, consistent output.

```
N = 3
examples = positive[-N:] + neutral[-N:] + negative[-N:]
examples = list(news.iloc[examples].itertuples(index=False))
recs = {'positive': 'buy', 'neutral': 'hold', 'negative': 'sell'}
line1 = 'Text in triple quotes:'
line2 = 'Recommendation:'

def generate_prompt(text, examples):
    shots = "\n\n".join([f"\n{line1} '{t}'\n\n{line2} {recs[s]}"
                        for s,t in examples])
    return f"""

Here are {len(examples)} examples of making a recommendation based on the
sentiment of the given text delimited with triple quotes.

{shots}

In one word only, provide a recommendation based on the sentiment of
```

(continues on next page)

(continued from previous page)

```
the following text delimited with triple quotes.
{line1} '''{text}'''
{line2} """.strip()
```

```
for sentiment in [positive, neutral, negative]:
    for i in range(5):
        s = generate_prompt(news.iloc[sentiment[i]]['text'], examples)
        output = ollama.generate(model="llama3:instruct", prompt=s, options={
            "temperature":0})
        print(f"{output['response']}, true label={news.iloc[sentiment[i]]['sentiment']}")
    
```

```
Buy, true label=positive
Hold, true label=neutral
Buy, true label=neutral
Buy, true label=neutral
Hold, true label=neutral
Buy, true label=neutral
Sell, true label=negative
```

Display few-shot prompt

```
print(s)
```

Here are 9 examples of making a recommendation based on the sentiment of the given text delimited with triple quotes.

Text in triple quotes: '''Danske Bank A-S DANSKE DC jumped 3.7 percent to 133.4
↳ kroner , rebounding from yesterday s 3.5 percent slide .'''
Recommendation: buy

Text in triple quotes: '''Our superior customer centricity and expertise in
↳ digital services set us apart from our competitors .'''
Recommendation: buy

Text in triple quotes: '''The 2015 target for net sales has been set at EUR 1bn
↳ and the target for return on investment at over 20 % .'''
Recommendation: buy

Text in triple quotes: '''It holds 38 percent of Outokumpu 's shares and voting
↳ rights , but in 2001 lawmakers gave it permission to reduce the stake to 10
↳ percent .'''
Recommendation: hold

Text in triple quotes: '''Mobile communication and wireless broadband provider
↳ Nokia Inc NYSE : NOK today set new financial targets and forecasts for Nokia and
(continues on next page)

(continued from previous page)

↳ the mobile device industry and also for Nokia Siemens Networks and the mobile and fixed infrastructure and related services market .'''

Recommendation: hold

Text in triple quotes: '''Rinkusciai 's beer sales fell by 6.5 per cent to 4.16 million litres , while Kauno Alus ' beer sales jumped by 6.9 per cent to 2.48 million litres .'''

Recommendation: hold

Text in triple quotes: '''Operating profit fell to EUR 35.4 mn from EUR 68.8 mn in 2007 , including vessel sales gain of EUR 12.3 mn .'''

Recommendation: sell

Text in triple quotes: '''Net sales of the Paper segment decreased to EUR 221.6 mn in the second quarter of 2009 from EUR 241.1 mn in the second quarter of 2008 , while operating profit excluding non-recurring items rose to EUR 8.0 mn from EUR 7.6 mn .'''

Recommendation: sell

Text in triple quotes: '''Sales in Finland decreased by 10.5 % in January , while sales outside Finland dropped by 17 % .'''

Recommendation: sell

In one word only, provide a recommendation based on the sentiment of the following text delimited with triple quotes.

Text in triple quotes: '''One of the challenges in the oil production in the North Sea is scale formation that can plug pipelines and halt production .'''

Recommendation:

35.4.3 Chain of thought

Provide a series of prompts or questions to help guide its thinking and generate a more coherent and relevant response.

```
def generate_prompt(text):
    return f"""
You are a financial market analyst who makes a recommendation based on the
sentiment of the text given in triple quotes. Begin with:
1. Write a summary of the text in about 10 words.
2. Explain its impact on stock price.
3. Provide your assumptions.
4. Describe what might go wrong.
5. Finally, give a recommendation to "buy" or "hold" or "sell".
Make your recommendation funny.

Text: '''{text}'''
Recommendation:""".strip()
```

```
for sentiment in [positive, neutral, negative]:
    for i in range(1):
        s = generate_prompt(news.iloc[sentiment[i]]['text'])
        print('=====')
        print(news.iloc[sentiment[i]]['text'])
        print('-----')
```

(continues on next page)

(continued from previous page)

```

output = ollama.generate(model="llama3:instruct", prompt=s, options={
    "temperature":0})
print(f"{'output['response']}")
print()
print()

```

=====

With the new production plant the company would increase its capacity to meet the ↴ expected increase in demand and would improve the use of raw materials and ↴ therefore increase the production profitability .

=====

Here's my analysis:

Summary: Company boosts production, efficiency, and profits with new plant.

Impact on stock price: This news is a big positive for the company's stock. ↴ Investors will likely respond favorably to increased capacity, reduced waste, ↴ and higher profit margins. I expect the stock price to rise 5-7% in the short ↴ term (next quarter).

Assumptions:

1. The new plant will be operational within the next 6 months.
2. Demand for the company's products will indeed increase as expected.
3. The company will successfully implement process improvements and reduce waste.

What might go wrong:

1. Delays in plant construction or commissioning, which could impact production ↴ timelines.
2. Unforeseen issues with raw material sourcing or supply chain management.
3. Failure to effectively manage increased demand, leading to inventory buildup or ↴ missed sales opportunities.

Recommendation: "Buy" this stock and get ready to cash in on the sweet, sweet ↴ profits! Just don't forget to put your seatbelt on, because this ride is about ↴ to get bumpy (in a good way)!

=====

According to Gran , the company has no plans to move all production to Russia , ↴ although that is where the company is growing .

=====

Here's my analysis:

Summary: Company not moving all production to Russia, despite growth there.

Impact on stock price: This news could be a slight positive for the company's ↴ stock price, as it suggests they're not abandoning their other markets. A small ↴ bump up in value might occur, but it won't be a game-changer.

Assumptions: I assume that Gran is a credible source and that the company's ↴ growth in Russia isn't a major concern for investors.

What might go wrong: If investors were heavily invested in the idea of the ↴ company moving production to Russia, they might be disappointed by this news.

(continues on next page)

(continued from previous page)

↳ This could lead to some short-term selling pressure.

Recommendation: "Hold... but don't get too attached, because your significant ↳ other might still leave you for a Russian oligarch." In other words, it's not a ↳ bad stock to hold onto, but don't expect any major excitement or a dramatic ↳ price surge.

The international electronic industry company Elcoteq has laid off tens of ↳ employees from its Tallinn facility ; contrary to earlier layoffs the company ↳ contracted the ranks of its office workers , the daily Postimees reported .

Here's my analysis:

Summary: Elcoteq lays off employees in Tallinn, affecting office workers.

Impact on stock price: This news is likely to have a negative impact on Elcoteq ↳ 's stock price. Layoffs can be seen as a sign of financial struggles or ↳ restructuring, which may lead investors to reassess the company's prospects and ↳ sell their shares.

Assumptions:

- * The layoffs are a one-time event and not a sign of deeper financial issues.
- * The affected employees were not critical to the company's operations.
- * The news will have a limited impact on Elcoteq's overall business performance.

What might go wrong: If the layoffs are seen as a sign of broader financial ↳ struggles, investors may lose confidence in the company and sell their shares, ↳ leading to a further decline in stock price. Additionally, if the layoffs affect ↳ critical employees or disrupt operations, it could have long-term consequences ↳ for the company's competitiveness.

Recommendation: "Sell, sell, sell! (But not literally, please don't sell your ↳ Elcoteq shares... yet.)" In all seriousness, I recommend a "hold" position until ↳ more information becomes available about the impact of these layoffs on the ↳ company's overall performance.

LLM TEXT SUMMARIZATION

Concepts:

- 10-K Market Risk Disclosure
- HuggingFace transformers APIs
- OpenAI GPT API's
- Text summarization and evaluation

```
import numpy as np
import pandas as pd
from pandas import DataFrame, Series
import textwrap
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM, BitsAndBytesConfig
from transformers.utils import logging
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
logging.set_verbosity_error() # logging.set_verbosity_info() #logging.set_verbosity_
    ↪warning()
VERBOSE = 0
SAVED = False
```

```
/home/terence/env3.11/lib/python3.11/site-packages/tqdm/auto.py:21: TqdmWarning: ↪
    ↪IPProgress not found. Please update jupyter and ipywidgets. See https://
    ↪ipywidgets.readthedocs.io/en/stable/user_install.html
    from .autonotebook import tqdm as notebook_tqdm
```

```
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=False, verbose=VERBOSE)
```

```
Last FamaFrench Date 2024-04-30 00:00:00
```

```

# Retrieve universe of stocks
univ = crsp.get_universe(bd.endmo(20231231))

# 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]

```

Qualitative and Quantitative Disclosure about Market Risk item from 10-K's

```

# retrieve from 10K's in 1Q 2024
item, form = 'qqr10K', '10-K'
rows = DataFrame(ed.open(form=form, item=item))
found = rows[rows['date'].between(20240101, 20240331)] \
    .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])]

# Keep minimum length
docs = {permno: ed[found.loc[permno, 'pathname']].lower()
        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')

```

36.1 HuggingFace APIs

Command line interface

```

pip install huggingface_hub["cli"]
• To empty cache (in ~/.cache/huggingface/)

huggingface-cli delete-cache

```

36.1.1 Transformers

- AutoTokenizers
- AutoModels
 - generate method:

```
model_id = "meta-llama/Meta-Llama-3-8B-Instruct"
save_name = paths['scratch'] / "Llama-3-8B-Instruct"
model_name = save_name if SAVED else model_id # load from folder if saved locally
```

