Predicting stock returns from earnings calls

Computing NEA - Victor Shao

Thursday 23rd March, 2023

Pages: 190

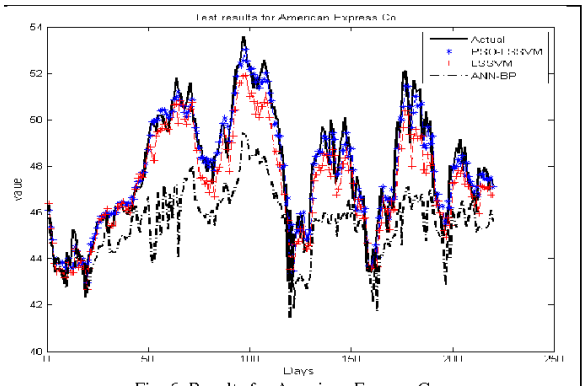


Fig. 6: Results for American Express Co

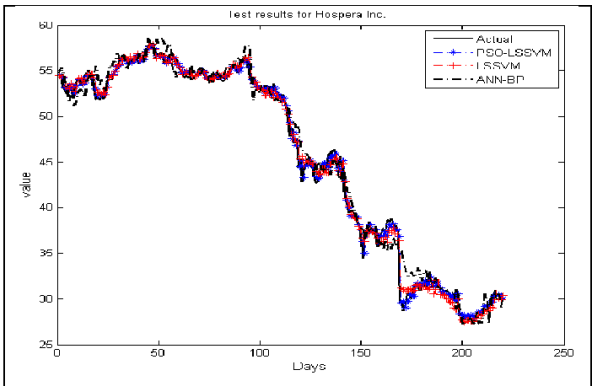


Fig. 9: Results for Hospera Company

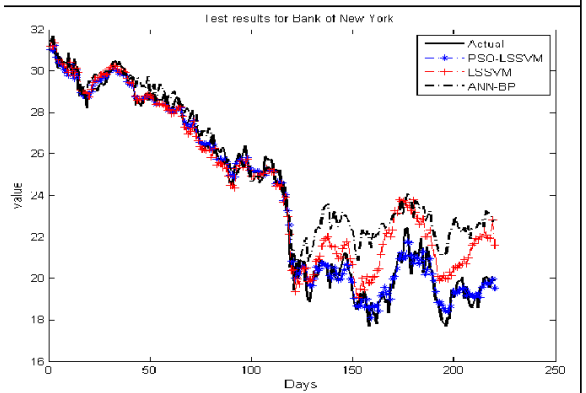


Fig. 7: Results for Bank of New York

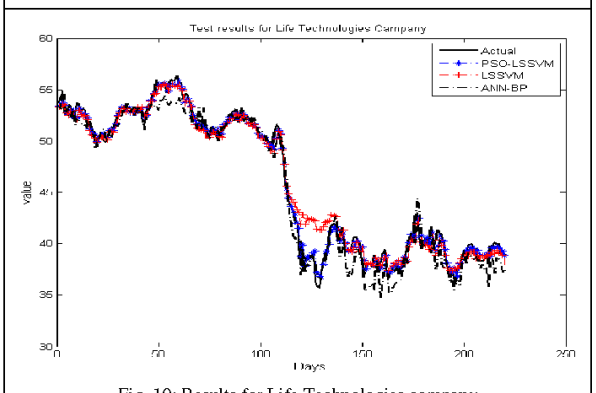


Fig. 10: Results for Life Technologies company

Table of Contents:

Analysis

Problem Area

Forecasting the movement of stock prices has been a widely researched topic in both industry and academia. Granted, these programmes may not predict the future with absolute certainty, nonetheless, we can encompass them with other methods to gain an upper edge in investing or trading. Hedge funds often use these models to analyse or find opportunities in the securities market. However, these advanced models are not widely available for retail investors due to their complexity and competitive edge.

Although the machine learning algorithm I aim to produce is not as sophisticated as one made by a billion-dollar hedge fund', my goal for this project is to produce a web application that can provide retail investors with an accessible and well-rounded stock forecast.

Existing stock forecast materials such as the one provided by https://simplywall.st is limited to a *Discounted Cash Flow (DCF)* model, which is used for finding the stock's intrinsic value. The DCF model discounts all future cash flow as a geometric series sum. The intrinsic value is a price range at which the stock is estimated to be "fair valued". The issue with this model is it ignores any sentiment or momentum indications for the stock – which are more correlated to the stock price in the short term. Thus, the solution is to provide a well-rounded forecasting model, based on a multitude of inputs, including the intrinsic value, momentum analysis, sentiment analysis, and various features related to earnings calls, or annual/quarterly reports.

Client/End User

The End Users of this product are retail investors. Certain many of them may want to "save" specific stock tickers and their relevant information – in addition to providing forecasts, the web application will offer an authentication mechanism where users can log in/out, and save relevant information in a "favourite list".

Existing stock forecast materials such as the one provided by https://simplywall.st offer a free plan where the user is limited to 5 companies searches per month. Most retail investor stock forecasting services are also restricted behind a paywall or subscription. My solution is to offer a free service for retail investors, equipt with the option to browse freely without any restrictions.

There may be retail investors who prefer a more in-depth description, thus in addition to providing a percentage figure forecast (i.e. >10%), it will be prudent to provide additional information on the probability of meeting such forecast and compare it with other classification labels (i.e. 60% chance of stock moving >10%; 90% chance of stock moving >5% etc.).

Per the regulations that govern providing financial advice, this web application does not issue stock recommendations, but rather, it is used as a forecasting tool which complements its own market research. The website will offer relative forecasts for each stock, including their expected trend and probability, however, it will be the end user's responsibility for due diligence.

Research Methodology

Interview:

I've asked my stock enthusiast friend a list of questions including, "How often do you research stock", "What are some problems with existing stock forecasting tools", or "What features do you think can show correlations with future stock price movements?". The feedback I've received was insightful, for example, we discussed how if an earnings transcript is released 1-3 days after the earnings transcript, the price of the stock may have already moved during that timeframe. Therefore, it may be beneficial to factor in the percentage change in the price of the stock between the 1-3 days interlude period, so the model will be more realistic.

Existing Websites:

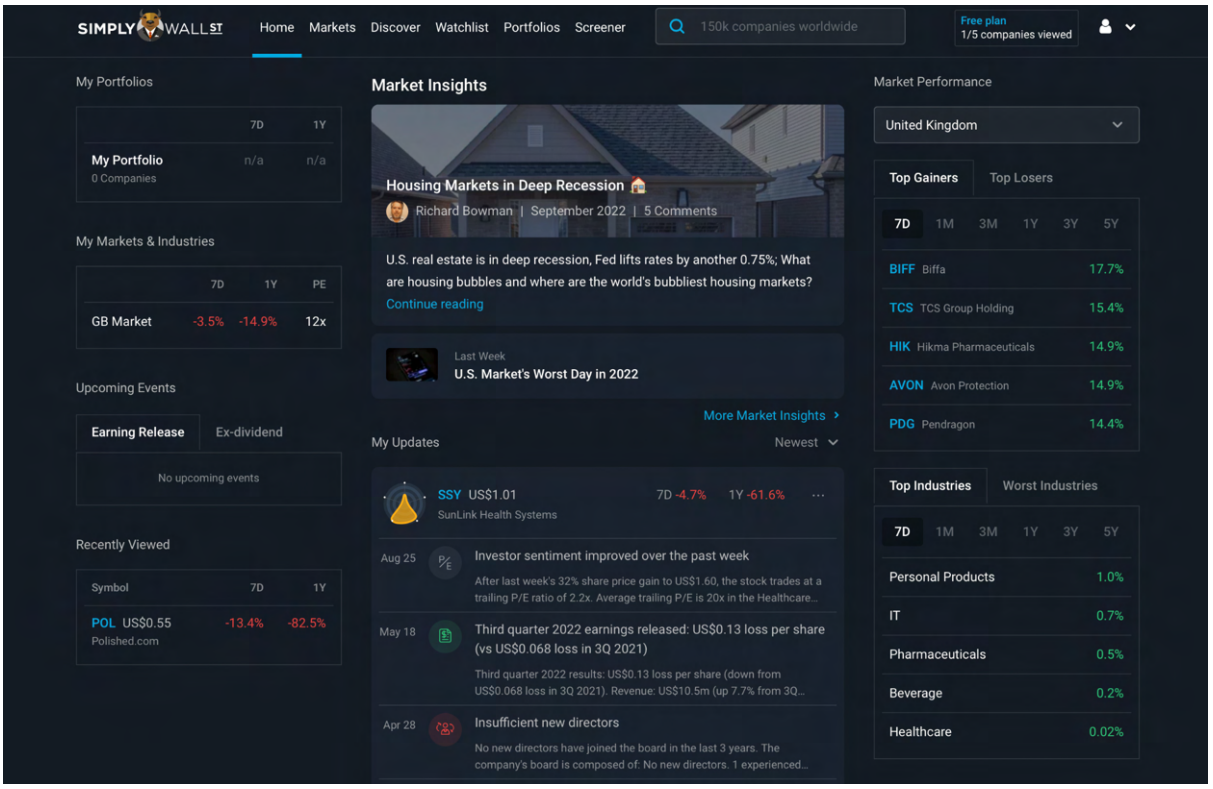


Figure 1

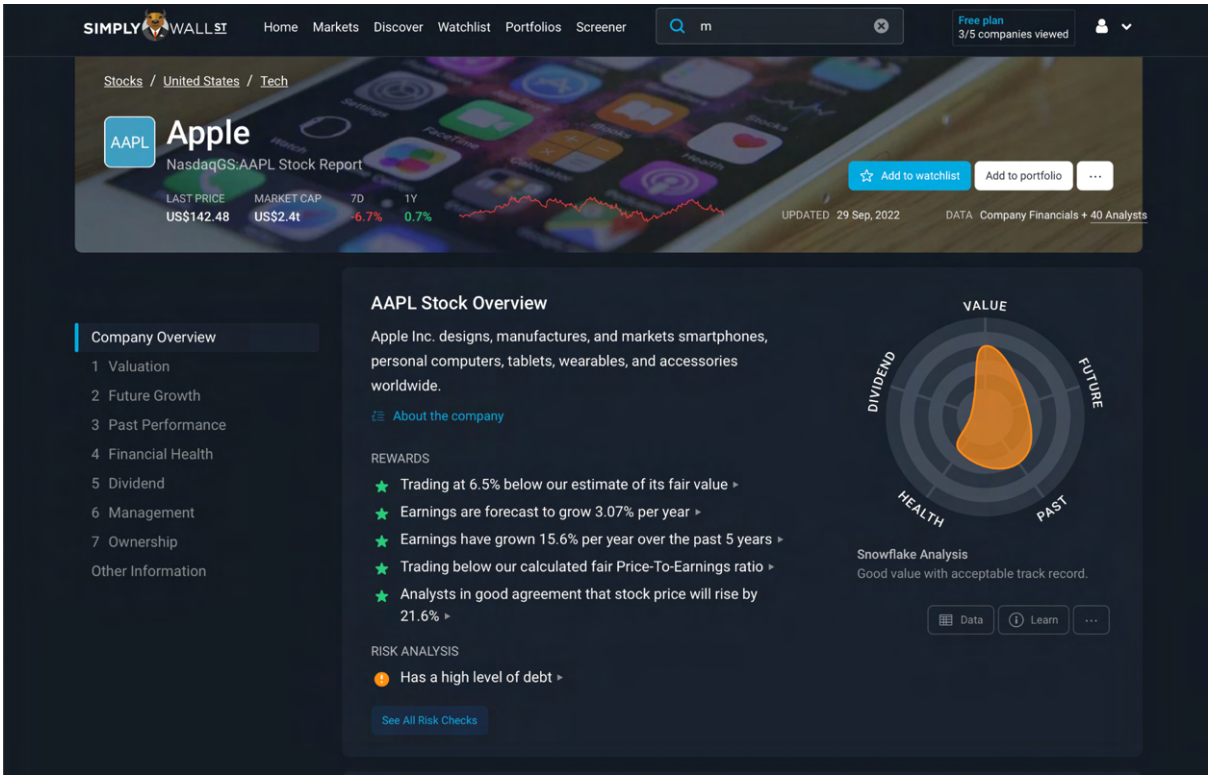


Figure 2

I have also investigated similar websites, such as https://simplywall.st, and https://finviz.com. Simplywall.st offers features such as analyst forecasts,

intrinsic value estimations, financial reports, and operating results of a specific stock. It also allows users to search specific stock tickers and save them to their accounts. Users can not only download relevant data to .pdf and .csv format but compare between different stocks.

The website offers graphical interpretations of different measurements, sometimes interactive. Simplywall.st's GUI is also appealing for the user in that it has lots of smooth edges, variety of colours (such as different shades of red to green for risks), the option for dark mode, fast response time, and clean background.

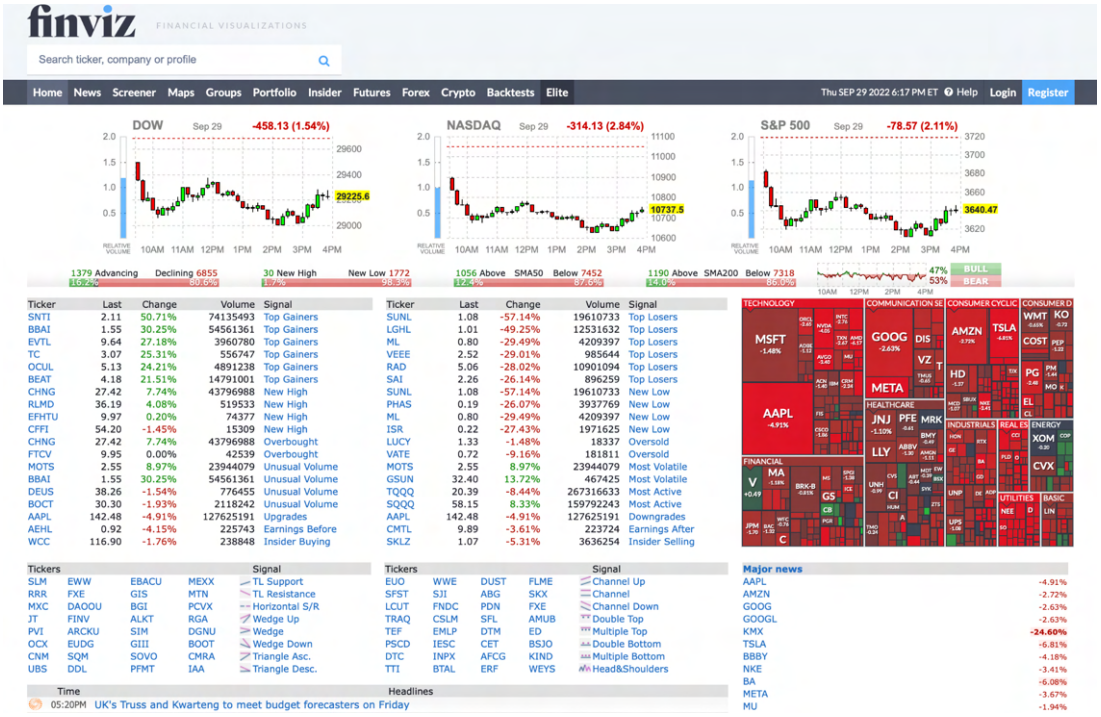


Figure 3

Index	-	P/E	6.99	EPS (ttm)	5.07	Insider Own	0.20%	Shs Outstand	182.00M	Perf Week	-5.57%
Market Cap	6.74B	Forward P/E	5.35	EPS next Y	6.62	Insider Trans	0.00%	Shs Float	173.82M	Perf Month	-30.24%
Income	963.00M	PEG	-	EPS next Q	0.38	Inst Own	86.20%	Short Float	5.55%	Perf Quarter	-22.27%
Sales	13.39B	P/S	0.50	EPS this Y	347.00%	Inst Trans	-1.09%	Short Ratio	1.56	Perf Half Y	-61.57%
Book/sh	31.13	P/B	1.14	EPS next Y	-5.02%	ROA	6.30%	Target Price	58.05	Perf Year	-29.95%
Cash/sh	8.62	P/C	4.11	EPS next 5Y	-	ROE	20.50%	52W Range	33.70 - 98.09	Perf YTD	-40.53%
Dividend	0.40	P/FCF	6.04	EPS past 5Y	24.80%	ROI	5.00%	52W High	-63.88%	Beta	2.36
Dividend %	1.13%	Quick Ratio	1.00	Sales past 5Y	5.50%	Gross Margin	27.90%	52W Low	5.13%	ATR	2.77
Employees	12200	Current Ratio	1.80	Sales Q/Q	28.60%	Oper. Margin	12.70%	RSI (14)	29.92	Volatility	5.50% 5.88%
Optionable	Yes	Debt/Eq	0.30	EPS Q/Q	81.50%	Profit Margin	7.20%	Rel Volume	2.13	Prev Close	35.23
Shortable	Yes	LT Debt/Eq	0.30	Earnings	Jul 20 AMC	Payout	5.70%	Avg Volume	6.19M	Price	35.43
Recom	2.20	SMA20	-19.65%	SMA50	-26.14%	SMA200	-42.04%	Volume	13,158,793	Change	0.57%

Figure 4

Finviz.com is a stock screener for retail traders. It's GUI design is not as appealing compared to simplywall.st due to its rudimentary design as shown in Figure 3 and 4. However, finviz offers more advanced financial information such as financial ratios as shown in Figure 4, in addition it includes more detailed insider, analyst information than simplywall.st. It also has the option for setting alerts, and interactive advanced stock charts. It also has similar features to simplywall.st, such as the options to compare prices, and save to watchlist. Both websites has a search bar and authentication for better user experience.

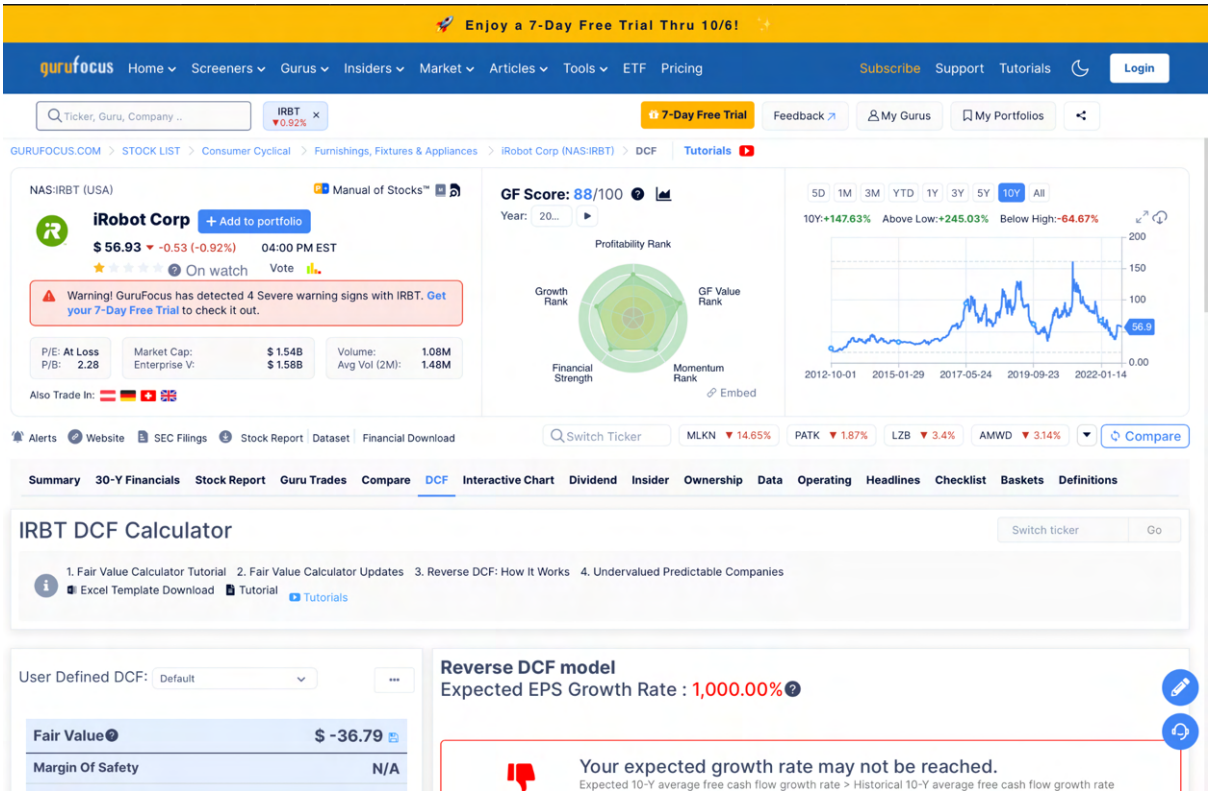


Figure 5

Figure 6

Gurufocus.com is another stock research tool, this website sits between simplywall.st and finviz in that it has a decent GUI (compared to finviz) while being comprehensive enough to include advanced features such as:

- Financial ratios
- Intrinsic value estimator (DCF calculator)
- Insider trading/Analyst ratings
- Comprehensive annual/quarterly reports, historical data

Unlike Finviz, each feature has their own unique tab for each stock (see Figure 6) which makes it easier to browse, enhancing user experience by increasing efficiency.

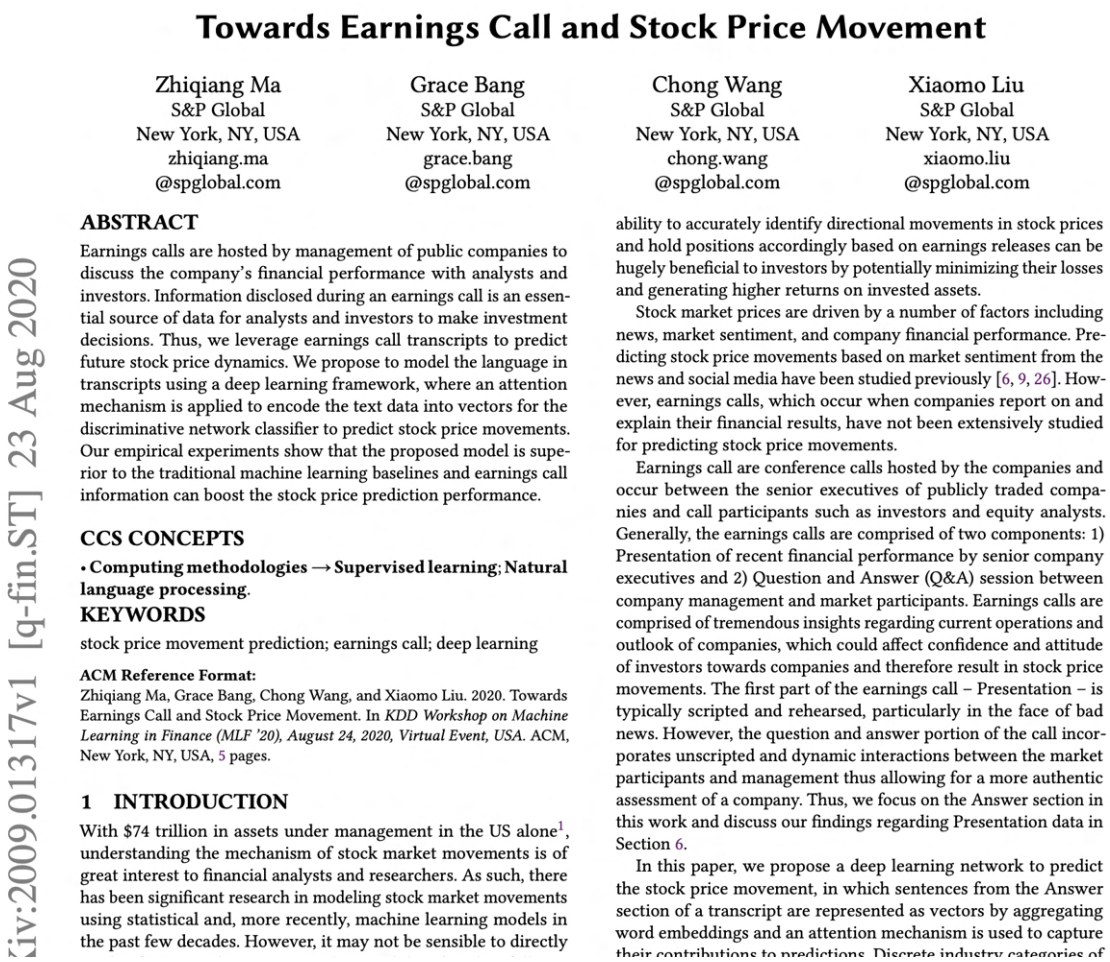
Other sources:

I have also research into various variables that may be correlated with forecasting stock prices, including intrinsic value, earnings call analysis,

relevant natural language processing and neural network models, features for momentum/technical analysis and more. This is done by browsing:

- Research/academic papers
- Online blogs
- Github repos
- YouTube tutorials.

Research/academic papers:



Towards Earnings Call and Stock Price Movement

Zhiqiang Ma, S&P Global, New York, NY, USA, zhiqiang.ma@spglobal.com
Grace Bang, S&P Global, New York, NY, USA, grace.bang@spglobal.com
Chong Wang, S&P Global, New York, NY, USA, chong.wang@spglobal.com
Xiaomo Liu, S&P Global, New York, NY, USA, xiaomo.liu@spglobal.com

ABSTRACT

Earnings calls are hosted by management of public companies to discuss the company's financial performance with analysts and investors. Information disclosed during an earnings call is an essential source of data for analysts and investors to make investment decisions. Thus, we leverage earnings call transcripts to predict future stock price dynamics. We propose to model the language in transcripts using a deep learning framework, where an attention mechanism is applied to encode the text data into vectors for the discriminative network classifier to predict stock price movements. Our empirical experiments show that the proposed model is superior to the traditional machine learning baselines and earnings call information can boost the stock price prediction performance.

CCS CONCEPTS

• **Computing methodologies** → **Supervised learning**; **Natural language processing**.

KEYWORDS

stock price movement prediction; earnings call; deep learning

ACM Reference Format:

Zhiqiang Ma, Grace Bang, Chong Wang, and Xiaomo Liu. 2020. Towards Earnings Call and Stock Price Movement. In *KDD Workshop on Machine Learning in Finance (MLF '20), August 24, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 5 pages.

1 INTRODUCTION

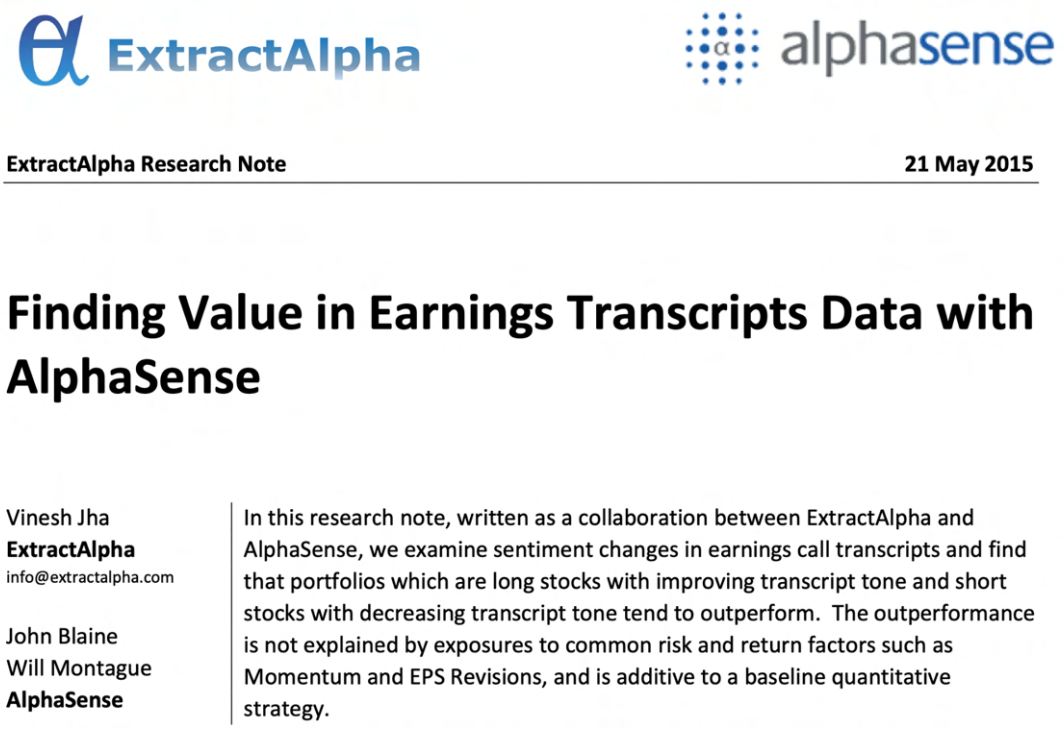
With $74 trillion in assets under management in the US alone[1], understanding the mechanism of stock market movements is of great interest to financial analysts and researchers. As such, there has been significant research in modeling stock market movements using statistical and, more recently, machine learning models in the past few decades. However, it may not be sensible to directly ability to accurately identify directional movements in stock prices and hold positions accordingly based on earnings releases can be hugely beneficial to investors by potentially minimizing their losses and generating higher returns on invested assets.

Stock market prices are driven by a number of factors including news, market sentiment, and company financial performance. Predicting stock price movements based on market sentiment from the news and social media have been studied previously [6, 9, 26]. However, earnings calls, which occur when companies report on and explain their financial results, have not been extensively studied for predicting stock price movements.

Earnings call are conference calls hosted by the companies and occur between the senior executives of publicly traded companies and call participants such as investors and equity analysts. Generally, the earnings calls are comprised of two components: 1) Presentation of recent financial performance by senior company executives and 2) Question and Answer (Q&A) session between company management and market participants. Earnings calls are comprised of tremendous insights regarding current operations and outlook of companies, which could affect confidence and attitude of investors towards companies and therefore result in stock price movements. The first part of the earnings call – Presentation – is typically scripted and rehearsed, particularly in the face of bad news. However, the question and answer portion of the call incorporates unscripted and dynamic interactions between the market participants and management thus allowing for a more authentic assessment of a company. Thus, we focus on the Answer section in this work and discuss our findings regarding Presentation data in Section 6.

