

---

# **Financial Data Science Python Notebooks**

**Terence Lim**

**Sep 09, 2023**



# CONTENTS

<b>1 Stock prices</b>	<b>3</b>
1.1 Sample selection . . . . .	3
1.2 Stock delistings . . . . .	7
1.3 Stock distributions and returns . . . . .	10
<b>2 Jegadeesh-Titman Rolling Portfolios</b>	<b>15</b>
2.1 Univariate sorts . . . . .	16
2.2 Newey-West correction . . . . .	19
2.3 Hypothesis Testing . . . . .	22
<b>3 Fama-French Portfolio Sorts</b>	<b>25</b>
3.1 Bivariate portfolio sorts . . . . .	27
3.2 Linear Regression . . . . .	39
3.3 CAPM . . . . .	39
<b>4 Fama-Macbeth Cross-sectional Regressions</b>	<b>41</b>
4.1 Cross-sectional regressions . . . . .	43
4.2 Kernel Regression . . . . .	47
4.3 Cross-sectional regressions and portfolio sorts . . . . .	47
<b>5 Weekly reversal</b>	<b>51</b>
5.1 Implementation Shortfall . . . . .	54
5.2 Information coefficient . . . . .	54
5.3 Structural break with unknown changepoint . . . . .	55
<b>6 Quant Factors</b>	<b>59</b>
6.1 Past prices . . . . .	62
6.2 Liquidity . . . . .	70
6.3 Fundamentals . . . . .	75
6.4 Earnings Estimates . . . . .	103
6.5 Performance evaluation . . . . .	113
<b>7 Event Study</b>	<b>121</b>
7.1 Key developments . . . . .	122
7.2 Event study . . . . .	122
<b>8 Multiple Testing</b>	<b>133</b>
<b>9 Economic Data Revisions</b>	<b>135</b>
9.1 Popular FRED series . . . . .	135

<b>10 Linear Regression Diagnostics</b>	<b>149</b>
10.1 Linear regression . . . . .	150
10.2 Residual plots . . . . .	153
<b>11 Econometric Forecasts</b>	<b>159</b>
11.1 Seasonality . . . . .	160
11.2 Autocorrelation . . . . .	162
11.3 Stationarity . . . . .	162
11.4 AR, ARMA, SARIMAX . . . . .	163
11.5 Forecasting . . . . .	165
11.6 Granger Causality . . . . .	169
11.7 Vector Autoregression . . . . .	172
<b>12 Approximate Factor Models</b>	<b>175</b>
12.1 FRED-MD . . . . .	175
12.2 Approximate factor model . . . . .	177
<b>13 State Space Models</b>	<b>183</b>
13.1 Hidden Markov Model . . . . .	185
13.2 Gaussian Mixture Model . . . . .	189
13.3 Dynamic Factor Models . . . . .	191
<b>14 Conditional Volatility</b>	<b>193</b>
14.1 GARCH . . . . .	195
<b>15 Value at Risk</b>	<b>199</b>
<b>16 Covariance Matrix</b>	<b>201</b>
16.1 PCA and scree plot . . . . .	203
<b>17 Term Structure</b>	<b>209</b>
17.1 Term Structure of Interest Rates . . . . .	209
17.2 Splines . . . . .	213
<b>18 Bond Returns</b>	<b>217</b>
<b>19 Options Pricing</b>	<b>225</b>
<b>20 Market Microstructure</b>	<b>227</b>
<b>21 Intraday liquidity</b>	<b>229</b>
<b>22 Event Risk</b>	<b>243</b>
<b>23 Topic Modelling</b>	<b>245</b>
23.1 Scrape FOMC Minutes text . . . . .	259
23.2 Topic models . . . . .	261
<b>24 Management Sentiment Analysis</b>	<b>271</b>
24.1 Loughran and MacDonald dictionaries . . . . .	272
<b>25 Business Text Analysis</b>	<b>281</b>
<b>26 10-K Business Descriptions</b>	<b>283</b>
<b>27 Bag-of-words Tf-Idf</b>	<b>285</b>
27.1 Logistic Regression . . . . .	285

27.2	Perceptron	286
27.3	Accuracy	286
27.4	Feature importances	286
<b>28</b>	<b>Principal Customers Network</b>	<b>289</b>
<b>29</b>	<b>Nodes Centrality</b>	<b>293</b>
29.1	Longest path	294
29.2	Ego graph	294
<b>30</b>	<b>Graph Centrality</b>	<b>297</b>
<b>31</b>	<b>Community Detection</b>	<b>307</b>
<b>32</b>	<b>Link Prediction</b>	<b>327</b>
32.1	Predict links	330
32.2	Evaluate accuracy of link prediction algorithms	332
<b>33</b>	<b>Spatial Regression</b>	<b>341</b>
<b>34</b>	<b>Classification Models</b>	<b>343</b>
34.1	Train Classification Models	346
<b>35</b>	<b>Regression Models</b>	<b>357</b>
35.1	Forward Selection	359
35.2	Partial Least Squares Regression	361
35.3	Ridge Regression	363
35.4	Lasso Regression	365
35.5	Gradient boost	370
35.6	Random Forest	373
35.7	Feature Importances	376
35.8	RMSE's	377
<b>36</b>	<b>Deep Averaging Networks</b>	<b>379</b>
<b>37</b>	<b>Temporal Convolutional Network</b>	<b>405</b>
37.1	Temporal Convolutional Net (TCN)	408
<b>38</b>	<b>Recurrent Neural Networks</b>	<b>419</b>
38.1	LSTM	422
<b>39</b>	<b>Language Modelling</b>	<b>431</b>
39.1	Language Modelling	444
<b>40</b>	<b>Reinforcement Learning</b>	<b>445</b>



UNDER CONSTRUCTION

30+ Projects in Financial Data Science, presented as Jupyter Notebooks, using the *FinDS* Python package

## Topics

notebook	Financial	Data	Science
stock_prices jegadeesh_titman	Stock distributions, delistings Overlapping portfolios; Momentum	CRSP stocks CRSP stocks	Sample selection Hypothesis testing; Newey-West correction
fama_french	Bivariate sorts; Value, Size; CAPM	CRSP stocks; Compustat	Linear regression; Quadratic programming
fama_macbeth	Cross-sectional Regressions; Beta	Ken French data library	Feature transformations; Kernel regression, LOOCV
weekly_reversals	Mean reversion; Implementation shortfall	CRSP stocks	Structural break tests
quant_factors	Factor zoo; Performance evaluation	CRSP stocks; Compustat; IBES	Clustering for unsupervised learning
event_study	Event studies	S&P key developments	Multiple testing; FFT
economic_releases	Macroeconomic analysis; Unemployment	ALFRED	Economic data revisions
regression_diagnostics	Regression analysis; Inflation	FRED	Linear regression diagnostics; Residual Analysis
econometric_forecast	Time series analysis; National Output	FRED	Stationarity, Autocorrelation
approximate_factors	Approximate factor models	FRED-MD	Unit Root; PCA; EM Algorithm
economic_states	State space models	FRED-MD	Gaussian Mixture; HMM; Kalman Filter
conditional_volatility	Value at risk; Conditional volatility	FRED cryptos and currencies	ARCH, GARCH; VaR, TVaR
covariance_matrix	Covariance matrix estimation; Portfolio risk	Ken French data library	Shrinkage
term_structure	Interest rates, yield curve	FRED	Splines, PCA
bond_returns	Bond portfolio returns	FRED	SVD
option_pricing	Binomial trees; the Greeks	OptionMetrics; FRED	Simulations
market_microstructure	Liquidity costs; Bid-ask spreads	TAQ tick data	Realized volatility; Variance ratio
event_risk	Earnings surprises	IBES; FRED-QD	Poisson regression; GLM's
customer_ego	Principal customers	Compustat customer segments	Graph Networks
bea_centrality	Input-output use tables	Bureau of Economic Analysis	Graph centrality
industry_community	Industry sectors	Hoberg&Phillips data library	Community detection
link_prediction	Product markets	Hoberg&Phillips data library	Links prediction
spatial_regression	Earnings surprises	IBES; Hoberg&Phillips data library	Spatial regression
fomc_topics	Fedspeak	FOMC meeting minutes	Topic modelling
mda_sentiment	Company filings	SEC Edgar	Sentiment analysis
business_description	Growth and value stocks	SEC Edgar	Part-of-speech tagging
classification_models	News classification	S&P key developments	Classification for supervised learning
regression_models	Macroeconomic forecasting	FRED-MD	Regression for supervised learning
deep_classifier	News classification	S&P key developments	Feedforward neural networks; Word embeddings; Deep averaging
convolutional_net	Macroeconomic forecasting	FRED-MD	Temporal convolutional networks; Vector autoregression
recurrent_net	Macroeconomic forecasting	FRED-MD	Elman recurrent networks; Kalman filter
fomc_language reinforce- ment_learning	Fedspeak Spending policy	FOMC meeting minutes Stocks, bonds, bills, and inflation	Language modelling; Transformers Reinforcement learning

## 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>

## STOCK PRICES

### UNDER CONSTRUCTION

- Sample selection, survivorship bias
- Stock delistings and post-delisting returns
- Stock distributions, split-adjustments, dividends and stock returns

```
import numpy as np
import pandas as pd
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
from finds.database.sql import SQL
from finds.database.redisdb import RedisDB
from finds.busday import BusDay
from finds.structured.crsp import CRSP
from finds.structured.finder import Finder
from finds.plots import plot_date, plot_bar, set_xtickbins
from finds.misc.show import Show
from secret import credentials, paths, CRSP_DATE
show = Show(ndigits=4, latex=None)
VERBOSE = 0
# %matplotlib qt
```

```
sql = SQL(**credentials['sql'], verbose=VERBOSE)
rdb = RedisDB(**credentials['redis'])
bd = BusDay(sql, verbose=VERBOSE)
crsp = CRSP(sql, bd, rdb=rdb, monthly=True, verbose=VERBOSE)
imgdir = paths['images']
```

## 1.1 Sample selection

### 1.1.1 Fama-French universe

Following Fama and French (1992), the CRSP universe is trimmed to only include US-domiciled, exchange-listed stocks:

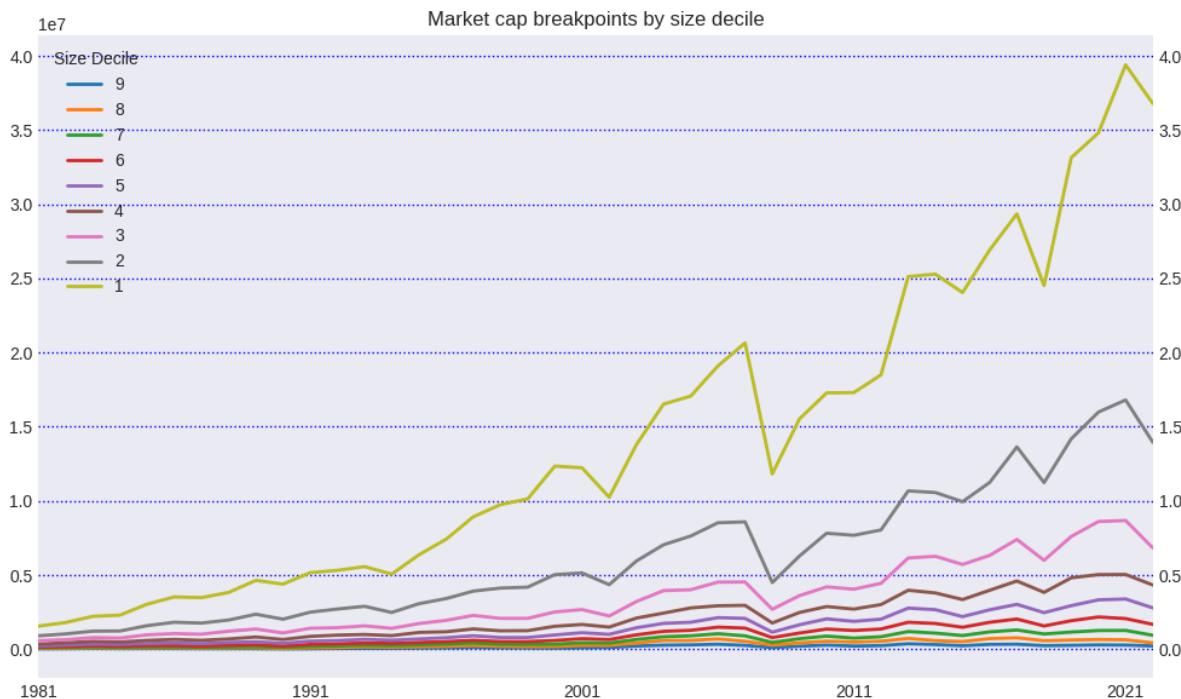
- Market cap must be available on date, with prc > 0.0
- shrcd in [10, 11], exchcd in [1, 2, 3]

The universe is often divided into 10 deciles (with the 10th decile comprising stocks with smallest market capitalization) with breakpoints determined only from those stocks listed on the NYSE. The plots below show the market cap breakpoints of and the number of stocks in each decile by year.

```
# retrieve universe of stocks annually from 1981
start = bd.endyr(19811231)
rebals = bd.date_range(start, CRSP_DATE, freq=12)
univs = {rebal: crsp.get_universe(date=rebal) for rebal in rebals}

# find market cap break points of and number of stocks in each size decile by year
cap, num = dict(), dict()
for date, univ in univs.items():
    cap[str(date//10000)] = {decile: min(univ.loc[univ['decile']==decile, 'cap'])
                             for decile in range(9, 0, -1)}
    num[str(date//10000)] = {decile: sum(univ['decile']==decile)
                            for decile in range(10, 0, -1)}
cap = DataFrame.from_dict(cap, orient='index')
num = DataFrame.from_dict(num, orient='index')
```

```
# plot market cap break points
fig, ax = plt.subplots(figsize=(10, 6))
ax.set_title('Market cap breakpoints by size decile')
cap.plot.line(ax=ax, lw=2)
ax.tick_params(labeltop=False, labelright=True)
ax.grid(visible=True, axis='y', color='b', ls=':', lw=1) # show y-axis grids
set_xtickbins(ax=ax, nbins=len(cap)//10)
plt.legend(title='Size Decile', loc='upper left')
plt.tight_layout()
```

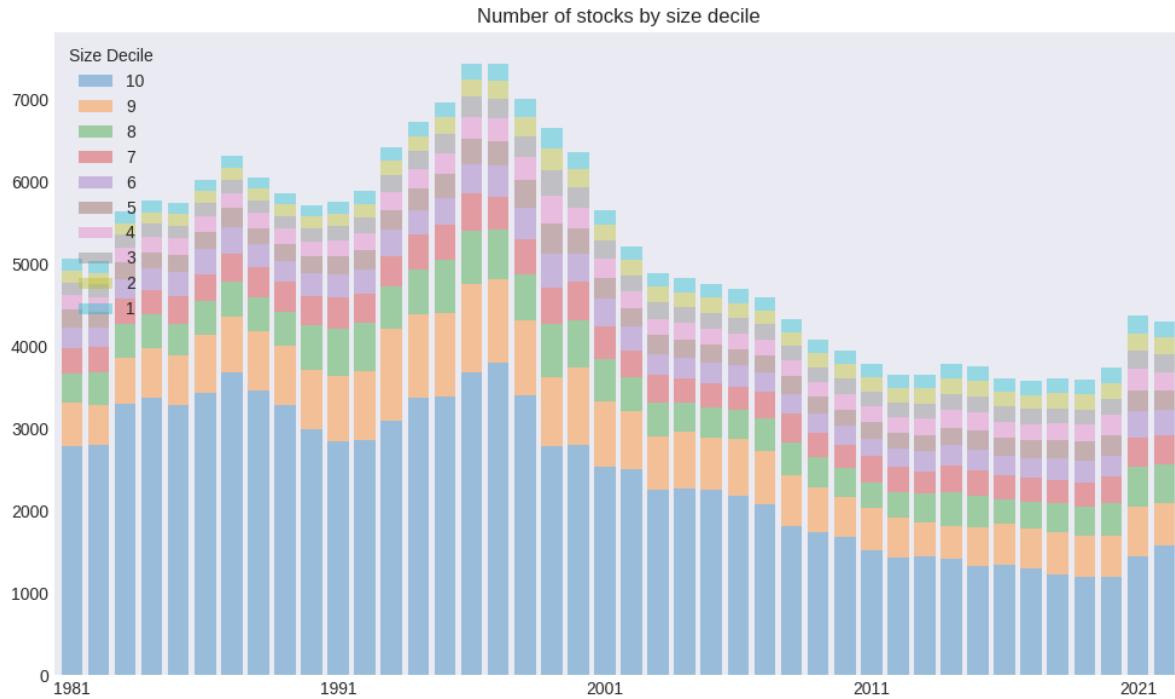


```
# plot number of stocks in each size decile
```

(continues on next page)

(continued from previous page)

```
fig, ax = plt.subplots(figsize=(10, 6))
ax.set_title('Number of stocks 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()
```

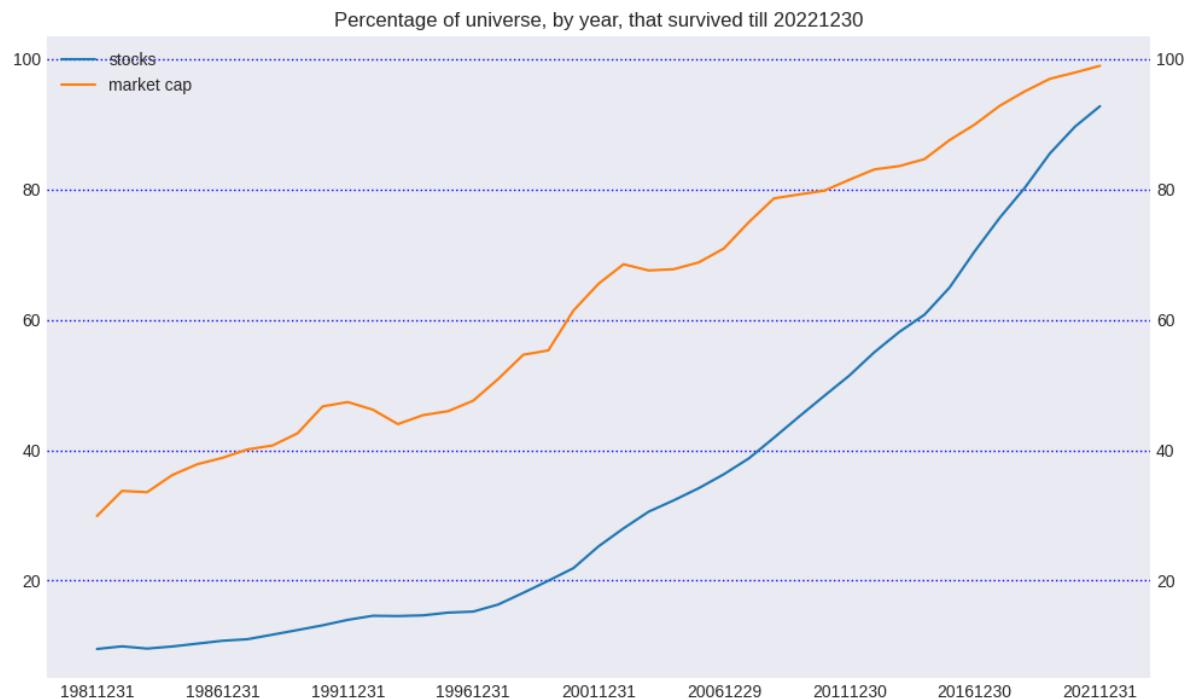


### 1.1.2 Survivorship bias

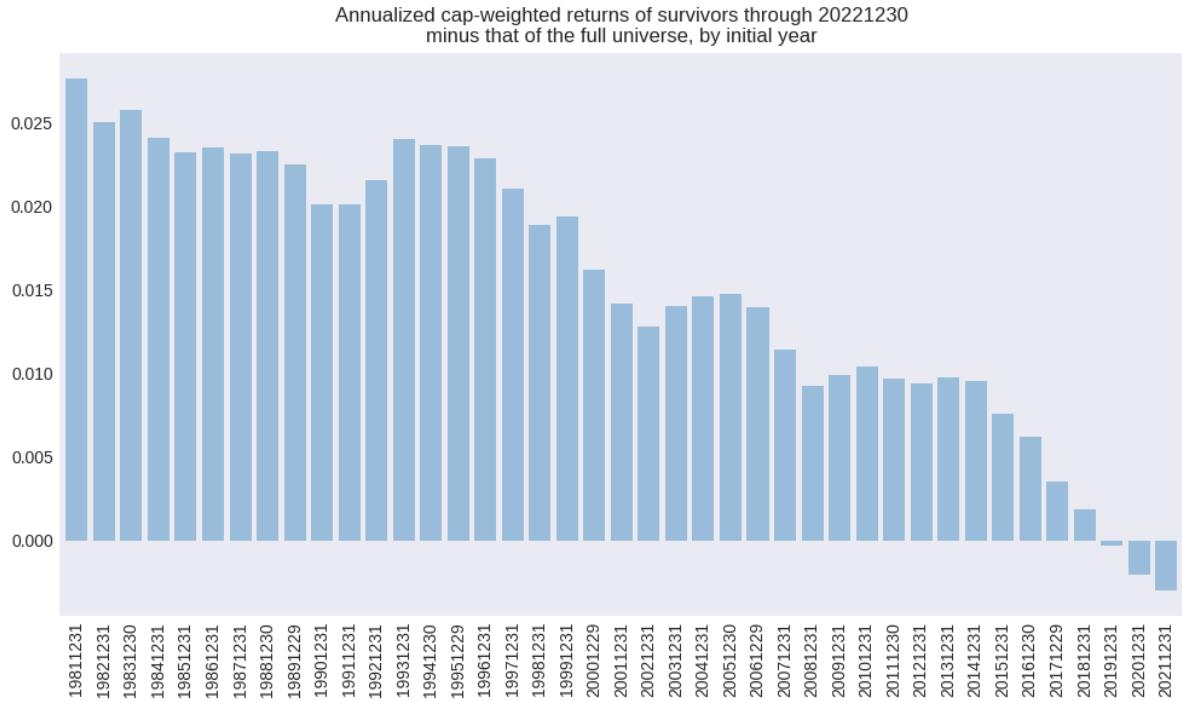
Sample bias occurs when some members of a population are systematically more likely to be selected in a sample than others. Survivorship bias is a particular form where only existing or “surviving” stocks are considered when measuring the performance of a portfolio. Stocks that have ceased to exist are not included, and therefore the portfolio’s return profile could be overestimated.

```
# compute cap-weighted returns through latest CRSP_DATE of survivors and all
diff, cap, num = dict(), dict(), dict()
for rebal in rebals[:-1]:
    univ = univs[rebal]
    nyears = np.round((CRSP_DATE - rebal) / 10000) # years elapsed
    df = crsp.get_ret(bd.offset(rebal, +1), CRSP_DATE, delist=True) \
        .reindex(univ.index) \
        .fillna(0) # include delist returns
    permnos = univ.index.intersection(univs[CRSP_DATE].index) # permnos of survivors
    total = ((1+df).dot(univ['cap']) / univ['cap'].sum())**(1/nyears)
    ret = ((1+df[permnos]).dot(univ.loc[permnos, 'cap'])) # annualized return
    / univ.loc[permnos, 'cap'].sum()**(1/nyears)
    diff[str(rebal)] = ret - total
    cap[str(rebal)] = univ.loc[permnos, 'cap'].sum() / univ['cap'].sum()
    num[str(rebal)] = len(permnos) / len(univ.index)
```

```
fig, ax = plt.subplots(figsize=(10, 6))
ax.set_title(f"Percentage of universe, by year, that survived till {CRSP_DATE}")
(DataFrame.from_dict({'stocks': num,
                     'market cap': cap},
                     orient='columns') * 100).plot.line(ax=ax)
ax.grid(visible=True, axis='y', color='b', ls=':', lw=1) # show y-axis grids
ax.tick_params(labeltop=False, labelright=True)
plt.tight_layout()
```



```
fig, ax = plt.subplots(figsize=(10, 6))
ax.set_title(f"Annualized cap-weighted returns of survivors through {CRSP_DATE}\n" +
            f"minus that of the full universe, by initial year")
Series(diff).plot.bar(ax=ax, width=.8, alpha=.4)
plt.tight_layout()
```



## 1.2 Stock delistings

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.

For CRSP Monthly, 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 to reflect the average recovery amount after delisting.

For CRSP Daily, a quick and close approximation of  $(1 + \text{DLRET}) (1 + \text{RET}) - 1$  on the last trading date can be applied.

```
avg, mab, num, frac, diff = dict(), dict(), dict(), dict(), dict()
for rebal, end in zip(rebals[:-1], rebals[1:]):
    univ = univs[rebal]
    ret = crsp.get_ret(bd.offset(rebal, +1), end, delist=False) \
        .reindex(univ.index) \
        .fillna(0)
    dlret = crsp.get_dlret(bd.offset(rebal, +1), end).reindex(univ.index)
    num[str(rebal)] = (~dlret.isna()).sum()
    frac[str(rebal)] = num[str(rebal)] / len(ret)
    diff[str(rebal)] = (1 + dlret[~dlret.isna().values]).to_list()
    avg[str(rebal)] = dlret[~dlret.isna().values].mean()
    mab[str(rebal)] = dlret[~dlret.isna().values].abs().mean()
```

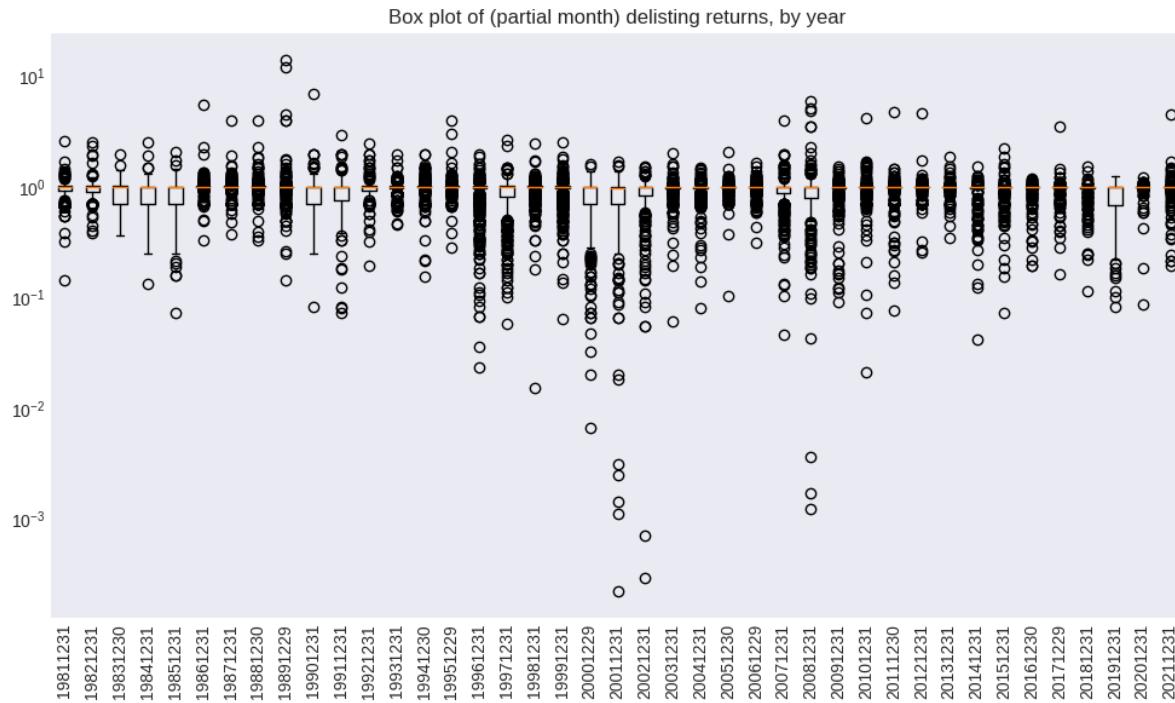
### Box plot of the distribution of delisting returns

```
fig, ax = plt.subplots(figsize=(10, 6))
ax.set_title("Box plot of (partial month) delisting returns, by year")
ax.boxplot(list(diff.values()), labels=list(diff.keys()))
```

(continues on next page)

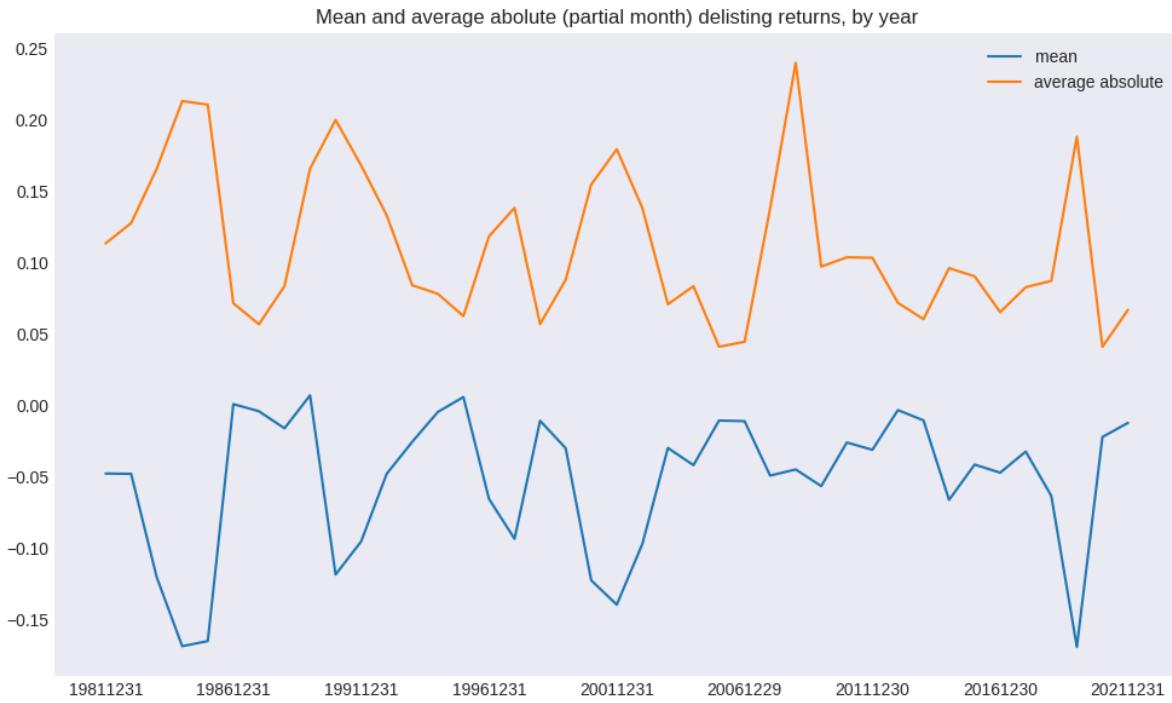
(continued from previous page)

```
ax.set_yscale('log')
ax.tick_params(axis='x', rotation=90) # set tick rotation
plt.tight_layout()
```



### Plot of the mean, and average absolute delisting returns

```
fig, ax = plt.subplots(figsize=(10, 6))
ax.set_title("Mean and average absolute (partial month) delisting returns, by year")
Series(avg).rename('mean').plot(ax=ax)
Series(mab).rename('average absolute').plot(ax=ax)
plt.legend()
plt.tight_layout()
```



```
fig, ax = plt.subplots(figsize=(10, 6))
ax.set_title("Number and fraction of delisted stocks, by year")
Series(num).rename('number').plot(ax=ax, color="C0")
ax.legend(loc='upper left')
ax.tick_params(axis='y', colors='C0')
bx = ax.twinx()
Series(frac).rename('fraction').plot(ax=bx, color="C1")
bx.tick_params(axis='y', colors='C1')
bx.legend(loc='upper right')
plt.tight_layout()
```



### 1.3 Stock distributions and returns

```
ticker = 'AAPL'
find = Finder(sql)          # to search identifier lookup tables
names = find(ticker)        # locate names records by ticker
show(names)
```

	date	comnam	ncusip	shrccls	ticker	permno	nameendt	permco
0	19821101	APPLE COMPUTER INC	03783310		AAPL	14593	20040609	
1	20040610	APPLE COMPUTER INC	03783310		AAPL	14593	20070110	
2	20070111	APPLE INC	03783310		AAPL	14593	20171227	
3	20171228	APPLE INC	03783310		AAPL	14593	20221230	

	shrcd	exchcd	siccd	tsymbol	naics	primexch	trdstat	secstat	permco
0	11	3	3573	AAPL	0	Q	A	R	7
1	11	3	3573	AAPL	334111	Q	A	R	7
2	11	3	3571	AAPL	334111	Q	A	R	7
3	11	3	3571	AAPL	334220	Q	A	R	7

```
# Get raw price and distributions history from CRSP Daily
where = f" where permno = {names['permno'].iloc[-1]} "
dist = sql.read_dataframe(f"select * from {crsp['dist'].key} {where}")
crsp_df = sql.read_dataframe(f"select * from {crsp['daily'].key} {where}\n    .set_index('date', inplace=False)
crsp_df
crsp_columns = crsp_df.columns
```

```
# Retrieve price history from yahoo finance
from yahoo import get_price
yahoo_df = get_price(ticker, start_date='19800101', verbose=VERBOSE)
yahoo_df.rename_axis(ticker)
yahoo_columns = yahoo_df.columns
```

```
cookies True crumb None
https://finance.yahoo.com/quote/^GSPC
```

```
# Merge yahoo and CRSP by date
df = crsp_df[['prc', 'ret', 'retx']] \
    .join(dist[dist['facpr'] != 0.0].set_index('exdt')['facpr']) \
    .join(dist[dist['divamt'] != 0.0].set_index('exdt')['divamt']) \
    .join(yahoo_df[['close', 'adjClose']], how='inner')
```

## Price Appreciation

The Factor to Adjust Price (*facpr*) can be used to adjust prices after distributions (usually stock dividends and splits) so that stock prices before and after the distribution are comparable. The compounded factor between two dates can be divided into raw stock prices to derive comparable split-adjusted prices.

```
# Cumulate factor to adjust pre-split prices prior to ex-date
facpr = df['facpr'].shift(-1).fillna(0) + 1
facpr = facpr.sort_index(ascending=False).cumprod()
```

```
# Split-adjust CRSP closing prices
df['price'] = df['prc'].abs().div(facpr)
```

## Holdings Returns

In addition to stock price appreciation, investors' total holding returns also include ordinary cash dividends (*divamt*). The dividend yield is obtained by dividing by the prior day's closing price, which can be compounded with the price adjustment factor to adjust raw stock prices for both splits and dividends received, the change in which then reflects total holding returns earned by an investor.

```
# Cumulate dividend factor to adjust stock prices on and prior to ex-date
divyld = df['divamt'].div(df['prc'].shift(1)).fillna(0) + 1
divyld = divyld.sort_index(ascending=False).cumprod()
```

```
# Dividend- and split-adjust CRSP closing prices
# (rescale to 20221231 Yahoo prices because 2023 CRSP dividends not available)
df['adjPrice'] = (df['prc'].abs().div(facpr * divyld) \
                  * (df.iloc[-1]['adjClose'] / df.iloc[-1]['close']))
```

## Adjusted closing prices

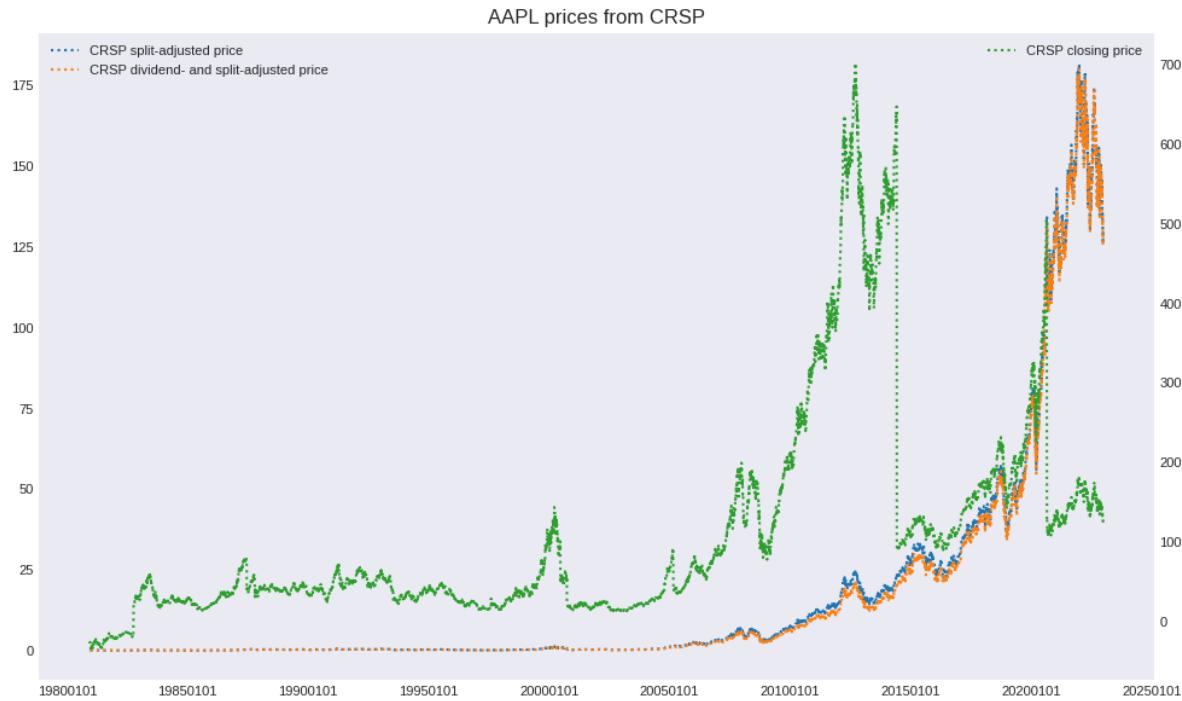
- The data item which Yahoo Finance calls the close “close” is the CRSP split-adjusted price
- The Yahoo Finance “adjclose” is the CRSP dividend- and split-adjusted: it reflects the total holdings returns that includes the effect of both stock splits and cash dividends received

```
fig, ax = plt.subplots(figsize=(10, 6))
plot_date(df[['price', 'adjPrice']], df['prc'],
          legend1=['CRSP split-adjusted price',
```

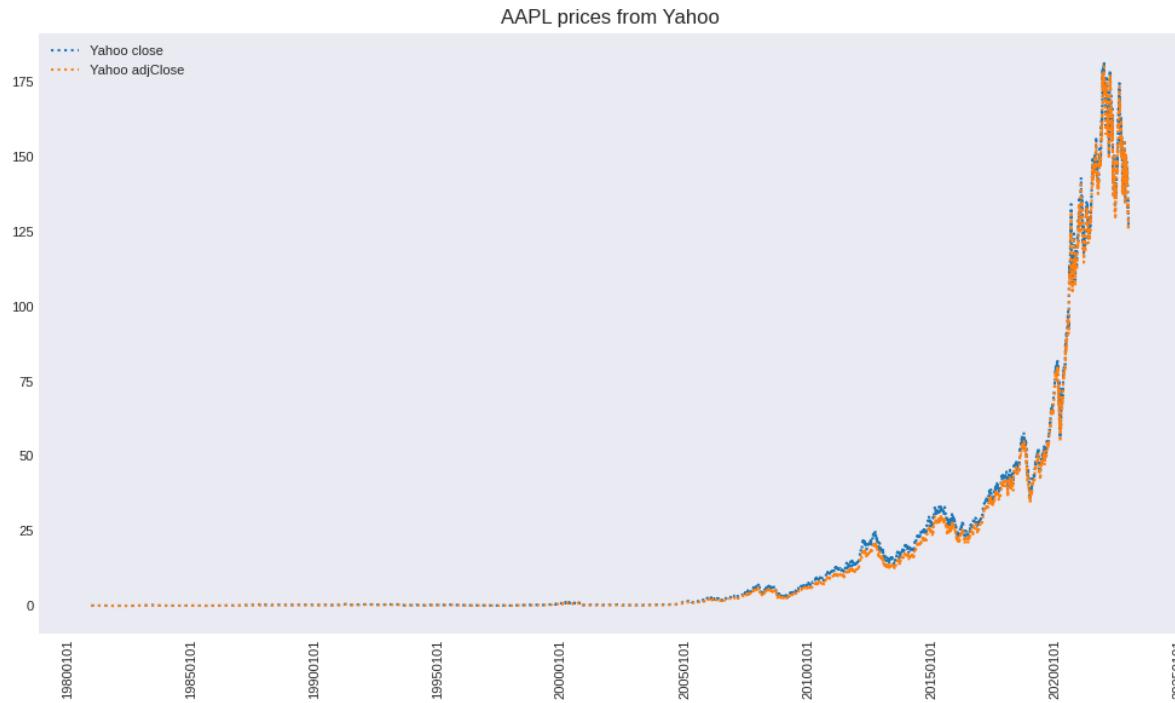
(continues on next page)

(continued from previous page)

```
'CRSP dividend- and split-adjusted price'],
legend2=['CRSP closing price'], title=f'{ticker} prices from CRSP',
ls=':', fontsize=8, rotation=0, cn=0, ax=ax)
plt.tight_layout()
```



```
fig, ax = plt.subplots(figsize=(10, 6))
plot_date(df[['close', 'adjClose']], legend1=['Yahoo close', 'Yahoo adjClose'],
          title=f'{ticker} prices from Yahoo',
          ls=':', fontsize=8, rotation=90, cn=0, ax=ax)
plt.tight_layout()
```



```
rets = DataFrame(data={
    'CRSP returns': [(df['retx'].iloc[1:] + 1).prod(),
                      (df['ret'].iloc[1:] + 1).prod()],
    'CRSP prices': [df['price'].iloc[-1] / df['price'].iloc[0],
                    df['adjPrice'].iloc[-1] / df['adjPrice'].iloc[0]],
    'Yahoo prices': [df['close'].iloc[-1] / df['close'].iloc[0],
                     df['adjClose'].iloc[-1] / df['adjClose'].iloc[0]],
}, index=['Total Price Appreciation', 'Total Holding Returns'])
show(rets, caption=f"{ticker} stock returns with and without dividends:")
```

	CRSP returns	CRSP prices
AAPL stock returns with and without dividends:		
Total Price Appreciation	1065.5725	1065.6044 \
Total Holding Returns	1369.8131	1368.1319
Yahoo prices		
AAPL stock returns with and without dividends:		
Total Price Appreciation	1068.0465	
Total Holding Returns	1372.5507	

## 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.



## JEGADEESH-TITMAN ROLLING PORTFOLIOS

### UNDER CONSTRUCTION

- Momentum effect
- Univariate spread portfolios
- Overlapping returns, Newey-West auto-correlation correction
- Hypothesis testing

```
import math
import numpy as np
import pandas as pd
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
from scipy.stats import kurtosis, skew
import statsmodels.formula.api as smf
import statsmodels.api as sm
from tqdm import tqdm
from finds.database import SQL, RedisDB
from finds.busday import BusDay
from finds.structured import CRSP
from finds.backtesting import fractiles
from finds.misc import Show
from finds.plots import plot_date
from secret import credentials, paths
show = Show(ndigits=4, latex=None)
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)
imgdir = paths['images']
```

```
# parameters for momentum portfolios
begrebal = 19260630 # first price date is 19251231
endrebal = 20221130
rebaldates = bd.date_range(begrebal, endrebal, 'endmo')
percentiles = [20, 80] # quintile spread percentile breakpoints
maxhold = 6 # hold each monthly-rebalanced portfolio for 6 months
```

## 2.1 Univariate sorts

### 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.

```

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),
          crsp.get_ret(start, end).rename('mom'),
          crsp.get_section('monthly', ['prc'], 'date', beg)[['prc']].rename('beg')]
    df = pd.concat(p, axis=1, join='inner').dropna()

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

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

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

    mom.append(float(ret) / nhold)
DataFrame({'mean': np.mean(mom), 'std': np.std(mom)},
          index=['Overlapping Returns'])

```

100% |██████████| 1158/1158 [14:26<00:00, 1.34it/s]

	mean	std
Overlapping Returns	0.004599	0.024366

## 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 = []
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),
          crsp.get_ret(start, end).rename('mom'),
          crsp.get_section('monthly', ['prc'], 'date', beg)['prc'].rename('beg')]
    df = pd.concat(p, axis=1, join='inner').dropna()

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

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

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

    # compute all 6 portfolios' monthly capwtd returns, and store eqlwtd average
    begret = bd.offset(rebaldate, 1)
    endret = bd.endmo(begret)
    rets = crsp.get_ret(begret, endret, delist=True)
    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 = crsp.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'])

```

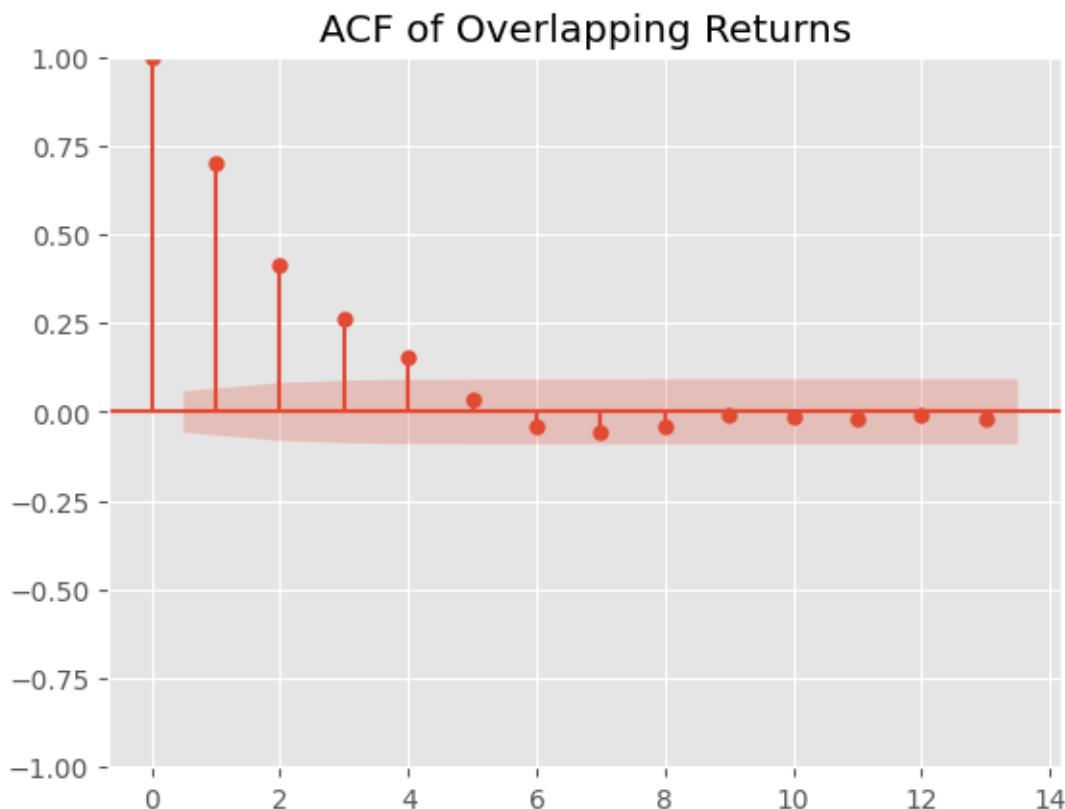
100% |██████████| 1158/1158 [14:43<00:00, 1.31it/s]

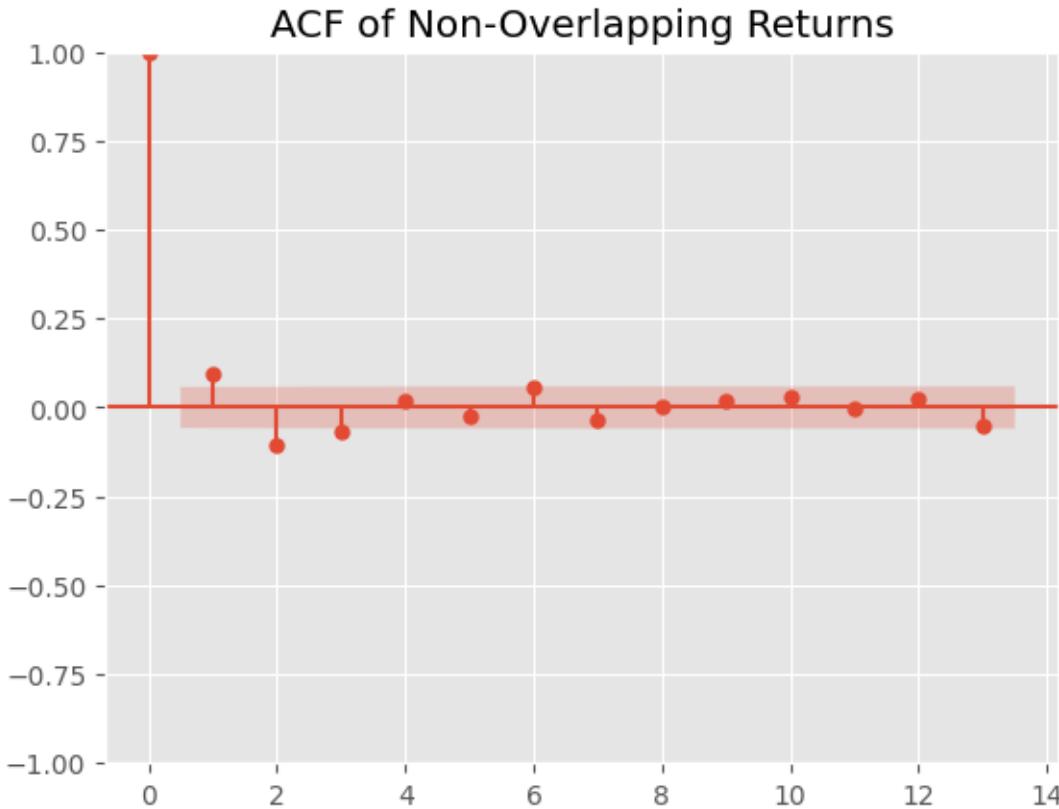
	mean	std
Non-overlapping Returns	0.004563	0.05115

### Plot autocorrelations of portfolio returns

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

```
sm.graphics.tsa.plot_acf(mom,
                         lags=13,
                         title='ACF of Overlapping Returns')
plt.savefig(imgdir / 'overlap.jpg')
sm.graphics.tsa.plot_acf(jt,
                         lags=13,
                         title='ACF of Non-Overlapping Returns')
plt.savefig(imgdir / 'nonoverlap.jpg')
```





## 2.2 Newey-West correction

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)))
res = []
keys = ['-- Overlapping --', '-- Non-overlapping --']
for out, label in zip([mom, jt], keys):
    data = DataFrame(out, columns=['ret'])

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

```

(continues on next page)

(continued from previous page)

```

for stat in ['params', 'bse', 'tvalues', 'pvalues'],  

    name='NeweyWest')    # coef, stderr, t-value, P>/z/  
  

# merge into intermediate dataframe with multicolm index  

c = pd.concat([a, b], axis=1)  

c.columns = pd.MultiIndex.from_product([[label], c.columns])  

res.append(c)  
  

show(pd.concat(res, axis=1),  

    caption='Uncorrected and Newey-West standard errors')

```

n = 1158      L = 6

-- Overlapping --		
uncorrected   NeweyWest		
Uncorrected and Newey-West standard errors		
params	0.0046	0.0046 \
bse	0.0007	0.0013
tvalues	6.4204	3.5748
pvalues	0.0000	0.0004
-- Non-overlapping --		
uncorrected   NeweyWest		
Uncorrected and Newey-West standard errors		
params	0.0046	0.0046
bse	0.0015	0.0015
tvalues	3.0345	3.1066
pvalues	0.0025	0.0019

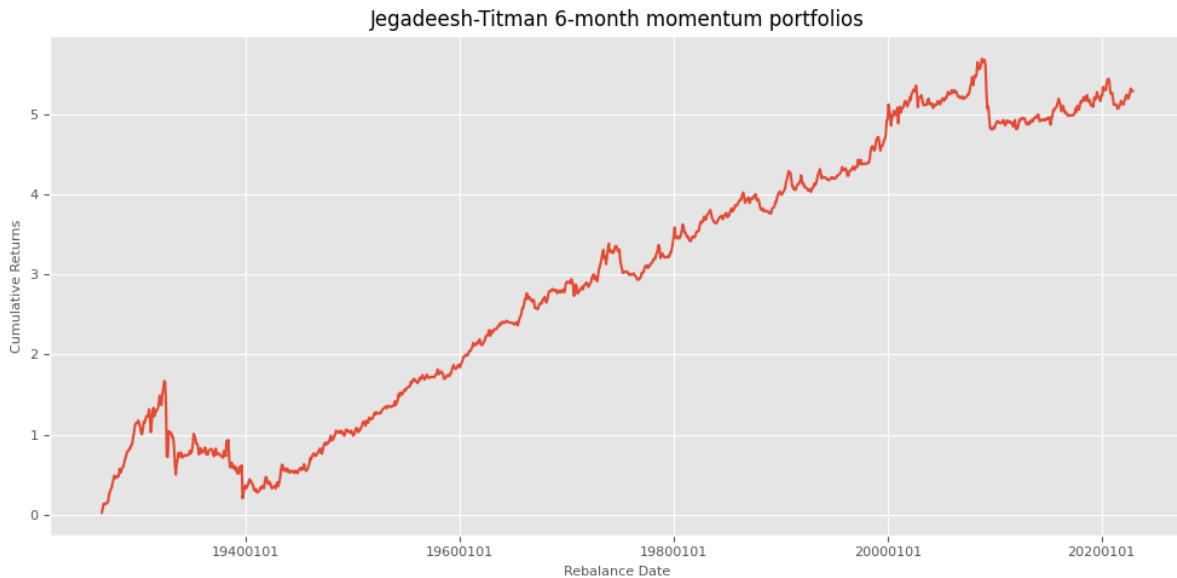
### Plot cumulative monthly average returns

- non-overlapping Jegadeesh-Titman 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')

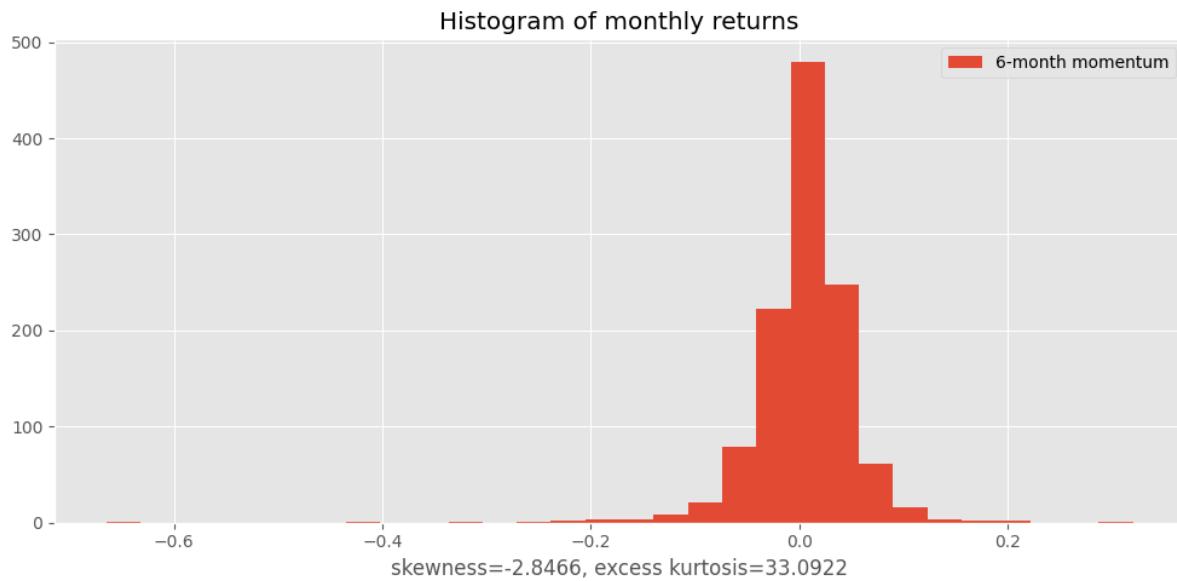
```



### Plot histogram of monthly returns

- non-overlapping Jegadeesh-Titman 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')
```



## 2.3 Hypothesis Testing

- The *null hypothesis* specifies the true value of a parameter to be tested, e.g.  $H_0 : \hat{\mu} = \mu_0$
- 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*  $C_\alpha$  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*  $\alpha$  if its *size* is less than or equal to  $\alpha$ .
- 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.

### 2.3.1 Confidence Interval

- A  $1 - \alpha$  *confidence interval* contains the values surrounding the test statistic that cannot be rejected when using a test size of  $\alpha$ , 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

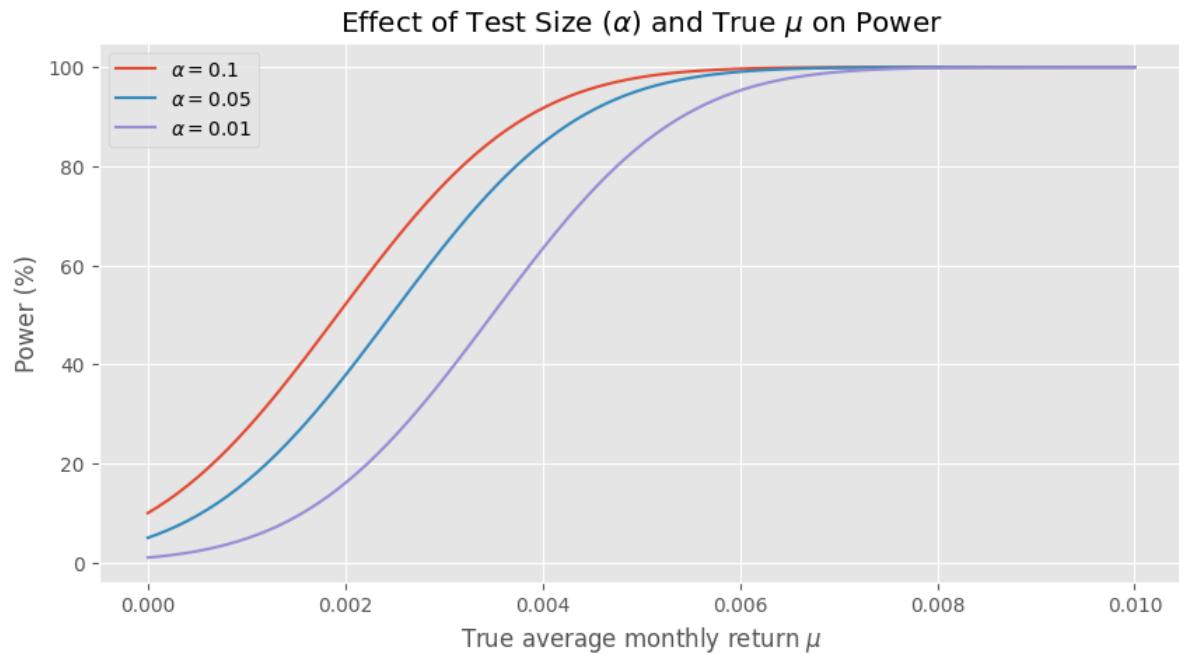
### 2.3.2 Power of Test

- 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 error is denoted by  $\beta$ , 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  $H_a : \hat{\mu} > \mu_0$ .

```
from scipy.stats import norm
alternative = np.linspace(0, 0.01, 100)      # up to annualized mean = 0.01*12 = 12%
sigma = 0.0015    # with n=1158, annualized std dev = 0.0015*sqrt(1128 x 12) = 17.7%
plt.figure(figsize=(10,5))
for alpha in [0.1, 0.05, 0.01]:
    power = 1 - norm.cdf(norm.ppf(1-alpha)*sigma, loc=alternative, scale=sigma)
    plt.plot(alternative, 100*power, label=f"$\alpha={alpha}$")
plt.title("Effect of Test Size ($\alpha$) and True $\mu$ on Power")
plt.ylabel('Power (%)')
plt.xlabel('True average monthly return $\mu$')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f2c68908850>
```





## FAMA-FRENCH PORTFOLIO SORTS

### UNDER CONSTRUCTION

- Value and size effect
- Bivariate portfolio sorts
- Fama-French research factors: HML, SMB, Mom, STRev
- Linear Regression
- CAPM and performance measurement

```
import numpy as np
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 Signals, Benchmarks, CRSP, PSTAT
from finds.busday import BusDay
from finds.backtesting import fractiles, bivariate_sorts, BackTest
from finds.plots import plot_date, plot_scatter, plot_hist
from finds.misc import Show
from tqdm import tqdm
from secret import credentials, paths, CRSP_DATE
```

```
show = Show(ndigits=4, latex=None)
VERBOSE = 0
# %matplotlib qt
```

```
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)
signals = Signals(user, verbose=VERBOSE)
bench = Benchmarks(sql, bd, verbose=VERBOSE)
backtest = BackTest(user, bench, 'RF', CRSP_DATE, verbose=VERBOSE)
LAST_DATE = bd.endmo(CRSP_DATE, -1) # last monthly rebalance date
```

### Construct HML

- 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

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

```
# subtract preferred stock, add back deferred investment tax credit
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])
df[label] = df['seq'] + df['txditz'].fillna(0) - df[label]
df.dropna(subset = [label], inplace=True)
df = df[df[label] > 0][['permno', 'gvkey', 'datadate', label]]
```

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

```
# construct b/m ratio
df['rebaldate'] = 0
for datadate in tqdm(sorted(df['datadate'].unique())):
    f = df['datadate'].eq(datadate)
    rebaldate = crsp.bd.endmo(datadate, abs(lag)) # 6 month lag
    capdate = crsp.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).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% |██████████| 734/734 [00:02<00:00, 254.00it/s]

219664

### 3.1 Bivariate portfolio sorts

```
label, benchname = 'hml', 'HML (mo)'
rebalend = LAST_DATE
rebalbeg = 19640601
holdings, smb, sizes = bivariate_sorts(stocks=crsp,
                                         label=label,
                                         signals=signals,
                                         rebalbeg=rebalbeg,
                                         rebalend=rebalend,
                                         window=12,
                                         months=[6])
size = {b: [int(np.mean(list(sizes[b[0]+s[0]].values()))) for s in "SB"]
        for b in ["High b/m", "Medium b/m", "Low b/m"]}
show(DataFrame(size, index=["Small size", "Big size"]),
      caption="Average number of stocks in bivariate-sorted subportfolios")
```

	High b/m	Medium b/m	
Average number of stocks in bivariate-sorted su...			
Small size	920	838	\
Big size	140	298	
	Low b/m		
Average number of stocks in bivariate-sorted su...			
Small size	666		
Big size	358		

#### Helpers to show histograms and comparisons of portfolio returns

```
def plot_ff(y, label, savefig=None):
    """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=" 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)
    if savefig:
        plt.savefig(imgdir / savefig)
    print(f"<Correlation of {label} vs {benchname}>\n"
          f" ({y.index[0]} - {y.index[-1]}): {corr:.4f}")
```

```
def plot_summary(y, label, savefig=None):
    """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 returns ({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()
```

(continues on next page)

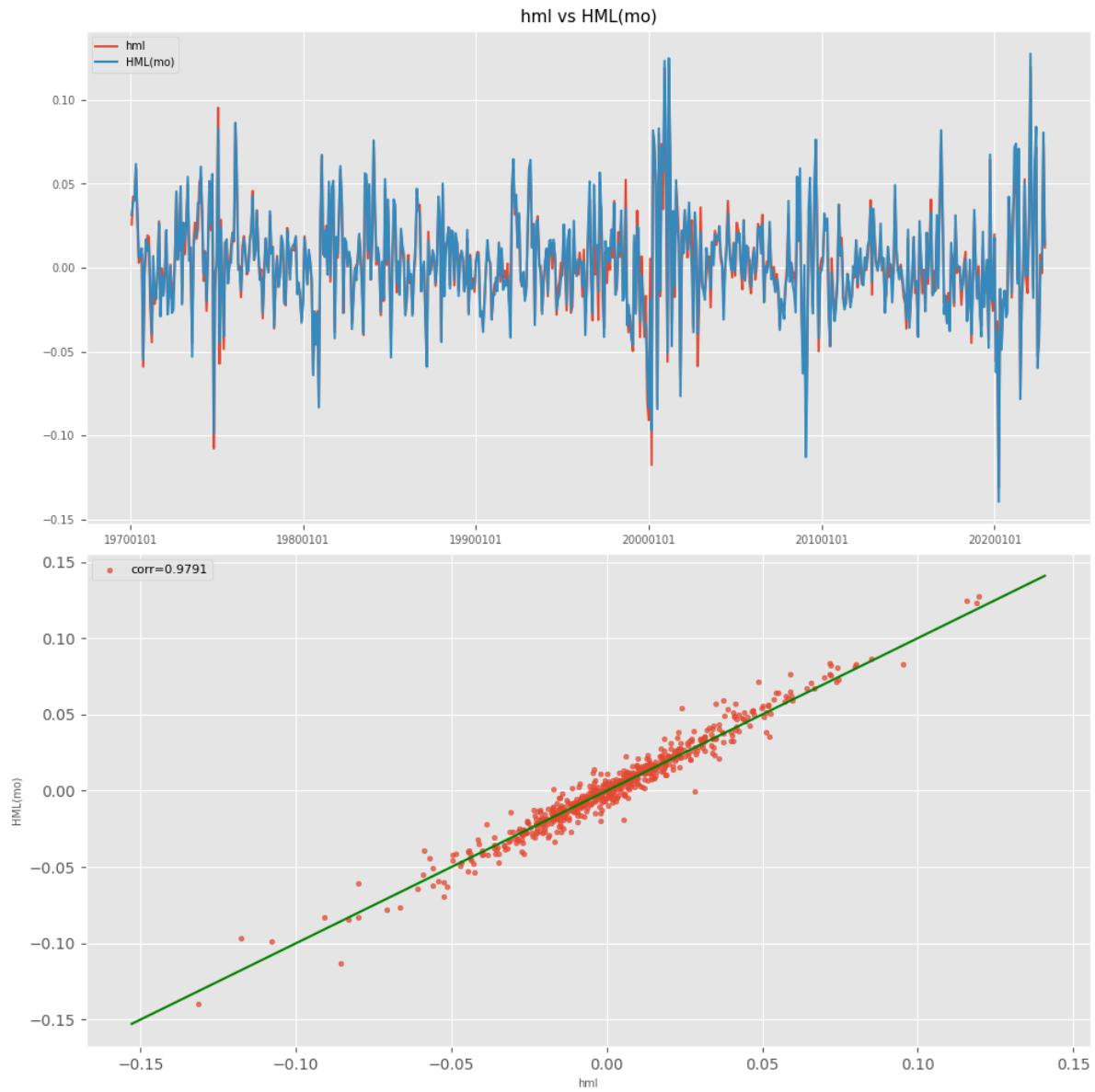
(continued from previous page)

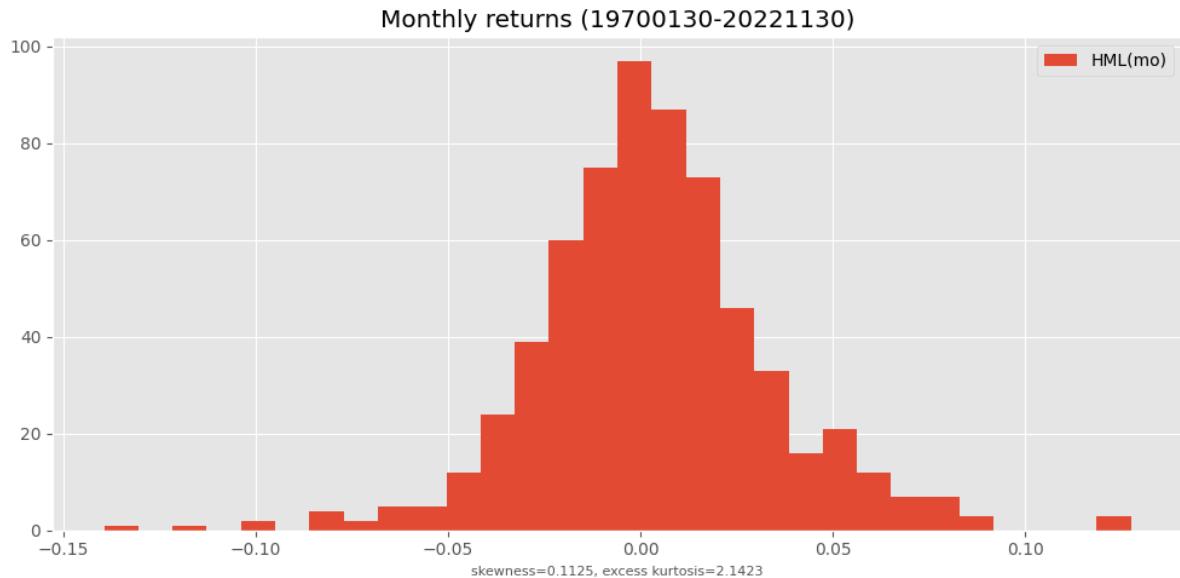
```
if savefig:  
    plt.savefig(imgdir / savefig)
```

```
# Plot histogram and comparison of HML returns  
result = backtest(crsp, holdings, label)  
y = backtest.fit([benchname], 19700101, LAST_DATE)  
plot_ff(y, label, savefig=imgdir / (label + '.jpg'))  
plot_summary(y, benchname, savefig=imgdir / (label + '_hist.jpg'))
```

```
/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
```

```
<Correlation of hml vs HML(mo) (19700130 - 20221130): 0.9791
```





```
# Linear regression on Mkt-Rf and intercept
x = bench.get_series('mkt-rf(mo)', beg=y.index[0], end=y.index[-1])
lm = smf.ols(f'Q("{benchname}")~Q("{x.name}")', data=pd.concat([y, x], axis=1))\
    .fit(cov_type='HAC', cov_kwds={'maxlags': 6})
print(f"Period: {rebalbeg}-{rebalend}")
print(lm.summary())
```

Period: 19640601-20221130

OLS Regression Results

Dep. Variable:	Q("HML(mo)")	R-squared:	0.048			
Model:	OLS	Adj. R-squared:	0.047			
Method:	Least Squares	F-statistic:	8.426			
Date:	Thu, 31 Aug 2023	Prob (F-statistic):	0.00383			
Time:	07:36:10	Log-Likelihood:	1325.0			
No. Observations:	635	AIC:	-2646.			
Df Residuals:	633	BIC:	-2637.			
Df Model:	1					
Covariance Type:	HAC					
	coef	std err	z	P> z	[0.025	0.975]
Intercept	0.0041	0.002	2.576	0.010	0.001	0.007
Q("mkt-rf(mo)")	-0.1464	0.050	-2.903	0.004	-0.245	-0.048
Omnibus:	47.215	Durbin-Watson:	1.656			
Prob(Omnibus):	0.000	Jarque-Bera (JB):	211.956			
Skew:	-0.021	Prob(JB):	9.43e-47			
Kurtosis:	5.830	Cond. No.	21.7			

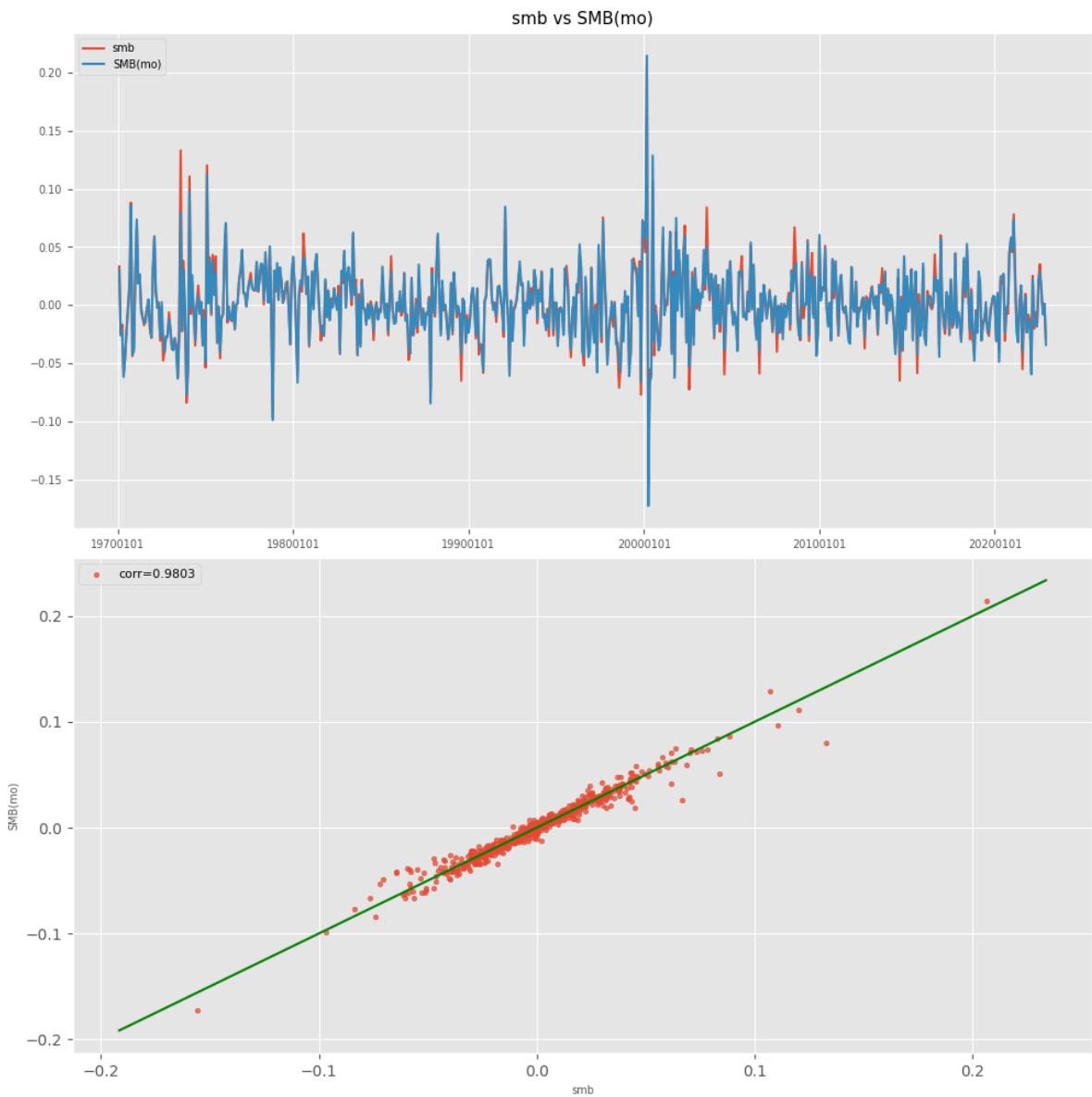
Notes:

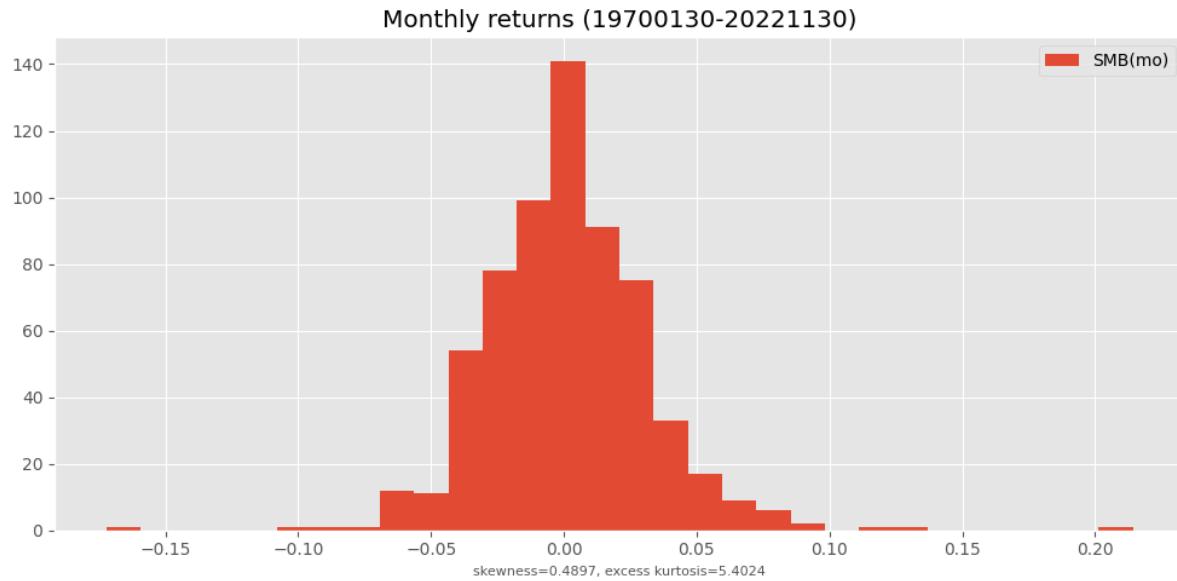
[1] Standard Errors are heteroscedasticity and autocorrelation robust (HAC) using 6 lags and without small sample correction

```
# Plot histogram and comparison of SMB returns
label, benchname = 'smb', 'SMB(mo)'
holdings = smb
result = backtest(crsp, holdings, label)
y = backtest.fit([benchname], 19700101, LAST_DATE)
plot_ff(y, label, savefig=imgdir / (label + '.jpg'))
plot_summary(y, benchname, savefig=imgdir / (label + '_hist.jpg'))
```

```
/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
```

<Correlation of smb vs SMB(mo) (19700130 - 20221130): 0.9803





```
# Linear regression on Mkt-Rf and intercept
x = bench.get_series('mkt-rf(mo)', beg=y.index[0], end=y.index[-1])
lm = smf.ols(f'Q("{benchname}") ~ Q("{x.name}")', data=pd.concat([y, x], axis=1))\
    .fit(cov_type='HAC', cov_kwds={'maxlags': 6})
print(f"Period: {rebalbeg}-{rebalend}")
print(lm.summary())
```

Period: 19640601-20221130

OLS Regression Results

Dep. Variable:	Q("SMB(mo)")	R-squared:	0.077			
Model:	OLS	Adj. R-squared:	0.075			
Method:	Least Squares	F-statistic:	42.24			
Date:	Thu, 31 Aug 2023	Prob (F-statistic):	1.63e-10			
Time:	07:36:25	Log-Likelihood:	1337.5			
No. Observations:	635	AIC:	-2671.			
Df Residuals:	633	BIC:	-2662.			
Df Model:	1					
Covariance Type:	HAC					
	coef	std err	z	P> z	[0.025	0.975]
Intercept	0.0001	0.001	0.100	0.920	-0.002	0.002
Q("mkt-rf(mo)")	0.1839	0.028	6.499	0.000	0.128	0.239
Omnibus:	113.510	Durbin-Watson:	2.078			
Prob(Omnibus):	0.000	Jarque-Bera (JB):	1030.412			
Skew:	0.485	Prob(JB):	1.77e-224			
Kurtosis:	9.165	Cond. No.	21.7			

Notes:

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

### Construct MOM

```
# 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 = 19270101, 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 = [crsp.get_universe(rebaldate),      # retrieve prices and construct signal
          crsp.get_ret(start, end).rename(label),
          crsp.get_section('monthly', ['prc'], 'date', beg)[['prc']].rename('beg'),
          crsp.get_section('monthly', ['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)
```

0% | 0/1151 [00:00<?, ?it/s]

100% |██████████| 1151/1151 [32:31<00:00, 1.70s/it]

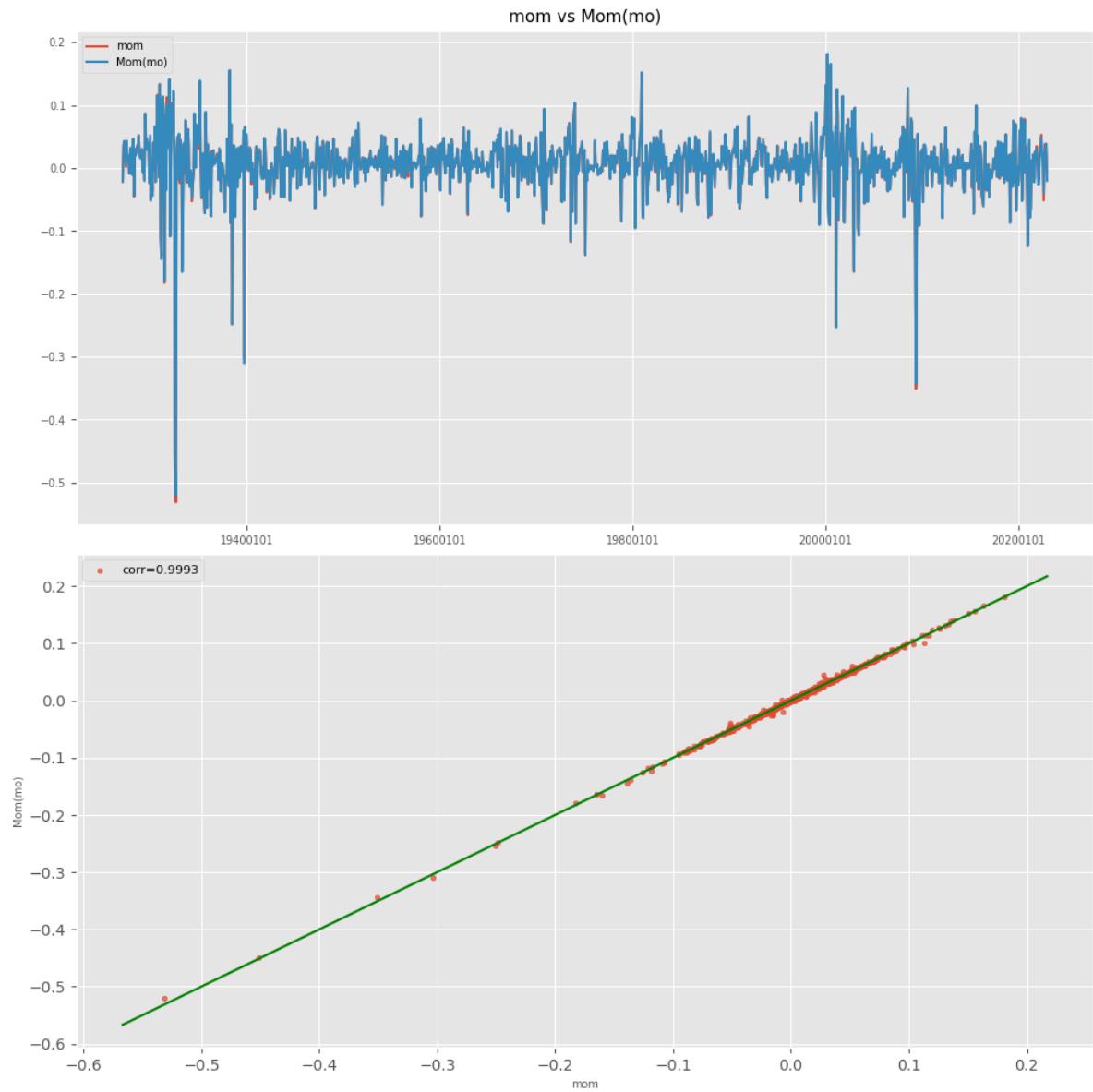
3357366

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

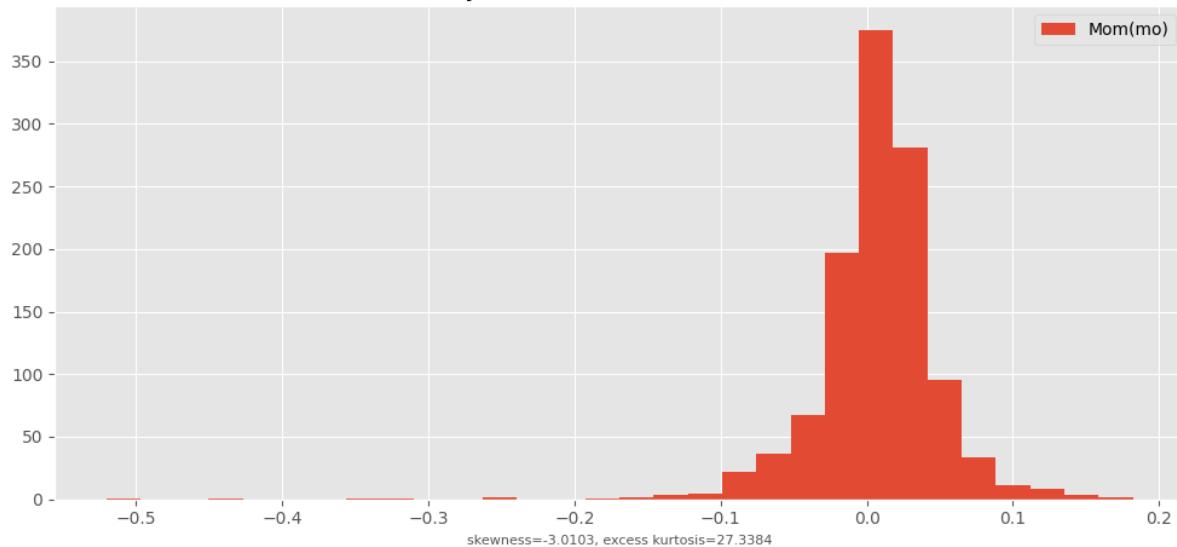
```
# plot histogram and comparison of returns
plot_ff(y, label, savefig=imgdir / (label + '.jpg'))
plot_summary(y, benchname, savefig=imgdir / (label + '_hist.jpg'))
```

```
/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
```

<Correlation of mom vs Mom(mo) (19270228 - 20221130): 0.9993



Monthly returns (19270228-20221130)



```
# Linear regression on Mkt-Rf and intercept
x = bench.get_series('mkt-rf(mo)', beg=y.index[0], end=y.index[-1])
lm = smf.ols(f'Q("{benchname}") ~ Q("{x.name}")', data=pd.concat([y, x], axis=1))\
    .fit(cov_type='HAC', cov_kwds={'maxlags': 6})
print(f"Period: {rebalbeg}-{rebalend}")
print(lm.summary())
```

Period: 19270101-20221130

OLS Regression Results

Dep. Variable:	Q("Mom(mo)")	R-squared:	0.118			
Model:	OLS	Adj. R-squared:	0.117			
Method:	Least Squares	F-statistic:	10.16			
Date:	Thu, 31 Aug 2023	Prob (F-statistic):	0.00148			
Time:	08:12:39	Log-Likelihood:	1958.6			
No. Observations:	1150	AIC:	-3913.			
Df Residuals:	1148	BIC:	-3903.			
Df Model:	1					
Covariance Type:	HAC					
	coef	std err	z	P> z	[0.025	0.975]
Intercept	0.0085	0.001	7.572	0.000	0.006	0.011
Q("mkt-rf(mo)")	-0.3009	0.094	-3.187	0.001	-0.486	-0.116
Omnibus:	612.170	Durbin-Watson:	1.926			
Prob(Omnibus):	0.000	Jarque-Bera (JB):	12217.497			
Skew:	-2.003	Prob(JB):	0.00			
Kurtosis:	18.457	Cond. No.	18.6			

Notes:

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

## Construct STRev

```

# 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 = [crsp.get_universe(rebaldate),
          crsp.get_section('monthly', ['prc'], 'date', beg)['prc'].rename('beg'),
          crsp.get_section('monthly', ['prc'], 'date', end)['prc'].rename('end'),
          crsp.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)

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

# plot histogram and comparison of returns
plot_ff(y, label, savefig=imgdir / (label + '.jpg'))
plot_summary(y, benchname, savefig=imgdir / (label + '_hist.jpg'))

# Linear regression on Mkt-Rf and intercept
x = bench.get_series('mkt-rf(mo)', beg=y.index[0], end=y.index[-1])
lm = smf.ols(f'Q("{benchname}")~Q("{x.name}")', data=pd.concat([y, x], axis=1))\
    .fit(cov_type='HAC', cov_kwds={'maxlags': 6})
print(f"Period: {rebalbeg}-{rebalend}")
print(lm.summary())

```

```

100%|██████████| 1163/1163 [30:36<00:00,  1.58s/it]
/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

```

```

<Correlation of strev vs ST_Rev(mo) (19260227 - 20221130): 0.9981
Period: 19260101-20221130

```

```
OLS Regression Results
```

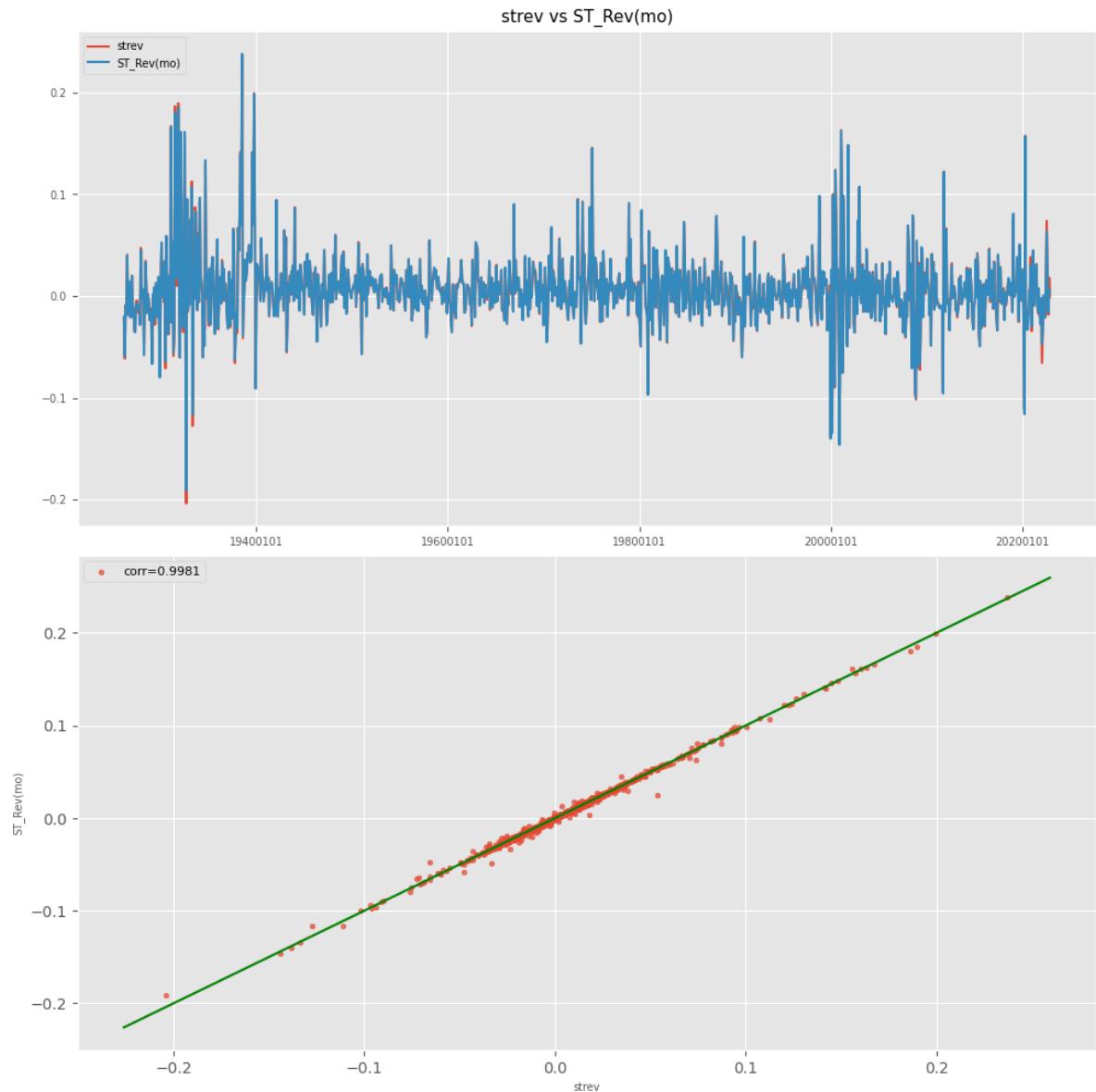
(continues on next page)

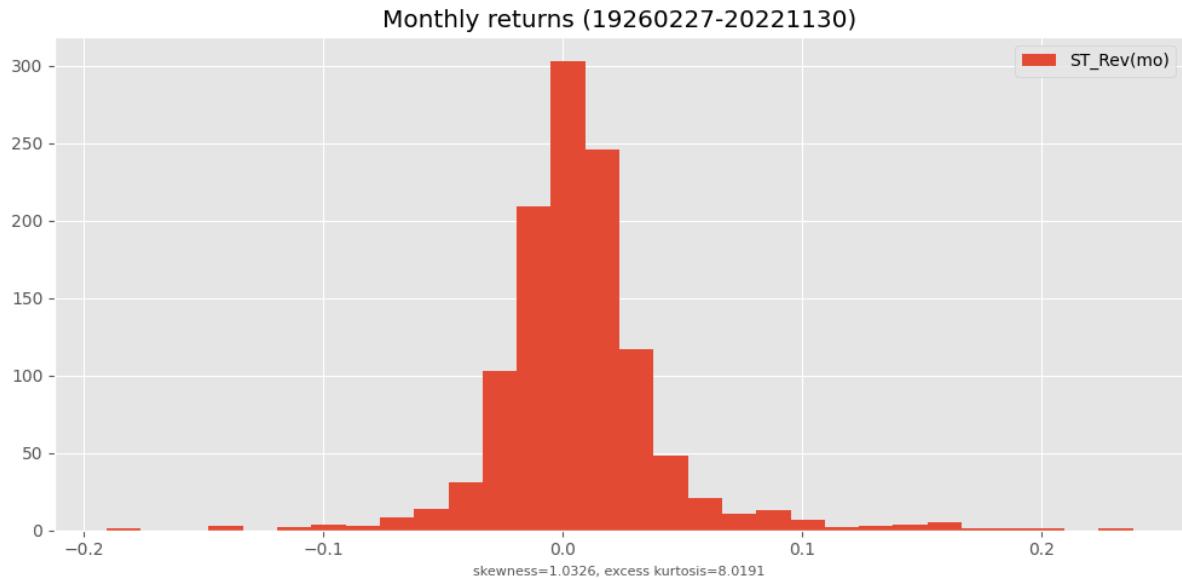
(continued from previous page)

Dep. Variable:	Q("ST_Rev(mo)")	R-squared:	0.049			
Model:	OLS	Adj. R-squared:	0.048			
Method:	Least Squares	F-statistic:	7.755			
Date:	Thu, 31 Aug 2023	Prob (F-statistic):	0.00544			
Time:	08:46:42	Log-Likelihood:	2286.8			
No. Observations:	1157	AIC:	-4570.			
Df Residuals:	1155	BIC:	-4560.			
Df Model:	1					
Covariance Type:	HAC					
<hr/>						
	coef	std err	z	P> z	[0.025	0.975]
Intercept	0.0059	0.001	5.587	0.000	0.004	0.008
Q("mkt-rf(mo)")	0.1421	0.051	2.785	0.005	0.042	0.242
<hr/>						
Omnibus:	265.445	Durbin-Watson:			1.921	
Prob(Omnibus):	0.000	Jarque-Bera (JB):			3905.523	
Skew:	0.629	Prob(JB):			0.00	
Kurtosis:	11.912	Cond. No.			18.7	
<hr/>						

Notes:

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





## 3.2 Linear Regression

FRM 7 + ISLP

- Ordinary least squares
- Intercept, slope
- Residuals and residual variance
- standard errors and asymptotic distribution
- RSS, F-test of regression, F-test of restricted regression
- RSS and TSS, R-square and correlation, Adjusted R-square

## 3.3 CAPM

- Modern portfolio theory (MPT): Markowitz's (1952) mean-variance model
- Capital asset pricing model (CAPM): Sharpe, Lintner, and Mossin derived an equilibrium model showing the relationship between the risk and expected return of a risky asset. The total risk of a risky asset can be 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 unique component that can be diversified away.
  - *Efficient frontier* – each point on this 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  $\sigma$
  - *Tangency portfolio* – a line is drawn from the risk-free rate becomes tangent to the efficient frontier at the point called the tangency portfolio
  - *Capital market line* – portfolios on the 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.

- *Two fund separation theorem* – the implication of the CML is that all investors should allocate to the risk-free asset and the tangency market portfolio
- *Security market line* – gives the relationship between the expected return for individual assets and risk as proxied by beta  $E(R_i) = r_f + \beta_i(E[R_M] - r_f)$

### Efficient frontier

<https://cvxopt.org/examples/tutorial/qp.html>

- 3x2 portfolio returns, allow shorts.

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.

### 3.3.1 Performance Measurement

FRM 4 -> Quant factors

- Sharpe ratio – slope of the capital market line is the fair equilibrium compensation:
- Treynor ratio – uses beta which is an appropriate measure of risk for a well-diversified portfolio:
- Jensen's alpha – the intercept of a CAPM regression should be zero in equilibrium:
- Sortino ratio – focuses on downside risk:
- Information Ratio – adjusts performance relative to a benchmark:

$$\frac{\text{average active return}}{\text{tracking error}} = \frac{\hat{R}_P - \hat{R}_B}{\sigma_{R_P - R_B}}$$

## FAMA-MACBETH CROSS-SECTIONAL REGRESSIONS

### UNDER CONSTRUCTION

- Beta, tests of the CAPM
- Polynomial regression, feature transformations
- Risk premiums from cross-sectional regressions
- Kernel regression and LOOCV

```
import numpy as np
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 pandas_datareader.famafrench import FamaFrenchReader
from tqdm import tqdm
from finds.database import SQL, RedisDB
from finds.structured import Signals, Benchmarks, CRSP
from finds.busday import BusDay
from finds.backtesting import RiskPremium
from finds.filters import winsorize
from finds.econs import least_squares
from finds.misc import Show
from secret import credentials, paths, CRSP_DATE
show = Show(ndigits=4, latex=None)
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)
```

(continues on next page)

(continued from previous page)

```
## Retrieve market and test asset returns
asset_names = ['25_Portfolios_ME_BETA_5x5']
# '25_Portfolios_ME_VAR_5x5', '25_Portfolios_ME_RESPAR_5x5', '25_Portfolios_5x5']

test_assets = {asset: FamaFrenchReader(asset, start=1900, end=2099).read()
               for asset in asset_names}
#for a in test_assets.values():
#    for p in range(2):
#        a[p].index = a[p].index.strftime('%Y%m').astype(int)

mkt = FamaFrenchReader('F-F_Research_Data_Factors', start=1900, end=2099).read()
mkt = mkt[0].rename(columns={'Mkt-RF': 'BETA'})
#mkt.index = mkt.index.strftime('%Y%m').astype(int)
```

```
# Show summary statistics of Mkt-Rf, every 25 years
periods=[(1925, 2024), (1925, 1949), (1950, 1974), (1975, 1999), (2000, 2024)]
df = []
for (beg, end) in periods:
    data = mkt[(mkt.index >= f"{beg}-01") & (mkt.index <= f"{end}-12")]['BETA']
    df.append(DataFrame({'of months': len(data),
                         'annual mean': 12*np.mean(data),
                         'annual stdev': np.std(data)*np.sqrt(12),
                         'sharpe ratio': np.mean(data)*np.sqrt(12)/np.std(data),
                         'skewness': skew(data),
                         'excess kurtosis': kurtosis(data, fisher=True)}))
    index=[f"\\{str(data.index[0])[:4]}-\\"
           f"\\{str(data.index[-1])[:4]}\\"])
show(pd.concat(df),
      caption="Summary statistics of Mkt-RF monthly returns")
```

	of months	annual mean
Summary statistics of Mkt-RF monthly returns		
1926-2023	1164	8.1160 \\
1926-1949	282	8.6630
1950-1974	300	6.4956
1975-1999	300	10.7464
2000-2023	282	6.4945
	annual stdev	sharpe ratio
Summary statistics of Mkt-RF monthly returns		
1926-2023	18.5122	0.4384 \\
1926-1949	26.6461	0.3251
1950-1974	13.5968	0.4777
1975-1999	15.3718	0.6991
2000-2023	15.9814	0.4064
	skewness	excess kurtosis
Summary statistics of Mkt-RF monthly returns		
1926-2023	0.1572	7.4086
1926-1949	0.4932	5.5027
1950-1974	-0.2875	0.8862
1975-1999	-0.7028	3.2154
2000-2023	-0.4912	0.7481

## 4.1 Cross-sectional regressions

### 4.1.1 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 are 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.

```

periods = [mkt.index[0], '1963-07']
out = {period: [] for period in periods}
for period in periods:

    for asset in asset_names:
        by_asset = []

        for p, wt in enumerate(['Value-weighted Test Assets', 'Equal-weighted Test Assets']):
            # subtract riskfree, and stack data as thin dataframe
            f = test_assets[asset][p]
            df = f.sub(mkt['RF'], axis=0).dropna().copy()
            rets = df.stack().reset_index(name='ret')\
                .rename(columns={'level_1':'port', 'level_0':'Date'})
            data = df.join(mkt[['BETA']], how='left')
            data = data[data.index >= period]

            # estimate test assets market betas from time-series of returns
            betas = least_squares(data,
                                  y=df.columns,
                                  x=['BETA'],
                                  stdres=True) [['BETA', 'stdres']]

            # orthogonalize beta^2 and residual-volatility regressors
            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)
            by_asset.append(r)

        out[period].append(by_asset)

    print(f'Completed {period} period')

```

(continues on next page)

(continued from previous page)

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

# time-series means and standard errors of the FM coefficients
sub = DataFrame({'mean': fm.mean(),
                  'stderr': fm.sem(),
                  'tstat': fm.mean() / fm.sem()}).T
sub.columns = pd.MultiIndex.from_tuples([(wt, col)
                                         for col in sub.columns])
sub.index = pd.MultiIndex.from_tuples([(asset, row)
                                         for row in sub.index])
by_asset.append(sub)
out[period].append(pd.concat(by_asset, axis=1))
p = periods[0]
show(pd.concat(out[p]),
      caption=[f"Fama-Macbeth Monthly CSR {p} to {df.index[-1]}", None])
```

Value-weighted Test Assets				
Intercept				
Fama-Macbeth Monthly CSR 1926-07 to 2023-06	mean	0.6599	\	
25_Portfolios_ME_BETA_5x5	stderr	0.1637		
	tstat	4.0310		
	BETA	BETA2	RES	
Fama-Macbeth Monthly CSR 1926-07 to 2023-06	mean	0.0861	-0.8213	0.1061
25_Portfolios_ME_BETA_5x5	stderr	0.2374	0.2630	0.0675
	tstat	0.3625	-3.1230	1.5724
Equal-weighted Test Assets				
Intercept				
Fama-Macbeth Monthly CSR 1926-07 to 2023-06	mean	0.8044	\	
25_Portfolios_ME_BETA_5x5	stderr	0.1561		
	tstat	5.1518		
	BETA	BETA2	RES	
Fama-Macbeth Monthly CSR 1926-07 to 2023-06	mean	-0.0080	-0.8349	0.1315
25_Portfolios_ME_BETA_5x5	stderr	0.2295	0.2597	0.0615
	tstat	-0.0349	-3.2146	2.1390

```
### Post-1963 time-period
p = periods[1]
show(pd.concat(out[p]),
      caption=[f"Fama-Macbeth Monthly CSR {p} to {df.index[-1]}", None])
```

Value-weighted Test Assets  
Intercept

(continues on next page)

(continued from previous page)

Fama-Macbeth Monthly CSR 1963-07 to 2023-06	mean	0.6599	\		
25_Portfolios_ME_BETA_5x5	stderr	0.1637			
	tstat	4.0310			
	BETA	BETA2	RES		
Fama-Macbeth Monthly CSR 1963-07 to 2023-06	mean	0.0861	-0.8213	0.1061	\
25_Portfolios_ME_BETA_5x5	stderr	0.2374	0.2630	0.0675	
	tstat	0.3625	-3.1230	1.5724	
	Equal-weighted Test Assets				
	Intercept				
Fama-Macbeth Monthly CSR 1963-07 to 2023-06	mean	0.8044	\		
25_Portfolios_ME_BETA_5x5	stderr	0.1561			
	tstat	5.1518			
	BETA	BETA2	RES		
Fama-Macbeth Monthly CSR 1963-07 to 2023-06	mean	-0.0080	-0.8349	0.1315	
25_Portfolios_ME_BETA_5x5	stderr	0.2295	0.2597	0.0615	
	tstat	-0.0349	-3.2146	2.1390	

### Clustered standard errors

```
### Compare 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:	2.811			
Date:	Thu, 31 Aug 2023	Prob (F-statistic):	0.0379			
Time:	10:40:36	Log-Likelihood:	-57761.			
No. Observations:	18000	AIC:	1.155e+05			
Df Residuals:	17996	BIC:	1.156e+05			
Df Model:	3					
Covariance Type:	nonrobust					
=====						
	coef	std err	t	P> t	[0.025	0.975]
-----						
Intercept	0.8044	0.189	4.256	0.000	0.434	1.175
BETA	-0.0080	0.165	-0.048	0.961	-0.332	0.316
BETA2	-0.8349	0.616	-1.356	0.175	-2.041	0.372
RES	0.1315	0.051	2.567	0.010	0.031	0.232
=====						
Omnibus:	1559.773	Durbin-Watson:			1.767	
Prob(Omnibus):	0.000	Jarque-Bera (JB):			10364.480	

(continues on next page)

(continued from previous page)

Skew:	-0.063	Prob (JB):	0.00
Kurtosis:	6.715	Cond. No.	20.8
=====			

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:	1.696			
Date:	Thu, 31 Aug 2023	Prob (F-statistic):	0.195			
Time:	10:40:36	Log-Likelihood:	-57761.			
No. Observations:	18000	AIC:	1.155e+05			
Df Residuals:	17996	BIC:	1.156e+05			
Df Model:	3					
Covariance Type:	hac-panel					
=====						
	coef	std err	t	P> t	[0.025	0.975]
Intercept	0.8044	0.199	4.045	0.000	0.394	1.215
BETA	-0.0080	0.191	-0.042	0.967	-0.403	0.387
BETA2	-0.8349	0.651	-1.282	0.212	-2.179	0.510
RES	0.1315	0.067	1.952	0.063	-0.008	0.271
=====						
Omnibus:	1559.773	Durbin-Watson:				1.767
Prob(Omnibus):	0.000	Jarque-Bera (JB):				10364.480
Skew:	-0.063	Prob(JB):				0.00
Kurtosis:	6.715	Cond. No.				20.8
=====						

Notes:

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

## 4.2 Kernel Regression

### 4.2.1 LOOCV

- Leave-one-out cross-validation

## 4.3 Cross-sectional regressions and portfolio sorts

Cross-sectional regression of monthly stock returns on fundamental stock exposures (winsored at 5% tail):

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

compared to time series of portfolio-sort monthly returns

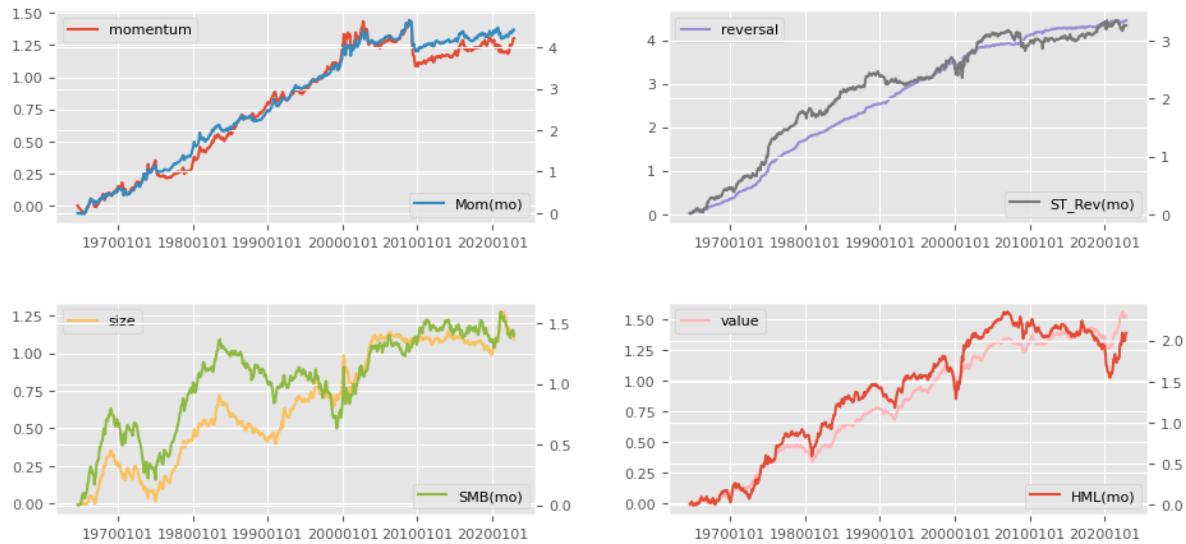
```
rebalbeg=19640601
rebalend=LAST_DATE
rebaldates = crsp.bd.date_range(rebalbeg, rebalend, 'endmo')
loadings = dict()
for pordate in tqdm(rebaldates):           # retrieve signal values every month
    date = bd.june_universe(pordate)
    univ = crsp.get_universe(date)
    cap = np.sqrt(crsp.get_cap(date))
    smb = -np.log(cap).rename('size')
    hml = signals('hml', date, bd.endmo(date, -12))['hml'].rename('value')
    #beta = signals('beta', pordate, bd.begmo(pordate))['beta']*2/3 + 1/3 #shrink
    mom = signals('mom', pordate)['mom'].rename('momentum')
    strev = -signals('strev', pordate)['strev'].rename('reversal')
    df = pd.concat((strev, hml, smb, mom),  # inner join of signals with univ
                   join='inner',
                   axis=1).reindex(univ.index).dropna()
    loadings[pordate] = winsorize(df, quantiles=[0.05, 0.95])

# Compute coefficients from FM cross-sectional regressions
riskpremium = RiskPremium(user, bench, 'RF', LAST_DATE)
out = riskpremium(stocks=crsp,          # FM regressions on standardized scores
                  loadings=loadings,
                  standardize=['value', 'size', 'momentum', 'reversal'])

# Compare time series of estimated risk premiums to portfolio-sort benchmark returns
benchnames = {'momentum': 'Mom(mo)',
              'reversal': 'ST_Rev(mo)',
              'size': 'SMB(mo)',
              'value': 'HML(mo)'}
out = riskpremium.fit(benchnames.values())  # to compare portfolio-sorts
riskpremium.plot(benchnames)
plt.savefig(imgdir / 'fm.jpg')
```

0% | 0/702 [00:00<?, ?it/s]

100% |██████████| 702/702 [16:00<00:00, 1.37s/it]



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

Factor Returns	mean	stderr	std
Fama-MacBeth Cross-sectional Regression Risk Pr...			
reversal	0.0063	0.0005	0.0143
value	0.0022	0.0004	0.0112
size	0.0016	0.0007	0.0175
momentum	0.0019	0.0007	0.0175

Factor Returns	count	tvalue	sharpe
Fama-MacBeth Cross-sectional Regression Risk Pr...			
reversal	701	11.7877	1.5423
value	701	5.1258	0.6706
size	701	2.3589	0.3086
momentum	701	2.8051	0.3670

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

Benchmarks	mean	stderr	std	count	tvalue	sharpe
Fama-French Portfolio-Sorts						
Mom(mo)	0.0063	0.0016	0.0422	701	3.9535	0.5173
ST_Rev(mo)	0.0047	0.0012	0.0315	701	3.9134	0.5120
SMB(mo)	0.0020	0.0012	0.0307	701	1.7260	0.2258
HML(mo)	0.0030	0.0011	0.0299	701	2.6371	0.3450

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

	reversal	value	size	
Correlation of FM Risk Premiums and FF Portfoli...				
reversal	1.0000	-0.0039	0.0881	\
value	-0.0039	1.0000	-0.1993	
size	0.0881	-0.1993	1.0000	
momentum	-0.4460	-0.1860	-0.0179	
Mom (mo)	-0.3965	-0.1618	0.1198	
ST_Rev (mo)	0.7942	-0.0399	0.0176	
SMB (mo)	0.1340	-0.2487	0.5747	
HML (mo)	0.0551	0.8178	-0.1421	
				momentum
				Mom (mo)
Correlation of FM Risk Premiums and FF Portfoli...				
reversal	-0.4460	-0.3965	\	
value	-0.1860	-0.1618		
size	-0.0179	0.1198		
momentum	1.0000	0.8860		
Mom (mo)	0.8860	1.0000		
ST_Rev (mo)	-0.2729	-0.3057		
SMB (mo)	-0.0416	-0.0282		
HML (mo)	-0.2076	-0.2053		
				ST_Rev (mo)
				SMB (mo)
Correlation of FM Risk Premiums and FF Portfoli...				
reversal	0.7942	0.1340	\	
value	-0.0399	-0.2487		
size	0.0176	0.5747		
momentum	-0.2729	-0.0416		
Mom (mo)	-0.3057	-0.0282		
ST_Rev (mo)	1.0000	0.1754		
SMB (mo)	0.1754	1.0000		
HML (mo)	0.0206	-0.1750		
				HML (mo)
Correlation of FM Risk Premiums and FF Portfoli...				
reversal	0.0551			
value	0.8178			
size	-0.1421			
momentum	-0.2076			
Mom (mo)	-0.2053			
ST_Rev (mo)	0.0206			
SMB (mo)	-0.1750			
HML (mo)	1.0000			



## WEEKLY REVERSAL

## UNDER CONSTRUCTION

- Contrarian trading (Lo and Mackinlay 1990, and others), mean reversion
- implementation shortfall
- structural break with unknown changepoint

```
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.busday import BusDay
from finds.structured import CRSP, Benchmarks
from finds.backtesting import fractiles
from finds.econs import lm
from finds.misc import PyR, row_formatted, Show
from secret import credentials, paths, CRSP_DATE
show = Show(ndigits=4, latex=None)
VERBOSE = 0      # 1
#%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)
bench = Benchmarks(sql, bd)
imgdir = paths['images']
```

## Construct weekly reversal

Retrieve weekly returns, standardize scores, and compute returns and i.c.

```
weekday = 3          # wednesday close-to-close
bd = BusDay(sql, endweek=weekday)  # Generate weekly cal
begweek = 19740102  # increased stocks coverage in CRSP in Jan 1973
middate = 19851231  # increased stocks traded in CRSP around this date
endweek = bd.endwk(CRSP_DATE, -1)
rebaldates = bd.date_range(begweek, endweek, freq='weekly')
```

(continues on next page)

(continued from previous page)

```
retdates = bd.date_tuples(rebaldates)
june_universe = 0 # to track when reached a June end to update universe
year = 0 # to track new year to retrieve prices in batch
results = []
lagged_weights = Series(dtype=float) # to track "turnover" of stock weights
```

Last FamaFrench Date 2023-04-28 00:00:00

Loop over weekly rebalance dates

```
for rebaldate, pastdates, nextdates in tqdm(zip(rebaldates[1:-1],
                                                retdates[:-1],
                                                retdates[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'] <= 9] # drop smallest half stocks

    # retrieve new annual batch of daily prices and returns when start 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 start of week
    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 minus returns to standardized weights in portfolio
    weights = ((past_week['ret'].mean() - past_week['ret']) / \
               (past_week['ret'].std(ddof=0) * len(past_week)))

    # turnover = total abs change in stock weight, scaled by total abs weight
    chg_weights = pd.concat([weights, -lagged_weights], axis=1) \
        .fillna(0) \
        .sum(axis=1)

    total_weight = weights.abs().sum() + lagged_weights.abs().sum()
    lagged_weights = weights

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

    # get next day's closing prices (or bid-ask quotes), to compute one-day delay cost
    next_day = prc[prc.index.get_level_values('date') == \
                  bd.offset(rebaldate, 1)] \
        .reset_index()
```

(continues on next page)

(continued from previous page)

```

.set_index('permno') \
.drop(columns='date') \
.reindex(chg_weights.index)
avgprc = next_day[['bidlo', 'askhi', 'prc']].abs().mean(axis=1)

# if no trade next day, then enter position at askhi (long) or bidlo (short)
bidask = next_day['askhi'].where(weights > 0, next_day['bidlo']).abs()
avgprc = next_day['prc'].where(next_day['prc'] > 0, bidask)

# delay slippage (positive is cost) = weights chg * drift of next day price
drift = avgprc.div(next_day['prc'].abs()) \
    .mul(1 + next_day['ret']) \
    .sub(1) \
    .fillna(0)

# accumulate weekly computations
results.append(DataFrame({'ret': weights.dot(next_week),
                           '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),
                           'delay': chg_weights.dot(drift)}),
               index=[rebaldate]))

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

# Show summary
cols = ['ic', 'vol', 'ret', 'delay', 'turnover']
indexes = ['Information coefficient', 'Cross-sectional Volatility',
           'Alpha (gross return)', 'Delay cost', 'Portfolio turnover']
show(pd.concat([df[cols].mean(axis=0).rename('mean'),
                df[cols].std(axis=0).rename('std')],
                axis=1).set_index(pd.Index(indexes)),
      caption=f'Summary of Weekly Mean Reversion Strategy {dates[0]}-{dates[-1]}')
)

```

2555it [01:13, 34.69it/s]

	mean	std
Summary of Weekly Mean Reversion Strategy 19740...		
Information coefficient	0.0398	0.0927
Cross-sectional Volatility	0.0583	0.0203
Alpha (gross return)	0.0024	0.0077
Delay cost	0.0019	0.0042
Portfolio turnover	0.7370	0.0384

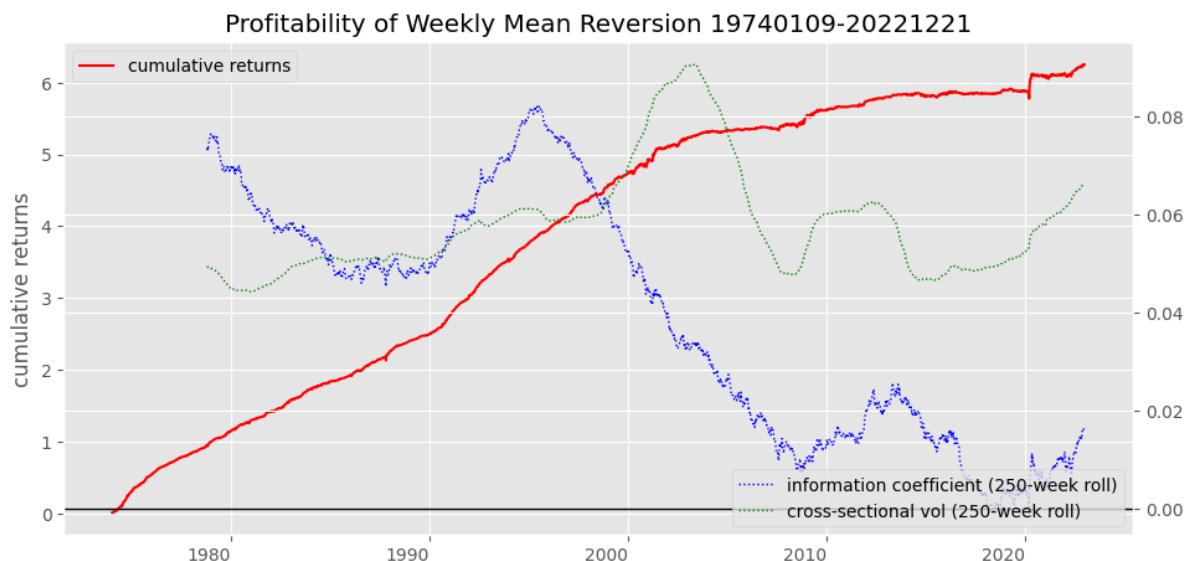
## 5.1 Implementation Shortfall

- decision price
- delay
- market impact
- opportunity cost

## 5.2 Information coefficient

- Grinold rule of thumb
- alpha = IC time volatility

```
## Plot returns, and rolling avg information coefficient and cross-sectional vol
fig, ax = plt.subplots(num=1, clear=True, figsize=(10, 5))
df['ret'].cumsum().plot(ax=ax, ls='-', color='r', rot=0)
ax.legend(['cumulative returns'], loc='upper 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')
#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 roll)",
           f"cross-sectional vol ({roll}-week roll)"],
           loc='lower right')
ax.set_title(f'Profitability of Weekly Mean Reversion {dates[0]}-{dates[-1]}')
plt.tight_layout(pad=2)
plt.savefig(imgdir / 'weekrev.jpg')
```



## 5.3 Structural break with unknown changepoint

- Welch/Chow test
- Andrews

```
# 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()
output['mean(post)'] = Y[df.index > output['breakpoints']].mean()
fstat = [0] + list(PyR(fstats_r[0]).values) + [0, 0] # pad beyond from and to

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

```
2.5 % breakpoints
Structural break test with unknown changepoint
sctest                                         1998-07-01  2001-07-03 \
                                                               \

97.5 % statistic  p.value
Structural break test with unknown changepoint
sctest                                         2002-07-31      23.1675      0.0 \
                                                               \

method  mean(pre)
Structural break test with unknown changepoint
sctest                                         aveF test      0.0035 \
                                                               \

mean(post)
Structural break test with unknown changepoint
sctest                                         0.001
```

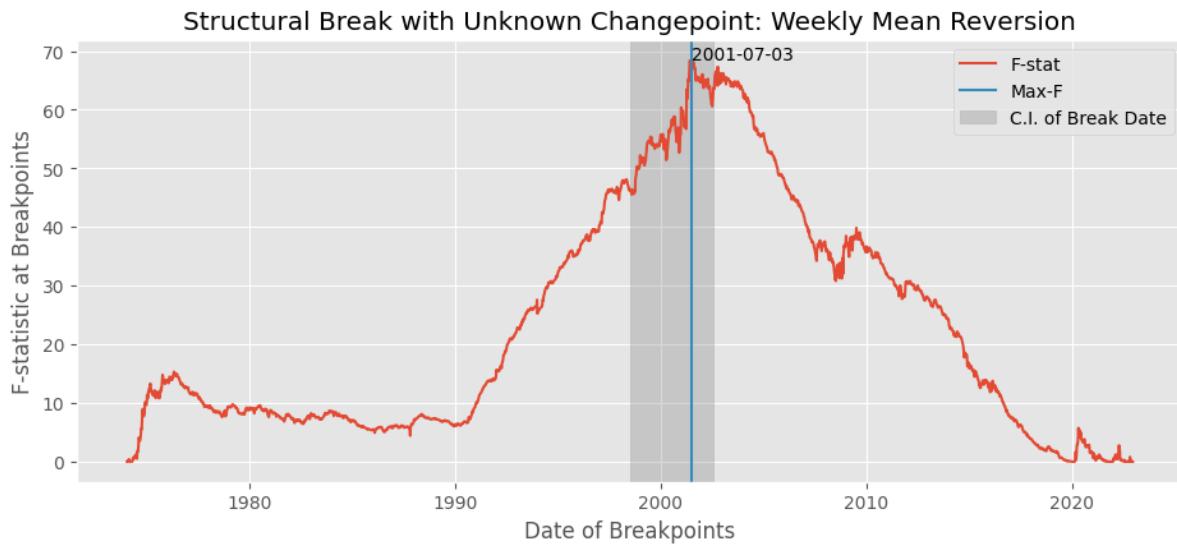
Plot breakpoint F-stats

```
fig, ax = plt.subplots(num=2, clear=True, figsize=(10, 5))
ax.plot(df.index, fstat, color='C0')
arg = np.nanargmax(fstat)
ax.axvline(df.index[arg], 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'])
```

(continues on next page)

(continued from previous page)

```
ax.annotate(df.index[arg].strftime('%Y-%m-%d'), xy=(df.index[arg], fstat[arg]))
ax.set_ylabel('F-statistic at Breakpoints')
ax.set_xlabel('Date of Breakpoints')
ax.set_title('Structural Break with Unknown Changepoint: '
             'Weekly Mean Reversion')
plt.tight_layout(pad=3)
plt.savefig(imgdir / 'break.jpg')
```



Compute gross annualized sharpe ratio and delay slippage

```
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)

    # 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 Return': res['ret'].mean(),
        'Std Dev Return': res['ret'].std(),
        'Market Beta': model.coefficients[1],
        'Jensen Alpha (annualized)': model.coefficients[0] * 52,
        'Appraisal Ratio': np.sqrt(52) * model.coefficients[0] / model.stderr,
        'Information Coefficient': res['ic'].mean(),
        'Cross-sectional Vol': res['vol'].mean(),
    }
```

(continues on next page)

(continued from previous page)

```

'Delay cost': res['delay'].mean(),
'Turnover Fraction': res['turnover'].mean(),
#'Abs Weight': res['absweight'].mean(),
'Num Stocks': int(res['n'].mean()),
}

# Display as formatted DataFrame
formats = dict.fromkeys(['start date', 'end date', 'Num Stocks'], '{:.0f}')
show(row_formatted(DataFrame(out), formats=formats, default='{:4f}'),
      caption="Subperiod Performance of Weekly Mean Reversion Strategy")

```

```
(get_series many) SELECT date, permno, ret FROM benchmarks WHERE date >=
19000000 AND date <= 29001231 AND permno IN ('Mkt-RF')
```

Full Period	
Subperiod Performance of Weekly Mean Reversion ...	
start date	19740109 \
end date	20221221
Sharpe Ratio	2.2938
Average Return	0.0024
Std Dev Return	0.0077
Market Beta	0.0928
Jensen Alpha (annualized)	0.1201
Information Coefficient	0.0398
Cross-sectional Vol	0.0583
Delay cost	0.0019
Turnover Fraction	0.7370
Num Stocks	2552

Pre-break Period	
Subperiod Performance of Weekly Mean Reversion ...	
start date	19740109 \
end date	20010703
Sharpe Ratio	4.2094
Average Return	0.0035
Std Dev Return	0.0061
Market Beta	0.0688
Jensen Alpha (annualized)	0.1792
Information Coefficient	0.0601
Cross-sectional Vol	0.0588
Delay cost	0.0033
Turnover Fraction	0.7401
Num Stocks	2658

Post-break Period	
Subperiod Performance of Weekly Mean Reversion ...	
start date	20010711
end date	20221221
Sharpe Ratio	0.8167
Average Return	0.0010
Std Dev Return	0.0092
Market Beta	0.1181
Jensen Alpha (annualized)	0.0446
Information Coefficient	0.0138
Cross-sectional Vol	0.0578

(continues on next page)

(continued from previous page)

Delay cost	0.0002
Turnover Fraction	0.7331
Num Stocks	2416

## QUANT FACTORS

### UNDER CONSTRUCTION

- return predicting signals (Green et al 2013, and others), factor zoo
- performance evaluation

```
from pandas import DataFrame, Series
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm
from datetime import datetime
from typing import List, Tuple, Any, Dict
from finds.database import SQL, RedisDB
from finds.structured import Stocks, Benchmarks, SignalsFrame, CRSP, PSTAT, ↵
    IBES
from finds.busday import BusDay
from finds.backtesting import BackTest, univariate_sorts, fractiles
from finds.finance import maximum_drawdown
from finds.misc import Show
from secret import credentials, paths, CRSP_DATE
show = Show(ndigits=4, latex=None)
pd.set_option('display.max_rows', None)
VERBOSE = 0
#%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['images'] / 'factors'
```

```
# TODO
#
```

(continues on next page)

(continued from previous page)

```
# drop penny stocks $1 and microcap (smallest quintile)
# decile spreads
#
# 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, and (optionally) save file and .jpg
```

Args:

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

(continues on next page)

(continued from previous page)

```

>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)
print(pd.Series(backtest.annualized,
                 name=label + suffix).to_frame().T.round(3).to_string())
if outdir: # output graph and summary statistics
    plt.savefig(outdir / (label + '.jpg'))

# 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><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"{{}.join(f"{{k:10s}}" for k in sub)}\n")
    f.write(f"{{:12s}}\n".format(label + ":"))
    f.write(f"{{}.join(f"{{backtest.annualized[k]:10.4f}}" for k in sub)}\n")
    f.write(f"\n</pre>\n<img src='{label}.jpg'><p>{datetime.now()}\n")
return summary

```

List subsets of signals to (optionally) generate and run backtest

```

testable = {'new', 'monthly', 'weekly', 'daily', 'pstann', 'pstqtr',
           'ibesf1', 'ibesltg', 'rdq_daily', 'ibesf1_hist', 'ibesf1_pstqtr',
           'ibesq1_pstqtr', 'summarize'} # subset of steps to rerun
regenerate = True # False to only run backtest (and not regenerate Signals)

# specify subsets for this run
#testable = {'rdq_daily', 'ibesf1_hist', 'ibesf1_pstqtr',
#           'ibesq1_pstqtr', 'summarize'} # subset of steps to rerun

# (optionally) initialize webpages to output results
if 'new' in testable:
    with open(outdir / 'index.html', 'wt') as f:
        f.write('<h1>Quant Factors</h1><br>')
        f.write('<br>')
        f.write('<p>\n')

```

## 6.1 Past prices

```

# Momentum and divyld from CRSP monthly
if 'monthly' in testable:
    if regenerate:
        beg, end = 19251231, LAST_DATE
        intervals = {'mom12m': (2, 12),
                      'mom36m': (13, 36),
                      'mom6m': (2, 6),
                      'mom1m': (1, 1)}
        for label, past in intervals.items():
            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'] = crsp.get_section(dataset='monthly',
                                                fields=['ret'],
                                                date_field='date',
                                                date=start).reindex(df.index)
                df[label] = crsp.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'] = crsp.get_section(dataset='monthly',
                                            fields=['ret'],
                                            date_field='date',
                                            date=start).reindex(df.index)
            df['end2'] = crsp.get_section(dataset='monthly',
                                            fields=['ret'],
                                            date_field='date',
                                            date=end2).reindex(df.index)
            df['mom2'] = crsp.get_ret(beg2, end2).reindex(df.index)
            df['mom1'] = crsp.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']

```

(continues on next page)

(continued from previous page)

```

# 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)

benchnames = ['Mkt-RF(mo)']
rebalbeg, rebalend = 19260101, LAST_DATE
for num, label in enumerate(['mom12m', 'mom6m', 'chmom', 'indmom', 
    'divyld', 'mom1m', 'mom36m']):
    holdings = univariate_sorts(crsp,
        label,
        SignalsFrame(signals.read(label)),
        rebalbeg,
        rebalend,
        window=1,
        months=[],
        maxdecile=8,
        pct=(10., 90.),
        leverage=leverage.get(label, 1))
    excess = backtest_pipeline(backtest,
        crsp,
        holdings,
        label,
        benchnames,
        overlap=0,
        outdir=outdir,
        suffix=(leverage.get(label, 1) < 0) * '(-)')

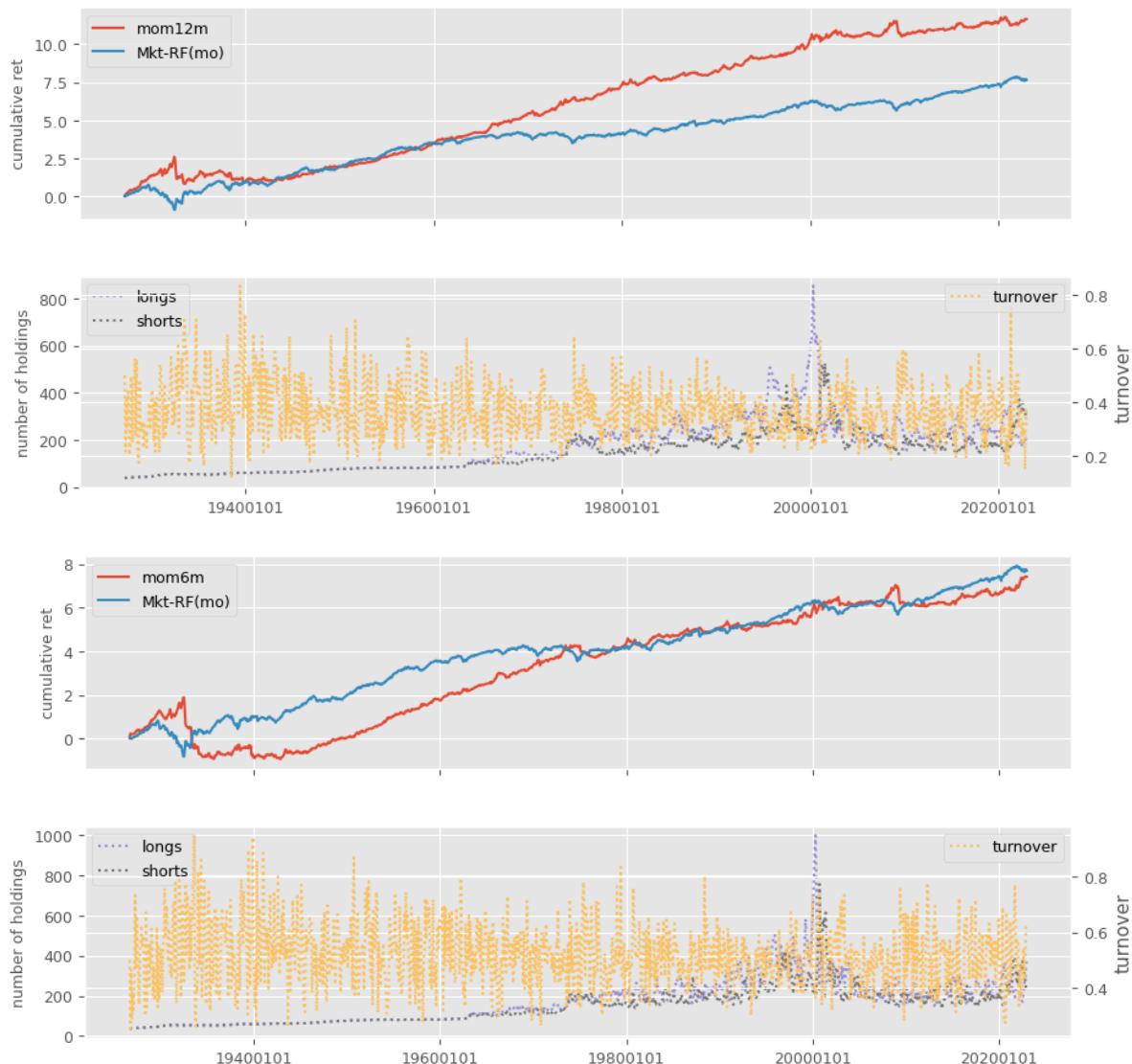
```

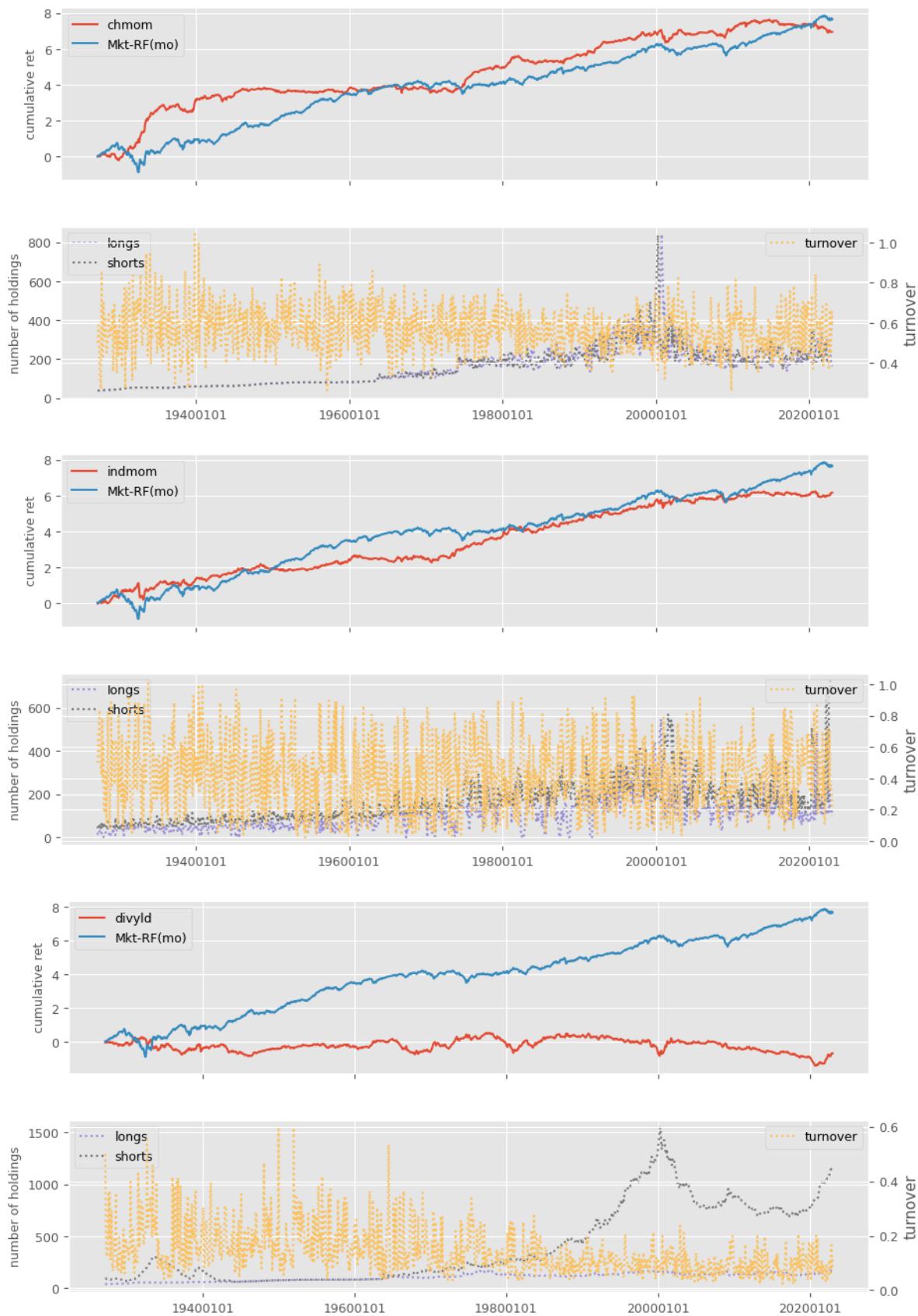
	excess	sharpe	alpha	appraisal	welch-t	welch-p	turnover	longs	shorts	buys	sells	
mom12m	0.116	0.469	0.152	0.651	-1.378	0.169	4.278	172.74	142.	784	4.248	4.308
	excess	sharpe	alpha	appraisal	welch-t	welch-p	turnover	longs	shorts	buys	sells	
mom6m	0.074	0.319	0.108	0.499	-0.461	0.645	5.957	171.15	150.	073	5.937	5.976
	excess	sharpe	alpha	appraisal	welch-t	welch-p	turnover	longs	shorts	buys	sells	
chmom	0.069	0.415	0.053	0.326	-1.801	0.072	6.535	156.97	157.	132	6.518	6.552
	excess	sharpe	alpha	appraisal	welch-t	welch-p	turnover	longs	shorts	buys	sells	
indmom	0.062	0.354	0.065	0.373	-1.116	0.265	4.857	106.519	161.	022	4.841	4.874
	excess	sharpe	alpha	appraisal	welch-t	welch-p	turnover	longs	shorts	buys	sells	
divyld	-0.007	-0.035	0.039	0.254	-0.765	0.445	1.61	106.845	406.	693	1.625	1.596

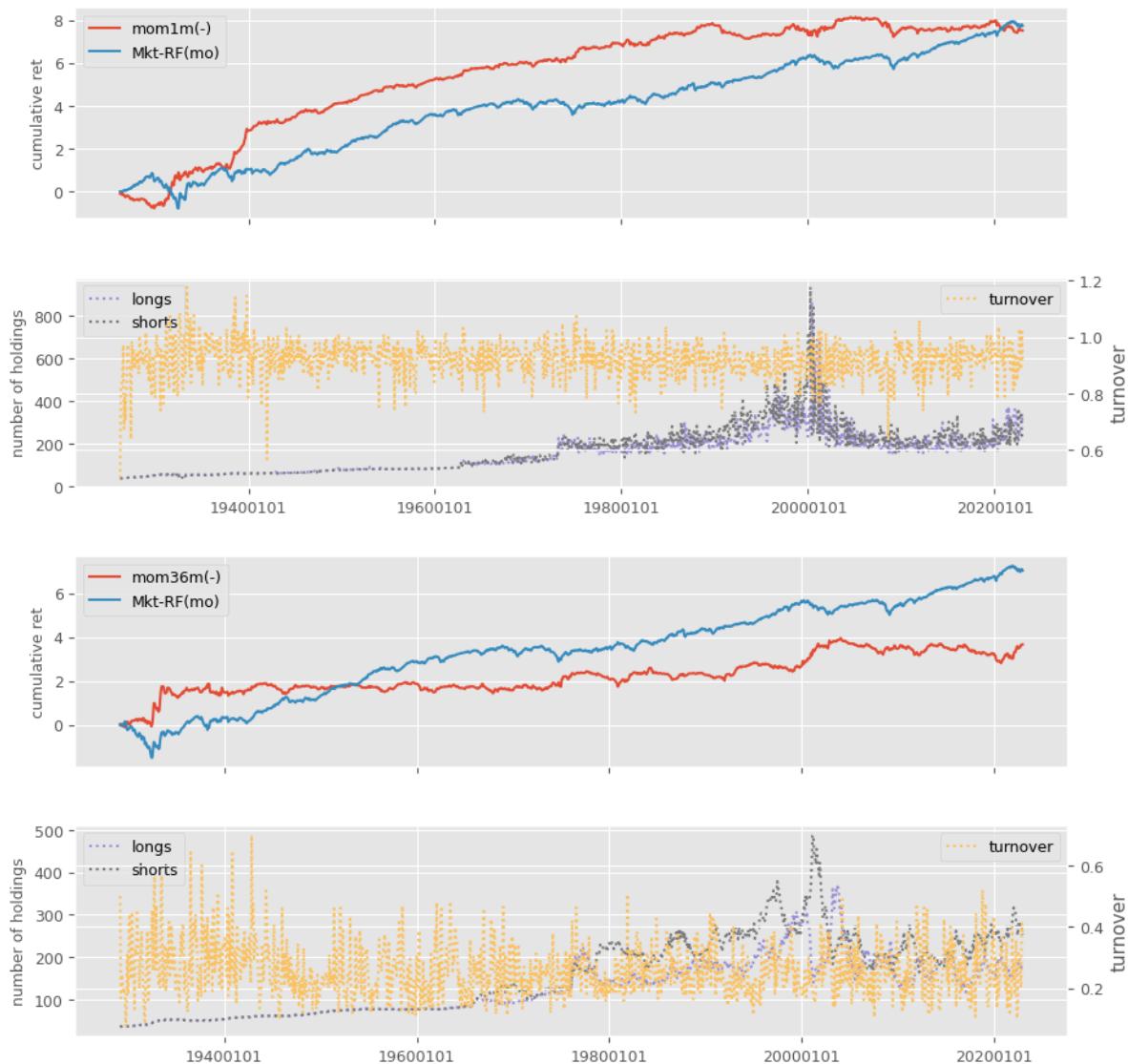
(continues on next page)

(continued from previous page)

	excess	sharpe	alpha	appraisal	welch-t	welch-p	turnover	longs	shorts
mom1m(-)	0.074	0.408	0.062	0.343	-2.308	0.021	10.611	157.552	167.075
mom36m(-)	0.038	0.185	0.023	0.114	-0.936	0.35	2.991	128.121	156.69
buys									2.982
sells									3.0







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

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

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

(continues on next page)

(continued from previous page)

```

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

```

```

# Weekly returns-based price response signals
if 'weekly' in testable:
    beg, end = 19260101, LAST_DATE
    columns = ['beta', 'idiovol', 'pricedelay']
    wd = BusDay(sql, endweek='Wed')    # custom weekly trading day calendar

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

    if regenerate:

        """
        # Pre-generate weekly returns and save in cache-store
        batchsize = 40
        r = bd.date_tuples(wd.date_range(beg, LAST_DATE))
        batches = [r[i:(i+batchsize)] for i in range(0, len(r), batchsize)]
        for batch in batches:
            crsp.cache_ret(batch, replace=True)
        """

        for date in wd.date_range(beg, end, 'weekly'):
            df = crsp.get_ret(wd.begwk(date), date)
            mkt = as_rolling(mkt,           # rolling window of weekly mkt returns
                             DataFrame(data=[df.mean()], columns=[date]),
                             width=width)
            weekly = as_rolling(weekly, # rolling window of weekly stock returns
                               df.rename(date),
                               width=width)
            valid = weekly.count(axis=1) >= mininvalid    # require min number weeks
            if valid.any():
                result = DataFrame([regress(mkt.values[0], y

```

(continues on next page)

(continued from previous page)

```

        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)

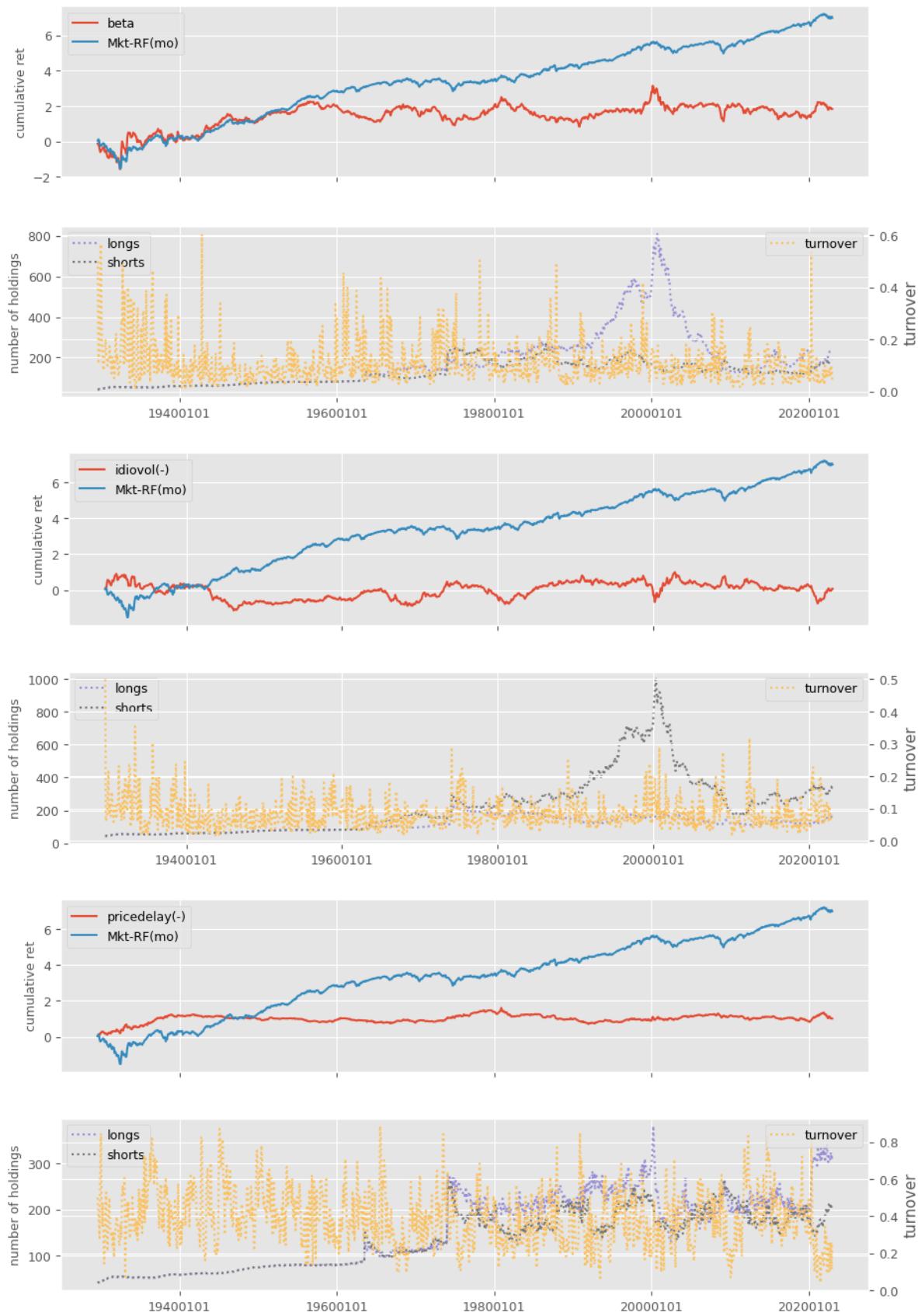
benchnames = ['Mkt-RF(mo)']
rebalbeg, rebalend = 19290601, LAST_DATE
for num, label in enumerate(columns):
    holdings = univariate_sorts(crsp,
                                 label,
                                 SignalsFrame(signals.read(label)),
                                 rebalbeg,
    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

```

```

Last FamaFrench Date 2023-04-28 00:00:00
      excess sharpe alpha appraisal welch-t welch-p turnover longs ▾
      ↵ shorts buys sells
beta  0.019  0.062 -0.075     -0.393   -0.538    0.591     1.238  183.442 126.
      ↵235  1.226  1.251
      excess sharpe alpha appraisal welch-t welch-p turnover longs ▾
      ↵ shorts buys sells
idiovol(-)  0.001  0.003  0.053      0.271   -0.332     0.74     0.96  115.702 ▾
      ↵229.826  0.968  0.953
      excess sharpe alpha appraisal welch-t welch-p turnover ▾
      ↵ longs shorts buys sells
pricedelay(-)  0.01  0.107  0.005      0.054   -0.437     0.662     4.511 154.
      ↵761  134.785  4.508  4.515

```



## 6.2 Liquidity

```

# Liquidity signals from daily stock returns
if 'daily' in testable:
    beg, end = 19830601, LAST_DATE      # nasdaq/volume from after 1982
    columns = ['ill', 'maxret', 'retvol', 'baspread', 'std_dolvol',
               'zerotrade', 'std_turn', 'turn']
    if regenerate:
        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)

```

(continues on next page)

(continued from previous page)

```

dolvol = pd.concat(dolvol, axis=0, ignore_index=True)

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 num, label in enumerate(columns + ['dolvol']):
    holdings = univariate_sorts(crsp,
                                 label,
                                 SignalsFrame(signals.read(label)),
                                 rebalbeg,
                                 rebalend,
                                 window=1,
                                 months=[],
                                 maxdecile=8,
                                 pct=(10., 90.),
                                 leverage=leverage.get(label, 1))
    excess = backtest_pipeline(backtest,
                               crsp,
                               holdings,
                               label,
                               benchnames,
                               overlap=0,
                               outdir=outdir,
                               suffix=(leverage.get(label, 1) < 0) * '(-)')

```

100% [██████████] 40/40 [22:26&lt;00:00, 33.67s/it]

	excess	sharpe	alpha	appraisal	welch-t	welch-p	turnover	longs	...
↳ shorts	buys	sells							...
ill(-)	0.034	0.294	0.043	0.371	-1.245	0.214	2.563	146.451	302.
↳ 127	2.556	2.569							...
↳ shorts	buys	sells							...
maxret(-)	0.055	0.251	0.123	0.698	-0.279	0.78	9.11	165.346	...
↳ 335.778	9.101	9.119							...
↳ shorts	buys	sells							...
retvol(-)	0.056	0.213	0.141	0.683	-0.191	0.849	7.483	162.215	...
↳ 370.73	7.475	7.492							...
↳ shorts	buys	sells							...
baspread	-0.035	-0.128	-0.126	-0.601	0.165	0.869	5.114	434.074	...
↳ 156.268	5.117	5.112							...
↳ shorts	buys	sells							...
std_dolvol	-0.0	-0.004	0.002	0.022	1.108	0.268	7.51	333.903	...
↳ 147.506	7.508	7.513							...
↳ longs	shorts	buys	sells						...
zerotrade(-)	0.024	0.112	-0.041	-0.234	0.496	0.62	4.267	350.	...
↳ 059	260.173	4.258	4.276						...
↳ excess	sharpe	alpha	appraisal	welch-t	welch-p	turnover	longs	...	...

