Building Factors using Natural Language Processing on Regulatory Filings

Alexander P. Jermann *
Dept. of Industrial Engineering & Operations Research
Columbia University
New York, 10027, NY
alexander.jermann@columbia.edu

1 Introduction

Every year (and quarter) every U.S. public company has to disclose a comprehensive summary of their financial standing to the U.S. Securities and Exchange Commission ("SEC") to comply with disclosure requirements. This report is public and highly standardized and includes a description of the business, its corporate structure and governance, risk factors, legal proceedings, financial statements, and more.

Empirical evidence shows that these regulatory reports are generally not entirely rewritten year-to-year but rather edited to fit the company's current financial standing. This means that there is a high textual similarity between reports. Cohen et al. (2018) made the discovery that the textual similarity between consecutive reports from the previous to the current year was correlated to short-term subsequent financial performance of the company. In other words, if a company changed a lot of the text from their previous to their most recent regulatory filing the subsequent financial performance was lower relative to companies with fewer text changes. Cohen et al. (2018) showed that the results were robust over longer periods of time from 1995-2014. This result is surprising as the efficient market hypothesis postulates that any public information is priced into the asset.

For this project, I provide a short introduction by giving a possible economic rationale for the existence of such factors and analyze the robustness and risks of said factor. Using the theoretical and empirical findings by Cohen et al. (2018), I then build a natural language processing ("NLP") model in Python that automatically compared the documents by filing year (and quarter) to derive statistics. I subsequently use those statistics as a signal to build an equity long/short portfolio to verify the validity of the theory in the years after the paper was published. In this project, I found that the factors provided robust results during the time period of 2015 to 2016 providing 6-7% yearly returns and that factors have since lost on their predictive power only rendering 2% returns.

2 Background and Relevant Theory

2.1 Regulatory Filings

Every U.S. public company is required to disclose information to shareholders information on an ongoing basis to comply with disclosure requirements. The annual and quarterly reports provide a comprehensive overview of the company's business and financial standing ranging from information on the business model, risk factors, legal proceedings, financial statements, corporate governance, and more (U.S. Securities and Exchange Comission). Companies try to be strategic as to what information is included in the report and how it is included. There is an inherent trade-off between complying with disclosure requirements and revealing as little information as possible that could negatively affect the firms subsequent financial performance as shown by (Dyer et al., 2017) and Lee (2012).

*alternate email: alexander.jermann@gmail.com

Preprint. Under review.

Zhang (2009) and Easton and Zmijewski (1993) find evidence of investor underreaction to information released in 10-K reports and a corresponding lag in the reaction of the underlying asset pricing. They identify that trading volumes and stock price movements happen during the weeks before the release of the official 10-K report and are sluggish after the release of the 10-K report and only get priced over the next few weeks. They suggest that this is due to the fact that metrics like earnings per share, dividends, sales growth are disclosed and made public weeks before during earnings calls. As a result, investors might view the filing as superfluous and largely ignore it. This fact is supported by the number of downloads of the 10-K reports on the S.E.C. online system EDGAR.

Lee (2012) explores an alternative explanation for the delayed response in pricing. He argues that while the semi-strong form of the efficient market hypothesis (see Fama (1970) for the hypothesis) states that capital markets incorporate all public information into security pricing in a timely and efficient manner, judgement and decision research suggests otherwise. Hirshleifer et al. (2003) find that investors, analysts have bounded rationality in the amount of information that they can process and can thus not attend to all information made available. Measuring complexity of documents in terms of word count, empirical evidence suggests that historically more complex documents are more heavily mispriced and have larger price adjustments in subsequent trading days.

Finally, Cohen et al. (2018) suggests companies are lazy and consistently use previous 10-K reports as templates for current reporting requirements. The author finds evidence that companies tend to only edit sections to reflect the current companies business and financial health. The authors find that companies that do not change their financial reports by a lot ("non-changers") tend to relatively outperform companies that make a lot of changes ("changers"). They find that a long/short portfolio that goes long on the top 20% ("non-changers" as in high textual similarity between documents) and short on the bottom 20% ("changers") by an average of 5-7% annually, supporting the assumption that companies only report what is necessary.

In summary, this section provided evidence that there is quantifiable evidence of investor inattention to 10-K reports, that the information released in 10-K and 10-Q reports has a significant lag and is only priced into the underlying security over the following weeks through gradual information processing or more immediate news reporting. This section has also shown that textual changes of subsequent reports have historically provided a signal of future financial performance of companies. Arguably, it thus makes sense to use a systematic approach to quantify changes of financial reports as a factor for investing.

2.2 Factor Investing

Factor investing is an investment approach where quantifiable characteristics or "factors" are used to explain differences in stock returns. Typical factors that were first published 30 years ago include: size, value, momentum, asset growth, and leverage (Value, 2016). The earliest paper on factors was published by Ross (1976). In his paper, Ross proposes an arbitrage pricing theory that holds that the expected return of assets can be modeled as the linear combination of various factors as follows:

$$r_j = a_j + \lambda_{j1} f_1 + \lambda_{j2} f_2 + ... + \lambda_{jn} f_n + \epsilon_j \tag{1}$$

where

- a_j is a constant for asset j
- f_n is a systematic factor
- λ_{jn} is called the factor loading and represents the relative sensitivity of the j-th asset to the n-th factor
- ϵ_j is the j-th asset idiosyncratic risk with mean 0.

More rigorously we define returns as $r \in \mathbb{R}^m$, factor loadings as $\Lambda \in \mathbb{R}^{m \times n}$, and factors as $f \in \mathbb{R}^n$ and is:

$$r = r_f + \Lambda f + \epsilon, \epsilon \sim \mathcal{N}(0, \Psi) \tag{2}$$

where ϵ follows a multivariate normal distribution with mean 0. In this model, we assume that

$$f \sim \mathcal{N}(\mu, \Omega) \tag{3}$$

where μ is the expected risk premium vector and Ω is the factor covariance matrix. We thus get the expected returns of:

$$\mathbb{E}(r) = r_f + \Lambda\mu, \; Cov(r) = \Lambda\Omega\Lambda^T + \Psi \tag{4}$$

It is assumed that the factors are known in a given model. In the subsequent sections we will use the similarity measures of the documents as the factors.

3 Methodology

To measure how effective textual changes between subsequent regulatory filings of individual companies are in predicting future returns of companies, we first download all the 10-K and 10-Q reports of the past few years for every U.S publicly traded company, define and compute the similarity measures between every document of a given company, and then map the factor to the historical asset price and then analyze how well the factor predicts future returns. As a final step we use the factor to rank assets and build a long/short portfolio that goes long on assets that have high factor values (top quintile or 20% and goes short on the bottom quintile or 20%). The next few subsections outline the methodology in more detail.

3.1 Data Collection & Scraping

In order to build the necessary sample dataset of 10-K and 10-Q reports, we had to scrape the SEC's Electronic Data Gathering, Analysis, and Retrieval ("EDGAR") site. For context, the average length of a 10-K report in 2017 was 60'000 words long which is the length of the Book Lord of the Flies by William Golding. The final dataset of 10-K and 10-Q reports between 2011 and 2018, contained 260'425 documents from all U.S. companies listed on the NYSE, AMEX, or NASDAQ in that time-period. This resulted in a dataset of approximately 80 gigabytes large. Since we were only interested in textual analysis of these documents we discard supplementary material such as images, tables, PDF files, excel files, etc. All the necessary code and documentation on how to download, pre-process, and get the dataset into the right format is linked in the code section 5.1 at the end of this paper.

3.2 Difference Measures

To measure the similarity between two documents we use two metrics that are common in the literature of textual analysis and natural language processing. Namely the Jaccard Similarity Measure and the Cosine similarity measures.

3.2.1 Jaccard Similarity Measure

The Jaccard Similarity measure compares members of two sets and compares which members are shared and which are distinct.Intuitively, the Jaccard Similarity measure is the size of the intersection, divided by the size of the union of the words. The value it returns will be between 0 and 1, where zero would be no words and common, and 1 would indicate that both sets have the same words.

Definition: Let A and B be words, then the Jaccard Similarity score is

$$J(A,B) = \frac{\mid A \cap B \mid}{\mid A \cup B \mid} \tag{5}$$

where $0 \geq J(A,B) \geq 1$. If both $A, B = \emptyset$ we define $J(A,B) = 1$.

In the context of comparing documents, it is a somewhat naive measure as it does not take into consideration word counts and frequencies of words as opposed to the cosine measure in the following section. Further, it does not take into consideration word similarity and contextual meaning of words. A project by Kai et al. (2017) showed that using word-embeddings to capture contextual information did not increase the predictive power of the factor. Intuitively, this makes sense since a 60'000 word document is being condensed into a single number between 0 and 1. So much information is lost in this mapping that the extra information captured by contextual information does not significantly improve the predictive power. We note however that we find that word count and frequency does improve the accuracy as we will see in the result section.

3.2.2 Cosine Similarity Measure

Let A and B be words. We map A and B into the vector space S, where S has dimension of the union between the word sets A and B

The dimensions of the vector-space are the set of the union between A and B. The vector $A \in$

$$C(A, B) = \frac{\mathbf{A} \cdot \mathbf{B}}{|| \mathbf{A} || \cdot || \mathbf{B} ||} \tag{6}$$

where the nominator is the dot product between the vectors $\mathbf{A}$ and $\mathbf{B}$ and the denominator is the Euclidean norm.

3.3 Statistical Factor Analysis

In this section, we introduce the tools and measures for analyzing a given alpha factors effectiveness at predicting future returns. As discussed in section 2.2, alpha factors express a predictive relationship between the set of information and future returns. It is important to note that these measure merely indicate a statistical relation and have to be rigorously back-tested before being able to implement them.

3.3.1 Information Coefficient

The information coefficient ("IC") provides a measure for the predictiveness of the alpha factor. It describes the correlation between predicted and actual stock returns. The values of the IC range between $[-1; 1]$ where -1 describes no correct prediction and +1 describes perfect prediction of future returns. As a rule of thumb any value above 0 is satisfactory and a mean value above 0.1 is excellent.

3.3.2 Return Analysis

To analyze the factors ability to generate returns, we analyze the historical performance using historical stock prices of assets. Since historical pricing information is public and easily accessible, and U.S. public companies are required by law to submit quarterly reports (which we base our factors on), the universe of stocks is large enough universe of stocks that we can use for analysis.

Our general methodology is to use the factors to rank the assets in 30 day periods and divide them into quintiles where we long the best performing quintile (the stocks with the highest similarity and thus the least changes) and short the bottom quintile (the stocks with the lowest similarity and thus many changes). We use mean period-wise return by factor quintiles, and cumulative return measures.

4 Results and Discussion

In this section we present the results of the experimentation. The main aim of this paper was to explore the factor predictiveness of textual similarity measures in predicting future returns after the main paper by Cohen et al. (2018) was published in 2018 using data from 1994 - 2015. In this paper we, apply the same factors to a dataset of U.S. public companies' 10-K and 10-Q reports from 2015 to 2018. The results are presented in two subsections, one for the Jaccard Similarity measure and one for the Cosine similarity measure. Please see the appendix in section 5 for the documentation and code used to compute the graphs in the following subsections.

4.1 Jaccard Similarity

See figures 1 - 5

4.2 Cosine Similarity

See figures 6 - 10

4.3 Discussion

Analyzing the results from sections 4.1 and 4.2 we can make several observations. Both, the jaccard similarity factor and the cosine similarity factor demonstrated relatively good information coefficients

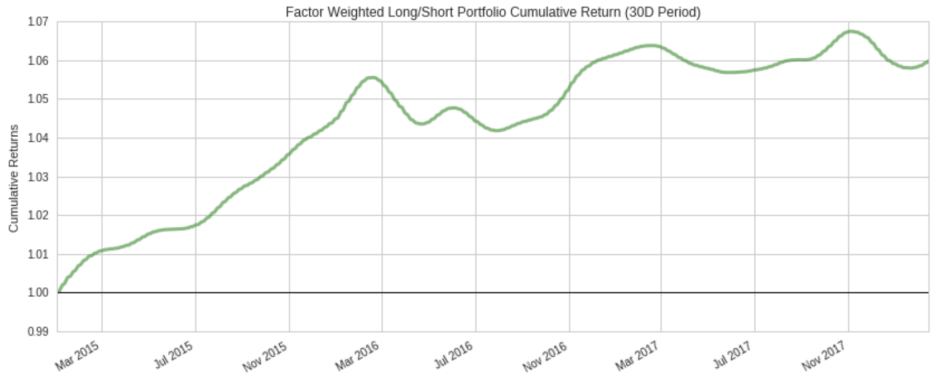


Figure 1: This graph shows the cumulative return of the factor weighted long/short portfolio over 30 day periods. We see that between March of 2015 and March 2016 the cumulative return is 6%. This result corresponds to the average range of cumulative returns of the factor in the years 1994-2015 as found by Cohen et al. (2018). However the subsequent year starting in March of 2016, the returns seem to flatten out to 2%.

Figure 2: On this bar plot we see the mean period-wise return by factor quantile, where the quantile number 1, represents the stocks that had the lowest similarity scores, meaning that they had the largest changes. We note that the absolute value of the mean return for the bottom quantile (around 25 basis points) exceeds the returns from going long the stocks with the highest factor values (20 basis points).

and subsequently were able to generate 6-7% return in the time period of March 2015 to March 2016 when using a long/short strategy. After March 2016 their performance of the two factors diverged, in that the cosine factor performed better than the jaccard similarity factor but still not as well as they did in the previous year or as found in the main paper by Cohen et al. (2018). We hypothesize that the cosine similarity factor performs better because it also takes into account word count and term frequencies, which adds more fine-grained information especially when considering negative words. In conclusion, we can hypothesisze that the predictive power of the textual similarity factor has subsided since the publishing of the paper. However, to confirm this we would have to continue to check the factor effectiveness in the coming years.

For future work, I would want to further explore the textual similarity between subsections of the regulatory reports, since Cohen et al. (2018) found that the Risk section was especially indicative of future performance.

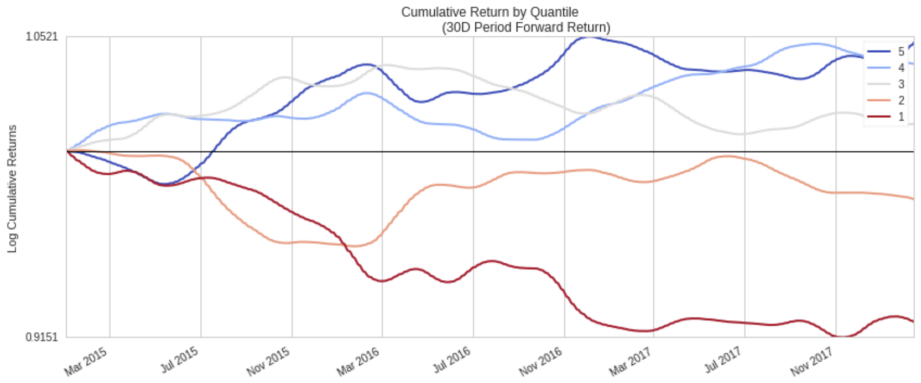


Figure 3: This graph shows the cumulative return of the factor weighted long/short portfolio in 30 day period by quantile. The graph shows nicely how there is a large spread between the bottom quantile (red) goes down, and the top quantile that goes up (Blue).

	30D	60D	90D
IC Mean	0.020	0.028	0.030
IC Std.	0.047	0.041	0.039
Risk-Adjusted IC	0.425	0.684	0.760
t-stat(IC)	11.698	18.816	20.887
p-value(IC)	0.000	0.000	0.000
IC Skew	-0.334	0.178	0.269
IC Kurtosis	-0.031	-0.681	-0.857

Figure 4: The table illustrates the information coefficient mean and distribution over time periods of 30, 60, and 90 days. We note that a number above zero indicates the factors ability to predict future returns. As a rule of thumb a value above zero is satisfactory and a value above 0.1 is exceptional.

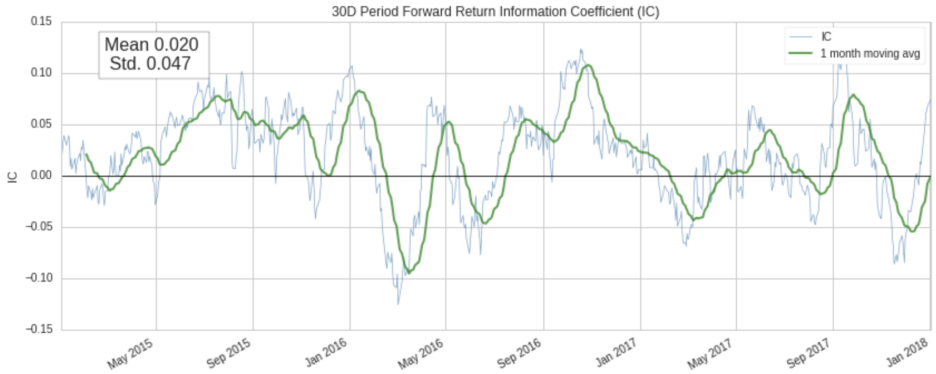


Figure 5: The time series shows the the 30 day revolving mean and standard deviation over time.

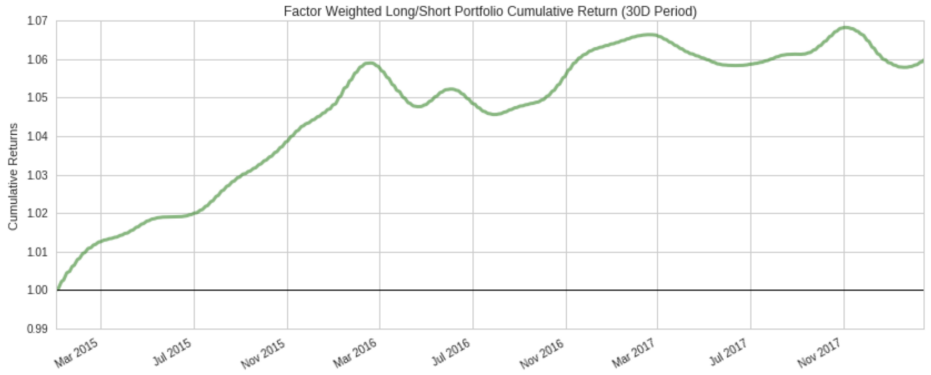


Figure 6: On the graph we see the cumulative return of the factor weighted long/short portfolio over 30 day periods for the cosine similarity measure. We see that between March of 2015 and March 2016 the cumulative return is 6%. This result corresponds to the average range of cumulative returns of the factor in the years 1994-2015 as found by Cohen et al. (2018).

Figure 7: On this bar plot we see the mean period-wise return by factor quantile, where the quantile number 1, represents the stocks that had the lowest similarity scores, meaning that they had the largest changes. We note that the absolute value of the mean return for the bottom quantile (around 25 basis points) exceeds the returns from going long the stocks with the highest factor values (20 basis points).

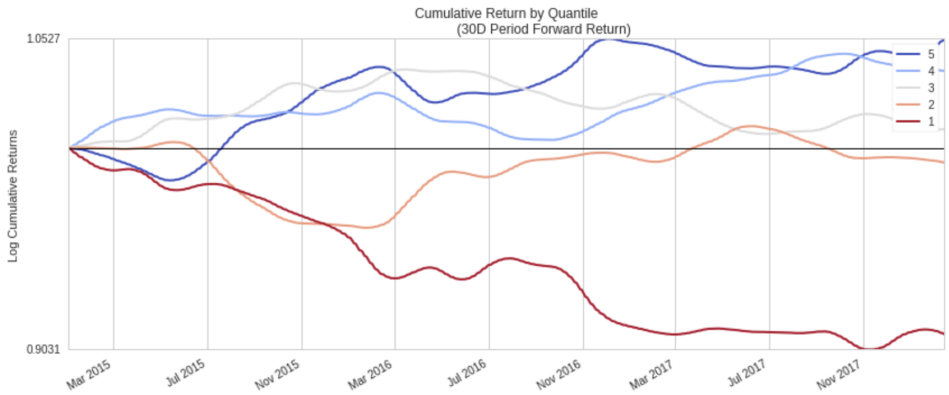


Figure 8: This graph shows the cumulative return of the factor weighted long/short portfolio in 30 day period by quantile. The graph shows nicely how there is a large spread between the bottom quantile (red) goes down, and the top quantile that goes up (Blue).

	30D	60D	90D
IC Mean	0.020	0.028	0.030
IC Std.	0.047	0.041	0.040
Risk-Adjusted IC	0.427	0.685	0.765
t-stat(IC)	11.730	18.842	21.032
p-value(IC)	0.000	0.000	0.000
IC Skew	-0.331	0.184	0.281
IC Kurtosis	-0.069	-0.682	-0.864

Figure 9: The table illustrates the information coefficient mean and distribution over time periods of 30, 60, and 90 days. We note that a number above zero indicates the factors ability to predict future returns. As a rule of thumb a value above zero is satisfactory and a value above 0.1 is exceptional.