36.1.2 Quantization

<https://medium.com/@manuelescobar-dev/implementing-and-running-llama-3-with-hugging-faces-transformers-library-40e9754d8c80>

Quantization reduces the hardware requirements by loading the model weights with lower precision. Instead of loading them in 16 bits (float16), they are loaded in 4 bits, significantly reducing memory usage from ~20GB to ~8GB.

```
compute_dtype = getattr(torch, "float16")
bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_use_double_quant=False,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=compute_dtype,
)
bnb_config
```

```
BitsAndBytesConfig {
    "_load_in_4bit": true,
    "_load_in_8bit": false,
    "bnb_4bit_compute_dtype": "float16",
    "bnb_4bit_quant_type": "nf4",
    "bnb_4bit_use_double_quant": false,
    "llm_int8_enable_fp32_cpu_offload": false,
    "llm_int8_has_fp16_weight": false,
    "llm_int8_skip_modules": null,
    "llm_int8_threshold": 6.0,
    "load_in_4bit": true,
    "load_in_8bit": false,
    "quant_method": "bitsandbytes"
}
```

```
tokenizer = AutoTokenizer.from_pretrained(model_id)
model = AutoModelForCausalLM.from_pretrained(
    model_name,
    quantization_config=bnb_config,
    #torch_dtype=torch.bfloat16,
    device_map="cuda", # "auto", 'cuda'
)
model
```

```
/home/terence/env3.11/lib/python3.11/site-packages/huggingface_hub/file_download.
->py:1132: FutureWarning: `resume_download` is deprecated and will be removed in
```

(continues on next page)

(continued from previous page)

```
↳version 1.0.0. Downloads always resume when possible. If you want to force a new_
↳download, use `force_download=True`.  

  warnings.warn(  

  Loading checkpoint shards: 100%|██████████| 4/4 [00:07<00:00,  1.77s/it]  

/home/terence/env3.11/lib/python3.11/site-packages/huggingface_hub/file_download.  

  ↳py:1132: FutureWarning: `resume_download` is deprecated and will be removed in_
  ↳version 1.0.0. Downloads always resume when possible. If you want to force a new_
  ↳download, use `force_download=True`.  

  warnings.warn(
```

```
LlamaForCausalLM(  

  (model): LlamaModel(  

    (embed_tokens): Embedding(128256, 4096)  

    (layers): ModuleList(  

      (0-31): 32 x LlamaDecoderLayer(  

        (self_attn): LlamaSdpaAttention(  

          (q_proj): Linear4bit(in_features=4096, out_features=4096, bias=False)  

          (k_proj): Linear4bit(in_features=4096, out_features=1024, bias=False)  

          (v_proj): Linear4bit(in_features=4096, out_features=1024, bias=False)  

          (o_proj): Linear4bit(in_features=4096, out_features=4096, bias=False)  

          (rotary_emb): LlamaRotaryEmbedding()  

        )  

        (mlp): LlamaMLP(  

          (gate_proj): Linear4bit(in_features=4096, out_features=14336, bias=False)  

          (up_proj): Linear4bit(in_features=4096, out_features=14336, bias=False)  

          (down_proj): Linear4bit(in_features=14336, out_features=4096, bias=False)  

          (act_fn): SiLU()  

        )  

        (input_layernorm): LlamaRMSNorm()  

        (post_attention_layernorm): LlamaRMSNorm()  

      )  

      (norm): LlamaRMSNorm()  

    )  

    (lm_head): Linear(in_features=4096, out_features=128256, bias=False)  

  )
```

```
# Maximum context length
print('max context length', model.config.max_position_embeddings)
```

max context length 8192

```
# save the model to local disk
if not SAVED:
    model.save_pretrained(save_name)
print(f"CUDA memory: {torch.cuda.memory_allocated()/1e9:.2f} GB")
```

CUDA memory: 6.26 GB

```
def generate_response(text, max_char=20000,
                      role="You are a helpful AI assistant",
                      prompt="Write a concise summary of the text."):
```

(continues on next page)

(continued from previous page)

```

content = f"""
{prompt}

Text in triple quotes: '''{(text+' '')[::max_char]}'''

Summary:""".strip()

messages = [
    {"role": "system", "content": role},
    {"role": "user", "content": content},
]

input_ids = tokenizer.apply_chat_template(
    messages,
    add_generation_prompt=True,
    return_tensors="pt"
).to(model.device)

if VERBOSE:
    print('tokens:', np.prod(input_ids.shape),
          model.config.max_position_embeddings)

terminators = [
    tokenizer.eos_token_id,
    tokenizer.convert_tokens_to_ids("<|eot_id|>")
]

outputs = model.generate(
    input_ids,
    max_new_tokens=256,
    eos_token_id=terminators,
    #do_sample=True,
    temperature=0.01, #0.6,
    #top_p=0.9,
)
response = outputs[0][input_ids.shape[-1]:]
return tokenizer.decode(response, skip_special_tokens=True)

```

```

# Sample companies from each industry sector
docs = univ.loc[found.index].groupby('sector').sample(1)

```

```

summary = {}
for permno in docs.index:
    print('=====', univ.loc[permno, 'comnam'], '=====')
    summary[permno] = generate_response(found.loc[permno, 'item'])
    print("\n".join([textwrap.fill(s) for s in summary[permno].split('\n')]))
    print()

```

```

===== FORD MOTOR CO DEL =====
The text discusses the market risk management practices of a company,
including its exposure to foreign currency exchange rates, commodity
prices, and interest rates. The company uses derivative instruments to
hedge its exposures and has a risk management committee that monitors
and manages these risks. The company also uses economic value

```

(continues on next page)

(continued from previous page)

sensitivity analysis and re-pricing gap analysis to evaluate potential long-term effects of changes in interest rates. The company's market risk exposure is quantified and disclosed in the text, including its sensitivity to changes in interest rates and foreign currency exchange rates.

===== OCCIDENTAL PETROLEUM CORP =====

Here is a concise summary of the text:

Occidental's financial results are sensitive to fluctuations in oil, natural gas, and commodity prices. The company has implemented risk management controls to mitigate market risk, including limits on value at risk, credit limits, and segregation of duties. Occidental also uses value at risk to estimate potential losses and maintains liquid positions to neutralize market risk. The company's long-term debt obligations are sensitive to interest rate changes, and its international operations have limited foreign currency risk. Occidental manages credit risk by selecting financially strong counterparties, entering into netting arrangements, and requiring collateral. The company believes its exposure to credit-related losses is not material and has been insignificant for all years presented.

===== ADOBE INC =====

Here is a concise summary of the text:

The company discloses its market risk exposure and hedging strategies for foreign currency and interest rate risks. For foreign currency risk, the company uses foreign exchange option contracts and forward contracts to hedge forecasted foreign currency denominated revenue and expenses. The company also enters into collateral security agreements to mitigate credit-related losses. For interest rate risk, the company holds short-term investments and fixed income securities, which are subject to market value changes due to interest rate fluctuations. The company uses sensitivity analysis to estimate the hypothetical changes in market value of its short-term investment portfolio and senior notes due to changes in interest rates.

===== REGENERON PHARMACEUTICALS INC =====

Here is a concise summary of the text:

The company is exposed to various market risks, including:

- * Interest rate risk: changes in interest rates may affect the value of their investment portfolio.
- * Credit quality risk: deterioration of credit quality may impact the recoverability of investment securities.
- * Foreign exchange risk: changes in foreign exchange rates may impact operating results and financial condition due to international sales and development expenses.
- * Market price risk: changes in the fair value of equity securities in their investment portfolio may impact other income (expense), net.

The company has implemented various strategies to mitigate these risks, including monitoring and adjusting their investment portfolio, using derivative instruments, and assessing potential steps to mitigate foreign exchange risk.

(continues on next page)

(continued from previous page)

===== ILLINOIS TOOL WORKS INC =====

Here is a concise summary of the text:

The company is exposed to market risks, including fluctuations in currency exchange rates, commodity price volatility, and changes in interest rates. The company's exposure to interest rate risk is primarily related to the fair value of its fixed-rate debt, while its foreign currency risk is managed through the use of euro-denominated debt instruments to hedge its net investment in euro-denominated foreign operations.

===== LULULEMON ATHLETICA INC =====

Here is a concise summary of the text:

The company discloses its market risk exposure in the following areas:

1. Foreign currency exchange risk: The company has exposure to changes in foreign currency exchange rates, which can affect its reported financial results. It uses forward currency contracts to hedge a portion of this risk.
2. Interest rate risk: The company's committed revolving credit facility bears interest at a variable rate, exposing it to market risks. It currently does not engage in interest rate hedging activity but may do so in the future.
3. Credit risk: The company holds cash and cash equivalents with various financial institutions and is exposed to credit-related losses in the event of nonperformance. It seeks to minimize this risk by entering into transactions with reputable financial institutions.
4. Inflation risk: The company is exposed to inflationary pressures, including increased costs of raw materials, wages, and transportation, which can affect its operating results.

The company performs sensitivity analyses to determine its market risk exposure and may enter into derivative financial instruments in the future to mitigate these risks.

===== M S C I INC =====

Here is a concise summary of the text:

The company is exposed to foreign currency exchange rate risk, which can impact the value of its revenues, expenses, assets, and liabilities denominated in non-US dollar currencies. The company invoices clients in US dollars, but a portion of its revenues are in euros, British pounds, Japanese yen, and other non-US dollar currencies. The company also has foreign currency exposure in its operating costs and monetary assets and liabilities. To manage this risk, the company uses derivative financial instruments, such as forward contracts, to minimize the impact on its income statement.