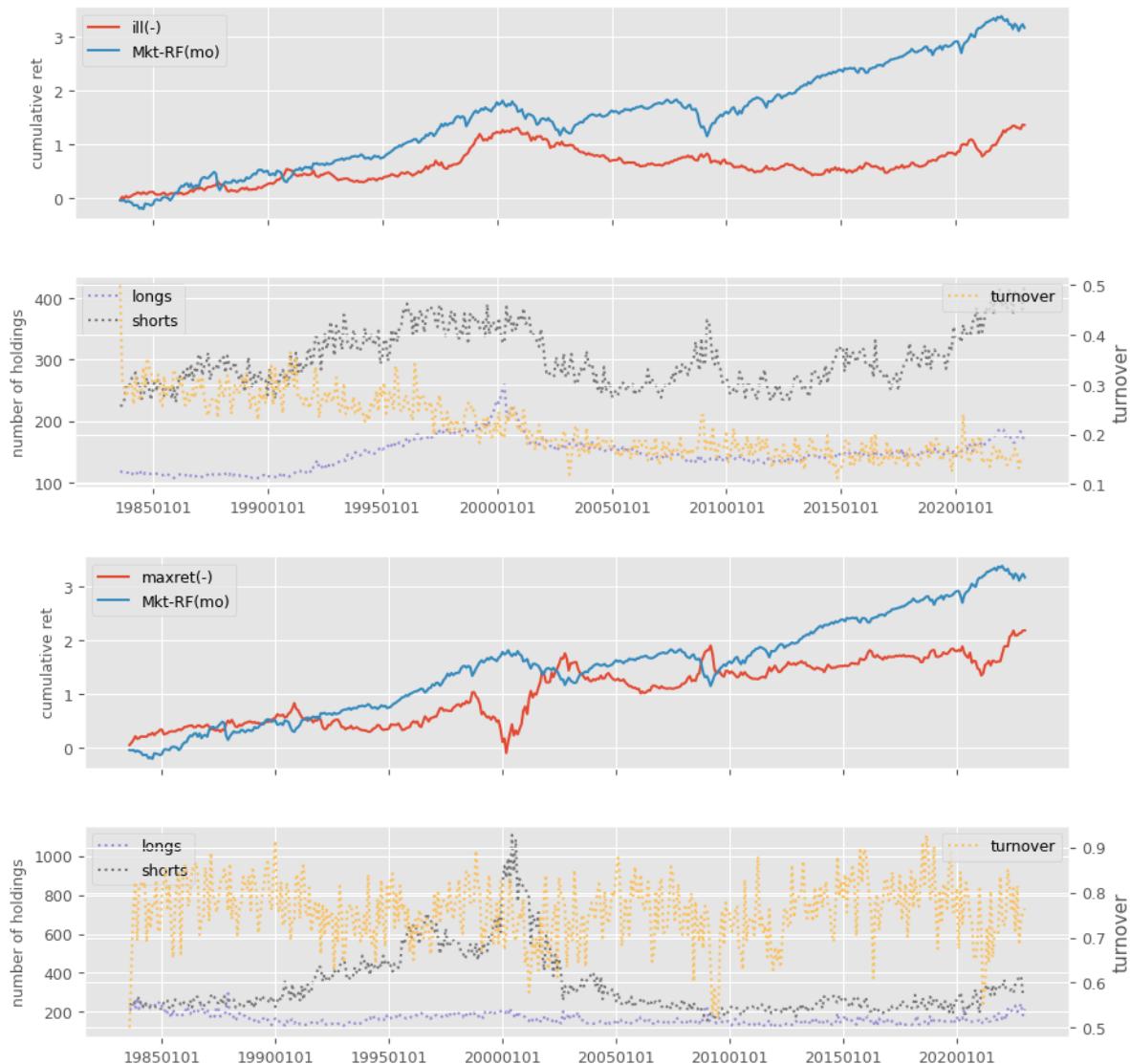
(continues on next page)

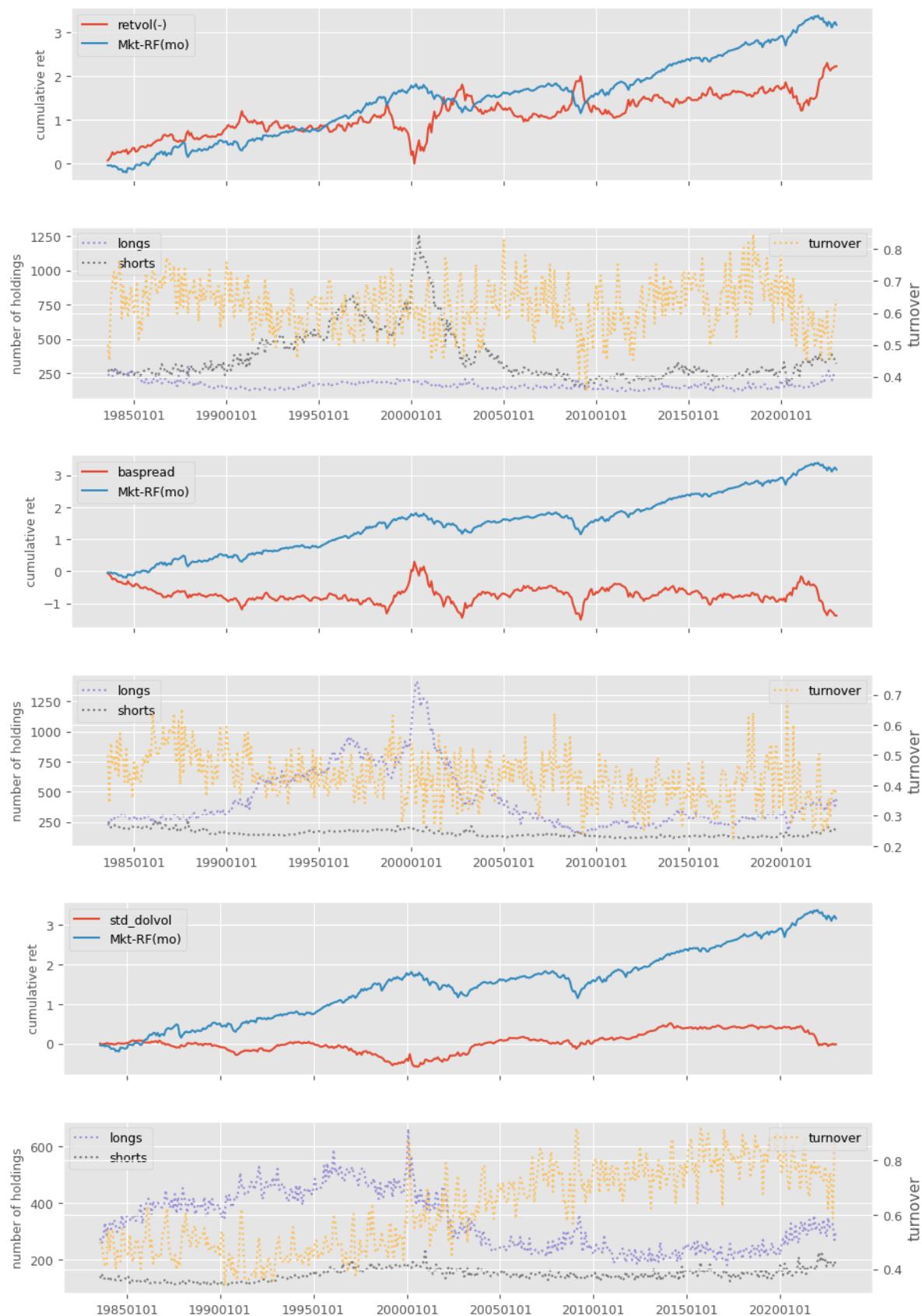
(continued from previous page)

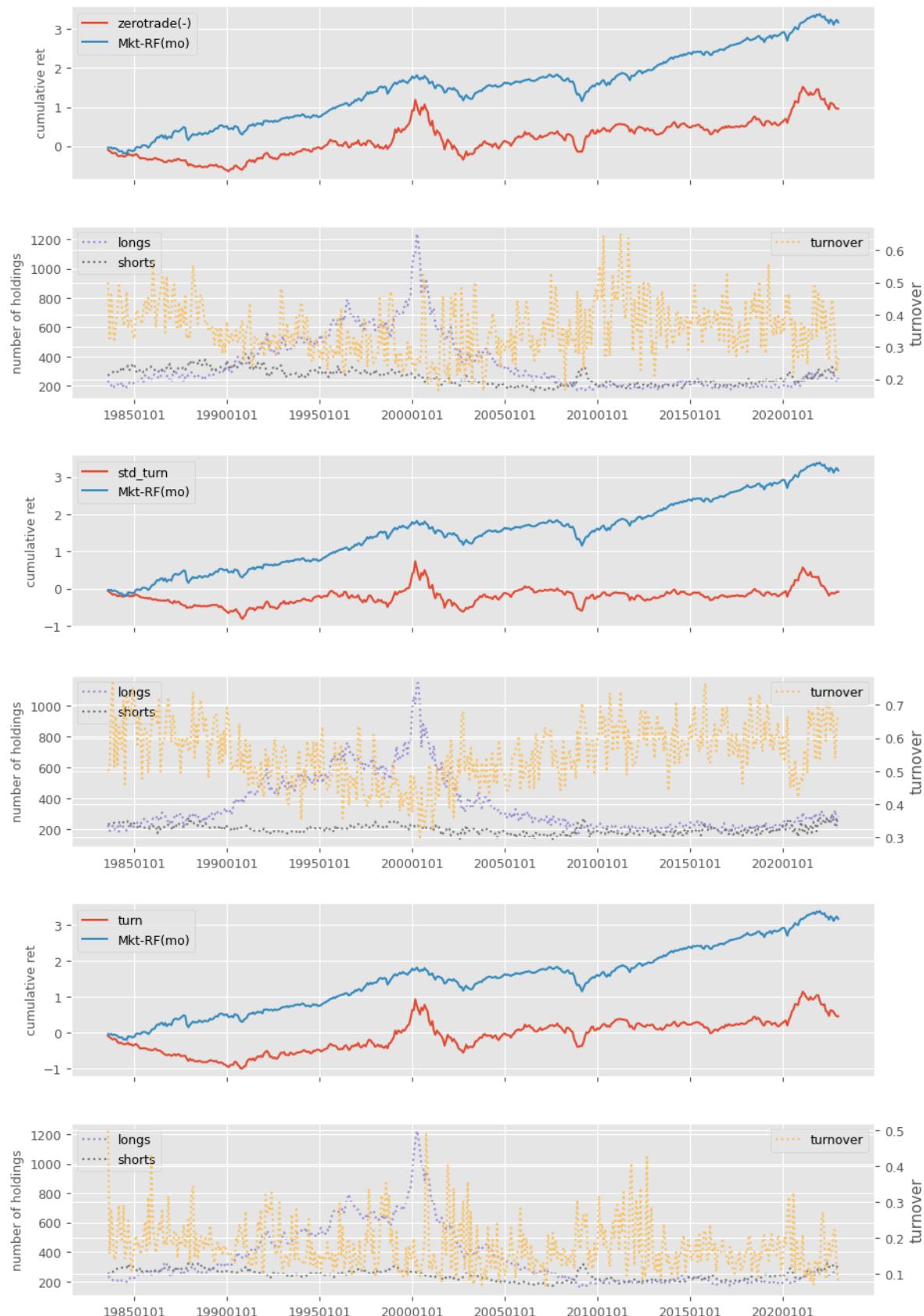
```

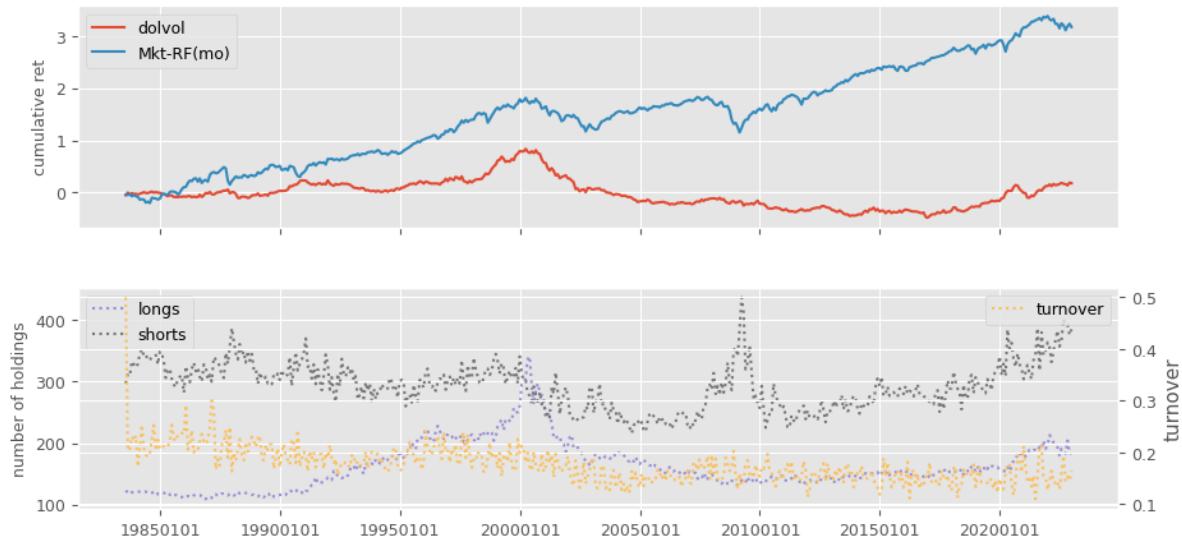
↳shorts  buys  sells
std_turn -0.002 -0.011 -0.057      -0.338      0.414      0.679      6.649  354.857 ↴
↳198.627  6.645  6.652
      excess  sharpe  alpha  appraisal  welch-t  welch-p  turnover  longs ↴
↳shorts  buys  sells
turn   0.012  0.052 -0.056      -0.312      0.481      0.631      2.115  359.409  242.
↳88  2.109  2.121
      excess  sharpe  alpha  appraisal  welch-t  welch-p  turnover  longs ↴
↳shorts  buys  sells
dolvol  0.005  0.043 -0.006      -0.061      -0.569      0.57      2.111  161.791 ↴
↳293.3  2.111  2.111

```









## 6.3 Fundamentals

```

# Fundamental signals from Compustat Annual
if 'pstann' in testable:
    columns = ['absacc', 'acc', 'agr', 'bm', 'cashpr', 'cfp', 'chcsho',
               'chinv', 'depr', 'dy', 'egr', 'ep', 'gma', 'grcapx',
               'grltnoa', 'hire', 'invest', 'lev', 'lgr',
               'pchdepr', 'pchgpm_pchsale', 'pchquick',
               'pchsale_pchinvt', '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

    if regenerate:
        # 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', 'inv', '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'])

```

(continues on next page)

(continued from previous page)

```

.dropna(subset=['ib'])

fund.index = list(zip(fund['permno'], fund['datadate'])) # multi-index
fund['rebaldate'] = bd.endmo(fund.datadate, numlag)

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

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

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

fund['bm'] = fund['ceq'] / fund['mve_f']
fund['cashpr'] = (fund['mve_f'] + fund['dltt'] - fund['at'])/fund['che']
fund['depr'] = fund['dp'] / fund['ppent']
fund['dy'] = fund['dvt'] / fund['mve_f']
fund['ep'] = fund['ib'] / fund['mve_f']
fund['lev'] = fund['lt'] / fund['mve_f']
fund['quick'] = (fund['act'] - fund['invt']) / fund['lct']
fund['rd_sale'] = fund['xrd'] / fund['sale']
fund['rd_mve'] = fund['xrd'] / fund['mve_f']
fund['realestate'] = ((fund['fatb'] + fund['fatl']) /
                      np.where(fund['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']))

```

(continues on next page)

(continued from previous page)

```

    / (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['avg_at']
fund['hire'] = ((fund['emp'] / lag['emp']) - 1).fillna(0)
fund['invest'] = (((fund['ppegt'] - lag['ppegt'])
                  + (fund['invt'] - lag['invt'])) / lag['at']))
fund['invest'] = fund['invest'] \
    .where(fund['invest'].notna(),
           ((fund['ppent'] - lag['ppent'])
            + (fund['invt'] - lag['invt'])) / lag['at']))
fund['lgr'] = (fund['lt'] / lag['lt'])
fund['pchdepr'] = ((fund['dp'] / fund['ppent'])
                    / (lag['dp'] / lag['ppent']))
fund['pchgm_pchsale'] = (((fund['sale'] - fund['cogs'])
                           / (lag['sale'] - lag['cogs']))
                           - (fund['sale'] / lag['sale'])))
fund['pchquick'] = (((fund['act'] - fund['invt']) / fund['lct'])
                     / ((lag['act'] - lag['invt']) / lag['lct']))
fund['pchsale_pchinvt'] = ((fund['sale'] / lag['sale'])
                            - (fund['invt'] / lag['invt']))
fund['pchsale_pchrect'] = ((fund['sale'] / lag['sale'])
                            - (fund['rect'] / lag['rect']))
fund['pchsale_pchxsga'] = ((fund['sale'] / lag['sale'])
                            - (fund['xsga'] / lag['xsga']))
fund['pchsaleinv'] = ((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']))

```

(continues on next page)

(continued from previous page)

```

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 num, label in enumerate(columns):
    holdings = univariate_sorts(crsp,
                                 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,
                               crsp,
                               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						↳
absacc(-)	0.014	0.1	0.029	0.209	-1.172	0.242	0.952	162.665
↳ 208.556	0.951	0.953						
	excess	sharpe	alpha	appraisal	welch-t	welch-p	turnover	longs
↳ shorts	buys	sells						↳
acc(-)	0.037	0.343	0.038	0.352	-1.204	0.229	0.961	189.726
↳ 287	0.952	0.97						

(continues on next page)

(continued from previous page)

	excess	sharpe	alpha	appraisal	welch-t	welch-p	turnover	longs	shorts
	↳ shorts	buys	sells						
agr(-)	0.043	0.374	0.06	0.552	-1.189	0.235	1.047	161.904	251.
↳ 009	1.039	1.054							
bm	0.046	0.292	0.044	0.282	-1.867	0.062	0.608	153.466	249.702
↳ 0.601	0.616								
cashpr(-)	0.042	0.338	0.051	0.421	-0.787	0.432	0.62	153.057	2
↳ 189.074	0.616	0.625							
cfp	0.038	0.268	0.053	0.383	-0.075	0.94	0.914	130.106	254.
↳ 973	0.908	0.921							
chcsho(-)	0.064	0.625	0.079	0.82	-0.864	0.388	1.034	161.005	2
↳ 213.608	1.02	1.048							
chinv(-)	0.043	0.401	0.054	0.511	-0.367	0.713	1.031	148.739	2
↳ 182.964	1.023	1.039							
depr	0.005	0.033	-0.024	-0.184	0.577	0.564	0.432	243.117	132.
↳ 209	0.425	0.44							
dy	-0.004	-0.021	0.041	0.242	-0.47	0.638	0.545	131.77	603.707
↳ 0.56	0.53								
egr(-)	0.042	0.376	0.059	0.557	-0.687	0.492	0.989	166.847	235.
↳ 175	0.981	0.996							
ep	0.044	0.276	0.061	0.401	-1.521	0.129	0.888	150.331	247.447
↳ 0.882	0.893								
gma	0.007	0.052	0.011	0.08	0.77	0.442	0.47	219.987	241.
↳ 987	0.464	0.475							
grcapx	-0.032	-0.313	-0.042	-0.426	1.449	0.148	1.01	207.441	159.
↳ 746	1.015	1.004							
grltnoa	0.02	0.172	0.022	0.188	-0.023	0.981	1.05	156.709	2
↳ 154.89	1.046	1.054							
hire	-0.021	-0.182	-0.036	-0.333	1.19	0.235	1.023	260.129	155.
↳ 052	1.024	1.021							

(continues on next page)

(continued from previous page)

	excess	sharpe	alpha	appraisal	welch-t	welch-p	turnover	longs
	↳ shorts	buys	sells					
invest(-)	0.038	0.331	0.051	0.449	-0.937	0.349	0.994	139.802
↳ 194.005	0.988	1.001						
	excess	sharpe	alpha	appraisal	welch-t	welch-p	turnover	longs
	↳ shorts	buys	sells					
lev	0.01	0.061	0.006	0.036	-1.507	0.132	0.445	215.107
↳ 592	0.447	0.443						
	excess	sharpe	alpha	appraisal	welch-t	welch-p	turnover	longs
	↳ shorts	buys	sells					
lgr	-0.017	-0.193	-0.027	-0.304	1.166	0.244	1.108	237.438
↳ 394	1.11	1.105						
	excess	sharpe	alpha	appraisal	welch-t	welch-p	turnover	longs
	↳ shorts	buys	sells					
pchdepr	0.003	0.038	0.004	0.039	-1.658	0.098	1.139	202.545
↳ 200.088	1.139	1.139						

```
/home/terence/Dropbox/github/data-science-notebooks/finds/backtesting/backtest.
↳ py:303: 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,
```

	excess	sharpe	alpha	appraisal	welch-t	welch-p	turnover	
	↳ longs	shorts	buys	sells				
pchgm_pchsale	0.004	0.043	-0.0	-0.001	-0.107	0.915	1.072	
↳ 146	181.539	1.071	1.074					
	excess	sharpe	alpha	appraisal	welch-t	welch-p	turnover	longs
	↳ shorts	buys	sells					
pchquick	-0.011	-0.128	-0.012	-0.139	0.299	0.765	1.164	170.091
↳ 172.706	1.167	1.161						
	excess	sharpe	alpha	appraisal	welch-t	welch-p	turnover	longs
	↳ longs	shorts	buys	sells				
pchsale_pchinv	0.028	0.311	0.028	0.307	-1.784	0.075	1.09	183.
↳ 266	173.726	1.083	1.097					
	excess	sharpe	alpha	appraisal	welch-t	welch-p	turnover	longs
	↳ longs	shorts	buys	sells				
pchsale_pchrect	-0.004	-0.048	-0.006	-0.072	-2.864	0.004	1.151	192.
↳ 844	193.427	1.152	1.15					
	excess	sharpe	alpha	appraisal	welch-t	welch-p	turnover	longs
	↳ longs	shorts	buys	sells				
pchsale_pchxsga	-0.011	-0.102	-0.013	-0.116	-0.669	0.504	1.103	166.
↳ 288	152.038	1.106	1.1					
	excess	sharpe	alpha	appraisal	welch-t	welch-p	turnover	longs
	↳ shorts	buys	sells					
pchsaleinv	0.019	0.223	0.018	0.209	-1.912	0.056	1.085	176.961
↳ 171.734	1.08	1.089						
	excess	sharpe	alpha	appraisal	welch-t	welch-p	turnover	longs
	↳ shorts	buys	sells					
pctacc(-)	0.013	0.121	0.025	0.234	0.742	0.459	1.011	152.658
↳ 176.584	1.009	1.014						
	excess	sharpe	alpha	appraisal	welch-t	welch-p	turnover	longs
	↳ shorts	buys	sells					

(continues on next page)

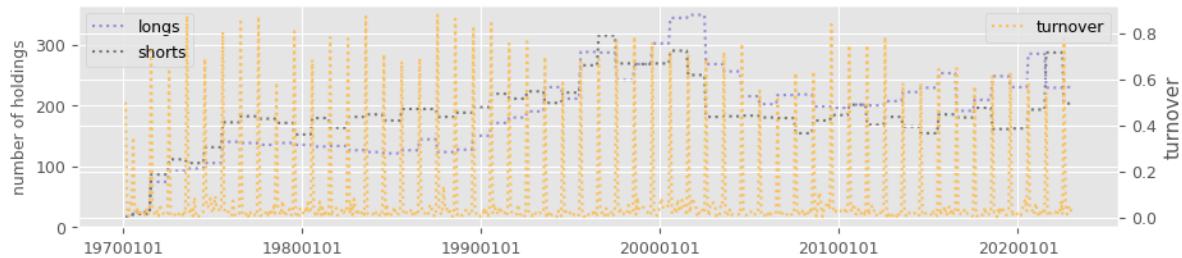
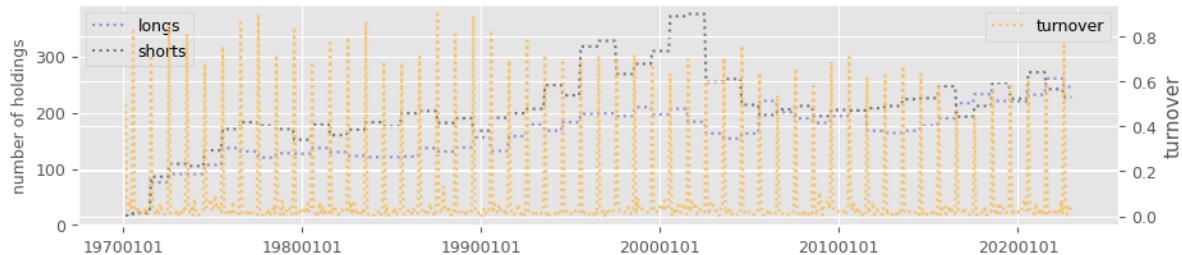
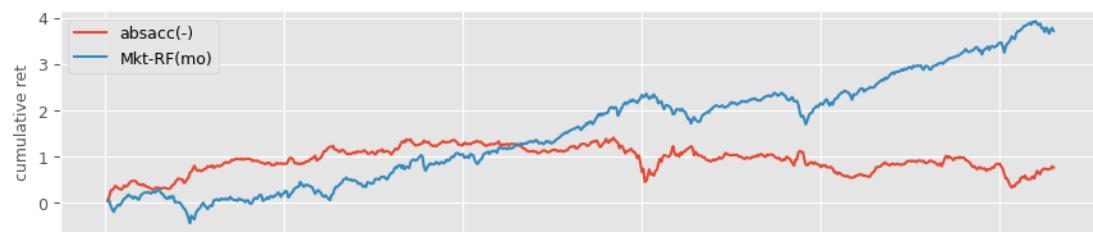
(continued from previous page)

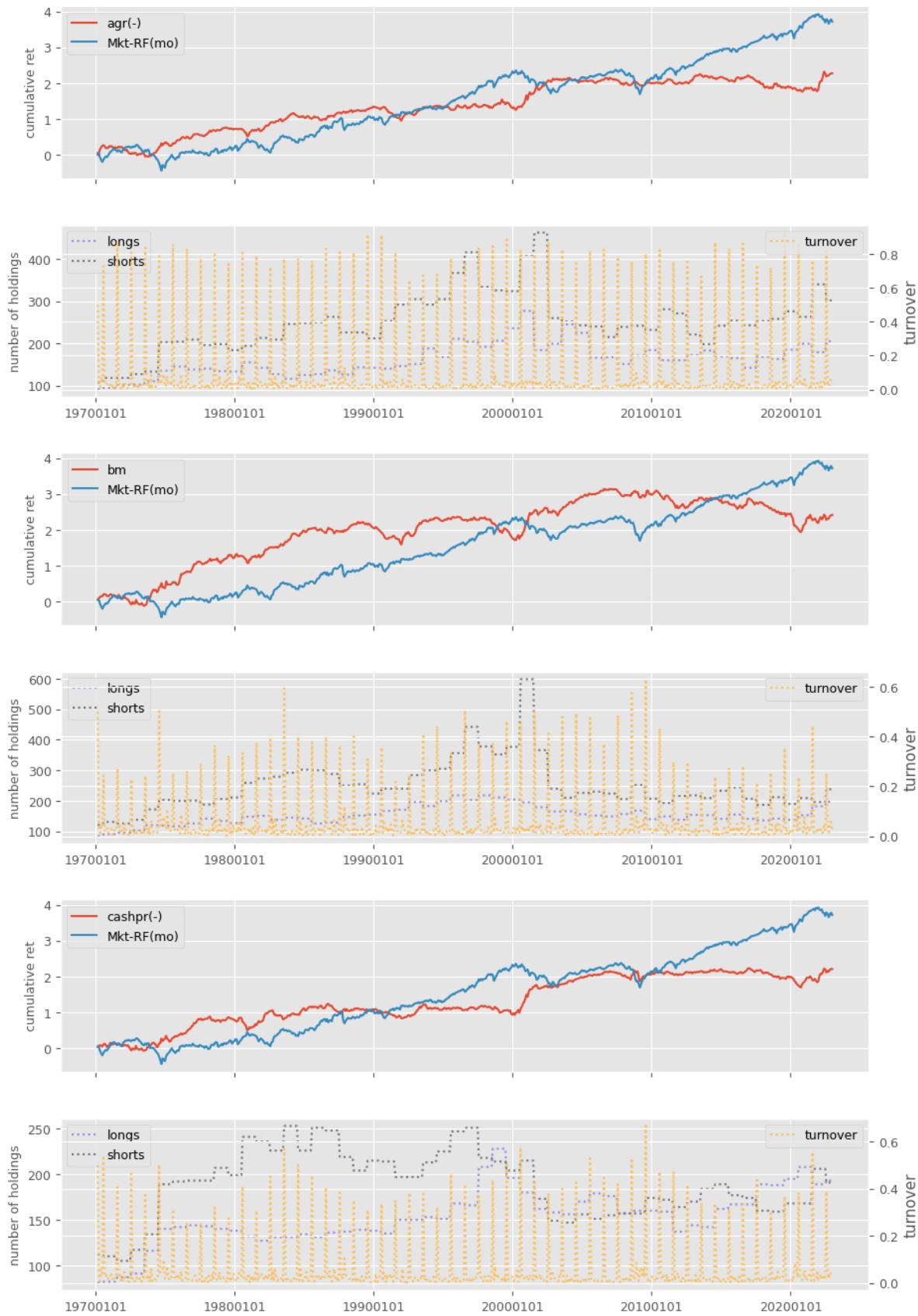
quick	-0.003	-0.019	-0.034	-0.24	0.517	0.605	0.564	310.723	123.
↳ 016	0.56	0.569							
	excess	sharpe	alpha	appraisal	welch-t	welch-p	turnover	longs	↳
	↳ shorts	buys	sells						
rd_sale	-0.014	-0.09	-0.02	-0.127	0.284	0.776	0.41	209.909	↳
↳ 135.71	0.414	0.407							
	excess	sharpe	alpha	appraisal	welch-t	welch-p	turnover	longs	↳
	↳ shorts	buys	sells						
rd_mve	0.025	0.154	0.014	0.09	0.367	0.714	0.537	120.745	136.
↳ 816	0.533	0.541							
	excess	sharpe	alpha	appraisal	welch-t	welch-p	turnover	longs	↳
	↳ shorts	buys	sells						
realestate	0.01	0.064	0.042	0.288	-1.422	0.156	0.553	90.98	↳
↳ 133.424	0.551	0.555							
	excess	sharpe	alpha	appraisal	welch-t	welch-p	turnover	longs	↳
	↳ shorts	buys	sells						
salecash	0.002	0.02	0.019	0.17	0.357	0.721	0.589	145.907	↳
↳ 358.499	0.59	0.587							
	excess	sharpe	alpha	appraisal	welch-t	welch-p	turnover	longs	↳
	↳ shorts	buys	sells						
salerec	0.025	0.195	0.039	0.313	0.929	0.354	0.393	176.923	↳
↳ 242.85	0.385	0.401							
	excess	sharpe	alpha	appraisal	welch-t	welch-p	turnover	longs	↳
	↳ shorts	buys	sells						
saleinv	0.009	0.088	0.029	0.31	1.526	0.128	0.497	179.732	↳
↳ 150.178	0.496	0.497							
	excess	sharpe	alpha	appraisal	welch-t	welch-p	turnover	longs	↳
	↳ shorts	buys	sells						
secured(-)	0.004	0.031	0.031	0.26	0.625	0.532	0.605	373.131	↳
↳ 175.09	0.607	0.604							
	excess	sharpe	alpha	appraisal	welch-t	welch-p	turnover	longs	↳
	↳ shorts	buys	sells						
sgr	-0.012	-0.092	-0.026	-0.214	0.151	0.88	1.011	255.545	157.
↳ 011	1.011	1.012							
	excess	sharpe	alpha	appraisal	welch-t	welch-p	turnover	longs	shorts
	↳ buys	buys	sells						
sp	0.047	0.322	0.045	0.311	-0.521	0.602	0.54	138.791	339.017
↳ 0.531	0.549								
	excess	sharpe	alpha	appraisal	welch-t	welch-p	turnover	longs	↳
	↳ shorts	buys	sells						
tang	0.02	0.164	0.008	0.069	0.805	0.421	0.544	322.165	158.
↳ 863	0.538	0.551							
	excess	sharpe	alpha	appraisal	welch-t	welch-p	turnover	longs	↳
	↳ shorts	buys	sells						
bm_ia	0.041	0.306	0.029	0.223	-1.42	0.156	0.915	159.276	178.
↳ 055	0.906	0.924							
	excess	sharpe	alpha	appraisal	welch-t	welch-p	turnover	longs	↳
	↳ shorts	buys	sells						
cfp_ia	-0.026	-0.185	-0.035	-0.258	0.094	0.925	1.015	145.016	155.
↳ 983	1.021	1.009							
	excess	sharpe	alpha	appraisal	welch-t	welch-p	turnover	longs	↳
	↳ shorts	buys	sells						
chatoia	0.022	0.234	0.024	0.257	-1.076	0.282	1.094	173.598	↳
↳ 172.991	1.089	1.099							
	excess	sharpe	alpha	appraisal	welch-t	welch-p	turnover	longs	↳
	↳ shorts	buys	sells						

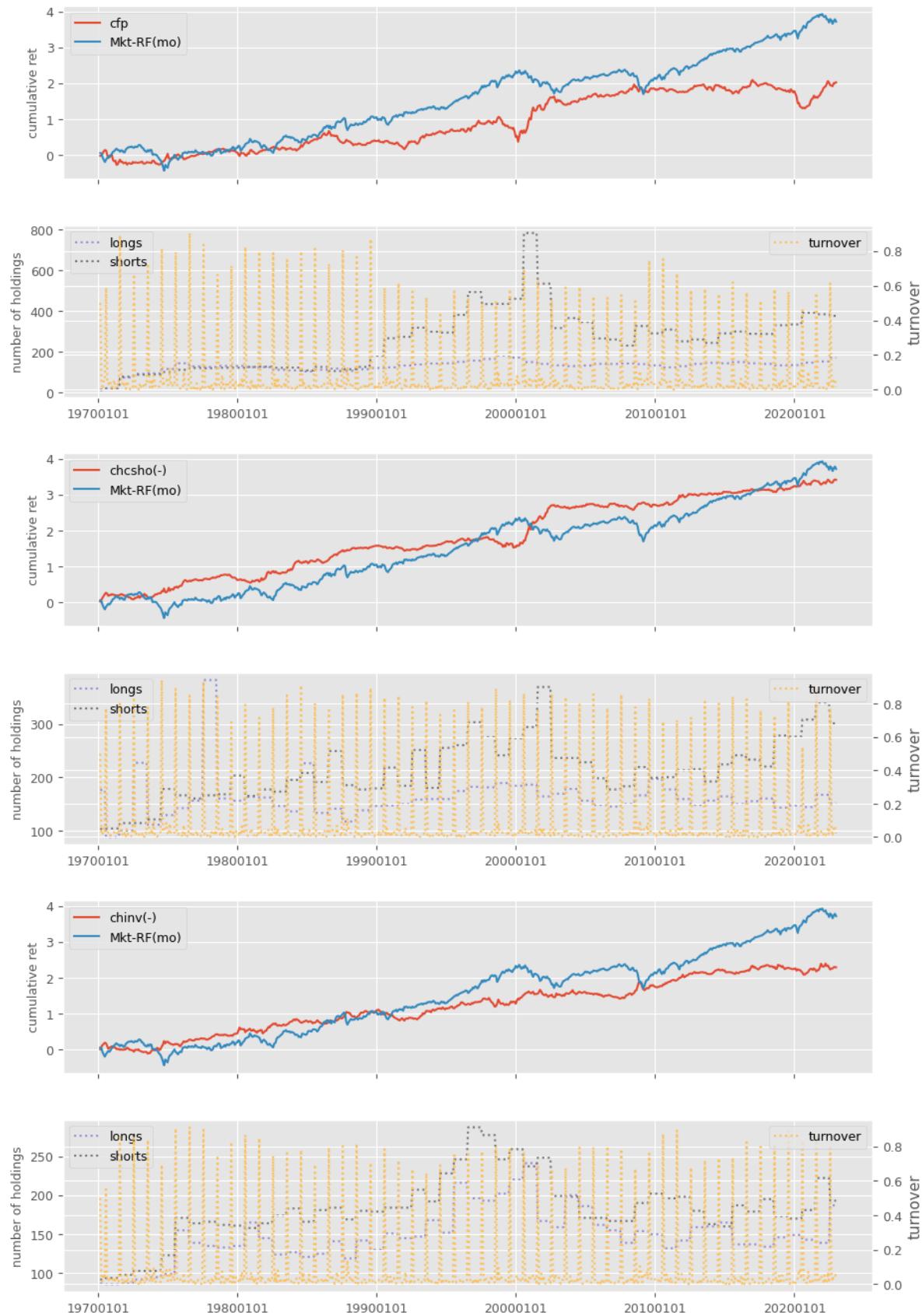
(continues on next page)

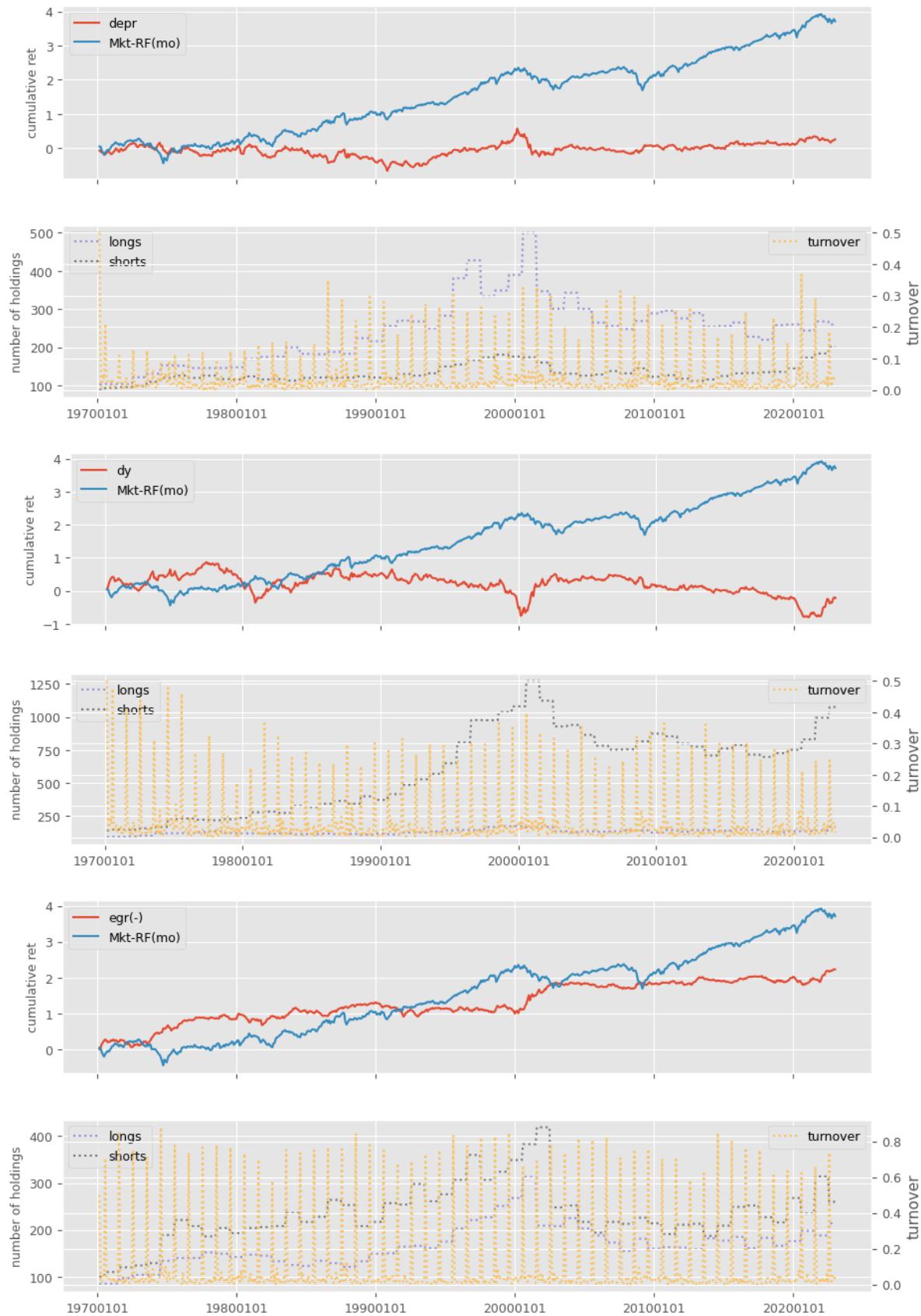
(continued from previous page)

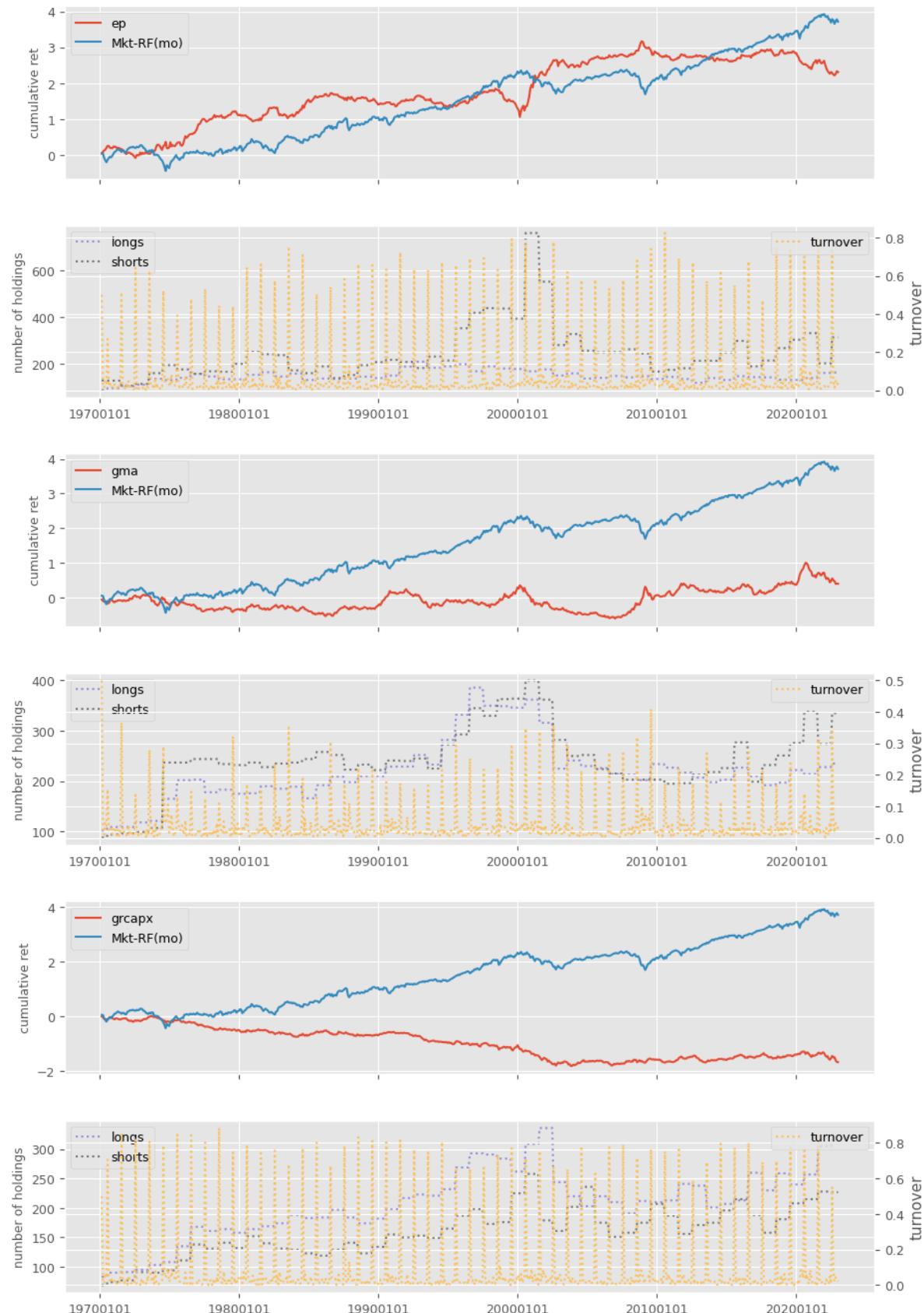
chpmia	0.022	0.167	0.025	0.196	0.023	0.981	1.158	87.027	87.
↳ 542	1.152	1.164							
↳ shorts buys sells									
pchcapx_ia	-0.031	-0.26	-0.032	-0.262	0.23	0.818	1.196	88.444	88.
↳ 78.29	1.202	1.189							
↳ shorts buys sells									
chempia	0.012	0.111	0.0	0.003	1.665	0.096	1.102	198.816	198.
↳ 144.383	1.097	1.107							
↳ shorts buys sells									
mve_ia(-)	0.019	0.177	0.008	0.08	-0.028	0.977	0.411	220.334	220.
↳ 124.131	0.405	0.417							

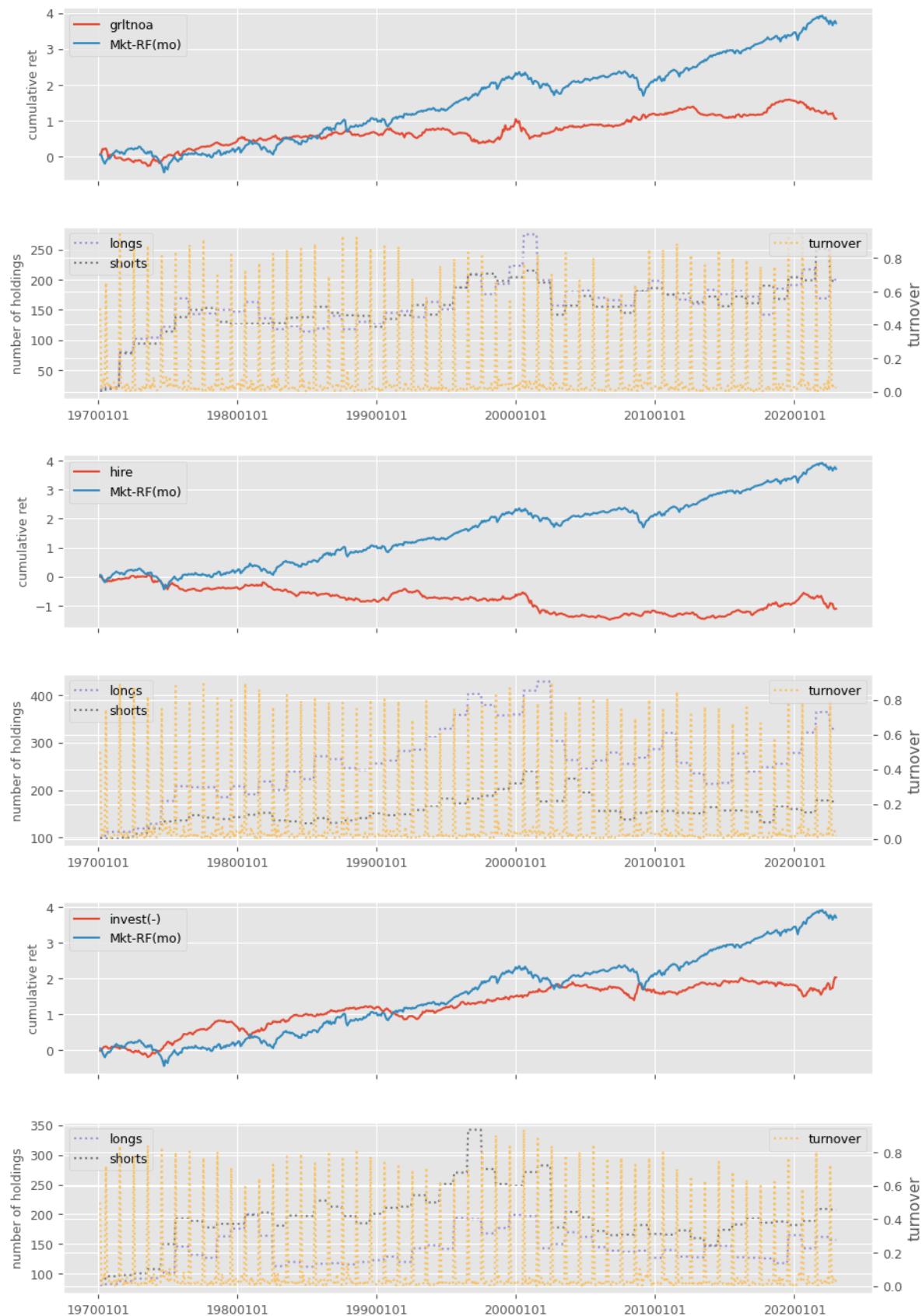


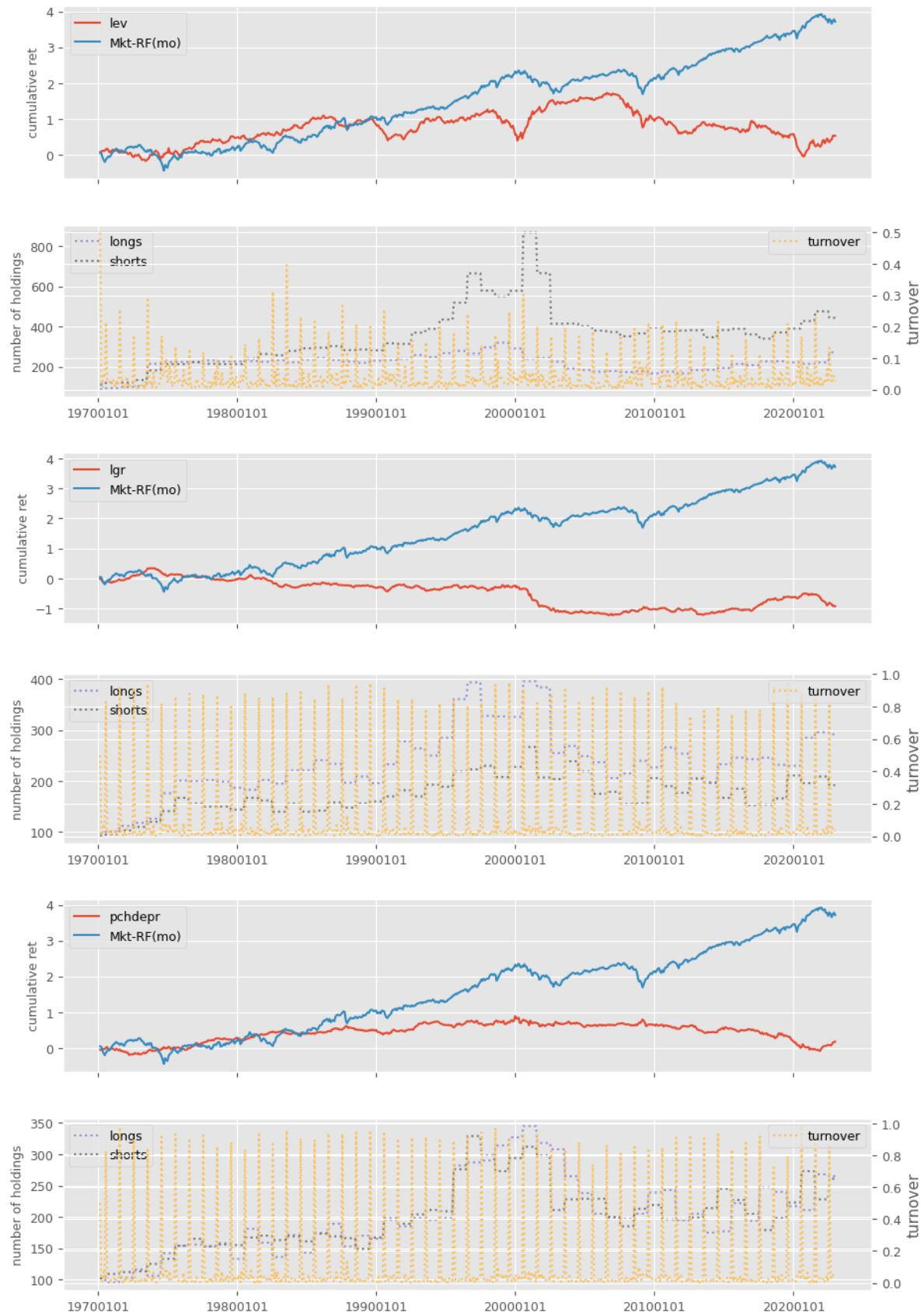


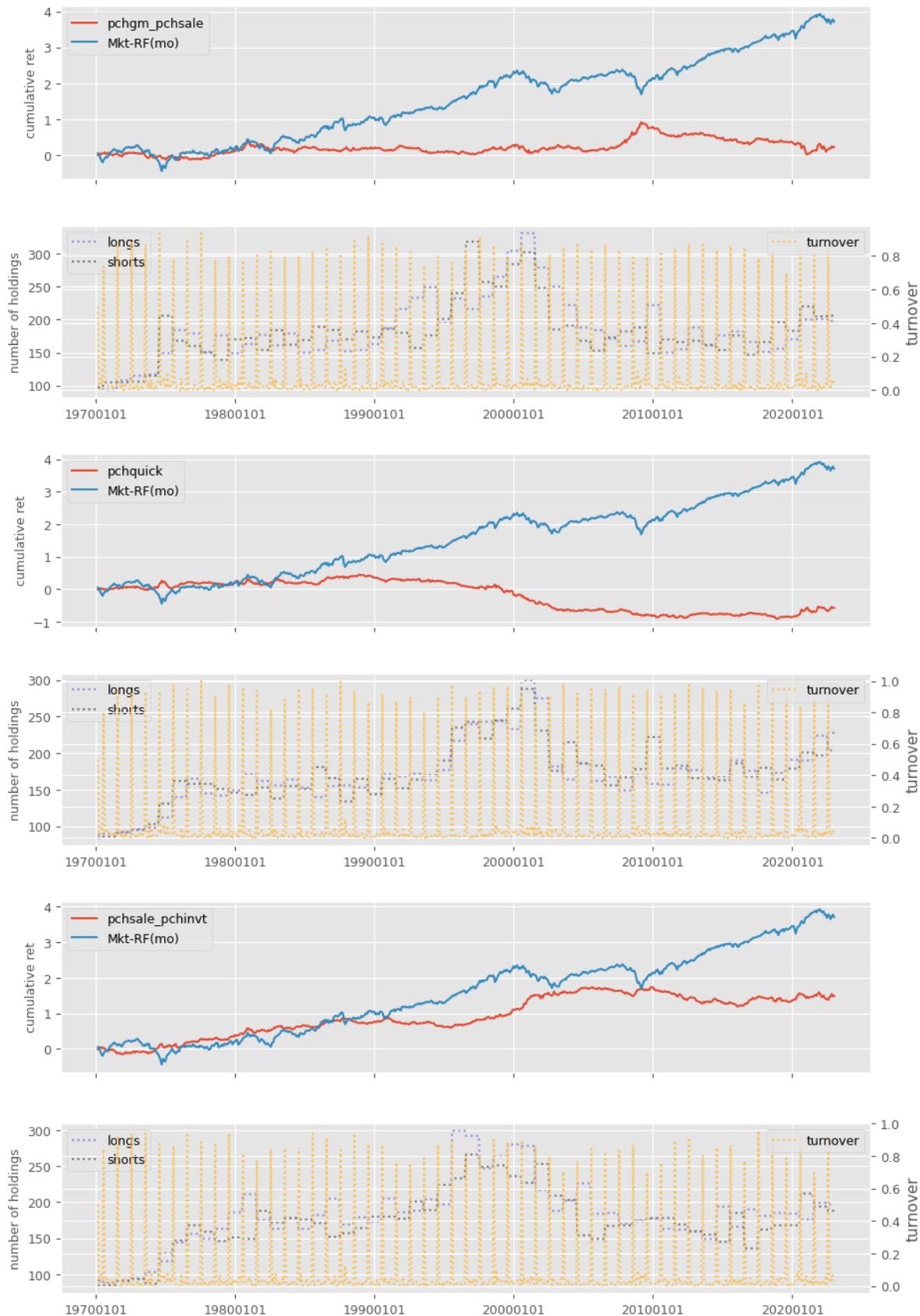


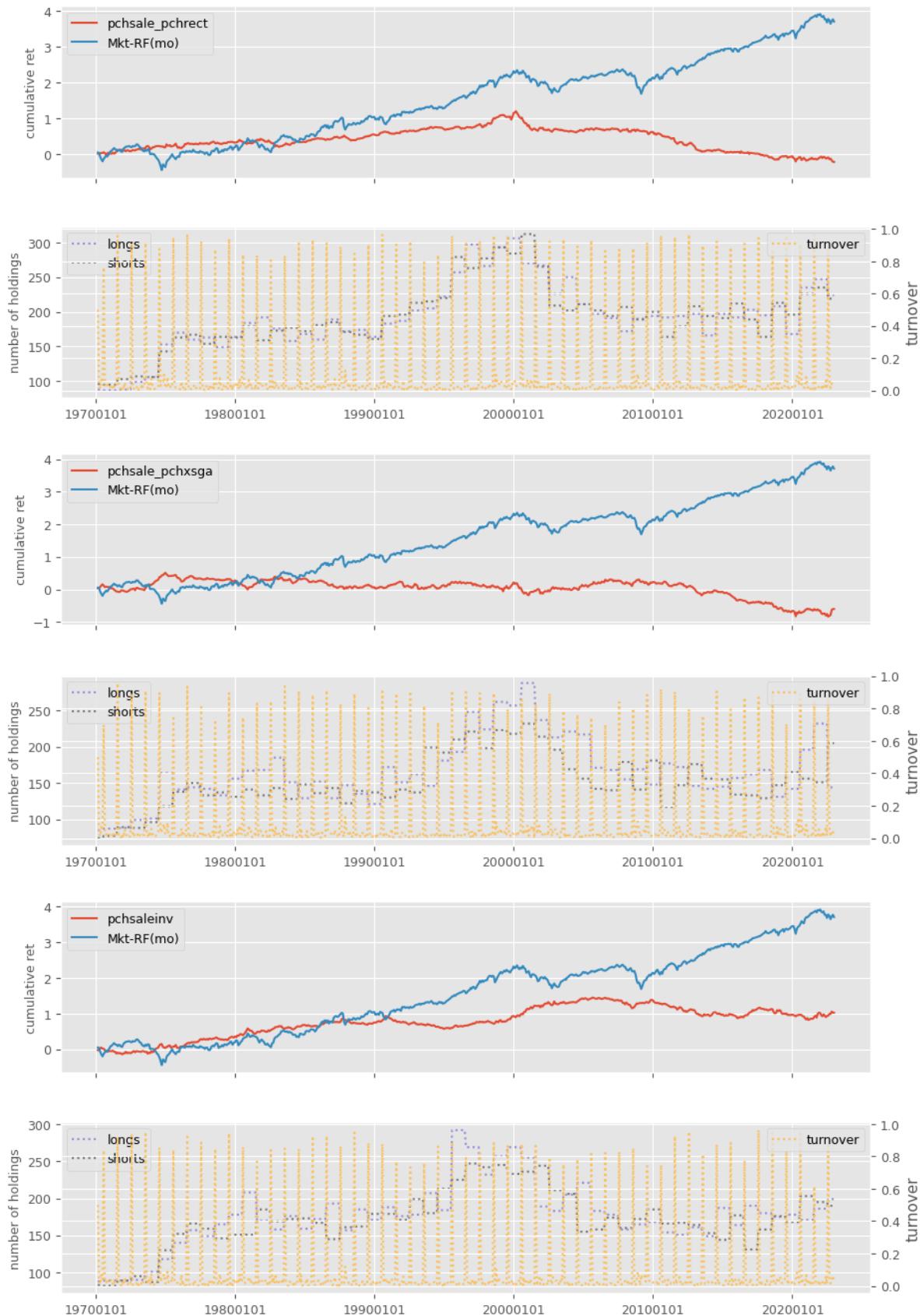


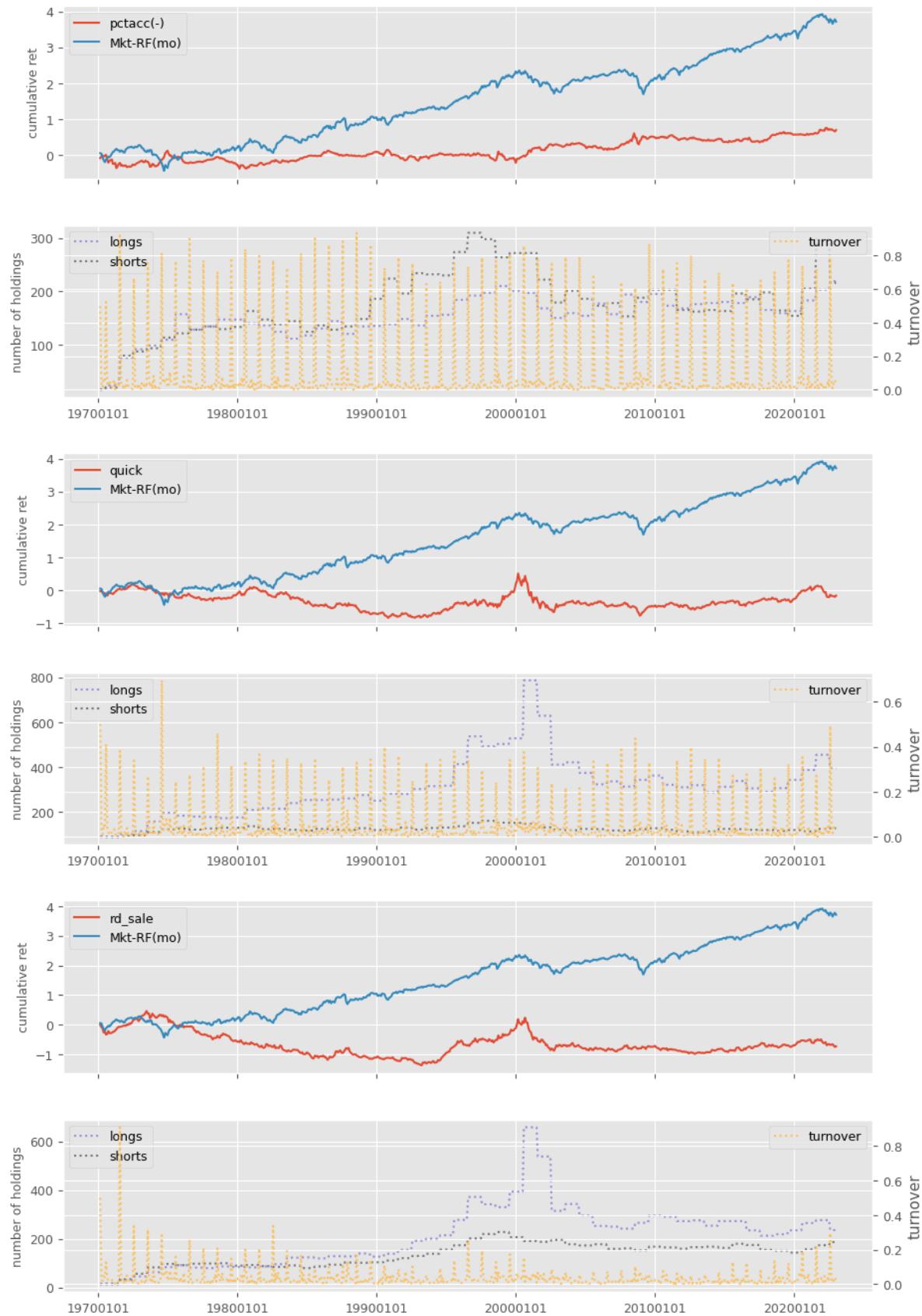


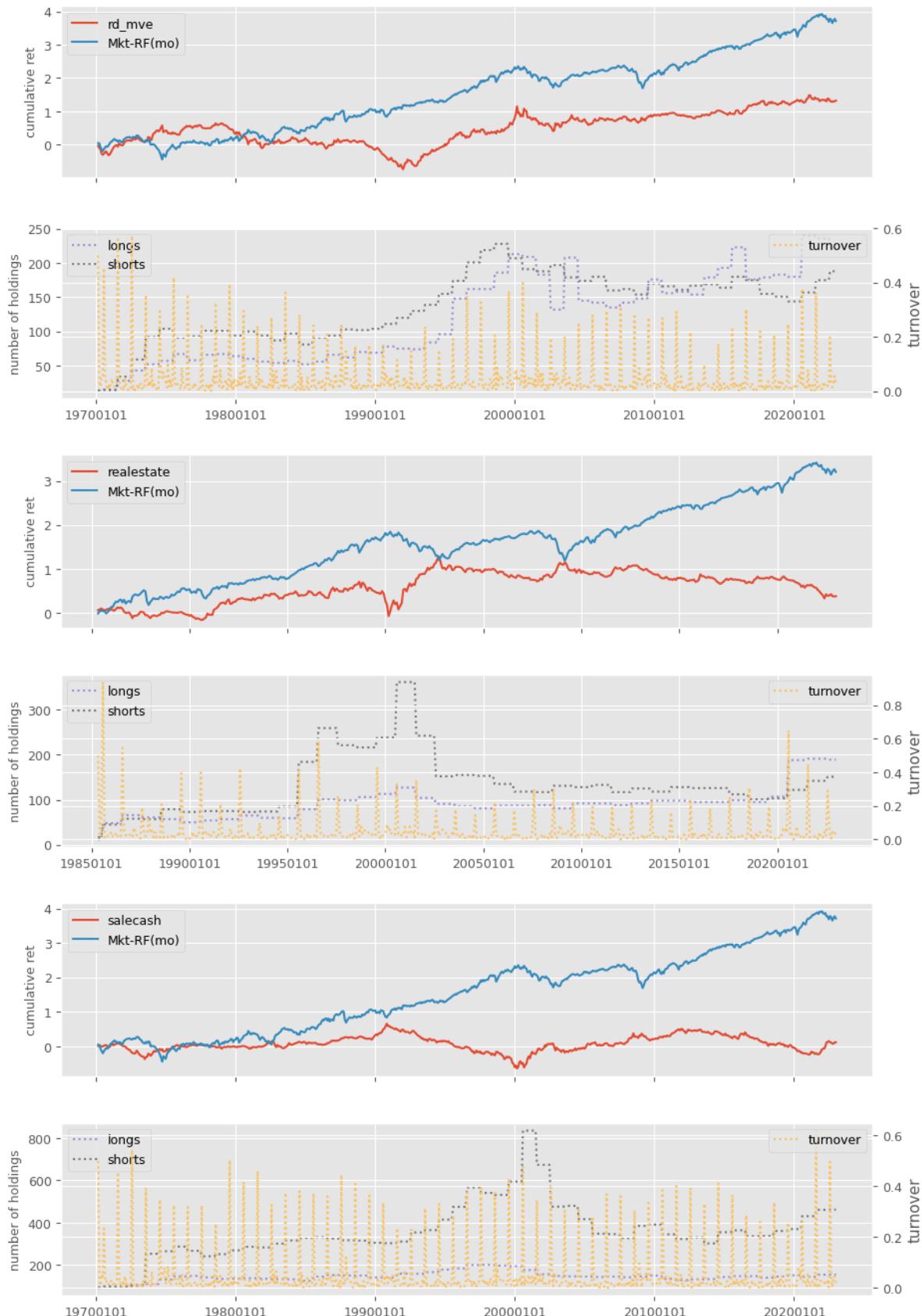


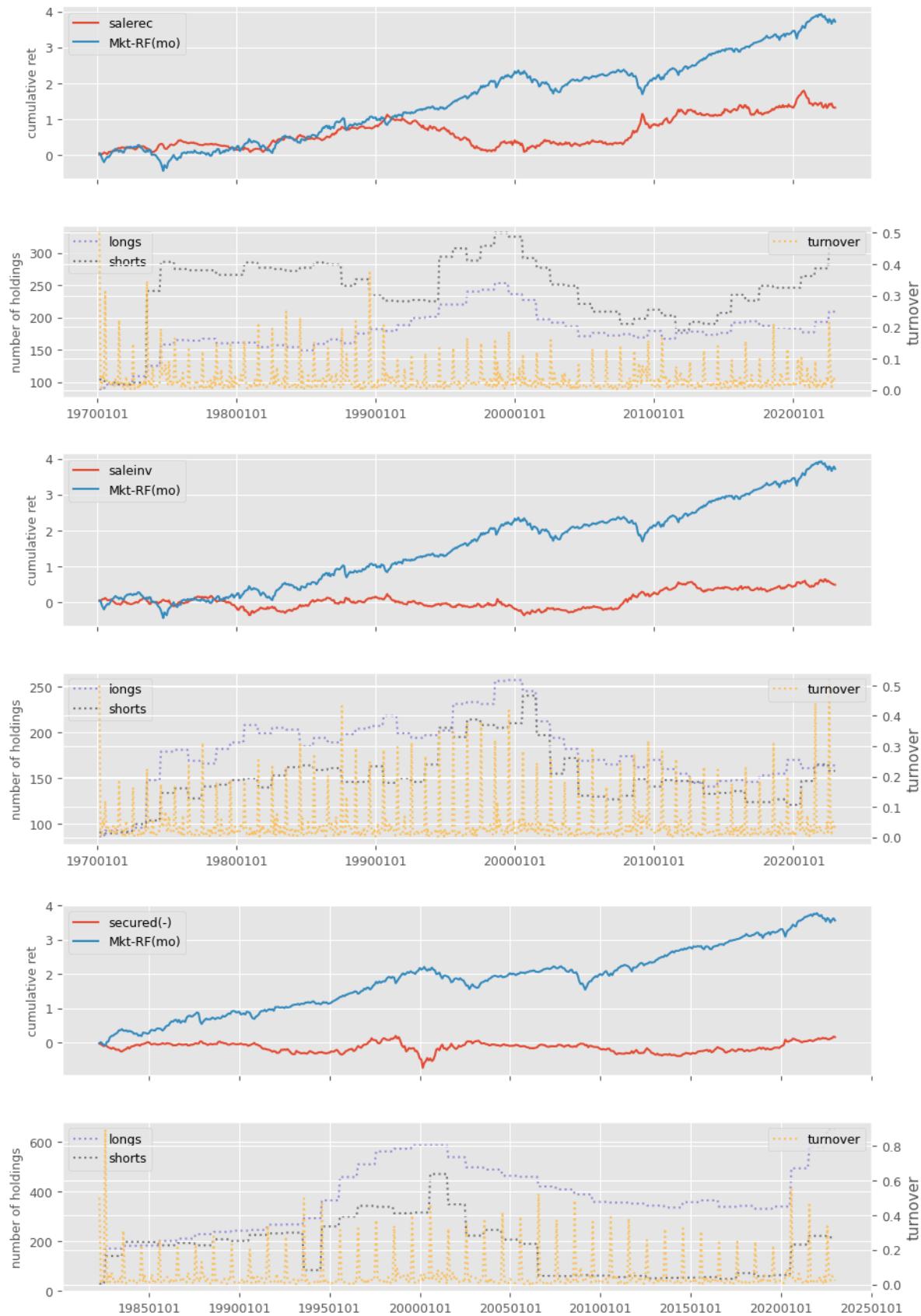


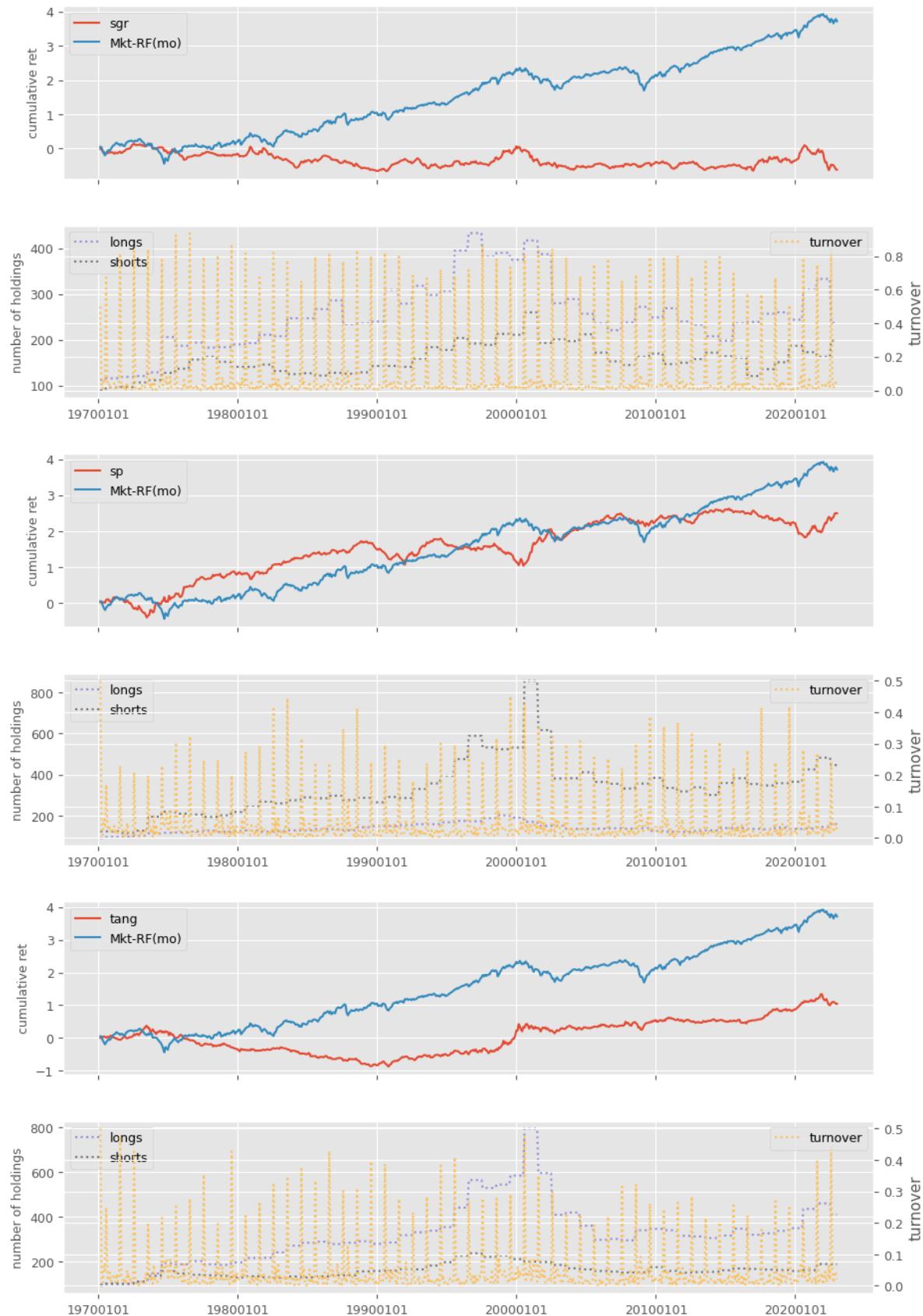


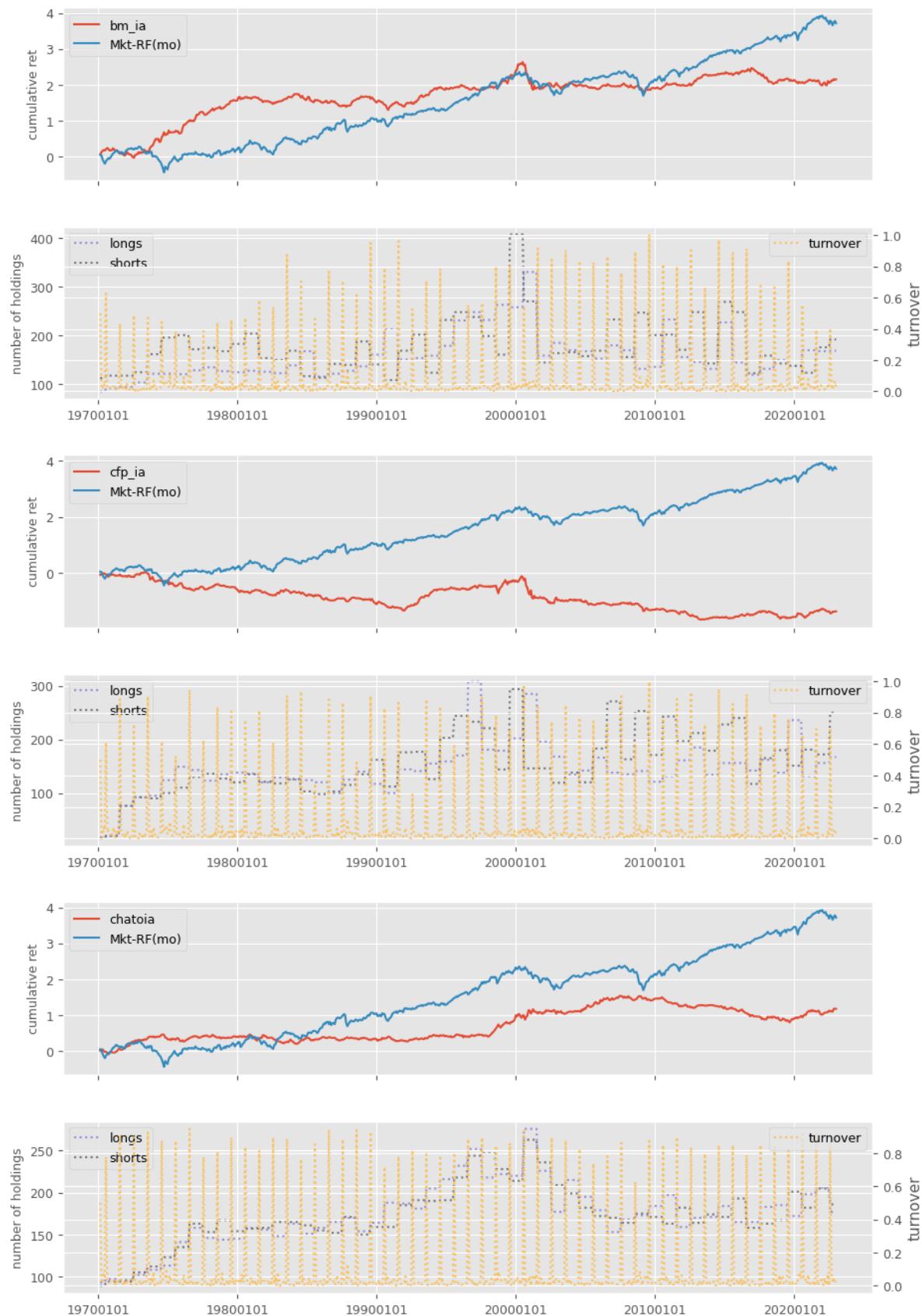


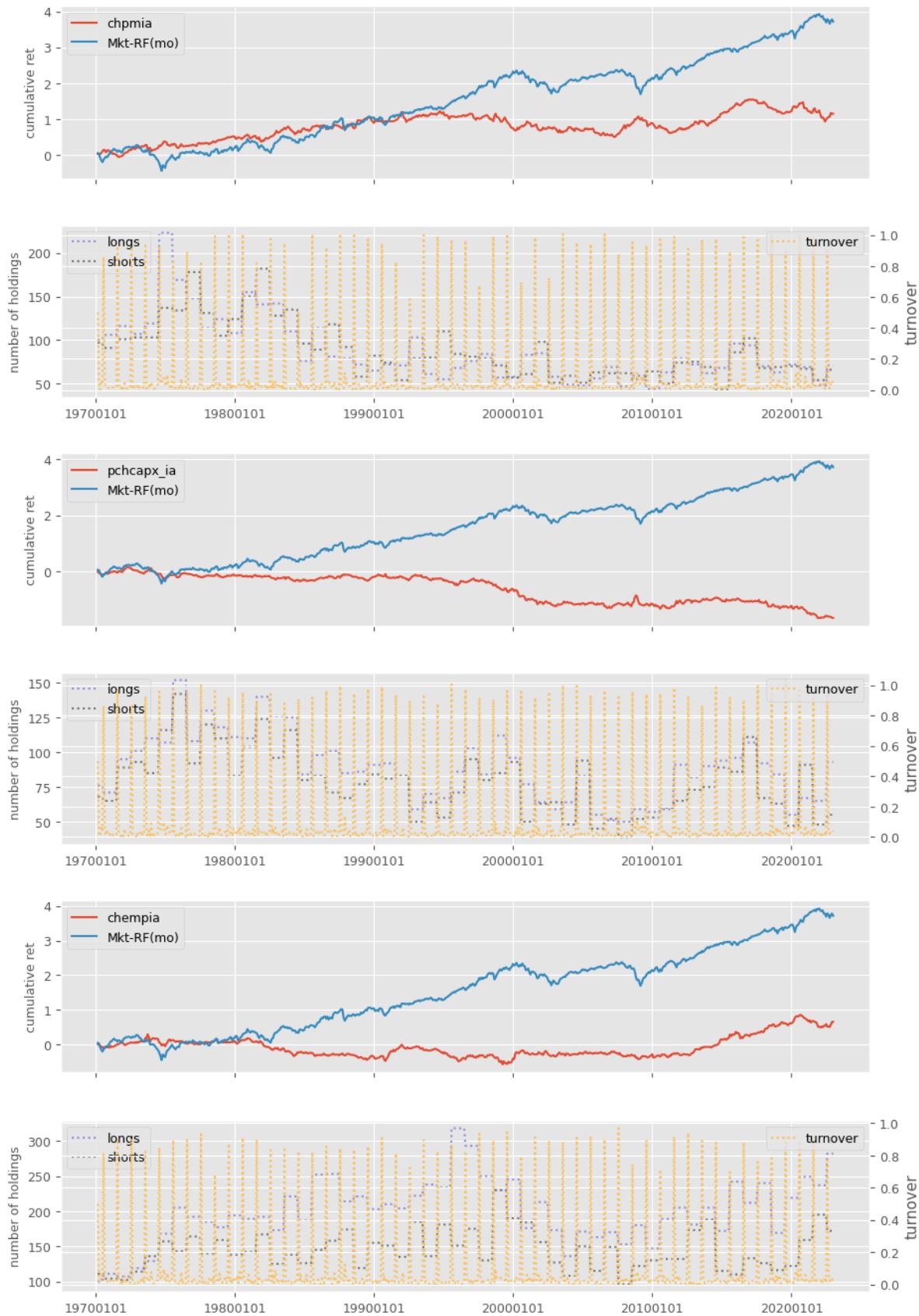


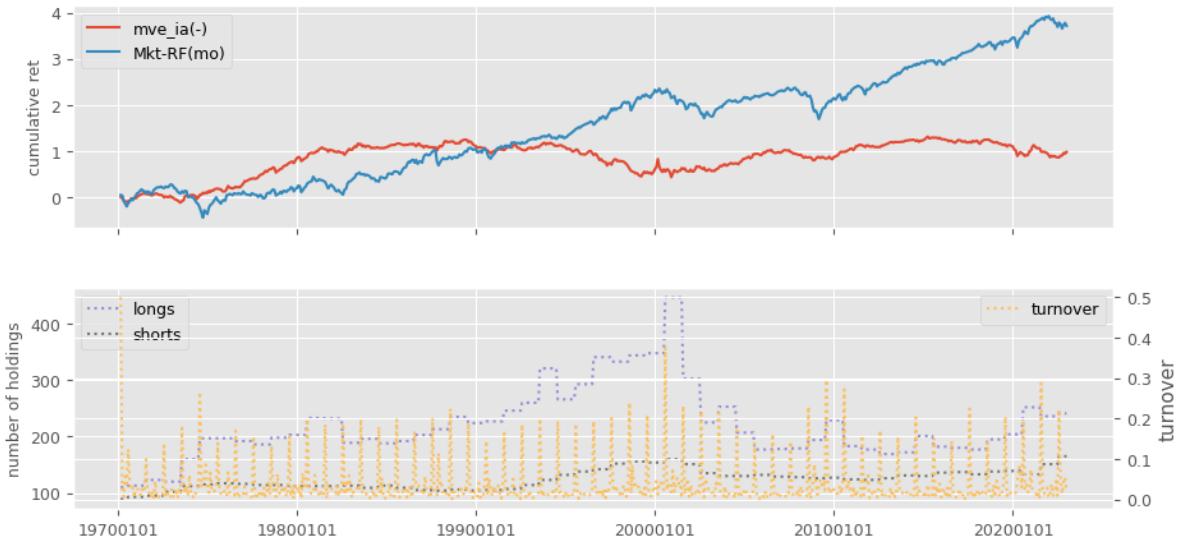












```

# Fundamental signals from Compustat Quarterly
if 'pstqtr' in testable:
    columns = ['stdacc', 'stdcf', 'roavol', 'sgrvol', 'cinvest', 'chtx',
               'rsup', 'roaq', 'cash', 'nincr']
    numlag = 4           # require 4 month lag of fiscal data
    end = LAST_DATE

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

```

(continues on next page)

(continued from previous page)

```

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)

rebalgbeg, rebalgend = 19700101, LAST_DATE
benchnames = ['Mkt-RF(mo)']
for num, label in enumerate(columns):
    holdings = univariate_sorts(crsp,
                                 label,
                                 SignalsFrame(signals.read(label)),
                                 rebalgbeg,
                                 rebalgend,
                                 window=3,
                                 months=[],
                                 maxdecile=8,
                                 pct=(10., 90.),
                                 leverage=leverage.get(label, 1))
    excess = backtest_pipeline(backtest,
                               crsp,
                               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						
stdacc(-)	0.017	0.138	0.041	0.359	-0.208	0.835	0.781	104.223
↳ 149.122	0.779	0.784						
	excess	sharpe	alpha	appraisal	welch-t	welch-p	turnover	longs
↳ shorts	buys	sells						

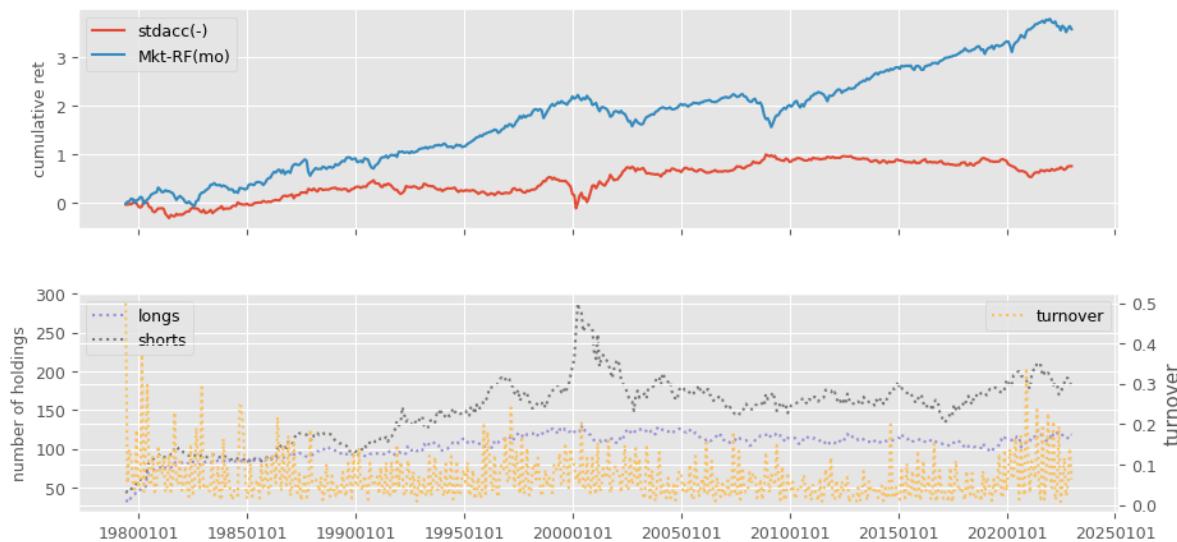
(continues on next page)

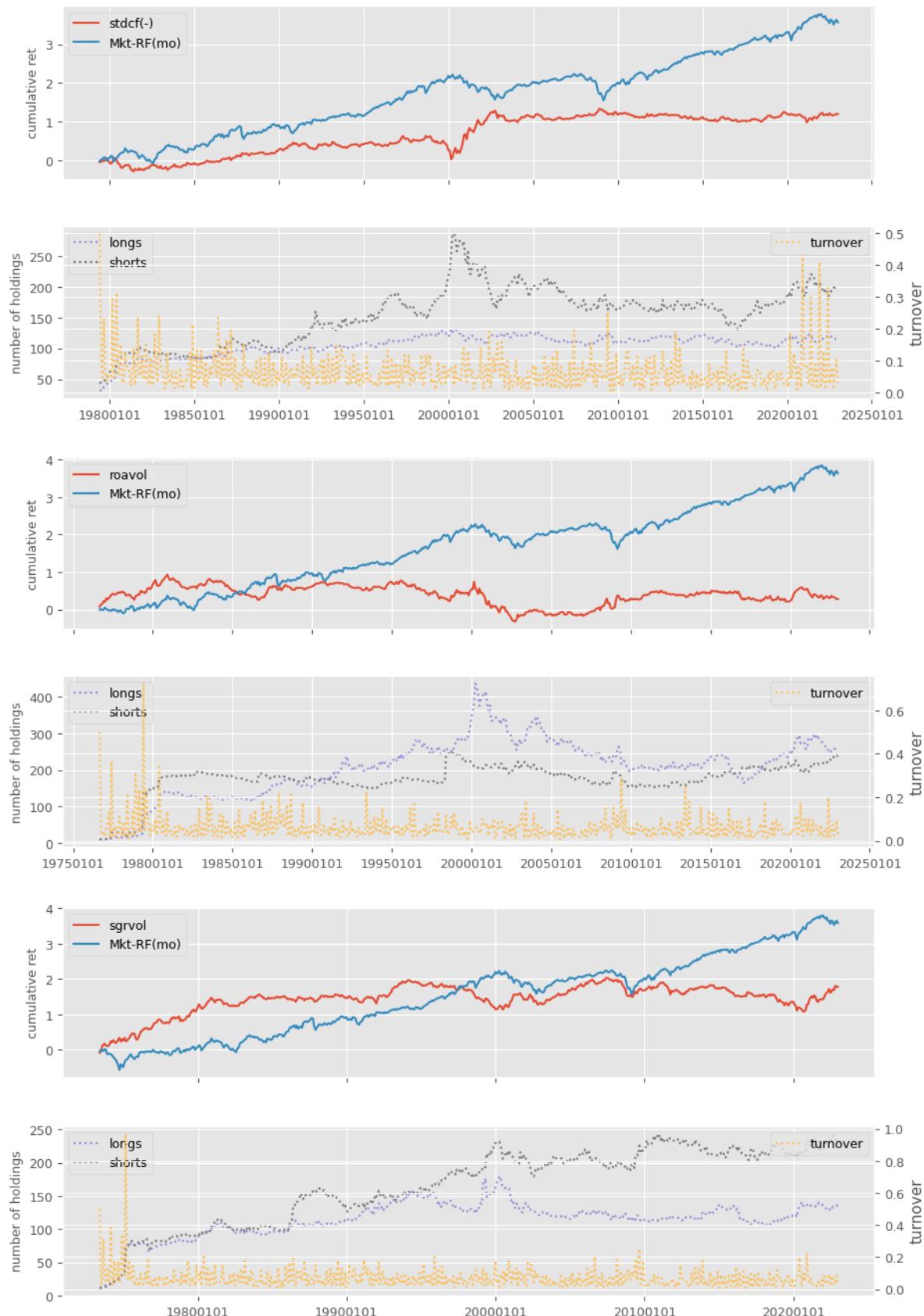
(continued from previous page)

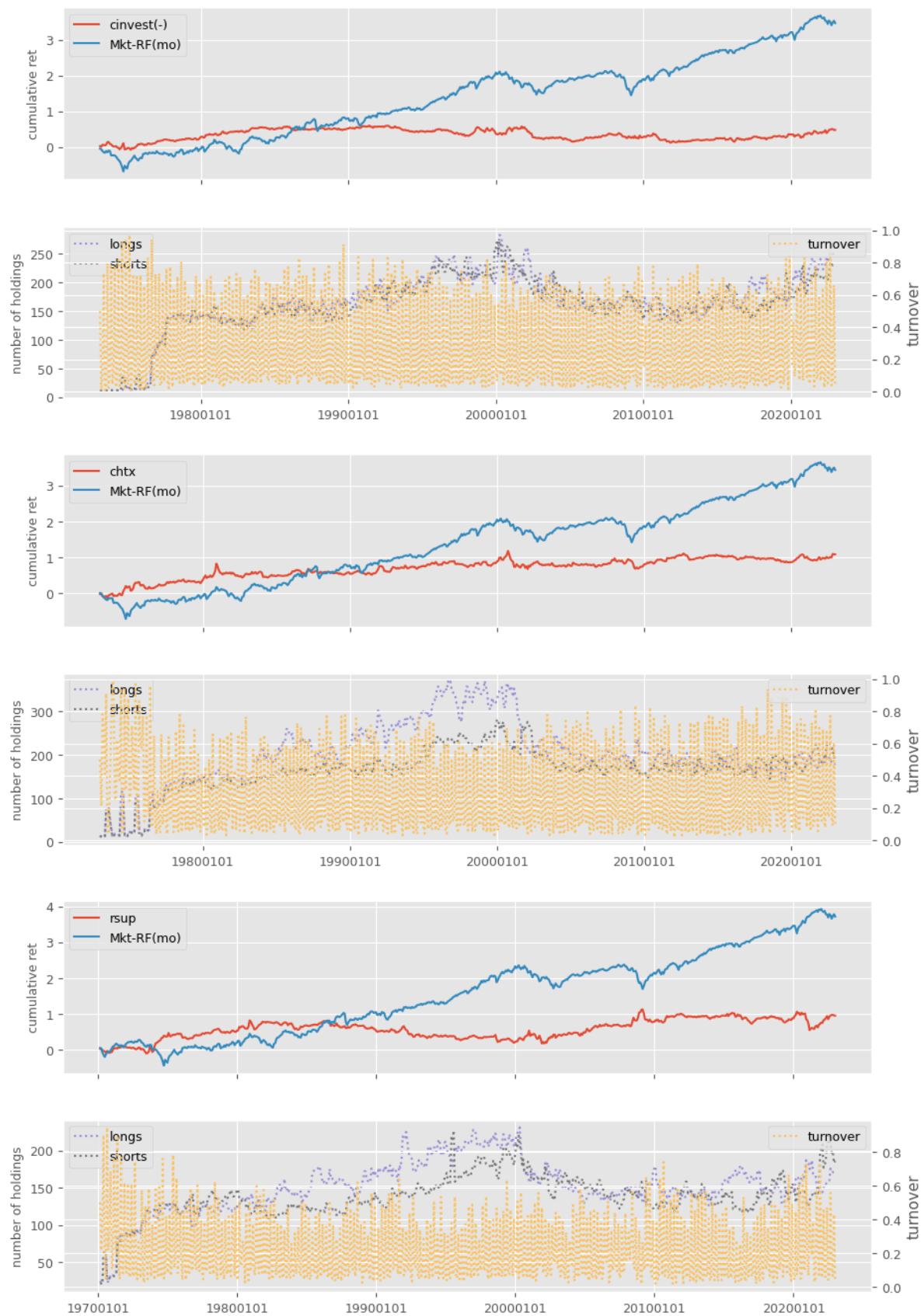
```

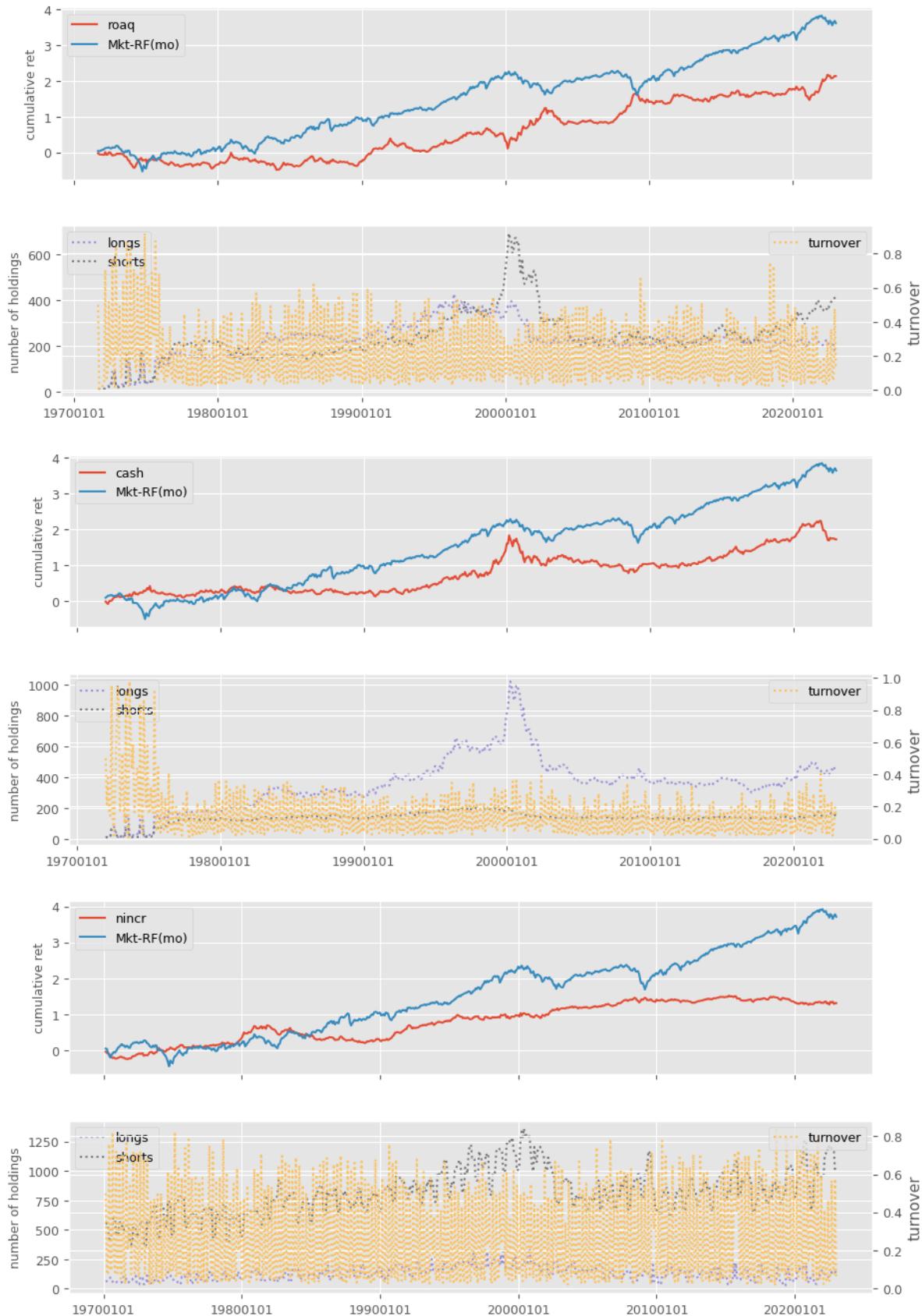
stdcf(-)  0.027  0.201  0.058      0.471  -0.678  0.498      0.755  105.559  -
↳154.275  0.751  0.76
      excess  sharpe  alpha  appraisal  welch-t  welch-p  turnover  longs  -
↳shorts  buys  sells
roavol  0.006  0.04 -0.011     -0.074  0.207  0.836      0.715  206.097  174.
↳855  0.709  0.72
      excess  sharpe  alpha  appraisal  welch-t  welch-p  turnover  longs  -
↳shorts  buys  sells
sgrvol  0.036  0.238  0.016      0.112  -0.768  0.443      0.777  114.173  165.
↳604  0.769  0.786
      excess  sharpe  alpha  appraisal  welch-t  welch-p  turnover  longs  -
↳shorts  buys  sells
cinvest(-)  0.01  0.109  0.011      0.132  -0.769  0.442      3.239  163.501  -
↳160.214  3.237  3.241
      excess  sharpe  alpha  appraisal  welch-t  welch-p  turnover  longs  -
↳shorts  buys  sells
chtx  0.022  0.18  0.016      0.136  -0.309  0.757      3.427  201.431  165.
↳623  3.419  3.434
      excess  sharpe  alpha  appraisal  welch-t  welch-p  turnover  longs  -
↳shorts  buys  sells
rsup  0.018  0.14  0.021      0.16  0.755  0.451      2.521  151.348  138.
↳096  2.514  2.527
      excess  sharpe  alpha  appraisal  welch-t  welch-p  turnover  longs  -
↳shorts  buys  sells
roaq  0.042  0.287  0.06      0.43  0.944  0.346      2.383  223.453  239.
↳357  2.372  2.395
      excess  sharpe  alpha  appraisal  welch-t  welch-p  turnover  longs  -
↳shorts  buys  sells
cash  0.034  0.224  0.008      0.061  -0.408  0.683      1.598  362.036  138.
↳008  1.585  1.61
      excess  sharpe  alpha  appraisal  welch-t  welch-p  turnover  longs  -
↳shorts  buys  sells
nincr  0.025  0.28  0.019      0.217  -0.692  0.489      3.073  130.677  814.
↳797  3.065  3.081

```









## 6.4 Earnings Estimates

```

# IBES Fiscal Year 1 signals: chfeps, chnanalyst, disp
if 'ibesf1' in testable:
    columns = ['chfeps', 'chnanalyst', 'disp']

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

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

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

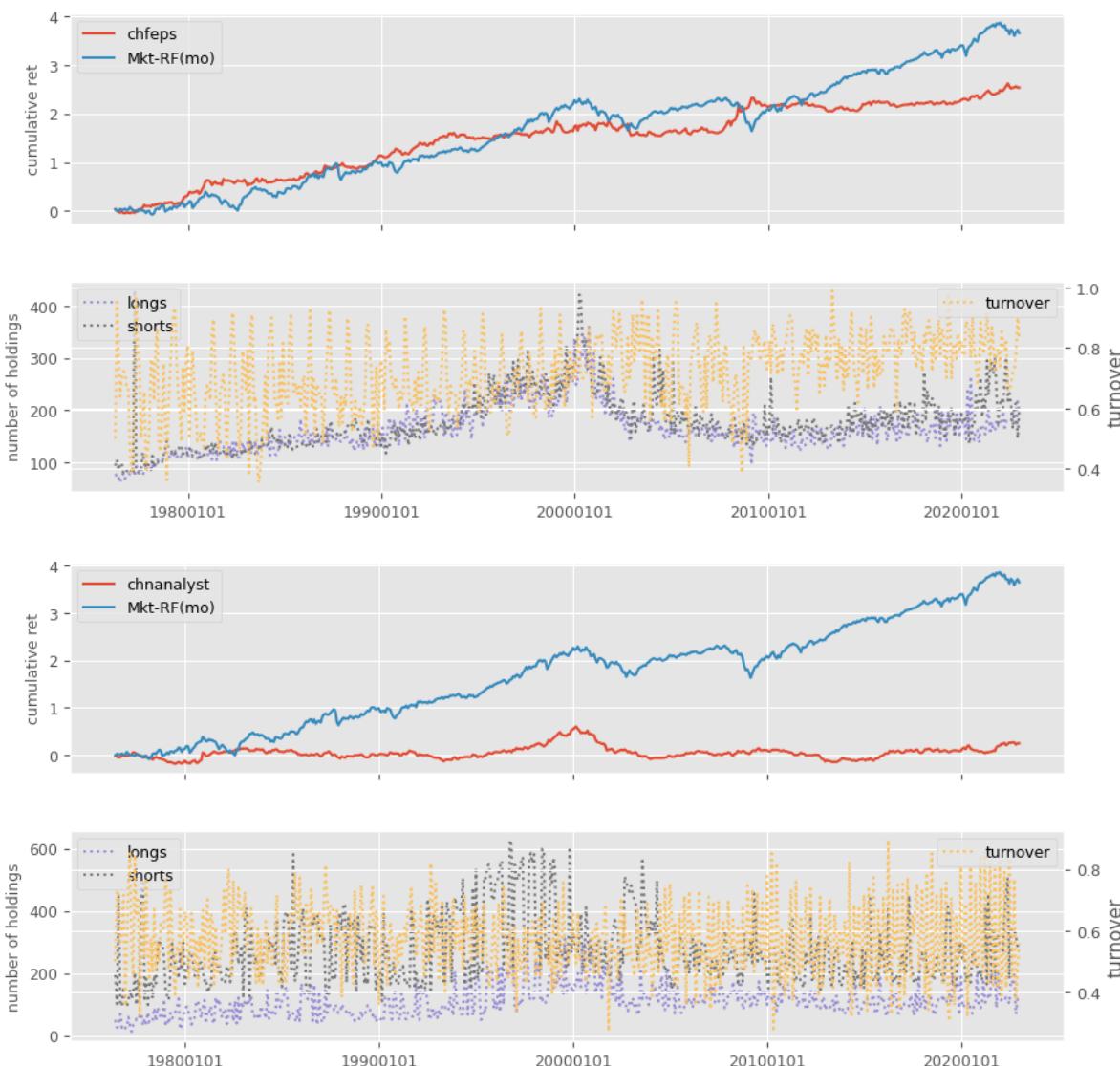
        lag3 = out.shift(3, fill_value=0)
        f3 = (lag3['permno'] == out['permno'])
        out.loc[f3, 'chnanalyst'] = out.loc[f3, 'numest']-lag3.loc[f3, 'numest']

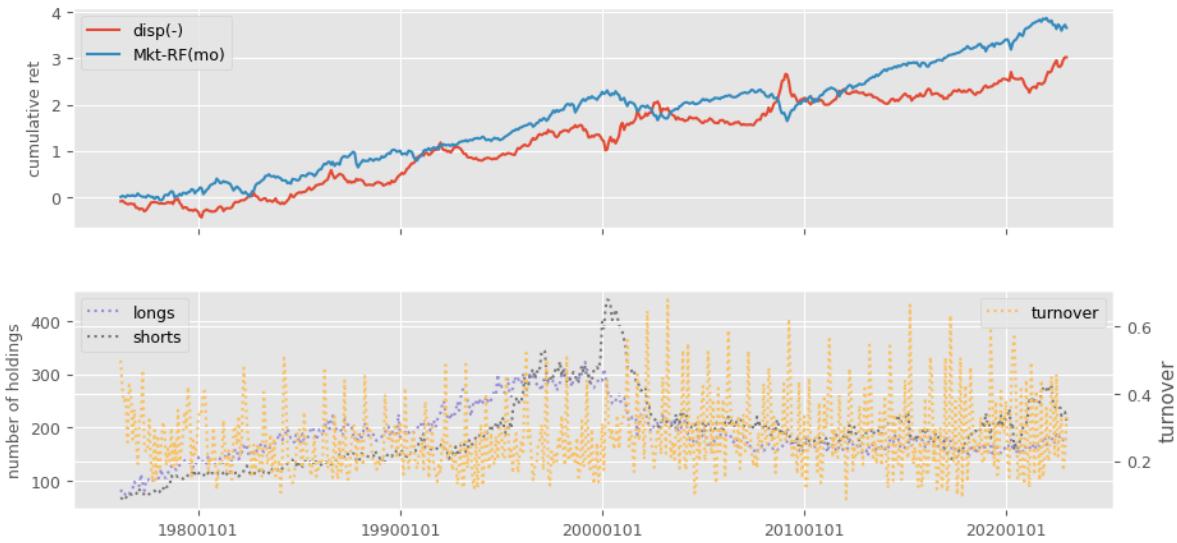
    for label in columns:
        signals.write(out, label, overwrite=True)

    rebalbeg, rebalend = 19760101, LAST_DATE
    benchnames = ['Mkt-RF(mo)']
    for num, label in enumerate(columns):
        holdings = univariate_sorts(crsp,
                                    label,
                                    SignalsFrame(signals.read(label)),
                                    rebalbeg,
                                    rebalend,
                                    window=3,
                                    months=[],
                                    maxdecile=8,
                                    pct=(10., 90.),
                                    leverage=leverage.get(label, 1))
        excess = backtest_pipeline(backtest,
                                    crsp,
                                    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							
chfeps	0.054	0.446	0.062		0.516	-0.72	0.472	8.658	164.546
chfeps	0.053	8.643	8.673						
chnanalyst	0.005	0.062	-0.001		-0.008	0.019	0.985	6.86	112.005
chnanalyst	273.968	6.858	6.862						
disp(-)	0.064	0.366	0.106		0.683	-0.084	0.933	3.226	188.549
disp(-)	188.302	3.211	3.241						





```

# IBES Long-term Growth signals: fgr5yr
if 'ibesltg' in testable:
    columns = ['fgr5yr']

    if regenerate:
        df = ibes.get_linked(dataset='summary',
                              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)

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

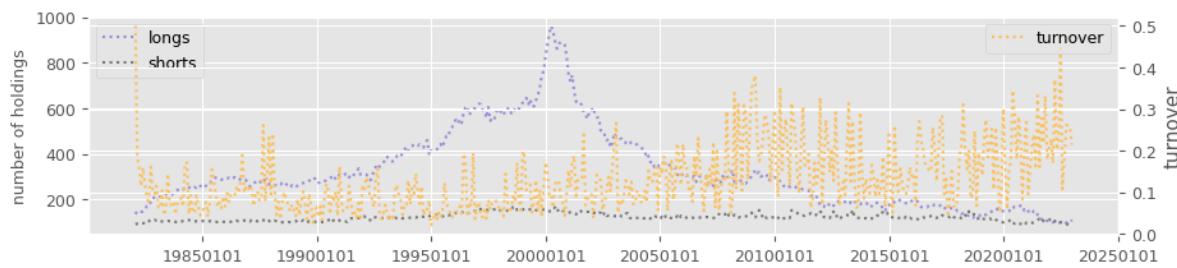
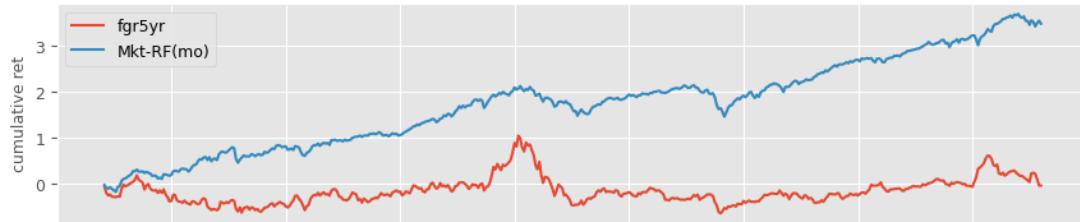
```

(continues on next page)

(continued from previous page)

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

	excess	sharpe	alpha	appraisal	welch-t	welch-p	turnover	longs	shorts	buys	sells
fgr5yr	-0.001	-0.005	-0.058	-0.302	0.054	0.957	1.563	323.36	121.	951	1.552 1.574



```
# Announcement date (rdq) in Quarterly, linked to CRSP daily: ear, aeavol
if 'rdq_daily' in testable:
    columns = ['ear', 'aeavol']

    if regenerate:
        # 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'))
        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 to 20-day average ten-days prior
        actual = crsp.get_window(dataset='daily',
```

(continues on next page)

(continued from previous page)

```

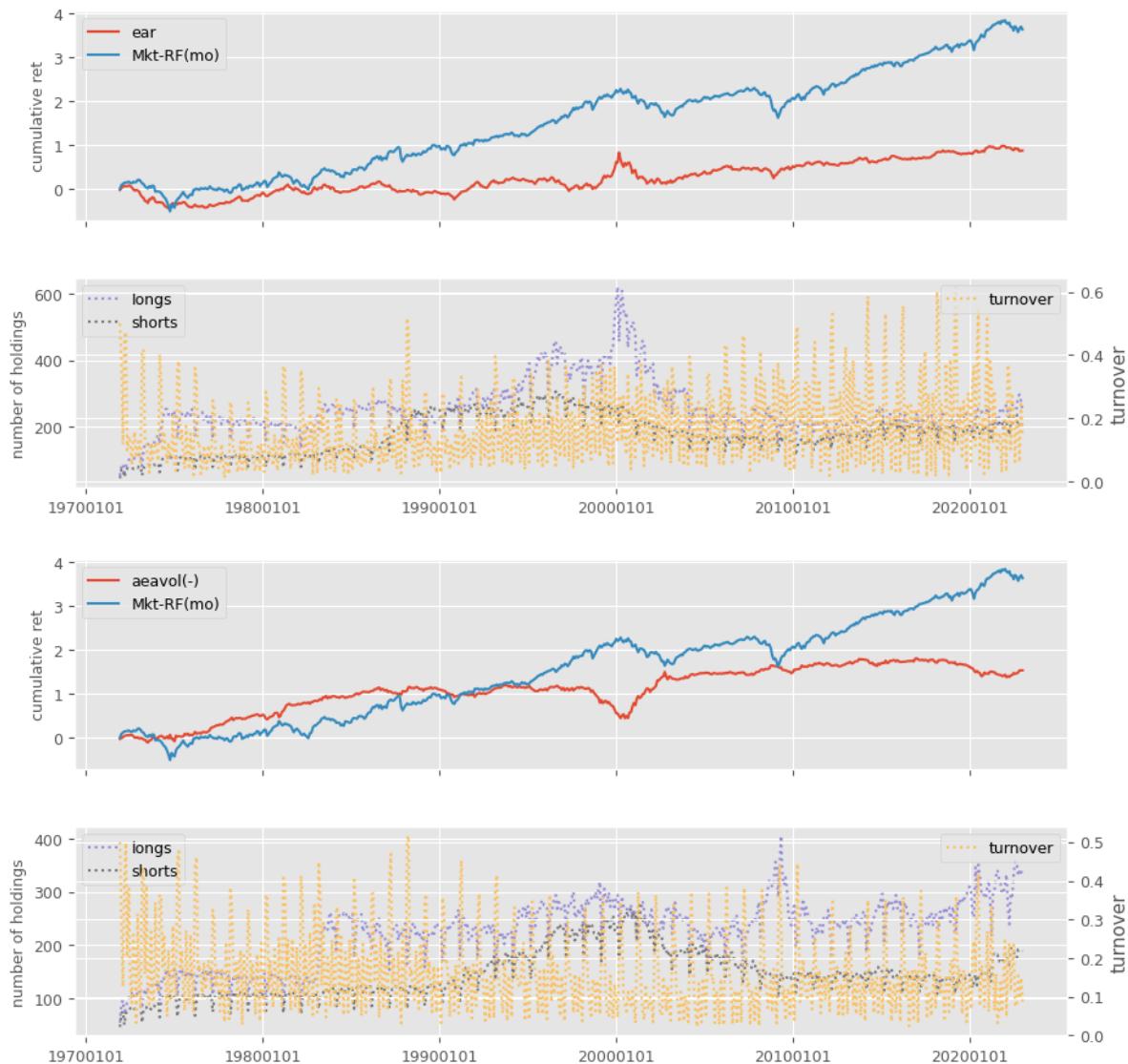
        field='vol',
        date_field='date',
        permnos=fund['permno'],
        dates=fund['rdq'],
        left=-1,
        right=1)
normal = crsp.get_window(dataset='daily',
                         field='vol',
                         date_field='date',
                         permnos=fund['permno'],
                         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)

rebalbeg, rebalend = 19700101, LAST_DATE
benchnames = ['Mkt-RF(mo)']
for num, label in enumerate(columns):
    holdings = univariate_sorts(crsp,
                                 label,
                                 SignalsFrame(signals.read(label)),
                                 rebalbeg,
                                 rebalend,
                                 window=3,
                                 months=[],
                                 maxdecile=8,
                                 pct=(10., 90.),
                                 leverage=leverage.get(label, 1))
    excess = backtest_pipeline(backtest,
                               crsp,
                               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	
ear	0.017	0.148	-0.005	-0.051	0.805	0.421	2.101	252.951	177.	967	2.093	2.11
aeavol(-)	0.03	0.274	0.044	0.417	-0.518	0.604	1.943	224.389	146.137	1.934	1.953	



```
# IBES Fiscal Year 1 linked to Quarterly PSTAT: sfeq
if 'ibesf1_pstqtr' in testable:
    beg, end = 19760101, LAST_DATE
    monthnum = lambda d: ((d//10000)-1900)*12 + ((d//100)%100) - 1
    if regenerate:
        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='summary',
                               fields=['fpedats', 'meanest'],
                               date_field='statpers',
                               where="fpi='1'")
        out = out.dropna() \
            .sort_values(['permno', 'statpers', 'fpedats']) \
```

(continues on next page)

(continued from previous page)

```

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

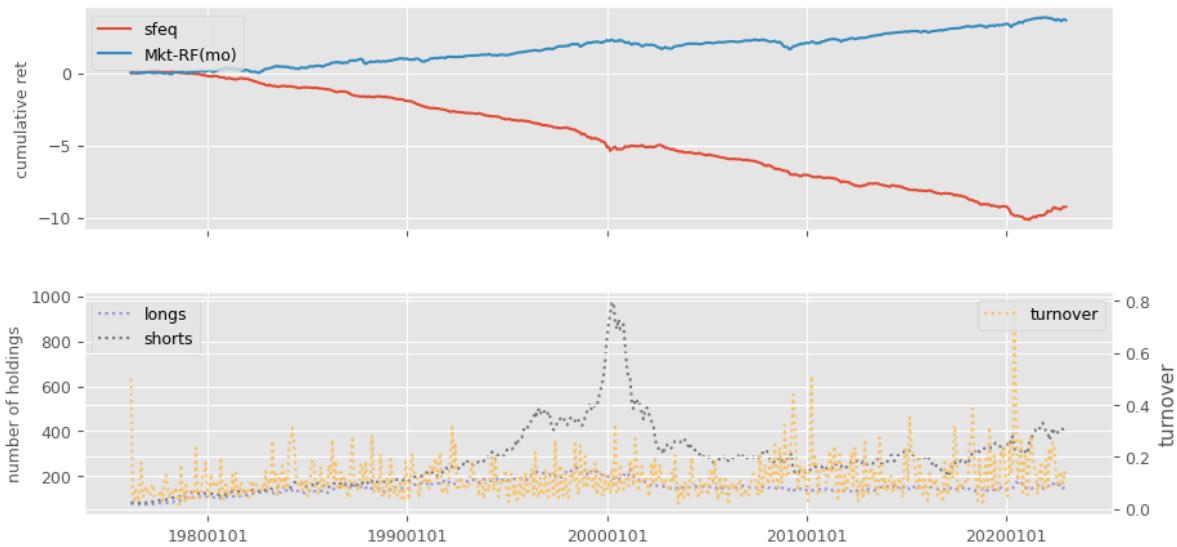
for num in range(4): # match ibes statpers to any data date in last 4 mos
    df['monthnum'] = monthnum(df['data date']) - num
    df = df.set_index(['permno', 'monthnum'], drop=False)
    out = out.join(df[['prccq']], how='left')
    out['sfeq'] = out['sfeq'].where(out['sfeq'].notna(),
            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

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

rebalbeg, rebalend = 19760101, LAST_DATE
benchnames = ['Mkt-RF(mo)']
label = 'sfeq'
holdings = univariate_sorts(crsp,
                             label,
                             SignalsFrame(signals.read(label)),
                             rebalbeg,
                             rebalend,
                             window=3,
                             months=[],
                             maxdecile=8,
                             pct=(10., 90.),
                             leverage=leverage.get(label, 1))
excess = backtest_pipeline(backtest,
                           crsp,
                           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	
sfeq	-0.197	-1.229	-0.188	-1.18	0.008	0.994	1.41	150.655	278.	453	1.466	1.355



```

# IBES Fiscal Year 1 linked to IBES price history: sfe
if 'ibesf1_hist' in testable:
    beg, end = 19760101, LAST_DATE
    if regenerate:
        # retrieve monthly price history
        df = ibes.get_linked(dataset='history',
                              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='summary',
                              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(crsp,
                                 label,
                                 SignalsFrame(signals.read(label)),
                                 rebalbeg,
    
```

(continues on next page)

(continued from previous page)

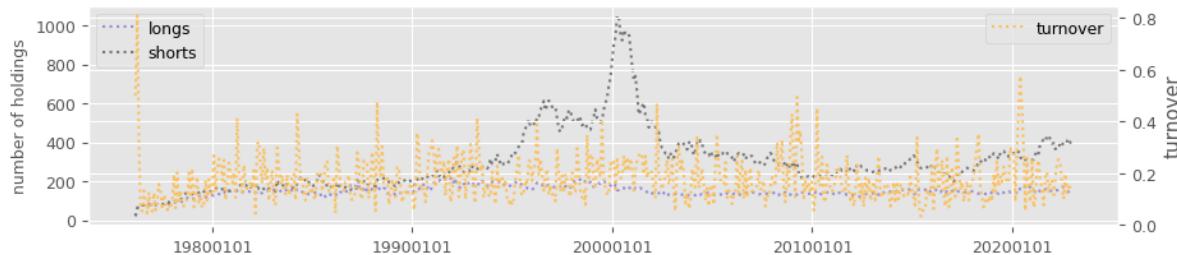
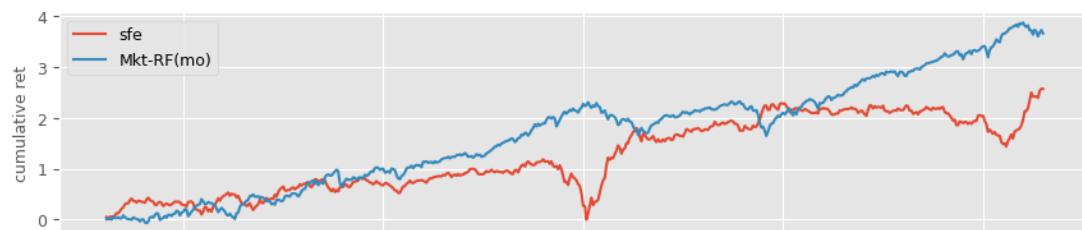
```

        rebalend,
        window=3,
        months=[],
        maxdecile=8,
        pct=(10., 90.),
        leverage=leverage.get(label, 1))

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

```

	excess	sharpe	alpha	appraisal	welch-t	welch-p	turnover	longs	shorts
sfe	0.055	0.301	0.081	0.468	0.067	0.947	2.117	149.174	300.
↳ 453	2.111	2.124							



```

# IBES Fiscal Quarter 1, linked to Quarterly: sue
if 'ibesq1_pstqtr' in testable:
    columns = ['sue']
    numlag = 4
    end = LAST_DATE

    if regenerate:
        # 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'])

```

(continues on next page)

(continued from previous page)

```

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='summary',
                      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='history',
                      fields=['price'],
                      date_field='statpers')
hist = df.dropna() \
    .sort_values(['permno', 'statpers']) \
    .drop_duplicates(['permno', 'statpers'], keep='last')
hist = hist.set_index(['permno', 'statpers'])
summ = summ.join(hist[['price']], how='left')
summ = summ.reset_index() \
    .set_index(['permno', 'rebalddate']) \
    .reindex(fund.index)

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

# sue with ibes surprise 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)

rebalbeg, rebalend = 19760101, LAST_DATE
benchnames = ['Mkt-RF(mo)']
for num, label in enumerate(columns):
    holdings = univariate_sorts(crsp,
                                 label,
                                 SignalsFrame(signals.read(label)),
                                 rebalbeg,
                                 rebalend,
                                 window=3,
                                 months=[],
                                 maxdecile=8,
                                 pct=(10., 90.),

```

(continues on next page)

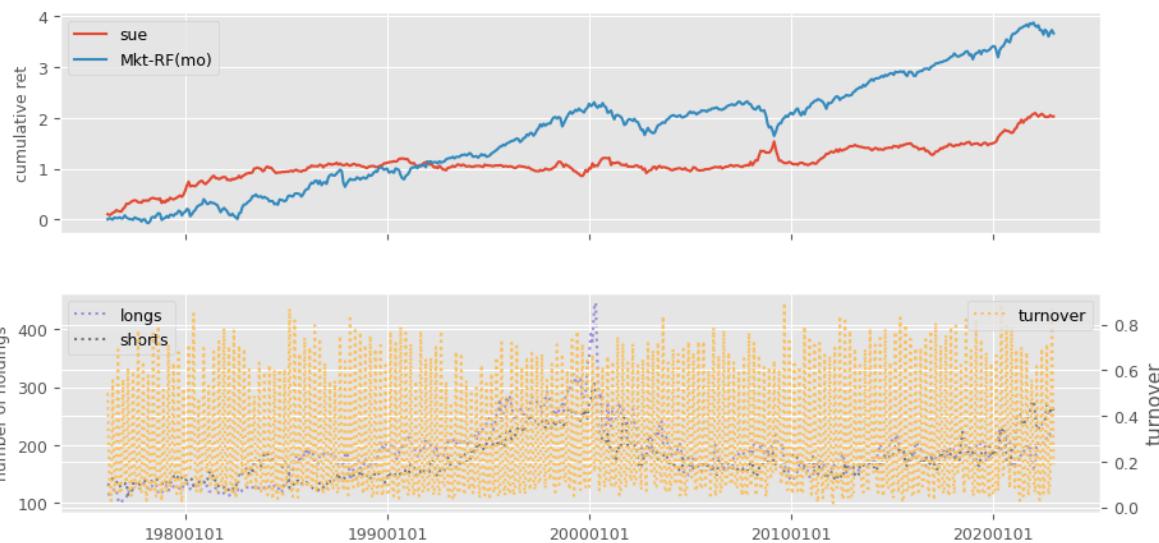
(continued from previous page)

```

        leverage=leverage.get(label, 1))
excess = backtest_pipeline(backtest,
                           crsp,
                           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
sue	0.043	0.407	0.044		0.413	0.209	0.834	3.497	187.091	174.	
808		3.486	3.509								



## 6.5 Performance evaluation

```

# Summarize all results
# sorted by Welch's t-value testing for difference in pre- and post-2002 mean
if 'summarize' in testable:
    zoo = backtest.read().sort_values(['begret', 'permno'])
    r = []
    for label in zoo.index:
        perf = backtest.read(label)
        excess = {'ret': backtest.fit(['Mkt-RF(mo)'])}
        excess['annualized'] = backtest.annualized
        excess['dd'] = maximum_drawdown(backtest.perf['excess'])
        post = {'ret': backtest.fit(['Mkt-RF(mo)'],
                                    beg=20020101).copy()}
        post['annualized'] = backtest.annualized.copy()
        s = label + ('(-)' if leverage.get(label, 1) < 0 else '')
        r.append(DataFrame({
            'Start': excess['ret'].index[0],

```

(continues on next page)

(continued from previous page)

```

'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(),
'Vol2002': post['ret']['excess'].std(ddof=0),
'Best': excess['ret']['excess'].idxmax(),
'BestRet': excess['ret']['excess'].max(),
'Worst': excess['ret']['excess'].idxmin(),
'WorstRet': excess['ret']['excess'].min(),
'Drawdown': (excess['dd'].iloc[1]/excess['dd'].iloc[0]) - 1,
'Beg': excess['dd'].index[0],
'End': excess['dd'].index[1],
'Turn2002+': post['annualized']['sells']/12,
'Long2002+': int(post['annualized']['longs']),
'Short2002+': int(post['annualized']['shorts'])), index=[s]))
df = pd.concat(r, axis=0).round(4).sort_values('Welch-t')
# with open(outdir / 'summary.tex', 'wt') as f:
#     _show = Show(ndigits=4, latex=True)
#     f.write(_show(df.loc[:, 'Vol2002'],
#                   caption="Factor Performance Summary Part I"))
#     f.write(_show(df.loc[:, 'Best'],
#                   caption="Factor Performance Summary Part II"))
show(df)

```

	Start	Sharpe Ratio	Alpha	Appraisal Ratio	Avg Ret	
pchsale_pchrect	19700227	-0.0478	-0.0060	-0.0721	-0.0003	\
mom1m(-)	19260331	0.4082	0.0615	0.3426	0.0065	
pchsaleinv	19700227	0.2231	0.0182	0.2091	0.0016	
bm	19700227	0.2918	0.0440	0.2825	0.0038	
chmom	19270228	0.4147	0.0532	0.3263	0.0060	
pchsale_pchinvt	19700227	0.3106	0.0277	0.3068	0.0023	
pchdepr	19700227	0.0380	0.0036	0.0394	0.0003	
ep	19700227	0.2762	0.0612	0.4006	0.0036	
lev	19700227	0.0613	0.0059	0.0362	0.0008	
realestate	19850430	0.0638	0.0424	0.2881	0.0008	
bm_ia	19700227	0.3064	0.0289	0.2230	0.0034	
mom12m	19270228	0.4693	0.1518	0.6512	0.0101	
ill(-)	19830729	0.2942	0.0429	0.3706	0.0029	
acc(-)	19700227	0.3428	0.0380	0.3524	0.0031	
agr(-)	19700227	0.3742	0.0596	0.5516	0.0036	
absacc(-)	19700227	0.0998	0.0291	0.2092	0.0012	
indmom	19270228	0.3536	0.0650	0.3735	0.0054	
chatoia	19700227	0.2343	0.0243	0.2567	0.0019	
invest(-)	19700227	0.3310	0.0506	0.4491	0.0032	
mom36m(-)	19290228	0.1850	0.0227	0.1140	0.0033	
chcsho(-)	19700227	0.6254	0.0794	0.8202	0.0054	
cashpr(-)	19700227	0.3384	0.0509	0.4208	0.0035	
cinvest(-)	19730228	0.1094	0.0115	0.1316	0.0008	
sgrvol	19730531	0.2384	0.0161	0.1123	0.0030	
divyld	19270228	-0.0354	0.0390	0.2539	-0.0006	
chfeps	19760331	0.4461	0.0620	0.5163	0.0045	
nincr	19700227	0.2799	0.0190	0.2172	0.0021	

(continues on next page)

(continued from previous page)

egr(-)	19700227	0.3760	0.0586	0.5565	0.0035
stdcf(-)	19790531	0.2006	0.0580	0.4709	0.0023
pchsale_pchxsga	19700227	-0.1025	-0.0128	-0.1157	-0.0010
dolvol	19830729	0.0426	-0.0063	-0.0605	0.0004
beta	19290731	0.0616	-0.0755	-0.3931	0.0016
sp	19700227	0.3225	0.0454	0.3112	0.0039
aeavol(-)	19711130	0.2742	0.0436	0.4166	0.0025
dy	19700227	-0.0206	0.0407	0.2419	-0.0003
mom6m	19260831	0.3192	0.1079	0.4992	0.0064
pricedelay(-)	19290731	0.1068	0.0053	0.0545	0.0009
cash	19711231	0.2242	0.0085	0.0611	0.0028
chinv(-)	19700227	0.4011	0.0536	0.5106	0.0036
idiovol(-)	19290731	0.0033	0.0527	0.2710	0.0001
chtx	19721229	0.1800	0.0163	0.1361	0.0018
maxret(-)	19830729	0.2514	0.1228	0.6978	0.0046
stdacc(-)	19790531	0.1383	0.0413	0.3591	0.0014
retvol(-)	19830729	0.2129	0.1410	0.6831	0.0047
pchgm_pchsale	19700227	0.0431	-0.0001	-0.0014	0.0004
disp(-)	19760227	0.3663	0.1059	0.6830	0.0054
cfp	19700227	0.2679	0.0529	0.3826	0.0032
mve_ia(-)	19700227	0.1773	0.0081	0.0799	0.0015
grltnoa	19700227	0.1722	0.0217	0.1875	0.0017
sfeq	19760227	-1.2295	-0.1881	-1.1802	-0.0165
chnanalyst	19760528	0.0624	-0.0007	-0.0084	0.0004
chpmia	19700227	0.1666	0.0255	0.1956	0.0018
fgr5yr	19820129	-0.0048	-0.0583	-0.3025	-0.0001
sfe	19760227	0.3010	0.0813	0.4680	0.0046
cfp_ia	19700227	-0.1851	-0.0354	-0.2580	-0.0021
sgr	19700227	-0.0918	-0.0261	-0.2139	-0.0010
baspread	19830729	-0.1276	-0.1259	-0.6013	-0.0029
roavol	19760831	0.0402	-0.0109	-0.0740	0.0005
sue	19760227	0.4070	0.0438	0.4130	0.0036
pchcapx_ia	19700227	-0.2599	-0.0317	-0.2624	-0.0026
rd_sale	19700227	-0.0898	-0.0196	-0.1272	-0.0012
pchquick	19700227	-0.1283	-0.0117	-0.1392	-0.0009
salecash	19700227	0.0196	0.0189	0.1704	0.0002
rd_mve	19700227	0.1543	0.0144	0.0903	0.0021
std_turn	19830729	-0.0112	-0.0569	-0.3379	-0.0002
turn	19830729	0.0520	-0.0561	-0.3123	0.0010
zerotrade(-)	19830729	0.1124	-0.0408	-0.2342	0.0020
quick	19700227	-0.0195	-0.0339	-0.2400	-0.0003
depr	19700227	0.0332	-0.0242	-0.1842	0.0004
secured(-)	19820331	0.0308	0.0312	0.2598	0.0003
pctacc(-)	19700227	0.1210	0.0246	0.2338	0.0011
rsup	19700227	0.1396	0.0207	0.1605	0.0015
gma	19700227	0.0523	0.0114	0.0801	0.0006
tang	19700227	0.1640	0.0080	0.0686	0.0016
ear	19711130	0.1480	-0.0053	-0.0509	0.0014
salerec	19700227	0.1946	0.0388	0.3131	0.0021
roaq	19710831	0.2874	0.0596	0.4304	0.0035
std_dolvol	19830729	-0.0039	0.0024	0.0217	-0.0000
lgr	19700227	-0.1926	-0.0268	-0.3040	-0.0015
hire	19700227	-0.1815	-0.0365	-0.3329	-0.0017
grcapx	19700227	-0.3130	-0.0420	-0.4262	-0.0026
saleinv	19700227	0.0878	0.0293	0.3101	0.0008
chempia	19700227	0.1113	0.0003	0.0026	0.0010