In this paper, we propose a deep learning network to predict the stock price movement, in which sentences from the Answer section of a transcript are represented as vectors by aggregating word embeddings and an attention mechanism is used to capture their contributions to predictions. Discrete industry categories of

Figure 7



ExtractAlpha Research Note 21 May 2015

Finding Value in Earnings Transcripts Data with AlphaSense

Vinesh Jha
ExtractAlpha
info@extractalpha.com

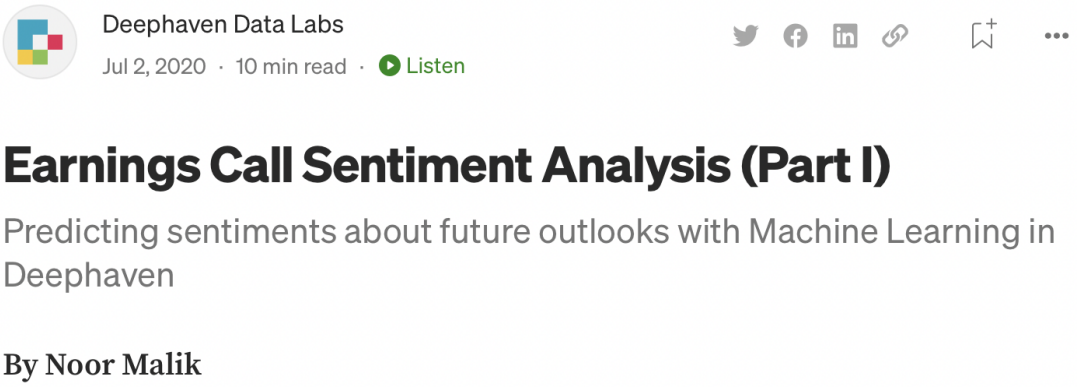
John Blaine
Will Montague
AlphaSense

In this research note, written as a collaboration between ExtractAlpha and AlphaSense, we examine sentiment changes in earnings call transcripts and find that portfolios which are long stocks with improving transcript tone and short stocks with decreasing transcript tone tend to outperform. The outperformance is not explained by exposures to common risk and return factors such as Momentum and EPS Revisions, and is additive to a baseline quantitative strategy.

Figure 8

Such as from S&P Global Market Intelligence, NYU Stern business school, and various graduate papers.

Online Blogs, and related Resources:



Deephaven Data Labs
Jul 2, 2020 · 10 min read · Listen

Earnings Call Sentiment Analysis (Part I)

Predicting sentiments about future outlooks with Machine Learning in Deephaven

By Noor Malik

Figure 7

Such as from medium.com and towardsdatascience.com

YouTube tutorials:

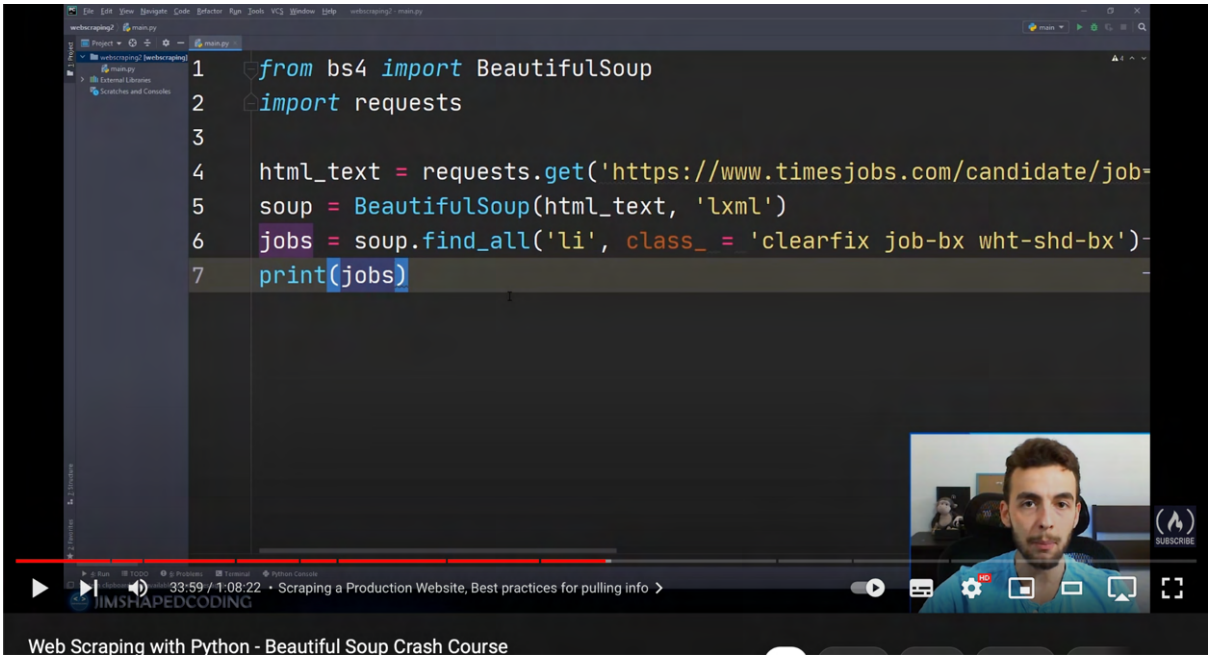


Figure 8 - Web scraping tutorials from YouTube for extracting data

All of these resources provided valuable insights on various feature extraction techniques, specifically those for earnings calls. In addition, papers such as “Technical analysis strategy optimization using a machine learning approach in stock market indices”, give a detailed analysis of the effectiveness of various momentum indicators in the context of stock forecasting via machine learning.

Features of proposed solutions

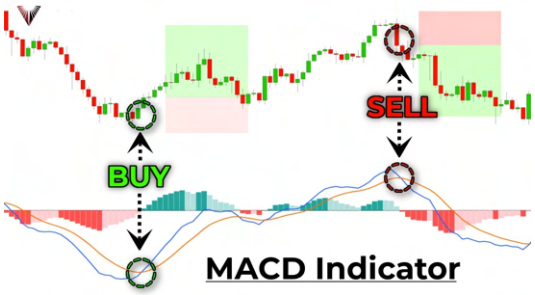
Website:

- Users can register and log in from the website. The authentication data will be stored in a non-relational database, specifically Google’s Firebase service. To register, the user will enter their email, password, and a confirmation password. This information will be sent to the Firebase server through a firebase API in Django.
- After logging into the website, users can “save” stock information to a firebase collection named “my favourites”. Users can create new collections, with the option to add/delete specific information. The stock information saved would be a record of:
    - Ticker name

    - Forecast how much the stock will increase (i.e. >10%)
    - Probability of each forecast (i.e. 70% for >10%)
- Users can choose to delete their account and reset their password, by communicating with the firebase server.
    - I may use mySQL instead of firebase if time and workload allows.
- The website will be using the Django Framework to communicate with the user and firebase.
- The machine learning algorithm will be operating in the back end.
- Appealing GUI for the user if time allows, a minimal viable product will be basic in terms of its UI, but functional.

Machine Learning:

- The machine learning algorithm will include stocks from both S&P500 and the Russell 2000 (US small cap stock index of 2000 companies), the features of each dataset include:
    - Sentiment analysis for earnings calls (using the Loughran and McDonald (L&M) Sentiment Master Dictionary)
    - Passage complexity analysis for earnings calls (using various Text, and indices, for example, the Flesch-Kinkaid score)
    - Similarities between earnings transcripts of the same stock ticker. For example how similar Apple's Q2 earnings call is to Apple's Q1 earnings call.
    - Similarities between annual and quarterly reports with previous years.
        - This feature will be included if time allows as interpreting historic annual and quarterly reports for 2500 companies is a time-consuming task for the programme to run effectively.
    - Percentage difference between stock price and intrinsic value (range of the stock's fair value)
        - This may be only attainable to an extent since I would need to either calculate the intrinsic value myself (difficult for each individual stock), or web scrape intrinsic value from websites like simplywall.st, unfortunately, this may not be possible since their service is limited to 5 companies per month

- 

    - Momentum Analysis, including moving average convergence/divergence (MACD indicator, see figure), Relative Strength Index, and Volumes to determine the stock's trend.
        - The Python library "yFinance" may not be as specialised for historical momentum indicators that include data from a decade or more ago.
    - Historical price change from the day before earnings call release to the day of earnings transcript release.
- The trained machine learning model can be serialised using *Pickle* (Python Library) and saved to a file. Therefore, I have to train the data first, store it into a file, and allow access to it on the website. When the user loads the website, the machine learning model is deserialised and able to be operated.
- When the user inputs the particular stock, the vector containing all the features of that stock will be computed, and then inputted into the deserialised machine learning model for predictions. The prediction results will then be displayed to the user.

Data models

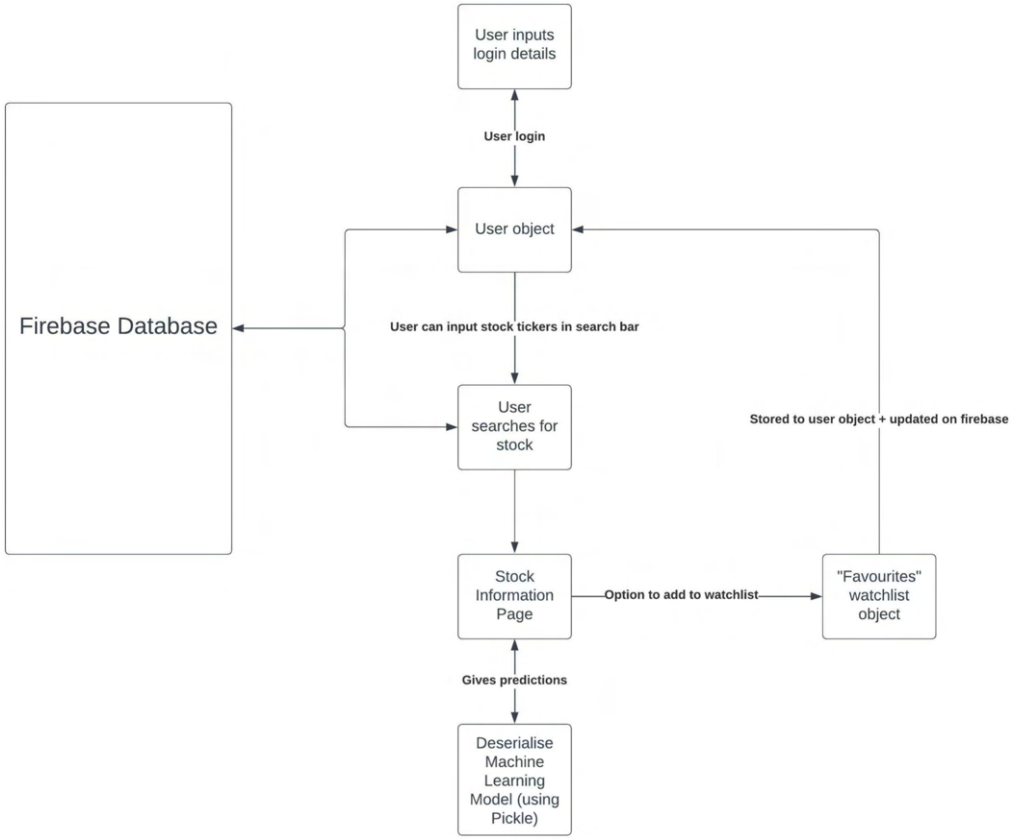


Figure 9 - Rough flow diagram of how the website interacts with database, user, and machine learning model.

As Figure 9 shows, the user can login (or register) for the website. The website will authenticate these information by querying the firebase database. Once the user is logged in, all the relevant information of this user will be stored in a user object. The user can then search for specific stock tickers by using the search bar on the website. When the user clicks on a specific stock, it will redirect them to a stock information page which shows them the relevant prediction and probability values. The user will be able to choose to save this “stock information page” to their watchlist. This information will be stored in the relevant user object, and subsequently to the Firebase.

On the stock information page, the website will:

1. Deserialise the machine learning algorithm using the Pickle library on Python
2. Calculates the various features of that stock and compiles it into a vector

3. The vector is ran inside the machine learning model, and the output values are then displayed to the user

Critical path

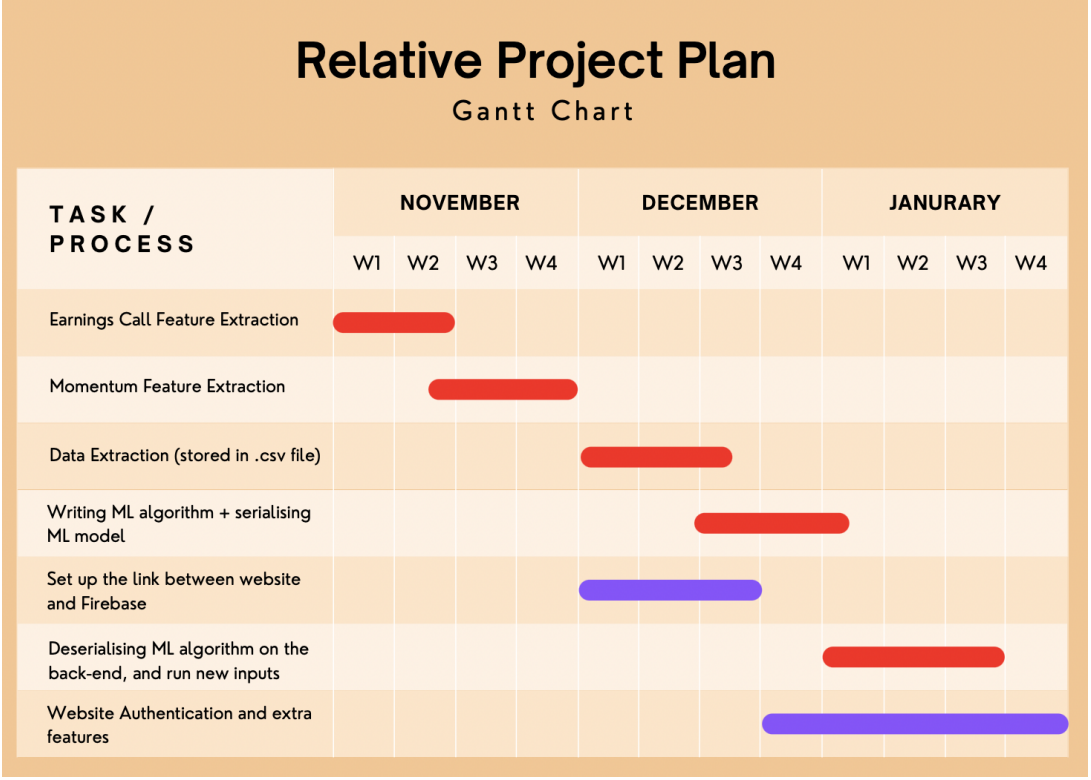


Figure 10 - Gantt chart showing the programming section's relative plan

The parts highlighted in red can be done in parallel with the purple parts. The red section is associated with the machine learning process, whereas the purple section is independent from it.

The red part starts with writing the programme for extracting various features from earnings calls of a set list of stock (e.g. S&P500), including sentiment, text-complexity and more features. Then I need to write the code for computing momentum analysis, such as volume and moving averages. Afterwards, I will need to run these code and extract the data onto a .csv file. This may take a dozen hours for the computer to successfully compute each feature vector from the set list of stocks. While the computer runs the code, I could start working on the website, specifically, linking it with Firebase and setting up User authentication methods.

After the programme extracts all necessary data to the .csv file, I would begin writing machine learning algorithms for it to be ran. Then, I would need to evaluate the different ML algorithms with one another and select the most optimal algorithm. This model will then be serialised using the Pickle library on Python, then saved as a file on the website. Once this is done, I need to be able to deserialise the data and allow new inputs to be ran as well.

Lastly, after the steps of Figure 10 is completed, I would start to “Prettify” the website to make it look more user friendly, and appealing. Then, it’s necessary for me to test all elements of the project and ensure there are no bugs.

Requirement Specification

The requirements specification is a document/contract with the client that outlines
what you will deliver. The contents need to have SMART (specific, measurable, achievable, realistic, timely) goals.

Website:

- Users have the option to register, login, delete their account, and edit their user information.
- Users can search for specific stock tickers through a search bar
- If the user wants to “predict” a stock, the programme will query, and compute the specific stock’s features into a “feature vector”, this is then fed into the deserialised machine learning algorithm for predictions.
- Users have the option to save the results of the stock information tab into a “favourites” list as a dataframe. Each dataframe will include information such as:
    - Stock ticker name
    - Date model is generated
    - Date model is saved
    - Prediction of the direction of stock’s movement
    - Probability of stock’s movement prediction
- Users can delete particular stock dataframes from their “favourites list”
- Users can create new “favourites” lists and name it as a string.
- The main page of the website will show the various “favourites” or other lists the user has when he logs in, as well as a stock search bar in the middle.

Machine Learning Model:

The machine learning model starts with feature extraction and includes:

- Sentiment analysis on earnings calls using FinBert Models
- Text complexity analysis on earnings calls
- Text similarity analysis on earnings calls
- MACD convergence/divergence
- Volume buy/sell ratio, and its deviation to average
- Potentially:
    - Word2Vec word embedding on annual reports analysis (if time allows)
    - Spotting variances in new annual/quarterly reports (i.e. new footnotes, and measure their sentiment using L&M dictionary)
    - Intrinsic value estimator (if web scraping beyond view limits allows, as some website restrict you to view companies per month.)
- Lastly, to normalise all above values using MinMaxScaler on Scikit Learn

The machine learning part will train using:

- S&P 500 companies historical data
    - The S&P500 is a common stock index used for tracking market performances in the US and consists of the leading 500 public traded US companies.
- Russell 2000 companies
    - The Russell 2000 is one of the most widely used index that tracks 2000 US small-cap companies. Small-cap companies are companies with a market capitalisation between $300 million to $2 billion.

I will then evaluate between difference machine learning models:

- Classification (Random Forest Classifier)
    - Binary vs Multiclass approach
- Deep learning models (i.e. LSTM)

The ML model will also be serialised using the Python Pickle library.

Design

Hierarchy chart

Figure 11 - Hierarchy Diagram of the machine learning algorithm (produced using lucid.app)

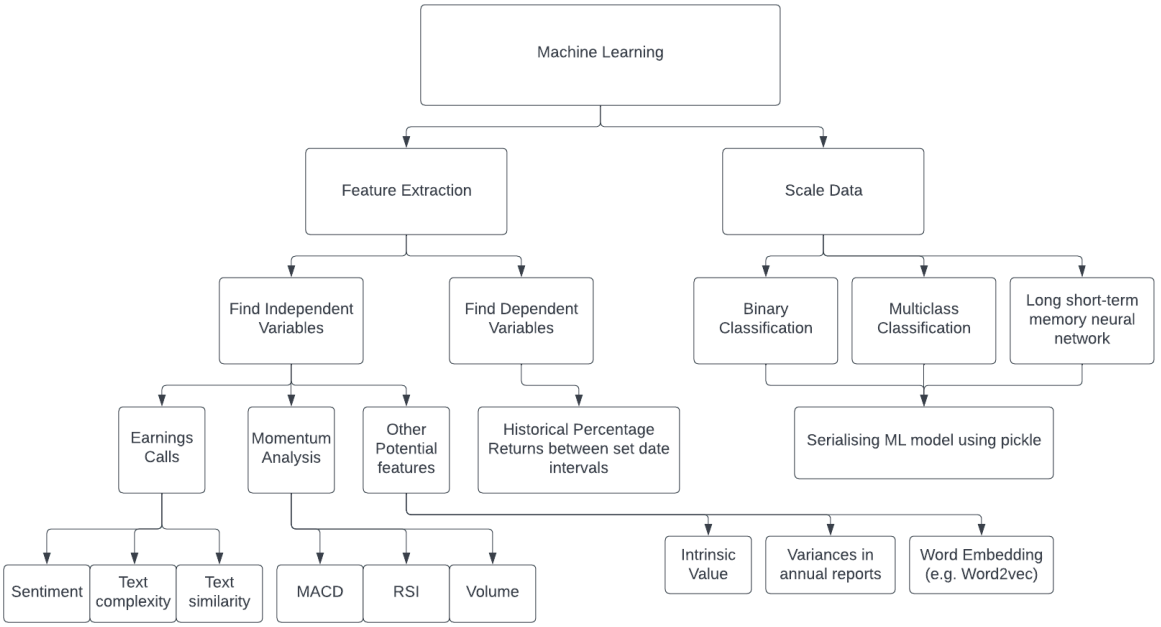


Figure 12 - Hierarchy Diagram of the website (produced using lucid.app)

The Hierarchy Diagrams above includes the most important modules and functions. I didn't include minor subroutines such as changing all texts to lowercase (needed before processing various earnings call text analysis).

Overall, the project is split between two parts: website, and machine learning. The user interacts with the web application; whereas the machine learning model is pre-computed outside, and then integrated in the web application.

Data structures / data modelling

Backend (via Firebase)

I will use Firebase instead of SQL since Firebase offers a wide range of custom features such as the option to sign in using a pre-existing Google, Facebook, Twitter etc. account. Furthermore, Firebase offers "Firebase hosting" which provides fast and secure hosting. These services are all free of cost as long as the stored data is less than 1 GiB; document write is limited to 20k/day; while document read is limited to 50k/day. Thus, I will be using Firebase due to the cost-effectiveness as well as its better security and flexibility.

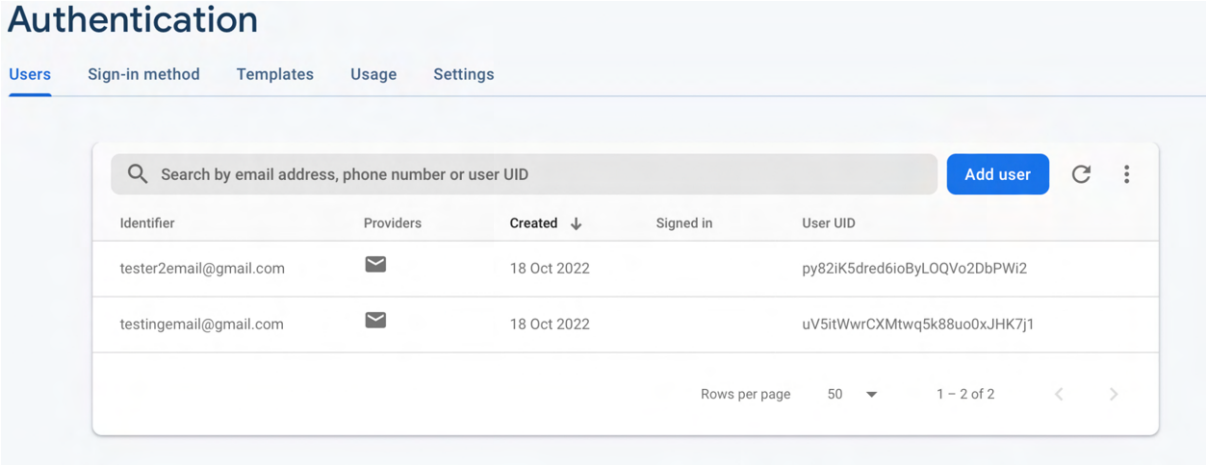


Figure 13a - Firebase Authenticator User Selections

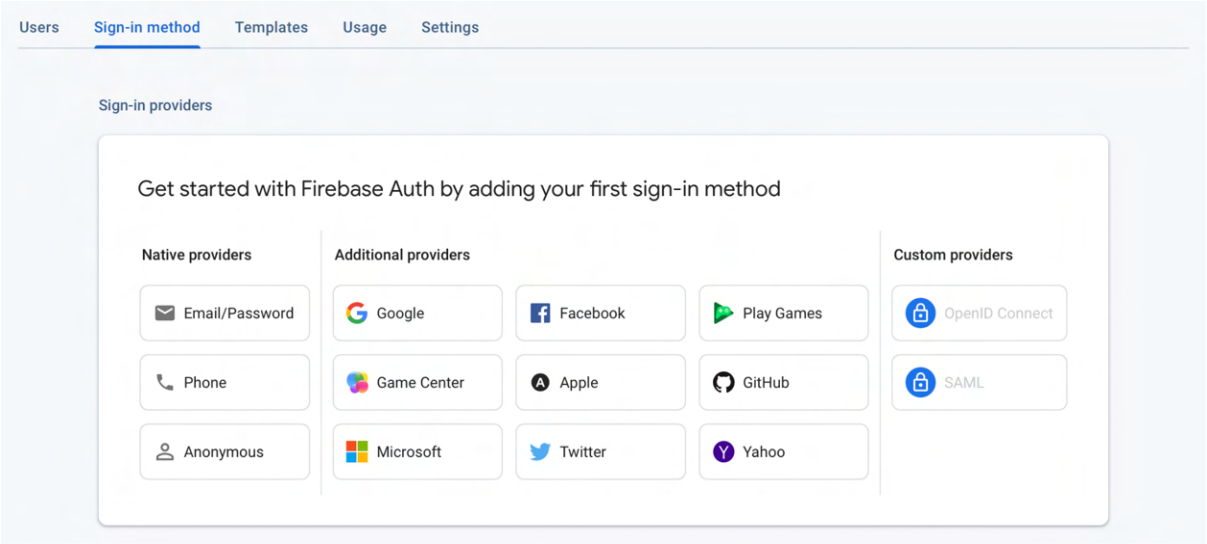


Figure 13b - Firebase Authenticator Sign-in providers

As shown in Figure 13b, Firebase provides a wide range of sign-in methods, including Email/Password and Google which I will use. To use this I would need to link Firebase with Django via the Python-Firebase API *"Pyrebase"*. I would access the authentication methods through Pyrebase as a CRUD API (CREATE, READ, UPDATE, and DELETE).

With reference to FIgure 13a, each user holds a unique User UID, along with their identifier (i.e. email), providers (e.g. email/password, Google log-in), and other user information.

For the database side of Firebase, I will use Firestore, which is a NoSQL document-oriented database. This consists of the three following objects: collections, documents, and fields. The data are stored in documents as fields, and the documents are stored in collections. Therefore, when a user wants to access

their saved lists of stock predictions, all of these informations will be inside a particular document (as fields) of a particular collection.
https://firebase.google.com/docs/firestore/data-model

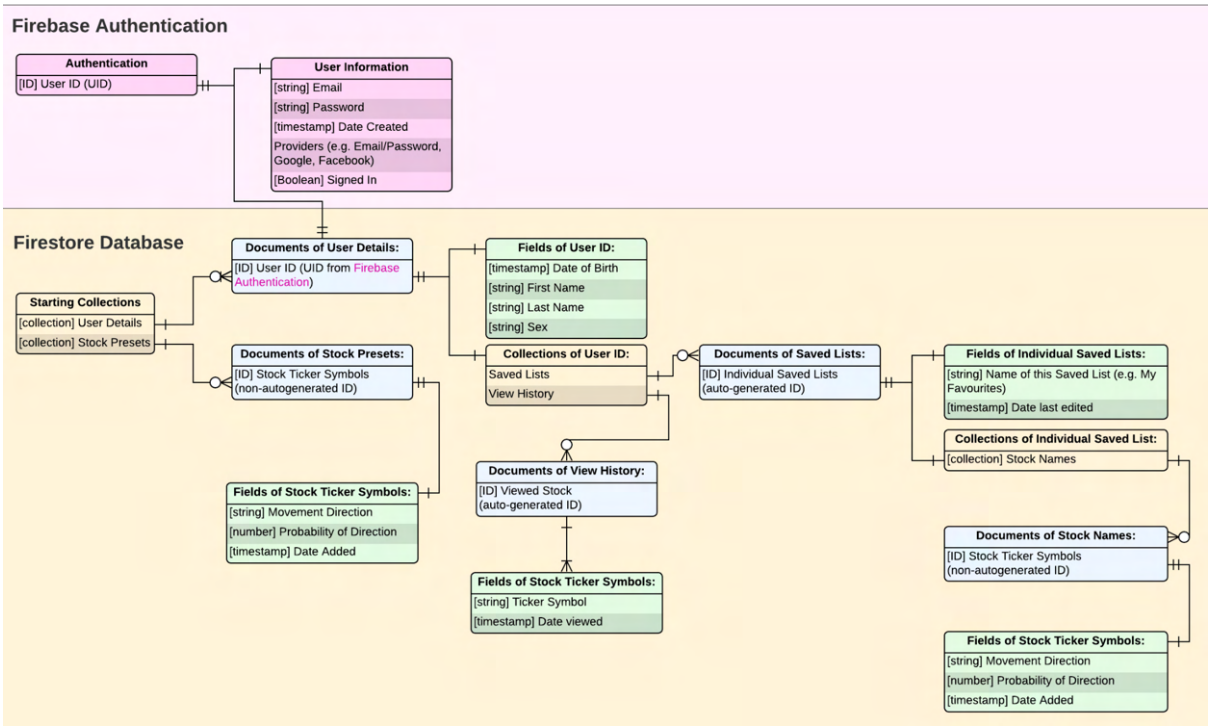


Figure 14a - Entity Relationship Diagram for Firebase Authentication and Firestore Database.