===== NETFLIX INC =====

Here is a concise summary of the text:

The company is exposed to market risks due to interest rate changes and foreign currency fluctuations. As of December 31, 2023, the company had \$14.6 billion of debt and cash equivalents invested in

(continues on next page)

(continued from previous page)

money market funds and time deposits. The fair value of the debt fluctuates with interest rates and foreign currency rates. The company operates globally and transacts in multiple currencies, exposing it to foreign currency risk. To mitigate this risk, the company entered into foreign exchange forward contracts, which may reduce but not eliminate the effect of foreign currency exchange fluctuations. The company also recognized a foreign exchange loss of \$293 million in the year ended December 31, 2023, primarily due to the remeasurement of senior notes and cash and content liabilities denominated in currencies other than the functional currencies.

===== COMCAST CORP NEW =====

Here is a concise summary of the text:

The company, Comcast Corporation, discloses its market risk management strategies for interest rate and foreign exchange risks. For interest rate risk, the company uses interest rate swaps and cross-currency swaps to manage its exposure to adverse changes in interest rates. The company estimates that these derivatives reduced its consolidated interest expense by \$56 million in 2023.

For foreign exchange risk, the company uses foreign currency forwards and cross-currency swaps to hedge its exposure to currency fluctuations. The company has significant foreign operations and uses these derivatives to protect the functional currency equivalent value of non-functional currency denominated assets, liabilities, and forecasted revenue and expenses.

The company also discloses its counterparty credit risk management strategy, which involves diversification and evaluation of the creditworthiness of counterparties. The company has agreements with counterparties that include collateral provisions, but as of December 31, 2023, it was not required to post collateral.

===== REPUBLIC SERVICES INC =====

Here is a concise summary of the text:

The company discloses its market risk exposure, including interest rate risk, fuel price risk, and commodities price risk. For interest rate risk, the company uses a combination of fixed and floating rate debt and has historically entered into swap agreements to manage exposure. For fuel price risk, the company may enter into fuel hedges to mitigate market risk, but currently has no hedges in place. For commodities price risk, the company has previously entered into derivative instruments to manage exposure, but currently has no hedges in place. The company provides quantitative and qualitative disclosures about its market risk exposure and how it manages that risk.

36.1.3 System prompt

The system prompt is an initial set of instructions that serve as the starting point when starting a new chat session. This defines things for the model and helps to focus its capabilities. It is a good place to define a role or a well-known person: the model will assume that role or person including their style.

Creating prompts based on the role or perspective of the person can be a useful technique for generating more relevant and engaging responses from language models.

```
role = "You are a helpful first-grade teacher."
prompt = "Write a simple summary of the text for a first-grader."
simple = {}
for permno in docs.index:
    print('=====', univ.loc[permno, 'comnam'], '=====')
    simple[permno] = generate_response(found.loc[permno, 'item'],
                                         role=role, prompt=prompt)
    print("\n".join([textwrap.fill(s) for s in simple[permno].split('\n')]))
print()
```

===== FORD MOTOR CO DEL =====

Hi there! So, you know how sometimes things can change, like the price of something or the value of money? Well, our company has to deal with those kinds of changes, and we have to make sure we're prepared.

We have a special team that helps us figure out how to manage those changes, and they use special tools like computers and math to help us make good decisions. They also work with other teams to make sure we're making the best choices for our company.

One of the things they do is use something called "derivatives" to help us manage our risks. It's like having insurance, but instead of protecting our stuff, it helps us protect our money.

We also have to think about things like interest rates and currency exchange rates, which can affect how much money we have. It's like trying to predict the weather, but instead of rain or sunshine, we're trying to predict how much money we'll have!

Our team is very good at this, and they work hard to make sure we're making the best decisions for our company. They also keep an eye on things and make sure we're not taking too much risk.

So, that's what our team does! They help us manage our risks and make good decisions for our company

===== OCCIDENTAL PETROLEUM CORP =====

Here's a summary of the text for a first-grader:

Occidental is a big company that sells oil and gas. They have to worry about how much money they make because of things like oil prices going up or down. They also have to be careful about how much money they borrow and how much they owe to other people. They have special rules to make sure they don't lose too much money. They also have to make sure they can pay back the money they borrow.

===== ADOBE INC =====

Here's a summary of the text for a first-grader:

(continues on next page)

(continued from previous page)

Our company does business in many countries, and we need to protect ourselves from changes in the value of money between those countries. We do this by using special contracts called "hedges" that help us keep our earnings and cash flows safe.

Imagine you have some money in a piggy bank, and the value of that money changes. If the value goes up, you're happy! But if it goes down, you might lose some of your money. That's what can happen when we do business in different countries and the value of their money changes.

To keep our money safe, we use hedges to protect ourselves from these changes. We do this by making special deals with other companies that help us keep our earnings and cash flows safe. It's like having a special insurance policy that helps us keep our money safe!

===== REGENERON PHARMACEUTICALS INC =====

Here's a summary of the text for a first-grader:

Imagine you have some money in a special account. Sometimes, the value of that money can go up or down. This can happen because of changes in interest rates, which are like the prices of loans. It's like if you borrowed money from a friend, and the interest rate changed, it would affect how much you have to pay back.

We also have to worry about the credit quality of the things we invest in, like bonds. It's like if you lent your friend some money, and they might not be able to pay you back. We have to make sure we're not lending too much to people who might not be able to pay us back.

Sometimes, we sell things to people in other countries, and we have to worry about the exchange rates. It's like if you went to a different country and exchanged your money for their money, and the rate changed. It could affect how much money you have.

Finally, we have to worry about the prices of the things we invest in, like stocks. It's like if you bought a toy, and its price went up or down. We have to be careful so we don't lose too much money.

That's it!

===== ILLINOIS TOOL WORKS INC =====

Here's a summary of the text for a first-grader:

The company we're talking about is like a big store that sells things all around the world. They have to deal with some risks, like if the money they borrow changes in value. This can happen if the interest rates change or if the money they borrow is in a different country's money. But don't worry, they have ways to protect themselves from these risks. They even have special notes that help them keep track of the money they borrow in different countries.

===== LULULEMON ATHLETICA INC =====

Here's a summary of the text for a first-grader:

(continues on next page)

(continued from previous page)

Imagine you have some money in a special account, and you want to buy something from another country. The problem is that the money in your account is in one kind of money, like dollars, but the thing you want to buy is in another kind of money, like euros. This can make it hard to know how much the thing really costs.

To help with this problem, some companies use special contracts to make sure they get the right amount of money. These contracts are like promises to buy or sell something at a certain price. They help the company know how much money they need to buy the thing they want.

Sometimes, the value of money can change, like if the dollar gets stronger or weaker. This can make it harder for the company to know how much money they need. To help with this, some companies use special tools, like special kinds of contracts, to try to keep the value of their money steady.

The company also talks about other kinds of risks, like the risk of not getting paid back if they borrow money, or the risk of not being able to sell their products because of things like inflation.

===== M S C I INC =====

Here's a summary of the text for a first-grader:

Our company does business with people from different countries, and that means we have to deal with different kinds of money. Sometimes, the value of one kind of money can change compared to another kind of money. This can affect how much money we make or spend. We try to protect ourselves from these changes by using special tools, like contracts, to help keep our money safe.

===== NETFLIX INC =====

Here's a summary of the text for a first-grader:

Our company does business in many different countries and uses different currencies. This means we can get hurt if the value of those currencies changes. For example, if the dollar gets weaker, it might cost us more money to buy things in other countries. We try to protect ourselves from these changes by using special contracts called "hedges." These hedges help us keep our money safe, even if the currencies change.

===== COMCAST CORP NEW =====

Here's a summary of the text for a first-grader:

Imagine you have some money in a special account, and you want to make sure it's safe. To do that, you might put some of that money in a special kind of account that helps keep it safe. This is like having a special kind of insurance for your money.

The company I work for does something similar. We have money that we use to buy things and pay for things, and we want to make sure that money is safe. So, we put some of that money in special accounts that help keep it safe. This is like having a special kind of insurance for our money.

(continues on next page)

(continued from previous page)

We also have to worry about something called "market risk." This is like when the price of something you want to buy goes up or down. We want to make sure that our money is safe, so we use special kinds of accounts to help keep it safe.

We also have to worry about something called "foreign exchange risk." This is like when you go to a different country and you need to exchange your money for the money of that country. We want to make sure that our money is safe, so we use special kinds of accounts to help keep it safe.

We also have to worry about something called "

===== REPUBLIC SERVICES INC =====

Here's a summary of the text for a first-grader:

The company that makes trash bags and recycling centers is talking about some big numbers and risks. They're worried about changes in interest rates, which are like the prices of loans. If interest rates go up or down, it could cost them more or less money. They're also worried about the prices of fuel, like gasoline, and how it affects their business. They're trying to find ways to make sure they're not hurt too much if the prices of fuel or other important things change.