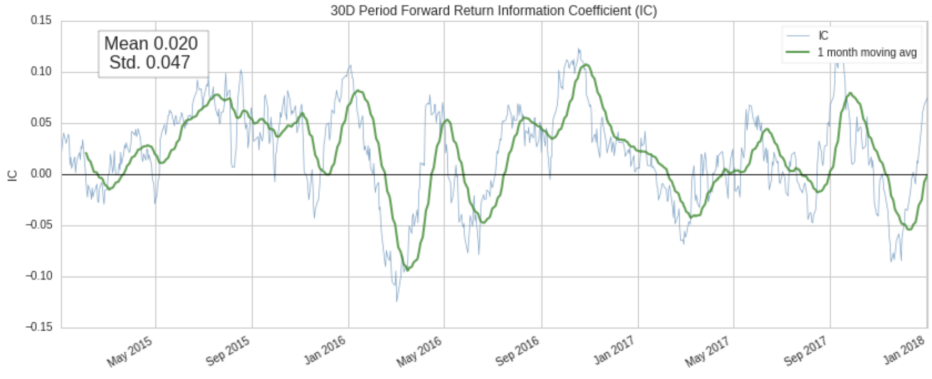


Figure 10: The time-series graph shows the mean IC measure over time.

References

Lauren Cohen, Christopher Malloy, and Quoc Nguyen. Lazy Prices. Working Paper 25084, National Bureau of Economic Research, sep 2018. URL http://www.nber.org/papers/w25084.

Travis Dyer, Mark Lang, and Lorien Stice-Lawrence. The evolution of 10-K textual disclosure: Evidence from Latent Dirichlet Allocation. *Journal of Accounting and Economics*, 64(2):221–245, 2017. ISSN 0165-4101. doi: https://doi.org/10.1016/j.jacceco.2017.07.002. URL http://www.sciencedirect.com/science/article/pii/S0165410117300484.

Peter D Easton and Mark E Zmijewski. SEC Form 1OK / 1OQ Reports and Annual Reports to Shareholders : Reporting Lags and Squared. 31(1):113–129, 1993.

Eugene F Fama. Efficient Capital Markets: A Review of Theory and Empirical Work. *The Journal of Finance*, 25(2):383–417, 1970. ISSN 00221082, 15406261. doi: 10.2307/2325486. URL http://www.jstor.org/stable/2325486.

David Hirshleifer, Siew Hong Teoh, Natasha Burns, Dick Dietrich, John Fellingham, Gerry Garvey, Jack Hirshleifer, Jack Hughes, Eugene Kandel, Sonya Seongyeon Lim, Bruce Miller, Ro Verrecchia, and Jerry Zimmerman. Limited attention , information disclosure , and financial reporting. 36: 337–386, 2003. doi: 10.1016/j.jacceco.2003.10.002.

Kuspa Kai, Victor Cheung, and Alex Lin. Lazy Prices: Vector Representations of Financial Disclosures and Market Outperformance, 2017.

Yen-Jung Lee. The Effect of Quarterly Report Readability on Information Efficiency of Stock Prices*. *Contemporary Accounting Research*, 29(4):1137–1170, dec 2012. ISSN 0823-9150. doi: 10.1111/j.1911-3846.2011.01152.x. URL https://doi.org/10.1111/j.1911-3846.2011.01152.x.

Stephen A Ross. The Arbitrage Theory of Capital Asset Pricing. pages 341–360, 1976.

U.S. Securities and Exchange Comission. Form 10-K & 10-Q. URL http://www.sec.gov/edgar.shtml.

Combining Value. Combining value and momentum. 14(2):33–48, 2016.

Haifeng You Æ Xiao-jun Zhang. Financial reporting complexity and investor underreaction to 10-K information. pages 559–586, 2009. doi: 10.1007/s11142-008-9083-2.

5 Appendix

5.1 Code

All the necessary code is available under the following link: https://github.com/jermann/gcm_project. The code and all the graphs is also made available in the appendix of this paper.

Global Capital Markets Project

Project by Alexander Jermann, Columbia University under supervision of Prof. Dr. Siddhartha Dastidar, Columbia University

Table of Contents

This notebook was used to pre-process all the 10-K and 10-Q reports in order to compute the similarity measures between subsequent 10-K and 10-Q reports. Running this notebook took a lot longer than I had anticipated for various reasons. First, the 10-K and 10-Q reports for over 2000 companies over 10 years took around 4 hours to download and another 6 hours to unzip. Second, the dataset requires around 60-100 gigabytes of storage. Third, computing similarity measures between so many documents also requires around 4 hours. For convenience, I saved all the intermediary steps as a CSV file and will save them together with all the datasets to a USB drive that I will also hand-in.

This notebook contains all the code that was used in this project to:

1. List of all U.S. equities.
2. Map the U.S. Stocks to the Central Index Key (CIK) that the S.E.C. uses internally.
3. Compilation of a CSV containing all the 10-K and 10-Q file summaries and clean-up statistics.
4. Defining Similarity measures
5. Computing similarities between 10-K reports
6. Computing similarities between 10-Q reports (to corresponding quarter of the following year)
7. Transposing statistics in a appropriate format for Alphawise.

Note: The second part of the project is in a second notebook which is run on a platform Quantopian, that allows to run the factor models online. Please see Notebook 2.

Data

The data can be downloaded from the following website https://sraf.nd.edu/data/stage-one-10-x-parse-data/ (https://sraf.nd.edu/data/stage-one-10-x-parse-data/)

In [1]:
```
# Importing built-in libraries (no need to install these)
import re
import os
from time import gmtime, strftime
from datetime import datetime, timedelta
import unicodedata

# Importing libraries you need to install
import pandas as pd
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
from iexfinance.stocks import get_historical_data
import requests
import bs4 as bs
from lxml import html
from tqdm import tqdm
```

Get all Stock tickers of US Equities

In this section we get a list of the stock tickers of all US Equities. More specifically, a list of all stocks (historical and present) traded on the New York Stock Exchange ("NYSE") on the National Association of Securities Dealers ("NASDAQ"), and the American Stock Exchange ("AMEX"). We do that by downloading the CSV's from the respective sites and then loading them using Pandas dataframes.

CSV downloaded from respective websites (Please note: website address might change in the future):

- NASDQ: https://old.nasdaq.com/screening/companies-by-name.aspx?letter=0&exchange=nasdaq&render=download (https://old.nasdaq.com/screening/companies-by-name.aspx?letter=0&exchange=nasdaq&render=download)
- AMEX: https://old.nasdaq.com/screening/companies-by-name.aspx?letter=0&exchange=amex&render=download (https://old.nasdaq.com/screening/companies-by-name.aspx?letter=0&exchange=amex&render=download)
- NYSE:https://old.nasdaq.com/screening/companies-by-name.aspx?letter=0&exchange=nyse&render=download (https://old.nasdaq.com/screening/companies-by-name.aspx?letter=0&exchange=nyse&render=download)

In [74]:
```
nasdaq = pd.read_csv('symbols_nasdaq.csv')
nyse = pd.read_csv('symbols_nyse.csv')
amex = pd.read_csv('symbols_amex.csv')
```

In [ ]:
```
nasdaq.head()
```

In [ ]:
```
nyse.head()
```

In [ ]:
```
amex.head()
```

In [78]:
```
# Delete uneeded columns
nasdaq.drop(labels='Unnamed: 8', axis='columns', inplace=True)
nyse.drop(labels='Unnamed: 8', axis='columns', inplace=True)
amex.drop(labels='Unnamed: 8', axis='columns', inplace=True)

# Merge all into a single list
tickers = list(set(list(nasdaq['Symbol']) + list(nyse['Symbol']) + list(
amex['Symbol'])))
```

In [80]:
```
# Example of 10 items from list
tickers[:10]
```

Out[80]:
```
['ELOX', 'VCLT', 'DS^B', 'UNTY', 'FR', 'KERN', 'MIC', 'MET', 'RFIL', 'B
A']
```

Map Stock Symbols to SEC CIK

This next section downloads the list of stock symbols to CIK mapping.

In [81]:
```
def MapStockToCik(tickers):
    url = 'http://www.sec.gov/cgi-bin/browse-edgar?CIK={}&Find=Search&ow
ner=exclude&action=getcompany'
    cik_re = re.compile(r'.*CIK=(\d{10}).*')

    cik_dict = {}
    for ticker in tqdm(tickers):
        results = cik_re.findall(requests.get(url.format(ticker)).text)
        if len(results):
            cik_dict[str(ticker).lower()] = str(results[0])

    return cik_dict
```

Note: the below code cell, took over an hour to run. For convenience saved results as a CSV file titled: `'ticker_cik.scv'`

In [84]:
```
cik_dict = MapStockToCik(tickers)
```

```
100%|██████████| 6989/6989 [1:08:21<00:00,  1.70it/s]
```

In [86]:
```
# Clean-up the mapping as a DataFrame
ticker_cik_df = pd.DataFrame.from_dict(data=cik_dict, orient='index')
ticker_cik_df.reset_index(inplace=True)
ticker_cik_df.columns = ['ticker', 'cik']
ticker_cik_df['cik'] = [str(cik) for cik in ticker_cik_df['cik']]
```

In [ ]:
```
ticker_cik_df
```

In [88]:
```
ticker_cik_df.to_csv('ticker_cik.csv')
```

In [89]:
```
print("Number of ticker-cik pairings:", len(ticker_cik_df))
print("Number of unique tickers:", len(set(ticker_cik_df['ticker'])))
print("Number of unique CIKs:", len(set(ticker_cik_df['cik'])))
```

```
Number of ticker-cik pairings: 5040
Number of unique tickers: 5040
Number of unique CIKs: 4851
```

In [92]:
```
ticker_cik_df = ticker_cik_df.sort_values(by='ticker')
ticker_cik_df.drop_duplicates(subset='cik', keep='first', inplace=True)
```

In [93]:
```
print("Number of ticker-cik pairings:", len(ticker_cik_df))
print("Number of unique tickers:", len(set(ticker_cik_df['ticker'])))
print("Number of unique CIKs:", len(set(ticker_cik_df['cik'])))
```

```
Number of ticker-cik pairings: 4851
Number of unique tickers: 4851
Number of unique CIKs: 4851
```

In [385]:
```
ticker_cik_df['cik_short'] = ticker_cik_df['cik'].str.lstrip("0")
```

In [387]:
```
ticker_cik_df
```

Out[387]:

	ticker	cik	cik_short
2025	a	0001090872	1090872
462	aa	0001675149	1675149
2159	aacg	0001420529	1420529
3011	aal	0000006201	6201
3896	aamc	0001555074	1555074
...	...	...	...
4179	zumz	0001318008	1318008
5011	zuo	0001423774	1423774
1415	zvo	0001305323	1305323
943	zyme	0001403752	1403752
2583	zyne	0001621443	1621443

4851 rows × 3 columns

Save the cleaned-up mapping as a CSV file.

In [386]:
```
ticker_cik_df.to_csv('ticker_cik_clean.csv')
```

Compile File Summary

In this section we compile a file summary that maps the stock ticker to the CIK and to the location of the file on the disk. This file also contains information on the filing date, the form type (i.e. 10-K or 10-Q), information on the sentiment of the documents (more detail to this in later section), and statistics on the clean-up (i.e. how many tables and exhibits, how many HTML tags were removed etc.). For a full list see the command `df1.columns` below.

In [ ]:
```
df1 = pd.read_csv('~/Desktop/LM_10X_Summaries_2018.csv')
```

In [ ]:
```
df1.columns
```

In [ ]:
```
filename_column = df1['FILE_NAME'].str.slice_replace(start=0, stop=16, r
epl='/Volumes/Alexander/Columbia/')
filename_column = filename_column.str.replace("\\", "/")
```

In [ ]:
```
df1['FILE_LOCATION'] = filename_column
```

In [ ]:
```
df1.drop('FILE_NAME', axis=1, inplace=True)
```

In [ ]:
```
df1.to_csv('10X_Summaries_AJ.csv')
```

To read file we have ro specify `index_col` as follows:

In [108]:
```
fs = pd.read_csv('10X_Summaries_AJ.csv', index_col=0)
```

In [ ]:
```
fs.head()
```

In [ ]:
```
fs[fs['FILING_DATE'] > 20110000]
```

Similarity Measures

This section defines the Jaccard and Cosine similarity measures between two words. We define them as follows:

Jaccard Similarity Index: Let A and B be words, then the Jaccard Similarity score is

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

where $0 \geq J(A,B) \geq 1$. If both $A, B = \emptyset$ we define $J(A,B) = 1$.

Cosine Similarity Measure: Let A and B be words. We map A and B into the vector space S, where S has dimension of the union between the word sets A and B

The dimensions of the vectorspace are the set of the union between A and B. The vector $A \in$

$$C(A,B) = \frac{\mathbf{A} \cdot \mathbf{B}}{|| \mathbf{A} || \cdot || \mathbf{B} ||}$$

where the nominator is the dot product between the vectors $\mathbf{A}$ and $\mathbf{B}$ and the denominator is the Euclidean norm.

Inputs

- whole documents
- compare 10ks
- compare 10qs

Using word sets

In [307]:
```
def jaccard_similarity(words_A, words_B):

    # Count number of words in both A and B
    words_intersect = len(words_A.intersection(words_B))

    # Count number of words in A or B
    words_union = len(words_A.union(words_B))

    # Compute Jaccard similarity score
    jaccard_score = words_intersect / words_union

    return jaccard_score
```

In [220]:
```
def c_cosine_similarity(words_A, words_B):

    # Get the union of words between A and B
    words = list(words_A.union(words_B))

    # Figure out which words are in A
    vector_A = [1 if x in words_A else 0 for x in words]

    # Figure out which words are in B
    vector_B = [1 if x in words_B else 0 for x in words]

    # Calculate cosine score using scikit-learn package
    array_A = np.array(vector_A).reshape(1, -1)
    array_B = np.array(vector_B).reshape(1, -1)
    cosine_score = cosine_similarity(array_A, array_B)[0,0]

    return cosine_score
```

As an exanple of how the two similarity measures work, let's consider the following three sentences as sets of words:

In [221]:
```
d_a = set(['we', 'expect', 'demand', 'to', 'increase','increase'])
d_b = set(['we', 'expect', 'worldwide', 'demand', 'to', 'increase'])
d_c = set(['we', 'expect', 'weakness', 'in', 'sales'])
```

In [308]:
```
print("Cosine similarity between A and B:", c_cosine_similarity(d_a, d_b
))
print("Cosine similarity between A and C:", c_cosine_similarity(d_a, d_c
))
print("Jaccard similarity between A and B:", ComputeJaccardSimilarity(d_
a, d_b))
print("Jaccard similarity between A and C:", ComputeJaccardSimilarity(d_
a, d_c))
```

```
Cosine similarity between A and B: 0.912870929175277
Cosine similarity between A and C: 0.39999999999999997
Jaccard similarity between A and B: 0.8333333333333334
Jaccard similarity between A and C: 0.25
```

Using TF-IDF

TF-IDF: stands for term frequency - inverse document frequency

In [217]:
```
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [219]:
```
vectorizer = TfidfVectorizer()
vectorizer.fit_transform(d_a)

vectorizer
```

Out[219]:
```
TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
                dtype=<class 'numpy.float64'>, encoding='utf-8',
                input='content', lowercase=True, max_df=1.0, max_featur
es=None,
                min_df=1, ngram_range=(1, 1), norm='l2', preprocessor=N
one,
                smooth_idf=True, stop_words=None, strip_accents=None,
                sublinear_tf=False, token_pattern='(?u)\\b\\w\\w+\\b',
                tokenizer=None, use_idf=True, vocabulary=None)
```

In [ ]:
```
# Scikit Learn
from sklearn.feature_extraction.text import CountVectorizer
import pandas as pd

# Create the Document Term Matrix
count_vectorizer = CountVectorizer(stop_words='english')
count_vectorizer = CountVectorizer()
sparse_matrix = count_vectorizer.fit_transform(documents)
```

Computing Similarities for 10Ks

Here we get a CSV file that contains information and the location of all the necessary files.

In [72]:
```
fs = pd.read_csv('10X_Summaries_AJ.csv', index_col=0)
```

```
/Users/alexander/anaconda3/lib/python3.7/site-packages/numpy/lib/arrays
etops.py:569: FutureWarning: elementwise comparison failed; returning s
calar instead, but in the future will perform elementwise comparison
  mask |= (ar1 == a)
```

In [ ]:
```
fs[(fs['FORM_TYPE'].str.startswith('10-K')) & (fs['FILING_DATE'] > 20110
000)]
```

In [366]:
```
ticker_cik_df
```

Out[366]:

	ticker	cik
2025	a	0001090872
462	aa	0001675149
2159	aacg	0001420529
3011	aal	0000006201
3896	aamc	0001555074
...	...	...
4179	zumz	0001318008
5011	zuo	0001423774
1415	zvo	0001305323
943	zyme	0001403752
2583	zyne	0001621443

4851 rows × 2 columns

In [416]:
```
filenames = fs[(fs['CIK'] == 1675149) & (fs['FORM_TYPE'].str.startswith(
'10-K')) & (fs['FILING_DATE'] > 20110000)]['FILE_LOCATION']
```

In [409]:
```
filenames.sort()
```

In [417]:
```
filenames
```

Out[417]:
```
977832     /Volumes/Alexander/Columbia/2017/QTR1/20170315...
1003695    /Volumes/Alexander/Columbia/2018/QTR1/20180226...
Name: FILE_LOCATION, dtype: object
```

In [ ]:
```
fs[fs['CIK'].isin([cik])]
```

In [443]:
```
def computeSimilarity10K(cik, start_date=20110000):

    path = '/Users/alexander/git/gcm/'
    os.chdir(path)

    # Get filenames for given CIK
    filenames = fs[(fs['CIK'].isin([cik])) & (fs['FORM_TYPE'].str.starts
with('10-K')) & (fs['FILING_DATE'] > 20110000)]['FILE_LOCATION']
    filenames = filenames.tolist()

    filenames.sort()

    # check if scores have already been calculated
    if os.path.exists('/Users/alexander/git/gcm/metrics/' + str(cik) +
'_sim_scores.csv'):
        return

    # Check enough files
    if len(filenames) < 2:
        return

    # Initialize dataframe to store sim scores
    dates = [x[38:46] for x in filenames]
    cosine_score = [0]*len(dates)
    jaccard_score = [0]*len(dates)
    data = pd.DataFrame(columns={'cosine_score': cosine_score,
                                 'jaccard_score': jaccard_score},
                        index=dates)

    doc1_loc = filenames[0]

    with open(doc1_loc, 'r') as file:
        doc1_text = file.read()

    for i in range(1, len(filenames)): #

        doc2_loc = filenames[i]

        with open(doc2_loc, 'r') as file:
            doc2_text = file.read()

        # Get set of words in A, B
        words_A = set(re.findall(r"[\w']+", doc1_text))
        words_B = set(re.findall(r"[\w']+", doc2_text))

        # Calculate similarity scores
        cosine_score = c_cosine_similarity(words_A, words_B)
        jaccard_score = jaccard_similarity(words_A, words_B)

        # Store score values
        date_B = doc2_loc[38:46]
        data.at[date_B, 'cosine_score'] = cosine_score
        data.at[date_B, 'jaccard_score'] = jaccard_score

        doc1_text = doc2_text
```

```
    # Save to csv data
    os.chdir('/Users/alexander/git/gcm/metrics')
    data.to_csv(str(cik) + '_sim_scores.csv', index=False)
    os.chdir('/Users/alexander/git/gcm/')

    # Read in second 10K
    # compute similarity
    # return similarity score
```

In [422]:
```
computeSimilarity10K(60512)
```

```
60512
Series([], Name: FILE_LOCATION, dtype: object)
```

In [444]:
```
tqdm._instances.clear()
```

In [445]:
```
#ticker_cik_df['cik']

cik_list = fs[(fs['FORM_TYPE'].str.startswith('10-K')) & (fs['FILING_DAT
E'] > 20110000)]['CIK']

for cik in tqdm(ticker_cik_df['cik_short']):
    computeSimilarity10K(cik)
```

```
100%|██████████| 4851/4851 [1:17:46<00:00,  1.18s/it]
```

Computing Similarities for 10-Qs

In [467]:
```
filenames = fs[(fs['CIK'].isin([1675149])) & (fs['FORM_TYPE'].str.starts
with('10-Q')) & (fs['FILING_DATE'] > 20110000)]['FILE_LOCATION']
```

In [468]:
```
[datetime.strptime(x[38:46], '%Y%m%d').strftime('%Y-%m-%d') for x in fil
enames]
```

Out[468]:
```
['2016-12-01',
 '2017-05-10',
 '2017-08-03',
 '2017-10-27',
 '2018-05-09',
 '2018-08-02',
 '2018-11-02']
```

In [475]:
```
def computeSimilarity10Q(cik, start_date=20110000):

    # Define how stringent we want to be about
    # "previous year"
    year_short = timedelta(345)
    year_long = timedelta(385)