(continues on next page)

(continued from previous page)

	Vol	Welch-t	Appraisal2002	Ret2002	Vol2002	Best	
pchsale_pchrect	0.0241	-2.8638	-0.5704	-0.0037	0.0243	19991231	\
mom1m(-)	0.0537	-2.3080	-0.2204	-0.0003	0.0520	19320130	
pchsaleinv	0.0251	-1.9123	-0.1762	-0.0008	0.0274	20211029	
bm	0.0450	-1.8675	-0.1753	-0.0003	0.0451	19750131	
chmom	0.0494	-1.8010	-0.0494	0.0013	0.0468	19330429	
pchsale_pchinvt	0.0261	-1.7845	-0.0464	-0.0000	0.0287	20000929	
pchdepr	0.0263	-1.6582	-0.2177	-0.0018	0.0260	19991231	
ep	0.0456	-1.5214	0.1572	0.0003	0.0414	20001130	
lev	0.0473	-1.5075	-0.2879	-0.0027	0.0507	20161130	
realestate	0.0458	-1.4225	0.0927	-0.0020	0.0372	20001130	
bm_ia	0.0382	-1.4202	-0.0058	0.0009	0.0318	20000229	
mom12m	0.0731	-1.3785	0.4569	0.0047	0.0703	20000229	
ill(-)	0.0338	-1.2447	0.2806	0.0011	0.0346	20200331	
acc(-)	0.0311	-1.2044	0.0908	0.0012	0.0333	20090130	
agr(-)	0.0331	-1.1891	0.2565	0.0017	0.0321	20220131	
absacc(-)	0.0413	-1.1717	-0.0572	-0.0011	0.0396	20080930	
indmom	0.0514	-1.1158	0.2298	0.0024	0.0450	20001229	
chatoia	0.0273	-1.0758	0.0443	0.0004	0.0274	20001130	
invest(-)	0.0335	-0.9373	0.2292	0.0016	0.0398	20221031	
mom36m(-)	0.0597	-0.9363	-0.0356	0.0004	0.0541	19320831	
chcsho(-)	0.0297	-0.8640	0.7284	0.0042	0.0259	20010228	
cashpr(-)	0.0355	-0.7869	0.1730	0.0021	0.0328	20220131	
cinvest(-)	0.0252	-0.7688	-0.1171	-0.0001	0.0250	20000929	
sgrvol	0.0433	-0.7680	-0.1551	0.0014	0.0454	20201130	
divyld	0.0554	-0.7647	-0.0748	-0.0025	0.0415	20010228	
chfeps	0.0350	-0.7199	0.4994	0.0033	0.0357	20080630	
nincr	0.0256	-0.6920	0.2133	0.0013	0.0209	19811030	
egr(-)	0.0323	-0.6873	0.4498	0.0025	0.0256	20001229	
stdcf(-)	0.0394	-0.6780	0.3555	0.0011	0.0310	20001130	
pchsale_pchxsga	0.0321	-0.6691	-0.2660	-0.0020	0.0335	20221031	
dolvol	0.0306	-0.5688	-0.0215	-0.0004	0.0286	20200331	
beta	0.0904	-0.5376	-0.5535	-0.0008	0.0772	19320730	
sp	0.0421	-0.5213	0.1738	0.0029	0.0402	19740131	
aeavol(-)	0.0315	-0.5183	0.3576	0.0017	0.0272	20010228	
dy	0.0569	-0.4701	0.0117	-0.0016	0.0437	20010228	
mom6m	0.0681	-0.4609	0.4435	0.0047	0.0651	20010228	
pricedelay(-)	0.0288	-0.4371	-0.0903	0.0002	0.0290	19321130	
cash	0.0433	-0.4080	-0.0251	0.0020	0.0397	20000229	
chinvt(-)	0.0311	-0.3675	0.3601	0.0030	0.0314	20080930	
idiovol(-)	0.0691	-0.3319	0.3329	-0.0012	0.0673	19321130	
chtx	0.0347	-0.3094	0.2196	0.0013	0.0293	19801128	
maxret(-)	0.0634	-0.2790	0.6880	0.0038	0.0593	20010228	
stdacc(-)	0.0358	-0.2079	0.3131	0.0011	0.0266	20000428	
retvol(-)	0.0762	-0.1908	0.6464	0.0041	0.0739	20001130	
pchgm_pchsale	0.0291	-0.1069	0.0519	0.0002	0.0334	20211029	
disp(-)	0.0507	-0.0841	0.8225	0.0052	0.0542	20001130	
cfp	0.0411	-0.0754	0.2649	0.0030	0.0393	20010228	
mve_ia(-)	0.0301	-0.0284	0.0974	0.0015	0.0263	20000229	
grltnoa	0.0334	-0.0233	0.2002	0.0016	0.0271	19990331	
sfeq	0.0463	0.0080	-1.1494	-0.0165	0.0512	20220131	
chnanalyst	0.0247	0.0191	0.0646	0.0005	0.0221	19801128	
chpmia	0.0377	0.0234	0.1888	0.0019	0.0394	19980831	
fgr5yr	0.0635	0.0545	-0.1150	0.0001	0.0493	20000229	
sfe	0.0525	0.0668	0.4127	0.0047	0.0508	20010228	

(continues on next page)

(continued from previous page)

cfp_ia	0.0401	0.0943	-0.3298	-0.0020	0.0320	20010131
sgr	0.0365	0.1513	-0.1144	-0.0007	0.0387	20200430
baspread	0.0791	0.1652	-0.5646	-0.0024	0.0812	20090430
roavol	0.0436	0.2071	0.0284	0.0009	0.0406	20090130
sue	0.0306	0.2094	0.5584	0.0039	0.0318	20090227
pchcapx_ia	0.0349	0.2300	-0.1260	-0.0022	0.0399	20080930
rd_sale	0.0447	0.2844	-0.1303	-0.0006	0.0337	20000229
pchquick	0.0243	0.2990	-0.1980	-0.0006	0.0231	20211029
salecash	0.0338	0.3567	0.2448	0.0008	0.0341	20001130
rd_mve	0.0467	0.3671	0.1423	0.0029	0.0378	20000831
std_turn	0.0575	0.4142	-0.3064	0.0009	0.0506	20000229
turn	0.0643	0.4809	-0.2450	0.0023	0.0607	20000229
zerotrade (-)	0.0622	0.4961	-0.1516	0.0034	0.0598	20000229
quick	0.0456	0.5175	-0.1972	0.0008	0.0370	20000229
depr	0.0424	0.5773	-0.0369	0.0015	0.0304	20000229
secured (-)	0.0375	0.6254	0.3753	0.0014	0.0284	20001130
pctacc (-)	0.0313	0.7416	0.2743	0.0022	0.0286	19980930
rsup	0.0372	0.7546	0.4193	0.0029	0.0404	19801128
gma	0.0413	0.7698	0.3281	0.0022	0.0452	20090130
tang	0.0345	0.8049	0.1572	0.0030	0.0311	20000229
ear	0.0331	0.8052	0.1948	0.0026	0.0256	20000229
salerec	0.0370	0.9285	0.5357	0.0039	0.0442	20090130
roaq	0.0418	0.9436	0.7615	0.0054	0.0429	20001130
std_dolvol	0.0315	1.1085	0.1598	0.0015	0.0299	20000229
lgr	0.0262	1.1662	-0.0205	0.0000	0.0263	19730731
hire	0.0333	1.1901	-0.0233	0.0002	0.0340	19811030
grcapx	0.0293	1.4490	-0.1244	-0.0005	0.0324	19970530
saleinv	0.0303	1.5258	0.5280	0.0030	0.0299	19871030
chempia	0.0323	1.6651	0.3580	0.0037	0.0325	20000229

	BestRet	Worst	WorstRet	Drawdown	Beg	End	
pchsale_pchrect	0.1156	20110729	-0.0897	-0.7802	20000229	20221230	\
mom1m (-)	0.3023	19320831	-0.2805	-0.6682	19890929	20090227	
pchsaleinv	0.1069	19741031	-0.0799	-0.4978	20050831	20210226	
bm	0.1952	20200331	-0.2009	-0.7468	20060731	20200930	
chmom	0.3721	20010228	-0.2282	-0.5541	20141031	20220630	
pchsale_pchinvt	0.1144	20011130	-0.0827	-0.4448	20050831	20160129	
pchdepr	0.1332	20000428	-0.1406	-0.6629	19991231	20211130	
ep	0.2241	20000229	-0.1896	-0.6611	20081128	20220930	
lev	0.1688	20090130	-0.2042	-0.8672	20060731	20200930	
realestate	0.3043	20000229	-0.2518	-0.6669	20020930	20220531	
bm_ia	0.1555	20000929	-0.2242	-0.6379	20000630	20090227	
mom12m	0.3018	19320831	-0.7666	-0.9625	19320531	19390930	
ill (-)	0.1206	20090430	-0.1272	-0.6201	20001130	20130930	
acc (-)	0.1286	20001229	-0.1221	-0.4028	20120731	20170630	
agr (-)	0.1758	20010131	-0.1205	-0.4119	20130628	20201030	
absacc (-)	0.1543	20000229	-0.2789	-0.7667	19861031	20200831	
indmom	0.2860	19320831	-0.4372	-0.6897	19320630	19330131	
chatoia	0.1469	20000831	-0.1143	-0.5422	20070629	20190830	
invest (-)	0.2001	20220729	-0.1532	-0.4225	20160229	20200831	
mom36m (-)	0.6077	19380630	-0.3220	-0.7423	20040130	20200930	
chcsho (-)	0.1632	20010131	-0.1257	-0.2681	19971128	19991029	
cashpr (-)	0.1519	20090130	-0.1758	-0.4260	20161230	20200930	
cinvest (-)	0.0899	19741031	-0.1740	-0.4349	19910531	20111031	
sgrvol	0.1580	20200331	-0.1433	-0.6886	19940228	20200930	
divyld	0.2050	19390930	-0.4011	-0.9669	19301231	20201030	

(continues on next page)

(continued from previous page)

chfeps	0.2032	20021031	-0.1595	-0.3106	19990129	20040528
nincr	0.0955	19990226	-0.0830	-0.4057	19811030	19881130
egr(-)	0.1240	19811030	-0.1456	-0.3388	19891229	19921231
stdcf(-)	0.2208	20000229	-0.2281	-0.4853	19970430	20000229
pchsale_pchxsga	0.1663	20200331	-0.1383	-0.8058	19741231	20220729
dolvol	0.0903	20020430	-0.1349	-0.7578	20000331	20161230
beta	0.8033	20001130	-0.3235	-0.9573	19560731	20190930
sp	0.1776	20021031	-0.1441	-0.5689	20150331	20200930
aeavol(-)	0.1471	20021031	-0.1099	-0.5578	19930930	20000630
dy	0.2417	20000229	-0.2530	-0.9167	19770531	20201231
mom6m	0.2335	19320730	-0.6317	-0.9870	19320531	19421031
pricedelay(-)	0.1842	19330429	-0.1501	-0.6177	19801128	19911231
cash	0.2273	20010228	-0.1908	-0.7067	20000229	20080630
chinv(-)	0.1138	20090130	-0.1004	-0.2734	19700630	19730731
idiovol(-)	0.3418	19330531	-0.4104	-0.9856	19301231	20210129
chtx	0.1925	20010228	-0.1398	-0.4446	20000929	20090529
maxret(-)	0.2802	20090430	-0.2737	-0.7180	19980930	20000229
stdacc(-)	0.1408	20000229	-0.2700	-0.5071	19981130	20000229
retvol(-)	0.3343	19991231	-0.3006	-0.7920	19980831	20000229
pchgm_pchsale	0.1294	20200831	-0.1340	-0.6147	20090227	20210129
disp(-)	0.2146	20090430	-0.2294	-0.5227	20090130	20140131
cfp	0.2207	20200331	-0.1837	-0.5675	20161230	20200930
mve_ia(-)	0.1721	20000331	-0.1664	-0.5985	19890531	20001130
grltnoa	0.1649	20000428	-0.1674	-0.4466	19991231	20001229
sfeq	0.1730	20200331	-0.2939	-1.0000	19770531	20210226
chnanalyst	0.1633	20010928	-0.1005	-0.5515	20000229	20130731
chpmia	0.1472	20201130	-0.1624	-0.5640	19940930	20070430
fgr5yr	0.2384	20010228	-0.2885	-0.8674	20000229	20081231
sfe	0.2313	20000229	-0.2623	-0.7366	19971231	20000229
cfp_ia	0.1521	20000929	-0.2560	-0.8842	19730430	20130430
sgr	0.1494	20220131	-0.1825	-0.6907	19730731	20220531
baspread	0.3527	20010228	-0.2888	-0.9398	19830729	20221230
roavol	0.2068	20000331	-0.1726	-0.7823	19801128	20020930
sue	0.1098	20090331	-0.1680	-0.3923	20090227	20101029
pchcapx_ia	0.1655	20081231	-0.1751	-0.8901	19720428	20211130
rd_sale	0.1830	20001130	-0.1842	-0.8712	19730629	19930331
pchquick	0.1153	19991231	-0.1173	-0.7665	19881130	20181130
salecash	0.1409	20000831	-0.1286	-0.7465	19901031	20000229
rd_mve	0.2326	19741031	-0.2032	-0.7844	19780731	19911231
std_turn	0.2880	20001130	-0.2615	-0.8093	20000229	20090227
turn	0.3013	20010228	-0.2655	-0.8252	20000229	20020930
zerotrade(-)	0.2750	20010228	-0.2608	-0.8317	20000229	20020930
quick	0.2292	20001130	-0.2601	-0.7894	20000229	20081128
depr	0.1778	19871030	-0.1921	-0.6552	19720731	19901031
secured(-)	0.1822	20000229	-0.2462	-0.6423	19980831	20000229
pctacc(-)	0.1450	20080930	-0.1493	-0.4085	19741231	19800829
rsup	0.1679	20210226	-0.1740	-0.5445	19801128	20020131
gma	0.1894	20161130	-0.1296	-0.6415	20000229	20061229
tang	0.1928	19730928	-0.1236	-0.7364	19730629	19901031
ear	0.2294	20000331	-0.1554	-0.5447	20000229	20021031
salerec	0.1576	20000831	-0.1527	-0.6778	19901031	20000831
roaq	0.1516	20000229	-0.1877	-0.4552	19980930	20000229
std_dolvol	0.1681	20000331	-0.1670	-0.5405	19850430	20000731
lgr	0.0657	20001229	-0.1356	-0.8208	19731231	20130628
hire	0.1186	20221031	-0.1552	-0.8297	19730731	20130628
grcapx	0.0830	20221031	-0.1194	-0.8635	19730928	20031031

(continues on next page)

(continued from previous page)

saleinv	0.1235	19800731	-0.1030	-0.4905	19780228	20000831
chempia	0.1590	19731130	-0.1306	-0.6364	19730928	19990331
			Turn2002+	Long2002+	Short2002+	
pchsale_pchrect	0.0937		203		201	
mom1m(-)	0.9185		218		229	
pchsaleinv	0.0872		176		173	
bm	0.0502		153		219	
chmom	0.5523		220		222	
pchsale_pchinvt	0.0876		182		174	
pchdepr	0.0901		225		217	
ep	0.0764		144		254	
lev	0.0342		197		393	
realestate	0.0380		103		130	
bm_ia	0.0833		160		181	
mom12m	0.3630		235		201	
ill(-)	0.1673		148		292	
acc(-)	0.0744		226		184	
agr(-)	0.0859		178		257	
absacc(-)	0.0706		197		226	
indmom	0.4331		140		219	
chatoia	0.0891		180		179	
invest(-)	0.0795		141		182	
mom36m(-)	0.2603		184		226	
chcsho(-)	0.0839		158		235	
cashpr(-)	0.0512		168		167	
cinvest(-)	0.2612		176		171	
sgrvol	0.0564		121		215	
divyld	0.0873		139		835	
chfeps	0.7760		163		183	
nincr	0.2628		124		862	
egr(-)	0.0789		181		232	
stdcf(-)	0.0583		114		179	
pchsale_pchxsga	0.0887		172		156	
dolvol	0.1546		157		279	
beta	0.0855		221		143	
sp	0.0452		136		374	
aeavol(-)	0.1458		258		154	
dy	0.0403		139		820	
mom6m	0.5046		235		211	
pricedelay(-)	0.3831		217		187	
cash	0.1041		394		140	
chinv(-)	0.0852		153		185	
idiovol(-)	0.0790		131		310	
chtx	0.2906		190		176	
maxret(-)	0.7590		154		256	
stdacc(-)	0.0556		112		169	
retvol(-)	0.6245		151		269	
pchgm_pchsale	0.0882		183		177	
disp(-)	0.2906		171		199	
cfp	0.0695		140		311	
mve_ia(-)	0.0337		202		133	
grltnoa	0.0844		176		175	
sfeq	0.1281		147		295	
chnanalyst	0.5630		119		259	
chpmia	0.0988		64		65	

(continues on next page)

(continued from previous page)

fgr5yr	0.1691	241	121
sfe	0.1739	141	306
cfp_ia	0.0832	153	184
sgr	0.0800	256	166
baspread	0.4015	306	137
roavol	0.0530	241	188
sue	0.3049	183	179
pchcapx_ia	0.0995	73	65
rd_sale	0.0255	272	167
pchquick	0.0955	177	183
salecash	0.0471	143	383
rd_mve	0.0398	175	168
std_turn	0.5773	252	187
turn	0.1686	248	221
zerotrade (-)	0.3687	245	222
quick	0.0439	350	119
depr	0.0367	270	136
secured (-)	0.0489	409	112
pctacc (-)	0.0790	174	187
rsup	0.2047	151	144
gma	0.0402	219	241
tang	0.0406	352	163
ear	0.2155	228	178
salerec	0.0301	181	226
roaq	0.1891	219	263
std_dolvol	0.7473	252	151
lgr	0.0895	246	187
hire	0.0811	274	162
grcapx	0.0805	233	187
saleinv	0.0404	162	141
chempia	0.0914	193	136

## EVENT STUDY

### UNDER CONSTRUCTION

- event study: CAR, BHAR, post-announcement drift (Kolari 2010 and others)
- cross-sectional correlation: cross-correlation, convolution, FFT
- multiple testing: Holm FWER, Benjamin-Hochberg FDR, Bonferroni p-values
- S&P Key Developments

```
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
from finds.database import SQL
from finds.busday import BusDay
from finds.structured import Benchmarks, Stocks, CRSP, PSTAT
from finds.backtesting import EventStudy
from finds.misc import Show
from secret import credentials, paths, CRSP_DATE
show = Show(ndigits=4, latex=None)
VERBOSE = 0
#
#%matplotlib qt
```

```
# open connections
sql = SQL(**credentials['sql'], verbose=VERBOSE)
user = SQL(**credentials['user'], verbose=VERBOSE)
bd = BusDay(sql)
keydev = PSTAT(sql, bd, verbose=VERBOSE)
crsp = CRSP(sql, bd, rdb=None, verbose=VERBOSE)
bench = Benchmarks(sql, bd, verbose=VERBOSE)
eventstudy = EventStudy(user, bench=bench, stocks=crsp, max_date=CRSP_DATE)
imgdir = paths['images'] / 'events'
```

Last FamaFrench Date 2023-04-28 00:00:00

## 7.1 Key developments

```
# sorted list of all eventids and roleids, provided in keydev class
events = sorted(keydev._event.keys())
roles = sorted(keydev._role.keys())
```

```
# 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)
```

## 7.2 Event study

```
# event window parameters
left, right, post = -1, 1, 21
end = bd.offset(CRSP_DATE, post - left)
beg = 19890101 # 20020101
minobs = 250
```

```
# to lookup prevailing exchange and share codes by permno and date
shrcd = crsp.build_lookup('permno', 'shrcd')
exchcd = crsp.build_lookup('permno', 'exchcd')

# run event study after screening stock universe
def event_pipeline(eventstudy: EventStudy,
                   stocks: Stocks,
                   beg: int,
                   end: int,
                   eventid: int,
                   roleid: int,
                   left: int,
                   right: int,
                   post: int,
                   mincap: float = 300000.) -> DataFrame:
    """helper to merge keydev events and crsp, and screen stock universe"""

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

(continues on next page)

(continued from previous page)

```

# Require in valid screen: 'cap', 'exchcd', 'shrcd'
stk = stocks.get_many(dataset='daily',
                      permnos=df['permno'],
                      date_field='date',
                      dates=stocks.bd.offset(df['announcedate'], left=1),
                      fields=['prc', 'shroud']).fillna(0)
df['cap'] = (stk['prc'].abs() * stk['shroud']).values
df['exchcd'] = [exchcd(row.permno, row.date) for row in stk.itertuples()]
df['shrcd'] = [shrcd(row.permno, row.date) for row in stk.itertuples()]
select = (df['cap'].gt(mincap) # require cap > $300M
          & df['exchcd'].isin([1, 2, 3]) # primary exchange
          & df['shrcd'].isin([10, 11])).values # domestic common stocks

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

```

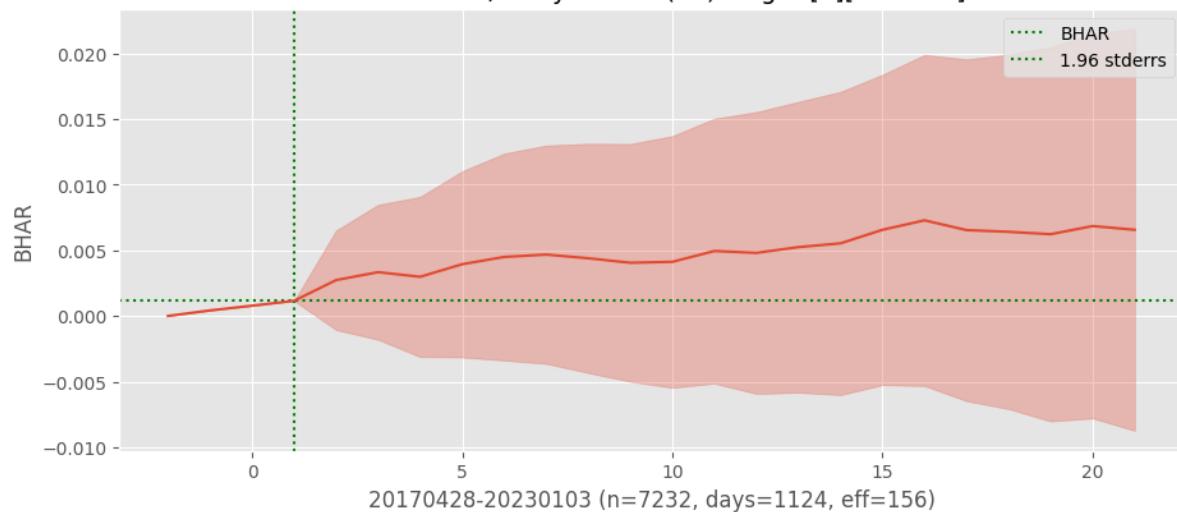
### Show subsample plots for selected events

```

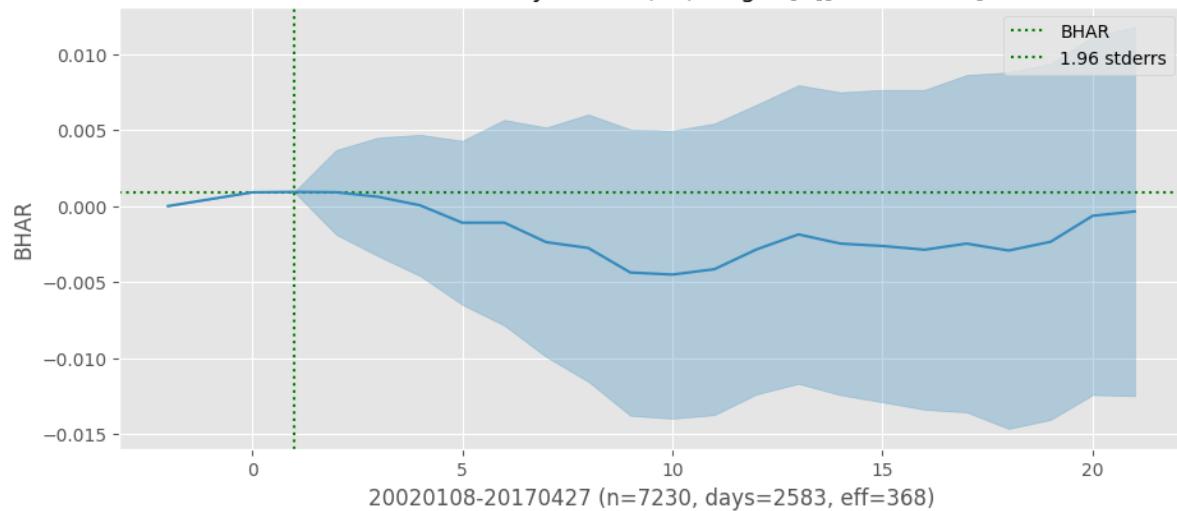
### Show by market cap and half-period
events_list = [[50, 1], [150, 1]] # top drift
midcap = 20000000 # arbitrary mid market cap breakpoint
for i, (eventid, roleid) in enumerate(events_list):
    #eventid, roleid = 50, 1
    #eventid, roleid = 83, 1
    df = event_pipeline(eventstudy,
                        stocks=crsp,
                        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['cap'].ge(midcap).values,
              'Small': df['cap'].lt(midcap).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}]",
                        drift=True,
                        ax=ax,
                        c=f"C{i*5+ifig}")
        plt.savefig(imgdir / (label + f"{eventid}_{roleid}.jpg"))

```

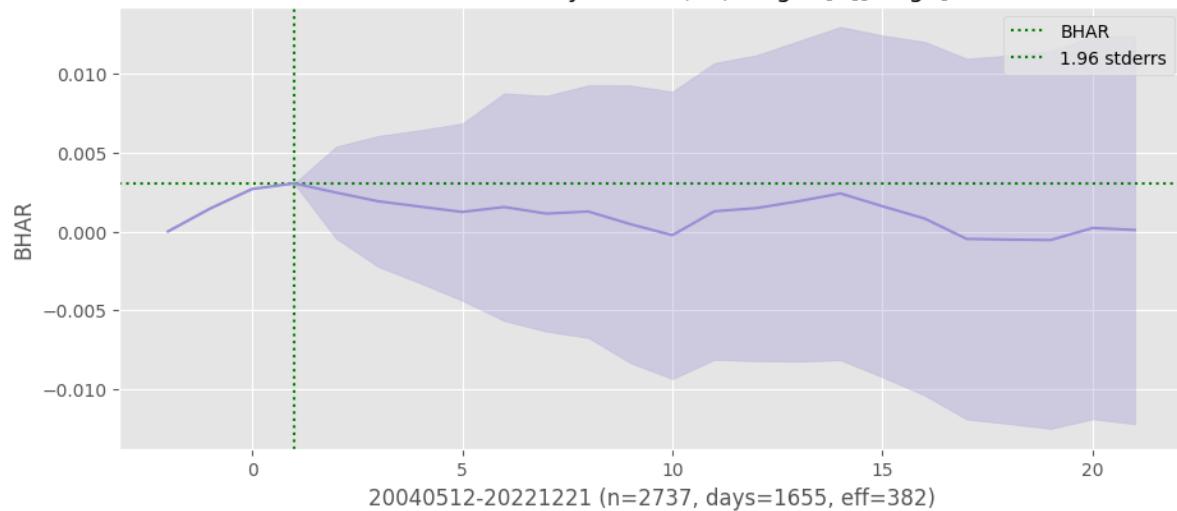
Shareholder/Analyst Calls (50) Target [1][FirstHalf]



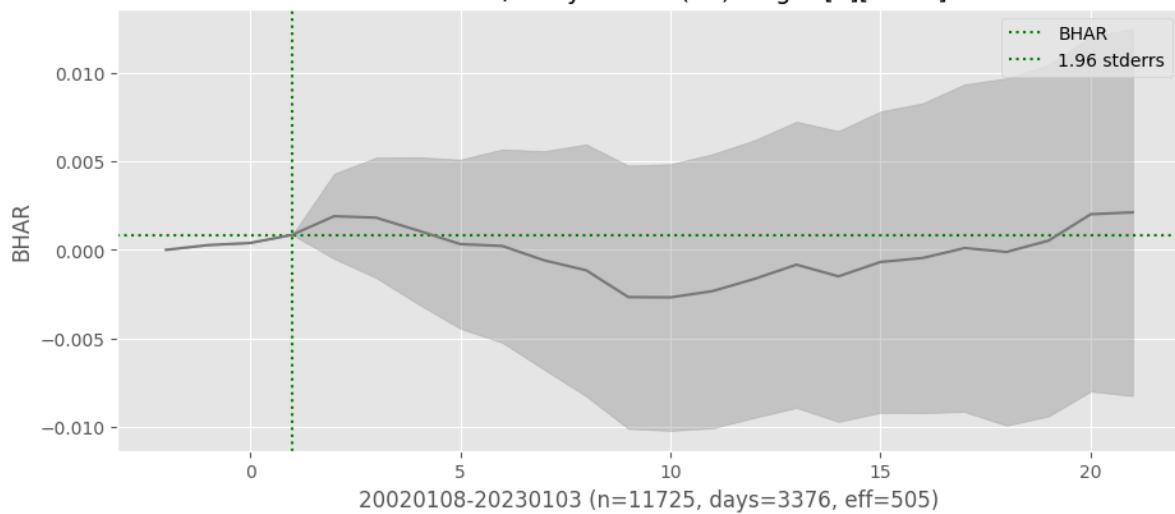
Shareholder/Analyst Calls (50) Target [1][SecondHalf]



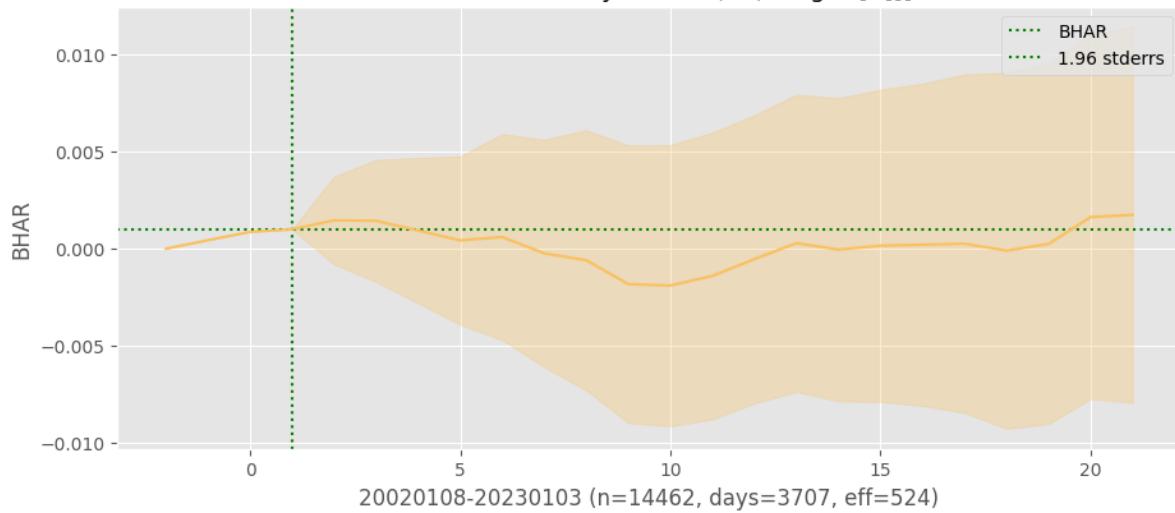
Shareholder/Analyst Calls (50) Target [1][Large]



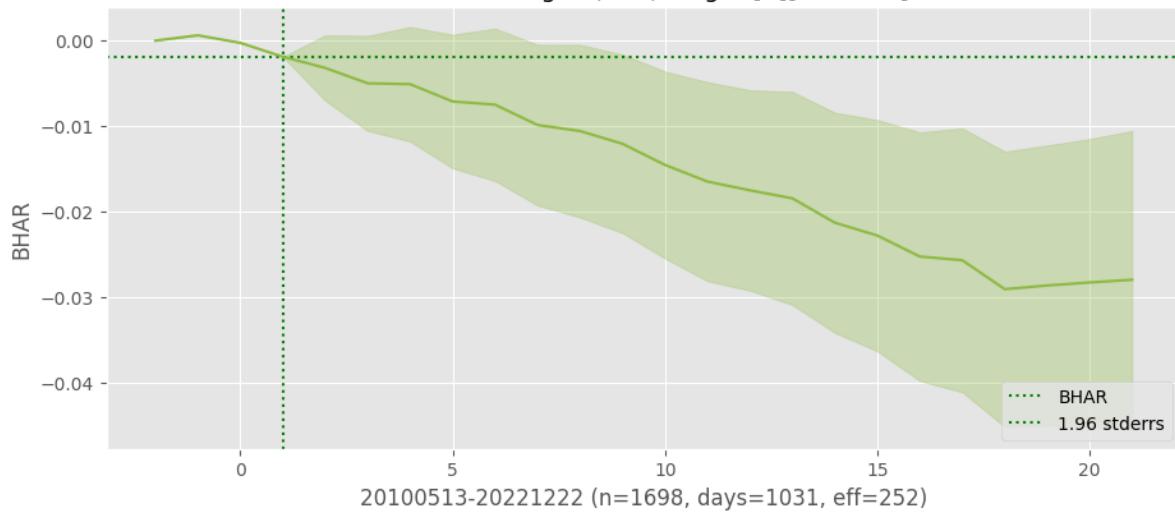
Shareholder/Analyst Calls (50) Target [1][Small]

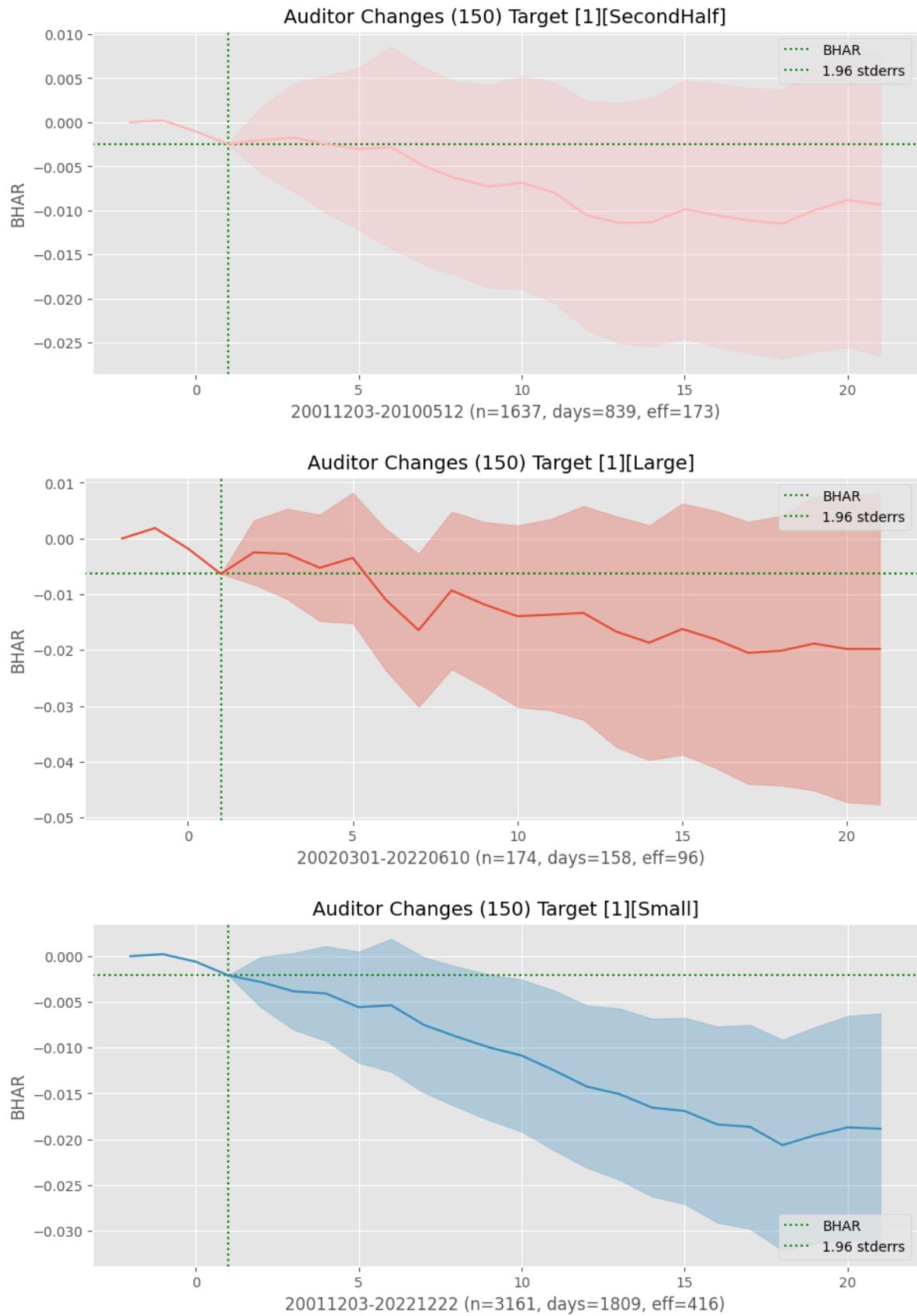


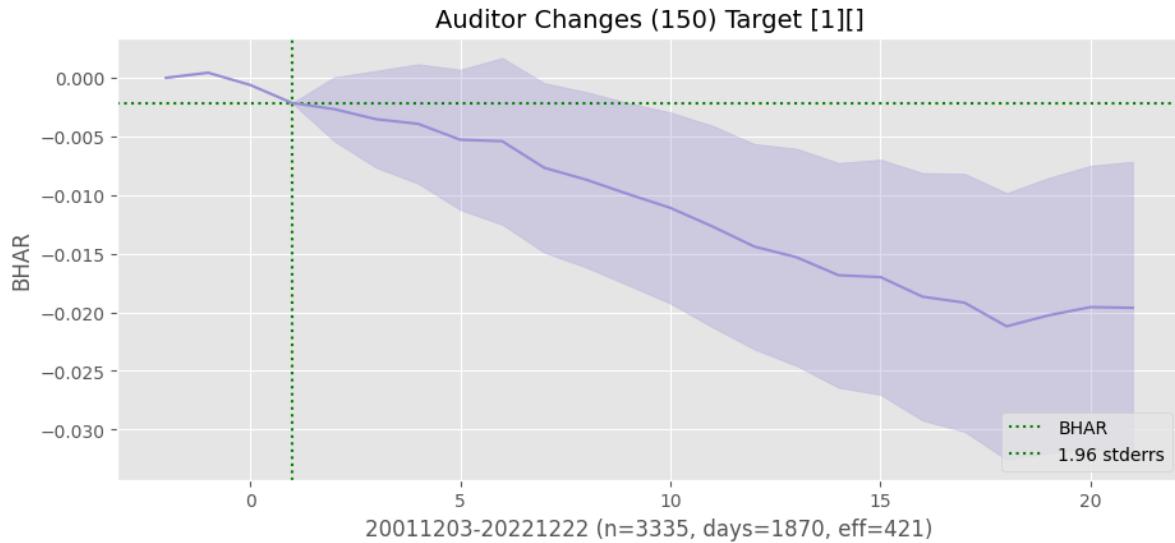
Shareholder/Analyst Calls (50) Target [1][ ]



Auditor Changes (150) Target [1][FirstHalf]



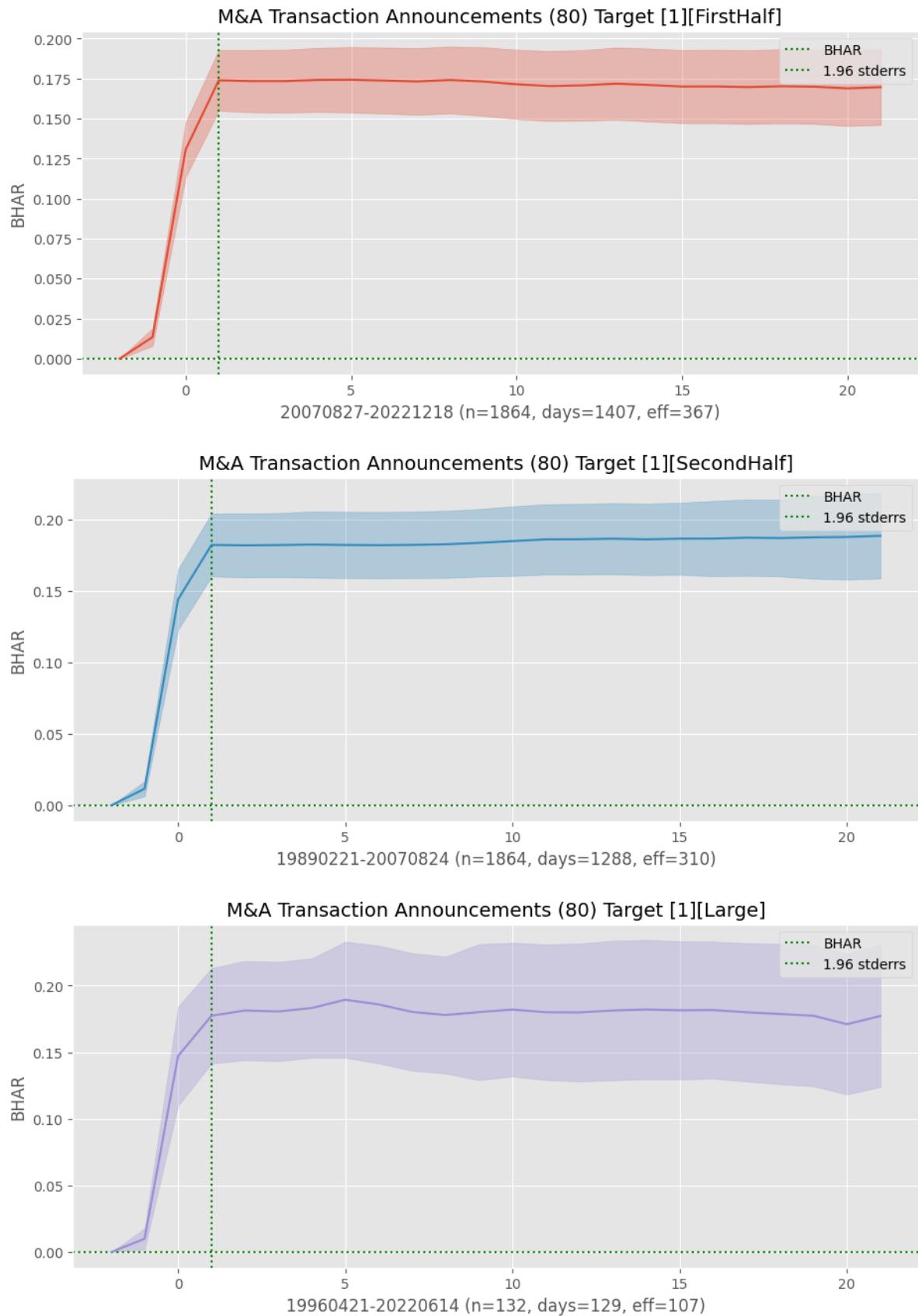


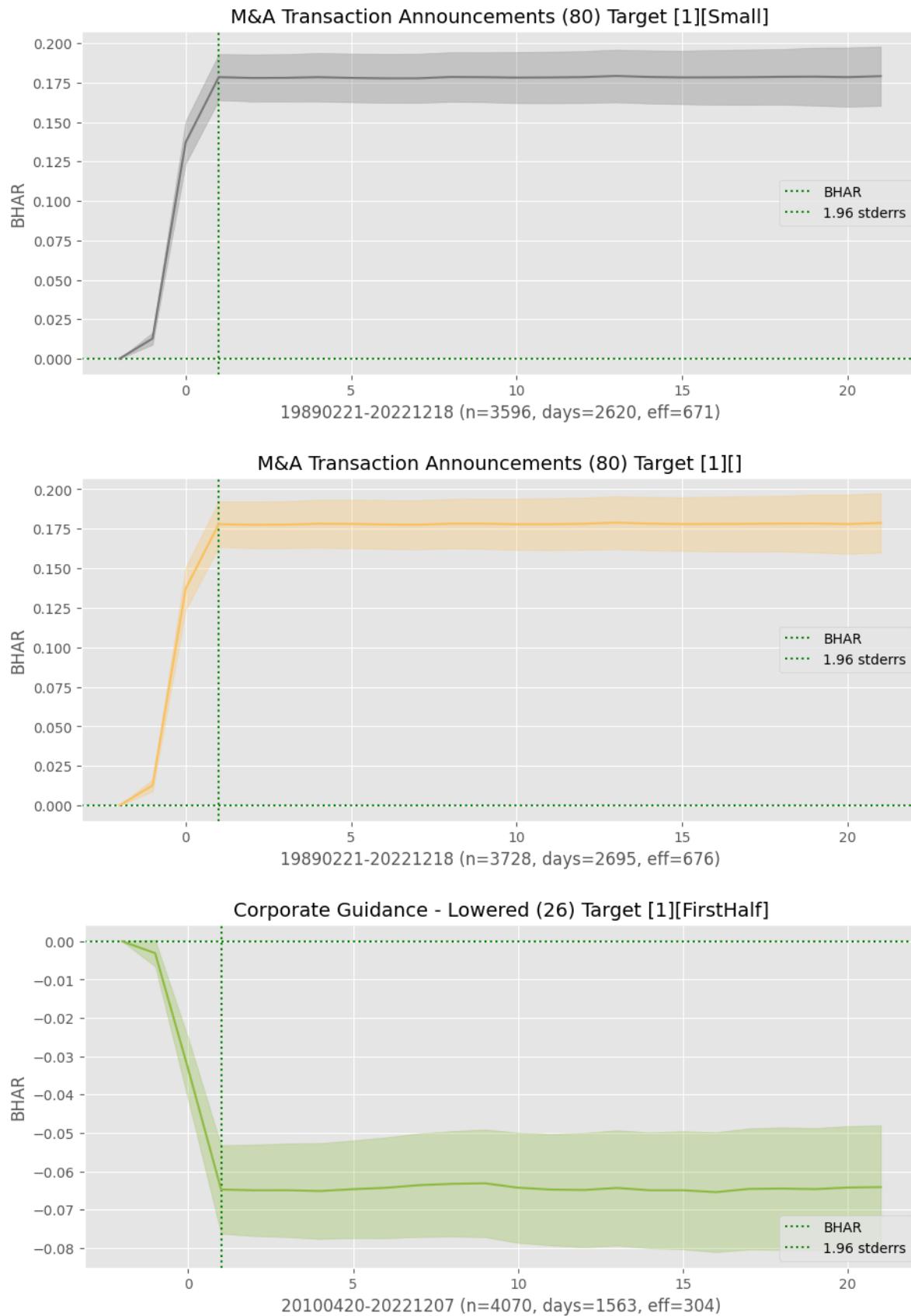


```

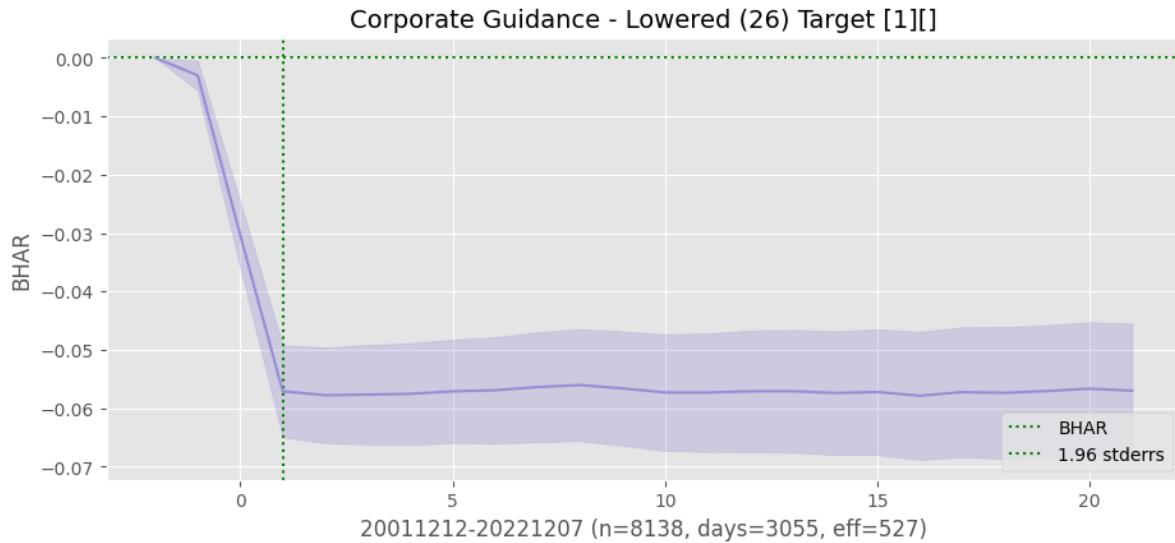
### Show by market cap and half-period
events_list = [[80,1], [26,1]] # top announcement window
for i, (eventid, roleid) in enumerate(events_list):
    #eventid, roleid = 50, 1
    #eventid, roleid = 83, 1
    df = event_pipeline(eventstudy,
                        stocks=crsp,
                        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['cap'].ge(midcap).values,
              'Small': df['cap'].lt(midcap).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}]",
                        drift=False,
                        ax=ax,
                        c=f"C{i*5+ifig}")
        plt.savefig(imgdir / (label + f"{eventid}_{roleid}.jpg"))

```









### Compute BHAR and CAR of all events

```

restart = 0
for i, eventid in tqdm(enumerate(events)):
    if eventid <= restart: # kludge to resume loop
        continue
    for roleid in roles:
        # retrieve all returns observations of this eventid, roleid
        df = event_pipeline(eventstudy,
                            stocks=crsp,
                            beg=beg,
                            end=end,
                            eventid=eventid,
                            roleid=roleid,
                            left=left,
                            right=right,
                            post=post)
        if df['announcedate'].nunique() < minobs: # 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)
        show(DataFrame.from_dict(car | bhar, orient='index'))

        fig, axes = plt.subplots(2, 1, clear=True, figsize=(5, 5), num=1)
        eventstudy.plot(model='sbhar', ax=axes[0], title=eventstudy.label,
                        fontsize=8, vline=[right])
        eventstudy.plot(model='scar', ax=axes[1], title='',
                        fontsize=8, vline=[right])
        plt.savefig(imgdir / f"{eventid}_{roleid}.jpg")
    
```

33it [11:47:02, 1289.68s/it]

### Summarize BHAR's of all events

```
# sorted by 3-day event window abnormal returns
df = eventstudy.read_summary(model='sbhar') \
    .set_index('label') \
    .drop(columns=['rho', 'tau', 'created']) \
    .sort_values('window', ascending=False)
```

```
# convert (eventid, roleid) to multiindex
df = df[df['days'] > 500].sort_values('post_t')
multiIndex = DataFrame(df.index.str.split('_').to_list()).astype(int)
df.index = pd.MultiIndex.from_frame(multiIndex, names=['eventid', 'roleid'])
```

```
# show summary
df['event'] = keydev._event[df.index.get_level_values(0)].values
df['role'] = keydev._role[df.index.get_level_values(1)].values
show(df.set_index(['event', 'role']).drop(columns=['model']),
      caption="Post-Announcement Drift")
```

## MULTIPLE TESTING

```
### Show actual two-sided p-values vs expected (with continuity correction)
pvals = list(norm.cdf(-df['post_t'].abs()) * 2)
argmin = np.argmin(pvals)
header = df.iloc[argmin][['event', 'role', 'days', 'effective', 'post_t']]\n    .to_dict()
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()
plt.savefig(imgdir / 'pvals.jpg')
```

```
### Bonferroni, Holm and Benjamin-Hochberg methods
alpha = 0.05
results = {}
for method in ['bonferroni', 'holm', 'fdr_bh']:
    tests = multipletests(pvals, alpha=alpha, method=method)
    results[method] = header | {f'pval(alpha={alpha})': pvals[argmin],
                                 'adj-pval': tests[1][argmin]}
show(DataFrame.from_dict(results, orient='index'),
      caption="Multiple Testing methods")
```



## ECONOMIC DATA REVISIONS

### UNDER CONSTRUCTION

- St Louis Fed FRED: popular series, api
- ALFRED: archival, releases, vintages, revisions
- FRED-MD: release dates

<https://journals.ala.org/index.php/dtpp/article/view/6383/8404>

```
import time
import numpy as np
import pandas as pd
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
from finds.readers import Alfred, fred_md, fred_qd
from finds.misc import Show
from finds.plots import plot_date
from datetime import datetime
from secret import credentials, paths
show = Show(ndigits=4, latex=None)
VERBOSE = 0
# %matplotlib qt
imgdir = paths['images'] / 'ts'
today = int(datetime.today().strftime('%Y%m%d'))
```

```
alf = Alfred(api_key=credentials['fred']['api_key'],
            savefile=imgdir / 'revisions.pkl',
            verbose=VERBOSE)
```

### 9.1 Popular FRED series

```
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']]
show(DataFrame.from_dict(popular, orient='index'),
     caption=f"Popular Series in FRED, retrieved {today}")
```

```

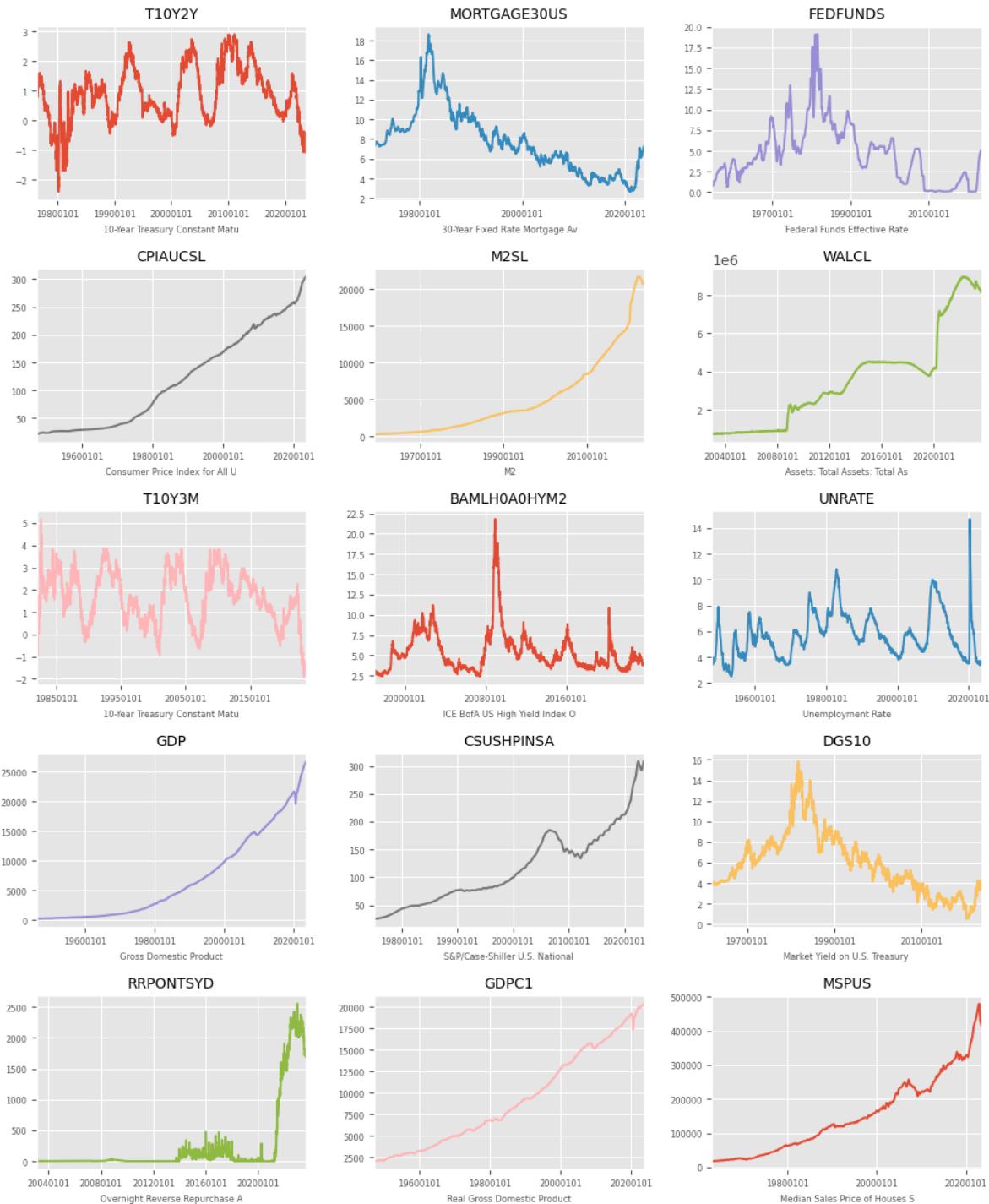
    ↵      title
Popular Series in FRED, retrieved 20230831
T10Y2Y          10-Year Treasury Constant Maturity
    ↵Minus 2-Yea... \
MORTGAGE30US    30-Year Fixed Rate Mortgage Average in_
    ↵the Uni...
FEDFUNDS
    ↵Effective Rate
CPIAUCSL        Consumer Price Index for All Urban_
    ↵Consumers: ...
M2SL
    ↵      M2
WALCL          Assets: Total Assets: Total Assets
    ↵(Less Elimi...
T10Y3M          10-Year Treasury Constant Maturity
    ↵Minus 3-Mon...
BAMLHOA0HYM2    ICE BofA US High Yield Index Option-
    ↵Adjusted S...
UNRATE
    ↵Unemployment Rate
GDP
    ↵Domestic Product
CSUSHPINSA      S&P/Case-Shiller U.S. National Home_
    ↵Price Index
DGS10          Market Yield on U.S. Treasury
    ↵Securities at 10...
RRPONTSYD      Overnight Reverse Repurchase
    ↵Agreements: Treas...
GDPC1          Real Gross
    ↵Domestic Product
MSPUS          Median Sales Price of Houses Sold for_
    ↵the Unit...
T10YIE          10-Year Breakeven
    ↵Inflation Rate
DFII10          Market Yield on U.S. Treasury
    ↵Securities at 10...

popularity
Popular Series in FRED, retrieved 20230831
T10Y2Y          100
MORTGAGE30US    100
FEDFUNDS        98
CPIAUCSL        95
M2SL            93
WALCL            95
T10Y3M            95
BAMLHOA0HYM2    94
UNRATE            93
GDP              92
CSUSHPINSA      91
DGS10            92
RRPONTSYD        92
GDPC1            90
MSPUS            91
T10YIE            90
DFII10            87

```

```
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=title, xlabel=alf.header(title)[:30],
              fontsize=6, ls='-', cn=cn, nbins=4)
plt.tight_layout()
```

```
/home/terence/Dropbox/github/data-science-notebooks/finds/plots.py:159:_
  ↵UserWarning: 'set_params()' not defined for locator of type <class 'pandas.
  ↵plotting._matplotlib.converter.PandasAutoDateLocator'>
    plt.locator_params(axis='x', nbins=nbins)  # numeric ticks
/tmp/ipykernel_4096016/2958181288.py:6: UserWarning: The figure layout has changed_
  ↵to tight
    plt.tight_layout()
```



```
# Traversing categories tree
node = 0
while True:
    node = alf.get_category(node)
    print(f"[{node['id']}]", node['name'],
          f"({#children = {len(node['children'])}})",
          f"({#series = {len(node['series'])}})")
```

(continues on next page)

(continued from previous page)

```

if not (node['children']):
    break
node = np.min([child['id'] for child in node['children']])
for i, row in enumerate(node['series']):
    print(i, row['id'], row['title'])

```

```

https://api.stlouisfed.org/fred/category?category_id=0&api_
  ↵key=eb428f329807459fa7bee89be4ebbc3b&file_type=json
https://api.stlouisfed.org/fred/category/children?category_id=0&api_
  ↵key=eb428f329807459fa7bee89be4ebbc3b&file_type=json
https://api.stlouisfed.org/fred/category/series?category_id=0&api_
  ↵key=eb428f329807459fa7bee89be4ebbc3b&file_type=json&offset=0
[0] Categories (#children = 8) (#series = 0)
https://api.stlouisfed.org/fred/category?category_id=1&api_
  ↵key=eb428f329807459fa7bee89be4ebbc3b&file_type=json
https://api.stlouisfed.org/fred/category/children?category_id=1&api_
  ↵key=eb428f329807459fa7bee89be4ebbc3b&file_type=json
https://api.stlouisfed.org/fred/category/series?category_id=1&api_
  ↵key=eb428f329807459fa7bee89be4ebbc3b&file_type=json&offset=0
https://api.stlouisfed.org/fred/category/series?category_id=1&api_
  ↵key=eb428f329807459fa7bee89be4ebbc3b&file_type=json&offset=1000
https://api.stlouisfed.org/fred/category/series?category_id=1&api_
  ↵key=eb428f329807459fa7bee89be4ebbc3b&file_type=json&offset=2000
https://api.stlouisfed.org/fred/category/series?category_id=1&api_
  ↵key=eb428f329807459fa7bee89be4ebbc3b&file_type=json&offset=3000
https://api.stlouisfed.org/fred/category/series?category_id=1&api_
  ↵key=eb428f329807459fa7bee89be4ebbc3b&file_type=json&offset=4000
[1] Production & Business Activity (#children = 16) (#series = 3274)
https://api.stlouisfed.org/fred/category?category_id=3&api_
  ↵key=eb428f329807459fa7bee89be4ebbc3b&file_type=json
https://api.stlouisfed.org/fred/category/children?category_id=3&api_
  ↵key=eb428f329807459fa7bee89be4ebbc3b&file_type=json
https://api.stlouisfed.org/fred/category/series?category_id=3&api_
  ↵key=eb428f329807459fa7bee89be4ebbc3b&file_type=json&offset=0
https://api.stlouisfed.org/fred/category/series?category_id=3&api_
  ↵key=eb428f329807459fa7bee89be4ebbc3b&file_type=json&offset=1000
https://api.stlouisfed.org/fred/category/series?category_id=3&api_
  ↵key=eb428f329807459fa7bee89be4ebbc3b&file_type=json&offset=2000
https://api.stlouisfed.org/fred/category/series?category_id=3&api_
  ↵key=eb428f329807459fa7bee89be4ebbc3b&file_type=json&offset=3000
[3] Industrial Production & Capacity Utilization (#children = 1) (#series = 2624)
https://api.stlouisfed.org/fred/category?category_id=33938&api_
  ↵key=eb428f329807459fa7bee89be4ebbc3b&file_type=json
https://api.stlouisfed.org/fred/category/children?category_id=33938&api_
  ↵key=eb428f329807459fa7bee89be4ebbc3b&file_type=json
https://api.stlouisfed.org/fred/category/series?category_id=33938&api_
  ↵key=eb428f329807459fa7bee89be4ebbc3b&file_type=json&offset=0
https://api.stlouisfed.org/fred/category/series?category_id=33938&api_
  ↵key=eb428f329807459fa7bee89be4ebbc3b&file_type=json&offset=1000
[33938] Automotive Production Seasonal Factor (#children = 0) (#series = 7)
0 G17MVSFAUTOS Regular Seasonal Factors: Auto Production
1 G17MVSFLTRUCKS Regular Seasonal Factors: Light Truck Production
2 G17MVSFTTRUCKS Regular Seasonal Factors: Total Truck Production
3 G17MVSFWAUTOS Weekday-Basis Seasonal Factors: Auto Production
4 G17MVSFWLTRUCKS Weekday-Basis Seasonal Factors: Light Truck Production
5 G17MVSFWTTRUCKS Weekday-Basis Seasonal Factors: Total Truck Production

```

(continues on next page)

(continued from previous page)

```
6 G17MVSFWWKDAYS Weekday-Basis Seasonal Factors: Weekdays for Calculation
```

[https://alfred.stlouisfed.org/graph/?graph\\_id=354151](https://alfred.stlouisfed.org/graph/?graph_id=354151)

```
# INDPRO by latest, vintage, revision number, time lag
series_id, freq = 'INDPRO', 'M' # https://www.bea.gov/gdp-revision-information
#series_id = 'CPIAUCSL'
series_id, freq = 'GDPC1', 'Q'
```

```
print(f"Latest revision retrieved {today}:")
print(alf(series_id,
          start=20200401,
          end=20200731,
          freq=freq,
          realtime=True))
```

```
Latest revision retrieved 20230831:
                    GDPC1  realtime_start  realtime_end
date
20200630  17378.712          20220929      99991231
```

```
print("First Release:")
print(alf(series_id,
          release=1,
          start=20200401,
          end=20200731,
          freq=freq,
          realtime=True))
```

```
First Release:
                    GDPC1  realtime_start  realtime_end
date
20200630  17205.822          20200730      20200826
```

```
print("Second Release:")
print(alf(series_id,
          release=2,
          start=20200401,
          end=20200731,
          freq=freq,
          realtime=True))
```

```
Second Release:
                    GDPC1  realtime_start  realtime_end
date
20200630  17282.188          20200827      20200929
```

```
print("Revised Up to 5-months Later:")
print(alf(series_id,
          release=pd.DateOffset(months=5),
          start=20200401,
```

(continues on next page)

(continued from previous page)

```
end=20200731,
freq=freq,
realtime=True))
```

```
Revised Up to 5-months Later:
      GDPC1  realtime_start  realtime_end
date
20200630  17302.511      20200930      20210728
```

```
print("Latest as of Vintage date 2020-06-30:")
print(alf(series_id,
          vintage=20200630,
          realtime=True,
          freq=freq,
          start=20200401))
```

```
Latest as of Vintage date 2020-06-30:
Empty DataFrame
Columns: [GDPC1, realtime_start, realtime_end]
Index: []
```

```
# revisions history up to N-months later
df = pd.concat([alf(series_id,
                     start=today - 60000,
                     freq=freq,
                     release=pd.DateOffset(months=m)) \
                     .rename(f"Revised up to {m}-months later")
                     for m in [1, 3, 9, 21, 33]], axis=1)
df.index = pd.DatetimeIndex(df.index.astype(str))
ax = df.plot(logy=False)
ax.set_title(f"{series_id}: Revisions up to N-months later")
if imgdir:
    plt.savefig(imgdir / 'release_months.jpg')
print(df)
```

	Revised up to 1-months later	Revised up to 3-months later
date		
2017-09-30	17156.946	17163.894 \
2017-12-31	17272.468	17286.497
2018-03-31	17385.831	17371.854
2018-06-30	18507.200	18511.576
2018-09-30	18671.497	18664.973
2019-03-31	18912.326	18910.332
2019-06-30	19023.820	19021.860
2019-09-30	19112.542	19121.112
2019-12-31	19219.767	19221.970
2020-03-31	18987.877	18977.363
2020-06-30	17205.822	17302.511
2020-09-30	18583.984	18596.521
2020-12-31	18780.325	18794.426
2021-03-31	19087.568	19086.375
2021-06-30	19358.176	19368.310
2021-09-30	19465.195	19478.893

(continues on next page)

(continued from previous page)

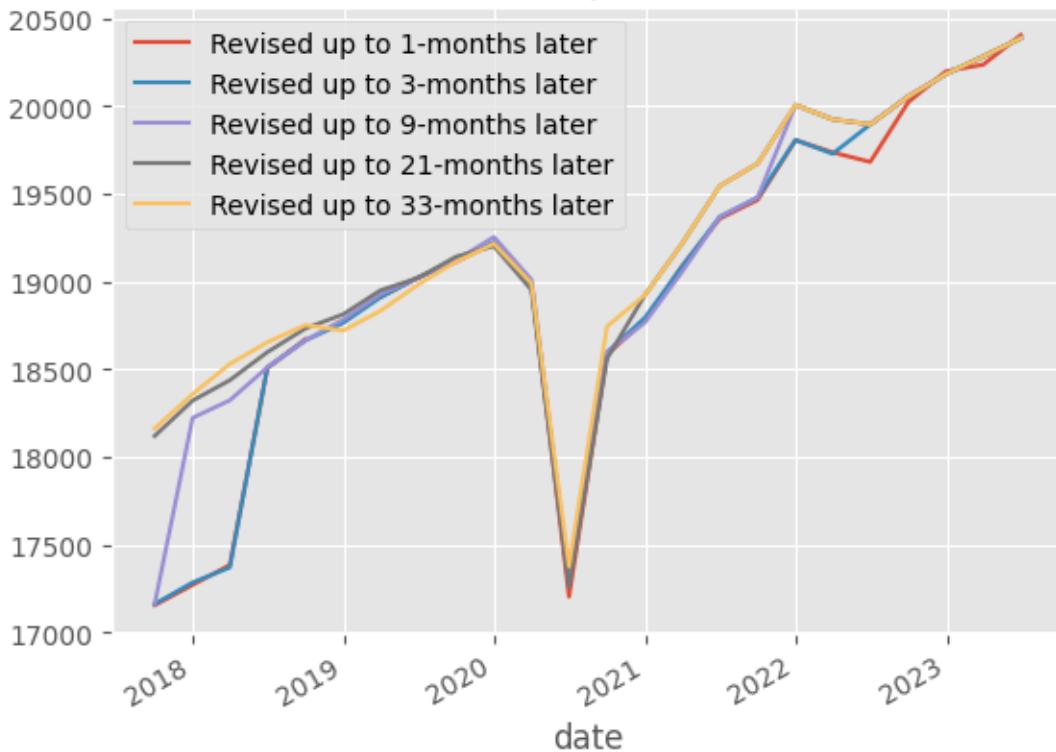
2021-12-31	19805.962	19806.290
2022-03-31	19735.895	19727.918
2022-06-30	19681.682	19895.271
2022-09-30	20021.721	20054.663
2022-12-31	20198.091	20182.491
2023-03-31	20235.878	20282.760
2023-06-30	20404.088	20386.467
2018-12-31	NaN	18765.256
Revised up to 9-months later		
date		
2017-09-30	17163.894	18120.843 \
2017-12-31	18223.758	18322.464
2018-03-31	18323.963	18438.254
2018-06-30	18511.576	18598.135
2018-09-30	18664.973	18732.720
2019-03-31	18927.281	18950.347
2019-06-30	19021.860	19020.599
2019-09-30	19121.112	19141.744
2019-12-31	19253.959	19202.310
2020-03-31	19010.848	18951.992
2020-06-30	17302.511	17258.205
2020-09-30	18596.521	18560.774
2020-12-31	18767.778	18924.262
2021-03-31	19055.655	19216.224
2021-06-30	19368.310	19544.248
2021-09-30	19478.893	19672.594
2021-12-31	20006.181	20006.181
2022-03-31	19924.088	19924.088
2022-06-30	19895.271	19895.271
2022-09-30	20054.663	20054.663
2022-12-31	20182.491	20182.491
2023-03-31	20282.760	20282.760
2023-06-30	20386.467	20386.467
2018-12-31	18783.548	18813.923
Revised up to 33-months later		
date		
2017-09-30	18163.558	
2017-12-31	18359.432	
2018-03-31	18530.483	
2018-06-30	18654.383	
2018-09-30	18752.355	
2019-03-31	18833.195	
2019-06-30	18982.528	
2019-09-30	19112.653	
2019-12-31	19215.691	
2020-03-31	18989.877	
2020-06-30	17378.712	
2020-09-30	18743.720	
2020-12-31	18924.262	
2021-03-31	19216.224	
2021-06-30	19544.248	
2021-09-30	19672.594	
2021-12-31	20006.181	
2022-03-31	19924.088	

(continues on next page)

(continued from previous page)

2022-06-30	19895.271
2022-09-30	20054.663
2022-12-31	20182.491
2023-03-31	20282.760
2023-06-30	20386.467
2018-12-31	18721.281

### GDPC1: Revisions up to N-months later



```
# revisions history by revision number
df = pd.concat([alf(series_id,
                     start=today - 60000,
                     freq=freq,
                     release=n).rename(f"release_{n}")
                 for n in range(1, 9)], axis=1)
#df.index = pd.DatetimeIndex(df.index.astype(str))
plot_date(df, title=f"Revisions of {series_id} by release number")
if imgdir:
    plt.savefig(imgdir / 'release_revisions.jpg')
show(df, caption=f"{series_id} Revisions, retrieved {today}")
```

	release 1	release 2	release 3	\
GDPC1 Revisions, retrieved 20230831				
20170930	17156.946	17169.733	17163.894	\
20171231	17272.468	17271.702	17286.497	
20180331	17385.831	17379.728	17371.854	
20180630	18507.200	18514.595	18511.576	
20180930	18671.497	18671.650	18664.973	

(continues on next page)

(continued from previous page)

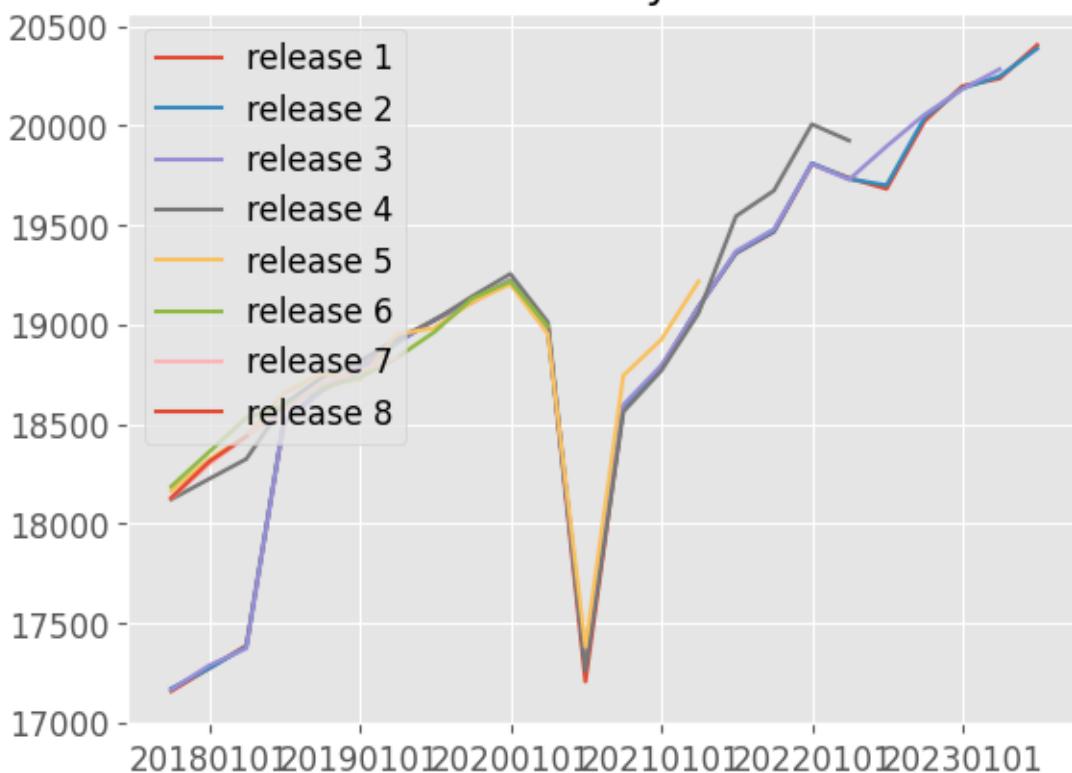
20181231	18784.632	18765.256	18783.548	
20190331	18912.326	18907.517	18910.332	
20190630	19023.820	19023.022	19021.860	
20190930	19112.542	19122.060	19121.112	
20191231	19219.767	19220.490	19221.970	
20200331	18987.877	18974.702	18977.363	
20200630	17205.822	17282.188	17302.511	
20200930	18583.984	18583.501	18596.521	
20201231	18780.325	18783.902	18794.426	
20210331	19087.568	19088.064	19086.375	
20210630	19358.176	19360.600	19368.310	
20210930	19465.195	19469.398	19478.893	
20211231	19805.962	19810.572	19806.290	
20220331	19735.895	19731.119	19727.918	
20220630	19681.682	19699.465	19895.271	
20220930	20021.721	20039.406	20054.663	
20221231	20198.091	20187.495	20182.491	
20230331	20235.878	20246.439	20282.760	
20230630	20404.088	20386.467	NaN	
release 4    release 5    release 6				
GDPC1 Revisions, retrieved 20230831				
20170930	18120.843	18163.558	18185.636	\
20171231	18223.758	18322.464	18359.432	
20180331	18323.963	18438.254	18530.483	
20180630	18598.135	18654.383	18590.004	
20180930	18732.720	18752.355	18679.599	
20181231	18813.923	18721.281	18733.741	
20190331	18927.281	18950.347	18833.195	
20190630	19020.599	18982.528	18962.175	
20190930	19141.744	19112.653	19130.932	
20191231	19253.959	19202.310	19215.691	
20200331	19010.848	18951.992	18989.877	
20200630	17258.205	17378.712	NaN	
20200930	18560.774	18743.720	NaN	
20201231	18767.778	18924.262	NaN	
20210331	19055.655	19216.224	NaN	
20210630	19544.248	NaN	NaN	
20210930	19672.594	NaN	NaN	
20211231	20006.181	NaN	NaN	
20220331	19924.088	NaN	NaN	
20220630	NaN	NaN	NaN	
20220930	NaN	NaN	NaN	
20221231	NaN	NaN	NaN	
20230331	NaN	NaN	NaN	
20230630	NaN	NaN	NaN	
release 7    release 8				
GDPC1 Revisions, retrieved 20230831				
20170930	18126.226	18127.994		
20171231	18296.685	18310.300		
20180331	18436.262	18437.127		
20180630	18565.697	NaN		
20180930	18699.748	NaN		
20181231	NaN	NaN		
20190331	18835.411	NaN		

(continues on next page)

(continued from previous page)

20190630	NaN	NaN
20190930	NaN	NaN
20191231	NaN	NaN
20200331	NaN	NaN
20200630	NaN	NaN
20200930	NaN	NaN
20201231	NaN	NaN
20210331	NaN	NaN
20210630	NaN	NaN
20210930	NaN	NaN
20211231	NaN	NaN
20220331	NaN	NaN
20220630	NaN	NaN
20220930	NaN	NaN
20221231	NaN	NaN
20230331	NaN	NaN
20230630	NaN	NaN

### Revisions of GDPC1 by release number



```
show(alf.observations(series_id, date=today - 60000, freq='Q'))
```

release	realtime_start	realtime_end	date	value
1	2017-10-27	2017-11-28	20170930	17156.946
2	2017-11-29	2017-12-20	20170930	17169.733
3	2017-12-21	2018-07-26	20170930	17163.894

(continues on next page)

(continued from previous page)

4	2018-07-27	2019-07-25	20170930	18120.843
5	2019-07-26	2020-07-29	20170930	18163.558
6	2020-07-30	2021-07-28	20170930	18185.636
7	2021-07-29	2022-09-28	20170930	18126.226
8	2022-09-29	9999-12-31	20170930	18127.994

```

# Release dates of series in FRED-MD collection
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-4, # 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_:
    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: # 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][1:]))

# special case of Consumer Sentiment (date convention)
df = alf('UMCSENT', release=1, realtime=True)
out['UMCSENT'] = df[df['realtime_start'] > end - 4].iloc[:1]

# special case of Claims (weekly averages)
df = alf('ICNSA', release=1, realtime=True)
out['CLAIMS'] = df[df['realtime_start'] > end - 4].iloc[:1]

```

monthly/current.csv

```

# Plot for a representative monthly cross-section
release = Series({k: max(v['realtime_start'])}
#                  for k,v in out.items() if v is not None and len(v) })
#                  .sort_values()
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'))

```

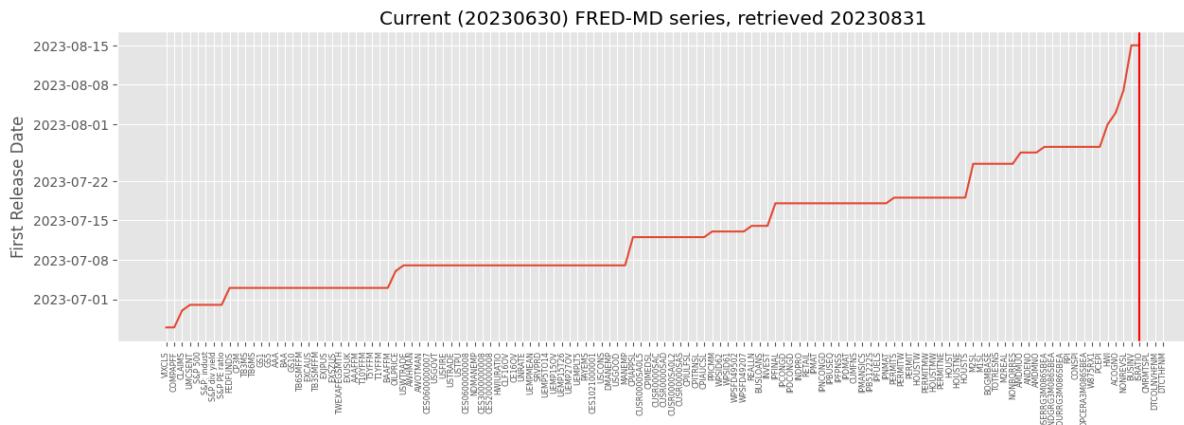
(continues on next page)

(continued from previous page)

```

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(pad=2)
if imgdir:
    plt.savefig(imgdir / 'fredmd_release.jpg')

```



```

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:
    show(alf.splice(series_id).iloc[-3:])

```

Recent values available to update missing in current FRED-MD

```

Series(release.values,
       index=[(s, alf.header(s))
              for s in release.index]).tail(len(md_missing))

```

```

(CONSPI, Nonrevolving consumer credit to Personal Income)
    20230728
(DPCERA3M086SBEA, Real personal consumption expenditures (chain-type quantity))
    20230728
(W875RX1, Real personal income excluding current transfer receipts)
    20230728
(PCEPI, Personal Consumption Expenditures: Chain-type Price Index)
    20230728
(HWI, Help Wanted Index for United States)
    20230801
(ACOGNO, Manufacturers' New Orders: Consumer Goods)
    20230803
(NONREVSL, Nonrevolving Consumer Credit Owned and Securitized)
    20230807
(BUSINV, Total Business Inventories)
    20230815
(ISRATIO, Total Business: Inventories to Sales Ratio)

```

(continues on next page)

(continued from previous page)

```
↪ 20230815
(CMRMTSPL, Real Manufacturing and Trade Industries Sales) ↵
↪ 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
dtype: object
```

## LINEAR REGRESSION DIAGNOSTICS

### UNDER CONSTRUCTION

- Linear regression diagnostics, HAC robust standard errors
- Residual analysis: outliers, leverage, influential points
- Multicollinearity, variance inflation factor

```
import numpy as np
import pandas as pd
import pandas as pd
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
import time
import os
import seaborn as sns
import patsy
import statsmodels.formula.api as smf
from statsmodels.stats.outliers_influence import variance_inflation_factor
from finds.readers import Alfred
from finds.plots import plot_fitted, plot_leverage, plot_scale, plot_qq
from finds.misc import Show
from secret import credentials, paths
show = Show(ndigits=4, latex=None)
VERBOSE = 0
# matplotlib qt
```

```
imgdir = paths['images'] / 'ts'
alf = Alfred(api_key=credentials['fred']['api_key'],
             savefile=paths['scratch'] / 'alfred.pkl')
```

```
# difference of logs of CPI and PPI monthly series from FRED
series_id, freq, start = 'CPIAUCSL', 'M', 0    #19740101
exog_id = 'WPSFD4131'
#exog_id = 'INDPRO'

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 changes in logs of the series
```

## 10.1 Linear regression

```
## Run Linear Regression (with one lag)
dmf = (f'{series_id} ~ {series_id}.shift(1) ')
model = smf.ols(formula=dmf, data=data).fit()
print(model.summary())
```

OLS Regression Results

Dep. Variable:	CPIAUCSL	R-squared:	0.435			
Model:	OLS	Adj. R-squared:	0.434			
Method:	Least Squares	F-statistic:	454.7			
Date:	Wed, 30 Aug 2023	Prob (F-statistic):	2.93e-75			
Time:	10:00:03	Log-Likelihood:	2720.3			
No. Observations:	593	AIC:	-5437.			
Df Residuals:	591	BIC:	-5428.			
Df Model:	1					
Covariance Type:	nonrobust					
975]	coef	std err	t	P> t	[0.025	0.
Intercept	0.0011	0.000	7.609	0.000	0.001	0.
CPIAUCSL.shift(1)	0.6566	0.031	21.323	0.000	0.596	0.
Omnibus:	62.951	Durbin-Watson:	2.031			
Prob(Omnibus):	0.000	Jarque-Bera (JB):	210.870			
Skew:	-0.459	Prob(JB):	1.62e-46			
Kurtosis:	5.774	Cond. No.	304.			

Notes:

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

```
## Run Linear Regression (with 2 lags)
dmf = (f'{series_id} ~ {series_id}.shift(1) + {series_id}.shift(2) ')
model = smf.ols(formula=dmf, data=data).fit()
print(model.summary())
```

OLS Regression Results

Dep. Variable:	CPIAUCSL	R-squared:	0.431
Model:	OLS	Adj. R-squared:	0.429
Method:	Least Squares	F-statistic:	223.2
Date:	Wed, 30 Aug 2023	Prob (F-statistic):	6.98e-73
Time:	10:00:03	Log-Likelihood:	2715.9
No. Observations:	592	AIC:	-5426.
Df Residuals:	589	BIC:	-5413.
Df Model:	2		
Covariance Type:	nonrobust		

(continues on next page)

(continued from previous page)

```
=====
            coef      std err          t      P>|t|      [0.025      0.
↳975]
=====
Intercept      0.0011      0.000      7.134      0.000      0.001      0.
↳001
CPIAUCSL.shift(1)  0.6401      0.041     15.550      0.000      0.559      0.
↳721
CPIAUCSL.shift(2)  0.0206      0.041      0.503      0.615     -0.060      0.
↳101
=====
Omnibus:          64.147  Durbin-Watson:          2.005
Prob(Omnibus):    0.000  Jarque-Bera (JB):      220.795
Skew:             -0.461  Prob(JB):            1.13e-48
Kurtosis:          5.846  Cond. No.           521.
=====
```

Notes:

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

```
## Run Linear Regression (with exog and lag)
dmf = (f'{series_id} ~ {series_id}.shift(1) + {exog_id}.shift(1)')
model = smf.ols(formula=dmf, data=data).fit()
print(model.summary())
```

```
OLS Regression Results
=====
Dep. Variable:      CPIAUCSL      R-squared:          0.462
Model:              OLS          Adj. R-squared:      0.460
Method:             Least Squares      F-statistic:       253.1
Date:              Wed, 30 Aug 2023      Prob (F-statistic): 4.20e-80
Time:              10:00:03          Log-Likelihood:     2734.8
No. Observations:  593          AIC:            -5464.
Df Residuals:      590          BIC:            -5451.
Df Model:           2
Covariance Type:  nonrobust
=====
            coef      std err          t      P>|t|      [0.025      0.
↳975]
=====
Intercept      0.0009      0.000      5.960      0.000      0.001      0.
↳001
CPIAUCSL.shift(1)  0.5632      0.035     16.262      0.000      0.495      0.
↳631
WPSFD4131.shift(1)  0.1893      0.035      5.440      0.000      0.121      0.
↳258
=====
Omnibus:          114.640  Durbin-Watson:          2.024
Prob(Omnibus):    0.000  Jarque-Bera (JB):      562.787
Skew:             -0.763  Prob(JB):            6.20e-123
Kurtosis:          7.522  Cond. No.           429.
=====
```

(continues on next page)

(continued from previous page)

Notes:

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

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

```
OLS Regression Results
=====
Dep. Variable: CPIAUCSL R-squared: 0.462
Model: OLS Adj. R-squared: 0.460
Method: Least Squares F-statistic: 173.6
Date: Wed, 30 Aug 2023 Prob (F-statistic): 4.98e-60
Time: 10:00:03 Log-Likelihood: 2734.8
No. Observations: 593 AIC: -5464.
Df Residuals: 590 BIC: -5451.
Df Model: 2
Covariance Type: HAC
=====
            coef    std err      t      P>|t|      [ 0.025      0.
975]
-----
Intercept  0.0009    0.000    5.474    0.000    0.001    0.
001
CPIAUCSL.shift(1)  0.5632    0.065    8.609    0.000    0.435    0.
692
WPSFD4131.shift(1)  0.1893    0.055    3.443    0.001    0.081    0.
297
=====
Omnibus: 114.640 Durbin-Watson: 2.024
Prob(Omnibus): 0.000 Jarque-Bera (JB): 562.787
Skew: -0.763 Prob(JB): 6.20e-123
Kurtosis: 7.522 Cond. No. 429.
=====
```

Notes:

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

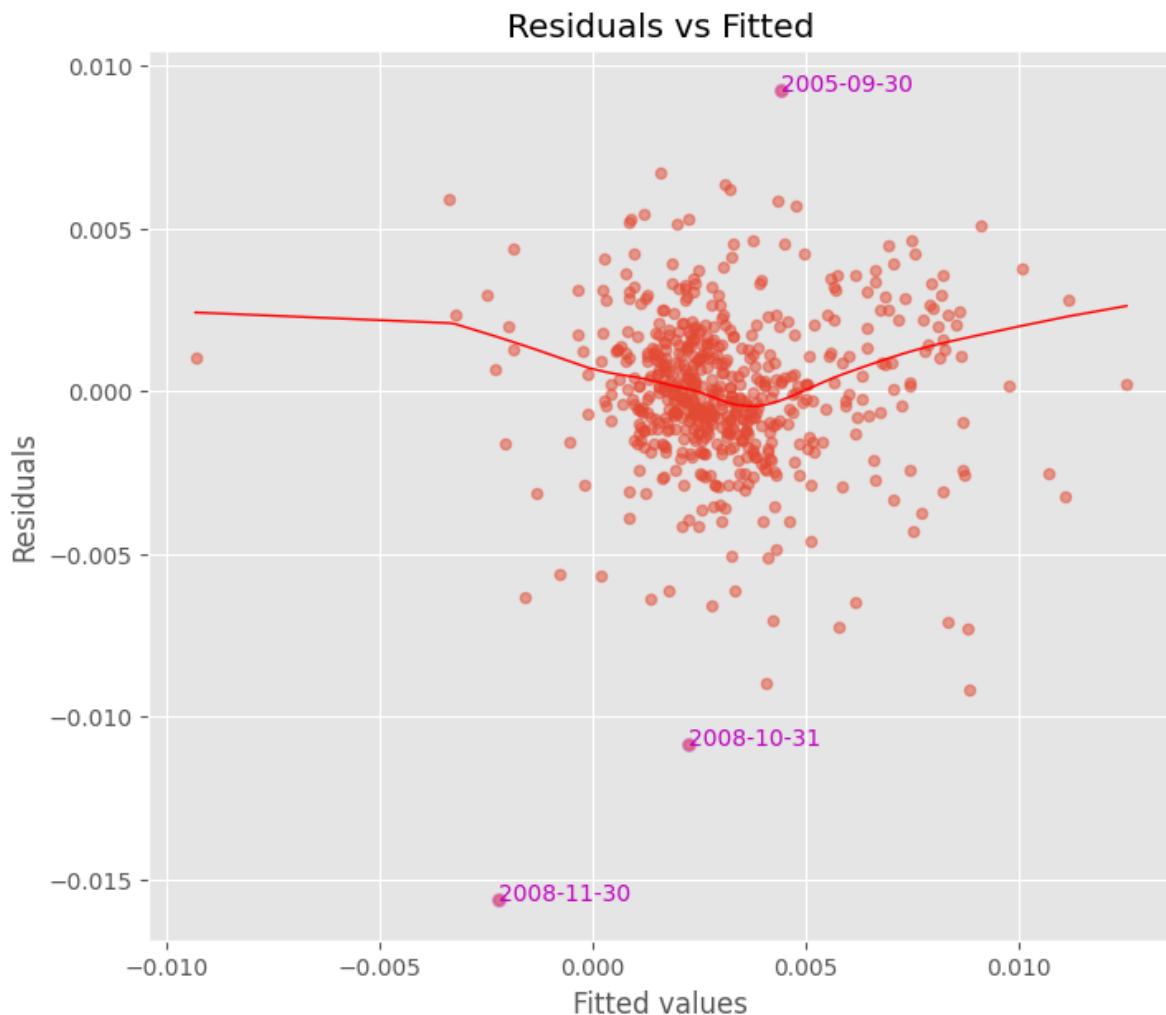
```
Y, X = patsy.dmatrices(dmf + ' - 1', data=data) # exclude intercept term
show(Series({X.design_info.column_names[i]: variance_inflation_factor(X, i)
             for i in range(X.shape[1])}, name='VIF').to_frame(),
      caption="Variance Inflation Factors")
```

```
VIF
Variance Inflation Factors
CPIAUCSL.shift(1)      2.0559
WPSFD4131.shift(1)      2.0559
```

## 10.2 Residual plots

```
## 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)
plt.savefig(imgdir / 'outliers.jpg')
show(z.to_frame().T, caption="Residual Outliers")
# plt.show()
```

date	2005-09-30	2008-10-31	2008-11-30
Residual Outliers			
outliers	0.0092	-0.0109	-0.0156



```
## QQ Plot of residuals and identify outliers
fig, ax = plt.subplots(clear=True, figsize=(8,7))
z = plot_qq(model.resid, ax=ax)
plt.savefig(imgdir / 'qq.jpg')
```

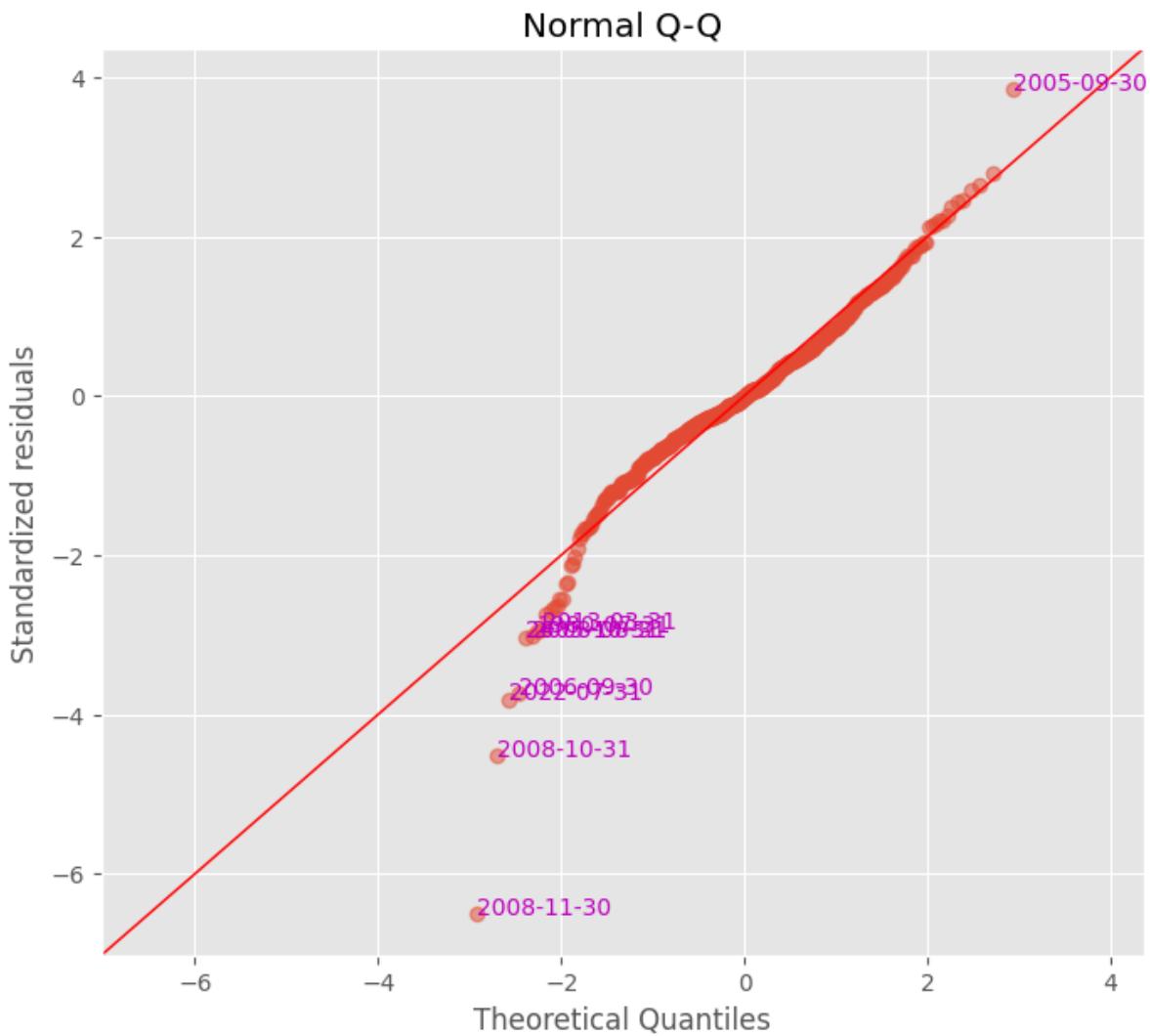
(continues on next page)

(continued from previous page)

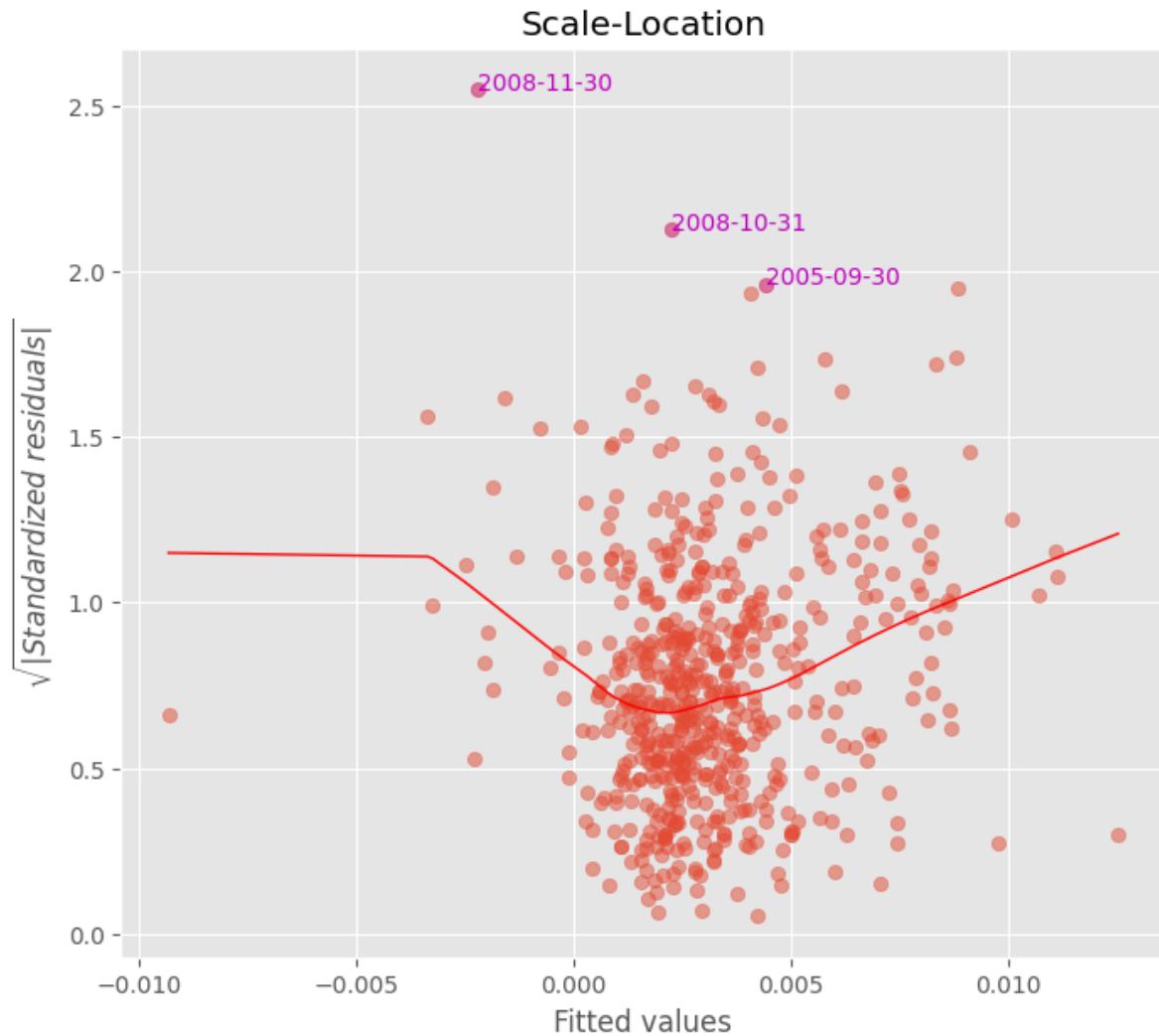
```
#plt.show()  
z
```

```
/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		
2008-11-30	-0.015636	-6.505301
2008-10-31	-0.010866	-4.521038
2022-07-31	-0.009155	-3.809182
2006-09-30	-0.008979	-3.735893
2005-10-31	-0.007286	-3.031274
2008-08-31	-0.007255	-3.018455
1980-07-31	-0.007105	-2.955960
2013-03-31	-0.007025	-2.922866
2005-09-30	0.009246	3.846821



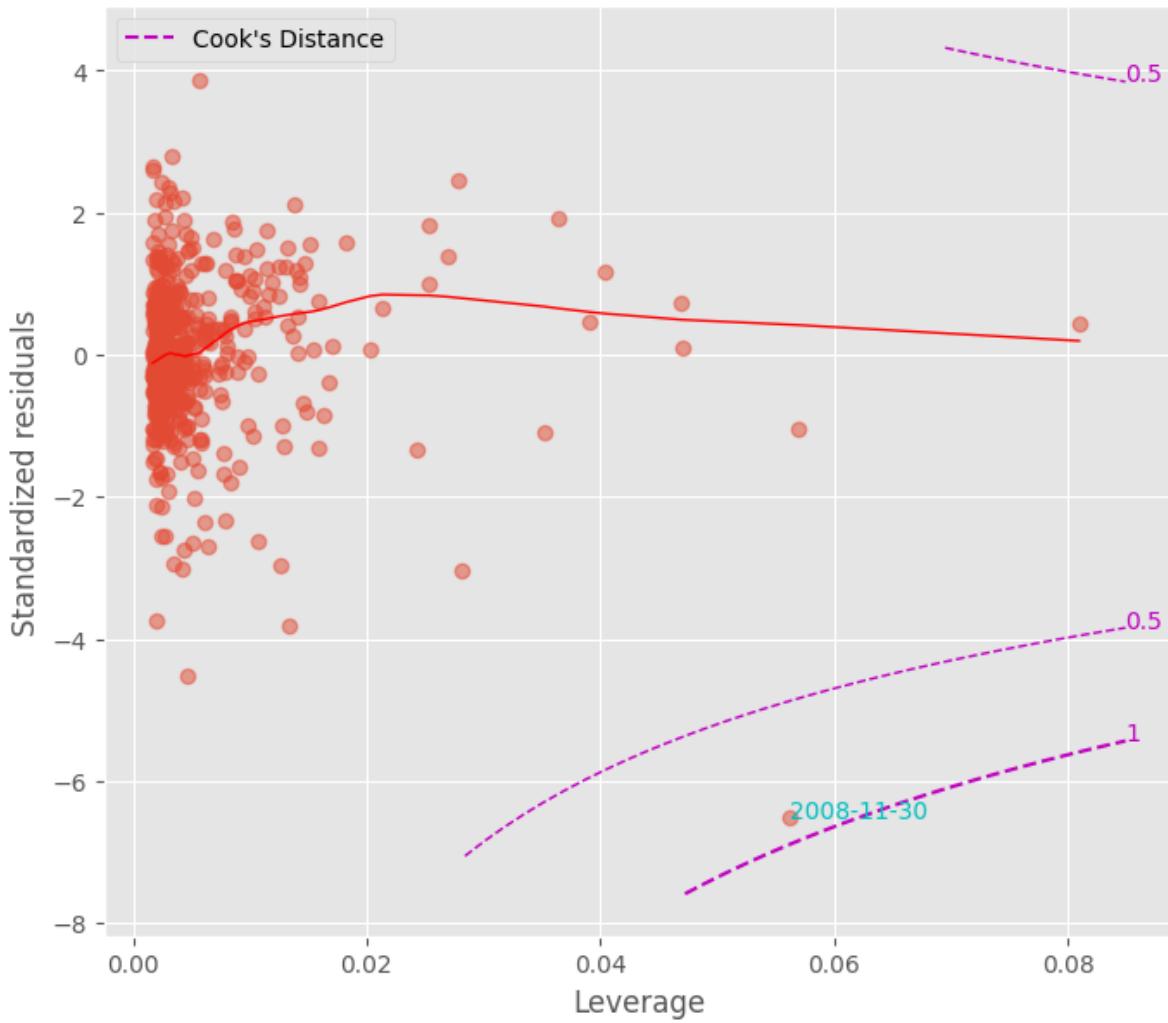
```
## Plot scale of residuals with outliers
fig, ax = plt.subplots(clear=True, figsize=(8,7))
plot_scale(model.fittedvalues, model.resid, ax=ax)
plt.savefig(imgdir / 'scale.jpg')
```



```
## Plot leverage and identify influential points
fig, ax = plt.subplots(clear=True, figsize=(8,7))
z = plot_leverage(model.resid, model.get_influence().hat_matrix_diag,
                  model.get_influence().cooks_distance[0],
                  ddof=len(model.params), ax=ax)
plt.savefig(imgdir / 'leverage.jpg')
z
```

```
Empty DataFrame
Columns: [influential, cook's D, leverage]
Index: []
```

## Residuals vs Leverage





## ECONOMETRIC FORECASTS

### UNDER CONSTRUCTION

- Trends: seasonality
- Autocorrelation Function: AR, MA, SARIMAX
- Unit root: integration order
- Forecasting: single-step, multi-step
- Granger causality, impulse response function, Vector Autoregression

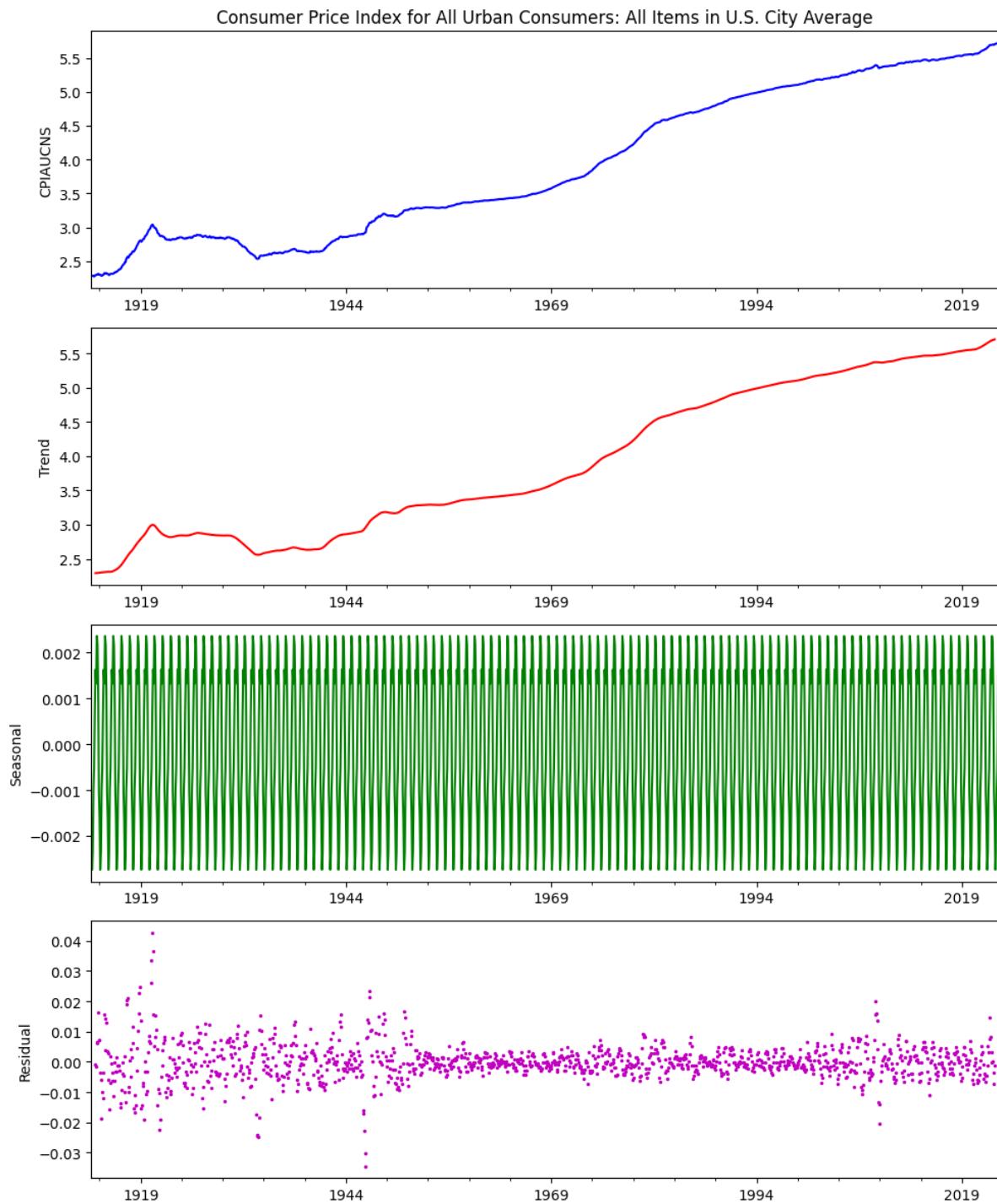
```
import numpy as np
import pandas as pd
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
import time
import os
import seaborn as sns
from statsmodels.tsa.ar_model import AutoReg, ar_select_order
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf, acf
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tsa.stattools import grangercausalitytests
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 finds.busday import BusDay
from finds.econs import integration_order
from finds.misc import Show
from secret import paths, credentials
show = Show(ndigits=4, latex=None)
VERBOSE = 0
# %matplotlib qt
```

```
imgdir = paths['images'] / 'ts'
alf = Alfred(api_key=credentials['fred']['api_key'])
```

```
series_id, freq, start = 'CPIAUCNS', 'M', 0 #19620101 # not seasonally adjusted
df = alf(series_id, log=1, freq=freq, start=start).dropna()
df.index = BusDay.to_datetime(df.index)
```

## 11.1 Seasonality

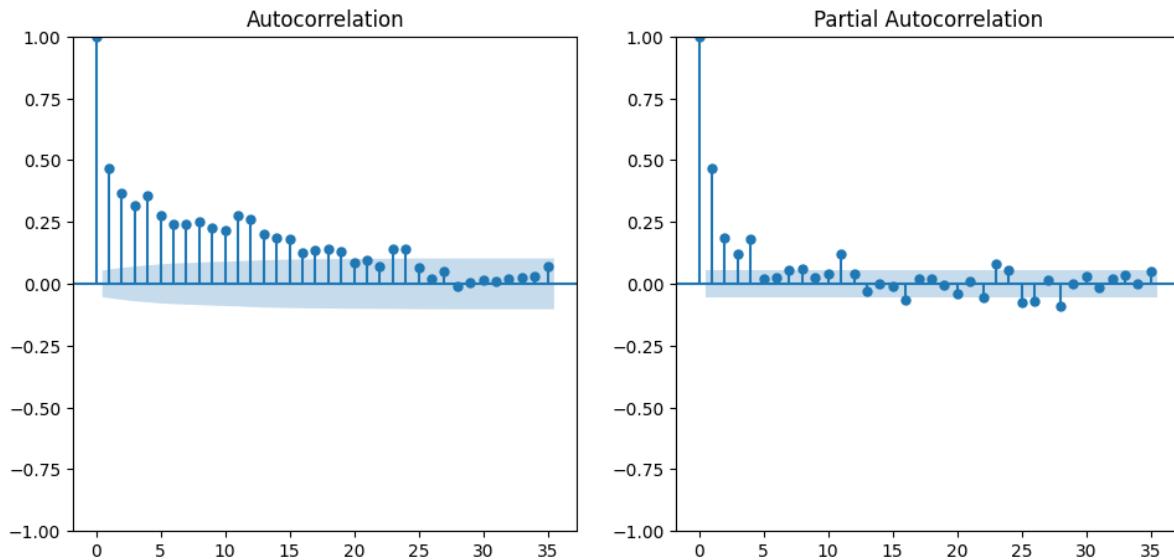
```
## Seasonality Decomposition Plot
result = seasonal_decompose(df, model = 'add')
fig, ax = plt.subplots(nrows=4, ncols=1, clear=True, figsize=(10, 12))
result.observed.plot(ax=ax[0], title=alf.header(result.observed.name),
                      ylabel=result.observed.name, xlabel='', c='b')
result.trend.plot(ax=ax[1], 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()
plt.savefig(imgdir / 'seasonal.jpg')
```



## 11.2 Autocorrelation

Plot ACF and PACF

```
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.tight_layout(pad=2)
plt.savefig(imgdir / 'acf.jpg')
```



## 11.3 Stationarity

Integration Order: Log CPI (Seasonally Adjusted)

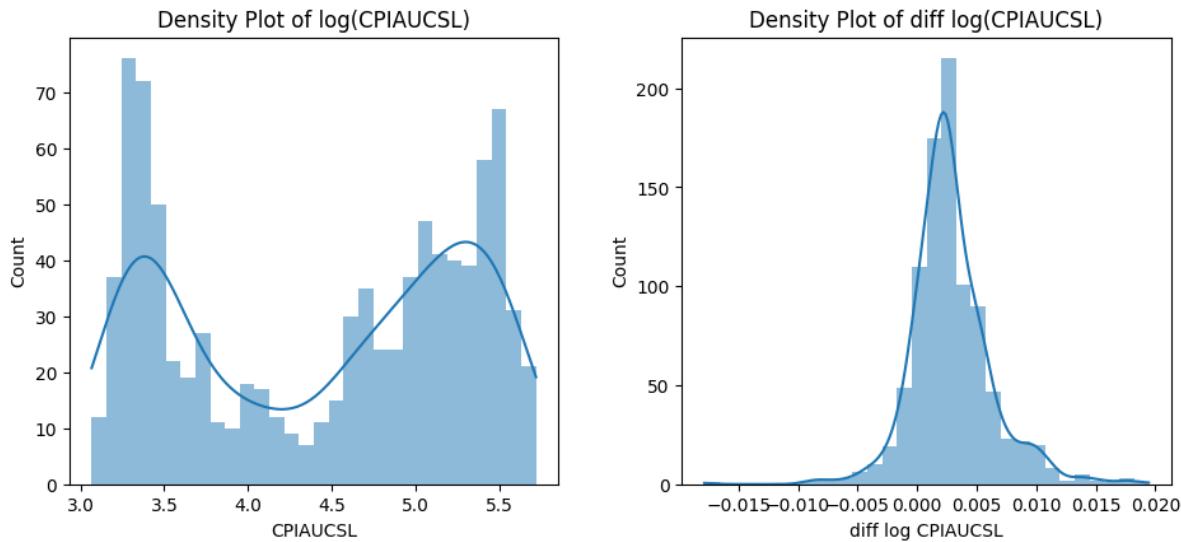
```
series_id = 'CPIAUCSL' # seasonally adjusted
#series_id = 'GDPC1'
df = alf(series_id, log=1, start=0).dropna()
df.index = BusDay.to_datetime(df.index)
p = integration_order(df, noprint=False, pvalue=0.05)
Series({series_id: p}, name='I(p)').to_frame()
```

```
Augmented Dickey-Fuller unit root test:
    Test Statistic  p-value  Lags Used  Obs Used  Critical Value (1%)  Critical Value (5%)
    ↵Value (5%)  Critical Value (10%)
I(0)      -0.154936  0.943684      15.0      903.0          -3.437612          -2.864746
          -2.568477
    Test Statistic  p-value  Lags Used  Obs Used  Critical Value (1%)  Critical Value (5%)
    ↵Value (5%)  Critical Value (10%)
I(1)      -4.575503  0.000143      14.0      903.0          -3.437612          -2.864746
          -2.568477
```

$I(p)$	
CPIAUCSL	1

### 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,
             #line_kws={"color": "r"},
             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,
             #line_kws={"color": "r"},
             ax=axes[1])
axes[1].set_title(f"Density Plot of diff log({series_id})")
plt.savefig(imgdir / 'order.jpg')
plt.tight_layout(pad=3)
```



## 11.4 AR, ARMA, SARIMAX

- AR(p) is simplest time-model, can nest in SARIMAX(p,d,q,s) with
- integration order  $I(d)$ , moving average  $MA(q)$ , seasonality  $S(s)$ , exogenous  $X$

```
split_date = '2021-12-31'
series_id, freq, start = 'CPIAUCNS', 'M', 0  # not seasonally adjust
log_df = alf(series_id, log=1)
log_df.index = BusDay.to_datetime(log_df.index)
log_df = log_df.loc[:split_date].dropna()
```

```

pdq = (1, 1, 3)    #(12, 1, 0)
seasonal_pdqs = (0, 0, 0, 12)
arima = SARIMAX(log_df,
                 order=pdq,
                 seasonal_order=seasonal_pdqs,
                 trend='c').fit()
fig = arima.plot_diagnostics(figsize=(10,6), lags=36)
plt.tight_layout(pad=2)
plt.savefig(imgdir / 'ar.jpg')
arima.summary()

```

```

/home/terence/env3.11/lib/python3.11/site-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency M will be used.
self._init_dates(dates, freq)
/home/terence/env3.11/lib/python3.11/site-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency M will be used.
self._init_dates(dates, freq)
This problem is unconstrained.

```

RUNNING THE L-BFGS-B CODE

```

* * *

Machine precision = 2.220D-16
N =           6      M =           10
At X0           0 variables are exactly at the bounds
At iterate      0      f= -3.76541D+00      |proj g|=  1.11379D+02

```

Warning: more than 10 function and gradient evaluations in the last line search. Termination may possibly be caused by a bad search direction.

```

At iterate      5      f= -3.78602D+00      |proj g|=  2.76533D-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

* * *

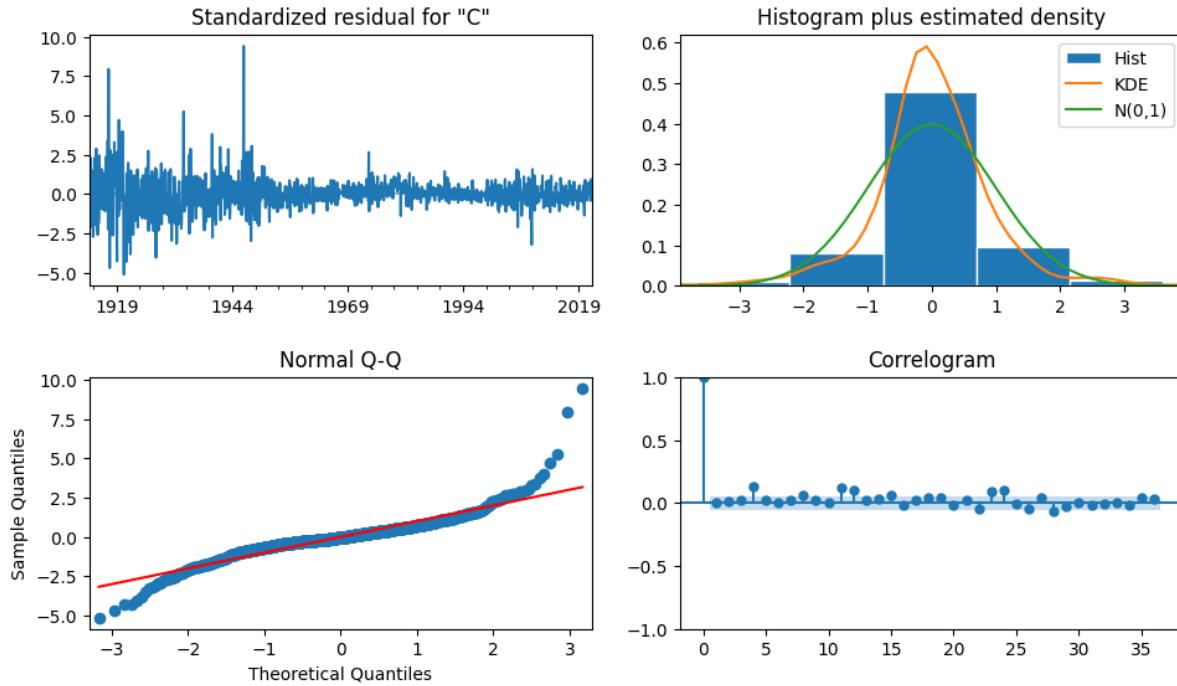
N   Tit   Tnf   Tnint   Skip   Nact   Projg   F
6      5      26      1      0      0   2.765D-01  -3.786D+00
F = -3.7860213397202140

```

(continues on next page)

(continued from previous page)

CONVERGENCE: REL\_REDUCTION\_OF\_F\_&lt;=\_FACTR\*EPSMCH



## 11.5 Forecasting

```
series_id, start = 'CPIAUCSL', 0
df = alf(series_id, log=1, diff=1, start=start).dropna()
df.index = BusDay.to_datetime(df.index)
df_train = df[df.index <= split_date]
df_test = df[df.index > split_date]
```

```
## Select AR lag order
"""ARMA select order is too slow and unstable in statsmodels
# Select ARMA lag order
from statsmodels.tsa.stattools import arma_order_select_ic
series_id = 'CPIAUCSL'
df = alf(series_id, log=1, diff=1).dropna()
df.index = BusDay.to_datetime(df.index)
split_date = '2021-06-30'
df_train = df[df.index <= split_date]
df_test = df[df.index > split_date]
res = arma_order_select_ic(df_train,
                           max_ar=36,
                           max_ma=12,
                           ic='aic',
                           trend='n')
print(' (p, q) = ', res.aic_min_order)
"""
```

```
"ARMA select order is too slow and unstable in statsmodels\n# Select ARMA lag
order\nfrom statsmodels.tsa.stattools import arma_order_select_ic\nseries_id =
'CPIAUCSL'\ndf = alf(series_id, log=1, diff=1).dropna()\ndf.index = BusDay.to_
datetime(df.index)\nsplit_date = '2021-06-30'\ndf_train = df[df.index <= split_
date]\ndf_test = df[df.index > split_date]\nres = arma_order_select_ic(df_train,\n
n
max_ar=36,\n
ic='aic',\n
n
max_ma=12,\n
trend='n')\nprint(
'(p, q) = ', res.aic_min_order)\n"
```

```
lags = ar_select_order(df_train,
    maxlag=36,
    ic='bic',
    old_names=False).ar_lags
print(' (BIC) lags= ', len(lags), ':', lags)
```

```
(BIC) lags= 12 : [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

```
/home/terence/env3.11/lib/python3.11/site-packages/statsmodels/tsa/base/tsa_model.


```

### One-step ahead predictions

```
model = AutoReg(df_train, lags=lags, old_names=False).fit()
print(model.summary())
```

AutoReg Model Results						
Dep. Variable:		CPIAUCSL	No. Observations:	899		
Model:		AutoReg(12)	Log Likelihood	4064.408		
Method:		Conditional MLE	S.D. of innovations	0.002		
Date:		Thu, 31 Aug 2023	AIC	-8100.815		
Time:		06:58:58	BIC	-8033.785		
Sample:		02-29-1948	HQIC	-8075.191		
		- 12-31-2021				
	coef	std err	z	P> z	[0.025	0.975]
const	0.0005	0.000	4.000	0.000	0.000	0.001
CPIAUCSL.L1	0.4431	0.033	13.433	0.000	0.378	0.508
CPIAUCSL.L2	0.0451	0.036	1.255	0.209	-0.025	0.116
CPIAUCSL.L3	0.0371	0.036	1.035	0.301	-0.033	0.107
CPIAUCSL.L4	0.0596	0.035	1.681	0.093	-0.010	0.129
CPIAUCSL.L5	0.0539	0.035	1.551	0.121	-0.014	0.122
CPIAUCSL.L6	-0.0043	0.035	-0.124	0.901	-0.072	0.064
CPIAUCSL.L7	0.1033	0.035	2.976	0.003	0.035	0.171
CPIAUCSL.L8	0.0200	0.035	0.574	0.566	-0.048	0.088
CPIAUCSL.L9	0.0707	0.035	2.039	0.041	0.003	0.139
CPIAUCSL.L10	0.1163	0.034	3.373	0.001	0.049	0.184

(continues on next page)

(continued from previous page)

CPIAUCSL.L11	0.0433	0.034	1.270	0.204	-0.024	0.110
CPIAUCSL.L12	-0.1793	0.031	-5.714	0.000	-0.241	-0.118
Roots						
	Real	Imaginary	Modulus	Frequency		
AR.1	0.8466	-0.7100j	1.1049	-0.1111		
AR.2	0.8466	+0.7100j	1.1049	0.1111		
AR.3	1.0858	-0.0000j	1.0858	-0.0000		
AR.4	1.2766	-0.0000j	1.2766	-0.0000		
AR.5	0.3489	-1.0659j	1.1215	-0.1997		
AR.6	0.3489	+1.0659j	1.1215	0.1997		
AR.7	-0.2554	-1.1060j	1.1351	-0.2861		
AR.8	-0.2554	+1.1060j	1.1351	0.2861		
AR.9	-0.8564	-0.8338j	1.1952	-0.3771		
AR.10	-0.8564	+0.8338j	1.1952	0.3771		
AR.11	-1.1442	-0.3382j	1.1931	-0.4543		
AR.12	-1.1442	+0.3382j	1.1931	0.4543		

```
/home/terence/env3.11/lib/python3.11/site-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
```

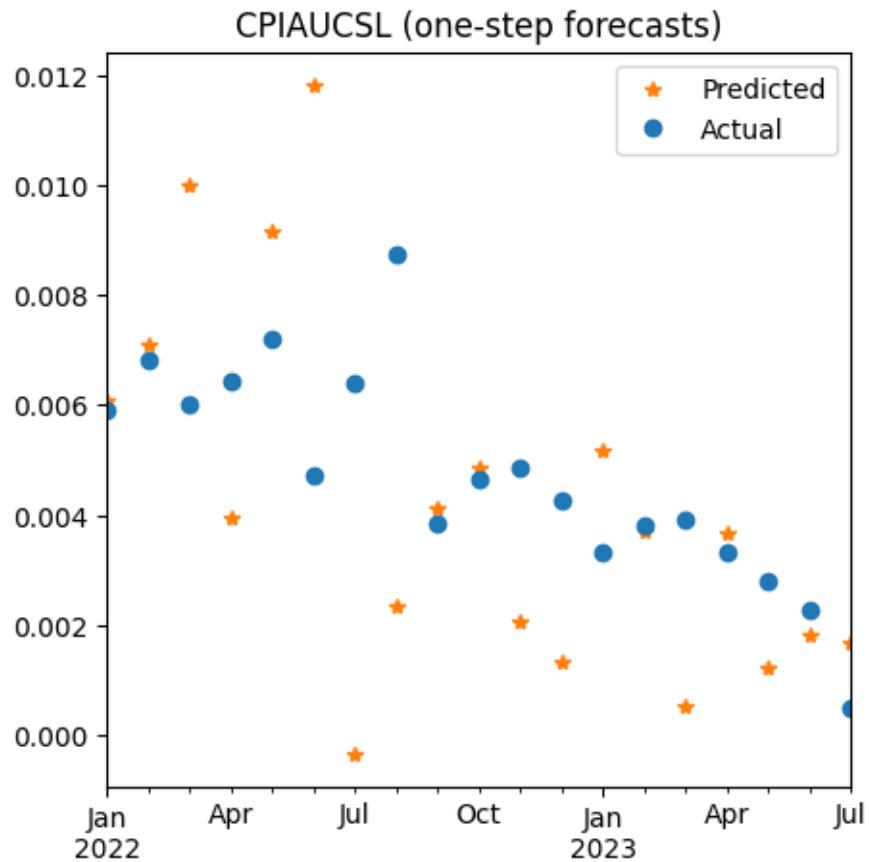
```
### Observations to predict are from the test split
all_dates = AutoReg(df, lags=lags, old_names=False)
```

```
/home/terence/env3.11/lib/python3.11/site-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
```

```
### Use model params from train split, start predictions from last train row
df_pred = all_dates.predict(model.params,
                             start=df_train.index[-1]).shift(1).iloc[1:]
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)} r2={1-mse/var}")
```

```
ST Forecast(19): rmse=0.003233725982906748 r2=0.1608176217788846
```

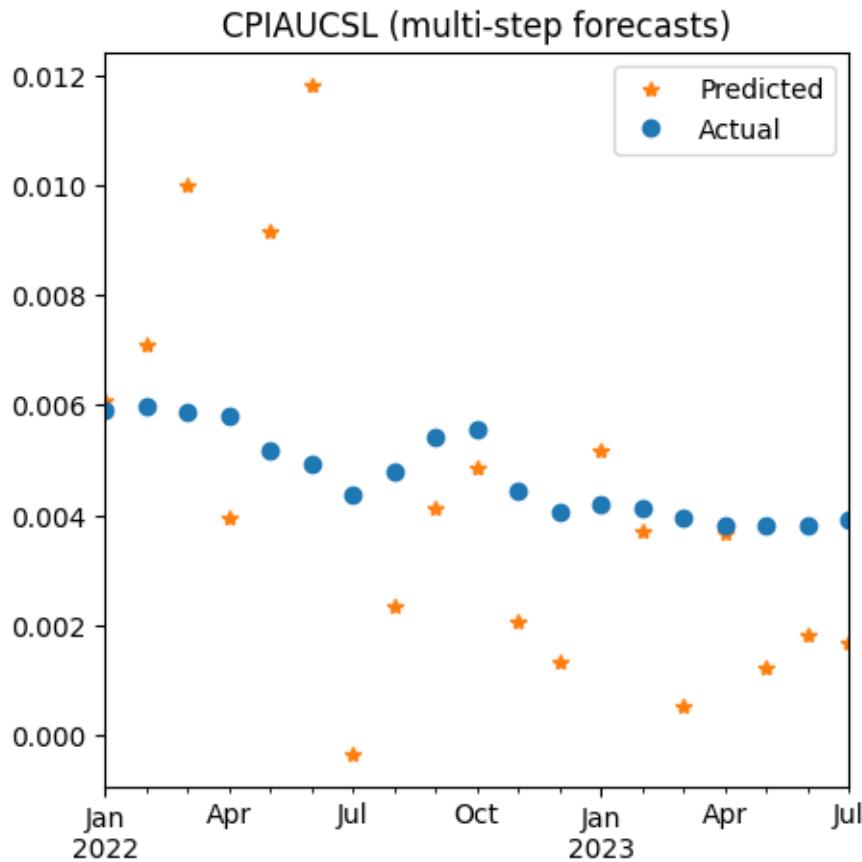
```
fig, ax = plt.subplots(clear=True, num=1, figsize=(5, 5))
df_test.plot(ax=ax, c='C1', ls='', marker='*')
df_pred.plot(ax=ax, c='C0', ls='', marker='o')
ax.legend(['Predicted', 'Actual'])
ax.set_title(series_id + " (one-step forecasts)")
ax.set_xlabel('')
plt.tight_layout(pad=2)
plt.savefig(imgdir / 'short.jpg')
```



### Multi-step ahead predictions

```
df_pred = all_dates.predict(model.params,
                            start=df_train.index[-1],
                            end=df_test.index[-1],
                            dynamic=0).shift(1).iloc[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} r2={1-mse/var:.4f}")
fig, ax = plt.subplots(clear=True, num=2, figsize=(5, 5))
df_test.plot(ax=ax, c='C1', ls='', marker='*')
df_pred.plot(ax=ax, c='C0', ls='', marker='o')
ax.legend(['Predicted', 'Actual'])
ax.set_title(series_id + " (multi-step forecasts)")
ax.set_xlabel('')
plt.tight_layout(pad=2)
plt.savefig(imgdir / 'long.jpg')
```

Long-term Forecasts: rmse=0.002876 r2=0.3363



## 11.6 Granger Causality

```

# Granger Causality: INDPROM vs CPI
variables = ['CPIAUCSL', 'INDPRO']
#variables = ['CPIAUCSL', 'WPSFD4131']
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))
    data = df.diff().dropna()

    print(f"Null Hypothesis: {exog_id} granger-causes {series_id}")
    res = grangercausalitytests(data, 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) ')
    #      f' + {exog_id}.shift(4) ')
    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: INDPRO granger-causes CPIAUCSL

Granger Causality
number of lags (no zero) 1
ssr based F test:      F=0.0229 , p=0.8797 , df_denom=734, df_num=1
ssr based chi2 test:  chi2=0.0230 , p=0.8795 , df=1
likelihood ratio test: chi2=0.0230 , p=0.8795 , df=1
parameter F test:      F=0.0229 , p=0.8797 , df_denom=734, df_num=1

Granger Causality
number of lags (no zero) 2
ssr based F test:      F=0.0166 , p=0.9835 , df_denom=731, df_num=2
ssr based chi2 test:  chi2=0.0335 , p=0.9834 , df=2
likelihood ratio test: chi2=0.0335 , p=0.9834 , df=2
parameter F test:      F=0.0166 , p=0.9835 , df_denom=731, df_num=2

Granger Causality
number of lags (no zero) 3
ssr based F test:      F=0.0969 , p=0.9618 , df_denom=728, df_num=3
ssr based chi2 test:  chi2=0.2934 , p=0.9613 , df=3
likelihood ratio test: chi2=0.2933 , p=0.9613 , df=3
parameter F test:      F=0.0969 , p=0.9618 , df_denom=728, df_num=3

OLS Regression Results
=====
Dep. Variable:          CPIAUCSL    R-squared:           0.391
Model:                 OLS         Adj. R-squared:      0.387
Method:                Least Squares  F-statistic:         53.04
Date:      Thu, 31 Aug 2023  Prob (F-statistic):  3.00e-39
Time:          06:59:01        Log-Likelihood:     3368.4
No. Observations:      735        AIC:                 -6727.
Df Residuals:          730        BIC:                 -6704.
Df Model:                   4
Covariance Type:        HAC
=====
            coef      std err          t      P>|t|      [ 0.025      0.
975]
-----
--+
Intercept      0.0012      0.000      6.712      0.000      0.001      0.
002
CPIAUCSL.shift(1)  0.6243      0.045     13.783      0.000      0.535      0.
713
INDPRO.shift(1)   -0.0020      0.014     -0.140      0.889     -0.029      0.
026
INDPRO.shift(2)    0.0023      0.016      0.143      0.886     -0.030      0.
034
INDPRO.shift(3)    0.0053      0.009      0.580      0.562     -0.013      0.
023
=====
Omnibus:            86.367  Durbin-Watson:           2.132
Prob(Omnibus):      0.000  Jarque-Bera (JB):        717.358
Skew:                 0.034  Prob(JB):            1.69e-156
Kurtosis:                7.839  Cond. No.                 317.
=====

Notes:

```

(continues on next page)

(continued from previous page)

```
[1] Standard Errors are heteroscedasticity and autocorrelation robust (HAC) using
  ↵0 lags and without small sample correction
Null Hypothesis: CPIAUCSL granger-causes INDPRO
```

Granger Causality

number of lags (no zero) 1

```
ssr based F test:          F=0.3960 , p=0.5294 , df_denom=734, df_num=1
ssr based chi2 test:      chi2=0.3976 , p=0.5283 , df=1
likelihood ratio test: chi2=0.3975 , p=0.5284 , df=1
parameter F test:         F=0.3960 , p=0.5294 , df_denom=734, df_num=1
```

Granger Causality

number of lags (no zero) 2

```
ssr based F test:          F=7.2923 , p=0.0007 , df_denom=731, df_num=2
ssr based chi2 test:      chi2=14.6844 , p=0.0006 , df=2
likelihood ratio test: chi2=14.5398 , p=0.0007 , df=2
parameter F test:         F=7.2923 , p=0.0007 , df_denom=731, df_num=2
```

Granger Causality

number of lags (no zero) 3

```
ssr based F test:          F=5.3865 , p=0.0011 , df_denom=728, df_num=3
ssr based chi2 test:      chi2=16.3148 , p=0.0010 , df=3
likelihood ratio test: chi2=16.1364 , p=0.0011 , df=3
parameter F test:         F=5.3865 , p=0.0011 , df_denom=728, df_num=3
```

#### OLS Regression Results

Dep. Variable:	INDPRO	R-squared:	0.090
Model:	OLS	Adj. R-squared:	0.085
Method:	Least Squares	F-statistic:	2.709
Date:	Thu, 31 Aug 2023	Prob (F-statistic):	0.0293
Time:	06:59:01	Log-Likelihood:	2404.7
No. Observations:	735	AIC:	-4799.
Df Residuals:	730	BIC:	-4776.
Df Model:	4		
Covariance Type:	HAC		
coef	std err	t	P> t
Intercept	0.0020	0.001	1.998
INDPRO.shift(1)	0.2542	0.143	1.776
CPIAUCSL.shift(1)	0.4172	0.214	1.945
CPIAUCSL.shift(2)	-0.4424	0.203	-2.181
CPIAUCSL.shift(3)	-0.1554	0.140	-1.113
Omnibus:	720.568	Durbin-Watson:	1.981
Prob(Omnibus):	0.000	Jarque-Bera (JB):	124405.818
Skew:	-3.897	Prob(JB):	0.00
Kurtosis:	66.257	Cond. No.	563.