36.2 ChatGPT LLM

TODO

36.2.1 OpenAI APIs

```
# TODO: AzureOpenAI
```

36.3 Text summarization and evaluation

TODO

36.3.1 BLEU

36.3.2 ROUGE

36.3.3 Readability

CHAPTER
THIRTYSEVEN

LLM FINETUNING

Concepts:

- Industry Classification
- Unslot APIs
- Performance Efficient Fine-tuning
- Supervised Fine-tuning

```
import numpy as np
import pandas as pd
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn import metrics
import torch
from unsloth import FastLanguageModel, is_bfloat16_supported
from datasets import Dataset
from trl import SFTTrainer
from transformers import TrainingArguments
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 credentials, paths
# %matplotlib qt
```

```
/home/terence/env3.11/lib/python3.11/site-packages/tqdm/auto.py:21: TqdmWarning:-
  ↵IPProgress not found. Please update jupyter and ipywidgets. See https://
  ↵ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

💡 **Unslot: Will patch your computer to enable 2x faster free finetuning.**

```
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)
```

(continues on next page)

(continued from previous page)

```
ed = Edgar(paths['10X'], zipped=True, verbose=0)
store = Store(paths['scratch'], ext='pkl')
```

Last FamaFrench Date 2024-04-30 00:00:00

Load 10-K business description text for industry classification task

```
# Retrieve universe of stocks
univ = crsp.get_universe(bd.endmo(20221231))

# 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]

#fullnames = Series({'Durbl': "Durable", 'Enrgy': "Energy",
#                    'HiTec': "Technology", 'Hlth': "Healthcare",
#                    'Manuf': "Manufacturing", 'NoDur': "Nondurable",
#                    'Other': "Other", 'Shops': "Retail",
#                    'Telcm': "Telecommunications", 'Utils': "Utilities"})
#univ['sector'] = fullnames[sectors[industry]].values

# same permnos from other earlier experiments
permnos = list(store.load('nouns').keys())

# retrieve 2023 bus10K's
item, form = 'bus10K', '10-K'
rows = DataFrame(ed.open(form=form, item=item))
found = rows[rows['date'].between(20230101, 20231231)] \
    .drop_duplicates(subset=['permno'], keep='last') \
    .set_index('permno') \
    .reindex(permnos)

# split documents into train/test sets
labels = univ.loc[permnos, 'sector']
train_index, test_index = train_test_split(permnos,
                                           stratify=labels,
                                           random_state=42,
                                           test_size=0.2)
Series(labels).value_counts().rename('count').to_frame()
```

```

  count
sector
Hlth      881
Other     762
HiTec     706
Manuf     344
Shops     321
Durbl     164
NoDur     145
Enrgy      94
Utils      92
Telcm      50

```

```

# helper to decode class label in text
class_labels = np.unique(labels)
def decode_label(text):
    """Extract label from output string text"""
    for lab in sorted(class_labels, key=lambda x: -len(x)): # longest strings first
        if lab.lower() in text.lower():
            return lab
    return ""
print(len(labels), len(train_index), len(test_index), class_labels)

```

```

3559 2847 712 ['Durbl' 'Enrgy' 'HiTec' 'Hlth' 'Manuf' 'NoDur' 'Other' 'Shops'
↳ 'Telcm'
'Utils']

```

37.1 Unsloth APIs

Unsloth's library uses several advanced techniques to make training large language models (LLMs) much faster and more efficient, for example, by optimizing matrix multiplications, which are a key part of LLM training, by chaining them together efficiently.

Installing unsloth:

```
!pip install "unsloth[colab-new] @ git+https://github.com/unslothai/unsloth.git"
```

major_version, minor_version = torch.cuda.get_device_capability() - must install separately since torch 2.2.1 breaks packages

- if major_version >= 8 (new GPUs like Ampere, Hopper GPUs (RTX 30xx, RTX 40xx, A100, H100, L40))


```
!pip install --no-deps packaging ninja einops flash-attn xformers trl peft
accelerate bitsandbytes
```
- else: older GPUs (V100, Tesla T4, RTX 20xx)


```
!pip install --no-deps xformers trl peft accelerate bitsandbytes
```

```

WARNING[XFORMERS]: xFormers can't load C++/CUDA extensions. xFormers was built for:
PyTorch 2.3.0+cu121 with CUDA 1201 (you have 2.2.1+cu121)
Python 3.11.9 (you have 3.11.9)
Please reinstall xformers (see https://github.com/facebookresearch/xformers
↳ #installing-xformers)

```

- install the corresponding xformers (<https://anaconda.org/xformers/xformers/files>)

```
!pip install xformers==0.0.25 # for torch 2.2.1+cu121
```

```
NEW_MODEL = False # True
model_id = "unsloth/llama-3-8b-bnb-4bit"
savedir = str(paths['scratch'] / 'finetuned_model')
output_dir = str(paths['scratch'] / "outputs")
max_seq_length = 8192
dtype = None # None for auto. Float16 for Tesla T4, V100, Bfloat16 for Ampere+
load_in_4bit = True # Use 4bit quantization to reduce memory usage. Can be False.
```

```
# 4bit pre quantized models from unsloth for 4x faster downloading + no OOMs
model, tokenizer = FastLanguageModel.from_pretrained(
    model_name = model_id if NEW_MODEL else savedir,
    max_seq_length = max_seq_length,
    dtype = dtype,
    load_in_4bit = load_in_4bit,
    # token = "hf_...", # if using gated models like meta-llama/Meta-Llama-3-8B
)
```

```
==((=====))==
  \\\  /| GPU: NVIDIA GeForce RTX 3080 Laptop GPU. Max memory: 15.739 GB.-
  ↵Platform = Linux.
  0^0/ \_/_ Pytorch: 2.2.1+cu121. CUDA = 8.6. CUDA Toolkit = 12.1.
  \ \ \ / Bfloat16 = TRUE. Xformers = 0.0.25. FA = True.
  "-____-" Free Apache license: http://github.com/unslotha/unsloth
```

Special tokens have been added in the vocabulary, make sure the associated word embeddings are fine-tuned or trained.
 Unsloth 2024.5 patched 32 layers with 32 QKV layers, 32 O layers and 32 MLP layers.

37.2 Low-rank adaptation

Parameter-efficient fine-tuning (PEFT) methods significantly decrease the computational and storage costs for large language models by only fine-tuning a small number of (extra) model parameters instead of all the model's parameters. Low-rank adaptation (LoRA) reduces the number of trainable parameters by learning pairs of rank-decomposition matrices while freezing the original weights.

The unsloth library replaces HuggingFace's peft:

```
from peft import AutoPeftModelForCausalLM
model = AutoPeftModelForCausalLM.from_pretrained(
    savedir, # YOUR MODEL YOU USED FOR TRAINING
    load_in_4bit = load_in_4bit,
)

from peft import LoraConfig, get_peft_model
config = LoraConfig(
    r=16,
    lora_alpha=16,
    target_modules=["query", "value"],
```

(continues on next page)

(continued from previous page)

```

    lora_dropout=0.1,
    bias="none",
    modules_to_save=["classifier"],
)
model = get_peft_model(model, config)

```

```

# Add LoRA adapters so we only need to update less than 10% of all parameters
if NEW_MODEL:
    model = FastLanguageModel.get_peft_model(
        model,
        r = 16, # Choose any number > 0 ! Suggested 8, 16, 32, 64, 128
        target_modules = ["q_proj", "k_proj", "v_proj", "o_proj",
                          "gate_proj", "up_proj", "down_proj",],
        lora_alpha = 16,
        lora_dropout = 0, # Supports any, but = 0 is optimized
        bias = "none", # Supports any, but = "none" is optimized
        use_gradient_checkpointing = "unsloth", # True or "unsloth" for long context
        random_state = 3407,
        use_rslora = False, # supports rank stabilized LoRA
        loftq_config = None, # supports LoftQ
    )

# [NOTE] To train only on completions (ignoring the user's input) read TRL's docs_
# [here] (https://huggingface.co/docs/trl/sft_trainer#train-on-completions-only).
# [NOTE] Remember to add the **EOS_TOKEN** to the tokenized output!! Otherwise you'll_
# get infinite generations!

```

37.3 HuggingFace datasets

```

EOS_TOKEN = tokenizer.eos_token # Must add EOS_TOKEN
def generate_prompt(text, label=''):
    return f"""
Below is an instruction that describes a task.
Write a response that appropriately completes the request.

### Instruction:
The text of a business description can be classified in industry categories like_
below.

""{'' or ''.join(class_labels)}.

In one word, please respond with the industry classification category
of the following text delimited in triple quotes.

'''{text}'''

### Response:
{label} "" + EOS_TOKEN

```

```

MAX_CHARS = 4096
def data_generator():
    for permno in train_index:

```

(continues on next page)

(continued from previous page)

```

text = ed[found.loc[permno, 'pathname']].replace('\n', '').lower()[:MAX_CHARS]
yield {"text": generate_prompt(text, label=univ.loc[permno, 'sector'])}
dataset = Dataset.from_generator(data_generator)

```

37.4 Supervised fine-tuning

Supervised Fine-Tuning adapts a pre-trained model to a specific task, by training the model on a new labeled dataset to predict the correct label for each input.

The TRL library provides the SFTTrainer class, which is designed to facilitate the SFT process. This class accepts a column in your training dataset that contains the prompt constructed from system instructions, questions, and answers.

https://huggingface.co/docs/trl/sft_trainer

```

MAX_EPOCHS = 16
trainer = SFTTrainer(
    model = model,
    tokenizer = tokenizer,
    train_dataset = dataset,
    dataset_text_field = "text",
    max_seq_length = max_seq_length,
    dataset_num_proc = 2,
    packing = False, # Can make training 5x faster for short sequences.
    args = TrainingArguments(
        per_device_train_batch_size = 2, # 2,
        gradient_accumulation_steps = 4,
        warmup_steps = 5,
        max_steps = -1, # 60, -1
        num_train_epochs = MAX_EPOCHS, # default 3
        learning_rate = 2e-4,
        fp16 = not is_bfloat16_supported(),
        bf16 = is_bfloat16_supported(),
        logging_strategy = "steps" if VERBOSE else "no", # "epoch", "no", "steps"
        logging_steps = 1, # defaults to 500 if logging_strategy="steps"
        optim = "adamw_8bit",
        weight_decay = 0.01,
        lr_scheduler_type = "linear",
        seed = 3407,
        save_strategy="epoch", # Save the model checkpoint every epoch
        output_dir = output_dir,
    ),
)

```

```

/home/terence/env3.11/lib/python3.11/site-packages/trl/trainer/sft_trainer.py:318:_
  ↵UserWarning: You passed a tokenizer with `padding_side` not equal to `right` to_
  ↵the SFTTrainer. This might lead to some unexpected behaviour due to overflow_
  ↵issues when training a model in half-precision. You might consider adding_
  ↵`tokenizer.padding_side = 'right'` to your code.
  warnings.warn(

```

```

# Show current memory stats
gpu_stats = torch.cuda.get_device_properties(0)

```

(continues on next page)

(continued from previous page)

```
start_gpu_memory = round(torch.cuda.max_memory_reserved() / 1024 / 1024 / 1024, 3)
max_memory = round(gpu_stats.total_memory / 1024 / 1024 / 1024, 3)
print(f"GPU = {gpu_stats.name}. Max memory = {max_memory} GB.")
print(f"{start_gpu_memory} GB of memory reserved.")
```

GPU = NVIDIA GeForce RTX 3080 Laptop GPU. Max memory = 15.739 GB.
5.75 GB of memory reserved.