    # Set path
    path = '/Users/alexander/git/gcm/metrics'
    os.chdir(path)

    # Get filenames for given CIK
    filenames = fs[(fs['CIK'].isin([cik])) & (fs['FORM_TYPE'].str.starts
with('10-Q')) & (fs['FILING_DATE'] > 20110000)]['FILE_LOCATION']
    filenames = filenames.tolist()
    filenames.sort()

    # check if scores have already been calculated
    if os.path.exists('/Users/alexander/git/gcm/metrics/' + str(cik) +
'_sim_scores.csv'):
        return

    # Check if enough files exist to compare
    # ... if there aren't enough files, exit
    if len(filenames) < 4:
        #print("No files to compare for CIK", cik)
        os.chdir('../..')
        return

    # Initialize dataframe to hold similarity scores
    dates = [datetime.strptime(x[38:46], '%Y%m%d').strftime('%Y-%m-%d')
for x in filenames]
    cosine_score = [0]*len(dates)
    jaccard_score = [0]*len(dates)
    data = pd.DataFrame(columns={'cosine_score': cosine_score,
                                 'jaccard_score': jaccard_score},
                        index=dates)

    # Iterate over each quarter...
    for j in range(3):

        # Get text and date of earliest filing from that quarter
        file_name_A = filenames[j]
        with open(file_name_A, 'r') as file:
            file_text_A = file.read()
        date_A = datetime.strptime(file_name_A[38:46], '%Y%m%d')

        # Iterate over the rest of the filings from that quarter...
        for i in range(j+3, len(filenames), 3):

            # Get name and date of the later file
            file_name_B = filenames[i]
            date_B = datetime.strptime(file_name_B[38:46], '%Y%m%d')

            # If B was not filed within ~1 year after A...
            if (date_B > (date_A + year_long)) or (date_B < (date_A + ye
```

```
ar_short)):

                #print(date_B.strftime('%Y-%m-%d'), "is not within a yea
r of", date_A.strftime('%Y-%m-%d'))

                # Record values as NaN
                data.at[date_B.strftime('%Y-%m-%d'), 'cosine_score'] =
'NaN'
                data.at[date_B.strftime('%Y-%m-%d'), 'jaccard_score'] =
'NaN'

                # Pretend as if we found new date_A in the next year
                date_A = date_A.replace(year=date_B.year)

                # Move to next filing
                continue

            # If B was filed within ~1 year of A...

            # Get file text
            with open(file_name_B, 'r') as file:
                file_text_B = file.read()

            # Get sets of words in A, B
            words_A = set(re.findall(r"[\w']+", file_text_A))
            words_B = set(re.findall(r"[\w']+", file_text_B))

            # Calculate similarity score
            cosine_score = c_cosine_similarity(words_A, words_B)
            jaccard_score = jaccard_similarity(words_A, words_B)

            # Store value (indexing by the date of document B)
            data.at[date_B.strftime('%Y-%m-%d'), 'cosine_score'] = cosin
e_score
            data.at[date_B.strftime('%Y-%m-%d'), 'jaccard_score'] = jacc
ard_score

            # Reset value for next loop
            # Don't re-read files, for efficiency
            file_text_A = file_text_B
            date_A = date_B

    # Save scores
    data.to_csv(str(cik)+'_sim_scores.csv', index=True)
```

In [476]:
```
computeSimilarity10Q(1675149)
```

In [483]:
```
tqdm._instances.clear()
```

In [478]:
```
for cik in tqdm(ticker_cik_df['cik_short']):
    computeSimilarity10Q(cik)
```

```
  0%|          | 1/4851 [00:00<59:35,  1.36it/s]
100%|██████████| 4851/4851 [1:55:51<00:00,  1.26s/it]
```

In [479]:
```
def get_data(cik, pathname_data):

    data_10k = True
    data_10q = True

    path = '/Users/alexander/git/gcm/10K_metrics'
    os.chdir(path)

    try:
        df_10k_sim_score = pd.read_csv(str(cik) + '_sim_scores.csv')
    except FileNotFoundError:
        data_10k = False


    path = '/Users/alexander/git/gcm/10Q_metrics'
    os.chdir(path)

    try:
        df_10q_sim_score = pd.read_csv(str(cik) + '_sim_scores.csv')
    except FileNotFoundError:
        data_10q = False


    if not (data_10k and data_10q):
        return

    if not data_10q:
        sim_scores = df_10k_sim_score

    elif not data_10k:
        sim_scores = df_10q_sim_score

    elif (data_10q and data_10k):
        sim_scores = pd.concat([df_10k_sim_score, df_10q_sim_score], axi
s='index')


    sim_scores.rename(columns={'Unnamed: 0': 'date'}, inplace=True)


    sim_scores['cik'] = cik

    # Save file in the data dir
    os.chdir(pathname_data)
    sim_scores.to_csv('%s_sim_scores_full.csv' % cik, index=False)

    return
```

In [480]:
```
pathname_data = '/Users/alexander/git/gcm/metrics_merged'
```

In [484]:
```
for cik in tqdm(ticker_cik_df['cik_short']):
    get_data(cik, pathname_data)
```

```
  0%|          | 0/4851 [00:00<?, ?it/s]/Users/alexander/anaconda3/lib/
python3.7/site-packages/ipykernel_launcher.py:34: FutureWarning: Sortin
g because non-concatenation axis is not aligned. A future version
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=Tru
e'.

100%|██████████| 4851/4851 [00:38<00:00, 124.68it/s]
```

In [485]:
```
def MakeDataset(file_list, pathname_full_data):

    '''
    Consolidates CIK datasets into a
    single dataset.

    Parameters
    ----------
    file_list : list
        List of .csv files to merge.
    pathname_full_data : str
        Path to directory to store
        full dataset.

    Returns
    -------
    None.

    '''

    # Initialize dataframe to store results
    data = pd.DataFrame(columns=['date', 'cosine_score', 'jaccard_score'
, 'cik'])

    # Iterate over files and merge all together
    for file_name in tqdm(file_list):
        new_data = pd.read_csv(file_name)
        data = data.append(new_data, sort=True)

    # Store result
    os.chdir(pathname_full_data)
    data.to_csv('all_sim_scores.csv', index=False)

    return
```

In [486]:
```
pathname_full_data = '/Users/alexander/git/gcm/final'
```

In [487]:
```
os.chdir(pathname_data)
file_list = [fname for fname in os.listdir() if not fname.startswith('.'
)]

MakeDataset(file_list, pathname_full_data)
```

```
100%|██████████| 3478/3478 [01:05<00:00, 53.26it/s]
```

In [494]:
```
sim_scores_full = pd.read_csv('all_sim_scores.csv')

# Cast CIKs as strings
sim_scores_full['cik'] = [str(x) for x in sim_scores_full['cik']]

# Merge to map tickers to CIKs
sim_scores_ticker = sim_scores_full.merge(ticker_cik_df, how='left', lef
t_on='cik', right_on='cik_short')

# Drop CIK column
#sim_scores_ticker.drop(labels=['cik'], axis='columns', inplace=True)

# Drop NaN values
sim_scores_ticker.dropna(axis='index', how='any', subset=['jaccard_scor
e', 'cosine_score'], inplace=True)
```

In [495]:
```
sim_scores_full.head()
```

Out[495]:

	cik	cosine_score	date	jaccard_score
0	1356090	NaN	NaN	NaN
1	1356090	0.868523	NaN	0.766920
2	1356090	0.867265	NaN	0.764305
3	1356090	0.557528	NaN	0.348981
4	1356090	0.587213	NaN	0.348520

In [497]:
```
sim_scores_ticker
```

Out[497]:

	cik_x	cosine_score	date	jaccard_score	ticker	cik_y	cik_short
1	1356090	0.868523	NaN	0.766920	xon	0001356090	1356090
2	1356090	0.867265	NaN	0.764305	xon	0001356090	1356090
3	1356090	0.557528	NaN	0.348981	xon	0001356090	1356090
4	1356090	0.587213	NaN	0.348520	xon	0001356090	1356090
5	1356090	0.865759	NaN	0.763239	xon	0001356090	1356090
...	...	...	...	...	...	...	...
102803	100885	0.890377	2017-07-20	0.801700	unp	0000100885	100885
102804	100885	0.793234	2017-10-26	0.651991	unp	0000100885	100885
102805	100885	0.867204	2018-04-26	0.765320	unp	0000100885	100885
102806	100885	0.869045	2018-07-19	0.768045	unp	0000100885	100885
102807	100885	0.788089	2018-10-25	0.644855	unp	0000100885	100885

68442 rows × 7 columns

In [ ]:
```
sim_scores_ticker.to_csv('sim_scores_ticker.csv')
```

Splitting Dataset

Only run the following cell if reruning notebook:

In [9]:
```
sim_scores_ticker = pd.read_csv('sim_scores_ticker.csv', index_col = 0)
```

In [10]:
```
sim_scores_ticker.drop(labels=['cik_x'], axis='columns', inplace=True)
```

In [11]:
```
sim_scores_ticker.rename(columns={'cik_y':'cik'}, inplace=True)
```

In [14]:
```
sim_scores_ticker
```

Out[14]:

	cosine_score	date	jaccard_score	ticker	cik	cik_short
1	0.868523	NaN	0.766920	xon	1356090	1356090
2	0.867265	NaN	0.764305	xon	1356090	1356090
3	0.557528	NaN	0.348981	xon	1356090	1356090
4	0.587213	NaN	0.348520	xon	1356090	1356090
5	0.865759	NaN	0.763239	xon	1356090	1356090
...	...	...	...	...	...	...
102803	0.890377	2017-07-20	0.801700	unp	100885	100885
102804	0.793234	2017-10-26	0.651991	unp	100885	100885
102805	0.867204	2018-04-26	0.765320	unp	100885	100885
102806	0.869045	2018-07-19	0.768045	unp	100885	100885
102807	0.788089	2018-10-25	0.644855	unp	100885	100885

68442 rows × 6 columns

In [ ]:
```
sim_scores_ticker[sim_scores_ticker['ticker'] == 'amzn']
```

In [15]:
```
def InitializeEmptyDataframe(start_date, end_date, tickers):

    '''
    Initializes an empty DataFrame with all correct indices
    (1 entry/ticker/day)

    Parameters
    ----------
    start_date : datetime.datetime
        Start date of dataframe.
    end_date : datetime.datetime
        End date of dataframe.
    tickers : list
        List of tickers.
    '''

    window_length_days = int((end_date - start_date).days)
    date_list = [start_date+timedelta(days=x) for x in range(0, window_l
ength_days)]
    long_date_list = date_list * len(tickers)
    long_date_list = [x.strftime('%Y-%m-%d') for x in long_date_list]
    list.sort(long_date_list)
    empty = pd.DataFrame(data={'date': long_date_list,
                                       'ticker': tickers*len(date_list),
                                 'jaccard_score': [np.nan]*len(tickers)*l
en(date_list),
                               'cosine_score': [np.nan]*len(tickers)*len(
date_list)})
    empty = empty.groupby(['date', 'ticker']).sum()

    empty['jaccard_score'] = np.nan
    empty['cosine_score'] = np.nan

    return empty
```

Create Dataset from 2011-2015

In [75]:
```
# Initialize empty dataframe
start_date = datetime(2011, 1, 1)
end_date = datetime(2015, 1, 1)
tickers = list(set(sim_scores_ticker['ticker']))

empty_data = InitializeEmptyDataframe(start_date, end_date, tickers)
```

In [76]:
```
empty_data.head()
```

Out[76]:

		jaccard_score	cosine_score
date	**ticker**		
2011-01-01	a	NaN	NaN
	aa	NaN	NaN
	aal	NaN	NaN
	aamc	NaN	NaN
	aame	NaN	NaN

In [77]:
```
sim_scores_formatted = sim_scores_ticker.dropna(axis='index', how='any',
subset=['jaccard_score', 'cosine_score'])
sim_scores_formatted = sim_scores_formatted.groupby(['date', 'ticker']).
agg('mean')
```

In [78]:
```
sim_scores_formatted.head()
```

Out[78]:

		cosine_score	jaccard_score	cik	cik_short
date	**ticker**				
2011-12-22	camp	0.755000	0.603712	730255	730255
2011-12-30	cag	0.817486	0.691281	23217	23217
	neog	0.662253	0.465062	711377	711377
2012-01-03	bby	0.825220	0.699897	764478	764478
	dri	0.874617	0.776287	940944	940944

In [79]:
```
sim_scores_formatted.head()
```

Out[79]:

		cosine_score	jaccard_score	cik	cik_short
date	**ticker**				
2011-12-22	camp	0.755000	0.603712	730255	730255
2011-12-30	cag	0.817486	0.691281	23217	23217
	neog	0.662253	0.465062	711377	711377
2012-01-03	bby	0.825220	0.699897	764478	764478
	dri	0.874617	0.776287	940944	940944

In [80]:
```
formatted_data = empty_data.join(sim_scores_formatted, how='left', on=[
'date', 'ticker'], lsuffix='_empty')
formatted_data.drop(labels=['cosine_score_empty', 'jaccard_score_empty'
], axis='columns', inplace=True)
```

In [81]:
```
formatted_data.head()
```

Out[81]:

		cosine_score	jaccard_score	cik	cik_short
date	ticker				
2011-01-01	a	NaN	NaN	NaN	NaN
	aa	NaN	NaN	NaN	NaN
	aal	NaN	NaN	NaN	NaN
	aamc	NaN	NaN	NaN	NaN
	aame	NaN	NaN	NaN	NaN

In [82]:
```
forward_filled_data = formatted_data.reset_index().sort_values(by=['tick
er', 'date'])
```

In [ ]:
```
forward_filled_data
```

In [84]:
```
forward_filled_data.fillna(method='ffill', limit=90, inplace=True)
```

In [85]:
```
forward_filled_data.head()
```

Out[85]:

	date	ticker	cosine_score	jaccard_score	cik	cik_short
0	2011-01-01	a	NaN	NaN	NaN	NaN
3478	2011-01-02	a	NaN	NaN	NaN	NaN
6956	2011-01-03	a	NaN	NaN	NaN	NaN
10434	2011-01-04	a	NaN	NaN	NaN	NaN
13912	2011-01-05	a	NaN	NaN	NaN	NaN

In [ ]:
```
forward_filled_data[forward_filled_data['ticker'] == 'tsla']
```

In [87]:
```
del forward_filled_data['cik_short']
```

In [89]:
```
forward_filled_data[(forward_filled_data['ticker'] == 'amzn') & (forward
_filled_data['date'] >  '2013-01-01')]
```

Out[89]:

	date	ticker	cosine_score	jaccard_score	cik
2546086	2013-01-02	amzn	0.872645	0.774005	1018724.0
2549564	2013-01-03	amzn	0.872645	0.774005	1018724.0
2553042	2013-01-04	amzn	0.872645	0.774005	1018724.0
2556520	2013-01-05	amzn	0.872645	0.774005	1018724.0
2559998	2013-01-06	amzn	0.872645	0.774005	1018724.0
...	...	...	...	...	...
5064158	2014-12-27	amzn	0.887086	0.796856	1018724.0
5067636	2014-12-28	amzn	0.887086	0.796856	1018724.0
5071114	2014-12-29	amzn	0.887086	0.796856	1018724.0
5074592	2014-12-30	amzn	0.887086	0.796856	1018724.0
5078070	2014-12-31	amzn	0.887086	0.796856	1018724.0

729 rows × 5 columns

In [46]:
```
forward_filled_data.dtypes
```

Out[46]:
```
date              object
ticker            object
cosine_score     float64
jaccard_score    float64
cik              float64
cik_short        float64
dtype: object
```

In [50]:
```
forward_filled_data.to_csv('lazy_prices_data_2011_2015.csv', index=False
)
```

In [520]:
```
forward_filled_data.to_csv('lazy_prices_data.csv', index=False)
```

In [52]:
```
forward_filled_data[forward_filled_data['ticker'] == 'amzn']
```

Out[52]:

	date	ticker	cosine_score	jaccard_score	cik
190	2011-01-01	amzn	NaN	NaN	NaN
3668	2011-01-02	amzn	NaN	NaN	NaN
7146	2011-01-03	amzn	NaN	NaN	NaN
10624	2011-01-04	amzn	NaN	NaN	NaN
14102	2011-01-05	amzn	NaN	NaN	NaN
...	...	...	...	...	...
5064158	2014-12-27	amzn	0.887086	0.796856	1018724.0
5067636	2014-12-28	amzn	0.887086	0.796856	1018724.0
5071114	2014-12-29	amzn	0.887086	0.796856	1018724.0
5074592	2014-12-30	amzn	0.887086	0.796856	1018724.0
5078070	2014-12-31	amzn	0.887086	0.796856	1018724.0

1461 rows × 5 columns

Create Dataset from 2015-2018

In [90]:
```
# Initialize empty dataframe
start_date = datetime(2015, 1, 1)
end_date = datetime(2018, 1, 1)
tickers = list(set(sim_scores_ticker['ticker']))

empty_data = InitializeEmptyDataframe(start_date, end_date, tickers)
```

In [91]:
```
empty_data.head()
```

Out[91]:

		jaccard_score	cosine_score
date	ticker		
2015-01-01	a	NaN	NaN
	aa	NaN	NaN
	aal	NaN	NaN
	aamc	NaN	NaN
	aame	NaN	NaN

In [92]:
```
sim_scores_formatted = sim_scores_ticker.dropna(axis='index', how='any',
subset=['jaccard_score', 'cosine_score'])
sim_scores_formatted = sim_scores_formatted.groupby(['date', 'ticker']).
agg('mean')
```

In [93]:
```
sim_scores_formatted.head()
```

Out[93]:

		cosine_score	jaccard_score	cik	cik_short
date	**ticker**				
2011-12-22	camp	0.755000	0.603712	730255	730255
2011-12-30	cag	0.817486	0.691281	23217	23217
	neog	0.662253	0.465062	711377	711377
2012-01-03	bby	0.825220	0.699897	764478	764478
	dri	0.874617	0.776287	940944	940944

In [94]:
```
sim_scores_formatted.head()
```

Out[94]:

		cosine_score	jaccard_score	cik	cik_short
date	**ticker**				
2011-12-22	camp	0.755000	0.603712	730255	730255
2011-12-30	cag	0.817486	0.691281	23217	23217
	neog	0.662253	0.465062	711377	711377
2012-01-03	bby	0.825220	0.699897	764478	764478
	dri	0.874617	0.776287	940944	940944

In [95]:
```
formatted_data = empty_data.join(sim_scores_formatted, how='left', on=[
'date', 'ticker'], lsuffix='_empty')
formatted_data.drop(labels=['cosine_score_empty', 'jaccard_score_empty'
], axis='columns', inplace=True)
```

In [96]:
```
formatted_data.head()
```

Out[96]:

		cosine_score	jaccard_score	cik	cik_short
date	**ticker**				
2015-01-01	a	NaN	NaN	NaN	NaN
	aa	NaN	NaN	NaN	NaN
	aal	NaN	NaN	NaN	NaN
	aamc	NaN	NaN	NaN	NaN
	aame	NaN	NaN	NaN	NaN

In [97]:
```
forward_filled_data = formatted_data.reset_index().sort_values(by=['tick
er', 'date'])
```

In [98]:
```
forward_filled_data
```

Out[98]:

	date	ticker	cosine_score	jaccard_score	cik	cik_short
0	2015-01-01	a	NaN	NaN	NaN	NaN
3478	2015-01-02	a	NaN	NaN	NaN	NaN
6956	2015-01-03	a	NaN	NaN	NaN	NaN
10434	2015-01-04	a	NaN	NaN	NaN	NaN
13912	2015-01-05	a	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...
3797975	2017-12-27	zyne	NaN	NaN	NaN	NaN
3801453	2017-12-28	zyne	NaN	NaN	NaN	NaN
3804931	2017-12-29	zyne	NaN	NaN	NaN	NaN
3808409	2017-12-30	zyne	NaN	NaN	NaN	NaN
3811887	2017-12-31	zyne	NaN	NaN	NaN	NaN

3811888 rows × 6 columns

In [99]:
```
forward_filled_data.fillna(method='ffill', limit=90, inplace=True)
```

In [100]:
```
forward_filled_data.head()
```

Out[100]:

	date	ticker	cosine_score	jaccard_score	cik	cik_short
0	2015-01-01	a	NaN	NaN	NaN	NaN
3478	2015-01-02	a	NaN	NaN	NaN	NaN
6956	2015-01-03	a	NaN	NaN	NaN	NaN
10434	2015-01-04	a	NaN	NaN	NaN	NaN
13912	2015-01-05	a	NaN	NaN	NaN	NaN

In [101]:
```
forward_filled_data[forward_filled_data['ticker'] == 'tsla']
```

Out[101]:

	date	ticker	cosine_score	jaccard_score	cik	cik_short
3130	2015-01-01	tsla	0.812659	0.684436	1519061.0	1519061.0
6608	2015-01-02	tsla	0.812659	0.684436	1519061.0	1519061.0
10086	2015-01-03	tsla	0.812659	0.684436	1519061.0	1519061.0
13564	2015-01-04	tsla	0.812659	0.684436	1519061.0	1519061.0
17042	2015-01-05	tsla	0.812659	0.684436	1519061.0	1519061.0
...	...	...	...	...	...	...
3797628	2017-12-27	tsla	0.877555	0.781317	1318605.0	1318605.0
3801106	2017-12-28	tsla	0.877555	0.781317	1318605.0	1318605.0
3804584	2017-12-29	tsla	0.877555	0.781317	1318605.0	1318605.0
3808062	2017-12-30	tsla	0.877555	0.781317	1318605.0	1318605.0
3811540	2017-12-31	tsla	0.877555	0.781317	1318605.0	1318605.0

1096 rows × 6 columns

In [102]:
```
del forward_filled_data['cik_short']
```

In [106]:
```
forward_filled_data[forward_filled_data['ticker'] == 'tsla']
```

Out[106]:

	date	ticker	cosine_score	jaccard_score	cik
3130	2015-01-01	tsla	0.812659	0.684436	1519061.0
6608	2015-01-02	tsla	0.812659	0.684436	1519061.0
10086	2015-01-03	tsla	0.812659	0.684436	1519061.0
13564	2015-01-04	tsla	0.812659	0.684436	1519061.0
17042	2015-01-05	tsla	0.812659	0.684436	1519061.0
...	...	...	...	...	...
3797628	2017-12-27	tsla	0.877555	0.781317	1318605.0
3801106	2017-12-28	tsla	0.877555	0.781317	1318605.0
3804584	2017-12-29	tsla	0.877555	0.781317	1318605.0
3808062	2017-12-30	tsla	0.877555	0.781317	1318605.0
3811540	2017-12-31	tsla	0.877555	0.781317	1318605.0

1096 rows × 5 columns

In [104]:
```
forward_filled_data.dtypes
```

Out[104]:
```
date              object
ticker            object
cosine_score     float64
jaccard_score    float64
cik              float64
dtype: object
```

In [107]:
```
forward_filled_data.to_csv('lazy_prices_data_2015_2018.csv', index=False
)
```

Resources

- The dataset of 10-K and 10-Q reports was downloaded from https://sraf.nd.edu/data/stage-one-10-x-parse-data/ (https://sraf.nd.edu/data/stage-one-10-x-parse-data/)
- This notebook uses parts from https://www.quantopian.com/posts/scraping-10-ks-and-10-qs-for-alpha (https://www.quantopian.com/posts/scraping-10-ks-and-10-qs-for-alpha)

In [ ]:
```

```

Global Capital Markets Project

Project by Alexander Jermann, Columbia University under supervision of Prof. Dr. Siddhartha Dastidar, Columbia University

Note: This notebook is the second part of a two part series. Please refer to Notebook 1 for the first part.

1. Prepare Data for Model

In [2]:
```
from quantopian.research import run_pipeline
from quantopian.pipeline import Pipeline
from quantopian.pipeline.filters import QTradableStocksUS