(continues on next page)

(continued from previous page)

=====

Notes:

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

## 11.7 Vector Autoregression

- Impulse Response Function

```
# Vector Autoregression: Impulse Response Function
model = VAR(data)
results = model.fit(3)
print(results.summary())
irf = results.irf(12)
#irf.plot(orth=False)
irf.plot_cum_effects(orth=False, figsize=(10, 6))
plt.savefig(imgdir / 'impulse.jpg')
```

Summary of Regression Results

=====

Model: VAR  
 Method: OLS  
 Date: Thu, 31, Aug, 2023  
 Time: 06:59:01

No. of Equations: 2.00000 BIC: -21.3075  
 Nobs: 735.000 HQIC: -21.3614  
 Log likelihood: 5790.88 FPE: 5.10743e-10  
 AIC: -21.3952 Det(Omega\_mle): 5.01152e-10

Results for equation INDPRO

=====

	coefficient	std. error	t-stat	prob
const	0.001959	0.000530	3.697	0.000
L1.INDPRO	0.268216	0.037126	7.225	0.000
L1.CPIAUCSL	0.410245	0.138383	2.965	0.003
L2.INDPRO	-0.060509	0.038070	-1.589	0.112
L2.CPIAUCSL	-0.426554	0.157572	-2.707	0.007
L3.INDPRO	0.072363	0.036677	1.973	0.048
L3.CPIAUCSL	-0.168769	0.139684	-1.208	0.227

Results for equation CPIAUCSL

=====

	coefficient	std. error	t-stat	prob
const	0.000893	0.000141	6.332	0.000
L1.INDPRO	0.004405	0.009885	0.446	0.656
L1.CPIAUCSL	0.540335	0.036845	14.665	0.000
L2.INDPRO	-0.000168	0.010136	-0.017	0.987
L2.CPIAUCSL	0.028925	0.041954	0.689	0.491

(continues on next page)

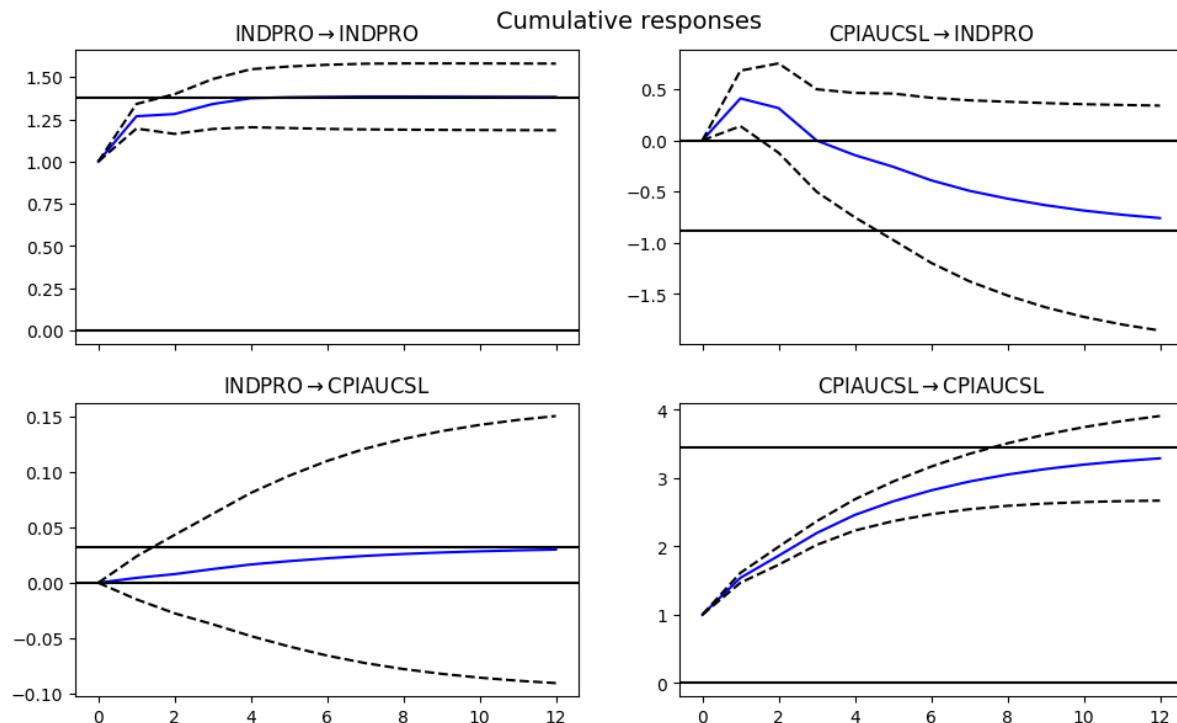
(continued from previous page)

L3.INDPRO	0.002588	0.009765	0.265	0.791
L3.CPIAUCSL	0.142322	0.037191	3.827	0.000
=====				

Correlation matrix of residuals

	INDPRO	CPIAUCSL
INDPRO	1.000000	0.098913
CPIAUCSL	0.098913	1.000000

```
/home/terence/env3.11/lib/python3.11/site-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
```





## APPROXIMATE FACTOR MODELS

### UNDER CONSTRUCTION

- PCA, EM
- Approximate factors and selection: Bai and Ng (2002), McCracken and Ng (2016)

```
import numpy as np
import pandas as pd
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
from finds.readers.alfred import Alfred, fred_md, fred_qd
from finds.econs import factors_em, mrsq, select_bai_ng, integration_order
from finds.filters import remove_outliers, is_outlier
from finds.unstructured.store import Store
from finds.misc.show import Show
from secret import credentials, paths
show = Show(ndigits=4, latex=None)
VERBOSE = 0
# %matplotlib qt
```

```
imgdir = paths['images'] / 'ts'
alf = Alfred(api_key=credentials['fred']['api_key'], verbose=-1)
store = Store(paths['scratch'])
```

### 12.1 FRED-MD

- Transformation Codes
- Stationarity, Integration Order

```
## Transformation Codes, and Stationarity
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)

print(f"Number of series by suggested tcode transformations ({md_date}):")
tcodes = pd.concat([Series(alf.tcode[i], name=i).to_frame().T
                    for i in range(1, 8)], axis=0).fillna(False)
tcodes = tcodes.join(qd_codes['transform'].value_counts().rename('fred-qd')) \
    .join(md_codes['transform'].value_counts().rename('fred-md')) \
```

(continues on next page)

(continued from previous page)

```

    .fillna(0) \
    .astype({'fred-qd': int, 'fred-md': int}) \
    .rename_axis(index='tcode')
show(tcodes)

```

quarterly/current.csv  
monthly/current.csv  
Number of series by suggested tcode transformations (20230630) :

tcode	diff	log	pct_change	fred-qd	fred-md
1	0	0	False	22	11
2	1	0	False	32	19
3	2	0	False	0	0
4	0	1	False	0	10
5	1	1	False	141	53
6	2	1	False	50	33
7	1	0	True	1	1

```

## Verify integration order
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():
        if tcode in range(1, 8):
            s = np.log(df[series_id]) if tcode in [4, 5, 6] else df[series_id]
            order = integration_order(s.dropna(), pvalue=0.05)
            expected_order = 2 if tcode == 7 else ((tcode - 1) % 3)
            stationary[series_id] = {'tcode': tcode,
                                      'I(p)': order,
                                      'different': order - expected_order,
                                      'title': alf.header(series_id)}
    #       print(series_id, tcode, expected_order, order)
    stationary = DataFrame.from_dict(stationary, orient='index')
    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]

```

```

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)
show(results,
      caption='FRED-MD order of integration, transformations and frequency')

```

Series by tcode, transformations and estimated order of integration:

```

diff  log  ...  I(1)  I(2)
FRED-MD order of integration, transformations a...
1      0   0  ...   0   0
2      1   0  ...  16   0
3      2   0  ...   0   0
4      0   1  ...   4   0
5      1   1  ...  39   0
6      2   1  ...  29   4
7      1   0  ...   1   0

[7 rows x 11 columns]

```

```

show(stationary[stationary['different'] > 0],
     max_colwidth=60,
     caption='FRED-MD series with unit root after transformations')

```

	title	tcode	...
FRED-MD series with unit root after transformat...			
NWPI	*** NWPI ***	1	...
TLBSNNBBDI	*** TLBSNNBBDI ***	1	...
TLBSNNCBBDI	*** TLBSNNCBBDI ***	1	...
HWI		1	...
Wanted Index for United States			Help
GFDEBTN		2	...
Federal Debt: Total Public Debt			
S&P div yield		2	...
Composite Common Stock: Dividend Yield			S&P's
CES2000000008		5	...
Earnings of Production and Nons...			Average Hourly
TLBSHNO		5	...
Nonprofit Organizations; Total ...			Households and
OPHMFG		5	...
Sector: Labor Productivity (Outp...			Manufacturing
SPCS20RSA		5	...
20-City Composite Home Price ...			S&P/Case-Shiller
SLCE		5	...
Consumption Expenditures & Gro...			State and Local

[11 rows x 4 columns]

## 12.2 Approximate factor model

- Bai and Ng, McCracken and Ng

```

# Verify BaiNg implementation on published FRED-MD and FRED-QD reports
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']]],

```

(continues on next page)

(continued from previous page)

```

        ['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 in X to NaN
X = data.loc[sample]
X = remove_outliers(X)

# compute factors EM and auto select number of components, r
Z = factors_em(X, p=2, verbose=VERBOSE)
r = select_bai_ng(Z, p=2)

# show marginal R2's of series to each component
mR2 = mrsq(Z, r).to_numpy()
show(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']),
caption=f"Fred-{freq[0].upper()} ID {freq} series:")

for k in range(r):
    args = np.argsort(-mR2[:, k])
    show(DataFrame.from_dict({mR2[arg, k].round(4):
                                {'series': cols[arg],
                                 'description': alf.header(cols[arg])},
                                for arg in args[:10]},
                                orient='index'),
caption=f"Factor: {1+k} Variance Explained={np.mean(mR2[:, k]):.4f}")

```

quarterly/2020-04.csv  
monthly/2015-05.csv

```

## Sanity check Extract factors: SVD == PCA
# pipe.fit through 20141231, pipe.transform through 20201231

df, t = fred_md()  #202104 # 201505
transforms = t['transform']
sample_date = 20141231
data = []
for col in df.columns:
    data.append(alf.transform(df[col], tcode=transforms[col], freq='m'))
data = pd.concat(data, axis=1).iloc[2:]
cols = list(data.columns)

```

(continues on next page)

(continued from previous page)

```
sample = data.index[((np.count_nonzero(np.isnan(data), axis=1)==0) |
                     (data.index <= sample_date)) & (data.index >= 19600301)]
train_sample = sample[sample <= sample_date]
test_sample = sample[sample <= 20191231]
```

monthly/current.csv

```
# replace missing and outliers with PCA EM and fixed number of components r=8
r = 8
X = data.loc[train_sample] # X = np.array(data.loc[train_sample])
X[is_outlier(X, method='iq10')] = np.nan
x = factors_em(X, kmax=r, p=0, verbose=0).to_numpy()
```

```
# Extract factors with SVD
y = ((x-x.mean(axis=0).reshape(1,-1))/x.std(axis=0, ddof=0).reshape(1,-1))
u, s, vT = np.linalg.svd(y, full_matrices=False)
#factors = DataFrame(u[:, :r], columns=np.arange(1, 1+r),
#                     index=pd.DatetimeIndex(train_sample.astype(str), freq='M'))
# Series(s[:r]**2 / np.sum(s**2), index=np.arange(1, r+1), name='R2').to_frame().T
```

	1	2	3	...	6	7	8
R2	0.151032	0.071875	0.068792	...	0.033761	0.031713	0.027361

[1 rows x 8 columns]

```
# Equivalent to sklearn PCA
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.decomposition import PCA
pipe = Pipeline([('scaler', StandardScaler()), ('pca', PCA(r))])
pipe.fit(x) # fit model on training data
X = data.loc[sample] # to transform on full sample data
X = factors_em(X, kmax=8, p=0, verbose=0) # replace missing (not outlier)
factors = DataFrame(StandardScaler().fit_transform(pipe.transform(X)),
                     index=pd.DatetimeIndex(sample.astype(str), freq='infer'),
                     columns=np.arange(1, 1+r))

# store approximate factors in local folder
store['approximate'] = dict(factors=factors)
```

```
/home/terence/env3.11/lib/python3.11/site-packages/sklearn/base.py:432:_
  UserWarning: X has feature names, but StandardScaler was fitted without feature_
  names
  warnings.warn(
```

```
Series(pipe.named_steps['pca'].explained_variance_ratio_,
       index=np.arange(1,r+1), name='R2').to_frame().T # sanity check
```

	1	2	3	...	6	7	8
R2	0.151032	0.071875	0.068792	...	0.033761	0.031713	0.027361

(continues on next page)

(continued from previous page)

[1 rows x 8 columns]

```
## Retrieve recession periods from FRED
vspan = alf.date_spans('USREC')
DataFrame(vspan, columns=['Start', 'End'])
```

	Start	End
0	1854-12-31	1854-12-31
1	1857-06-30	1858-12-31
2	1860-10-31	1861-06-30
3	1865-04-30	1867-12-31
4	1869-06-30	1870-12-31
5	1873-10-31	1879-03-31
6	1882-03-31	1885-05-31
7	1887-03-31	1888-04-30
8	1890-07-31	1891-05-31
9	1893-01-31	1894-06-30
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

```
## Plot extracted factors
fig = plt.figure(figsize=(9, 10), num=1, clear=True)
for col in factors.columns:
    ax = fig.add_subplot(4, 2, col)
    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 vspan:
        if b >= min(factors.index):
            ax.axvspan(max(a, min(factors.index)), min(b, max(factors.index)),
                        alpha=0.3, color='grey')
```

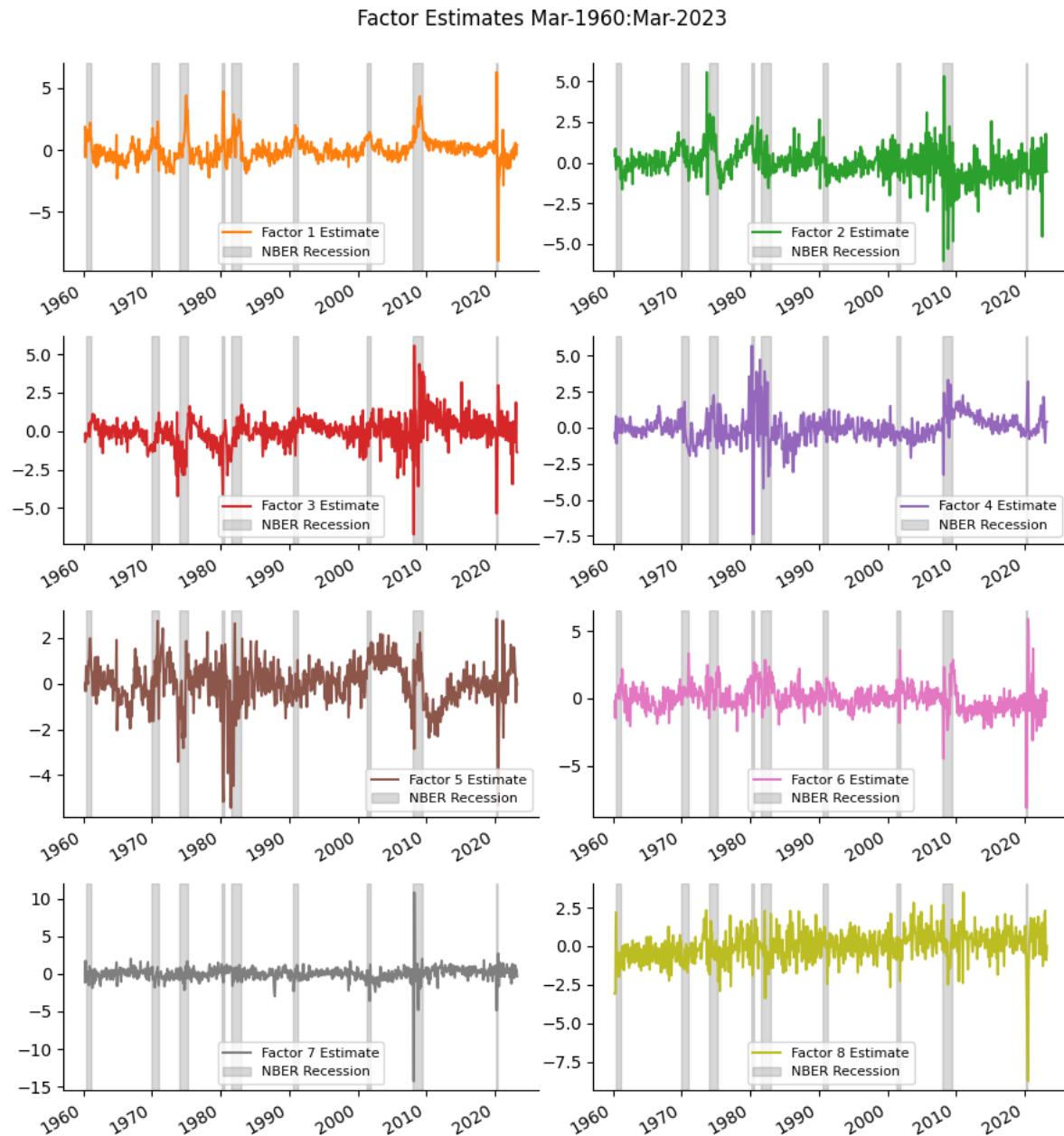
(continues on next page)

(continued from previous page)

```

ax.legend([f"Factor {col} Estimate", 'NBER Recession'], fontsize=8)
ax.spines["top"].set_visible(False)
ax.spines["right"].set_visible(False)
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.suptitle(f"Factor Estimates {factors.index[0]}:{b}-{Y}:"
            f"{factors.index[-1]}:{b}-{Y}", fontsize=12)
plt.savefig(imgdir / 'approximate.jpg')

```





## STATE SPACE MODELS

### UNDER CONSTRUCTION

- Gaussian Mixture Model
- Hidden Markov Models
  - Viterbi algorithm
- Kalman Filter and Dynamic Factor Models

```
import re
import time
import random
from datetime import datetime
import numpy as np
import pandas as pd
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
import statsmodels.api as sm
from hmmlearn import hmm
from sklearn.mixture import GaussianMixture
from sklearn.preprocessing import StandardScaler
from tqdm import tqdm
from typing import List, Dict
from finds.readers import fred_md, fred_qd, Alfred
from finds.misc import Show
from secret import paths, credentials

show = Show(ndigits=4, latex=None)
VERBOSE = 0
# %matplotlib qt
imgdir = paths['images'] / 'states'
```

```
# Load and pre-process time series from FRED
alf = Alfred(api_key=credentials['fred']['api_key'])
vspan = alf.date_spans('USREC') # to indicate recession periods in the plots
```

```
today = int(datetime.today().strftime('%Y%m%d'))
beg = 19600301
end = 20221231 # should be current less 4-6 months?
split = 20220601 # forecast last two quarters
```

### FRED-MD

- replace outliers

- apply tcode transformations
- train/test split: train up to MINUS two quarters, forecast through current

```

#
#
# TODO: use EM to replace outliers in data
#
#
# Retrieve FRED-MD series and apply tcode transformations
df, t = fred_md()           # from vintage April 2020?
data = []
for col in df.columns:
    data.append(alf.transform(df[col], tcode=t['transform'][col], freq='m'))
mdf = pd.concat(data, axis=1).iloc[2:]
mdata = mdf[(mdf.index >= beg) & (mdf.index <= end)].dropna(axis=1)
mdata = (mdata - mdata.mean(axis=0)) / mdata.std(axis=0, ddof=0)
mdata.index = pd.DatetimeIndex(mdata.index.astype(str), freq='m')
mdata

```

monthly/current.csv

	RPI	W875RX1	DPCERA3M086SBEA	CMRMTSPL	RETAIL	INDPRO	
1960-03-31	-0.055906	-0.239279		1.361862	-2.436357	-0.373310	-1.128282 \
1960-04-30	0.070437	0.197310		1.518351	0.617775	1.351158	-1.018832
1960-05-31	-0.014758	0.007897		-2.731964	-2.719109	-1.263778	-0.315496
1960-06-30	-0.149637	-0.427661		-0.336036	0.517670	-0.390060	-1.503229
1960-07-31	-0.062582	-0.079816		-0.067982	-0.728275	-0.947820	-0.556224
...	...	...		...	...	...	...
2022-08-31	0.031812	0.268079		0.259747	1.120513	0.145295	-0.095970
2022-09-30	-0.105971	-0.000502		-0.002497	-0.128454	-0.313545	0.095777
2022-10-31	-0.107340	-0.775597		0.005934	-0.260730	0.360985	-0.314157
2022-11-30	-0.214863	-0.510423		-0.803318	-0.939377	-1.145908	-0.538102
2022-12-31	-0.227146	-0.418519		-0.540658	1.027194	-0.762278	-1.797629
	IPFPNSS	IPFINAL	IPCONGD	IPDCONGD	...	PCEPI	
1960-03-31	-0.544828	-0.288448	-0.030594	-0.670564	...	-0.251034 \	
1960-04-30	0.046300	-0.071603	0.510438	-0.099990	...	1.588834	
1960-05-31	0.397676	0.466734	0.398859	0.223473	...	-1.342564	
1960-06-30	-1.370566	-1.260830	-0.785716	-0.387215	...	-0.311868	
1960-07-31	-0.785820	-0.725112	-1.008340	-1.242847	...	0.682163	
...	...	...	...	...	...	...	
2022-08-31	0.117463	0.254708	0.145430	-0.254268	...	1.841999	
2022-09-30	-0.137272	-0.231848	-0.406570	-0.193276	...	0.386374	
2022-10-31	0.060215	0.222922	0.347623	0.342962	...	0.386478	
2022-11-30	-0.534353	-0.557796	-0.308918	-0.744716	...	-1.270842	
2022-12-31	-1.433904	-0.978260	-0.728798	-0.377401	...	0.156942	
	DDURRG3M086SBEA	DNDGRG3M086SBEA	DSERRG3M086SBEA	CES0600000008			
1960-03-31	-1.387819	0.286975	-0.688081	-0.005209 \			
1960-04-30	1.257955	0.711232	0.889271	-2.259533			
1960-05-31	-0.394294	-1.073753	0.360705	2.259270			
1960-06-30	-0.801636	0.036861	-0.425122	-1.129832			
1960-07-31	0.869009	0.293779	0.307193	1.124537			
...	...	...	...	...	...	...	
2022-08-31	2.128771	-0.404350	3.054382	-0.364202			
2022-09-30	-0.105260	0.628192	-0.070539	0.354791			

(continues on next page)

(continued from previous page)

2022-10-31	-2.555433	1.999584	-0.833222	0.083418	
2022-11-30	-0.531799	-1.413324	-0.351634	0.258909	
2022-12-31	1.050942	-0.980203	1.076286	-0.272668	
	CES2000000008	CES3000000008	DTCOLNVHFN	DTCTHFNM	INVEST
1960-03-31	3.346166	-1.006546	0.145331	0.048471	0.322440
1960-04-30	-7.559300	0.000557	0.363396	0.275403	2.250783
1960-05-31	4.621913	0.000557	-0.202301	-0.103697	0.351217
1960-06-30	-1.268962	0.000557	0.214617	0.374620	-0.935720
1960-07-31	0.838260	0.000557	-0.457746	-0.165399	3.001718
...	...	...	...	...	...
2022-08-31	0.306571	-0.772418	0.051969	0.071119	-0.431431
2022-09-30	-0.003621	0.682301	0.217847	-0.008291	-0.680035
2022-10-31	-0.172316	0.081215	0.013922	0.108611	-0.412855
2022-11-30	-0.069131	0.837302	0.067908	-0.003047	0.892882
2022-12-31	0.333477	-1.268091	-0.111931	-0.058777	0.819264

[754 rows x 120 columns]

## 13.1 Hidden Markov Model

```

def hmm_summary(markov: hmm.GaussianHMM, X: DataFrame,
                 lengths: List[int], matrix: bool = False) -> Dict:
    """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 = markov.n_features + {"diag": m * n, # parms in mean and cov matrix
                            "full": m * n * (n-1) / 2.0,
                            "tied": n * (n-1) / 2.0,
                            "spherical": m}[markov.covariance_type]
    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:') \
            .reset_index()
        matrix['Stationary'] = markov.get_stationary_distribution()
    
```

(continues on next page)

(continued from previous page)

```
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(1,16):
        markov = hmm.GaussianHMM(n_components=n_components,
                                  covariance_type=covariance_type,
                                  verbose=False,
                                  tol=1e-6,
                                  random_state=42,
                                  n_iter=100)\.
            .fit(mdata.values, [len(mdata)])
        result = hmm_summary(markov, mdata, [len(mdata)])
        #print(n_components, Series(results, name=covariance_type).to_frame().T)
        result.update({'covariance_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: -8356.536935223692 is not greater than -8356.
↳ 536922138932. Delta is -1.3084760212223046e-05
Model is not converging. Current: -20040.65225614991 is not greater than -20040.
↳ 65225604072. Delta is -1.0919029591605067e-07
Fitting a model with 96108 free scalar parameters with only 90480 data points will
↳ result in a degenerate solution.
Fitting a model with 103515 free scalar parameters with only 90480 data points
↳ will result in a degenerate solution.
Fitting a model with 110924 free scalar parameters with only 90480 data points
↳ will result in a degenerate solution.
Model is not converging. Current: -98907.84473039334 is not greater than -98907.
↳ 84472817722. Delta is -2.216111170127988e-06
Model is not converging. Current: -96262.11285136703 is not greater than -96262.
↳ 11280305496. Delta is -4.831206751987338e-05
Model is not converging. Current: -91059.2537900895 is not greater than -91059.
↳ 25375594033. Delta is -3.414916864130646e-05
Model is not converging. Current: -90807.24961393392 is not greater than -90807.
↳ 24959916719. Delta is -1.4766730600968003e-05
Model is not converging. Current: -89268.3968799929 is not greater than -89268.
↳ 39576843736. Delta is -0.0011115555389551446
Model is not converging. Current: -48070.79235763028 is not greater than -48070.
↳ 7923533622. Delta is -4.268076736479998e-06
Model is not converging. Current: -113338.9446084779 is not greater than -113338.
↳ 94460542541. Delta is -3.052497049793601e-06
Model is not converging. Current: -112081.42704057258 is not greater than -112081.
↳ 42703677774. Delta is -3.7948484532535076e-06
Model is not converging. Current: -108461.4527899802 is not greater than -108461.
↳ 45278697292. Delta is -3.0072842491790652e-06
Model is not converging. Current: -107122.20983928096 is not greater than -107122.
↳ 20983800037. Delta is -1.2805830920115113e-06
Model is not converging. Current: -102844.95197449482 is not greater than -102844.
↳ 95197112675. Delta is -3.3680698834359646e-06
```

```
## Find best bic's
best_bic = []
for covariance_type in ["full", "diag", "tied", "spherical"]:
    result = results[results['covariance_type'] == covariance_type]
    argmin = np.argmin(result['bic'])
    best_bic.append(result.iloc[argmin])
best_bic = pd.concat(best_bic, axis=0)
show(best_bic, caption="HMM best bic by covariance type:")
```

	aic	bic	parameters
HMM best bic by covariance type:			
0	114858.0027	148438.3513	7260 \
19	205978.9381	222741.36	3624
30	114858.0027	148438.3513	7260
59	210187.904	220590.4114	2249
	NLL	covariance_type	n_components
HMM best bic by covariance type:			
0	50169.0013	full	1
19	99365.469	diag	5
30	50169.0013	tied	1
59	102844.952	spherical	15

```
## display estimated transition and stationary distributions of best_bic
n_components = best_bic[best_bic['covariance_type'] == 'diag']['n_components']
n_components = int(n_components.iloc[0])
markov = hmm.GaussianHMM(n_components=n_components,
                          covariance_type='diag',
                          verbose=False,
                          tol=1e-6,
                          random_state=42,
                          n_iter=100) \
    .fit(mdata.values, [len(mdata)])
pred = DataFrame(markov.predict(mdata), columns=['state'], index=mdata.index)
matrix = hmm_summary(markov, mdata, [len(mdata)], matrix=True)['matrix']
show(matrix, caption="HMM stationary and transition probabilities")
```

Transition Matrix:	0	1	2	3
HMM stationary and transition probabilities				
0	0.8779	0.0355	0.0353	0.0512 \
1	0.0450	0.9053	0.0073	0.0424
2	0.3431	0.0577	0.5137	0.0855
3	0.1052	0.0980	0.0227	0.7664
4	0.0000	0.0000	1.0000	0.0000

Transition Matrix:	4	Stationary
HMM stationary and transition probabilities		
0	0.0000	0.4144
1	0.0000	0.3638
2	0.0000	0.0464
3	0.0076	0.1740
4	0.0000	0.0013

## Plot predicted states

```

def plot_states(modelname: str, labels: np.ndarray, beg: int, end: int,
                 series_ids = ['IPMANSICS', 'SPASTT01USM661N']):
    """helper to plot predicted states"""

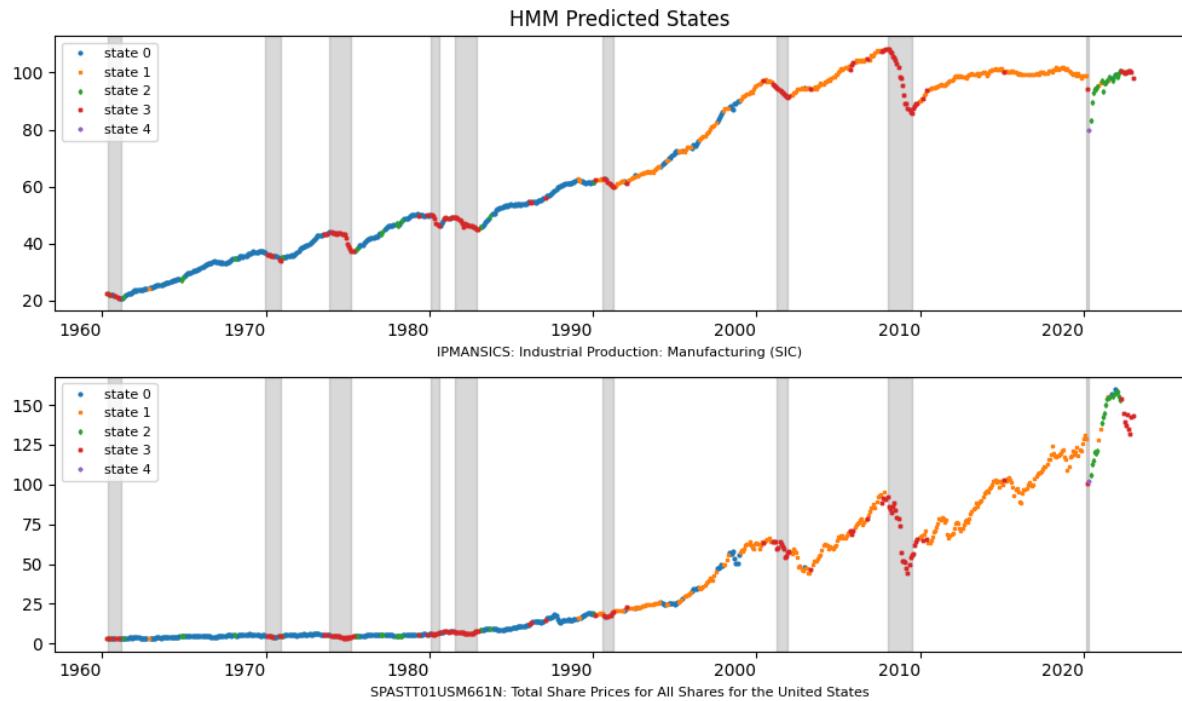
    # n_components markers
    n_components = len(np.unique(labels))
    markers = ["o", "s", "d", "X", "P", "8", "H", "*", "x", "+"][:n_components]

    fig, axes = plt.subplots(len(series_ids),
                           ncols=1,
                           figsize=(10, 3 * len(series_ids)),
                           num=1,
                           clear=True)
    axes[0].set_title(f" {modelname.upper()} Predicted States", {'fontsize':12})

    # plot each selected series, with states colored
    for series_id, ax in zip(series_ids, axes.ravel()):
        df = alf(series_id)
        df.index = pd.DatetimeIndex(df.index.astype(str), freq='infer')
        df = df[(df.index >= beg) & (df.index <= end)]
        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)}",
                      {'fontsize':8})
        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)
    plt.tight_layout()
    plt.savefig(imgdir / f" {modelname.lower()} .jpg")

```

```
plot_states('HMM', pred.values.flatten(), min(pred.index), max(pred.index))
```

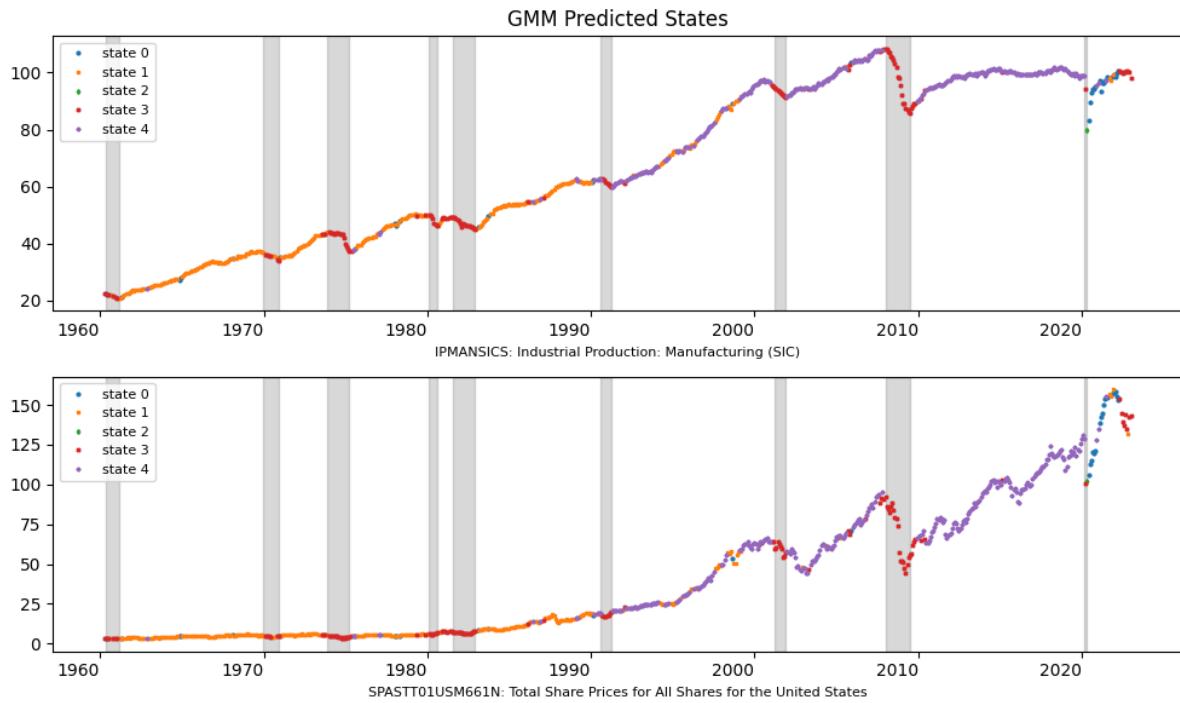


## 13.2 Gaussian Mixture Model

```

gmm = GaussianMixture(n_components=n_components,
                      covariance_type='diag') \
      .fit(mdata)
labels = gmm.predict(mdata)
plot_states('GMM', labels, min(mdata.index), max(mdata.index))

```



```
## Compare persistance of HMM and GMM
dist = DataFrame({
    'Hidden Markov': (sorted(matrix.iloc[:, -1])
                      + [np.mean(pred[:, -1].values == pred[1:,].values)]),
    'Gaussian Mixture': (sorted(Series(labels).value_counts().sort_index()
                                 / len(labels))
                          + [np.mean(labels[:, -1] == labels[1:,])]),
    index=[f'Stationary probability of state {n_components-s-1}'
           for s in range(n_components)]
           + ['Average persistance of states'])
})
show(dist, caption="Compare HMM with GMM:")
```

	Hidden Markov	Gaussian Mixture
Compare HMM with GMM:		
Stationary probability of state 4	0.0013	0.0013
Stationary probability of state 3	0.0464	0.0451
Stationary probability of state 2	0.1740	0.1658
Stationary probability of state 1	0.3638	0.3660
Stationary probability of state 0	0.4144	0.4218
Average persistance of states	0.8526	0.8181

```
## Retrieve FRED-MD series and apply tcode transformations
"""
df, t = fred_md(202004)          # from vintage April 2020?
data = []
for col in df.columns:
    data.append(alf.transform(df[col], tcode=t['transform'][col], freq='m'))
mdf = pd.concat(data, axis=1).iloc[2:]
mdata = mdf[(mdf.index >= beg) & (mdf.index <= end)].dropna(axis=1)
mdata = (mdata - mdata.mean(axis=0)) / mdata.std(axis=0, ddof=0)
mdata.index = pd.DatetimeIndex(mdata.index.astype(str), freq='m')
```

(continues on next page)

(continued from previous page)

'''

```

"\ndf, t = fred_md(202004)      # from vintage April 2020?\nfor col in df.columns:\n    data.append(alf.transform(df[col], tcode=t['transform'][col],\n        freq='m'))\nmdf = pd.concat(data, axis=1).iloc[2:]\nmdata = mdf[(mdf.index >= beg) & (mdf.index <= end)].dropna(axis=1)\nmdata = (mdata - mdata.mean(axis=0)) / mdata.std(axis=0, ddof=0)\nmdata.index = pd.DatetimeIndex(mdata.index.\n        astype(str), freq='m')\n"

```

### 13.3 Dynamic Factor Models

```

dynamic_factors = dict()
seq_len = 16
for i in [1, 2, 3, 4]:
    mod = sm.tsa.DynamicFactorMQ(endog=mdata,
        factors=1,                      # num factor blocks
        factor_multiplicities=i,        # num factors in block
        factor_orders=2,                # order of factor VAR
        idiosyncratic_ar1=False)        # False=white noise
    fitted = mod.fit_em(disp=20,
        maxiter=200,
        full_output=True)
    dynamic_factors[i] = DataFrame(fitted.factors.filtered.iloc[seq_len+1:])
    dynamic_factors[i].columns = list(range(len(dynamic_factors[i].columns)))
#print(fitted.summary(0))

```

```

EM start iterations, llf=-1.166e+05
EM converged at iteration 14, llf=-1.1573e+05, convergence criterion=9.3327e-07 <-
    tolerance=1e-06
EM start iterations, llf=-1.1229e+05
EM iteration 20, llf=-1.095e+05, convergence criterion=1.7942e-05
EM iteration 40, llf=-1.0947e+05, convergence criterion=4.786e-06
EM converged at iteration 57, llf=-1.0947e+05, convergence criterion=9.1845e-07 <-
    tolerance=1e-06
EM start iterations, llf=-1.0663e+05
EM iteration 20, llf=-1.044e+05, convergence criterion=1.3871e-05
EM iteration 40, llf=-1.0438e+05, convergence criterion=3.7032e-06
EM converged at iteration 57, llf=-1.0438e+05, convergence criterion=9.9877e-07 <-
    tolerance=1e-06
EM start iterations, llf=-1.0269e+05
EM iteration 20, llf=-99741, convergence criterion=1.3581e-05
EM iteration 40, llf=-99727, convergence criterion=3.2774e-06
EM converged at iteration 57, llf=-99724, convergence criterion=9.3062e-07 <-
    tolerance=1e-06

```

```

### Plot dynamic factors
fig, axes = plt.subplots(len(dynamic_factors), 1, figsize=(9, 10), num=1, clear=True)
for dynamic_factor, ax in zip(dynamic_factors.values(), axes):
    dynamic_factor.plot(ax=ax, style='--', legend=False)
    for a, b in vspans:
        if a >= min(dynamic_factor.index):

```

(continues on next page)

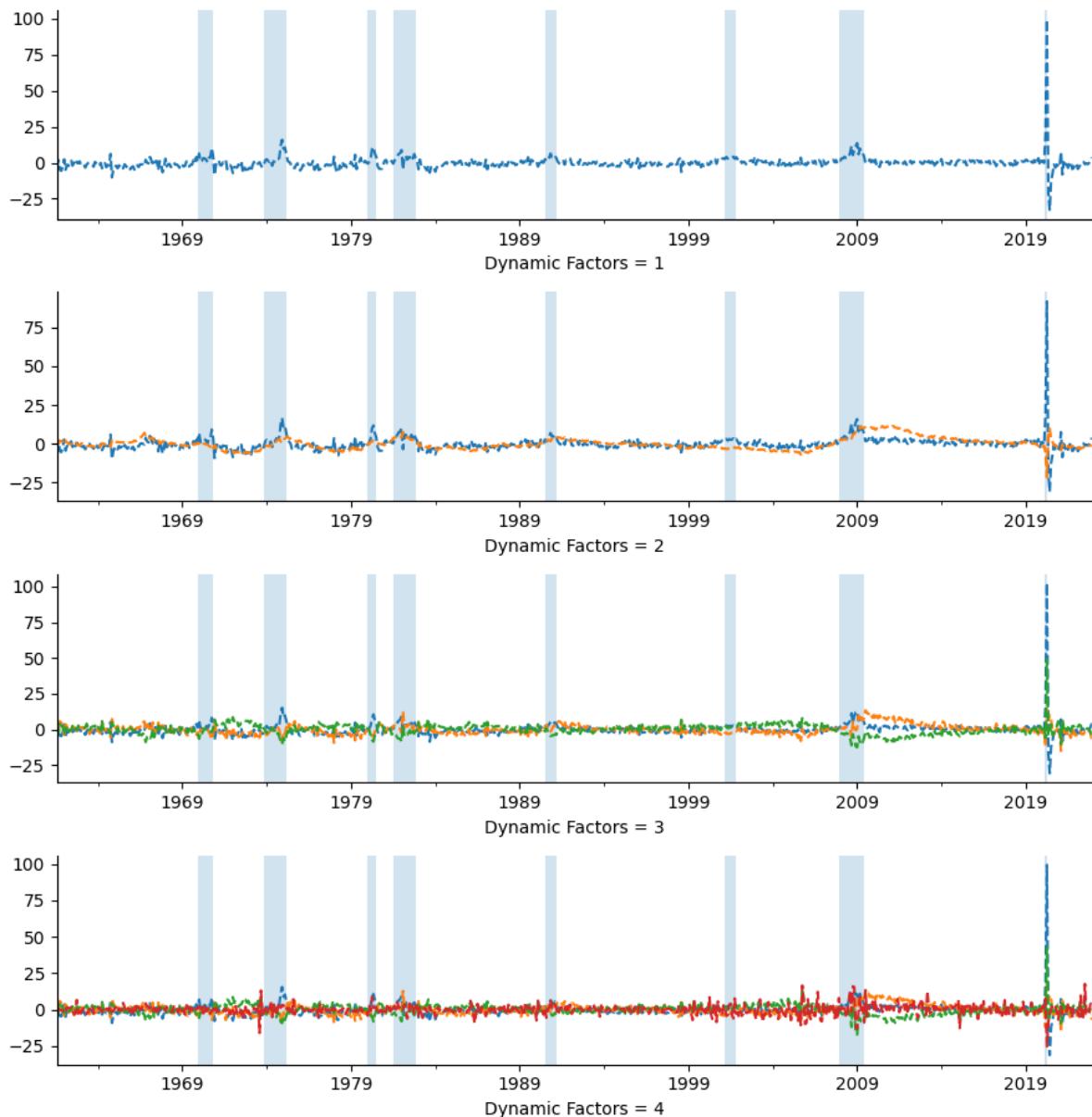
(continued from previous page)

```

ax.axvspan(a, min(b, max(dynamic_factor.index)), alpha=0.2)
ax.spines["top"].set_visible(False)
ax.spines["right"].set_visible(False)
ax.set_xlabel(f"Dynamic Factors = {len(dynamic_factor.columns)}")
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.suptitle(f"Fitted Dynamic Factors ", fontsize=12)
plt.savefig(imgdir / 'dynamic.jpg')

```

Fitted Dynamic Factors



---

CHAPTER  
**FOURTEEN**

---

## CONDITIONAL VOLATILITY

### UNDER CONSTRUCTION

- Value at Risk, Expected Shortfall, GARCH, EWMA
- Cryptos: bitcoin, etherium
- Historical VaR, ES
- pof, LR: Theoretical ES, VaR - normal, t(6)
- ARCH, GARCH, EWMA
- Copulas
- Power Laws, EVT
- Bootstrapping with circular block

```
import numpy as np
from scipy.stats import chi2, norm, t, jarque_bera, kurtosis, skew
import pandas as pd
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
import seaborn as sns
from finds.readers import Alfred
from finds.finance import kupiecLR, pof, halflife, RiskMeasure
from finds.misc import Show
from secret import credentials, paths
show = Show(ndigits=4, latex=None)
pd.set_option('display.max_rows', None)
VERBOSE = 0
#%matplotlib qt
imgdir = paths['images'] / 'ts'
```

```
alf = Alfred(api_key=credentials['fred']['api_key'], verbose=VERBOSE)
```

```
## Histogram, VaR and ES of Cryptocurrencies
labels = ['CBBTCUSD', 'CBETHUSD', 'CBBCHUSD', 'CBLTCUSD']
alpha = 0.95    # VaR parameter
out = {}
fig, axes = plt.subplots(2, 2, num=1, clear=True, figsize=(10, 5))
for label, ax in zip(labels, list(axes.ravel())):
    rets = alf(label, log=1, diff=1).dropna()
    sigma = rets.std()
    risk_measure = RiskMeasure(rets, alpha=0.95)
```

(continues on next page)

(continued from previous page)

```

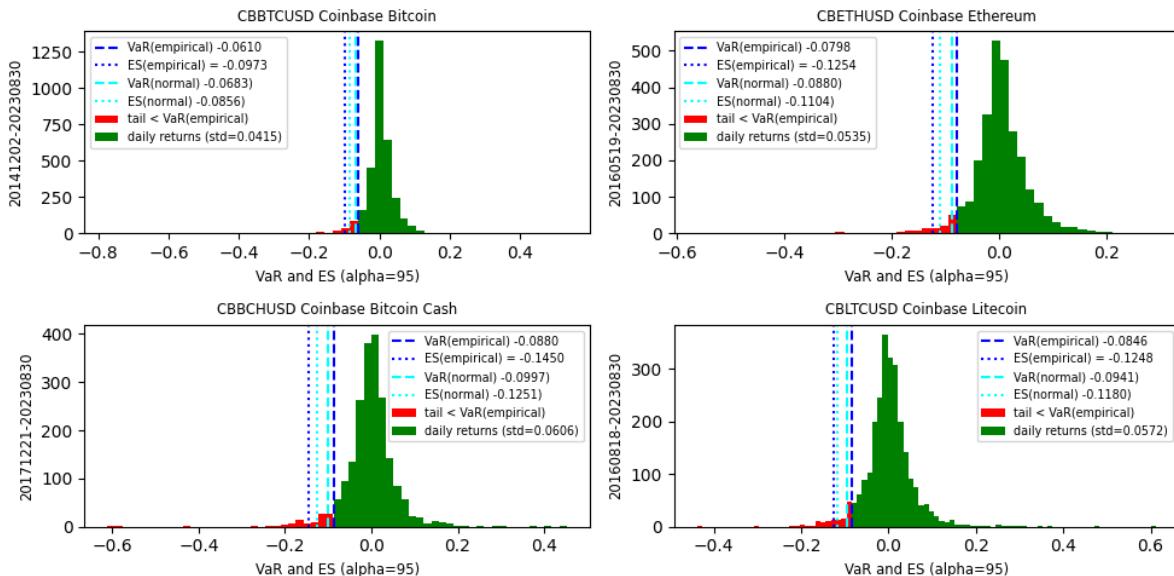
var = risk_measure.value_at_risk()
es = risk_measure.expected_shortfall()
var_normal = risk_measure.value_at_risk(normal=True)
es_normal = risk_measure.expected_shortfall(normal=True)

n, bins, _ = ax.hist(rets[rets < var], color='red', bins=30)
stepsize = (bins[-1] - bins[0]) / (len(bins) - 1)
bins = np.arange(var, max(rets) + stepsize, stepsize)
ax.hist(rets[rets >= var], color='green', bins=bins)

ax.axvline(var, color='blue', ls='--')
ax.axvline(es, color='blue', ls=':')
ax.axvline(var_normal, color='cyan', ls='--')
ax.axvline(es_normal, color='cyan', ls=':')
ax.legend([f'VaR(empirical) {var:.4f}', 
           f'ES(empirical) = {es:.4f}', 
           f'VaR(normal) {var_normal:.4f})', 
           f'ES(normal) {es_normal:.4f})', 
           'tail < VaR(empirical)', 
           'daily returns (std={sigma:.4f})'], fontsize='x-small')
ax.set_title(f"{{label}} {{alf.header(label) [:60]}}",
            {'fontsize' : 'small'})
ax.set_ylabel(f'{min(rets.index)}-{max(rets.index)}',
              fontsize='small')
ax.set_xlabel(f'VaR and ES (alpha={alpha*100:.0f})',
              fontsize='small')
out[label] = {'std dev': np.std(rets, ddof=1),
              'skewness': skew(rets),
              'excess kurtosis': kurtosis(rets)-3,
              'jarque-bera pvalue': jarque_bera(rets)[1]}

plt.tight_layout()
plt.savefig(imgdir / 'es.jpg')

```



```

# Non-normal: Jacque-Bera, excess kurtosis, mixtures
show(DataFrame(out).T)

```

	std	dev	skewness	excess	kurtosis	jarque-bera	pvalue
CBBTCUSD	0.0415	0.0415	-1.8691	50.6324	0.0	0.0	0.0
CBETHUSD	0.0535	0.0535	-0.4478	5.3968	0.0	0.0	0.0
CBBCHUSD	0.0606	0.0606	-0.4462	12.0454	0.0	0.0	0.0
CBLTCUSD	0.0572	0.0572	0.7669	9.8600	0.0	0.0	0.0

## 14.1 GARCH

```
## GARCH(1,1) wrapper around rugarch
import rpy2.robj as ro
from rpy2.robj.packages import importr
from finds.misc import PyR
def rugarch(rets: Series, savefig: str = '') -> np.array:
    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(rets.values).ro)
    ro.r['show'](model)

    if savefig:
        for which in [4, 5, 10, 11]:
            ro.r['plot'](model, which=which)
            PyR.savefig(imgdir / f'{savefig}{which}.png')
    return PyR(ro.r['sigma'](model)).values # fitted volatility values
```

```
## Retrieve Bitcoin from FRED and plot GARCH, EWMA and Daily Returns
series_id = 'CBBTCUSD'
X = alf(series_id, log=1, diff=1).dropna()
X.index = pd.DatetimeIndex(X.index.astype(str), freq='infer')
Y = np.square(X)
```

```
lam = 0.97
asset = np.sqrt((Y.ewm(alpha=1 - lam).mean()).rename('EWMA')).to_frame()
asset['EWMA(0.94)'] = np.sqrt((Y.ewm(alpha=1 - 0.94).mean()))
asset['GARCH'] = rugarch(X)
```

```
*-----*
*      GARCH Model Fit      *
*-----*

Conditional Variance Dynamics
-----
GARCH Model      : SGARCH(1,1)
Mean Model       : ARFIMA(0,0,0)
Distribution     : norm

Optimal Parameters
-----
Estimate  Std. Error  t value Pr(>|t|)
```

(continues on next page)

(continued from previous page)

```
mu      0.002039  0.000547  3.725 0.000195
omega   0.000082  0.000011  7.299 0.000000
alpha1   0.194184  0.017569 11.053 0.000000
beta1   0.785250  0.015994 49.096 0.000000
```

Robust Standard Errors:

	Estimate	Std. Error	t value	Pr(> t )
mu	0.002039	0.000604	3.3742	0.000740
omega	0.000082	0.000031	2.6541	0.007952
alpha1	0.194184	0.060987	3.1840	0.001453
beta1	0.785250	0.045039	17.4349	0.000000

LogLikelihood : 6038.942

Information Criteria

---

Akaike	-3.8208
Bayes	-3.8131
Shibata	-3.8208
Hannan-Quinn	-3.8180

Weighted Ljung-Box Test on Standardized Residuals

---

	statistic	p-value
Lag[1]	2.504	0.113554
Lag[2*(p+q)+(p+q)-1][2]	4.927	0.042536
Lag[4*(p+q)+(p+q)-1][5]	10.122	0.008565
d.o.f=0		

H0 : No serial correlation

Weighted Ljung-Box Test on Standardized Squared Residuals

---

	statistic	p-value
Lag[1]	0.04635	0.8295
Lag[2*(p+q)+(p+q)-1][5]	2.19280	0.5736
Lag[4*(p+q)+(p+q)-1][9]	3.00410	0.7583
d.o.f=2		

Weighted ARCH LM Tests

---

	Statistic	Shape	Scale	P-Value
ARCH Lag[3]	1.426	0.500	2.000	0.2324
ARCH Lag[5]	1.686	1.440	1.667	0.5449
ARCH Lag[7]	2.002	2.315	1.543	0.7170

Nyblom stability test

---

Joint Statistic: 1.1778

Individual Statistics:

mu	0.31192
omega	0.32500
alpha1	0.08342
beta1	0.22086

Asymptotic Critical Values (10% 5% 1%)

(continues on next page)

(continued from previous page)

```
Joint Statistic:          1.07 1.24 1.6
Individual Statistic:    0.35 0.47 0.75
```

Sign Bias Test

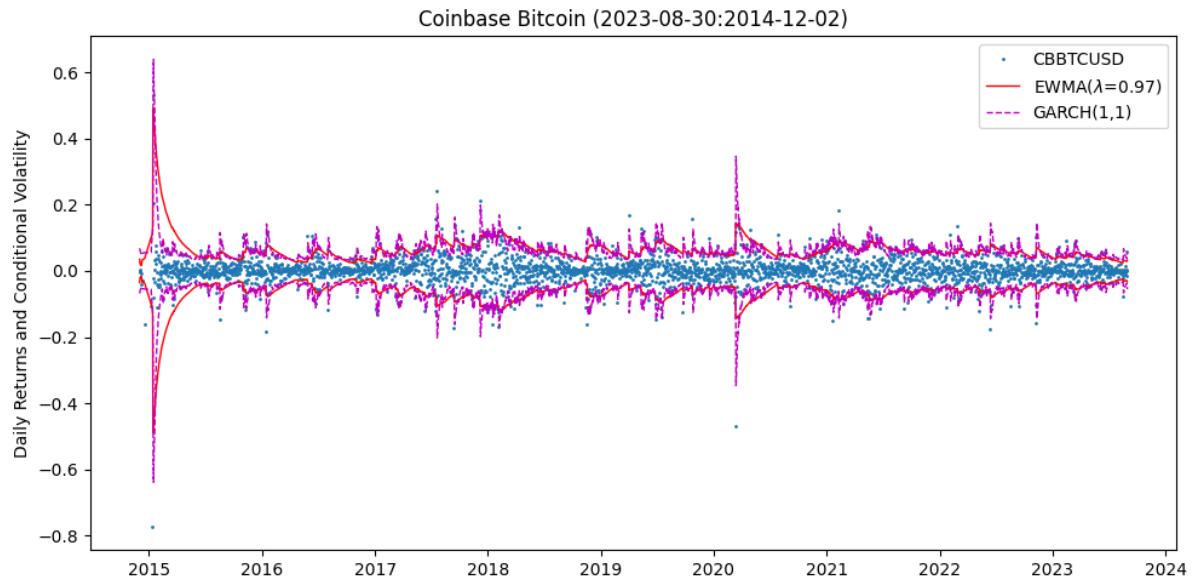
```
-----  
          t-value    prob  sig  
Sign Bias      1.315 0.18856  
Negative Sign Bias  1.346 0.17826  
Positive Sign Bias  1.705 0.08829  *  
Joint Effect     4.721 0.19340
```

Adjusted Pearson Goodness-of-Fit Test:

```
-----  
  group  statistic  p-value(g-1)  
1      20      529.7  3.209e-100  
2      30      537.6  5.292e-95  
3      40      566.4  8.888e-95  
4      50      579.2  1.058e-91
```

Elapsed time : 0.2695932

```
# Conditional Volatility Models EWMA(94), EWMA(97), GARCH(1, 1): Bitcoin
fig, ax = plt.subplots(num=1, clear=True, figsize=(10, 5))
ax.plot(X.shift(-1), ls='', marker='.', markersize=2)
ax.plot(norm.ppf(1 - alpha) * asset['EWMA'], lw=1, ls='-', c='r')
ax.plot(norm.ppf(1 - alpha) * asset['GARCH'], lw=1, ls='--', c='m')
ax.set_title(alf.header(series_id)
            + f" ({max(Y.index).strftime('%Y-%m-%d')})"
            + f" {min(Y.index).strftime('%Y-%m-%d')})")
ax.set_ylabel('Daily Returns and Conditional Volatility')
ax.legend([series_id] + [f"EWMA({lam:.2f})"] + ['GARCH(1,1)'])
ax.plot(-norm.ppf(1 - alpha) * asset['EWMA'], lw=1, ls='-', c='r')
ax.plot(-norm.ppf(1 - alpha) * asset['GARCH'], lw=1, ls='--', c='m')
plt.tight_layout()
plt.savefig(imgdir / 'ewma.jpg')
```



## VALUE AT RISK

```
# Statistical summary of conditional volatility models
y = X.shift(-1)
results = {}
for label, x in zip([series_id, 'EWMA(0.94)', f'EWMA({lam:.2f})','GARCH(1,1)'],
                    [1.0, asset['EWMA(0.94)'], asset['EWMA'], asset['GARCH']]):
    if isinstance(x, (int, float)):
        pof_ = pof((y / np.std(y))[126:], x) # skip first six months
    else:
        pof_ = pof(y[126:], x[126:]) # skip first six months
    results[label] = {'std dev': np.std(y.div(x).dropna()),
                      'skewness': skew(y.div(x).dropna()),
                      'excess kurtosis': kurtosis(y.div(x).dropna()) - 3,
                      f'pof({int(100*alpha)})': pof_['s']/pof_['n'],
                      'pof p-value': pof_['pvalue']}
show(DataFrame.from_dict(results, orient='index'), caption=series_id)
```

	std	dev	skewness	excess kurtosis	pof(95)	pof	p-value
CBBTCUSD							
CBBTCUSD	0.0415	0.0415	-1.8686	50.6217	0.0349	0.0001	
EWMA(0.94)	1.1386	1.1386	-1.0813	15.4336	0.0462	0.3257	
EWMA(0.97)	1.0772	1.0772	-1.4521	17.8767	0.0425	0.0530	
GARCH(1,1)	1.0249	1.0249	-0.7791	8.6911	0.0415	0.0279	



## COVARIANCE MATRIX

### UNDER CONSTRUCTION

- Covariance Matrix Estimation: PCA, SVD, Shrinkage
- TODO: Risk Decomposition, Black-Litterman

```
from typing import Tuple
import numpy as np
from numpy.linalg import inv, svd
from sklearn.decomposition import PCA
from scipy.stats import multivariate_normal
import pandas as pd
from pandas import DataFrame, Series
from tqdm import tqdm
import matplotlib.pyplot as plt
from sklearn.covariance import LedoitWolf, OAS, EmpiricalCovariance
from finds.database import SQL, RedisDB
from finds.structured import CRSP
from finds.readers import FFReader
from finds.busday import BusDay
from finds.finance import halflife
from finds.plots import plot_bar
from finds.misc import Show
from secret import credentials, paths
```

```
# %matplotlib qt
VERBOSE = 0
show = Show(ndigits=4, latex=None)
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)
```

```
## Retrieve industry returns from Ken French Data Library website
name, item, suffix = ('49_Industry_Portfolios', 0, '49vw')
df = FFReader.fetch(name=name,
                     item=item,
                     suffix=suffix,
                     date_formatter=bd.endmo)
```

```

/home/terence/Dropbox/github/data-science-notebooks/finds/readers/ffreader.py:151:_
  ↵FutureWarning: The argument 'date_parser' is deprecated and will be removed in a_
  ↵future version. Please use 'date_format' instead, or read your data in as 'object
  ↵' dtype and then call 'to_datetime'.
  df = pdr.data.DataReader(name=name,
/home/terence/Dropbox/github/data-science-notebooks/finds/readers/ffreader.py:151:_
  ↵FutureWarning: The argument 'date_parser' is deprecated and will be removed in a_
  ↵future version. Please use 'date_format' instead, or read your data in as 'object
  ↵' dtype and then call 'to_datetime'.
  df = pdr.data.DataReader(name=name,
/home/terence/Dropbox/github/data-science-notebooks/finds/readers/ffreader.py:151:_
  ↵FutureWarning: The argument 'date_parser' is deprecated and will be removed in a_
  ↵future version. Please use 'date_format' instead, or read your data in as 'object
  ↵' dtype and then call 'to_datetime'.
  df = pdr.data.DataReader(name=name,
/home/terence/Dropbox/github/data-science-notebooks/finds/readers/ffreader.py:151:_
  ↵FutureWarning: The argument 'date_parser' is deprecated and will be removed in a_
  ↵future version. Please use 'date_format' instead, or read your data in as 'object
  ↵' dtype and then call 'to_datetime'.
  df = pdr.data.DataReader(name=name,
/home/terence/Dropbox/github/data-science-notebooks/finds/readers/ffreader.py:151:_
  ↵FutureWarning: The argument 'date_parser' is deprecated and will be removed in a_
  ↵future version. Please use 'date_format' instead, or read your data in as 'object
  ↵' dtype and then call 'to_datetime'.
  df = pdr.data.DataReader(name=name,
/home/terence/Dropbox/github/data-science-notebooks/finds/readers/ffreader.py:151:_
  ↵FutureWarning: The argument 'date_parser' is deprecated and will be removed in a_
  ↵future version. Please use 'date_format' instead, or read your data in as 'object
  ↵' dtype and then call 'to_datetime'.
  df = pdr.data.DataReader(name=name,
/home/terence/Dropbox/github/data-science-notebooks/finds/readers/ffreader.py:151:_
  ↵FutureWarning: The argument 'date_parser' is deprecated and will be removed in a_
  ↵future version. Please use 'date_format' instead, or read your data in as 'object
  ↵' dtype and then call 'to_datetime'.
  df = pdr.data.DataReader(name=name,
/home/terence/Dropbox/github/data-science-notebooks/finds/readers/ffreader.py:151:_
  ↵FutureWarning: The argument 'date_parser' is deprecated and will be removed in a_
  ↵future version. Please use 'date_format' instead, or read your data in as 'object
  ↵' dtype and then call 'to_datetime'.
  df = pdr.data.DataReader(name=name,

```

```

# show missing dates
nans = df.isnull()
missing = {col: {'first': min(nans[nans[col]].index),
                 'last': max(nans[nans[col]].index),
                 'missing': np.sum(nans[col])}
            for col in df if nans[col].sum() > 0}
df = df[df.index > 19690630]
Y = df.to_numpy()
show(DataFrame.from_dict(missing, orient='index'),
      latex=False,
      caption="Missing Values in FF-49 Industry Monthly Returns")

```

	first	last	missing
Missing Values in FF-49 Industry Monthly Returns			
Soda49vw	19260731	19630628	444

(continues on next page)

(continued from previous page)

Hlth49vw	19260731	19690630	516
Rubb49vw	19260731	19440630	60
FabPr49vw	19260731	19630628	444
Guns49vw	19260731	19630628	444
Gold49vw	19260731	19630628	444
PerSv49vw	19260731	19270630	12
Softw49vw	19260731	19650630	468
Paper49vw	19260731	19290629	36

## 16.1 PCA and scree plot

```
# PCA of returns covariances by SVD
# SVD: u S vT = x (T samples x N stocks)
means = Y.mean(axis=0, keepdims=True)
X = Y.copy()
```

```
x = np.array(X - means) # pre-process: demean by column
u, s, vT = np.linalg.svd(x, full_matrices=False)
v = vT.T
k = 10
print(np.cumsum(s[:k]**2/np.sum(np.diag(s**2))))
print('u:', u.shape, 's:', s.shape, 'vT:', vT.shape, 'v', v.shape)
```

```
[0.56081683 0.62434391 0.66423103 0.70158944 0.73044243 0.75277385
 0.77084836 0.78503209 0.79886157 0.81079137]
u: (648, 49) s: (49,) vT: (49, 49) v (49, 49)
```

```
# sklearn PCA: X (T samples x N features/stocks), sanity check same results
pca = PCA() # note: PCA first demeans input X by column mean_
y = pca.fit_transform(X) # project X (stock returns) onto the components
print(pca.explained_variance_ratio_[:k])
print('y:', y.shape, 'x:', x.shape, 'components_:', pca.components_.shape)
```

```
[0.56081683 0.06352708 0.03988711 0.03735841 0.02885299 0.02233142
 0.01807451 0.01418374 0.01382948 0.01192979]
y: (648, 49) x: (648, 49) components_: (49, 49)
```

```
# assert: s == singular_values_ (aka 2-norm of the projection on components)
print('singular values:', np.allclose(pca.singular_values_, s))
```

```
singular values: True
```

```
# assert: s**2 / len(y) == explained_variance_
print('singular values:', np.allclose(pca.singular_values_, s))
```

```
singular values: True
```

```
# assert: x @ v == transform(x) (aka projection on components) (aka returns)
print('projections:', [np.allclose((x @ v)[:,i], -y[:,i]) or
                      np.allclose((x @ v)[:,i], y[:,i]) for i in range(k)])
```

projections: [True, True, True, True, True, True, True, True, True]

```
# assert: u @ s == transform(x) (aka factor returns)
print('projections:', [np.allclose(u[:,i]*s[i], -y[:,i]) or
                      np.allclose(u[:,i]*s[i], y[:,i]) for i in range(k)])
```

projections: [True, True, True, True, True, True, True, True, True]

```
# assert: cols of v == rows of components_ (right SVD eigenvectors) (weights)
print('components:', [np.allclose(pca.components_[i,:], -v[:,i]) or
                      np.allclose(pca.components_[i,:], v[:,i])
                      for i in range(k)])
```

components: [True, True, True, True, True, True, True, True, True]

```
# assert: covariance matrix == loadings.T @ loadings
loadings = np.diag(pca.singular_values_) @ pca.components_
print('covariance matrix:', np.allclose(x.T @ x, loadings.T @ loadings))
```

covariance matrix: True

```
### Projection on first component, and average "market" factor
t = pca.components_
top_k = 4
DataFrame({'frac weights +ve': np.mean(t[:top_k, :] >= 0, axis=1),
           'sum weights': np.sum(t[:top_k, :], axis=1),
           'sum abs weights': np.sum(np.abs(t[:top_k, :]), axis=1),
           'corr with eql-wtd market returns':
               [np.corrcoef(x.mean(axis=1), y[:,i])[0,1]
                for i in range(top_k)]},
           index=[f"PC{i+1}" for i in range(top_k)])
```

	frac weights +ve	sum weights	sum abs weights
PC1	0.000000	-6.836258	6.836258
PC2	0.244898	0.085580	4.141710
PC3	0.428571	0.159702	3.708582
PC4	0.326531	-0.972123	5.289028

	corr with eql-wtd market returns
PC1	-0.998857
PC2	0.004209
PC3	0.006223
PC4	-0.036660

```
### Plot components/portfolio weights distribution
fig, axes = plt.subplots(nrows=1, ncols=3, num=1, clear=True, figsize=(10, 4))
```

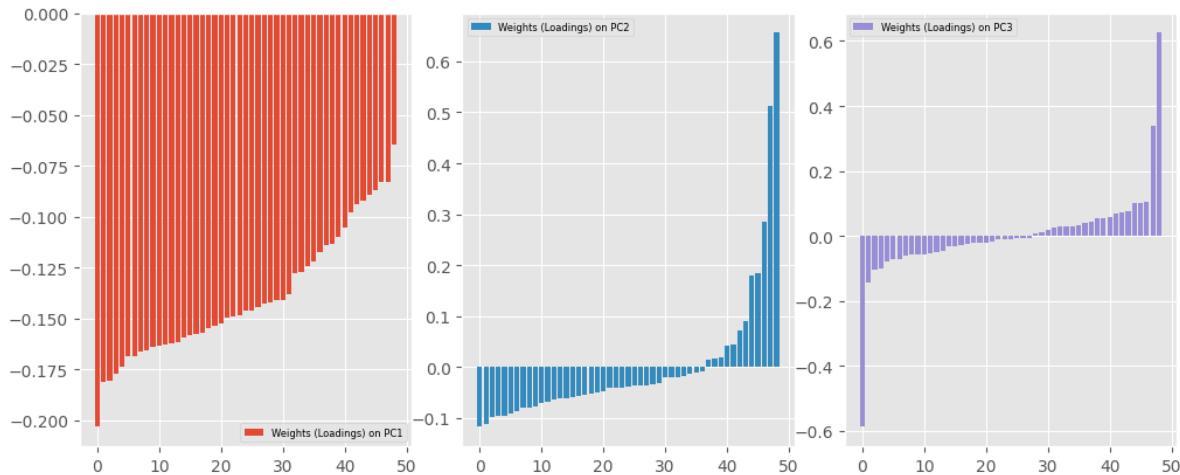
(continues on next page)

(continued from previous page)

```

for i, ax in enumerate(np.ravel(axes)):
    ax.bar(np.arange(t.shape[1]),
           np.sort(t[i, :]),
           color=f"C{i}")
ax.legend([f"Weights (Loadings) on PC{i+1}"], fontsize=6)
plt.tight_layout(pad=0)
plt.savefig(imgdir / 'weights.jpg')

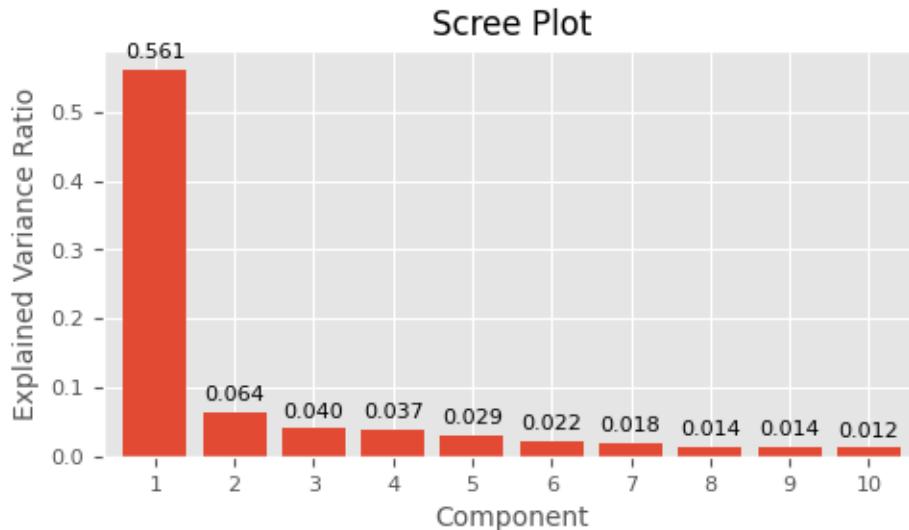
```



```

### Scree Plot
fig, ax = plt.subplots(1, 1, num=1, clear=True, figsize=(5, 3))
k=10
plot_bar(Series(pca.explained_variance_ratio_[:k],
                 index=np.arange(1, k+1)),
          ylabel='Explained Variance Ratio',
          xlabel="Component",
          legend=None,
          title='Scree Plot',
          ax=ax,
          fontsize=8,
          labels=[f"{i:.3f}" for i in pca.explained_variance_ratio_[:k]])
plt.tight_layout(pad=1)
plt.savefig(imgdir / 'explained.jpg')

```



### GMV portfolio volatility

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

```
# Helper to compute EWMA covariance matrix estimate
def ewma(X, alpha=0.03, demean=False):
    weights = (1 - alpha)**np.arange(len(X)) [::-1]
    if demean:
        X = X - X.mean(axis=0, keepdims=True)
    return (weights.reshape((1, -1)) * X.T) @ X / weights.sum()
```

```
# Rolling monthly evaluation
r = {}      # collect results of covariance matrix models
start_eval = 20000101
for retdate in tqdm(df.index[(df.index >= start_eval)]):
    x_train = Y[df.index < retdate, :]
    x_test = Y[df.index == retdate, :]
    keep = 5 * 12 # keep five years
    N = x_train.shape[1]
    r[retdate] = {}

    cov = EmpiricalCovariance().fit(x_train[-keep:, :]).covariance_
    r[retdate]['Full Covariance'] = float(gmv(cov, x_test))

    for alpha in [0.1, 0.06, 0.03, 0.01, 0.003]:
        r[retdate][f'EWMA({halflife(alpha=alpha):.0f}mo)'] = \
            float(gmv(ewma(x_train, alpha=alpha), x_test))

    r[retdate]['Eye'] = float(gmv(np.identity(x_train.shape[1]), x_test))
    r[retdate]['Diagonal'] = float(gmv(np.diagflat(np.diag(cov)), x_test))
```

(continues on next page)

(continued from previous page)

```

for k in [2, 5, 10, 15, 20]:
    r[retdate][f"PC 1-{k}"] = float(gmv(PCA(k).fit(x_train[-keep:, :])\
        .get_covariance(), x_test))

r[retdate]['LW'] = float(gmv(LedoitWolf().fit(x_train[-keep:, :])\
    .covariance_, x_test))

r[retdate]['OAS'] = float(gmv(OAS().fit(x_train[-keep:, :])\
    .covariance_, x_test))

```

100% |██████████| 282/282 [23:31<00:00, 5.00s/it]

```

ts = DataFrame.from_dict(r, orient='index')
vol = np.std(ts, axis=0)
show(vol, caption='Realized volatility of minimum variance portfolios')

```

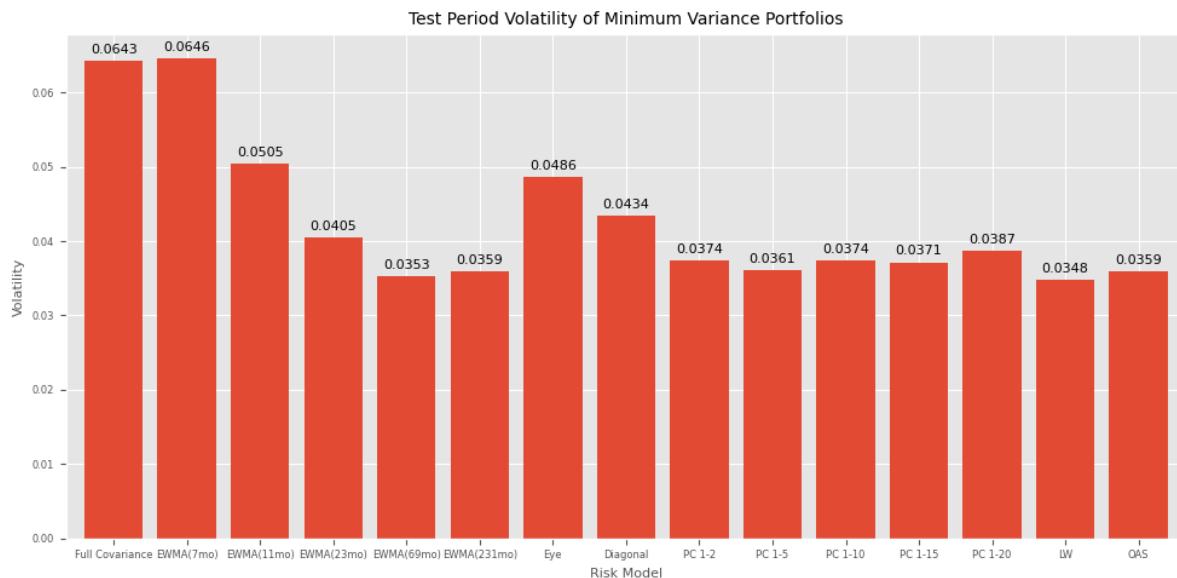
	0
Realized volatility of minimum variance portfolios	
Full Covariance	0.0643
EWMA (7mo)	0.0646
EWMA (11mo)	0.0505
EWMA (23mo)	0.0405
EWMA (69mo)	0.0353
EWMA (231mo)	0.0359
Eye	0.0486
Diagonal	0.0434
PC 1-2	0.0374
PC 1-5	0.0361
PC 1-10	0.0374
PC 1-15	0.0371
PC 1-20	0.0387
LW	0.0348
OAS	0.0359

Plot evaluation period realized volatility of minimum variance portfolios

```

fig, ax = plt.subplots(1, 1, num=1, clear=True, figsize=(10, 5))
plot_bar(vol,
    ylabel='Volatility',
    xlabel='Risk Model',
    title='Test Period Volatility of Minimum Variance Portfolios',
    labels=[f"{v:.4f}" for v in vol],
    legend='',
    fontsize=6,
    ax=ax)
plt.tight_layout()
plt.savefig(imgdir / 'gmv.jpg')

```



Newey-west, Scholes Williams Beta  $1/T \sum_t e_t^2 + 2/T \sum_L \sum_T w_l e_t e_{t-l}$ :  $w_l = 1/(L+1)$

Risk Decomposition: MCR, BL equilibrium, Risk Parity  $u.T @ x = \text{beta}$ , since  $u$  is standardized (orthogonal) factor returns  $u @ \text{beta} = \text{stock's return due to orthogonal component}$   $(u @ \text{beta})^2 = \text{variation of stock returns due to orthogonal component}$

## TERM STRUCTURE

### UNDER CONSTRUCTION

- yield curve, duration, DV01, bootstrap
- term structure shifts
- splines

```
from typing import List, Dict, Any
import re
from datetime import datetime
import numpy as np
import pandas as pd
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
from finds.readers import Alfred
from finds.busday import BusDay
from finds.misc import Show
from finds.finance import Interest
from secret import credentials, paths
VERBOSE = 0
show = Show(ndigits=4, latex=None)
# %matplotlib qt
imgdir = paths['images'] / 'ts'
```

```
alf = Alfred(api_key=credentials['fred']['api_key'])
```

### 17.1 Term Structure of Interest Rates

```
## list of monthly Constant Maturity Treasury, excluding inflation-indexed
c = alf.get_category(115) # Fed H.15 Selected Interest Rates
print(c['id'], c['name'])
t = Series({s['id']: s['title'] for s in c['series']}
           if (s['frequency']=='Monthly'
               and 'Inflation' not in s['title']
               and 'DISCONT' not in s['title']}).rename('title')
show(t.to_frame(), formatters=[lambda x: x.split(',') [0]],
      caption="Constant Maturity Treasuries in FRED")
```

```

https://api.stlouisfed.org/fred/category?category_id=115&api_
  ↵key=eb428f329807459fa7bee89be4ebbc3b&file_type=json
https://api.stlouisfed.org/fred/category/children?category_id=115&api_
  ↵key=eb428f329807459fa7bee89be4ebbc3b&file_type=json
https://api.stlouisfed.org/fred/category/series?category_id=115&api_
  ↵key=eb428f329807459fa7bee89be4ebbc3b&file_type=json&offset=0
https://api.stlouisfed.org/fred/category/series?category_id=115&api_
  ↵key=eb428f329807459fa7bee89be4ebbc3b&file_type=json&offset=1000
115 Treasury Constant Maturity

```

```

  ↵title
Constant Maturity Treasuries in FRED
GS1                               Market Yield on U.S. Treasury Securities at
  ↵1-...
GS10                             Market Yield on U.S. Treasury Securities at
  ↵10...
GS1M                             Market Yield on U.S. Treasury Securities at
  ↵1-...
GS2                               Market Yield on U.S. Treasury Securities at
  ↵2-...
GS20                             Market Yield on U.S. Treasury Securities at
  ↵20...
GS3                               Market Yield on U.S. Treasury Securities at
  ↵3-...
GS30                             Market Yield on U.S. Treasury Securities at
  ↵30...
GS3M                             Market Yield on U.S. Treasury Securities at
  ↵3-...
GS5                               Market Yield on U.S. Treasury Securities at
  ↵5-...
GS6M                             Market Yield on U.S. Treasury Securities at
  ↵6-...
GS7                               Market Yield on U.S. Treasury Securities at
  ↵7-...

```

```

## retrieve CMT yields, and infer maturity from label
data = [alf(s, freq='M') for s in t.index]
data = pd.concat([alf(s, freq='M') for s in t.index], axis=1, join='inner')
data.columns = [int(re.sub('\D', '', col)) * (1 if col.endswith('M') else 12)
               for col in data.columns] # infer maturity in months from label
data = data.sort_index(axis=1) # sort columns by ascending maturity
show(data)

```

	1	3	6	12	24	36	60	84	120	240	360
date											
20010731	3.67	3.59	3.56	3.62	4.04	4.31	4.76	5.06	5.24	5.75	5.61
20010831	3.53	3.44	3.39	3.47	3.76	4.04	4.57	4.84	4.97	5.58	5.48
20010930	2.68	2.69	2.71	2.82	3.12	3.45	4.12	4.51	4.73	5.53	5.48
20011031	2.27	2.20	2.17	2.33	2.73	3.14	3.91	4.31	4.57	5.34	5.32
20011130	1.99	1.91	1.92	2.18	2.78	3.22	3.97	4.42	4.65	5.33	5.12
...	...	...	...	...	...	...	...	...	...	...	...
20230331	4.49	4.86	4.99	4.68	4.30	4.09	3.82	3.77	3.66	3.94	3.77
20230430	4.17	5.07	4.99	4.68	4.02	3.76	3.54	3.50	3.46	3.80	3.68
20230531	5.49	5.31	5.27	4.91	4.13	3.82	3.59	3.58	3.57	3.96	3.86

(continues on next page)

(continued from previous page)

```

20230630  5.20  5.42  5.42  5.24  4.64  4.27  3.95  3.85  3.75  4.04  3.87
20230731  5.39  5.49  5.53  5.37  4.83  4.47  4.14  4.03  3.90  4.15  3.96

[265 rows x 11 columns]

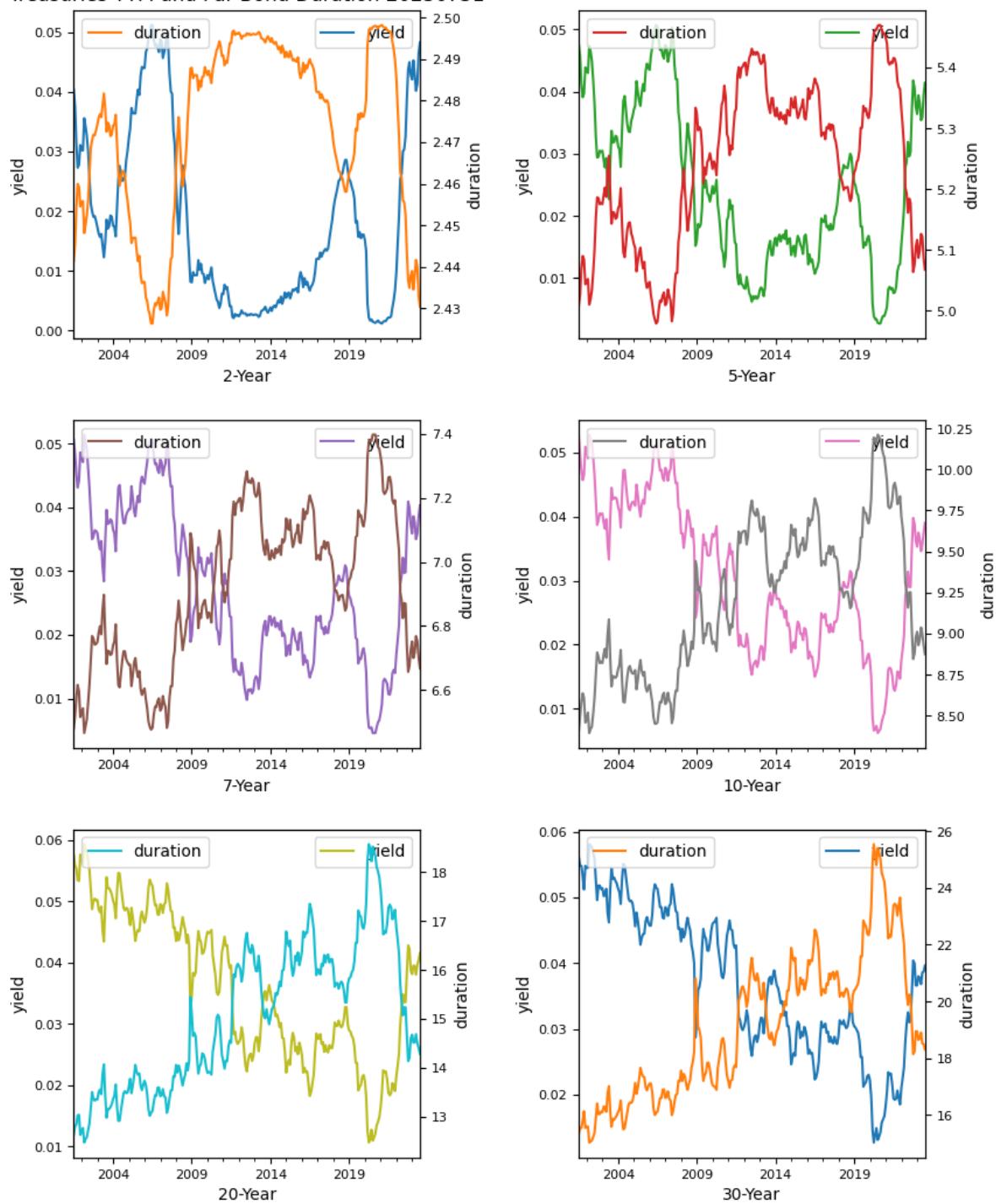
```

```

## Plot history of yields and durations
yields = pd.concat([data[yr * 12].rename(yr) / 100
                    for yr in [2, 5, 7, 10, 20, 30]], axis=1)
yields.index = pd.to_datetime(yields.index, format='%Y%m%d')
durations = pd.concat([yields[yr].apply(Interest.par_duration, n=yr, m=2)
                       for yr in yields], axis=1)
fig, axes = plt.subplots(3, 2, clear=True, num=1, figsize=(9, 11))
for i in range(6):
    ax = axes[i // 2, i % 2]
    title = (i==0)*f"Treasuries YTM and Par Bond Duration {data.index[-1]}"
    yields.iloc[:, i].rename('yield')\
        .plot(ax=ax, c=f"C{i*2}", fontsize=8, legend=True,
              ylabel='yield', title=title)
    bx = ax.twinx()
    durations.iloc[:, i]\.
        rename('duration')\
        .plot(ax=bx, c=f"C{i*2+1}", fontsize=8, ylabel='duration',
              legend=True)
    ax.set_xlabel(f"yields.columns[i])-Year")
plt.tight_layout(pad=2)
plt.savefig(imgdir / 'term.jpg')

```

Treasuries YTM and Par Bond Duration 20230731



## 17.2 Splines

- Interpolate yield curve

```

## https://home.treasury.gov/policy-issues/financing-the-government/
## interest-rate-statistics/treasury-yield-curve-methodology
curve_date = 20211231 #data.index[-1] # for latest date available 20021231 #
from scipy.interpolate import CubicSpline
yields = data.loc[curve_date].values
maturities = data.columns.to_list()
yield_curve = CubicSpline(x=maturities, y=yields, bc_type='clamped')

## Bootstrap nominal annual rates from interpolated ytm semi-annually
ytm = [yield_curve(t) / 100 for t in range(6, 361, 6)]
m = 2
nominal = np.array([]) # to store annualized rates from bootstrap
for y in ytm:
    nominal = np.append(nominal,
                         Interest.bootstrap_rates(y, nominal=nominal, m=m))
spot_rates = (((1 + np.array(nominal/2))**2) - 1) * 100 # annualize effective

## Sanity-check spot rate results
for n in range(1, 1 + len(ytm)): # discounted cash flows all close to par=1?
    assert abs(Interest.discounted_cash_flow(flops=ytm[n-1] / m,
                                              spot=nominal[:n] / m)
               + Interest.present_value(1, n=n, spot=nominal[n-1] / m) - 1) < 0.001

f = np.array(Interest.forward_rates(spot=nominal / m)) # as single-period rate
for n in range(1, 1+len(ytm)): # compounded forwards all close to nominal
    assert abs(np.prod(1 + f[:n])
               - (1 + nominal[n-1] / m)**n) < 0.001
forwards = (((1 + np.array(f))**2) - 1) * 100 # annual effective forward

# Retrieve reconstructed yield curve history
## https://sites.google.com/view/jingcynthiawu/yield-data
def fetch_liuwu(file_id='1_u9cRxmOSiwp_tFvlaORuhS-zwl935s0'):
    src = "https://drive.google.com/uc?export=download&id={}".format(file_id)
    x = pd.ExcelFile(src)
    df = x.parse()
    dates = np.where(df.iloc[:, 0].astype(str).str[0].str.isdigit())[0]
    return DataFrame(np.exp(df.iloc[dates, 1:361].astype(float).values/100) - 1,
                     index=BusDay.to_monthend(df.iloc[dates, 0].values),
                     columns=np.arange(1, 361))
df = fetch_liuwu()
reconstructed_rates = df.loc[curve_date, :] * 100 # as of curve_date
f = np.array(Interest.forward_rates(spot=reconstructed_rates / 1200))
reconstructed_forward = (((1 + np.array(f))**12) - 1) * 100 # annualize

## Plot historical reconstructed rates as 3D surface
from mpl_toolkits.mplot3d import Axes3D
r = df.dropna()
X, Y = np.meshgrid((r.index//10000) + (((r.index//100)%100)-1)/12,
                    r.columns.astype(float)/12)

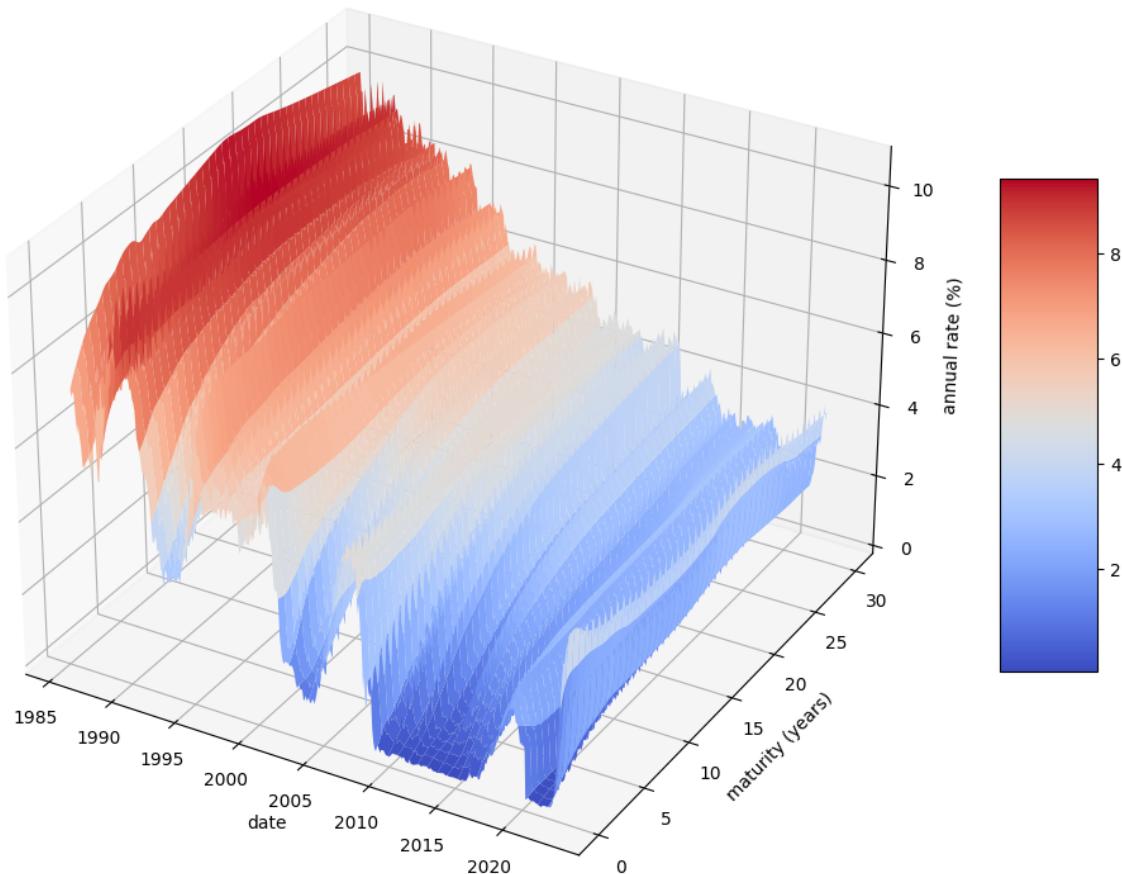
```

(continues on next page)

(continued from previous page)

```
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 rate (%)')
fig.colorbar(f, shrink=0.5, aspect=5)
plt.tight_layout(pad=0)
plt.savefig(imgdir / 'reconstructed.jpg')
```

Reconstructed Treasury Interest Rates [Liu and Wu (2020)]



```
# Plot yield curves as of recent date
semi_annual = np.arange(6, 361, 6)      # enumerate semi-annual months
fig, ax = plt.subplots(num=1, clear=True, figsize=(12,7))
ax.plot(maturities,
        yields, 'o',
        label='CMT nominal yields (assuming par bond)')
ax.plot(semi_annual,
```

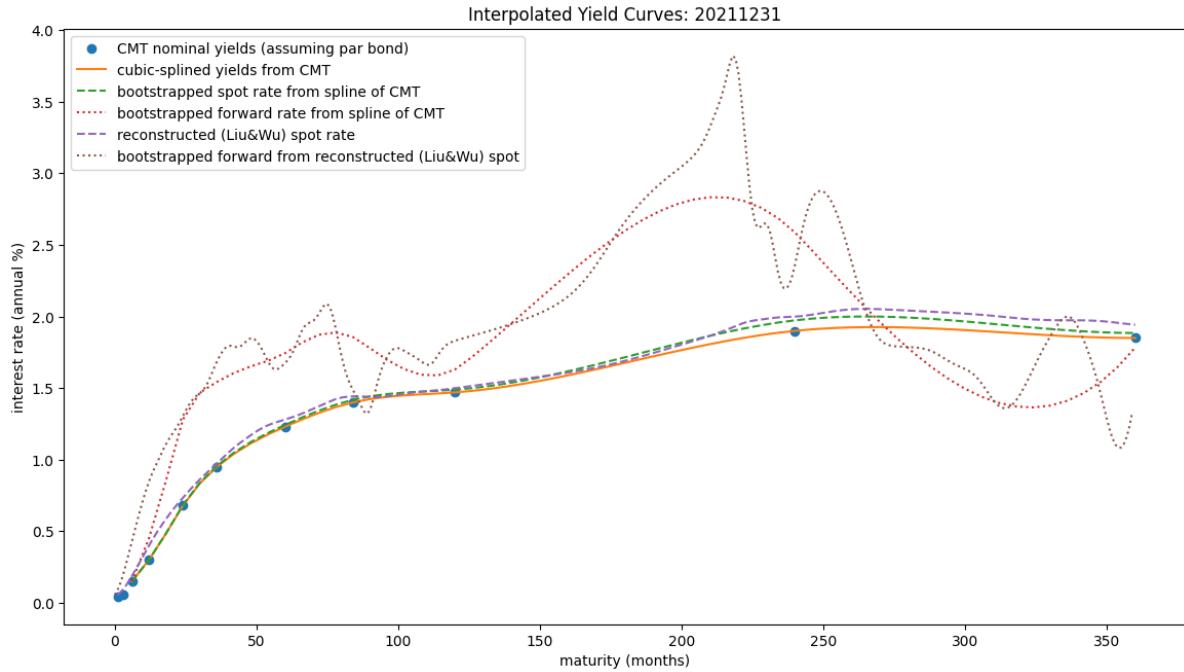
(continues on next page)

(continued from previous page)

```

yield_curve(semi_annual),
label="cubic-splined yields from CMT")
ax.plot(semi_annual,
        spot_rates, '--',
        label="bootstrapped spot rate from spline of CMT")
ax.plot(semi_annual,
        forwards, ':',
        label="bootstrapped forward rate from spline of CMT")
ax.plot(reconstructed_rates, '--',
        label="reconstructed (Liu&Wu) spot rate")
ax.plot(reconstructed_forward, ':',
        label="bootstrapped forward from reconstructed (Liu&Wu) spot")
ax.set_title(f"Interpolated Yield Curves: {curve_date}")
ax.set_xlabel('maturity (months)')
ax.set_ylabel('interest rate (annual %)')
ax.legend()
plt.tight_layout(pad=2)
plt.savefig(imgdir / 'curve.jpg')

```





## BOND RETURNS

### UNDER CONSTRUCTION

- PCA
- bond portfolio returns

```
import numpy as np
import pandas as pd
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
import statsmodels.api as sm
from finds.readers import Alfred
from finds.misc import Show
from secret import credentials, paths
# %matplotlib qt
VERBOSE = 0
show = Show(ndigits=4, latex=None)
imgdir = paths['images'] / 'ts'
```

```
alf = Alfred(api_key=credentials['fred']['api_key'], verbose=VERBOSE)
```

### Bond Return Components

```
## get Merrill Lynch bond indexes
c = alf.get_category(32413)
print(c['id'], c['name'])
t = Series({s['id']: s['title'] for s in c['series']})
t
```

```
https://api.stlouisfed.org/fred/category?category_id=32413&api_
  ↵key=eb428f329807459fa7bee89be4ebbc3b&file_type=json
https://api.stlouisfed.org/fred/category/children?category_id=32413&api_
  ↵key=eb428f329807459fa7bee89be4ebbc3b&file_type=json
https://api.stlouisfed.org/fred/category/series?category_id=32413&api_
  ↵key=eb428f329807459fa7bee89be4ebbc3b&file_type=json&offset=0
https://api.stlouisfed.org/fred/category/series?category_id=32413&api_
  ↵key=eb428f329807459fa7bee89be4ebbc3b&file_type=json&offset=1000
32413 BofA Merrill Lynch Total Bond Return Index Values
```

BAMLCC0A0CMTRIV	ICE BofA US Corporate Index Total Return Index...
BAMLCC0A1AAATRIV	ICE BofA AAA US Corporate Index Total Return I...
BAMLCC0A2AATRIV	ICE BofA AA US Corporate Index Total Return In...

(continues on next page)

(continued from previous page)

BAMLCC0A3ATRIV	ICE BofA Single-A US Corporate Index Total Ret...
BAMLCC0A4BBBTRIV	ICE BofA BBB US Corporate Index Total Return I...
BAMLCC1A013YTRIV	ICE BofA 1-3 Year US Corporate Index Total Ret...
BAMLCC2A035YTRIV	ICE BofA 3-5 Year US Corporate Index Total Ret...
BAMLCC3A057YTRIV	ICE BofA 5-7 Year US Corporate Index Total Ret...
BAMLCC4A0710YTRIV	ICE BofA 7-10 Year US Corporate Index Total Re...
BAMLCC7A01015YTRIV	ICE BofA 10-15 Year US Corporate Index Total R...
BAMLCC8A015PYTRIV	ICE BofA 15+ Year US Corporate Index Total Ret...
BAMLEM1BRRAAA2ACRPITRIV	ICE BofA AAA-A Emerging Markets Corporate Plus...
BAMLEM1RAAA2ALCRPIUSTRIV	ICE BofA AAA-A US Emerging Markets Liquid Corp...
BAMLEM2BRRBBBCRPITRIV	ICE BofA BBB Emerging Markets Corporate Plus I...
BAMLEM2RBBLCRPIUSTRIV	ICE BofA BBB US Emerging Markets Liquid Corpor...
BAMLEM3BRRBBCRPITRIV	ICE BofA BB Emerging Markets Corporate Plus In...
BAMLEM3RBBLCRPIUSTRIV	ICE BofA BB US Emerging Markets Liquid Corpora...
BAMLEM4BRRBLCRPITRIV	ICE BofA B & Lower Emerging Markets Corporate ...
BAMLEM4RBLLCRPIUSTRIV	ICE BofA B & Lower US Emerging Markets Liquid ...
BAMLEM5BCOCRPITRIV	ICE BofA Crossover Emerging Markets Corporate ...
BAMLEMALLCRPIASIAUSTRIV	ICE BofA Asia US Emerging Markets Liquid Corpo...
BAMLEMCPITRIV	ICE BofA Emerging Markets Corporate Plus Index...
BAMLEMCLLCRPPIUSTRIV	ICE BofA US Emerging Markets Liquid Corporate ...
BAMLEMEBCRPITRIV	ICE BofA Euro Emerging Markets Corporate Plus ...
BAMLEMELLCRPIEMEAUSTRIV	ICE BofA EMEA US Emerging Markets Liquid Corpo...
BAMLEMFLFLCRPIUSTRIV	ICE BofA Financial US Emerging Markets Liquid ...
BAMLEMSFCRPITRIV	ICE BofA Private Sector Financial Emerging Mar...
BAMLEMHBHYCRPITRIV	ICE BofA High Yield Emerging Markets Corporate...
BAMLEMHGGLCRPIUSTRIV	ICE BofA High Grade US Emerging Markets Liquid...
BAMLEMHYHYLCRPPIUSTRIV	ICE BofA High Yield US Emerging Markets Liquid...
BAMLEMIBHGCRPITRIV	ICE BofA High Grade Emerging Markets Corporate...
BAMLEMILLLCRPILAUSTRIV	ICE BofA Latin America US Emerging Markets Liq...
BAMLEMNFNFLCRPIUSTRIV	ICE BofA Non-Financial US Emerging Markets Liq...
BAMLEMNSNFCRPITRIV	ICE BofA Non-Financial Emerging Markets Corpor...
BAMLEMPBPUBSICRPITRIV	ICE BofA Public Sector Issuers Emerging Market...
BAMLEMPTPRVICRPITRIV	ICE BofA Private Sector Issuers Emerging Marke...
BAMLEMPUPUBSLCRPIUSTRIV	ICE BofA Public Sector Issuers US Emerging Mar...
BAMLEMPVPRIVSLCRPIUSTRIV	ICE BofA Private Sector Issuers US Emerging Ma...
BAMLEMRACRPIASIASTRIV	ICE BofA Asia Emerging Markets Corporate Plus ...
BAMLEMRECRPIEMEATRIV	ICE BofA EMEA Emerging Markets Corporate Plus ...
BAMLEMRLCRPILATRIV	ICE BofA Latin America Emerging Markets Corpor...
BAMLEMUBCRPIUSTRIV	ICE BofA US Emerging Markets Corporate Plus In...
BAMLEMXOCOLCRPIUSTRIV	ICE BofA Crossover US Emerging Markets Liquid ...
BAMLHE00EHYITRIV	ICE BofA Euro High Yield Index Total Return In...
BAMLHYH0A0HYM2TRIV	ICE BofA US High Yield Index Total Return Inde...
BAMLHYH0A1BBTRIV	ICE BofA BB US High Yield Index Total Return I...
BAMLHYH0A2BTRIV	ICE BofA Single-B US High Yield Index Total Re...
BAMLHYH0A3CMTRIV	ICE BofA CCC & Lower US High Yield Index Total...

```
b = []    # accumulate bond returns
for s in t.index:
    b.append(alf(s, start=19961231) )
bonds = pd.concat(b, axis=1)
```

```
## Show blocks of data available
v = bonds.notna().sum(axis=1).rename('count')
v = pd.concat([v, (v != v.shift()).cumsum().rename('notna')], axis=1)
```

(continues on next page)

(continued from previous page)

```
g = v.reset_index().groupby(['notna', 'count'])['date'].agg(['first','last'])
g
```

		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	20230829

```
start_date = 19981231
rets = bonds.loc[bonds.index >= start_date,
                 bonds.loc[start_date].notna().values]
rets = pd.concat([alf.transform(rets[col], log=1, diff=1)
                  for col in rets.columns], axis=1)
show(Series(alf.header(rets.columns),
            index=rets.columns,
            name='title').to_frame().rename_axis('series'),
      max_colwidth=88,
      caption="Bond Index Total Returns")
```

	title
Bond Index Total Returns	
BAMLCC0A0CMTRIV	ICE BofA US Corporate Index Total Return Index...
BAMLCC0A1AAATRIV	ICE BofA AAA US Corporate Index Total Return I...
BAMLCC0A2AATRIV	ICE BofA AA US Corporate Index Total Return In...
BAMLCC0A3ATRIV	ICE BofA Single-A US Corporate Index Total Ret...
BAMLCC0A4BBBTRIV	ICE BofA BBB US Corporate Index Total Return I...
BAMLCC1A013YTRIV	ICE BofA 1-3 Year US Corporate Index Total Ret...
BAMLCC2A035YTRIV	ICE BofA 3-5 Year US Corporate Index Total Ret...
BAMLCC3A057YTRIV	ICE BofA 5-7 Year US Corporate Index Total Ret...
BAMLCC4A0710YTRIV	ICE BofA 7-10 Year US Corporate Index Total Re...
BAMLCC7A01015YTRIV	ICE BofA 10-15 Year US Corporate Index Total R...
BAMLCC8A015PYTRIV	ICE BofA 15+ Year US Corporate Index Total Ret...
BAMLEM1BRRAAA2ACRPITRIV	ICE BofA AAA-A Emerging Markets Corporate Plus...
BAMLEM2BRRBBBCRPITRIV	ICE BofA BBB Emerging Markets Corporate Plus I...
BAMLEM3BRRBBCRPITRIV	ICE BofA BB Emerging Markets Corporate Plus In...
BAMLEM4BRRBLCRPITRIV	ICE BofA B & Lower Emerging Markets Corporate ...
BAMLEM5BCOCRPITRIV	ICE BofA Crossover Emerging Markets Corporate ...
BAMLEMCBPITRIV	ICE BofA Emerging Markets Corporate Plus Index...
BAMLEMEBCRPIETRIV	ICE BofA Euro Emerging Markets Corporate Plus ...
BAMLEMFSFCRPITRIV	ICE BofA Private Sector Financial Emerging Mar...
BAMLEMHBHYCRPITRIV	ICE BofA High Yield Emerging Markets Corporate...
BAMLEMIBHGCRPITRIV	ICE BofA High Grade Emerging Markets Corporate...
BAMLEMNSNFCRPITRIV	ICE BofA Non-Financial Emerging Markets Corpor...
BAMLEMPBPUBSICRPITRIV	ICE BofA Public Sector Issuers Emerging Market...
BAMLEMPTPRVICRPITRIV	ICE BofA Private Sector Issuers Emerging Marke...
BAMLEMRACRPIASIAITRIV	ICE BofA Asia Emerging Markets Corporate Plus ...
BAMLEMRECRPIEMEATRIV	ICE BofA EMEA Emerging Markets Corporate Plus ...
BAMLEMRLCRPILATRIV	ICE BofA Latin America Emerging Markets Corpor...
BAMLEMUBCRPIUSTRIV	ICE BofA US Emerging Markets Corporate Plus In...
BAMLHE00EHYITRIV	ICE BofA Euro High Yield Index Total Return In...

(continues on next page)

(continued from previous page)

BAMLHYH0A0HYM2TRIV	ICE BofA US High Yield Index Total Return Inde...
BAMLHYH0A1BBTRIV	ICE BofA BB US High Yield Index Total Return I...
BAMLHYH0A2BTRIV	ICE BofA Single-B US High Yield Index Total Re...
BAMLHYH0A3CMTRIV	ICE BofA CCC & Lower US High Yield Index Total...

```
# Marginal Variance Explained
x = np.array(rets.iloc[1:].replace(np.nan, 0))
d = rets.iloc[1:].index.rename(None)
c = rets.columns
r = 3
"""
from finds.alfred import marginalRsq
mR2 = marginalRsq(x, standardize=True)
print(f"Explained by {r} factors: {np.sum(np.mean(mR2[:r,:], axis=1)):.3f}"
      f" ({len(x)} obs)")
df = DataFrame({'explained': np.mean(mR2, axis=1)},
               index=np.arange(1, len(mR2) + 1))
df.iloc[4]
"""


```

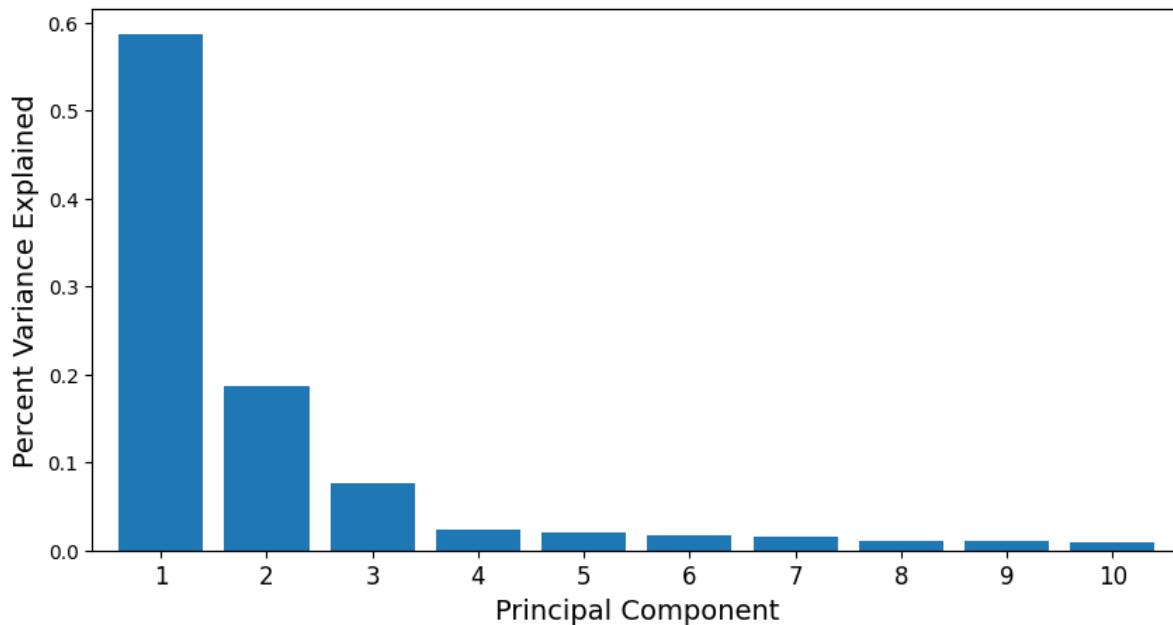
```
'\nfrom finds.alfred import marginalRsq\nmR2 = marginalRsq(x, standardize=True)\nprint(f"Explained by {r} factors: {np.sum(np.mean(mR2[:r,:], axis=1)):.3f}"\n      f" ({len(x)} obs)")\nndf = DataFrame({'explained': np.mean(mR2, axis=1)},\n               index=np.arange(1, len(mR2) + 1))\nndf.iloc[4]\n'
```

```
## Same calculation with sklearn
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.decomposition import PCA
pipe = Pipeline([('scaler', StandardScaler()), ('pca', PCA())])
pipe.fit(x)
print(pipe.named_steps['pca'].explained_variance_ratio_) # sanity check
scree = Series(pipe.named_steps['pca'].explained_variance_ratio_,
               index=np.arange(1, x.shape[1]+1))
```

```
[5.86789902e-01 1.86999891e-01 7.70507495e-02 2.31692776e-02
 2.09236720e-02 1.74466819e-02 1.59969576e-02 1.11997496e-02
 1.08857821e-02 9.62694118e-03 7.23593993e-03 5.86580950e-03
 5.18815697e-03 4.67376507e-03 4.08324626e-03 2.59381179e-03
 2.45324538e-03 2.30697359e-03 1.37010289e-03 9.79850816e-04
 8.05769526e-04 6.38160060e-04 4.08352451e-04 3.82511512e-04
 3.14728492e-04 2.77963772e-04 9.32990978e-05 8.59047248e-05
 5.50770945e-05 4.00939820e-05 3.59304585e-05 1.58570390e-05
 5.84577715e-06]
```

```
## Scree plot
fig, ax = plt.subplots(num=1, clear=True, figsize=(10, 5))
scree[:10].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.savefig(imgdir / 'scree.jpg')
```

## Scree Plot: PCA of FRED BofA Bond Return Indexes



```
# Rsquare of components explained by Interest Rate Indicators
rates = ['BAA10Y', 'T10Y2Y', 'DGS10'] # compare rate levels
rates = pd.concat([alf(s) for s in rates], axis=1).reindex(d).pad()
rates.index = pd.DatetimeIndex(rates.index.astype(str), freq='infer')
```

```
/tmp/ipykernel_4082727/620974196.py:3: FutureWarning: DataFrame.pad/Series.pad is
  deprecated. Use DataFrame.ffill/Series.ffill instead
  rates = pd.concat([alf(s) for s in rates], axis=1).reindex(d).pad()
```

```
factors = DataFrame(pipe.transform(x)[:, :3],
                     columns=np.arange(1, 4),
                     index=pd.DatetimeIndex(d.astype(str), freq='infer'))
```

```
rsq = [sm.OLS(factors[col].cumsum(), rates).fit().rsquared
       for col in factors.columns]
res = DataFrame({'Rsq of rate levels': rsq}, index=factors.columns)
res['Rsq of rate changes'] = [sm.OLS(factors[col],
                                      rates.diff().fillna(0))\
                                      .fit().rsquared
       for col in factors.columns]
show(res, max_colwidth=75,
      caption="R-squared of each PC explained by Interest Rate Indicators")
```

Rsq of rate levels	
R-squared of each PC explained by Interest Rate...	
1	0.7131 \
2	0.2942
3	0.9211

Rsq of rate changes	
R-squared of each PC explained by Interest Rate...	

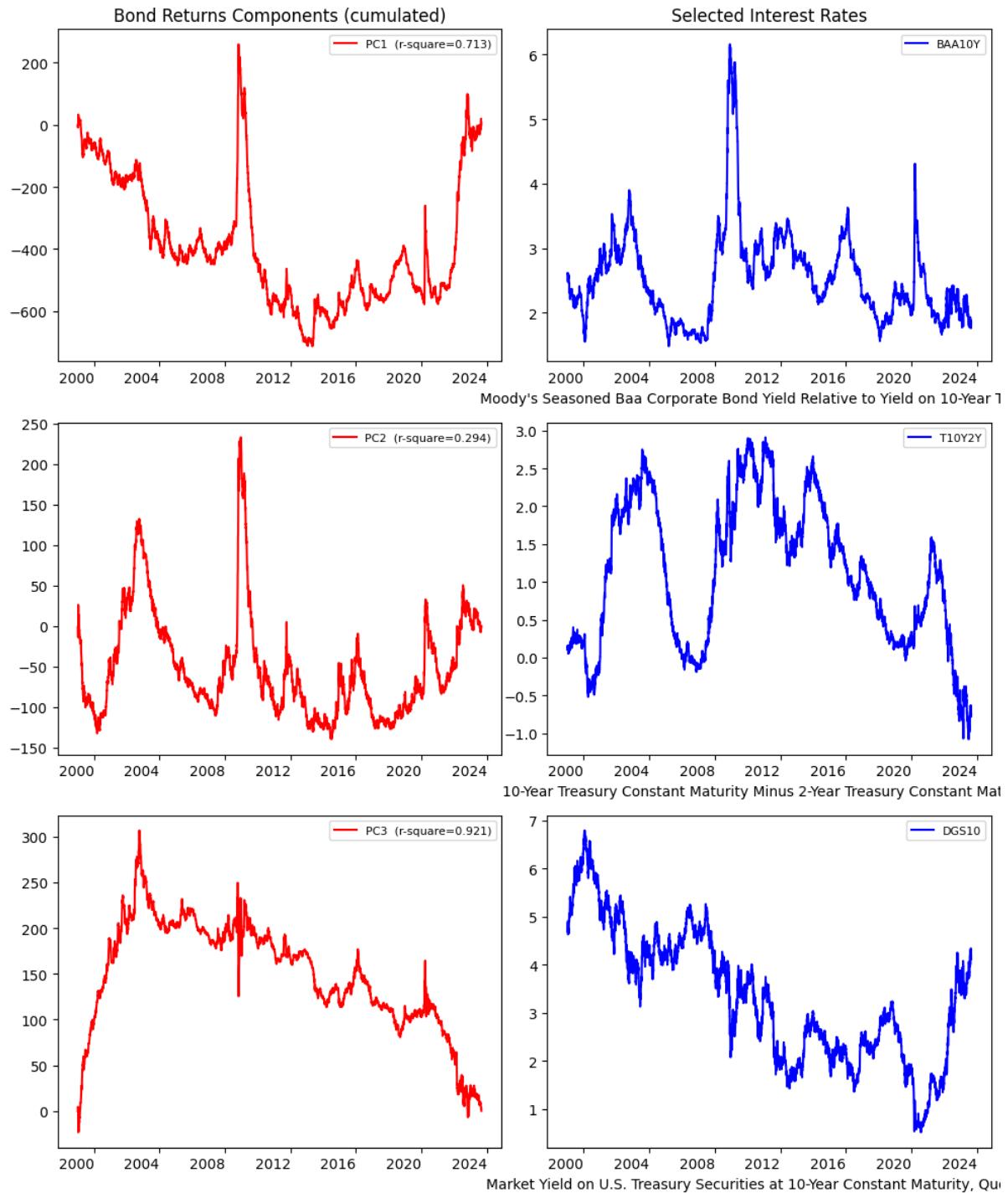
(continues on next page)

(continued from previous page)

1	0.5263
2	0.5016
3	0.0499

```
# Plot of cumulative components and interest rates levels
fig = plt.figure(figsize=(10, 12), num=1, clear=True)
for isub, col in enumerate(factors.columns):
    ax = fig.add_subplot(3, 2, (col * 2) - 1)
    flip = -1 if col == 3 else 1
    (flip*factors[col]).cumsum().plot(ax=ax, color='r')
    ax.legend([f"PC{col} (r-square={rsg[isub]:.3})"], fontsize=8)
    ax.xaxis.set_tick_params(rotation=0)
    if not isub:
        ax.set_title('Bond Returns Components (cumulated)')

    ax = fig.add_subplot(3, 2, (isub + 1) * 2)
    rates.iloc[:, isub].plot(ax=ax, color='b')
    ax.legend([f"{rates.columns[isub]}"], fontsize=8)
    ax.xaxis.set_tick_params(rotation=0)
    if not isub:
        ax.set_title('Selected Interest Rates')
    ax.set_xlabel(alf.header(rates.columns[isub])[:80])
plt.savefig(imgdir / 'components.jpg')
plt.tight_layout()
```





## OPTIONS PRICING

### UNDER CONSTRUCTION

- Binomial pricing
- Simulation
- the Greeks

```
from pandas import DataFrame, Series
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm
from datetime import datetime
from typing import List, Tuple, Any, Dict
from finds.database import SQL, RedisDB
from finds.structured import Stocks, Signals, SignalsFrame
from finds.busday import BusDay
from finds.misc import Show
from secret import credentials, paths, CRSP_DATE
show = Show(ndigits=4, latex=None)
pd.set_option('display.max_rows', None)
VERBOSE = 0
#%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)
signals = Signals(user, verbose=VERBOSE)
outdir = paths['images'] / 'glm'
```



## MARKET MICROSTRUCTURE

### UNDER CONSTRUCTION

- NYSE Daily TAQ tick data
- Intraday spreads: quoted, effective, price impact, realized; Lee-Ready tick test
- Intraday volatility: variance ratio, Parkinsons (HL) and Klass-Garman (OHLC) methods

```
import numpy as np
import pandas as pd
import time
from pandas import DataFrame, Series
from matplotlib import colors
import matplotlib.pyplot as plt
from tqdm import tqdm
import multiprocessing
from finds.database.sql import SQL
from finds.database.redisdb import RedisDB
from finds.structured.crsp import CRSP
from finds.busday import BusDay
from finds.readers.taq import opentaq, itertaq, bin_trades, bin_quotes, TAQ
from finds.plots import plot_time
from finds.misc.show import row_formatted, Show
from finds.filters import weighted_average
from finds.finance import Volatility
from finds.unstructured.store import Store
from secret import credentials, paths

VERBOSE = 0
show = Show(ndigits=4, latex=None)
%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)
imgdir = paths['images']
taqdir = paths['taq']
storedir = paths['downloads'] / 'store'
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]
```

Last FamaFrench Date 2023-06-30 00:00:00

### Loop over stocks in TAQ daily

```
"""Non-parallel code
# filter by usual CRSP investment universe
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 Tag trades, nbbo and master files
    for ct, cq, mast in itertaq(trades,
                                 quotes,
                                 master,
                                 cusips=univ['ncusip'],
                                 open_t=open_t,
                                 close_t=None,
                                 verbose=VERBOSE):
        header = {'date':date}
        header.update(univ.loc[mast['CUSIP'][:8],
                               ['permno', 'decile', 'exchcd', 'siccd']])
        header.update(mast[['Symbol', 'Round_Lot']])
        store[header['Symbol']] = dict(header=header, ct=ct, cq=cq, mast=mast)
        quotes.close()
        trades.close()

    # combine into large dataframe
    daily_df = DataFrame(daily_all)
    bins_df = {k: DataFrame(bins_all[k]) for k in bins_all.keys()}
"""

```

## INTRADAY LIQUIDITY

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
bin_keys = ['effective', 'realized', 'impact',
            'quoted', 'volume', 'offersize', 'bidsize',
            'ret', 'retq', 'counts']
def intraday(date):
    """Compute intraday liquidity for one date, parallelizable"""
    store = Store(storedir / str(date), verbose=VERBOSE)
    symbols = sorted(store)
    shareclass = []
    daily_all = []
    bins_all = {k: [] for k in bin_keys}
    for num, symbol in enumerate(symbols):
        if num >= max_num:
            return (daily_all, bins_all)
        header, ct, cq, mast = store[symbol]

        # 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)

```

(continues on next page)

(continued from previous page)

```

daily[f"tvar{v}{u}"] = bt['ret'].var(ddof=0) * len(bt)
daily[f"tvarHL{v}{u}"] = ((Volatility.HL(bt['maxtrade'],
                                         bt['mintrade']))**2)
                                         * len(bt))
daily[f"tvarOHLC{v}{u}"] = ((Volatility.OHLC(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}"] = ((Volatility.HL(bq['maxmid'],
                                         bq['minmid']))**2)
                                         * len(bq))
daily[f"qvarOHLC{v}{u}"] = ((Volatility.OHLC(bq['firstmid'],
                                                bq['maxmid'],
                                                bq['minmid'],
                                                bq['mid']))**2)
                                         * len(bq))
daily[f"tunch{v}{u}"] = np.mean(np.abs(bt['ret']) < EPSILON)
daily[f"qunch{v}{u}"] = np.mean(np.abs(bq['retq']) < EPSILON)
daily[f"tzero{v}{u}"] = np.mean(bt['counts'] == 0)

# Collect final (i.e. 5 minute bins) bt and bq intraday series
df = bq.join(bt, how='left')
for s in ['effective', 'realized', 'impact', 'quoted']:
    bins_all[s].append({**header,
                         **(df[s]/df['mid']).to_dict()})
for s in ['volume', 'offersize', 'bidsize', 'ret', 'retq', 'counts']:
    bins_all[s].append({**header,
                         **df[s].to_dict()})

# Collect daily means
daily.update(df[['bidsize', 'offersize', 'quoted', 'mid']].mean())
daily.update(df[['volume', 'counts']].sum())
daily.update(weighted_average(df[['effective', 'impact', 'realized',
                                    'vwap', 'volume']],
                               weights='volume'))
daily_all.append(daily)

```

```

# Run intraday jobs in parallel pool
p = multiprocessing.Pool()
data = p.map(intraday, dates)
p.close()

```

```

# Combine data
daily_df = pd.concat([DataFrame(data[j][0]) for j in range(len(data))],
                     ignore_index=True)
bins_df = {k: pd.concat([DataFrame(data[j][1][k]) for j in range(len(data))],
                     ignore_index=True)
           for k in bin_keys}

```

```

# Initialize folder to store data
store = Store(paths['scratch'])

```

```
# Save extracted data
store.dump(daily_df, 'tick.daily')
store.dump(bins_df, 'tick.bins')
#store.dump(shareclass, 'tick.shrccls')
```

```
# Fetch extracted data
daily_df = store.load('tick.daily')
bins_df = store.load('tick.bins')
```

### Daily average of liquidity measures, by market cap

```
# 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'])
```

```
# 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)
```

```
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]
```

(continues on next page)

(continued from previous page)

```
formats.update({k: '{:.4f}' for k in result.columns})
results.update(result)
```

```
rel = [s.capitalize() + ' (% price)' for s in spr]
daily_df[rel] = daily_df[spr].div(daily_df['mid'], axis=0) # scale spreads
result = 100*groupby[rel].mean()
formats.update({k: '{:.4f}' for k in result.columns})
results.update(result)
```

```
# summarize large and small trade effective spreads
spr = ['large_spread', 'small_spread']
result = 100*groupby[spr].mean()
result.columns = ['Large trade (% spread)', 'Small trade (% spread)']
formats.update({k: '{:.4f}' for k in result.columns})
results.update(result)
```

```
spr = ['large_trades', 'small_trades']
result = groupby[spr].mean()
result.columns = ['Large trade (# trades)', 'Small trade (# trades)']
formats.update({k: '{:.0f}' for k in result.columns})
results.update(result)
```

```
spr = ['large_volume', 'small_volume']
result = groupby[spr].mean()
result.columns = ['Large trade (avg volume)', 'Small trade (avg volume)']
formats.update({k: '{:.0f}' for k in result.columns})
results.update(result)
```

```
# display table of results
show(row_formatted(DataFrame(results).T, formats),
      caption="Average Liquidity by Market Cap")
```

Size	large	medium	small	tiny
Average Liquidity by Market Cap				
Number of Stock/Days	2063	2572	4297	4618
Midquote Price	128.98	64.91	27.76	7.92
VWAP	128.97	64.92	27.75	7.90
Number of trades	25178	8407	3658	837
Volume (shares)	3031056	995483	533736	254773
Volatility(trade price)	0.0145	0.0211	0.0359	0.0807
Volatility(midquote)	0.0152	0.0223	0.0355	0.0918
Volatility(HL trade price)	0.0176	0.0217	0.0335	0.0690
Volatility(HL midquote)	0.0144	0.0198	0.0288	0.0594
Volatility(OHLC trade price)	0.0184	0.0218	0.0324	0.0612
Volatility(OHLC midquote)	0.0138	0.0183	0.0255	0.0452
Offersize (lots)	8.8	9.8	11.9	13.6
Bidsize (lots)	8.9	17.0	14.1	16.5
Quoted \$ spread	0.0630	0.0672	0.0781	0.0841
Effective \$ spread	0.0379	0.0442	0.0406	0.0457
Impact \$ spread	0.0273	0.0244	0.0248	0.0186
Realized \$ spread	0.0106	0.0199	0.0158	0.0270
Quoted (% price)	0.0350	0.0834	0.2500	1.1582

(continues on next page)

(continued from previous page)

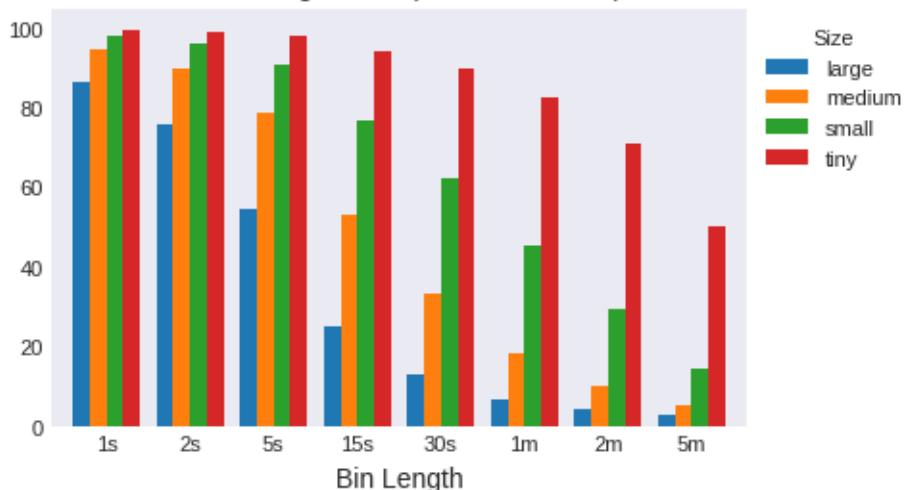
Effective (% price)	0.0200	0.0421	0.1287	0.6638
Impact (% price)	0.0165	0.0345	0.0904	0.2669
Realized (% price)	0.0036	0.0077	0.0384	0.3973
Large trade (% spread)	0.0191	0.0420	0.1310	0.6398
Small trade (% spread)	0.0190	0.0400	0.1304	0.6755
Large trade (# trades)	3133	1140	423	108
Small trade (# trades)	22045	7267	3236	729
Large trade (avg volume)	1734	1608	2635	2324
Small trade (avg volume)	61	57	71	85

```
## Summarize unchanged midquote and trade price, and zero-volume bins
def plot_helper(result, xticks, keys, legend, xlabel, title, ylim=[], figsize=(5,3), num=1, fontsize=8):
    """helper to plot bar graphs"""
    fig, ax = plt.subplots(num=num, clear=True, figsize=figsize)
    result.plot(kind='bar',
                fontsize=fontsize,
                rot=0,
                width=0.8,
                xlabel='',
                ax=ax)
    if ylim:
        ax.set_ylim(*ylim)
    ax.set_xticklabels(xticks, fontsize=fontsize)
    ax.legend(keys, loc='upper left', bbox_to_anchor=(1.0, 1.0),
              fontsize=fontsize, title=legend, title_fontsize=8)
    ax.set_xlabel(xlabel, fontsize=fontsize + 2)
    ax.set_title(title, fontsize=fontsize + 2)
    plt.subplots_adjust(right=0.8, bottom=0.15)
    plt.tight_layout()
    return ax
```

```
xticks = [f"v\{u}" for v, u in intervals]      # x-axis: bin lengths
keys = list(groupby.indices.keys())             # legend labels
```

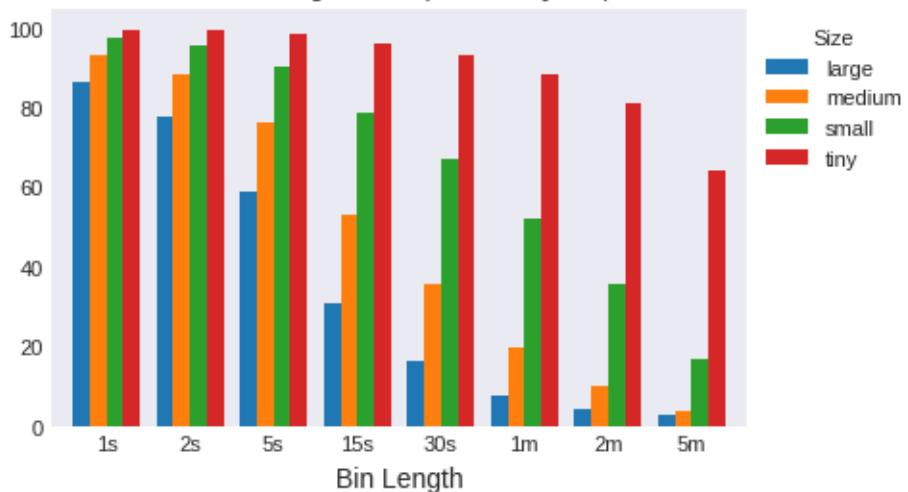
```
labels = [f"tunch\{v\}\{u\}" for v, u in intervals]
result = groupby[labels].median()*100
ax = plot_helper(result.T,
                  title="% Unchanged Bins (Last Trade Price)",
                  xticks=xticks,
                  xlabel="Bin Length",
                  keys=keys,
                  legend='Size',
                  num=1)
plt.savefig(imgdir / 'tunch.jpg')
```

% Unchanged Bins (Last Trade Price)



```
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)
plt.savefig(imgdir / 'qunch.jpg')
```

% Unchanged Bins (Last MidQuote)

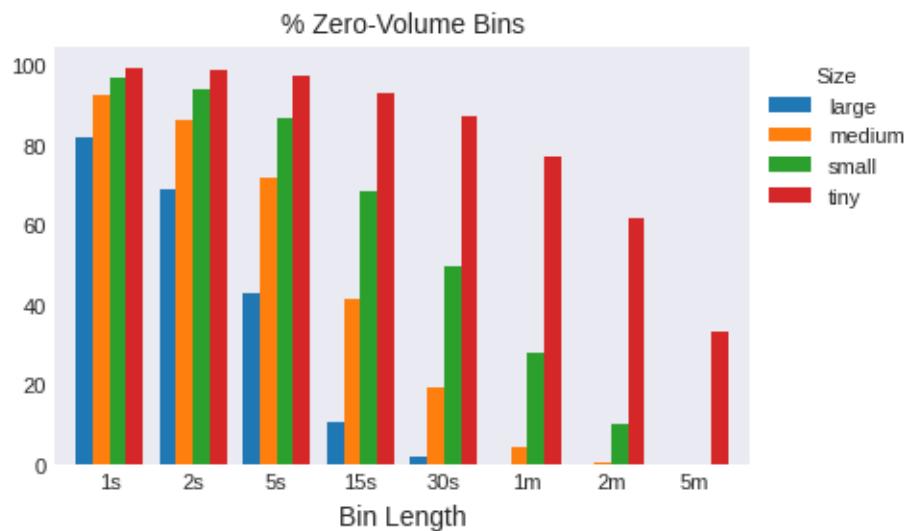


```
labels = [f"tzzero{v}{u}" for v, u in intervals]
result = groupby[labels].median()*100
ax = plot_helper(result.T,
                  title="% Zero-Volume Bins",
                  xticks=xticks,
                  xlabel="Bin Length",
```

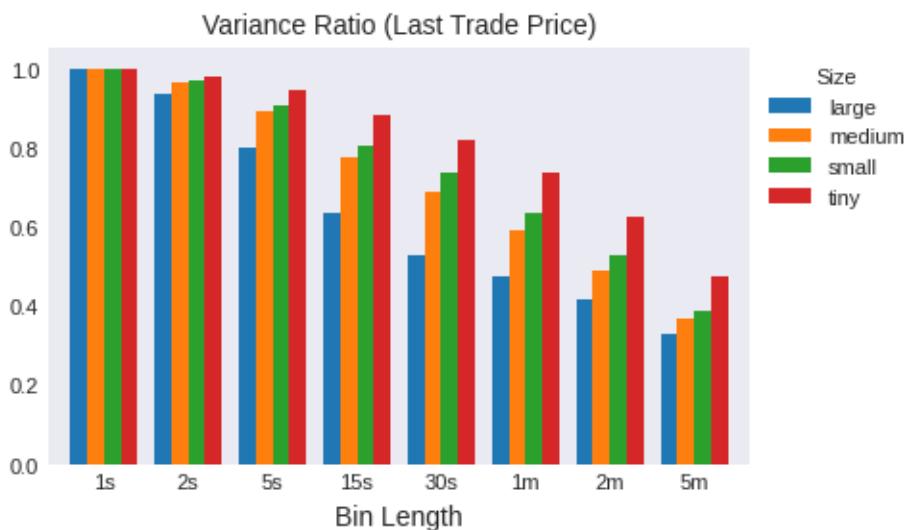
(continues on next page)

(continued from previous page)

```
        keys=keys,
        legend='Size',
        num=3)
plt.savefig(imgdir / 'tzero.jpg')
```



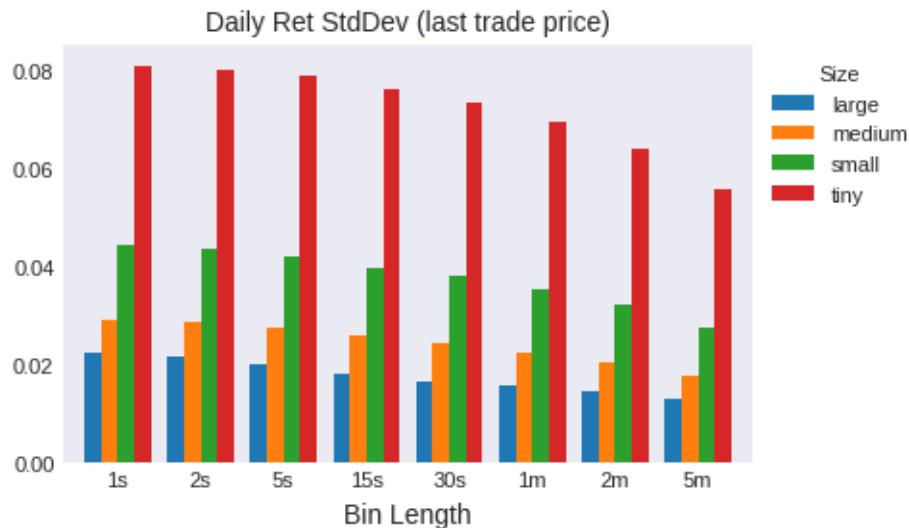
```
labels = [f"tvar{v}{u}" for v, u in intervals]
result = groupby[labels].median()
result = result.div(result['tvar1s'].values, axis=0)
ax = plot_helper(result.T,
                  title="Variance Ratio (Last Trade Price)",
                  xticks=xticks,
                  xlabel="Bin Length",
                  keys=keys,
                  legend='Size',
                  num=4)
plt.savefig(imgdir / 'tvratio.jpg')
```



```

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)
plt.savefig(imgdir / 'tstd.jpg')

```

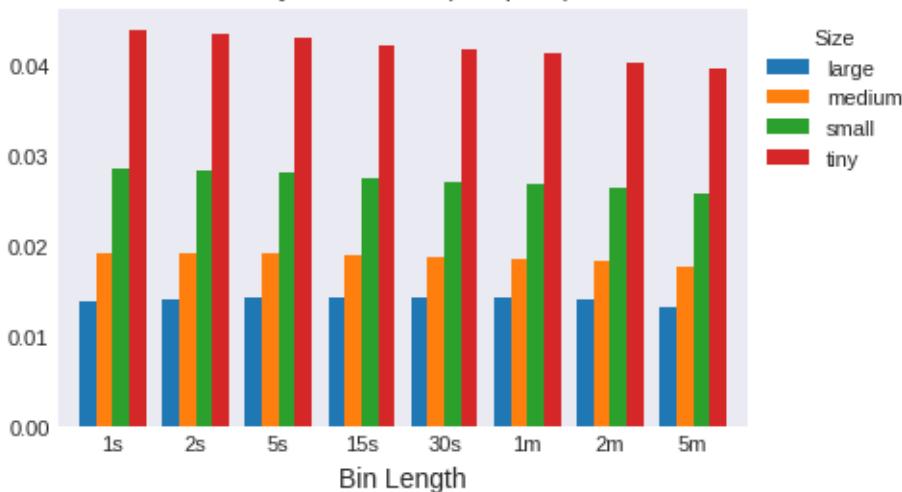


```

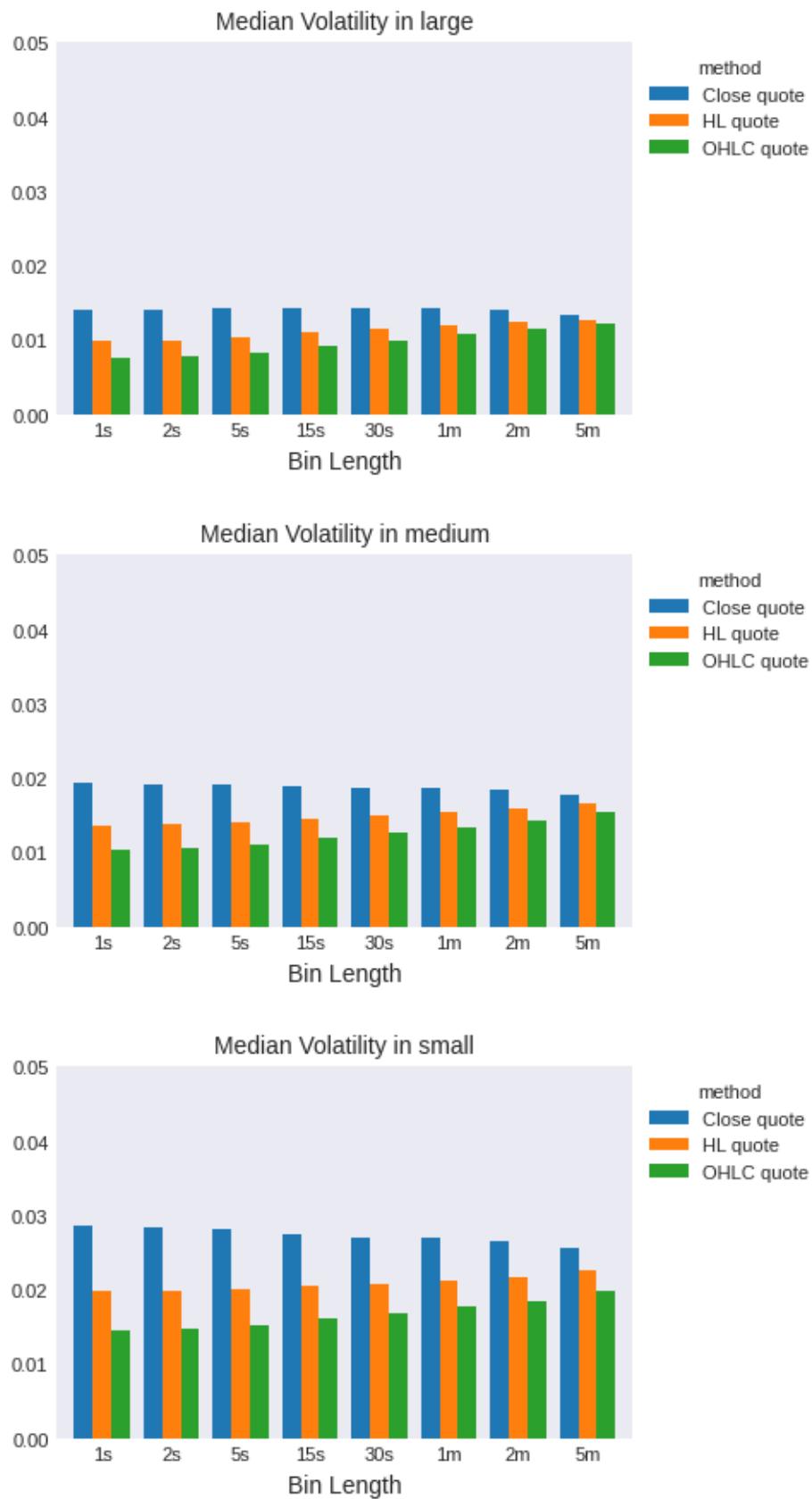
labels = [f"qvar{v}{u}" for v, u in intervals]
result = np.sqrt(groupby[labels].median())
ax = plot_helper(result.T,
                  title="Daily Ret StdDev (midquote)",
                  xticks=xticks,
                  xlabel="Bin Length",
                  keys=keys,
                  legend='Size',
                  num=6)
plt.savefig(imgdir / 'qstd.jpg')

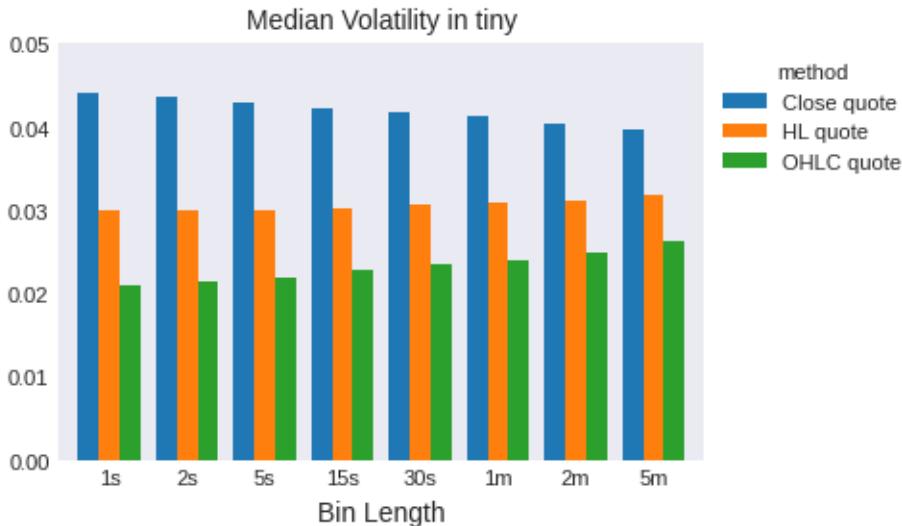
```

Daily Ret StdDev (midquote)