```
# Helpers to generate input tokens and response
def generate_inputs(permno):
    """Transform text corresponding to permno into tokenized input"""
    text = ed[found.loc[permno, 'pathname']].replace('\n', '').lower()[:MAX_CHARS]
    inputs = tokenizer([generate_prompt(text)], return_tensors="pt").to("cuda")
    return inputs
```

```
def generate_output(inputs):
    """Generate response given tokenized input, and return decoded output text"""
    terminators = [
        tokenizer.eos_token_id,
        tokenizer.convert_tokens_to_ids("<|eot_id|>")
    ]
    outputs = model.generate(
        **inputs,
        max_new_tokens=16,
        eos_token_id=terminators,
        pad_token_id=tokenizer.eos_token_id,
        #do_sample=True,
        #temperature=0.01, #0.6,
        #top_p=0.9,
        do_sample=False,
        use_cache=True,
    )
    response = outputs[0][len(inputs[0]):]
    return tokenizer.decode(response, skip_special_tokens=True)
```

```
def evaluate_output(permnos, verbose=VERBOSE):
    """Return predicted and true labels"""
    y_pred, y_true = [], []
    for permno in tqdm(permnos):
        inputs = generate_inputs(permno)
        output = generate_output(inputs).replace('\n', ' ')
        label = decode_label(output)
        gold = univ.loc[permno, 'sector']
        y_pred.append(label)
        y_true.append(gold)
        if verbose:
            print(permno, gold, ' [', label, '] *****', output, '*****')
    return y_pred, y_true
```

```
# Evaluate before fine-tuning
if False:
    FastLanguageModel.for_inference(model) # Enable native 2x faster inference
    y_pred, y_true = evaluate_output(test_index)
```

(continues on next page)

(continued from previous page)

```
# generate classification report
report = metrics.classification_report(y_true=y_true, y_pred=y_pred)
print(f"Classification Report (Test Set) before fine-tuning:")
print(report)
```

Train the model

```
# Training loop
trainer_stats = trainer.train()
```

```
==((=====))==  Unsloth - 2x faster free finetuning | Num GPUs = 1
    \\  /|  Num examples = 2,847 | Num Epochs = 16
  0^0/ \_/_ \  Batch size per device = 2 | Gradient Accumulation steps = 4
  \       /  Total batch size = 8 | Total steps = 5,696
  "-____-"  Number of trainable parameters = 41,943,040
100%|██████████| 5696/5696 [20:39:52<00:00, 13.06s/it]
```

```
{'train_runtime': 74392.7458, 'train_samples_per_second': 0.612, 'train_steps_per_
↳second': 0.077, 'train_loss': 0.494993617025654, 'epoch': 16.0}
```

```
# Show final memory and time stats
used_memory = round(torch.cuda.max_memory_reserved() / 1024 / 1024 / 1024, 3)
used_memory_for_lora = round(used_memory - start_gpu_memory, 3)
used_percentage = round(used_memory / max_memory*100, 3)
lora_percentage = round(used_memory_for_lora/max_memory*100, 3)
print(f'{trainer_stats.metrics["train_runtime"]} seconds used for training.')
print(f'Peak reserved memory = {used_memory} GB.')
print(f'Peak reserved memory for training = {used_memory_for_lora} GB.')
print(f'Peak reserved memory % of max memory = {used_percentage} %.')
print(f'Peak reserved memory for training % of max memory = {lora_percentage} %.)
```

```
74392.7458 seconds used for training.
Peak reserved memory = 9.1 GB.
Peak reserved memory for training = 3.35 GB.
Peak reserved memory % of max memory = 57.818 %.
Peak reserved memory for training % of max memory = 21.285 %.
```

```
# Save fine-tuned model
# **[NOTE]** This ONLY saves the LoRA adapters, and not the full model.
if True:
    model.save_pretrained(savedir) # Local saving
    tokenizer.save_pretrained(savedir)
# model.push_to_hub("your_name/lora_model", token = "...") # Online saving
# tokenizer.push_to_hub("your_name/lora_model", token = "...")
```

```
# Run inference on train split
if False:
    FastLanguageModel.for_inference(model) # Enable native 2x faster inference
```

(continues on next page)

(continued from previous page)

```

y_pred, y_true = evaluate_output(train_index)

# generate classification report
report = metrics.classification_report(y_true=y_true, y_pred=y_pred)
print(f"Classification Report (Training set):")
print(report)

```

```

# Run inference on test split
if True:
    FastLanguageModel.for_inference(model)    # Enable native 2x faster inference
    y_pred, y_true = evaluate_output(test_index)

# generate classification report
report = metrics.classification_report(y_true=y_true, y_pred=y_pred)
print(f"Classification Report (Test Set) after fine-tuning:")
print(report)

```

100% |██████████| 712/712 [09:10<00:00, 1.29it/s]

| Classification Report (Test Set) after fine-tuning: | | | | |
|---|-----------|--------|----------|---------|
| | precision | recall | f1-score | support |
| | 0.00 | 0.00 | 0.00 | 0 |
| Durbl | 0.33 | 0.61 | 0.43 | 33 |
| Enrgy | 1.00 | 0.47 | 0.64 | 19 |
| HiTec | 0.74 | 0.59 | 0.66 | 141 |
| Hlth | 0.96 | 0.89 | 0.92 | 176 |
| Manuf | 0.70 | 0.71 | 0.71 | 69 |
| NoDur | 0.70 | 0.55 | 0.62 | 29 |
| Other | 0.80 | 0.70 | 0.75 | 153 |
| Shops | 0.77 | 0.78 | 0.78 | 64 |
| Telcm | 1.00 | 0.90 | 0.95 | 10 |
| Utils | 0.65 | 0.94 | 0.77 | 18 |
| accuracy | | | 0.72 | 712 |
| macro avg | 0.70 | 0.65 | 0.66 | 712 |
| weighted avg | 0.79 | 0.72 | 0.75 | 712 |

```

/home/terence/env3.11/lib/python3.11/site-packages/sklearn/metrics/_classification.
  ↵py:1509: UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in
  ↵labels with no true samples. Use `zero_division` parameter to control this
  ↵behavior.
  ↵_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/terence/env3.11/lib/python3.11/site-packages/sklearn/metrics/_classification.
  ↵py:1509: UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in
  ↵labels with no true samples. Use `zero_division` parameter to control this
  ↵behavior.
  ↵_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/terence/env3.11/lib/python3.11/site-packages/sklearn/metrics/_classification.
  ↵py:1509: UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in
  ↵labels with no true samples. Use `zero_division` parameter to control this
  ↵behavior.
  ↵_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))

```


LLM RAG, CHATBOTS AND AGENTS

Concepts:

- Retrieval Augmented Generation (RAG)
- Chatbots
- LLM Agents

References:

- Lim, T., 2010, Measuring the value of corporate philanthropy: Social impact, business benefits, and investor returns. published by CECP.

```
# Chroma requires SQLite > 3.35, if you encounter issues with having too
# low of a SQLite version, try installing 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')
import sqlite_version    # hack to use lower version of SQLite

# !ollama pull mxbai-embed-large  # vector embeddings model in Ollama
import ollama

from langchain_community.llms import Ollama
from langchain_community.text_splitters import RecursiveCharacterTextSplitter
# import pdfplumber    # PDF document loaders
# from langchain_community.document_loaders import PyPDFLoader
from langchain_community.document_loaders import UnstructuredMarkdownLoader
import chromadb
from tqdm import tqdm
import textwrap

def wrap(text):
    """Helper to wrap text for pretty printing"""
    return "\n".join([textwrap.fill(s) for s in text.split('\n')])
```

38.1 Retrieval Augmented Generation

38.1.1 Langchain APIs

Common interface to models served locally (e.g. Ollama) or remotely (e.g. cloud-based server)

<https://python.langchain.com/v0.1/docs/integrations/llms/ollama/>

```
# temperature set low but nonzero so that chat responses can vary slightly
llm = Ollama(model="llama3:instruct", temperature=0.3)
```

Read and split input documents into chunks of around 1000 characters with overlap. The document is a textbook on the subject of measuring the value of corporate philanthropy, which will drive the examples of a chatbot, and a collaborative conversation between multiple LLM agents.

```
loader = UnstructuredMarkdownLoader('MVC.P.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: 198
[Document(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\n(through
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.', metadata={'source': 'MVC.P.md'}),
Document(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
companies wishing to adapt current methodologies in\nthe field to
their own corporate giving programs.\nTo realize meaningful benefits,
philanthropy cannot be treated as just\nanother "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', metadata={'source': 'MVC.P.md'})]
```

38.1.2 Vector Database

Chroma DB is an open-source vector store that is utilized for the storage and retrieval of vector embeddings. It performs fast search of text data, and is used by semantic search engines and large language models.

```
client = chromadb.Client()
#client.delete_collection(name="docs")
collection = client.get_or_create_collection(name="docs")
```

Ollama can also serve word embeddings models, such as the “mxbai-embed-large” model.