import alphalens
```

As the first step we import the factor data that we saved in a CSV file in the previous notebook. To do that Quantopian has a page called Self-Serve Data that allows you to upload your own data. Once that is done, we can import it into our environment using the following command:

In [3]:
```
from quantopian.pipeline.data.user_5b5ea3e147c3ff0043f28115 import lazy_
prices_2015_2018
```

To use the data we use a Pipeline, which provides a tool for cross-sectional analysis of asset data. It allows us to make computation for many assets at the same time. In our case we use the similarity scores for many assets over time to rank assets.

In [4]:
```
def make_pipeline():

    jaccard_score = lazy_prices_2015_2018.jaccard_score.latest
    cosine_score = lazy_prices_2015_2018.cosine_score.latest

    screen = (QTradableStocksUS() & jaccard_score.notnull() & cosine_sco
re.notnull())

    return Pipeline(columns={'jaccard_score': jaccard_score, 'cosine_sco
re': cosine_score}, screen=screen)
```

We create two datasets one between

In [5]:
```
data = run_pipeline(make_pipeline(), '2015-01-01', '2018-01-01')
```

Pipeline Execution Time: 2 Minutes, 0.56 Seconds

In [6]:
```
data.tail(10)
```

Out[6]:

		cosine_score	jaccard_score
2018-01-02 00:00:00+00:00	Equity(50310 [DFIN])	0.773923	0.631219
	Equity(50312 [LKSD])	0.759163	0.609053
	Equity(50320 [ELF])	0.854101	0.740132
	Equity(50350 [COUP])	0.727309	0.548890
	Equity(50361 [ADSW])	0.871912	0.772911
	Equity(50366 [CWH])	0.604587	0.394254
	Equity(50368 [XOG])	0.806468	0.675054
	Equity(50376 [CDEV])	0.612506	0.436896
	Equity(50398 [FRTA])	0.792387	0.656128
	Equity(50533 [CNDT])	0.784652	0.641971

2. Prices of Assets

In [7]:
```
assets = data.index.levels[1]
```

In [8]:
```
pricing_end_date = '2018-08-01' # End date of pricing later so we can ge
t future returns
prices = get_pricing(assets,
                     start_date='2015-01-01',
                     end_date=pricing_end_date,
                     fields='open_price')
```

3. Factor Predictiveness Analysis

3.1 Jaccard Factor

In [27]:
```
jaccard_factor = data[['jaccard_score']]
```

In [28]:
```
factor_data_j = alphalens.utils.get_clean_factor_and_forward_returns(
    jaccard_factor,
    prices=prices,
    quantiles=5,
    periods =(30, 60, 90),
)
```