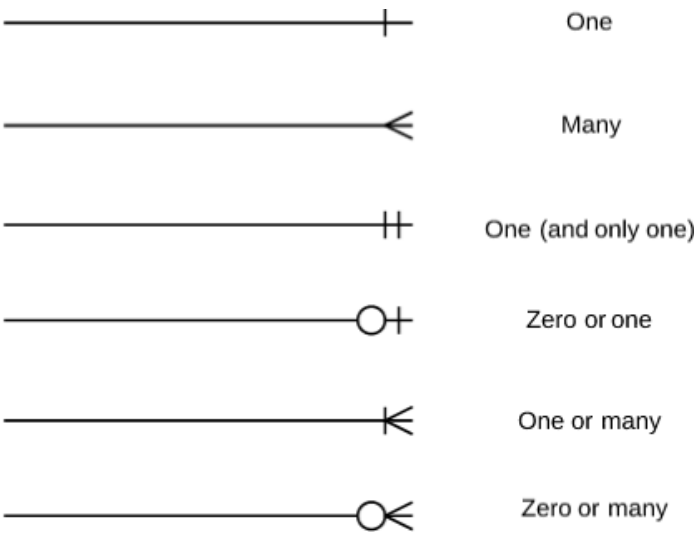


Figure 14b - Key for Entity Relationship Diagram
https://www.lucidchart.com/pages/ER-diagram-symbols-and-meaning

With reference to Figure 14 and document-oriented databases, collections are highlighted as orange, documents are highlighted blue, whereas the fields are highlighted green. When a user creates an account, they will produce a User

Identification Number (UID) through the Firebase Authenticator (highlighted pink). This UID number will be added as a document to the Firestore Collection labelled “User Details”. Each UID document has further information on the user including their names, sex, and date of birth. In addition, the specific user can access their “saved lists”, and “view history”.

The “saved lists” collection may contain various stored lists (documents) that are created by the user, these will be represented using Firestore’s auto-generated IDs. These document objects are then split into a field object, and a collections object. The field objects includes information about when the saved list is last edited, and the name of the saved list (for example, the user may call it “my favourite watchlist”). In the collection object, it offers multiple individual stock ticker symbols in the form of document objects. These stock ticker symbol doesn’t use auto-generated IDs, and instead use the stock symbol as it is, for more efficient query. Inside each stock ticker symbol document, it includes various field objects including: the forecasted movement direction (i.e. Up 10%), its probability (i.e. 65% chance), and the date it’s been added to the saved list.

For the “View History” document object, the ID is auto-generated. This is because, if the ID were the stock ticker symbol (e.g. AAPL) instead, there’s the chance for repetition as a user can access the stock symbol AAPL more than once. Hence, below the “View History” document includes the field objects “Ticker Symbol” and “Date viewed” to distinguish between the different view histories performed by the specific user.

Going back to the starter collection, there’s also the document object labelled “Stock Presets”. This document object is used to save forecasted information of a particular stock that has already been computed through the machine learning algorithm. For example, if User A searched for the stock labelled “AAPL”, and it returned Movement “UP” and Probability: 65%, this information would be saved to the document object with ID: AAPL. User B can then search for “AAPL” and the information would be automatically sent to him, instead of re-computing using the machine learning model, thus saving time and computation. The reason why this ID is not auto-generated (using AAPL instead of py82iK5dred6ioByLOQVo2DbPWi2, for example) is because there can only be one “most-recent” AAPL prediction, instead of multiple AAPL predictions stretched amongst different random generated IDs. Most importantly, in order to guarantee legitimacies with the predictions, there will be a 2nd option (via a button) for the user to use the machine learning algorithm (instead of using preset). Nonetheless, the stock preset documents will delete itself if the “date added” surpasses 30 days compared to the present day to automate forecast integrity.

File structure for earnings transcripts

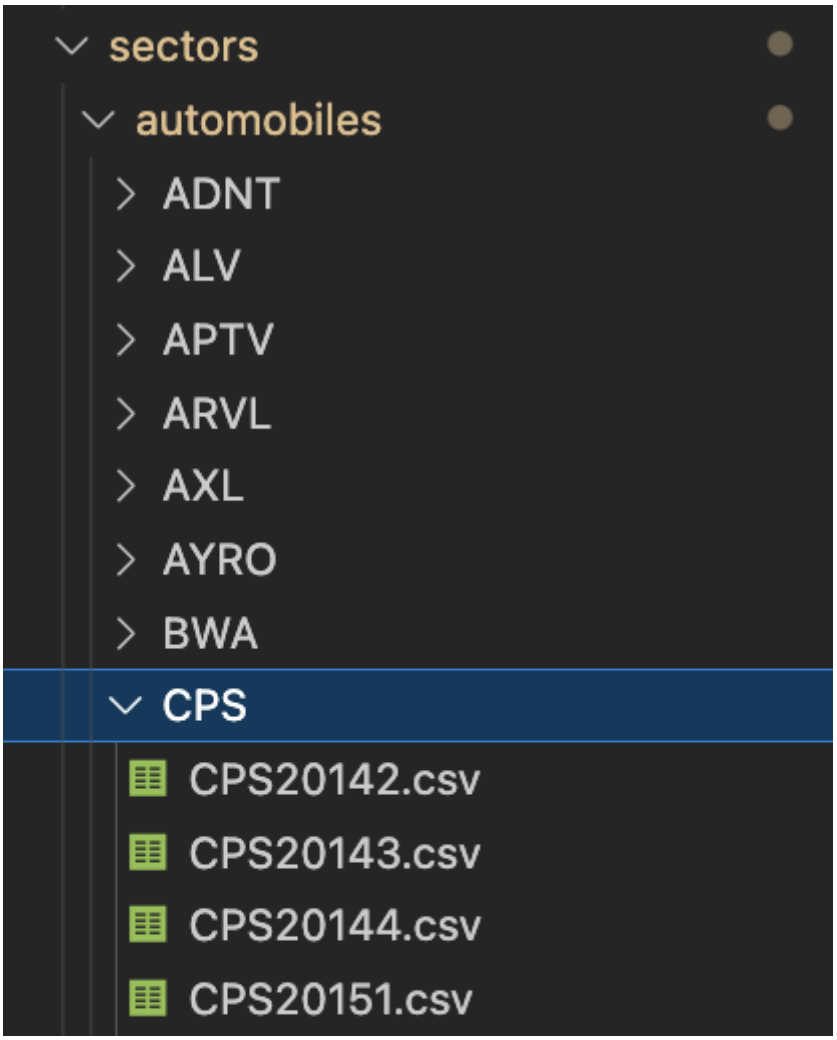


Figure 15

Before analysing and performing feature extraction, we have to prepare the historical earnings transcript data of all the stocks first. I chose to download each earnings transcript as .csv files. The way I organised it can be summed up as `“Sectors/[sector]/[stock]/[sector][year][quarter].csv”` (see figure 15) where [sector] is defined as a value in sectorlist, [stock] as one particular stock within that sector. [year] and [quarter] are defined as the release date of the earnings call.

```
sectorlist = ["automobiles", "banks", "capital-goods", "commercial-services",
"consumer-durables", "consumer-retailing", "consumer-services",
"diversified-financials",
"energy", "food-beverage-tobacco", "healthcare", "household", "insurance", "materials",
"media",
"pharmaceuticals-biotech", "real-estate", "retail", "semiconductors", "software",
"tech", "telecom", "transportation", "utilities"]
```

Figure 16

Figure 16 shows what sectors I will include under my ./sectors directory.

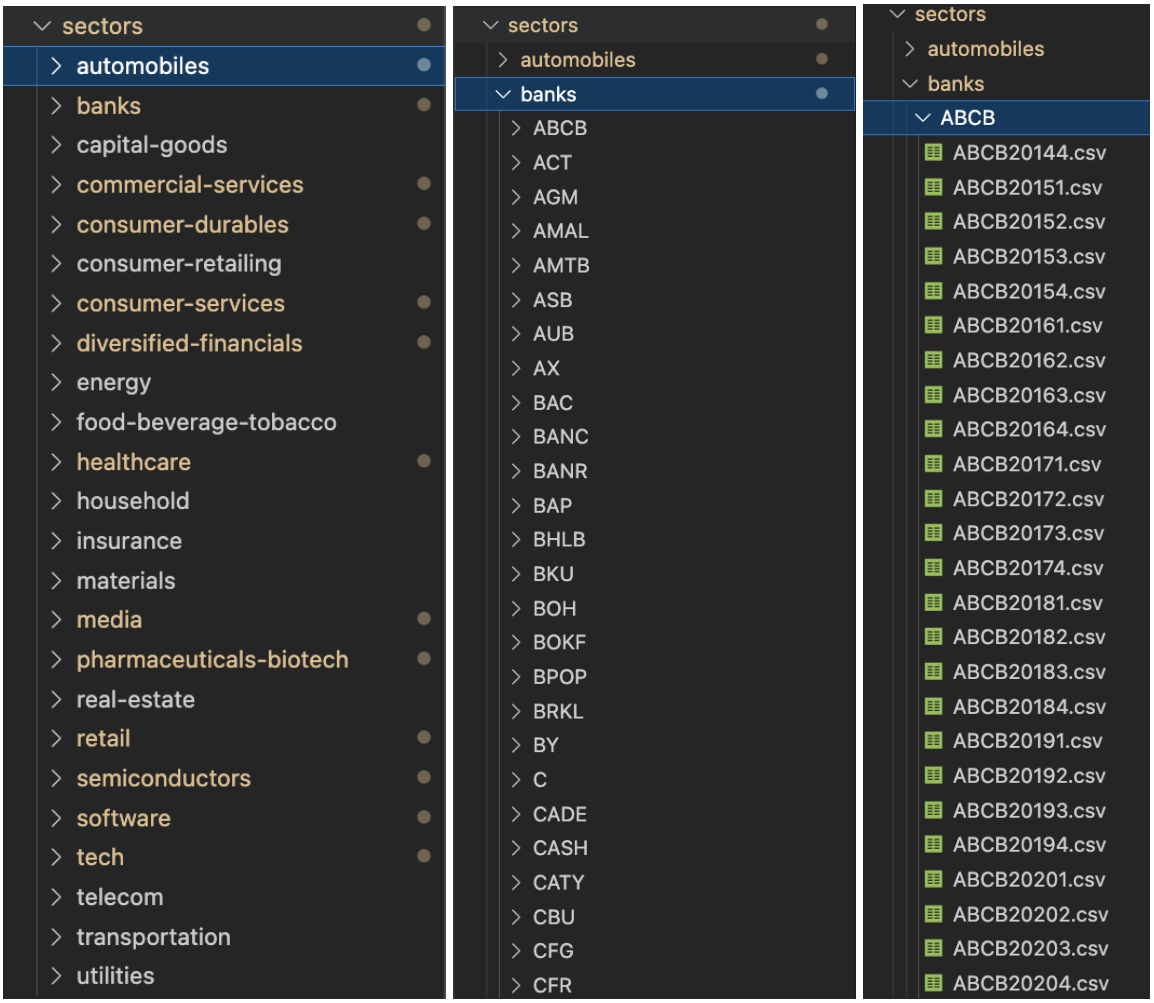


Figure 17a, 17b, 17c

Figure 17 shows what falls under sectors, particular stocks, and earnings transcripts. Each earnings transcript is identified by its year and quarter so ABCB20153.csv indicates the earnings transcript is from 2015, quarter 3.

Summary of the features extracted from each transcript

To train the machine learning algorithm, it requires input data, which are obtained by processing earnings transcripts using various techniques described below. These techniques extract different "features" from the earnings transcripts, and the resulting features are stored in a separate file named earnings_transcript.csv for each transcript.

Each earnings transcript.csv file holds the following information and features (note that the number counters indicate the row index of the .csv file):

Meta Data:

0. Year of transcript release

1. Quarter of transcript release
2. Date of transcript release
3. Earnings Transcript contents

Independent Variables (inputs of the machine learning model)

4. EPS surprise value

The "EPS surprise value" indicates how much the EPS number released by a company exceeds or falls short of the predicted value by a financial analyst. This information can be used to determine whether the company has exceeded or missed the forecast.

Features extracted from Pre-release/Safe Harbour materials

5. Whole pre-release's net sentiment index
6. Whole pre-release's positive sentiment index
7. Whole pre-release's negative sentiment index
8. Whole pre-release's average text complexity index

Numbers 5 to 8 only analyses the contents in pre-releases. The hypothesis is that if there is a huge difference between the pre-release and Q&A section, it may be a signal of deteriorating future business performances, thus potentially affecting the stock price.

The "net sentiment index" measures the average sentiment of all pre-release sentences. To derive the positive sentiment index, the number of positive-sentiment sentences is divided by the total number of sentences. A similar approach is used for the negative sentiment index but with negative sentences.

"Average text complexity index" refers to the average text complexity for the pre-release materials (i.e. flesch reading ease index). The hypothesis here is that higher text complexity may indicate a lack of clarity, hesitation, or uncertainty in their communication, potentially leading to a negative impact on the stock price due to uncertain future business performance.

9. Specific forward-looking statement's net sentiment index
10. Specific forward-looking statement's positive sentiment index
11. Specific forward-looking statement's negative sentiment index
12. Specific forward-looking statements average text complexity index

13. Non-Specific Forward-looking statement's net sentiment index

14. Non-Specific Forward-looking statement's positive sentiment index
15. Non-Specific Forward-looking statement's negative sentiment index
16. Non-Specific Forward-looking statement's average text complexity index

17. Not Foward-looking statement's net sentiment index
18. Not Foward-looking statement's positive sentiment index
19. Not Foward-looking statement's negative sentiment index
20. Not Foward looking statement's average text complexity index

21: Specific forward-looking statement index (calculated as the number of sentences classed as specific forward-looking, over the total number of sentences)
22: Non-Specific forward-looking statement index (calculated as the number of sentences classed as non-specific forward-looking, over the total number of sentences)

Specific forward-looking statements refer to statements that involve a precise estimation of future events (e.g. expected revenue).

Non-specific forward-looking statements, on the other hand, are more general in nature and do not involve specific figures or estimates. They may include statements regarding future plans or strategies, but without committing to a precise outcome.

Not forward-looking statements are statements that do not estimate future events or projections, but instead, they describe the current state of affairs or historical data.

Numbers 9 to 22 essentially tries to find any relationship between sentiment and semantical differences between these three forward-looking classifications and tests whether they have an effect of future stock performances.

Features extracted from the Questions & Answers:

"Questions" are those from analysts
"Answers" are replies from the management

23. Whole Q&A's net sentiment index
24. Whole Q&A's positive sentiment index
25. Whole Q&A's negative sentiment index
26. Whole Q&A's average text complexity index

27. All Question's net sentiment index
28. All Question's positive sentiment index
29. All Question's negative sentiment index

30. All Question’s average text complexity index

Combining all the management replies:

31. All Reply’s net sentiment index
32. All Reply’s positive sentiment index
33. All Reply’s negative sentiment index
34 .All Reply’s average text complexity index

35. Specific forward-looking statement’s net sentiment index
36. Specific forward-looking statement’s positive sentiment index
37. Specific forward-looking statement’s negative sentiment index

38. Non-Specific Forward-looking statement’s net sentiment index
39. Non-Specific Forward-looking statement’s positive sentiment index
40. Non-Specific Forward-looking statement’s negative sentiment index

41. Not Forward-looking statement’s net sentiment index
42. Not Forward-looking statement’s positive sentiment index
43. Not Forward-looking statement’s negative sentiment index

In the Q&A section, analysts asks questions, and management replies (an answer). Hence, we can compare the difference between the two parties and their general sentiment and semantical state, potentially giving a hint of future business performances.

Analysing specific words:

These features only look at sentences that contain the required word.

44: "margin" - average sentiment (i.e. what’s the average sentiment for sentences that mentioned the word “margin”)
45: "margin" - positive sentiment index
46: "margin" - negative sentiment index
47: "cost" - average sentiment
48: "cost" - positive sentiment index
49: "cost" - negative sentiment index
50: "revenue" - average sentiment
51: "revenue" - positive sentiment index
52: "revenue" - negative sentiment index
53: "earnings/EBIDTA" - average sentiment
54: "earnings/EBIDTA" - positive sentiment index
55: "earnings/EBIDTA" - negative sentiment index

56: "growth" - average sentiment
57: "growth" - positive sentiment index
58: "growth" - negative sentiment index
59: "leverage/debt" - average sentiment
60: "leverage/debt" - positive sentiment index
61: "leverage/debt" - negative sentiment index
62: "industry/sector" – average sentiment
63: "industry/sector" – positive sentiment index
64: "industry/sector" – negative sentiment index
65: "operation" - average sentiment
66: "operation" - positive sentiment index
67: "operation" - negative sentiment index
68: "cashflow" - average sentiment
69: "cashflow" - positive sentiment index
70: "cashflow" - negative sentiment index
71: "dividend/share buyback" - average sentiment
72: "dividend/share buyback" - positive sentiment index
73: "dividend/share buyback" - negative sentiment index

For numbers 44 to 73, it finds all sentences where a particular word appears, it then checks the sentiment value for this sentence. For example, if the sentences associated with the word “earnings” is positive, this may imply they are achieving higher earnings in the future, hence potentially bullish for the stock price.

Finding text similarity between transcripts

74: Text similarity between the current and the 2nd most recent earnings transcript
75: Text similarity between the current and the 3rd most recent earnings transcript
76: Text similarity between the current and the 4th most recent earnings transcript

77: Text similarity between the current and the 2nd most recent earnings transcript (only looking at the pre-release materials)
78: Text similarity between the current and the 3rd most recent earnings transcript (only looking at the pre-release materials)
79: Text similarity between the current and the 4th most recent earnings transcript (only looking at the pre-release materials)

80: Text similarity between the current and the 2nd most recent earnings transcript (only looking at the management replies)
81: Text similarity between the current and the 3rd most recent earnings transcript (only looking at the management replies)

82: Text similarity between the current and the 4th most recent earnings transcript (only looking at the management replies)

83: Text similarity between the current and the 2nd most recent earnings transcript (only looking at the analyst questions)
84: Text similarity between the current and the 3rd most recent earnings transcript (only looking at the analyst questions)
85: Text similarity between the current and the 4th most recent earnings transcript (only looking at the analyst questions)

The hypothesis here is that text dissimilarity may imply operational changes in the business. For example, if there is a sudden drop in the frequency of the term 'iPad' and a corresponding increase in mentions of 'Macbooks' by Apple during the second quarter of 2022, it is reasonable to infer that they may have reoriented their attention towards Macbooks, which may have an impact on the stock price.

Dependent Variable (output of the machine learning model):

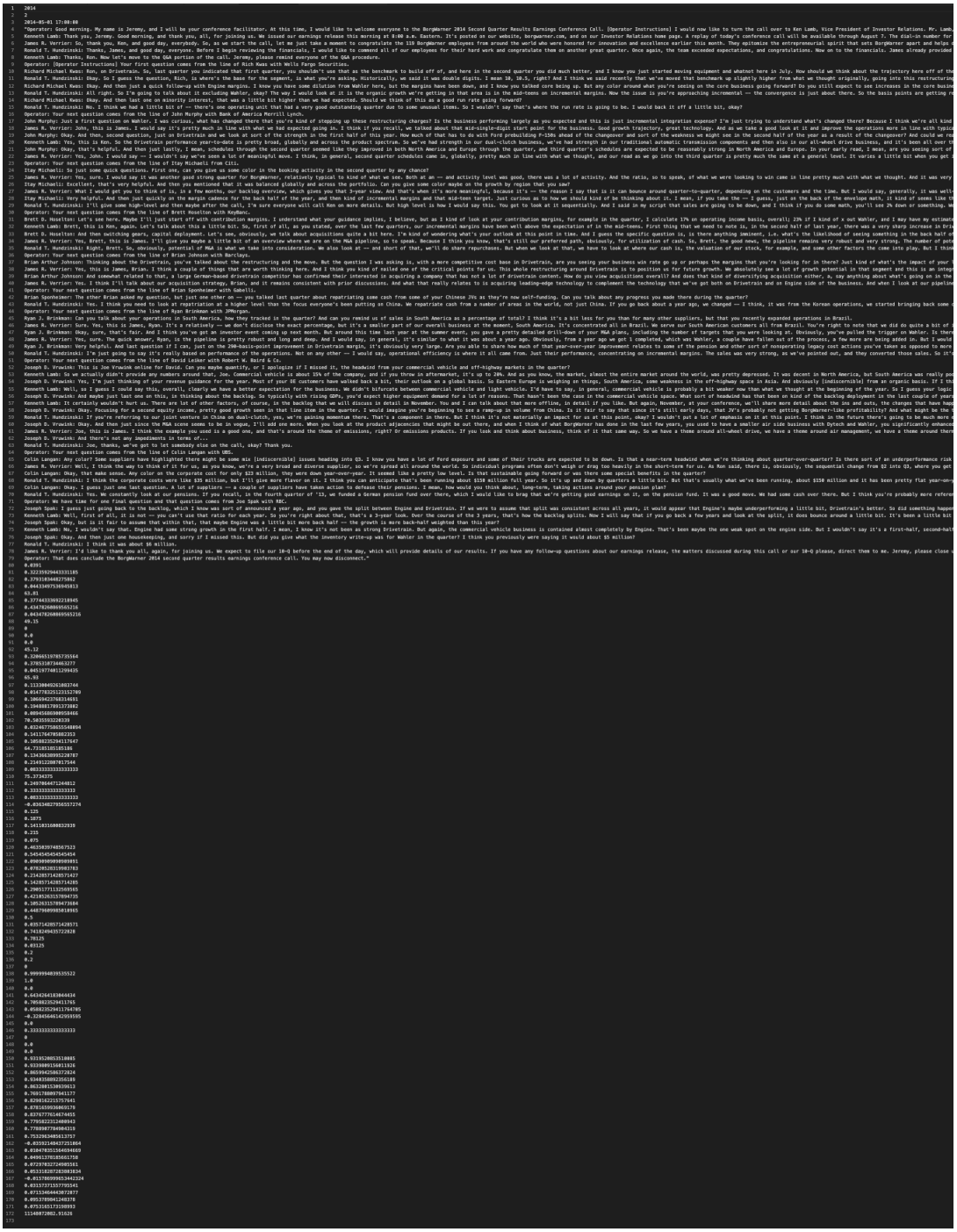
The dependent variable is what I want to predict (i.e. stock returns after the earnings call). Day 0 represents the day the earnings call is broadcasted. The reason I have multiple dependent variables is to see which time series is more accurate.

86: Stock price difference between Day 0 and Day 10
87: Stock price difference between Day 0 and Day 30
88: Stock price difference between Day 0 and Day 50
89: Stock price difference between Day 0 and Day 70
90: Stock price difference between Day 0 and Day 90
91: Stock price difference between Day 1 and Day 10
92: Stock price difference between Day 1 and Day 30
93: Stock price difference between Day 1 and Day 50
94: Stock price difference between Day 1 and Day 70
95: Stock price difference between Day 1 and Day 90

Market Cap:

I included market cap so I can use it as a filter, for example, create a model that only looks at small-cap stocks, or one that only looks at large-cap etc.

96: Market Cap



Sample of what a .csv file contains

Aho-Corasick String Searching Data Structure and Algorithm

In different parts of my programme, I would need to search for specific words in a body of text. I chose to use an Aho-Corasick algorithm, a finite state machine algorithm, since it's designed to work with large sets of words. As geeksforgeeks.org mentions, it holds a time complexity of **O(n + m + z)** where **n** is the length of text, **m**

is the total number of characters in all words, and **z** is the number of total occurrences of that word in the text.

The Aho-Corasick Algorithm I implement will be guided by the 3 websites below:
https://www.geeksforgeeks.org/aho-corasick-algorithm-pattern-searching/
https://en.wikipedia.org/wiki/Aho%E2%80%93Corasick_algorithm
https://cp-algorithms.com/string/aho_corasick.html#construction-of-an-automaton

The implementation I made consists of two classes, TrieNode and AhoCorasick.

TrieNode

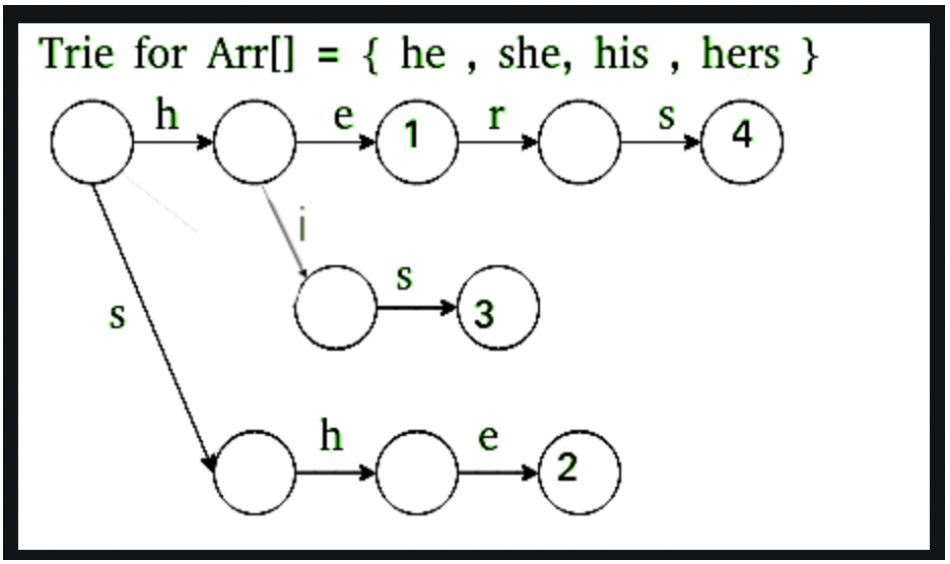


Figure 18 - Trie Node Implementation (from geeksforgeeks)

The TrieNode class represents a node in the Trie data structure, which is used to store a set of words by dividing their characters. Figure 18 shows how the word “he” is broken down into the “hers” branch until the number “1”, which represents it’s the first word of the Arr[ ].

Within the TrieNode class, there are the following attributes:

- **Children:** the keys of the dictionary that maps characters to child nodes (i.e. edges of the tree).
- **is_word:** the boolean value which indicates whether this node represents the end of a word (ie. number “1” in figure 18 for the word “he”)
- **fail:** used to point to the “fail” node of the algorithm
- **word:** the word represented when **is_word** is True

AhoCorasick

The AhoCorasick class has the following methods:

- **__init__(self, words)**: constructs the instance of the class and builds the trie and Aho-Corasick automaton using a given set of words.
- **build_trie(self, phrase)**: builds the trie from a given set of words by using the TrieNode class.
- **build_ac_automata(self)**: builds the finite state machine Aho-Corasick automaton using the trie constructed by build_trie. This is done by implementing a queue with the children of the roots. Then, it breadth first traverse through the trie structure. When the word is identified through the **fail** attribute, the is_word is set to True.
- **remove_words(self, text)**: A class that can be used to remove all occurrences of the said word after it's been identified, essentially, it deletes all instances of a particular word from a phrase.

UML Diagram

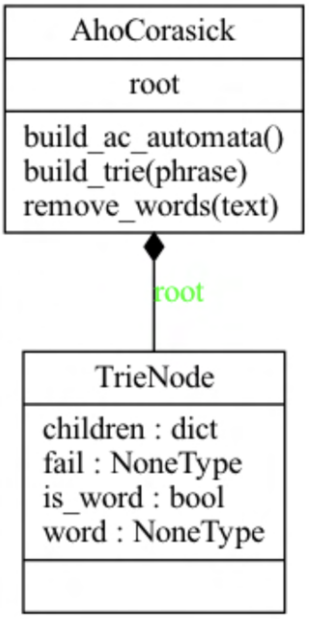


Figure 19 - UML Diagram for TrieNode and AhoCorasick Classes

Machine Learning Datastructures

I will be using the python library Pickle to serialise the machine learning model into a .pkl file format. This means it converts objects into a byte stream and stored within a "trained models file directory". This byte stream can then be loaded back into memory and used as the original object, which are used for predicting the data. Serialising the trained model using Pickle also ensures its reproducibility as the

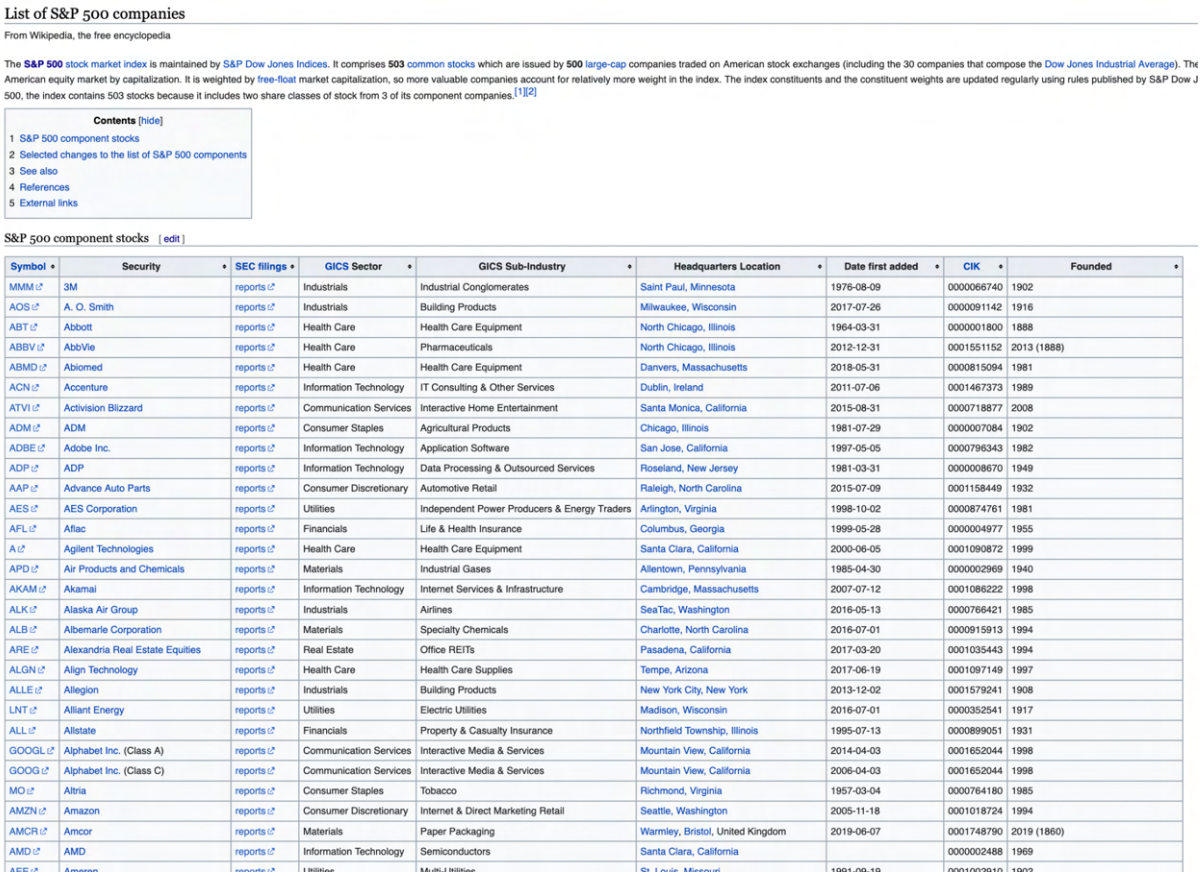
model being deserialised is in the exact same state as it was trained and serialised, this is very important for a stock prediction tool.