```
# store each document in a vector embedding 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]
    )
```

Example of Question-Answer prompt

```
# an example prompt
query = "Why is it important to measure 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 most similar document chunks from the vector database

```
# retrieve the most relevant doc
results = collection.query(
    query_embeddings=[response["embedding"]],
    n_results=3,
)
data = ". ".join(results['documents'][0])
print(textwrap.fill(data))
```

A model for measuring the influence of corporate philanthropic initiatives on employee engagement.. consumer satisfaction and companies philanthropic record only when companies are perceived to have strong product quality and innovation capabilities and or operate in consumer oriented industries 22 Designing a measurement framework for loyalty should begin with an assessment of the perceptions customers have already developed as a result of a company s corporate

(continues on next page)

(continued from previous page)

philanthropic initiatives and whether these perceptions are contributing to higher loyalty scores Figure 9 suggests such a framework based on the literature reviewed Figure 9 A Framework for Measuring Customer Loyalty and Corporate Philanthropy Source Adapted from Bhattacharya C B Sen S 2004 and Smith V Langford P 2009 A company's marketing department is likely already to have implemented its own customer loyalty metrics in which case it is sensible to leverage these along with customized deliberate customer research It is imperative that the additional factors affecting loyalty scores e.g. philanthropic strategies criteria and disclosures are linked to can be successful in financial value There is a significant creating long term opportunity for companies to lead the business value industry in developing standards or differentiating themselves to the investor community through their disclosures about philanthropic efforts Documentation of the measurement process should be an important part of establishing quality disclosures and standards A high quality measurement process is a critical input for good management and demonstrates that a company recognizes how its philanthropic strategies can be successful in creating long term business value The Dow Jones Sustainability Indexes questionnaire also asks if the company has in place a measurement system although it does not provide guidance about what Dow Jones considers to be a good system The review and findings summarized in this report suggest that companies could

Generate a response combining the query and data we retrieved in previous step

```
prompt = f"""
Only use relevant information in the following text delimited by triple quotes:

'''{data}''' 

Respond to this prompt: {query}.

In your response, only use information in the given text.
Do not mention that you referred to the text."""

output = llm.invoke(prompt)
print(textwrap.fill(output))
```

It is imperative that the additional factors affecting loyalty scores, such as philanthropic strategies and disclosures, are linked to can be successful in financial value. This suggests that measuring the value of corporate philanthropy is important because it can create long-term business value. Additionally, a high-quality measurement process demonstrates that a company recognizes its philanthropic strategies can be successful in creating long-term business value, which can lead to differentiating themselves to the investor community through their disclosures about philanthropic efforts.

```
# show the supporting source document chunk ids
print(set(results['ids'][0]))
```

```
{'95', '120', '187'}
```

38.2 Chatbot

The chatbot is an LLM instance, where a record of the past conversation is included in the next prompt, so the chatbot replies to the user's question with memory of past queries and responses.

```
system_prompt = "You are a competent AI business manager."
```

```
instruction = """
I will provide you with ideas.
Please summarize and briefly explain all
the ideas, including those I have provided you earlier, in a bulleted list.
"""
```

```
# Hard-code the sequence of user queries.
user_queries = [
    "Idea: Sum up the amount of charitable contributions.",
    "Idea: Measure the dollar value of goods contributed.",
    "Idea: Count total employee volunteering hours.",
]
```

Concatenate user and response turns to form prompt

```
# Loop over number of incremental user queries and build up memory of responses

# These special input symbols and sequences are needed to construct the prompt
# including the past conversation. See
# https://llama.meta.com/docs/model-cards-and-prompt-formats/meta-llama-3/
tokens = {'bos_token': '<|begin_of_text|>',
          'eos_token': '<|end_of_text|>',
          'pad_token': '<|reserved_special_token_250|>',
          'system_header': '<|start_header_id|>system<|end_header_id|>',
          'assistant_header': '<|start_header_id|>assistant<|end_header_id|>',
          'user_header': '<|start_header_id|>user<|end_header_id|>',
          'eot': '<|eot_id|>'}
}

responses = [] # memory of past AI-assistant turns
for turn in tqdm(range(len(user_queries))):
    # 1. Start with system message
    prompt = tokens['bos_token']
    prompt += "\n".join([tokens['system_header'],
                        system_prompt,
                        tokens['eot']]) + '\n\n'

    # 2. Add user instruction and first user prompt
    prompt += "\n".join([tokens['user_header'],
                        instruction,
                        user_queries[0],
                        tokens['eot']]) + '\n\n'

    # 3. Loop through past turns of AI response and user prompt
    for response, query in zip(responses, user_queries[1:][:len(responses)]):
        # a. Add AI response from memory
        prompt += "\n".join([tokens['assistant_header'],
                            response,
                            tokens['eot']]) + '\n\n'
```

(continues on next page)

(continued from previous page)

```
        response,
        tokens['eot']]) + '\n\n'

    # b. Add next user prompt
    prompt += "\n".join([tokens['user_header'],
                        query,
                        tokens['eot']]) + '\n\n'

    # 4. End with assistant header to elicit AI response
    prompt += tokens['assistant_header']

    response = llm.invoke(prompt)
    responses.append(response)
```

100% |██████████| 3/3 [00:16<00:00, 5.53s/it]

Show the final response from the last turn of the Chatbot

```
print('FINAL RESPONSE:')
print(wrap(response))
```

FINAL RESPONSE:

Here's the updated list with the new idea:

* **Sum up the amount of charitable contributions**: This idea likely refers to calculating and aggregating the total amount of donations made by a company or organization to various charities. This could be an important metric for tracking philanthropic efforts and measuring social responsibility.

* **Measure the dollar value of goods contributed**: This idea suggests quantifying the monetary value of goods, products, or services donated by a company or individual to charitable causes. This could include in-kind donations, product drives, or other forms of non-monetary giving.

* **Count total employee volunteering hours**: This idea focuses on tracking and aggregating the total number of hours that employees spend volunteering for charitable organizations or community events. This metric can help measure employee engagement, corporate social responsibility, and the impact of company-sponsored volunteer programs.

Please provide more ideas, and I'll keep adding them to the list!

Show the last prompt that was asked of the chatbot: it includes the entire memory of the past conversation.

```
print("FINAL PROMPT:")
print(prompt)
```

FINAL PROMPT:

<|begin_of_text|><|start_header_id|>system<|end_header_id|>
You are a competent AI business manager.
<|eot_id|>

(continues on next page)

(continued from previous page)

```
<|start_header_id|>user<|end_header_id|>

I will provide you with ideas.
Please summarize and briefly explain all
the ideas, including those I have provided you earlier, in a bulleted list.

Idea: Sum up the amount of charitable contributions.
<|eot_id|>

<|start_header_id|>assistant<|end_header_id|>
Since this is the first idea you've provided, I'll start by summarizing it:

* **Sum up the amount of charitable contributions**: This idea likely refers to
  ↵calculating and aggregating the total amount of donations made by a company or
  ↵organization to various charities. This could be an important metric for
  ↵tracking philanthropic efforts and measuring social responsibility.

Please go ahead and provide more ideas, and I'll add them to the list!
<|eot_id|>

<|start_header_id|>user<|end_header_id|>
Idea: Measure the dollar value of goods contributed.
<|eot_id|>

<|start_header_id|>assistant<|end_header_id|>
Here's the updated list with the new idea:

* **Sum up the amount of charitable contributions**: This idea likely refers to
  ↵calculating and aggregating the total amount of donations made by a company or
  ↵organization to various charities. This could be an important metric for
  ↵tracking philanthropic efforts and measuring social responsibility.
* **Measure the dollar value of goods contributed**: This idea suggests
  ↵quantifying the monetary value of goods, products, or services donated by a
  ↵company or individual to charitable causes. This could include in-kind donations,
  ↵product drives, or other forms of non-monetary giving.

Please provide more ideas, and I'll keep adding them to the list!
<|eot_id|>

<|start_header_id|>user<|end_header_id|>
Idea: Count total employee volunteering hours.
<|eot_id|>

<|start_header_id|>assistant<|end_header_id|>
```

38.3 Agents

In multi-agent LLM systems, multiple LLM agents interact with each other or work in collaboration to achieve complex tasks or goals. Frameworks such as Microsoft's Autogen provides a multi-agent conversation framework as a high-level abstraction with an open-source library to build complex workflows.

Below, we model a simpler workflow of a conversation between three LLM agents, which perform the roles of a Chief Executive Officer, a Chief Giving Officer (who manages the company's corporate philanthropy program), and an AI Assistant. Over several turns, they gradually explore and build up a plan for measuring the value of their program, drawing only from information in the RAG document store.

38.3.1 Role-playing

AI Assistant agent

This agent responds to a query using RAG. To reduce hallucination, the prompt is clearly written to stick to the retrieved text, and to simply respond "I don't know" if the answer cannot be found in the text. This agent performs the role of an AI Corporate Giving expert assistant (named CorGi), since its responses only uses information retrieved from the corporate philanthropy textbook.

```
def rag_agent(text, n_results=3):
    """Role of a RAG-based question-answering agent"""
    response = ollama.embeddings(prompt=text, model="mxbai-embed-large")

    results = collection.query(query_embeddings=[response["embedding"]],
                               n_results=n_results)
    data = "\n".join(results['documents'][0])

    rag_prompt=f"""
    You are a helpful AI assistant.
    Only use the information in the following text delimited by triple quotes:
    """
    rag_prompt+=f"""
    {data}
    """

    Provide an answer to this question: {text}.

    You answer must only contain information in the given text.
    If the information requested is not in the text, say "I don't know".
    Be concise.
    Do not mention that you referred to the text."""

    # generate a response combining the prompt and data
    output = llm.invoke(rag_prompt)
    return output, results['ids'][0]
```

Chief Giving Officer agent

This role collects and summarizes of the responses received from the AI Assistant agent, to present to the Chief Executive Officer.

```
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
```

(continues on next page)

(continued from previous page)

of your company's corporate philanthropy program.
Only use the most relevant information about the plan
given in the following text delimited by triple quotes:

```
'''{text}'''  
  
Please summarize and explain all the parts of the plan given in the text using a  
→bulleted list.  
Be complete and concise.  
In your summary, do not use any information that is not in the given text.  
'''  
    output = llm.invoke(cgo_prompt)  
    return output
```