```
Dropped 0.1% entries from factor data: 0.1% in forward returns computat
ion and 0.0% in binning phase (set max_loss=0 to see potentially suppre
ssed Exceptions).
max_loss is 35.0%, not exceeded: OK!
```

In [29]:
```
alphalens.tears.create_full_tear_sheet(factor_data_j, by_group=False);
```

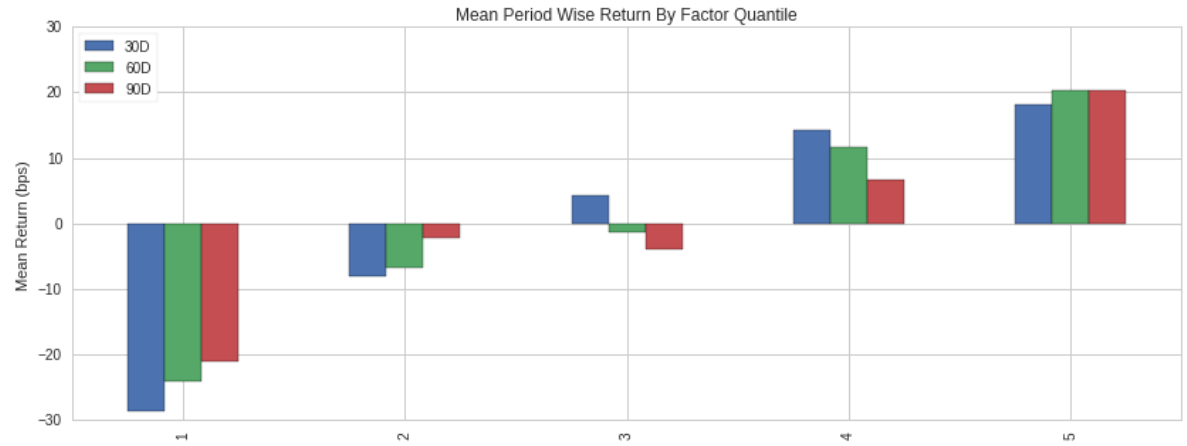
Quantiles Statistics

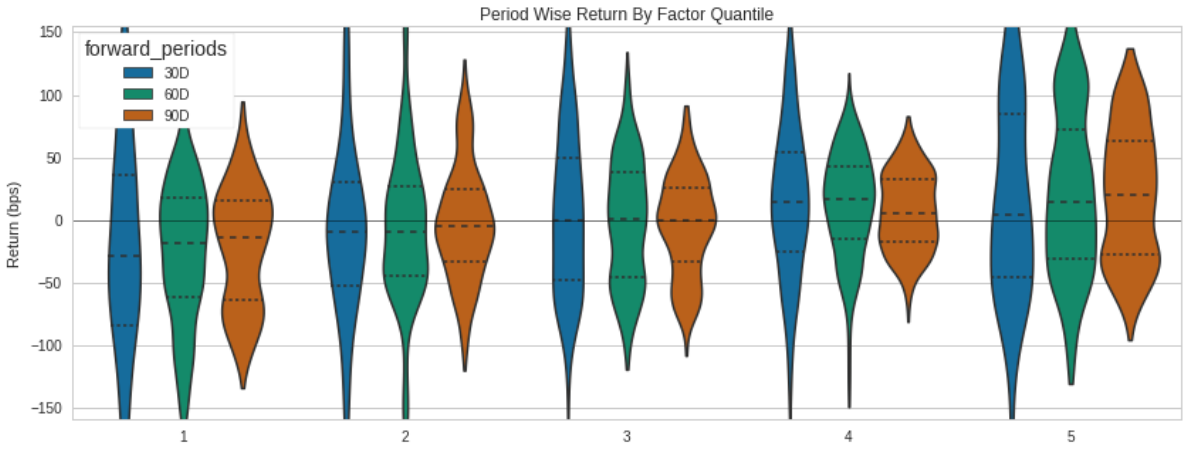
factor_quantile	min	max	mean	std	count	count %
1	0.058986	0.617053	0.479202	0.108507	213065	20.028313
2	0.584740	0.711563	0.653385	0.029964	212617	19.986201
3	0.687982	0.760984	0.728453	0.016805	212614	19.985919
4	0.748986	0.812557	0.778404	0.014801	212607	19.985261
5	0.797175	0.936800	0.839111	0.025475	212916	20.014307

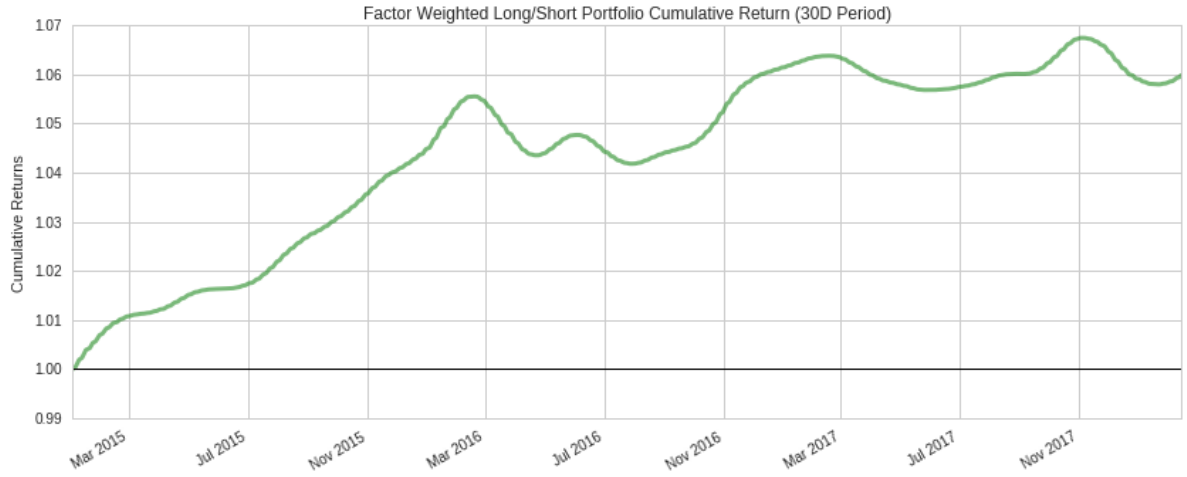
Returns Analysis

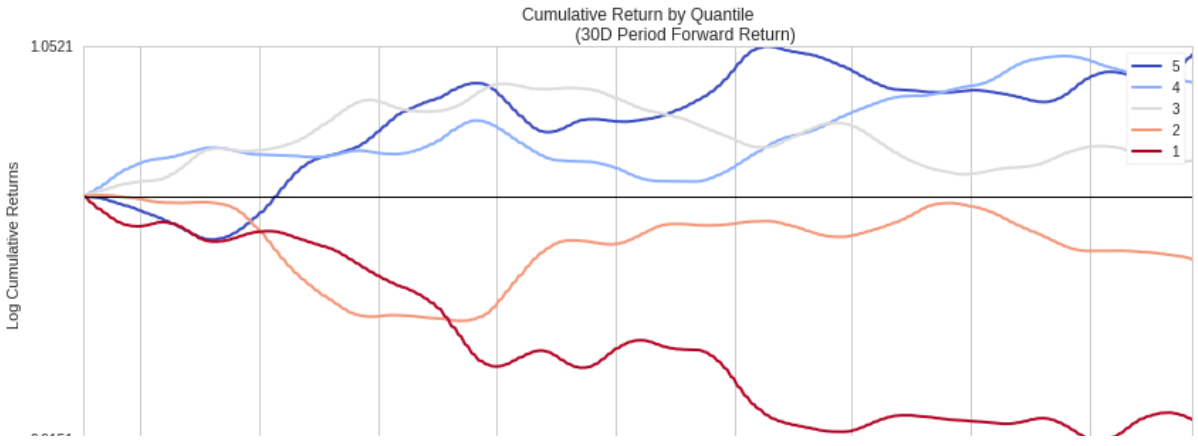
	30D	60D	90D
Ann. alpha	0.025	0.021	0.018
beta	-0.060	-0.063	-0.054
Mean Period Wise Return Top Quantile (bps)	18.231	20.380	20.392
Mean Period Wise Return Bottom Quantile (bps)	-28.702	-24.029	-21.089
Mean Period Wise Spread (bps)	46.933	44.362	41.428

<matplotlib.figure.Figure at 0x7f6ceaf180b8>

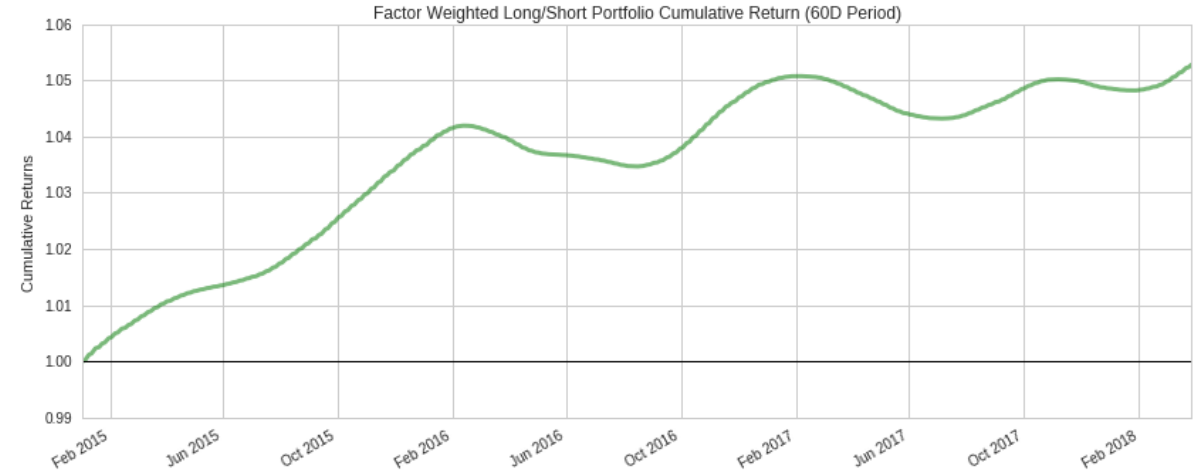

Mean Period Wise Return By Factor Quantile
30D
60D
90D
Mean Return (bps)
30
20
10
0
−10
−20
−30
1
2
3
4
5

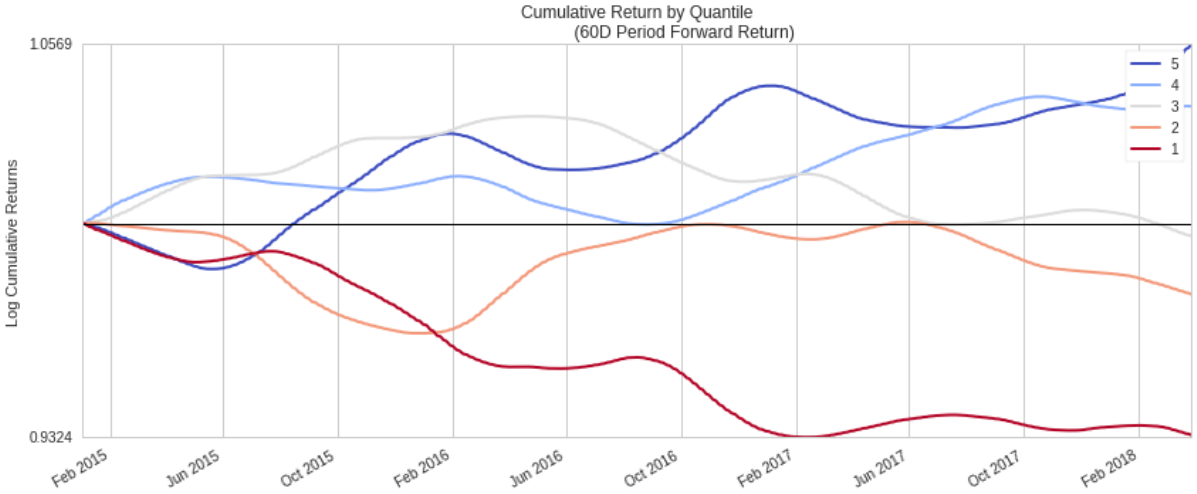

Period Wise Return By Factor Quantile
forward_periods
30D
60D
90D
Return (bps)
150
100
50
0
−50
−100
−150
1
2
3
4
5

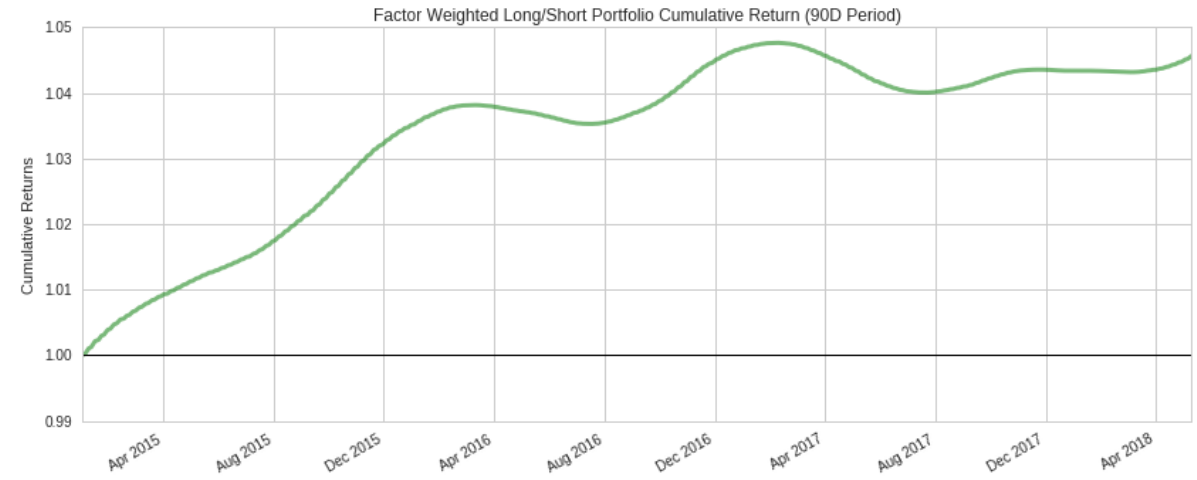

Factor Weighted Long/Short Portfolio Cumulative Return (30D Period)
Cumulative Returns
1.07
1.06
1.05
1.04
1.03
1.02
1.01
1.00
0.99
Mar 2015
Jul 2015
Nov 2015
Mar 2016
Jul 2016
Nov 2016
Mar 2017
Jul 2017
Nov 2017

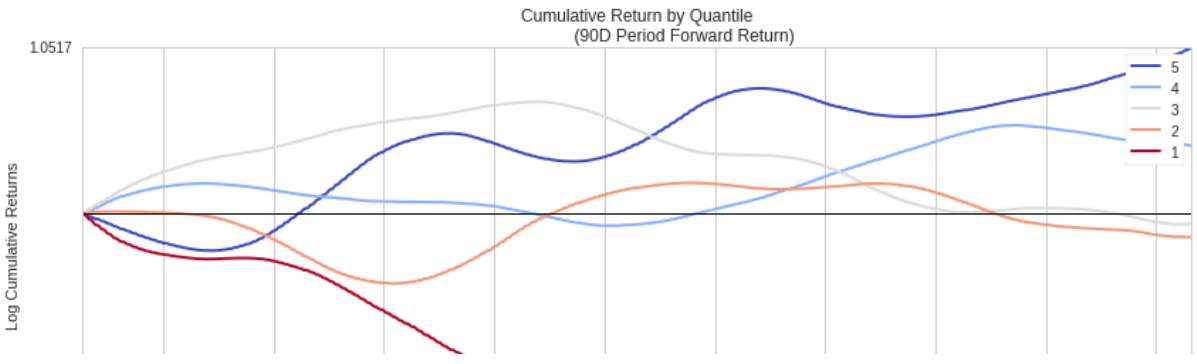

Cumulative Return by Quantile
(30D Period Forward Return)
Log Cumulative Returns
1.0521
5
4
3
2
1

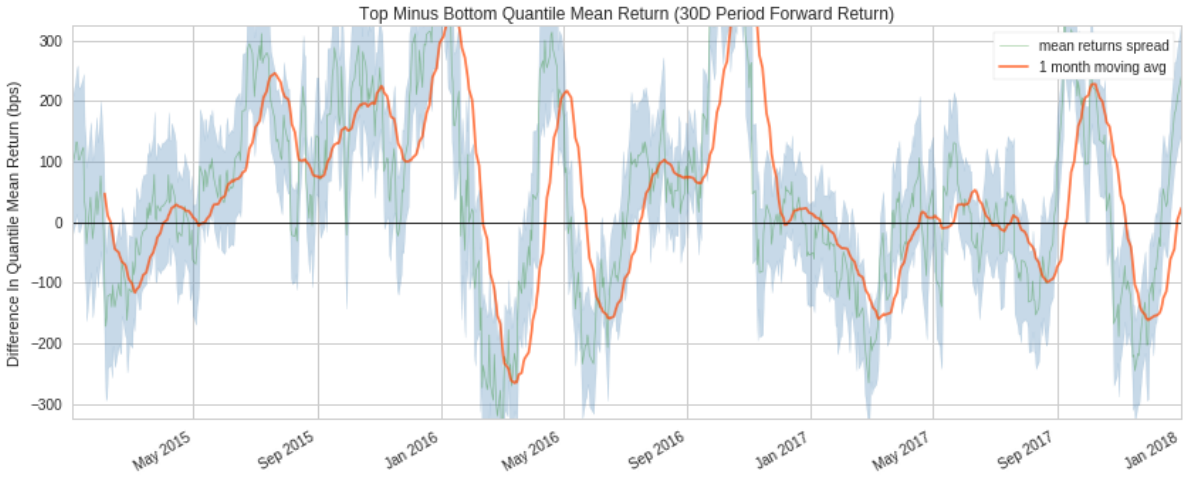

0.9151
Mar 2015
Jul 2015
Nov 2015
Mar 2016
Jul 2016
Nov 2016
Mar 2017
Jul 2017
Nov 2017

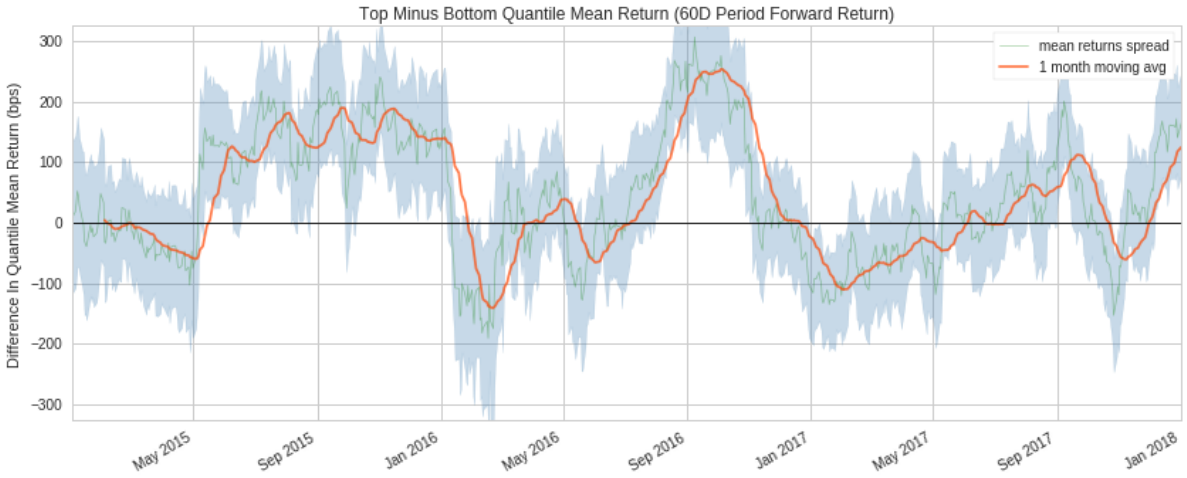

Factor Weighted Long/Short Portfolio Cumulative Return (60D Period)
Cumulative Returns
1.06
1.05
1.04
1.03
1.02
1.01
1.00
0.99
Feb 2015
Jun 2015
Oct 2015
Feb 2016
Jun 2016
Oct 2016
Feb 2017
Jun 2017
Oct 2017
Feb 2018

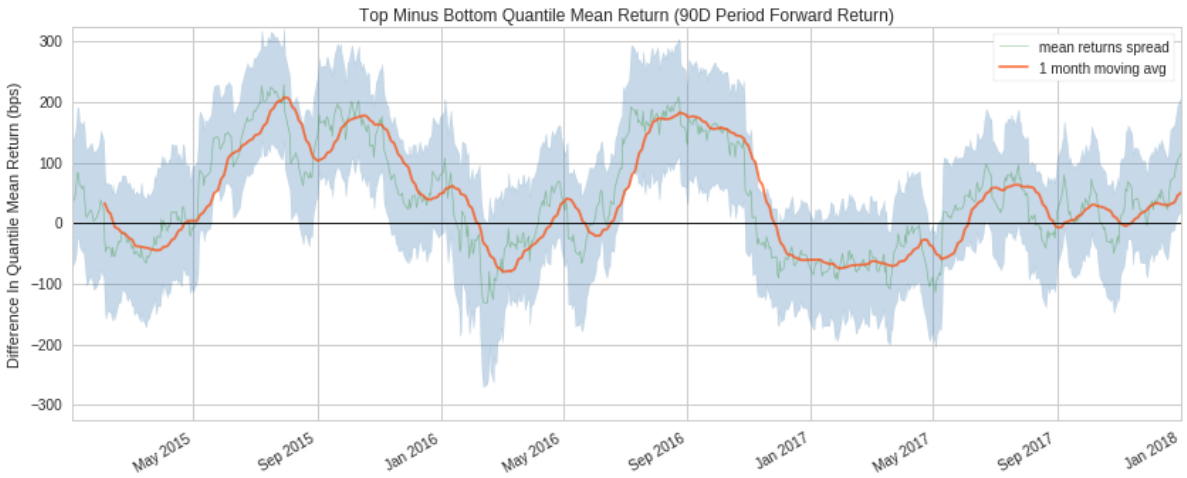

Cumulative Return by Quantile
(60D Period Forward Return)
Log Cumulative Returns
1.0569
0.9324
5
4
3
2
1
Feb 2015
Jun 2015
Oct 2015
Feb 2016
Jun 2016
Oct 2016
Feb 2017
Jun 2017
Oct 2017
Feb 2018


Factor Weighted Long/Short Portfolio Cumulative Return (90D Period)
Cumulative Returns
1.05
1.04
1.03
1.02
1.01
1.00
0.99
Apr 2015
Aug 2015
Dec 2015
Apr 2016
Aug 2016
Dec 2016
Apr 2017
Aug 2017
Dec 2017
Apr 2018


Cumulative Return by Quantile
(90D Period Forward Return)
Log Cumulative Returns
1.0517
5
4
3
2
1

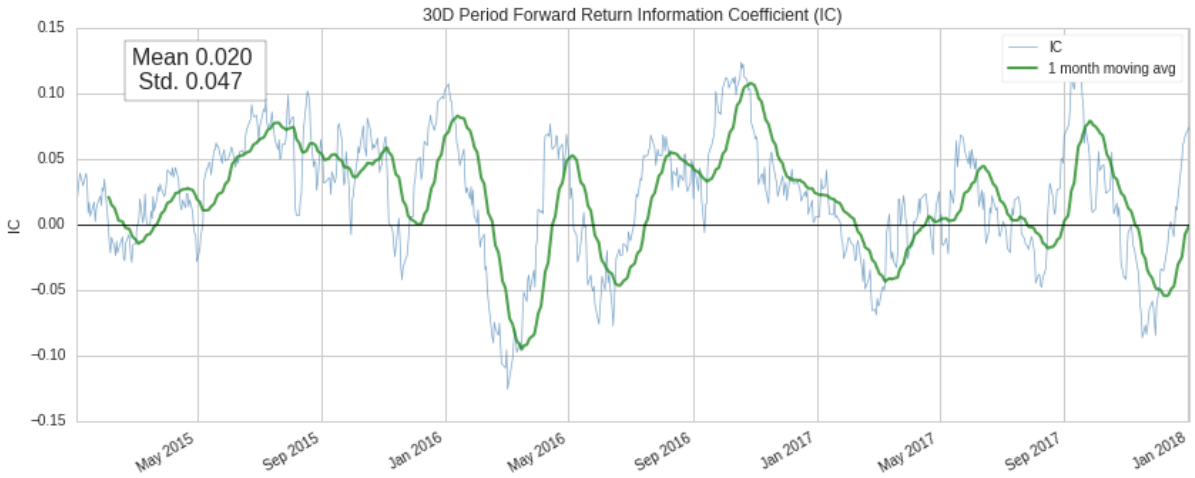


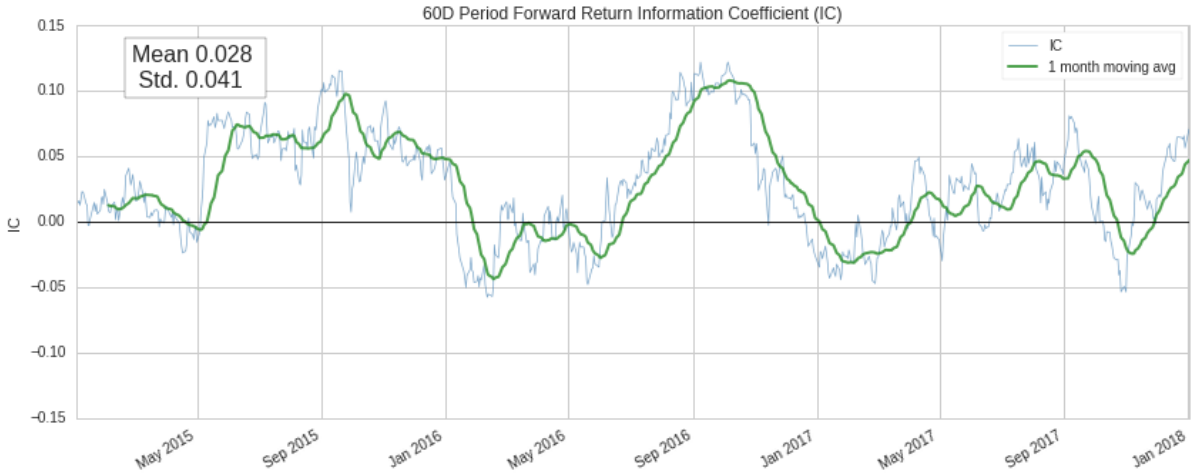


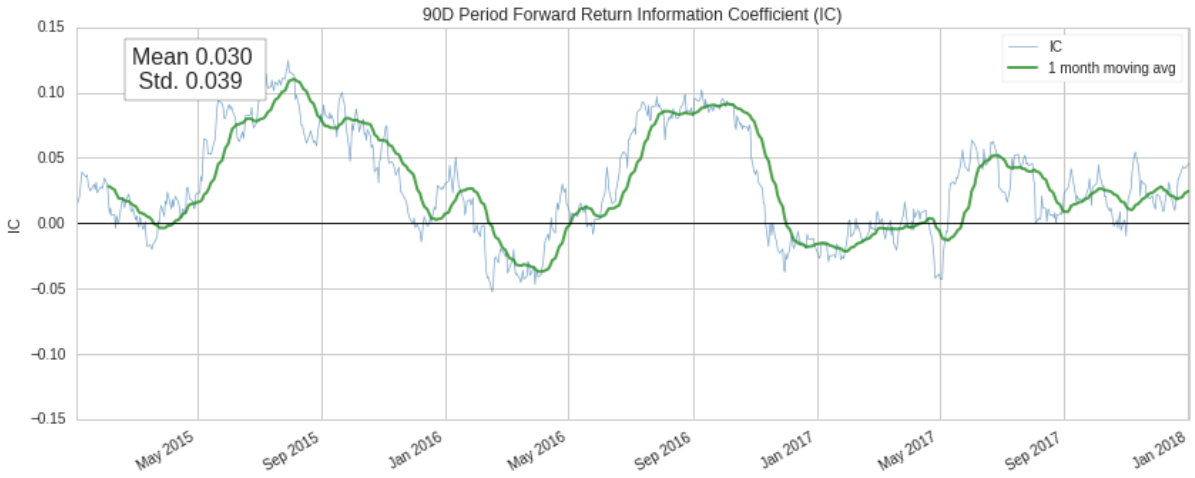


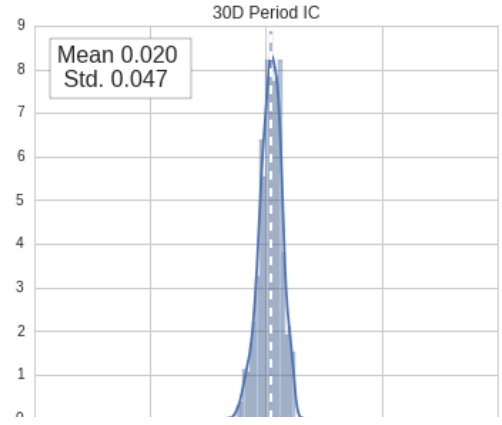
Information Analysis

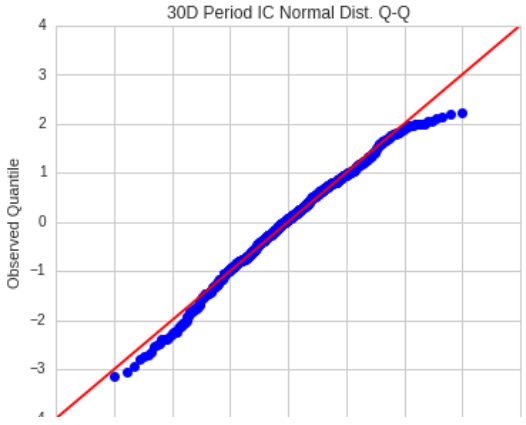
	30D	60D	90D
IC Mean	0.020	0.028	0.030
IC Std.	0.047	0.041	0.039
Risk-Adjusted IC	0.425	0.684	0.760
t-stat(IC)	11.698	18.816	20.887
p-value(IC)	0.000	0.000	0.000
IC Skew	-0.334	0.178	0.269
IC Kurtosis	-0.031	-0.681	-0.857


30D Period Forward Return Information Coefficient (IC)
