1. **Modify EDGAR_Pac.py & EDGAR_DownloadForms.ipynb**

- *EDGAR_Pac.py & EDGAR_DownloadForms.ipynb*
  We modified the EDGAR_DownloadForms.ipynb file and downloaded the 10-Q forms of the S&P 500 companies with cik available in .txt format. However, due to the lack of UTF-8 support on our local machines, we had to switch to using the edgar-crawler package as an alternative.

- *Edgar-crawler*
  This is an open-source tool to retrieve any financial reports from the SEC EDGAR database. We used the edgar_crawler.py module to download all the 10Q of the S&P 500 companies across 2019 to 2023. The filings are in htm format so we had to extract all the natural language text for further analysis. We modified the extract_items.py module to extract the text and save them into a json file including cik, filing_date and other information. As the format of 10Q is not consistent, extracting only the MDA section would result in less effective input data. We chose to extract all the text in the 10Q.
  https://github.com/nlpaueb/edgar-crawler?tab=readme-ov-file#Accompanying-Resources

  We use a specific CIK list for each quarter to download the 10-Q filings. Since the filing dates for 10-Qs often lag behind the quarter's end, we rely on the CIK list for the S&P 500 from the previous quarter. Notably, the constituents for 2019 Q1, Q2, and 2023 Q3 are absent from the historical data. For 2019 Q1, we used the 2018 Q4 CIK list and skipped 2019 Q2 and 2023 Q3. Additionally, the number of filings in the first quarter is typically around 100. As a result, the total number of filings in our sample is 6,390.

2. **Load_MasterDictionary.py and Generic_Parser.ipynb**

- We updated the Generic_Parser.ipynb file by creating three matrices required: tf_matrix (number of documents × number of negative words in lm_dictionary), idf_matrix (number of documents × number of negative words in lm_dictionary), and doc_length_matrix (number of documents × 1) in the main function.
- Additionally, we introduced the get_proportion function, which takes in the three matrices mentioned above as arguments and return the tf_idf_matrix (number of documents × number of negative words in lm_dictionary) and proportion_weights (proportion/term weights).
- The final output is a csv file that contains a table with the dimension (number of documents × 2), where the columns represent quintiles (ranging 1 to 5 based on proportion_weights, with 1 being the least negative and 5 the most negative) and corresponding filing dates for further processing.

3. The Return and Plotting Matrix

- Connect to WRDS, I use script retrieves daily returns for S&P 500 returns from CRSP, focusing on dates from August 1, 2017, onward. Then I merge this data with additional company information from crsp.msenames and drop duplicates. Next, I link the merged data with the Compustat link table (crsp.ccmxpf_linktable) to incorporate identifiers like gvkey and further company-specific information. I merge the linked data with Compustat's names table to include additional attributes like cik, resulting in a dataset that contains the date, company name, ticker, cik, and daily returns for each stock.

- Then I sorted the sp500 dataset by ticker, cik, and date, removed duplicate entries, filtered out rows with missing tickers, and ensured the cik column had no missing values and was converted to integer type for a clean and well-structured dataset.

- I loaded a list of S&P 500 tickers from an Excel file (provided by TA) and filtered the sp500 dataset to keep only those tickers, as some permno codes are the same for different stocks.

- Then, I created an info column by combining the ticker and cik values, pivoted the data to create a table of returns (ret) indexed by date and grouped by info, and ensured unique columns by retaining the most detailed version of each ticker-related column in the resulting pivot table.

info	AAL_6201	AAPL_320193	AAP_1158449	AA_4281	ABBV_1551152	ABC_1140859	ABMD_815094	ABNB_1559720	ABT_1800	ACE_896159	...	XRAY_818479
date												
2017-08-01	0.012292	0.008875	0.018213	0.012505	0.006723	-0.008953	NaN	NaN	-0.000610	0.010993	...	-0.008222
2017-08-02	-0.011947	0.047251	0.003770	-0.006375	0.004689	-0.018391	NaN	NaN	0.001628	0.005740	...	-0.005852
2017-08-03	0.001982	-0.009991	-0.034504	0.018087	0.003960	-0.104744	NaN	NaN	-0.001422	0.006379	...	0.003107
2017-08-04	0.004946	0.005271	0.004343	-0.007896	0.000000	-0.010770	NaN	NaN	0.002034	-0.000467	...	0.003912
2017-08-07	-0.004331	0.015474	-0.013693	0.008357	0.003240	-0.003093	NaN	NaN	0.000812	-0.003271	...	0.012177
...	...	...	...	...	...	...	...	...	...	...	...	...
2023-12-22	-0.002787	-0.005548	NaN	0.001870	0.015401	0.003807	NaN	-0.009079	0.005070	0.003409	...	0.000853
2023-12-26	-0.013976	-0.002841	NaN	0.005040	-0.002065	0.005171	NaN	-0.014773	0.001834	0.005889	...	0.009086
2023-12-27	-0.008505	0.000518	NaN	0.003900	0.001682	0.000686	NaN	-0.015643	0.005127	0.002792	...	0.002532
2023-12-28	-0.000715	0.002226	NaN	0.001110	-0.000839	-0.003623	NaN	0.003295	0.005556	0.007814	...	0.002807
2023-12-29	-0.017167	-0.005424	NaN	0.000185	0.001422	0.009288	NaN	-0.006277	-0.002989	0.006996	...	0.000000