Chief Executive Officer agent

This LLM agent generates questions, given the text of the plan, for the other agents to answer and add improvements.

```
def ceo_agent(text):  
    """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.  
The plan is given in the text delimited by triple quotes:  
  
'''{text}'''  
  
Please ask a simple, general question for an idea to improve the your company's plan  
that is different than the text.  
Be concise."""  
  
    output = llm.invoke(ceo_prompt)  
    return output
```

Loop over several turns of the conversation between the three agents, who collaboratively develop the 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 = rag_agent(question)  
    docs.extend(doc)  
    print('Corgi-AI >>>', wrap(answer))  
    print('\n-----\n')  
    memory = cgo_agent("\n".join([memory, answer]))  
    print('CGO >>>', wrap(memory))  
    print('\n-----')  
    print('Source documents:', set(docs))
```

```
Turn 1...  
CEO >>> What metrics would you recommend tracking to gauge the program's
```

(continues on next page)

(continued from previous page)

impact on employee engagement and morale?

Corgi-AI >>> How are outcome metrics designed and tracked? Draws from knowledge and experience of third-party domain-area experts engaged to collect (and/or supervise the collection of) data and then to conduct evaluation analysis.

Outcome metrics for gauging program's impact on employee engagement and morale: How are outcome metrics designed and tracked?

CGO >>> Here is a summary of the plan for measuring the value of the company's corporate philanthropy program:

- ****Measure monetary donations**:** The plan involves tracking the amount of monetary donations made by the company.
 - ****Outcome metrics designed and tracked with expert input**:** To design and track outcome metrics, the company will draw from the knowledge and experience of third-party domain-area experts who will collect (and/or supervise the collection of) data and conduct evaluation analysis.
 - ****Metrics for program impact on employee engagement and morale**:** The plan also includes designing and tracking outcome metrics to gauge the program's impact on employee engagement and morale.
-

Turn 2...

CEO >>> What opportunities are there to leverage our corporate philanthropy program as a recruitment tool to attract top talent who share our values?

Corgi-AI >>> Philanthropy provides access to new relationships and opportunities, whereby the company can find test and demonstrate new ideas, technologies, and products. This could potentially attract top talent who share our values.

CGO >>> Here is a summary of the plan for measuring the value of the company's corporate philanthropy program:

- ****Measure monetary donations**:** The plan involves tracking the amount of monetary donations made by the company.
- ****Outcome metrics designed and tracked with expert input**:** The company will draw from the knowledge and experience of third-party domain-area experts to design and track outcome metrics. These experts will collect data and conduct evaluation analysis.

(continues on next page)

(continued from previous page)

- **Metrics for program impact on employee engagement and morale**:**
The plan includes designing and tracking outcome metrics to gauge the program's impact on employee engagement and morale.

Note: There is no mention of specific metrics or targets in the text, only that the company will design and track them with expert input.

Turn 3...

CEO >>> What steps can be taken to ensure that the value of the corporate philanthropy program is communicated effectively to stakeholders, including employees and customers?

Corgi-AI >>> To ensure effective communication of corporate philanthropy program value to stakeholders, including employees and customers:

- * Leverage existing customer loyalty metrics
- * Conduct customized deliberate customer research
- * Measure perceptions customers have developed as a result of company's corporate philanthropic initiatives and whether these perceptions are contributing to higher loyalty scores.

CGO >>> Here is a summary of the plan for measuring the value of the company's corporate philanthropy program:

- **Measure monetary donations**:** The plan involves tracking the amount of monetary donations made by the company.
- **Outcome metrics designed and tracked with expert input**:** The company will design and track outcome metrics with the help of third-party domain-area experts. These experts will collect data and conduct evaluation analysis to measure the impact of the corporate philanthropy program.
- **Metrics for program impact on employee engagement and morale**:**
The plan includes designing and tracking outcome metrics to gauge the program's impact on employee engagement and morale.
- **Leverage existing customer loyalty metrics**:** To communicate the value of the corporate philanthropy program to stakeholders, including customers, the company will use its existing customer loyalty metrics.
- **Conduct customized deliberate customer research**:** The company will conduct research specifically designed to understand how customers perceive the corporate philanthropic initiatives and whether these perceptions contribute to higher loyalty scores.
- **Measure perceptions and loyalty scores**:** The plan involves measuring the perceptions customers have developed as a result of the company's corporate philanthropic initiatives and whether these perceptions are contributing to higher loyalty scores.

(continues on next page)

(continued from previous page)

Turn 4...

CEO >>> What steps can be taken to ensure that the impact of the corporate philanthropy program on employee engagement and morale is also reflected in the company's overall employee retention rates?

Corgi-AI >>> Companies can engage employees through group volunteer programs and awareness of their philanthropic initiatives which raise employee motivation, productivity, and a sense of identification with the organization. This can contribute to business value by enhancing employee engagement.

CGO >>> Here is a summary of the plan for measuring the value of the company's corporate philanthropy program:

- ****Measure monetary donations**:** Track the amount of monetary donations made by the company to measure the financial aspect of the program.
- ****Outcome metrics designed and tracked with expert input**:** Design and track outcome metrics with the help of third-party domain-area experts to measure the impact of the corporate philanthropy program. These experts will collect data and conduct evaluation analysis.
- ****Metrics for program impact on employee engagement and morale**:** Design and track outcome metrics to gauge the program's impact on employee engagement and morale, which can contribute to business value by enhancing employee engagement.
- ****Leverage existing customer loyalty metrics**:** Use existing customer loyalty metrics to communicate the value of the corporate philanthropy program to stakeholders, including customers.
- ****Conduct customized deliberate customer research**:** Conduct research specifically designed to understand how customers perceive the company's corporate philanthropic initiatives and whether these perceptions contribute to higher loyalty scores.
- ****Measure perceptions and loyalty scores**:** Measure the perceptions customers have developed as a result of the company's corporate philanthropic initiatives and whether these perceptions are contributing to higher loyalty scores.

Turn 5...

CEO >>> What if we also tracked the number of employee volunteer hours and days taken off for volunteering, to better understand how our corporate philanthropy program affects work-life balance and overall job satisfaction?

(continues on next page)

(continued from previous page)

Corgi-AI >>> Positive job-related behaviors include objective metrics such as reduced absenteeism and lower employee turnover. Additionally, enhanced work effort, advocacy, and cooperative conduct are also considered positive outcomes.

CGO >>> Here is a summary of the plan for measuring the value of the company's corporate philanthropy program:

- ****Measure monetary donations**:** Track the amount of monetary donations made by the company to measure the financial aspect of the program.
- ****Outcome metrics designed and tracked with expert input**:** Design and track outcome metrics with the help of third-party domain-area experts to measure the impact of the corporate philanthropy program. These experts will collect data and conduct evaluation analysis.
- ****Metrics for program impact on employee engagement and morale**:** Design and track outcome metrics to gauge the program's impact on employee engagement and morale, which can contribute to business value by enhancing employee engagement.
- ****Leverage existing customer loyalty metrics**:** Use existing customer loyalty metrics to communicate the value of the corporate philanthropy program to stakeholders, including customers.
- ****Conduct customized deliberate customer research**:** Conduct research specifically designed to understand how customers perceive the company's corporate philanthropic initiatives and whether these perceptions contribute to higher loyalty scores.
- ****Measure perceptions and loyalty scores**:** Measure the perceptions customers have developed as a result of the company's corporate philanthropic initiatives and whether these perceptions are contributing to higher loyalty scores.

Note: The plan also mentions positive job-related behaviors, such as reduced absenteeism, lower employee turnover, enhanced work effort, advocacy, and cooperative conduct, but this is not part of the specific measurement plan.

Turn 6...

CEO >>> What are some potential metrics or indicators that could be used to measure the program's impact on the company's reputation and brand equity?

Corgi-AI >>> Enhancing the company's reputation or brand.

CGO >>> Here is a summary of the plan for measuring the value of the company's

(continues on next page)

(continued from previous page)

corporate philanthropy program:

- **Measure monetary donations:** Track the amount of monetary donations made by the company to measure the financial aspect of the program.
- **Outcome metrics designed and tracked with expert input:** Design and track outcome metrics with the help of third-party domain-area experts to measure the impact of the corporate philanthropy program. These experts will collect data and conduct evaluation analysis.
- **Metrics for program impact on employee engagement and morale:** Design and track outcome metrics to gauge the program's impact on employee engagement and morale, which can contribute to business value by enhancing employee engagement.
- **Leverage existing customer loyalty metrics:** Use existing customer loyalty metrics to communicate the value of the corporate philanthropy program to stakeholders, including customers.
- **Conduct customized deliberate customer research:** Conduct research specifically designed to understand how customers perceive the company's corporate philanthropic initiatives and whether these perceptions contribute to higher loyalty scores.
- **Measure perceptions and loyalty scores:** Measure the perceptions customers have developed as a result of the company's corporate philanthropic initiatives and whether these perceptions are contributing to higher loyalty scores.

Turn 7...

CEO >>> What steps can be taken to ensure that the metrics and data collected from the program's impact on employee engagement and morale are also used to inform and improve the overall corporate philanthropy strategy?

Corgi-AI >>> To ensure that metrics and data collected from the program's impact on employee engagement and morale are used to inform and improve the overall corporate philanthropy strategy, CEOs can:

- * View the goal of their philanthropic programs as creating business innovation and building new market knowledge.
- * Recognize that efficient philanthropists tend to view their philanthropic programs' goals in this way.

These steps can help leverage the benefits of philanthropy for long-term financial value by enhancing employee engagement and morale.

CGO >>> Here is a summary of the plan for measuring the value of the company's corporate philanthropy program:

(continues on next page)

(continued from previous page)

- ****Measure monetary donations**:** Track the amount of monetary donations made by the company to measure the financial aspect of the program.
- ****Outcome metrics designed and tracked with expert input**:** Design and track outcome metrics with the help of third-party domain-area experts to measure the impact of the corporate philanthropy program. These experts will collect data and conduct evaluation analysis.
- ****Metrics for program impact on employee engagement and morale**:** Design and track outcome metrics to gauge the program's impact on employee engagement and morale, which can contribute to business value by enhancing employee engagement.
- ****Leverage existing customer loyalty metrics**:** Use existing customer loyalty metrics to communicate the value of the corporate philanthropy program to stakeholders, including customers.
- ****Conduct customized deliberate customer research**:** Conduct research specifically designed to understand how customers perceive the company's corporate philanthropic initiatives and whether these perceptions contribute to higher loyalty scores.
- ****Measure perceptions and loyalty scores**:** Measure the perceptions customers have developed as a result of the company's corporate philanthropic initiatives and whether these perceptions are contributing to higher loyalty scores.
- ****Use metrics to inform and improve strategy**:** View the goal of the philanthropic programs as creating business innovation and building new market knowledge. Recognize that efficient philanthropists tend to view their philanthropic programs' goals in this way, which can help leverage the benefits of philanthropy for long-term financial value by enhancing employee engagement and morale.

Note: The plan does not explicitly mention how often or when these metrics should be measured, but it implies that they will be used to inform and improve the overall corporate philanthropy strategy.

Turn 8...

CEO >>> What are some potential non-monetary metrics we could track to measure the value of our corporate philanthropy program, such as volunteer hours or community partnerships?

Corgi-AI >>> Some potential non-monetary metrics we could track to measure the value of our corporate philanthropy program include:

- * Volunteer hours
- * Community partnerships

CGO >>> Here is a summary of the plan for measuring the value of the company's

(continues on next page)

(continued from previous page)

corporate philanthropy program:

- **Measure monetary donations:** Track the amount of monetary donations made by the company to measure the financial aspect of the program.
- **Outcome metrics designed and tracked with expert input:** Design and track outcome metrics with the help of third-party domain-area experts to measure the impact of the corporate philanthropy program. These experts will collect data and conduct evaluation analysis.
- **Metrics for program impact on employee engagement and morale:** Design and track outcome metrics to gauge the program's impact on employee engagement and morale, which can contribute to business value by enhancing employee engagement.
- **Leverage existing customer loyalty metrics:** Use existing customer loyalty metrics to communicate the value of the corporate philanthropy program to stakeholders, including customers.
- **Conduct customized deliberate customer research:** Conduct research specifically designed to understand how customers perceive the company's corporate philanthropic initiatives and whether these perceptions contribute to higher loyalty scores.
- **Measure perceptions and loyalty scores:** Measure the perceptions customers have developed as a result of the company's corporate philanthropic initiatives and whether these perceptions are contributing to higher loyalty scores.
- **Use metrics to inform and improve strategy:** View the goal of the philanthropic programs as creating business innovation and building new market knowledge. Recognize that efficient philanthropists tend to view their philanthropic programs' goals in this way, which can help leverage the benefits of philanthropy for long-term financial value by enhancing employee engagement and morale.
- **Non-monetary metrics:** Consider tracking non-monetary metrics such as volunteer hours and community partnerships to measure the value of the corporate philanthropy program.

Turn 9...

CEO >>> What steps can we take to ensure that our employees are aware of and feel invested in the corporate philanthropy program, beyond just tracking metrics?

Corgi-AI >>> Companies can engage employees through group volunteer programs and awareness of their philanthropic initiatives, which raise employee motivation, productivity, and a sense of identification with the organization.

(continues on next page)

(continued from previous page)

CGO >>> Here is a summary of the plan for measuring the value of the company's corporate philanthropy program:

- **Measure monetary donations**: Track the amount of monetary donations made by the company to measure the financial aspect of the program.
- **Outcome metrics designed and tracked with expert input**: Design and track outcome metrics with the help of third-party domain-area experts to measure the impact of the corporate philanthropy program. These experts will collect data and conduct evaluation analysis.
- **Metrics for program impact on employee engagement and morale**: Design and track outcome metrics to gauge the program's impact on employee engagement and morale, which can contribute to business value by enhancing employee engagement.
- **Leverage existing customer loyalty metrics**: Use existing customer loyalty metrics to communicate the value of the corporate philanthropy program to stakeholders, including customers.
- **Conduct customized deliberate customer research**: Conduct research specifically designed to understand how customers perceive the company's corporate philanthropic initiatives and whether these perceptions contribute to higher loyalty scores.
- **Measure perceptions and loyalty scores**: Measure the perceptions customers have developed as a result of the company's corporate philanthropic initiatives and whether these perceptions are contributing to higher loyalty scores.
- **Use metrics to inform and improve strategy**: View the goal of the philanthropic programs as creating business innovation and building new market knowledge. Recognize that efficient philanthropists tend to view their philanthropic programs' goals in this way, which can help leverage the benefits of philanthropy for long-term financial value by enhancing employee engagement and morale.
- **Non-monetary metrics**: Consider tracking non-monetary metrics such as volunteer hours and community partnerships to measure the value of the corporate philanthropy program.

Turn 10...

CEO >>> What steps can be taken to ensure that the impact of our corporate philanthropy program on employee engagement and morale is not only measured but also effectively communicated to employees, fostering a sense of ownership and motivation?

Corgi-AI >>> To ensure the impact of our corporate philanthropy program on employee engagement and morale is effectively communicated to employees, fostering a sense of ownership and motivation:

- * Understand the diverse range of emotional needs of employees

(continues on next page)

(continued from previous page)

- * Design programs that fulfill those needs
 - * Measure objective metrics such as reduced absenteeism, lower employee turnover, and greater efficiency
 - * Assess subjective outcomes like enhanced work effort, advocacy, and co-operative conduct in performance reviews
-

CGO >>> Here is a summary of the plan for measuring the value of the company's corporate philanthropy program:

- ****Monetary donations**:** Track the amount of monetary donations made by the company to measure the financial aspect of the program.
- ****Outcome metrics**:** Design and track outcome metrics with expert input to measure the impact of the corporate philanthropy program. This includes metrics for:
 - + Program impact on employee engagement and morale
 - + Customer loyalty and perceptions
- ****Existing customer loyalty metrics**:** Use existing customer loyalty metrics to communicate the value of the corporate philanthropy program to stakeholders, including customers.
- ****Deliberate customer research**:** Conduct research specifically designed to understand how customers perceive the company's corporate philanthropic initiatives and whether these perceptions contribute to higher loyalty scores.
- ****Perceptions and loyalty scores**:** Measure the perceptions customers have developed as a result of the company's corporate philanthropic initiatives and whether these perceptions are contributing to higher loyalty scores.
- ****Strategy improvement**:** View the goal of the philanthropic programs as creating business innovation and building new market knowledge. Recognize that efficient philanthropists tend to view their philanthropic programs' goals in this way, which can help leverage the benefits of philanthropy for long-term financial value by enhancing employee engagement and morale.
- ****Non-monetary metrics**:** Consider tracking non-monetary metrics such as:
 - + Volunteer hours
 - + Community partnerships
- ****Employee engagement and morale**:** To ensure the impact of the corporate philanthropy program on employee engagement and morale is effectively communicated to employees, fostering a sense of ownership and motivation:
 - + Understand the diverse range of emotional needs of employees
 - + Design programs that fulfill those needs
 - + Measure objective metrics such as:
 - Reduced absenteeism
 - Lower employee turnover
 - Greater efficiency

(continues on next page)

(continued from previous page)

- + Assess subjective outcomes like:
 - Enhanced work effort
 - Advocacy
 - Co-operative conduct in performance reviews

Source documents: {'15', '194', '5', '98', '120', '87', '2', '1', '102', '109', '94', '8', '97', '4', '85', '86', '28', '66', '95'}

```
print("FINAL ANSWER")
print(wrap(memory))
```

FINAL ANSWER

Here is a summary of the plan for measuring the value of the company's corporate philanthropy program:

- ****Monetary donations**:** Track the amount of monetary donations made by the company to measure the financial aspect of the program.
- ****Outcome metrics**:** Design and track outcome metrics with expert input to measure the impact of the corporate philanthropy program. This includes metrics for:
 - + Program impact on employee engagement and morale
 - + Customer loyalty and perceptions
- ****Existing customer loyalty metrics**:** Use existing customer loyalty metrics to communicate the value of the corporate philanthropy program to stakeholders, including customers.
- ****Deliberate customer research**:** Conduct research specifically designed to understand how customers perceive the company's corporate philanthropic initiatives and whether these perceptions contribute to higher loyalty scores.
- ****Perceptions and loyalty scores**:** Measure the perceptions customers have developed as a result of the company's corporate philanthropic initiatives and whether these perceptions are contributing to higher loyalty scores.
- ****Strategy improvement**:** View the goal of the philanthropic programs as creating business innovation and building new market knowledge. Recognize that efficient philanthropists tend to view their philanthropic programs' goals in this way, which can help leverage the benefits of philanthropy for long-term financial value by enhancing employee engagement and morale.
- ****Non-monetary metrics**:** Consider tracking non-monetary metrics such as:
 - + Volunteer hours
 - + Community partnerships
- ****Employee engagement and morale**:** To ensure the impact of the corporate philanthropy program on employee engagement and morale is effectively communicated to employees, fostering a sense of ownership and motivation:

(continues on next page)

(continued from previous page)

- + Understand the diverse range of emotional needs of employees
- + Design programs that fulfill those needs
- + Measure objective metrics such as:
 - Reduced absenteeism
 - Lower employee turnover
 - Greater efficiency
- + Assess subjective outcomes like:
 - Enhanced work effort
 - Advocacy
 - Co-operative conduct in performance reviews

For attribution, show the source reference documents which supported the responses.

```
print("Source documents reference id's:", set(docs))
```

```
Source documents reference id's: {'15', '194', '5', '98', '120', '87', '2', '1',  
'102', '109', '94', '8', '97', '4', '85', '86', '28', '66', '95'}
```