Mean 0.020
Std. 0.047
IC
1 month moving avg
IC
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
May 2015
Sep 2015
Jan 2016
May 2016
Sep 2016
Jan 2017
May 2017
Sep 2017
Jan 2018


60D Period Forward Return Information Coefficient (IC)
Mean 0.028
Std. 0.041
IC
1 month moving avg
IC
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
May 2015
Sep 2015
Jan 2016
May 2016
Sep 2016
Jan 2017
May 2017
Sep 2017
Jan 2018

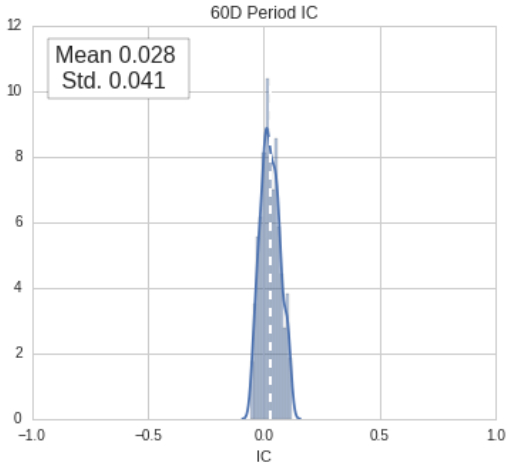

90D Period Forward Return Information Coefficient (IC)
Mean 0.030
Std. 0.039
IC
1 month moving avg
IC
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
May 2015
Sep 2015
Jan 2016
May 2016
Sep 2016
Jan 2017
May 2017
Sep 2017
Jan 2018

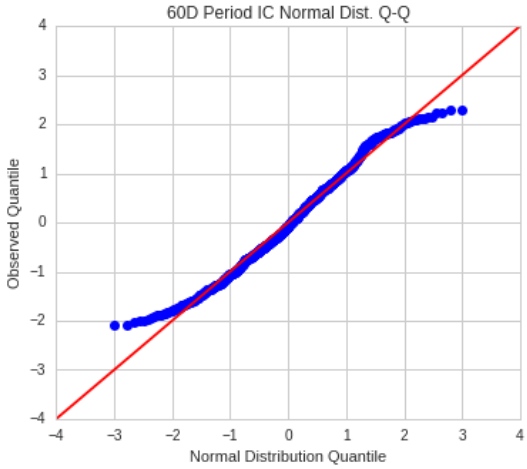

30D Period IC
Mean 0.020
Std. 0.047
9
8
7
6
5
4
3
2
1

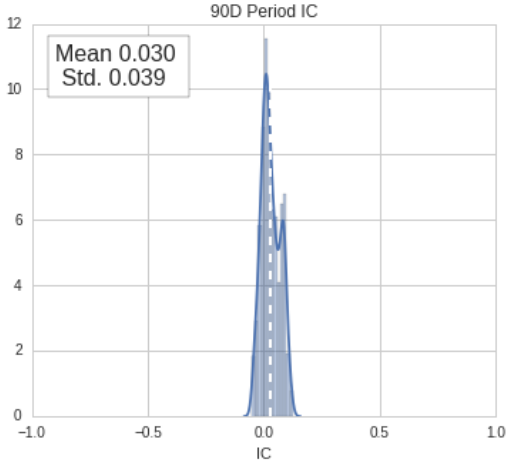

30D Period IC Normal Dist. Q-Q
Observed Quantile
4
3
2
1
0
−1
−2
−3

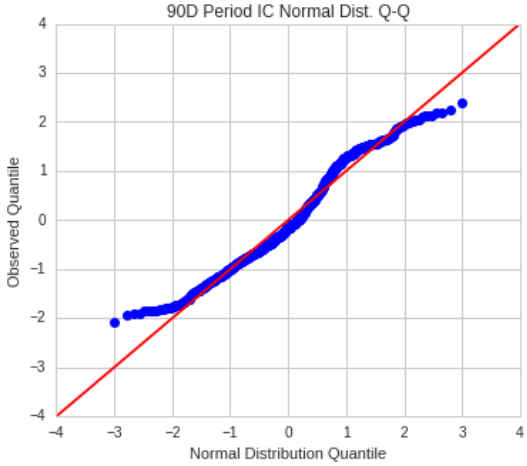

−1.0
−0.5
0.0
0.5
1.0
IC

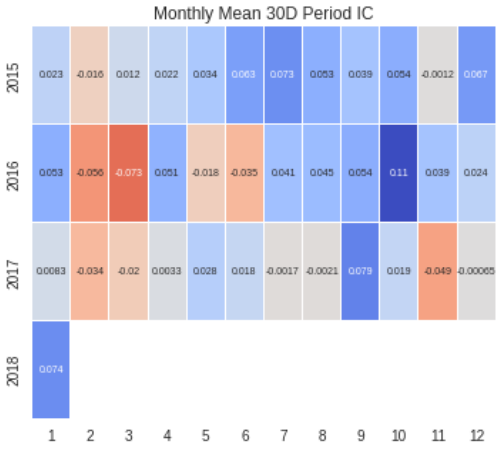

−4
−3
−2
−1
0
1
2
3
4
Normal Distribution Quantile

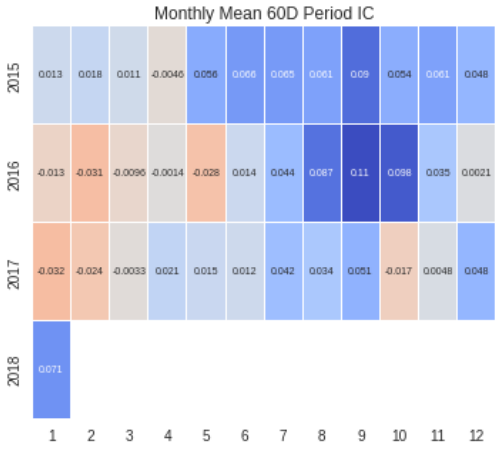

60D Period IC
Mean 0.028
Std. 0.041
IC

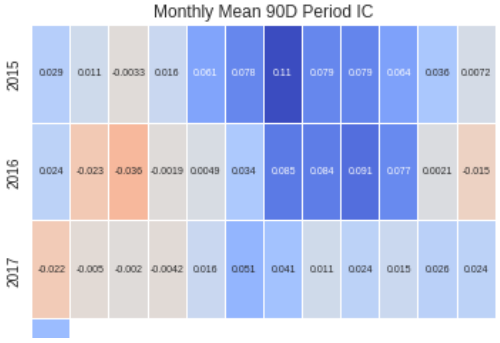

60D Period IC Normal Dist. Q-Q
Observed Quantile
Normal Distribution Quantile


90D Period IC
Mean 0.030
Std. 0.039
IC


90D Period IC Normal Dist. Q-Q
Observed Quantile
Normal Distribution Quantile


Monthly Mean 30D Period IC
2015
2016
2017
2018
1
2
3
4
5
6
7
8
9
10
11
12

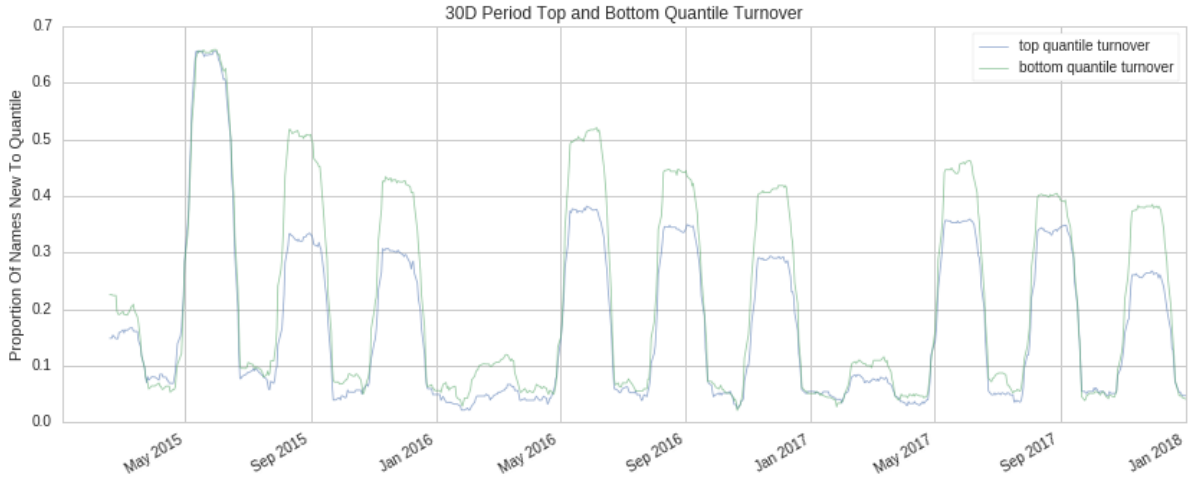

Monthly Mean 60D Period IC
2015
2016
2017
2018
1
2
3
4
5
6
7
8
9
10
11
12

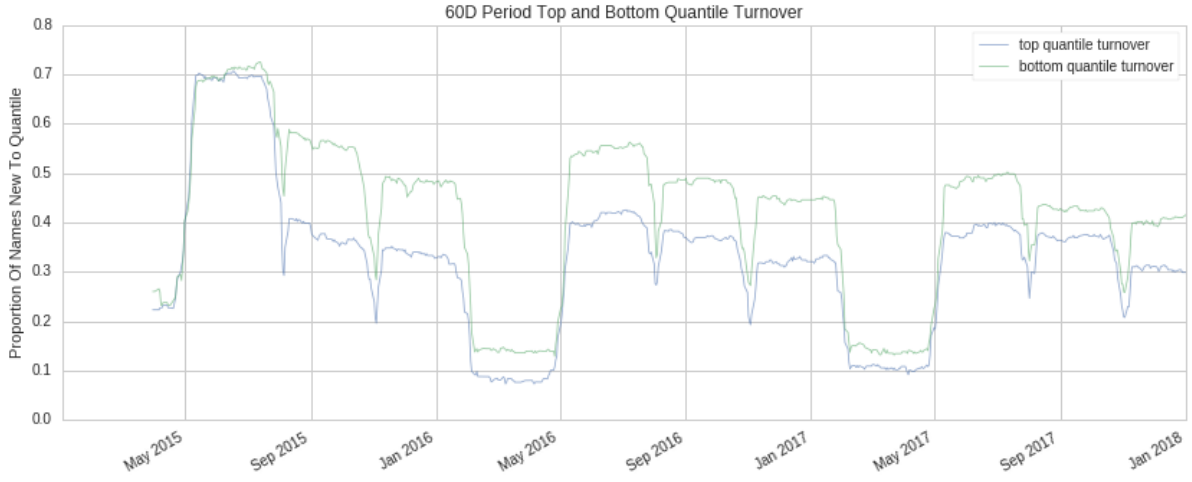

Monthly Mean 90D Period IC
2015
2016
2017

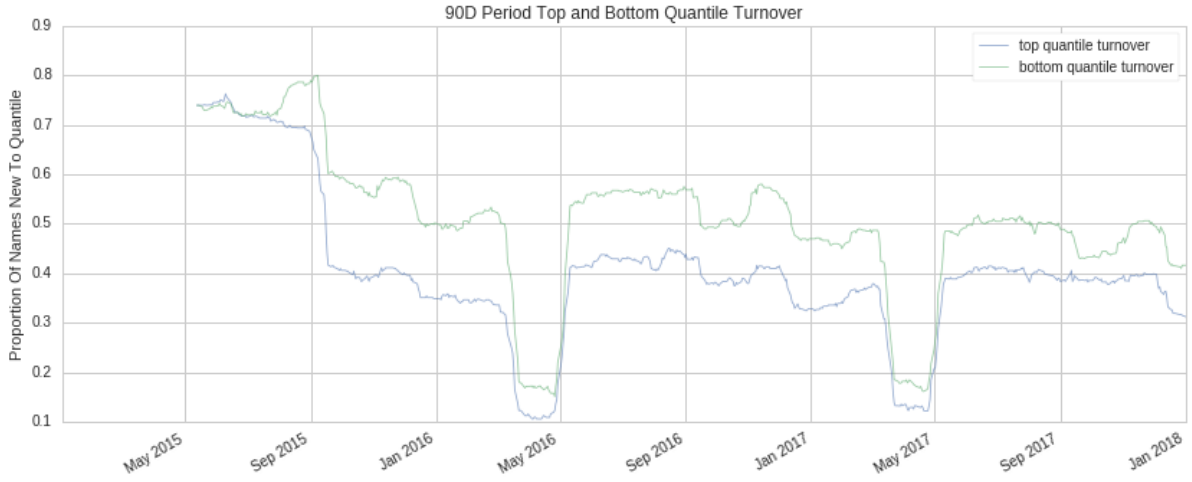
Turnover Analysis

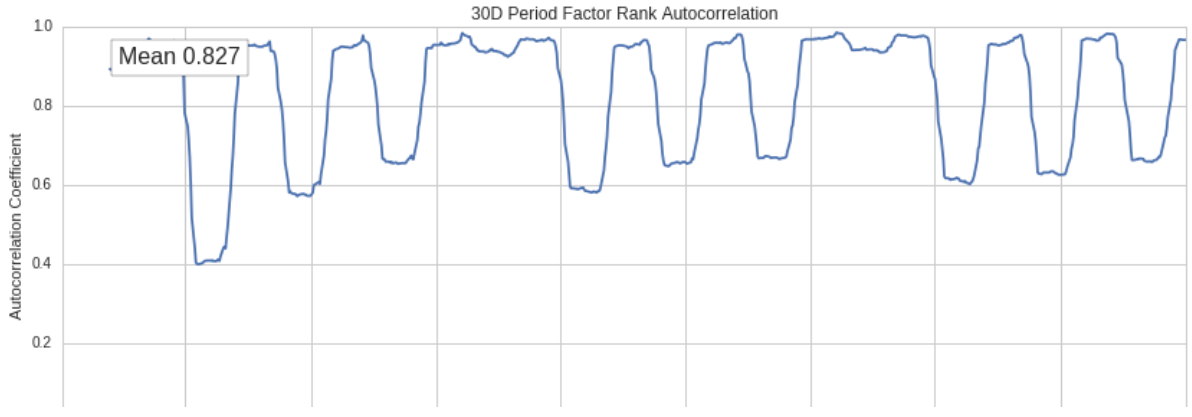
	30D	60D	90D
Quantile 1 Mean Turnover	0.220	0.418	0.509
Quantile 2 Mean Turnover	0.257	0.479	0.569
Quantile 3 Mean Turnover	0.247	0.463	0.558
Quantile 4 Mean Turnover	0.239	0.449	0.541
Quantile 5 Mean Turnover	0.173	0.329	0.402

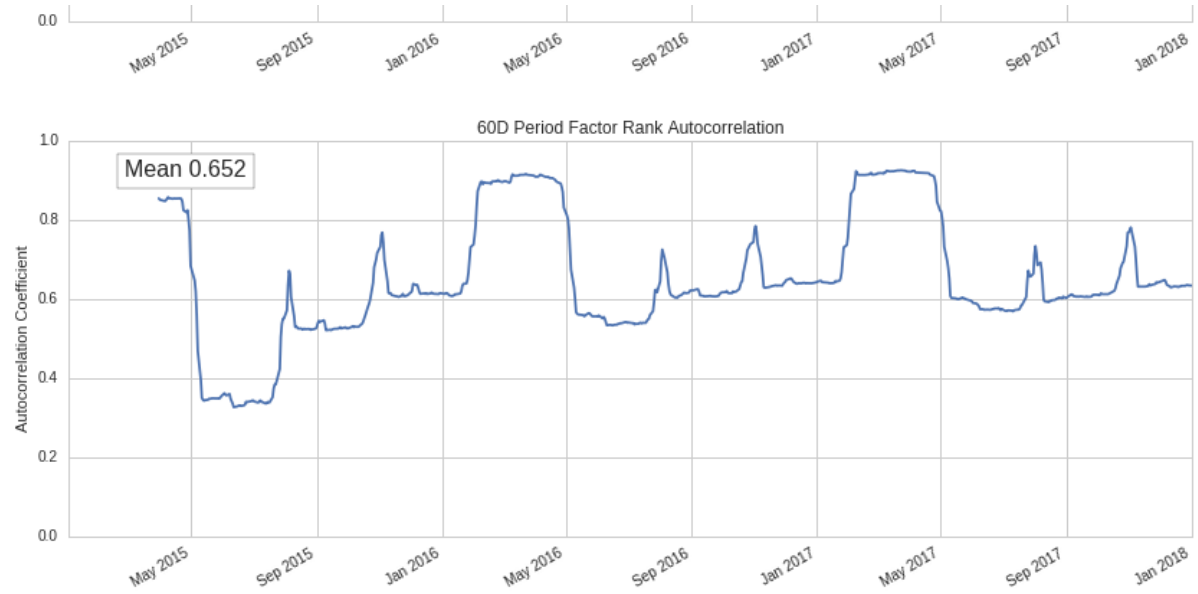
	30D	60D	90D
Mean Factor Rank Autocorrelation	0.827	0.652	0.571

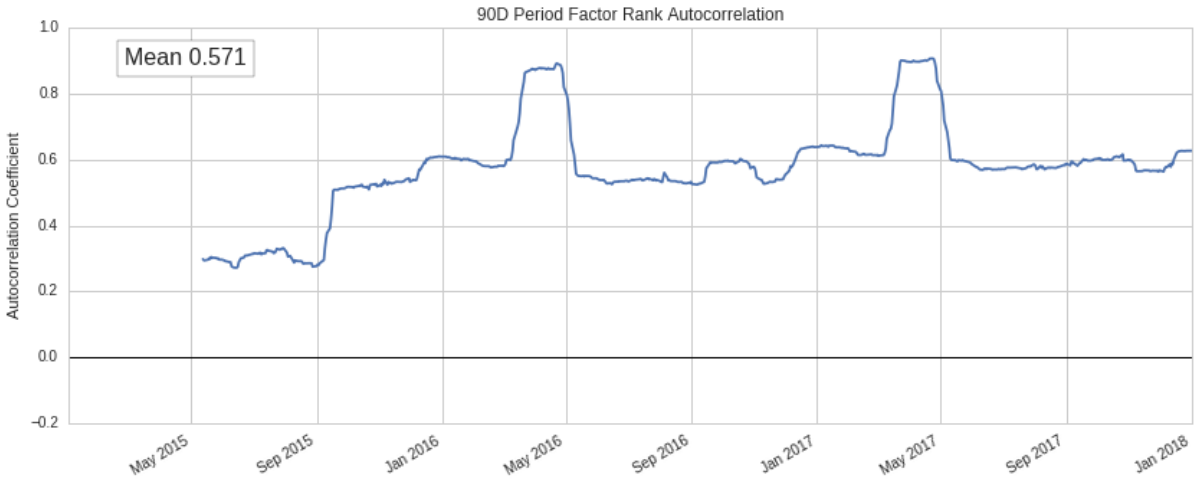

30D Period Top and Bottom Quantile Turnover
top quantile turnover
bottom quantile turnover
Proportion Of Names New To Quantile
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
May 2015
Sep 2015
Jan 2016
May 2016
Sep 2016
Jan 2017
May 2017
Sep 2017
Jan 2018


60D Period Top and Bottom Quantile Turnover
top quantile turnover
bottom quantile turnover
Proportion Of Names New To Quantile
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
May 2015
Sep 2015
Jan 2016
May 2016
Sep 2016
Jan 2017
May 2017
Sep 2017
Jan 2018


90D Period Top and Bottom Quantile Turnover
top quantile turnover
bottom quantile turnover
Proportion Of Names New To Quantile
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
May 2015
Sep 2015
Jan 2016
May 2016
Sep 2016
Jan 2017
May 2017
Sep 2017
Jan 2018


30D Period Factor Rank Autocorrelation
Mean 0.827
Autocorrelation Coefficient
0.2
0.4
0.6
0.8
1.0





3.2 Cosine Factor

In [9]:
```
cosine_factor = data[['cosine_score']]
```

In [10]:
```
factor_data_c3 = alphalens.utils.get_clean_factor_and_forward_returns(
    cosine_factor,
    prices=prices,
    quantiles=5,
    periods =(30, 60, 90),
)
```