I will also create multiple machine learning models tailored to various industries, each with its own corresponding pickle file. For instance, a model specific to the banking sector, another model for the technology sector, and another for the consumer durables sector etc.

Algorithms

Web Scraping Stock's Earnings Transcripts:

Before conducting feature extraction on the earnings transcripts of different stocks, it is necessary to identify the list of stocks first. I will be analysing the entire set of stocks from the S&P 500 index, and the Russell 2000.



List of S&P 500 companies

From Wikipedia, the free encyclopedia

The **S&P 500** stock market index is maintained by S&P Dow Jones Indices. It comprises **503** common stocks which are issued by **500** large-cap companies traded on American stock exchanges (including the 30 companies that compose the Dow Jones Industrial Average). The American equity market by capitalization. It is weighted by free-float market capitalization, so more valuable companies account for relatively more weight in the index. The index constituents and the constituent weights are updated regularly using rules published by S&P Dow J 500, the index contains 503 stocks because it includes two share classes of stock from 3 of its component companies.[1][2]

Contents [hide]

1 S&P 500 component stocks
2 Selected changes to the list of S&P 500 components
3 See also
4 References
5 External links

S&P 500 component stocks [edit]

Symbol	Security	SEC filings	GICS Sector	GICS Sub-Industry	Headquarters Location	Date first added	CIK	Founded
MMM	3M	reports	Industrials	Industrial Conglomerates	Saint Paul, Minnesota	1976-08-09	0000066740	1902
AOS	A. O. Smith	reports	Industrials	Building Products	Milwaukee, Wisconsin	2017-07-26	0000091142	1916
ABT	Abbott	reports	Health Care	Health Care Equipment	North Chicago, Illinois	1964-03-31	0000001800	1888
ABBV	AbbVie	reports	Health Care	Pharmaceuticals	North Chicago, Illinois	2012-12-31	0001551152	2013 (1888)
ABMD	Abiomed	reports	Health Care	Health Care Equipment	Danvers, Massachusetts	2018-05-31	0000815094	1981
ACN	Accenture	reports	Information Technology	IT Consulting & Other Services	Dublin, Ireland	2011-07-06	0001467373	1989
ATVI	Activision Blizzard	reports	Communication Services	Interactive Home Entertainment	Santa Monica, California	2015-08-31	0000718877	2008
ADM	ADM	reports	Consumer Staples	Agricultural Products	Chicago, Illinois	1981-07-29	0000007084	1902
ADBE	Adobe Inc.	reports	Information Technology	Application Software	San Jose, California	1997-05-05	0000796343	1982
ADP	ADP	reports	Information Technology	Data Processing & Outsourced Services	Roseland, New Jersey	1981-03-31	0000008670	1949
AAP	Advance Auto Parts	reports	Consumer Discretionary	Automotive Retail	Raleigh, North Carolina	2015-07-09	0001158449	1932
AES	AES Corporation	reports	Utilities	Independent Power Producers & Energy Traders	Arlington, Virginia	1998-10-02	0000874761	1981
AFL	Aflac	reports	Financials	Life & Health Insurance	Columbus, Georgia	1999-05-28	0000004977	1955
A	Agilent Technologies	reports	Health Care	Health Care Equipment	Santa Clara, California	2000-06-05	0001090872	1999
APD	Air Products and Chemicals	reports	Materials	Industrial Gases	Allentown, Pennsylvania	1985-04-30	0000002969	1940
AKAM	Akamai	reports	Information Technology	Internet Services & Infrastructure	Cambridge, Massachusetts	2007-07-12	0001086222	1998
ALK	Alaska Air Group	reports	Industrials	Airlines	SeaTac, Washington	2016-05-13	0000766421	1985
ALB	Albemarle Corporation	reports	Materials	Specialty Chemicals	Charlotte, North Carolina	2016-07-01	0000915913	1994
ARE	Alexandria Real Estate Equities	reports	Real Estate	Office REITs	Pasadena, California	2017-03-20	0001035443	1994
ALGN	Align Technology	reports	Health Care	Health Care Supplies	Tempe, Arizona	2017-06-19	0001097149	1997
ALLE	Allegion	reports	Industrials	Building Products	New York City, New York	2013-12-02	0001579241	1908
LNT	Alliant Energy	reports	Utilities	Electric Utilities	Madison, Wisconsin	2016-07-01	0000352541	1917
ALL	Allstate	reports	Financials	Property & Casualty Insurance	Northfield Township, Illinois	1995-07-13	0000899051	1931
GOOGL	Alphabet Inc. (Class A)	reports	Communication Services	Interactive Media & Services	Mountain View, California	2014-04-03	0001652044	1998
GOOG	Alphabet Inc. (Class C)	reports	Communication Services	Interactive Media & Services	Mountain View, California	2006-04-03	0001652044	1998
MO	Altria	reports	Consumer Staples	Tobacco	Richmond, Virginia	1957-03-04	0000764180	1985
AMZN	Amazon	reports	Consumer Discretionary	Internet & Direct Marketing Retail	Seattle, Washington	2005-11-18	0001018724	1994
AMCR	Amcor	reports	Materials	Paper Packaging	Warmley, Bristol, United Kingdom	2019-06-07	0001748790	2019 (1860)
AMD	AMD	reports	Information Technology	Semiconductors	Santa Clara, California		0000002488	1969

Figure 15 - A list of S&P 500 companies
(https://en.wikipedia.org/wiki/List_of_S%26P_500_companies)

There are 505 stocks in the S&P 500 index and can be accessed from the wikipedia page as illustrated in Figure 15.

```python
#Collects S&P500 tickers
spy_list_requests = requests.get('http://en.wikipedia.org/wiki/List_of_S%26P_500_companies')
soup = BeautifulSoup(spy_list_requests.text, 'lxml')
table = soup.find('table', {'class': 'wikitable sortable'})
tickers = []
for row in table.findAll('tr')[1:]:
    ticker = row.findAll('td')[0].text
    tickers.append(ticker)

spy_tickers_list = [s.replace('\n', '') for s in tickers]

print(spy_tickers_list)
```

```
['MMM', 'AOS', 'ABT', 'ABBV', 'ABMD', 'ACN', 'ATVI', 'ADM', 'ADBE', 'ADP', 'AAP', 'AES', 'AFL', 'A'
'ALB', 'ARE', 'ALGN', 'ALLE', 'LNT', 'ALL', 'GOOGL', 'GOOG', 'MO', 'AMZN', 'AMCR', 'AMD', 'AEE', 'A
'AMT', 'AWK', 'AMP', 'ABC', 'AME', 'AMGN', 'APH', 'ADI', 'ANSS', 'AON', 'APA', 'AAPL', 'AMAT', 'APT
'T', 'ATO', 'ADSK', 'AZO', 'AVB', 'AVY', 'BKR', 'BALL', 'BAC', 'BBWI', 'BAX', 'BDX', 'WRB', 'BRK.B'
```

Figure 16 - Code for getting a list of S&P 500 companies' ticker symbols

With reference to Figure 16, the Python libraries Beautiful Soup and Requests are used for web scrapping the list of S&P 500 companies.

For the Russell 2000 index, there exists a downloadable .csv file from the website https://stockmarketmba.com/stocksintherussell2000.php. This means I can use the Pandas library from Python to obtain the stock ticker symbols and append it into a list.

```python
mylist1 = [1,2,3,4]
mylist2 = [2,3,4,5,6,7,8]
final_list = mylist1+mylist2
final_list = set(final_list)
print(final_list)
```

Figure 17a - Checking Duplicates

```
{1, 2, 3, 4, 5, 6, 7, 8}
```

Figure 17b - Output

After getting these two set of stock lists, I will then need to check for duplicates. This could be done by converting it to a *set* which automatically deletes duplicates.

Now that I've gotten a list of stocks (for example, named **final_ticker_list**), I could use a for loop to loop around each individual ticker symbol and obtain the earnings transcript of that particular stock by calling their relevant feature extraction functions. One of these functions is to web scrape the relevant earnings call transcript of that particular stock. This would be done using the website https://roic.ai/transcripts/ followed by it's ticker symbol, year and quarter. For

example, if I want to look at the earnings transcript of Apple for 2022 quarter 3, and 2021 quarter 4 respectively:
https://roic.ai/transcripts/AAPL?y=2022&q=3
https://roic.ai/transcripts/AAPL?y=2021&q=4
Afterwards, Requests and BeautifulSoup would be used to web scrape the relevant earnings call transcript contents for further feature extraction.

Splitting Earnings Transcript [split_transcript()]:

Earnings calls begin with the **"Safe Harbour"** statements to discuss quarterly operating results and future outlook. This is presided over by the **"Questions and Answers"** section which involves a Q&A exchange between the company and the public. Safe Harbour as the name suggests is a place for the management to provide prepared remarks without interference by analysts. The hypothesis is that there may be a difference in sentiment and text complexity between the 'Safe Harbour' and 'Questions and Answers' sections, hence potentially hint at the stock's future outlook.

Note that in this NEA doc, I will use "safe habour" and "pre-release" interchangeably.

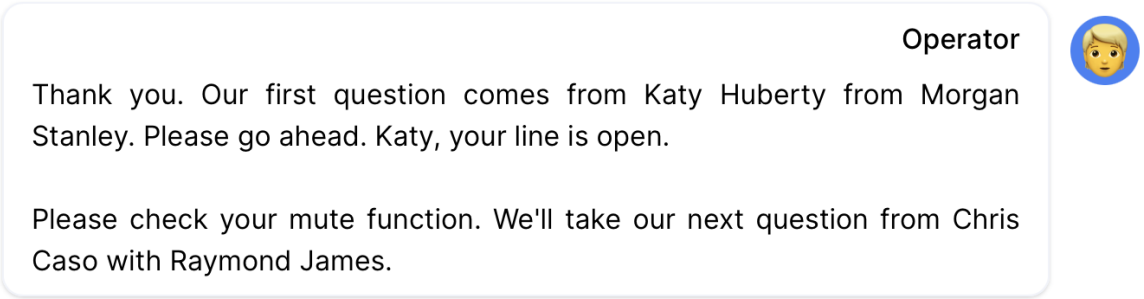


Figure 18 - Point of separation between 'Safe Harbour' and 'Questions and Answers' for Apple

So in order to split the earnings transcript, we can identify various points of separation between the "Safe Harbour" and "Question and Answers" sections as illustrated by Figure 18.

After looking through several dozen earnings calls, the point of separation may potentially be when the words "first question" and "operator" are both in the same paragraph, or if the words "first question" or "go ahead" are in the paragraph, then the programme will notice this paragraph as the "point of separation"

In the actual code, the split_transcript function may work by receiving a string input of the entire earnings transcript. In the earnings transcript string, since each paragraph (or "speech bubbles") are separated by a line break "\n", it could be split using the python .split("\n") method which turns it into a list. The for loop would loop through this new list and checks a particular conditional IF statement for each value (i.e. paragraph) of that list. In each of these paragraphs, the program looks for keywords such as "first question", "operator instructions" or "go ahead"; and if the IF statement is met, two new lists are created. The two new lists then runs through a function called clean_text() which I'll discuss below.

Clean_Text Function [clean_text()]:

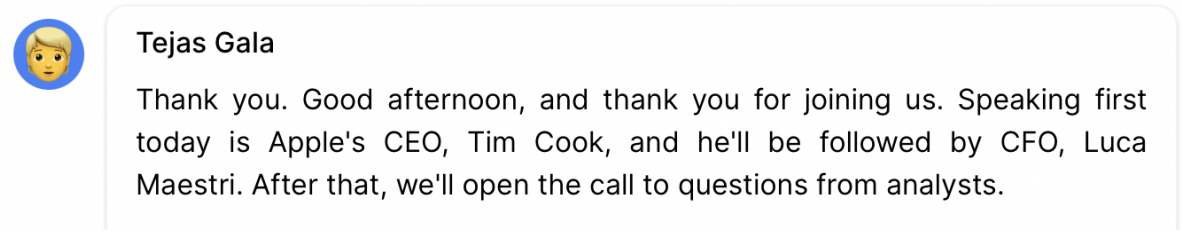


Figure 21 - Sample comment in earnings transcript

Certain functions, such as **split_transcript()**, need an additional clean_text function. This function uses the RegEx library from Python as it offers a wide range of string manipulation methods. The main purpose of this function is to standardise all earnings call inputs with features including:

- Changing all characters to lowercase
- Deletes comments made by the Operator[1] (see figure 18)
- Turns 'end sentence.start' to 'end sentence. start'
- Remove multi-spaced words to a single space
  (e.g. 'double space' to 'double space')
- Deletes all speaker names (e.g. Tejas Gala, Tim Cook with reference to Figure 21) and any mentions of it throughout the entire transcript. It also makes sure embedded words are not deleted, such as not deleting the word 'estimate' even though 'tim' is embedded within, or 'gala' in 'galaxy'.

```
def clean_text(transcript): #transcript is in format ["a", "b", "c"]
   transcript = '\n'.join(transcript)
   transcript = transcript.lower()
```

[1] The operator facilitates the earnings call, by introducing the speakers and keeping track of time, all non information that I would like to leave out.

```
    # turns 'end sentence.start' to 'end sentence. start' with space in between
    transcript = re.sub(r'\.([a-zA-Z])', r'. \1', transcript)
    transcript = re.sub(r'\?([a-zA-Z])', r'. \1', transcript)
    transcript = re.sub(r'\!([a-zA-Z])', r'. \1', transcript)
    # replace q1,2,3,4 with q
    transcript = re.sub("q[1-4]", "q", transcript)
    # replace 20xx with 2000
    transcript = re.sub("20[0-2][0-9]", "2000", transcript)
    # deletes all commments that begins with 'Operator: ...'
    temp = transcript.split('\n') #TURNS BACK TO LIST
    i = 0
    r = 0
    try:
        while (i != len(temp) - 1) and r < 80:
            r+=1
            if 'operator:' in temp[i]:
                del temp[i]
            i += 1
    except:
        temp = temp
    temp = '\n'.join(temp)
    temp = re.sub(r'\.([a-zA-Z])', r'. \1', temp)
    temp = re.sub(r'\?([a-zA-Z])', r'. \1', temp)
    temp = re.sub(r'\!([a-zA-Z])', r'. \1', temp)
    temp = temp.split('\n') #TURNS BACK TO LIST

    #deletes speaker name:
    arr_speaker_name = []
    for i in range(0, len(temp)):
        a = temp[i].split()[0:5]  # gets the first 5 words
        for j in range(0, len(a)):
            if ':' in a[j]:
                k = list(a[j])
                del k[-1]
                p = (a[0:j])
                for l in p:
                    if l != '' and l not in arr_speaker_name:
                        arr_speaker_name.append(l)
                if ''.join(k) != '' and ''.join(k) not in arr_speaker_name:
                    arr_speaker_name.append(''.join(k))
    temp = '\n'.join(temp)

    temp = re.sub(':', ' ', temp)
    for i in arr_speaker_name:  # removes all speaker names from transcript
        try:
            temp = re.sub('\s+', ' ', temp)  # replace multiple space to single space
            temp = re.sub(r'\s'+i+r'\s', ' ', temp) #makes sure embedded words aren't
deleted, such as 'tim' in estimate
            temp = re.sub(r'\s' + i+r'\.', ' ', temp)
            temp = re.sub(r'\s' + i+r'\?', ' ', temp)
            temp = re.sub(r'\s' + i+r'\,', ' ', temp)
            temp = re.sub(r'\s' + i+r'\'', ' ', temp)
        except:
            temp = re.sub('\s+', ' ', temp)
            continue
    temp = re.sub('\s+', ' ', temp)  # replace multiple space to single space
```

```
    return temp #returns a string
```

Code for Clean_Text Function

Is the speaker an Analyst or a part of the Management?

In the Q&A section, analysts ask a question and management provides them with a reply, hence, we can classify each Q&A speech bubble between "**Questions**" and "**Answers**". We can then aggregate each of these classes and perform separate semantical analyses (e.g. find the average sentiment value for analyst questions).

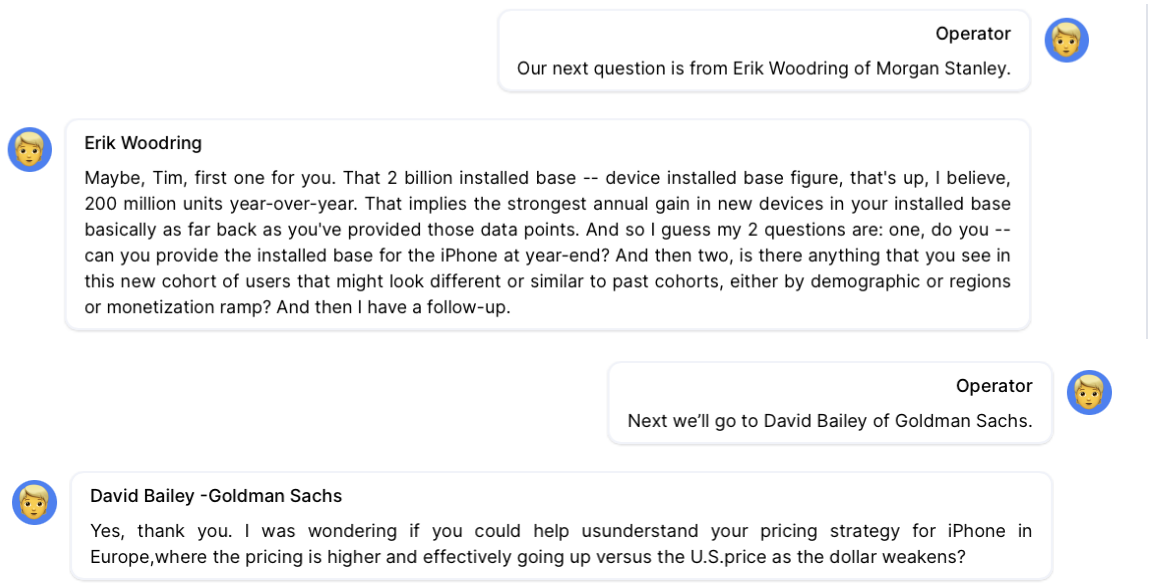


Figure 22a - Snapshot of Apple 2023 Q1 Earnings transcript

Figure 22b - Snapshot of Apple 2007 Q4 Earnings transcript

The way it determines whether the speaker is an analyst or a part of the management is based on what the operator says. With reference to Figure 22a, the operator speech bubble includes the speaker "Erik Woodring", and the words "next question". Hence in this instance, the conditions that the speaker is an analyst is:

```
If said by operator
AND
If a speaker's name is in the speech bubble
AND
If "next question" is in the speech bubble
```

However this is isn't true in every case, for example, in Figure 22b, the conditions will be:

```
If said by operator
AND
If speaker name is in the speech bubble
```

Matching Algorithm between EPS date and Earnings Transcript date

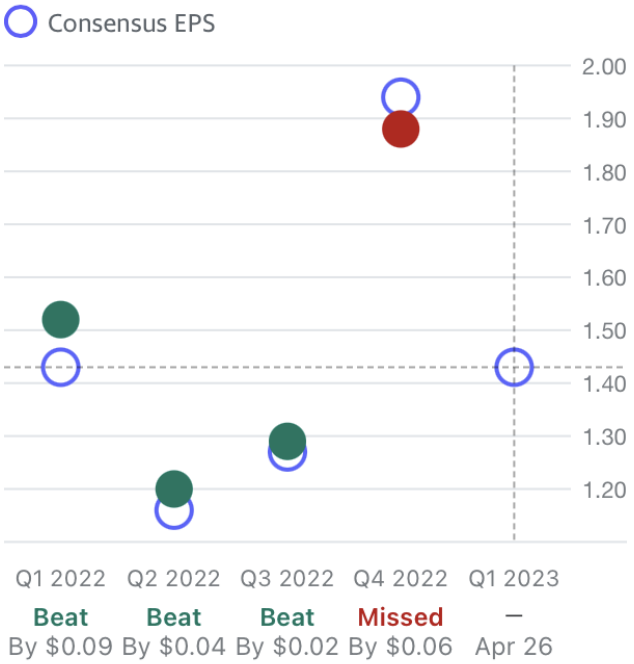


Figure 23 - Apple's Earnings Estimates

Earnings per share (EPS) is a metric used to indicate what portion of a company's profit is allocated to each outstanding share of its common stock. Analysts make estimates of the anticipated EPS value prior to the company's release of its earnings results. Hence, analysts can compare the estimated and actual EPS values and determine whether the company exceeded or fell short of expectations.

Figure 23 is an example of Apple's consensus actual vs predicted EPS. As we can see, Apple beat expectations in Q1 Q2 and Q3, but fell short in Q4. In my programme, I used a percentage figure to represent the magnitude of beat/miss instead of subtracting it to normalise different EPS scales of different companies.

The programme will accept two arguments, 'stock' and 'sector' to get the directory location of the particular stock. The script will then loop through every earnings transcript.csv file within this sub-directory and get the **transcript_date** (represents the date of which the earnings transcript is released). The script will then compare this **transcript_date** with a corresponding **EPS_date** (represents the date the EPS is released).

The main purpose of this algorithm is to match the EPS date **(EPS_date)** with the earnings transcript date **(transcript_date).** This is so the EPS value can be appended

to the correct file location. For example, if the EPS data is for 2017 Q1, we want to append this value to the file that holds th 2017 Q1 earnings transcript.

By using the yfinance API, we can get a historical dataframe of the “EPS surprise value”, sorted reverse-chronologically. However, the earnings transcript.csv files have gaps in between.

```
On an abstracted level, for example, the dates on a yfinance
dataframe may look like this:
2020
2019
2018
2017
2016

However, the dates of the earnings transcript.csv files sorted
chronologically may look like:
2020
2019
2017
2016

In this example, we see that there is no earnings transcript file
for the year 2018.

Similarly, the yfinance dataframe may have gaps too.
```

In short:
1. Basis Case: If the 2 dates matches, the script adds the EPS_value to the .csv file.
2. However, if the dates do not match, the script checks which case it falls under to and increment a loop counter accordingly.

Furthermore, it adds a NaN value to the EPS_value location of that .csv file in case 2, but not case 1.

```
Case 1:
Current transcript_date_list is less than EPS_date_list, hence EPS_date_counter increments by
1 until they match[2]

transcript_date_list     EPS_date_list
```

[2] The EPS_date_list is [2022, 2021, 2020] sorted reverse-chornoologically, thus, you increase the index_counter to get the “previous year”.

```
2020                    2022
2020                    2021
2020                    2020 --> Match!

Case 2:
Current EPS_date_list is less than transcript_date_list, hence transcript_date_counter
increments by 1 until they match[3]

transcript_date_list    EPS_date_list
2022                    2020
2021                    2020
2020                    2020 --> Match!
```

Trace tables with different scenarios:

```
In the case that 2020_Q3 is missing from transcript_date_list:
transcript_date_list = [2021_Q1, 2020_Q4, 2020_Q2, 2020_Q1, 2019_Q4]
EPS_date_list = [2021_Q1, 2020_Q4, 2020_Q3, 2020_Q2, 2020_Q1, 2019_Q4]

transcript_date_list    EPS_date_list
2021_Q1                 2021_Q1  --> Match!
2020_Q4                 2020_Q4  --> Match!
2020_Q2                 2020_Q3  --> case 1, EPS_date_counter decreases, NaN doesn't replace EPS_data
2020_Q2                 2020_Q2  --> Match!

------------------------------------------------------------------------------------------------------
--

In the case that 2020_Q3 is missing from EPS_date_list:
transcript_date_list = [2021_Q1, 2020_Q4, 2020_Q3, 2020_Q2, 2020_Q1, 2019_Q4]
EPS_date_list = [2021_Q1, 2020_Q4, 2020_Q2, 2020_Q1, 2019_Q4]

transcript_date_list    EPS_date_list
2021_Q1                 2021_Q1  --> Match!
2020_Q4                 2020_Q4  --> Match!
2020_Q3                 2020_Q2  --> case 2, NaN added to 2020_Q3 because this EPS_data doesn't exist
2020_Q2                 2020_Q2  --> Match!
2020_Q1                 2020_Q1  --> Match!
2019_Q4                 2019_Q4  --> Match!

------------------------------------------------------------------------------------------------------
--

in the case that 2021 and 2020 are missing in EPS_date_list:
transcript_date_list = [2021_Q1, 2020_Q4, 2020_Q3, 2020_Q2, 2020_Q1, 2019_Q4]
EPS_date_list = [2019_Q4, 2019_Q3, 2019_Q2, 2019_Q1]
```

[3] The EPS_date_list is [2022, 2021, 2020] sorted reverse-chornoologically, thus, you increase the index_counter to get the "previous year".

```
transcript_date_list    EPS_date_list
2021_Q1                 2019_Q4  --> case 2, np.NaN is appended to 2021_Q1.csv
2020_Q4                 2019_Q4  --> case 2, np.NaN is appended to 2020_Q4.csv
2020_Q3                 2019_Q4  --> case 2, np.NaN is appended to 2020_Q3.csv
2020_Q2                 2019_Q4  --> case 2, np.NaN is appended to 2020_Q2.csv
2020_Q1                 2019_Q4  --> case 2, np.NaN is appended to 2020_Q1.csv
2019_Q4                 2019_Q4  --> Match!
```

User Interface:

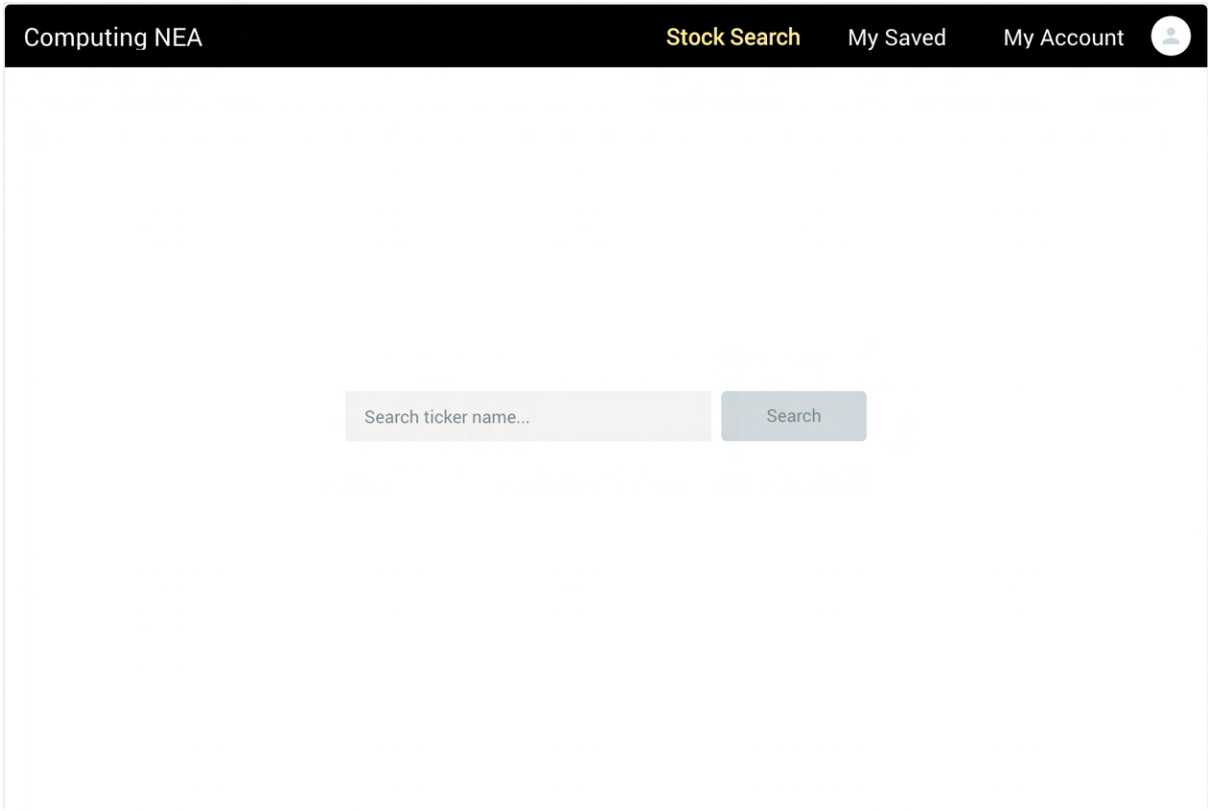


Figure 24 - User interface for the Stock Search Page

As Figure 24 illustrates, the ‘Stock Search’ page consists of a search bar and a ‘search button’. The user will be able to type the ticker name in the search bar and search for that stock. When this page is opened, the ‘Stock Search’ button in the menu will be highlighted with the Hex colour code #FFE898, giving a yellow-orange hue. This is so the user knows what page they are on. The user is able to interact with the menu by clicking on other links such as opening the page “My saved”, or “My Account”.