```
## 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]])
    result = []
    for col in [c for c in vol_df.columns if "qvar" in c]:
        if col[4] == 'H':
            m = 'HL'
        elif col[4] == 'O':
            m = 'OHLC'
        else:
            m = 'Close'
        result.append({'method': m + ' ' + {'t': 'trade', 'q': 'quote'}[col[0]],
                      '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')
    ax = plot_helper(result,
                      title="Median Volatility in " + " ".join(split_label),
                      xticks=xticks,
                      xlabel="Bin Length",
                      keys=result.columns,
                      num=1+ifig,
                      ylim=[0.0, 0.05],
                      legend='method')
    plt.savefig(imgdir / ('tick_' + "_".join(split_label) + '.jpg'))
```

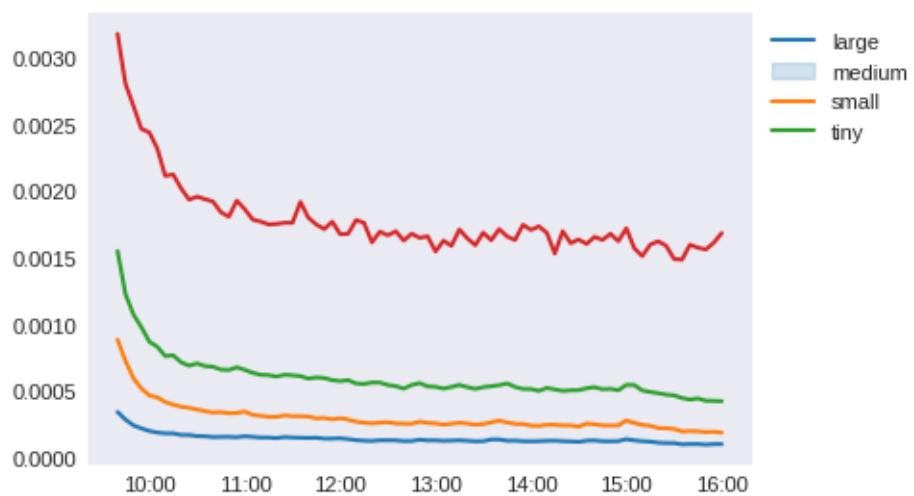




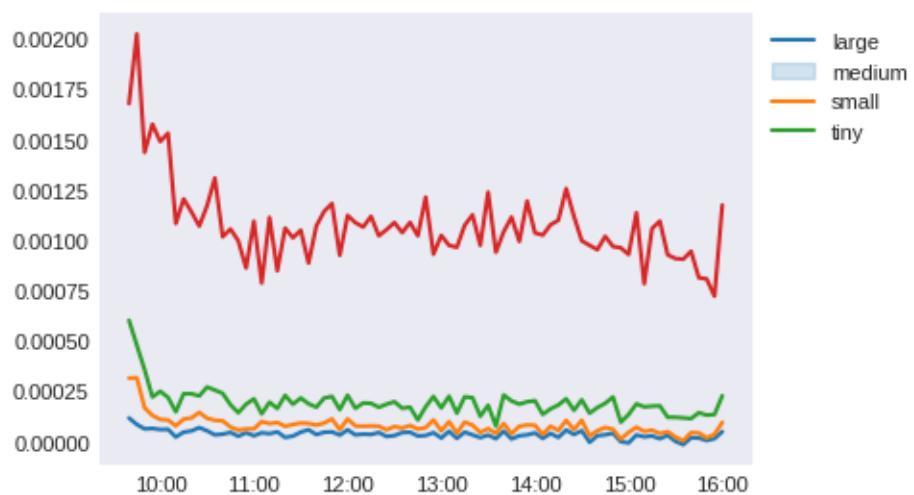
```
## Plot intraday spreads, depths and volumes
keys = ['effective', 'realized', 'impact', 'quoted',
        'volume', 'counts', 'offersize', 'bidsize']
for num, key in enumerate(keys):
    df = bins_df[key].drop(columns=['Round_Lot', 'Symbol'])
    df.index = list(zip(df['permno'], df['date']))

    # 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']) \
        .median().T
    fig, ax = plt.subplots(1, 1, num=num+1, clear=True, figsize=(5, 3))
    plot_time(df.iloc[1:], title='Median ' + key.capitalize(),
               ax=ax, fontsize=8, loc1='upper center',
               legend1=None)
    ax.legend(df.columns, loc='upper left', bbox_to_anchor=(1.0, 1.0),
              fontsize=8)
    plt.subplots_adjust(right=0.8)
    plt.tight_layout()
    plt.savefig(imgdir / (key + '.jpg'))
```

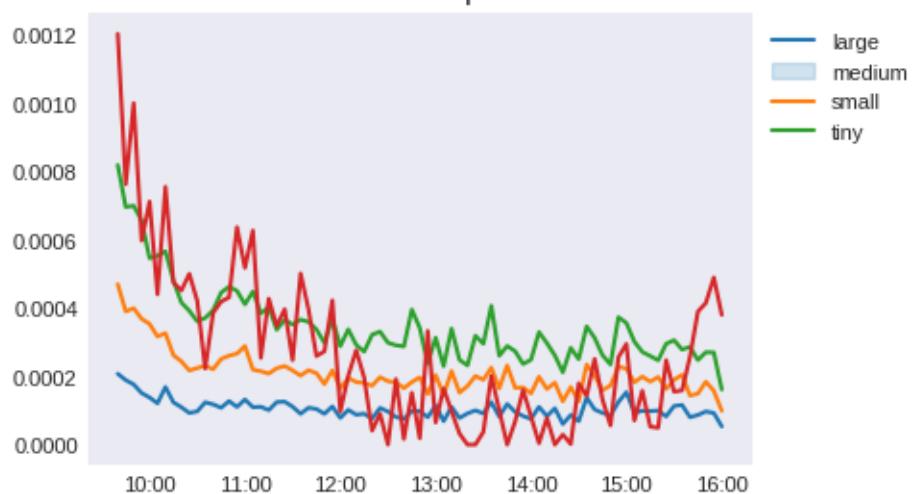
Median Effective



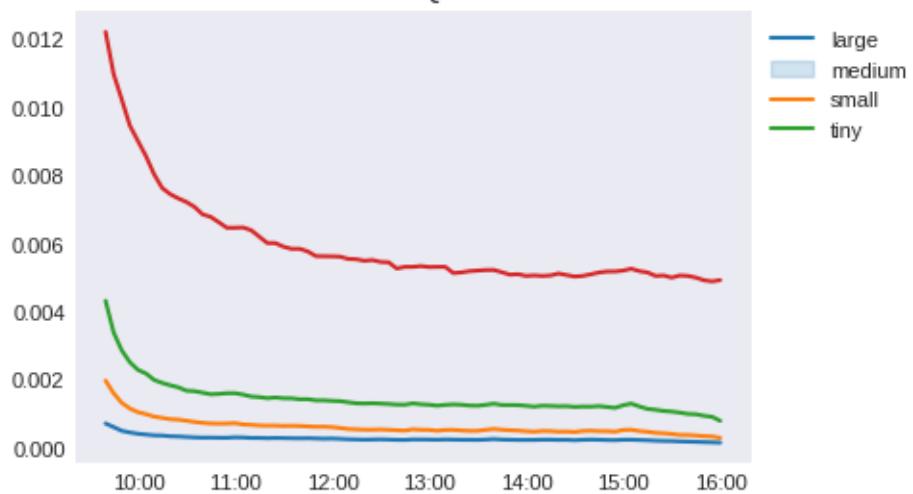
Median Realized



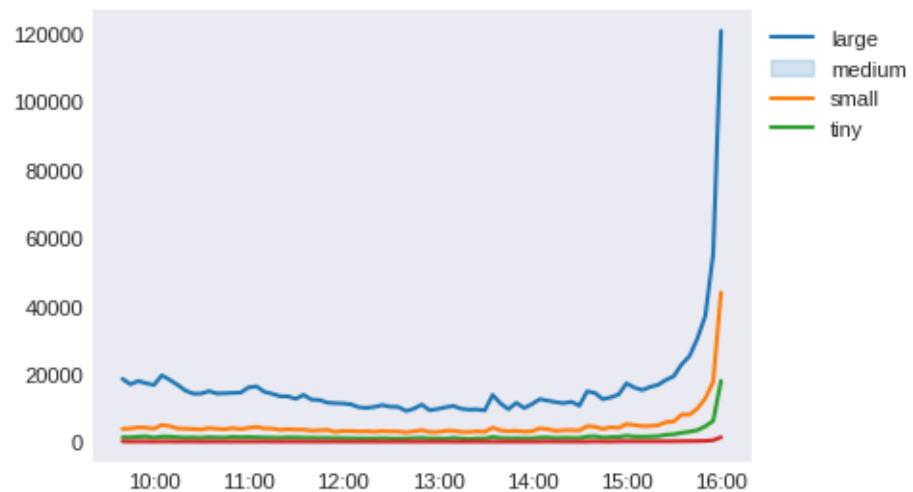
Median Impact



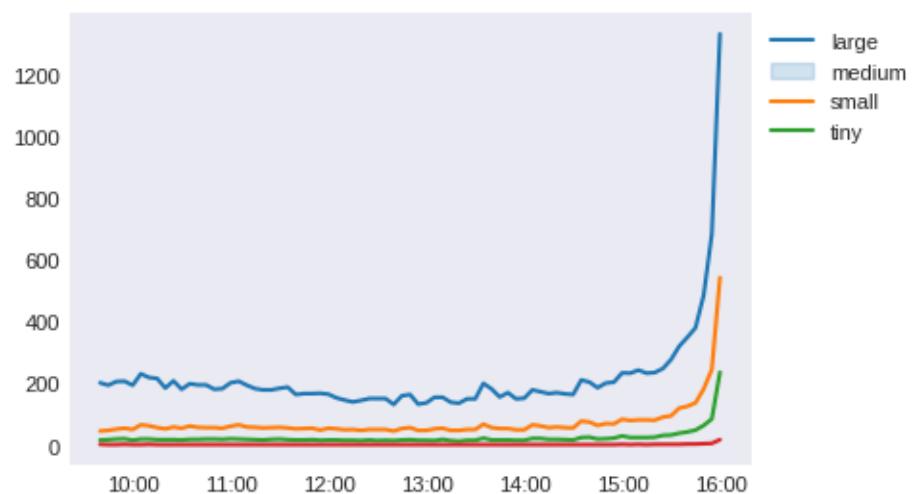
Median Quoted



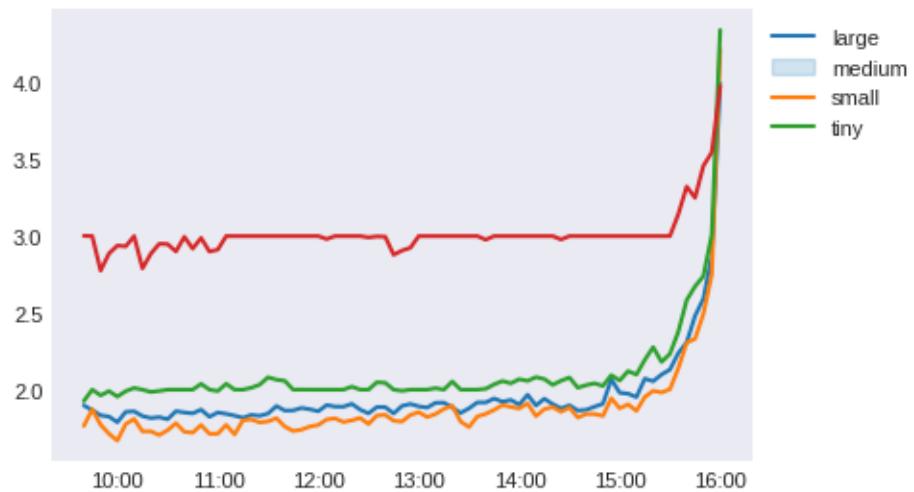
Median Volume



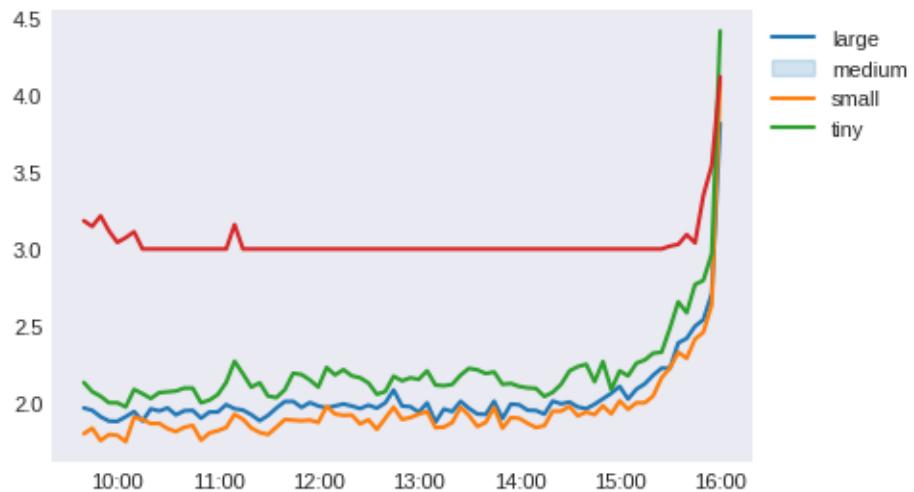
Median Counts



Median Offersize



Median Bidsize



---

CHAPTER  
**TWENTYTWO**

---

## EVENT RISK

### UNDER CONSTRUCTION

- Poisson regression; GLM's
- Earnings surprises

```
from pandas import DataFrame, Series
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm
from datetime import datetime
from typing import List, Tuple, Any, Dict
from finds.database import SQL, RedisDB
from finds.structured import Stocks, Signals, SignalsFrame
from finds.busday import BusDay
from finds.misc import Show
from secret import credentials, paths, CRSP_DATE
show = Show(ndigits=4, latex=None)
pd.set_option('display.max_rows', None)
VERBOSE = 0
#%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)
signals = Signals(user, verbose=VERBOSE)
outdir = paths['images'] / 'glm'
```

### TODO

- quarterly number of earnings surprise
- conditional on FRED-QD and number of exposures = companies

```
df = signals.read('sue')
```



---

CHAPTER  
**TWENTYTHREE**

---

## TOPIC MODELLING

### UNDER CONSTRUCTION

- Topic Models: FOMC meeting minutes text
- Matrix Decomposition: NMF, LSA, LDA, PLSI

```
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.mongodb import MongoDB
from finds.unstructured import Unstructured
from finds.unstructured.store import Store
from finds.readers.fomcreader import FOMCReader
from finds.readers.alfred import Alfred
from secret import credentials, paths
# %matplotlib qt
VERBOSE = 1      # 0
SHOW = dict(ndigits=4, latex=True)  # None
```

```
mongodb = MongoDB(**credentials['mongodb'])
fomc = Unstructured(mongodb, 'FOMC')
imgdir = paths['images'] / 'fomc'
```

```
{'host': 'omen3080', 'version': '5.0.20', 'process': 'mongod', 'pid': 4153196,
 'uptime': 931.0, 'uptimeMillis': 931239, 'uptimeEstimate': 931, 'localTime': _
 _datetime.datetime(2023, 8, 31, 18, 13, 46, 746000), 'asserts': {'regular': 0,
 'warning': 0, 'msg': 0, 'user': 19, 'tripwire': 0, 'rollovers': 0}, 'catalogStats':
 {'collections': 7, 'capped': 0, 'timeseries': 0, 'views': 0,
 'internalCollections': 3, 'internalViews': 0}, 'connections': {'current': 6,
 'available': 51194, 'totalCreated': 8, 'active': 3, 'threaded': 6,
 'exhaustIsMaster': 0, 'exhaustHello': 1, 'awaitingTopologyChanges': 2},
 'electionMetrics': {'stepUpCmd': {'called': 0, 'successful': 0},
 'priorityTakeover': {'called': 0, 'successful': 0}, 'catchUpTakeover': {'called':
 0, 'successful': 0}, 'electionTimeout': {'called': 0, 'successful': 0},
 'freezeTimeout': {'called': 0, 'successful': 0}, 'numStepDownsCausedByHigherTerm': 0}
```

(continues on next page)

(continued from previous page)

```
↳: 0, 'numCatchUps': 0, 'numCatchUpsSucceeded': 0, 'numCatchUpsAlreadyCaughtUp': 0, 'numCatchUpsSkipped': 0, 'numCatchUpsTimedOut': 0, 'numCatchUpsFailedWithError': 0, 'numCatchUpsFailedWithReplSetAbortPrimaryCatchUpCmd': 0, 'averageCatchUpOps': 0.0}, 'extra_info': {'note': 'fields vary by platform', 'user_time_us': 3076896, 'system_time_us': 740900, 'maximum_resident_set_kb': 137284, 'input_blocks': 131336, 'output_blocks': 8056, 'page_reclaims': 22026, 'page_faults': 622, 'voluntary_context_switches': 34419, 'involuntary_context_switches': 250}, 'flowControl': {'enabled': True, 'targetRateLimit': 10000000000, 'timeAcquiringMicros': 51, 'locksPerKiloOp': 0.0, 'sustainerRate': 0, 'isLagged': False, 'isLaggedCount': 0, 'isLaggedTimeMicros': 0}, 'freeMonitoring': {'state': 'undecided'}, 'globalLock': {'totalTime': 931307000, 'currentQueue': {'total': 0, 'readers': 0, 'writers': 0}, 'activeClients': {'total': 0, 'readers': 0, 'writers': 0}}, 'indexBulkBuilder': {'count': 0, 'resumed': 0, 'filesOpenedForExternalSort': 0, 'filesClosedForExternalSort': 0}, 'indexStats': {'count': 8, 'features': {'2d': {'count': 0, 'accesses': 0}, '2dsphere': {'count': 0, 'accesses': 0}, 'collation': {'count': 0, 'accesses': 0}, 'compound': {'count': 0, 'accesses': 0}, 'hashed': {'count': 0, 'accesses': 0}, 'id': {'count': 7, 'accesses': 0}, 'normal': {'count': 1, 'accesses': 0}, 'partial': {'count': 0, 'accesses': 0}, 'single': {'count': 1, 'accesses': 0}, 'sparse': {'count': 0, 'accesses': 0}, 'text': {'count': 0, 'accesses': 0}, 'ttl': {'count': 0, 'accesses': 0}, 'unique': {'count': 1, 'accesses': 0}, 'wildcard': {'count': 0, 'accesses': 0}}}, 'locks': {'ParallelBatchWriterMode': {'acquireCount': {'r': 37}}, 'FeatureCompatibilityVersion': {'acquireCount': {'r': 1887, 'w': 33}}, 'ReplicationStateTransition': {'acquireCount': {'w': 972}}, 'Global': {'acquireCount': {'r': 1887, 'w': 28, 'W': 5}}, 'Database': {'acquireCount': {'r': 8, 'w': 27, 'R': 1, 'W': 1}}, 'Collection': {'acquireCount': {'r': 16, 'w': 25, 'W': 2}}, 'Mutex': {'acquireCount': {'r': 53}}, 'logicalSessionRecordCache': {'activeSessionsCount': 1, 'sessionsCollectionJobCount': 4, 'lastSessionsCollectionJobDurationMillis': 0, 'lastSessionsCollectionJobTimestamp': datetime.datetime(2023, 8, 31, 18, 13, 15, 977000), 'lastSessionsCollectionJobEntriesRefreshed': 0, 'lastSessionsCollectionJobEntriesEnded': 0, 'lastSessionsCollectionJobCursorsClosed': 0, 'transactionReaperJobCount': 4, 'lastTransactionReaperJobDurationMillis': 0, 'lastTransactionReaperJobTimestamp': datetime.datetime(2023, 8, 31, 18, 13, 15, 977000), 'lastTransactionReaperJobEntriesCleanedUp': 0, 'sessionCatalogSize': 0}, 'network': {'bytesIn': 276721, 'bytesOut': 9479574, 'physicalBytesIn': 264527, 'physicalBytesOut': 9479574, 'numSlowDNSOperations': 0, 'numSlowSSLOperations': 0, 'numRequests': 204, 'tcpFastOpen': {'kernelSetting': 1, 'serverSupported': True, 'clientSupported': True, 'accepted': 0}, 'compression': {'snappy': {'compressor': {'bytesIn': 0, 'bytesOut': 0}, 'decompressor': {'bytesIn': 0, 'bytesOut': 0}}, 'zstd': {'compressor': {'bytesIn': 0, 'bytesOut': 0}, 'decompressor': {'bytesIn': 0, 'bytesOut': 0}}, 'zlib': {'compressor': {'bytesIn': 0, 'bytesOut': 0}, 'decompressor': {'bytesIn': 0, 'bytesOut': 0}}}, 'serviceExecutors': {'passthrough': {'threadsRunning': 6, 'clientsInTotal': 6, 'clientsRunning': 6, 'clientsWaitingForData': 0}, 'fixed': {'threadsRunning': 1, 'clientsInTotal': 0, 'clientsRunning': 0, 'clientsWaitingForData': 0}, 'opLatencies': {'reads': {'latency': 17995, 'ops': 3}, 'writes': {'latency': 2074, 'ops': 6}, 'commands': {'latency': 12921, 'ops': 192}, 'transactions': {'latency': 0, 'ops': 0}, 'opcounters': {'insert': 6, 'query': 5, 'update': 2, 'delete': 0, 'getmore': 1, 'command': 204}, 'opcountersRepl': {'insert': 0, 'query': 0, 'update': 0, 'delete': 0, 'getmore': 0, 'command': 0}, 'readConcernCounters': {'nonTransactionOps': {'none': 2, 'noneInfo': {'CWRC': {'local': 0, 'available': 0, 'majority': 0}, 'implicitDefault': {'local': 2, 'available': 0}}, 'available': 0}, 'local': 0, 'available': 0, 'majority': 0, 'snapshot': {}}}}
```

(continues on next page)

(continued from previous page)

```

↳ 'withClusterTime': 0, 'withoutClusterTime': 0}, 'linearizable': 0},
↳ 'transactionOps': {'none': 0, 'noneInfo': {'CWRC': {'local': 0, 'majority': 0},
↳ 'implicitDefault': {'local': 0}}, 'local': 0, 'majority': 0, 'snapshot': {
↳ 'withClusterTime': 0, 'withoutClusterTime': 0}}}, 'scramCache': {'SCRAM-SHA-1': {
↳ 'count': 0, 'hits': 0, 'misses': 0}, 'SCRAM-SHA-256': {'count': 0, 'hits': 0,
↳ 'misses': 0}}, 'security': {'authentication': {'saslSupportedMechsReceived': 0,
↳ 'mechanisms': {'MONGODB-X509': {'speculativeAuthenticate': {'received': 0,
↳ 'successful': 0}, 'clusterAuthenticate': {'received': 0, 'successful': 0},
↳ 'authenticate': {'received': 0, 'successful': 0}}, 'SCRAM-SHA-1': {
↳ 'speculativeAuthenticate': {'received': 0, 'successful': 0}, 'clusterAuthenticate':
↳ {'received': 0, 'successful': 0}, 'authenticate': {'received': 0, 'successful':
↳ 0}}, 'SCRAM-SHA-256': {'speculativeAuthenticate': {'received': 0, 'successful':
↳ 0}, 'clusterAuthenticate': {'received': 0, 'successful': 0}, 'authenticate': {
↳ 'received': 0, 'successful': 0}}}}}, 'storageEngine': {'name': 'wiredTiger',
↳ 'supportsCommittedReads': True, 'oldestRequiredTimestampForCrashRecovery':_
↳ Timestamp(0, 0), 'supportsPendingDrops': True, 'dropPendingIdents': 0,
↳ 'supportsSnapshotReadConcern': True, 'readOnly': False, 'persistent': True,
↳ 'backupCursorOpen': False, 'supportsResumableIndexBuilds': True}, 'tcmalloc': {
↳ 'generic': {'current_allocated_bytes': 108290160, 'heap_size': 151408640},
↳ 'tcmalloc': {'pageheap_free_bytes': 41582592, 'pageheap_unmapped_bytes': 0, 'max_
↳ total_thread_cache_bytes': 1073741824, 'current_total_thread_cache_bytes':_
↳ 1049944, 'total_free_bytes': 1535888, 'central_cache_free_bytes': 329656,
↳ 'transfer_cache_free_bytes': 156288, 'thread_cache_free_bytes': 1049944,
↳ 'aggressive_memory_decommit': 0, 'pageheap_committed_bytes': 151408640,
↳ 'pageheap_scavenge_count': 0, 'pageheap_commit_count': 70, 'pageheap_total_
↳ commit_bytes': 151408640, 'pageheap_decommit_count': 0, 'pageheap_total_decommit_
↳ bytes': 0, 'pageheap_reserve_count': 70, 'pageheap_total_reserve_bytes':_
↳ 151408640, 'spinlock_total_delay_ns': 0, 'release_rate': 1.0, 'formattedString':_
↳ '-----\nMALLOC: 108290736 ( 103.
↳ 3 MiB) Bytes in use by application\nMALLOC: + 41582592 ( 39.7 MiB) Bytes_
↳ in page heap freelist\nMALLOC: + 329656 ( 0.3 MiB) Bytes in central_
↳ cache freelist\nMALLOC: + 156288 ( 0.1 MiB) Bytes in transfer cache_
↳ freelist\nMALLOC: + 1049368 ( 1.0 MiB) Bytes in thread cache freelists\
↳ nMALLOC: + 4849664 ( 4.6 MiB) Bytes in malloc metadata\nMALLOC: -----
↳ -----nMALLOC: = 156258304 ( 149.0 MiB) Actual memory used (physical + swap)\\
↳ nMALLOC: + 0 ( 0.0 MiB) Bytes released to OS (aka unmapped)\\
↳ nMALLOC: -----nMALLOC: = 156258304 ( 149.0 MiB) Virtual address_
↳ space used\nMALLOC:\nMALLOC: 1007 Spans in use\nMALLOC: -
↳ 36 Thread heaps in use\nMALLOC: 4096
↳ Tcmalloc page size\n-----\nCall_
↳ ReleaseFreeMemory() to release freelist memory to the OS (via madvise()).\nBytes_
↳ released to the OS take up virtual address space but no physical memory.\n'},
↳ 'tenantMigrations': {'currentMigrationsDonating': 0, 'currentMigrationsReceiving':
↳ 0, 'totalSuccessfulMigrationsDonated': 0, 'totalSuccessfulMigrationsReceived':
↳ 0, 'totalFailedMigrationsDonated': 0, 'totalFailedMigrationsReceived': 0},
↳ 'trafficRecording': {'running': False}, 'transactions': {'retriedCommandsCount':_
↳ 0, 'retriedStatementsCount': 0, 'transactionsCollectionWriteCount': 0,
↳ 'currentActive': 0, 'currentInactive': 0, 'currentOpen': 0, 'totalAborted': 0,
↳ 'totalCommitted': 0, 'totalStarted': 0, 'totalPrepared': 0,
↳ 'totalPreparedThenCommitted': 0, 'totalPreparedThenAborted': 0, 'currentPrepared':
↳ 0}, 'transportSecurity': {'1.0': 0, '1.1': 0, '1.2': 0, '1.3': 0, 'unknown':_
↳ 0}, 'twoPhaseCommitCoordinator': {'totalCreated': 0, 'totalStartedTwoPhaseCommit':
↳ 0, 'totalAbortedTwoPhaseCommit': 0, 'totalCommittedTwoPhaseCommit': 0,
↳ 'currentInSteps': {'writingParticipantList': 0, 'waitingForVotes': 0,
↳ 'writingDecision': 0, 'waitingForDecisionAcks': 0, 'deletingCoordinatorDoc': 0}},
↳ 'wiredTiger': {'uri': 'statistics:', 'block_manager': {'block cache cached'}
```

(continues on next page)

(continued from previous page)

```
↳ 'blocks updated': 0, 'block cache cached bytes updated': 0, 'block cache evicted'_
↳ 'blocks': 0, 'block cache file size causing bypass': 0, 'block cache lookups': 0,_
↳ 'block cache number of blocks not evicted due to overhead': 0, 'block cache_
↳ number of bypasses because no-write-allocate setting was on': 0, 'block cache_
↳ number of bypasses due to overhead on put': 0, 'block cache number of bypasses'_
↳ 'on get': 0, 'block cache number of bypasses on put because file is too small': 0,_
↳ 'block cache number of eviction passes': 0, 'block cache number of hits'_
↳ 'including existence checks': 0, 'block cache number of misses including'_
↳ 'existence checks': 0, 'block cache number of put bypasses on checkpoint I/O': 0,_
↳ 'block cache removed blocks': 0, 'block cache total blocks': 0, 'block cache_
↳ total blocks inserted on read path': 0, 'block cache total blocks inserted on'_
↳ 'write path': 0, 'block cache total bytes': 0, 'block cache total bytes inserted'_
↳ 'on read path': 0, 'block cache total bytes inserted on write path': 0, 'blocks'_
↳ 'pre-loaded': 107, 'blocks read': 154, 'blocks written': 126, 'bytes read':_
↳ 4685824, 'bytes read via memory map API': 0, 'bytes read via system call API': 0,_
↳ 'bytes written': 1167360, 'bytes written for checkpoint': 1167360, 'bytes'_
↳ 'written via memory map API': 0, 'bytes written via system call API': 0, 'mapped'_
↳ 'blocks read': 0, 'mapped bytes read': 0, 'number of times the file was remapped'_
↳ 'because it changed size via fallocate or truncate': 0, 'number of times the'_
↳ 'region was remapped via write': 0}, 'cache': {'application threads page read'_
↳ 'from disk to cache count': 105, 'application threads page read from disk to'_
↳ 'cache time (usecs)': 11367, 'application threads page write from cache to disk'_
↳ 'count': 62, 'application threads page write from cache to disk time (usecs)':_
↳ 20803, 'bytes allocated for updates': 307071, 'bytes belonging to page images in'_
↳ 'the cache': 9918765, 'bytes belonging to the history store table in the cache':_
↳ 588, 'bytes currently in the cache': 10253408, 'bytes dirty in the cache'_
↳ 'cumulative': 986248, 'bytes not belonging to page images in the cache': 334643,_
↳ 'bytes read into cache': 9184042, 'bytes written from cache': 907453, 'cache'_
↳ 'overflow score': 0, 'checkpoint blocked page eviction': 0, 'checkpoint of'_
↳ 'history store file blocked non-history store page eviction': 0, 'eviction calls'_
↳ 'to get a page': 29, 'eviction calls to get a page found queue empty': 26,_
↳ 'eviction calls to get a page found queue empty after locking': 0, 'eviction'_
↳ 'currently operating in aggressive mode': 0, 'eviction empty score': 0, 'eviction'_
↳ 'gave up due to detecting an out of order on disk value behind the last update on'_
↳ 'the chain': 0, 'eviction gave up due to detecting an out of order tombstone'_
↳ 'ahead of the selected on disk update': 0, 'eviction gave up due to detecting an'_
↳ 'out of order tombstone ahead of the selected on disk update after validating the'_
↳ 'update chain': 0, 'eviction gave up due to detecting out of order timestamps on'_
↳ 'the update chain after the selected on disk update': 0, 'eviction gave up due to'_
↳ 'needing to remove a record from the history store but checkpoint is running': 0,_
↳ 'eviction passes of a file': 0, 'eviction server candidate queue empty when'_
↳ 'topping up': 0, 'eviction server candidate queue not empty when topping up': 0,_
↳ 'eviction server evicting pages': 0, 'eviction server slept, because we did not'_
↳ 'make progress with eviction': 3, 'eviction server unable to reach eviction goal'_
↳ ': 0, 'eviction server waiting for a leaf page': 0, 'eviction state': 64,_
↳ 'eviction walk most recent sleeps for checkpoint handle gathering': 0, 'eviction'_
↳ 'walk target pages histogram - 0-9': 0, 'eviction walk target pages histogram -'_
↳ '10-31': 0, 'eviction walk target pages histogram - 128 and higher': 0, 'eviction'_
↳ 'walk target pages histogram - 32-63': 0, 'eviction walk target pages histogram -'_
↳ '64-128': 0, 'eviction walk target pages reduced due to history store cache'_
↳ 'pressure': 0, 'eviction walk target strategy both clean and dirty pages': 0,_
↳ 'eviction walk target strategy only clean pages': 0, 'eviction walk target'_
↳ 'strategy only dirty pages': 0, 'eviction walks abandoned': 0, 'eviction walks'_
↳ 'gave up because they restarted their walk twice': 0, 'eviction walks gave up'_
↳ 'because they saw too many pages and found no candidates': 0, 'eviction walks'_
↳ 'gave up because they saw too many pages and found too few candidates': 0,
```

(continues on next page)

(continued from previous page)

```

↳ 'eviction walks reached end of tree': 0, 'eviction walks restarted': 0,
↳ 'eviction walks started from root of tree': 0, 'eviction walks started from
↳ saved location in tree': 0, 'eviction worker thread active': 4, 'eviction worker
↳ thread created': 0, 'eviction worker thread evicting pages': 3, 'eviction worker
↳ thread removed': 0, 'eviction worker thread stable number': 0, 'files with
↳ active eviction walks': 0, 'files with new eviction walks started': 0, 'force re-
↳ tuning of eviction workers once in a while': 0, 'forced eviction - history store
↳ pages failed to evict while session has history store cursor open': 0, 'forced
↳ eviction - history store pages selected while session has history store cursor
↳ open': 0, 'forced eviction - history store pages successfully evicted while
↳ session has history store cursor open': 0, 'forced eviction - pages evicted that
↳ were clean count': 0, 'forced eviction - pages evicted that were clean time
↳ (usecs)': 0, 'forced eviction - pages evicted that were dirty count': 0, 'forced
↳ eviction - pages evicted that were dirty time (usecs)': 0, 'forced eviction -
↳ pages selected because of a large number of updates to a single item': 0,
↳ 'forced eviction - pages selected because of too many deleted items count': 0,
↳ 'forced eviction - pages selected count': 0, 'forced eviction - pages selected
↳ unable to be evicted count': 0, 'forced eviction - pages selected unable to be
↳ evicted time': 0, 'hazard pointer blocked page eviction': 0, 'hazard pointer
↳ check calls': 3, 'hazard pointer check entries walked': 5, 'hazard pointer
↳ maximum array length': 3, 'history store score': 0, 'history store table insert
↳ calls': 0, 'history store table insert calls that returned restart': 0, 'history
↳ store table max on-disk size': 0, 'history store table on-disk size': 4096,
↳ 'history store table out-of-order resolved updates that lose their durable
↳ timestamp': 0, 'history store table out-of-order updates that were fixed up by
↳ reinserting with the fixed timestamp': 0, 'history store table reads': 0,
↳ 'history store table reads missed': 0, 'history store table reads requiring
↳ squashed modifies': 0, 'history store table truncation by rollback to stable to
↳ remove an unstable update': 0, 'history store table truncation by rollback to
↳ stable to remove an update': 0, 'history store table truncation to remove an
↳ update': 0, 'history store table truncation to remove range of updates due to
↳ key being removed from the data page during reconciliation': 0, 'history store
↳ table truncation to remove range of updates due to out-of-order timestamp update
↳ on data page': 0, 'history store table writes requiring squashed modifies': 0,
↳ 'in-memory page passed criteria to be split': 0, 'in-memory page splits': 0,
↳ 'internal pages evicted': 0, 'internal pages queued for eviction': 0, 'internal
↳ pages seen by eviction walk': 0, 'internal pages seen by eviction walk that are
↳ already queued': 0, 'internal pages split during eviction': 0, 'leaf pages split
↳ during eviction': 0, 'maximum bytes configured': 16098787328, 'maximum
↳ milliseconds spent at a single eviction': 0, 'maximum page size seen at eviction
↳ ': 0, 'modified pages evicted': 3, 'modified pages evicted by application threads
↳ ': 0, 'operations timed out waiting for space in cache': 0, 'overflow pages read
↳ into cache': 0, 'page split during eviction deepened the tree': 0, 'page written
↳ requiring history store records': 0, 'pages currently held in the cache': 124,
↳ 'pages evicted by application threads': 0, 'pages evicted in parallel with
↳ checkpoint': 3, 'pages queued for eviction': 0, 'pages queued for eviction post
↳ lru sorting': 0, 'pages queued for urgent eviction': 3, 'pages queued for urgent
↳ eviction during walk': 0, 'pages queued for urgent eviction from history store
↳ due to high dirty content': 0, 'pages read into cache': 116, 'pages read into
↳ cache after truncate': 7, 'pages read into cache after truncate in prepare state
↳ ': 0, 'pages removed from the ordinary queue to be queued for urgent eviction': 0,
↳ 'pages requested from the cache': 965, 'pages seen by eviction walk': 0,
↳ 'pages seen by eviction walk that are already queued': 0, 'pages selected for
↳ eviction unable to be evicted': 0, 'pages selected for eviction unable to be
↳ evicted because of active children on an internal page': 0, 'pages selected for
↳ eviction unable to be evicted because of failure in reconciliation': 0, 'pages

```

(continues on next page)

(continued from previous page)

```
↳selected for eviction unable to be evicted because of race between checkpoint
↳and out of order timestamps handling': 0, 'pages walked for eviction': 0, 'pages
↳written from cache': 65, 'pages written requiring in-memory restoration': 0,
↳'percentage overhead': 8, 'the number of times full update inserted to history
↳store': 0, 'the number of times reverse modify inserted to history store': 0,
↳'total milliseconds spent inside reentrant history store evictions in a
↳reconciliation': 0, 'tracked bytes belonging to internal pages in the cache': 0
↳12950, 'tracked bytes belonging to leaf pages in the cache': 10240458, 'tracked
↳dirty bytes in the cache': 0, 'tracked dirty pages in the cache': 0, 'unmodified
↳pages evicted': 0}, 'capacity': {'background fsync file handles considered': 0,
↳'background fsync file handles synced': 0, 'background fsync time (msecs)': 0,
↳'bytes read': 4530176, 'bytes written for checkpoint': 698601, 'bytes written
↳for eviction': 0, 'bytes written for log': 1003804288, 'bytes written total': 0
↳1004502889, 'threshold to call fsync': 0, 'time waiting due to total capacity
↳(usecs)': 0, 'time waiting during checkpoint (usecs)': 0, 'time waiting during
↳eviction (usecs)': 0, 'time waiting during logging (usecs)': 0, 'time waiting
↳during read (usecs)': 0}, 'checkpoint-cleanup': {'pages added for eviction': 3,
↳'pages removed': 0, 'pages skipped during tree walk': 0, 'pages visited': 143},
↳'connection': {'auto adjusting condition resets': 63, 'auto adjusting condition
↳wait calls': 5750, 'auto adjusting condition wait raced to update timeout and
↳skipped updating': 0, 'detected system time went backwards': 0, 'files currently
↳open': 16, 'hash bucket array size for data handles': 512, 'hash bucket array
↳size general': 512, 'memory allocations': 42516, 'memory frees': 41098, 'memory
↳re-allocations': 3128, 'pthread mutex condition wait calls': 15192, 'pthread
↳mutex shared lock read-lock calls': 15410, 'pthread mutex shared lock write-lock
↳calls': 1141, 'total fsync I/Os': 133, 'total read I/Os': 1193, 'total write I/Os
↳': 190}, 'cursor': {'Total number of entries skipped by cursor next calls': 0,
↳'Total number of entries skipped by cursor prev calls': 0, 'Total number of
↳entries skipped to position the history store cursor': 0, 'Total number of times
↳a search near has exited due to prefix config': 0, 'cached cursor count': 19,
↳'cursor bulk loaded cursor insert calls': 0, 'cursor close calls that result in
↳cache': 7705, 'cursor create calls': 126, 'cursor insert calls': 72, 'cursor
↳insert key and value bytes': 284145, 'cursor modify calls': 0, 'cursor modify
↳key and value bytes affected': 0, 'cursor modify value bytes modified': 0,
↳'cursor next calls': 828, 'cursor next calls that skip due to a globally visible
↳history store tombstone': 0, 'cursor next calls that skip greater than or equal
↳to 100 entries': 0, 'cursor next calls that skip less than 100 entries': 826,
↳'cursor operation restarted': 0, 'cursor prev calls': 33, 'cursor prev calls
↳that skip due to a globally visible history store tombstone': 0, 'cursor prev
↳calls that skip greater than or equal to 100 entries': 0, 'cursor prev calls
↳that skip less than 100 entries': 33, 'cursor remove calls': 1, 'cursor remove
↳key bytes removed': 12, 'cursor reserve calls': 0, 'cursor reset calls': 8255,
↳'cursor search calls': 314, 'cursor search history store calls': 0, 'cursor
↳search near calls': 50, 'cursor sweep buckets': 11307, 'cursor sweep cursors
↳closed': 0, 'cursor sweep cursors examined': 108, 'cursor sweeps': 1126, 'cursor
↳truncate calls': 0, 'cursor update calls': 0, 'cursor update key and value bytes
↳': 0, 'cursor update value size change': 0, 'cursors reused from cache': 7686,
↳'open cursor count': 6}, 'data-handle': {'connection data handle size': 440,
↳'connection data handles currently active': 25, 'connection sweep candidate
↳became referenced': 0, 'connection sweep dhandles closed': 0, 'connection sweep
↳dhandles removed from hash list': 17, 'connection sweep time-of-death sets': 155,
↳'connection sweeps': 93, 'connection sweeps skipped due to checkpoint gathering
↳handles': 0, 'session dhandles swept': 11, 'session sweep attempts': 335}, 'lock
↳': {'checkpoint lock acquisitions': 17, 'checkpoint lock application thread wait
↳time (usecs)': 0, 'checkpoint lock internal thread wait time (usecs)': 0,
↳'dhandle lock application thread time waiting (usecs)': 0, 'dhandle lock
```

(continues on next page)

(continued from previous page)

```

↳internal thread time waiting (usecs)': 0, 'dhandle read lock acquisitions': 3804,
↳ 'dhandle write lock acquisitions': 63, 'durable timestamp queue lock'
↳application thread time waiting (usecs)': 0, 'durable timestamp queue lock'
↳internal thread time waiting (usecs)': 0, 'durable timestamp queue read lock'
↳acquisitions': 0, 'durable timestamp queue write lock acquisitions': 0,
↳'metadata lock acquisitions': 16, 'metadata lock application thread wait time
↳(usecs)': 0, 'metadata lock internal thread wait time (usecs)': 0, 'read
↳timestamp queue lock application thread time waiting (usecs)': 0, 'read
↳timestamp queue lock internal thread time waiting (usecs)': 0, 'read timestamp
↳queue read lock acquisitions': 0, 'read timestamp queue write lock acquisitions
↳': 0, 'schema lock acquisitions': 44, 'schema lock application thread wait time
↳(usecs)': 0, 'schema lock internal thread wait time (usecs)': 0, 'table lock'
↳application thread time waiting for the table lock (usecs)': 0, 'table lock'
↳internal thread time waiting for the table lock (usecs)': 0, 'table read lock
↳acquisitions': 0, 'table write lock acquisitions': 13, 'txn global lock'
↳application thread time waiting (usecs)': 0, 'txn global lock internal thread
↳time waiting (usecs)': 0, 'txn global read lock acquisitions': 66, 'txn global
↳write lock acquisitions': 46}, 'log': {'busy returns attempting to switch slots
↳': 0, 'force archive time sleeping (usecs)': 0, 'log bytes of payload data': 135688,
↳ 'log bytes written': 142720, 'log files manually zero-filled': 0, 'log
↳ flush operations': 9308, 'log force write operations': 10335, 'log force write
↳ operations skipped': 10308, 'log records compressed': 14, 'log records not
↳ compressed': 10, 'log records too small to compress': 69, 'log release advances
↳ write LSN': 17, 'log scan operations': 6, 'log scan records requiring two reads
↳': 0, 'log server thread advances write LSN': 27, 'log server thread write LSN
↳ walk skipped': 1917, 'log sync operations': 44, 'log sync time duration (usecs)
↳': 200004, 'log sync_dir operations': 1, 'log sync_dir time duration (usecs)': 8968,
↳ 'log write operations': 93, 'logging bytes consolidated': 142208, 'maximum
↳ log file size': 104857600, 'number of pre-allocated log files to create': 2,
↳ 'pre-allocated log files not ready and missed': 1, 'pre-allocated log files
↳ prepared': 2, 'pre-allocated log files used': 0, 'records processed by log scan
↳': 15, 'slot close lost race': 0, 'slot close unbuffered waits': 0, 'slot
↳ closures': 44, 'slot join atomic update races': 0, 'slot join calls atomic
↳ updates raced': 0, 'slot join calls did not yield': 93, 'slot join calls found
↳ active slot closed': 0, 'slot join calls slept': 0, 'slot join calls yielded': 0,
↳ 'slot join found active slot closed': 0, 'slot joins yield time (usecs)': 0,
↳ 'slot transitions unable to find free slot': 0, 'slot unbuffered writes': 0,
↳ 'total in-memory size of compressed records': 279252, 'total log buffer size': 33554432,
↳ 'total size of compressed records': 127908, 'written slots coalesced': 0,
↳ 'yields waiting for previous log file close': 0}, 'perf': {'file system read
↳ latency histogram (bucket 1) - 10-49ms': 0, 'file system read latency histogram
↳ (bucket 2) - 50-99ms': 0, 'file system read latency histogram (bucket 3) - 100-
↳ 249ms': 0, 'file system read latency histogram (bucket 4) - 250-499ms': 0, 'file
↳ system read latency histogram (bucket 5) - 500-999ms': 0, 'file system read
↳ latency histogram (bucket 6) - 1000ms+'. 0, 'file system write latency histogram
↳ (bucket 1) - 10-49ms': 0, 'file system write latency histogram (bucket 2) - 50-
↳ 99ms': 0, 'file system write latency histogram (bucket 3) - 100-249ms': 0, 'file
↳ system write latency histogram (bucket 4) - 250-499ms': 0, 'file system write
↳ latency histogram (bucket 5) - 500-999ms': 0, 'file system write latency
↳ histogram (bucket 6) - 1000ms+'. 0, 'operation read latency histogram (bucket 1) -
↳ 100-249us': 0, 'operation read latency histogram (bucket 2) - 250-499us': 0,
↳ 'operation read latency histogram (bucket 3) - 500-999us': 0, 'operation read
↳ latency histogram (bucket 4) - 1000-9999us': 0, 'operation read latency
↳ histogram (bucket 5) - 10000us+'. 0, 'operation write latency histogram (bucket
↳ 1) - 100-249us': 0, 'operation write latency histogram (bucket 2) - 250-499us': 0,
↳ 'operation write latency histogram (bucket 3) - 500-999us': 0, 'operation

```

(continues on next page)

(continued from previous page)

```
↳ write latency histogram (bucket 4) - 1000-9999us': 0, 'operation write latency'
↳ histogram (bucket 5) - 10000us+: 0}, 'reconciliation': {'approximate byte size
↳ of timestamps in pages written': 0, 'approximate byte size of transaction IDs in
↳ pages written': 336, 'fast-path pages deleted': 0, 'leaf-page overflow keys': 0,
↳ 'maximum milliseconds spent in a reconciliation call': 0, 'maximum milliseconds
↳ spent in building a disk image in a reconciliation': 0, 'maximum milliseconds
↳ spent in moving updates to the history store in a reconciliation': 0, 'page
↳ reconciliation calls': 71, 'page reconciliation calls for eviction': 3, 'page
↳ ': 0, 'page reconciliation calls that resulted in values with prepared transaction metadata
↳ ': 0, 'page reconciliation calls that resulted in values with timestamps': 0,
↳ 'page reconciliation calls that resulted in values with transaction ids': 17,
↳ 'pages deleted': 9, 'pages written including an aggregated newest start durable
↳ timestamp ': 0, 'pages written including an aggregated newest stop durable
↳ timestamp ': 0, 'pages written including an aggregated newest stop timestamp ': 0,
↳ 'pages written including an aggregated newest stop transaction ID': 0, 'pages
↳ written including an aggregated newest transaction ID ': 0, 'pages written
↳ including an aggregated oldest start timestamp ': 0, 'pages written including an
↳ aggregated prepare': 0, 'pages written including at least one prepare state': 0,
↳ 'pages written including at least one start durable timestamp': 0, 'pages
↳ written including at least one start timestamp': 0, 'pages written including at
↳ least one start transaction ID': 17, 'pages written including at least one stop
↳ durable timestamp': 0, 'pages written including at least one stop timestamp': 0,
↳ 'pages written including at least one stop transaction ID': 0, 'records written
↳ including a prepare state': 0, 'records written including a start durable
↳ timestamp': 0, 'records written including a start timestamp': 0, 'records
↳ written including a start transaction ID': 42, 'records written including a stop
↳ durable timestamp': 0, 'records written including a stop timestamp': 0, 'records
↳ written including a stop transaction ID': 0, 'split bytes currently awaiting free
↳ ': 0, 'split objects currently awaiting free': 0}, 'session': {'attempts to
↳ remove a local object and the object is in use': 0, 'flush_tier operation calls
↳ ': 0, 'local objects removed': 0, 'open session count': 15, 'session query
↳ timestamp calls': 0, 'table alter failed calls': 0, 'table alter successful calls
↳ ': 0, 'table alter triggering checkpoint calls': 0, 'table alter unchanged and
↳ skipped': 0, 'table compact failed calls': 0, 'table compact failed calls due to
↳ cache pressure': 0, 'table compact running': 0, 'table compact skipped as
↳ process would not reduce file size': 0, 'table compact successful calls': 0,
↳ 'table compact timeout': 0, 'table create failed calls': 0, 'table create
↳ successful calls': 1, 'table drop failed calls': 0, 'table drop successful calls
↳ ': 0, 'table rename failed calls': 0, 'table rename successful calls': 0, 'table
↳ salvage failed calls': 0, 'table salvage successful calls': 0, 'table truncate
↳ failed calls': 0, 'table truncate successful calls': 0, 'table verify failed
↳ calls': 0, 'table verify successful calls': 0, 'tiered operations dequeued and
↳ processed': 0, 'tiered operations scheduled': 0, 'tiered storage local retention
↳ time (secs)': 0}, 'thread-state': {'active filesystem fsync calls': 0, 'active
↳ filesystem read calls': 0, 'active filesystem write calls': 0}, 'thread-yield': {
↳ 'application thread time evicting (usecs)': 0, 'application thread time waiting
↳ for cache (usecs)': 0, 'connection close blocked waiting for transaction state
↳ stabilization': 0, 'connection close yielded for lsm manager shutdown': 0, 'data
↳ handle lock yielded': 0, 'get reference for page index and slot time sleeping
↳ (usecs)': 0, 'page access yielded due to prepare state change': 0, 'page acquire
↳ busy blocked': 0, 'page acquire eviction blocked': 0, 'page acquire locked
↳ blocked': 0, 'page acquire read blocked': 0, 'page acquire time sleeping (usecs)
↳ ': 0, 'page delete rollback time sleeping for state change (usecs)': 0, 'page
↳ reconciliation yielded due to child modification': 0}, 'transaction': {'Number
↳ of prepared updates': 0, 'Number of prepared updates committed': 0, 'Number of
↳ prepared updates repeated on the same key': 0, 'Number of prepared updates'
```

(continues on next page)

(continued from previous page)

```

↳ 'rolled back': 0, 'oldest pinned transaction ID rolled back for eviction': 0,
↳ 'prepared transactions': 0, 'prepared transactions committed': 0, 'prepared
↳ transactions currently active': 0, 'prepared transactions rolled back': 0,
↳ 'query timestamp calls': 933, 'race to read prepared update retry': 0, 'rollback
↳ to stable calls': 0, 'rollback to stable history store records with stop
↳ timestamps older than newer records': 0, 'rollback to stable inconsistent
↳ 'checkpoint': 0, 'rollback to stable keys removed': 0, 'rollback to stable keys
↳ 'restored': 0, 'rollback to stable pages visited': 0, 'rollback to stable
↳ 'restored tombstones from history store': 0, 'rollback to stable restored updates
↳ from history store': 0, 'rollback to stable skipping delete rle': 0, 'rollback
↳ to stable skipping stable rle': 0, 'rollback to stable sweeping history store
↳ 'keys': 0, 'rollback to stable tree walk skipping pages': 0, 'rollback to stable
↳ 'updates aborted': 0, 'rollback to stable updates removed from history store': 0,
↳ 'sessions scanned in each walk of concurrent sessions': 15795, 'set timestamp
↳ 'calls': 0, 'set timestamp durable calls': 0, 'set timestamp durable updates': 0,
↳ 'set timestamp oldest calls': 0, 'set timestamp oldest updates': 0, 'set
↳ 'timestamp stable calls': 0, 'set timestamp stable updates': 0, 'transaction
↳ 'begins': 47, 'transaction checkpoint currently running': 0, 'transaction
↳ 'checkpoint currently running for history store file': 0, 'transaction checkpoint
↳ 'generation': 17, 'transaction checkpoint history store file duration (usecs)': 3,
↳ 'transaction checkpoint max time (msecs)': 78, 'transaction checkpoint min time
↳ (msecs)': 31, 'transaction checkpoint most recent duration for gathering all
↳ 'handles (usecs)': 67, 'transaction checkpoint most recent duration for gathering
↳ 'applied handles (usecs)': 0, 'transaction checkpoint most recent duration for
↳ 'gathering skipped handles (usecs)': 45, 'transaction checkpoint most recent
↳ 'handles applied': 0, 'transaction checkpoint most recent handles skipped': 12,
↳ 'transaction checkpoint most recent handles walked': 25, 'transaction checkpoint
↳ 'most recent time (msecs)': 34, 'transaction checkpoint prepare currently running
↳ ': 0, 'transaction checkpoint prepare max time (msecs)': 0, 'transaction
↳ 'checkpoint prepare min time (msecs)': 0, 'transaction checkpoint prepare most
↳ 'recent time (msecs)': 0, 'transaction checkpoint prepare total time (msecs)': 0,
↳ 'transaction checkpoint scrub dirty target': 0, 'transaction checkpoint scrub
↳ 'time (msecs)': 0, 'transaction checkpoint stop timing stress active': 0,
↳ 'transaction checkpoint total time (msecs)': 673, 'transaction checkpoints': 16,
↳ 'transaction checkpoints due to obsolete pages': 0, 'transaction checkpoints
↳ 'skipped because database was clean': 0, 'transaction fsync calls for checkpoint
↳ 'after allocating the transaction ID': 16, 'transaction fsync duration for
↳ 'checkpoint after allocating the transaction ID (usecs)': 9207, 'transaction
↳ 'range of IDs currently pinned': 0, 'transaction range of IDs currently pinned by
↳ 'a checkpoint': 0, 'transaction range of timestamps currently pinned': 0,
↳ 'transaction range of timestamps pinned by a checkpoint': 0, 'transaction range
↳ 'of timestamps pinned by the oldest active read timestamp': 0, 'transaction range
↳ 'of timestamps pinned by the oldest timestamp': 0, 'transaction read timestamp of
↳ 'the oldest active reader': 0, 'transaction rollback to stable currently running
↳ ': 0, 'transaction walk of concurrent sessions': 1056, 'transactions committed': 0,
↳ '13, 'transactions rolled back': 34, 'update conflicts': 0,
↳ 'concurrentTransactions': {'write': {'out': 0, 'available': 128, 'totalTickets': 128}, 'read': {'out': 0, 'available': 128, 'totalTickets': 128}}, 'snapshot-
↳ 'window-settings': {'total number of SnapshotTooOld errors': 0, 'minimum target
↳ 'snapshot window size in seconds': 300, 'current available snapshot window size
↳ 'in seconds': 0, 'latest majority snapshot timestamp available': 'Dec 31
↳ 19:00:00:0', 'oldest majority snapshot timestamp available': 'Dec 31 19:00:00:0',
↳ 'pinned timestamp requests': 0, 'min pinned timestamp': Timestamp(4294967295,
↳ 4294967295)}, 'oplog': {'visibility timestamp': Timestamp(0, 0)}, 'mem': {'bits
↳ ': 64, 'resident': 134, 'virtual': 1563, 'supported': True}, 'metrics': {
↳ 'apiVersions': {'': ['default']}, 'aggStageCounters': {'$_

```

(continues on next page)

(continued from previous page)

```

↳internalApplyOplogUpdate': 0, '$_internalBoundedSort': 0, '$_
↳internalConvertBucketIndexStats': 0, '$_internalFindAndModifyImageLookup': 0, '$_
↳internalInhibitOptimization': 0, '$_internalReshardingIterateTransaction': 0, '$_
↳internalReshardingOwnershipMatch': 0, '$_internalSetWindowFields': 0, '$_
↳internalSplitPipeline': 0, '$_internalUnpackBucket': 0, '$_unpackBucket': 0, '$_
↳$addFields': 0, '$bucket': 0, '$bucketAuto': 0, '$changeStream': 0, '$collStats
↳': 0, '$count': 0, '$currentOp': 0, '$documents': 0, '$facet': 0, '$geoNear': 0,
↳'$graphLookup': 0, '$group': 0, '$indexStats': 0, '$limit': 0, '$_
↳$listLocalSessions': 0, '$listSessions': 0, '$lookup': 0, '$match': 0, '$merge': 0,
↳'$mergeCursors': 0, '$operationMetrics': 0, '$out': 0, '$planCacheStats': 0, '$_
↳$project': 0, '$queue': 0, '$redact': 0, '$replaceRoot': 0, '$replaceWith': 0, '$_
↳$sample': 0, '$set': 2, '$setWindowFields': 0, '$skip': 0, '$sort': 0, '$_
↳$sortByCount': 0, '$unionWith': 0, '$unset': 0, '$unwind': 0}, 'changeStreams': {
↳'largeEventsFailed': 0}, 'commands': {'<UNKNOWN>': 0, '_addShard': {'failed': 0,
↳'total': 0}, '_cloneCollectionOptionsFromPrimaryShard': {'failed': 0, 'total': 0}
↳, '_configsvrAbortReshardCollection': {'failed': 0, 'total': 0}, '_
↳configsvrAddShard': {'failed': 0, 'total': 0}, '_configsvrAddShardToZone': {
↳'failed': 0, 'total': 0}, '_configsvrBalancerCollectionStatus': {'failed': 0,
↳'total': 0}, '_configsvrBalancerStart': {'failed': 0, 'total': 0}, '_
↳configsvrBalancerStatus': {'failed': 0, 'total': 0}, '_configsvrBalancerStop': {
↳'failed': 0, 'total': 0}, '_configsvrCleanupReshardCollection': {'failed': 0,
↳'total': 0}, '_configsvrClearJumboFlag': {'failed': 0, 'total': 0}, '_
↳configsvrCommitChunkMerge': {'failed': 0, 'total': 0}, '_
↳configsvrCommitChunkMigration': {'failed': 0, 'total': 0}, '_
↳configsvrCommitChunkSplit': {'failed': 0, 'total': 0}, '_
↳configsvrCommitChunksMerge': {'failed': 0, 'total': 0}, '_
↳configsvrCommitMovePrimary': {'failed': 0, 'total': 0}, '_
↳configsvrCommitReshardCollection': {'failed': 0, 'total': 0}, '_
↳configsvrCreateDatabase': {'failed': 0, 'total': 0}, '_configsvrDropCollection': {
↳'failed': 0, 'total': 0}, '_configsvrDropDatabase': {'failed': 0, 'total': 0},
↳'_configsvrEnableSharding': {'failed': 0, 'total': 0}, '_
↳configsvrEnsureChunkVersionIsGreater Than': {'failed': 0, 'total': 0}, '_
↳configsvrMoveChunk': {'failed': 0, 'total': 0}, '_configsvrMovePrimary': {'failed
↳': 0, 'total': 0}, '_configsvrRefineCollectionShardKey': {'failed': 0, 'total': 0},
↳'_configsvrRemoveChunks': {'failed': 0, 'total': 0}, '_configsvrRemoveShard
↳': {'failed': 0, 'total': 0}, '_configsvrRemoveShardFromZone': {'failed': 0,
↳'total': 0}, '_configsvrRemoveTags': {'failed': 0, 'total': 0}, '_
↳configsvrRenameCollectionMetadata': {'failed': 0, 'total': 0}, '_
↳configsvrRepairShardedCollectionChunksHistory': {'failed': 0, 'total': 0}, '_
↳configsvrReshardCollection': {'failed': 0, 'total': 0}, '_
↳configsvrSetAllowMigrations': {'failed': 0, 'total': 0}, '_
↳configsvrShardCollection': {'failed': 0, 'total': 0}, '_
↳configsvrUpdateZoneKeyRange': {'failed': 0, 'total': 0}, '_
↳flushDatabaseCacheUpdates': {'failed': 0, 'total': 0}, '_
↳flushDatabaseCacheUpdatesWithWriteConcern': {'failed': 0, 'total': 0}, '_
↳flushReshardingStateChange': {'failed': 0, 'total': 0}, '_
↳flushRoutingTableCacheUpdates': {'failed': 0, 'total': 0}, '_
↳flushRoutingTableCacheUpdatesWithWriteConcern': {'failed': 0, 'total': 0}, '_
↳getNextSessionMods': {'failed': 0, 'total': 0}, '_getUserCacheGeneration': {
↳'failed': 0, 'total': 0}, '_isSelf': {'failed': 0, 'total': 0}, '_killOperations
↳': {'failed': 0, 'total': 0}, '_mergeAuthzCollections': {'failed': 0, 'total': 0}
↳, '_migrateClone': {'failed': 0, 'total': 0}, '_recvChunkAbort': {'failed': 0,
↳'total': 0}, '_recvChunkCommit': {'failed': 0, 'total': 0}, '_recvChunkStart': {
↳'failed': 0, 'total': 0}, '_recvChunkStatus': {'failed': 0, 'total': 0}, '_
↳shardsvrAbortReshardCollection': {'failed': 0, 'total': 0}, '_
↳shardsvrCleanupReshardCollection': {'failed': 0, 'total': 0}, '_

```

(continues on next page)

(continued from previous page)

```

↳shardsvrCloneCatalogData': {'failed': 0, 'total': 0}, '_
↳shardsvrCommitReshardCollection': {'failed': 0, 'total': 0}, '_
↳shardsvrCreateCollection': {'failed': 0, 'total': 0}, '_
↳shardsvrCreateCollectionParticipant': {'failed': 0, 'total': 0}, '_
↳shardsvrDropCollection': {'failed': 0, 'total': 0}, '_
↳shardsvrDropCollectionIfUUIDNotMatching': {'failed': 0, 'total': 0}, '_
↳shardsvrDropCollectionParticipant': {'failed': 0, 'total': 0}, '_
↳shardsvrDropDatabase': {'failed': 0, 'total': 0}, '_
↳shardsvrDropDatabaseParticipant': {'failed': 0, 'total': 0}, '_
↳shardsvrMovePrimary': {'failed': 0, 'total': 0}, '_
↳shardsvrRefineCollectionShardKey': {'failed': 0, 'total': 0}, '_
↳shardsvrRenameCollection': {'failed': 0, 'total': 0}, '_
↳shardsvrRenameCollectionParticipant': {'failed': 0, 'total': 0}, '_
↳shardsvrRenameCollectionParticipantUnblock': {'failed': 0, 'total': 0}, '_
↳shardsvrReshardCollection': {'failed': 0, 'total': 0}, '_
↳shardsvrReshardingOperationTime': {'failed': 0, 'total': 0}, '_
↳shardsvrSetAllowMigrations': {'failed': 0, 'total': 0}, '_shardsvrShardCollection
': {'failed': 0, 'total': 0}, '_transferMods': {'failed': 0, 'total': 0},
↳'abortTransaction': {'failed': 0, 'total': 0}, 'aggregate': {'allowDiskUseTrue': 0,
↳0, 'failed': 0, 'total': 0}, 'appendOplogNote': {'failed': 0, 'total': 0},
↳'applyOps': {'failed': 0, 'total': 0}, 'authenticate': {'failed': 0, 'total': 0},
↳'autoSplitVector': {'failed': 0, 'total': 0}, 'availableQueryOptions': {'failed
': 0, 'total': 0}, 'buildInfo': {'failed': 0, 'total': 0}, 'checkShardingIndex':
{'failed': 0, 'total': 0}, 'cleanupOrphaned': {'failed': 0, 'total': 0},
↳'cloneCollectionAsCapped': {'failed': 0, 'total': 0}, 'collMod': {'failed': 0,
↳0, 'total': 0}, 'validator': {'failed': 0, 'jsonSchema': 0, 'total': 0}, 'collStats
': {'failed': 0, 'total': 0}, 'commitTransaction': {'failed': 0, 'total': 0},
↳'compact': {'failed': 0, 'total': 0}, 'connPoolStats': {'failed': 0, 'total': 0},
↳'connPoolSync': {'failed': 0, 'total': 0}, 'connectionStatus': {'failed': 0,
↳0, 'total': 0}, 'convertToCapped': {'failed': 0, 'total': 0},
↳'coordinateCommitTransaction': {'failed': 0, 'total': 0}, 'count': {'failed': 0,
↳0, 'total': 0}, 'create': {'failed': 0, 'total': 0}, 'validator': {'failed': 0,
↳0, 'total': 0}, 'createIndexes': {'failed': 0, 'total': 0},
↳'createRole': {'failed': 0, 'total': 0}, 'createUser': {'failed': 0, 'total': 0},
↳'currentOp': {'failed': 0, 'total': 0}, 'dataSize': {'failed': 0, 'total': 0},
↳'dbCheck': {'failed': 0, 'total': 0}, 'dbHash': {'failed': 0, 'total': 0},
↳'dbStats': {'failed': 0, 'total': 0}, 'delete': {'failed': 0, 'total': 0},
↳'distinct': {'failed': 0, 'total': 1}, 'donorAbortMigration': {'failed': 0,
↳0, 'total': 0}, 'donorForgetMigration': {'failed': 0, 'total': 0},
↳'donorStartMigration': {'failed': 0, 'total': 0}, 'driverOIDTest': {'failed': 0,
↳0, 'total': 0}, 'drop': {'failed': 0, 'total': 0}, 'dropAllRolesFromDatabase': {
↳'failed': 0, 'total': 0}, 'dropAllUsersFromDatabase': {'failed': 0, 'total': 0},
↳'dropConnections': {'failed': 0, 'total': 0}, 'dropDatabase': {'failed': 0,
↳0, 'total': 0}, 'dropIndexes': {'failed': 0, 'total': 0}, 'dropRole': {'failed': 0,
↳0, 'total': 0}, 'dropUser': {'failed': 0, 'total': 0}, 'endSessions': {'failed': 0,
↳0, 'total': 0}, 'explain': {'failed': 0, 'total': 0}, 'features': {'failed': 0,
↳0, 'total': 0}, 'filemd5': {'failed': 0, 'total': 0}, 'find': {'failed': 0, 'total
': 5}, 'findAndModify': {'arrayFilters': 0, 'failed': 0, 'pipeline': 0, 'total': 0},
↳'flushRouterConfig': {'failed': 0, 'total': 0}, 'fsync': {'failed': 0, 'total': 0},
↳'fsyncUnlock': {'failed': 0, 'total': 0}, 'getCmdLineOpts': {'failed': 0,
↳0, 'total': 0}, 'getDatabaseVersion': {'failed': 0, 'total': 0},
↳'getDefaultRWConcern': {'failed': 0, 'total': 0}, 'getDiagnosticData': {'failed
': 0, 'total': 0}, 'getFreeMonitoringStatus': {'failed': 0, 'total': 0},
↳'getLastError': {'failed': 0, 'total': 0}, 'getLog': {'failed': 0, 'total': 0},
↳'getMore': {'failed': 0, 'total': 1}, 'getParameter': {'failed': 0, 'total': 0},
↳'getShardMap': {'failed': 0, 'total': 0}, 'getShardVersion': {'failed': 0, 'total
': 0}

```

(continues on next page)

(continued from previous page)

```

': 0}, 'getnonce': {'failed': 0, 'total': 0}, 'grantPrivilegesToRole': {'failed': 0, 'total': 0}, 'grantRolesToRole': {'failed': 0, 'total': 0}, 'grantRolesToUser': {'failed': 0, 'total': 0}, 'hello': {'failed': 1, 'total': 140}, 'hostInfo': {'failed': 0, 'total': 0}, 'insert': {'failed': 0, 'total': 6}, 'internalRenameIfOptionsAndIndexesMatch': {'failed': 0, 'total': 0}, 'invalidateUserCache': {'failed': 0, 'total': 0}, 'isMaster': {'failed': 0, 'total': 53}, 'killAllSessions': {'failed': 0, 'total': 0}, 'killAllSessionsByPattern': {'failed': 0, 'total': 0}, 'killCursors': {'failed': 0, 'total': 0}, 'killOp': {'failed': 0, 'total': 0}, 'killSessions': {'failed': 0, 'total': 0}, 'listCollections': {'failed': 0, 'total': 0}, 'listCommands': {'failed': 0, 'total': 0}, 'listDatabases': {'failed': 0, 'total': 0}, 'listIndexes': {'failed': 0, 'total': 8}, 'lockInfo': {'failed': 0, 'total': 0}, 'logRotate': {'failed': 0, 'total': 0}, 'logout': {'failed': 0, 'total': 0}, 'mapReduce': {'failed': 0, 'total': 0}, 'mergeChunks': {'failed': 0, 'total': 0}, 'moveChunk': {'failed': 0, 'total': 0}, 'ping': {'failed': 0, 'total': 0}, 'planCacheClear': {'failed': 0, 'total': 0}, 'planCacheClearFilters': {'failed': 0, 'total': 0}, 'planCacheListFilters': {'failed': 0, 'total': 0}, 'planCacheSetFilter': {'failed': 0, 'total': 0}, 'prepareTransaction': {'failed': 0, 'total': 0}, 'profile': {'failed': 0, 'total': 0}, 'reIndex': {'failed': 0, 'total': 0}, 'recipientForgetMigration': {'failed': 0, 'total': 0}, 'recipientSyncData': {'failed': 0, 'total': 0}, 'refreshSessions': {'failed': 0, 'total': 0}, 'renameCollection': {'failed': 0, 'total': 0}, 'replSetAbortPrimaryCatchUp': {'failed': 0, 'total': 0}, 'replSetFreeze': {'failed': 0, 'total': 0}, 'replSetGetConfig': {'failed': 0, 'total': 0}, 'replSetGetRBID': {'failed': 0, 'total': 0}, 'replSetGetStatus': {'failed': 0, 'total': 0}, 'replSetHeartbeat': {'failed': 0, 'total': 0}, 'replSetInitiate': {'failed': 0, 'total': 0}, 'replSetMaintenance': {'failed': 0, 'total': 0}, 'replSetReconfig': {'failed': 0, 'total': 0}, 'replSetRequestVotes': {'failed': 0, 'total': 0}, 'replSetResizeOplog': {'failed': 0, 'total': 0}, 'replSetStepDown': {'failed': 0, 'total': 0}, 'replSetStepDownWithForce': {'failed': 0, 'total': 0}, 'replSetStepUp': {'failed': 0, 'total': 0}, 'replSetSyncFrom': {'failed': 0, 'total': 0}, 'replSetUpdatePosition': {'failed': 0, 'total': 0}, 'revokePrivilegesFromRole': {'failed': 0, 'total': 0}, 'revokeRolesFromRole': {'failed': 0, 'total': 0}, 'revokeRolesFromUser': {'failed': 0, 'total': 0}, 'rolesInfo': {'failed': 0, 'total': 0}, 'rotateCertificates': {'failed': 0, 'total': 0}, 'saslContinue': {'failed': 0, 'total': 0}, 'saslStart': {'failed': 0, 'total': 0}, 'serverStatus': {'failed': 0, 'total': 2}, 'setDefaultRWConcern': {'failed': 0, 'total': 0}, 'setFeatureCompatibilityVersion': {'failed': 0, 'total': 0}, 'setFreeMonitoring': {'failed': 0, 'total': 0}, 'setIndexCommitQuorum': {'failed': 0, 'total': 0}, 'setParameter': {'failed': 0, 'total': 0}, 'setProfilingFilterGlobally': {'failed': 0, 'total': 0}, 'setShardVersion': {'failed': 0, 'total': 0}, 'shardingState': {'failed': 0, 'total': 0}, 'shutdown': {'failed': 0, 'total': 0}, 'splitChunk': {'failed': 0, 'total': 0}, 'splitVector': {'failed': 0, 'total': 0}, 'startRecordingTraffic': {'failed': 0, 'total': 0}, 'startSession': {'failed': 0, 'total': 0}, 'stopRecordingTraffic': {'failed': 0, 'total': 0}, 'top': {'failed': 0, 'total': 0}, 'update': {'arrayFilters': 0, 'failed': 0, 'pipeline': 2, 'total': 2}, 'updateRole': {'failed': 0, 'total': 0}, 'updateUser': {'failed': 0, 'total': 0}, 'usersInfo': {'failed': 0, 'total': 0}, 'validate': {'failed': 0, 'total': 0}, 'validateDBMetadata': {'failed': 0, 'total': 0}, 'voteCommitIndexBuild': {'failed': 0, 'total': 0}, 'waitForFailPoint': {'failed': 0, 'total': 0}, 'whatsmyuri': {'failed': 0, 'total': 0}, 'cursor': {'moreThanOneBatch': 1, 'timedOut': 0}, 'totalOpened': 1, 'lifespan': {'greaterThanOrEqual10Minutes': 0, 'lessThan10Minutes': 0, 'lessThan15Seconds': 0, 'lessThan1Minute': 0}, 'lessThan1Second': 1, 'lessThan30Seconds': 0, 'lessThan5Seconds': 0, 'open': {'noTimeout': 0, 'pinned': 0, 'total': 0}, 'document': {'deleted': 0, 'inserted': 0}

```

(continues on next page)

(continued from previous page)

```

': 6, 'returned': 244, 'updated': 1}, 'dotsAndDollarsFields': {'inserts': 0,
'updates': 0}, 'getLastError': {'wtime': {'num': 0, 'totalMillis': 0}, 'wtimeouts':
0, 'default': {'unsatisfiable': 0, 'wtimeouts': 0}}, 'mongos': {'cursor': {
'moreThanOneBatch': 0, 'totalOpened': 0}}, 'operation': {'scanAndOrder': 0,
'transactionTooLargeForCacheErrors': 0,
'transactionTooLargeForCacheErrorsConvertedToWriteConflict': 0, 'writeConflicts':
0}, 'operatorCounters': {'expressions': {'$_internalJsEmit': 0, '$abs': 0, '$acos':
0, '$acosh': 0, '$add': 0, '$allElementsTrue': 0, '$and': 0, '$anyElementTrue':
0, '$arrayElemAt': 0, '$arrayToObject': 0, '$asin': 0, '$asinh': 0, '$atan': 0,
'$atan2': 0, '$atanh': 0, '$avg': 0, '$binarySize': 0, '$bsonSize': 0, '$ceil':
0, '$cmp': 0, '$concat': 0, '$concatArrays': 0, '$cond': 0, '$const': 0, '$convert':
0, '$cos': 0, '$cosh': 0, '$dateAdd': 0, '$dateDiff': 0, '$dateFromParts': 0,
'$dateFromString': 0, '$dateSubtract': 0, '$dateToParts': 0, '$dateToString': 0,
'$dateTrunc': 0, '$dayOfMonth': 0, '$dayOfWeek': 0, '$dayOfYear': 0, '$degreesToRadians':
0, '$divide': 0, '$eq': 0, '$exp': 0, '$filter': 0, '$first': 0, '$floor': 0, '$function':
0, '$getField': 0, '$gt': 0, '$gte': 0, '$hour': 0, '$ifNull': 0, '$in': 0, '$indexOfArray':
0, '$indexOfBytes': 0, '$indexOfCP': 0, '$isArray': 0, '$isNumber': 0, '$isoDayOfWeek':
0, '$isoWeek': 0, '$isoWeekYear': 0, '$last': 0, '$let': 0, '$literal': 0, '$ln': 0,
'$log': 0, '$log10': 0, '$lt': 0, '$lte': 0, '$ltrim': 0, '$map': 0, '$max': 0,
'$mergeObjects': 0, '$meta': 0, '$millisecond': 0, '$min': 0, '$minute': 0, '$mod':
0, '$month': 0, '$multiply': 0, '$ne': 0, '$not': 0, '$objectToArray': 0, '$or': 0,
'$pow': 0, '$radiansToDegrees': 0, '$rand': 0, '$range': 0, '$reduce': 0, '$regexFind':
0, '$regexFindAll': 0, '$regexMatch': 0, '$replaceAll': 0, '$replaceOne': 0, '$reverseArray':
0, '$round': 0, '$rtrim': 0, '$second': 0, '$setDifference': 0, '$setEquals': 0,
'$setField': 0, '$setIntersection': 0, '$setIsSubset': 0, '$setUnion': 0, '$sin': 0, '$sinh':
0, '$size': 0, '$slice': 0, '$split': 0, '$sqrt': 0, '$stdDevPop': 0, '$stdDevSamp': 0,
'$strLenBytes': 0, '$strLenCP': 0, '$strcasecmp': 0, '$substr': 0, '$substrBytes': 0,
'$substrCP': 0, '$subtract': 0, '$sum': 0, '$switch': 0, '$tan': 0, '$tanh': 0,
'$toBool': 0, '$toDate': 0, '$toDecimal': 0, '$toDouble': 0, '$toHashedIndexKey': 0,
'$ToInt': 0, '$toLong': 0, '$toLower': 0, '$toObjectId': 0, '$toString': 0, '$toUpperCase':
0, '$trim': 0, '$trunc': 0, '$type': 0, '$unsetField': 0, '$week': 0, '$year': 0, '$zip': 0},
'groupAccumulators': {'$_internalJsReduce': 0, '$accumulator': 0, '$addToSet': 0, '$avg': 0,
'$count': 0, '$first': 0, '$last': 0, '$max': 0, '$mergeObjects': 0, '$min': 0, '$push': 0,
'$stdDevPop': 0, '$stdDevSamp': 0, '$sum': 0}, 'match': {'$all': 0, '$alwaysFalse': 0,
'$alwaysTrue': 0, '$and': 0, '$bitsAllClear': 0, '$bitsAllSet': 0, '$bitsAnyClear': 0,
'$bitsAnySet': 0, '$comment': 0, '$elemMatch': 0, '$eq': 0, '$exists': 0, '$expr': 0,
'$geoIntersects': 0, '$geoWithin': 0, '$gt': 0, '$gte': 0, '$in': 0, '$jsonSchema': 0,
'$lt': 4, '$lte': 0, '$mod': 0, '$ne': 0, '$near': 0, '$nearSphere': 0, '$nin': 0,
'$nor': 0, '$not': 0, '$or': 0, '$regex': 0, '$sampleRate': 0, '$size': 0, '$text': 0,
'$type': 0, '$where': 0}, 'windowAccumulators': {'$addToSet': 0, '$avg': 0, '$count': 0,
'$covariancePop': 0, '$covarianceSamp': 0, '$denseRank': 0, '$derivative': 0, '$documentNumber': 0,
'$expMovingAvg': 0, '$first': 0, '$integral': 0, '$last': 0, '$max': 0, '$min': 0,
'$push': 0, '$rank': 0, '$shift': 0, '$stdDevPop': 0, '$stdDevSamp': 0, '$sum': 0}, 'query': {
'deleteManyCount': 0, 'planCacheTotalSizeEstimateBytes': 0, 'updateDeleteManyDocumentsMaxCount': 0,
'updateDeleteManyDocumentsTotalCount': 0, 'updateDeleteManyDurationMaxMs': 0, 'updateDeleteManyDurationTotalMs': 0,
'updateManyCount': 0, 'updateOneOpStyleBroadcastWithExactIDCount': 0, 'multiPlanner': {
'classicCount': 0, 'classicMicros': 0, 'classicWorks': 0, 'sbeCount': 0, 'sbeMicros': 0,
'sbeNumReads': 0, 'histograms': {'classicMicros': 0, 'lowerBound': 0, 'count': 0}, 'lowerBound': 0, 'count': 0,
'lowerBound': 1024, 'count': 0, 'lowerBound': 4096, 'count': 0, 'lowerBound': 16384, 'count': 0, 'lowerBound': 65536,
'count': 0, 'lowerBound': 262144, 'count': 0, 'lowerBound': 1048576, 'count': 0}}}
```

(continues on next page)

(continued from previous page)

```

': 0}, {'lowerBound': 4194304, 'count': 0}, {'lowerBound': 16777216, 'count': 0},
{'lowerBound': 67108864, 'count': 0}, {'lowerBound': 268435456, 'count': 0}, {
'lowerBound': 1073741824, 'count': 0}], 'classicNumPlans': [{['lowerBound': 0,
'count': 0}, {'lowerBound': 2, 'count': 0}, {'lowerBound': 4, 'count': 0}, {
'lowerBound': 8, 'count': 0}, {'lowerBound': 16, 'count': 0}, {'lowerBound': 32,
'count': 0}], 'classicWorks': [{['lowerBound': 0, 'count': 0}, {'lowerBound': 128,
'count': 0}, {'lowerBound': 256, 'count': 0}, {'lowerBound': 512, 'count': 0}, {
'lowerBound': 1024, 'count': 0}, {'lowerBound': 2048, 'count': 0}, {'lowerBound':
4096, 'count': 0}, {'lowerBound': 8192, 'count': 0}, {'lowerBound': 16384,
'count': 0}, {'lowerBound': 32768, 'count': 0}], 'sbeMicros': [{['lowerBound': 0,
'count': 0}, {'lowerBound': 1024, 'count': 0}, {'lowerBound': 4096, 'count': 0},
{'lowerBound': 16384, 'count': 0}, {'lowerBound': 65536, 'count': 0}, {
'lowerBound': 262144, 'count': 0}, {'lowerBound': 1048576, 'count': 0}, {
'lowerBound': 4194304, 'count': 0}, {'lowerBound': 16777216, 'count': 0}, {
'lowerBound': 67108864, 'count': 0}, {'lowerBound': 268435456, 'count': 0}, {
'lowerBound': 1073741824, 'count': 0}], 'sbeNumPlans': [{['lowerBound': 0, 'count':
0}, {'lowerBound': 2, 'count': 0}, {'lowerBound': 4, 'count': 0}, {'lowerBound':
8, 'count': 0}, {'lowerBound': 16, 'count': 0}, {'lowerBound': 32, 'count': 0}],
'sbeNumReads': [{['lowerBound': 0, 'count': 0}, {'lowerBound': 128, 'count': 0},
{'lowerBound': 256, 'count': 0}, {'lowerBound': 512, 'count': 0}, {'lowerBound':
1024, 'count': 0}, {'lowerBound': 2048, 'count': 0}, {'lowerBound': 4096,
'count': 0}, {'lowerBound': 8192, 'count': 0}, {'lowerBound': 16384, 'count': 0},
{'lowerBound': 32768, 'count': 0}}]}, 'queryExecutor': {'scanned': 1,
'scannedObjects': 483, 'collectionScans': {'nonTailable': 2, 'total': 2}},
'record': {'moves': 0}, 'repl': {'executor': {'pool': {'inProgressCount': 0}},
'queues': {'networkInProgress': 0, 'sleepers': 0}, 'unsignaledEvents': 0,
'shuttingDown': False, 'networkInterface': 'DEPRECATED: getDiagnosticString is
deprecated in NetworkInterfaceTL'}, 'apply': {'attemptsToBecomeSecondary': 0,
'batchSize': 0, 'batches': {'num': 0, 'totalMillis': 0}, 'ops': 0}, 'buffer': {
'count': 0, 'maxSizeBytes': 0, 'sizeBytes': 0}, 'initialSync': {'completed': 0,
'failedAttempts': 0, 'failures': 0}, 'network': {'bytes': 0, 'getmores': {'num': 0,
'totalMillis': 0, 'numEmptyBatches': 0}, 'notPrimaryLegacyUnacknowledgedWrites':
0, 'notPrimaryUnacknowledgedWrites': 0, 'oplogGetMoresProcessed': {'num': 0,
'totalMillis': 0}, 'ops': 0}, 'readersCreated': 0, 'replSetUpdatePosition': {'num':
0}, 'reconfig': {'numAutoReconfigsForRemovalOfNewlyAddedFields': 0},
'stateTransition': {'lastStateTransition': '', 'userOperationsKilled': 0,
'userOperationsRunning': 0}, 'syncSource': {'numSelections': 0,
'numSyncSourceChangesDueToSignificantlyCloserNode': 0, 'numTimesChoseDifferent':
0, 'numTimesChoseSame': 0, 'numTimesCouldNotFind': 0}}, 'ttl': {'deletedDocuments':
0, 'passes': 15}}, 'ok': 1.0}

```

```

## retrieve recessions dates for plotting
alf = Alfred(api_key=credentials['fred']['api_key'])
vspan = alf.date_spans()

```

## 23.1 Scrape FOMC Minutes text

```
dates = fomc['minutes'].distinct('date')           # check dates stored in MongoDB

## retrieve from FOMC website
catalog = FOMCReader.fetch()          # check for new dates in FOMC site, and retrieve
print(f"FOMC: {len(catalog)} dates {min(catalog.keys()):}-{max(catalog.keys()):}")
docs = {d: FOMCReader.fetch(url) for d, url in catalog.items() if d not in dates}
print('NEW:', ", ".join([f"{k}: {len(v)} chars" for k, v in docs.items()]))
```

FOMC: 244 dates 19930203-20230726  
NEW:

### Filter raw text

- Minutes file usually start with sentences about:
  - developments in domestic financial markets and system open market transactions; or
  - discussion of economic and financial outlook and implementation of monetary policy; sometimes
  - staff presentations on ad-hoc policy topics
  - discussion ad-hoc issues such as “balance sheet normalization” in recent years
- keep text approximately from (i.e. skip earlier organizational items):
  - “Developments in Financial Markets” or
  - “Discussion of Guideliness for Policy Normalizatio” 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”
- skip earlier which merely re-affirmation of general policy statement
- delete after adjourn line, i.e. after schedule next meeting,
- ignore notation votes, minutes approvals, signature, and footnotes
- ignore intermeeting conference calls

```
import subprocess
import tempfile
def edit(text: str) -> str:
    """helper to spawn editor and write/read/write to tempfile"""
    with tempfile.NamedTemporaryFile(suffix=".tmp") as f: # save temp file
        f.write(text.encode("utf-8"))
        f.flush()
        subprocess.call([os.environ.get('EDITOR', 'emacs'), "-nw", f.name])
        f.seek(0)
    return f.read().decode("utf-8")           # keep edited text
```

```

if docs:
    ## loop to edit out head and tail of document
    results = list()
    for date, initial_message in docs.items():  # manually trim text
        edited_text = edit(initial_message)
        results.append({'date': date, 'text' : edited_text})
    results = sorted(results, key = lambda x: x['date'])  # sort by date

    ## save the new edited docs
    Store(paths['scratch'], ext='gz').dump(results, 'fomc')
    for doc in results: # store docs for new dates
        fomc.insert('minutes', doc, keys=['date'])

```

```

# Retrieve and preprocess text
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...
...	...
20230201	Developments in Financial Markets and Open Mar...
20230322	Recent Developments in the Banking Sector \nBe...
20230503	Developments in Financial Markets and Open Mar...
20230614	Developments in Financial Markets and Open Mar...
20230726	Developments in Financial Markets and Open Mar...

[244 rows x 1 columns]

```

## set stop words and vectorize inputs
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']
ngram_range = (1,1) # (1, 2)
max_df, min_df, max_features = 0.5, 6, 5000 # some reasonable constraints
tfidf_vectorizer = sklearn.feature_extraction.text.TfidfVectorizer(
    strip_accents='unicode',
    lowercase=True,
    stop_words=StopWords,
    ngram_range=ngram_range,           # (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") #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

```

(continues on next page)

(continued from previous page)

```
max_df=max_df,
min_df=min_df,
max_features=max_features,
token_pattern=r"\b[\^\d\W] [\^\d\W] [\^\d\W]+\b")
```

## 23.2 Topic models

```
# Define models
n_components = 4
algos = {
    'LSA': (TruncatedSVD(n_components=n_components),
             tfidf_vectorizer),
    'LDA': (LatentDirichletAllocation(n_components=n_components,
                                       learning_method='batch', #'online',
                                       # learning_offset = 50.0,
                                       max_iter = 40,
                                       random_state = 42), tf_vectorizer),
    'PLSI': (NMF(n_components=n_components,
                  beta_loss='kullback-leibler',
                  solver='mu',
                  alpha_W=0.00005,
                  alpha_H=0.00005,
                  l1_ratio=0.5,
                  max_iter=1000,
                  random_state = 42),
              tfidf_vectorizer),
    'NMF': (NMF(n_components=n_components,
                  random_state=42,
                  beta_loss='frobenius',
                  alpha_W=0.00005,
                  alpha_H=0.00005,
                  l1_ratio=0.5),
              tfidf_vectorizer)}
```

```
## Fit and plot models
scores = dict()
dates = dict()
for ifig, (name, (base, vectorizer)) in enumerate(algos.items()):
    vectorized = vectorizer.fit_transform(docs.to_list())
    feature_names = vectorizer.get_feature_names_out()
    model = base.fit(vectorized)
    topics = model.transform(vectorized)
    if name == 'LSA': # Additional step for LSA
        kmeans = KMeans(n_clusters=n_components,
                         n_init=5,
                         random_state=37) \
            .fit(topics) # find centroids of the latent factors
        topics = kmeans.transform(topics) # distance of document to centroid
        topics = -(topics / topics.sum(axis=1, keepdims=True)) # as similarity
        scores[name] = softmax(model.components_, axis=1) # scale word scores
    else:
        scores[name] = model.components_
```

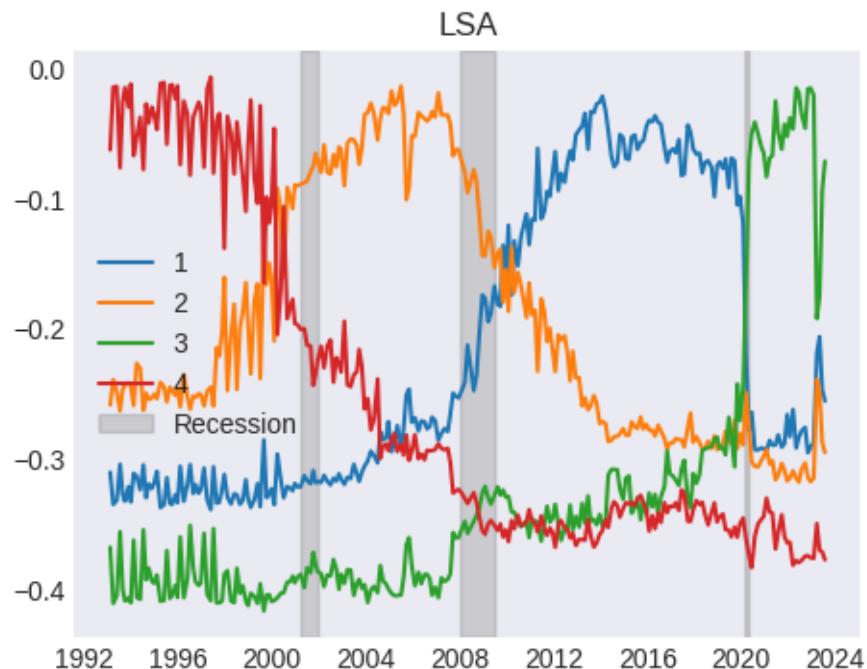
(continues on next page)

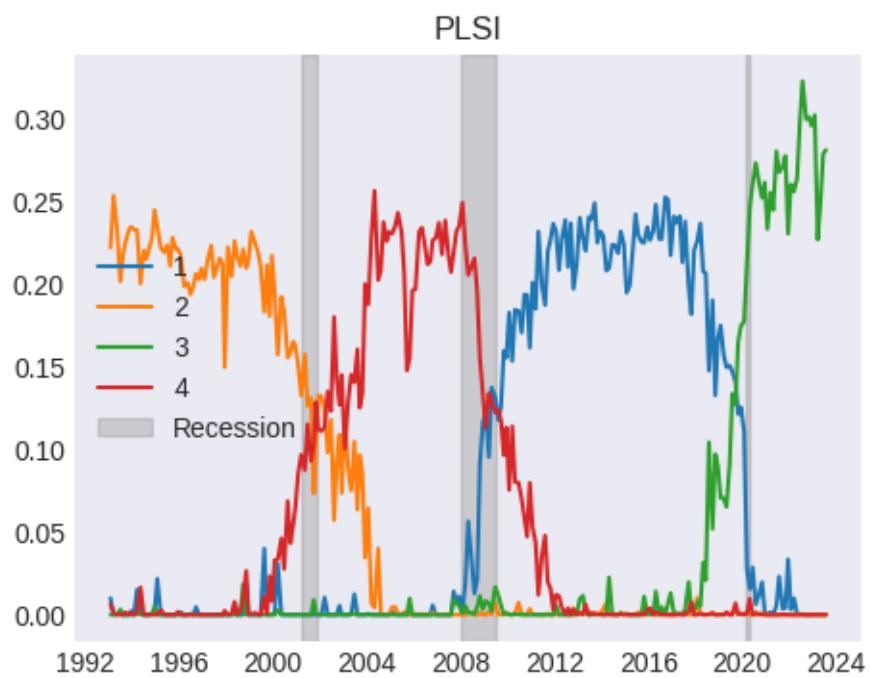
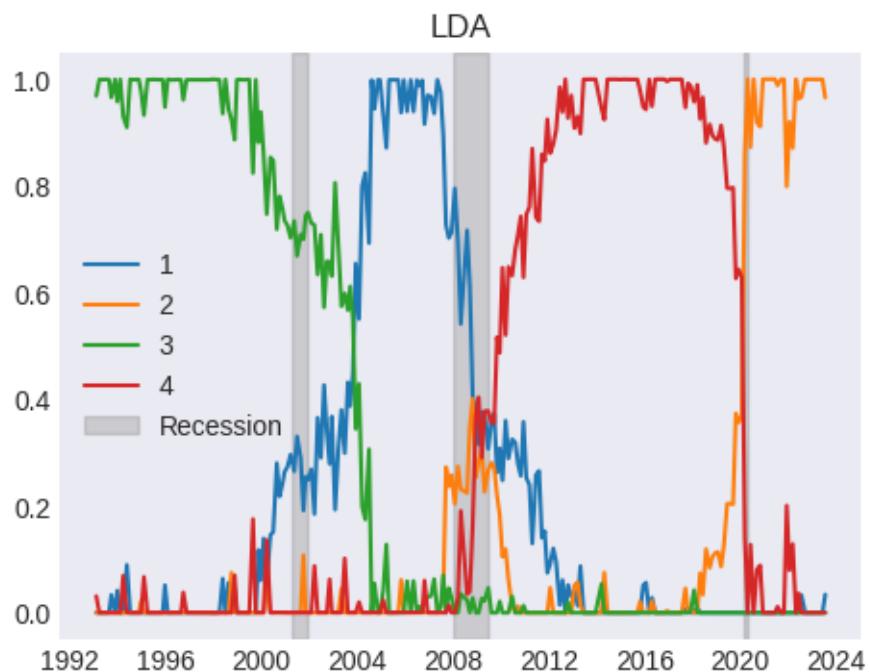
(continued from previous page)

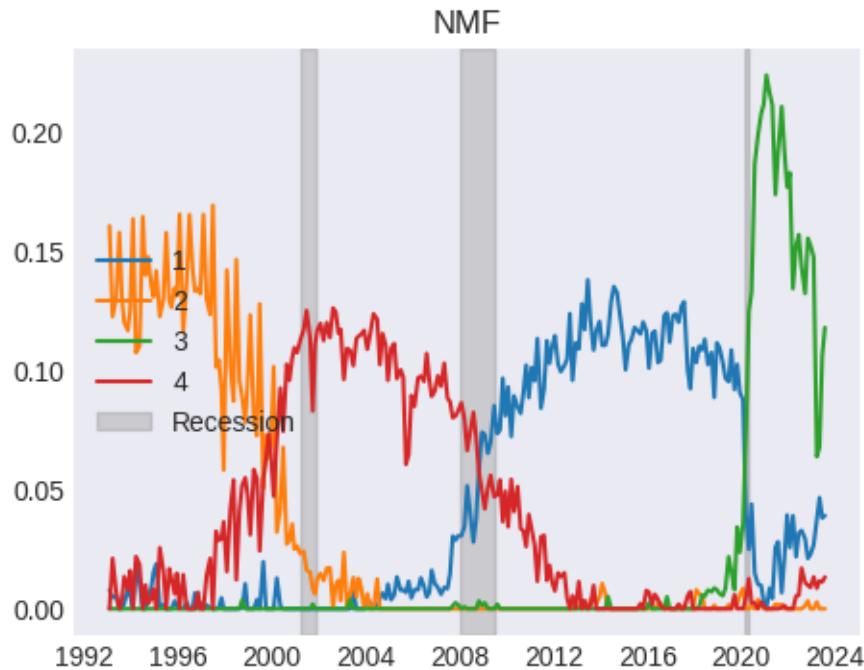
```

fig, ax = plt.subplots(num=1 + ifig, clear=True, figsize=(5, 4))
date = pd.DatetimeIndex(docs.index.astype(str))
ax.plot(date, topics)
for a,b in vspans:
    if b >= min(date):
        ax.axvspan(a, min(b, max(date)), alpha=0.3, color='grey')
ax.set_title(name)
ax.legend([f"i{1}" for i in range(n_components)] + ['Recession'],
          loc='center left')
plt.tight_layout(pad=2)
for topic in range(topics.shape[1]):
    arg = DataFrame({'t': np.argmax(topics, axis=1),
                     'dt': docs.index})
    g = (arg!=arg.shift()).cumsum().groupby('t').agg(['first', 'last']) - 1
    g['topic'] = arg.loc[g.iloc[:,1], 't'].values
    dates[name] = {topic: [(arg['dt'].iloc[row[0]],
                           arg['dt'].iloc[row[1]]) for row in g.itertuples(index=False, name=None)
                           if row[2] == topic]
                  for topic in range(topics.shape[1])}
plt.savefig(imgdir / (name + '_topics.jpg'))

```







```
## display top features
figsize = (10, 10) # (5, 5)
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]
                 for i in components.argsort()[:-top_n - 1:-1]}
        print("Topic", topic+1, dates[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"Topic {topic+1} {dates[name]}")
        regime = ", ".join([f"{d[0]}-{d[1]}" if d[0] != d[1] else f"{d[0]}"
                            for d in dates[name][topic]])
        ax.set_xlabel(regime if len(regime) < 75 else '-- many --',
                      fontsize=8,
                      loc='left')
    plt.tight_layout()
    plt.savefig(imgdir / (name + '_words.jpg'))
plt.show()
```

```
Topic 1 {0: [(20091104, 20091104), (20100127, 20100127), (20100428, 20200129)], 1: [(19990824, 19990824), (20000321, 20000516), (20000822, 20090923), (20091216, 20091216), (20100316, 20100316)], 2: [(20200315, 20230726)], 3: [(19930203, 19990629), (19991005, 20000202), (20000628, 20000628)]}
['ranges', 'restraint', 'accommodation', 'pandemic', 'program', 'billion', 'mbs', 'sheet', 'couple', 'normalization', 'guidance', 'mandate', 'aggregates',
```

(continues on next page)

(continued from previous page)

```

↳ 'facility', 'purchase', 'rrp', 'software', 'per', 'principal', 'final']
Topic 2 {0: [(20091104, 20091104), (20100127, 20100127), (20100428, 20200129)], 1:
↳ [(19990824, 19990824), (20000321, 20000516), (20000822, 20090923), (20091216,
↳ 20091216), (20100316, 20100316)], 2: [(20200315, 20230726)], 3: [(19930203,
↳ 19990629), (19991005, 20000202), (20000628, 20000628)]}
['ranges', 'aggregates', 'restraint', 'acceptable', 'direction', 'giving', 'careful
↳ ', 'civilian', 'finished', 'lesser', 'final', 'durable', 'positions',
↳ 'contemplated', 'called', 'prospective', 'behavior', 'adjustment', 'nation',
↳ 'upper']
Topic 3 {0: [(20091104, 20091104), (20100127, 20100127), (20100428, 20200129)], 1:
↳ [(19990824, 19990824), (20000321, 20000516), (20000822, 20090923), (20091216,
↳ 20091216), (20100316, 20100316)], 2: [(20200315, 20230726)], 3: [(19930203,
↳ 19990629), (19991005, 20000202), (20000628, 20000628)]}
['pandemic', 'covid', 'virus', 'goals', 'vaccinations', 'ranges', 'pre', 'remarked
↳ ', 'repo', 'bottlenecks', 'ukraine', 'distancing', 'achieve', 'social', 'flow',
↳ 'health', 'coronavirus', 'russia', 'aggregates', 'restrictive']
Topic 4 {0: [(20091104, 20091104), (20100127, 20100127), (20100428, 20200129)], 1:
↳ [(19990824, 19990824), (20000321, 20000516), (20000822, 20090923), (20091216,
↳ 20091216), (20100316, 20100316)], 2: [(20200315, 20230726)], 3: [(19930203,
↳ 19990629), (19991005, 20000202), (20000628, 20000628)]}
['ranges', 'normalization', 'restraint', 'aggregates', 'couple', 'acceptable', 'rrp
↳ ', 'symmetric', 'careful', 'giving', 'lesser', 'monitoring', 'positions',
↳ 'underutilization', 'principal', 'behavior', 'districts', 'per', 'velocity',
↳ 'directs']
Topic 1 {0: [(20031209, 20081216), (20090318, 20090318), (20090812, 20090923)], 1:
↳ [(20200315, 20230726)], 2: [(19930203, 20031028)], 3: [(20090128, 20090128),
↳ (20090429, 20090624), (20091104, 20200129)]}
['software', 'tech', 'house', 'going', 'headline', 'contraction', 'mortgages',
↳ 'paper', 'accommodation', 'institutions', 'losses', 'probably', 'commodities',
↳ 'contained', 'strains', 'decelerated', 'transportation', 'commodity', 'summer',
↳ 'preceding']
Topic 2 {0: [(20031209, 20081216), (20090318, 20090318), (20090812, 20090923)], 1:
↳ [(20200315, 20230726)], 2: [(19930203, 20031028)], 3: [(20090128, 20090128),
↳ (20090429, 20090624), (20091104, 20200129)]}
['pandemic', 'goals', 'facility', 'sheet', 'repo', 'functioning', 'overnight', 'mbs
↳ ', 'remarked', 'billion', 'pre', 'banking', 'covid', 'health', 'guidance',
↳ 'achieve', 'virus', 'balances', 'quality', 'flow']
Topic 3 {0: [(20031209, 20081216), (20090318, 20090318), (20090812, 20090923)], 1:
↳ [(20200315, 20230726)], 2: [(19930203, 20031028)], 3: [(20090128, 20090128),
↳ (20090429, 20090624), (20091104, 20200129)]}
['ranges', 'restraint', 'aggregates', 'final', 'acceptable', 'durable', 'adjustment
↳ ', 'positions', 'direction', 'called', 'prospective', 'behavior', 'finished',
↳ 'appreciable', 'civilian', 'established', 'accordingly', 'growing', 'careful',
↳ 'persisting']
Topic 4 {0: [(20031209, 20081216), (20090318, 20090318), (20090812, 20090923)], 1:
↳ [(20200315, 20230726)], 2: [(19930203, 20031028)], 3: [(20090128, 20090128),
↳ (20090429, 20090624), (20091104, 20200129)]}
['program', 'accommodation', 'couple', 'mandate', 'billion', 'sheet',
↳ 'normalization', 'mbs', 'purchase', 'principal', 'per', 'guidance', 'soma', 'met
↳ ', 'tools', 'reserves', 'approach', 'repurchase', 'facility', 'sovereign']
Topic 1 {0: [(20090128, 20090128), (20090429, 20090624), (20090923, 20190918)], 1:
↳ [(19930203, 20010821), (20011211, 20020319), (20030129, 20030129)], 2:
↳ [(20191030, 20230726)], 3: [(20011002, 20011106), (20020507, 20021210),
↳ (20030318, 20081216), (20090318, 20090318), (20090812, 20090812)]}
['program', 'normalization', 'mandate', 'mbs', 'accommodation', 'billion',
↳ 'purchase', 'couple', 'sheet', 'guidance', 'principal', 'per', 'rrp', 'soma',

```

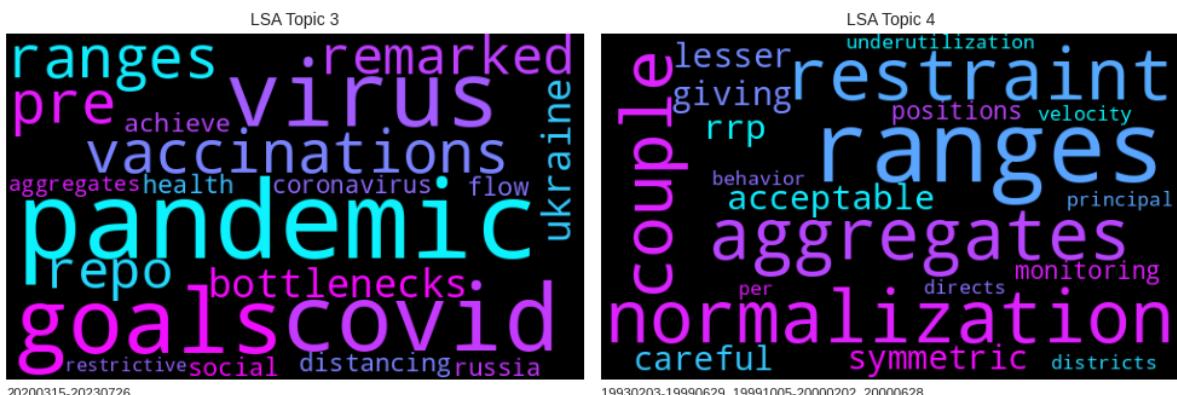
(continues on next page)

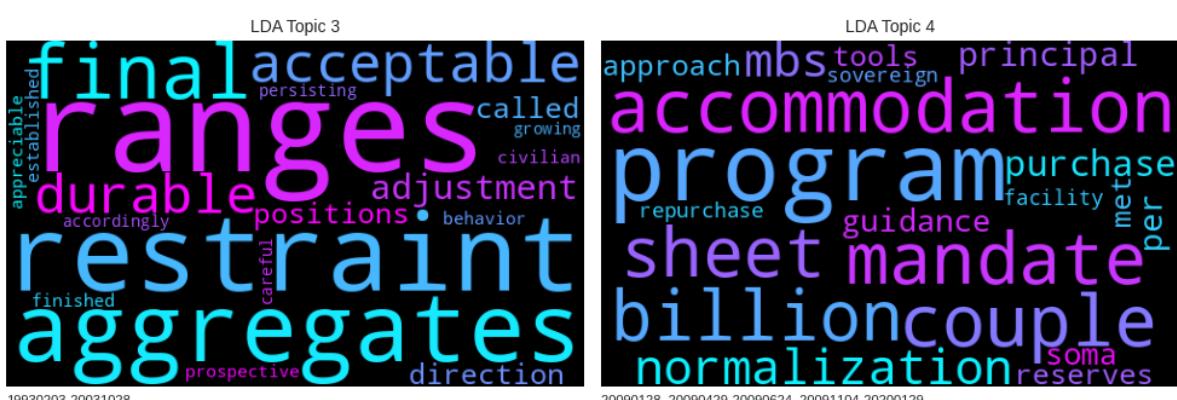
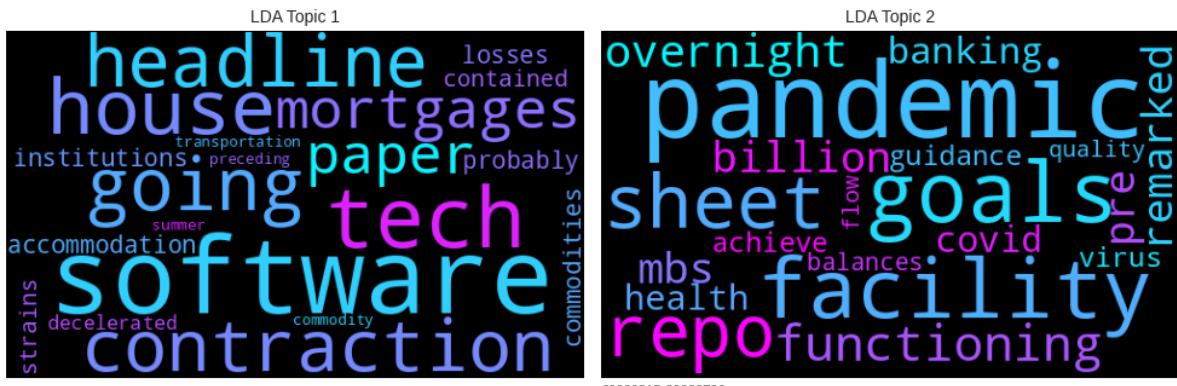
(continued from previous page)

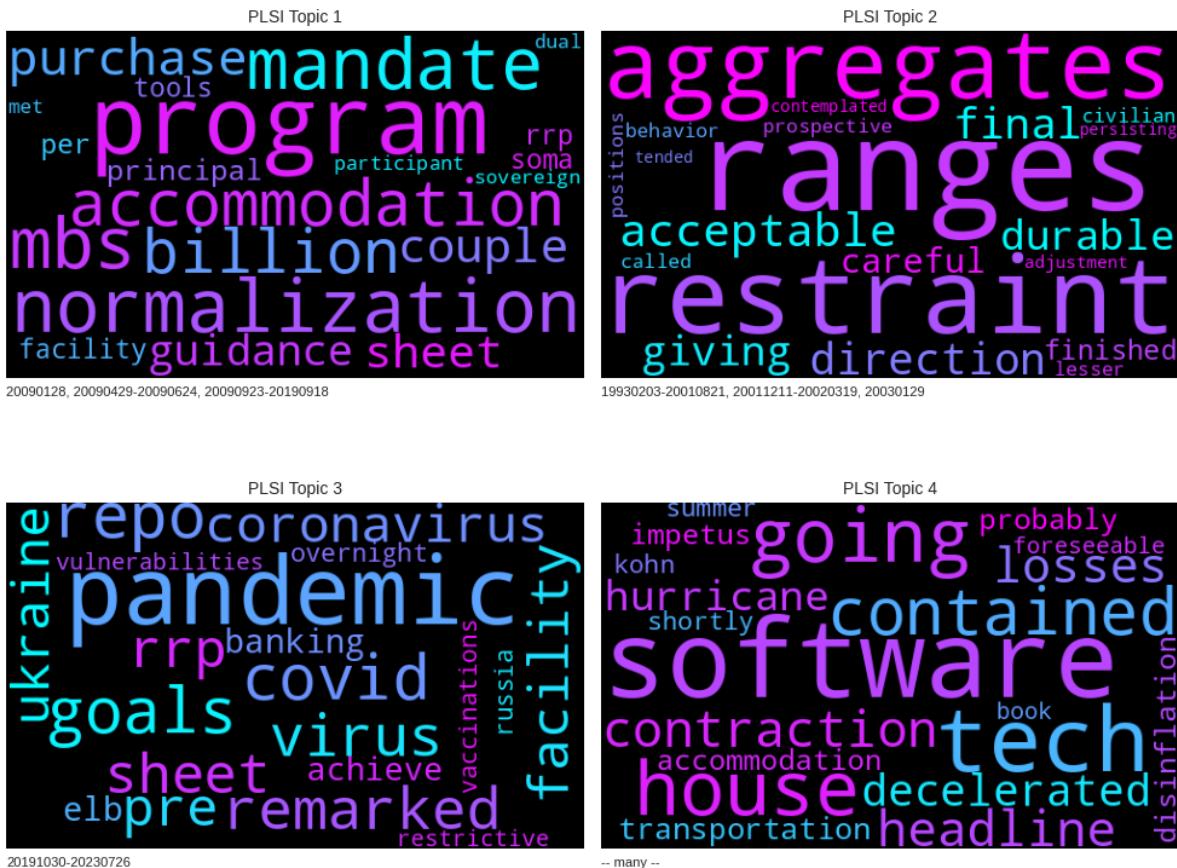
```

↳ 'facility', 'tools', 'participant', 'met', 'dual', 'sovereign']
Topic 2 {0: [(20090128, 20090128), (20090429, 20090624), (20090923, 20190918)], 1:_
↳ [(19930203, 20010821), (20011211, 20020319), (20030129, 20030129)], 2:_
↳ [(20191030, 20230726)], 3: [(20011002, 20011106), (20020507, 20021210),_
↳ (20030318, 20081216), (20090318, 20090318), (20090812, 20090812)]}
['ranges', 'restraint', 'aggregates', 'acceptable', 'direction', 'final', 'durable
↳ ', 'giving', 'careful', 'finished', 'prospective', 'positions', 'civilian',
↳ 'called', 'lesser', 'behavior', 'adjustment', 'contemplated', 'persisting',
↳ 'tended']
Topic 3 {0: [(20090128, 20090128), (20090429, 20090624), (20090923, 20190918)], 1:_
↳ [(19930203, 20010821), (20011211, 20020319), (20030129, 20030129)], 2:_
↳ [(20191030, 20230726)], 3: [(20011002, 20011106), (20020507, 20021210),_
↳ (20030318, 20081216), (20090318, 20090318), (20090812, 20090812)]}
['pandemic', 'goals', 'repo', 'covid', 'virus', 'remarked', 'sheet', 'facility',
↳ 'rrp', 'pre', 'coronavirus', 'ukraine', 'banking', 'elb', 'achieve', 'overnight',
↳ 'russia', 'vaccinations', 'restrictive', 'vulnerabilities']
Topic 4 {0: [(20090128, 20090128), (20090429, 20090624), (20090923, 20190918)], 1:_
↳ [(19930203, 20010821), (20011211, 20020319), (20030129, 20030129)], 2:_
↳ [(20191030, 20230726)], 3: [(20011002, 20011106), (20020507, 20021210),_
↳ (20030318, 20081216), (20090318, 20090318), (20090812, 20090812)]}
['software', 'tech', 'house', 'going', 'contained', 'contraction', 'headline',
↳ 'losses', 'decelerated', 'hurricane', 'transportation', 'probably',
↳ 'accommodation', 'disinflation', 'impetus', 'shortly', 'kohn', 'summer',
↳ 'foreseeable', 'book']
Topic 1 {0: [(20081029, 20200129)], 1: [(19930203, 19990629), (19991005, 19991005),
↳ (19991221, 20000202)], 2: [(20200315, 20230726)], 3: [(19990824, 19990824),_
↳ (19991116, 19991116), (20000321, 20080916)]}
['normalization', 'program', 'mandate', 'mbs', 'couple', 'accommodation', 'billion
↳ ', 'guidance', 'sheet', 'purchase', 'per', 'principal', 'rrp', 'tools', 'soma',
↳ 'participant', 'facility', 'met', 'districts', 'transitory']
Topic 2 {0: [(20081029, 20200129)], 1: [(19930203, 19990629), (19991005, 19991005),
↳ (19991221, 20000202)], 2: [(20200315, 20230726)], 3: [(19990824, 19990824),_
↳ (19991116, 19991116), (20000321, 20080916)]}
['ranges', 'restraint', 'aggregates', 'acceptable', 'giving', 'careful', 'lesser',
↳ 'positions', 'behavior', 'direction', 'contemplated', 'civilian', 'finished',
↳ 'monitoring', 'adjustment', 'durable', 'upper', 'tentative', 'established',
↳ 'velocity']
Topic 3 {0: [(20081029, 20200129)], 1: [(19930203, 19990629), (19991005, 19991005),
↳ (19991221, 20000202)], 2: [(20200315, 20230726)], 3: [(19990824, 19990824),_
↳ (19991116, 19991116), (20000321, 20080916)]}
['pandemic', 'covid', 'goals', 'virus', 'pre', 'vaccinations', 'remarked', 'repo',
↳ 'facility', 'bottlenecks', 'rrp', 'ukraine', 'social', 'achieve', 'distancing',
↳ 'health', 'sheet', 'flow', 'coronavirus', 'guidance']
Topic 4 {0: [(20081029, 20200129)], 1: [(19930203, 19990629), (19991005, 19991005),
↳ (19991221, 20000202)], 2: [(20200315, 20230726)], 3: [(19990824, 19990824),_
↳ (19991116, 19991116), (20000321, 20080916)]}
['software', 'tech', 'foreseeable', 'final', 'press', 'decelerated', 'contained',
↳ 'going', 'profits', 'impetus', 'probably', 'shortly', 'acceleration',
↳ 'disinflation', 'partners', 'house', 'liquidation', 'unit', 'contraction',
↳ 'prospective']

```









---

CHAPTER  
**TWENTYFOUR**

---

## MANAGEMENT SENTIMENT ANALYSIS

- Sentiment Analysis: e.g. Loughran and McDonald (2011) sentiment word list
- SEC Edgar 10-K Company Filings: e.g. Cohen, Malloy and Nguyen (2020),

```
import re
import requests
import time
import numpy as np
import pandas as pd
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
from wordcloud import WordCloud
import sklearn.feature_extraction
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import CountVectorizer
from tqdm import tqdm
from finds.database.sql import SQL
from finds.database.redisdb import RedisDB
from finds.database.mongodb import MongoDB
from finds.structured.crsp import CRSP
from finds.structured.signals import Signals
from finds.unstructured import Unstructured
from finds.unstructured.store import Store
from finds.busday import BusDay
from finds.filters import weighted_average
from finds.backtesting import fractiles
from finds.readers.edgar import Edgar
from finds.readers.alfred import Alfred
from finds.misc.show import Show
from secret import credentials, paths
# %matplotlib qt
VERBOSE = 0
show = Show(ndigits=4, latex=None)
```

```
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)
signals = Signals(user)
mongodb = MongoDB(**credentials['mongodb'], verbose=VERBOSE)
wordlists = Unstructured(mongodb, 'WordLists')
```

(continues on next page)

(continued from previous page)

```
store = Store(paths['scratch'], ext='pkl')
imgdir = paths['images'] / 'edgar'
item, form = 'mda10K', '10-K'

def _print(*args, **kwargs):
    """print messages if VERBOSE"""
    if VERBOSE:
        print(*args, **kwargs)
```

Last FamaFrench Date 2023-06-30 00:00:00

## 24.1 Loughran and MacDonald dictionaries

```
# Load Loughran and MacDonald sentiment words and stopwords
wordlists = Unstructured(mongodb, 'WordLists')
sentiments = {sent: wordlists['lm', sent] for sent in ['positive', 'negative']}
```

```
# Pre-process with sklearn methods
tf_vectorizer = CountVectorizer(
    strip_accents='unicode',
    lowercase=True,
    stop_words=stop_words,
    # tokenizer=CustomTokenizer(),
    token_pattern=r"\b[\^\d\W][^\d\W][^\d\W]+\b")
analyzer = tf_vectorizer.build_analyzer()
```

```
# Construct sentiment feature all years for usual universe
univs = {yr+1: crsp.get_universe(bd.endmo(yr*10000 + 1231)).assign(year=yr+1)
         for yr in range(1992, 2020)}
rows = DataFrame(ed.open(form=form, item=item)) # open mda10K archive
permnos = rows['permno'].unique().astype(int)

tic = time.time()
results = []
for i, permno in tqdm(enumerate(permnos)): # Loop over all permnos

    # retrieve all valid mda's for this permno by year
    mdas = {}
    dates = {}
    files = rows[rows['permno'].eq(permno)].to_dict('records')
    for i, f in enumerate(files):
        year = int(f['date']) // 10000
        if ((f['date'] // 100) % 100) <= 3: # if filing date <= Mar
            year = year - 1 # then assign to previous year
        if (year in univs and
            (year not in mdas or f['date'] < dates[year])):
            tokens = Series(analyzer(ed[f['pathname']]), dtype='object')
            if len(tokens):
                mdas[year] = tokens
                dates[year] = f['date']
```

(continues on next page)

(continued from previous page)

```

# compute sentiment (count frequency) scores for permno by year
sentiment = {year: (mda.isin(sentiments['positive']).sum()
                     - mda.isin(sentiments['negative']).sum()) / len(mda)
             for year, mda in mdas.items()} # compute sentiment by year

# 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 = [" ".join(mdas[year]), " ".join(mdas[year-1])]
        cos = cosine_similarity(tf_vectorizer.fit_transform(corpus))
        result['mdacos'] = cos[0, 1]
    _print(i, int(time.time()-tic), result)
    results.append(result)

```

12632it [21:08, 9.96it/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(data, 'mdacos', overwrite=True))

```

```

(signals_write) mdasent 107105
(signals_write) mdachg 88292
(signals_write) mdacos 88292
107105 88292 88292

```

```

# right join data with univ, to identify univ with missing mda
data = pd.concat([data[data['year']==year]\
                  .drop(columns=['year'])\
                  .set_index('permno')\
                  .join(univ[['year']], how='right')\
                  .reset_index()
                  for year, univ in univs.items() if year <= 2020],
                  ignore_index=True)

```

```

# save sentiment dataframe in scratch folder
store.dump(data, 'sentiment')
data = store.load('sentiment')

```

```

# Stacked Bar Plot of universe coverage by year
y1 = data[data['mdasent'].notna()]\n    .groupby('year')[['permno']]\n    .count()\ny0 = data[data['mdasent'].isna()]\n
```

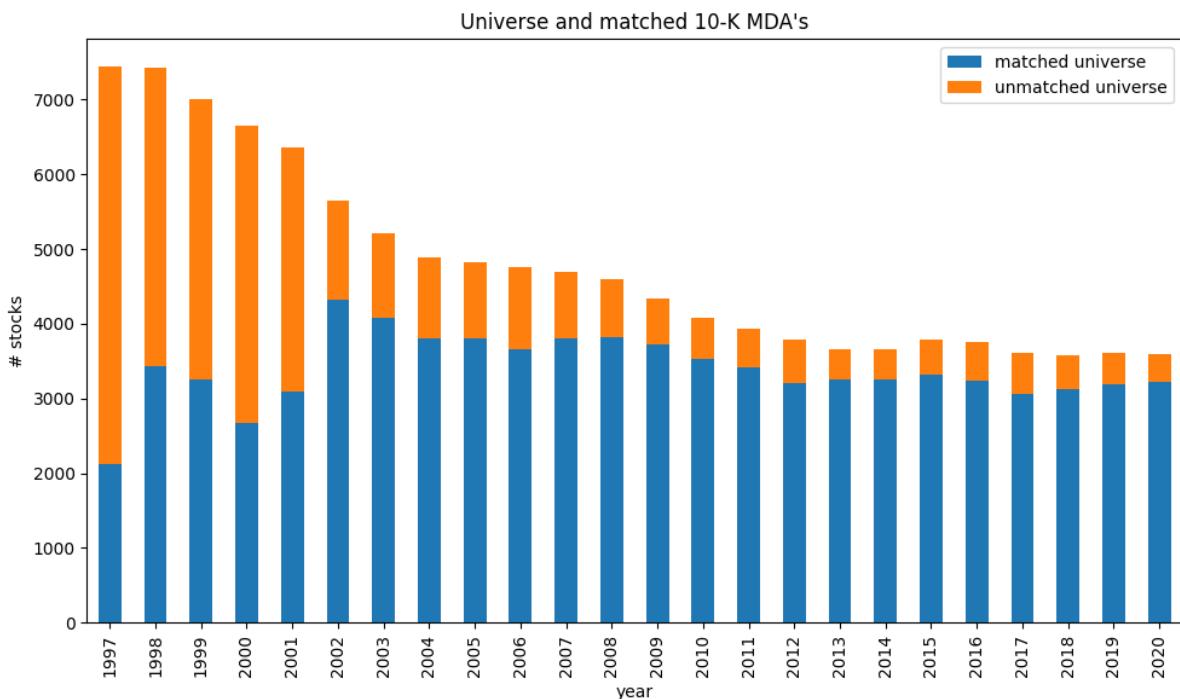
(continues on next page)

(continued from previous page)

```

.groupby('year')['permno']\
.count()\
.reindex(y1.index)
fig, ax = plt.subplots(1, 1, clear=True, num=1, figsize=(10, 6))
y1.plot(kind='bar',
        label='matched universe',
        color='C0',
        ax=ax,
        rot=90)
y0.plot(kind='bar',
        label='unmatched universe',
        color='C1',
        ax=ax,
        rot=90,
        bottom=y1)
ax.set_ylabel('# stocks')
ax.set_title("Universe and matched 10-K MDA's")
ax.legend()
plt.tight_layout()
plt.savefig(imgdir / 'coverage.jpg')

```



```

# Stacked Bar Plot of filings date, by month and day-of-week
y = DataFrame.from_records([{'date': int(d),
                            'day': bd.datetime(int(d)).strftime('%A'),
                            'month': bd.datetime(int(d)).strftime('%B')}
                           for d in data.loc[data['mdasent'].notna(), 'date']])
z = pd.concat([y['month'].value_counts()/len(y),
               y['day'].value_counts()/len(y)])
fig, ax = plt.subplots(1, 1, clear=True, num=1, figsize=(10, 6))
z.plot(kind='bar',
        color='C0',
        ax=ax)

```

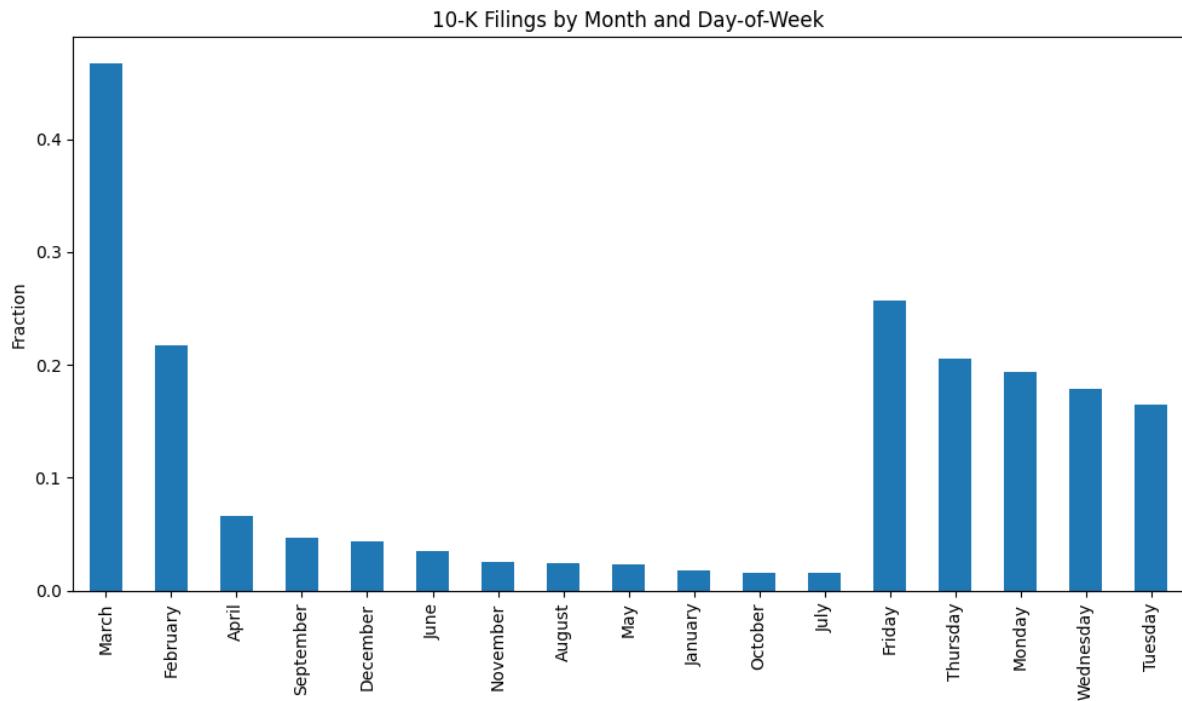
(continues on next page)

(continued from previous page)

```

    ax=ax,
    rot=90)
ax.set_ylabel('Fraction')
ax.set_title("10-K Filings by Month and Day-of-Week")
plt.tight_layout()
plt.savefig(imgdir / 'calendar.jpg')

```



### Plot distributions of sentiment, change, similarity univ by year

- visualize with economy-wide profitability (CP Corporate Profits from FRED) Align year as caldate, e.g. for caldate year 2019:
- filings from April 2019 to Mar 2020 (i.e. year = filing year-1 if month<=3)
- universe is as of year end prior to filing date
- economic time series is average annual value ending Dec 2019
- concurrent return year is Jan 2019 to Dec 2019
- next return year (i.e. lagged filings) is April 2020-Mar 2021 #data = data.dropna(subset=['date'])

```

alf = Alfred(api_key=credentials['fred']['api_key'], verbose=VERBOSE)
series_id = 'CP' # Corporate Profits
#series_id = 'UNRATE'
#series_id = 'WILL5000IND'
econ = alf(series_id)
econ = econ.to_frame() \
    .assign(year=econ.index // 10000) \
    .groupby('year') \
    .mean()

for i, sent in enumerate(['mdasent', 'mdachg', 'mdacos']):

```

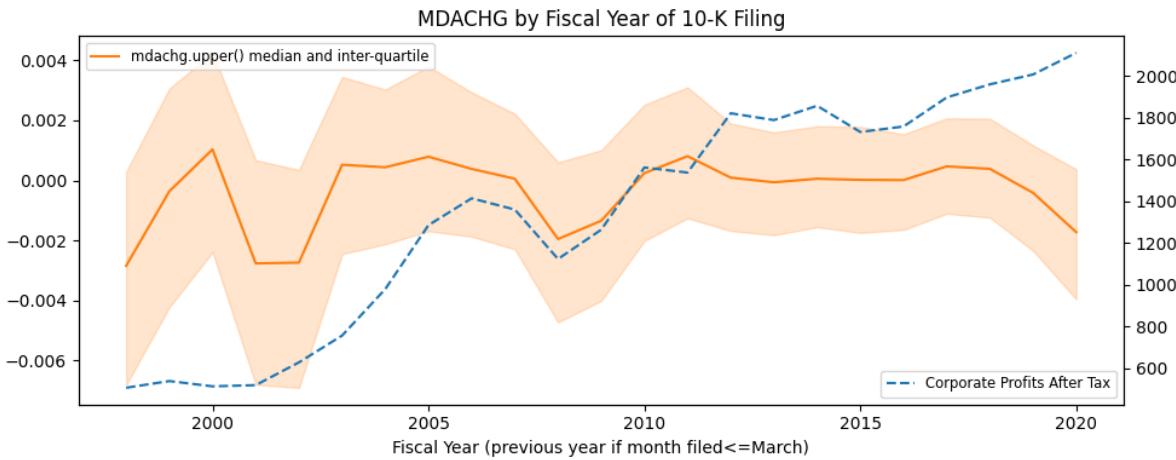
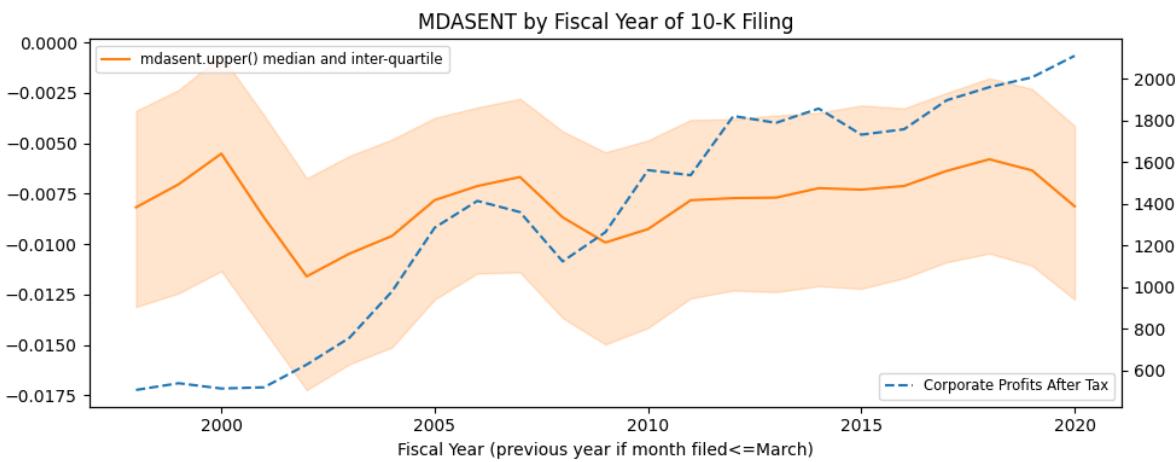
(continues on next page)

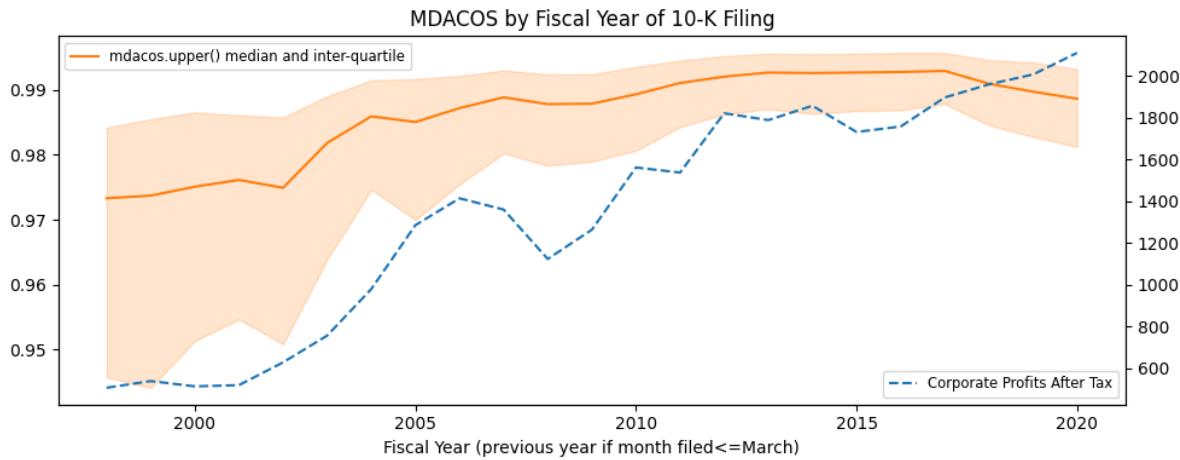
(continued from previous page)

```

g = data[data['currlen'].gt(500)].dropna().groupby('year')
iq1, iq2, iq3 = [g[sent].quantile(p) for p in [.25, .5, .75]]
y = iq2.index.astype(int)
fig, ax = plt.subplots(1, 1, clear=True, num=1+i, figsize=(10, 4))
ax.plot(y, iq2, ls='-', color='C1')
ax.fill_between(y, iq1, iq3, alpha=0.2, color='C1')
ax.set_title(f'{sent.upper()} by Fiscal Year of 10-K Filing')
ax.set_xlabel("Fiscal Year (previous year if month filed<=March)")
ax.legend([f'{sent}.upper() median and inter-quartile'],
          fontsize='small',
          loc='upper left')
#econ.index = ax.get_xticks() # boxplot(by) has sequential xticks
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')
plt.tight_layout()
plt.savefig(imgdir / f'{sent}.jpg')

```





### Quintile cap-weighted spread portfolio weights

- same year filings [yr]0101:[yr]1231 = bd.begyr(caldate) to caldate
- lagged [yr+1]0401:[yr+2]0331 = bd.begmo(caldate,4) - bd.endmo(caldate,15)

```

for ifig, key in enumerate(['mdasent', 'mdachg', 'mdacos']):
    ret1 = {} # to collect year-ahead spread returns
    ret0 = {} # to collect current-year spread returns
    for year in sorted(np.unique(data['year'])): # loop over years

        # compute current year spread returns
        beg = bd.begyr(year)
        end = bd.endyr(year)
        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, delist=True), how='left')

        if len(univ):
            sub = fractiles(univ[key], [20, 80])
            pos = weighted_average(univ.loc[sub==1, ['cap', 'ret']],
                                   'cap')['ret']
            neg = weighted_average(univ.loc[sub==3, ['cap', 'ret']],
                                   'cap')['ret']
            ret0[end] = {'ret': pos - neg,
                         'npos': sum(sub==1),
                         'nneg': sum(sub==3)}
            _print(end, len(univ), pos, neg)

        # compute year ahead spread returns
        beg = bd.begmo(end, 4)
        end = bd.endmo(end, 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, delist=True), how='left')

        if len(univ):
            sub = fractiles(univ[key], [20, 80])
            pos = weighted_average(univ.loc[sub==1, ['cap', 'ret']],
                                   'cap')['ret']

```

(continues on next page)

(continued from previous page)

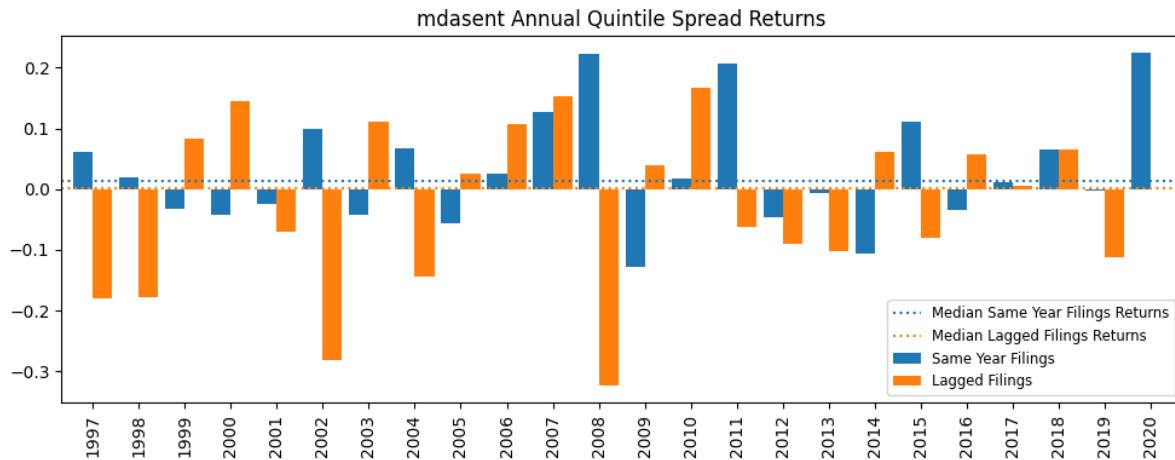
```

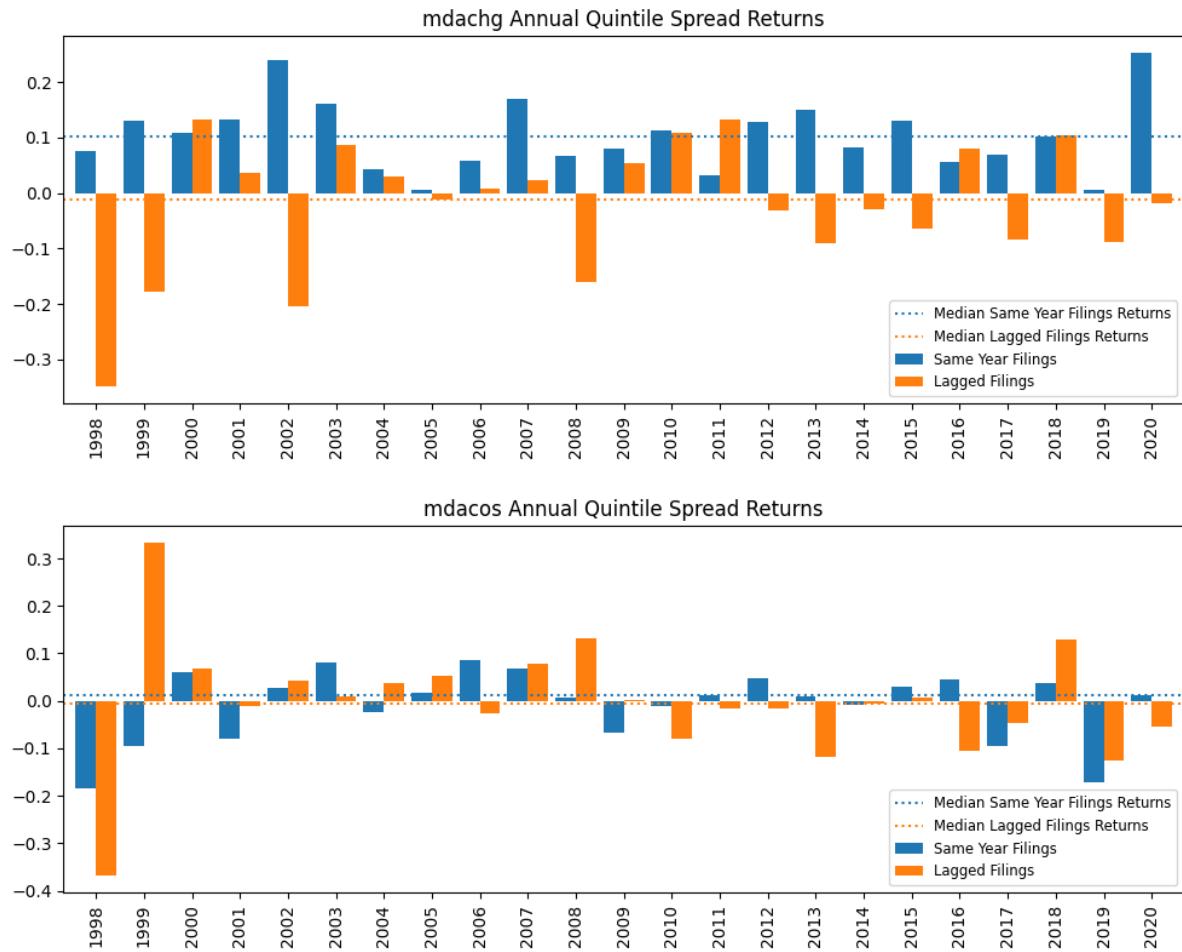
neg = weighted_average(univ.loc[sub==3, ['cap', 'ret']],
                      'cap')['ret']
ret1[end] = {'ret': pos - neg,
             'npos': sum(sub==1),
             'nneg': sum(sub==3)}
print(end, len(univ), pos, neg)

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

fig, ax = plt.subplots(1, 1, clear=True, num=1+ifig, figsize=(10, 4))
pd.concat([r0['ret'].rename('Same Year Filings').to_frame(),
           r1['ret'].rename('Lagged Filings').to_frame()],
           join='outer',
           axis=1).plot(kind='bar',
                         ax=ax,
                         width=.85)
ax.set_title(f"{{key}} Annual Quintile Spread Returns")
ax.axhline(r0['ret'].median(), linestyle=':', color='C0',
           label='Median Same Year Filings Returns')
ax.axhline(r1['ret'].median(), linestyle=':', color='C1',
           label='Median Lagged Filings Returns')
# ax.legend(['Median Same Year Filings Returns',
#           'Median Lagged Filings Returns',
#           'Same Year Filings Returns',
#           'Lagged Filings Returns'],
#           ax.legend(fontsize='small',
#                     loc='lower right')
plt.tight_layout()
plt.savefig(imgdir / f'{key}RET.jpg')

```







## BUSINESS TEXT ANALYSIS

### UNDER CONSTRUCTION

- Spacy
- Syntactic analysis, POS tags, named entity recognition
- Logistic regression, Perceptron, stochastic gradient descent
- Growth and Value stocks