```
Dropped 0.1% entries from factor data: 0.1% in forward returns computat
ion and 0.0% in binning phase (set max_loss=0 to see potentially suppre
ssed Exceptions).
max_loss is 35.0%, not exceeded: OK!
```

In [11]:
```
alphalens.tears.create_full_tear_sheet(factor_data_c3, by_group=False);
```

Quantiles Statistics

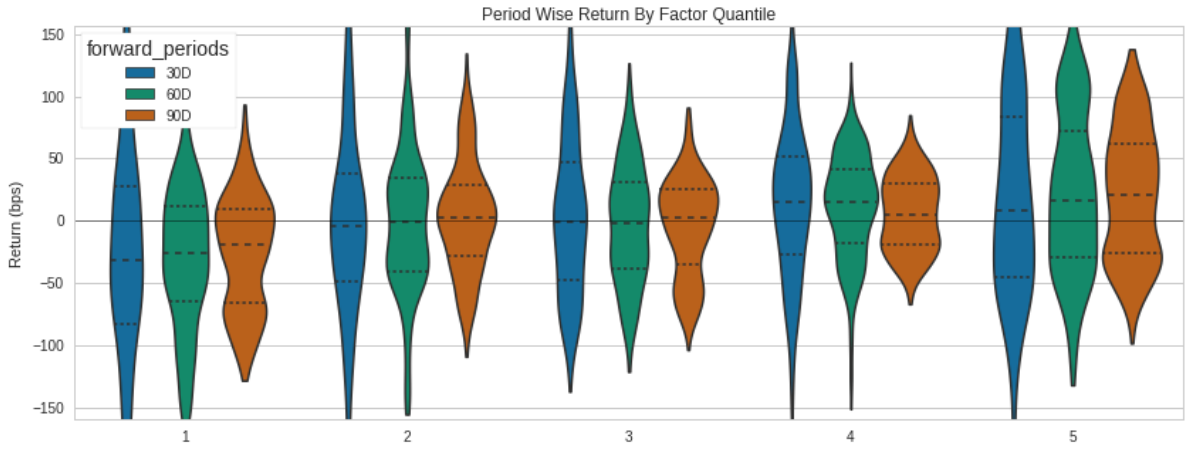
factor_quantile	min	max	mean	std	count	count %
1	0.236408	0.767642	0.663874	0.092597	213065	20.028313
2	0.744706	0.831777	0.793650	0.020593	212617	19.986201
3	0.817089	0.865000	0.843825	0.010916	212606	19.985167
4	0.857090	0.896678	0.875717	0.009240	212615	19.986013
5	0.887518	0.967373	0.912464	0.014885	212916	20.014307

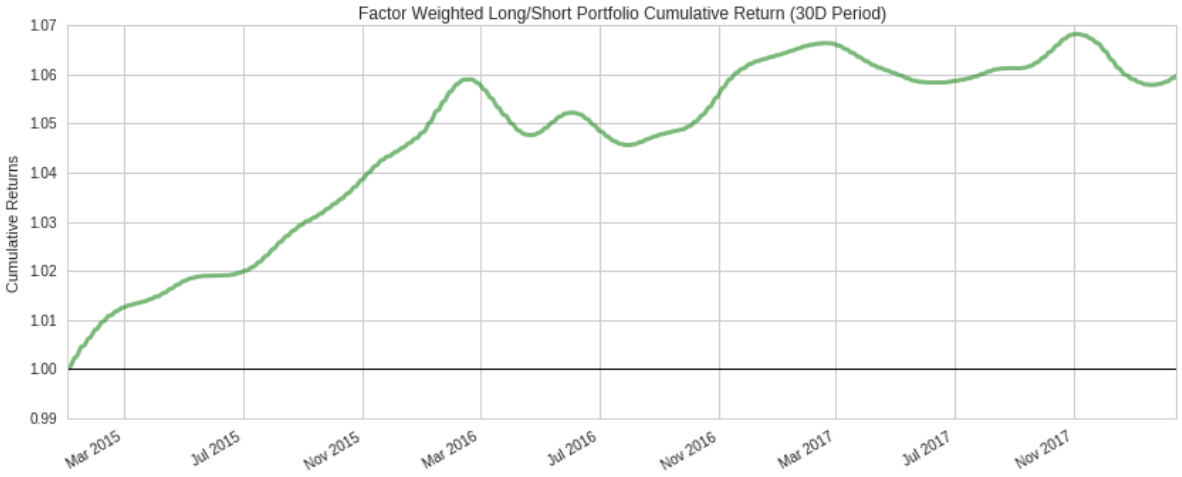
Returns Analysis

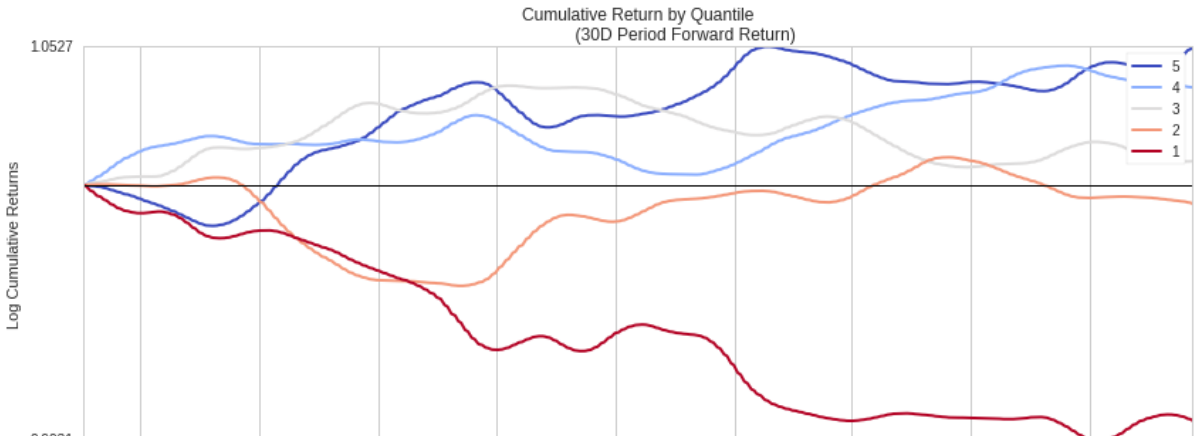
	30D	60D	90D
Ann. alpha	0.024	0.021	0.018
beta	-0.061	-0.065	-0.057
Mean Period Wise Return Top Quantile (bps)	19.503	21.346	20.886
Mean Period Wise Return Bottom Quantile (bps)	-33.839	-29.379	-25.643
Mean Period Wise Spread (bps)	53.342	50.683	46.468

<matplotlib.figure.Figure at 0x7f78e295d0b8>

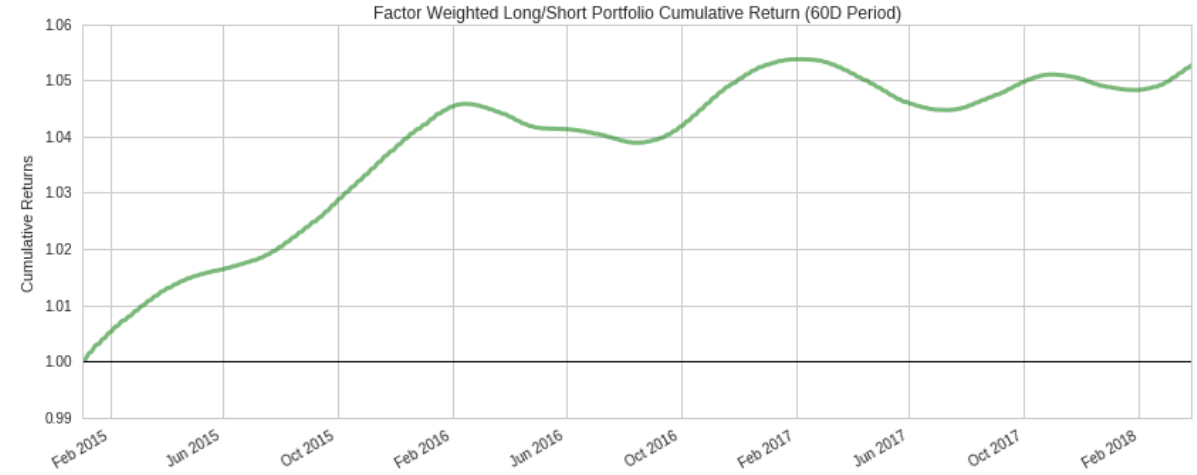

Mean Period Wise Return By Factor Quantile
30D
60D
90D
Mean Return (bps)
30
20
10
0
−10
−20
−30
−40
1
2
3
4
5

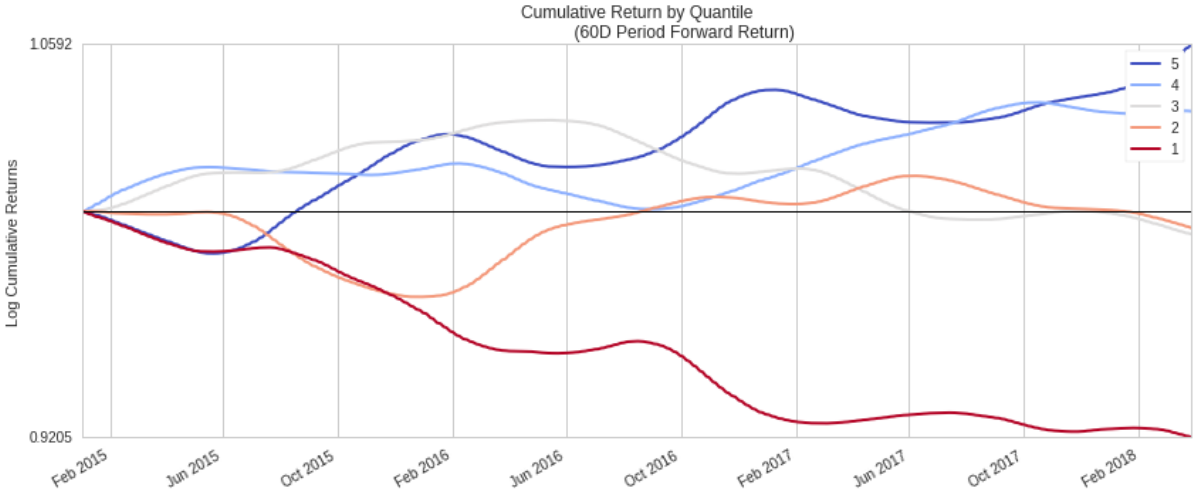

Period Wise Return By Factor Quantile
forward_periods
30D
60D
90D
Return (bps)
150
100
50
0
−50
−100
−150
1
2
3
4
5

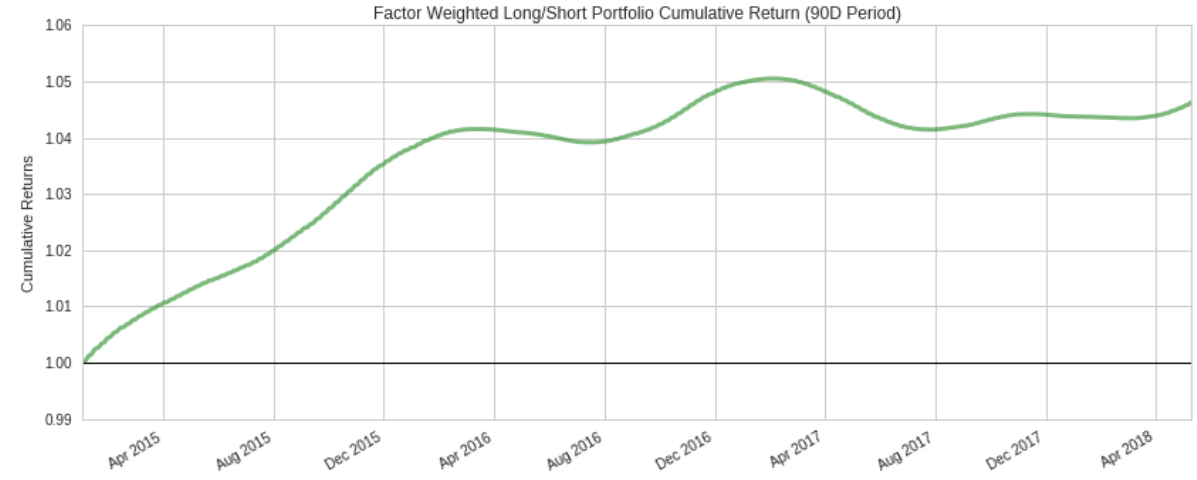

Factor Weighted Long/Short Portfolio Cumulative Return (30D Period)
Cumulative Returns
1.07
1.06
1.05
1.04
1.03
1.02
1.01
1.00
0.99
Mar 2015
Jul 2015
Nov 2015
Mar 2016
Jul 2016
Nov 2016
Mar 2017
Jul 2017
Nov 2017

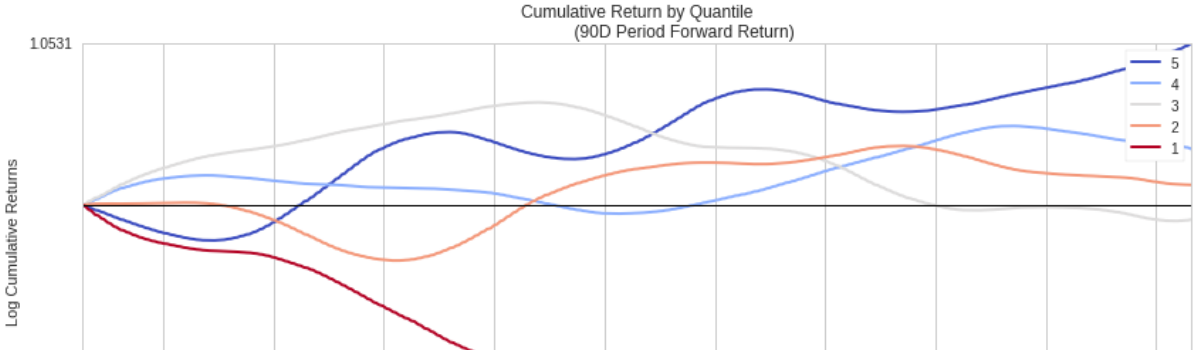

Cumulative Return by Quantile
(30D Period Forward Return)
Log Cumulative Returns
1.0527
5
4
3
2
1

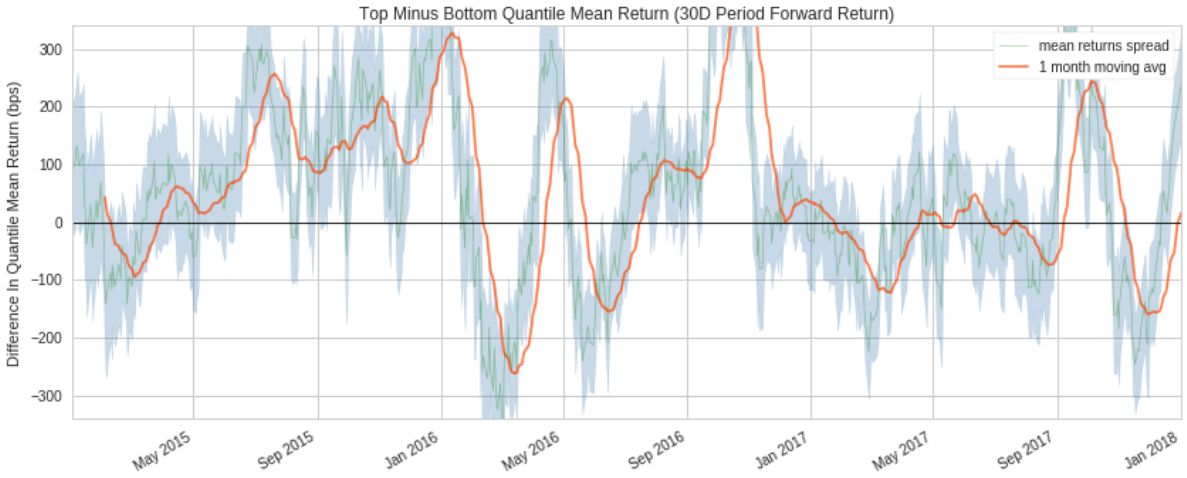

0.9031
Mar 2015
Jul 2015
Nov 2015
Mar 2016
Jul 2016
Nov 2016
Mar 2017
Jul 2017
Nov 2017

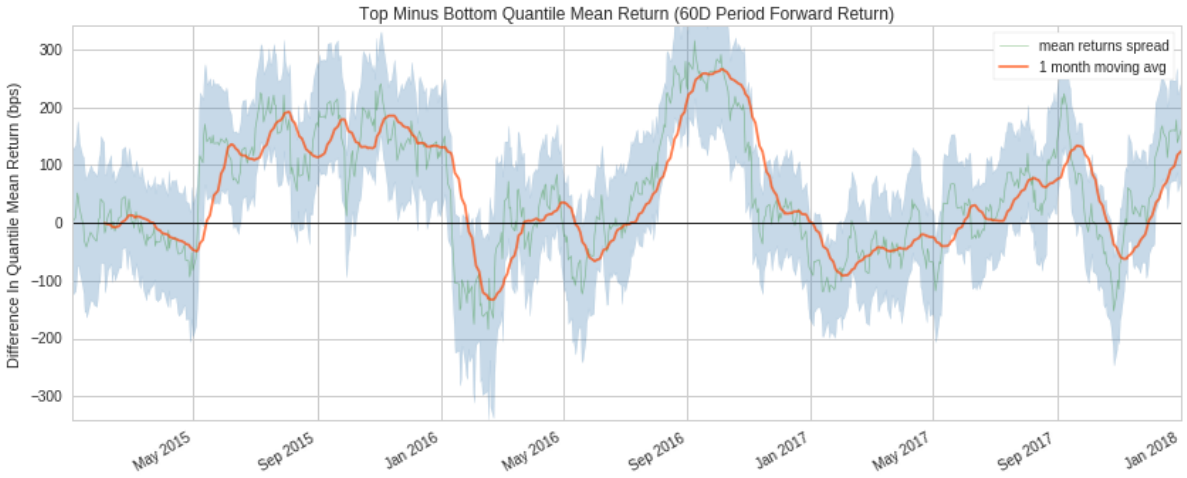

Factor Weighted Long/Short Portfolio Cumulative Return (60D Period)
Cumulative Returns
1.06
1.05
1.04
1.03
1.02
1.01
1.00
0.99
Feb 2015
Jun 2015
Oct 2015
Feb 2016
Jun 2016
Oct 2016
Feb 2017
Jun 2017
Oct 2017
Feb 2018

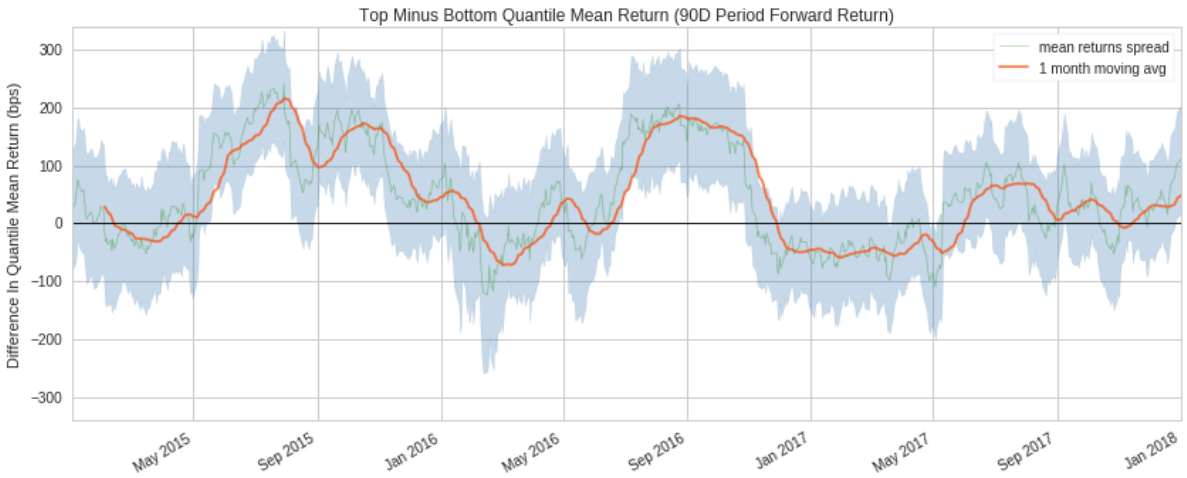

Cumulative Return by Quantile
(60D Period Forward Return)
Log Cumulative Returns
1.0592
0.9205
5
4
3
2
1
Feb 2015
Jun 2015
Oct 2015
Feb 2016
Jun 2016
Oct 2016
Feb 2017
Jun 2017
Oct 2017
Feb 2018


Factor Weighted Long/Short Portfolio Cumulative Return (90D Period)
Cumulative Returns
1.06
1.05
1.04
1.03
1.02
1.01
1.00
0.99
Apr 2015
Aug 2015
Dec 2015
Apr 2016
Aug 2016
Dec 2016
Apr 2017
Aug 2017
Dec 2017
Apr 2018


Cumulative Return by Quantile
(90D Period Forward Return)
Log Cumulative Returns
1.0531
5
4
3
2
1



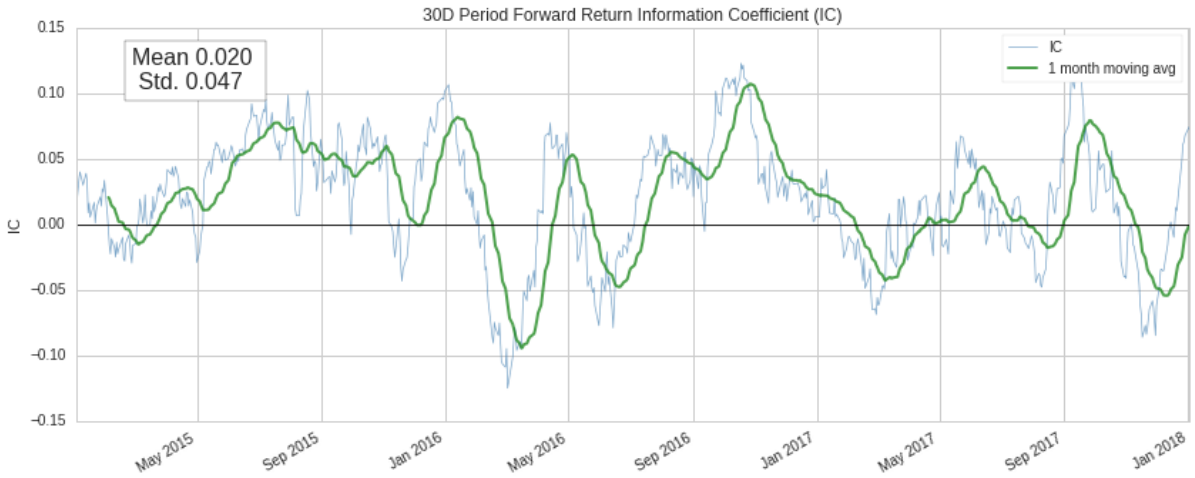


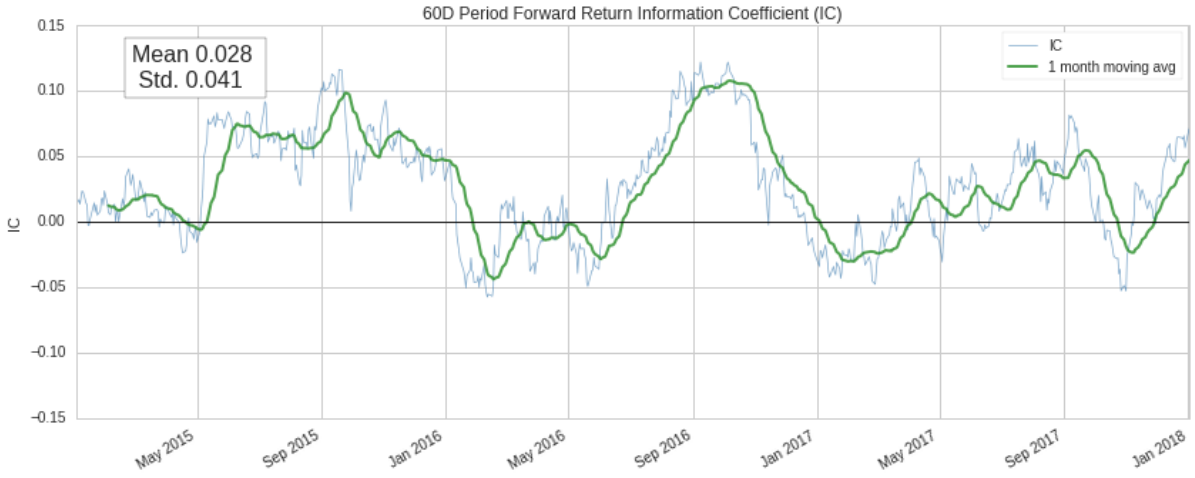


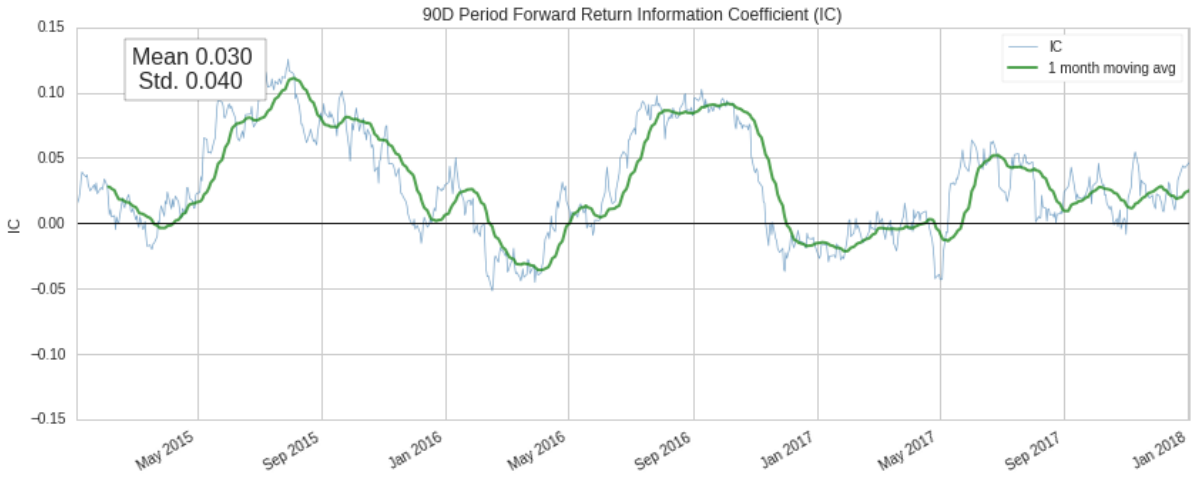
Information Analysis

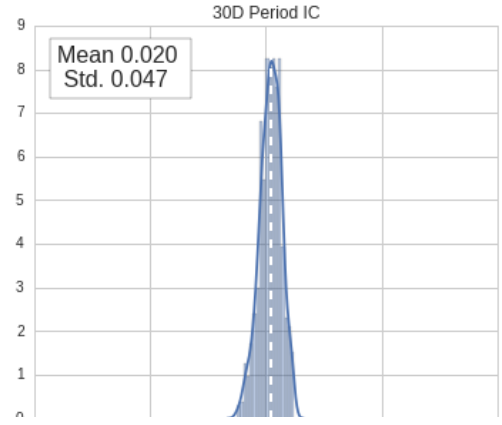
	30D	60D	90D
IC Mean	0.020	0.028	0.030
IC Std.	0.047	0.041	0.040
Risk-Adjusted IC	0.427	0.685	0.765
t-stat(IC)	11.730	18.842	21.032
p-value(IC)	0.000	0.000	0.000
IC Skew	-0.331	0.184	0.281
IC Kurtosis	-0.069	-0.682	-0.864

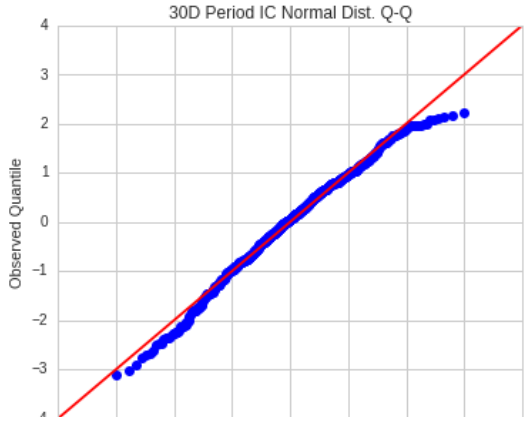
```
/venvs/py35/lib/python3.5/site-packages/statsmodels/nonparametric/kdeto
ols.py:20: VisibleDeprecationWarning: using a non-integer number instea