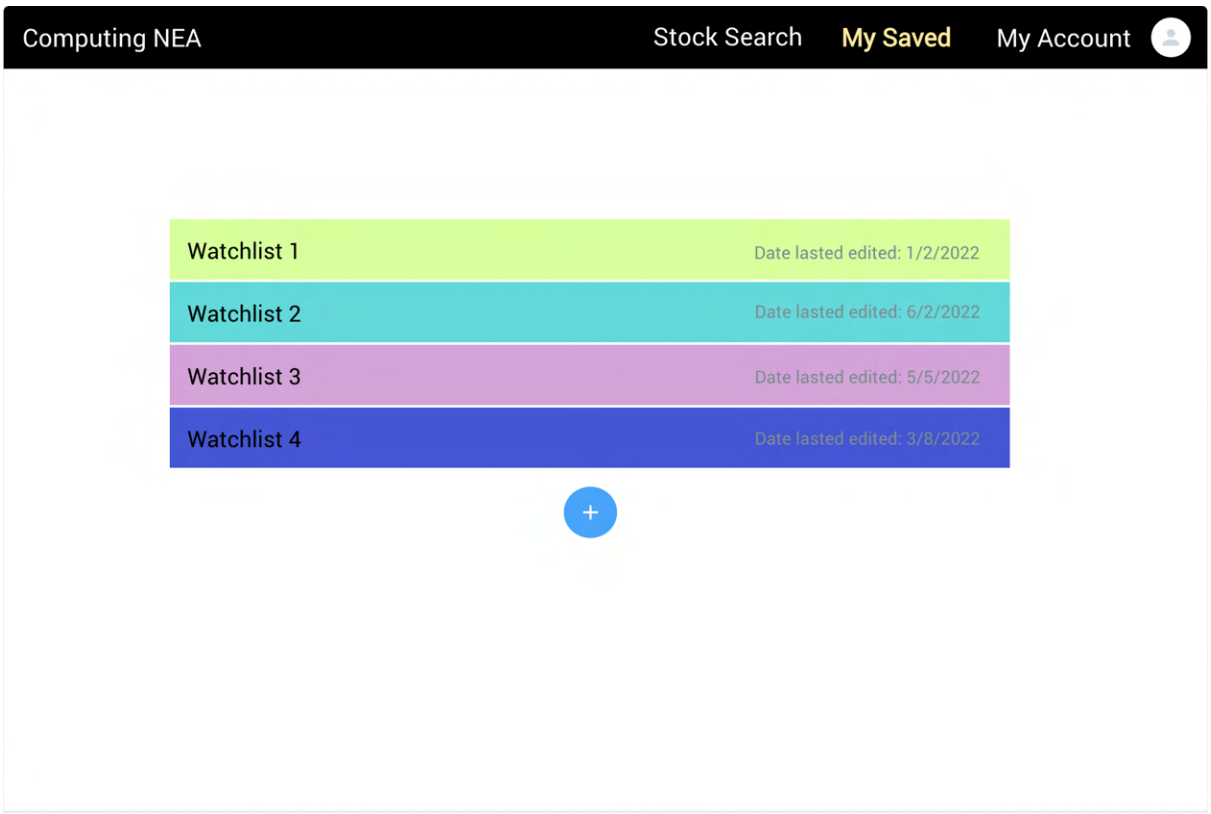


Figure 25 - Able to change colours of watchlist via hex colour thingy?

When the user clicks on the “My Saved” page, like before, the colour will switch to Hex colour code #FFE898. On this page, you can view your existing saved lists and the date you last edited them. You can also add new saved lists using the blue add button (Hex colour #00A6FF), as well as the option to change the colour of each list. For example, with reference to figure 25, Watchlist 1 can be configured with a green colour, Watchlist 2 with a cyan colour etc.

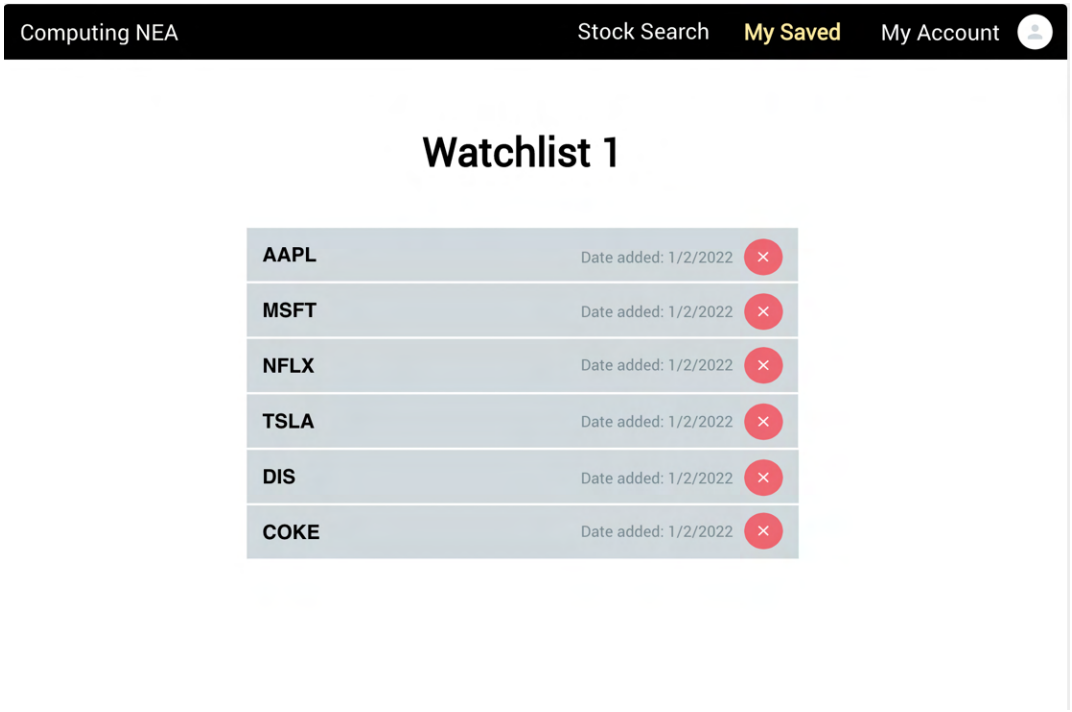


Figure 26 - UI for each saved list

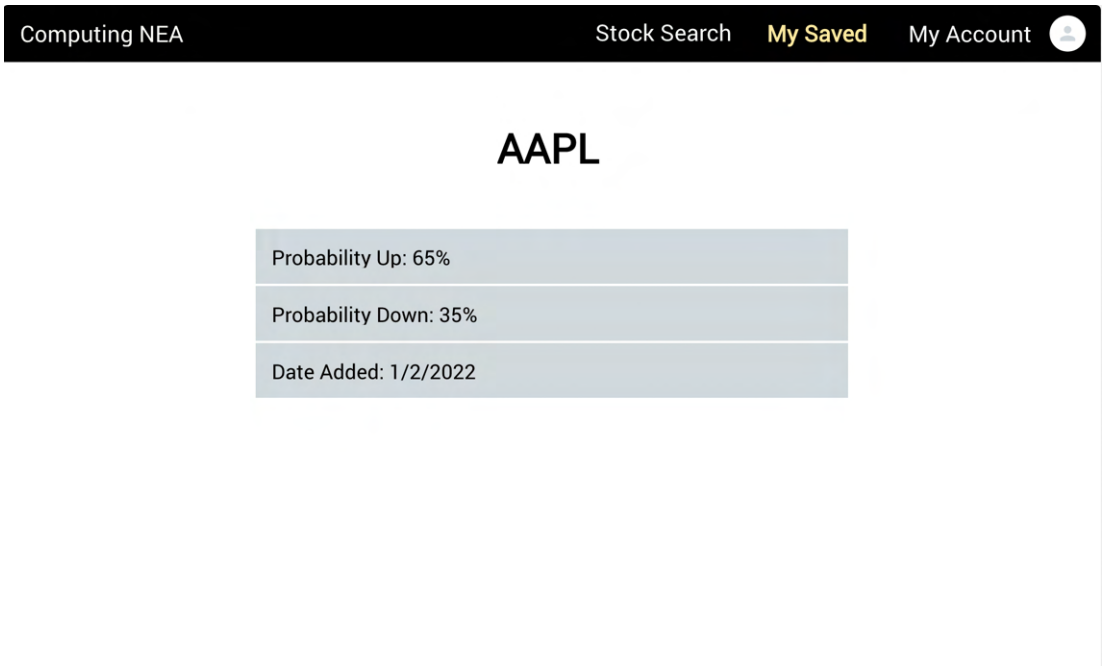


Figure 27 - UI for saved stock information

When you click on each saved list, it takes the user to a new page detailing the set of stocks and the date they were added. It also gives the user the option to delete any stocks in the watchlist by pressing the button with the red cross. From the saved list, the user is able to click on any of the saved stocks and view their saved information as shown in Figure 27.

As an example, the user can search for the ticker name “AAPL” in the “Stock Search” page, which then redirects them to the stock information page on AAPL. On the stock information page, the user can save the ticker “AAPL” into “Watchlist 1”. Afterwards, the user is able to access the entirety of “Watchlist 1” as shown in Figure 26 and delete any saved stock tickers if they want. If the user wants to look at the stock information of “AAPL” that they saved on 1/2/2022, they can click on the “AAPL” name in Figure 26 which redirects them to the page detailed in Figure 27.

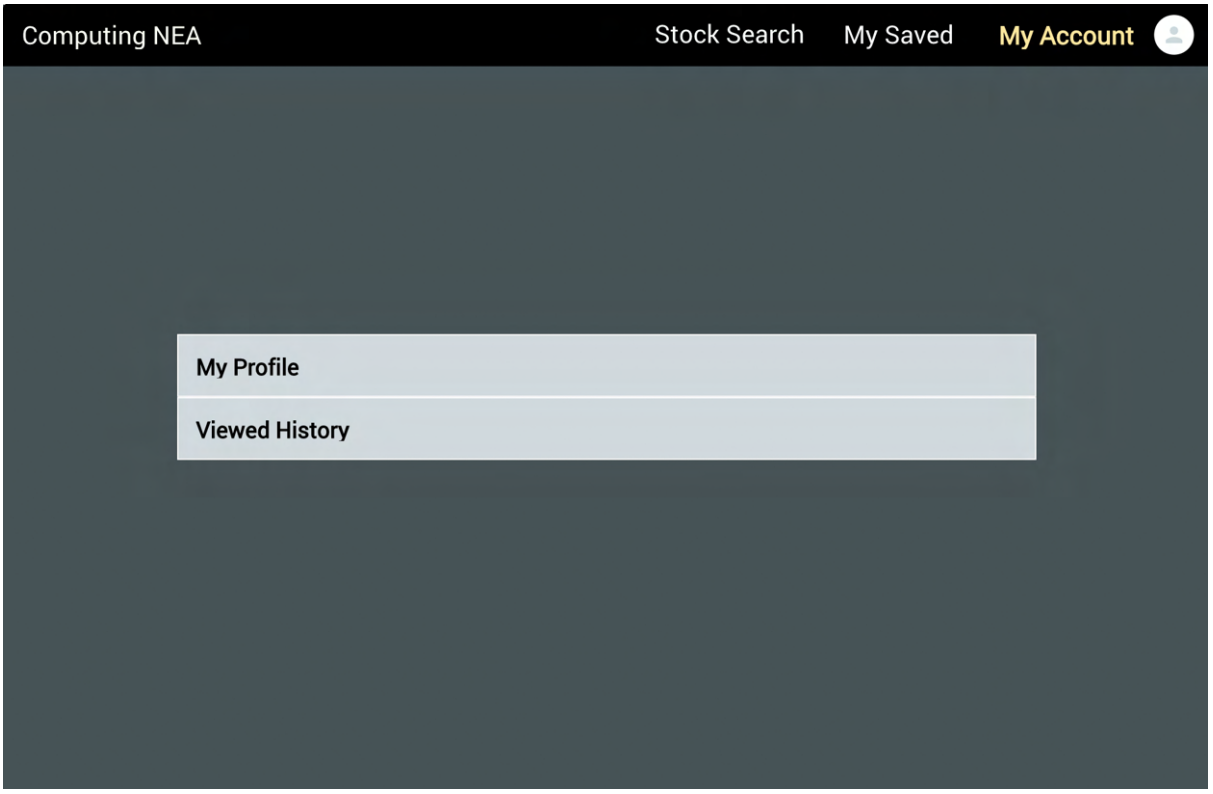


Figure 28 - UI for “My Account” Page

When the user clicks on the “My Account” page, Figure 28 will be displayed to the user. Here, the user can click change their profile settings or view their viewed history.

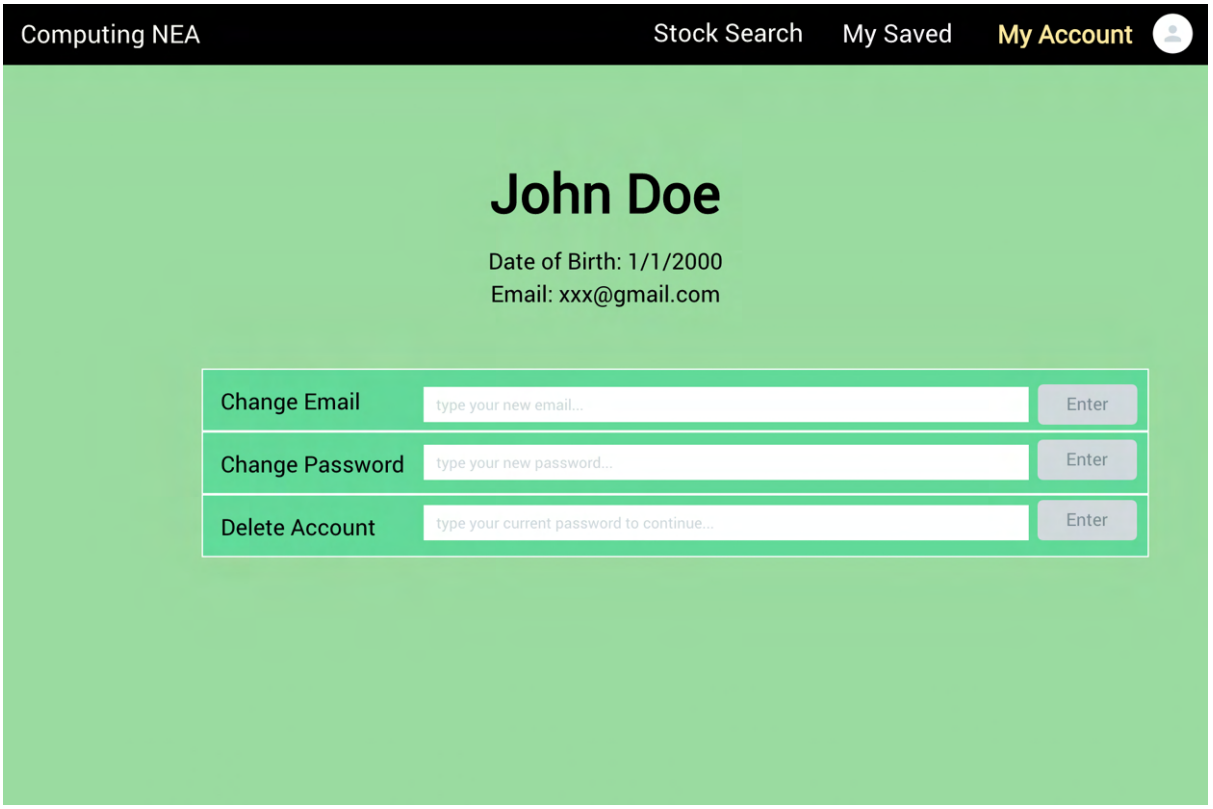


Figure 29 - UI for My Profile

If the user were to click on the “My Profile” tab in Figure 28, the page will be redirected as shown in Figure 29. This page shows key user information such as their Name, Date of Birth, and current email address. Furthermore, the user is able to change their Email or Password (This is done by sending a request to the Firebase authentication server). In addition, the user can delete their account, but only if they confirm it by entering their current password.

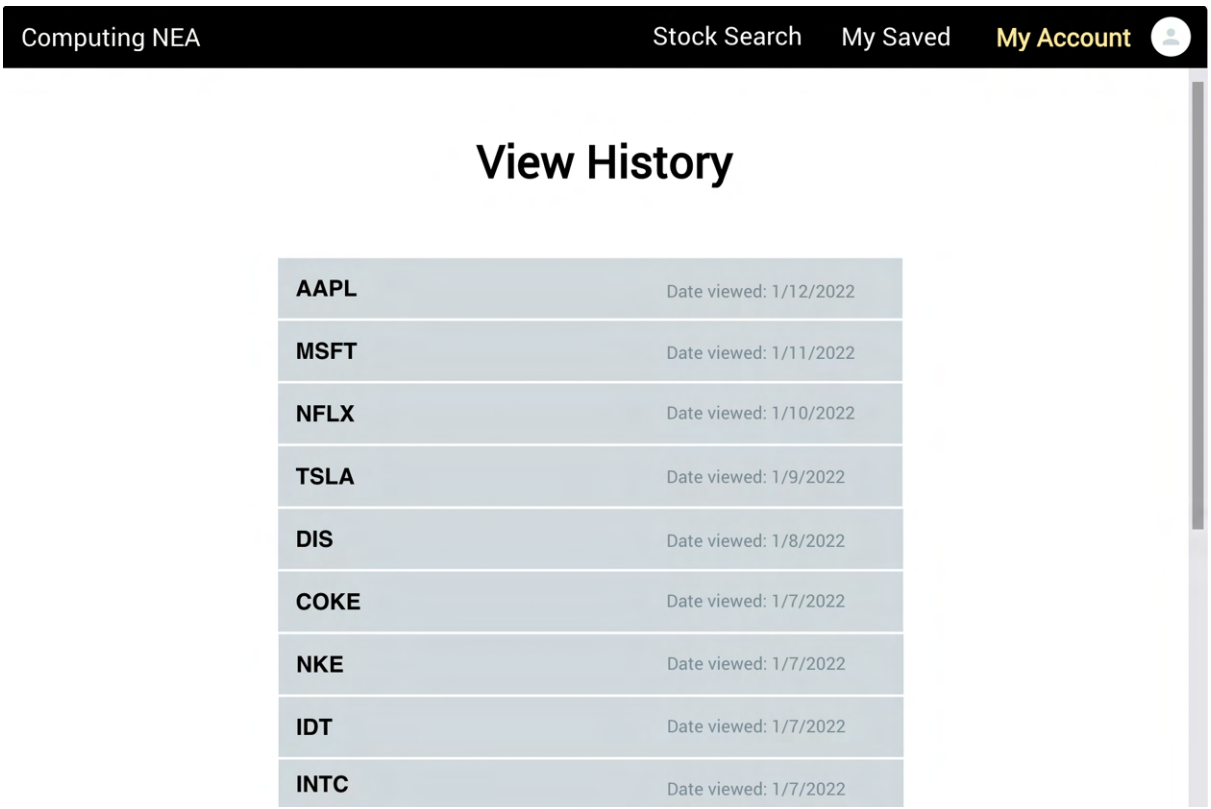


Figure 30 - UI of “View History” page

The second option on the “My account” page is for the user to view their viewed history. Here, the user can look at what stock tickers they previously viewed, along with the date viewed. If the user has multiple viewed tickers, they can scroll down to check for older view history.

Hardware & software requirements:

1. Requires Python, and Jupyter Notebook installed.
2. Requires the below libraries to be installed:

```
Package                       Version
----------------------------- ----------
absl-py                       1.1.0
aiohttp                       3.8.1
aiosignal                     1.2.0
appdirs                       1.4.4
appnope                       0.1.3
argon2-cffi                   21.3.0
argon2-cffi-bindings          21.2.0
astroid                       2.15.0
asttokens                     2.0.5
astunparse                    1.6.3
```

```
async-generator               1.10
async-timeout                 4.0.2
attrs                         21.4.0
backcall                      0.2.0
backports.functools-lru-cache 1.6.4
beautifulsoup4                4.11.1
bleach                        5.0.1
blis                          0.7.8
brotlipy                      0.7.0
bs4                           0.0.1
CacheControl                  0.12.11
cached-property               1.5.2
cachetools                    5.2.0
catalogue                     2.0.7
certifi                       2022.6.15
cffi                          1.15.1
charset-normalizer            2.1.0
click                         8.1.3
colorama                      0.4.5
cryptography                  37.0.4
cssselect                     1.2.0
cycler                        0.11.0
cymem                         2.0.6
dataclasses                   0.8
datasets                      2.3.2
debugpy                       1.6.0
decorator                     5.1.1
defusedxml                    0.7.1
dill                          0.3.5.1
en-core-web-sm                3.4.0
entrypoints                   0.4
et-xmlfile                    1.1.0
etils                         0.6.0
exceptiongroup                1.0.1
executing                     0.8.3
fake-useragent                1.1.1
fastjsonschema                2.16.1
feedparser                    6.0.10
filelock                      3.7.1
financedatabase               1.0.2
firebase-admin                6.0.1
flatbuffers                   1.12
flit_core                     3.7.1
fonttools                     4.34.4
frozendict                    2.3.4
frozenlist                    1.3.0
fsspec                        2022.5.0
gast                          0.4.0
gcloud                        0.18.3
google-api-core               2.11.0
google-api-python-client      2.74.0
google-auth                   2.16.0
google-auth-httplib2          0.1.0
google-auth-oauthlib          0.4.6
google-cloud-core             2.3.2
google-cloud-firestore        2.9.1
```

```
google-cloud-storage          2.7.0
google-crc32c                 1.5.0
google-pasta                  0.2.0
google-resumable-media        2.4.1
googleapis-common-protos      1.56.4
graphviz                      0.20.1
grpcio                        1.51.1
grpcio-status                 1.51.1
h11                           0.14.0
h5py                          3.6.0
html5lib                      1.1
httplib2                      0.21.0
huggingface-hub               0.11.1
idna                          3.3
importlib-metadata            4.11.4
importlib-resources           5.8.0
ipykernel                     6.15.1
ipython                       8.4.0
ipython-genutils              0.2.0
ipywidgets                    7.7.1
isort                         5.12.0
jedi                          0.18.1
Jinja2                        3.1.2
joblib                        1.2.0
jsonschema                    4.7.2
jupyter                       1.0.0
jupyter-client                7.3.4
jupyter-console               6.4.4
jupyter-core                  4.10.0
jupyterlab-pygments           0.2.2
jupyterlab-widgets            1.1.1
jws                           0.1.3
keras                         2.9.0
Keras-Preprocessing           1.1.2
kiwisolver                    1.4.4
langcodes                     3.3.0
lazy-object-proxy             1.9.0
libclang                      14.0.1
lightgbm                      3.3.5
lxml                          4.9.1
Markdown                      3.4.1
MarkupSafe                    2.1.1
matplotlib                    3.5.2
matplotlib-inline             0.1.3
mccabe                        0.7.0
mistune                       0.8.4
msgpack                       1.0.4
multidict                     6.0.2
multiprocess                  0.70.13
multitasking                  0.0.11
munkres                       1.1.4
murmurhash                    1.0.7
mysql-connector-python        8.0.32
nbclient                      0.5.13
nbconvert                     6.4.5
nbformat                      5.4.0
```

```
nest-asyncio                  1.5.5
nltk                          3.7
notebook                      6.4.12
numpy                         1.24.1
oauth2client                  4.1.3
oauthlib                      3.2.0
openpyxl                      3.1.0
opt-einsum                    3.3.0
outcome                       1.2.0
packaging                     21.3
pandas                        1.5.3
pandas-datareader             0.10.0
pandocfilters                 1.5.0
parse                         1.19.0
parso                         0.8.3
pathy                         0.6.2
patsy                         0.5.2
pexpect                       4.8.0
pickleshare                   0.7.5
Pillow                        9.2.0
pip                           22.1.2
platformdirs                  3.1.0
preshed                       3.0.6
prometheus-client             0.14.1
promise                       2.3
prompt-toolkit                3.0.30
proto-plus                    1.22.2
protobuf                      3.20.1
psutil                        5.9.1
ptyprocess                    0.7.0
pure-eval                     0.2.2
py-readability-metrics        1.4.5
pyarrow                       8.0.0
pyasn1                        0.4.8
pyasn1-modules                0.2.8
pycparser                     2.21
pycryptodome                  3.17
pydantic                      1.9.1
pydotplus                     2.0.2
pyee                          8.2.2
Pygments                      2.12.0
PyJWT                         2.6.0
pylint                        2.17.0
pyOpenSSL                     22.0.0
pyparsing                     3.0.9
pyphen                        0.13.2
pyppeteer                     1.0.2
pyquery                       2.0.0
pyrsistent                    0.18.1
pysentiment2                  0.1.1
PySocks                       1.7.1
python-dateutil               2.8.2
python-jwt                    2.0.1
pytz                          2022.7
PyYAML                        6.0
pyzmq                         23.2.0
```

```
regex                          2022.7.9
requests                       2.28.2
requests-html                  0.10.0
requests-oauthlib              1.3.1
requests-toolbelt              0.10.1
responses                      0.18.0
rsa                            4.8
sacremoses                     0.0.53
scikit-learn                   1.2.1
scipy                          1.8.1
seaborn                        0.11.2
selenium                       4.6.0
Send2Trash                     1.8.0
sentence-transformers          2.2.2
sentencepiece                  0.1.96
setuptools                     63.2.0
sgmllib3k                      1.0.0
shellingham                    1.4.0
six                            1.16.0
smart-open                     5.2.1
sniffio                        1.3.0
sortedcontainers               2.4.0
soupsieve                      2.3.1
spacy                          3.4.0
spacy-legacy                   3.0.9
spacy-loggers                  1.0.3
srsly                          2.4.3
stack-data                     0.3.0
statsmodels                    0.13.2
tensorboard                    2.9.1
tensorboard-data-server        0.6.1
tensorboard-plugin-wit         1.8.1
tensorflow-datasets            4.6.0
tensorflow-estimator           2.9.0
tensorflow-macos               2.9.2
tensorflow-metadata            1.9.0
tensorflow-metal               0.5.0
termcolor                      1.1.0
terminado                      0.15.0
testpath                       0.6.0
textstat                       0.7.3
thepassiveinvestor             1.1.2
thinc                          8.1.0
thinc-apple-ops                0.1.0.dev1
threadpoolctl                  3.1.0
tokenizers                     0.12.1
toml                           0.10.2
tomli                          2.0.1
tomlkit                        0.11.6
torch                          1.13.1
torchaudio                     0.12.0
torchvision                    0.13.0
tornado                        6.2
tqdm                           4.64.0
traitlets                      5.3.0
transformers                   4.26.0
```

```
trio                          0.22.0
trio-websocket                0.9.2
typer                         0.4.2
typing_extensions             4.3.0
unicodedata2                  14.0.0
Unidecode                     1.3.6
uritemplate                   4.1.1
urllib3                       1.26.10
w3lib                         2.1.1
wasabi                        0.9.1
wcwidth                       0.2.5
webencodings                  0.5.1
websockets                    10.4
Werkzeug                      2.1.2
wheel                         0.37.1
widgetsnbextension            3.6.1
wrapt                         1.14.1
wsproto                       1.2.0
xgboost                       1.7.3
xxhash                        3.0.0
yahoo-fin                     0.8.9.1
yahoofinancials               1.12
yarl                          1.7.2
yfinance                      0.2.10
zipp                          3.8.0
```

Implementation & Technical Solution

Preparing the Earnings Call Transcripts (finding index 0-3):

Prepare_csv_files.ipynb

Prepare Stock information URLs:

```
numblist = [1,2,3,4,5,6,7]

sectorlist = ["automobiles", "banks", "capital-goods", "commercial-services",
"consumer-durables", "consumer-retailing", "consumer-services", "diversified-financials",
"energy", "food-beverage-tobacco", "healthcare", "household", "insurance", "materials", "media",
"pharmaceuticals-biotech", "real-estate", "retail", "semiconductors", "software",
"tech", "telecom", "transportation", "utilities"]

urllist = []

for sector in sectorlist:
   for number in numblist:
       urllist.append('https://simplywall.st/stocks/us/' + sector + '/market-cap-large?page=' +
str(number))


count = 0
count2 = 0
df = pd.DataFrame(columns=['href','ticker'])

for i in range(0, len(urllist)+1):
   if count == 7:
           count = 0
           df.to_csv('hrefs/ref_' + (sectorlist[count2]) + '.csv', index=False)
           count2 += 1
           df = pd.DataFrame(columns=['href','ticker'])

   try:
       url = urllist[i]
       print(url)
       count+=1
       html_text = requests.get(url).text

       soup = BeautifulSoup(html_text, "html.parser")
       body = soup.find('tbody').find_all('tr')

       for j in body:
           temp = j.find_all('td')[1]
           insert_row = {"href": (temp.a['href']), "ticker": ''.join(temp.find('b').contents)}
           df = pd.concat([df, pd.DataFrame([insert_row])])

   except:
       continue
```

Figure 3 - Code for finding Ticker name, and URLS

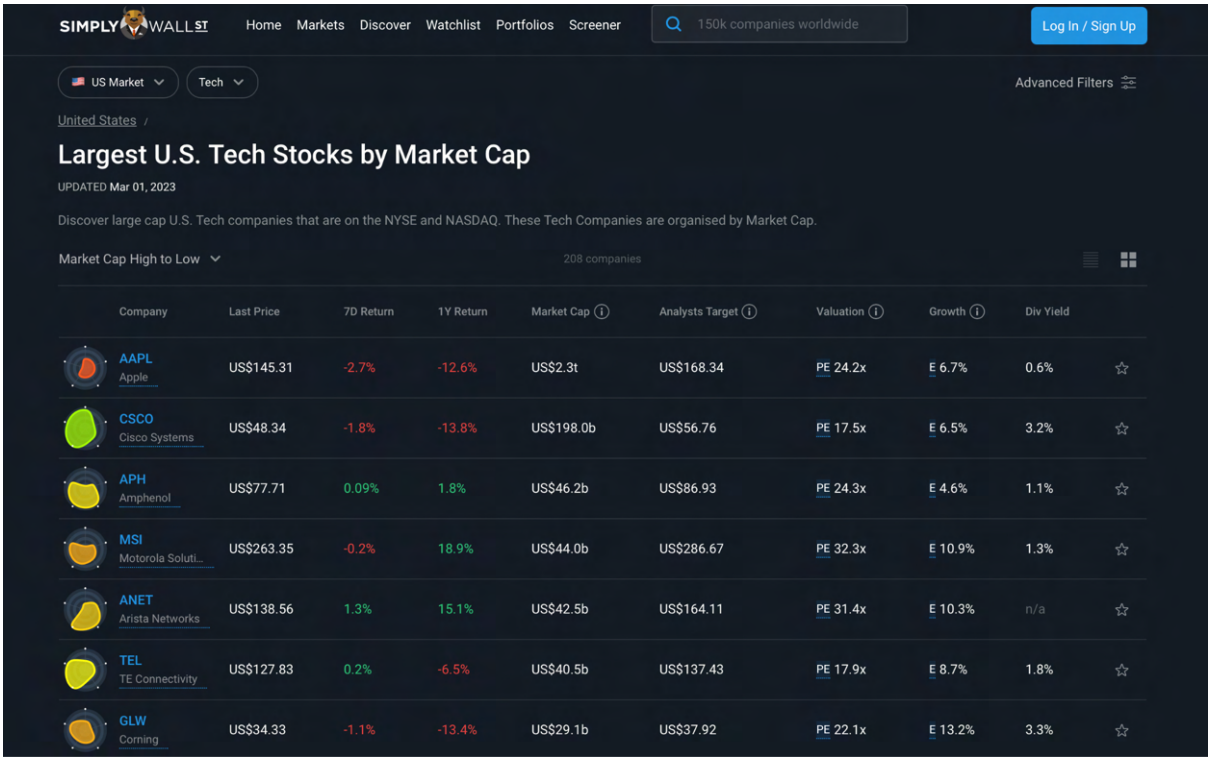


Figure 4 - largest US stocks sorted by market cap for the sector "tech"

In the Figure 3 code snippet, I am trying to find a list of stocks and their respective sector to analyse. I found https://simplywall.st/stocks/us/ to be very helpful as it contains thousands of tickers (sorted by market cap) per each industry (see Figure 4).

First, I created a list called "urllist" to get the website urls for each sector, such as:
https://simplywall.st/stocks/us/tech/market-cap-large?page=1
https://simplywall.st/stocks/us/utilities/market-cap-large?page=1
Then I incremented the page number to find the data of the next page, e.g. https://simplywall.st/stocks/us/utilities/market-cap-large?page=2.
Using these urls, I can cycle throug heach one and use Request and BeautifulSoup to parse the html data and get the URL reference of each stock's page. This is because each stock's URL is formatted differently and it will be inefficient to find a way to guess what their URLs will be. For example in the list below, even though we would assume Apple and Microsoft to be a tech stock in the same industry, however, Apple belongs to tech and Microsoft to software. Similarly, JP Morgan is under the New York Stock Exchange (NYSE), whereas Apple is under Nasdaq.

https://simplywall.st/stocks/us/tech/nasdaq-aapl/apple
https://simplywall.st/stocks/us/software/nasdaq-msft/microsoft
https://simplywall.st/stocks/us/banks/nyse-jpm/jpmorgan-chase

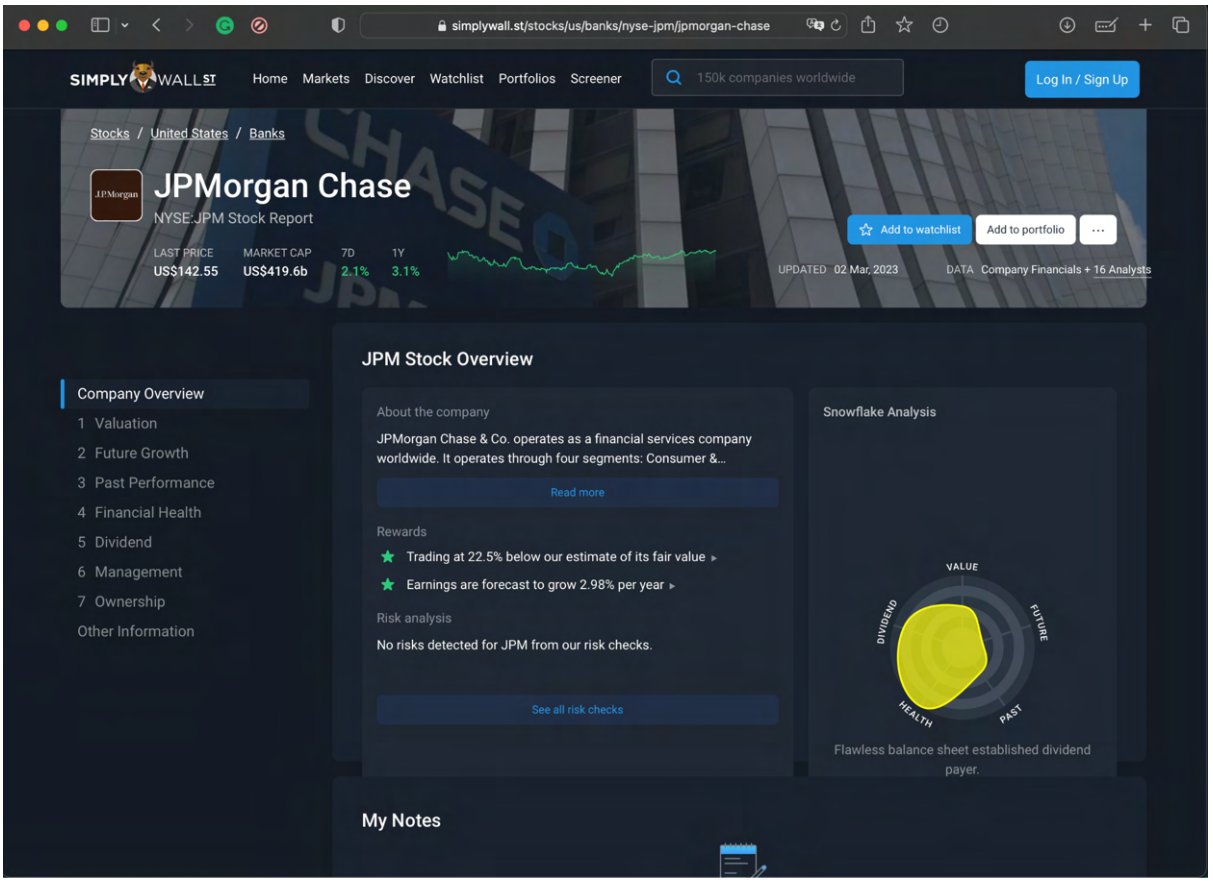


Figure 5 - The URL of
https://simplywall.st/stocks/us/banks/nyse-jpm/jpmorgan-chase

A try, except block will also skip any errors in case Request or BeautifulSoup couldn't find that page. After parsing the relevant HTML, the "insert_row" content will display:
{'href': '/stocks/us/automobiles/nasdaq-tsla/tesla', 'ticker': 'TSLA'} or
{'href': '/stocks/us/automobiles/nyse-gm/general-motors', 'ticker': 'GM'}
This will be added to a pandas dataframe and written on a .csv file named ref_[sector].csv, under the folder name "hrefs" (see below Figure 6).

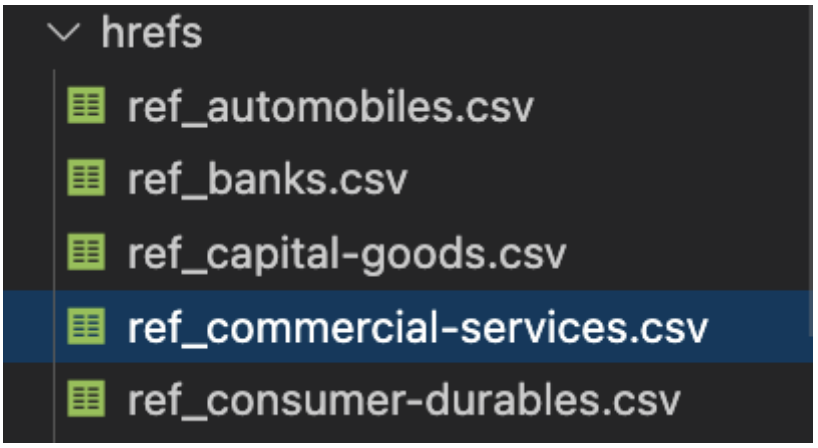


Figure 6 - hrefs folder

Prepare the “Fair Value” Data

```
for sector in sectorlist:
   df = pd.read_csv('hrefs/ref_' + (sector) + '.csv')

   hrefs = df["href"]
   df_fair_value = pd.DataFrame()
   for href in hrefs:
       time.sleep(1.5)
       url = "https://simplywall.st" + str(href) + "/valuation"
       html_text = requests.get(url).text
       soup = BeautifulSoup(html_text, "html.parser")
       body = soup.find_all('blockquote')
       count = 0
       fair_value = np.NaN
       for i in body:
           count += 1
           if count < 25:
               Try:
                   content = ''.join(i.find('p').find_all('span')[3].contents)
                   if content[2] == "$":
                       fair_value = content
               except:
                   continue
           else:
               break
       insert_row = {"fair_value": fair_value}

       df_fair_value = pd.concat([df_fair_value, pd.DataFrame([insert_row])])

   df_initial = pd.read_csv('hrefs/ref_'+sector+'.csv')

   temp2 = df_fair_value.to_numpy()

   df_initial.insert(loc=2, column="fair_value", value=temp2)
   df_initial.to_csv('hrefs/ref_' + (sector) + '.csv', index=False)
```

Figure 7 - Code for finding the estimated “fair value” for each stock

As illustrated by Figure 7, I then find the “fair value” of each stock using various HTML parsing techniques and conditions.

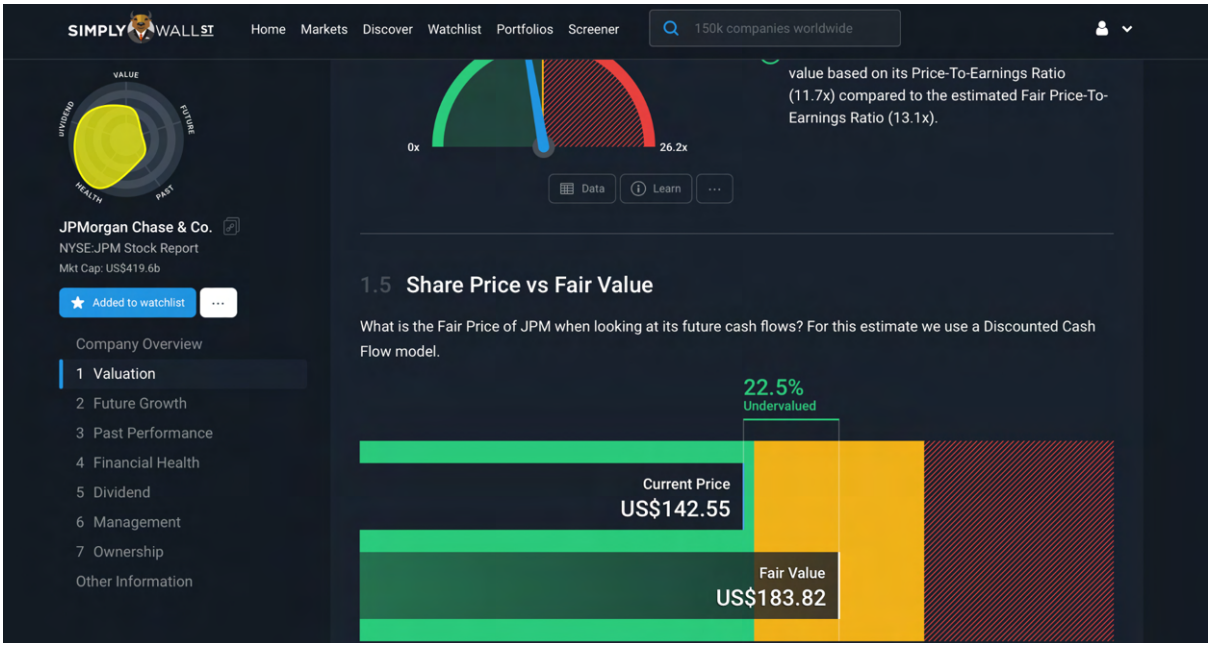


Figure 8 - Fair Value

As demonstrated in Figure 8, the fair value is estimated to be $183.82, I would need to try to find this piece of data in the HTML.

Finally, I used a pandas concat method to add the “fair_value” to the pandas dataframe and update the relevant href files, as referenced with Figure 9 below.

```
1  href,ticker,fair_value
2  /stocks/us/automobiles/nasdaq-tsla/tesla,TSLA, ($169.12)
3  /stocks/us/automobiles/nyse-f/ford-motor,F, ($38.05)
4  /stocks/us/automobiles/nyse-gm/general-motors,GM, ($62.65)
5  /stocks/us/automobiles/nyse-race/ferrari,RACE, ($63.33)
```

Figure 9 - First 4 rows of hrefs/ref_automobiles.csv

```
for sector in sectorlist:
   df_initial = pd.read_csv('hrefs/ref_'+sector+'.csv')
   print(df_initial)
   df_initial = df_initial.drop_duplicates()
   df_initial.to_csv('hrefs/ref_'+sector+'.csv', index=False)
```

Figure 10 - Deletes duplicate values in .csv files

With reference to Figure 10, the programme then gets rid of any duplicates in .csv files located under /hrefs/.

Finding Transcript Data:

```
def find_transcript(ticker, date_yr, date_qtr, daterange_index):
   try:
```

```
        url = 'https://roic.ai/transcripts/' + ticker + '?y=' + str(date_yr) + '&q=' + str(date_qtr)
        html_text = requests.get(url).text
        soup = BeautifulSoup(html_text, "html.parser")
        # the website below is used to "beautify" JSON files (e.g. indent it)
        # https://jsonformatter.org/
        # finds the 15th <script/> and strips the unnecsssary data so it can be read in JSON
        script = soup.find_all('script')[15].text.strip()
        data = json.loads(script)
        transcript_data = data['props']['pageProps']['transcriptdata']['content']  # loads the transcript
content
        exact_date =
datetime.strptime(data['props']['pageProps']['data']['data']['earningscalls'][daterange_index]['date'],
'%Y-%m-%d %H:%M:%S')
    except:
        transcript_data = np.NaN
        exact_date = np.NaN

    return transcript_data, exact_date

def ticker_transcript(ticker):
    # loops around each ticker, and gets transcript data
    try:
        url = 'https://roic.ai/transcripts/' + ticker

        html_text = requests.get(url).text
        soup = BeautifulSoup(html_text, "html.parser")

        script = soup.find_all('script')[15].text.strip()
        # finds json data (i.e. transcript)
        data = json.loads(script)
        first_er_date = data['props']['pageProps']['data']['data']['earningscalls'][0]
        last_er_date = data['props']['pageProps']['data']['data']['earningscalls'][-1]
        # reads the json data and finds the first and last date for which the transcript is held and its
set as limits
        # perhaps I worded it badly...
        # first_er_date_yr represents the most recent transcript's date (i.e. 2022)
        # last_er_date represents the last transcript's date (e.g. 2006)

        first_er_date_yr = first_er_date['year']
        first_er_date_qtr = first_er_date['quarter']
        last_er_date_yr = last_er_date['year']
        # calculates the year differences between the dates of first and last transcript so it can be
looped
        yr_difference = first_er_date_yr-last_er_date_yr
        date_yr = first_er_date_yr
        date_qtr = first_er_date_qtr

        arr_ticker_infos = np.empty((0, 4), str)
        arr_full_transcript = np.array([], str) #array of all of the transcripts, every year and quarter
        daterange_index = -1

        for i in range(0, yr_difference + 1):
            #there are two seperate loops below:
            if date_yr == first_er_date_yr:
                # the first loop is to loop through the most recent transcript as the quarter does not end
with 4
```

```
                # as of when this programme is written, the most recent transcript is Q2 of 2022
                for j in range(0, first_er_date_qtr):
                    daterange_index += 1
                    #finds the relevant 13 initial dataset as described in page 2 of the Doc
                    #it finds them by calling the function find_transcript() located below
                    full_transcript, exact_date = find_transcript(ticker, date_yr, date_qtr,
daterange_index)
                    arr_full_transcript = np.append(arr_full_transcript, full_transcript)
                    arr_ticker_infos = np.append(arr_ticker_infos, np.array([[ticker, date_yr, date_qtr,
exact_date]]), axis=0)
                    date_qtr -= 1
                    if date_qtr == 0:
                        date_qtr = 4
            else:
                # the second loop loops through the rest of the transcripts
                for j in range(0, 4):
                    daterange_index += 1

                    full_transcript, exact_date = find_transcript(ticker, date_yr, date_qtr,
daterange_index)
                    arr_full_transcript = np.append(arr_full_transcript, full_transcript)
                    arr_ticker_infos = np.append(arr_ticker_infos, np.array([[ticker, date_yr, date_qtr,
exact_date]]), axis=0)

                    date_qtr -= 1
                    if date_qtr == 0:
                        date_qtr = 4
            date_yr -= 1
    except:
        arr_full_transcript = np.NaN
        arr_ticker_infos = np.NaN

    return arr_full_transcript, arr_ticker_infos
```

Figure 11 - Find, Parse, and Save Transcripts Data code

There are two functions in Figure 11, `ticker_transcript(ticker)` and `find_transcript(ticker, date_yr, date_qtr, daterange_index)`.

The function `ticker_transcript(ticker)` uses the stock ticker symbol as an argument. As previously mentioned in the **Design Section,** the website roic.ai is used as earnings transcript are available there, thus the url can be represented as `url = 'https://roic.ai/transcripts/' + ticker`.

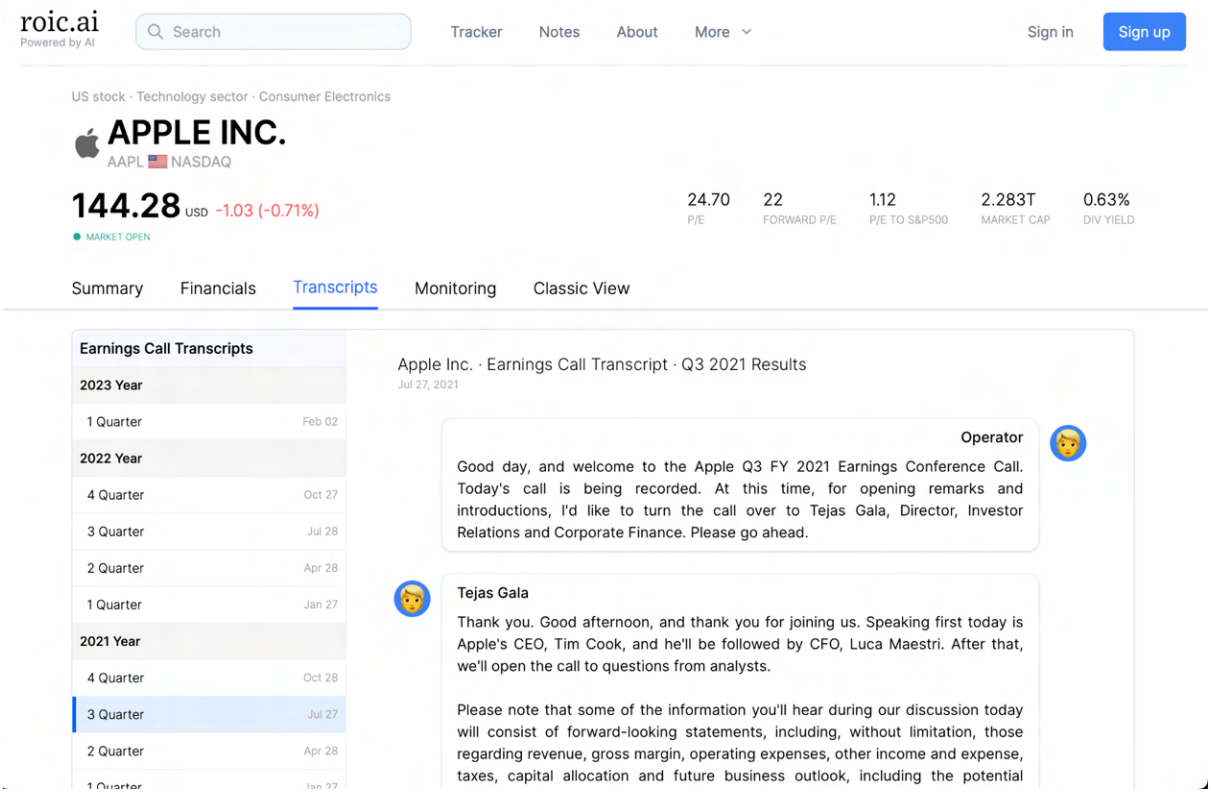


Figure 12 - Earnings Transcript on Roic.ai for https://roic.ai/transcripts/AAPL?y=2021&q=3

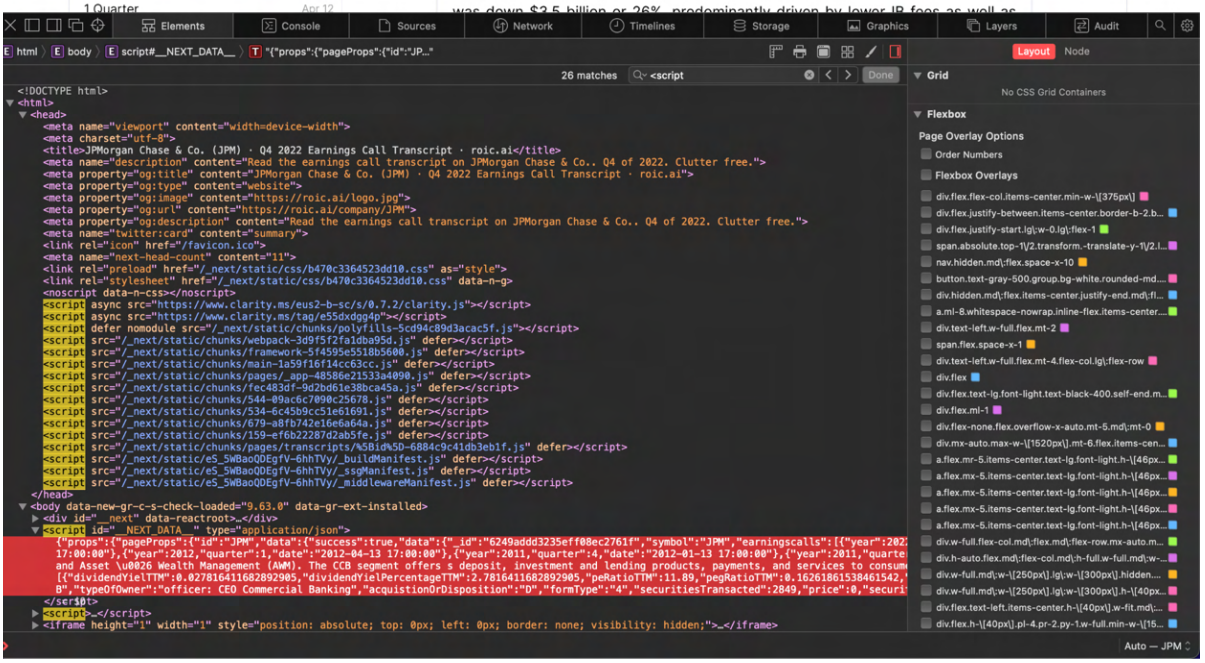


Figure 13 - Inspect element location of where the transcript's JSON data are stored in roic.ai for https://roic.ai/transcripts/JPM?y=2022&q=4

Using Request and Beautifulsoup, I can get the entirety of the earnings call transcript

by parsing the relevant HTML and JSON data. The highlighted <script id> in Figure 13 shows the contents of the earnings transcript stored in JSON format.

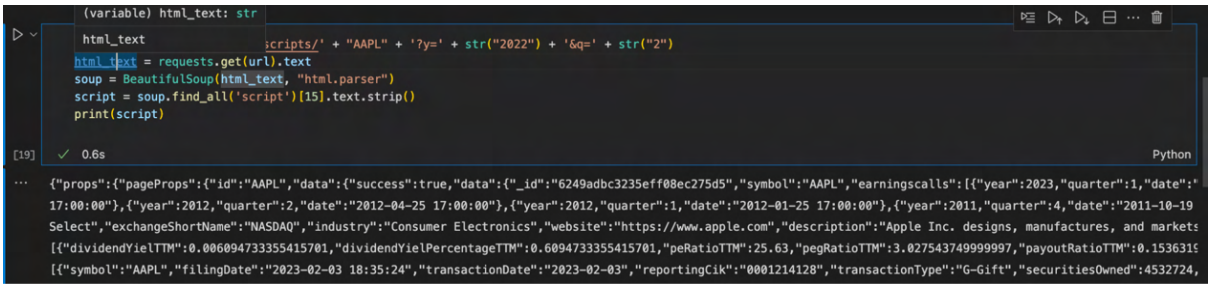


Figure 14 - Accessing the contents of the <script> tag

Referencing figure 14, we can search for all the contents that is enclosed within the <script> HTML tag. Furthermore, the content that is highlighted in Figure 13 is on the 15th index so we can take this unformatted JSON information and “beautify” it using https://jsonformatter.org/# (see Figure 15)

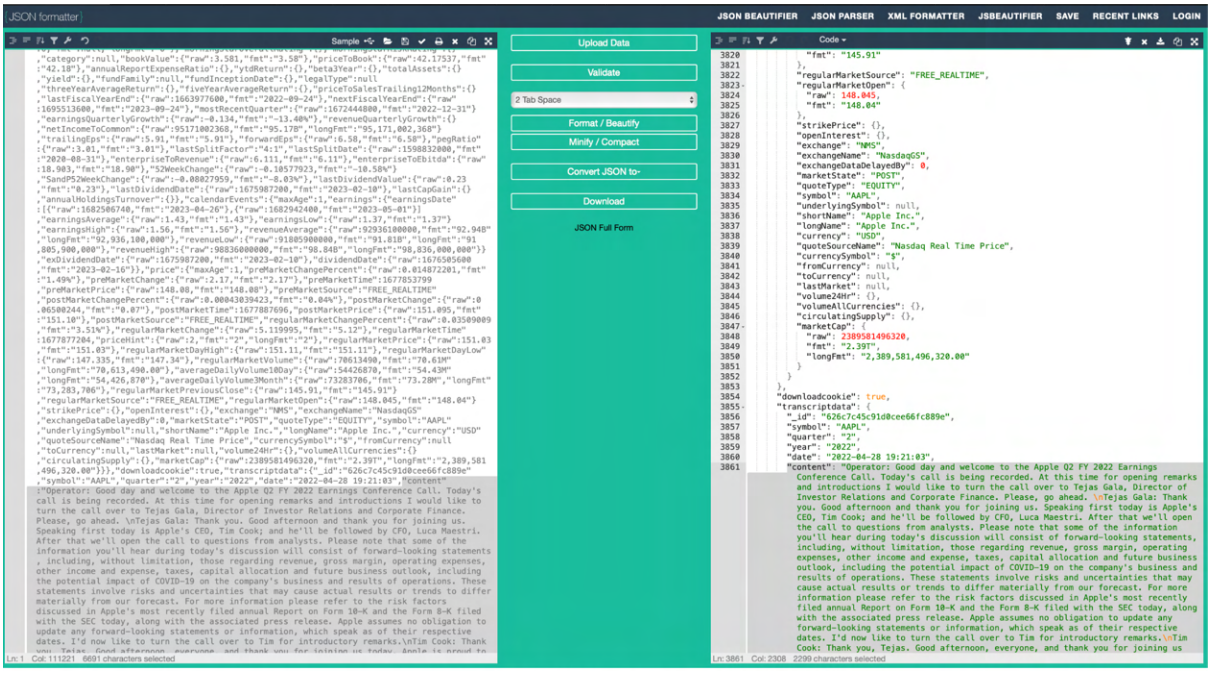


Figure 15 - “Beautifying” JSON formatting

With reference to Figure 15, we can easily find the location of the earnings transcript contents after formatting the JSON file. Thus running the python command `transcript_data = data['props']['pageProps']['transcriptdata']['content']` will offer the contents of the transcript data.

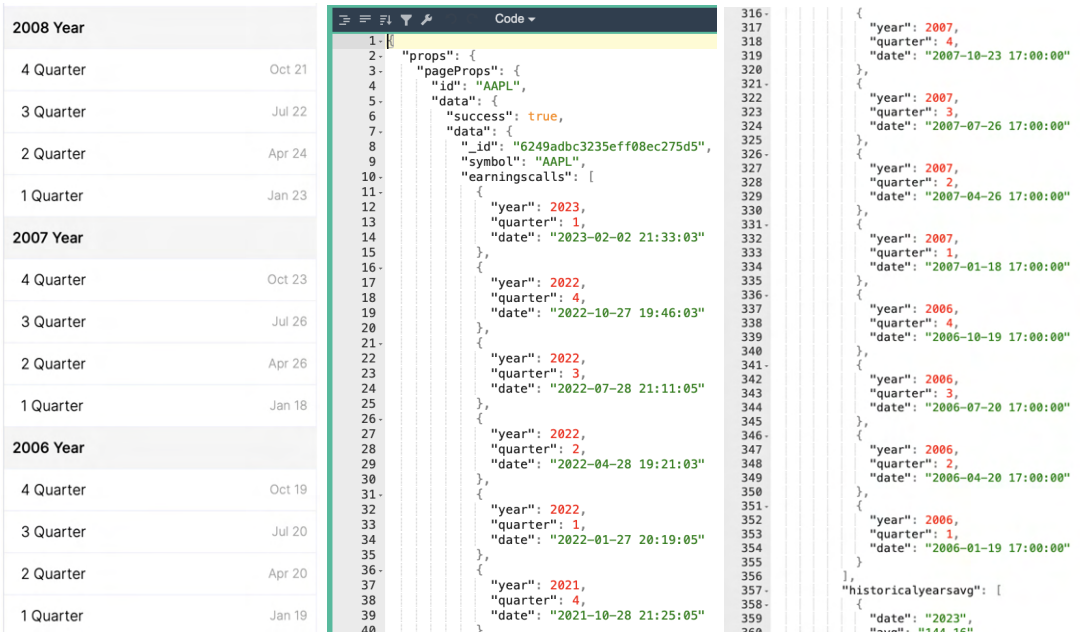


Figure 16a, 16b, 16c

However, before getting the transcript_data, we must know their URL links. For example, if we want to get Apple's 2023 Q1 and 2022 Q4 earnings transcript, their respective URL is: https://roic.ai/transcripts/AAPL?y=2023&q=1, https://roic.ai/transcripts/AAPL?y=2022&q=4.