```
import re
import json
import gzip
import requests
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 sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.linear_model import LogisticRegression, LogisticRegressionCV
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from wordcloud import WordCloud
import spacy
from tqdm import tqdm
from finds.database.sql import SQL
from finds.database.redisdb import Redis
from finds.database.mongodb import MongoDB
from finds.structured.crsp import CRSP
from finds.structured.pstat import PSTAT
from finds.structured.benchmarks import Benchmarks
from finds.structured.signals import Signals
from finds.backtesting.backtest import BackTest
from finds.busday import BusDay
from finds.unstructured import Unstructured
from finds.unstructured.store import Store
from finds.readers.sectoring import Sectoring
from finds.readers.edgar import Edgar
from finds.misc.show import Show
from finds.plots import plot_date
from secret import credentials, paths
# %matplotlib qt
VERBOSE = 0
show = Show(ndigits=4, latex=None)
```

```
sql = SQL(**credentials['sql'], verbose=VERBOSE)
user = SQL(**credentials['user'], verbose=VERBOSE)
bd = BusDay(sql)
rdb = Redis(**credentials['redis'])
crsp = CRSP(sql, bd, rdb, verbose=VERBOSE)
pstat = PSTAT(sql, bd, verbose=VERBOSE)
bench = Benchmarks(sql, bd, verbose=VERBOSE)
ed = Edgar(paths['10X'], zipped=True, verbose=VERBOSE)
imgdir = paths['images'] / 'edgar'
store = Store(paths['scratch'], ext='pkl')
item, form = 'bus10K', '10-K'
```

## 10-K BUSINESS DESCRIPTIONS

```
# Retrieve universe of stocks
# 5-year growth and book-to-price, by 2022, 1997, 1972
# NYSE top-half market cap
univ = crsp.get_universe(20181231)

lookup = crsp.build_lookup('permno', 'comnam', fillna="") # company name
comnam = lookup(univ.index)
univ['comnam'] = comnam

lookup_sic = pstat.build_lookup('lpermno', 'sic', fillna=0) # sic from PSTAT
sic_ = Series(lookup_sic(univ.index, date=20181231), univ.index)
univ['siccd'] = univ['siccd'].where(sic_.isin([0, 9999]), sic_)

lookup_naics = pstat.build_lookup('lpermno', 'naics', fillna=0) # naics from PSTAT
naics_ = Series(lookup_naics(univ.index, date=20181231), univ.index)
univ['naics'] = univ['naics'].where(sic_.isin([0, 9999]), naics_)

# Retrieve business descriptions text; extract nouns from POS tags
nlp = spacy.load("en_core_web_lg") # Load a spaCy language pipeline
if 'bus' not in store: # store processed text if necessary
    rows = DataFrame(ed.open(form=form, item=item)) # open bus10K archive
    bus = {}
    restart = 0
    for i, permno in tqdm(enumerate(univ.index)):
        found = rows[rows['permno'].eq(permno) & rows['date'].between(20190101, 20190331)]
        if len(found) and i >= restart:
            doc = nlp(found.iloc[0]['pathname'][:nlp.max_length].lower())
            bus[permno] = " ".join([re.sub("[^a-zA-Z]+", "", token.lemma_) for token in doc if token.pos_ in ['NOUN'] and len(token.lemma_) > 2])
        store.dump(bus, 'bus') # serialize
    bus = store.load('bus')
    keys = list(bus.keys())
    corpus = list(bus.values())
```



## BAG-OF-WORDS TF-IDF

```
vectorizer = TfidfVectorizer(max_df=0.5, min_df=10)
tfidf = vectorizer.fit_transform(corpus)
X = tfidf
```

### Retrieve Fama-French sector scheme

```
# populate codes49 industry, company name, and legacy sector
codes = Sectoring(sql, scheme='codes49', fillna="")      # codes49 industry
sic = Sectoring(sql, scheme='sic', fillna=0)
codes49 = Series(codes[univ['siccd']])
replace = univ['siccd'].isin([0, 9999]).values
codes49[replace] = codes[sic[univ.loc[replace, 'naics']]]
univ['industry'] = codes49.values
```

```
codes12 = Sectoring(sql, scheme='codes12', fillna="")  # [5,10,12,17,30,38,48,49]
sic = Sectoring(sql, scheme='sic', fillna=0)      # cross-walk naics to sic
legacy = Series(codes12[univ['siccd']])           # convert sic to legacy sector
replace = (legacy.eq("").values | univ['siccd'].isin([0, 9999]).values)
legacy[replace] = codes12[sic[univ.loc[replace, 'naics']]] # convert naics
univ['legacy'] = legacy.tolist()
y = univ['legacy'].reindex(keys)
print(y.groupby(y).count().to_string())
```

## 27.1 Logistic Regression

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  $y \in \{0, 1\}$

## 27.2 Perceptron

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  $y \in \{-1, +1\}$

## 27.3 Accuracy

- confusion matrix, precision, recall
- auc, roc

```
## Confusion Matrix
print(confusion_matrix(y, res.clf.predict(X)))
```

```
fig, ax = plt.subplots(figsize=(10, 6))
ConfusionMatrixDisplay.from_predictions(y, res.clf.predict(X), ax=ax)
fig.tight_layout()
plt.savefig(imgdir / 'logistic_cf.jpg')
```

```
fig, ax = plt.subplots(figsize=(5, 3)) # accuracy vs alpha
ax.semilogx(res.Cs, res.valid_accuracy, #, drawstyle="steps-post")
ax.semilogx(res.Cs, res.train_accuracy, #, drawstyle="steps-post")
argmax = np.argmax(res.valid_accuracy)
ax.annotate(f"res.valid_accuracy[{argmax}]:.4f",
            xy=(res.Cs[argmax], res.valid_accuracy[argmax]))
ax.plot(res.Cs[argmax], res.valid_accuracy[argmax], "o")
ax.set_xlabel("Regularization parameter (C)")
ax.set_ylabel("accuracy")
ax.set_title("Softmax Regression: Accuracy vs Complexity")
ax.legend(['Cross-Validation Accuracy', 'Training Accuracy'])
plt.tight_layout()
plt.savefig(imgdir / 'logistic.jpg')
```

## 27.4 Feature importances

```
top_n = 20
words = {}
feature_names = vectorizer.get_feature_names_out()
for topic, lab in enumerate(res.clf.classes_):
    importance = res.clf.coef_[topic, :]
    words[lab] = [feature_names[i]
                  for i in importance.argsort()[:-top_n-1:-1]]
    freqs = {feature_names[i]: importance[i]
              for i in importance.argsort()[:-top_n-1:-1]}
```

(continues on next page)

(continued from previous page)

```
fig, ax = plt.subplots(figsize=(3.5, 3), clear=True)
wc = WordCloud(height=500, width=500, colormap='cool')
ax.imshow(wc.generate_from_frequencies(freqs))
ax.axis("off")
ax.set_title(lab)
plt.tight_layout()
plt.savefig(imgdir / f"logistic_wc{topic}.jpg")
out = DataFrame.from_dict(words, orient='columns')
show(out, index=False, **SHOW)
```



---

CHAPTER  
**TWENTYEIGHT**

---

## PRINCIPAL CUSTOMERS NETWORK

### UNDER CONSTRUCTION

- Graphs: ego network, induced subgraph
- Supply chain: principal customers

```
import numpy as np
import pandas as pd
import networkx as nx
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
from pandas.api import types
import numpy.ma as ma
from numpy.ma import masked_invalid as valid
from finds.database import SQL
from finds.graph import graph_info, graph_draw, nodes_centrality
from finds.misc import Show
from secret import paths, credentials
# %matplotlib qt
VERBOSE = 0
show = Show(ndigits=4, latex=None)
sql = SQL(**credentials['sql'], verbose=VERBOSE)
imgdir = paths['images'] / 'supplychain'
```

```
# Retrieve principal customers info
year = 2016
cust = sql.read_dataframe(f"select gvkey, cgvkey, stic, ctic, conm, cconnm "
                           f"  from customer "
                           f"  where srcdate >= {year}0101 "
                           f"    and srcdate <= {year}1231")
```

```
# 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()
```

```
# Construct Directed Graph
vertices = set(cust['stic']).union(cust['ctic'])
edges = cust[['stic', 'ctic']].values.tolist()  # supplier --> customer

G = nx.DiGraph()
```

(continues on next page)

(continued from previous page)

```
G.add_nodes_from(vertices)
G.add_edges_from(edges)
```

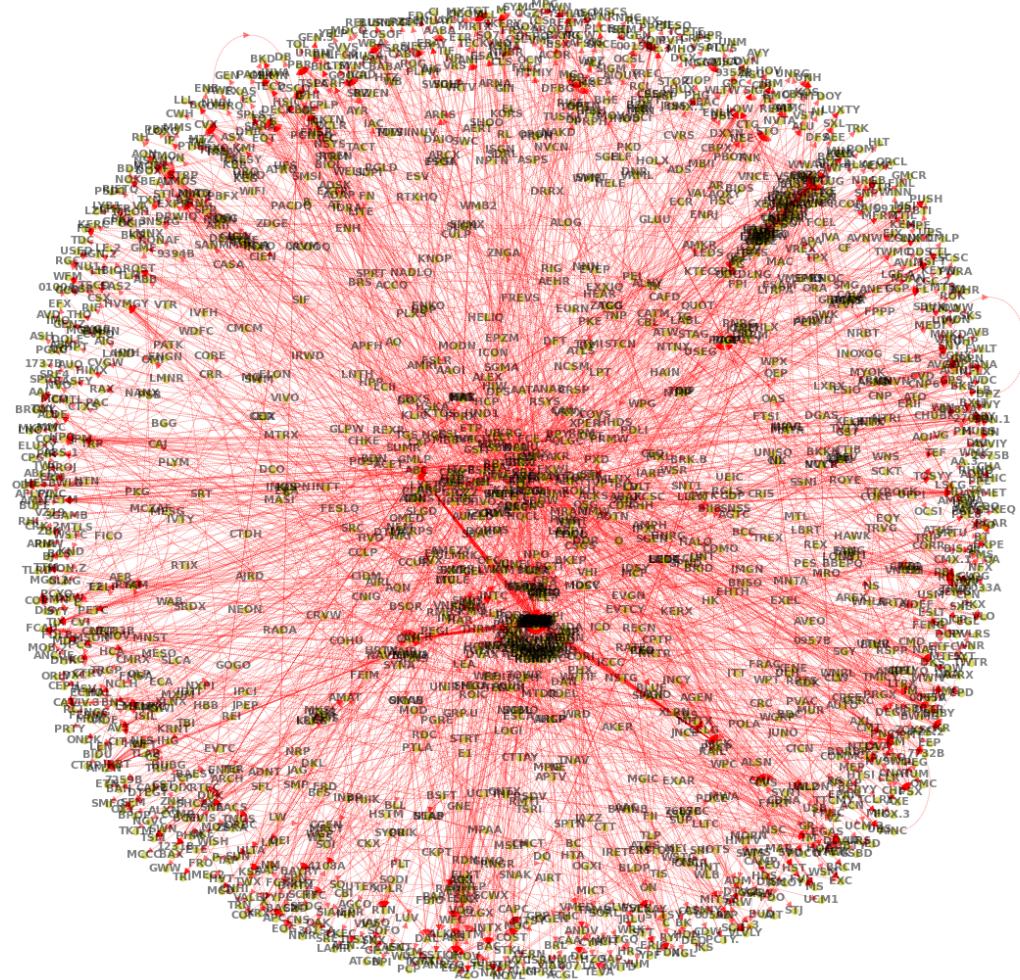
### 1. Show graph properties

```
graph_info(G)
```

```
{'triad_003': 1014214260,
 'triad_012': 5493913,
 'triad_102': 1826,
 'triad_021D': 7858,
 'triad_021U': 21706,
 'triad_021C': 1036,
 'triad_111D': 0,
 'triad_111U': 2,
 'triad_030T': 59,
 'triad_030C': 0,
 'triad_201': 0,
 'triad_120D': 0,
 'triad_120U': 0,
 'triad_120C': 0,
 'triad_210': 0,
 'triad_300': 0,
 'transitivity': 0.0037252178305341582,
 'average_clustering': 0.004935791976903955,
 'weakly_connected': False,
 'weakly_connected_components': 89,
 'size_largest_weak_component': 1628,
 'strongly_connected': False,
 'strongly_connected_components': 1829,
 'size_largest_strong_component': 2,
 'directed': True,
 'weighted': False,
 'edges': 3052,
 'nodes': 1830,
 'selfloops': 11,
 'density': 0.0009118422978903937}
```

### 2. Display graph

```
pos = graph_draw(G,
                  figsize=(12, 12),
                  savefig=imgdir / 'graph.jpg',
                  font_color='k',
                  node_color='y')
plt.show()
```



```
# 3. Node properties
G.remove_edges_from(nx.selfloop_edges(G)) # remove self-loops, if any
```



---

CHAPTER  
**TWENTYNINE**

---

## **NODES CENTRALITY**

- show top nodes' centrality properties

```
centrality = DataFrame.from_dict(nodes_centrality(G))
n = 5
for c in centrality.columns:
    df = centrality[[c]].sort_values(by=c, ascending=False) [:n]
    print(pd.concat((lookup[df.index].rename('name'), df), axis=1))
```

		name	clustering
ODC	OIL DRI CORP	AMERICA	0.5
FORM		FORMFACTOR INC	0.5
FVE	FIVE STAR SENIOR	LIVING INC	0.5
MKC		MCCORMICK & CO INC	0.5
PTR	PETROCHINA CO	LTD	0.5
		name	in_degree
WMT		WALMART INC	0.062329
MCK		MCKESSON CORP	0.022963
RDS.A	ROYAL DUTCH SHELL	PLC	0.022963
CAH	CARDINAL HEALTH	INC	0.022417
ABC	AMERISOURCEBERGEN	CORP	0.019683
		name	out_degree
AKAM	AKAMAI TECHNOLOGIES	INC	0.015856
DHX	DHI GROUP	INC	0.013669
KRG	KITE REALTY GROUP	TRUST	0.011482
NPO	ENPRO INDUSTRIES	INC	0.011482
CBL	CBL & ASSOCIATES	PPTYS INC	0.010935
		name	eigenvector
ZIOP	ZIOPHARM ONCOLOGY	INC	0.500000
XON	INTREXON CORP		0.500000
FCSC	FIBROCELL SCIENCE	INC	0.500000
OGEN	ORAGENICS INC		0.500000
AET	AETNA INC		0.000835
		name	pagerank
WMT	WALMART INC		0.025525
AET	AETNA INC		0.020512
CVS	CVS HEALTH CORP		0.019657
MCK	MCKESSON CORP		0.011308
CAH	CARDINAL HEALTH INC		0.010130
		name	hub
KRG	KITE REALTY GROUP	TRUST	0.019795
BRX1	BRIXMOR OPERATING	PRTNRS LP	0.018452
BRX	BRIXMOR PROPERTY	GROUP INC	0.018452
DDR	DDR CORP		0.014801

(continues on next page)

(continued from previous page)

```

RPT  RAMCO-GERSHENSON PROPERTIES  0.014442
      name  authority
WMT      WALMART INC  0.117030
BBY      BEST BUY CO INC  0.035143
DKS  DICKS SPORTING GOODS INC  0.033133
BBBY  BED BATH & BEYOND INC  0.031484
PETM  PETSMART INC  0.029829
      name  betweenness
ABC  AMERISOURCEBERGEN CORP  0.000101
CAH  CARDINAL HEALTH INC  0.000068
CVS  CVS HEALTH CORP  0.000047
ESRX  EXPRESS SCRIPTS HOLDING CO  0.000038
MCK  MCKESSON CORP  0.000037
      name  closeness
WMT  WALMART INC  0.063601
MCK  MCKESSON CORP  0.040170
CAH  CARDINAL HEALTH INC  0.036602
CVS  CVS HEALTH CORP  0.036326
ABC  AMERISOURCEBERGEN CORP  0.035619

```

## 29.1 Longest path

## 29.2 Ego graph

- induce ego-graph of max betweenness node and neighbors

```

c = 'betweenness'
center = centrality.index[np.argmax(centrality[c])]
all_neighbors = list(nx.all_neighbors(G, center))  # predecessors and successors
neighbors = list(nx.neighbors(G, center))  # successors only
ego = G.subgraph([center] + all_neighbors).copy()
graph_info(ego, fast=True)

```

```

{'weakly_connected': True,
 'weakly_connected_components': 1,
 'size_largest_weak_component': 39,
 'strongly_connected': False,
 'strongly_connected_components': 39,
 'size_largest_strong_component': 1,
 'directed': True,
 'weighted': False,
 'edges': 41,
 'nodes': 39,
 'selfloops': 0,
 'density': 0.02766531713900135}

```

```

node_color = (dict.fromkeys(all_neighbors, 'b')
              | dict.fromkeys(neighbors, 'g')
              | {center: 'cyan'})
labels = ({ticker: ticker for ticker in ego.nodes}

```

(continues on next page)

(continued from previous page)

```

| {ticker: lookup[ticker] for ticker in [center] + neighbors})

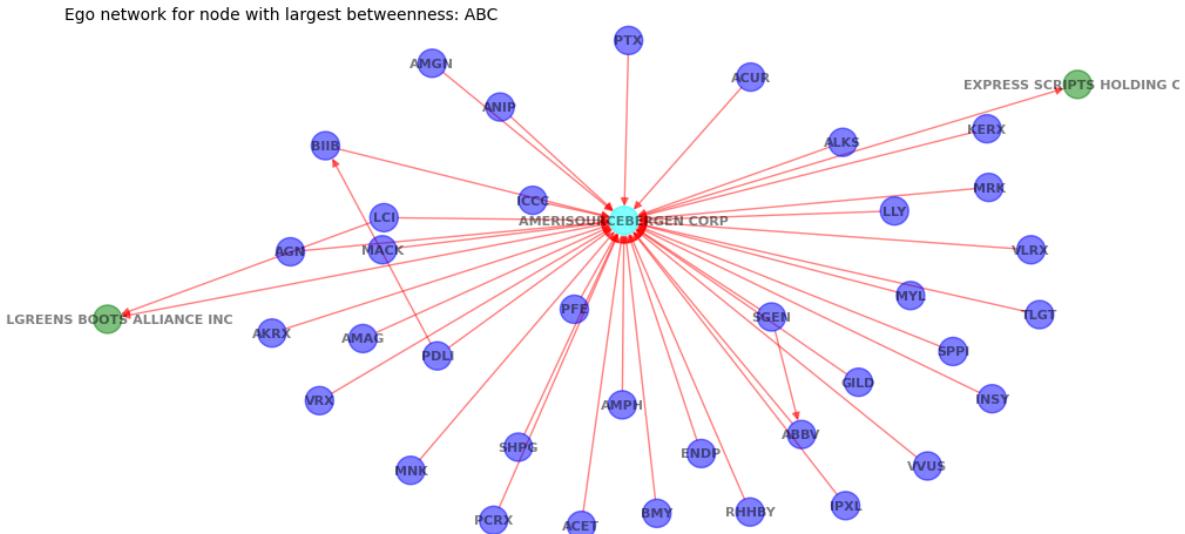
graph_draw(ego,
    figsize=(10, 5),
    savefig=imgdir / f" {center}.png",
    node_size=300,
    width=1,
    node_color=node_color,
    labels=labels,
    style='-' ,
    title=f"Ego network for node with largest {c}: {center}")

```

```

{'SHPG': array([-0.25082933, -0.35567791]),
 'WBA': array([-1.          , -0.03256569]),
 'BMY': array([ 0.00178938, -0.52295215]),
 'LLY': array([0.43506672, 0.24086706]),
 'ACUR': array([0.17293448, 0.57909669]),
 'MRK': array([0.60701898, 0.30035638]),
 'VVUS': array([ 0.49540816, -0.40330805]),
 'PTX': array([-0.04990963, 0.67154546]),
 'IPXL': array([ 0.3459714 , -0.50353382]),
 'ENDP': array([ 0.08266948, -0.37185982]),
 'PFE': array([-0.14875308, -0.00745675]),
 'TLGT': array([ 0.69966816, -0.02100429]),
 'AMGN': array([-0.40817628, 0.61391159]),
 'ABBV': array([ 0.26554787, -0.32381893]),
 'BIIIB': array([-0.60259212, 0.40572203]),
 'AMPH': array([-0.06136585, -0.24890341]),
 'AGN': array([-0.66651497, 0.13778222]),
 'ESRX': array([0.76898448, 0.56049694]),
 'PDLI': array([-0.39867019, -0.12518804]),
 'RHHBY': array([ 0.17165874, -0.51965105]),
 'SPPI': array([ 0.54197759, -0.11407883]),
 'KERX': array([0.60375536, 0.4485514 ]),
 'ANIP': array([-0.2840836 , 0.50338873]),
 'LCI': array([-0.49590841, 0.22394769]),
 'PCRX': array([-0.29596525, -0.54054871]),
 'ACET': array([-0.13594692, -0.55326711]),
 'SGEN': array([ 0.21083251, -0.02688948]),
 'MNK': array([-0.44720782, -0.41432912]),
 'AMAG': array([-0.53428029, -0.08141577]),
 'MYL': array([0.46438575, 0.02716273]),
 'ICCC': array([-0.22462841, 0.26772037]),
 'ABC': array([-0.05812201, 0.21695896]),
 'GILD': array([ 0.36950431, -0.19184001]),
 'INSY': array([ 0.61316937, -0.23343322]),
 'MACK': array([-0.4987529, 0.1428751]),
 'AKRX': array([-0.70008699, -0.06749587]),
 'VRX': array([-0.6135401 , -0.23826206]),
 'VLRX': array([0.68409957, 0.1428459 ]),
 'ALKS': array([0.34089183, 0.41425084])}

```



## GRAPH CENTRALITY

### UNDER CONSTRUCTION

- Centrality: eigenvector, hub, authority, pagerank,
- BEA: Input-Output Use Table, e.g. Choi and Foerster (2017)

```
import time
import numpy as np
import pandas as pd
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
import networkx as nx
from finds.database import RedisDB
from finds.readers import Sectoring, BEA
from finds.graph import graph_info, nodes_centrality, graph_draw
from finds.misc import Show
from secret import credentials, paths
# %matplotlib qt
VERBOSE = 0
show = Show(ndigits=4, latex=None)
```

```
LAST_YEAR = 2021
years = np.arange(1947, LAST_YEAR)
vintages = [1997, 1963, 1947]    # when sectoring schemes were revised
rdb = RedisDB(**credentials['redis'])
bea = BEA(rdb, **credentials['bea'], verbose=VERBOSE)
imgdir = paths['images'] / 'bea'
```

Read IOUse tables from BEA website

```
ioUses = dict()
for vintage in vintages:
    for year in [y for y in years if y >= vintage]:
        df = bea.read_ioUse(year, vintage=vintage)
        ioUses[(vintage, year)] = df
print(f"{len(ioUses)} tables through sectoring vintage year {vintage}")
```

```
24 tables through sectoring vintage year 1997
82 tables through sectoring vintage year 1963
156 tables through sectoring vintage year 1947
```

```
## 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']
yearc = {} # collect annual table
```

Populate and plot graph of first and last table years

```
ifig, year = 0, 1947
vintage = 1947
year0 = 1947
#vintage = 1997
#year0 = 2019
year1 = 2020
for ifig, year in enumerate([year0, year1]):
    # keep year, drop invalid rows
    ioUse = ioUses[(vintage, 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.bea_industry_[list(G.nodes)].to_dict()

    # update master table industry flow values
    master = master.join(data.groupby(['colcode'])['datavalue'].sum(),
                          how='outer').rename(columns={'datavalue': 'user'})
    master = master.join(data.groupby(['rowcode'])['datavalue'].sum(),
                          how='outer').rename(columns={'datavalue': 'maker'})
    master = master.fillna(0).astype(int)
    # inweight~supply~authority~eigenvector~pagerank, outweight~demand~hub

    centrality = DataFrame(nodes_centrality(G))
    master = master.join(centrality, how='left')
    master['bea'] = BEA.bea_industry_[master.index].to_list()
    yearc[year] = master[['pagerank', 'bea']].set_index('bea')

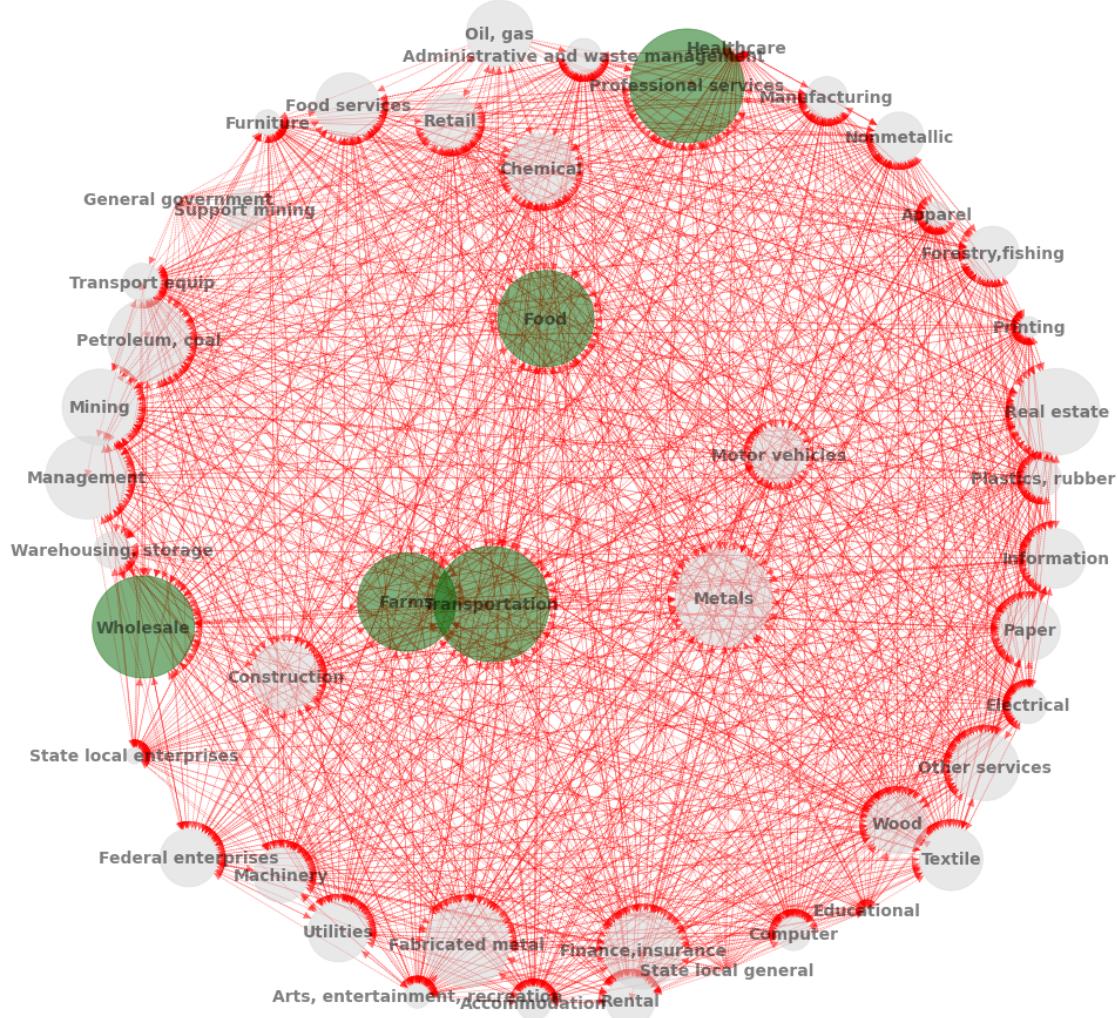
    # visualize graph
    score = centrality['pagerank']
    node_size = score.to_dict()
    node_color = {node: colors[0] for node in G.nodes()}
    if ifig == 0:
        center_name = score.index[score.argmax()]
    else:
```

(continues on next page)

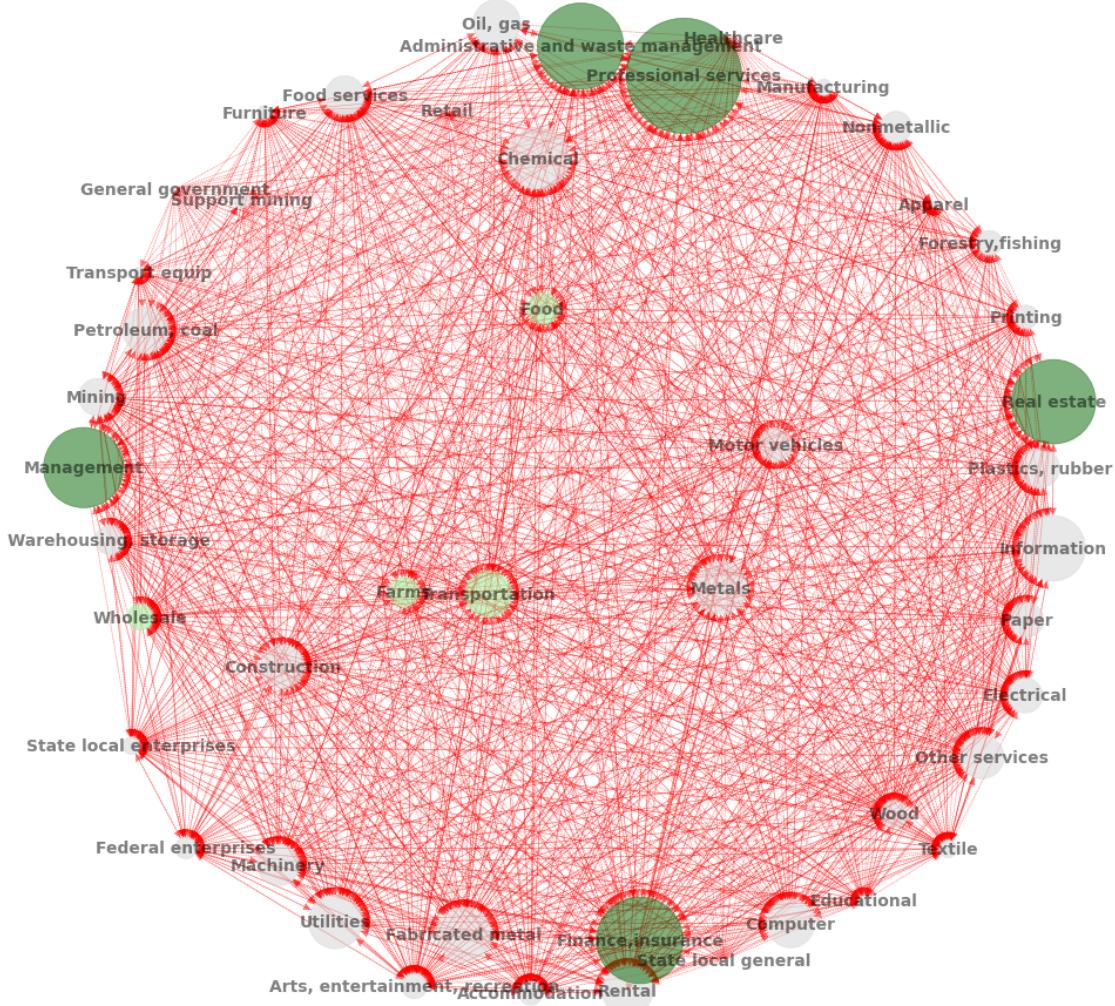
(continued from previous page)

```
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"By Pagerank: {year} IO-Use ({vintage} vintage)")
if imgdir:
    plt.savefig(imgdir / f"{year}.jpg")
```

By Pagerank: 1947 IO-Use (1947 vintage)



By Pagerank: 2020 IO-Use (1947 vintage)



```
## Display node centrality
c = pd.concat([yearc[year0].rank(ascending=False).astype(int),
               yearc[year1].rank(ascending=False).astype(int)],
              axis=1)
c.columns = pd.MultiIndex.from_product([[year0, year1], yearc[year0].columns])
c
```

	1947	2020
	pagerank	pagerank
bea		
Farms	4	27
Forestry, fishing	27	32
Oil, gas	16	13
Mining	12	21
Support mining	31	38

(continues on next page)

(continued from previous page)

Utilities	20	9
Construction	14	17
Food	5	25
Textile	18	36
Apparel	39	43
Wood	21	29
Paper	19	23
Printing	42	33
Petroleum, coal	10	14
Chemical	13	7
Plastics, rubber	33	19
Nonmetallic	28	26
Metals	6	10
Fabricated metal	8	8
Machinery	26	18
Computer	38	12
Electrical	35	22
Motor vehicles	24	24
Transport equip	32	42
Furniture	40	40
Manufacturing	30	39
Wholesale	3	30
Retail	25	45
Transportation	1	15
Warehousing, storage	34	28
Information	22	6
Finance, insurance	11	2
Real estate	7	4
Rental	29	11
Professional services	2	1
Management	9	5
Administrative and waste management	36	3
Educational	45	41
Healthcare	44	44
Arts, entertainment, recreation	41	31
Accommodation	37	34
Food services	17	20
Other services	15	16
Federal enterprises	23	35
General government	46	46
State local enterprises	43	37
State local general	46	46

```
## Display correlation of centrality ranks
c.corr().round(3)
```

	1947	2020
	pagerank	pagerank
1947 pagerank	1.00	0.62
2020 pagerank	0.62	1.00

```
# Display latest graph and node centrality
year = LAST_YEAR - 1
ioUse = ioUses[(1997, year)]
```

(continues on next page)

(continued from previous page)

```

data = ioUse[(~ioUse['rowcode'].str.startswith(drop) &
              ~ioUse['colcode'].str.startswith(drop))].copy()

## extract cross data; generate and load edges (as tuples) to graph
data = data[(data['colcode'] != data['rowcode'])]
data['weights'] = data['datavalue'] / data['datavalue'].sum()
edges = data.loc[data['weights'] > 0, [tail, head, 'weights']].values.tolist()
G = nx.DiGraph()
G.add_weighted_edges_from(edges, weight='weight')

## update master table 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.bea_industry_[master.index].to_list()
show(master.drop(columns=['self', 'clustering']),\
      ndigits=3, \
      caption=f"Centrality of BEA Input-Output Use Table {year}")

```

	user	maker	in_degree
Centrality of BEA Input-Output Use Table 2020			
111CA	194547	286362	0.4203 \
113FF	5057	70809	0.3913
211	114812	262906	0.4203
212	34128	88125	0.7246
213	27222	11272	0.0580
...	...	...	...
GFE	20912	61973	0.7826
GFGD	276183	0	0.0000
GFGN	164988	0	0.0000
GSLE	186131	33750	0.8841
GSLG	809034	0	0.0000

	out_degree	eigenvector
Centrality of BEA Input-Output Use Table 2020		
111CA	0.6232	0.0500 \
113FF	0.5362	0.0124
211	0.5942	0.0809
212	0.6522	0.0352
213	0.5942	0.0029
...	...	...
GFE	0.6812	0.0155
GFGD	0.6957	0.0000
GFGN	0.7826	0.0000
GSLE	0.6812	0.0123
GSLG	0.8116	0.0000

(continues on next page)

(continued from previous page)

```

    pagerank    hub    authority
Centrality of BEA Input-Output Use Table 2020
111CA          0.0088  0.0103  0.0133 \
113FF          0.0061  0.0003  0.0020
211           0.0229  0.0064  0.0046
212           0.0139  0.0020  0.0050
213           0.0027  0.0019  0.0002
...
GFE           0.0039  0.0015  0.0047
GFGD          0.0001  0.0181  -0.0000
GFGN          0.0001  0.0125  0.0000
GSLE          0.0031  0.0139  0.0023
GSLG          0.0001  0.0509  0.0000

    betweenness  closeness
Centrality of BEA Input-Output Use Table 2020
111CA          0.0008  0.6330 \
113FF          0.0006  0.6216
211           0.0161  0.6330
212           0.0052  0.7841
213           0.0000  0.5149
...
GFE           0.0065  0.8214
GFGD          0.0000  0.0000
GFGN          0.0000  0.0000
GSLE          0.0031  0.8961
GSLG          0.0000  0.0000

    bea
Centrality of BEA Input-Output Use Table 2020
111CA          Farms
113FF          Forestry, fishing
211           Oil, gas
212           Mining
213           Support mining
...
GFE           ...
GFGD          Federal enterprises
GFGN          Defense
GSLE          Nondefense
GSLG          State local enterprises
              State local general

[70 rows x 11 columns]

```

Series(graph\_info(G, fast=True))

weakly_connected	True
weakly_connected_components	1
size_largest_weak_component	70
strongly_connected	False
strongly_connected_components	10
size_largest_strong_component	61
directed	True
weighted	True

(continues on next page)

(continued from previous page)

```
negatively_weighted      False
edges                   3235
nodes                   70
selfloops                0
density                 0.669772
dtype: object
```



## COMMUNITY DETECTION

### UNDER CONSTRUCTION

- Community Detection and Industry Sectoring
- TNIC (Text-based Network Industry Classification), see Hoberg and Phillips (2016)

```
import zipfile
import io
import time
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
import networkx.algorithms.community as nx_comm
from finds.database import SQL
from finds.readers import requests_get, Sectoring
from finds.busday import BusDay
from finds.structured import PSTAT
from finds.misc import Show
from finds.graph import graph_info, community_quality, community_detection
from secret import credentials, paths
# %matplotlib qt
VERBOSE = 0
show = Show(ndigits=4, latex=False)
```

```
sql = SQL(**credentials['sql'], verbose=VERBOSE)
bd = BusDay(sql)
pstat = PSTAT(sql, bd, verbose=VERBOSE)
imgdir = paths['images'] / 'tnic' # None
tnic_scheme = 'tnic3'
```

Last FamaFrench Date 2023-06-30 00:00:00

Retrieve TNIC scheme from Hoberg and Phillips website

```
## https://hobergphillips.tuck.dartmouth.edu/industryclass.htm
root = 'https://hobergphillips.tuck.dartmouth.edu/idata/'
source = root + tnic_scheme + '_data.zip'
if source.startswith('http'):
```

(continues on next page)

(continued from previous page)

```
response = requests_get(source)
source = io.BytesIO(response.content)
with zipfile.ZipFile(source).open(tnic_scheme + "_data.txt") as f:
    tnic_data = pd.read_csv(f, sep='\s+')
tnic_data
```

	year	gvkey1	gvkey2	score
0	1988	1011	3226	0.1508
1	1988	1011	6282	0.0851
2	1988	1011	6734	0.0258
3	1988	1011	7609	0.0097
4	1988	1011	9526	0.0369
...	...	...	...	...
25657913	2021	345556	345556	NaN
25657914	2021	345920	345920	NaN
25657915	2021	345980	345980	NaN
25657916	2021	347007	347007	NaN
25657917	2021	349972	349972	NaN

[25657918 rows x 4 columns]

```
# Loop over representative years for community detection
years = [1989, 1999, 2009, 2019] # [1999, 2019]:
collect = {'info': {}, 'modularity': {}, 'community': {} } # to collect metrics
num = 0
for year in years:

    # extract one year of tnic as data frame
    tnic = tnic_data[tnic_data.year == year].dropna()
    nodes = DataFrame(index=sorted(set(tnic['gvkey1']).union(tnic['gvkey2']))) 

    # with gvkey, lookup permno, sic and naics codes
    for code in ['lpermno', 'sic', 'naics']:
        lookup = pstat.build_lookup('gvkey', code, fillna=0)
        nodes[code] = lookup(nodes.index)
    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,
                                       naics[nodes['naics']])
    Series(np.sum(nodes > 0, axis=0)).rename('Non-missing').to_frame().T

    # apply sectoring schemes, and store in nodes DataFrame
    schemes = {'sic': ([f"codes{c}" for c in [5, 10, 12, 17, 30, 38, 48, 49]] +
                       ['sic2', 'sic3']),
               'naics': ['bea1947', 'bea1963', 'bea1997']}
    codes = {} # intermediate to combine raw sic/naics to sector scheme
    for key, sub in schemes.items():
        for scheme in sub:
            if scheme not in codes:
                fillna = 0 if scheme.startswith('sic') else ''
                codes[scheme] = Sectoring(sql, scheme, fillna=fillna)
                nodes[scheme] = codes[scheme][nodes[key]]
            nodes = nodes[nodes[scheme].ne(codes[scheme].fillna())]
```

(continues on next page)

(continued from previous page)

```

nodes

# create edges
edges = tnic[tnic['gvkey1'].isin(nodes.index) &
             tnic['gvkey2'].isin(nodes.index)]
edges = list(edges[['gvkey1', 'gvkey2', 'score']]\.
             .itertuples(index=False, name=None))

# populate graph
g = nx.Graph()
g.add_weighted_edges_from(edges)

# remove self-loops: not necessary
g.remove_edges_from(nx.selfloop_edges(g))

# graph info
collect['info'][year] = Series(graph_info(g, fast=True)).rename(year)

# Plot degree distribution
# num = num + 1
# fig, ax = plt.subplots(clear=True, num=num, figsize=(10, 6))
# Series(nx.degree_histogram(g)).hist(grid=False, ax=ax, bins=100)
# ax.set_title(f'Degree Distribution of {tnic_scheme.upper()} links {year}')
# plt.tight_layout(pad=3)
# plt.savefig(os.path.join(imgdir, f'degree{year}' + figext))

# evaluate modularity of sectoring schemes
modularity = {}
for scheme in sorted(chain(*schemes.values())):
    communities = nodes.loc[list(g.nodes), scheme]\.
        .reset_index()\.
        .groupby(scheme) ['index']\.
        .apply(list)\.
        .to_list()      # list of list of symbols
    modularity[scheme] = community_quality(g, communities)
df = DataFrame.from_dict(modularity, orient='index').sort_index()
collect['modularity'][year] = df
show(df, caption=f"Modularity of sectoring schemes {year}")

# detect communities and report modularity
communities = community_detection(g)
tic = time.time()
quality = {}
for key, community in communities.items():
    quality[key] = community_quality(g, community)
    print('total elapsed:', round(time.time() - tic, 0), key)
df = DataFrame.from_dict(quality, orient='index').sort_index()
collect['community'][year] = df
show(df, caption=f"Modularity community detection algorithms {year}")

# Plot Fama-French codes49 industry representation as heatmap
for ifig, detection in enumerate(['label', 'greedy', 'louvain']):
    scheme = 'codes49'
    industry = []
    for i, community in enumerate(sorted(community[detection]),
                                  key=len,

```

(continues on next page)

(continued from previous page)

```

        reverse=True) ) :
industry.append(nodes[scheme] [list(community)] \
    .value_counts() \
    .rename(i+1))

df = pd.concat(industry, axis=1) \
    .dropna(axis=0, how='all') \
    .fillna(0) \
    .astype(int) \
    .reindex(codes[scheme] \
        .sectors['name']) \
    .drop_duplicates(keep='first'))

num = num + 1
fig, ax = plt.subplots(num=num, clear=True, figsize=(5, 12))
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('{scheme} industry')
fig.subplots_adjust(left=0.4)
plt.tight_layout(pad=3)
plt.savefig(imgdir / f'{detection}_{year}.jpg')

```

### Modularity of sectoring schemes 1989

	communities	modularity	coverage	performance
bea1947	42	0.1534	0.6464	0.9307
bea1963	59	0.1901	0.4979	0.9515
bea1997	63	0.1884	0.4940	0.9525
codes10	10	0.1568	0.7480	0.8491
codes12	12	0.1614	0.7190	0.8821
codes17	17	0.1556	0.7027	0.8428
codes30	30	0.1538	0.6939	0.9156
codes38	37	0.1484	0.6794	0.9130
codes48	48	0.1987	0.5186	0.9560
codes49	49	0.1986	0.5177	0.9584
codes5	5	0.1595	0.7739	0.7900
sic2	69	0.1951	0.4958	0.9562
sic3	258	0.1985	0.2984	0.9696

3.0 secs: label\_propagation  
5.0 secs: louvain  
38.0 secs: greedy  
total elapsed: 0.0 label  
total elapsed: 1.0 louvain  
total elapsed: 1.0 greedy

(continues on next page)

(continued from previous page)

Modularity community detection algorithms 1989

	communities	modularity	coverage	performance
greedy	33	0.2951	0.9437	0.6972
label	152	0.2358	0.9845	0.7812
louvain	31	0.3452	0.6924	0.8630

Modularity of sectoring schemes 1999

	communities	modularity	coverage	performance
bea1947	42	0.2775	0.7493	0.9330
bea1963	60	0.2938	0.6254	0.9582
bea1997	64	0.2924	0.6228	0.9591
codes10	10	0.2903	0.8650	0.8477
codes12	12	0.2988	0.8543	0.8780
codes17	17	0.2792	0.8435	0.8133
codes30	30	0.2856	0.8063	0.9124
codes38	37	0.2780	0.7919	0.9018
codes48	48	0.3120	0.6701	0.9523
codes49	49	0.3130	0.6727	0.9587
codes5	5	0.2919	0.8867	0.7964
sic2	71	0.3128	0.6503	0.9539
sic3	263	0.2671	0.4278	0.9660

2.0 secs: label\_propagation

9.0 secs: louvain

101.0 secs: greedy

total elapsed: 1.0 label

total elapsed: 1.0 louvain

total elapsed: 2.0 greedy

Modularity community detection algorithms 1999

	communities	modularity	coverage	performance
greedy	41	0.3170	0.9802	0.7217
label	146	0.3421	0.9733	0.8481
louvain	27	0.3601	0.9631	0.8268

Modularity of sectoring schemes 2009

	communities	modularity	coverage	performance
bea1947	41	0.3185	0.8310	0.9339
bea1963	58	0.3441	0.7318	0.9593
bea1997	62	0.3427	0.7291	0.9599
codes10	10	0.3256	0.9137	0.8484
codes12	12	0.3285	0.9068	0.8795
codes17	17	0.2829	0.8442	0.8322
codes30	30	0.3243	0.8915	0.9062
codes38	36	0.3136	0.8524	0.9012
codes48	48	0.3533	0.7505	0.9586
codes49	49	0.3531	0.7497	0.9622
codes5	5	0.3279	0.9335	0.8006
sic2	69	0.3500	0.7271	0.9585
sic3	241	0.2960	0.5228	0.9689

1.0 secs: label\_propagation

3.0 secs: louvain

(continues on next page)

(continued from previous page)

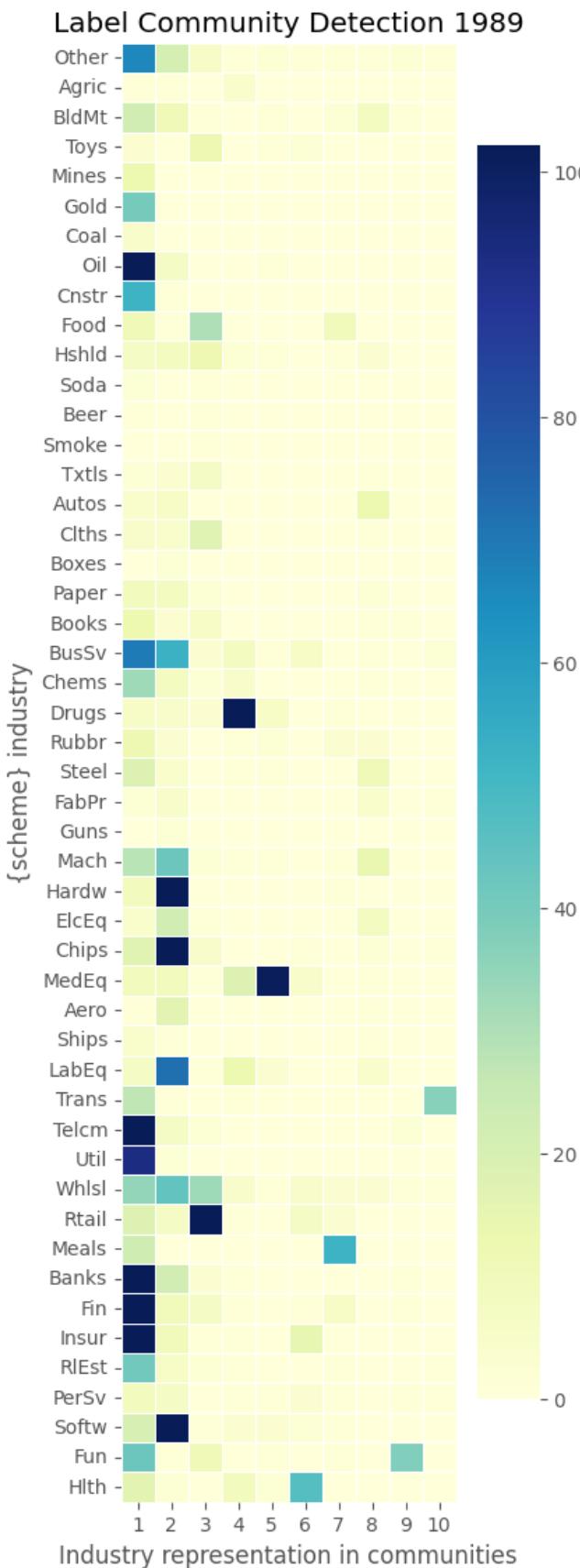
```

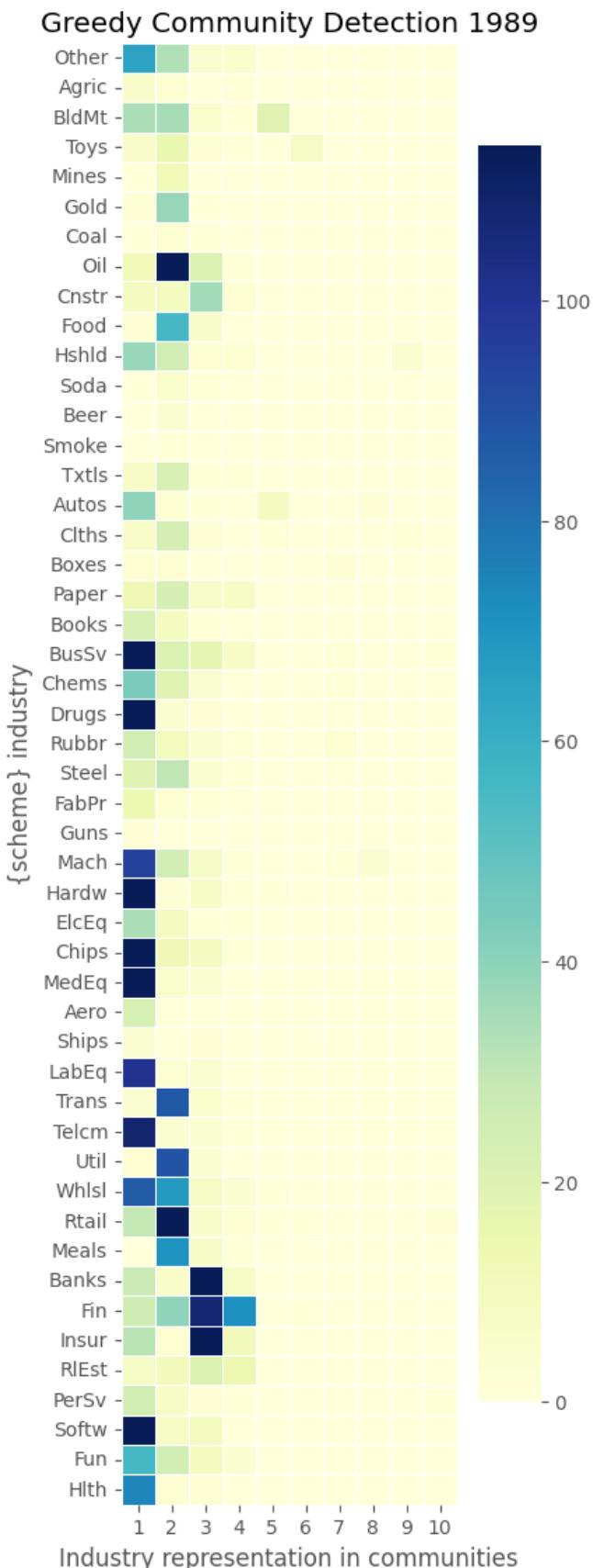
27.0 secs: greedy
total elapsed: 0.0 label
total elapsed: 1.0 louvain
total elapsed: 1.0 greedy
Modularity community detection algorithms 2009
-----
      communities  modularity  coverage  performance
greedy          36      0.2981      0.9810      0.6656
label          124      0.3651      0.9662      0.9226
louvain         27      0.3973      0.9305      0.8521

Modularity of sectoring schemes 2019
-----
      communities  modularity  coverage  performance
bea1947         41      0.4666      0.7492      0.9275
bea1963         59      0.4643      0.6834      0.9525
bea1997         63      0.4639      0.6825      0.9530
codes10          10      0.4713      0.9250      0.8419
codes12          12      0.4859      0.9082      0.8795
codes17          17      0.3787      0.7500      0.8006
codes30          30      0.4828      0.9002      0.9003
codes38          36      0.4692      0.7684      0.8923
codes48          48      0.4749      0.7086      0.9486
codes49          49      0.4749      0.7081      0.9528
codes5           5      0.4743      0.9370      0.8067
sic2            69      0.4688      0.6941      0.9481
sic3           233      0.4152      0.6211      0.9615

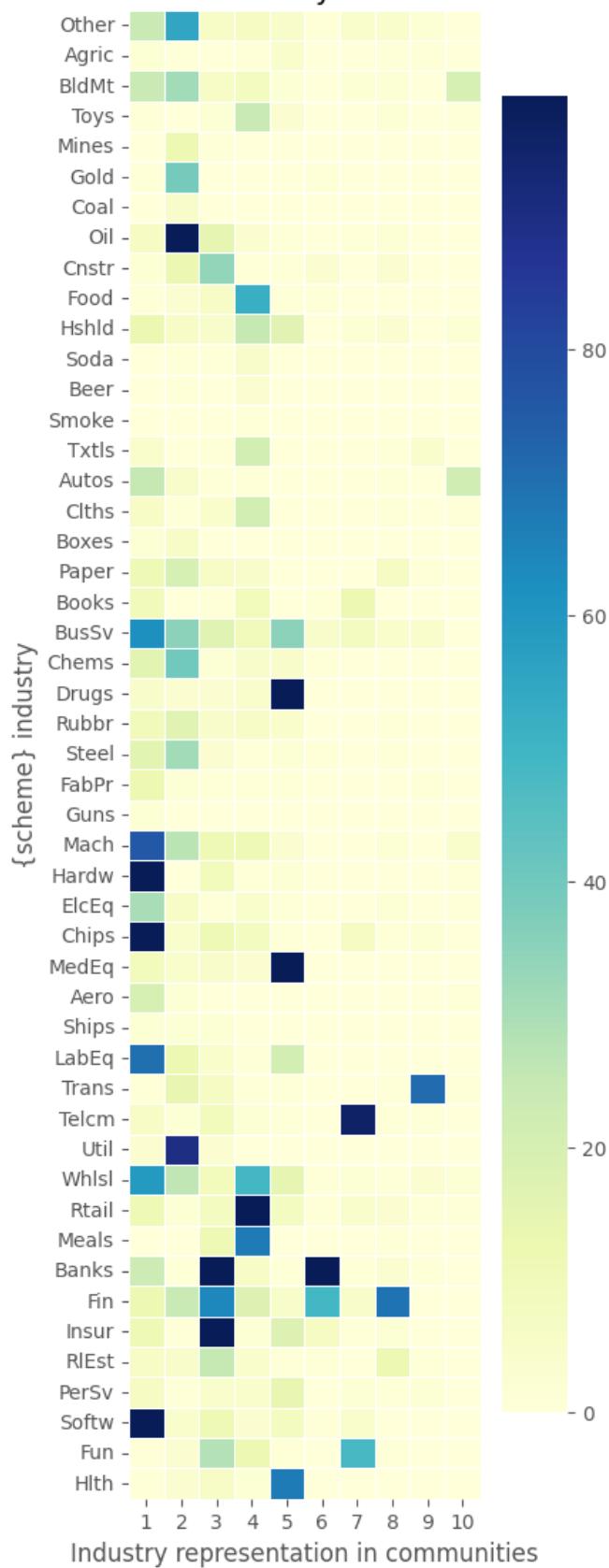
2.0 secs: label_propagation
4.0 secs: louvain
28.0 secs: greedy
total elapsed: 0.0 label
total elapsed: 1.0 louvain
total elapsed: 1.0 greedy
Modularity community detection algorithms 2019
-----
      communities  modularity  coverage  performance
greedy          26      0.4936      0.9775      0.7110
label          91      0.4951      0.9830      0.8971
louvain         21      0.4978      0.9848      0.8398

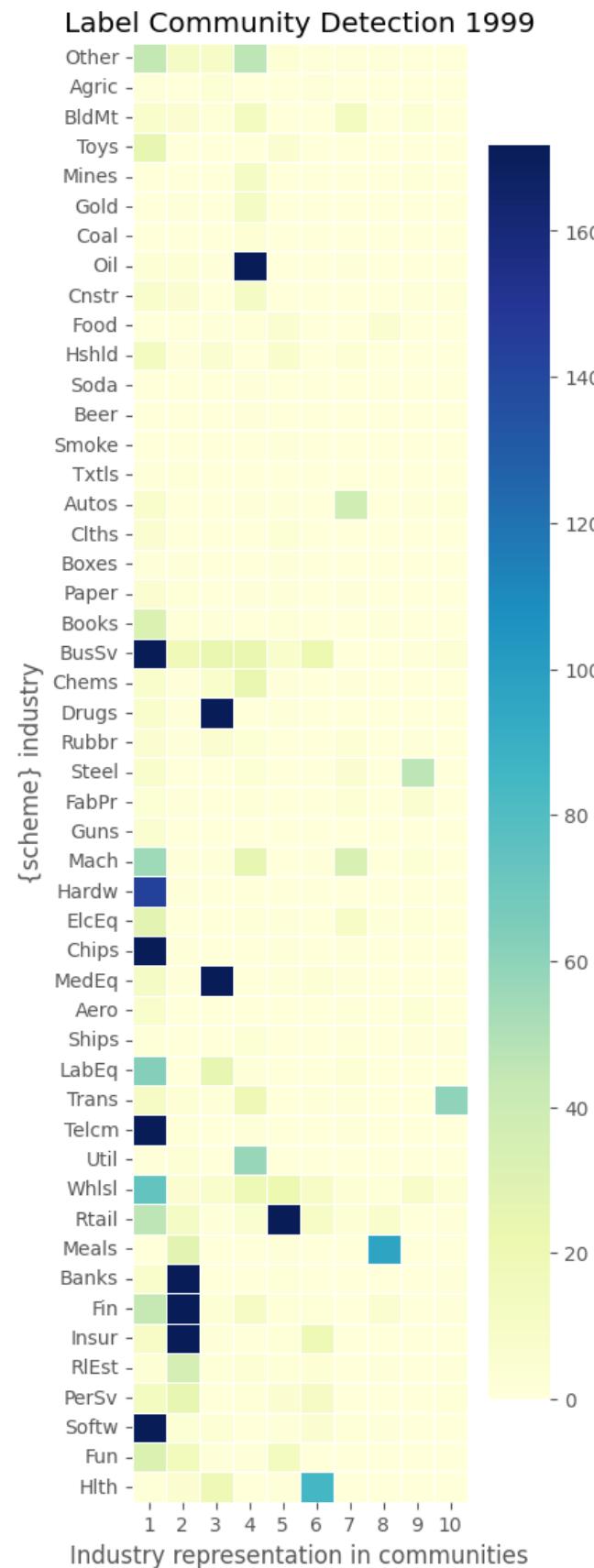
```



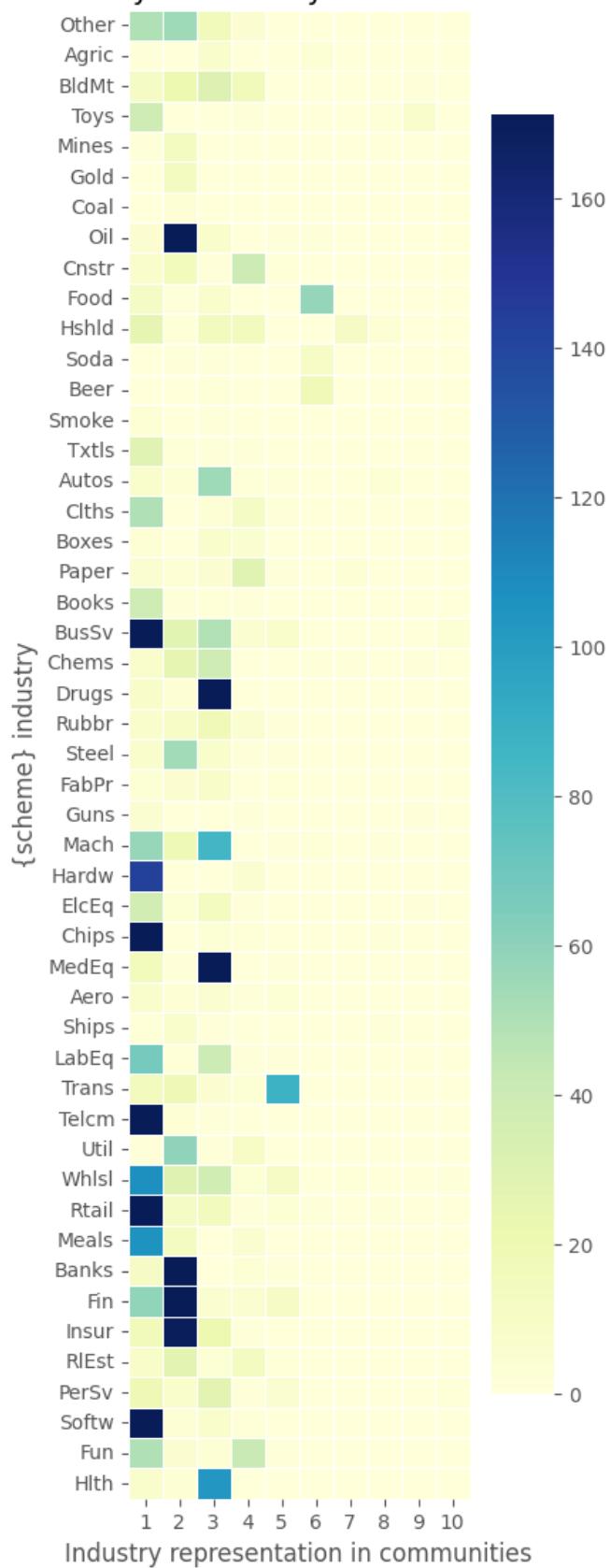


## Louvain Community Detection 1989

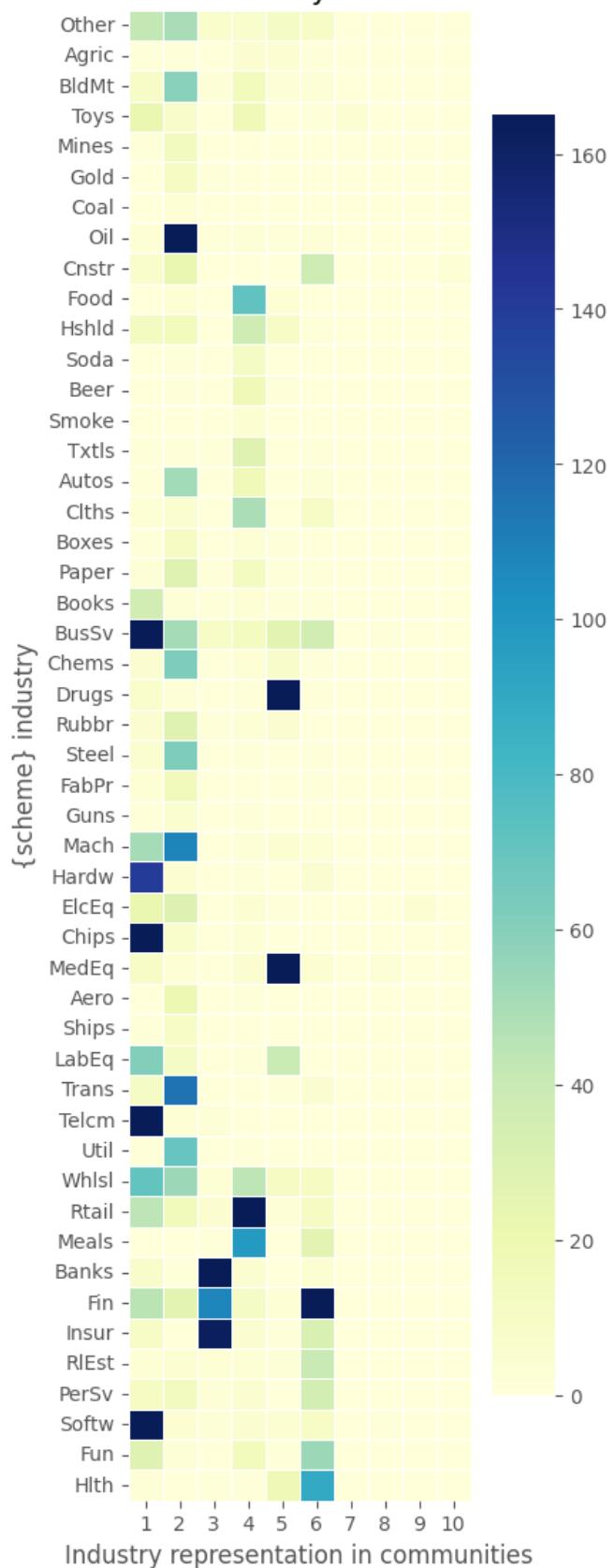




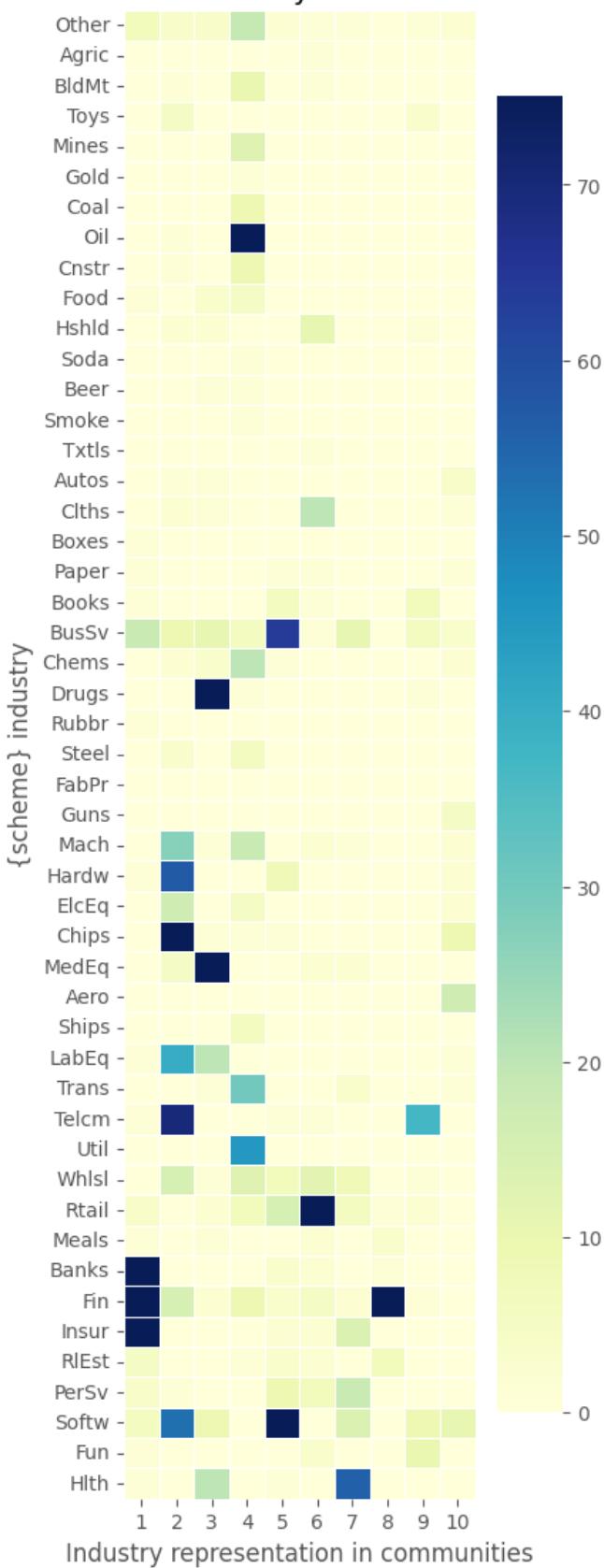
## Greedy Community Detection 1999

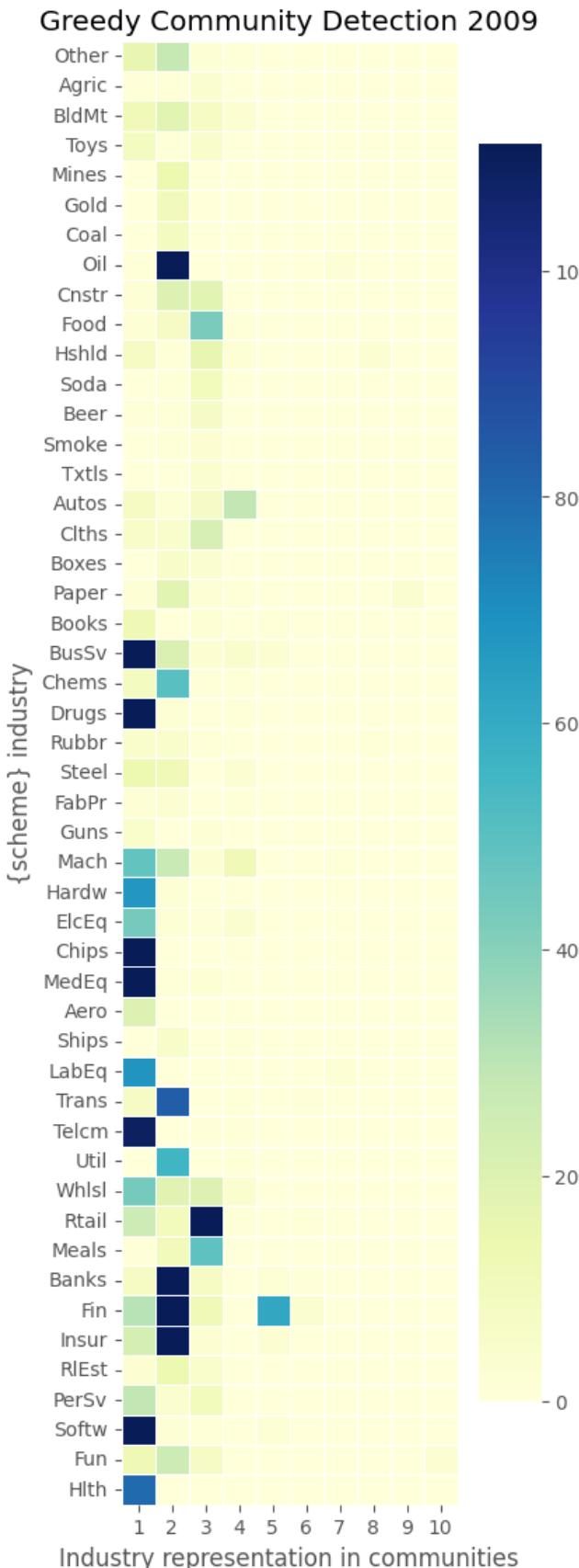


## Louvain Community Detection 1999

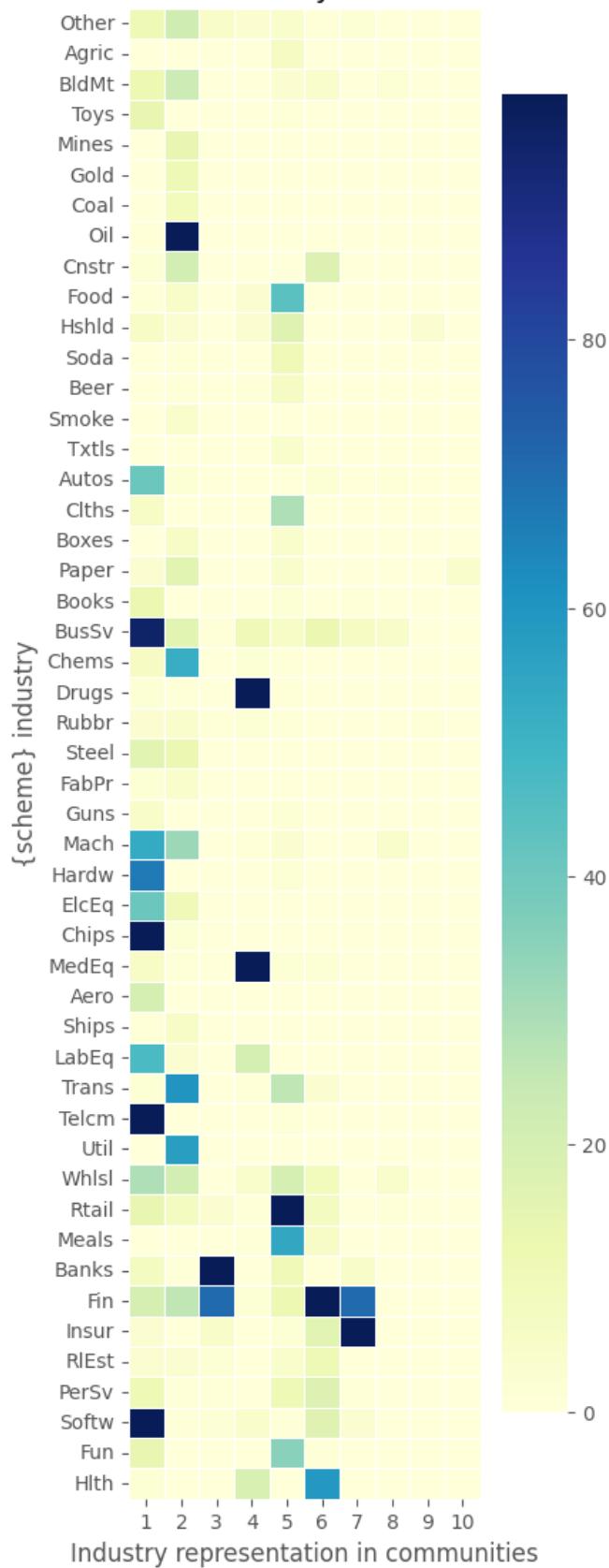


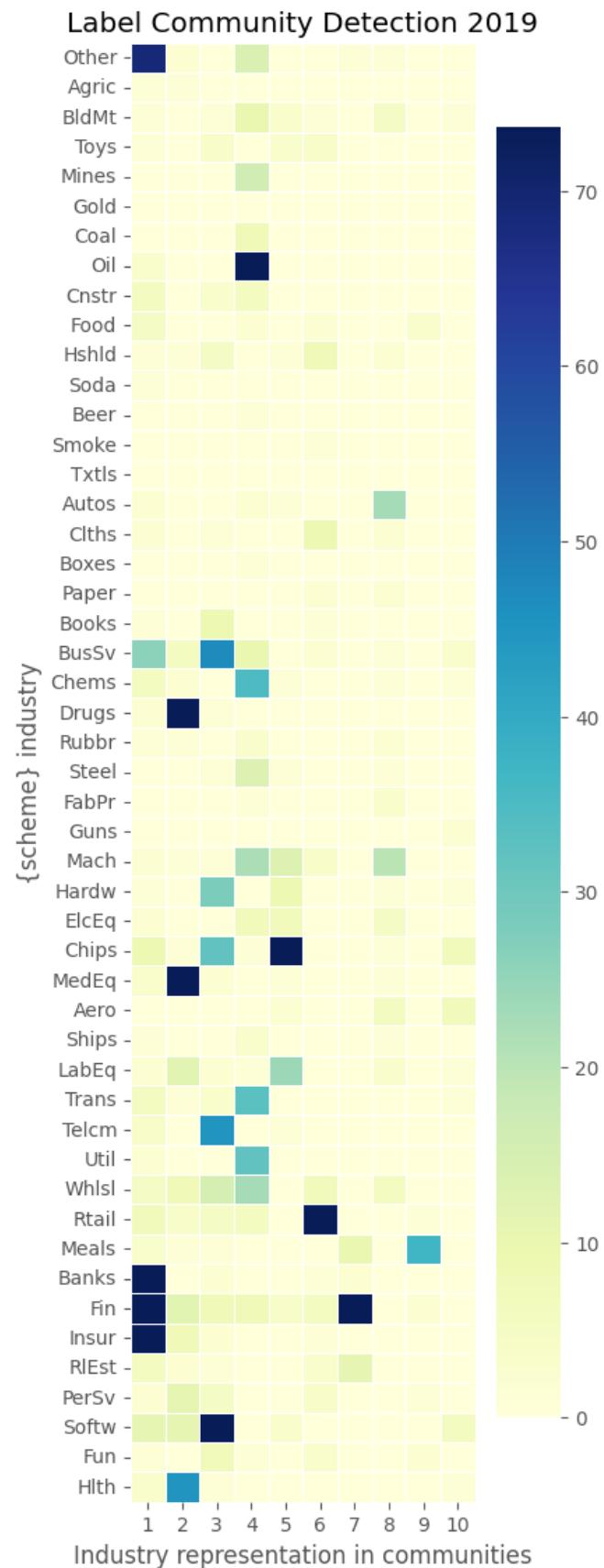
## Label Community Detection 2009

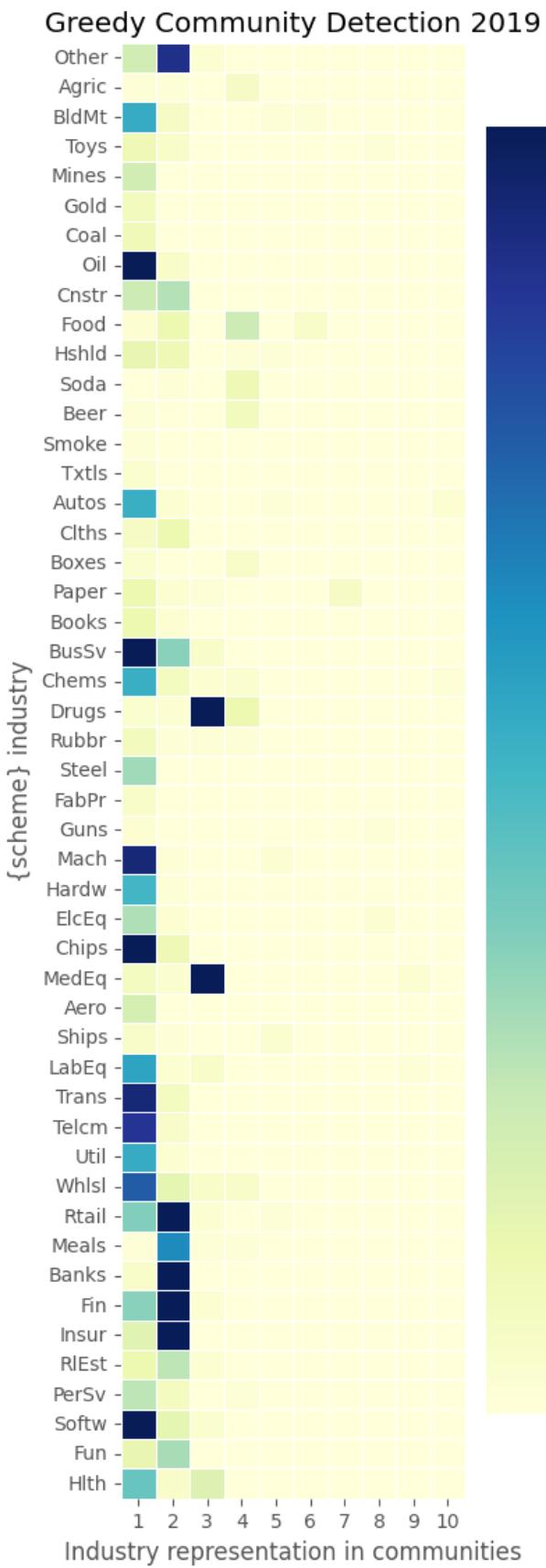




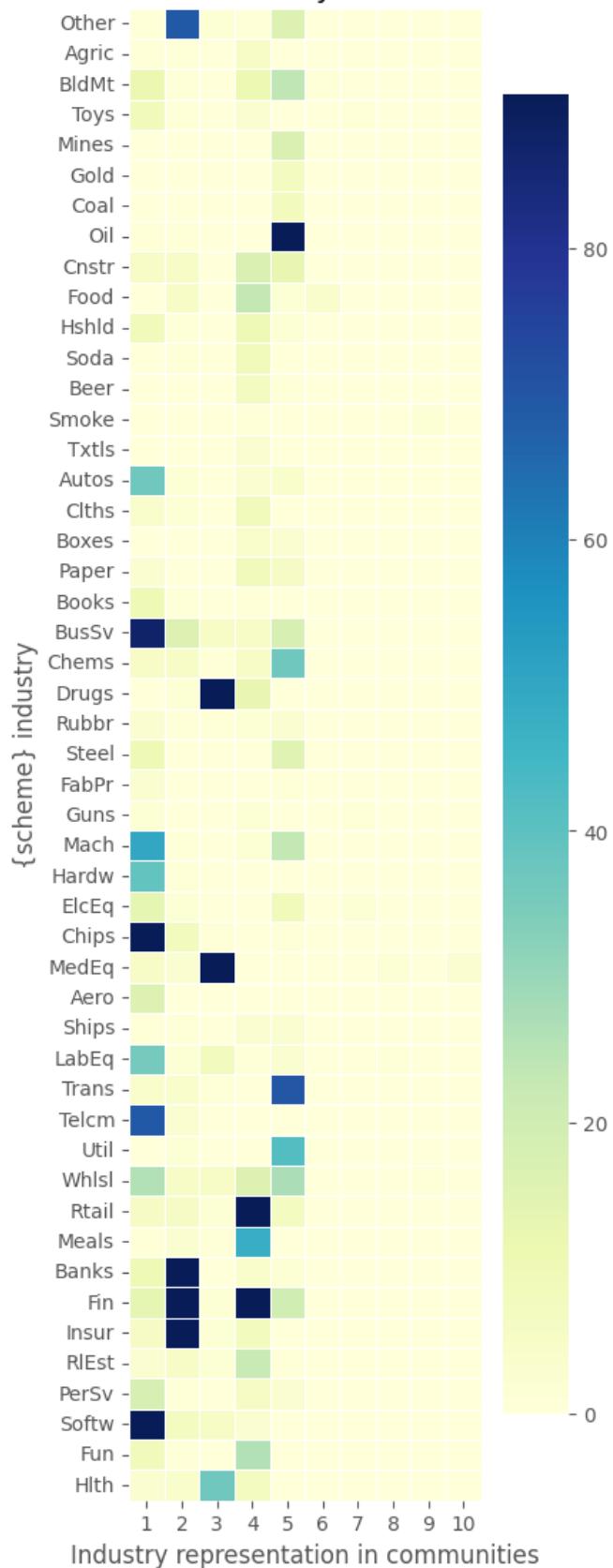
## Louvain Community Detection 2009







## Louvain Community Detection 2019



Display latest year info

```
show(collect['info'][2019],
    caption=f"tnic_scheme {year} graph info:")
```

```
tnic3 2019 graph info:
-----
2019
connected           False
connected_components 14
size_largest_component 3635
directed            False
weighted             True
negatively_weighted False
edges                318913
nodes                3669
selfloops             0
density              0.047394
```

```
show(collect['community'][2019],
    caption=f"tnic_scheme {year} community detection:")
```

```
tnic3 2019 community detection:
-----
communities  modularity  coverage  performance
greedy        26          0.4936    0.9775    0.7110
label         91          0.4951    0.9830    0.8971
louvain       21          0.4978    0.9848    0.8398
```

```
show(collect['modularity'][2019],
    caption=f"{year} modularity of industry sectoring crosswalk schemes:")
```

```
2019 modularity of industry sectoring crosswalk schemes:
-----
communities  modularity  coverage  performance
bea1947       41          0.4666    0.7492    0.9275
bea1963       59          0.4643    0.6834    0.9525
bea1997       63          0.4639    0.6825    0.9530
codes10        10          0.4713    0.9250    0.8419
codes12        12          0.4859    0.9082    0.8795
codes17        17          0.3787    0.7500    0.8006
codes30        30          0.4828    0.9002    0.9003
codes38        36          0.4692    0.7684    0.8923
codes48        48          0.4749    0.7086    0.9486
codes49        49          0.4749    0.7081    0.9528
codes5         5           0.4743    0.9370    0.8067
sic2          69          0.4688    0.6941    0.9481
sic3          233         0.4152    0.6211    0.9615
```



---

CHAPTER  
**THIRTYTWO**

---

## LINK PREDICTION

### UNDER CONSTRUCTION

- Link prediction: resource\_allocation, jaccard coefficient, adamic\_adar, preferential\_attachment
- Accuracy: precision, recall, ROC curve, AUC, confusion matrix,
- Imbalanced sample

```
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.sql import SQL
from finds.database.redisdb import RedisDB
from finds.busday import BusDay
from finds.structured.crsp import CRSP
from finds.structured.pstat import PSTAT
from finds.readers import requests_get
from finds.graph import graph_info, link_prediction
from finds.misc.show import Show
from secret import credentials, paths
# %matplotlib qt
VERBOSE = 0
show = Show(ndigits=4, latex=None)
```

```
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)
imgdir = paths['images'] / 'tnic'
```

Retrieve TNIC schemes from Hoberg and Phillips website

```
# 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'
```

(continues on next page)

(continued from previous page)

```

if source.startswith('http'):
    response = requests_get(source)
    source = io.BytesIO(response.content)
    with zipfile.ZipFile(source).open(scheme + "_data.txt") as f:
        tnic_data[scheme] = pd.read_csv(f, sep='\s+')
for k,v in tnic_data.items():
    print(k, v.shape)

```

```

tnic2 (50402140, 4)
tnic3 (25657918, 4)

```

```

# extract one year of both tnic schemes, merge in permno, and require in univ
year = 2019
capsize = 10 # large cap (large than NYSE median)
univ = crsp.get_universe(bd.endyr(year))
univ = univ[univ['decile'] <= capsize]
lookup = pstat.build_lookup('gvkey', 'lpermno', fillna=0)
nodes = {}
tnic = {}
edges = {}
for scheme in ['tnic2', 'tnic3']:
    tnic[scheme] = tnic_data[scheme][tnic_data[scheme].year == year].dropna()
    gvkeys = sorted(set(tnic[scheme]['gvkey1']).union(tnic[scheme]['gvkey2']))
    df = DataFrame(index=gvkeys, data=lookup(gvkeys), columns=['permno'])
    nodes[scheme] = df[df['permno'].gt(0) & df['permno'].isin(univ.index)].drop_duplicates()
    nodes['tnic2'] = nodes['tnic2'][nodes['tnic2'].index.isin(nodes['tnic3'].index)]
    nodes['tnic3'] = nodes['tnic3'][nodes['tnic3'].index.isin(nodes['tnic2'].index)]

```

```

# create graphs of tnic2 (full graph) and tnic3 (subgraph) 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 = {'info':{}}
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['info'].update({scheme: Series(graph_info(g, fast=True))})

    # Plot degree distribution

```

(continues on next page)

(continued from previous page)

```

fig, ax = plt.subplots(clear=True, figsize=(5, 4))
degree = nx.degree_histogram(g)
degree = DataFrame(data={'degree': degree[1:]},   # exclude degree 0
                   index=np.arange(1, len(degree)))
degree['bin'] = (degree.index // (2*capsize) + 1) * (2*capsize)
degree.groupby('bin').sum().plot(kind='bar', ax=ax, fontsize=6)
ax.set_title(f'Degree Distribution of {scheme.upper()} links {year}')
plt.tight_layout(pad=3)
plt.savefig(imgdir / f'degree_{scheme}_{year}.jpg')

G[scheme] = g

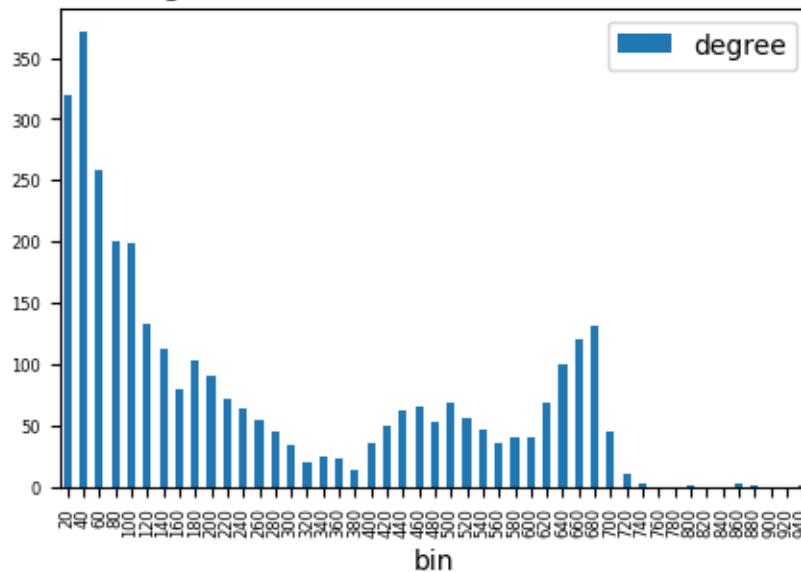
```

```

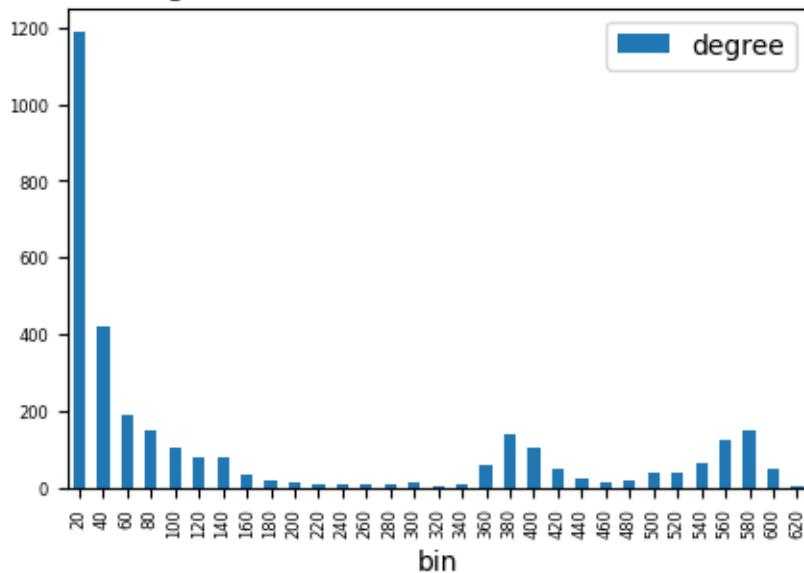
tnic2 nodes = 3253 edges = 808592
tnic3 nodes = 3253 edges = 517840

```

Degree Distribution of TNIC2 links 2019



## Degree Distribution of TNIC3 links 2019



```
show(DataFrame(results['info']),      # Display graph properties
      caption=f"Graph info of TNIC schemes {year}")
```

	tnic2	tnic3
Graph info of TNIC schemes 2019		
connected	False	False
connected_components	2	32
size_largest_component	3251	3195
directed	False	False
weighted	True	True
negatively_weighted	False	False
edges	404296	258920
nodes	3253	3253
selfloops	0	0
density	0.076435	0.048951

## 32.1 Predict links

- jaccard\_coefficient
- resource\_allocation
- adamic\_adar
- preferential\_attachment

```
links = link_prediction(G['tnic3'])
```

```
443.0 secs: resource_allocations
967.0 secs: jaccard_coefficient
```

(continues on next page)

(continued from previous page)

```
1453.0 secs: adamic_adar
1459.0 secs: preferential_attachment
```

```
# Sanity check that tnic3 and prediction edges strictly in tnic2
def isin(e1, e2):
    """helper to count number of edges e1 are in e2"""
    num = sum([e[:2] in e2 for e in e1])
    return num, len(e1), num/len(e1)
```

```
records = [[src, tgt, *isin(G[src].edges, G[tgt].edges)]
           for src, tgt in zip(['tnic3', 'tnic2'],
                               ['tnic2', 'tnic3'])]
records.extend([[src, 'tnic2', *isin(links[src], G['tnic2'].edges)]
               for src in ['jaccard_coefficient',
                           'resource_allocation',
                           'adamic_adar',
                           'preferential_attachment']])

show(DataFrame.from_records(records,
                            columns=['source',
                                      'target',
                                      'source edges in target',
                                      'total source edges',
                                      'fraction']),
     index=False, caption="Counts of edges")
```

	source	target	source edges in target	
Counts of edges				
0	tnic3	tnic2	258920	\
1	tnic2	tnic3	258920	
2	jaccard_coefficient	tnic2	145376	
3	resource_allocation	tnic2	145376	
4	adamic_adar	tnic2	145376	
5	preferential_attachment	tnic2	145376	
	total source edges	fraction		
Counts of edges				
0	258920	1.0000		
1	404296	0.6404		
2	5030458	0.0289		
3	5030458	0.0289		
4	5030458	0.0289		
5	5030458	0.0289		

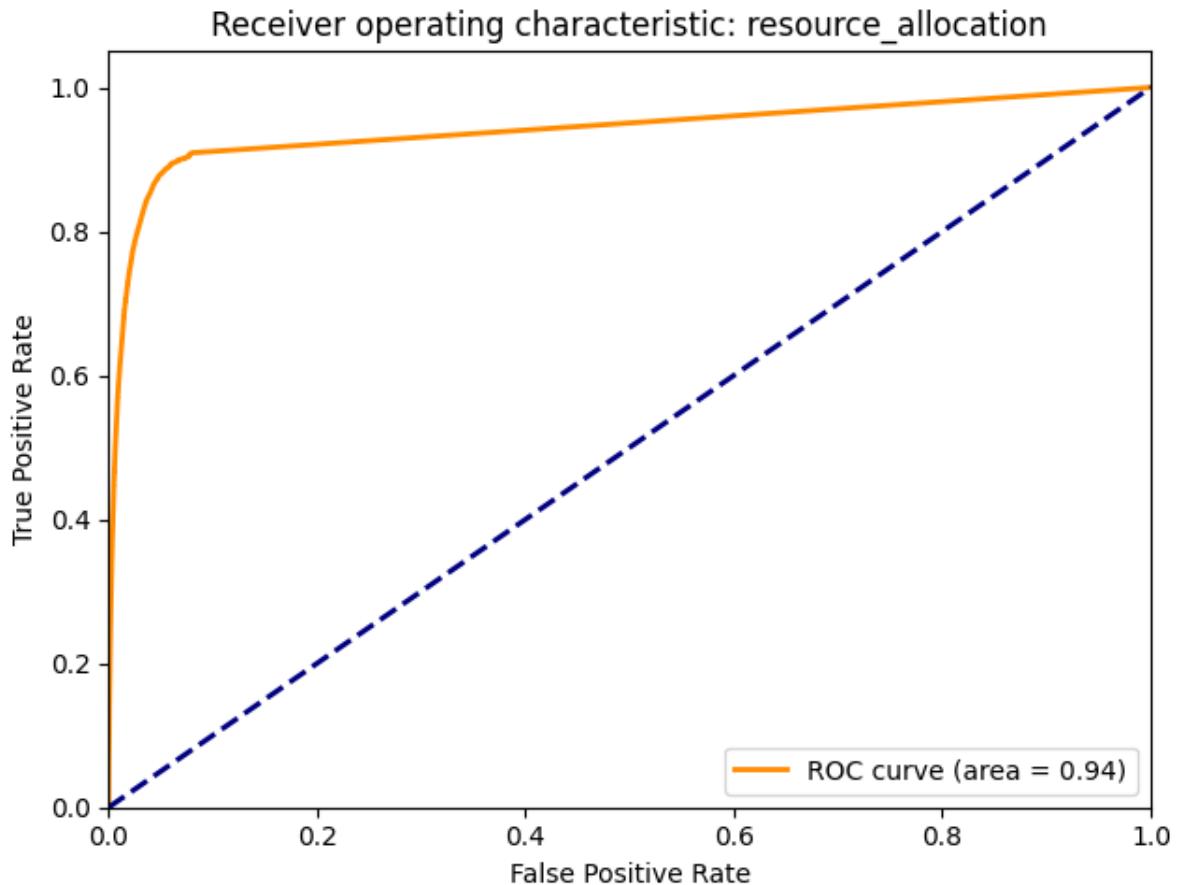
## 32.2 Evaluate accuracy of link prediction algorithms

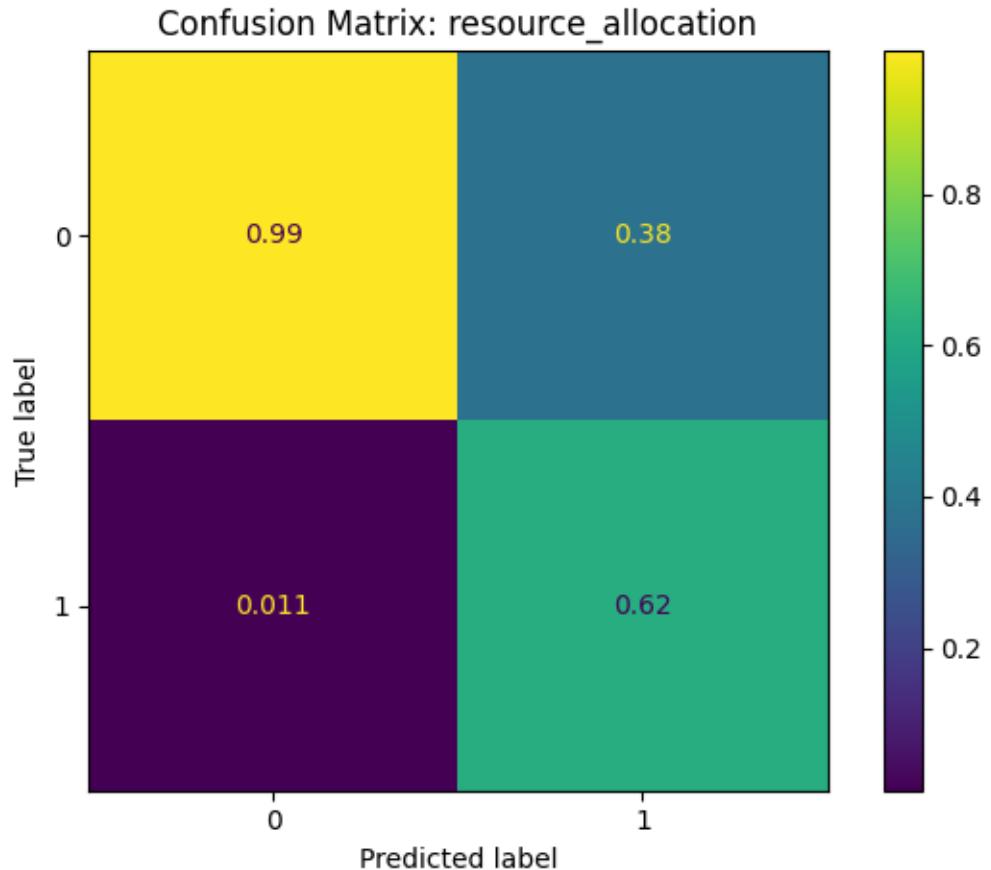
- roc
- auc
- confusion matrix

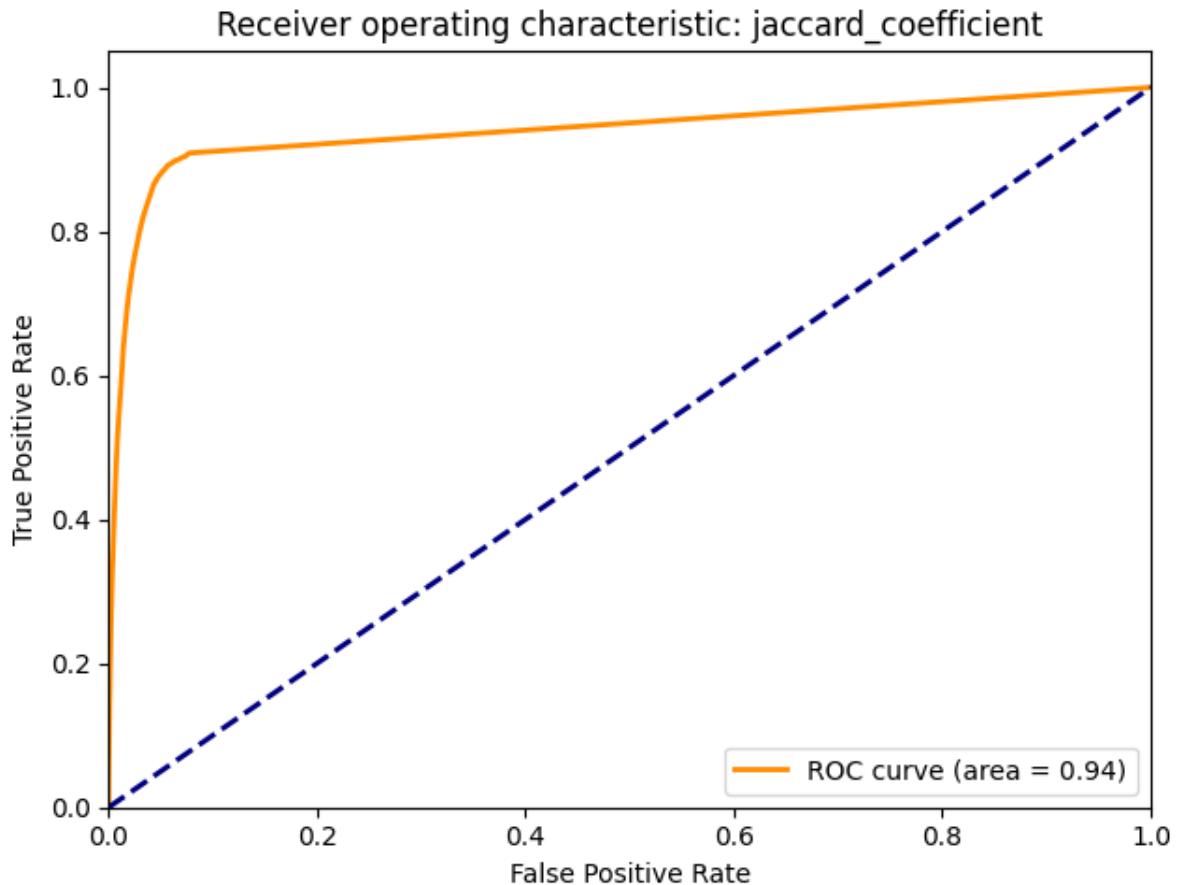
```
def make_sample(prediction, edges):
    """helper to transform prediction to labels and scores for roc and auc"""
    scores = [e[-1] for e in prediction]
    label = [e[:2] for e in prediction]
    gold = [e[:2] in edges for e in prediction]
    return gold, scores, label # actual, predicted score, predicted label
```

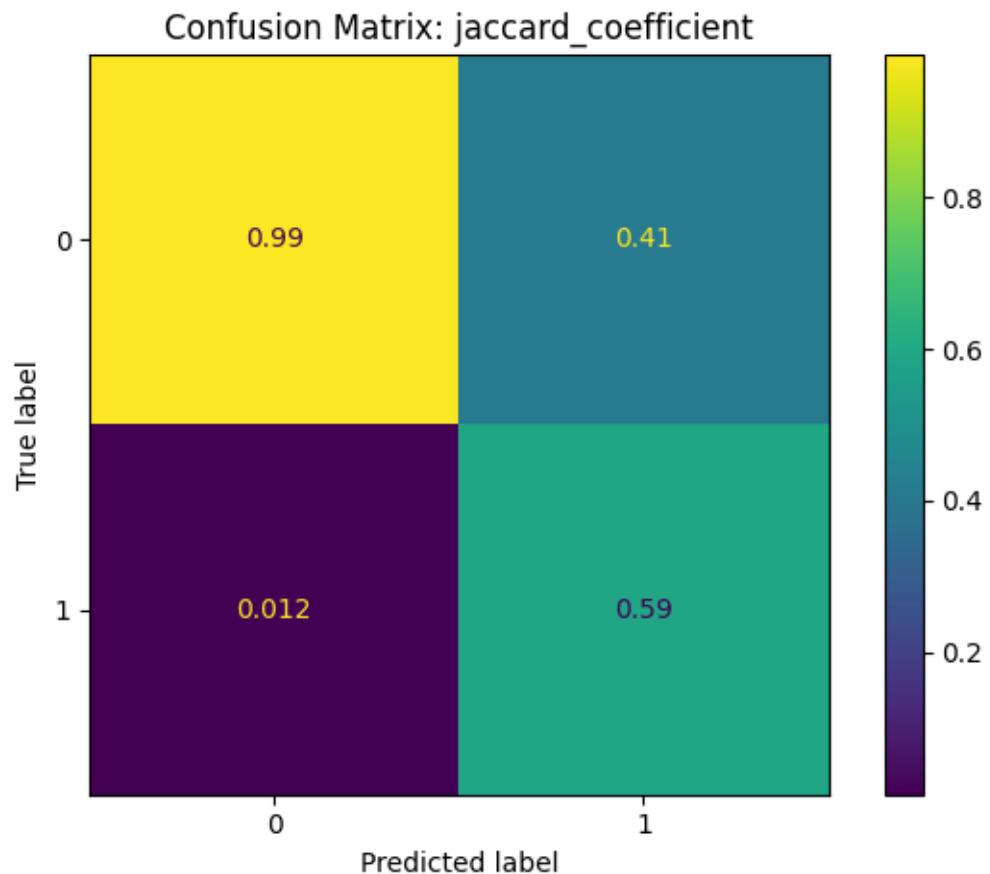
```
for ifig, (method, pred) in enumerate(links.items()):
    y, scores, label = make_sample(pred, G['tnic2'].edges)
    fpr, tpr, thresholds = metrics.roc_curve(y, scores)
    roc_auc = metrics.auc(fpr, tpr)
    plt.figure(clear=True, figsize=(6.5, 5))
    plt.plot(fpr,
              tpr,
              color="darkorange",
              lw=2,
              label="ROC curve (area = %0.2f)" % roc_auc)
    plt.plot([0, 1], [0, 1], color="navy", lw=2, linestyle="--")
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title(f"Receiver operating characteristic: {method}")
    plt.legend(loc="lower right")
    plt.tight_layout()
    plt.savefig(imgdir / f'{method}_auc.jpg')

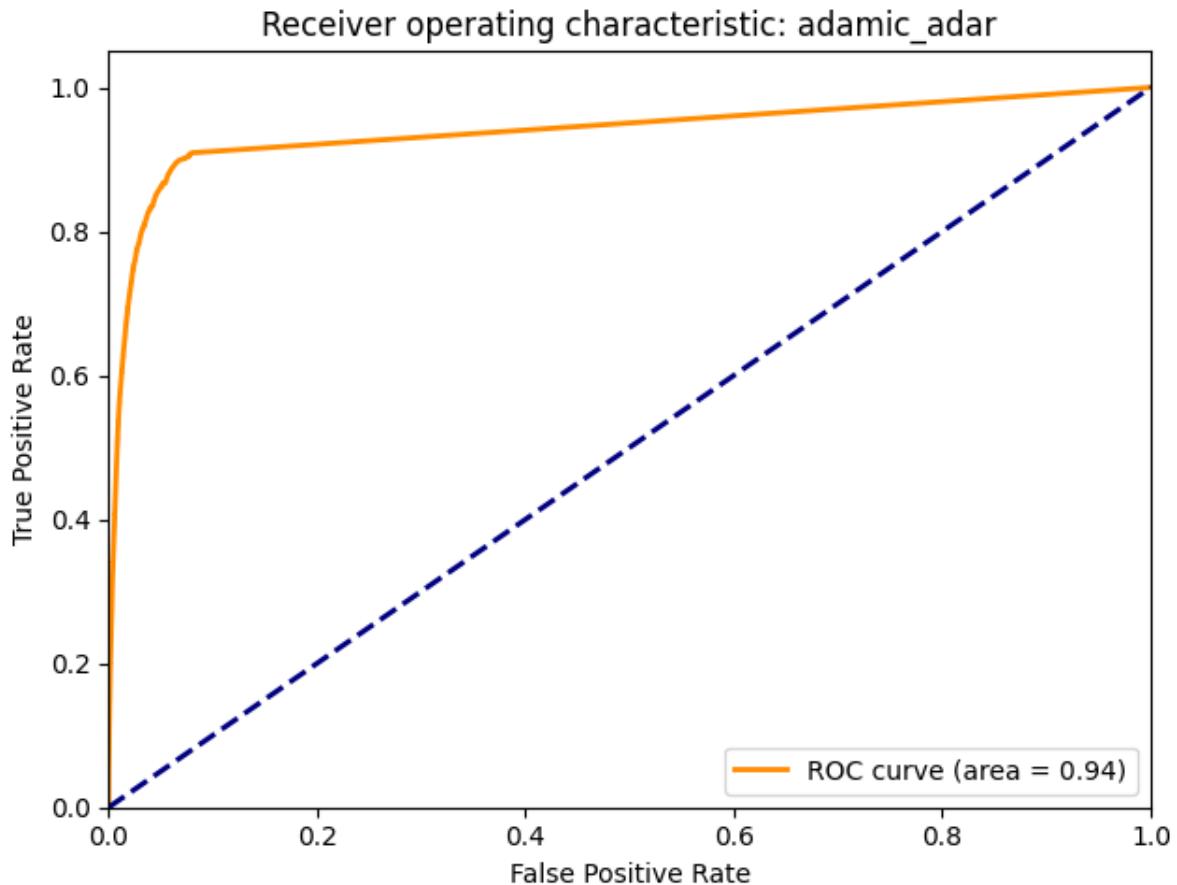
    thresh = scores[sum(y)] # set threshold to equal class size
    cm = metrics.confusion_matrix(y, [score > thresh for score in scores],
                                   normalize='pred')
    disp = metrics.ConfusionMatrixDisplay(confusion_matrix=cm)
    disp.plot()
    plt.title(f"Confusion Matrix: {method}")
    plt.tight_layout()
    plt.savefig(imgdir / f'{method}_confusion.jpg')
```

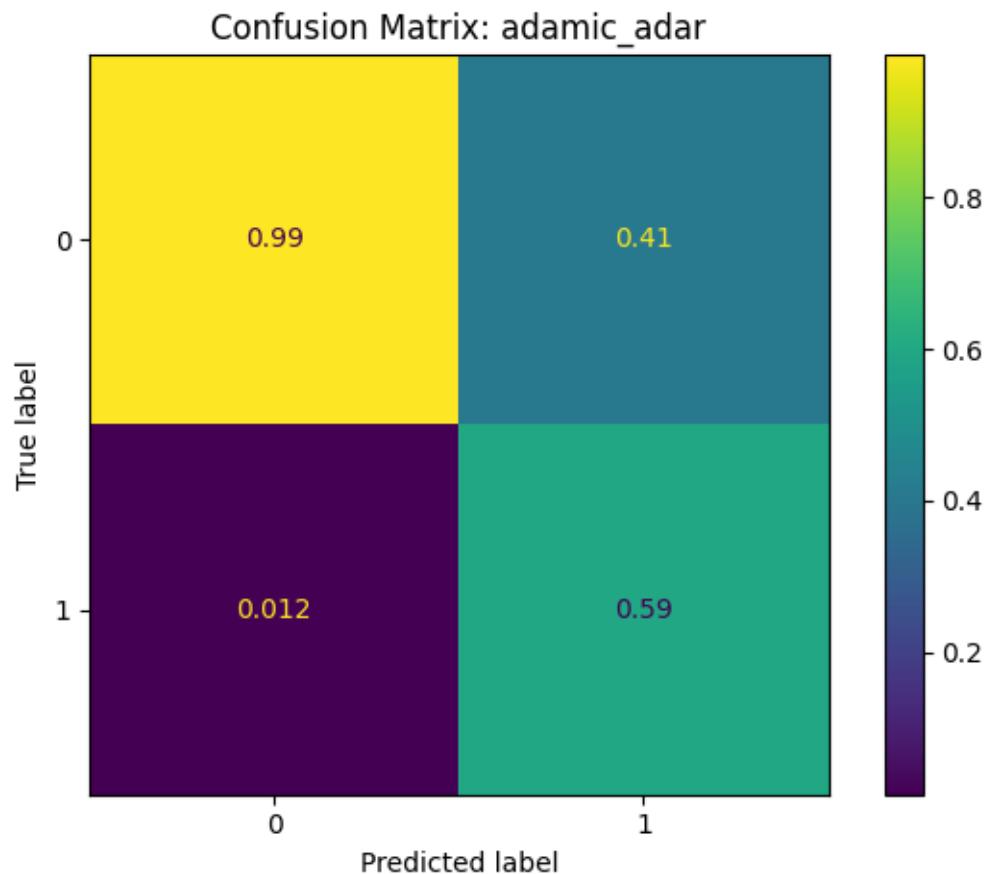


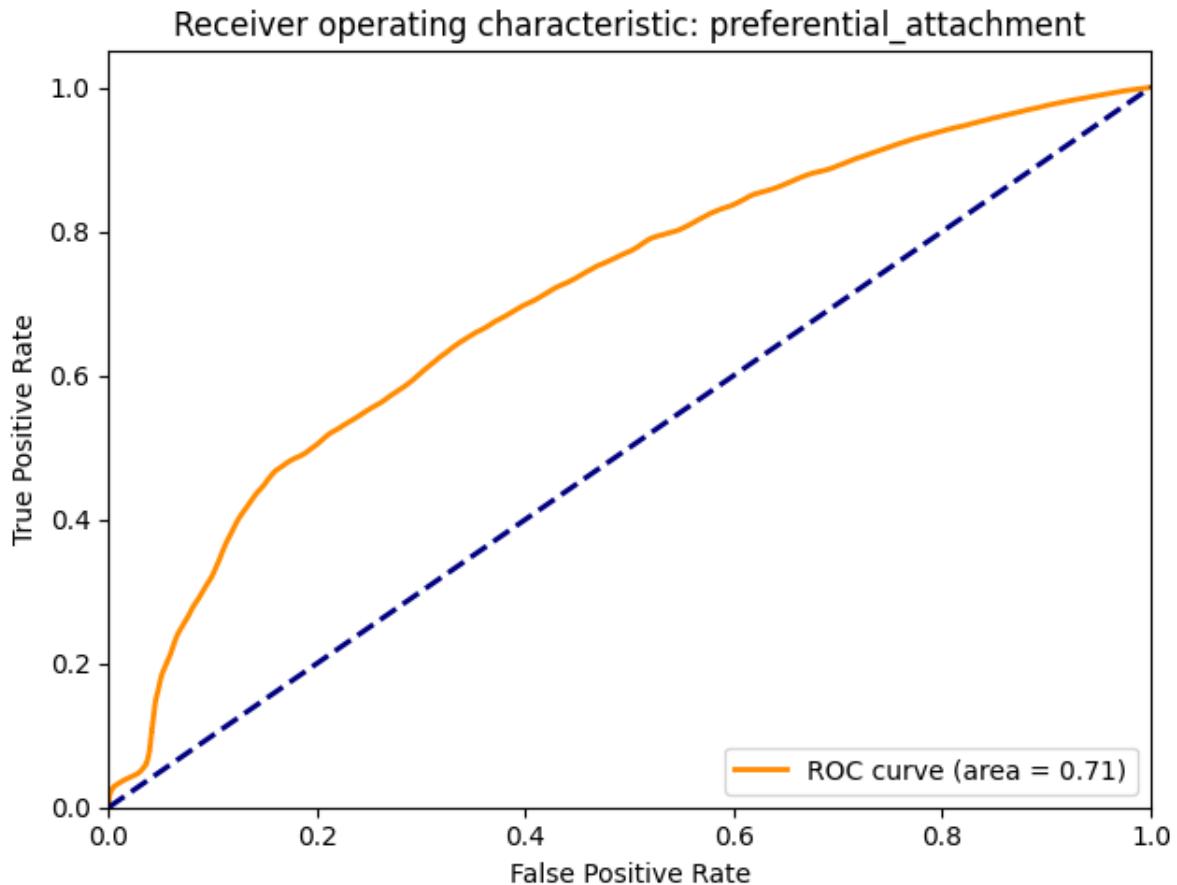


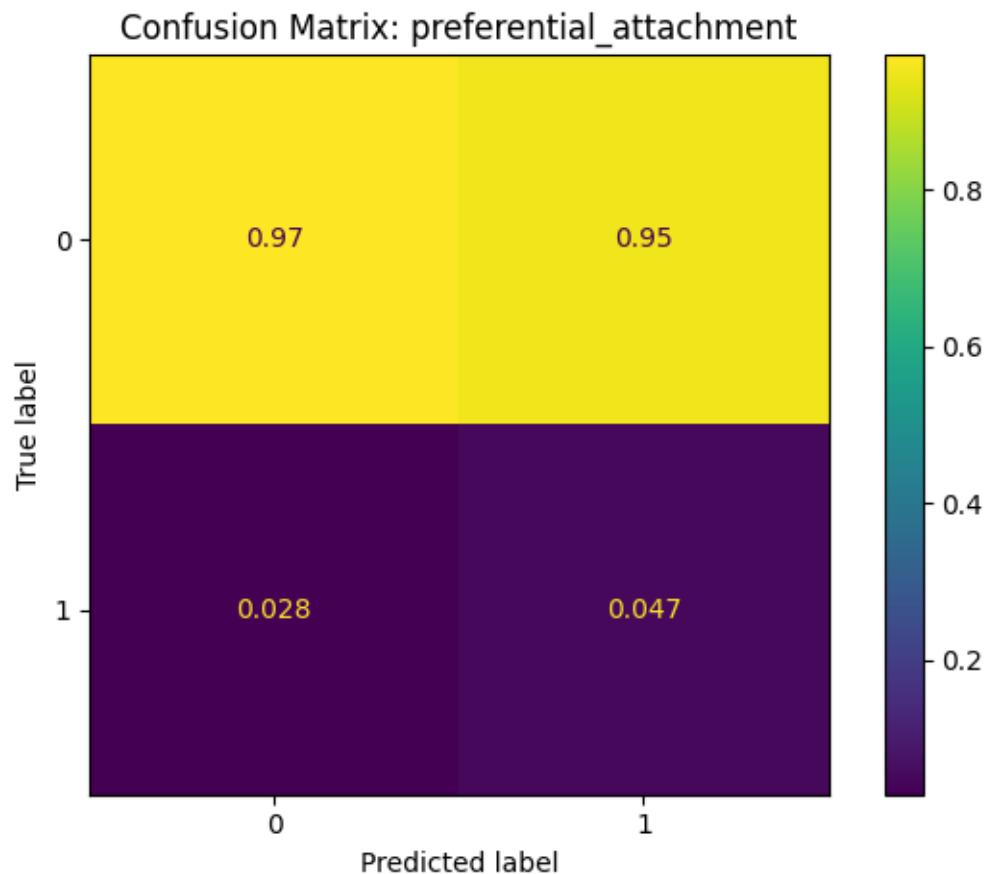












---

CHAPTER  
THIRTYTHREE

---

## SPATIAL REGRESSION

### UNDER CONSTRUCTION

- Quarterly Earnings Surprises
- TNIC links

```
from pandas import DataFrame, Series
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm
from datetime import datetime
from typing import List, Tuple, Any, Dict
from finds.database import SQL, RedisDB
from finds.structured import Stocks, Signals, SignalsFrame
from finds.busday import BusDay
from finds.misc import Show
from secret import credentials, paths, CRSP_DATE
show = Show(ndigits=4, latex=None)
pd.set_option('display.max_rows', None)
VERBOSE = 0
#%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)
signals = Signals(user, verbose=VERBOSE)
outdir = paths['images'] / 'glm'
```



---

CHAPTER  
THIRTYFOUR

---

## CLASSIFICATION MODELS

### UNDER CONSTRUCTION

- Gensim: preprocessing, phrases
- Text classification, S&P Key Developments
- sklearn multi-class: naivebayes, logistic, linearsvc, svm, decisiontree

```
import time
import re
import numpy as np
import pandas as pd
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
from collections import Counter
from wordcloud import WordCloud
import gensim
from gensim.parsing.preprocessing import strip_tags, strip_punctuation, \
    strip_multiple_whitespaces, strip_numeric, remove_stopwords, strip_short, \
    stem_text, preprocess_documents, preprocess_string
from gensim.models.phrases import Phrases, ENGLISH_CONNECTOR_WORDS
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction import text
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
from tqdm import tqdm
from finds.database.mongodb import MongoDB
from finds.unstructured import Unstructured
from finds.structured.pstat import PSTAT
from finds.misc.show import Show
from finds.plots import plot_bar
from secret import paths, credentials
# %matplotlib qt
VERBOSE = 0
show = Show(ndigits=4, latex=None)
```

```
mongodb = MongoDB(**credentials['mongodb'], verbose=VERBOSE)
keydev = Unstructured(mongodb, 'KeyDev')
imgdir = paths['images'] / 'classify'
events_ = PSTAT._event
roles_ = PSTAT._role
```

```
# Show event sample counts
# counts = {e: keydev['events'].count_documents({'keydeveventtypeid': e})
#             for e in keydev['events'].distinct('keydeveventtypeid')}
```

### Retrieve headline+situation text

- in lower case, exclude numeric

```
#[16, 83] #[65, 80]: #[101, 192, 65, 80, 27, 86]
events = [28, 16, 83, 41, 81, 23, 87, 45, 80, 97, 231, 46, 31, 77, 29,
          232, 101, 42, 47, 86, 93, 3, 22, 102, 82]
corpus = {}
for event in events:
    docs = keydev['events'].find({'keydeveventtypeid': {'$eq': event}},
                                  {'_id': 0})
    corpus[event] = [doc['headline'] + ' ' + doc['situation'] for doc in docs]
```

```
STOPWORDS = gensim.parsing.preprocessing.STOPWORDS      # modify stopwords
stopwords = ['inr', 'yen', 'jpy', 'eur', 'dkk', 'cny', 'sfr']
gensim.parsing.preprocessing.STOPWORDS = STOPWORDS.union(stopwords)
```

```
## 1. Preprocess with gensim
CUSTOM_FILTERS = [lambda x: x.lower(),
                  strip_tags,
                  strip_punctuation,
                  strip_multiple_whitespaces,
                  strip_numeric,
                  remove_stopwords,
                  strip_short,
                  stem_text]
for event, docs in tqdm(corpus.items()):
    corpus[event] = [preprocess_string(doc, CUSTOM_FILTERS) for doc in docs]
```

100% |██████████| 25/25 [05:41<00:00, 13.67s/it]

```
## 2. Transform to Bigram corpus with gensim model
phrase_model = Phrases([doc for docs in corpus.values() for doc in docs],
                       min_count=2,
                       threshold=1,
                       connector_words=ENGLISH_CONNECTOR_WORDS)
bigram_corpus = {}
for event, docs in tqdm(corpus.items()):
    bigram_corpus[event] = phrase_model[docs]
```

100% |██████████| 25/25 [00:00<00:00, 19414.48it/s]

```
## 3. Train/test split of bigram_corpus with sklearn
data = " ".join(doc) for docs in bigram_corpus.values() for doc in docs]
labels = []
for event, docs in bigram_corpus.items():
    labels.extend([event] * len(docs))
```

```

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 = events_[sorted(np.unique(y_train))].to_frame()
summary['n_train'] = Series(y_train).value_counts()
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()
show(summary, caption='Stratified Train/Test Split by Event')

```

	event
Stratified Train/Test Split by Event	
3	Seeking Acquisitions/Investments \
16	Executive/Board Changes - Other
22	Strategic Alliances
23	Client Announcements
28	Announcements of Earnings
29	Corporate Guidance - New/Confirmed
31	Business Expansions
41	Product-Related Announcements
42	Debt Financing Related
45	Dividend Affirmations
46	Dividend Increases
47	Dividend Decreases
77	Changes in Company Bylaws/Rules
80	M&A Transaction Announcements
81	M&A Transaction Closings
82	M&A Transaction Cancellations
83	Private Placements
86	Follow-on Equity Offerings
87	Fixed Income Offerings
93	Shelf Registration Filings
97	Special/Extraordinary Shareholders Meeting
101	Executive Changes - CEO
102	Executive Changes - CFO
231	Buyback Tranche Update
232	Buyback Transaction Announcements
	n_train n_test frac_train frac_test
Stratified Train/Test Split by Event	
3	10189 2547 0.0104 0.0104
16	102415 25604 0.1046 0.1046
22	9636 2409 0.0098 0.0098
23	72549 18138 0.0741 0.0741
28	195312 48828 0.1994 0.1994
29	15545 3886 0.0159 0.0159
31	21209 5302 0.0217 0.0217
41	74537 18635 0.0761 0.0761
42	12147 3037 0.0124 0.0124
45	42963 10741 0.0439 0.0439
46	22136 5534 0.0226 0.0226
47	12129 3032 0.0124 0.0124
77	17884 4471 0.0183 0.0183

(continues on next page)

(continued from previous page)

80	34644	8661	0.0354	0.0354
81	72764	18191	0.0743	0.0743
82	8061	2015	0.0082	0.0082
83	90405	22602	0.0923	0.0923
86	11750	2937	0.0120	0.0120
87	55722	13930	0.0569	0.0569
93	10234	2559	0.0104	0.0105
97	26058	6514	0.0266	0.0266
101	13033	3258	0.0133	0.0133
102	9358	2340	0.0096	0.0096
231	24784	6196	0.0253	0.0253
232	13936	3484	0.0142	0.0142

```
## 4. TfIdf vectorize with sklearn: fit on train
max_df, min_df, max_features = 0.5, 200, 10000
tfidf_vectorizer = text.TfidfVectorizer(
    encoding='latin-1',
    strip_accents='unicode',
    lowercase=True,
    stop_words=stop_words,
    max_df=max_df,
    min_df=min_df,
    max_features=max_features,
    token_pattern=r'\b[a-z]+\b',
#    token_pattern=r"\b[^d\W][^d\W][^d\W]+\b", #r'\b[^d\W]+\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()
show(DataFrame([[x_train.shape, x_test.shape]],
               index=['data shape:'],
               columns=['train', 'test']),
               caption="n_sample x n_features")
print(len(feature_names), 'train:', x_train.shape, 'test:', x_test.shape)
```

10000 train: (979400, 10000) test: (244851, 10000)

## 34.1 Train Classification Models

- SVM: kernel
- regularization?

```
models = {
    'naivebayes': MultinomialNB(),
    'linearsvc': LinearSVC(class_weight='balanced',
                           multi_class='ovr',
                           penalty='l2',
                           verbose=VERBOSE),
    'logistic': LogisticRegression(class_weight='balanced',
                                   verbose=VERBOSE,
                                   penalty='l2',
```

(continues on next page)

(continued from previous page)

```

        multi_class='multinomial',
        max_iter=1000),
'decisiontree': DecisionTreeClassifier(class_weight='balanced',
                                         random_state=0),
}
results = {}
for name, clf in tqdm(models.items()):
    tic = time.time()
    clf.fit(x_train, y_train)
    train_score = clf.score(x_train, y_train) # evaluate train set accuracy
    test_score = clf.score(x_test, y_test) # and test set accuracy
    toc = time.time() - tic
    results.update({name: {'train_score': train_score,
                          'test_score': test_score,
                          'elapsed': toc}})
show(DataFrame.from_dict(results, orient='index'),
      caption="Train and Test Accuracy")
res = DataFrame.from_dict(results, orient='index')

```

100% |██████████| 4/4 [34:07&lt;00:00, 511.87s/it]

Show decision tree fitted parameters

```

dt = models['decisiontree']
x = dt.decision_path(x_test[-1])
show(DataFrame.from_dict({'Tree depth': dt.get_depth(),
                          'Tree n_leaves': dt.get_n_leaves()},
                          columns=['property'],
                          orient='index'),
      caption="Fitted Decision Tree")
"""
Tree depth: 431
Tree n_leaves: 59217
Length of a test document: 30
Length of its decision path: 262

Tree depth: 636
Tree n_leaves: 80185
Length of a test document: 20
Length of its decision path: 313
"""

```

```
'\nTree depth: 431\nTree n_leaves: 59217\nLength of a test document: 30\nLength of its decision path: 262\n\nTree depth: 636\nTree n_leaves: 80185\nLength of a test document: 20\nLength of its decision path: 313\n'
```

```

## Plot Train and Test Accuracy
y = res[['train_score', 'test_score']]
fig, ax = plt.subplots(num=1, clear=True, figsize=(10, 6))
plot_bar(y,
          ax=ax,
          labels=[round(value, 3) for row in y.itertuples(index=False, name=None)
                  for value in row],
          fontsize=12,

```

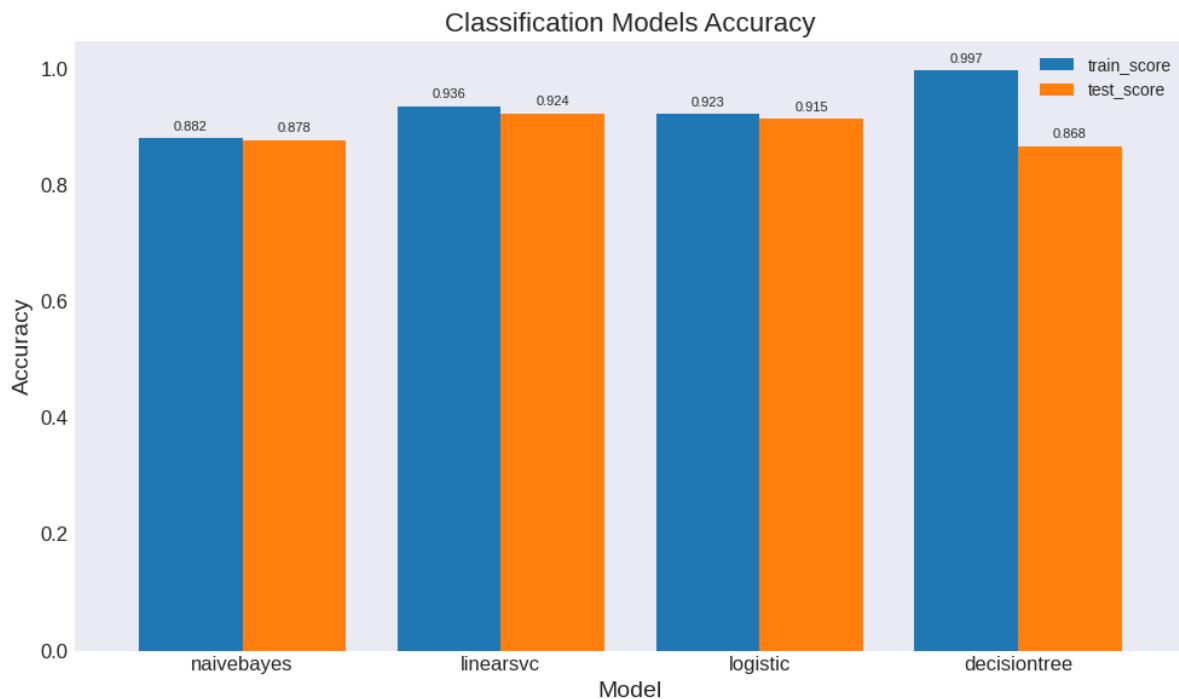
(continues on next page)

(continued from previous page)

```

        title='Classification Models Accuracy',
        loc='lower right',
        ylabel='Accuracy',
        xlabel='Model')
plt.tight_layout()
plt.savefig(imgdir / 'mse.jpg')

```



```

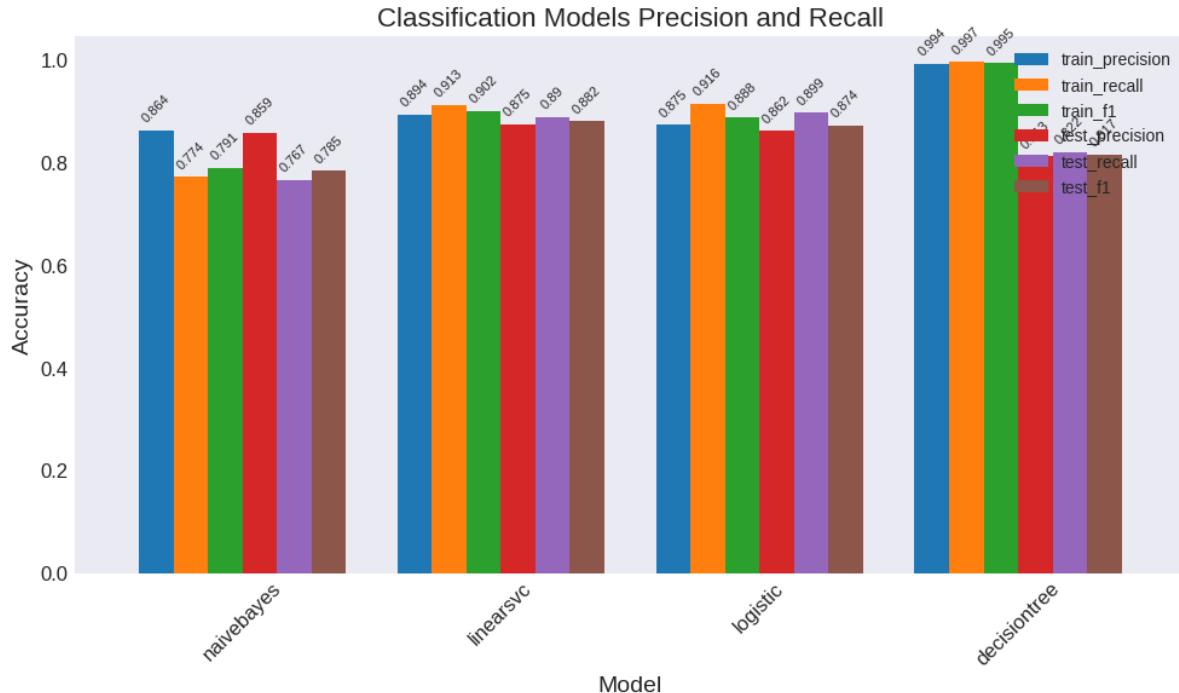
## Plot precision, recall, f1
scores = {}
for ifig, (name, clf) in enumerate(models.items()):
    train = metrics.precision_recall_fscore_support(y_train,
                                                    clf.predict(x_train),
                                                    average='macro')[:3]
    test = metrics.precision_recall_fscore_support(y_test,
                                                    clf.predict(x_test),
                                                    average='macro')[:3]
    scores[name] = np.append(train, test)
y = DataFrame(scores,
              index=[t + '_' + s
                     for t in ['train', 'test']
                     for s in ['precision', 'recall', 'f1']]).T
fig, ax = plt.subplots(num=1, clear=True, figsize=(10, 6))
plot_bar(y,
          ax=ax,
          labels=[round(value, 3) for row in y.itertuples(index=False, name=None)
                  for value in row],
          fontsize=12,
          ylabel='Accuracy',
          title='Classification Models Precision and Recall',
          xlabel='Model',
          rotation=45,

```

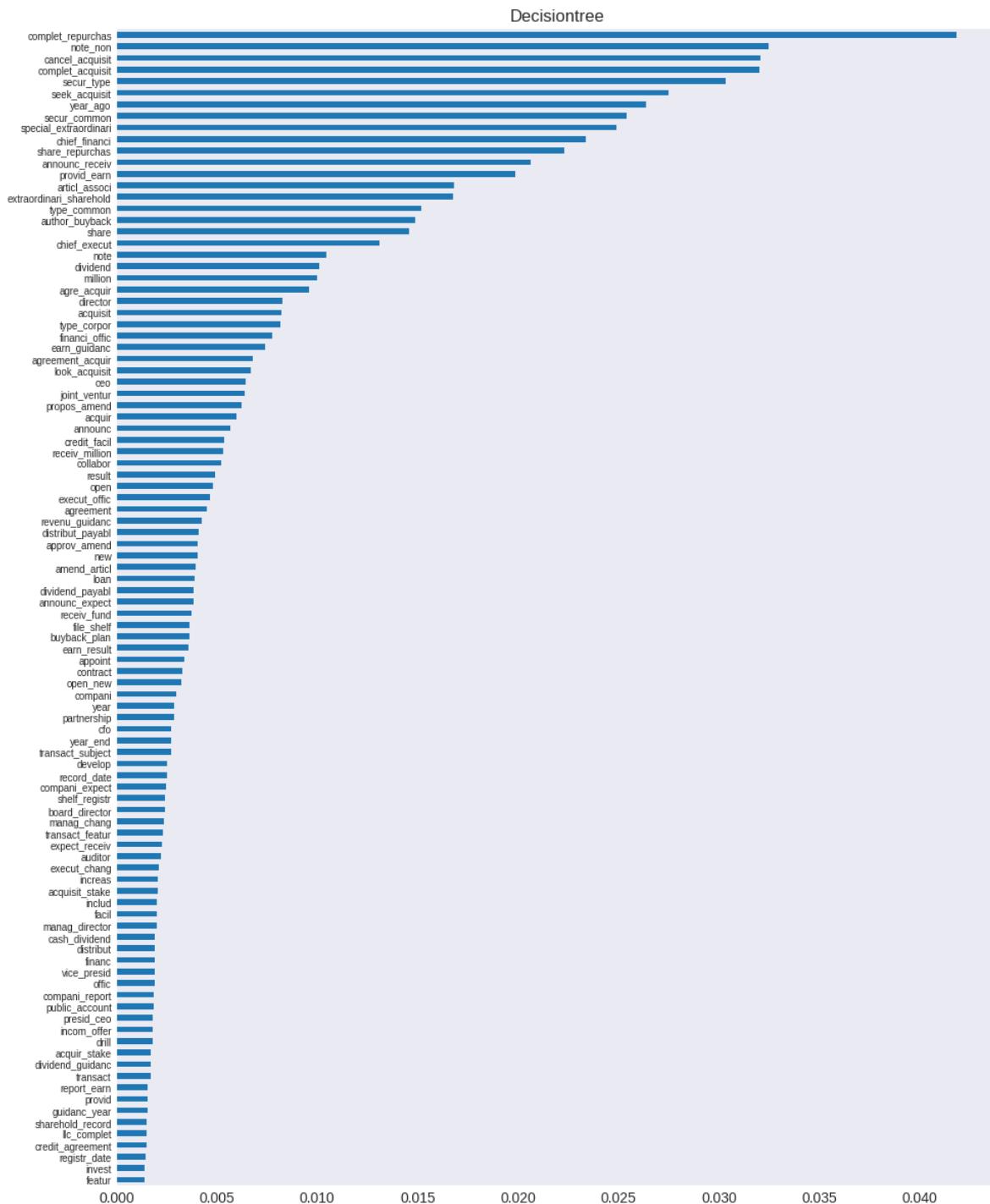
(continues on next page)

(continued from previous page)

```
loc='lower left')
plt.tight_layout()
plt.savefig(imgdir / 'accuracy.jpg')
```



```
## Plot Feature Importances
top_n = 100
for ifig, name in enumerate(['decisiontree']):
    clf = models[name]
    imp = clf.feature_importances_.flatten()
    words = Series({feature_names[i].replace(' ', '_'): imp[i]
                    for i in np.argsort(abs(imp))})
    fig, ax = plt.subplots(clear=True, num=1, figsize=(10, 12))
    words.iloc[-top_n:].plot(kind='barh', color='C0', ax=ax)
    ax.set_title(name.capitalize())
    ax.yaxis.set_tick_params(labelsize=7)
    plt.tight_layout()
    plt.savefig(imgdir / f"{name}.jpg")
```



```
top_n = 6
for ifig, name in enumerate(['logistic', 'naivebayes', 'linearsvc']):
    clf = models[name]
    fig, axes = plt.subplots(5, 5, figsize=(10, 12), num=1+ifig, clear=True)
    axes = [ax for axs in axes for ax in axs]
    for topic, ax in enumerate(axes[:len(clf.classes_)]):
        print("topic %d %s:" % (topic, events_[clf.classes_[topic]]))
```

(continues on next page)

(continued from previous page)

```

assert hasattr(clf, 'coef_') or hasattr(clf, 'feature_log_prob_')
if hasattr(clf, 'coef_'):
    importance = clf.coef_[topic, :]
else:
    importance = clf.feature_log_prob_[topic, :]
words = {feature_names[i].replace(' ', '_'): importance[i]
         for i in importance.argsort()[:-top_n - 1:-1]}
Series(words).plot(kind='barh', color='C0', ax=ax)
ax.set_title(events_[clf.classes_[topic]], fontdict={'fontsize':8})
ax.yaxis.set_tick_params(labelsize=7)
plt.tight_layout()
plt.savefig(imgdir / f"{name}.jpg")

```

```

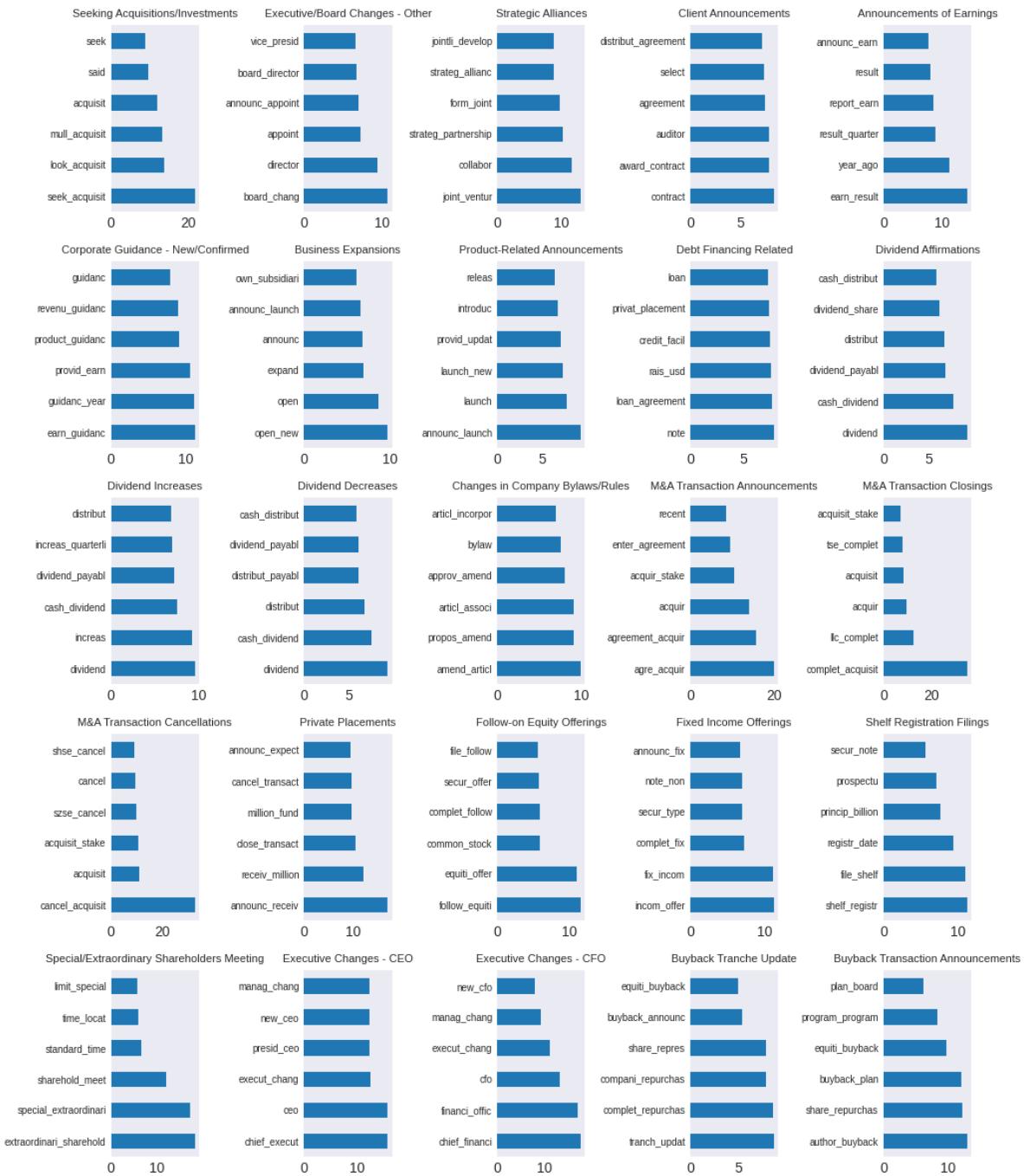
topic 0 Seeking Acquisitions/Investments:
topic 1 Executive/Board Changes - Other:
topic 2 Strategic Alliances:
topic 3 Client Announcements:
topic 4 Announcements of Earnings:
topic 5 Corporate Guidance - New/Confirmed:
topic 6 Business Expansions:
topic 7 Product-Related Announcements:
topic 8 Debt Financing Related:
topic 9 Dividend Affirmations:
topic 10 Dividend Increases:
topic 11 Dividend Decreases:
topic 12 Changes in Company Bylaws/Rules:
topic 13 M&A Transaction Announcements:
topic 14 M&A Transaction Closings:
topic 15 M&A Transaction Cancellations:
topic 16 Private Placements:
topic 17 Follow-on Equity Offerings:
topic 18 Fixed Income Offerings:
topic 19 Shelf Registration Filings:
topic 20 Special/Extraordinary Shareholders Meeting:
topic 21 Executive Changes - CEO:
topic 22 Executive Changes - CFO:
topic 23 Buyback Tranche Update:
topic 24 Buyback Transaction Announcements:
topic 0 Seeking Acquisitions/Investments:
topic 1 Executive/Board Changes - Other:
topic 2 Strategic Alliances:
topic 3 Client Announcements:
topic 4 Announcements of Earnings:
topic 5 Corporate Guidance - New/Confirmed:
topic 6 Business Expansions:
topic 7 Product-Related Announcements:
topic 8 Debt Financing Related:
topic 9 Dividend Affirmations:
topic 10 Dividend Increases:
topic 11 Dividend Decreases:
topic 12 Changes in Company Bylaws/Rules:
topic 13 M&A Transaction Announcements:
topic 14 M&A Transaction Closings:
topic 15 M&A Transaction Cancellations:
topic 16 Private Placements:
topic 17 Follow-on Equity Offerings:

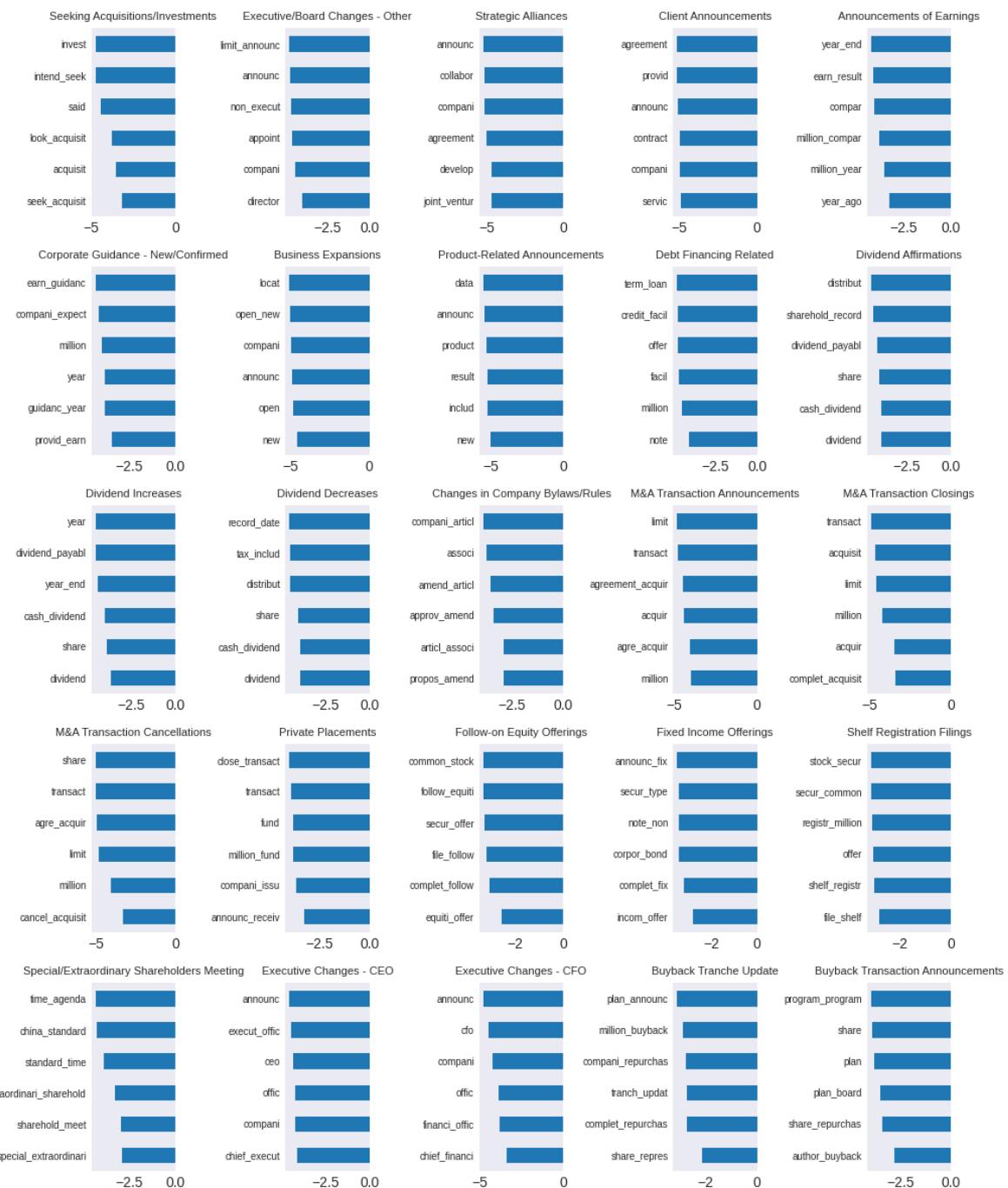
```

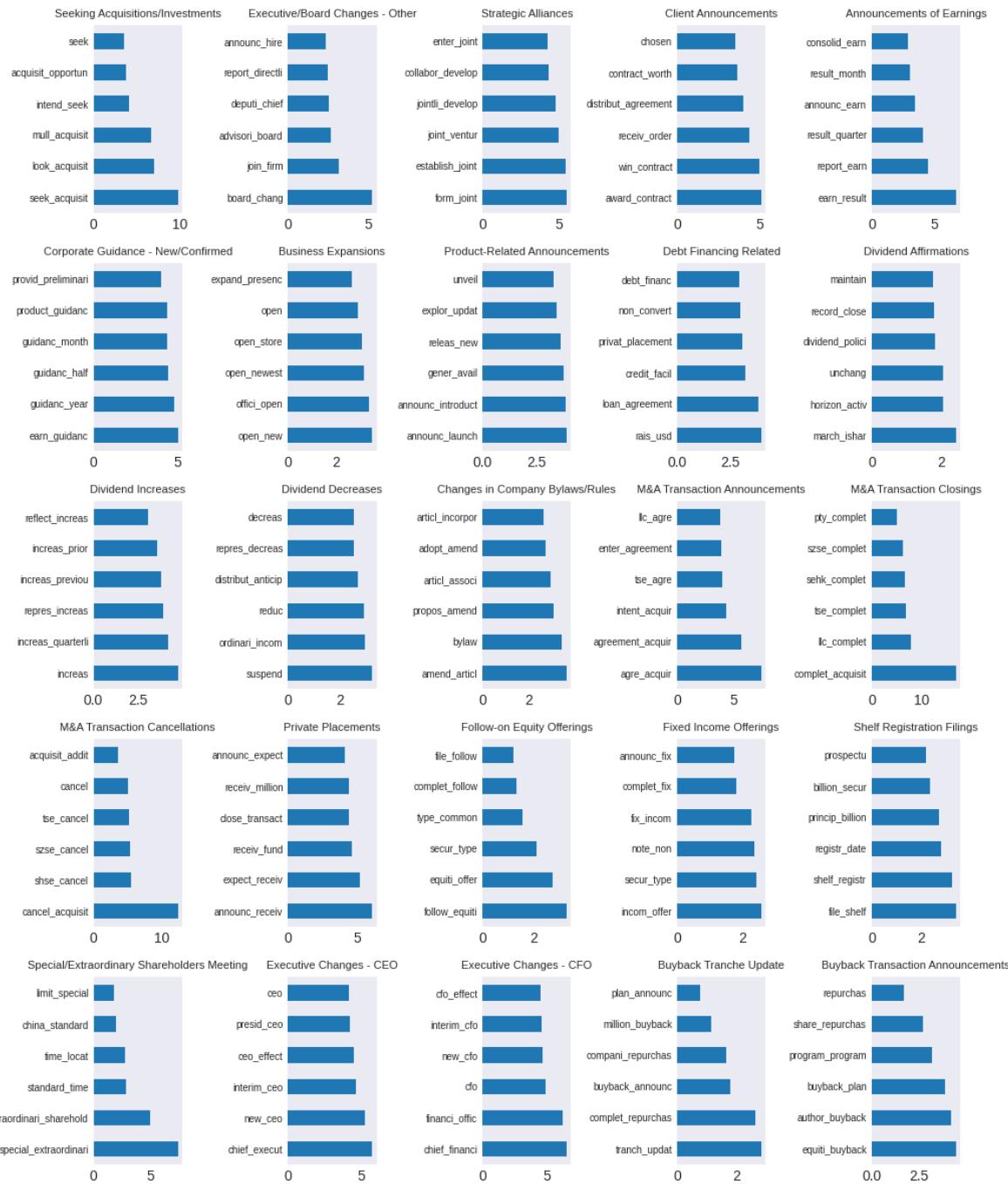
(continues on next page)

(continued from previous page)

```
topic 18 Fixed Income Offerings:  
topic 19 Shelf Registration Filings:  
topic 20 Special/Extraordinary Shareholders Meeting:  
topic 21 Executive Changes - CEO:  
topic 22 Executive Changes - CFO:  
topic 23 Buyback Tranche Update:  
topic 24 Buyback Transaction Announcements:  
topic 0 Seeking Acquisitions/Investments:  
topic 1 Executive/Board Changes - Other:  
topic 2 Strategic Alliances:  
topic 3 Client Announcements:  
topic 4 Announcements of Earnings:  
topic 5 Corporate Guidance - New/Confirmed:  
topic 6 Business Expansions:  
topic 7 Product-Related Announcements:  
topic 8 Debt Financing Related:  
topic 9 Dividend Affirmations:  
topic 10 Dividend Increases:  
topic 11 Dividend Decreases:  
topic 12 Changes in Company Bylaws/Rules:  
topic 13 M&A Transaction Announcements:  
topic 14 M&A Transaction Closings:  
topic 15 M&A Transaction Cancellations:  
topic 16 Private Placements:  
topic 17 Follow-on Equity Offerings:  
topic 18 Fixed Income Offerings:  
topic 19 Shelf Registration Filings:  
topic 20 Special/Extraordinary Shareholders Meeting:  
topic 21 Executive Changes - CEO:  
topic 22 Executive Changes - CFO:  
topic 23 Buyback Tranche Update:  
topic 24 Buyback Transaction Announcements:
```









## REGRESSION MODELS

### UNDER CONSTRUCTION

- subset selection, partial least squares, ridge, gradient boost, random forest
- sklearn, statsmodels

```
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.plots import plot_date
from finds.misc.show import Show
from secret import credentials, paths
# %matplotlib qt
VERBOSE = 0
show = Show(ndigits=4, latex=None)

imgdir = paths['images'] / 'regression'
alf = Alfred(api_key=credentials['fred']['api_key'], verbose=-1)

### Get FRED-MD INDPRO data
df, t = fred_md(202205)
transforms = t['transform']

### Splice in common updates: source of PE ratio, Commercial Paper
for col in ['S&P PE ratio', 'CP3M']:
    df[col] = alf.splice(col)
df['COMPAPFF'] = df['COMPAPFF'].fillna(method='ffill') # forward fill 20200430

### Apply transformations
transformed = []
freq = 'M'
beg = 19640701 # 19620701
end = 20220331 # ignore 2020?
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)]
```

(continues on next page)

(continued from previous page)

```
### 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

### Split time series train (through 2017) and test set (17+ quarters)
target_id = 'INDPRO'
def ts_split(X, Y, end=20171231):
    """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):
    return {col: col + '_' + str(lag) for col in columns}
def columns_unmap(columns):
    cols = [col.split('_') for col in columns]
    return [col[0] for col in cols], [int(col[1]) for col in cols]

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)
test = Series(name='test', dtype=float) # collect test and train errors
test_robust = Series(name='test_no_extremes', dtype=float)
train = Series(name='train', dtype=float)
final_models = {} # collect final fitted models
```

```
monthly/2022-05.csv
original: (693, 127) dropna: (693, 122)
series_id
CP3M          393
ACOGNO        332
UMCSENT        163
TWEXAFEGSMTH   103
ANDENO         44
Name: count, dtype: int64
```

## 35.1 Forward Selection

```

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()

### split train/test and forward select
X_train, X_test, Y_train, Y_test = ts_split(X, Y)
tic = time.time()
selected = []
models = {}

### find best bic, and show selection criteria scores
ic = 'bic' # select by information criterion
for i in range(1, 32):
    select = forward_select(Y_train,
                            X_train,
                            selected,
                            ic=ic)
    models.update({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']]
show(subset, caption='Forward Selection Subset', max_colwidth=60)

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')

### Plot BIC vs number selected
fig, ax = plt.subplots(num=1, clear=True, figsize=(5, 3))
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()

```

(continues on next page)

(continued from previous page)

```

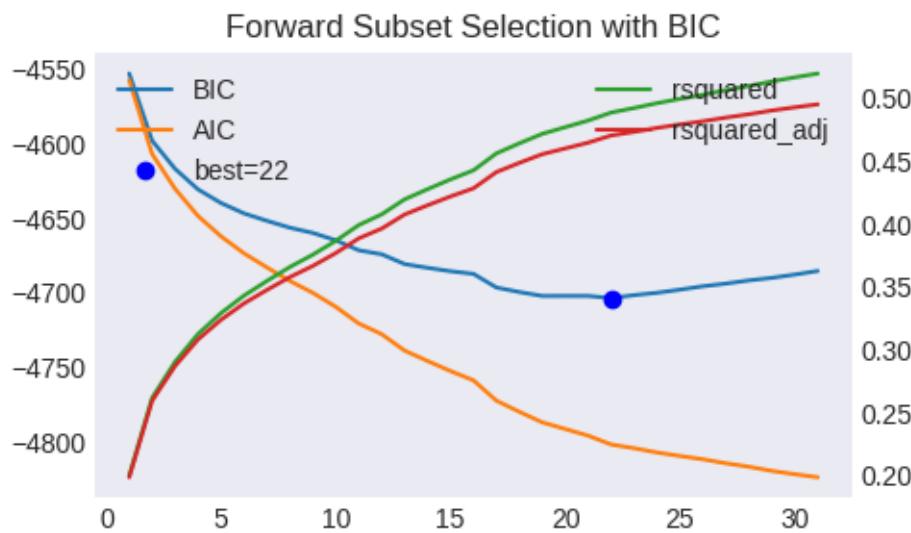
plt.savefig(imgdir / 'forward.jpg')

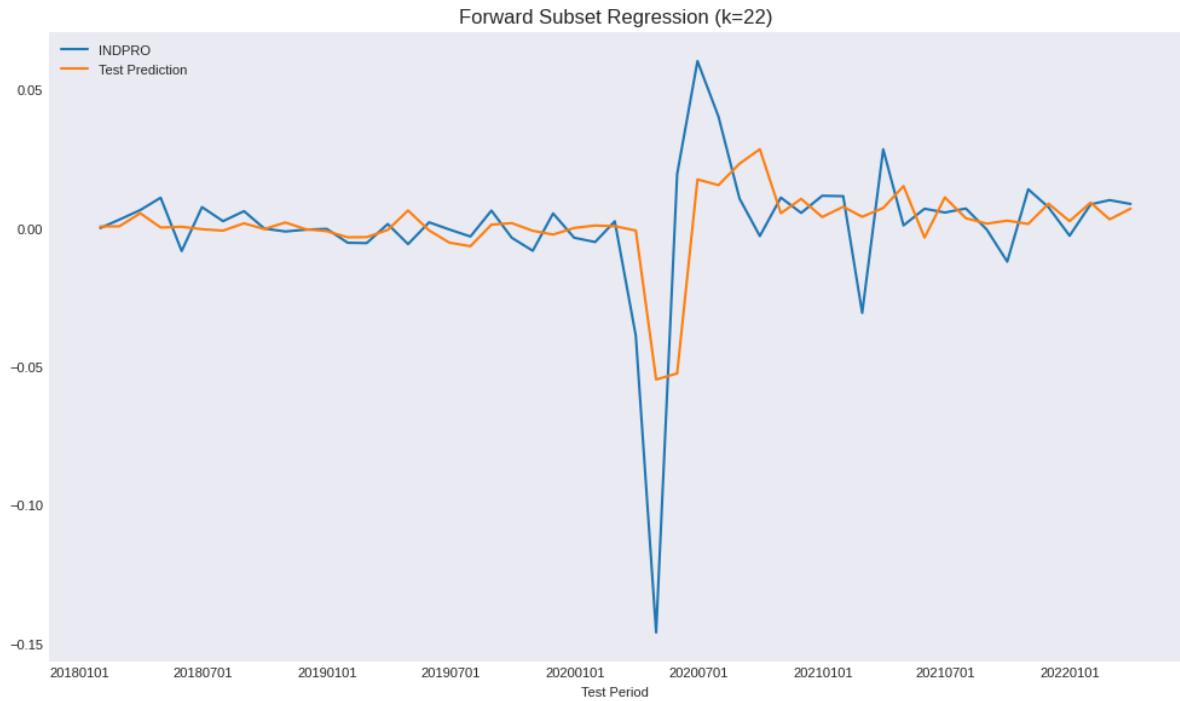
### evaluate train and test mse
X_subset = X_train[subset['select']]
model = sm.OLS(Y_train, X_subset).fit()
name = f"Forward Subset Regression (k={len(subset)}) "
Y_pred = model.predict(X_test[subset['select']])
test[name] = mean_squared_error(Y_test, Y_pred)
robust = Y_test.abs() < 2.33 * np.std(Y_test)
test_robust[name] = mean_squared_error(Y_test[robust], Y_pred[robust])
train[name] = mean_squared_error(Y_train, model.predict(X_subset))
final_models[name] = model
fig, ax = plt.subplots(figsize=(10, 6))
plot_date(pd.concat([Y_test, Y_pred], axis=1),
          legend1=[target_id, 'Test Prediction'],
          title=name,
          xlabel='Test Period',
          fontsize=8,
          ax=ax)
plt.tight_layout()
plt.savefig(imgdir / 'forward_test.jpg')

DataFrame({'name': name,
           'train': np.sqrt(train[name]),
           'test_robust': np.sqrt(test_robust[name]),
           'test': np.sqrt(test[name])}, index=['RMSE'])

```

	name	train	test_robust	test
RMSE	Forward Subset Regression (k=22)	0.005459	0.015444	0.020698





## 35.2 Partial Least Squares Regression

```
### - 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=-1,
                            verbose=VERBOSE,
                            cv=kf,
                            scoring='neg_mean_squared_error').mean()
    mse.loc[i] = -score

### show CV results and best model
fig, ax = plt.subplots(clear=True, num=1, figsize=(5, 3))
mse.plot(ylabel='Mean Squared Error',
          xlabel='Number of Components',
```

(continues on next page)

(continued from previous page)

```

        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()
plt.savefig(imgdir / 'pls.jpg')

### evaluate train and test mse
model = PLSRegression(n_components=best).fit(X_train, Y_train)
name = f"PLS Regression"
Y_pred = Series(index=Y_test.index,
                 data=model.predict(X_test).reshape((-1,)))
test[name] = mean_squared_error(Y_test, Y_pred)
robust = Y_test.abs() < 2.33 * np.std(Y_test)
test_robust[name] = mean_squared_error(Y_test[robust], Y_pred[robust])
train[name] = mean_squared_error(Y_train, model.predict(X_train))
final_models[name] = model

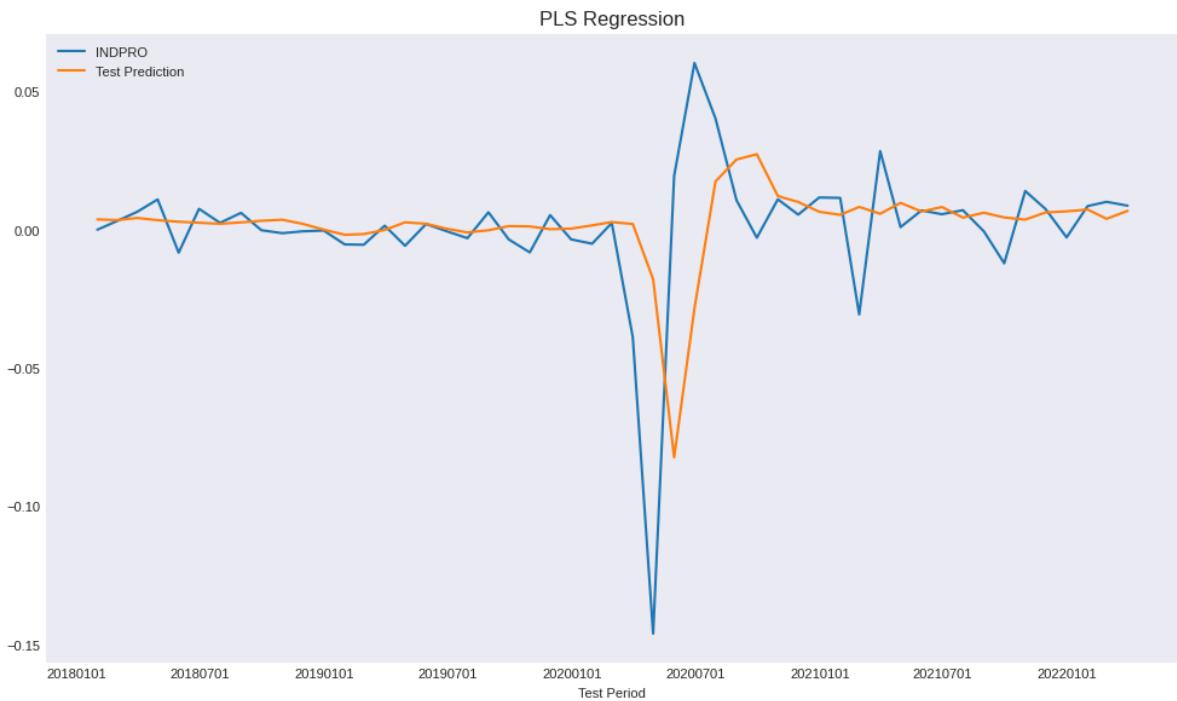
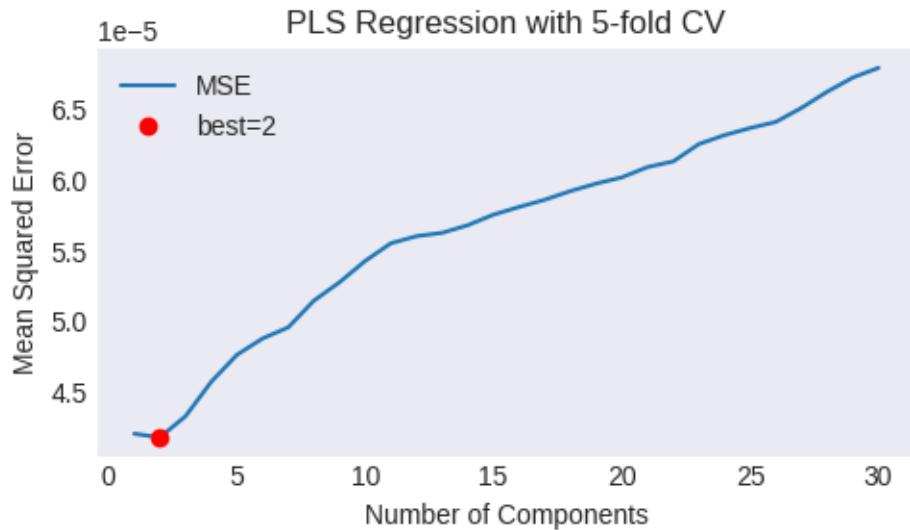
fig, ax = plt.subplots(figsize=(10, 6))
plot_date(pd.concat([Y_test, Y_pred], axis=1),
          legend1=[target_id, 'Test Prediction'],
          title=name,
          xlabel='Test Period',
          fontsize=8,
          ax=ax)
plt.tight_layout()
plt.savefig(imgdir / 'pls_test.jpg')

DataFrame({'name': name,
           'train': np.sqrt(train[name]),
           'test_robust': np.sqrt(test_robust[name]),
           'test': np.sqrt(test[name])}, index=['RMSE'])

#

```

	name	train	test_robust	test
RMSE	PLS Regression	0.00597	0.018655	0.028461



### 35.3 Ridge Regression

```
from sklearn.linear_model import Ridge, RidgeCV
from sklearn.pipeline import Pipeline
alphas = 10**np.linspace(8, -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)
```

(continues on next page)

(continued from previous page)

```

### 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(num=1, clear=True, figsize=(5, 3))
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()
plt.savefig(imgdir / 'ridge.jpg')

### 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)

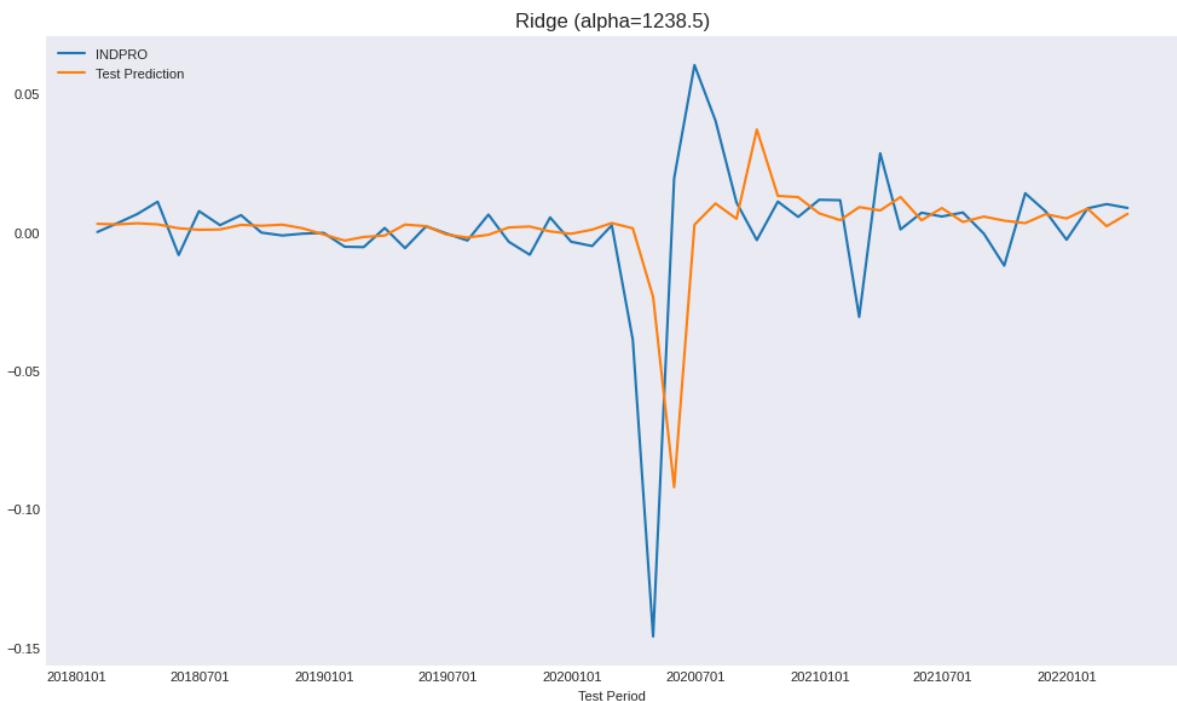
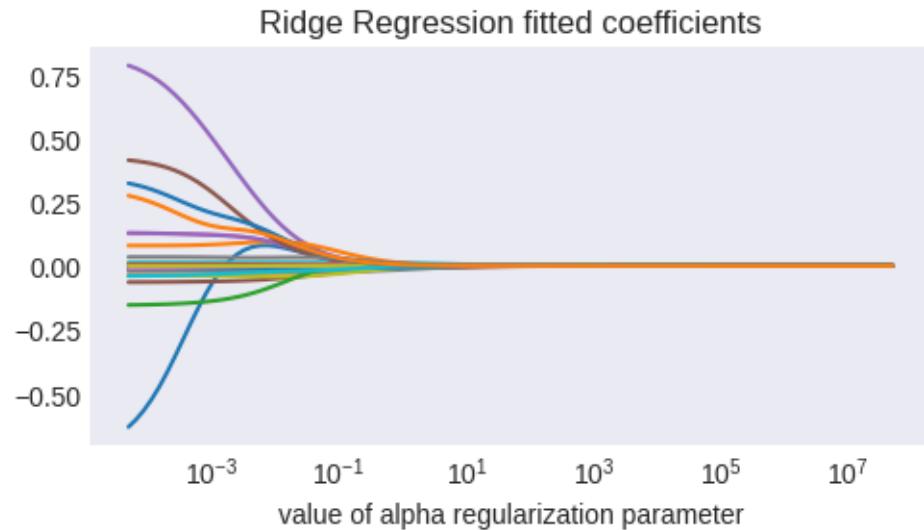
name = f'Ridge (alpha={model.alpha_:.1f})'
Y_pred = Series(index=Y_test.index,
                 data=model.predict(X_test).reshape((-1,)))
test[name] = mean_squared_error(Y_test, Y_pred)
robust = Y_test.abs() < 2.33 * np.std(Y_test)
test_robust[name] = mean_squared_error(Y_test[robust], Y_pred[robust])
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, Y_pred], axis=1),
          legend1=[target_id, 'Test Prediction'],
          title=name,
          xlabel='Test Period',
          fontsize=8,
          ax=ax)
plt.tight_layout()
plt.savefig(imgdir / 'ridge_test.jpg')

DataFrame({'name': name,
           'train': np.sqrt(train[name]),
           'test_robust': np.sqrt(test_robust[name]),
           'test': np.sqrt(test[name])}, index=['RMSE'])

```

	name	train	test_robust	test
RMSE	Ridge (alpha=1238.5)	0.005482	0.020216	0.027426



## 35.4 Lasso Regression

```
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
```

(continues on next page)

(continued from previous page)

```

coefs = [Lasso(max_iter=10000, alpha=alpha) \
         .fit(X_subset, Y_train).coef_ for alpha in alphas]
fig, ax = plt.subplots(num=3, clear=True, figsize=(5, 3))
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()
plt.savefig(imgdir / 'lasso.jpg')

### Lassocv 10-Fold CV
model = LassoCV(alphas=None,
                  cv=10,
                  n_jobs=-1,
                  verbose=VERBOSE,
                  max_iter=10000).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)
robust = Y_test.abs() < 2.33 * np.std(Y_test)
test_robust[name] = mean_squared_error(Y_test[robust], Y_pred[robust])
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, Y_pred], axis=1),
          legend1=[target_id, 'Test Prediction'],
          title=name,
          xlabel='Test Period',
          fontsize=8,
          ax=ax)
plt.tight_layout()
plt.savefig(imgdir / 'lasso_test.jpg')

DataFrame({'name': name,
           'train': np.sqrt(train[name]),
           'test_robust': np.sqrt(test_robust[name]),
           'test': np.sqrt(test[name])}, index=['RMSE'])

### Display nonzero coefs
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')
show(df, max_colwidth=70, caption="Lasso: Nonzero Coefficients")

```

```

/home/terence/env3.11/lib/python3.11/site-packages/sklearn/linear_model/_  

  coordinate_descent.py:617: ConvergenceWarning: Objective did not converge. You  

  might want to increase the number of iterations. Duality gap: 3.939433662976735e-  

  06, tolerance: 3.06196989500973e-06

```

(continues on next page)

(continued from previous page)

```

model = cd_fast.enet_coordinate_descent_gram(
/home/terence/env3.11/lib/python3.11/site-packages/sklearn/linear_model/_co
ordinate_descent.py:617: ConvergenceWarning: Objective did not converge. You
might want to increase the number of iterations. Duality gap: 4.275721962663029e-
06, tolerance: 3.2902038096105723e-06
model = cd_fast.enet_coordinate_descent_gram(
/home/terence/env3.11/lib/python3.11/site-packages/sklearn/linear_model/_co
ordinate_descent.py:617: ConvergenceWarning: Objective did not converge. You
might want to increase the number of iterations. Duality gap: 4.
9042555804960225e-06, tolerance: 3.06196989500973e-06
model = cd_fast.enet_coordinate_descent_gram(

```

	lags
Lasso: Nonzero Coefficients	
NDMANEMP	1 \
CLAIMS	1
CES2000000008	1
S&P div yield	2
USTRADE	1
IPNMAT	3
AAA	1
CES102100001	3
EXCAUS	3
S&P: indust	3
TB3SMFFM	1
USGOOD	1
IPBUSEQ	2
AWOTMAN	2
ISRATIO	2
PERMITMW	1
T1YFFM	1
HWIURATIO	1
RETAIL	2
M2REAL	3
INDPRO	3
IPMANSICS	3
UNRATE	1
CES102100001	1
BAA	2
IPDMAT	1
BAA	3
T5YFFM	3
PERMITMW	2
VIXCLS	2
PCEPI	1
HOUSTMW	1
UEMP5TO14	3
IPMAT	3
TB3MS	2
TB6MS	1
UEMPLT5	1
USWTRADE	1
HWI	3
	desc
Lasso: Nonzero Coefficients	

(continues on next page)

(continued from previous page)

NDMANEMP	All Employees, Nondurable Goods \ Initial Claims
CLAIMS	
CES2000000008	Average Hourly Earnings of Production and Nons...
S&P div yield	S&P's Composite Common Stock: Dividend Yield
USTRADE	All Employees, Retail Trade
IPNMAT	Industrial Production: Non-Durable Goods Mater...
AAA	Moody's Seasoned Aaa Corporate Bond Yield
CES1021000001	All Employees, Mining, Quarrying, and Oil and ...
EXCAUS	Canadian Dollars to U.S. Dollar Spot Exchange ...
S&P: indust	S&P's Common Stock Price Index: Industrials
TB3SMFFM	3-Month Treasury Bill Minus Federal Funds Rate
USGOOD	All Employees, Goods-Producing
IPBUSEQ	Industrial Production: Equipment: Business Equ...
AWOTMAN	Average Weekly Overtime Hours of Production an...
ISRATIO	Total Business: Inventories to Sales Ratio
PERMITMW	New Privately-Owned Housing Units Authorized i...
T1YFFM	1-Year Treasury Constant Maturity Minus Federa...
HWIURATIO	Ratio of Help Wanted/No. Unemployed
RETAIL	Retail and Food Services Sales
M2REAL	Real M2 Money Stock
INDPRO	Industrial Production: Total Index
IPMANSICS	Industrial Production: Manufacturing (SIC)
UNRATE	Unemployment Rate
CES1021000001	All Employees, Mining, Quarrying, and Oil and ...
BAA	Moody's Seasoned Baa Corporate Bond Yield
IPDMAT	Industrial Production: Durable Goods Materials
BAA	Moody's Seasoned Baa Corporate Bond Yield
T5YFFM	5-Year Treasury Constant Maturity Minus Federa...
PERMITMW	New Privately-Owned Housing Units Authorized i...
VIXCLS	CBOE Volatility Index: VIX
PCEPI	Personal Consumption Expenditures: Chain-type ...
HOUSTMW	New Privately-Owned Housing Units Started: Tot...
UEMP5TO14	Number Unemployed for 5-14 Weeks
IPMAT	Industrial Production: Materials
TB3MS	3-Month Treasury Bill Secondary Market Rate, D...
TB6MS	6-Month Treasury Bill Secondary Market Rate, D...
UEMPLT5	Number Unemployed for Less Than 5 Weeks
USWTRADE	All Employees, Wholesale Trade
HWI	Help Wanted Index for United States

### coef

Lasso: Nonzero Coefficients

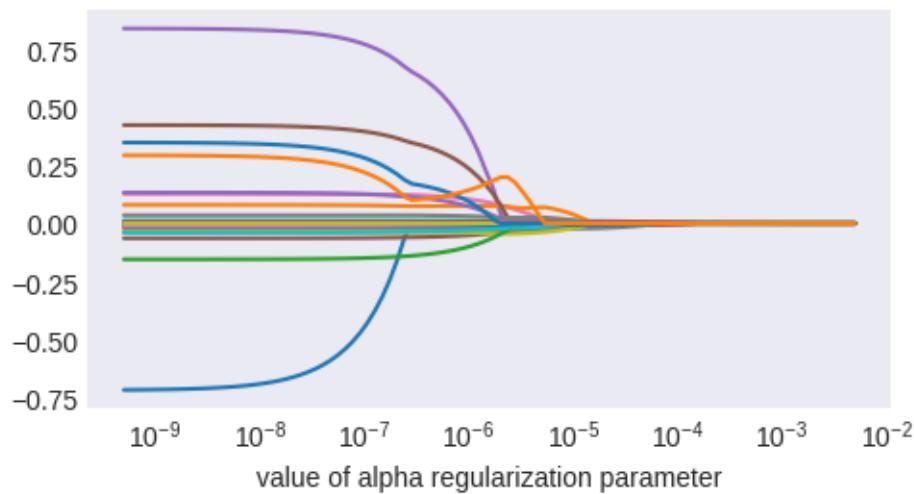
NDMANEMP	0.0008
CLAIMS	-0.0006
CES2000000008	0.0006
S&P div yield	-0.0005
USTRADE	0.0005
IPNMAT	0.0005
AAA	0.0005
CES1021000001	-0.0004
EXCAUS	-0.0004
S&P: indust	0.0004
TB3SMFFM	0.0004
USGOOD	0.0004
IPBUSEQ	0.0003
AWOTMAN	0.0003

(continues on next page)

(continued from previous page)

ISRATIO	-0.0003
PERMITMW	0.0003
T1YFFM	0.0003
HWIURATIO	0.0002
RETAIL	0.0002
M2REAL	0.0002
INDPRO	0.0001
IPMANSICS	0.0001
UNRATE	-0.0001
CES1021000001	0.0001
BAA	-0.0001
IPDMAT	0.0001
BAA	-0.0001
T5YFFM	0.0001
PERMITMW	0.0001
VIXCLS	-0.0001
PCEPI	0.0001
HOUSTMW	0.0001
UEMP5TO14	-0.0001
IPMAT	0.0001
TB3MS	0.0001
TB6MS	0.0000
UEMPLT5	-0.0000
USWTRADE	0.0000
HWI	0.0000

Lasso fitted coefficients





## 35.5 Gradient boost

```

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=-1,
                           verbose=VERBOSE,
                           scoring='neg_mean_squared_error').mean()
    mse.loc[i] = -score

fig, ax = plt.subplots(clear=True, num=1, figsize=(5, 3))
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()

```

(continues on next page)

(continued from previous page)

```

plt.savefig(imgdir / 'boosting.jpg')

### 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)
robust = Y_test.abs() < 2.33 * np.std(Y_test)
test_robust[name] = mean_squared_error(Y_test[robust], Y_pred[robust])
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, Y_pred], axis=1),
          legend1=[target_id, 'Test Prediction'],
          title=name,
          xlabel='Test Period',
          fontsize=8,
          ax=ax)
plt.tight_layout()
plt.savefig(imgdir / 'boosting_test.jpg')

DataFrame({'name': name,
           'train': np.sqrt(train[name]),
           'test_robust': np.sqrt(test_robust[name]),
           'test': np.sqrt(test[name])}, index=['RMSE'])

### Show feature importance
top_n = 10
imp = Series(model.feature_importances_, index=X.columns).sort_values()
show(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'),
      max_colwidth=70, caption="Gradient Boosting: Feature Importances")

```

	importance	series_id	lags
Gradient Boosting: Feature Importances			
1	0.0709	USGOOD	1 \
2	0.0501	CMRMTSPL	2
3	0.0439	IPNMAT	1
4	0.0386	M2REAL	3
5	0.0342	UNRATE	1
6	0.0300	HWIURATIO	1
7	0.0271	DMANEMP	1
8	0.0255	ISRATIO	2
9	0.0254	NDMANEMP	1
10	0.0245	HWIURATIO	3

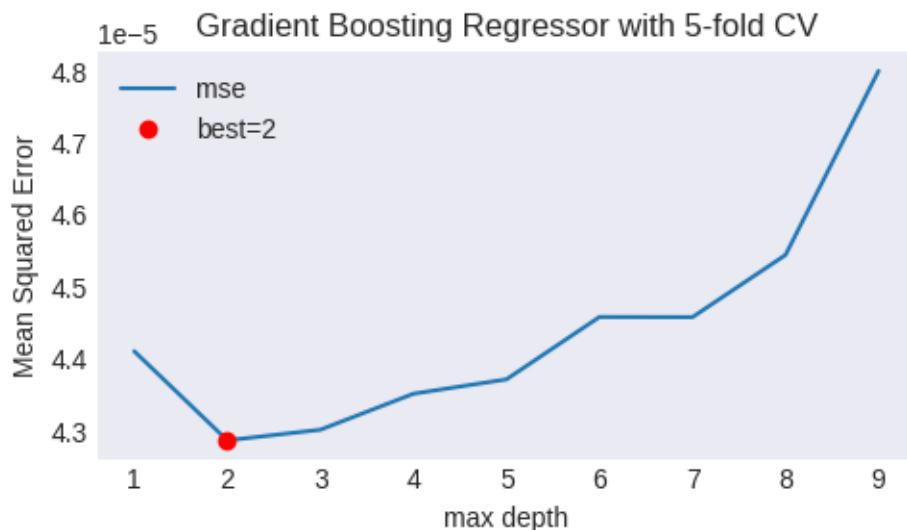
(continues on next page)

(continued from previous page)

```

↳description
Gradient Boosting: Feature Importances
1                                         All Employees, Goods-
↳Producing
2                                         Real Manufacturing and Trade
↳Industries Sales
3                                         Industrial Production: Non-Durable Goods
↳Mater...
4                                         Real M2
↳Money Stock
5
↳Unemployment Rate
6                                         Ratio of Help Wanted/No.
↳Unemployed
7                                         All Employees, ...
↳Durable Goods
8                                         Total Business: Inventories to ...
↳Sales Ratio
9                                         All Employees, ...
↳Nondurable Goods
10                                         Ratio of Help Wanted/No.
↳Unemployed

```





## 35.6 Random Forest

```

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=-1,
                           verbose=VERBOSE,
                           scoring='neg_mean_squared_error').mean()
    mse.loc[i] = -score
    print(i, np.sqrt(abs(score)))

fig, ax = plt.subplots(clear=True, num=1, figsize=(5, 3))
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()

```

(continues on next page)

(continued from previous page)

```

plt.savefig(imgdir / 'forest.jpg')

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)
robust = Y_test.abs() < 2.33 * np.std(Y_test)
test_robust[name] = mean_squared_error(Y_test[robust], Y_pred[robust])
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, Y_pred], axis=1),
          legend1=[target_id, 'Test Prediction'],
          title=name,
          xlabel='Test Period',
          fontsize=8,
          ax=ax)
plt.tight_layout()
plt.savefig(imgdir / 'forest_test.jpg')

DataFrame({'name': name,
           'train': np.sqrt(train[name]),
           'test_robust': np.sqrt(test_robust[name]),
           'test': np.sqrt(test[name])}, index=['RMSE'])
fig, ax = plt.subplots(num=1, clear=True, figsize=(10, 5))
pd.concat([np.sqrt(r.to_frame()) for r in [train, test, test_robust]], 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()
plt.savefig(imgdir / 'rmse.jpg')

```

```

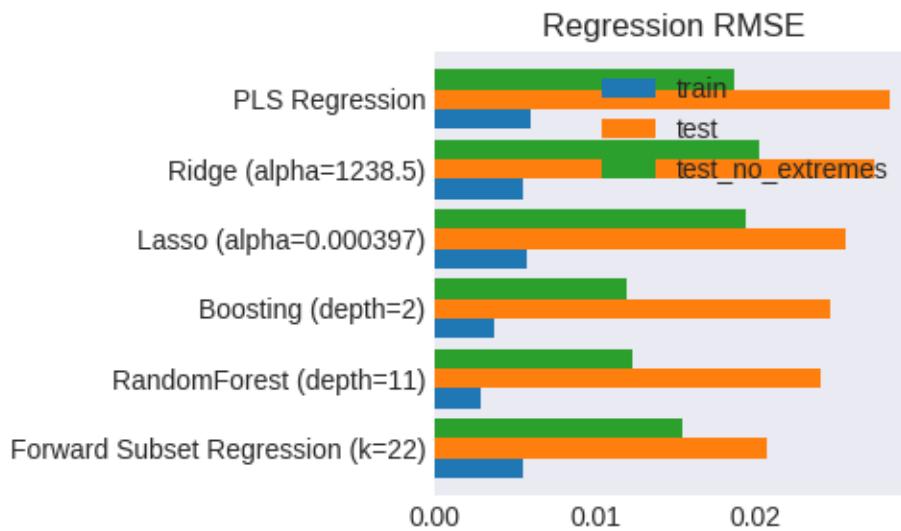
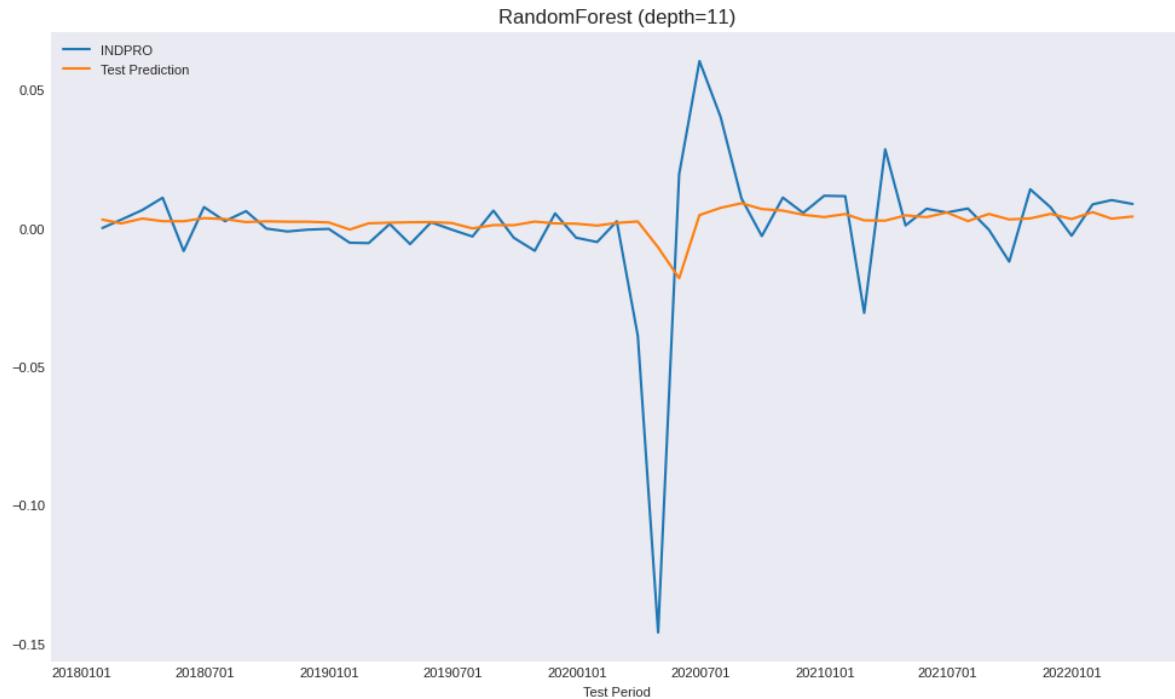
3 0.006658748700334789
4 0.006595819691834476
5 0.006554813834111895
6 0.006525145608947737
7 0.006498070117454981
8 0.0065098589283977635
9 0.006500171449029663
10 0.00651232983315365
11 0.006483624005866995
12 0.0065079602075351186
13 0.006498528900854759
14 0.006500489917980369
15 0.00649950719243888
16 0.006508985777345642
17 0.0065033396940369125
18 0.006504936979291603

```

(continues on next page)

(continued from previous page)

19 0.00650865115862167



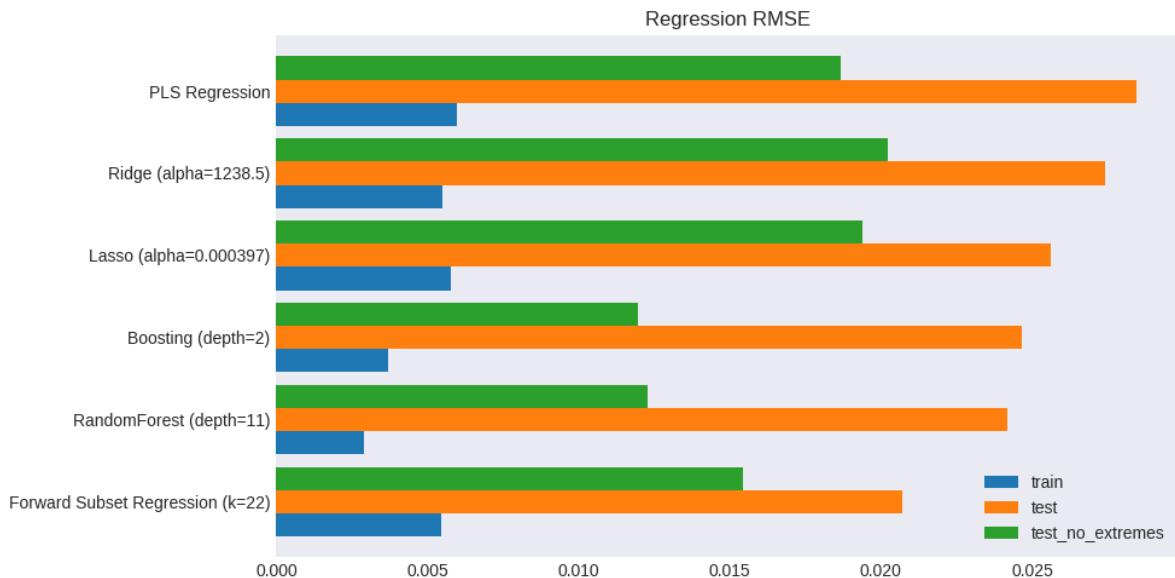
## 35.7 Feature Importances

```
top_n = 10
imp = Series(model.feature_importances_, index=X.columns).sort_values()
show(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'),
      max_colwidth=70, caption="Feature Importances with Random Forest")
```

	importance	series_id	lags
Feature Importances with Random Forest			
1	0.0510	USGOOD	1 \
2	0.0351	IPNMAT	1
3	0.0265	UNRATE	1
4	0.0224	M2REAL	3
5	0.0222	MANEMP	1
6	0.0185	DMANEMP	1
7	0.0180	TB6SMFFM	1
8	0.0159	CMRMTSPL	2
9	0.0158	PAYEMS	1
10	0.0141	NDMANEMP	1
↳ description			
Feature Importances with Random Forest			
1			All Employees, Goods-
↳ Producing			↳
2			Industrial Production: Non-Durable Goods
↳ Mater...			↳
3			↳
↳ Unemployment Rate			Real M2
4			↳
↳ Money Stock			All Employees, ↳
5			↳
↳ Manufacturing			All Employees, ↳
6			↳
↳ Durable Goods			6-Month Treasury Bill Minus Federal
7			↳
↳ Funds Rate			Real Manufacturing and Trade
8			↳
↳ Industries Sales			All Employees, Total
9			↳
↳ Nonfarm			All Employees, ↳
10			↳
↳ Nondurable Goods			

## 35.8 RMSE's

```
# np.sqrt(train.rename('train').to_frame().join(test.rename('test')))\n\nfig, ax = plt.subplots(num=1, clear=True, figsize=(10, 5))\npd.concat([np.sqrt(r.to_frame()) for r in [train, test, test_robust]], axis=1)\\n    .sort_values('test')\\n    .plot.barh(ax=ax, width=0.85)\nax.yaxis.set_tick_params(labelsize=10)\nax.set_title('Regression RMSE')\nax.figure.subplots_adjust(left=0.35)\nplt.tight_layout()\nplt.savefig(imgdir / 'rmse.jpg')\n\n#\n# TODO\n# 1. predict 5 years, raw (don't no extremes)\n#
```





## DEEP AVERAGING NETWORKS

- Text classification
- NLTK, Textual
- Word vectors: GloVe, relativize, frozen, fine-tuning
- Feedforward Neural Networks: torch, deep averaging networks
- Initialization, dropout, Adam, batching, nonlinearity

### Notes

- jupyter-notebook –NotebookApp.iopub\_data\_rate\_limit=1.0e12
- drop Spacy in frozen dan, use GloVe for frozen and fine-tune

```
import numpy as np
import time
import re
import csv, gzip, json
import pandas as pd
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import seaborn as sns
from tqdm import tqdm
from collections import Counter
from nltk.tokenize import RegexpTokenizer
import torch
import torch.nn as nn
import random
import pickle
from finds.database.mongodb import MongoDB
from finds.unstructured import Unstructured
from finds.structured.pstat import PSTAT
from finds.unstructured.textual import Textual
from finds.misc.show import Show
from secret import credentials, paths
# %matplotlib qt
VERBOSE = 0
show = Show(ndigits=4, latex=None)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(device)
```

cuda

```

mongodb = MongoDB(**credentials['mongodb'])
keydev = Unstructured(mongodb, 'KeyDev')
imgdir = paths['images'] / 'classify'
events_ = PSTAT._event
roles_ = PSTAT._role
memdir = paths['scratch']

```

```

{'host': 'omen3080', 'version': '5.0.20', 'process': 'mongod', 'pid': 4153196,
 'uptime': 160319.0, 'uptimeMillis': 160319056, 'uptimeEstimate': 160319,
 'localTime': datetime.datetime(2023, 9, 2, 14, 30, 14, 562000), 'asserts': {
     'regular': 0, 'warning': 0, 'msg': 0, 'user': 215, 'tripwire': 0, 'rollovers': 0
 }, 'catalogStats': {'collections': 7, 'capped': 0, 'timeseries': 0, 'views': 0,
     'internalCollections': 3, 'internalViews': 0}, 'connections': {'current': 3,
     'available': 51197, 'totalCreated': 63, 'active': 2, 'threaded': 3,
     'exhaustIsMaster': 0, 'exhaustHello': 0, 'awaitingTopologyChanges': 1},
     'electionMetrics': {'stepUpCmd': {'called': 0, 'successful': 0},
     'priorityTakeover': {'called': 0, 'successful': 0}, 'catchUpTakeover': {'called':
     0, 'successful': 0}, 'electionTimeout': {'called': 0, 'successful': 0},
     'freezeTimeout': {'called': 0, 'successful': 0}, 'numStepDownsCausedByHigherTerm
     ': 0, 'numCatchUps': 0, 'numCatchUpsSucceeded': 0, 'numCatchUpsAlreadyCaughtUp': 0,
     'numCatchUpsSkipped': 0, 'numCatchUpsTimedOut': 0, 'numCatchUpsFailedWithError
     ': 0, 'numCatchUpsFailedWithNewTerm': 0,
     'numCatchUpsFailedWithReplSetAbortPrimaryCatchUpCmd': 0, 'averageCatchUpOps': 0.
     0}, 'extra_info': {'note': 'fields vary by platform', 'user_time_us': 789054356,
     'system_time_us': 392463045, 'maximum_resident_set_kb': 1539464, 'input_blocks': 11188136,
     'output_blocks': 1146352, 'page_reclaims': 398780, 'page_faults': 51490,
     'voluntary_context_switches': 5739326, 'involuntary_context_switches': 64287},
     'flowControl': {'enabled': True, 'targetRateLimit': 1000000000,
     'timeAcquiringMicros': 56964, 'locksPerKiloOp': 0.0, 'sustainerRate': 0,
     'isLagged': False, 'isLaggedCount': 0, 'isLaggedTimeMicros': 0}, 'freeMonitoring
     ': {'state': 'undecided'}, 'globalLock': {'totalTime': 160319123000,
     'currentQueue': {'total': 0, 'readers': 0, 'writers': 0}, 'activeClients': {
     'total': 0, 'readers': 0, 'writers': 0}}, 'indexBulkBuilder': {'count': 0,
     'resumed': 0, 'filesOpenedForExternalSort': 0, 'filesClosedForExternalSort': 0},
     'indexStats': {'count': 8, 'features': {'2d': {'count': 0, 'accesses': 0},
     '2dsphere': {'count': 0, 'accesses': 0}, 'collation': {'count': 0, 'accesses': 0},
     'compound': {'count': 0, 'accesses': 0}, 'hashed': {'count': 0, 'accesses': 0},
     'id': {'count': 7, 'accesses': 0}, 'normal': {'count': 1, 'accesses': 0},
     'partial': {'count': 0, 'accesses': 0}, 'single': {'count': 1, 'accesses': 0},
     'sparse': {'count': 0, 'accesses': 0}, 'text': {'count': 0, 'accesses': 0}, 'ttl
     ': {'count': 0, 'accesses': 0}, 'unique': {'count': 1, 'accesses': 0}, 'wildcard
     ': {'count': 0, 'accesses': 0}}}, 'locks': {'ParallelBatchWriterMode': {
     'acquireCount': {'r': 2712}}, 'FeatureCompatibilityVersion': {'acquireCount': {'r
     ': 564699, 'w': 2708}}, 'ReplicationStateTransition': {'acquireCount': {'w': 163022}},
     'Global': {'acquireCount': {'r': 564699, 'w': 2703, 'W': 5}}, 'Database
     ': {'acquireCount': {'r': 8, 'w': 2702, 'R': 1, 'W': 1}}, 'Collection': {
     'acquireCount': {'r': 16, 'w': 2700, 'W': 2}}, 'Mutex': {'acquireCount': {'r': 4817}},
     'logicalSessionRecordCache': {'activeSessionsCount': 1,
     'sessionsCollectionJobCount': 535, 'lastSessionsCollectionJobDurationMillis': 0,
     'lastSessionsCollectionJobTimestamp': datetime.datetime(2023, 9, 2, 14, 28, 15,
     977000), 'lastSessionsCollectionJobEntriesRefreshed': 0,
     'lastSessionsCollectionJobEntriesEnded': 0,
     'lastSessionsCollectionJobCursorsClosed': 0, 'transactionReaperJobCount': 535,
     'lastTransactionReaperJobDurationMillis': 0, 'lastTransactionReaperJobTimestamp
     ': datetime.datetime(2023, 9, 2, 14, 28, 15, 977000),
     'lastTransactionReaperJobEntriesCleanedUp': 0, 'sessionCatalogSize': 0}, 'network
     
```

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

```
↳nMALLOC: -----\\nMALLOC: = 1616781312 ( 1541.9 MiB) Virtual address_
↳space used\\nMALLOC:\\nMALLOC: 32172 Spans in use\\nMALLOC: _
↳ 33 Thread heaps in use\\nMALLOC: 4096
↳ Tcmalloc page size\\n-----
↳ReleaseFreeMemory() to release freelist memory to the OS (via madvise()).\\nBytes_
↳released to the OS take up virtual address space but no physical memory.\\n'}},
↳'tenantMigrations': {'currentMigrationsDonating': 0, 'currentMigrationsReceiving_
': 0, 'totalSuccessfulMigrationsDonated': 0, 'totalSuccessfulMigrationsReceived_
': 0, 'totalFailedMigrationsDonated': 0, 'totalFailedMigrationsReceived': 0},
↳'trafficRecording': {'running': False}, 'transactions': {'retriedCommandsCount':_
0, 'retriedStatementsCount': 0, 'transactionsCollectionWriteCount': 0,
↳'currentActive': 0, 'currentInactive': 0, 'currentOpen': 0, 'totalAborted': 0,
↳'totalCommitted': 0, 'totalStarted': 0, 'totalPrepared': 0,
↳'totalPreparedThenCommitted': 0, 'totalPreparedThenAborted': 0, 'currentPrepared_
': 0}, 'transportSecurity': {'1.0': 0, '1.1': 0, '1.2': 0, '1.3': 0, 'unknown':_
0}, 'twoPhaseCommitCoordinator': {'totalCreated': 0, 'totalStartedTwoPhaseCommit_
': 0, 'totalAbortedTwoPhaseCommit': 0, 'totalCommittedTwoPhaseCommit': 0,
↳'currentInSteps': {'writingParticipantList': 0, 'waitingForVotes': 0,
↳'writingDecision': 0, 'waitingForDecisionAcks': 0, 'deletingCoordinatorDoc': 0}},
↳'wiredTiger': {'uri': 'statistics:', 'block-manager': {'block cache cached_
blocks updated': 0, 'block cache cached bytes updated': 0, 'block cache evicted_
blocks': 0, 'block cache file size causing bypass': 0, 'block cache lookups': 0,
↳'block cache number of blocks not evicted due to overhead': 0, 'block cache_
number of bypasses because no-write-allocate setting was on': 0, 'block cache_
number of bypasses due to overhead on put': 0, 'block cache number of bypasses_
on get': 0, 'block cache number of bypasses on put because file is too small': 0,
↳'block cache number of eviction passes': 0, 'block cache number of hits_
including existence checks': 0, 'block cache number of misses including_
existence checks': 0, 'block cache number of put bypasses on checkpoint I/O': 0,
↳'block cache removed blocks': 0, 'block cache total blocks': 0, 'block cache_
total blocks inserted on read path': 0, 'block cache total blocks inserted on_
write path': 0, 'block cache total bytes': 0, 'block cache total bytes inserted_
on read path': 0, 'block cache total bytes inserted on write path': 0, 'blocks_
pre-loaded': 112, 'blocks read': 31435, 'blocks written': 13598, 'bytes read':_
689737728, 'bytes read via memory map API': 0, 'bytes read via system call API':_
0, 'bytes written': 131997696, 'bytes written for checkpoint': 131997696, 'bytes_
written via memory map API': 0, 'bytes written via system call API': 0, 'mapped_
blocks read': 0, 'mapped bytes read': 0, 'number of times the file was remapped_
because it changed size via fallocate or truncate': 0, 'number of times the_
region was remapped via write': 0}, 'cache': {'application threads page read_
from disk to cache count': 28610, 'application threads page read from disk to_
cache time (usecs)': 2187549, 'application threads page write from cache to disk_
count': 5643, 'application threads page write from cache to disk time (usecs)':_
935391, 'bytes allocated for updates': 306203, 'bytes belonging to page images_
in the cache': 1538128908, 'bytes belonging to the history store table in the_
cache': 588, 'bytes currently in the cache': 1560878378, 'bytes dirty in the_
cache cumulative': 133255158, 'bytes not belonging to page images in the cache':_
22749469, 'bytes read into cache': 1424193434, 'bytes written from cache':_
78147848, 'cache overflow score': 0, 'checkpoint blocked page eviction': 0,
↳'checkpoint of history store file blocked non-history store page eviction': 0,
↳'eviction calls to get a page': 14737, 'eviction calls to get a page found queue_
empty': 11452, 'eviction calls to get a page found queue empty after locking':_
316, 'eviction currently operating in aggressive mode': 0, 'eviction empty score_
': 0, 'eviction gave up due to detecting an out of order on disk value behind_
the last update on the chain': 0, 'eviction gave up due to detecting an out of_
order tombstone ahead of the selected on disk update': 0, 'eviction gave up due_
```

(continues on next page)

(continued from previous page)

↵to detecting an out of order tombstone ahead of the selected on disk update  
 ↵after validating the update chain': 0, 'eviction gave up due to detecting out of  
 ↵order timestamps on the update chain after the selected on disk update': 0,  
 ↵'eviction gave up due to needing to remove a record from the history store but  
 ↵checkpoint is running': 0, 'eviction passes of a file': 0, 'eviction server  
 ↵candidate queue empty when topping up': 0, 'eviction server candidate queue not  
 ↵empty when topping up': 0, 'eviction server evicting pages': 0, 'eviction server  
 ↵slept, because we did not make progress with eviction': 3684, 'eviction server  
 ↵unable to reach eviction goal': 0, 'eviction server waiting for a leaf page': 23,  
 ↵'eviction state': 64, 'eviction walk most recent sleeps for checkpoint handle  
 ↵gathering': 0, 'eviction walk target pages histogram - 0-9': 0, 'eviction walk  
 ↵target pages histogram - 10-31': 0, 'eviction walk target pages histogram - 128  
 ↵and higher': 0, 'eviction walk target pages histogram - 32-63': 0, 'eviction  
 ↵walk target pages histogram - 64-128': 0, 'eviction walk target pages reduced  
 ↵due to history store cache pressure': 0, 'eviction walk target strategy both  
 ↵clean and dirty pages': 0, 'eviction walk target strategy only clean pages': 0,  
 ↵'eviction walk target strategy only dirty pages': 0, 'eviction walks abandoned':  
 ↵0, 'eviction walks gave up because they restarted their walk twice': 0,  
 ↵'eviction walks gave up because they saw too many pages and found no candidates  
 ↵': 0, 'eviction walks gave up because they saw too many pages and found too few  
 ↵candidates': 0, 'eviction walks reached end of tree': 0, 'eviction walks  
 ↵restarted': 0, 'eviction walks started from root of tree': 0, 'eviction walks  
 ↵started from saved location in tree': 0, 'eviction worker thread active': 4,  
 ↵'eviction worker thread created': 0, 'eviction worker thread evicting pages':  
 ↵2340, 'eviction worker thread removed': 0, 'eviction worker thread stable number  
 ↵': 0, 'files with active eviction walks': 0, 'files with new eviction walks  
 ↵started': 0, 'force re-tuning of eviction workers once in a while': 0, 'forced  
 ↵eviction - history store pages failed to evict while session has history store  
 ↵cursor open': 0, 'forced eviction - history store pages selected while session  
 ↵has history store cursor open': 0, 'forced eviction - history store pages  
 ↵successfully evicted while session has history store cursor open': 0, 'forced  
 ↵eviction - pages evicted that were clean count': 0, 'forced eviction - pages  
 ↵evicted that were clean time (usecs)': 0, 'forced eviction - pages evicted that  
 ↵were dirty count': 0, 'forced eviction - pages evicted that were dirty time  
 ↵(usecs)': 0, 'forced eviction - pages selected because of a large number of  
 ↵updates to a single item': 0, 'forced eviction - pages selected because of too  
 ↵many deleted items count': 7, 'forced eviction - pages selected count': 0,  
 ↵'forced eviction - pages selected unable to be evicted count': 0, 'forced  
 ↵eviction - pages selected unable to be evicted time': 0, 'hazard pointer blocked  
 ↵page eviction': 1, 'hazard pointer check calls': 2340, 'hazard pointer check  
 ↵entries walked': 130, 'hazard pointer maximum array length': 2, 'history store  
 ↵score': 0, 'history store table insert calls': 0, 'history store table insert  
 ↵calls that returned restart': 0, 'history store table max on-disk size': 0,  
 ↵'history store table on-disk size': 4096, 'history store table out-of-order  
 ↵resolved updates that lose their durable timestamp': 0, 'history store table out-  
 ↵of-order updates that were fixed up by reinserting with the fixed timestamp': 0,  
 ↵'history store table reads': 0, 'history store table reads missed': 0, 'history  
 ↵store table reads requiring squashed modifies': 0, 'history store table  
 ↵truncation by rollback to stable to remove an unstable update': 0, 'history  
 ↵store table truncation by rollback to stable to remove an update': 0, 'history  
 ↵store table truncation to remove an update': 0, 'history store table truncation  
 ↵to remove range of updates due to key being removed from the data page during  
 ↵reconciliation': 0, 'history store table truncation to remove range of updates  
 ↵due to out-of-order timestamp update on data page': 0, 'history store table  
 ↵writes requiring squashed modifies': 0, 'in-memory page passed criteria to be  
 ↵split': 0, 'in-memory page splits': 0, 'internal pages evicted': 0, 'internal

(continues on next page)

(continued from previous page)

```
↳ pages queued for eviction': 0, 'internal pages seen by eviction walk': 0,  
↳ 'internal pages seen by eviction walk that are already queued': 0, 'internal  
↳ pages split during eviction': 0, 'leaf pages split during eviction': 0, 'maximum  
↳ bytes configured': 16098787328, 'maximum milliseconds spent at a single eviction  
↳ ': 0, 'maximum page size seen at eviction': 368, 'modified pages evicted': 2339,  
↳ 'modified pages evicted by application threads': 0, 'operations timed out  
↳ waiting for space in cache': 0, 'overflow pages read into cache': 0, 'page split  
↳ during eviction deepened the tree': 0, 'page written requiring history store  
↳ records': 0, 'pages currently held in the cache': 28628, 'pages evicted by  
↳ application threads': 0, 'pages evicted in parallel with checkpoint': 2332,  
↳ 'pages queued for eviction': 0, 'pages queued for eviction post lru sorting': 0,  
↳ 'pages queued for urgent eviction': 2340, 'pages queued for urgent eviction  
↳ during walk': 0, 'pages queued for urgent eviction from history store due to  
↳ high dirty content': 0, 'pages read into cache': 28623, 'pages read into cache  
↳ after truncate': 2340, 'pages read into cache after truncate in prepare state':  
↳ 0, 'pages removed from the ordinary queue to be queued for urgent eviction': 0,  
↳ 'pages requested from the cache': 4155672, 'pages seen by eviction walk': 0,  
↳ 'pages seen by eviction walk that are already queued': 0, 'pages selected for  
↳ eviction unable to be evicted': 1, 'pages selected for eviction unable to be  
↳ evicted because of active children on an internal page': 0, 'pages selected for  
↳ eviction unable to be evicted because of failure in reconciliation': 0, 'pages  
↳ selected for eviction unable to be evicted because of race between checkpoint  
↳ and out of order timestamps handling': 0, 'pages walked for eviction': 0, 'pages  
↳ written from cache': 5646, 'pages written requiring in-memory restoration': 24,  
↳ 'percentage overhead': 8, 'the number of times full update inserted to history  
↳ store': 0, 'the number of times reverse modify inserted to history store': 0,  
↳ 'total milliseconds spent inside reentrant history store evictions in a  
↳ reconciliation': 0, 'tracked bytes belonging to internal pages in the cache':  
↳ 2313897, 'tracked bytes belonging to leaf pages in the cache': 1558564481,  
↳ 'tracked dirty bytes in the cache': 484, 'tracked dirty pages in the cache': 1,  
↳ 'unmodified pages evicted': 0}, 'capacity': {'background fsync file handles  
↳ considered': 0, 'background fsync file handles synced': 0, 'background fsync  
↳ time (msecs)': 0, 'bytes read': 678219776, 'bytes written for checkpoint':  
↳ 77706843, 'bytes written for eviction': 0, 'bytes written for log': 1007979264,  
↳ 'bytes written total': 1085686107, 'threshold to call fsync': 0, 'time waiting  
↳ due to total capacity (usecs)': 0, 'time waiting during checkpoint (usecs)': 0,  
↳ 'time waiting during eviction (usecs)': 0, 'time waiting during logging (usecs)  
↳ ': 0, 'time waiting during read (usecs)': 0}, 'checkpoint-cleanup': {'pages  
↳ added for eviction': 2333, 'pages removed': 0, 'pages skipped during tree walk':  
↳ 0, 'pages visited': 7919}, 'connection': {'auto adjusting condition resets':  
↳ 12350, 'auto adjusting condition wait calls': 991449, 'auto adjusting condition  
↳ wait raced to update timeout and skipped updating': 0, 'detected system time  
↳ went backwards': 0, 'files currently open': 18, 'hash bucket array size for data  
↳ handles': 512, 'hash bucket array size general': 512, 'memory allocations':  
↳ 7100328, 'memory frees': 7013090, 'memory re-allocations': 642636, 'pthread  
↳ mutex condition wait calls': 2624717, 'pthread mutex shared lock read-lock calls  
↳ ': 2870497, 'pthread mutex shared lock write-lock calls': 178019, 'total fsync I/  
↳ Os': 19169, 'total read I/Os': 35130, 'total write I/Os': 22269}, 'cursor': {  
↳ 'Total number of entries skipped by cursor next calls': 76, 'Total number of  
↳ entries skipped by cursor prev calls': 20, 'Total number of entries skipped to  
↳ position the history store cursor': 0, 'Total number of times a search near has  
↳ exited due to prefix config': 0, 'cached cursor count': 12, 'cursor bulk loaded  
↳ cursor insert calls': 0, 'cursor close calls that result in cache': 1334635,  
↳ 'cursor create calls': 230, 'cursor insert calls': 7936, 'cursor insert key and  
↳ value bytes': 4073675, 'cursor modify calls': 0, 'cursor modify key and value  
↳ bytes affected': 0, 'cursor modify value bytes modified': 0, 'cursor next calls
```

(continues on next page)

(continued from previous page)

```

↳': 241946935, 'cursor next calls that skip due to a globally visible history':
↳'store tombstone': 0, 'cursor next calls that skip greater than or equal to 100':
↳'entries': 0, 'cursor next calls that skip less than 100 entries': 241946933,
↳'cursor operation restarted': 0, 'cursor prev calls': 2363, 'cursor prev calls':
↳'that skip due to a globally visible history store tombstone': 0, 'cursor prev':
↳'calls that skip greater than or equal to 100 entries': 0, 'cursor prev calls':
↳'that skip less than 100 entries': 2363, 'cursor remove calls': 56, 'cursor':
↳'remove key bytes removed': 1446, 'cursor reserve calls': 0, 'cursor reset calls':
↳': 1635777, 'cursor search calls': 13446, 'cursor search history store calls': 0,
↳'cursor search near calls': 244837, 'cursor sweep buckets': 1902993, 'cursor':
↳'sweep cursors closed': 0, 'cursor sweep cursors examined': 15788, 'cursor sweeps':
↳': 194309, 'cursor truncate calls': 0, 'cursor update calls': 0, 'cursor update':
↳'key and value bytes': 0, 'cursor update value size change': 0, 'cursors reused':
↳'from cache': 1334623, 'open cursor count': 6}, 'data-handle': {'connection data':
↳'handle size': 440, 'connection data handles currently active': 29, 'connection':
↳'sweep candidate became referenced': 0, 'connection sweep dhandles closed': 0,
↳'connection sweep dhandles removed from hash list': 2485, 'connection sweep time':
↳'of-death sets': 21361, 'connection sweeps': 16031, 'connection sweeps skipped':
↳'due to checkpoint gathering handles': 0, 'session dhandles swept': 37, 'session':
↳'sweep attempts': 50025}, 'lock': {'checkpoint lock acquisitions': 2671,
↳'checkpoint lock application thread wait time (usecs)': 4, 'checkpoint lock':
↳'internal thread wait time (usecs)': 0, 'dhandle lock application thread time':
↳'waiting (usecs)': 0, 'dhandle lock internal thread time waiting (usecs)': 0,
↳'dhandle read lock acquisitions': 657605, 'dhandle write lock acquisitions':
↳: 5003, 'durable timestamp queue lock application thread time waiting (usecs)': 0,
↳'durable timestamp queue lock internal thread time waiting (usecs)': 0, 'durable':
↳'timestamp queue read lock acquisitions': 0, 'durable timestamp queue write lock':
↳'acquisitions': 0, 'metadata lock acquisitions': 2670, 'metadata lock application':
↳'thread wait time (usecs)': 7, 'metadata lock internal thread wait time (usecs)': 0,
↳'read timestamp queue lock application thread time waiting (usecs)': 0, 'read':
↳'timestamp queue lock internal thread time waiting (usecs)': 0, 'read timestamp':
↳'queue read lock acquisitions': 0, 'read timestamp queue write lock acquisitions':
↳': 0, 'schema lock acquisitions': 2702, 'schema lock application thread wait':
↳'time (usecs)': 23, 'schema lock internal thread wait time (usecs)': 0, 'table':
↳'lock application thread time waiting for the table lock (usecs)': 0, 'table lock':
↳'internal thread time waiting for the table lock (usecs)': 0, 'table read lock':
↳'acquisitions': 0, 'table write lock acquisitions': 15, 'txn global lock':
↳'application thread time waiting (usecs)': 0, 'txn global lock internal thread':
↳'time waiting (usecs)': 0, 'txn global read lock acquisitions': 14247, 'txn':
↳'global write lock acquisitions': 7797}, 'log': {'busy returns attempting to':
↳'switch slots': 2, 'force archive time sleeping (usecs)': 0, 'log bytes of':
↳'payload data': 3413228, 'log bytes written': 4317696, 'log files manually zero-':
↳'filled': 0, 'log flush operations': 1602583, 'log force write operations':
↳: 1778799, 'log force write operations skipped': 1775471, 'log records compressed':
↳': 2527, 'log records not compressed': 169, 'log records too small to compress':
↳: 10738, 'log release advances write LSN': 2671, 'log scan operations': 6, 'log':
↳'scan records requiring two reads': 0, 'log server thread advances write LSN':
↳: 3326, 'log server thread write LSN walk skipped': 159506, 'log sync operations':
↳: 5996, 'log sync time duration (usecs)': 27502822, 'log sync_dir operations': 1,
↳'log sync_dir time duration (usecs)': 8968, 'log write operations': 13434,
↳'logging bytes consolidated': 4317184, 'maximum log file size': 104857600,
↳'number of pre-allocated log files to create': 2, 'pre-allocated log files not':
↳'ready and missed': 1, 'pre-allocated log files prepared': 2, 'pre-allocated log':
↳'files used': 0, 'records processed by log scan': 15, 'slot close lost race': 0,
↳'slot close unbuffered waits': 0, 'slot closures': 5997, 'slot join atomic':
↳'update races': 0, 'slot join calls atomic updates raced': 0, 'slot join calls':

```

(continues on next page)

(continued from previous page)

```
↳ 'did not yield': 13434, 'slot join calls found active slot closed': 0, 'slot join calls slept': 0, 'slot join calls yielded': 0, 'slot join found active slot closed': 0, 'slot joins yield time (usecs)': 0, 'slot transitions unable to find free slot': 0, 'slot unbuffered writes': 0, 'total in-memory size of compressed records': 4126548, 'total log buffer size': 33554432, 'total size of compressed records': 2797250, 'written slots coalesced': 0, 'yields waiting for previous log file close': 0}, 'perf': {'file system read latency histogram (bucket 1) - 10-49ms': 0, 'file system read latency histogram (bucket 2) - 50-99ms': 0, 'file system read latency histogram (bucket 3) - 100-249ms': 0, 'file system read latency histogram (bucket 4) - 250-499ms': 0, 'file system read latency histogram (bucket 5) - 500-999ms': 0, 'file system read latency histogram (bucket 6) - 1000ms+'. 0, 'file system write latency histogram (bucket 1) - 10-49ms': 1, 'file system write latency histogram (bucket 2) - 50-99ms': 0, 'file system write latency histogram (bucket 3) - 100-249ms': 0, 'file system write latency histogram (bucket 4) - 250-499ms': 0, 'file system write latency histogram (bucket 5) - 500-999ms': 0, 'file system write latency histogram (bucket 6) - 1000ms+'. 0, 'operation read latency histogram (bucket 1) - 100-249us': 5, 'operation read latency histogram (bucket 2) - 250-499us': 1, 'operation read latency histogram (bucket 3) - 500-999us': 1, 'operation read latency histogram (bucket 4) - 1000-9999us': 3, 'operation read latency histogram (bucket 5) - 10000us+'. 2, 'operation write latency histogram (bucket 1) - 100-249us': 3, 'operation write latency histogram (bucket 2) - 250-499us': 0, 'operation write latency histogram (bucket 3) - 500-999us': 0, 'operation write latency histogram (bucket 4) - 1000-9999us': 0, 'operation write latency histogram (bucket 5) - 10000us+'. 3}, 'reconciliation': {'approximate byte size of timestamps in pages written': 0, 'approximate byte size of transaction IDs in pages written': 38760, 'fast-path pages deleted': 0, 'leaf-page overflow keys': 0, 'maximum milliseconds spent in a reconciliation call': 0, 'maximum milliseconds spent in building a disk image in a reconciliation': 0, 'maximum milliseconds spent in moving updates to the history store in a reconciliation': 0, 'page reconciliation calls': 12671, 'page reconciliation calls for eviction': 2339, 'page reconciliation calls that resulted in values with prepared transaction metadata': 0, 'page reconciliation calls that resulted in values with timestamps': 0, 'page reconciliation calls that resulted in values with transaction ids': 1720, 'pages deleted': 7028, 'pages written including an aggregated newest start durable timestamp': 0, 'pages written including an aggregated newest stop durable timestamp': 0, 'pages written including an aggregated newest stop timestamp': 0, 'pages written including an aggregated newest stop transaction ID': 0, 'pages written including an aggregated newest transaction ID': 0, 'pages written including an aggregated oldest start timestamp': 0, 'pages written including an aggregated prepare': 0, 'pages written including at least one prepare state': 0, 'pages written including at least one start durable timestamp': 0, 'pages written including at least one start transaction ID': 1720, 'pages written including at least one stop durable timestamp': 0, 'pages written including at least one stop timestamp': 0, 'pages written including at least one stop transaction ID': 0, 'records written including a prepare state': 0, 'records written including a start durable timestamp': 0, 'records written including a start timestamp': 0, 'records written including a stop durable timestamp': 0, 'records written including a stop timestamp': 0, 'records written including a stop transaction ID': 0, 'split bytes currently awaiting free': 0, 'split objects currently awaiting free': 0}, 'session': {'attempts to remove a local object and the object is in use': 0, 'flush_tier operation calls': 0, 'local objects removed': 0, 'open session count': 15, 'session query timestamp calls': 0, 'table alter failed calls': 0, 'table alter successful calls': 0},
```

(continues on next page)

(continued from previous page)

```

↳ 'table alter triggering checkpoint calls': 0, 'table alter unchanged and skipped
↳ ': 0, 'table compact failed calls': 0, 'table compact failed calls due to cache
↳ pressure': 0, 'table compact running': 0, 'table compact skipped as process
↳ would not reduce file size': 0, 'table compact successful calls': 0, 'table
↳ compact timeout': 0, 'table create failed calls': 0, 'table create successful
↳ calls': 1, 'table drop failed calls': 0, 'table drop successful calls': 0,
↳ 'table rename failed calls': 0, 'table rename successful calls': 0, 'table
↳ salvage failed calls': 0, 'table salvage successful calls': 0, 'table truncate
↳ failed calls': 0, 'table truncate successful calls': 0, 'table verify failed
↳ calls': 0, 'table verify successful calls': 0, 'tiered operations dequeued and
↳ processed': 0, 'tiered operations scheduled': 0, 'tiered storage local retention
↳ time (secs)': 0}, 'thread-state': {'active filesystem fsync calls': 0, 'active
↳ filesystem read calls': 0, 'active filesystem write calls': 0}, 'thread-yield': {
↳ 'application thread time evicting (usecs)': 0, 'application thread time waiting
↳ for cache (usecs)': 0, 'connection close blocked waiting for transaction state
↳ stabilization': 0, 'connection close yielded for lsm manager shutdown': 0, 'data
↳ handle lock yielded': 0, 'get reference for page index and slot time sleeping
↳ (usecs)': 0, 'page access yielded due to prepare state change': 0, 'page acquire
↳ busy blocked': 0, 'page acquire eviction blocked': 0, 'page acquire locked
↳ blocked': 0, 'page acquire read blocked': 0, 'page acquire time sleeping (usecs)
↳ ': 0, 'page delete rollback time sleeping for state change (usecs)': 0, 'page
↳ reconciliation yielded due to child modification': 0}, 'transaction': {'Number
↳ of prepared updates': 0, 'Number of prepared updates committed': 0, 'Number of
↳ prepared updates repeated on the same key': 0, 'Number of prepared updates
↳ rolled back': 0, 'oldest pinned transaction ID rolled back for eviction': 0,
↳ 'prepared transactions': 0, 'prepared transactions committed': 0, 'prepared
↳ transactions currently active': 0, 'prepared transactions rolled back': 0,
↳ 'query timestamp calls': 160308, 'race to read prepared update retry': 0,
↳ 'rollback to stable calls': 0, 'rollback to stable history store records with
↳ stop timestamps older than newer records': 0, 'rollback to stable inconsistent
↳ checkpoint': 0, 'rollback to stable keys removed': 0, 'rollback to stable keys
↳ restored': 0, 'rollback to stable pages visited': 0, 'rollback to stable
↳ restored tombstones from history store': 0, 'rollback to stable restored updates
↳ from history store': 0, 'rollback to stable skipping delete rle': 0, 'rollback
↳ to stable skipping stable rle': 0, 'rollback to stable sweeping history store
↳ keys': 0, 'rollback to stable tree walk skipping pages': 0, 'rollback to stable
↳ updates aborted': 0, 'rollback to stable updates removed from history store': 0,
↳ 'sessions scanned in each walk of concurrent sessions': 2844119, 'set timestamp
↳ calls': 0, 'set timestamp durable calls': 0, 'set timestamp durable updates': 0,
↳ 'set timestamp oldest calls': 0, 'set timestamp oldest updates': 0, 'set
↳ timestamp stable calls': 0, 'set timestamp stable updates': 0, 'transaction
↳ begins': 246058, 'transaction checkpoint currently running': 0, 'transaction
↳ checkpoint currently running for history store file': 0, 'transaction checkpoint
↳ generation': 2671, 'transaction checkpoint history store file duration (usecs)': 0
↳ 3, 'transaction checkpoint max time (msecs)': 2748, 'transaction checkpoint min
↳ time (msecs)': 23, 'transaction checkpoint most recent duration for gathering
↳ all handles (usecs)': 107, 'transaction checkpoint most recent duration for
↳ gathering applied handles (usecs)': 29, 'transaction checkpoint most recent
↳ duration for gathering skipped handles (usecs)': 45, 'transaction checkpoint
↳ most recent handles applied': 1, 'transaction checkpoint most recent handles
↳ skipped': 13, 'transaction checkpoint most recent handles walked': 30,
↳ 'transaction checkpoint most recent time (msecs)': 39, 'transaction checkpoint
↳ prepare currently running': 0, 'transaction checkpoint prepare max time (msecs)
↳ ': 442, 'transaction checkpoint prepare min time (msecs)': 0, 'transaction
↳ checkpoint prepare most recent time (msecs)': 0, 'transaction checkpoint prepare
↳ total time (msecs)': 524, 'transaction checkpoint scrub dirty target': 0,

```

(continues on next page)

(continued from previous page)

```
↳ 'transaction checkpoint scrub time (msecs)': 0, 'transaction checkpoint stop'_
↳ 'timing stress active': 0, 'transaction checkpoint total time (msecs)': 123150,
↳ 'transaction checkpoints': 2670, 'transaction checkpoints due to obsolete pages
↳ ': 0, 'transaction checkpoints skipped because database was clean': 0,
↳ 'transaction fsync calls for checkpoint after allocating the transaction ID':_
↳ 2670, 'transaction fsync duration for checkpoint after allocating the_
↳ transaction ID (usecs)': 12751, 'transaction range of IDs currently pinned': 0,
↳ 'transaction range of IDs currently pinned by a checkpoint': 0, 'transaction_
↳ range of timestamps currently pinned': 0, 'transaction range of timestamps pinn
↳ ed by a checkpoint': 0, 'transaction range of timestamps pinned by the_
↳ oldest active read timestamp': 0, 'transaction range of timestamps pinned by the_
↳ oldest timestamp': 0, 'transaction read timestamp of the oldest active reader':_
↳ 0, 'transaction rollback to stable currently running': 0, 'transaction walk of_
↳ concurrent sessions': 182801, 'transactions committed': 84, 'transactions rolled_
↳ back': 245974, 'update conflicts': 0}, 'concurrentTransactions': {'write': {'out
↳ ': 0, 'available': 128, 'totalTickets': 128}, 'read': {'out': 0, 'available':_
↳ 128, 'totalTickets': 128}}, 'snapshot-window-settings': {'total number of_
↳ SnapshotTooOld errors': 0, 'minimum target snapshot window size in seconds': 300,
↳ 'current available snapshot window size in seconds': 0, 'latest majority
↳ snapshot timestamp available': 'Dec 31 19:00:00:0', 'oldest majority snapshot
↳ timestamp available': 'Dec 31 19:00:00:0', 'pinned timestamp requests': 0, 'min
↳ pinned timestamp': Timestamp(4294967295, 4294967295)}, 'oplog': {'visibility
↳ timestamp': Timestamp(0, 0)}}, 'mem': {'bits': 64, 'resident': 1489, 'virtual':_
↳ 2960, 'supported': True}, 'metrics': {'apiVersions': {'': ['default']}},
↳ 'aggStageCounters': {'$_internalApplyOplogUpdate': 0, '$_internalBoundedSort': 0,
↳ '$_internalConvertBucketIndexStats': 0, '$_internalFindAndModifyImageLookup': 0,
↳ '$_internalInhibitOptimization': 0, '$_internalReshardingIterateTransaction': 0,
↳ '$_internalReshardingOwnershipMatch': 0, '$_internalSetWindowFields': 0, '$_
↳ internalSplitPipeline': 0, '$_internalUnpackBucket': 0, '$_unpackBucket': 0,
↳ '$addFields': 0, '$bucket': 0, '$bucketAuto': 0, '$changeStream': 0, '$collStats
↳ ': 0, '$count': 0, '$currentOp': 0, '$documents': 0, '$facet': 0, '$geoNear': 0,
↳ '$graphLookup': 0, '$group': 0, '$indexStats': 0, '$limit': 0, '
↳ '$listLocalSessions': 0, '$listSessions': 0, '$lookup': 0, '$match': 0, '$merge':_
↳ 0, '$mergeCursors': 0, '$operationMetrics': 0, '$out': 0, '$planCacheStats': 0,
↳ '$project': 0, '$queue': 0, '$redact': 0, '$replaceRoot': 0, '$replaceWith': 0,
↳ '$sample': 0, '$set': 20, '$setWindowFields': 0, '$skip': 0, '$sort': 0, '
↳ $sortByCount': 0, '$unionWith': 0, '$unset': 0, '$unwind': 0}, 'changeStreams': {
↳ 'largeEventsFailed': 0}, 'commands': {'<UNKNOWN>': 0, '_addShard': {'failed': 0,
↳ 'total': 0}, '_cloneCollectionOptionsFromPrimaryShard': {'failed': 0, 'total': 0}
↳ , '_configsvrAbortReshardCollection': {'failed': 0, 'total': 0}, '_
↳ configsvrAddShard': {'failed': 0, 'total': 0}, '_configsvrAddShardToZone': {
↳ 'failed': 0, 'total': 0}, '_configsvrBalancerCollectionStatus': {'failed': 0,
↳ 'total': 0}, '_configsvrBalancerStart': {'failed': 0, 'total': 0}, '_
↳ configsvrBalancerStatus': {'failed': 0, 'total': 0}, '_configsvrBalancerStop': {
↳ 'failed': 0, 'total': 0}, '_configsvrCleanupReshardCollection': {'failed': 0,
↳ 'total': 0}, '_configsvrClearJumboFlag': {'failed': 0, 'total': 0}, '_
↳ configsvrCommitChunkMerge': {'failed': 0, 'total': 0}, '_
↳ configsvrCommitChunkMigration': {'failed': 0, 'total': 0}, '_
↳ configsvrCommitChunkSplit': {'failed': 0, 'total': 0}, '_
↳ configsvrCommitChunksMerge': {'failed': 0, 'total': 0}, '_
↳ configsvrCommitMovePrimary': {'failed': 0, 'total': 0}, '_
↳ configsvrCommitReshardCollection': {'failed': 0, 'total': 0}, '_
↳ configsvrCreateDatabase': {'failed': 0, 'total': 0}, '_configsvrDropCollection': {
↳ 'failed': 0, 'total': 0}, '_configsvrDropDatabase': {'failed': 0, 'total': 0},
↳ '_configsvrEnableSharding': {'failed': 0, 'total': 0}, '_
↳ configsvrEnsureChunkVersionIsGreaterThan': {'failed': 0, 'total': 0}, '_
```

(continues on next page)

(continued from previous page)

```

↳ configsrvMoveChunk': {'failed': 0, 'total': 0}, '_configsvrMovePrimary': {'failed': 0, 'total': 0}, '_configsvrRefineCollectionShardKey': {'failed': 0, 'total': 0}, '_configsvrRemoveChunks': {'failed': 0, 'total': 0}, '_configsvrRemoveShard': {'failed': 0, 'total': 0}, '_configsvrRemoveShardFromZone': {'failed': 0, 'total': 0}, '_configsvrRemoveTags': {'failed': 0, 'total': 0}, '_configsvrRenameCollectionMetadata': {'failed': 0, 'total': 0}, '_configsvrRepairShardedCollectionChunksHistory': {'failed': 0, 'total': 0}, '_configsvrReshardCollection': {'failed': 0, 'total': 0}, '_configsvrSetAllowMigrations': {'failed': 0, 'total': 0}, '_configsvrShardCollection': {'failed': 0, 'total': 0}, '_configsvrUpdateZoneKeyRange': {'failed': 0, 'total': 0}, '_flushDatabaseCacheUpdates': {'failed': 0, 'total': 0}, '_flushDatabaseCacheUpdatesWithWriteConcern': {'failed': 0, 'total': 0}, '_flushReshardingStateChange': {'failed': 0, 'total': 0}, '_flushRoutingTableCacheUpdates': {'failed': 0, 'total': 0}, '_flushRoutingTableCacheUpdatesWithWriteConcern': {'failed': 0, 'total': 0}, '_getNextSessionMods': {'failed': 0, 'total': 0}, '_getUserCacheGeneration': {'failed': 0, 'total': 0}, '_isSelf': {'failed': 0, 'total': 0}, '_killOperations': {'failed': 0, 'total': 0}, '_mergeAuthzCollections': {'failed': 0, 'total': 0}, '_migrateClone': {'failed': 0, 'total': 0}, '_recvChunkAbort': {'failed': 0, 'total': 0}, '_recvChunkCommit': {'failed': 0, 'total': 0}, '_recvChunkStart': {'failed': 0, 'total': 0}, '_recvChunkStatus': {'failed': 0, 'total': 0}, '_shardsvrAbortReshardCollection': {'failed': 0, 'total': 0}, '_shardsvrCleanupReshardCollection': {'failed': 0, 'total': 0}, '_shardsvrCloneCatalogData': {'failed': 0, 'total': 0}, '_shardsvrCommitReshardCollection': {'failed': 0, 'total': 0}, '_shardsvrCreateCollection': {'failed': 0, 'total': 0}, '_shardsvrCreateCollectionParticipant': {'failed': 0, 'total': 0}, '_shardsvrDropCollection': {'failed': 0, 'total': 0}, '_shardsvrDropCollectionIfUUIDNotMatching': {'failed': 0, 'total': 0}, '_shardsvrDropCollectionParticipant': {'failed': 0, 'total': 0}, '_shardsvrDropDatabase': {'failed': 0, 'total': 0}, '_shardsvrDropDatabaseParticipant': {'failed': 0, 'total': 0}, '_shardsvrMovePrimary': {'failed': 0, 'total': 0}, '_shardsvrRefineCollectionShardKey': {'failed': 0, 'total': 0}, '_shardsvrRenameCollection': {'failed': 0, 'total': 0}, '_shardsvrRenameCollectionParticipant': {'failed': 0, 'total': 0}, '_shardsvrRenameCollectionParticipantUnblock': {'failed': 0, 'total': 0}, '_shardsvrReshardCollection': {'failed': 0, 'total': 0}, '_shardsvrReshardingOperationTime': {'failed': 0, 'total': 0}, '_shardsvrSetAllowMigrations': {'failed': 0, 'total': 0}, '_shardsvrShardCollection': {'failed': 0, 'total': 0}, '_transferMods': {'failed': 0, 'total': 0}, '_abortTransaction': {'failed': 0, 'total': 0}, 'aggregate': {'allowDiskUseTrue': 0, 'failed': 0, 'total': 0}, '_appendOplogNote': {'failed': 0, 'total': 0}, '_applyOps': {'failed': 0, 'total': 0}, '_authenticate': {'failed': 0, 'total': 0}, '_autoSplitVector': {'failed': 0, 'total': 0}, '_availableQueryOptions': {'failed': 0, 'total': 0}, '_buildInfo': {'failed': 0, 'total': 0}, '_checkShardingIndex': {'failed': 0, 'total': 0}, '_cleanupOrphaned': {'failed': 0, 'total': 0}, '_cloneCollectionAsCapped': {'failed': 0, 'total': 0}, '_collMod': {'failed': 0, 'total': 0}, '_collStats': {'failed': 0, 'total': 0}, '_commitTransaction': {'failed': 0, 'total': 0}, '_compact': {'failed': 0, 'total': 0}, '_connPoolStats': {'failed': 0, 'total': 0}, '_connPoolSync': {'failed': 0, 'total': 0}, '_connectionStatus': {'failed': 0, 'total': 0}, '_convertToCapped': {'failed': 0, 'total': 0}, '_coordinateCommitTransaction': {'failed': 0, 'total': 0}, '_count': {'failed': 0, 'total': 0}, '_create': {'failed': 0, 'total': 0}, '_validator': {'failed': 0, 'total': 0}

```

(continues on next page)

(continued from previous page)

```
↳ 'jsonSchema': 0, 'total': 0}}, 'createIndexes': {'failed': 0, 'total': 0},
↳ 'createRole': {'failed': 0, 'total': 0}, 'createUser': {'failed': 0, 'total': 0},
↳ 'currentOp': {'failed': 0, 'total': 0}, 'dataSize': {'failed': 0, 'total': 0},
↳ 'dbCheck': {'failed': 0, 'total': 0}, 'dbHash': {'failed': 0, 'total': 0},
↳ 'dbStats': {'failed': 0, 'total': 0}, 'delete': {'failed': 0, 'total': 0},
↳ 'distinct': {'failed': 0, 'total': 2}, 'donorAbortMigration': {'failed': 0,
↳ 'total': 0}, 'donorForgetMigration': {'failed': 0, 'total': 0},
↳ 'donorStartMigration': {'failed': 0, 'total': 0}, 'driverOIDTest': {'failed': 0,
↳ 'total': 0}, 'drop': {'failed': 0, 'total': 0}, 'dropAllRolesFromDatabase': {
↳ 'failed': 0, 'total': 0}, 'dropAllUsersFromDatabase': {'failed': 0, 'total': 0},
↳ 'dropConnections': {'failed': 0, 'total': 0}, 'dropDatabase': {'failed': 0,
↳ 'total': 0}, 'dropIndexes': {'failed': 0, 'total': 0}, 'dropRole': {'failed': 0,
↳ 'total': 0}, 'dropUser': {'failed': 0, 'total': 0}, 'endSessions': {'failed': 0,
↳ 'total': 0}, 'explain': {'failed': 0, 'total': 0}, 'features': {'failed': 0,
↳ 'total': 0}, 'filemd5': {'failed': 0, 'total': 0}, 'find': {'failed': 0, 'total
↳ ': 669}, 'findAndModify': {'arrayFilters': 0, 'failed': 0, 'pipeline': 0, 'total
↳ ': 0}, 'flushRouterConfig': {'failed': 0, 'total': 0}, 'fsync': {'failed': 0,
↳ 'total': 0}, 'fsyncUnlock': {'failed': 0, 'total': 0}, 'getCmdLineOpts': {'failed
↳ ': 0, 'total': 0}, 'getDatabaseVersion': {'failed': 0, 'total': 0},
↳ 'getDefaultRWConcern': {'failed': 0, 'total': 0}, 'getDiagnosticData': {'failed
↳ ': 0, 'total': 0}, 'getFreeMonitoringStatus': {'failed': 0, 'total': 0},
↳ 'getLastError': {'failed': 0, 'total': 0}, 'getLog': {'failed': 0, 'total': 0},
↳ 'getMore': {'failed': 0, 'total': 364}, 'getParameter': {'failed': 0, 'total': 0}
↳ , 'getShardMap': {'failed': 0, 'total': 0}, 'getShardVersion': {'failed': 0,
↳ 'total': 0}, 'getnonce': {'failed': 0, 'total': 0}, 'grantPrivilegesToRole': {
↳ 'failed': 0, 'total': 0}, 'grantRolesToRole': {'failed': 0, 'total': 0},
↳ 'grantRolesToUser': {'failed': 0, 'total': 0}, 'hello': {'failed': 21, 'total': 0
↳ : 16596}, 'hostInfo': {'failed': 0, 'total': 0}, 'insert': {'failed': 0, 'total': 0
↳ : 6}, 'internalRenameIfOptionsAndIndexesMatch': {'failed': 0, 'total': 0},
↳ 'invalidateUserCache': {'failed': 0, 'total': 0}, 'isMaster': {'failed': 0,
↳ 'total': 5581}, 'killAllSessions': {'failed': 0, 'total': 0},
↳ 'killAllSessionsByPattern': {'failed': 0, 'total': 0}, 'killCursors': {'failed': 0
↳ , 'total': 0}, 'killOp': {'failed': 0, 'total': 0}, 'killSessions': {'failed': 0
↳ , 'total': 0}, 'listCollections': {'failed': 0, 'total': 0}, 'listCommands': {
↳ 'failed': 0, 'total': 0}, 'listDatabases': {'failed': 0, 'total': 0},
↳ 'listIndexes': {'failed': 0, 'total': 1070}, 'lockInfo': {'failed': 0, 'total': 0
↳ : 0}, 'logRotate': {'failed': 0, 'total': 0}, 'logout': {'failed': 0, 'total': 0},
↳ 'mapReduce': {'failed': 0, 'total': 0}, 'mergeChunks': {'failed': 0, 'total': 0},
↳ 'moveChunk': {'failed': 0, 'total': 0}, 'ping': {'failed': 0, 'total': 0},
↳ 'planCacheClear': {'failed': 0, 'total': 0}, 'planCacheClearFilters': {'failed': 0
↳ , 'total': 0}, 'planCacheListFilters': {'failed': 0, 'total': 0},
↳ 'planCacheSetFilter': {'failed': 0, 'total': 0}, 'prepareTransaction': {'failed
↳ ': 0, 'total': 0}, 'profile': {'failed': 0, 'total': 0}, 'reIndex': {'failed': 0,
↳ 'total': 0}, 'recipientForgetMigration': {'failed': 0, 'total': 0},
↳ 'recipientSyncData': {'failed': 0, 'total': 0}, 'refreshSessions': {'failed': 0,
↳ 'total': 0}, 'renameCollection': {'failed': 0, 'total': 0},
↳ 'replSetAbortPrimaryCatchUp': {'failed': 0, 'total': 0}, 'replSetFreeze': {
↳ 'failed': 0, 'total': 0}, 'replSetGetConfig': {'failed': 0, 'total': 0},
↳ 'replSetGetRBID': {'failed': 0, 'total': 0}, 'replSetGetStatus': {'failed': 0,
↳ 'total': 0}, 'replSetHeartbeat': {'failed': 0, 'total': 0}, 'replSetInitiate': {
↳ 'failed': 0, 'total': 0}, 'replSetMaintenance': {'failed': 0, 'total': 0},
↳ 'replSetReconfig': {'failed': 0, 'total': 0}, 'replSetRequestVotes': {'failed': 0
↳ , 'total': 0}, 'replSetResizeOplog': {'failed': 0, 'total': 0}, 'replSetStepDown
↳ : {'failed': 0, 'total': 0}, 'replSetStepDownWithForce': {'failed': 0, 'total': 0
↳ : 0}, 'replSetStepUp': {'failed': 0, 'total': 0}, 'replSetSyncFrom': {'failed': 0,
↳ 'total': 0}, 'replSetUpdatePosition': {'failed': 0, 'total': 0},
```

(continues on next page)

(continued from previous page)

```

↳ 'revokePrivilegesFromRole': {'failed': 0, 'total': 0}, 'revokeRolesFromRole': {
↳ 'failed': 0, 'total': 0}, 'revokeRolesFromUser': {'failed': 0, 'total': 0},
↳ 'rolesInfo': {'failed': 0, 'total': 0}, 'rotateCertificates': {'failed': 0,
↳ 'total': 0}, 'saslContinue': {'failed': 0, 'total': 0}, 'saslStart': {'failed': 0,
↳ 'total': 0}, 'serverStatus': {'failed': 0, 'total': 15}, 'setDefaultRWConcern
↳ ': {'failed': 0, 'total': 0}, 'setFeatureCompatibilityVersion': {'failed': 0,
↳ 'total': 0}, 'setFreeMonitoring': {'failed': 0, 'total': 0},
↳ 'setIndexCommitQuorum': {'failed': 0, 'total': 0}, 'setParameter': {'failed': 0,
↳ 'total': 0}, 'setProfilingFilterGlobally': {'failed': 0, 'total': 0},
↳ 'setShardVersion': {'failed': 0, 'total': 0}, 'shardingState': {'failed': 0,
↳ 'total': 0}, 'shutdown': {'failed': 0, 'total': 0}, 'splitChunk': {'failed': 0,
↳ 'total': 0}, 'splitVector': {'failed': 0, 'total': 0}, 'startRecordingTraffic': {
↳ 'failed': 0, 'total': 0}, 'startSession': {'failed': 0, 'total': 0},
↳ 'stopRecordingTraffic': {'failed': 0, 'total': 0}, 'top': {'failed': 0, 'total': 0},
↳ 'update': {'arrayFilters': 0, 'failed': 0, 'pipeline': 20, 'total': 19},
↳ 'updateRole': {'failed': 0, 'total': 0}, 'updateUser': {'failed': 0, 'total': 0},
↳ 'usersInfo': {'failed': 0, 'total': 0}, 'validate': {'failed': 0, 'total': 0},
↳ 'validateDBMetadata': {'failed': 0, 'total': 0}, 'voteCommitIndexBuild': {'failed
↳ ': 0, 'total': 0}, 'waitForFailPoint': {'failed': 0, 'total': 0}, 'whatsmyuri': {
↳ 'failed': 0, 'total': 0}}, 'cursor': {'moreThanOneBatch': 120, 'timedOut': 0,
↳ 'totalOpened': 120, 'lifespan': {'greaterThanOrEqual10Minutes': 1,
↳ 'lessThan10Minutes': 0, 'lessThan15Seconds': 0, 'lessThan1Minute': 0,
↳ 'lessThan1Second': 80, 'lessThan30Seconds': 0, 'lessThan5Seconds': 39}, 'open': {
↳ 'noTimeout': 0, 'pinned': 0, 'total': 0}}, 'document': {'deleted': 0, 'inserted
↳ ': 6, 'returned': 6030383, 'updated': 2}, 'dotsAndDollarsFields': {'inserts': 0,
↳ 'updates': 0}, 'getLastError': {'wtime': {'num': 0, 'totalMillis': 0}, 'wtimeouts
↳ ': 0, 'default': {'unsatisfiable': 0, 'wtimeouts': 0}}, 'mongos': {'cursor': {
↳ 'moreThanOneBatch': 0, 'totalOpened': 0}}, 'operation': {'scanAndOrder': 0,
↳ 'transactionTooLargeForCacheErrors': 0,
↳ 'transactionTooLargeForCacheErrorsConvertedToWriteConflict': 0, 'writeConflicts
↳ ': 0}, 'operatorCounters': {'expressions': {'$_internalJsEmit': 0, '$abs': 0, '$
↳ acos': 0, '$acosh': 0, '$add': 0, '$allElementsTrue': 0, '$and': 0, '$anyElementTrue
↳ ': 0, '$arrayElemAt': 0, '$arrayToObject': 0, '$asin': 0, '$asinh': 0, '$atan': 0,
↳ '$atan2': 0, '$atanh': 0, '$avg': 0, '$binarySize': 0, '$bsonSize': 0, '$ceil': 0,
↳ '$cmp': 0, '$concat': 0, '$concatArrays': 0, '$cond': 0, '$const': 0, '$convert': 0,
↳ '$cos': 0, '$cosh': 0, '$dateAdd': 0, '$dateDiff': 0, '$dateFromParts': 0, '$dateFrom
↳ String': 0, '$dateSubtract': 0, '$dateToString': 0, '$dateTrunc': 0, '$dayOfMonth': 0,
↳ '$dayOfWeek': 0, '$dayOfYear': 0, '$degreesToRadians': 0, '$divide': 0, '$eq': 0,
↳ '$exp': 0, '$filter': 0, '$first': 0, '$floor': 0, '$function': 0, '$getField': 0,
↳ '$gt': 0, '$gte': 0, '$hour': 0, '$ifNull': 0, '$in': 0, '$indexOfArray': 0, '$index
↳ OfBytes': 0, '$indexOfCP': 0, '$isArray': 0, '$isNumber': 0, '$isoDayOfWeek': 0,
↳ '$isoWeek': 0, '$isoWeekYear': 0, '$last': 0, '$let': 0, '$literal': 0, '$ln': 0,
↳ '$log': 0, '$log10': 0, '$lt': 0, '$lte': 0, '$ltrim': 0, '$map': 0, '$max': 0,
↳ '$mergeObjects': 0, '$meta': 0, '$millisecond': 0, '$min': 0, '$minute': 0, '$mod': 0,
↳ '$month': 0, '$multiply': 0, '$ne': 0, '$not': 0, '$objectToArray': 0, '$or': 0,
↳ '$pow': 0, '$radiansToDegrees': 0, '$rand': 0, '$range': 0, '$reduce': 0, '$regexFind': 0,
↳ '$regexFindAll': 0, '$regexMatch': 0, '$replaceAll': 0, '$replaceOne': 0, '$reverseArray': 0,
↳ '$round': 0, '$rtrim': 0, '$second': 0, '$setDifference': 0, '$setEquals': 0, '$setField': 0,
↳ '$setIntersection': 0, '$setIsSubset': 0, '$setUnion': 0, '$sin': 0, '$sinh': 0, '$size': 0,
↳ '$slice': 0, '$split': 0, '$sqrt': 0, '$stdDevPop': 0, '$stdDevSamp': 0, '$strLenBytes': 0,
↳ '$strLenCP': 0, '$strcasecmp': 0, '$substr': 0, '$substrCP': 0, '$subtract': 0, '$sum': 0,
↳ '$switch': 0, '$tanh': 0, '$toBool': 0, '$toDate': 0, '$toDecimal': 0, '$toDouble': 0,
↳ '$toHashedIndexKey': 0, '$toInt': 0, '$toLong': 0, '$toLower': 0, '$toObjectId': 0

```

(continues on next page)

(continued from previous page)

```

        '$toString': 0, '$toUpper': 0, '$trim': 0, '$trunc': 0, '$type': 0, '$unsetField': 0, '$week': 0, '$year': 0, '$zip': 0}, 'groupAccumulators': {'$_internalJsReduce': 0, '$accumulator': 0, '$addToSet': 0, '$avg': 0, '$count': 0, '$first': 0, '$last': 0, '$max': 0, '$mergeObjects': 0, '$min': 0, '$push': 0, '$stdDevPop': 0, '$stdDevSamp': 0, '$sum': 0}, 'match': {'$all': 0, '$alwaysFalse': 0, '$alwaysTrue': 0, '$and': 0, '$bitsAllClear': 0, '$bitsAllSet': 0, '$bitsAnyClear': 0, '$bitsAnySet': 0, '$comment': 0, '$elemMatch': 0, '$eq': 118, '$exists': 6, '$expr': 0, '$geoIntersects': 0, '$geoWithin': 0, '$gt': 0, '$gte': 0, '$in': 8, '$jsonSchema': 0, '$lt': 535, '$lte': 0, '$mod': 0, '$ne': 0, '$near': 0, '$nearSphere': 0, '$nin': 0, '$nor': 0, '$not': 0, '$or': 0, '$regex': 0, '$sampleRate': 0, '$size': 0, '$text': 0, '$type': 0, '$where': 0}, 'windowAccumulators': {'$addToSet': 0, '$avg': 0, '$count': 0, '$covariancePop': 0, '$covarianceSamp': 0, '$denseRank': 0, '$derivative': 0, '$documentNumber': 0, '$expMovingAvg': 0, '$first': 0, '$integral': 0, '$last': 0, '$max': 0, '$min': 0, '$push': 0, '$rank': 0, '$shift': 0, '$stdDevPop': 0, '$stdDevSamp': 0, '$sum': 0}, 'query': {'deleteManyCount': 0, 'planCacheTotalSizeEstimateBytes': 0, 'updateDeleteManyDocumentsMaxCount': 0, 'updateDeleteManyDocumentsTotalCount': 0, 'updateDeleteManyDurationMaxMs': 0, 'updateDeleteManyDurationTotalMs': 0, 'updateManyCount': 0, 'updateOneOpStyleBroadcastWithExactIDCount': 0, 'multiPlanner': {'classicCount': 0, 'classicMicros': 0, 'classicWorks': 0, 'sbeCount': 0, 'sbeMicros': 0, 'sbeNumReads': 0, 'histograms': {'classicMicros': [{['lowerBound': 0, 'count': 0}, {'lowerBound': 1024, 'count': 0}, {'lowerBound': 4096, 'count': 0}, {'lowerBound': 16384, 'count': 0}, {'lowerBound': 65536, 'count': 0}, {'lowerBound': 262144, 'count': 0}, {'lowerBound': 1048576, 'count': 0}, {'lowerBound': 4194304, 'count': 0}, {'lowerBound': 16777216, 'count': 0}, {'lowerBound': 67108864, 'count': 0}, {'lowerBound': 268435456, 'count': 0}, {'lowerBound': 1073741824, 'count': 0}], 'classicNumPlans': [{['lowerBound': 0, 'count': 0}, {'lowerBound': 2, 'count': 0}, {'lowerBound': 4, 'count': 0}, {'lowerBound': 8, 'count': 0}, {'lowerBound': 16, 'count': 0}, {'lowerBound': 32, 'count': 0}, {'lowerBound': 64, 'count': 0}, {'lowerBound': 128, 'count': 0}, {'lowerBound': 256, 'count': 0}, {'lowerBound': 512, 'count': 0}, {'lowerBound': 1024, 'count': 0}, {'lowerBound': 2048, 'count': 0}, {'lowerBound': 4096, 'count': 0}, {'lowerBound': 8192, 'count': 0}, {'lowerBound': 16384, 'count': 0}, {'lowerBound': 32768, 'count': 0}], 'sbeMicros': [{['lowerBound': 0, 'count': 0}, {'lowerBound': 4096, 'count': 0}, {'lowerBound': 16384, 'count': 0}, {'lowerBound': 65536, 'count': 0}, {'lowerBound': 262144, 'count': 0}, {'lowerBound': 1048576, 'count': 0}, {'lowerBound': 4194304, 'count': 0}, {'lowerBound': 16777216, 'count': 0}, {'lowerBound': 67108864, 'count': 0}, {'lowerBound': 268435456, 'count': 0}, {'lowerBound': 1073741824, 'count': 0}], 'sbeNumPlans': [{['lowerBound': 0, 'count': 0}, {'lowerBound': 2, 'count': 0}, {'lowerBound': 4, 'count': 0}, {'lowerBound': 8, 'count': 0}, {'lowerBound': 16, 'count': 0}, {'lowerBound': 32, 'count': 0}, {'lowerBound': 64, 'count': 0}, {'lowerBound': 128, 'count': 0}, {'lowerBound': 256, 'count': 0}, {'lowerBound': 512, 'count': 0}, {'lowerBound': 1024, 'count': 0}, {'lowerBound': 2048, 'count': 0}, {'lowerBound': 4096, 'count': 0}, {'lowerBound': 8192, 'count': 0}, {'lowerBound': 16384, 'count': 0}, {'lowerBound': 32768, 'count': 0}], 'sbeNumReads': [{['lowerBound': 0, 'count': 0}, {'lowerBound': 128, 'count': 0}, {'lowerBound': 256, 'count': 0}, {'lowerBound': 512, 'count': 0}, {'lowerBound': 1024, 'count': 0}, {'lowerBound': 2048, 'count': 0}, {'lowerBound': 4096, 'count': 0}, {'lowerBound': 8192, 'count': 0}, {'lowerBound': 16384, 'count': 0}, {'lowerBound': 32768, 'count': 0}], 'queryExecutor': {'scanned': 10, 'scannedObjects': 241944000, 'collectionScans': {'nonTailable': 128, 'total': 128}}, 'record': {'moves': 0, 'repl': {'executor': {'pool': {'inProgressCount': 0}, 'queues': {'networkInProgress': 0, 'sleepers': 0}, 'unsignaledEvents': 0}, 'shuttingDown': False, 'networkInterface': 'DEPRECATED: getDiagnosticString is deprecated in NetworkInterfaceTL'}, 'apply': {'attemptsToBecomeSecondary': 0, 'batchSize': 0, 'batches': {'num': 0, 'totalMillis': 0}, 'ops': 0}, 'buffer': {'count': 0, 'maxSizeBytes': 0, 'sizeBytes': 0}, 'initialSync': {'completed': 0, 'failedAttempts': 0, 'failures': 0}, 'network': {'bytes': 0, 'getmores': {'num': 0, 'totalMillis': 0}, 'numEmptyBatches': 0}, 'notPrimaryLegacyUnacknowledgedWrites': 0}}

```

(continues on next page)

(continued from previous page)

```

': 0, 'notPrimaryUnacknowledgedWrites': 0, 'oplogGetMoresProcessed': {'num': 0,
'totalMillis': 0}, 'ops': 0, 'readersCreated': 0, 'replSetUpdatePosition': {'num':
0}}, 'reconfig': {'numAutoReconfigsForRemovalOfNewlyAddedFields': 0},
'stateTransition': {'lastStateTransition': '', 'userOperationsKilled': 0,
'userOperationsRunning': 0}, 'syncSource': {'numSelections': 0,
'numSyncSourceChangesDueToSignificantlyCloserNode': 0, 'numTimesChoseDifferent': 0,
'numTimesChoseSame': 0, 'numTimesCouldNotFind': 0}}, 'ttl': {'deletedDocuments': 18,
'passes': 2671}}, 'ok': 1.0}

```

```

## Retrieve headline+situation text
events = [28, 16, 83, 41, 81, 23, 87, 45, 80, 97, 231, 46, 31, 77, 29,
232, 101, 42, 47, 86, 93, 3, 22, 102, 82]

```

```

initial = False
if initial:
    lines = []
    event_all = []
    tokenizer = RegexpTokenizer(r"\b[^\d\W][^\d\W][^\d\W]+\b")
    for event in events:
        docs = keydev['events']\n            .find({'keydeveventtypeid': {'$eq': event}}, {'_id': 0})
        doc = [tokenizer.tokenize((d['headline'] + " " + d['situation']))\n            .lower()]
        for d in docs]
        lines.extend(doc)
        event_all.extend([event] * len(doc))
    with gzip.open(memdir / 'lines.json.gz', 'wt') as f:
        json.dump(lines, f)
    with gzip.open(memdir / 'event_all.json.gz', 'wt') as f:
        json.dump(event_all, f)
    print(lines[1000000])
else:
    with gzip.open(memdir / 'lines.json.gz', 'rt') as f:
        lines = json.load(f)
    with gzip.open(memdir / 'event_all.json.gz', 'rt') as f:
        event_all = json.load(f)

```

```

## Encode class labels
from sklearn.preprocessing import LabelEncoder
events_encoder = LabelEncoder().fit(event_all)      # .inverse_transform()
num_classes = len(np.unique(event_all))
y_all = events_encoder.transform(event_all)
Series(event_all).value_counts()\
    .rename(index=events_)\
    .rename('count')\
    .to_frame()

```

	count
Announcements of Earnings	244140
Executive/Board Changes - Other	128019
Private Placements	113007
Product-Related Announcements	93172
M&A Transaction Closings	90955

(continues on next page)

(continued from previous page)

Client Announcements	90687
Fixed Income Offerings	69652
Dividend Affirmations	53704
M&A Transaction Announcements	43305
Special/Extraordinary Shareholders Meeting	32572
Buyback Tranche Update	30980
Dividend Increases	27670
Business Expansions	26511
Changes in Company Bylaws/Rules	22355
Corporate Guidance - New/Confirmed	19431
Buyback Transaction Announcements	17420
Executive Changes - CEO	16291
Debt Financing Related	15184
Dividend Decreases	15161
Follow-on Equity Offerings	14687
Shelf Registration Filings	12793
Seeking Acquisitions/Investments	12736
Strategic Alliances	12045
Executive Changes - CFO	11698
M&A Transaction Cancellations	10076

```
## Split into stratified train and test indices
from sklearn.model_selection import train_test_split
train_idx, test_idx = train_test_split(np.arange(len(y_all)),
                                       random_state=42,
                                       stratify=y_all,
                                       test_size=0.2)
```

```
## Load spacy vocab
import spacy
lang = 'en_core_web_lg' # python -m spacy download en_core_web_lg
nlp = spacy.load(lang, disable=['parser', 'tagger', 'ner', 'lemmatizer'])
for w in ['inr', 'yen', 'jpy', 'eur', 'dkk', 'cny', 'sfr']:
    nlp.vocab[w].is_stop = True # Mark customized stop words
n_vocab, vocab_dim = nlp.vocab.vectors.shape
print('Language:', lang, ' vocab:', n_vocab, ' dim:', vocab_dim)
```

Language: en\_core\_web\_lg vocab: 514157 dim: 300

```
## Precompute word embeddings input
def form_input(line, nlp):
    """Return spacy average vector from valid words"""
    tokens = [tok.vector for tok in nlp(" ".join(line))
              if not(tok.is_stop
                     or tok.is_punct
                     or tok.is_oov
                     or tok.is_space)]
    if len(tokens):
        return np.array(tokens).mean(axis=0)
    else:
        return np.zeros(nlp.vocab.vectors.shape[1])
```

```

args = {'dtype': 'float32'}
if initial:
    args.update({'shape': (len(lines), vocab_dim), 'mode': 'w+'})
    X = np.memmap(memdir / "X.{}_{}".format(*args['shape'])),
        **args)
    for i, line in tqdm(enumerate(lines)):
        X[i] = form_input(line, nlp).astype(args['dtype'])
else:
    args.update({'shape': (1224251, vocab_dim), 'mode': 'r'})
    X = np.memmap(memdir / "X.{}_{}".format(*args['shape'])), **args)

```

```

## Pytorch Feed Forward Network
class FFNN(nn.Module):
    """Deep Averaging Network for classification"""
    def __init__(self, vocab_dim, num_classes, hidden, dropout=0.3):
        super().__init__()
        V = nn.Linear(vocab_dim, hidden[0])
        nn.init.xavier_uniform_(V.weight)
        L = [V, nn.Dropout(dropout)]
        for g, h in zip(hidden, hidden[1:] + [num_classes]):
            W = nn.Linear(g, h)
            nn.init.xavier_uniform_(W.weight)
            L.extend([nn.ReLU(), W])
        self.network = nn.Sequential(*L)
        self.classifier = nn.LogSoftmax(dim=-1) # output is (N, C) logits

    def forward(self, x):
        """Return tensor of log probabilities"""
        return self.classifier(self.network(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

```

```

## Training Loops
accuracy = {} # to store computed metrics
max_layers, hidden = 1, 300 #3, 300
batch_sz, lr, num_lr, step_sz, eval_skip = 64, 0.01, 4, 10, 5 #3, 3, 3 #
num_epochs = step_sz * num_lr + 1
for layers in [max_layers]:

    # Instantiate model, optimizer, scheduler, loss_function
    model = FFNN(vocab_dim=vocab_dim,
                  num_classes=num_classes,
                  hidden=[hidden]*layers) \
                  .to(device)
    optimizer = torch.optim.Adam(model.parameters(), lr=lr)

```

(continues on next page)

(continued from previous page)

```

scheduler = torch.optim.lr_scheduler.StepLR(optimizer,
                                             gamma=0.1,
                                             step_size=step_sz)
loss_function = nn.NLLLoss()
accuracy[layers] = {}

# Loop over epochs and batches
for epoch in range(0, num_epochs):
    tic = time.time()
    idxs = [i for i in train_idx]
    random.shuffle(idxs)
    batches = [idxs[i:(i+batch_sz)] for i in range(0, len(idxs), batch_sz)]

    total_loss = 0.0
    model.train()
    for i, batch in enumerate(batches):      # loop over minibatches
        x = torch.FloatTensor(np.array(X[batch])) \
            .to(device)
        y = torch.tensor(np.array([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
        print(i, len(batches), i/len(batches), total_loss, end='\r')
    scheduler.step()                      # scheduler step
    model.eval()
    print(f"Loss on epoch {epoch}: {total_loss:.1f}")
    #model.save(imgdir / f"dan{layers}.pt")

    with torch.no_grad():
        if epoch % eval_skip == 0:
            gold = np.asarray([int(y) for y in y_all])
            batches = [test_idx[i:(i+128)] \
                       for i in range(0, len(test_idx), 128)]
            test_gold, test_pred = [], []
            for batch in tqdm(batches):
                test_pred.extend(model.predict(
                    torch.FloatTensor(np.array(X[batch])).to(device)))
                test_gold.extend(gold[batch])
            test_correct = (np.asarray(test_pred) ==
                           np.asarray(test_gold)).sum()

            batches = [train_idx[i:(i+128)] \
                       for i in range(0, len(train_idx), 128)]
            train_gold, train_pred = [], []
            for batch in tqdm(batches):
                train_pred.extend(model.predict(
                    torch.FloatTensor(np.array(X[batch])).to(device)))
                train_gold.extend(gold[batch])
            train_correct = (np.asarray(train_pred) ==
                           np.asarray(train_gold)).sum()

            accuracy[layers][epoch] = {
                'loss': total_loss,

```

(continues on next page)

(continued from previous page)

```

        'train': train_correct/len(train_idx),
        'test': test_correct/len(test_idx)}
    print(layers, epoch, int(time.time()-tic),
          optimizer.param_groups[0]['lr'],
          train_correct/len(train_idx), test_correct/len(test_idx))

```

Loss on epoch 0: 5324.06058546 5324.0379179120065

100%|██████████| 1913/1913 [00:00<00:00, 4894.66it/s]
100%|██████████| 7652/7652 [00:01<00:00, 4865.88it/s]

1 0 16 0.01 0.8959638554216868 0.8951035527729109  
Loss on epoch 1: 4997.96058546 4997.9266692250975  
Loss on epoch 2: 4960.26058546 4960.1847788915045  
Loss on epoch 3: 4951.46058546 4951.4285500943665  
Loss on epoch 4: 4944.66058546 4944.63189046084956  
Loss on epoch 5: 4945.76058546 4945.65234553441456

100%|██████████| 1913/1913 [00:00<00:00, 4871.00it/s]
100%|██████████| 7652/7652 [00:01<00:00, 4897.74it/s]

1 5 12 0.01 0.9031927710843374 0.9010500263425512  
Loss on epoch 6: 4958.36058546 4958.3483284115795  
Loss on epoch 7: 5004.76058546 5004.72262148559155  
Loss on epoch 8: 4991.26058546 4991.21699206531055  
Loss on epoch 9: 4975.76058546 4975.74128325283554  
Loss on epoch 10: 4215.3058546 4215.25833353400255

100%|██████████| 1913/1913 [00:00<00:00, 4794.91it/s]
100%|██████████| 7652/7652 [00:01<00:00, 4860.43it/s]

1 10 12 0.001 0.9113549111701041 0.9086587353124962  
Loss on epoch 11: 4087.2058546 4087.21495052054523  
Loss on epoch 12: 4041.6058546 4041.59882415458565  
Loss on epoch 13: 4007.9058546 4007.90326675027643  
Loss on epoch 14: 3983.9058546 3983.9181403480476  
Loss on epoch 15: 3965.2058546 3965.20702140033255

100%|██████████| 1913/1913 [00:00<00:00, 4909.41it/s]
100%|██████████| 7652/7652 [00:01<00:00, 4981.19it/s]

1 15 12 0.001 0.9121799060649377 0.9090181375612107  
Loss on epoch 16: 3943.9058546 3943.90357608720666  
Loss on epoch 17: 3942.5058546 3942.53987898305065  
Loss on epoch 18: 3924.0058546 3924.02518970891837  
Loss on epoch 19: 3909.5058546 3909.48357108398347  
Loss on epoch 20: 3840.3058546 3840.28533599725056

```
100% [██████████] | 1913/1913 [00:00<00:00, 4811.84it/s]
100% [██████████] | 7652/7652 [00:01<00:00, 4891.78it/s]
```

```
1 20 13 0.0001 0.9146017970185828 0.9111541304711845
Loss on epoch 21: 3833.8058546 3833.81362865865237
Loss on epoch 22: 3826.3058546 3826.30845699086876
Loss on epoch 23: 3820.9058546 3820.91938536241655
Loss on epoch 24: 3809.1058546 3809.09616566449445
Loss on epoch 25: 3819.5058546 3819.54453152790673
```

```
100% [██████████] | 1913/1913 [00:00<00:00, 4734.76it/s]
100% [██████████] | 7652/7652 [00:01<00:00, 4923.37it/s]
```

```
1 25 12 0.0001 0.9148284664080049 0.9113910092260191
Loss on epoch 26: 3814.8058546 3814.83310318738238
Loss on epoch 27: 3814.0058546 3813.97460565343556
Loss on epoch 28: 3812.3058546 3812.34203559532765
Loss on epoch 29: 3804.8058546 3804.80524399527353
Loss on epoch 30: 3796.5058546 3796.45055302418773
```

```
100% [██████████] | 1913/1913 [00:00<00:00, 4997.94it/s]
100% [██████████] | 7652/7652 [00:01<00:00, 5003.26it/s]
```

```
1 30 12 1e-05 0.915077598529712 0.9114849439046604
Loss on epoch 31: 3800.5058546 3800.45695696026146
Loss on epoch 32: 3797.5058546 3797.45131069421774
Loss on epoch 33: 3800.9058546 3800.85432272194943
Loss on epoch 34: 3794.3058546 3794.28749283030637
Loss on epoch 35: 3795.6058546 3795.58340572379537
```

```
100% [██████████] | 1913/1913 [00:00<00:00, 4853.00it/s]
100% [██████████] | 7652/7652 [00:01<00:00, 4939.08it/s]
```

```
1 35 12 1e-05 0.9150245047988564 0.911574794466839
Loss on epoch 36: 3793.1058546 3793.06682578101755
Loss on epoch 37: 3796.0058546 3796.01393349841243
Loss on epoch 38: 3803.5058546 3803.46042640507233
Loss on epoch 39: 3793.1058546 3793.09306661225865
Loss on epoch 40: 3800.3058546 3800.32398629561076
```

```
100% [██████████] | 1913/1913 [00:00<00:00, 4772.00it/s]
100% [██████████] | 7652/7652 [00:01<00:00, 4755.91it/s]
```

```
1 40 13 1.0000000000000002e-06 0.9151000612619972 0.9114971962540483
```

```
from sklearn import metrics
print(model)      # show accuracy metrics for this layer
pd.concat([
    Series({'Accuracy': metrics.accuracy_score(test_gold, test_pred),
```

(continues on next page)

(continued from previous page)

```

'Precision': metrics.precision_score(test_gold, test_pred,
                                      average='weighted'),
'Recall': metrics.recall_score(test_gold, test_pred,
                                 average='weighted')},
name='Test Set').to_frame().T,
Series({'Accuracy': metrics.accuracy_score(train_gold, train_pred),
        'Precision': metrics.precision_score(train_gold, train_pred,
                                              average='weighted'),
        'Recall': metrics.recall_score(train_gold, train_pred,
                                       average='weighted')},
name='Train Set').to_frame().T], axis=0)
"""
          Accuracy  Precision   Recall
Test Set  0.910648  0.906198  0.910648
Train Set 0.914899  0.913187  0.914899
"""

```

```

FFNN(
    (network): Sequential(
        (0): Linear(in_features=300, out_features=300, bias=True)
        (1): Dropout(p=0.3, inplace=False)
        (2): ReLU()
        (3): Linear(in_features=300, out_features=25, bias=True)
    )
    (classifier): LogSoftmax(dim=-1)
)

```

```

'\n          Accuracy  Precision   Recall\nTest Set  0.910648  0.906198  0.
˓→910648\nTrain Set  0.914899  0.913187  0.914899\n\n'
```

```

fig, ax = plt.subplots(num=1, clear=True, figsize=(10, 6))
DataFrame.from_dict({err: {k: v[err] for k, v in accuracy[max_layers].items()} for err in ['train', 'test']}).plot(ax=ax)
ax.set_title(f'Accuracy of DAN with frozen embedding weights')
ax.set_xlabel('Steps')
ax.set_ylabel('Accuracy')
ax.legend(['Train Set', 'Test Set'], loc='upper left')
plt.tight_layout()
plt.savefig(imgdir / f"frozen_accurac.jpg")

```

```

## Confusion Matrix
labels = [events_[e] for e in events_encoder.classes_]
cf_train = DataFrame(confusion_matrix(train_gold, train_pred),
                      index=pd.MultiIndex.from_product([['Actual'], labels]),
                      columns=pd.MultiIndex.from_product([['Predicted'], labels]))
cf_test = DataFrame(confusion_matrix(test_gold, test_pred),
                      index=pd.MultiIndex.from_product([['Actual'], labels]),
                      columns=pd.MultiIndex.from_product([['Predicted'], labels]))
for num, (title, cf) in enumerate({'Training':cf_train, 'Test':cf_test}.items()):
    fig, ax = plt.subplots(num=1+num, clear=True, figsize=(10, 6))
    sns.heatmap(cf, ax=ax, annot=False, fmt='d', cmap='viridis', robust=True,

```

(continues on next page)

(continued from previous page)

```

        yticklabels=[f'{lab} {e}' for lab, e in
                    zip(labels, events_encoder.classes_)],
        xticklabels=events_encoder.classes_)
    ax.set_title(f'{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.savefig(imgdir / f"frozen_{title}.jpg")
    plt.tight_layout()

```

```

## DAN with GloVe embeddings and fine-tune weights
textdata = Textual() # class for text pre-processing

if initial:
    vocab = textdata.counter(lines) # count words for vocab
    textdata(vocab.most_common(20000), 0) # vocab is most common 20000
    textdata.dump('textdata.json', imgdir)

    x_all = textdata[lines] # convert str docs to word indexes
    with gzip.open(imgdir / 'x_all.csv.gz', 'wt', newline="") as f:
        csv.writer(f).writerows(x_all)
else:
    x_all = []
    with gzip.open(imgdir / 'x_all.csv.gz', 'rt') as f:
        for row in csv.reader(f):
            x_all.append(row)
    textdata.load('textdata.json', imgdir)

# hackish convert json str to int
for i in range(len(x_all)):
    for j in range(len(x_all[i])):
        x_all[i][j] = int(x_all[i][j])
print('vocab size', textdata.n)

```

```

## Relativize glove embeddings
# wget http://nlp.stanford.edu/data/wordvecs/glove.6B.zip
vocab_dim = 300
glove_prefix = paths['scratch'] / f"glove.6B.{vocab_dim}d.rel.pkl"
if initial:
    glove = textdata.relativize(glove_prefix + ".txt")
    with open(glove_prefix, "wb") as f:
        pickle.dump(glove, f)
else:
    with open(glove_prefix, "rb") as f:
        glove = pickle.load(f)
print('glove dimensions', glove.shape)

```

```

## train_test split stratified by y_all
textdata.form_splits(y_all, random_state=42, test_size=0.2)

```

```

## define DAN with tunable word embeddings
class DAN(nn.Module):
    """Deep Averaging Network for classification"""
    def __init__(self, vocab_dim, num_classes, hidden, embedding,
                 dropout=0.3, requires_grad=False):
        super().__init__()
        self.embedding = nn.EmbeddingBag.from_pretrained(embedding)
        self.embedding.weight.requires_grad = requires_grad
        V = nn.Linear(vocab_dim, hidden[0])
        nn.init.xavier_uniform_(V.weight)
        L = [V, nn.Dropout(dropout)]
        for g, h in zip(hidden, hidden[1:] + [num_classes]):
            W = nn.Linear(g, h)
            nn.init.xavier_uniform_(W.weight)
            L.extend([nn.ReLU(), W])
        self.network = nn.Sequential(*L)
        self.classifier = nn.LogSoftmax(dim=-1) # output is (N, C) logits

    def tune(self, requires_grad=False):
        self.embedding.weight.requires_grad = requires_grad

    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

```

```

layers = 2
hidden = vocab_dim #100, 300
model = DAN(vocab_dim,
            num_classes,
            hidden=[hidden]*layers,
            embedding=torch.FloatTensor(glove)).to(device)
print(model)

```

```

## Training loop
accuracy = dict()
for tune in [False, True]:
    # define model, optimizer, scheduler, loss_function
    model.tune(tune)
    batch_sz, lr, num_lr, step_sz, eval_skip = 64, 0.01, 4, 5, 5 #3, 3, 3 #
    num_epochs = step_sz * num_lr + 1
    optimizer = torch.optim.Adam(model.parameters(), lr=lr)
    scheduler = torch.optim.lr_scheduler.StepLR(optimizer, gamma=0.1,

```

(continues on next page)

(continued from previous page)

```

loss_function = nn.NLLLoss()
accuracy[tune] = dict()

# Loop over epochs and batches
for epoch in range(0, num_epochs):
    tic = time.time()
    batches = textdata.form_batches(batch_sz)

    total_loss = 0.0
    model.train()
    for batch in tqdm(batches):    # train by batch
        x = textdata.form_input([x_all[idx] for idx in batch]).to(device)
        y = torch.tensor([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
    scheduler.step()             # scheduler step for learning rate
    model.eval()
    model.save(imgdir / f"danGloVe.pt")
    print(f"Loss on epoch {epoch} ({tune}): {total_loss:.1f}")

    with torch.no_grad():
        if epoch % eval_skip == 0:
            test_pred = [model.predict(textdata.form_input(
                [x_all[i]]).to(device))[0] for i in textdata.test_idx]
            test_gold = [int(y_all[idx]) for idx in textdata.test_idx]
            test_correct = (np.asarray(test_pred) ==
                            np.asarray(test_gold)).sum()
            train_pred = [model.predict(textdata.form_input(
                [x_all[i]]).to(device))[0] for i in textdata.train_idx]
            train_gold = [int(y_all[idx]) for idx in textdata.train_idx]
            train_correct = (np.asarray(train_pred) ==
                            np.asarray(train_gold)).sum()
            accuracy[tune][epoch] = {
                'loss': total_loss,
                'train': train_correct/len(train_gold),
                'test': test_correct/len(test_gold)}
            print(tune, epoch, int(time.time()-tic),
                  optimizer.param_groups[0]['lr'],
                  train_correct/len(train_gold),
                  test_correct/len(test_gold))

```

```

## Confusion matrix
labels = [events_[e] for e in events_encoder.classes_]
cf_train = DataFrame(confusion_matrix(train_gold, train_pred),
                     index=pd.MultiIndex.from_product([['Actual'], labels]),
                     columns=pd.MultiIndex.from_product([['Predicted'], labels]))
cf_test = DataFrame(confusion_matrix(test_gold, test_pred),
                     index=pd.MultiIndex.from_product([['Actual'], labels]),
                     columns=pd.MultiIndex.from_product([['Predicted'], labels]))

```

```

#
# First half of sample fixed weights, second half start allow tuning
#
for num, (title, cf) in enumerate({'Training':cf_train,'Test':cf_test}.items()):
    fig, ax = plt.subplots(num=1+num, clear=True, figsize=(10,6))
    sns.heatmap(cf, ax=ax, annot=False, fmt='d', cmap='viridis', robust=True,
                yticklabels=[f'{label} {e}' for label, e in
                            zip(labels, events_encoder.classes_)],
                xticklabels=events_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.savefig(imgdir / f"tuned_{title}.jpg")
    plt.tight_layout()

```

```

fig, ax = plt.subplots(num=1, clear=True, figsize=(10, 6))
DataFrame.from_dict({sample: {(tuning * len(accuracy[tuning]) + epoch): acc[sample]
                                for tuning in [False, True]
                                for epoch, acc in enumerate(accuracy[tuning].values())}
                     for sample in ['train', 'test']}).plot(ax=ax)
ax.axvline((len(accuracy[False])), c='grey', alpha=0.5)
ax.set_title(f'Accuracy of DAN with fine-tuned GloVe weights')
ax.set_xlabel('Steps')
ax.set_ylabel('Accuracy')
ax.legend(['Train Set', 'Test Set', 'Start Fine-Tuning'], loc='upper left')
plt.tight_layout()
plt.savefig(imgdir / f"tuned_accuracy.jpg")

```

```

# 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
"""

if False:
    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)

```

(continues on next page)

(continued from previous page)

```
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])
```

## TEMPORAL CONVOLUTIONAL NETWORK

*UNDER CONSTRUCTION*

- Vector Autoregression

```
import numpy as np
import pandas as pd
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
import re
import time
from datetime import datetime
import random
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import statsmodels.api as sm
from statsmodels.tsa.api import VAR
from finds.plots import plot_bands
from finds.misc.show import Show
from finds.unstructured.store import Store
from secret import credentials, paths

# %matplotlib qt
VERBOSE = 0
show = Show(ndigits=4, latex=None)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```
imgdir = paths['images'] / 'ts'
store = Store(paths['scratch'])
factors = store['approximate'].factors
```

### Vector Autoregression Model

predict multiple time series (of the extracted approximate factors)

```
maxlags = 16
train_split = '2014-12-31' # training period up to this date
test_split = '2019-12-31' # split test period into two (due to COVID)
train_index = factors.index[factors.index <= train_split]
test1_index = factors.index[(factors.index >= train_index[-maxlags]) &
                           (factors.index <= test_split)]
test2_index = factors.index[(factors.index >= test1_index[-maxlags])]
```

(continues on next page)

(continued from previous page)

```
test_end = max(test2_index).strftime('%Y-%m-%d')
train_data = factors.loc[train_index].copy()
test1_data = factors.loc[test1_index].copy()
test2_data = factors.loc[test2_index].copy()
M = train_data.shape[1]      # M is number of vectors to autoregress
model = VAR(train_data, freq='M')
```

### Auto Select Lag Order

The lagged coefficients estimated from the Vector Autoregression produce a multi-period cumulative forecast

```
show(DataFrame({ic: model.fit(maxlags=maxlags, ic=ic).k_ar
                for ic in ['aic', 'fpe', 'hqic', 'bic']},
               index=['optimal p:'])\
        .rename_axis(columns='IC:'),
        caption="Optimal number of VAR(p) lags selected by Information Criterion")
```

IC:	aic	fpe	hqic	bic
Optimal number of VAR(p) lags selected by Infor...	9	9	3	2
optimal p:				

```
### Collect one-period ahead forecasts and errors in train and test sets
results = {p: model.fit(p) for p in range(1, maxlags+1)}    # VAR(p) models
from sklearn.metrics import mean_squared_error as mse
mean_error = dict()  # to collect unconditional mean error each month
var_error= {p: dict() for p in results}  # collect error each month for each p
```

```
for sample in [train_data, test1_data, test2_data]:  # loop over 3 subsamples

    for i in range(maxlags, len(sample)):  # start at maxlag'th obs

        # accumulate to error of unconditional mean forecast
        mean_error[sample.index[i]] = mse(sample.iloc[i].values,
                                           train_data.mean().values)

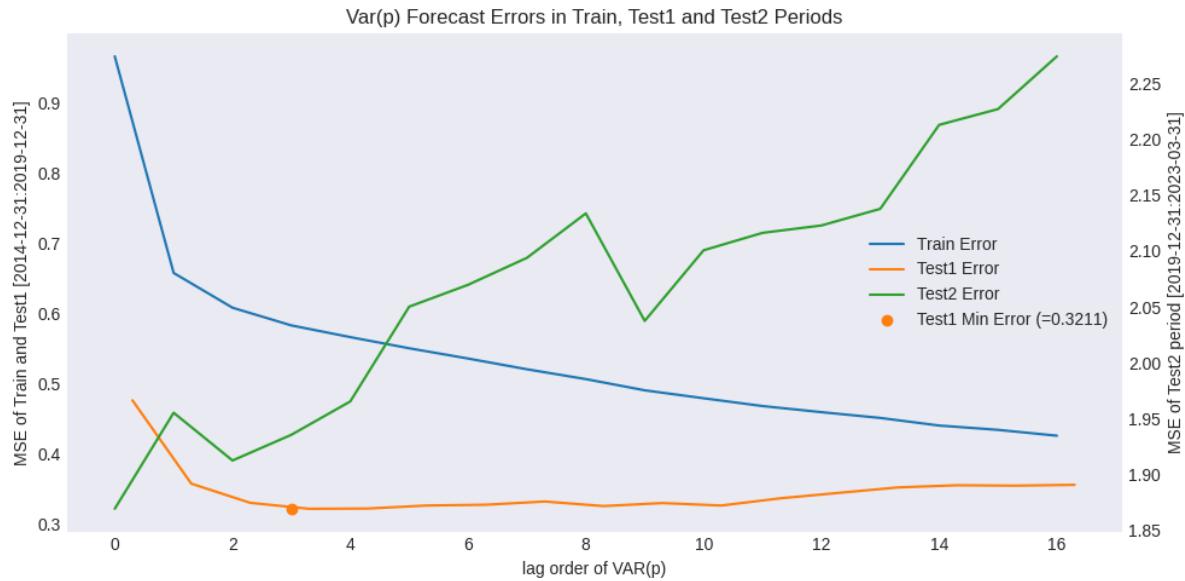
        # accumulate to error of VAR(p) model forecasts
        for p in var_error:

            # include p prior for VAR(p) to forecast i'th obs
            pred = results[p].forecast(sample.iloc[(i-p):i].values, 1)
            var_error[p][sample.index[i]] = mse([sample.iloc[i].values], pred)
```

```
# Collect mean test and train set errors of all VAR(p) models
errors = {0: Series(mean_error, name="TrainSampleMean")}  # VAR(0)
errors.update({p: Series(var_error[p], name=f"VAR({p})")
               for p in var_error})
out = [Series({'Train Error': e.loc[e.index <= train_split].mean(),
              'Test1 Error': e.loc[test1_index].mean(),
              'Test2 Error': e.loc[test2_index].mean()},
              name=e.name) for p, e in errors.items()]
out = pd.concat(out, axis=1).T.rename_axis(columns="1961-07-31...2019-12-31:")
out
```

1961-07-31...2019-12-31:	Train Error	Test1 Error	Test2 Error
TrainSampleMean	0.966422	0.475856	1.869434
VAR(1)	0.657480	0.356948	1.955375
VAR(2)	0.607881	0.329619	1.912644
VAR(3)	0.582625	0.321119	1.935668
VAR(4)	0.565950	0.321482	1.965509
VAR(5)	0.550050	0.325775	2.050291
VAR(6)	0.535415	0.326891	2.069849
VAR(7)	0.520095	0.331472	2.094007
VAR(8)	0.506079	0.325011	2.133762
VAR(9)	0.490080	0.329260	2.037579
VAR(10)	0.478760	0.325770	2.100757
VAR(11)	0.467626	0.336032	2.116336
VAR(12)	0.458930	0.343839	2.122953
VAR(13)	0.450689	0.351576	2.137847
VAR(14)	0.439834	0.354656	2.212973
VAR(15)	0.433670	0.354110	2.227124
VAR(16)	0.425329	0.355261	2.274311

```
### Plot Train and Test Error of all VAR(p) Models
fig, ax = plt.subplots(1, 1, figsize=(10, 5), num=1, clear=True)
ax.plot(np.arange(len(out)), out['Train Error'], color="C0")
ax.plot(np.arange(len(out))+.3, out['Test1 Error'], color="C1")
ax.plot([], [], color="C2") # dummy for legend labels
argmin = out['Test1 Error'].argmin()
ax.plot(argmin, out.iloc[argmin]['Test1 Error'], 'o', color="C1")
bx = ax.twinx()
bx.plot(np.arange(len(out)), out['Test2 Error'], color=f"C2")
ax.set_title(f'Var(p) Forecast Errors in Train, Test1 and Test2 Periods')
ax.set_ylabel(f'MSE of Train and Test1 [{train_split}:{test_split}]')
bx.set_ylabel(f'MSE of Test2 period [{test_split}:{test_end}]')
ax.set_xlabel('lag order of VAR(p)')
ax.legend(['Train Error', 'Test1 Error', 'Test2 Error',
           f'Test1 Min Error (=out.iloc[argmin]["Test1 Error"]:.4f)'],
           loc='center right')
plt.tight_layout()
plt.savefig(imgdir / 'varerr.jpg')
```



## 37.1 Temporal Convolutional Net (TCN)

```

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

```

```

### Form input data from training set
seq_len = 16      # length of each input sequence for TCN
train_exs = [train_data.iloc[i-(seq_len+1):i].values
             for i in range(seq_len+1, len(train_data))]

```

```

### Fit TCN models with increasing layers of convolution and dropout rates
n_features = train_data.shape[1]      # number of input planes
batch_size = 8
step_size = 100          # learning rate scheduler step size
lr = 0.01                # initial learning rate
num_lr = 6
res = []                  # to collect results summaries
tcn_error = dict()        # to store prediction errors

```

```

kernel_sizes = [1, 2, 4]
dropouts = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5]
blocks = [0, 1, 2]
for block in blocks:
    for parm in (dropouts if block else kernel_sizes):
        if block:
            dropout = parm
            kernel_size = 3
        else:
            dropout = 0.0
            kernel_size = parm
        modelname = f"TCN{block}_{kernel_size}_{dropout*100:.0f}"

```

(continues on next page)

(continued from previous page)

```

# 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(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):
    idxs = np.arange(len(train_exs))    # shuffle idxs into batches
    random.shuffle(idxs)
    batches = [idxs[i:(i+batch_size)]
               for i in range(0, len(idxs), batch_size)]

    total_loss = 0.0                  # train by batch
    model.train()
    for batch in batches:
        # input has shape (batch_size=8, n_features=8, seq_len=16)
        # Creating a tensor from a list of numpy.ndarrays is extremely
        # slow. Please consider converting the list to a single
        # numpy.ndarray with numpy.array() before converting to a tensor
        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)
        loss.backward()
        optimizer.step()
    scheduler.step()
    if VERBOSE and (epoch % (step_size//2)) == 0:
        print(epoch, num_epochs, block, dropout,
              optimizer.param_groups[0]['lr'], total_loss/len(batches))
    model.eval()

    # Compute MSE of one-period ahead forecast error in train and test sets
    e = dict()
    for x in [train_data, test1_data, test2_data]:
        for i in range(seq_len, len(x)):
            X = torch.tensor(x.iloc[(i-seq_len):i].values.T) \
                .unsqueeze(0).float().to(device)
            pred = model(X)
            e[x.index[i]] = mse([x.iloc[i].values],
                               pred[:, :, -1].cpu().detach().numpy())
    model.save(imgdir / (modelname + '.pt'))
    e = Series(e, name=modelname)
    tcn_error[modelname] = e
    res.append(Series({
        'blocks': block, 'dropout': dropout, 'kernel_size': kernel_size,

```

(continues on next page)

(continued from previous page)

```

    'Train Error': float(e[e.index <= train_split].mean()),
    'Test1 Error': float(e[test1_index].mean()),
    'Test2 Error': float(e[test2_index].mean())}, name=modelname)
#print(pd.concat(res, axis=1).T)
res = pd.concat(res, axis=1).T.astype({'blocks': int, 'kernel_size': int})
res

```

```

TCN(
    (classifier): Sequential(
        (0): ConstantPad1d(padding=(0, 0), value=0)
        (1): Conv1d(8, 8, kernel_size=(1,), stride=(1,)))
    )
)
TCN(
    (classifier): Sequential(
        (0): ConstantPad1d(padding=(1, 0), value=0)
        (1): Conv1d(8, 8, kernel_size=(2,), stride=(1,)))
    )
)
TCN(
    (classifier): Sequential(
        (0): ConstantPad1d(padding=(3, 0), value=0)
        (1): Conv1d(8, 8, kernel_size=(4,), stride=(1,)))
    )
)
kernel 3 dilation 1
TCN(
    (network): Sequential(
        (0): CausalConv1dBlock(
            (network): Sequential(
                (0): ConstantPad1d(padding=(2, 0), value=0)
                (1): Conv1d(8, 8, kernel_size=(3,), stride=(1,)))
                (2): ReLU()
                (3): ConstantPad1d(padding=(2, 0), value=0)
                (4): Conv1d(8, 8, kernel_size=(3,), stride=(1,)))
                (5): ReLU()
                (6): Dropout(p=0.0, inplace=False)
            )
        )
    )
    (classifier): Conv1d(8, 8, kernel_size=(1,), stride=(1,)))
)
kernel 3 dilation 1
TCN(
    (network): Sequential(
        (0): CausalConv1dBlock(
            (network): Sequential(
                (0): ConstantPad1d(padding=(2, 0), value=0)
                (1): Conv1d(8, 8, kernel_size=(3,), stride=(1,)))
                (2): ReLU()
                (3): ConstantPad1d(padding=(2, 0), value=0)
                (4): Conv1d(8, 8, kernel_size=(3,), stride=(1,)))
                (5): ReLU()
                (6): Dropout(p=0.1, inplace=False)
            )
        )
    )
)

```

(continues on next page)

(continued from previous page)

```

        )
        (classifier): Conv1d(8, 8, kernel_size=(1,), stride=(1,))
    )
kernel 3 dilation 1
TCN(
    (network): Sequential(
        (0): CausalConv1dBlock(
            (network): Sequential(
                (0): ConstantPad1d(padding=(2, 0), value=0)
                (1): Conv1d(8, 8, kernel_size=(3,), stride=(1,))
                (2): ReLU()
                (3): ConstantPad1d(padding=(2, 0), value=0)
                (4): Conv1d(8, 8, kernel_size=(3,), stride=(1,))
                (5): ReLU()
                (6): Dropout (p=0.2, inplace=False)
            )
        )
    )
    (classifier): Conv1d(8, 8, kernel_size=(1,), stride=(1,))
)
kernel 3 dilation 1
TCN(
    (network): Sequential(
        (0): CausalConv1dBlock(
            (network): Sequential(
                (0): ConstantPad1d(padding=(2, 0), value=0)
                (1): Conv1d(8, 8, kernel_size=(3,), stride=(1,))
                (2): ReLU()
                (3): ConstantPad1d(padding=(2, 0), value=0)
                (4): Conv1d(8, 8, kernel_size=(3,), stride=(1,))
                (5): ReLU()
                (6): Dropout (p=0.3, inplace=False)
            )
        )
    )
    (classifier): Conv1d(8, 8, kernel_size=(1,), stride=(1,))
)
kernel 3 dilation 1
TCN(
    (network): Sequential(
        (0): CausalConv1dBlock(
            (network): Sequential(
                (0): ConstantPad1d(padding=(2, 0), value=0)
                (1): Conv1d(8, 8, kernel_size=(3,), stride=(1,))
                (2): ReLU()
                (3): ConstantPad1d(padding=(2, 0), value=0)
                (4): Conv1d(8, 8, kernel_size=(3,), stride=(1,))
                (5): ReLU()
                (6): Dropout (p=0.4, inplace=False)
            )
        )
    )
    (classifier): Conv1d(8, 8, kernel_size=(1,), stride=(1,))
)
kernel 3 dilation 1
TCN(

```

(continues on next page)

(continued from previous page)

```

(network): Sequential(
    (0): CausalConv1dBlock(
        (network): Sequential(
            (0): ConstantPad1d(padding=(2, 0), value=0)
            (1): Conv1d(8, 8, kernel_size=(3,), stride=(1,))
            (2): ReLU()
            (3): ConstantPad1d(padding=(2, 0), value=0)
            (4): Conv1d(8, 8, kernel_size=(3,), stride=(1,))
            (5): ReLU()
            (6): Dropout (p=0.5, inplace=False)
        )
    )
)
(classifier): Conv1d(8, 8, kernel_size=(1,), stride=(1,))
)
kernel 3 dilation 1
kernel 3 dilation 2
TCN(
    (network): Sequential(
        (0): CausalConv1dBlock(
            (network): Sequential(
                (0): ConstantPad1d(padding=(2, 0), value=0)
                (1): Conv1d(8, 8, kernel_size=(3,), stride=(1,))
                (2): ReLU()
                (3): ConstantPad1d(padding=(2, 0), value=0)
                (4): Conv1d(8, 8, kernel_size=(3,), stride=(1,))
                (5): ReLU()
                (6): Dropout (p=0.0, inplace=False)
            )
        )
    )
    (1): CausalConv1dBlock(
        (network): Sequential(
            (0): ConstantPad1d(padding=(4, 0), value=0)
            (1): Conv1d(8, 8, kernel_size=(3,), stride=(1,), dilation=(2,))
            (2): ReLU()
            (3): ConstantPad1d(padding=(4, 0), value=0)
            (4): Conv1d(8, 8, kernel_size=(3,), stride=(1,), dilation=(2,))
            (5): ReLU()
            (6): Dropout (p=0.0, inplace=False)
        )
    )
)
(classifier): Conv1d(8, 8, kernel_size=(1,), stride=(1,))
)
kernel 3 dilation 1
kernel 3 dilation 2
TCN(
    (network): Sequential(
        (0): CausalConv1dBlock(
            (network): Sequential(
                (0): ConstantPad1d(padding=(2, 0), value=0)
                (1): Conv1d(8, 8, kernel_size=(3,), stride=(1,))
                (2): ReLU()
                (3): ConstantPad1d(padding=(2, 0), value=0)
                (4): Conv1d(8, 8, kernel_size=(3,), stride=(1,))
                (5): ReLU()
            )
        )
    )
)

```

(continues on next page)

(continued from previous page)

```

        (6): Dropout(p=0.1, inplace=False)
    )
)
(1): CausalConv1dBlock(
    (network): Sequential(
        (0): ConstantPad1d(padding=(4, 0), value=0)
        (1): Conv1d(8, 8, kernel_size=(3,), stride=(1,), dilation=(2,))
        (2): ReLU()
        (3): ConstantPad1d(padding=(4, 0), value=0)
        (4): Conv1d(8, 8, kernel_size=(3,), stride=(1,), dilation=(2,))
        (5): ReLU()
        (6): Dropout(p=0.1, inplace=False)
    )
)
)
(classifier): Conv1d(8, 8, kernel_size=(1,), stride=(1,))
)
kernel 3 dilation 1
kernel 3 dilation 2
TCN(
    (network): Sequential(
        (0): CausalConv1dBlock(
            (network): Sequential(
                (0): ConstantPad1d(padding=(2, 0), value=0)
                (1): Conv1d(8, 8, kernel_size=(3,), stride=(1,))
                (2): ReLU()
                (3): ConstantPad1d(padding=(2, 0), value=0)
                (4): Conv1d(8, 8, kernel_size=(3,), stride=(1,))
                (5): ReLU()
                (6): Dropout(p=0.2, inplace=False)
            )
)
        (1): CausalConv1dBlock(
            (network): Sequential(
                (0): ConstantPad1d(padding=(4, 0), value=0)
                (1): Conv1d(8, 8, kernel_size=(3,), stride=(1,), dilation=(2,))
                (2): ReLU()
                (3): ConstantPad1d(padding=(4, 0), value=0)
                (4): Conv1d(8, 8, kernel_size=(3,), stride=(1,), dilation=(2,))
                (5): ReLU()
                (6): Dropout(p=0.2, inplace=False)
            )
)
        )
    )
(classifier): Conv1d(8, 8, kernel_size=(1,), stride=(1,))
)
kernel 3 dilation 1
kernel 3 dilation 2
TCN(
    (network): Sequential(
        (0): CausalConv1dBlock(
            (network): Sequential(
                (0): ConstantPad1d(padding=(2, 0), value=0)
                (1): Conv1d(8, 8, kernel_size=(3,), stride=(1,))
                (2): ReLU()
                (3): ConstantPad1d(padding=(2, 0), value=0)
            )
)
)
)

```

(continues on next page)

(continued from previous page)

```

(4): Conv1d(8, 8, kernel_size=(3,), stride=(1,))
(5): ReLU()
(6): Dropout(p=0.3, inplace=False)
)
)
(1): CausalConv1dBlock(
(network): Sequential(
(0): ConstantPad1d(padding=(4, 0), value=0)
(1): Conv1d(8, 8, kernel_size=(3,), stride=(1,), dilation=(2,))
(2): ReLU()
(3): ConstantPad1d(padding=(4, 0), value=0)
(4): Conv1d(8, 8, kernel_size=(3,), stride=(1,), dilation=(2,))
(5): ReLU()
(6): Dropout(p=0.3, inplace=False)
)
)
)
(classifier): Conv1d(8, 8, kernel_size=(1,), stride=(1,))
)
kernel 3 dilation 1
kernel 3 dilation 2
TCN(
(network): Sequential(
(0): CausalConv1dBlock(
(network): Sequential(
(0): ConstantPad1d(padding=(2, 0), value=0)
(1): Conv1d(8, 8, kernel_size=(3,), stride=(1,))
(2): ReLU()
(3): ConstantPad1d(padding=(2, 0), value=0)
(4): Conv1d(8, 8, kernel_size=(3,), stride=(1,))
(5): ReLU()
(6): Dropout(p=0.4, inplace=False)
)
)
(1): CausalConv1dBlock(
(network): Sequential(
(0): ConstantPad1d(padding=(4, 0), value=0)
(1): Conv1d(8, 8, kernel_size=(3,), stride=(1,), dilation=(2,))
(2): ReLU()
(3): ConstantPad1d(padding=(4, 0), value=0)
(4): Conv1d(8, 8, kernel_size=(3,), stride=(1,), dilation=(2,))
(5): ReLU()
(6): Dropout(p=0.4, inplace=False)
)
)
)
(classifier): Conv1d(8, 8, kernel_size=(1,), stride=(1,))
)
kernel 3 dilation 1
kernel 3 dilation 2
TCN(
(network): Sequential(
(0): CausalConv1dBlock(
(network): Sequential(
(0): ConstantPad1d(padding=(2, 0), value=0)
(1): Conv1d(8, 8, kernel_size=(3,), stride=(1,))

```

(continues on next page)

(continued from previous page)

```

        (2): ReLU()
        (3): ConstantPad1d(padding=(2, 0), value=0)
        (4): Conv1d(8, 8, kernel_size=(3,), stride=(1,))
        (5): ReLU()
        (6): Dropout(p=0.5, inplace=False)
    )
)
(1): CausalConv1dBlock(
    network): Sequential(
        (0): ConstantPad1d(padding=(4, 0), value=0)
        (1): Conv1d(8, 8, kernel_size=(3,), stride=(1,), dilation=(2,))
        (2): ReLU()
        (3): ConstantPad1d(padding=(4, 0), value=0)
        (4): Conv1d(8, 8, kernel_size=(3,), stride=(1,), dilation=(2,))
        (5): ReLU()
        (6): Dropout(p=0.5, inplace=False)
    )
)
)
(classifier): Conv1d(8, 8, kernel_size=(1,), stride=(1,))
)

```

	blocks	dropout	kernel_size	Train Error	Test1 Error	Test2 Error
TCN0_1_0	0	0.0		1 0.657199	0.358790	1.960907
TCN0_2_0	0	0.0		2 0.609275	0.330528	1.906794
TCN0_4_0	0	0.0		4 0.572158	0.321935	1.956305
TCN1_3_0	1	0.0		3 0.420763	0.390504	1.798904
TCN1_3_10	1	0.1		3 0.446327	0.384417	2.004865
TCN1_3_20	1	0.2		3 0.476245	0.419338	2.024344
TCN1_3_30	1	0.3		3 0.492200	0.391346	2.291207
TCN1_3_40	1	0.4		3 0.519895	0.368835	1.940071
TCN1_3_50	1	0.5		3 0.544130	0.362974	1.971947
TCN2_3_0	2	0.0		3 0.336047	0.411147	2.053564
TCN2_3_10	2	0.1		3 0.412089	0.394387	2.213243
TCN2_3_20	2	0.2		3 0.405027	0.388860	2.088544
TCN2_3_30	2	0.3		3 0.452836	0.384957	2.190841
TCN2_3_40	2	0.4		3 0.468511	0.378591	2.072996
TCN2_3_50	2	0.5		3 0.514378	0.379931	2.036257

```

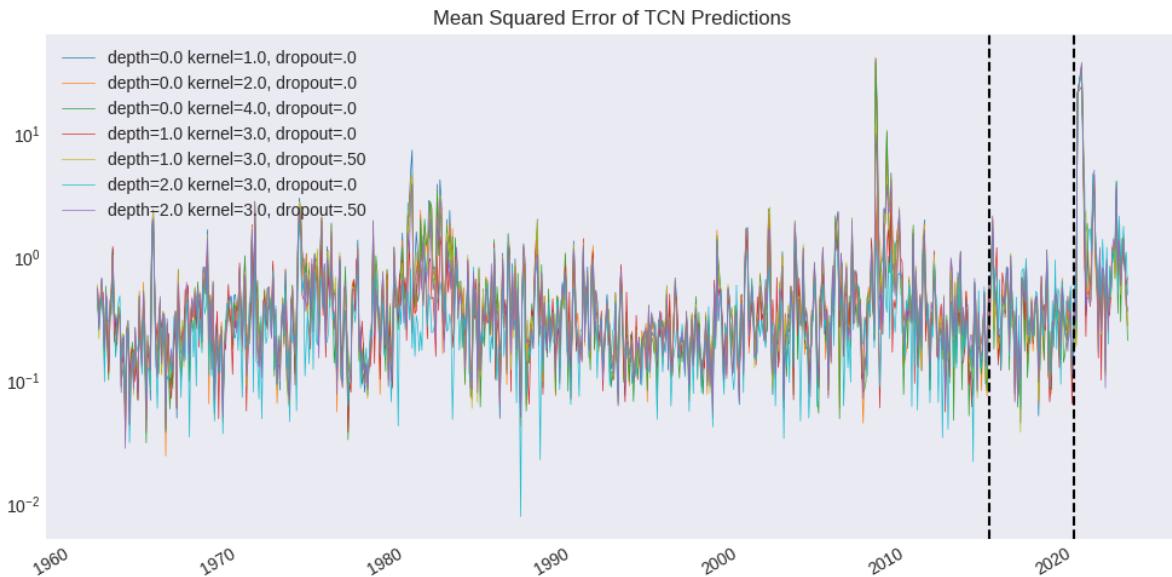
### Plot monthly mean squarejd error
fig, ax = plt.subplots(1, 1, figsize=(10, 5), num=1, clear=True)
ax.set_yscale('log')
legend = []
for col, (modelname, parm) in enumerate(res.iterrows()):
    if parm.dropout in [0.0, 0.5]:
        tcn_error[modelname].plot(ax=ax, c=f'C{col}', lw=.5, ls='--')
        legend.append(f"depth={parm.blocks} kernel={parm.kernel_size}, "
                      f"dropout={100*parm.dropout:.0f}")
# for a,b in vspans:
#     if a >= train_index[maxlags]:
#         ax.axvspan(a, min(b, max(test2_index)), alpha=0.3, color='grey')
ax.set_title('Mean Squared Error of TCN Predictions')
ax.axvline(max(train_index), color='black', linestyle='--')
ax.axvline(max(test1_index), color='black', linestyle='--')
ax.legend(legend, loc='upper left')

```

(continues on next page)

(continued from previous page)

```
plt.tight_layout()
plt.savefig(imgdir / 'tcnmse.jpg')
```



```
### Display train and test error of TCN models by dropout parameters
res[res['blocks'].gt(0)]\ 
    .rename(columns={s+' Error': s for s in ['Train', 'Test1', 'Test2']})\ 
    .pivot(index=['dropout'], columns=['blocks'])\ 
    .drop(columns=['kernel_size'])\ 
    .swaplevel(0, 1, 1).round(4).sort_index(axis=1)
```

blocks	1			2		
	Test1	Test2	Train	Test1	Test2	Train
dropout						
0.0	0.3905	1.7989	0.4208	0.4111	2.0536	0.3360
0.1	0.3844	2.0049	0.4463	0.3944	2.2132	0.4121
0.2	0.4193	2.0243	0.4762	0.3889	2.0885	0.4050
0.3	0.3913	2.2912	0.4922	0.3850	2.1908	0.4528
0.4	0.3688	1.9401	0.5199	0.3786	2.0730	0.4685
0.5	0.3630	1.9719	0.5441	0.3799	2.0363	0.5144

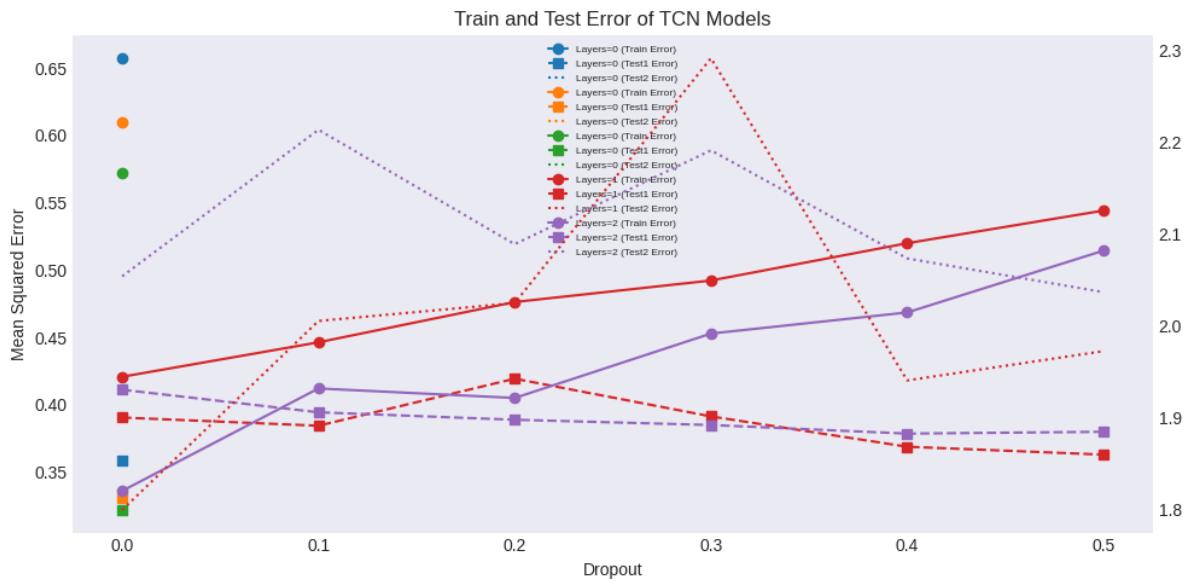
```
### Display train and test error of TCN models by size parameters
res[res['blocks'].eq(0)]\ 
    .rename(columns={s+' Error': s for s in ['Train', 'Test1', 'Test2']})\ 
    .pivot(index=['kernel_size'], columns=['blocks'])\ 
    .drop(columns=['dropout'])\ 
    .swaplevel(0, 1, 1).round(4).sort_index(axis=1)
```

blocks	0		
	Test1	Test2	Train
kernel_size			
1	0.3588	1.9609	0.6572
2	0.3305	1.9068	0.6093
4	0.3219	1.9563	0.5722

```
### Plot Train and Test Error of TCN Models
fig, ax = plt.subplots(1, 1, figsize=(10, 5), num=1, clear=True)
bx = ax.twiny()
col = 0
for block in np.unique(res.blocks):
    select_block = res['blocks'].eq(block)
    for kernel_size in np.unique(res['kernel_size'][select_block]):
        select = res['blocks'].eq(block) & res['kernel_size'].eq(kernel_size)

        Series(index=res['dropout'][select],
               data=res['Train Error'][select].values,
               name=f"Layers={block}.0f (Train Error)") \
            .plot(ax=ax, color=f"C{col}", style='-', marker='o')
        Series(index=res['dropout'][select],
               data=res['Test1 Error'][select].values,
               name=f"Layers={block}.0f (Test1 Error)") \
            .plot(ax=ax, color=f"C{col}", style='--', marker='s')
        ax.plot([],[], color=f"C{col}", ls=':', label=f"Layers={block}.0f (Test2 Error)")
        Series(index=res['dropout'][select],
               data=res['Test2 Error'][select].values) \
            .plot(ax=bx, color=f"C{col}", style=':')

    col = col + 1
ax.set_title('Train and Test Error of TCN Models')
ax.set_ylabel('Mean Squared Error')
ax.set_xlabel('Dropout')
ax.legend(loc='upper center', fontsize=6)
plt.tight_layout()
plt.savefig(imgdir / 'tcn.jpg')
```



---

CHAPTER  
THIRTYEIGHT

---

## RECURRENT NEURAL NETWORKS

### UNDER CONSTRUCTION

- RNN, Elman Network, LSTM
- compare Linear Dynamic Factor Models

```
import numpy as np
import pandas as pd
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
import re
import time
from datetime import datetime
import statsmodels.api as sm
import random
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from sklearn.preprocessing import StandardScaler
from tqdm import tqdm
from finds.readers.alfred import fred_md, fred_qd, Alfred
from finds.misc.show import Show
from secret import paths, credentials
```

```
# %matplotlib qt
VERBOSE = 0
show = Show(ndigits=4, latex=None)
imgdir = paths['images'] / 'states'
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```
### Load and pre-process time series from FRED
alf = Alfred(api_key=credentials['fred']['api_key'])
vspan = alf.date_spans('USREC') # to indicate recession periods in the plots
beg = 19600301
end = 20200131
# TODO: end = train up to MINUS two quarters, forecast through current
```

```
### Retrieve FRED-MD series and apply tcode transformations
df, t = fred_md(202004) # from vintage April 2020?
data = []
for col in df.columns:
```

(continues on next page)

(continued from previous page)

```

data.append(alf.transform(df[col], tcode=t['transform'][col], freq='m'))
mdf = pd.concat(data, axis=1).iloc[2:]
mdata = mdf[(mdf.index >= beg) & (mdf.index <= end)].dropna(axis=1)
mdata = (mdata - mdata.mean(axis=0)) / mdata.std(axis=0, ddof=0)
mdata.index = pd.DatetimeIndex(mdata.index.astype(str), freq='m')
mdata

```

monthly/2020-04.csv

	RPI	W875RX1	DPCERA3M086SBEA	CMRMTSPL	RETAIL	INDPRO	
1960-03-31	-0.134881	-0.275771	2.216461	-2.860458	-0.488916	-1.494126	\
1960-04-30	0.140605	0.203278	2.471089	0.718774	1.808191	-1.351744	
1960-05-31	-0.045275	-0.004559	-4.482796	-3.197753	-1.675079	-0.432646	
1960-06-30	-0.338482	-0.482476	-0.559540	0.602494	-0.511228	-1.983791	
1960-07-31	-0.148985	-0.100803	-0.122954	-0.860250	-1.254201	-0.747608	
...	...	...	...	...	...	...	
2019-09-30	-0.079715	-0.081618	-0.174630	-0.239203	-0.752893	-0.747932	
2019-10-31	-0.598714	-0.639323	-0.385917	-0.656535	-0.108856	-0.828301	
2019-11-30	0.202376	0.355149	-0.057531	0.283946	-0.217945	0.983266	
2019-12-31	-0.644414	-0.535997	-0.324231	-0.230203	-0.374542	-0.815438	
2020-01-31	0.203651	-0.063344	0.048876	0.250612	0.246743	-0.934552	
	IPFPNSS	IPFINAL	IPCONGD	IPDCONGD	...	DDURRG3M086SBEA	
1960-03-31	-0.747423	-0.399146	-0.045307	-1.035927	...	-1.469820	\
1960-04-30	0.044685	-0.109636	0.617368	-0.158355	...	1.329945	
1960-05-31	0.516409	0.612768	0.480566	0.339132	...	-0.418471	
1960-06-30	-1.854133	-1.702628	-0.970468	-0.600344	...	-0.849520	
1960-07-31	-1.070579	-0.984785	-1.243071	-1.916083	...	0.918360	
...	...	...	...	...	...	...	
2019-09-30	-1.076215	-1.247348	-0.984334	-1.453438	...	-0.071819	
2019-10-31	-0.648297	-0.596663	-0.251664	-1.417370	...	0.074042	
2019-11-30	1.686887	2.054174	2.128329	2.857134	...	-0.704147	
2019-12-31	-1.127034	-1.336095	-1.517772	-1.297690	...	-0.239385	
2020-01-31	-1.599474	-2.152708	-1.360395	0.204324	...	1.763764	
	DNDGRG3M086SBEA	DSERRG3M086SBEA	CES0600000008	CES2000000008			
1960-03-31	0.295721	-0.643390	-0.003771	3.306562	\		
1960-04-30	0.735110	0.960469	-2.253155	-7.467190			
1960-05-31	-1.113542	0.316095	2.255746	4.566901			
1960-06-30	0.036686	-0.482642	-1.125930	-1.252825			
1960-07-31	0.302768	0.421378	1.123500	0.828946			
...	...	...	...	...	...	...	
2019-09-30	-0.163996	-0.071125	-0.204290	-0.307419			
2019-10-31	1.267315	0.290075	-0.202566	0.000637			
2019-11-30	-0.295116	-0.421273	-0.201345	-0.037554			
2019-12-31	0.522518	1.193464	0.604609	0.457551			
2020-01-31	-0.802817	-0.654722	-0.703859	-0.456666			
	CES3000000008	MZMSL	DTCOLNVHFNM	DTCTHFNM	INVEST		
1960-03-31	-1.010361	0.316937	0.142584	0.047443	0.324377		
1960-04-30	0.001676	-0.193935	0.356025	0.269794	2.267079		
1960-05-31	0.001676	0.188096	-0.197676	-0.101655	0.353369		
1960-06-30	0.001676	0.186521	0.210401	0.367009	-0.943151		
1960-07-31	0.001676	0.247264	-0.447705	-0.162111	3.023607		
...	...	...	...	...	...	...	

(continues on next page)

(continued from previous page)

2019-09-30	0.098370	0.138877	-0.163879	-0.281842	0.375870
2019-10-31	0.000978	0.522365	0.080482	0.200078	-1.320725
2019-11-30	0.484033	-0.370009	-0.095845	-0.056103	0.600435
2019-12-31	-0.386623	-0.767687	0.246058	0.036262	-0.996952
2020-01-31	-0.672961	0.661603	-0.219878	-0.078393	0.361933

[719 rows x 123 columns]

```
### Retrieve FRED-QD series and apply tcode transformations
df, t = fred_qd(202004)      # from vintage April 2020
data = []
for col in df.columns:
    data.append(alf.transform(df[col], tcode=t['transform'][col], freq='q'))
df = pd.concat(data, axis=1).iloc[2:]
qdata = df[(df.index >= beg) & (df.index <= end)].dropna(axis=1)
qdata.index = pd.DatetimeIndex(qdata.index.astype(str), freq='q')
qdata
```

quarterly/2020-04.csv

	GDPC1	PCECC96	PCDG	PCESV	PCND	GPDIC1	
1960-03-31	0.022224	0.009521	0.031666	0.009043	0.002506	0.095788	\
1960-06-30	-0.005408	0.012535	0.022501	0.011003	0.010896	-0.097837	
1960-09-30	0.004881	-0.004005	-0.008030	-0.001174	-0.005880	-0.002544	
1960-12-31	-0.012917	0.001294	-0.025450	0.009625	0.000690	-0.117411	
1961-03-31	0.006728	-0.000372	-0.056541	0.009798	0.005951	0.025621	
...	...	...	...	...	...	...	
2018-12-31	0.002710	0.003565	0.003155	0.003444	0.004208	0.007390	
2019-03-31	0.007623	0.002829	0.000639	0.002416	0.005331	0.014937	
2019-06-30	0.004985	0.011135	0.030572	0.006846	0.015664	-0.016345	
2019-09-30	0.005204	0.007750	0.019467	0.005429	0.009520	-0.002477	
2019-12-31	0.005261	0.004535	0.006805	0.005952	-0.001400	-0.015571	
	FPI	Y033RC1Q027SBEA	PNFI	PRFI	...	TLBSNNB	
1960-03-31	0.032856	0.034200	0.034166	0.030306	...	0.037652	\
1960-06-30	-0.018342	0.016711	0.014503	-0.085422	...	0.018321	
1960-09-30	-0.021666	-0.044133	-0.017656	-0.030290	...	0.018804	
1960-12-31	-0.001999	-0.035770	-0.002352	-0.001084	...	-0.005096	
1961-03-31	-0.008523	-0.033095	-0.014372	0.003940	...	0.038024	
...	...	...	...	...	...	...	
2018-12-31	0.006561	0.017770	0.011718	-0.011973	...	0.021630	
2019-03-31	0.007913	-0.000245	0.010812	-0.002634	...	0.014793	
2019-06-30	-0.003600	0.002076	-0.002531	-0.007507	...	0.004232	
2019-09-30	-0.002080	-0.009578	-0.005777	0.011351	...	0.012694	
2019-12-31	-0.001371	-0.010886	-0.006189	0.015724	...	0.012437	
	TLBSNNBBDI	TABSNNB	TNWBSNNB	TNWBSNNBBDI	CNCF	S&P 500	
1960-03-31	279155.40	0.009183	0.004308	86.68	-0.051251	-0.026073	\
1960-06-30	280144.95	0.003057	0.000375	-22.94	0.014782	-0.003680	
1960-09-30	275164.71	0.007952	0.006017	-47.83	0.036742	-0.006262	
1960-12-31	273738.35	0.003843	0.005440	8.20	0.000101	-0.006964	
1961-03-31	291545.95	0.009200	0.003994	45.35	-0.025001	0.113813	
...	...	...	...	...	...	...	
2018-12-31	398935.93	0.013740	0.008576	12.73	-0.012680	-0.057381	

(continues on next page)

(continued from previous page)

```

2019-03-31    399732.21  0.016181  0.017095      2.61  0.012799  0.011111
2019-06-30    392693.84  0.011550  0.016331     -3.44  0.021996  0.057399
2019-09-30    388971.84  0.013807  0.014530     -4.63  0.022218  0.025918
2019-12-31    389233.63  0.005609  0.001158     -6.33  0.011764  0.042307

      S&P: indust  S&P  div  yield  S&P  PE  ratio
1960-03-31    -0.029896    0.2278   -0.057371
1960-06-30    -0.006778    0.0876   -0.024349
1960-09-30    -0.012285    0.0276   0.001369
1960-12-31    -0.006569    0.0257   0.000796
1961-03-31     0.111342   -0.3889   0.128986
...
2018-12-31    -0.058307    0.1617   -0.098323
2019-03-31     0.011686    0.0233   -0.033627
2019-06-30     0.059265   -0.0731   0.013658
2019-09-30     0.025926   -0.0106   0.004248
2019-12-31     0.041578   -0.0445   0.035413

[240 rows x 211 columns]

```

## 38.1 LSTM

in pytorch for time series

```

class LSTM(nn.Module):
    def __init__(self, n_features, hidden_size, num_layers=1):
        super().__init__()
        self.lstm = nn.LSTM(input_size=n_features,
                            hidden_size=hidden_size,
                            num_layers=num_layers)
        self.linear = nn.Linear(in_features=hidden_size,
                               out_features=n_features)

    def forward(self, x, hidden_state=None):
        """
        x: shape (seq_len, batch, input_size)
        h: of shape (num_layers * num_directions, batch, hidden_size)
        c: of shape (num_layers * num_directions, batch, hidden_size)
        output: shape (seq_len, batch, num_directions * hidden_size)
        """
        output, (h, c) = self.lstm(x, hidden_state)
        return self.linear(output), (h.detach(), c.detach())

```

```

### Create input data for LSTM, with sequence length 16 (months)
seq_len = 16
train_exs = [mdata.iloc[i-(seq_len+1):i].values
             for i in range(seq_len+1, len(mdata))]
n_features = mdata.shape[1]

```

```

### Train LSTM
hidden_factors = dict()
prediction_errors = dict()

```

(continues on next page)

(continued from previous page)

```

for hidden_size in [1, 2, 3, 4]:
    model = LSTM(n_features=n_features,
                 hidden_size=hidden_size).to(device)
    print(model)

# Set optimizer and learning rate scheduler, with step_size=30
lr, num_lr, step_size = 0.001, 3, 400
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()

batch_size, num_epochs = 32, step_size*num_lr
for i in tqdm(range(num_epochs)):    # Run training loop per epoch
    idx = np.arange(len(train_exs))  # shuffle idxs into batches
    random.shuffle(idx)
    batches = [idx[i:(i+batch_size)] for i in range(0, len(idx), batch_size)]
    total_loss = 0.0
    model.train()
    for batch in batches:    # train each batch
        # train_ex input has shape (seq_len, batch_size=16, n_features)
        nparray = np.array([[train_exs[idx][seq] for idx in batch]
                           for seq in range(seq_len+1)])
        train_ex = torch.tensor(nparray).float()
        model.zero_grad()
        y_pred, hidden_state = model.forward(train_ex[:-1].to(device))
        loss = loss_function(y_pred[-1],
                             train_ex[-1].to(device))
        total_loss += float(loss)
        loss.backward()
        optimizer.step()
    scheduler.step()

# collect predictions and hidden states, and compute mse
with torch.no_grad():    # reduce memory consumption for eval
    hidden_state = []
    prediction_error = []
    mse = nn.MSELoss()
    for i in range(seq_len+1, len(mdata)):
        # single test example shaped (seq_len=12, batch_size=1, n_features)
        test_ex = torch.tensor(mdata[i-(seq_len+1):i].values) \
            .float() \
            .unsqueeze(dim=1) \
            .to(device)
        y_pred, (h, c) = model.forward(test_ex[:-1], None)
        prediction_error.append(float(mse(y_pred[-1], test_ex[-1])))
        hidden_state.append(h[0][0].cpu().numpy())
    hidden_factors[hidden_size] = DataFrame(hidden_state,
                                             index=mdata.index \
                                             [(1+seq_len):len(mdata)])
prediction_errors[f"Hidden Size {hidden_size}"] = np.mean(prediction_error)
print(prediction_errors)

```

```
LSTM(
    (lstm): LSTM(123, 1)
    (linear): Linear(in_features=1, out_features=123, bias=True)
)
```

100% |██████████| 1200/1200 [00:49<00:00, 24.44it/s]

```
{'Hidden Size 1': 0.8803263414237235}
LSTM(
    (lstm): LSTM(123, 2)
    (linear): Linear(in_features=2, out_features=123, bias=True)
)
```

100% |██████████| 1200/1200 [00:48<00:00, 24.60it/s]

```
{'Hidden Size 1': 0.8803263414237235, 'Hidden Size 2': 0.812816398053767}
LSTM(
    (lstm): LSTM(123, 3)
    (linear): Linear(in_features=3, out_features=123, bias=True)
)
```

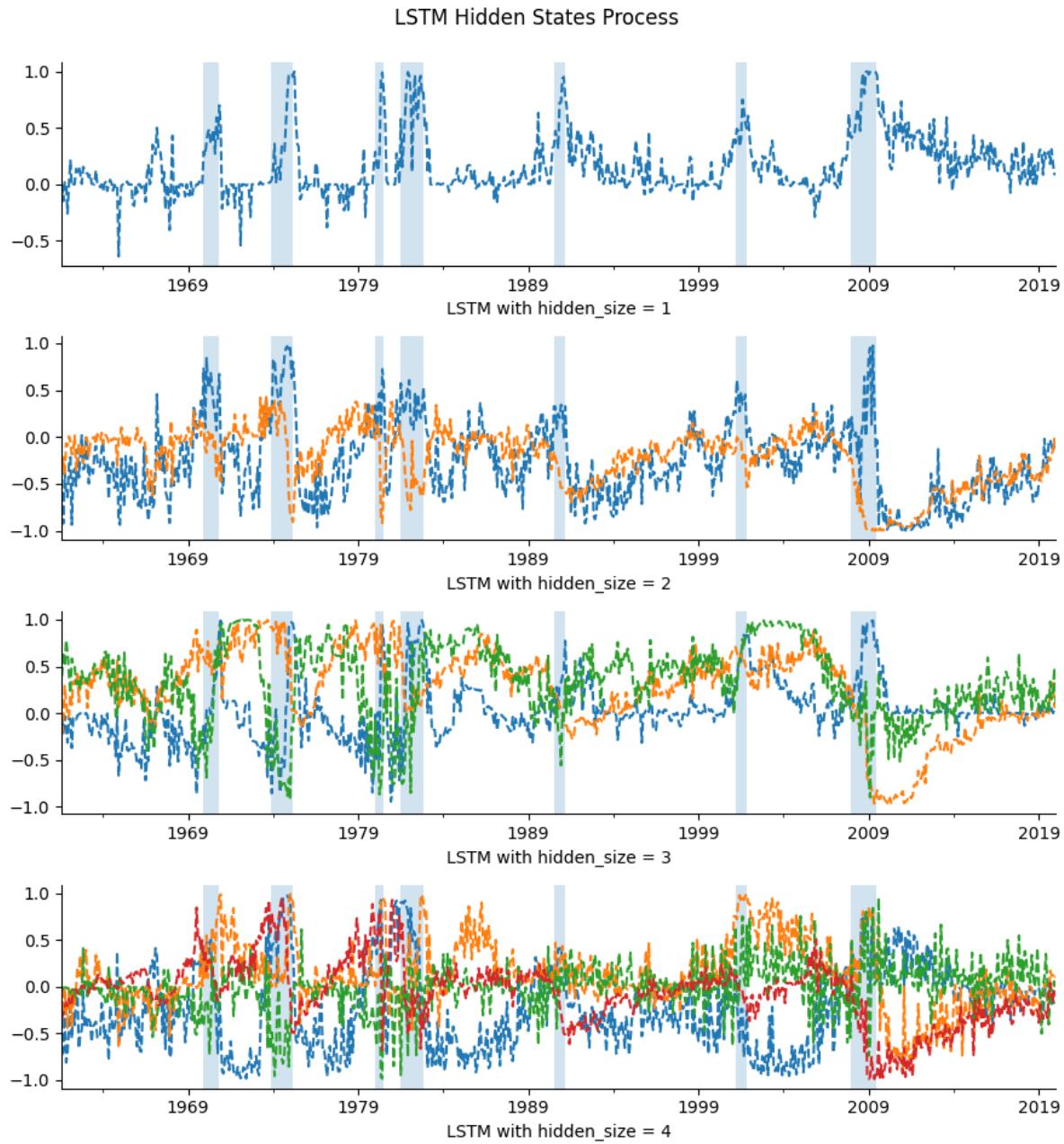
100% |██████████| 1200/1200 [00:45<00:00, 26.51it/s]

```
{'Hidden Size 1': 0.8803263414237235, 'Hidden Size 2': 0.812816398053767, 'Hidden
  ↵Size 3': 0.7557395131796854}
LSTM(
    (lstm): LSTM(123, 4)
    (linear): Linear(in_features=4, out_features=123, bias=True)
)
```

100% |██████████| 1200/1200 [00:44<00:00, 26.87it/s]

```
{'Hidden Size 1': 0.8803263414237235, 'Hidden Size 2': 0.812816398053767, 'Hidden
  ↵Size 3': 0.7557395131796854, 'Hidden Size 4': 0.7086005449889392}
```

```
## Plot LSTM hidden states sequence
fig, axes = plt.subplots(len(hidden_factors), 1, figsize=(9,10),
                       num=1, clear=True)
for hidden_factor, ax in zip(hidden_factors.values(), axes):
    hidden_factor.plot(ax=ax, style='--', legend=False)
    for a,b in vspans:
        if a >= min(hidden_factor.index):
            ax.axvspan(a, min(b, max(hidden_factor.index)), alpha=0.2)
    ax.spines["top"].set_visible(False)
    ax.spines["right"].set_visible(False)
    ax.set_xlabel(f'LSTM with hidden_size = {len(hidden_factor.columns)}')
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.suptitle(f'LSTM Hidden States Process', fontsize=12)
plt.savefig(imgdir / 'lstm.jpg')
```



```
## Dynamic Factor Models
dynamic_factors = dict()
for i in [1, 2, 3, 4]:
    mod = sm.tsa.DynamicFactorMQ(endog=mdata,
                                  factors=1,                      # num factor blocks
                                  factor_multiplicities=i,          # num factors in block
                                  factor_orders=2,                  # order of factor VAR
                                  idiosyncratic_ar1=False)          # False=white noise
    fitted = mod.fit_em(disp=20,
                         maxiter=200,
                         full_output=True)
    dynamic_factors[i] = DataFrame(fitted.factors.filtered.iloc[seq_len+1:])
    dynamic_factors[i].columns = list(range(len(dynamic_factors[i].columns)))

```

(continues on next page)

(continued from previous page)

```

mse = nn.MSELoss()
prediction_errors[f"Dynamic Factors {i}"] = float(
    mse(torch.tensor(fitted.fittedvalues.iloc[mod.factor_orders+1:]).values,
         torch.tensor(mdata.iloc[mod.factor_orders+1:]).values)))
#print(fitted.summary(0))

```

```

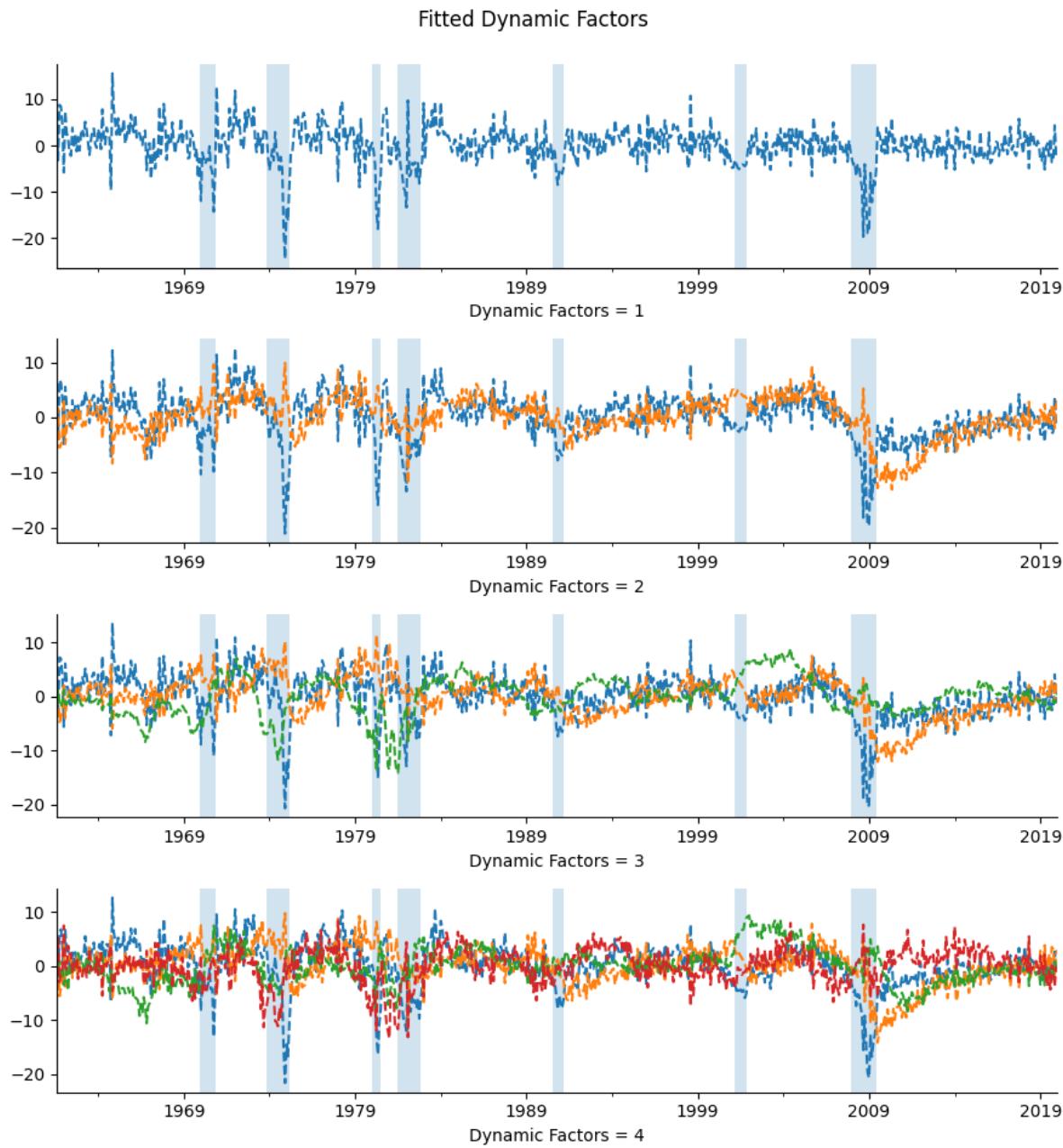
EM start iterations, llf=-1.1843e+05
EM iteration 20, llf=-1.1774e+05, convergence criterion=3.1004e-05
EM converged at iteration 29, llf=-1.1773e+05, convergence criterion=9.4463e-07 <-
    ↪tolerance=1e-06
EM start iterations, llf=-1.1327e+05
EM iteration 20, llf=-1.1147e+05, convergence criterion=7.3906e-06
EM iteration 40, llf=-1.1146e+05, convergence criterion=2.7273e-06
EM iteration 60, llf=-1.1146e+05, convergence criterion=1.0928e-06
EM converged at iteration 62, llf=-1.1146e+05, convergence criterion=9.9257e-07 <-
    ↪tolerance=1e-06
EM start iterations, llf=-1.0902e+05
EM iteration 20, llf=-1.062e+05, convergence criterion=1.8777e-05
EM iteration 40, llf=-1.0617e+05, convergence criterion=3.7322e-06
EM converged at iteration 56, llf=-1.0617e+05, convergence criterion=9.9476e-07 <-
    ↪tolerance=1e-06
EM start iterations, llf=-1.0478e+05
EM iteration 20, llf=-1.0301e+05, convergence criterion=2.4994e-05
EM iteration 40, llf=-1.0299e+05, convergence criterion=4.3864e-06
EM iteration 60, llf=-1.0298e+05, convergence criterion=2.0342e-06
EM iteration 80, llf=-1.0298e+05, convergence criterion=1.0751e-06
EM converged at iteration 83, llf=-1.0298e+05, convergence criterion=9.7809e-07 <-
    ↪tolerance=1e-06

```

```

### Plot dynamic factors
fig, axes = plt.subplots(len(dynamic_factors), 1, figsize=(9,10), num=1, clear=True)
for dynamic_factor, ax in zip(dynamic_factors.values(), axes):
    dynamic_factor.plot(ax=ax, style='--', legend=False)
    for a,b in vspans:
        if a >= min(dynamic_factor.index):
            ax.axvspan(a, min(b, max(dynamic_factor.index)), alpha=0.2)
    ax.spines["top"].set_visible(False)
    ax.spines["right"].set_visible(False)
    ax.set_xlabel(f"Dynamic Factors = {len(dynamic_factor.columns)}")
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.suptitle(f"\"Fitted Dynamic Factors\"", fontsize=12)
plt.savefig(imgdir / 'dynamic.jpg')

```



```
## Correlation of LSTM hidden state process with (linear) dynamic factors
rsq = dict()
for k, hidden_factor in hidden_factors.items():
    rsq[k] = [sm.OLS(y, sm.add_constant(dynamic_factors[len(dynamic_factors)]))\
              .fit().rsquared for _, y in hidden_factor.iteritems()]
print('Average variance of LSTM hidden states explained by dynamic factors')
DataFrame({k: np.mean(r) for k, r in rsq.items()},\
          index=['R-square']).rename_axis("# hidden states in LSTM:", axis=1)
```

AttributeError

Traceback (most recent call last)

/tmp/ipykernel\_279548/1932558065.py in ?()

(continues on next page)

(continued from previous page)

```

1 ## Correlation of LSTM hidden state process with (linear) dynamic factors
2 rsq = dict()
3 for k, hidden_factor in hidden_factors.items():
4     rsq[k] = [sm.OLS(y, sm.add_constant(dynamic_factors[len(dynamic_
+ factors)]))\
5                 .fit().rsquared for _, y in hidden_factor.iteritems()]
6 print('Average variance of LSTM hidden states explained by dynamic factors
+')
7 DataFrame({k: np.mean(r) for k, r in rsq.items()},
8             index=['R-square']).rename_axis("# hidden states in LSTM:",_
+axis=1)

~/env3.11/lib/python3.11/site-packages/pandas/core/generic.py in ?(self, name)
5985         and name not in self._accessors
5986         and self._info_axis._can_hold_identifiers_and_holds_name(name)
5987     ):
5988         return self[name]
-> 5989     return object.__getattribute__(self, name)

AttributeError: 'DataFrame' object has no attribute 'iteritems'

```

```

## Mixed Frequency Dynamic Factor Model
scaler = StandardScaler()\
    .fit(qdata['GDPC1'].values.reshape((-1, 1)))
gdp = DataFrame(scaler.transform(qdata['GDPC1'].values.reshape((-1, 1))),
    index = qdata.index, columns=['GDPC1'])

```

```

mod = sm.tsa.DynamicFactorMQ(endog=mdata,
                               endog_quarterly=gdp,
                               factors=1,                      # num factor blocks
                               factor_multiplicities=8,          # num factors in block
                               factor_orders=2,                 # order of factor VAR
                               idiosyncratic_ar1=False)         # False=white noise
fitted = mod.fit_em(disp=1, maxiter=200, full_output=True)
dynamic_factor = DataFrame(fitted.factors.filtered.iloc[seq_len+1:])
dynamic_factor.columns = list(np.arange(len(dynamic_factor.columns)))

```

### Plot fitted GDP values in 2007-2010

```

# TODO: fit through current
beg = '2006-12-31'
end = '2010-12-31'
fig, ax = plt.subplots(figsize=(9,5), num=1, clear=True)
y = fitted.fittedvalues['GDPC1']
y = y[(y.index > beg) & (y.index <= end)]
ax.plot_date(y.index,
              (scaler.inverse_transform(y.to_numpy()).reshape(-1,1))/3).cumsum(),
              fmt='-_o',
              color='C0')
x = gdp.copy()
x.index = pd.DatetimeIndex(x.index.astype(str), freq=None)
x = x[(x.index > beg) & (x.index <= end)]
ax.plot_date(x.index, scaler.inverse_transform(x).cumsum(), fmt='-_o', color='C1')
ax.legend(['monthly fitted', 'quarterly actual'], loc='upper left')

```

(continues on next page)

(continued from previous page)

```
ax.set_title('Quarterly GDP and Monthly Estimates from Dynamic Factor Model')
plt.tight_layout()
plt.savefig(imgdir / "mixedfreq.jpg")
```

```
# Show "Nowcast" of GDP
# Series({'Last Date of Monthly Data': mdata.index[-1].strftime('%Y-%m-%d'),
#          'Last Date of Quarterly Data': qdata.index[-1].strftime('%Y-%m-%d'),
#          'Forecast of Q1 GDP quarterly rate':
#              scaler.inverse_transform(fitted.forecast('2020-03') ['GDPC1'] [[-1]]) [0],
#          'Forecast of Q2 GDP quarterly rate':
#              scaler.inverse_transform(fitted.forecast('2020-06') ['GDPC1'] [[-1]]) [0]},
#          name = 'Forecast').to_frame()
```



## LANGUAGE MODELLING

### UNDER CONSTRUCTION

- Language modelling of FOMC meeting minutes

```
import numpy as np
import pandas as pd
from pandas import Series, DataFrame
import os
import matplotlib.pyplot as plt
from finds.database.mongodb import MongoDB
from finds.unstructured import Unstructured
from finds.unstructured.store import Store
from finds.readers.fomcreader import FOMCReader
from secret import credentials, paths
# %matplotlib qt
VERBOSE = 1      # 0
SHOW = dict(ndigits=4, latex=True)  # None
```

```
mongodb = MongoDB(**credentials['mongodb'])
fomc = Unstructured(mongodb, 'FOMC')
imgdir = paths['images'] / 'fomc'
```

```
{'host': 'omen3080', 'version': '5.0.20', 'process': 'mongod', 'pid': 4153196,
 'uptime': 931.0, 'uptimeMillis': 931239, 'uptimeEstimate': 931, 'localTime': -
 datetime.datetime(2023, 8, 31, 18, 13, 46, 746000), 'asserts': {'regular': 0,
 'warning': 0, 'msg': 0, 'user': 19, 'tripwire': 0, 'rollovers': 0}, 'catalogStats':
 {'collections': 7, 'capped': 0, 'timeseries': 0, 'views': 0,
 'internalCollections': 3, 'internalViews': 0}, 'connections': {'current': 6,
 'available': 51194, 'totalCreated': 8, 'active': 3, 'threaded': 6,
 'exhaustIsMaster': 0, 'exhaustHello': 1, 'awaitingTopologyChanges': 2},
 'electionMetrics': {'stepUpCmd': {'called': 0, 'successful': 0},
 'priorityTakeover': {'called': 0, 'successful': 0}, 'catchUpTakeover': {'called':
 0, 'successful': 0}, 'electionTimeout': {'called': 0, 'successful': 0},
 'freezeTimeout': {'called': 0, 'successful': 0}, 'numStepDownsCausedByHigherTerm':
 0, 'numCatchUps': 0, 'numCatchUpsSucceeded': 0, 'numCatchUpsAlreadyCaughtUp': -
 0, 'numCatchUpsSkipped': 0, 'numCatchUpsTimedOut': 0, 'numCatchUpsFailedWithError':
 0, 'numCatchUpsFailedWithNewTerm': 0, 'numCatchUpsFailedWithReplSetAbortPrimaryCatchUpCmd': 0,
 'averageCatchUpOps': 0.0}, 'extra_info': {'note': 'fields vary by platform', 'user_time_us': 3076896,
 'system_time_us': 740900, 'maximum_resident_set_kb': 137284, 'input_blocks': -
 131336, 'output_blocks': 8056, 'page_reclaims': 22026, 'page_faults': 622,
 'voluntary_context_switches': 34419, 'involuntary_context_switches': 250},
```

(continues on next page)

(continued from previous page)

```
'flowControl': {'enabled': True, 'targetRateLimit': 10000000000, 'timeAcquiringMicros': 51, 'locksPerKiloOp': 0.0, 'sustainerRate': 0, 'isLagged': False, 'isLaggedCount': 0, 'isLaggedTimeMicros': 0}, 'freeMonitoring': {'state': 'undecided'}, 'globalLock': {'totalTime': 931307000, 'currentQueue': {'total': 0, 'readers': 0, 'writers': 0}, 'activeClients': {'total': 0, 'readers': 0, 'writers': 0}}, 'indexBulkBuilder': {'count': 0, 'resumed': 0, 'filesOpenedForExternalSort': 0, 'filesClosedForExternalSort': 0}, 'indexStats': {'count': 8, 'features': {'2d': {'count': 0, 'accesses': 0}, '2dsphere': {'count': 0, 'accesses': 0}, 'collation': {'count': 0, 'accesses': 0}, 'compound': {'count': 0, 'accesses': 0}, 'hashed': {'count': 0, 'accesses': 0}, 'id': {'count': 7, 'accesses': 0}, 'normal': {'count': 1, 'accesses': 0}, 'partial': {'count': 0, 'accesses': 0}, 'single': {'count': 1, 'accesses': 0}, 'sparse': {'count': 0, 'accesses': 0}, 'text': {'count': 0, 'accesses': 0}, 'ttl': {'count': 0, 'accesses': 0}, 'unique': {'count': 1, 'accesses': 0}, 'wildcard': {'count': 0, 'accesses': 0}}, 'locks': {'ParallelBatchWriterMode': {'acquireCount': {'r': 37}}, 'FeatureCompatibilityVersion': {'acquireCount': {'r': 1887, 'w': 33}}, 'ReplicationStateTransition': {'acquireCount': {'w': 972}}, 'Global': {'acquireCount': {'r': 1887, 'w': 28, 'W': 5}}, 'Database': {'acquireCount': {'r': 8, 'w': 27, 'R': 1, 'W': 1}}, 'Collection': {'acquireCount': {'r': 16, 'w': 25, 'W': 2}}, 'Mutex': {'acquireCount': {'r': 53}}, 'logicalSessionRecordCache': {'activeSessionsCount': 1, 'sessionsCollectionJobCount': 4, 'lastSessionsCollectionJobDurationMillis': 0, 'lastSessionsCollectionJobTimestamp': datetime.datetime(2023, 8, 31, 18, 13, 15, 977000), 'lastSessionsCollectionJobEntriesRefreshed': 0, 'lastSessionsCollectionJobEntriesEnded': 0, 'lastSessionsCollectionJobCursorsClosed': 0, 'transactionReaperJobCount': 4, 'lastTransactionReaperJobDurationMillis': 0, 'lastTransactionReaperJobTimestamp': datetime.datetime(2023, 8, 31, 18, 13, 15, 977000), 'lastTransactionReaperJobEntriesCleanedUp': 0, 'sessionCatalogSize': 0}, 'network': {'bytesIn': 276721, 'bytesOut': 9479574, 'physicalBytesIn': 264527, 'physicalBytesOut': 9479574, 'numSlowDNSOperations': 0, 'numSlowSSLOperations': 0, 'numRequests': 204, 'tcpFastOpen': {'kernelSetting': 1, 'serverSupported': True, 'clientSupported': True, 'accepted': 0}, 'compression': {'snappy': {'compressor': {'bytesIn': 0, 'bytesOut': 0}, 'decompressor': {'bytesIn': 0, 'bytesOut': 0}}, 'zstd': {'compressor': {'bytesIn': 0, 'bytesOut': 0}, 'decompressor': {'bytesIn': 0, 'bytesOut': 0}}, 'zlib': {'compressor': {'bytesIn': 0, 'bytesOut': 0}, 'decompressor': {'bytesIn': 0, 'bytesOut': 0}}}, 'serviceExecutors': {'passthrough': {'threadsRunning': 6, 'clientsInTotal': 6, 'clientsRunning': 6, 'clientsWaitingForData': 0}, 'fixed': {'threadsRunning': 1, 'clientsInTotal': 0, 'clientsRunning': 0, 'clientsWaitingForData': 0}}, 'opLatencies': {'reads': {'latency': 17995, 'ops': 3}, 'writes': {'latency': 2074, 'ops': 6}, 'commands': {'latency': 12921, 'ops': 192}, 'transactions': {'latency': 0, 'ops': 0}, 'opcounters': {'insert': 6, 'query': 5, 'update': 2, 'delete': 0, 'getmore': 1, 'command': 204}, 'opcountersRepl': {'insert': 0, 'query': 0, 'update': 0, 'delete': 0, 'getmore': 0, 'command': 0}, 'readConcernCounters': {'nonTransactionOps': {'none': 2, 'noneInfo': {'CWRC': {'local': 0, 'available': 0, 'majority': 0}, 'implicitDefault': {'local': 2, 'available': 0}}, 'local': 0, 'available': 0, 'majority': 0, 'snapshot': {'withClusterTime': 0, 'withoutClusterTime': 0}, 'linearizable': 0}, 'transactionOps': {'none': 0, 'noneInfo': {'CWRC': {'local': 0, 'majority': 0}, 'implicitDefault': {'local': 0}}, 'local': 0, 'majority': 0, 'snapshot': {'withClusterTime': 0, 'withoutClusterTime': 0}}, 'scramCache': {'SCRAM-SHA-1': {'count': 0, 'hits': 0, 'misses': 0}, 'SCRAM-SHA-256': {'count': 0, 'hits': 0, 'misses': 0}}, 'security': {'authentication': {'saslSupportedMechsReceived': 0, 'mechanisms': {'MONGODB-X509': {'speculativeAuthenticate': {'received': 0, 'successful': 0}, 'clusterAuthenticate': {'received': 0, 'successful': 0}}}}}}
```

(continues on next page)

(continued from previous page)

```

← 'authenticate': {'received': 0, 'successful': 0}}, 'SCRAM-SHA-1': {
← 'speculativeAuthenticate': {'received': 0, 'successful': 0}, 'clusterAuthenticate':
← ': {'received': 0, 'successful': 0}, 'authenticate': {'received': 0, 'successful':
← ': 0}}, 'SCRAM-SHA-256': {'speculativeAuthenticate': {'received': 0, 'successful':
← ': 0}, 'clusterAuthenticate': {'received': 0, 'successful': 0}, 'authenticate': {
← 'received': 0, 'successful': 0}}}}}, 'storageEngine': {'name': 'wiredTiger',
← 'supportsCommittedReads': True, 'oldestRequiredTimestampForCrashRecovery':_
← Timestamp(0, 0), 'supportsPendingDrops': True, 'dropPendingIdents': 0,
← 'supportsSnapshotReadConcern': True, 'readOnly': False, 'persistent': True,
← 'backupCursorOpen': False, 'supportsResumableIndexBuilds': True}, 'tcmalloc': {
← 'generic': {'current_allocated_bytes': 108290160, 'heap_size': 151408640},
← 'tcmalloc': {'pageheap_free_bytes': 41582592, 'pageheap_unmapped_bytes': 0, 'max_
← total_thread_cache_bytes': 1073741824, 'current_total_thread_cache_bytes':_
← 1049944, 'total_free_bytes': 1535888, 'central_cache_free_bytes': 329656,
← 'transfer_cache_free_bytes': 156288, 'thread_cache_free_bytes': 1049944,
← 'aggressive_memory_decommit': 0, 'pageheap_committed_bytes': 151408640,
← 'pageheap_scavenge_count': 0, 'pageheap_commit_count': 70, 'pageheap_total_
← commit_bytes': 151408640, 'pageheap_decommit_count': 0, 'pageheap_total_decommit_
← bytes': 0, 'pageheap_reserve_count': 70, 'pageheap_total_reserve_bytes':_
← 151408640, 'spinlock_total_delay_ns': 0, 'release_rate': 1.0, 'formattedString':_
← '-----\nMALLOC: 108290736 ( 103.
← 3 MiB) Bytes in use by application\nMALLOC: + 41582592 ( 39.7 MiB) Bytes_
← in page heap freelist\nMALLOC: + 329656 ( 0.3 MiB) Bytes in central_
← cache freelist\nMALLOC: + 156288 ( 0.1 MiB) Bytes in transfer cache_
← freelist\nMALLOC: + 1049368 ( 1.0 MiB) Bytes in thread cache freelists\
← \nMALLOC: + 4849664 ( 4.6 MiB) Bytes in malloc metadata\nMALLOC: -----
← \nMALLOC: = 156258304 ( 149.0 MiB) Actual memory used (physical + swap)\\
← \nMALLOC: + 0 ( 0.0 MiB) Bytes released to OS (aka unmapped)\\
← \nMALLOC: ----- \nMALLOC: = 156258304 ( 149.0 MiB) Virtual address_
← \nMALLOC: 1007 Spans in use\nMALLOC: 36 Thread heaps in use\nMALLOC: 4096
← Tcmalloc page size\n-----\nCall_
← ReleaseFreeMemory() to release freelist memory to the OS (via madvise()).\nBytes_
← released to the OS take up virtual address space but no physical memory.\n'},
← 'tenantMigrations': {'currentMigrationsDonating': 0, 'currentMigrationsReceiving':
← ': 0, 'totalSuccessfulMigrationsDonated': 0, 'totalSuccessfulMigrationsReceived':
← ': 0, 'totalFailedMigrationsDonated': 0, 'totalFailedMigrationsReceived': 0},
← 'trafficRecording': {'running': False}, 'transactions': {'retriedCommandsCount':_
← 0, 'retriedStatementsCount': 0, 'transactionsCollectionWriteCount': 0,
← 'currentActive': 0, 'currentInactive': 0, 'currentOpen': 0, 'totalAborted': 0,
← 'totalCommitted': 0, 'totalStarted': 0, 'totalPrepared': 0,
← 'totalPreparedThenCommitted': 0, 'totalPreparedThenAborted': 0, 'currentPrepared':
← ': 0}, 'transportSecurity': {'1.0': 0, '1.1': 0, '1.2': 0, '1.3': 0, 'unknown':_
← 0}, 'twoPhaseCommitCoordinator': {'totalCreated': 0, 'totalStartedTwoPhaseCommit':
← ': 0, 'totalAbortedTwoPhaseCommit': 0, 'totalCommittedTwoPhaseCommit': 0,
← 'currentInSteps': {'writingParticipantList': 0, 'waitingForVotes': 0,
← 'writingDecision': 0, 'waitingForDecisionAcks': 0, 'deletingCoordinatorDoc': 0}},
← 'wiredTiger': {'uri': 'statistics:', 'block-manager': {'block cache cached_
← blocks updated': 0, 'block cache cached bytes updated': 0, 'block cache evicted_
← blocks': 0, 'block cache file size causing bypass': 0, 'block cache lookups': 0,
← 'block cache number of blocks not evicted due to overhead': 0, 'block cache_
← number of bypasses because no-write-allocate setting was on': 0, 'block cache_
← number of bypasses due to overhead on put': 0, 'block cache number of bypasses_
← on get': 0, 'block cache number of bypasses on put because file is too small': 0,
← 'block cache number of eviction passes': 0, 'block cache number of hits':_
← including existence checks': 0, 'block cache number of misses including':_

```

(continues on next page)

(continued from previous page)

```
↳existence checks': 0, 'block cache number of put bypasses on checkpoint I/O': 0,  
↳'block cache removed blocks': 0, 'block cache total blocks': 0, 'block cache  
↳total blocks inserted on read path': 0, 'block cache total blocks inserted on  
↳write path': 0, 'block cache total bytes': 0, 'block cache total bytes inserted  
↳on read path': 0, 'block cache total bytes inserted on write path': 0, 'blocks  
↳pre-loaded': 107, 'blocks read': 154, 'blocks written': 126, 'bytes read':  
↳4685824, 'bytes read via memory map API': 0, 'bytes read via system call API': 0,  
↳'bytes written': 1167360, 'bytes written for checkpoint': 1167360, 'bytes  
↳written via memory map API': 0, 'bytes written via system call API': 0, 'mapped  
↳blocks read': 0, 'mapped bytes read': 0, 'number of times the file was remapped  
↳because it changed size via mallocate or truncate': 0, 'number of times the  
↳region was remapped via write': 0}, 'cache': {'application threads page read  
↳from disk to cache count': 105, 'application threads page read from disk to  
↳cache time (usecs)': 11367, 'application threads page write from cache to disk  
↳count': 62, 'application threads page write from cache to disk time (usecs)':  
↳20803, 'bytes allocated for updates': 307071, 'bytes belonging to page images in  
↳the cache': 9918765, 'bytes belonging to the history store table in the cache':  
↳588, 'bytes currently in the cache': 10253408, 'bytes dirty in the cache'  
↳cumulative': 986248, 'bytes not belonging to page images in the cache': 334643,  
↳'bytes read into cache': 9184042, 'bytes written from cache': 907453, 'cache  
↳overflow score': 0, 'checkpoint blocked page eviction': 0, 'checkpoint of  
↳history store file blocked non-history store page eviction': 0, 'eviction calls  
↳to get a page': 29, 'eviction calls to get a page found queue empty': 26,  
↳'eviction calls to get a page found queue empty after locking': 0, 'eviction  
↳currently operating in aggressive mode': 0, 'eviction empty score': 0, 'eviction  
↳gave up due to detecting an out of order on disk value behind the last update on  
↳the chain': 0, 'eviction gave up due to detecting an out of order tombstone  
↳ahead of the selected on disk update': 0, 'eviction gave up due to detecting an  
↳out of order tombstone ahead of the selected on disk update after validating the  
↳update chain': 0, 'eviction gave up due to detecting out of order timestamps on  
↳the update chain after the selected on disk update': 0, 'eviction gave up due to  
↳needing to remove a record from the history store but checkpoint is running': 0,  
↳'eviction passes of a file': 0, 'eviction server candidate queue empty when  
↳topping up': 0, 'eviction server candidate queue not empty when topping up': 0,  
↳'eviction server evicting pages': 0, 'eviction server slept, because we did not  
↳make progress with eviction': 3, 'eviction server unable to reach eviction goal  
↳': 0, 'eviction server waiting for a leaf page': 0, 'eviction state': 64,  
↳'eviction walk most recent sleeps for checkpoint handle gathering': 0, 'eviction  
↳walk target pages histogram - 0-9': 0, 'eviction walk target pages histogram -  
↳10-31': 0, 'eviction walk target pages histogram - 128 and higher': 0, 'eviction  
↳walk target pages histogram - 32-63': 0, 'eviction walk target pages histogram -  
↳64-128': 0, 'eviction walk target pages reduced due to history store cache  
↳pressure': 0, 'eviction walk target strategy both clean and dirty pages': 0,  
↳'eviction walk target strategy only clean pages': 0, 'eviction walk target  
↳strategy only dirty pages': 0, 'eviction walks abandoned': 0, 'eviction walks  
↳gave up because they restarted their walk twice': 0, 'eviction walks gave up  
↳because they saw too many pages and found no candidates': 0, 'eviction walks  
↳gave up because they saw too many pages and found too few candidates': 0,  
↳'eviction walks reached end of tree': 0, 'eviction walks restarted': 0,  
↳'eviction walks started from root of tree': 0, 'eviction walks started from  
↳saved location in tree': 0, 'eviction worker thread active': 4, 'eviction worker  
↳thread created': 0, 'eviction worker thread evicting pages': 3, 'eviction worker  
↳thread removed': 0, 'eviction worker thread stable number': 0, 'files with  
↳active eviction walks': 0, 'files with new eviction walks started': 0, 'force re-  
tuning of eviction workers once in a while': 0, 'forced eviction - history store  
↳pages failed to evict while session has history store cursor open': 0, 'forced
```

(continues on next page)

(continued from previous page)

```

↳ eviction - history store pages selected while session has history store cursor
↳ 'open': 0, 'forced eviction - history store pages successfully evicted while
↳ session has history store cursor open': 0, 'forced eviction - pages evicted that
↳ were clean count': 0, 'forced eviction - pages evicted that were clean time
↳ '(usecs)': 0, 'forced eviction - pages evicted that were dirty count': 0, 'forced
↳ eviction - pages evicted that were dirty time (usecs)': 0, 'forced eviction -
↳ pages selected because of a large number of updates to a single item': 0,
↳ 'forced eviction - pages selected because of too many deleted items count': 0,
↳ 'forced eviction - pages selected count': 0, 'forced eviction - pages selected
↳ unable to be evicted count': 0, 'forced eviction - pages selected unable to be
↳ evicted time': 0, 'hazard pointer blocked page eviction': 0, 'hazard pointer
↳ check calls': 3, 'hazard pointer check entries walked': 5, 'hazard pointer
↳ maximum array length': 3, 'history store score': 0, 'history store table insert
↳ calls': 0, 'history store table insert calls that returned restart': 0, 'history
↳ store table max on-disk size': 0, 'history store table on-disk size': 4096,
↳ 'history store table out-of-order resolved updates that lose their durable
↳ timestamp': 0, 'history store table out-of-order updates that were fixed up by
↳ reinserting with the fixed timestamp': 0, 'history store table reads': 0,
↳ 'history store table reads missed': 0, 'history store table reads requiring
↳ squashed modifies': 0, 'history store table truncation by rollback to stable to
↳ remove an unstable update': 0, 'history store table truncation by rollback to
↳ stable to remove an update': 0, 'history store table truncation to remove an
↳ update': 0, 'history store table truncation to remove range of updates due to
↳ key being removed from the data page during reconciliation': 0, 'history store
↳ table truncation to remove range of updates due to out-of-order timestamp update
↳ on data page': 0, 'history store table writes requiring squashed modifies': 0,
↳ 'in-memory page passed criteria to be split': 0, 'in-memory page splits': 0,
↳ 'internal pages evicted': 0, 'internal pages queued for eviction': 0, 'internal
↳ pages seen by eviction walk': 0, 'internal pages seen by eviction walk that are
↳ already queued': 0, 'internal pages split during eviction': 0, 'leaf pages split
↳ during eviction': 0, 'maximum bytes configured': 16098787328, 'maximum
↳ milliseconds spent at a single eviction': 0, 'maximum page size seen at eviction
↳ ': 0, 'modified pages evicted': 3, 'modified pages evicted by application threads
↳ ': 0, 'operations timed out waiting for space in cache': 0, 'overflow pages read
↳ into cache': 0, 'page split during eviction deepened the tree': 0, 'page written
↳ requiring history store records': 0, 'pages currently held in the cache': 124,
↳ 'pages evicted by application threads': 0, 'pages evicted in parallel with
↳ checkpoint': 3, 'pages queued for eviction': 0, 'pages queued for eviction post
↳ lru sorting': 0, 'pages queued for urgent eviction': 3, 'pages queued for urgent
↳ eviction during walk': 0, 'pages queued for urgent eviction from history store
↳ due to high dirty content': 0, 'pages read into cache': 116, 'pages read into
↳ cache after truncate': 7, 'pages read into cache after truncate in prepare state
↳ ': 0, 'pages removed from the ordinary queue to be queued for urgent eviction': 0,
↳ 'pages requested from the cache': 965, 'pages seen by eviction walk': 0,
↳ 'pages seen by eviction walk that are already queued': 0, 'pages selected for
↳ eviction unable to be evicted': 0, 'pages selected for eviction unable to be
↳ evicted because of active children on an internal page': 0, 'pages selected for
↳ eviction unable to be evicted because of failure in reconciliation': 0, 'pages
↳ selected for eviction unable to be evicted because of race between checkpoint
↳ and out of order timestamps handling': 0, 'pages walked for eviction': 0, 'pages
↳ written from cache': 65, 'pages written requiring in-memory restoration': 0,
↳ 'percentage overhead': 8, 'the number of times full update inserted to history
↳ store': 0, 'the number of times reverse modify inserted to history store': 0,
↳ 'total milliseconds spent inside reentrant history store evictions in a
↳ reconciliation': 0, 'tracked bytes belonging to internal pages in the cache': 0,
↳ 12950, 'tracked bytes belonging to leaf pages in the cache': 10240458, 'tracked

```

(continues on next page)

(continued from previous page)

```
↳dirty bytes in the cache': 0, 'tracked dirty pages in the cache': 0, 'unmodified_
↳pages evicted': 0}, 'capacity': {'background fsync file handles considered': 0,
↳'background fsync file handles synced': 0, 'background fsync time (msecs)': 0,
↳'bytes read': 4530176, 'bytes written for checkpoint': 698601, 'bytes written_
↳for eviction': 0, 'bytes written for log': 1003804288, 'bytes written total':_
↳1004502889, 'threshold to call fsync': 0, 'time waiting due to total capacity_
↳(usecs)': 0, 'time waiting during checkpoint (usecs)': 0, 'time waiting during_
↳eviction (usecs)': 0, 'time waiting during logging (usecs)': 0, 'time waiting_
↳during read (usecs)': 0}, 'checkpoint-cleanup': {'pages added for eviction': 3,
↳'pages removed': 0, 'pages skipped during tree walk': 0, 'pages visited': 143},
↳'connection': {'auto adjusting condition resets': 63, 'auto adjusting condition_
↳wait calls': 5750, 'auto adjusting condition wait raced to update timeout and_
↳skipped updating': 0, 'detected system time went backwards': 0, 'files currently_
↳open': 16, 'hash bucket array size for data handles': 512, 'hash bucket array_
↳size general': 512, 'memory allocations': 42516, 'memory frees': 41098, 'memory_
↳re-allocations': 3128, 'pthread mutex condition wait calls': 15192, 'pthread_
↳mutex shared lock read-lock calls': 15410, 'pthread mutex shared lock write-lock_
↳calls': 1141, 'total fsync I/Os': 133, 'total read I/Os': 1193, 'total write I/Os_
↳': 190}, 'cursor': {'Total number of entries skipped by cursor next calls': 0,
↳'Total number of entries skipped by cursor prev calls': 0, 'Total number of_
↳entries skipped to position the history store cursor': 0, 'Total number of times_
↳a search near has exited due to prefix config': 0, 'cached cursor count': 19,
↳'cursor bulk loaded cursor insert calls': 0, 'cursor close calls that result in_
↳cache': 7705, 'cursor create calls': 126, 'cursor insert calls': 72, 'cursor_
↳insert key and value bytes': 284145, 'cursor modify calls': 0, 'cursor modify_
↳key and value bytes affected': 0, 'cursor modify value bytes modified': 0,
↳'cursor next calls': 828, 'cursor next calls that skip due to a globally visible_
↳history store tombstone': 0, 'cursor next calls that skip greater than or equal_
↳to 100 entries': 0, 'cursor next calls that skip less than 100 entries': 826,
↳'cursor operation restarted': 0, 'cursor prev calls': 33, 'cursor prev calls_
↳that skip due to a globally visible history store tombstone': 0, 'cursor prev_
↳calls that skip greater than or equal to 100 entries': 0, 'cursor prev calls_
↳that skip less than 100 entries': 33, 'cursor remove calls': 1, 'cursor remove_
↳key bytes removed': 12, 'cursor reserve calls': 0, 'cursor reset calls': 8255,
↳'cursor search calls': 314, 'cursor search history store calls': 0, 'cursor_
↳search near calls': 50, 'cursor sweep buckets': 11307, 'cursor sweep cursors_
↳closed': 0, 'cursor sweep cursors examined': 108, 'cursor sweeps': 1126, 'cursor_
↳truncate calls': 0, 'cursor update calls': 0, 'cursor update key and value bytes_
↳': 0, 'cursor update value size change': 0, 'cursors reused from cache': 7686,
↳'open cursor count': 6}, 'data-handle': {'connection data handle size': 440,
↳'connection data handles currently active': 25, 'connection sweep candidate_
↳became referenced': 0, 'connection sweep dhandles closed': 0, 'connection sweep_
↳dhandles removed from hash list': 17, 'connection sweep time-of-death sets': 155,
↳'connection sweeps': 93, 'connection sweeps skipped due to checkpoint gathering_
↳handles': 0, 'session dhandles swept': 11, 'session sweep attempts': 335}, 'lock_
↳': {'checkpoint lock acquisitions': 17, 'checkpoint lock application thread wait_
↳time (usecs)': 0, 'checkpoint lock internal thread wait time (usecs)': 0,
↳'dhandle lock application thread time waiting (usecs)': 0, 'dhandle lock_
↳internal thread time waiting (usecs)': 0, 'dhandle read lock acquisitions': 3804,
↳'dhandle write lock acquisitions': 63, 'durable timestamp queue lock_
↳application thread time waiting (usecs)': 0, 'durable timestamp queue lock_
↳internal thread time waiting (usecs)': 0, 'durable timestamp queue read lock_
↳acquisitions': 0, 'durable timestamp queue write lock acquisitions': 0,
↳'metadata lock acquisitions': 16, 'metadata lock application thread wait time_
↳(usecs)': 0, 'metadata lock internal thread wait time (usecs)': 0, 'read_
↳timestamp queue lock application thread time waiting (usecs)': 0, 'read_
↳timestamp queue lock internal thread wait time (usecs)': 0}
```

(continues on next page)

(continued from previous page)

```

timestamp queue lock internal thread time waiting (usecs)': 0, 'read timestamp
queue read lock acquisitions': 0, 'read timestamp queue write lock acquisitions
': 0, 'schema lock acquisitions': 44, 'schema lock application thread wait time
(usecs)': 0, 'schema lock internal thread wait time (usecs)': 0, 'table lock
application thread time waiting for the table lock (usecs)': 0, 'table lock
internal thread time waiting for the table lock (usecs)': 0, 'table read lock
acquisitions': 0, 'table write lock acquisitions': 13, 'txn global lock
application thread time waiting (usecs)': 0, 'txn global lock internal thread
time waiting (usecs)': 0, 'txn global read lock acquisitions': 66, 'txn global
write lock acquisitions': 46}, 'log': {'busy returns attempting to switch slots
': 0, 'force archive time sleeping (usecs)': 0, 'log bytes of payload data':_
135688, 'log bytes written': 142720, 'log files manually zero-filled': 0, 'log
flush operations': 9308, 'log force write operations': 10335, 'log force write
operations skipped': 10308, 'log records compressed': 14, 'log records not
compressed': 10, 'log records too small to compress': 69, 'log release advances
write LSN': 17, 'log scan operations': 6, 'log scan records requiring two reads
': 0, 'log server thread advances write LSN': 27, 'log server thread write LSN
walk skipped': 1917, 'log sync operations': 44, 'log sync time duration (usecs)
': 200004, 'log sync_dir operations': 1, 'log sync_dir time duration (usecs)':_
8968, 'log write operations': 93, 'logging bytes consolidated': 142208, 'maximum
log file size': 104857600, 'number of pre-allocated log files to create': 2,
'pre-allocated log files not ready and missed': 1, 'pre-allocated log files
prepared': 2, 'pre-allocated log files used': 0, 'records processed by log scan
': 15, 'slot close lost race': 0, 'slot close unbuffered waits': 0, 'slot
closures': 44, 'slot join atomic update races': 0, 'slot join calls atomic
updates raced': 0, 'slot join calls did not yield': 93, 'slot join calls found
active slot closed': 0, 'slot join calls slept': 0, 'slot join calls yielded': 0,
'slot join found active slot closed': 0, 'slot joins yield time (usecs)': 0,
'slot transitions unable to find free slot': 0, 'slot unbuffered writes': 0,
'total in-memory size of compressed records': 279252, 'total log buffer size':_
33554432, 'total size of compressed records': 127908, 'written slots coalesced':_
0, 'yields waiting for previous log file close': 0}, 'perf': {'file system read
latency histogram (bucket 1) - 10-49ms': 0, 'file system read latency histogram
(bucket 2) - 50-99ms': 0, 'file system read latency histogram (bucket 3) - 100-
249ms': 0, 'file system read latency histogram (bucket 4) - 250-499ms': 0, 'file
system read latency histogram (bucket 5) - 500-999ms': 0, 'file system read
latency histogram (bucket 6) - 1000ms+'. 0, 'file system write latency histogram
(bucket 1) - 10-49ms': 0, 'file system write latency histogram (bucket 2) - 50-
99ms': 0, 'file system write latency histogram (bucket 3) - 100-249ms': 0, 'file
system write latency histogram (bucket 4) - 250-499ms': 0, 'file system write
latency histogram (bucket 5) - 500-999ms': 0, 'file system write latency
histogram (bucket 6) - 1000ms+'. 0, 'operation read latency histogram (bucket 1)-
100-249us': 0, 'operation read latency histogram (bucket 2) - 250-499us': 0,
'operation read latency histogram (bucket 3) - 500-999us': 0, 'operation read
latency histogram (bucket 4) - 1000-9999us': 0, 'operation read latency
histogram (bucket 5) - 10000us+'. 0, 'operation write latency histogram (bucket
1) - 100-249us': 0, 'operation write latency histogram (bucket 2) - 250-499us':_
0, 'operation write latency histogram (bucket 3) - 500-999us': 0, 'operation
write latency histogram (bucket 4) - 1000-9999us': 0, 'operation write latency
histogram (bucket 5) - 10000us+'. 0, 'reconciliation': {'approximate byte size
of timestamps in pages written': 0, 'approximate byte size of transaction IDs in
pages written': 336, 'fast-path pages deleted': 0, 'leaf-page overflow keys': 0,
'maximum milliseconds spent in a reconciliation call': 0, 'maximum milliseconds
spent in building a disk image in a reconciliation': 0, 'maximum milliseconds
spent in moving updates to the history store in a reconciliation': 0, 'page
reconciliation calls': 71, 'page reconciliation calls for eviction': 3, 'page

```

(continues on next page)

(continued from previous page)

```
↳ reconciliation calls that resulted in values with prepared transaction metadata
↳ ': 0, 'page reconciliation calls that resulted in values with timestamps': 0,
↳ 'page reconciliation calls that resulted in values with transaction ids': 17,
↳ 'pages deleted': 9, 'pages written including an aggregated newest start durable'
↳ 'timestamp ': 0, 'pages written including an aggregated newest stop durable'
↳ 'timestamp ': 0, 'pages written including an aggregated newest stop timestamp ': 0
↳ 0, 'pages written including an aggregated newest stop transaction ID': 0, 'pages
↳ written including an aggregated newest transaction ID ': 0, 'pages written
↳ including an aggregated oldest start timestamp ': 0, 'pages written including an
↳ aggregated prepare': 0, 'pages written including at least one prepare state': 0,
↳ 'pages written including at least one start durable timestamp': 0, 'pages
↳ written including at least one start timestamp': 0, 'pages written including at
↳ least one start transaction ID': 17, 'pages written including at least one stop
↳ durable timestamp': 0, 'pages written including at least one stop timestamp': 0,
↳ 'pages written including at least one stop transaction ID': 0, 'records written
↳ including a prepare state': 0, 'records written including a start durable
↳ timestamp': 0, 'records written including a start timestamp': 0, 'records
↳ written including a start transaction ID': 42, 'records written including a stop
↳ durable timestamp': 0, 'records written including a stop timestamp': 0, 'records
↳ written including a stop transaction ID': 0, 'split bytes currently awaiting free
↳ ': 0, 'split objects currently awaiting free': 0}, 'session': {'attempts to
↳ remove a local object and the object is in use': 0, 'flush_tier operation calls
↳ ': 0, 'local objects removed': 0, 'open session count': 15, 'session query
↳ timestamp calls': 0, 'table alter failed calls': 0, 'table alter successful calls
↳ ': 0, 'table alter triggering checkpoint calls': 0, 'table alter unchanged and
↳ skipped': 0, 'table compact failed calls': 0, 'table compact failed calls due to
↳ cache pressure': 0, 'table compact running': 0, 'table compact skipped as
↳ process would not reduce file size': 0, 'table compact successful calls': 0,
↳ 'table compact timeout': 0, 'table create failed calls': 0, 'table create
↳ successful calls': 1, 'table drop failed calls': 0, 'table drop successful calls
↳ ': 0, 'table rename failed calls': 0, 'table rename successful calls': 0, 'table
↳ salvage failed calls': 0, 'table salvage successful calls': 0, 'table truncate
↳ failed calls': 0, 'table truncate successful calls': 0, 'table verify failed
↳ calls': 0, 'table verify successful calls': 0, 'tiered operations dequeued and
↳ processed': 0, 'tiered operations scheduled': 0, 'tiered storage local retention
↳ time (secs)': 0}, 'thread-state': {'active filesystem fsync calls': 0, 'active
↳ filesystem read calls': 0, 'active filesystem write calls': 0}, 'thread-yield': {
↳ 'application thread time evicting (usecs)': 0, 'application thread time waiting
↳ for cache (usecs)': 0, 'connection close blocked waiting for transaction state
↳ stabilization': 0, 'connection close yielded for lsm manager shutdown': 0, 'data
↳ handle lock yielded': 0, 'get reference for page index and slot time sleeping
↳ (usecs)': 0, 'page access yielded due to prepare state change': 0, 'page acquire
↳ busy blocked': 0, 'page acquire eviction blocked': 0, 'page acquire locked
↳ blocked': 0, 'page acquire read blocked': 0, 'page acquire time sleeping (usecs)
↳ ': 0, 'page delete rollback time sleeping for state change (usecs)': 0, 'page
↳ reconciliation yielded due to child modification': 0}, 'transaction': {'Number
↳ of prepared updates': 0, 'Number of prepared updates committed': 0, 'Number of
↳ prepared updates repeated on the same key': 0, 'Number of prepared updates
↳ rolled back': 0, 'oldest pinned transaction ID rolled back for eviction': 0,
↳ 'prepared transactions': 0, 'prepared transactions committed': 0, 'prepared
↳ transactions currently active': 0, 'prepared transactions rolled back': 0,
↳ 'query timestamp calls': 933, 'race to read prepared update retry': 0, 'rollback
↳ to stable calls': 0, 'rollback to stable history store records with stop
↳ timestamps older than newer records': 0, 'rollback to stable inconsistent
↳ checkpoint': 0, 'rollback to stable keys removed': 0, 'rollback to stable keys
↳ restored': 0, 'rollback to stable pages visited': 0, 'rollback to stable
```

(continues on next page)

(continued from previous page)

```

↳ restored tombstones from history store': 0, 'rollback to stable restored updates'_
↳ from history store': 0, 'rollback to stable skipping delete rle': 0, 'rollback_
↳ to stable skipping stable rle': 0, 'rollback to stable sweeping history store_
↳ keys': 0, 'rollback to stable tree walk skipping pages': 0, 'rollback to stable_
↳ updates aborted': 0, 'rollback to stable updates removed from history store': 0,
↳ 'sessions scanned in each walk of concurrent sessions': 15795, 'set timestamp_
↳ calls': 0, 'set timestamp durable calls': 0, 'set timestamp durable updates': 0,
↳ 'set timestamp oldest calls': 0, 'set timestamp oldest updates': 0, 'set_
↳ timestamp stable calls': 0, 'set timestamp stable updates': 0, 'transaction_
↳ begins': 47, 'transaction checkpoint currently running': 0, 'transaction_
↳ checkpoint currently running for history store file': 0, 'transaction checkpoint_
↳ generation': 17, 'transaction checkpoint history store file duration (usecs)': 3,
↳ 'transaction checkpoint max time (msecs)': 78, 'transaction checkpoint min time_
↳ (msecs)': 31, 'transaction checkpoint most recent duration for gathering all_
↳ handles (usecs)': 67, 'transaction checkpoint most recent duration for gathering_
↳ applied handles (usecs)': 0, 'transaction checkpoint most recent duration for_
↳ gathering skipped handles (usecs)': 45, 'transaction checkpoint most recent_
↳ handles applied': 0, 'transaction checkpoint most recent handles skipped': 12,
↳ 'transaction checkpoint most recent handles walked': 25, 'transaction checkpoint_
↳ most recent time (msecs)': 34, 'transaction checkpoint prepare currently running
↳ ': 0, 'transaction checkpoint prepare max time (msecs)': 0, 'transaction_
↳ checkpoint prepare min time (msecs)': 0, 'transaction checkpoint prepare most_
↳ recent time (msecs)': 0, 'transaction checkpoint prepare total time (msecs)': 0,
↳ 'transaction checkpoint scrub dirty target': 0, 'transaction checkpoint scrub_
↳ time (msecs)': 0, 'transaction checkpoint stop timing stress active': 0,
↳ 'transaction checkpoint total time (msecs)': 673, 'transaction checkpoints': 16,
↳ 'transaction checkpoints due to obsolete pages': 0, 'transaction checkpoints_
↳ skipped because database was clean': 0, 'transaction fsync calls for checkpoint_
↳ after allocating the transaction ID': 16, 'transaction fsync duration for_
↳ checkpoint after allocating the transaction ID (usecs)': 9207, 'transaction_
↳ range of IDs currently pinned': 0, 'transaction range of IDs currently pinned by_
↳ a checkpoint': 0, 'transaction range of timestamps currently pinned': 0,
↳ 'transaction range of timestamps pinned by a checkpoint': 0, 'transaction range_
↳ of timestamps pinned by the oldest active read timestamp': 0, 'transaction range_
↳ of timestamps pinned by the oldest timestamp': 0, 'transaction read timestamp of_
↳ the oldest active reader': 0, 'transaction rollback to stable currently running
↳ ': 0, 'transaction walk of concurrent sessions': 1056, 'transactions committed':_
↳ 13, 'transactions rolled back': 34, 'update conflicts': 0},
↳ 'concurrentTransactions': {'write': {'out': 0, 'available': 128, 'totalTickets':_
↳ 128}, 'read': {'out': 0, 'available': 128, 'totalTickets': 128}}, 'snapshot-
↳ window-settings': {'total number of SnapshotTooOld errors': 0, 'minimum target_
↳ snapshot window size in seconds': 300, 'current available snapshot window size_
↳ in seconds': 0, 'latest majority snapshot timestamp available': 'Dec 31_
↳ 19:00:00:0', 'oldest majority snapshot timestamp available': 'Dec 31 19:00:00:0',
↳ 'pinned timestamp requests': 0, 'min pinned timestamp': Timestamp(4294967295,_
↳ 4294967295)}, 'oplog': {'visibility timestamp': Timestamp(0, 0)}}, 'mem': {'bits
↳ ': 64, 'resident': 134, 'virtual': 1563, 'supported': True}, 'metrics': {
↳ 'apiVersions': {'': ['default']}, 'aggStageCounters': {'$_
↳ internalApplyOplogUpdate': 0, '$_internalBoundedSort': 0, '$_
↳ internalConvertBucketIndexStats': 0, '$_internalFindAndModifyImageLookup': 0, '$_
↳ internalInhibitOptimization': 0, '$_internalReshardingIterateTransaction': 0, '$_
↳ internalReshardingOwnershipMatch': 0, '$_internalSetWindowFields': 0, '$_
↳ internalSplitPipeline': 0, '$_internalUnpackBucket': 0, '$_unpackBucket': 0, '$_
↳ addFields': 0, '$bucket': 0, '$bucketAuto': 0, '$changeStream': 0, '$collStats
↳ ': 0, '$count': 0, '$currentOp': 0, '$documents': 0, '$facet': 0, '$geoNear': 0,
↳ '$graphLookup': 0, '$group': 0, '$indexStats': 0, '$limit': 0, '$

```

(continues on next page)

(continued from previous page)

```
↳ '$listLocalSessions': 0, '$listSessions': 0, '$lookup': 0, '$match': 0, '$merge': 0, '$mergeCursors': 0, '$operationMetrics': 0, '$out': 0, '$planCacheStats': 0, '$project': 0, '$queue': 0, '$redact': 0, '$replaceRoot': 0, '$replaceWith': 0, '$sample': 0, '$set': 2, '$setWindowFields': 0, '$skip': 0, '$sort': 0, '$sortByCount': 0, '$unionWith': 0, '$unset': 0, '$unwind': 0}, 'changeStreams': { 'largeEventsFailed': 0}, 'commands': {'<UNKNOWN>': 0, '_addShard': {'failed': 0, 'total': 0}, '_cloneCollectionOptionsFromPrimaryShard': {'failed': 0, 'total': 0}, '_configsrvAbortReshardCollection': {'failed': 0, 'total': 0}, '_configsrvAddShard': {'failed': 0, 'total': 0}, '_configsrvAddShardToZone': { 'failed': 0, 'total': 0}, '_configsrvBalancerCollectionStatus': {'failed': 0, 'total': 0}, '_configsrvBalancerStart': {'failed': 0, 'total': 0}, '_configsrvBalancerStatus': {'failed': 0, 'total': 0}, '_configsrvBalancerStop': { 'failed': 0, 'total': 0}, '_configsrvCleanupReshardCollection': {'failed': 0, 'total': 0}, '_configsrvClearJumboFlag': {'failed': 0, 'total': 0}, '_configsrvCommitChunkMerge': {'failed': 0, 'total': 0}, '_configsrvCommitChunkMigration': {'failed': 0, 'total': 0}, '_configsrvCommitChunkSplit': {'failed': 0, 'total': 0}, '_configsrvCommitChunksMerge': {'failed': 0, 'total': 0}, '_configsrvCommitMovePrimary': {'failed': 0, 'total': 0}, '_configsrvCommitReshardCollection': {'failed': 0, 'total': 0}, '_configsrvCreateDatabase': {'failed': 0, 'total': 0}, '_configsrvDropCollection': {'failed': 0, 'total': 0}, '_configsrvDropDatabase': {'failed': 0, 'total': 0}, '_configsrvEnableSharding': {'failed': 0, 'total': 0}, '_configsrvEnsureChunkVersionIsGreaterThan': {'failed': 0, 'total': 0}, '_configsrvMoveChunk': {'failed': 0, 'total': 0}, '_configsrvMovePrimary': {'failed': 0, 'total': 0}, '_configsrvRefineCollectionShardKey': {'failed': 0, 'total': 0}, '_configsrvRemoveChunks': {'failed': 0, 'total': 0}, '_configsrvRemoveShard': {'failed': 0, 'total': 0}, '_configsrvRemoveShardFromZone': {'failed': 0, 'total': 0}, '_configsrvRemoveTags': {'failed': 0, 'total': 0}, '_configsrvRenameCollectionMetadata': {'failed': 0, 'total': 0}, '_configsrvRepairShardedCollectionChunksHistory': {'failed': 0, 'total': 0}, '_configsrvReshardCollection': {'failed': 0, 'total': 0}, '_configsrvSetAllowMigrations': {'failed': 0, 'total': 0}, '_configsrvShardCollection': {'failed': 0, 'total': 0}, '_configsrvUpdateZoneKeyRange': {'failed': 0, 'total': 0}, '_flushDatabaseCacheUpdates': {'failed': 0, 'total': 0}, '_flushDatabaseCacheUpdatesWithWriteConcern': {'failed': 0, 'total': 0}, '_flushReshardingStateChange': {'failed': 0, 'total': 0}, '_flushRoutingTableCacheUpdates': {'failed': 0, 'total': 0}, '_flushRoutingTableCacheUpdatesWithWriteConcern': {'failed': 0, 'total': 0}, '_getNextSessionMods': {'failed': 0, 'total': 0}, '_getUserCacheGeneration': { 'failed': 0, 'total': 0}, '_isSelf': {'failed': 0, 'total': 0}, '_killOperations': {'failed': 0, 'total': 0}, '_mergeAuthzCollections': {'failed': 0, 'total': 0}, '_migrateClone': {'failed': 0, 'total': 0}, '_recvChunkAbort': {'failed': 0, 'total': 0}, '_recvChunkCommit': {'failed': 0, 'total': 0}, '_recvChunkStart': { 'failed': 0, 'total': 0}, '_recvChunkStatus': {'failed': 0, 'total': 0}, '_shardsrvAbortReshardCollection': {'failed': 0, 'total': 0}, '_shardsrvCleanupReshardCollection': {'failed': 0, 'total': 0}, '_shardsrvCloneCatalogData': {'failed': 0, 'total': 0}, '_shardsrvCommitReshardCollection': {'failed': 0, 'total': 0}, '_shardsrvCreateCollection': {'failed': 0, 'total': 0}, '_shardsrvCreateCollectionParticipant': {'failed': 0, 'total': 0}, '_shardsrvDropCollection': {'failed': 0, 'total': 0}, '_shardsrvDropCollectionIfUUIDNotMatching': {'failed': 0, 'total': 0}, '_shardsrvDropCollectionParticipant': {'failed': 0, 'total': 0}, '_shardsrvDropDatabase': {'failed': 0, 'total': 0}, '_
```

(continues on next page)

(continued from previous page)

```

↳shardsvrDropDatabaseParticipant': {'failed': 0, 'total': 0}, '_
↳shardsvrMovePrimary': {'failed': 0, 'total': 0}, '_
↳shardsvrRefineCollectionShardKey': {'failed': 0, 'total': 0}, '_
↳shardsvrRenameCollection': {'failed': 0, 'total': 0}, '_
↳shardsvrRenameCollectionParticipant': {'failed': 0, 'total': 0}, '_
↳shardsvrRenameCollectionParticipantUnblock': {'failed': 0, 'total': 0}, '_
↳shardsvrReshardCollection': {'failed': 0, 'total': 0}, '_
↳shardsvrReshardingOperationTime': {'failed': 0, 'total': 0}, '_
↳shardsvrSetAllowMigrations': {'failed': 0, 'total': 0}, '_shardsvrShardCollection
': {'failed': 0, 'total': 0}, '_transferMods': {'failed': 0, 'total': 0},
↳'abortTransaction': {'failed': 0, 'total': 0}, 'aggregate': {'allowDiskUseTrue':_
0, 'failed': 0, 'total': 0}, 'appendOplogNote': {'failed': 0, 'total': 0},
↳'applyOps': {'failed': 0, 'total': 0}, 'authenticate': {'failed': 0, 'total': 0},
↳'autoSplitVector': {'failed': 0, 'total': 0}, 'availableQueryOptions': {'failed
': 0, 'total': 0}, 'buildInfo': {'failed': 0, 'total': 0}, 'checkShardingIndex':_
{'failed': 0, 'total': 0}, 'cleanupOrphaned': {'failed': 0, 'total': 0},
↳'cloneCollectionAsCapped': {'failed': 0, 'total': 0}, 'collMod': {'failed': 0,
'total': 0, 'validator': {'failed': 0, 'jsonSchema': 0, 'total': 0}}, 'collStats
': {'failed': 0, 'total': 0}, 'commitTransaction': {'failed': 0, 'total': 0},
↳'compact': {'failed': 0, 'total': 0}, 'connPoolStats': {'failed': 0, 'total': 0},
↳'connPoolSync': {'failed': 0, 'total': 0}, 'connectionStatus': {'failed': 0,
'total': 0}, 'convertToCapped': {'failed': 0, 'total': 0},
↳'coordinateCommitTransaction': {'failed': 0, 'total': 0}, 'count': {'failed': 0,
'total': 0}, 'create': {'failed': 0, 'total': 0, 'validator': {'failed': 0,
'jsonSchema': 0, 'total': 0}}, 'createIndexes': {'failed': 0, 'total': 0},
↳'createRole': {'failed': 0, 'total': 0}, 'createUser': {'failed': 0, 'total': 0},
↳'currentOp': {'failed': 0, 'total': 0}, 'dataSize': {'failed': 0, 'total': 0},
↳'dbCheck': {'failed': 0, 'total': 0}, 'dbHash': {'failed': 0, 'total': 0},
↳'dbStats': {'failed': 0, 'total': 0}, 'delete': {'failed': 0, 'total': 0},
↳'distinct': {'failed': 0, 'total': 1}, 'donorAbortMigration': {'failed': 0,
'total': 0}, 'donorForgetMigration': {'failed': 0, 'total': 0},
↳'donorStartMigration': {'failed': 0, 'total': 0}, 'driverOIDTest': {'failed': 0,
'total': 0}, 'drop': {'failed': 0, 'total': 0}, 'dropAllRolesFromDatabase': {_
'failed': 0, 'total': 0}, 'dropAllUsersFromDatabase': {'failed': 0, 'total': 0},
↳'dropConnections': {'failed': 0, 'total': 0}, 'dropDatabase': {'failed': 0,
'total': 0}, 'dropIndexes': {'failed': 0, 'total': 0}, 'dropRole': {'failed': 0,
'total': 0}, 'dropUser': {'failed': 0, 'total': 0}, 'endSessions': {'failed': 0,
'total': 0}, 'explain': {'failed': 0, 'total': 0}, 'features': {'failed': 0,
'total': 0}, 'filemd5': {'failed': 0, 'total': 0}, 'find': {'failed': 0, 'total
': 5}, 'findAndModify': {'arrayFilters': 0, 'failed': 0, 'pipeline': 0, 'total':_
0}, 'flushRouterConfig': {'failed': 0, 'total': 0}, 'fsync': {'failed': 0, 'total
': 0}, 'fsyncUnlock': {'failed': 0, 'total': 0}, 'getCmdLineOpts': {'failed': 0,
'total': 0}, 'getDatabaseVersion': {'failed': 0, 'total': 0}, 'getDefaultRWConcern':_
{'failed': 0, 'total': 0}, 'getDiagnosticData': {'failed
': 0, 'total': 0}, 'getFreeMonitoringStatus': {'failed': 0, 'total': 0},
↳'getLastErrorMessage': {'failed': 0, 'total': 0}, 'getLog': {'failed': 0, 'total': 0},
↳'getMore': {'failed': 0, 'total': 1}, 'getParameter': {'failed': 0, 'total': 0},
↳'getShardMap': {'failed': 0, 'total': 0}, 'getShardVersion': {'failed': 0, 'total
': 0}, 'getnonce': {'failed': 0, 'total': 0}, 'grantPrivilegesToRole': {'failed
': 0, 'total': 0}, 'grantRolesToRole': {'failed': 0, 'total': 0}, 'grantRolesToUser':_
{'failed': 0, 'total': 0}, 'hello': {'failed': 1, 'total':_
140}, 'hostInfo': {'failed': 0, 'total': 0}, 'insert': {'failed': 0, 'total': 6},
↳'internalRenameIfOptionsAndIndexesMatch': {'failed': 0, 'total': 0}, 'invalidateUserCache':_
{'failed': 0, 'total': 0}, 'isMaster': {'failed': 0, 'total': 0}, 'killAllSessions':_
{'failed': 0, 'total': 0}, 'killAllSessionsByPattern': {'failed': 0, 'total': 0}, 'killCursors':_
{'failed': 0, 'total': 0}

```

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

```
$anyElementTrue': 0, '$arrayElemAt': 0, '$arrayToObject': 0, '$asin': 0, '$asinh': 0, '$atan': 0, '$atan2': 0, '$atanh': 0, '$avg': 0, '$binarySize': 0, '$bsonSize': 0, '$ceil': 0, '$cmp': 0, '$concat': 0, '$concatArrays': 0, '$cond': 0, '$const': 0, '$convert': 0, '$cos': 0, '$cosh': 0, '$dateAdd': 0, '$dateDiff': 0, '$dateFromParts': 0, '$dateFromString': 0, '$dateSubtract': 0, '$dateToParts': 0, '$dateToString': 0, '$dateTrunc': 0, '$dayOfMonth': 0, '$dayOfWeek': 0, '$dayOfYear': 0, '$degreesToRadians': 0, '$divide': 0, '$eq': 0, '$exp': 0, '$filter': 0, '$first': 0, '$floor': 0, '$function': 0, '$getField': 0, '$gt': 0, '$gte': 0, '$hour': 0, '$ifNull': 0, '$in': 0, '$indexOfArray': 0, '$indexOfBytes': 0, '$indexOfCP': 0, '$isArray': 0, '$isNumber': 0, '$isoDayOfWeek': 0, '$isoWeek': 0, '$isoWeekYear': 0, '$last': 0, '$let': 0, '$literal': 0, '$ln': 0, '$log': 0, '$log10': 0, '$lt': 0, '$lte': 0, '$ltrim': 0, '$map': 0, '$max': 0, '$mergeObjects': 0, '$meta': 0, '$millisecond': 0, '$min': 0, '$minute': 0, '$mod': 0, '$month': 0, '$multiply': 0, '$ne': 0, '$not': 0, '$objectToArray': 0, '$or': 0, '$pow': 0, '$radiansToDegrees': 0, '$rand': 0, '$range': 0, '$reduce': 0, '$regexFind': 0, '$regexFindAll': 0, '$regexMatch': 0, '$replaceAll': 0, '$replaceOne': 0, '$reverseArray': 0, '$round': 0, '$rtrim': 0, '$second': 0, '$setDifference': 0, '$setEquals': 0, '$setField': 0, '$setIntersection': 0, '$setIsSubset': 0, '$setUnion': 0, '$sin': 0, '$sinh': 0, '$size': 0, '$slice': 0, '$split': 0, '$sqrt': 0, '$stdDevPop': 0, '$stdDevSamp': 0, '$strLenBytes': 0, '$strLenCP': 0, '$strcasecmp': 0, '$substr': 0, '$substrBytes': 0, '$substrCP': 0, '$subtract': 0, '$sum': 0, '$switch': 0, '$tan': 0, '$tanh': 0, '$toBool': 0, '$toDate': 0, '$toDecimal': 0, '$toDouble': 0, '$toHashedIndexKey': 0, '$toInt': 0, '$toLong': 0, '$toLower': 0, '$toObjectId': 0, '$toString': 0, '$toUpperCase': 0, '$trim': 0, '$trunc': 0, '$type': 0, '$unsetField': 0, '$week': 0, '$year': 0, '$zip': 0}, 'groupAccumulators': {'$_internalJsReduce': 0, '$accumulator': 0, '$addToSet': 0, '$avg': 0, '$count': 0, '$first': 0, '$last': 0, '$max': 0, '$mergeObjects': 0, '$min': 0, '$push': 0, '$stdDevPop': 0, '$stdDevSamp': 0, '$sum': 0}, 'match': {'$all': 0, '$alwaysFalse': 0, '$alwaysTrue': 0, '$and': 0, '$bitsAllClear': 0, '$bitsAllSet': 0, '$bitsAnyClear': 0, '$bitsAnySet': 0, '$comment': 0, '$elemMatch': 0, '$eq': 0, '$exists': 0, '$expr': 0, '$geoIntersects': 0, '$geoWithin': 0, '$gt': 0, '$gte': 0, '$in': 0, '$jsonSchema': 0, '$lt': 4, '$lte': 0, '$mod': 0, '$ne': 0, '$near': 0, '$nearSphere': 0, '$nin': 0, '$nor': 0, '$not': 0, '$or': 0, '$regex': 0, '$sampleRate': 0, '$size': 0, '$text': 0, '$type': 0, '$where': 0}, 'windowAccumulators': {'$addToSet': 0, '$avg': 0, '$count': 0, '$covariancePop': 0, '$covarianceSamp': 0, '$denseRank': 0, '$derivative': 0, '$documentNumber': 0, '$expMovingAvg': 0, '$first': 0, '$integral': 0, '$last': 0, '$max': 0, '$min': 0, '$push': 0, '$rank': 0, '$shift': 0, '$stdDevPop': 0, '$stdDevSamp': 0, '$sum': 0}, 'query': {'deleteManyCount': 0, 'planCacheTotalSizeEstimateBytes': 0, 'updateDeleteManyDocumentsMaxCount': 0, 'updateDeleteManyDocumentsTotalCount': 0, 'updateDeleteManyDurationMaxMs': 0, 'updateDeleteManyDurationTotalMs': 0, 'updateManyCount': 0, 'updateOneOpStyleBroadcastWithExactIDCount': 0, 'multiPlanner': {'classicCount': 0, 'classicMicros': 0, 'classicWorks': 0, 'sbeCount': 0, 'sbeMicros': 0, 'sbeNumReads': 0, 'histograms': {'classicMicros': 0, 'lowerBound': 0, 'count': 0}, {'lowerBound': 1024, 'count': 0}, {'lowerBound': 4096, 'count': 0}, {'lowerBound': 16384, 'count': 0}, {'lowerBound': 65536, 'count': 0}, {'lowerBound': 262144, 'count': 0}, {'lowerBound': 1048576, 'count': 0}, {'lowerBound': 4194304, 'count': 0}, {'lowerBound': 16777216, 'count': 0}, {'lowerBound': 67108864, 'count': 0}, {'lowerBound': 268435456, 'count': 0}, {'lowerBound': 1073741824, 'count': 0}], 'classicNumPlans': [{"lowerBound": 0, "count": 0}, {"lowerBound": 2, "count": 0}, {"lowerBound": 4, "count": 0}, {"lowerBound": 8, "count": 0}, {"lowerBound": 16, "count": 0}, {"lowerBound": 32, "count": 0}], 'classicWorks': [{"lowerBound": 0, "count": 0}, {"lowerBound": 128, "count": 0}, {"lowerBound": 256, "count": 0}, {"lowerBound": 512, "count": 0}, {"lowerBound": 1024, "count": 0}, {"lowerBound": 2048, "count": 0}, {"lowerBound": 4096, "count": 0}, {"lowerBound": 8192, "count": 0}, {"lowerBound": 16384, "count": 0}, {"lowerBound": 32768, "count": 0}, {"lowerBound": 65536, "count": 0}, {"lowerBound": 131072, "count": 0}, {"lowerBound": 262144, "count": 0}, {"lowerBound": 524288, "count": 0}, {"lowerBound": 1048576, "count": 0}, {"lowerBound": 2097152, "count": 0}, {"lowerBound": 4194304, "count": 0}, {"lowerBound": 8388608, "count": 0}, {"lowerBound": 16777216, "count": 0}, {"lowerBound": 33554432, "count": 0}, {"lowerBound": 67108864, "count": 0}, {"lowerBound": 134217728, "count": 0}, {"lowerBound": 268435456, "count": 0}, {"lowerBound": 536870912, "count": 0}, {"lowerBound": 1073741824, "count": 0}, {"lowerBound": 2147483648, "count": 0}, {"lowerBound": 4294967296, "count": 0}, {"lowerBound": 8589934592, "count": 0}, {"lowerBound": 17179869184, "count": 0}, {"lowerBound": 34359738368, "count": 0}, {"lowerBound": 68719476736, "count": 0}, {"lowerBound": 137438953472, "count": 0}, {"lowerBound": 274877906944, "count": 0}, {"lowerBound": 549755813888, "count": 0}, {"lowerBound": 1099511627776, "count": 0}, {"lowerBound": 2199023255552, "count": 0}, {"lowerBound": 4398046511104, "count": 0}, {"lowerBound": 8796093022208, "count": 0}, {"lowerBound": 17592186044416, "count": 0}, {"lowerBound": 35184372088832, "count": 0}, {"lowerBound": 70368744177664, "count": 0}, {"lowerBound": 140737488355328, "count": 0}, {"lowerBound": 281474976710656, "count": 0}, {"lowerBound": 562949953421312, "count": 0}, {"lowerBound": 1125899906842624, "count": 0}, {"lowerBound": 2251799813685248, "count": 0}, {"lowerBound": 4503599627370496, "count": 0}, {"lowerBound": 9007199254740992, "count": 0}, {"lowerBound": 18014398509481984, "count": 0}, {"lowerBound": 36028797018963968, "count": 0}, {"lowerBound": 72057594037927936, "count": 0}, {"lowerBound": 144115188075855872, "count": 0}, {"lowerBound": 288230376151711744, "count": 0}, {"lowerBound": 576460752303423488, "count": 0}, {"lowerBound": 1152921504606846976, "count": 0}, {"lowerBound": 2305843009213693952, "count": 0}, {"lowerBound": 4611686018427387904, "count": 0}, {"lowerBound": 9223372036854775808, "count": 0}, {"lowerBound": 18446744073709551616, "count": 0}, {"lowerBound": 36893488147419103232, "count": 0}, {"lowerBound": 73786976294838206464, "count": 0}, {"lowerBound": 147573952589676412928, "count": 0}, {"lowerBound": 295147905179352825856, "count": 0}, {"lowerBound": 590295810358705651712, "count": 0}, {"lowerBound": 1180591620717411303424, "count": 0}, {"lowerBound": 2361183241434822606848, "count": 0}, {"lowerBound": 4722366482869645213696, "count": 0}, {"lowerBound": 9444732965739290427392, "count": 0}, {"lowerBound": 18889465931478580854784, "count": 0}, {"lowerBound": 37778931862957161609568, "count": 0}, {"lowerBound": 75557863725914323219136, "count": 0}, {"lowerBound": 151115727458228646438272, "count": 0}, {"lowerBound": 302231454916457292876544, "count": 0}, {"lowerBound": 604462909832914585753088, "count": 0}, {"lowerBound": 1208925819665829171506176, "count": 0}, {"lowerBound": 2417851639331658343012352, "count": 0}, {"lowerBound": 4835703278663316686024704, "count": 0}, {"lowerBound": 9671406557326633372049408, "count": 0}, {"lowerBound": 19342813114653266744098816, "count": 0}, {"lowerBound": 38685626229306533488197632, "count": 0}, {"lowerBound": 77371252458613066976395264, "count": 0}, {"lowerBound": 154742504917226133952785128, "count": 0}, {"lowerBound": 309485009834452267905570256, "count": 0}, {"lowerBound": 618970019668904535811140512, "count": 0}, {"lowerBound": 123794003933780907162228024, "count": 0}, {"lowerBound": 247588007867561814324456048, "count": 0}, {"lowerBound": 495176015735123628648912096, "count": 0}, {"lowerBound": 990352031470247257297824192, "count": 0}, {"lowerBound": 1980704062940494514595648384, "count": 0}, {"lowerBound": 3961408125880989029191296768, "count": 0}, {"lowerBound": 7922816251761978058382593536, "count": 0}, {"lowerBound": 15845632503523956116765187072, "count": 0}, {"lowerBound": 31691265007047912233530374144, "count": 0}, {"lowerBound": 63382530014095824467060748288, "count": 0}, {"lowerBound": 126765060028191648934121496576, "count": 0}, {"lowerBound": 253530120056383297868242993152, "count": 0}, {"lowerBound": 507060240112766595736485986304, "count": 0}, {"lowerBound": 1014120480225333191472971972608, "count": 0}, {"lowerBound": 2028240960450666382945943945216, "count": 0}, {"lowerBound": 4056481920901332765891887890432, "count": 0}, {"lowerBound": 8112963841802665531783775780864, "count": 0}, {"lowerBound": 16225927683605331063567551561728, "count": 0}, {"lowerBound": 32451855367210662127135103123456, "count": 0}, {"lowerBound": 64903710734421324254270206246912, "count": 0}, {"lowerBound": 129807421468842648508540412493824, "count": 0}, {"lowerBound": 259614842937685297017080824987648, "count": 0}, {"lowerBound": 519229685875370594034161649975296, "count": 0}, {"lowerBound": 1038459371750741188068323299950592, "count": 0}, {"lowerBound": 2076918743501482376136646599901184, "count": 0}, {"lowerBound": 4153837487002964752273293199802368, "count": 0}, {"lowerBound": 8307674974005929504546586399604736, "count": 0}, {"lowerBound": 16615349948011859009093172799209472, "count": 0}, {"lowerBound": 33230699896023718018186345598418944, "count": 0}, {"lowerBound": 66461399792047436036372691196837888, "count": 0}, {"lowerBound": 132922799584094872072745382393675776, "count": 0}, {"lowerBound": 265845599168189744145490764787351552, "count": 0}, {"lowerBound": 531691198336379488290981529574703056, "count": 0}, {"lowerBound": 1063382396673598976581963059149406112, "count": 0}, {"lowerBound": 2126764793347197953163926118298812224, "count": 0}, {"lowerBound": 4253529586694395906327852236597624448, "count": 0}, {"lowerBound": 8507059173388791812655704473195248896, "count": 0}, {"lowerBound": 17014118346777583625311408946384977792, "count": 0}, {"lowerBound": 34028236693555167250622817892769555584, "count": 0}, {"lowerBound": 68056473387110334501245635785539111688, "count": 0}, {"lowerBound": 136112946774220669002491271571078223376, "count": 0}, {"lowerBound": 272225893548441338004982543142156446752, "count": 0}, {"lowerBound": 544451787096882676009965086284312893504, "count": 0}, {"lowerBound": 108890357419376535201993017256862577008, "count": 0}, {"lowerBound": 217780714838753070403986034513725154016, "count": 0}, {"lowerBound": 435561429677506140807972069027450308032, "count": 0}, {"lowerBound": 871122859355012281615944138054900616064, "count": 0}, {"lowerBound": 1742245718700024563231888276109801232128, "count": 0}, {"lowerBound": 3484491437400049126463776552219602464256, "count": 0}, {"lowerBound": 6968982874800098252927553104439204928512, "count": 0}, {"lowerBound": 13937965749600196505855106208878409857224, "count": 0}, {"lowerBound": 27875931499200393011710212417756819714448, "count": 0}, {"lowerBound": 55751862998400786023420424835513639428896, "count": 0}, {"lowerBound": 111503725996801572046840849671027278957792, "count": 0}, {"lowerBound": 223007451993603144093681699342054557915584, "count": 0}, {"lowerBound": 446014903987206288187363398684109115831168, "count": 0}, {"lowerBound": 892029807974412576374726797368218231662336, "count": 0}, {"lowerBound": 1784059615948825152749453594736436463324672, "count": 0}, {"lowerBound": 3568119231897650305498907189472872926649344, "count": 0}, {"lowerBound": 7136238463795300610997814378945745853298688, "count": 0}, {"lowerBound": 14272476927590601221995628757891491706597776, "count": 0}, {"lowerBound": 28544953855181202443991257515782983413195552, "count": 0}, {"lowerBound": 57089907710362404887982515031565966826391104, "count": 0}, {"lowerBound": 114179815420724809775965230063131933652782208, "count": 0}, {"lowerBound": 228359630841449619551930460126263867305564416, "count": 0}, {"lowerBound": 456719261682899239103860920252527734611128832, "count": 0}, {"lowerBound": 913438523365798478207721840505055469222257664, "count": 0}, {"lowerBound": 1826877046731596956415443681010110938444515328, "count": 0}, {"lowerBound": 3653754093463193912830887362020221876889030656, "count": 0}, {"lowerBound": 7307508186926387825661774724040443753778061312, "count": 0}, {"lowerBound": 14615016373852775651323559448080887507556122624, "count": 0}, {"lowerBound": 29230032747705551302647118896161775015112252448, "count": 0}, {"lowerBound": 58460065495411102605294237792323550030224504896, "count": 0}, {"lowerBound": 11692013099082220521058847558464710006044900976, "count": 0}, {"lowerBound": 23384026198164441042117695116929420012089801952, "count": 0}, {"lowerBound": 46768052396328882084235390233858840024179603904, "count": 0}, {"lowerBound": 93536104792657764168470780467717680048359207808, "count": 0}, {"lowerBound": 187072209585315528336941560935435360096718415616, "count": 0}, {"lowerBound": 374144419170631056673883121870870720193436831232, "count": 0}, {"lowerBound": 748288838341262113347766243741741440386873662464, "count": 0}, {"lowerBound": 149657767668252422669553248748348288077374732492, "count": 0}, {"lowerBound": 299315535336504845339106497496696576154749464984, "count": 0}, {"lowerBound": 598631070673009690678212994993393152309498929968, "count": 0}, {"lowerBound": 1197262141346019381356425989986786304618988959936, "count": 0}, {"lowerBound": 2394524282692038762712851979973572609237977919872, "count": 0}, {"lowerBound": 4789048565384077525425703959947145218475955839744, "count": 0}, {"lowerBound": 9578097130768155050851407919894290436951911679488, "count": 0}, {"lowerBound": 19156194261536310101702815839788580873853823358976, "count": 0}, {"lowerBound": 38312388523072620203405631679577161747707646717952, "count": 0}, {"lowerBound": 76624777046145240406811263359154323495415293435904, "count": 0}, {"lowerBound": 153249554092290480813622526787308646985830586871808, "count": 0}, {"lowerBound": 306499108184580961627245053574617293971661173743616, "count": 0}, {"lowerBound": 612998216369161923254490107149234587943322347487232, "count": 0}, {"lowerBound": 122599643273832384650888201429846917886664464974464, "count": 0}, {"lowerBound": 245199286547664769301776402859693835773328929948928, "count": 0}, {"lowerBound": 490398573095329538603552805719387671546657859897856, "count": 0}, {"lowerBound": 980797146190659077207105611438775343093315719795712, "count": 0}, {"lowerBound": 196159429238131815441421122287750666618663143959424, "count": 0}, {"lowerBound": 392318858476263630882842244575501333237326287918848, "count": 0}, {"lowerBound": 784637716952527261765684489151002666474652575837696, "count": 0}, {"lowerBound": 1569275433905054523531368978302005332943105151675392, "count": 0}, {"lowerBound": 3138550867810109047062737956604010665886210303350784, "count": 0}, {"lowerBound": 6277101735620218094125475913208021331772420606701568, "count": 0}, {"lowerBound": 12554203471240436188250951826416042663544841213403136, "count": 0}, {"lowerBound": 25108406942480872376501903652832085327089682426806272, "count": 0}, {"lowerBound": 50216813884961744753003807305664170654179364853612544, "count": 0}, {"lowerBound": 10043362776992358950600761461132834130835873770722588, "count": 0}, {"lowerBound": 20086725553984717901201522922265668261671747541445176, "count": 0}, {"lowerBound": 40173451107969435802403045844531336523343495082890352, "count": 0}, {"lowerBound": 80346902215938871604806091689062673046686980165780704, "count": 0}, {"lowerBound": 160693804438777743209612183378125346093373960331561408, "count": 0}, {"lowerBound": 321387608877555486419224366756250692186747920663122816, "count": 0}, {"lowerBound": 642775217755110972838448733512501843734495841326245632, "count": 0}, {"lowerBound": 1285550435510221945676894467025036887468991682652491264, "count": 0}, {"lowerBound": 257110087102044389135378893405007377493798336530498252, "count": 0}, {"lowerBound": 514220174204088778270757786810014754987596673060996504, "count": 0}, {"lowerBound": 1028440348408177556541515573620029109851193346121993008, "count": 0}, {"lowerBound": 2056880696816355113083031147240058219702386692243986016, "count": 0}, {"lowerBound": 4113761393632710226166062294480116439404773384487972032, "count": 0}, {"lowerBound": 8227522787265420452332124588960223278809546768959944064, "count": 0}, {"lowerBound": 1645504557453084090466248917792044655761909353791988128, "count": 0}, {"lowerBound": 3291009114906168180932497835584089311523818707583976256, "count": 0}, {"lowerBound": 6582018229812336361864995671168178623047637415167952512, "count": 0}, {"lowerBound": 1316403645962467272372991134233635724609527423033590504, "count": 0}, {"lowerBound": 263280729192493454474598226846727449218905484606718008, "count": 0}, {"lowerBound": 526561458384986908949196453693454898437810969213436016, "count": 0}, {"lowerBound": 1053122916769973817898392907386909796756621938426772032, "count": 0}, {"lowerBound": 2106245833539947635796785814773819593513243876853544064, "count": 0}, {"lowerBound": 4212491667079895271593571629547639187026487753707088128, "count": 0}, {"lowerBound": 8424983334159790543187143259095278374052955507414176256, "count": 0}, {"lowerBound": 1684996666831958108637426651819055674810591007882835252, "count": 0}, {"lowerBound": 3369993333663916217274853303638111349621182015765670504, "count": 0}, {"lowerBound": 6739986667327832434549706607276222692423640311531341008, "count": 0}, {"lowerBound": 13479973334655664869099413214552445384847280623062680016, "count": 0}, {"lowerBound": 26959946669311329738198826428504890768694561246125360032, "count": 0}, {"lowerBound": 53919893338622659476397652857009781537389122492250720064, "count": 0}, {"lowerBound": 107839786673245318552753105714019563074778244844501440128, "count": 0}, {"lowerBound": 215679573346490637105506211428039126145564889689002880256, "count": 0}, {"lowerBound": 431359146692981274211012422856078252281129779378005760512, "count": 0}, {"lowerBound": 862718293385962548422024845712156504562259558760011521024, "count": 0}, {"lowerBound": 1725436586771925096844049694242313009124519117320023042048, "count": 0}, {"lowerBound": 3450873173543850193688099388484626018249038234640046084096, "count": 0}, {"lowerBound": 6901746347087700387376198776969252036498076469280092168192, "count": 0}, {"lowerBound": 13803492694175400774752397553938504072976152938560184336384, "count": 0}, {"lowerBound": 27606985388350801549504795107877008145952305877120368672768, "count": 0}, {"lowerBound": 5521397077670160309900959021575
```

(continues on next page)

(continued from previous page)

```

': 4096, 'count': 0}, {'lowerBound': 8192, 'count': 0}, {'lowerBound': 16384,
'count': 0}, {'lowerBound': 32768, 'count': 0}], 'sbeMicros': [{'lowerBound': 0,
'count': 0}, {'lowerBound': 1024, 'count': 0}, {'lowerBound': 4096, 'count': 0},
{'lowerBound': 16384, 'count': 0}, {'lowerBound': 65536, 'count': 0}, {'lowerBound':
262144, 'count': 0}, {'lowerBound': 1048576, 'count': 0}, {'lowerBound': 4194304,
'count': 0}, {'lowerBound': 16777216, 'count': 0}, {'lowerBound': 67108864,
'count': 0}, {'lowerBound': 268435456, 'count': 0}, {'lowerBound': 1073741824,
'count': 0}], 'sbeNumPlans': [{['lowerBound': 0, 'count': 0}, {'lowerBound': 2, 'count': 0},
{'lowerBound': 4, 'count': 0}, {'lowerBound': 8, 'count': 0}, {'lowerBound': 16, 'count': 0},
{'lowerBound': 32, 'count': 0}], 'sbeNumReads': [{['lowerBound': 0, 'count': 0}, {'lowerBound': 128, 'count': 0},
{'lowerBound': 256, 'count': 0}, {'lowerBound': 512, 'count': 0}, {'lowerBound': 1024,
'count': 0}, {'lowerBound': 2048, 'count': 0}, {'lowerBound': 4096, 'count': 0},
{'lowerBound': 8192, 'count': 0}, {'lowerBound': 16384, 'count': 0}, {'lowerBound': 32768,
'count': 0}]}}, 'queryExecutor': {'scanned': 1, 'scannedObjects': 483, 'collectionScans': {'nonTailable': 2, 'total': 2}},
'record': {'moves': 0}, 'repl': {'executor': {'pool': {'inProgressCount': 0}}, 'queues': {'networkInProgress': 0, 'sleepers': 0}, 'unsignaledEvents': 0}, 'shuttingDown': False, 'networkInterface': 'DEPRECATED: getDiagnosticString is deprecated in NetworkInterfaceTL'}, 'apply': {'attemptsToBecomeSecondary': 0, 'batchSize': 0, 'batches': {'num': 0, 'totalMillis': 0}, 'ops': 0}, 'buffer': {'count': 0, 'maxSizeBytes': 0, 'sizeBytes': 0}, 'initialSync': {'completed': 0, 'failedAttempts': 0, 'failures': 0}, 'network': {'bytes': 0, 'getmores': {'num': 0, 'totalMillis': 0, 'numEmptyBatches': 0}, 'notPrimaryLegacyUnacknowledgedWrites': 0, 'notPrimaryUnacknowledgedWrites': 0, 'oplogGetMoresProcessed': {'num': 0, 'totalMillis': 0}, 'ops': 0}, 'readersCreated': 0, 'replSetUpdatePosition': {'num': 0}, 'reconfig': {'numAutoReconfigsForRemovalOfNewlyAddedFields': 0}, 'stateTransition': {'lastStateTransition': '', 'userOperationsKilled': 0}, 'userOperationsRunning': 0, 'syncSource': {'numSelections': 0}, 'numSyncSourceChangesDueToSignificantlyCloserNode': 0, 'numTimesChoseDifferent': 0, 'numTimesChoseSame': 0, 'numTimesCouldNotFind': 0}, 'ttl': {'deletedDocuments': 0, 'passes': 15}}, 'ok': 1.0}

```

```

# Retrieve and preprocess text
dates = fomc['minutes'].distinct('date')           # check dates stored in MongoDB
docs = Series({doc['date']: doc['text'] for doc in fomc.select('minutes')},
              name='minutes').sort_index()
DataFrame(docs)

```

## 39.1 Language Modelling

## REINFORCEMENT LEARNING

### UNDER CONSTRUCTION

- Spending policy

```
from pandas import DataFrame, Series
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm
from datetime import datetime
from typing import List, Tuple, Any, Dict
from finds.database import SQL, RedisDB
from finds.structured import Stocks, Signals, SignalsFrame
from finds.busday import BusDay
from finds.misc import Show
from secret import credentials, paths, CRSP_DATE
show = Show(ndigits=4, latex=None)
pd.set_option('display.max_rows', None)
VERBOSE = 0
#%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)
signals = Signals(user, verbose=VERBOSE)
outdir = paths['images'] / 'glm'
```