- After that I queried and processed S&P 500 return data from CRSP, renaming and formatting the date column. Then, using a rolling window of 90 days, I calculated the

beta values for each stock in return_df_clean by performing linear regressions between individual stock returns and S&P 500 returns, storing the results in beta_matrix. NaN values were handled by skipping windows containing missing data.

```
data = {
    'AAPL': [0.01, 0.02, -0.01, 0.03, -0.02, 0.01, np.nan, 0.04],
    'MSFT': [0.02, np.nan, 0.01, -0.02, 0.03, 0.01, 0.02, -0.01],
    'GOOG': [0.03, 0.01, -0.02, np.nan, 0.01, 0.02, 0.03, 0.00]
}

dfs = pd.DataFrame(data, index=pd.date_range(start='2023-01-01', periods=8))
df_filled = dfs.fillna(0)
cumulative_returns_3d = (df_filled + 1).rolling(window=3).apply(lambda x: x.prod(), raw=True) - 1
next_3_day_cumulative_returns = cumulative_returns_3d.shift(-3)
```

- For plotting I define function that compute rolling mean and then shift it back so that we can match quantile based on cik and filling date (with function that push it back if it is not a trade date).

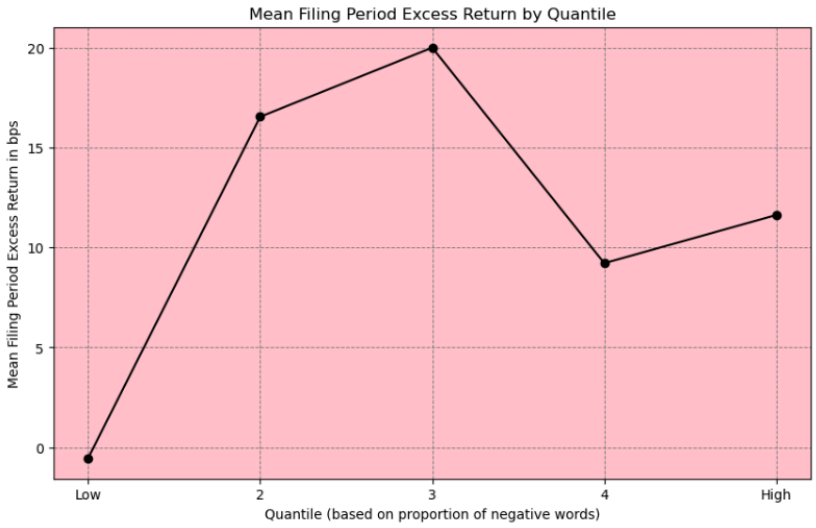
```
def get_closest_trade_date(date, trade_dates):
    future_dates = trade_dates[trade_dates >= date]
    if not future_dates.empty:
        return future_dates[0]
    return None
```

```
cumulative_returns_3d.shift(-3)
```

	AAPL	MSFT	GOOG
2023-01-01	0.040094	-0.010200	-0.010200
2023-01-02	-0.000694	0.019494	-0.010200
2023-01-03	0.019494	0.019494	0.030200
2023-01-04	-0.010200	0.061106	0.061106
2023-01-05	0.050400	0.019898	0.050600
2023-01-06	NaN	NaN	NaN
2023-01-07	NaN	NaN	NaN
2023-01-08	NaN	NaN	NaN

```
[0.02, -0.01, 0.03]
```

```
[0.02, -0.01, 0.03]
```



4. Conclusion

It appears that the "McDonald's effect" does not persist based on our findings, and this could be attributed to several factors.

Firstly, the paper indicates that companies have become less inclined to use negative language, possibly due to an increased awareness of how public perception and investor reactions are shaped by sentiment analysis. This shift towards more neutral or positive language could diminish the impact of traditionally negative indicators, leading to reduced predictive power in our analysis.

Secondly, it's important to note that we are only analyzing 10-Q filings, which provide a retrospective view of a company's financial health rather than forward-looking performance forecasts. As 10-Qs are intended primarily to update investors on past performance and current risks, they may not fully capture the optimistic or pessimistic outlook that could directly influence future stock returns.

Lastly, the dictionary we used for sentiment analysis is from 2018, which may limit its effectiveness in capturing evolving language trends. Corporate communication styles and the terminology used in financial reports have likely changed over time, leading to discrepancies between the dictionary's vocabulary and the language currently in use. An outdated dictionary might fail to accurately classify sentiment, thereby reducing the reliability of our findings.

To address these challenges, we may consider updating our sentiment dictionary, incorporating more recent language trends, and expanding the analysis to include other documents such as earnings forecasts or conference call transcripts that contain forward-looking statements, providing a more holistic view of sentiment in relation to company performance.