d of an integer will result in an error in the future
  y = X[:m/2+1] + np.r_[0,X[m/2+1:],0]*1j
```


30D Period Forward Return Information Coefficient (IC)
Mean 0.020
Std. 0.047
IC
1 month moving avg
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
IC
May 2015
Sep 2015
Jan 2016
May 2016
Sep 2016
Jan 2017
May 2017
Sep 2017
Jan 2018


60D Period Forward Return Information Coefficient (IC)
Mean 0.028
Std. 0.041
IC
1 month moving avg
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
IC
May 2015
Sep 2015
Jan 2016
May 2016
Sep 2016
Jan 2017
May 2017
Sep 2017
Jan 2018

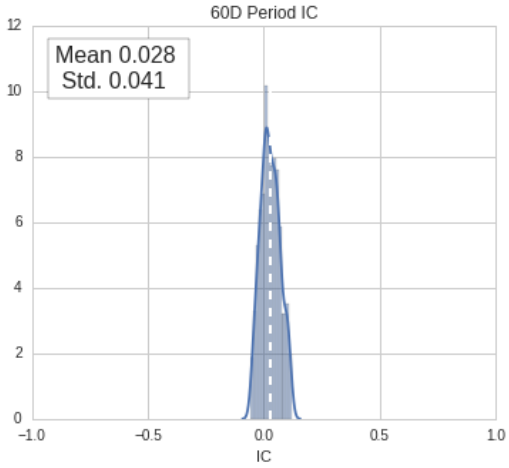

90D Period Forward Return Information Coefficient (IC)
Mean 0.030
Std. 0.040
IC
1 month moving avg
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
IC
May 2015
Sep 2015
Jan 2016
May 2016
Sep 2016
Jan 2017
May 2017
Sep 2017
Jan 2018

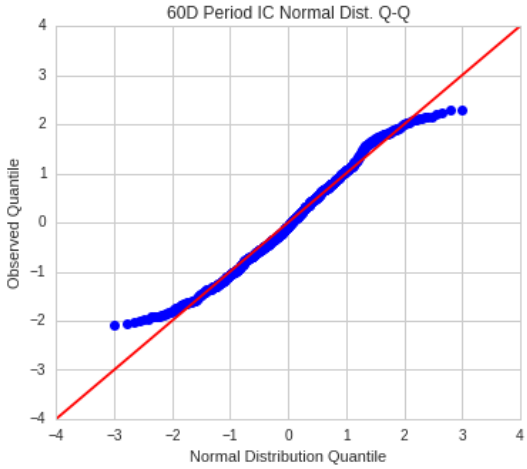

30D Period IC
Mean 0.020
Std. 0.047
9
8
7
6
5
4
3
2
1

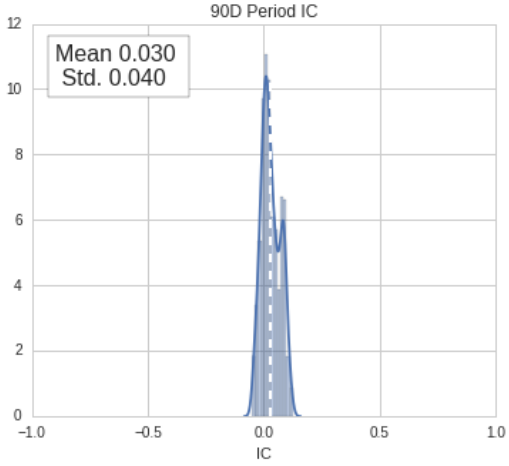

30D Period IC Normal Dist. Q-Q
Observed Quantile
4
3
2
1
0
−1
−2
−3

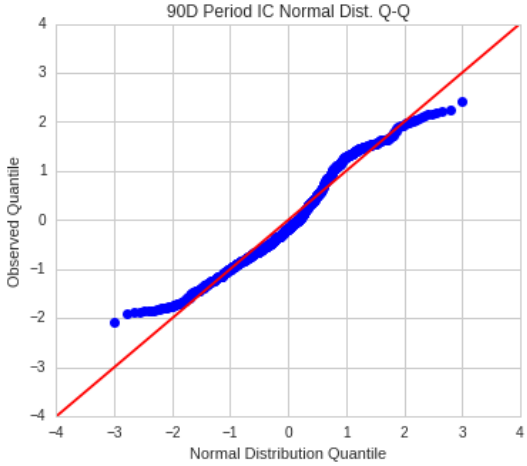

IC

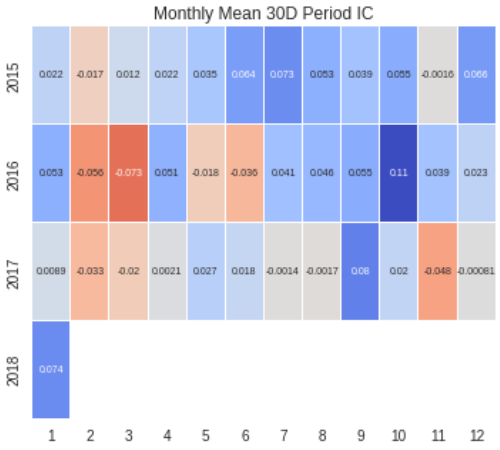

Normal Distribution Quantile

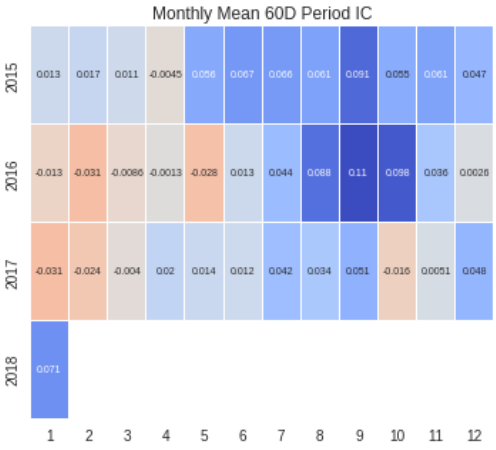

60D Period IC
Mean 0.028
Std. 0.041
IC

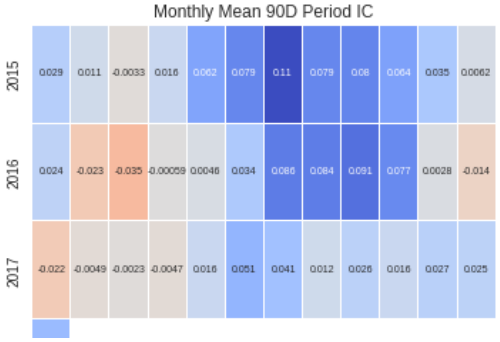

60D Period IC Normal Dist. Q-Q
Observed Quantile
Normal Distribution Quantile


90D Period IC
Mean 0.030
Std. 0.040
IC


90D Period IC Normal Dist. Q-Q
Observed Quantile
Normal Distribution Quantile


Monthly Mean 30D Period IC
2015
2016
2017
2018
1 2 3 4 5 6 7 8 9 10 11 12

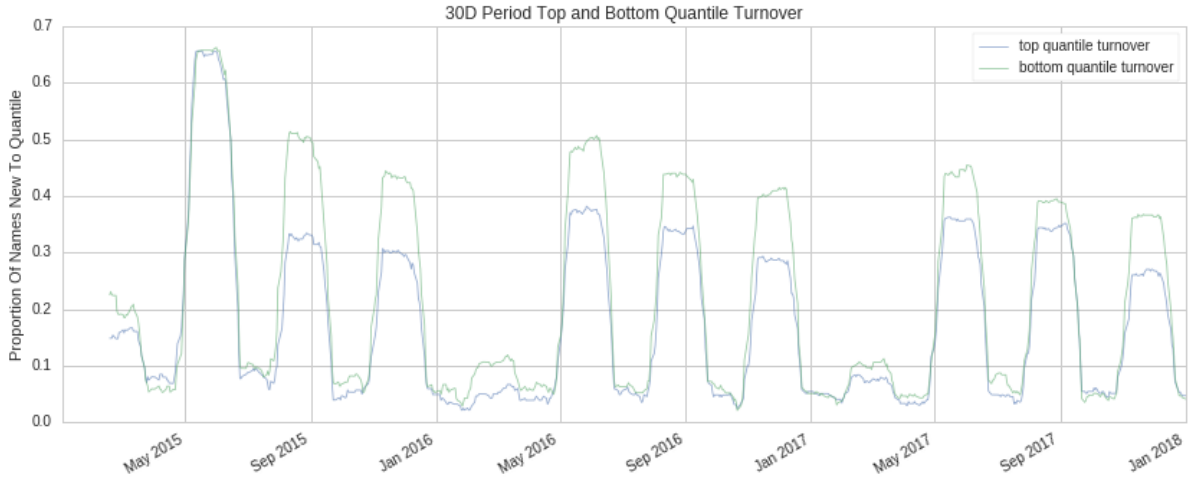

Monthly Mean 60D Period IC
2015
2016
2017
2018
1 2 3 4 5 6 7 8 9 10 11 12

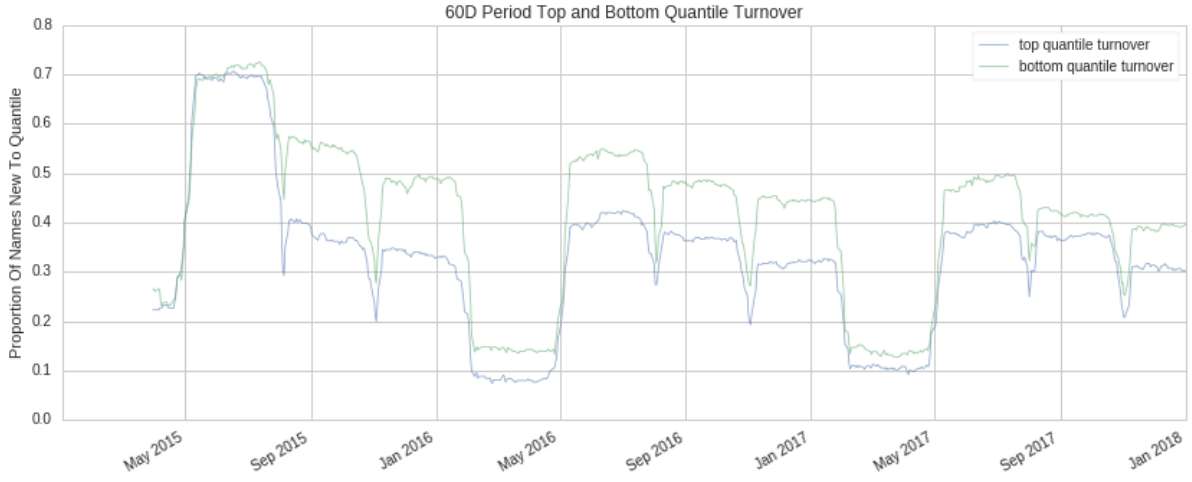

Monthly Mean 90D Period IC
2015
2016
2017

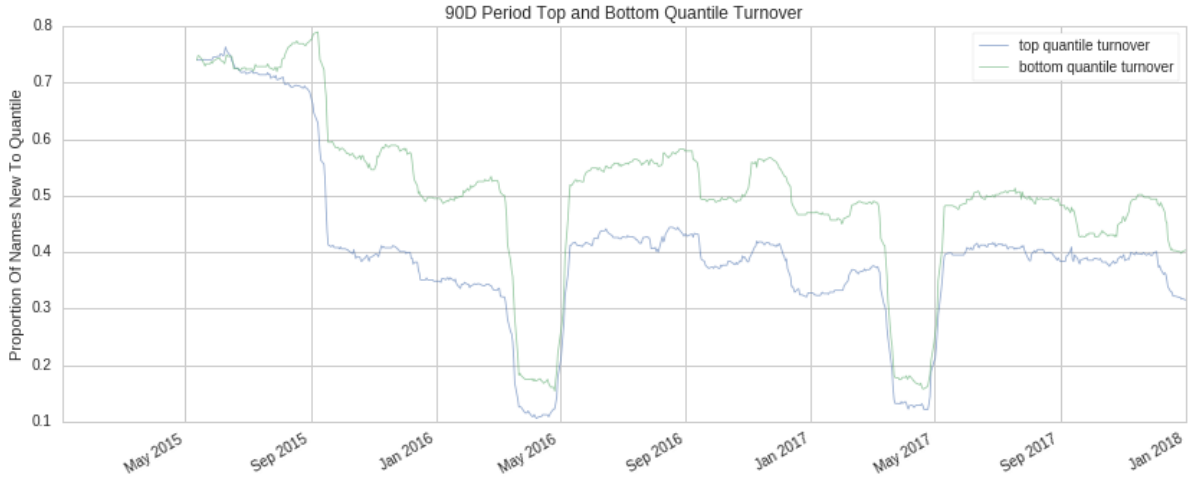
Turnover Analysis

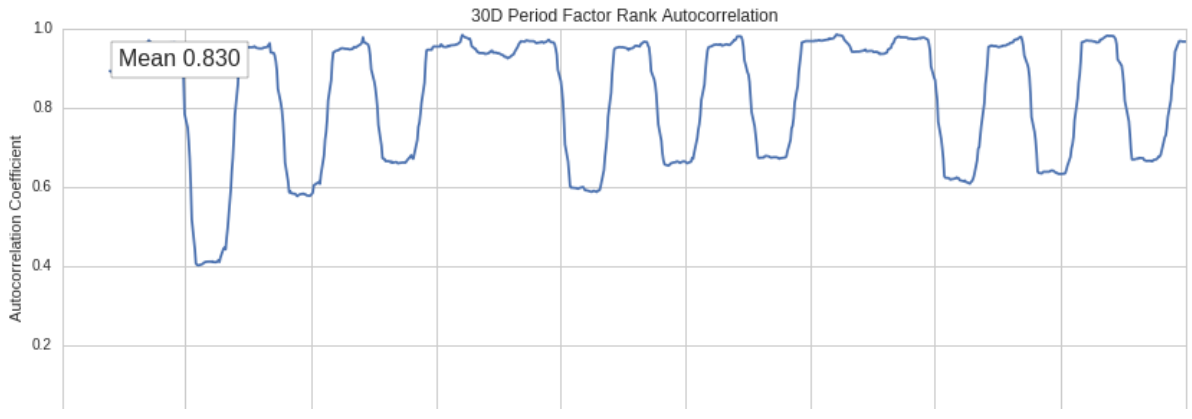
	30D	60D	90D
Quantile 1 Mean Turnover	0.217	0.414	0.506
Quantile 2 Mean Turnover	0.257	0.480	0.571
Quantile 3 Mean Turnover	0.248	0.465	0.561
Quantile 4 Mean Turnover	0.240	0.449	0.540
Quantile 5 Mean Turnover	0.173	0.329	0.402

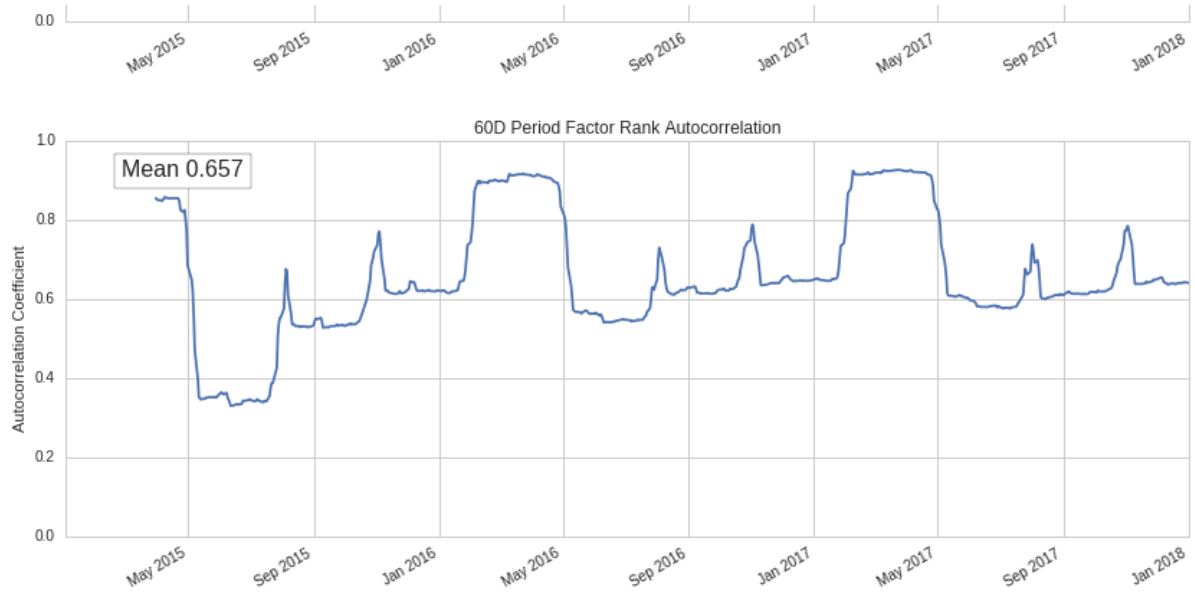
	30D	60D	90D
Mean Factor Rank Autocorrelation	0.83	0.657	0.577

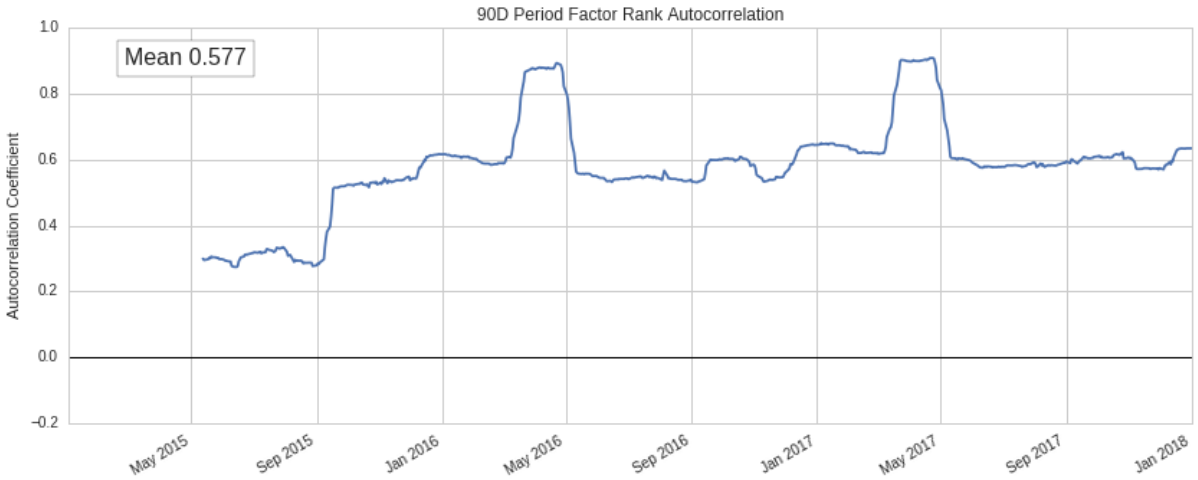

30D Period Top and Bottom Quantile Turnover
top quantile turnover
bottom quantile turnover
Proportion Of Names New To Quantile
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
May 2015
Sep 2015
Jan 2016
May 2016
Sep 2016
Jan 2017
May 2017
Sep 2017
Jan 2018


60D Period Top and Bottom Quantile Turnover
top quantile turnover
bottom quantile turnover
Proportion Of Names New To Quantile
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
May 2015
Sep 2015
Jan 2016
May 2016
Sep 2016
Jan 2017
May 2017
Sep 2017
Jan 2018


90D Period Top and Bottom Quantile Turnover
top quantile turnover
bottom quantile turnover
Proportion Of Names New To Quantile
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
May 2015
Sep 2015
Jan 2016
May 2016
Sep 2016
Jan 2017
May 2017
Sep 2017
Jan 2018


30D Period Factor Rank Autocorrelation
Mean 0.830
Autocorrelation Coefficient
0.2
0.4
0.6
0.8
1.0





Resources

This notebook follows the code from: https://www.quantopian.com/posts/analyzing-alpha-in-10-ks-and-10-qs

In [ ]: