Executive Summary

The media can have quite an influence on the public's opinion about certain topics. That is no different when considering investor attitudes towards company stock market performance. The purpose of this paper is to examine how authors' opinions and attitudes portrayed through news articles affect the performance of stocks within the S&P 500 index. This analysis focuses on the various readability index scores, word counts and structures, and finally sentiment scores from a day's news to predict the stock price movement of the following day.

The process begins by web scraping news article contents and passing the text through readability and sentiment models to generate scores. The readability scores determine how easily an audience can interpret and comprehend the article contents. The sentiment scores produce ratings of how the emotion of the article contents can measured in terms of positivity, neutrality, and negativity. Incorporating these same features in a decaying, lagging manner from multiple days prior provide for a more in-depth analysis since some news can have an affect over multiple trading days. This led

Add final results and findings…

Introduction

This project is designed to explore the relevance of news articles and how it reflects in the movement of stock prices. The sentiment of news articles can portray an attitude toward a certain topic or company. This attitude could then be passed on the thousands of readers and affect their behaviors within the market with respect to the stock mentioned in that article. To evaluate if this hypothesis is present, news must be collected, audited, and then evaluated in comparison with stock returns.

Data

This project required several types of data to be collected from various sources. The diverse data types covered were news articles, the S&P 500 index composition, and stock open and close prices. The main dataset that includes the news article details was collected from the GDELT 2.0 Event Database. GDELT releases event files in 15-minute increments every day of the week. To collect all files, a script was developed to navigate and download the event data by adjusting the timestamp to the GDELT file links. The data spans the entire year of 2021 from January 1st to December 31st. Before processing, the GDELT data contained 26 columns including columns regarding dataset ID, website, URL, title, people, topics, etc. Downloading the entire data files caused some storage issues as each monthly file totaled about approximately 15GB. After processing to reduce storage, three columns were retained: dataset ID which was converted to a date stamp, website, and URL. This reduced the monthly file size to about 15MB. The date column was used to specify input and output rows when trying to use a day's news to predict the next day's stock price movement. The website was use for filtering, while the URL column was required for web scraping. The S&P 500 index stock composition was pulled from Wikipedia. The stock open and close prices were gathered from Yahoo Finance. The web scraping stage of the project began with three different tasks: extracting text from news articles, gathering the list of S&P 500 companies and tickers, and downloading the stock price open and close prices for 2021 for the S&P 500 companies.

When deciding which websites to gather news from, the goal was to select 3-5 popular and reputable, but unbiased websites that produce an adequate volume of news on a daily basis. Some limiting factors in the selection included websites that have paywalls that limit web scraping actions along with websites without a consistent webpage format that prevents consistent web scraping. The three websites that were selected were marketwatch.com, yahoo.com, and prnewswire.com. All three websites rank in the top 20 in terms of volume of news and have dependable formatting that allowed for site specific templates to be created. All three scraping templates were developed using the *BeautifulSoup* package. An issue that arrived when web scraping news articles from a year prior was that some URLs were dead and no longer active. To handle this, specific error handling techniques were needed to continue along with other active URLs. If a URL was no longer active, then this row of data was removed from the dataset. The *BeautifulSoup* package was also used to gather the S&P 500 companies and tickers chart from Wikipedia. To download the historical stock prices for each of the S&P 500 tickers, a web driver package from *selenium* and *ChromeDriverManager* was required to navigate around and click icons on Yahoo Finance.

***update and format chart with most current data**

rank	website	counts
1	iheart.com	161,187
2	msn.com	81,981
3	medium.com	48,707
4	reuters.com	37,931
5	yahoo.com	28,885
6	indiatimes.com	26,354
7	prnewswire.com	22,544
8	dailymail.co.uk	18,859
9	texasguardian.com	18,031
10	apnews.com	17,731
11	sandiegosun.com	16,727
12	newyorktelegraph.com	16,668
13	ianslive.in	13,846
14	sfgate.com	12,787
15	chron.com	12,676
16	marketwatch.com	12,166
17	washingtontimes.com	11,970
18	onenewspage.com	11,206
19	thenews.com.pk	10,850
20	thehindu.com	10,523

Methodology
** format citations for final draft*

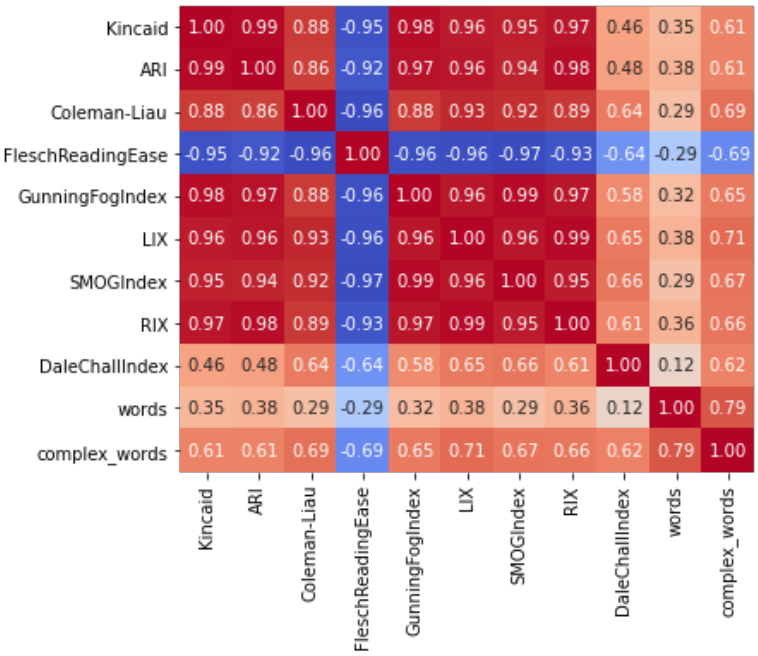
Once the web scraping and data collection processes were complete, the rest of this project focused on data processing and computing. There were two main methodologies that required extensive amounts of troubleshooting resulting in plenty of trial and error; these include computing sentiment/readability scores, assigning these scores to the appropriate stocks, and finally stock return modeling. The first two parts were combined together and run on a single script on Amazon Elastic Compute Cloud (EC2). The main challenge of this project was scaling computations from a few months to an entire year of data. This task occurred throughout the duration of the project from loading data in, to quantifying sentiment and readability scores for monthly datasets. To resolve this issue, processes needed to be transferred to a cloud computing environment such as (EC2). The process starts by loading in the GDELT data, navigating to the URL, web scraping the article contents, computing both sentiment and readability scores, tagging articles with associated stock tickers, rearranging those article tags to be their own unique rows, and finally saving the new dataset. This new dataset was then used to create some additional features before modeling occurred.

In developing sentiment scores for each article, a natural language processing (NLP) application programming interface (API) was deployed from *HuggingFace*. *HuggingFace* is an open-source, artificial intelligence community specializing in NLP technologies. The model of choice from *HuggingFace* was the *mrm8488/distilroberta-finetuned-financial-news-sentiment-analysis model* which is a "fine-tuned version of the *distilroberta-base* on the financial_phrasebank dataset." (https://huggingface.co/mrm8488/distilroberta-finetuned-financial-news-sentiment-analysis) This financial specific language training was critical in the selection of this model. The *distileroberta-base model* consists of 6 layers, 768 dimension and 12 heads, totalizing 82M parameters and runs twice as fast as its predecessor the *RoBERTa-base model* which has a total of 125M parameters (https://huggingface.co/distilroberta-base). The *RoBERTa-base model* is a transformers model, that is self-supervised on raw English text which eliminated the process of human labeling. The model objective was to use Masked language modeling (MLM), which involves masking 15% of the words in a sentence and using the context of the sentence to predict those masked words (https://huggingface.co/roberta-base).

The *mrm8488/distilroberta-finetuned-financial-news-sentiment-analysis model* works best when text is passed in on a sentence-by-sentence basis. This was achieved by scraping the news contents and splitting the texts at periods with a space afterwards. The output of the model produces three sentiment scores: positive, neutral, and negative. The combination of these three scores sums to one. A net sentiment scores was manually calculated by subtracting the negative sentiment score from the positive sentiment score. After computing the sentiment scores for each sentence, the scores were aggregated together to find summary scores, such as mean, median, minimum, and maximum, for each sentiment type of positive, neutral, negative, and net. In total, there are 16 summary sentiment scores for each article.

The readability scores were determined using the *readability* package. The package consists of traditional readability measures focused on basic characteristics of raw text. The results are essentially linear regressions based on the forementioned basic characteristics such as words, syllables, and sentences (https://pypi.org/project/readability/). The types of output that were collected from this package were the *readability grades* and the *sentence info* portion of the results. The entirety of the *readability grades* was selected for analysis including Kincaid, ARI, Coleman-Liau, Flesch Reading Ease, Gunning Fog Index, LIX, SMOG Index, RIX, and Dale Chall Index. Only two measurements , word count and complex word count, were selected from the *sentence info*. To see how each of the scores correlated with one another and evaluate if the number of features could be reduced, a correlation matrix was produced and showed the readability scores were highly, positively correlated, with exception of Flesch Reading Ease which was highly, negatively correlated because its readability scale works in the opposite direction of the other scores. The word count basically had no correlation with the readability scores, however the complex word count did have a slight positive correlation. After examining the correlation matrix, both word counts were removed because word count did not provide much information and complex word count was assumed to already be captured within the readability scores. Since the readability scores were highly correlated, each score was scaled and added together to find an average readability score to reduce the feature count.

(**update correlation chart)**



After collecting both the sentiment and readability scores, each article needed to be scanned to detect which of the S&P 500 stocks were mentioned in the contents of the article. To accomplish this, the *fuzzywuzzy* package was imported. *Fuzzywuzzy* is a string-matching package that applies Levenshtein

Distance to differentiate distances between different character sequences (https://pypi.org/project/fuzzywuzzy/). Once applying *fuzzywuzzy* extraction of string matches between article contents and company names that were scraped from Wikipedia, the matching function used was the *token set ratio.* The *token set ratio* tokenizes the strings while converting all letters to lower case and removing punctuation. Then, the common tokens are removed, and a ratio is calculated producing a rating from 0 to 100. A filter was then applied to only display tags with a rating of 90 or above. The main reason for using the *token set ratio* was it is most useful when comparing sets of strings that greatly differ in length, which is what is happening in this example of comparing one or two word company names to paragraphs of news text (https://towardsdatascience.com/string-matching-with-fuzzywuzzy-e982c61f8a84). Once this process was completed, a list of stock tickers was produced for each article except for articles that did not result in any stock tags. The rows of data with no tags were dropped from the dataset. However, to reduce complications for future modeling, each stock tag for each URL article needed to be separated to its own individual row. The final output would then be a row of data for each stock mentioned within each article along with the associated sentiment and readability scores for the article.
(**Add histogram of article tags)**

In the modeling portion of this study, the two focal points were on stock returns relative to the average return of the individual companies in the S&P 500 and a classification prediction of a daily stock price increasing or decreasing. Both models would include the features that were mentioned above, specifically to use today's news to predict the return of a stock between tomorrow's open and close price. Some additional features that may be created are some lagging sentiment scores meaning that today's news could be used to predict stock movement for multiple days into the future in addition to tomorrow. To create this lag feature, an exponential decay factor would be applied to lessen the effect of news as time moves on. A similar process was also implemented for days where a company had no news sentiment rating. The default for this specific day was zero, but using the lagging look back period, a supplemental score can be generating by discounting the previous days' sentiment scores. The last feature to be added would be considered an importance factor which can be represented by the volume of news per day, per stock. An aggregated sentiment score from two news articles should be evaluated in a different manner than a sentiment score produced by 15 news articles. **Add details about model selection and tuning…**

Additionally, a long-short portfolio construction is also a possibility. This process involves separating the S&P 500 stocks into a specified number of bins based on their net sentiment scores, and then calculating their raw and relative returns to the rest of the S&P 500 stocks. Ideally, the ordered bin sentiment scores should be highly negatively correlated with the bin aggregated average returns.

Results

Add results after modeling/portfolio construction is complete

Conclusion

Explain how results can be applied to the real-world application. Added suggestions for improvement below…

There are ways to proceed with room for improvement. The first being to gather additional years of data along with selecting more news sources when filtering on websites. This would allow for more training data while providing additional news perspectives. Another improvement that could be made would be in regards to the entire script that is ran on AWS EC2. Adjusting the script to run parallel

processing could cut run time in half. Finally, the analysis of the feature and prediction horizon could be adjusted to examine which timeframe produces the best results. Currently, the study uses today's news along with some lagging news to predict tomorrow's returns. Instead, investigating weekly, monthly or quarterly timelines could produce different results with more significance. This could be more beneficial for long-term investors rather than short-term investors.