A problem arises since we don't know when Apple started releasing earnings transcript, was their first ever earnings transcript dated all the way back in the 1990s? According to Figure 16a, roic.ai's database only holds earnings transcript since the 2006.

Another problem also arises… We don't know the date of Apple's most recent earnings transcript. Companies release quarterlies at different times, and every company uses a different definition of "Quarterly". For some companies, "Q4" may mean being released in December, for others however, this may mean as late as March. Therefore, we must find out what's the range of dates for a particular stock. In Apple's example, they started releasing them in 2006_Q1, whereas the most recent one is 2023_Q1. Figure 16b and 16c shows the JSON data of when each quarterly is released.

For simplicity sake, the "first" earnings call represents the most recent earnings call (i.e. 2023_Q1), whereas the "last" earnings call represents the oldest (i.e. 2006_Q1). This is because the way the JSON is formatted, so when I transform it into a python list, the "first index" of the list will be the most recent earnings call.

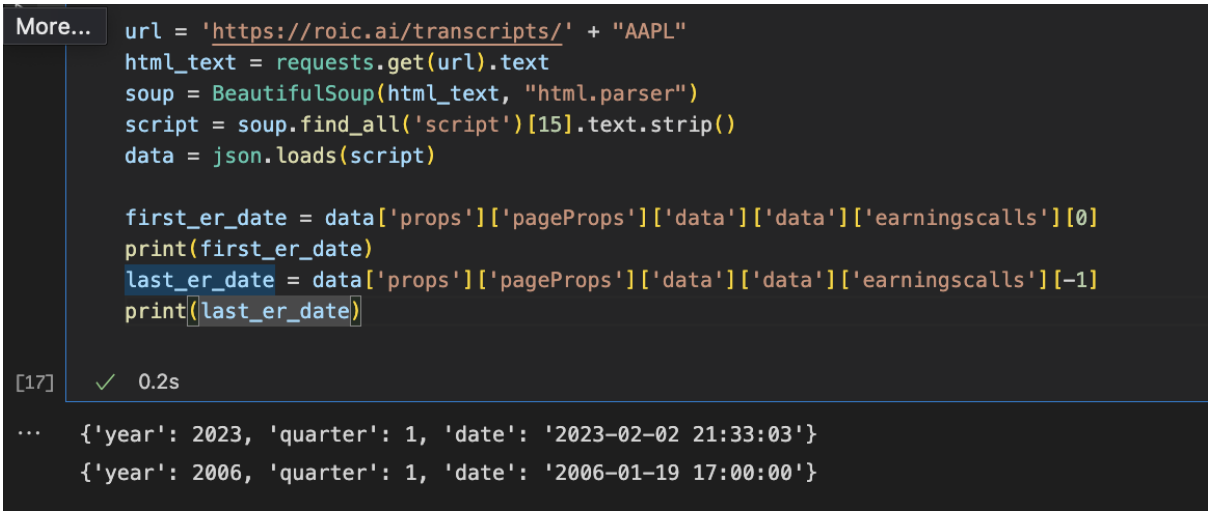


Figure 17 - the date range of the earnings transcripts

With reference to the function `ticker_transcript(ticker)`, Figure 16b, Figure 16c, and Figure 17, we can find the "first" earnings transcript and the "last" earnings transcript of that particular stock.

Furthermore, we can find the integer value of the **year** and **quarter** using the python command: `first_er_date_yr = first_er_date['year']` and `first_er_date_qtr = first_er_date['quarter']`. (ER represents earnings report, which is synonymous with earnings call).

Now, that we know the first and last earnings call dates, we can find the "range" i.e. the difference in years (`yr_difference = first_er_date_yr-last_er_date_yr`). `Yr_difference` will be used as the "range" value for the for loop, thus with this information in mind, we can cycle through the each earnings transcript and run the function `find_transcript(ticker,date_yr,date_qtr,daterange_index)` to find the transcript contents.

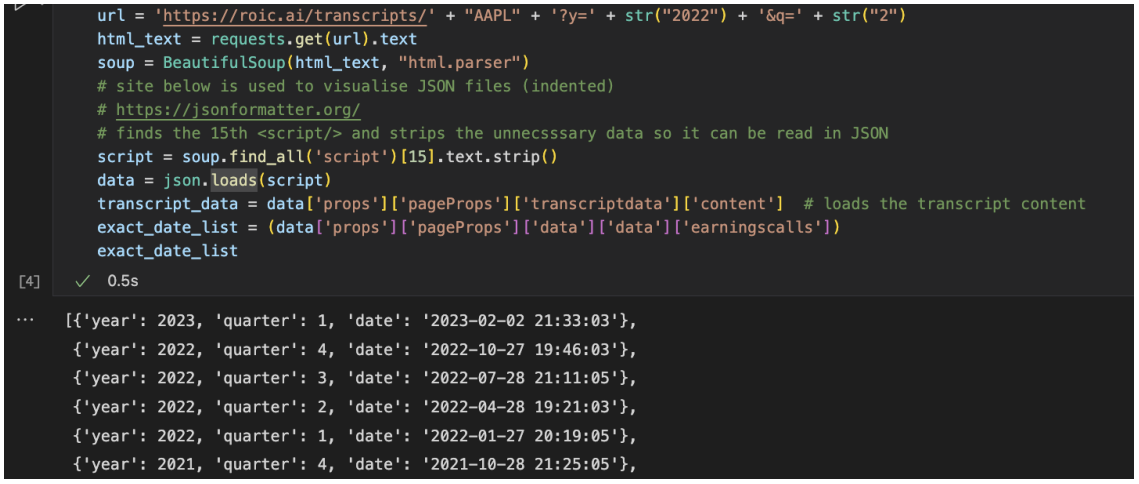


Figure 18 - daterange_index

In `find_transcript(ticker,date_yr,date_qtr,daterange_index)`, it also takes the argument `daterange_index`. With reference to Figure 18, the `exact_date_list` represents a list that contains all the dates of the earnings release, where every one of them maps onto a

particular earnings transcript (1:1). `daterange_index` will be used to determine which datetime value (i.e. 2023-02-02 21:33:03) maps onto the current set of earnings transcript.

Saves Transcript Data to .csv

```
#Finds transcript and creates the .csv files
for sector in sectorlist:
   df_stock_list = pd.read_csv('hrefs/ref_'+sector+'.csv')['ticker']
   for stock in df_stock_list:
       try:
           try:
               filelist = os.listdir("sectors/"+sector)
           except:
               filelist = []
           if stock not in filelist:
               arr_full_transcript, arr_ticker_infos = ticker_transcript(stock)
               FOLDERPATH = 'sectors/' + sector + '/' + str(stock)
               if not os.path.exists(FOLDERPATH):
                   os.makedirs(FOLDERPATH)
               for i in range(0,len(arr_ticker_infos)):
                   year = arr_ticker_infos[i][1]
                   quarter = arr_ticker_infos[i][2]
                   release_date = arr_ticker_infos[i][3]
                   single_transcript = arr_full_transcript[i]
                   df_transcript_details = pd.DataFrame([year, quarter, release_date, single_transcript])
                   FILEPATH = 'sectors/'+sector+'/'+str(stock)+'/'+str(stock)+str(year)+str(quarter)+".csv"
                   df_transcript_details.to_csv(FILEPATH, header=False, index=False)
       except:
           continue
```

Figure 19 - code for saving transcript data to .csv file

Figure 20a, 20b - an example for the the stock “ADNT” for 2017 Q2

Figure 19 uses python's **OS** and **pandas** library to save transcript data to a .csv file. Looking at Figure 20a, each .csv file contains [year, quarter, datetime, transcript_data] in that order. Furthermore, Figure 20b shows how the .csv files for each stock are organised. For example, the stock ADNT works in the automobiles sector, thus the directory path is **sectors/automobiles/ADNT**. Then, under the subdirectory **./ADNT**, it has .csv files organised by [stock][year][quarter].csv, such as **ADNT20193.csv.**

Deletes all .csv files that hold (transcript_data = NaN)

```
#deletes all csv files with NaN values as transcript data
for sector in sectorlist:
   filelist = os.listdir("sectors/"+sector)
   try:
       filelist.remove('.DS_Store')
   except:
       pass
   for stock in filelist:
       sector_files = glob.glob('sectors/'+str(sector)+'/'+str(stock)+'/'+str(stock)+'20*[0-9]**[0-9]*[1-4].*')
       for directory_link in sector_files:
           try:
               df_transcript = pd.read_csv(directory_link)
               my_transcript = df_transcript.loc[2]
               if str(my_transcript[0]) == "nan":
                   os.remove(directory_link)
           except:
               pass
```

Figure 21 - code for deleting .csv files with NaN values for transcript_data

Deletes any stock directories with less than 5 .csv files

```
#deletes stock directories with less than 5 earnings calls
for sector in sectorlist:
   filelist = os.listdir("sectors/"+sector)
   try:
       filelist.remove('.DS_Store')
   except:
       pass
   for stock in filelist:
       sector_files = glob.glob('sectors/'+sector+'/'+str(stock))
       for directory_link in sector_files:
           if len(os.listdir(directory_link)) < 5:
               shutil.rmtree(directory_link, ignore_errors=True)
```

Figure 22 - code for deleting stock directories with less than 5 earnings transcripts

Some stocks may be newly IPOed (inital public offering), or newly merged, or perhaps roic.ai just doesn't have their past earnings calls. Hence, in some cases, these stocks may only have 3 or 4 earnings transcript .csv files. I decided to delete these data since they may only add noise.

Sorts the stock directory into an orderedlist

```
for sector in sectorlist:
   filelist = os.listdir("../sectors/"+sector)
   try:
       filelist.remove('.DS_Store')
   except:
       pass
   for stock in filelist:
       all_files = glob.glob('../sectors/'+str(sector)+'/'+str(stock)+'/'+str(stock)+'20*[0-9]**[0-9]*[1-4].*')
       all_files.sort(reverse=True)
       print(all_files)
```

```
returns:
['../sectors/automobiles/XPEL/XPEL20223.csv', '../sectors/automobiles/XPEL/XPEL20222.csv', '../sectors/automobiles/XPEL/XPEL20221.csv',
'../sectors/automobiles/XPEL/XPEL20214.csv', '../sectors/automobiles/XPEL/XPEL20213.csv', '../sectors/automobiles/XPEL/XPEL20212.csv',
'../sectors/automobiles/XPEL/XPEL20211.csv', '../sectors/automobiles/XPEL/XPEL20204.csv', '../sectors/automobiles/XPEL/XPEL20202.csv',
'../sectors/automobiles/XPEL/XPEL20201.csv', '../sectors/automobiles/XPEL/XPEL20194.csv', '../sectors/automobiles/XPEL/XPEL20193.csv',
'../sectors/automobiles/XPEL/XPEL20192.csv', '../sectors/automobiles/XPEL/XPEL20191.csv', '../sectors/automobiles/XPEL/XPEL20184.csv',
'../sectors/automobiles/XPEL/XPEL20183.csv', '../sectors/automobiles/XPEL/XPEL20182.csv', '../sectors/automobiles/XPEL/XPEL20181.csv',
'../sectors/automobiles/XPEL/XPEL20174.csv', '../sectors/automobiles/XPEL/XPEL20173.csv', '../sectors/automobiles/XPEL/XPEL20172.csv',
'../sectors/automobiles/XPEL/XPEL20171.csv', '../sectors/automobiles/XPEL/XPEL20164.csv', '../sectors/automobiles/XPEL/XPEL20163.csv',
'../sectors/automobiles/XPEL/XPEL20162.csv', '../sectors/automobiles/XPEL/XPEL20161.csv', '../sectors/automobiles/XPEL/XPEL20154.csv',
'../sectors/automobiles/XPEL/XPEL20153.csv', '../sectors/automobiles/XPEL/XPEL20152.csv', '../sectors/automobiles/XPEL/XPEL20151.csv']
```

Figure 23 - code and output for sorting each stock's directory into an ordered list

Using `import glob`, we can get an ordered list of what's in a particular's stock directory. For example, if we want to find the directory paths of the stock "XPEL", we can run the code in Figure 23 and find it. We can then loop around this list to access each individual earnings transcript.

Find_speaker_names.ipynb

```
def find_speaker_name(mytranscript, speaker_name_list):
   for sentence in mytranscript:
       # finds the speaker:
       colon_pos = sentence.find(":")
       speaker_name = sentence[:colon_pos]

       # appends speaker name to speaker_name_list:
       speaker_name_list.append(speaker_name)
```

Figure 24 - code for finding the speaker_name given a transcript

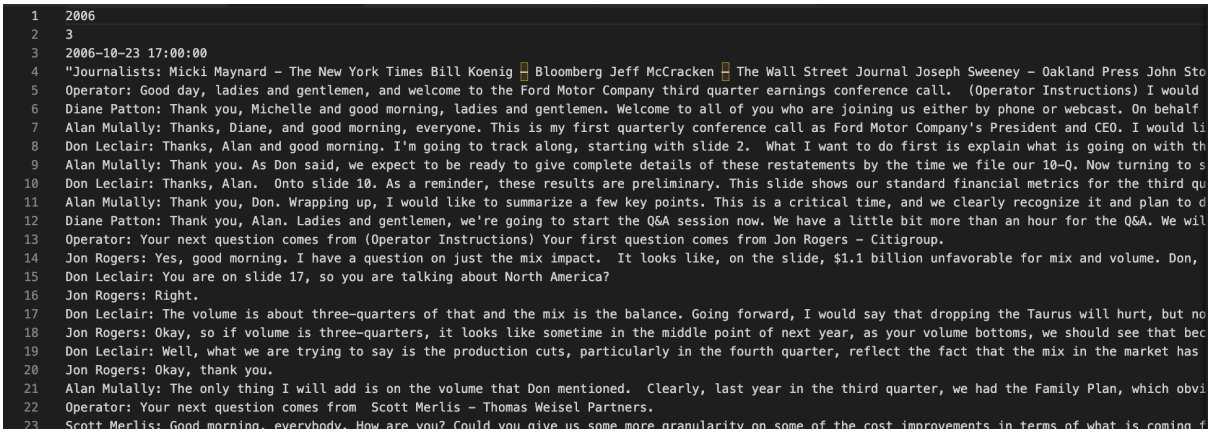


Figure 25 - Content snapshot of F20063.csv

Referencing figure 25, speaker names are separated by a colon, such as Diane Patton, Alan Mulally, Don Leclair etc.

The function `find_speaker_name(mytranscript, speaker_name_list)` takes in the variable `mytranscript` which holds the transcript data, and `speaker_name_list`, which starts off as an empty list (this function will append to it, then return).

```
def find_all_speaker_names(sector, stock):
    speaker_name_list = []
    sector_files = glob.glob('sectors/'+str(sector)+'/'+str(stock)+'/'+str(stock)+'20*[0-9]**[0-9]*[1-4].*')
    sector_files.sort(reverse=True)
    for path in sector_files: # for every path of the stock
        mytranscript = pd.read_csv(path).iloc[[2]].values[0][0]
        mytranscript = mytranscript.splitlines() # finds transcript
        find_speaker_name(mytranscript, speaker_name_list) # finds all speaker names
    speaker_name_list = list(set(speaker_name_list)) #removes all speaker names duplicates
    write_path = "sectors/"+sector+"/"+stock+"/"+"speaker names.csv"
    if not os.path.exists(write_path):
        np.savetxt(write_path, speaker_name_list, delimiter =", ", fmt ='% s')
```

Figure 26

Referencing Figure 26, when `find_all_speaker_names(sector, stock)` is called, it will cycle through all the .csv files (i.e. transcripts) of a single stock, then produce a new .csv file under the /[sector]/[stock] directory holding all the names of the speakers.

This is made possible using **glob** which allows us to cycle through all the files of a single stock directory, then extract the speaker names of each transcript, add them to the speaker_name_list for that particular stock.

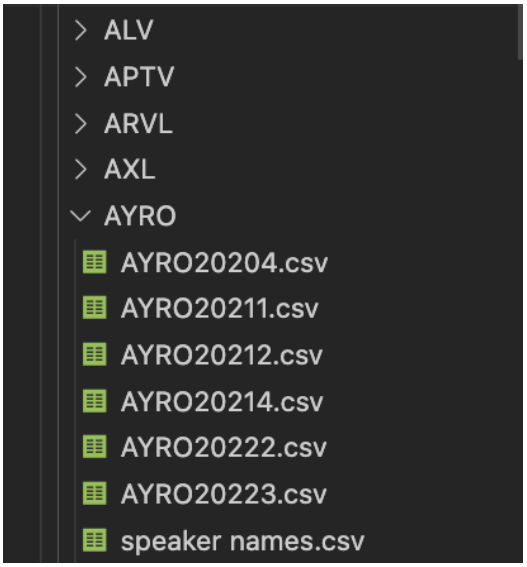


Figure 27a - AYRO's directory

Figure 27b - Content of AYRO's *speaker names.csv*

For example in figure 27a, speaker names.csv is created under the AYRO directory. And 27b shows the speaker names of AYRO, stored in speaker names.csv.

```
for sector in sectorlist:
   filelist = os.listdir("sectors/"+sector)
   try:
       filelist.remove('.DS_Store')
   except:
       pass
   for stock in filelist:
       find_all_speaker_names(sector, stock)
```

Figure 28 - Iterating through each sector and stock

Figure 28 essentially iterates through every sector, and every stock, then calling the function `find_all_speaker_names` so every stock has their own "speaker names.csv" file.

Dividing_transcript.ipynb

Dividing between safe harbour and Q&A:

```
def get_transcript(path):
   mytranscript = pd.read_csv(path).iloc[[2]].values[0][0]
   mytranscript = re.sub(r'[^A-Za-z0-9.,:!\'\n ]', '', mytranscript)
   mytranscript = re.sub('[^\S\n]+', ' ', mytranscript) #replaces multiple spaces to single space, without
deleting newlines \n in the process
   mytranscript = mytranscript.splitlines() # finds transcript
   return mytranscript
```

Figure 29a - Getting the contents of a transcript

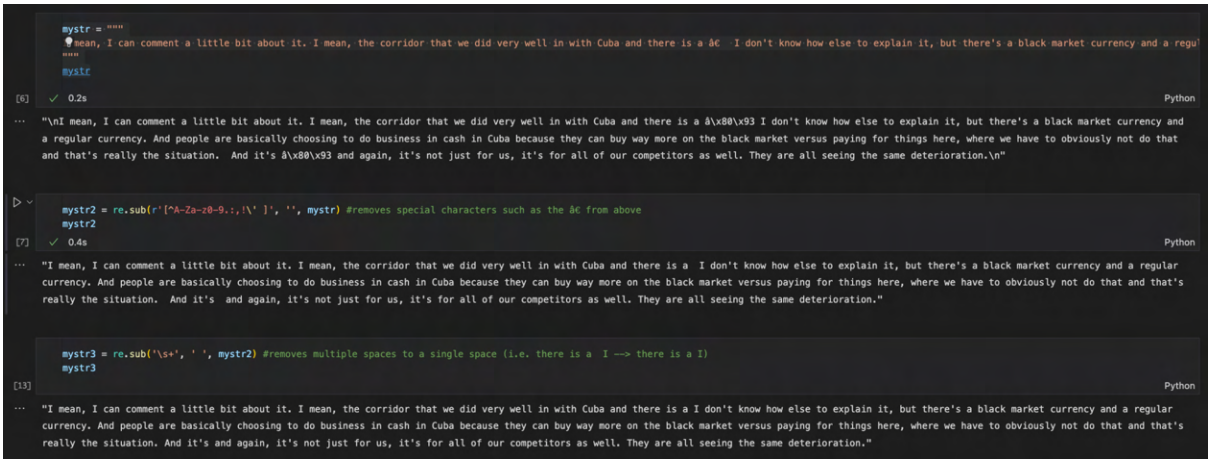


Figure 29b - cleaning the text using regex example

Using pandas read_csv function, we can read the transcript data from our .csv files. Furthermore, we can clean the function by running the command `mytranscript = re.sub(r'[^A-Za-z0-9.,:!\'\n ]', '', mytranscript)`. Using python's **regex** library, we can set the rules so that it deletes non-essential special characters. Essentially, it keeps the alphabets, numbers, and `.,:!\'\n` (which includes full stops, commas, colons, exclamation marks, apostrophes and line breaks). This would reduce the noise that special characters (e.g. åáâ, see Figure 29b) cause in later sentiment and semantical analysis. Furthermore, this function will replace words that are separated by multiple spaces to a single space (e.g. This is, This is). Mytranscript is then passed onto a splitlines function which identifies the line breaks and separates them accordingly into a list. This allows us to iterate through the "mytranscript list" and individually analyse each "speech bubble"/"paragraphs".

```
def split_transcript(mytranscript):
   transcript_safe_harbour, transcript_questions = "", ""
   for i in range(0, len(mytranscript)):
       speech_bubble = mytranscript[i].lower()
       speech_bubble = re.sub(r'[^\w\s:]', ' ', speech_bubble) # regex: replaces all punctuations (except for ":") with 1 open space so the IF condition below can run smoothly
       if (i > 1) and (("operator:" in speech_bubble) and (("question" in speech_bubble) or ("go ahead" in speech_bubble) or ("operator instructions" in speech_bubble))):
           transcript_safe_harbour = mytranscript[0:i]
           transcript_questions = mytranscript[i:]
           break
       elif (i > 1 ) and ("operator" in speech_bubble) and ("question" in speech_bubble):
           transcript_safe_harbour = mytranscript[0:i+1]
           transcript_questions = mytranscript[i+1:]
           break
       elif (i > 1 ) and ("operator:" in speech_bubble) and ("first" in speech_bubble):
           transcript_safe_harbour = mytranscript[0:i]
           transcript_questions = mytranscript[i:]
           break
   return transcript_safe_harbour, transcript_questions
```

Figure 30 - function for splitting transcript into safe harbour and Q&A

As mentioned previously in the **Design** Section, earnings calls are separated by a safe harbour section and a Q&A section. To divide these, there are a few universal conditions that are used to identify the *"point of separation"*.

Given the transcript's content (after it has been formatted using `get_transcript`), the programme will iterate through each "speech bubble" of the transcript and checks whether it meets certain conditions (highlighted in orange in Figure 30). When these conditions are met, it will identify which index it represents and assign `transcript_safe_harbour` and `transcript_questions` accordingly.

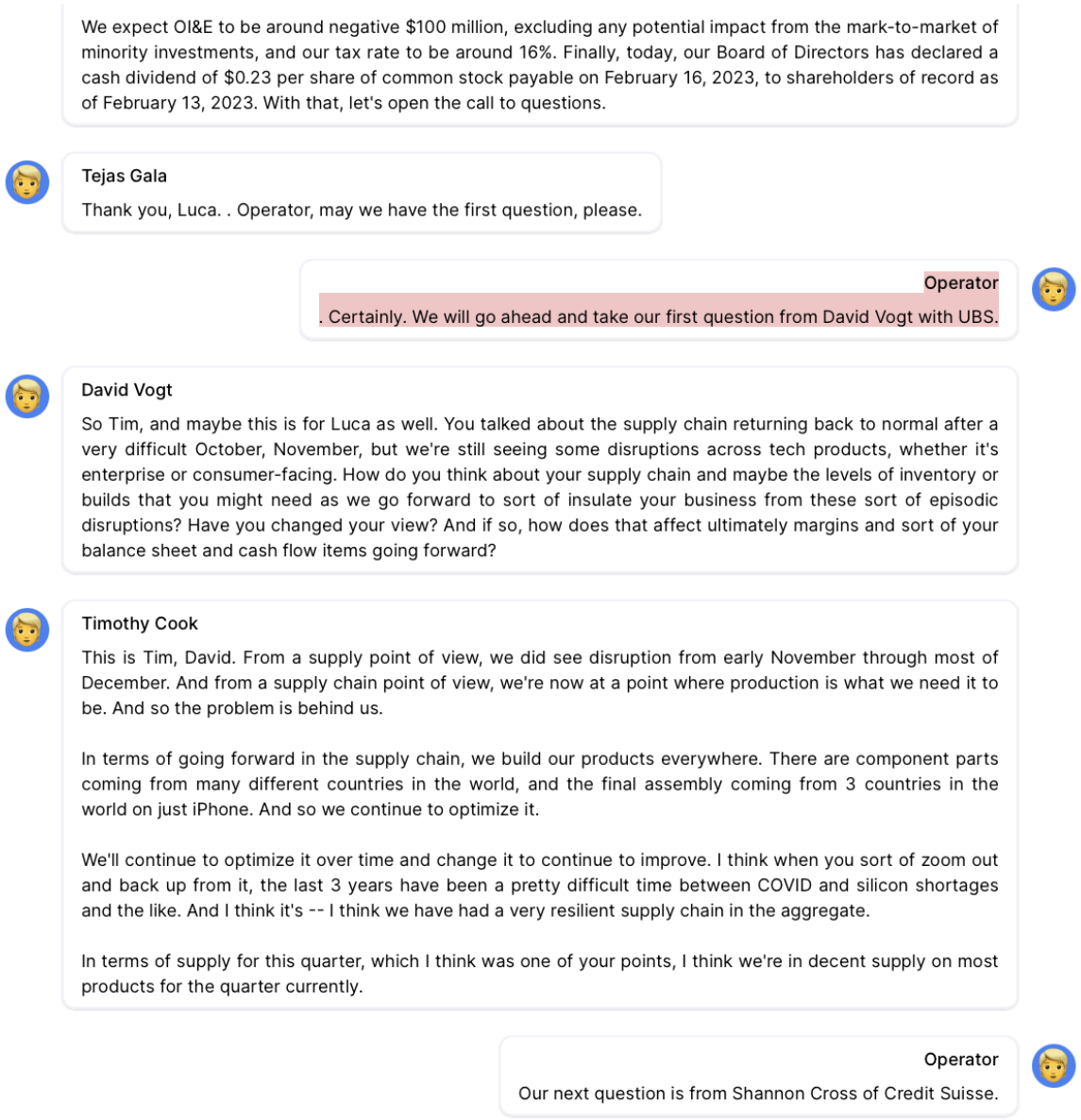


Figure 31 - An example of the "point of separation" for Apple 2023 Q1

With reference to Figure 31, we can see that the point of separation is when the speech bubble is said by the "operator", and looking into this speech bubble, "first question", and "go ahead" is also mentioned. Furthermore, the index of this speech

bubble is greater than one, so we can safely say that this is the point of separation between the safe harbour and Q&A sections.

Dividing between analyst questions and management replies:

As previously mentioned in the **Design Section**: In the Q&A section, analysts ask a question and management provides them with a reply, hence, we can classify each Q&A speech bubble between "**Questions**" and "**Answers**". We can then aggregate each class and perform separate semantical analyses (e.g. find the average sentiment value for analyst questions).

Specifically, there are two universal conditions that will help us identify who is the analyst:

```
If said by operator
AND
If the speaker name is in the speech bubble
```

The first condition is simple, and we can do this easily: `if "operator:" in speech_bubble`

```
def get_file_speaker_names(sector, stock):
   write_path = "sectors/"+sector+"/"+stock+"/"+"speaker names.csv"
   speaker_names = np.loadtxt(write_path, delimiter='\n', dtype=str)
   return speaker_names
```

Figure 33 - Function which reads the speaker names .csv files

For the second condition, we must get a list of speaker names. With reference to Figure 33, the `get_file_speaker_names(sector, stock)` function will read the speaker names.csv file and return it as a list, containing all the possible speaker names.

However, checking whether a speaker's name is in this speech bubble is a bit more nuanced.

```
'Operator: Okay, thank you sir. And that will come from Michael
Rollins with Citi. Please go ahead.

Mike Rollins: Thanks for taking the questions. Two, if I could.
First, as we think about your move to consumption-based pricing
and we also think about some of the changes in the sizes of the
buckets for data, can you give us a sense of what's happening as
```

```
you increase the data buckets, you try to encourage customers to
spend more on data?
```

Figure 34 - The problem

As shown in figure 34, the operator mentions Michael Rollins, but the speaker's name is registered as Mike Rollins. So, if we use the command `if myname in sentence:` it will return False. This is because the string “Mike Rollins” is not in the string “Michael Rollins”.

This highlights the importance of using fuzzy string matching or other techniques to account for variations in names and ensure the accurate identification of speakers in conversation analysis.

```
# Solution:
speech_bubble = "The next question is from Michael Rollins"
speaker_name = "Mike Rollins"
myspeaker_name_list = speaker_name.split()
if (myspeaker_name_list[0] in speech_bubble) or (myspeaker_name_list[-1] in speech_bubble):
   print(speaker_name)
```

Figure 35 - Solution to the problem

With reference to Figure 35, a simplified method is to split the speaker’s name into a list, then compared each name inside the list with whether it’s in the speech_bubble.

```
# finds a list of analyst names for a single .csv file
def find_analyst_names(speaker_names, transcript_questions):
   analyst_names = []
   # the programme recognises the question is being asked by an analyst when the following conditions are met
   for index in range(0, len(transcript_questions)-2):
       speech_bubble = transcript_questions[index].lower()
       speech_bubble = re.sub(r'[^\w\s:]', ' ', speech_bubble)
       if "operator:" in speech_bubble:
           for name in speaker_names:
               namelist = name.split()
               if name.lower() != "operator":
                   for name_2 in namelist: # cycle through each name in the name_list
                       name_2 = name_2.lower()
                       if (name_2 in speech_bubble) and len(name_2) > 2:
                           analyst_names.append(name)
                   if "unidentified" in name.lower().split(): # finds name such as "Unidentified Analyst"
                       analyst_names.append(name)

   analyst_names = list(set(analyst_names)) # replaces duplicates
   return analyst_names
```

Figure 36 - Function for finding Analyst names

Hence, with all of the above in mind, we can write a function that finds the names of all the analysts. First, it splits the Q&A section into speech bubbles, then it checks whether the conditions are met. I also included an additional condition that ignores any abbreviated initial from the name (for example, the `A.` in `Gayn A. Erickson`).

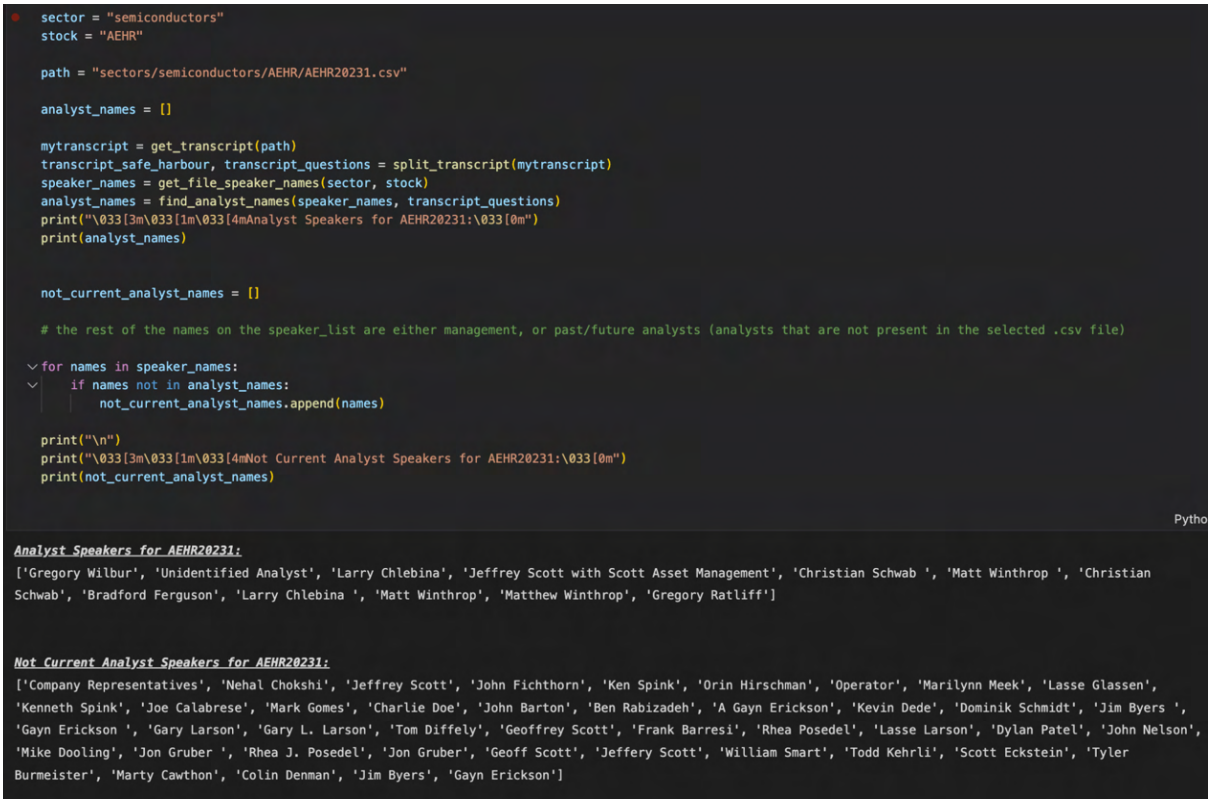


Figure 37 - code to get analyst and management names

Figure 37 shows how to get a list of analyst speakers and another list of management speakers for the stock AEHR.

```
def get_analyst_management_sentences(analyst_names, transcript_questions):
    analyst_sentences = []
    management_sentences = []

    for index in range(0, len(transcript_questions)-2):
        speech_bubble = transcript_questions[index]
        colon_pos = speech_bubble.find(":")
        speaker_name = speech_bubble[:colon_pos]
        if speaker_name in analyst_names:
            print("analyst:", speaker_name)
            analyst_sentences.append(speech_bubble)

        elif speaker_name.lower() == "operator":
            print("operator:", speaker_name)

        else:
            print("management:", speaker_name)
            management_sentences.append(speech_bubble)

    return analyst_sentences, management_sentences
```

Figure 38 - Code for getting analyst and management sentences

Now that we know who is an analyst, and who is a part of the management, we can find the analyst and management sentences as shown in Figure 38.

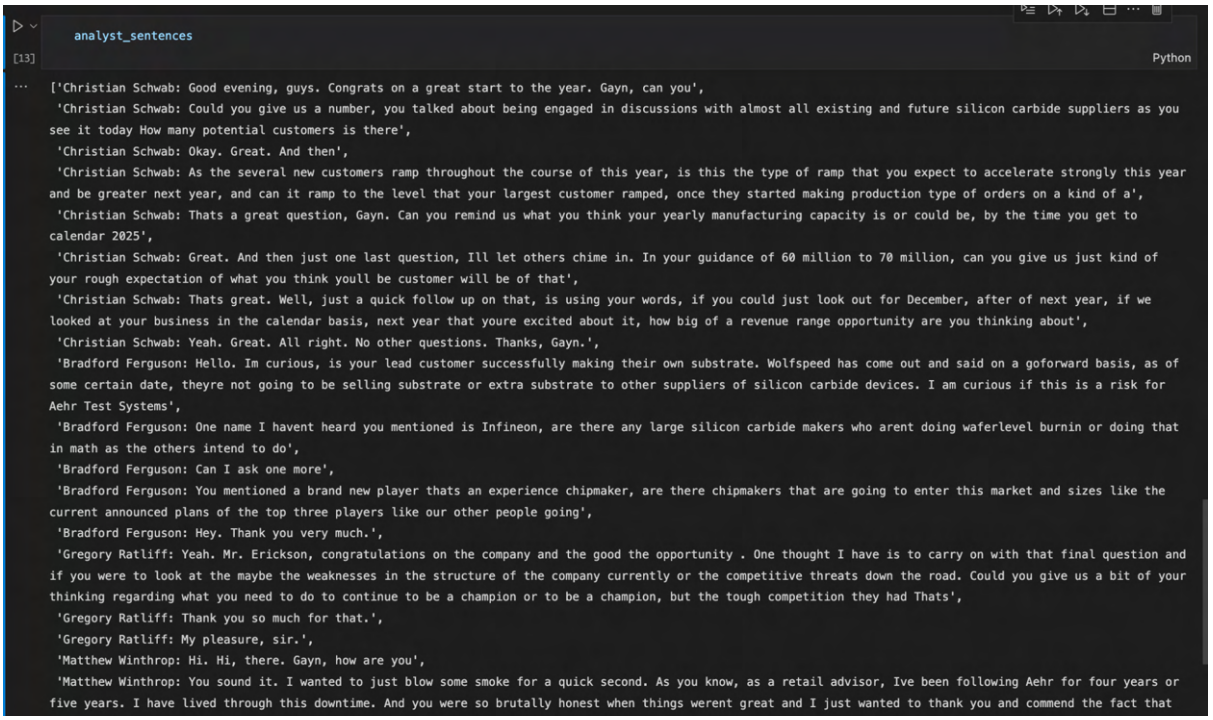


Figure 39 - An example of getting analyst sentences

Figure 40 - An example of getting management sentences

Feature Extracting the EPS (index 4):

feature_extract_EPS.ipynb:

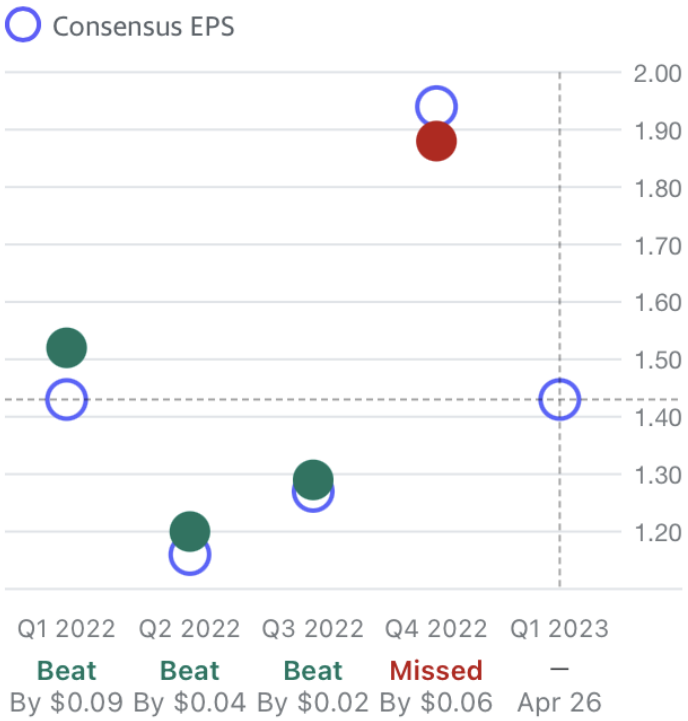


Figure 41 - Apple's Earnings Estimates

Earnings per share (EPS) is a metric used to indicate what portion of a company's profit is allocated to each outstanding share of its common stock. Analysts make estimates of the anticipated EPS value prior to the company's release of its earnings results. Hence, analysts can compare the estimated and actual EPS values and determine whether the company exceeded or fell short of expectations.

Figure 41 is an example of Apple's consensus actual vs predicted EPS. As we can see, Apple beat expectations in Q1 Q2 and Q3, but fell short in Q4. In my programme, I used a percentage figure to represent the magnitude of beat/miss instead of subtracting it to normalise different EPS scales of different companies.

I have written two different methods for obtaining the EPS data. The first was my original code that is designed to web scrape and parses data from https://www.marketbeat.com. The second approach was using the yFinance python API to obtain the EPS value. I ended up running the second method, as this approach provided structured pandas dataframes directly when called, allowing for easier data manipulation. Furthermore, yFinance also offers a longer timeframe of data, with some records dating back to 1998, in comparison, the data available on

https://www.marketbeat.com can sometimes be limited to 2010 or 2014 for the same stocks.

```
def append_row_toCSV(path, toadd_data):
    mydata = pd.read_csv(path)
    newdata = pd.DataFrame([toadd_data], columns=mydata.columns)
    mydata = pd.concat([mydata, newdata], ignore_index=True)
    mydata.to_csv(path, index=False)
```

Figure 42 - Appends a row to a CSV file, given the path

After obtaining the relevant EPS data, the function append_row_toCSV will be run to add the EPS data onto the corresponding earnings transcript.

Getting EPS data (1st method - marketbeat):

```
# finds earnings estimate and and EPS
def add_EPS_estimate_reported(url):
    html_text = requests.get(url).text
    soup = BeautifulSoup(html_text, "html.parser")
    body = soup.find("table", class_="scroll-table sort-table", id="earnings-history")

    body = body.find("tbody").find_all("tr")
    body.pop(0)
    mylist = []
    for value in body:
        while True:
            date_EPS = value.find("td")
            consensus_EPS = date_EPS.find_next("td").find_next("td")
            reported_EPS = consensus_EPS.find_next("td")
            if str(date_EPS.contents[0])[0] == "<":
                break
            date_EPS = date_EPS.contents
            consensus_EPS = consensus_EPS.contents
            reported_EPS = reported_EPS.contents
            mylist2 = [date_EPS,consensus_EPS,reported_EPS]
            mylist.append(mylist2)
            break
    return mylist

def check_day_diff(mydatestr, quarterURL):
    date_1 = datetime.strptime(mydatestr, f'%m/%d/%Y')

    myurl = pd.read_csv(quarterURL)
    date_string = myurl.loc[1][0]
    date_2 = datetime.strptime(date_string, f'%Y-%m-%d %H:%M:%S')

    day_difference = abs((date_2 - date_1).days)
    if day_difference < 5:
        return True
    else:
        return False
```

Figure 43 - Get EPS Method 1

JSON and HTML parsing techniques are used to extract EPS data from marketbeat.com. With reference to `add_EPS_estimate_reported(url)`, the code finds all <td> tags and iterates through them using the function `.find_next`. It will find the **release date of the EPS**, **consensus estimation for the EPS**, and the **reported result of the EPS.** This information will be added to a multi-dimensional list structure for later use (i.e. code in Figure 45).

In addition, the `check_day_diff(mydatestr, quarterURL)` function verifies whether the EPS release date aligns with the earnings call release date by checking if the difference between the two falls within a range of ±5. This margin of error is used because the release dates may differ across various websites.

```
    url = 'https://www.marketbeat.com/stocks/NASDAQ/'+"MSFT"+'/earnings/'
    html_text = requests.get(url).text
    soup = BeautifulSoup(html_text, "html.parser")
    body = soup.find("table", class_="scroll-table sort-table", id="earnings-history")

    body = body.find("tbody").find_all("tr")
    body.pop(0)
    mylist = []
    body
    for value in body:
        while True:
            date_EPS = value.find("td")
            consensus_EPS = date_EPS.find_next("td").find_next("td")
            reported_EPS = consensus_EPS.find_next("td")
            if str(date_EPS.contents[0])[0] == "<":
                break
            date_EPS = date_EPS.contents
            consensus_EPS = consensus_EPS.contents
            reported_EPS = reported_EPS.contents
            mylist2 = [date_EPS,consensus_EPS,reported_EPS]
            mylist.append(mylist2)
            break

    mylist
[76]  ✓ 1.1s
...  [[['1/24/2023'], ['$2.27'], ['$2.32']],
      [['10/25/2022'], ['$2.29'], ['$2.35']],
      [['7/26/2022'], ['$2.28'], ['$2.23']],
      [['4/26/2022'], ['$2.18'], ['$2.22']],
      [['1/25/2022'], ['$2.29'], ['$2.48']],
      [['10/25/2021'], ['$2.08'], ['$2.27']],
      [['7/26/2021'], ['$1.92'], ['$2.17']],
      [['4/27/2021'], ['$1.76'], ['$1.95']],
      [['1/25/2021'], ['$1.64'], ['$2.03']],
      [['10/27/2020'], ['$1.53'], ['$1.82']],
      [['7/22/2020'], ['$1.34'], ['$1.46']],
      [['4/29/2020'], ['$1.27'], ['$1.40']],
      [['1/29/2020'], ['$1.32'], ['$1.51']],
      [['10/23/2019'], ['$1.24'], ['$1.38']],
      [['7/18/2019'], ['$1.21'], ['$1.37']],
      [['4/24/2019'], ['$1.00'], ['$1.1375']],
```

Figure 44 - Example of the output for `add_EPS_estimate_reported(url)`

Figure 44 shows the output when querying Microsoft's historical EPS data. The output is a multi-dimensioned structured list such that it's [date, consensus_EPS, reported_EPS].

```
# sorts directory into a list, then add consensus and reported EPS to each stock's csv files

for sector in sectorlist:

   filelist = os.listdir("sectors/"+sector)
   try:
       filelist.remove('.DS_Store')
   except:
       pass
   for stock in filelist:
       all_files =
glob.glob('sectors/'+str(sector)+'/'+str(stock)+'/'+str(stock)+'20*[0-9]**[0-9]*[1-4].*')
       all_files.sort(reverse=True) #gives a SORTED list of all .csv paths of that particular stock
       url = 'https://www.marketbeat.com/stocks/NASDAQ/'+stock+'/earnings/'
       eps_list = add_EPS_estimate_reported(url)
       i=0
       try:
           for quarterURL in all_files: #loops around each .csv path
               date = ''.join(eps_list[i][0])
               #checks if the data on consensus/reported EPS match the quarterly report one
               mybool = check_day_diff(date, quarterURL)
               #gets the consensus data, turning it into a string in the format i.e. $0.48 then getting
rid of the $ sign --> to get 0.48
               consensus_EPS = ''.join(eps_list[i][1]).replace("$","")
               reported_EPS = ''.join(eps_list[i][2]).replace("$","")

               percentage = (float(reported_EPS)/float(consensus_EPS))-1
               print(percentage)

               if mybool == True:
                   path = quarterURL
                   append_row_toCSV(path, percentage) #adds percentage of surprise to csv file
               i+=1
       except:
           pass
       break
   break
```

Figure 45 - Iterating through each stocks and appending the EPS value on their corresponding earnings transcript.csv

As figure 45 shows, it will iterate through seach sector and stock, then find their corresponding EPS values by running the code in figure 43. It will use the output of Figure 44 to determine the percentage beat/miss of the EPS value, then append it to the relevant .csv file.

Getting EPS data (2nd method - yfinance):

The second method uses the yFinance API library in python.

```
import yfinance as yf


msft = yf.Ticker("msft")
earnings_list = msft.get_earnings_dates(limit=200)
earnings_list.reset_index(inplace=True)
earnings_list = earnings_list[earnings_list['Surprise(%)'].notna()]
eps_list = earnings_list.reset_index(drop=True)

eps_list
```

✓ 3.8s

	Earnings Date	EPS Estimate	Reported EPS	Surprise(%)
0	2023-01-24 11:00:00-05:00	2.30	2.32	0.0109
1	2022-10-25 12:00:00-04:00	2.30	2.35	0.0204
2	2022-07-26 12:00:00-04:00	2.29	2.23	-0.0275
3	2022-04-26 12:00:00-04:00	2.19	2.22	0.0160
4	2022-01-25 11:00:00-05:00	2.31	2.48	0.0727
...	...	...	...	...
116	1994-01-19 00:00:00-05:00	0.03	0.03	0.0189
117	1993-10-20 00:00:00-04:00	0.02	0.02	0.0181
118	1993-07-28 00:00:00-04:00	0.03	0.03	0.0172
119	1993-04-14 00:00:00-04:00	0.02	0.03	0.0196
120	1993-01-16 00:00:00-05:00	0.02	0.02	-0.0041

121 rows × 4 columns

Figure 46 - Output result for MSFT's EPS data

With reference to Figure 46, yFinance provides data going back to Jan, 1993, whereas marketbeat is restricted to March, 2015 for Microsoft.

Sometimes, it may not be possible to make a yFinance API call for a certain stock at a given time. In such cases, the stock will be added to the `not_found_stock.txt` text document so that it can be attempted again at a later time when the API is available.

Throughout this programme, note that **date_csv** represents the release date of the earnings transcript, and **date_yfinance** represents the release date of the EPS.

```
#Adjusted for Yfinance
#check if the difference between the two dates are <5
def check_day_diff(date_yfinance,QuarterURL):
   myurl = pd.read_csv(QuarterURL)
   date_string = myurl.loc[1][0]
   date_csv = datetime.strptime(date_string, f'%Y-%m-%d %H:%M:%S')
   day_difference = (date_csv - date_yfinance).days

   if abs(day_difference) < 5:
       return "match"

   elif day_difference > 0:
       return "case 2"

   elif day_difference < 0:
```

```
        return "case 1"
```

Figure 47 - check_day_diff function, adjusted for yfinance

With reference to the contents under "***Matching Algorithm between EPS date and Earnings Transcript date***" of the Design section, there are three scenarios that may arise:

1. Match
2. Case 1 (see **Matching Algorithm between EPS date and Earnings Transcript date section**)
3. Case 2 (see **Matching Algorithm between EPS date and Earnings Transcript date section**)

```
def get_EPS(stock, sector):
   all_files = glob.glob('sectors/'+str(sector)+'/'+str(stock)+'/'+str(stock)+'20*[0-9]**[0-9]*[1-4].*')
   all_files.sort(reverse=True) # a reverse-sorted list of all .csv paths of that particular stock
   try: # first exception handling checks if the yfinance API is able to retrieve the data for this
particular stock, if not, see the exception handling below
       yf_stock = yf.Ticker(stock)
       earnings_list = yf_stock.get_earnings_dates(limit=200) #uses yfinance and gets historical eps
consensus/reported data
       earnings_list.reset_index(inplace=True)
       earnings_list = earnings_list[earnings_list['Surprise(%)'].notna()]
       eps_list = earnings_list.reset_index(drop=True)
   except: # if exception is thrown, yfinance couldn't find the data, it might be because of a temporary
API error so the stock ticker symbol is stored in the file not_found_stock.txt, so it could be ran again
later
       with open("not_found_list_stocks.txt", "a") as file_object:
           file_object.write(sector)
           file_object.write("\n")
           file_object.write(stock)
           file_object.write("\n")
       return
 # sets the index_counters of the relevant lists
   index_csv = 0
   index_yfinance= 0
   error_count = 0
   try: # second exception handling, see below for explanation
       while index_csv < len(all_files): # while the current csv index is below the number of all csvs -->
then it loops around each csv path
           # gets the path of the csv file
           path = all_files[index_csv]

           # finds the EPS % surprise by referencing the index_yfinance variable
           percentage_surprise = float(eps_list['Surprise(%)'][index_yfinance])
           # finds which date the EPS data is added on yfinance
           date_yfinance = eps_list['Earnings Date'][index_yfinance]

           # fixes the "Cannot subtract tz-naive and tz-aware datetime-like objects" error
           date_yfinance = date_yfinance.replace(tzinfo=None)

           # this function checks if the yfinance date matches the date on the transcript.csv file
```

```
            mycase = check_day_diff(date_yfinance, path)

            # reads the csv file and returns it as a pandas dataframe
            read_csv = pd.read_csv(path)

            # if the function check_day_diff returns “match”, in other words, if the date on yfinance
matches the date on the csv file...
            if mycase == "match":
                # furthermore if the csv file dataframe has more than 4 rows (more than 3 index), the
dataframe will delete every value beyond the 4th row
                # this is because:
                # index 0 is reserved for the year,
                # index 1 is reserved for the quarter,
                # index 2 is reserved for the date in which the earnings transcript is reported
                # index 3 is reserved for the content of the earnings transcript
                # index 4 is reserved for the EPS surprise value (the value we are trying to get)
                # there is a chance that index 4 returns NaN (see explanation for 1st exception handling)
                # so when the function is "re-ran" with reference to the not_found_stock.txt file, the new
value of the EPS data does not "fill" the spaces for index 5 and beyond
                # so it's safe to wipe everything beyond the 3rd index, and then add the EPS value

                if len(read_csv) > 3:
                    read_csv = read_csv[:3]
                index_csv += 1
                index_yfinance += 1

                # EPS surprise is printed on index 4
                newdata = pd.DataFrame([percentage_surprise], columns=read_csv.columns)
                read_csv = pd.concat([read_csv, newdata], ignore_index=True)
                read_csv.to_csv(path, index=False)

            elif mycase == "case 2":
                # in the event of "case 2", the date on the csv is bigger than the date on yfinance
                # so we need the index_counter of the transcripts to decrease while the index_counter of the
EPS stays the same, until they match
                # to decrease date_csv, the index_counter of the transcript will have to increment since the
list is in a descending order
                index_csv += 1
                if len(read_csv) > 3:
                    read_csv = read_csv[:3]
                newdata = pd.DataFrame([np.NaN], columns=read_csv.columns)
                read_csv = pd.concat([read_csv, newdata], ignore_index=True)
                read_csv.to_csv(path, index=False)

            elif mycase == "case 1":
                index_yfinance += 1

        # the second exception is thrown when the below condition's index is out of bounds
        # percentage_surprise = float(eps_list['Surprise(%)'][index_yfinance])
        # the reason that it's out of bound is because the length of the data yfinance provides is smaller than
the number of transcript.csv files
        # for example for the stock XPEL, yfinance API returns the following:
        #                    Earnings Date  EPS Estimate  Reported EPS  Surprise(%)
        # 0  2022-11-09 03:00:00-05:00          0.44          0.48       0.0909
        # 1  2022-08-09 04:00:00-04:00          0.34          0.43       0.2760
        # 2  2022-05-10 04:00:00-04:00          0.24          0.28       0.1814
```

```
   # 3  2022-02-28 02:00:00-05:00          0.30          0.22      -0.2763
   # 4  2021-11-09 03:00:00-05:00          0.32          0.32       0.0127
   # 5  2021-08-09 04:00:00-04:00          0.26          0.37       0.4453
   # 6  2021-05-10 03:00:00-04:00          0.18          0.25       0.4286
   # 7  2021-03-11 02:00:00-05:00          0.20          0.22       0.1282
   # 8  2020-11-10 12:00:00-05:00          0.17          0.24       0.4118
   # 9  2020-08-12 03:00:00-04:00          0.05          0.14       1.8000
   # 10 2020-05-14 04:00:00-04:00          0.06          0.06       0.0909
   # 11 2020-03-16 04:00:00-04:00          0.10          0.17       0.7000
   # 12 2019-11-08 11:00:00-05:00          0.11          0.16       0.4545
   #
   #
   # however, the csv file (transcript data) contains data between 2022_Q3 to 2015_Q1
   # in this case, the exception is thrown.
   # When the exception is thrown, the rest of the csv files will append the value "NaN" to make sure index
4 of the csv file is filled with a value.

   except:
       rest_of_files = all_files[index_csv:]
       for path in rest_of_files:
           read_csv = pd.read_csv(path)
           if len(read_csv) > 3:
                   read_csv = read_csv[:3]
           newdata = pd.DataFrame([np.NaN], columns=read_csv.columns)
           read_csv = pd.concat([read_csv, newdata], ignore_index=True)
           read_csv.to_csv(path, index=False)
```

Figure 48 - Writes EPS data to csv files, adjusted for finance

```
#Adjusted for Yfinance
for sector in sectorlist:
   print(sector, "sector start ..............................................................")
   filelist = os.listdir("sectors/"+sector)
   try:
       filelist.remove('.DS_Store')
   except:
       pass
   for stock in filelist:
       print(stock, "start")
       get_EPS(stock, sector) #calls get_EPS function and notes EPS info on the relevant .csv files
       print(stock, "end")
   break
```

Figure 49 - Iterates through every sector, stock and runs the get_EPS() function

Going through not_found_list_stocks.txt:

```
def modified_get_EPS(stock, sector):
   all_files =
glob.glob('sectors/'+str(sector)+'/'+str(stock)+'/'+str(stock)+'20*[0-9]**[0-9]*[1-4].*'
)
   all_files.sort(reverse=True)
   try:
```

```
        yf_stock = yf.Ticker(stock)
        earnings_list = yf_stock.get_earnings_dates(limit=200) #uses yfinance and gets
historical eps consensus/reported data
        earnings_list.reset_index(inplace=True)
        earnings_list = earnings_list[earnings_list['Surprise(%)'].notna()]
        eps_list = earnings_list.reset_index(drop=True)
    except:
        print("ERROR", stock, "NOT FOUND!!!!!!")
        return False
    index_csv = 0
    index_yfinance= 0
    error_count = 0
    try:
        while index_csv < len(all_files):
            path = all_files[index_csv]
            percentage_surprise = float(eps_list['Surprise(%)'][index_yfinance])
            date_yfinance = eps_list['Earnings Date'][index_yfinance]
            date_yfinance = date_yfinance.replace(tzinfo=None)
            mycase = check_day_diff(date_yfinance, path)
            read_csv = pd.read_csv(path)

            if mycase == "match":
                if len(read_csv) > 3:
                    read_csv = read_csv[:3]
                index_csv += 1
                index_yfinance += 1
                newdata = pd.DataFrame([percentage_surprise], columns=read_csv.columns)
                read_csv = pd.concat([read_csv, newdata], ignore_index=True)
                read_csv.to_csv(path, index=False)

            elif mycase == "case 2":
                index_csv += 1
                if len(read_csv) > 3:
                    read_csv = read_csv[:3]
                newdata = pd.DataFrame([np.NaN], columns=read_csv.columns)
                read_csv = pd.concat([read_csv, newdata], ignore_index=True)
                read_csv.to_csv(path, index=False)

            elif mycase == "case 1":
                index_yfinance += 1
    except:
        rest_of_files = all_files[index_csv:]
        for path in rest_of_files:
            read_csv = pd.read_csv(path)
            if len(read_csv) > 3:
                    read_csv = read_csv[:3]
            newdata = pd.DataFrame([np.NaN], columns=read_csv.columns)
            read_csv = pd.concat([read_csv, newdata], ignore_index=True)
            read_csv.to_csv(path, index=False)

    return True
```

Figure 50 - Slightly modified get_EPS function

With reference to Figure 50, the function goes through each stock in the not_found_list_stocks.txt and fetch their EPS data using yfinance again, if the fetch

was successful, that particular stock will be erased in not_found_list_stocks.txt. Then it will try again.

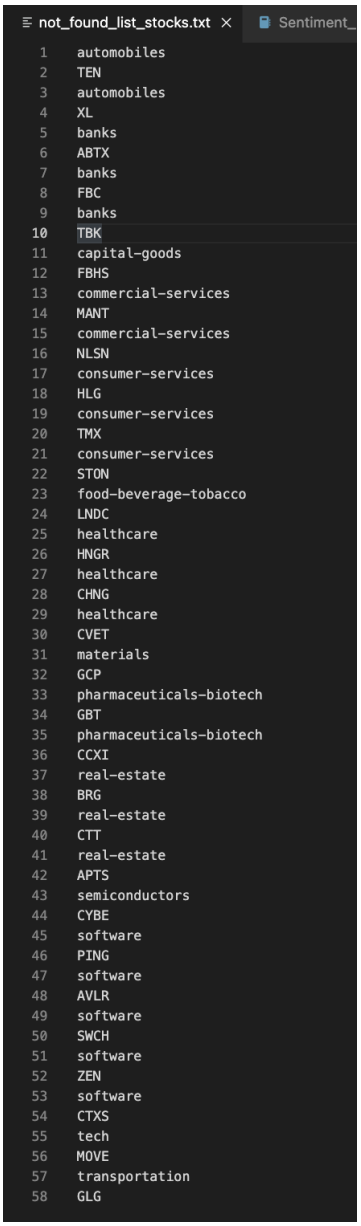


Figure 51 - leftovers of not_found_list_stocks.txt file after multiple loops

FinBERT Sentiment and Classification:

Sentiment_FLS_classify.ipynb:

Figure 52 - Installing the transformers library to access FinBERT machine learning models

With reference to the section **"Summary of the features extracted from each transcript"** in the Design section, a large majority of the features include processing the sentiment value of particular sentences, as well as determining whether they should be considered "forward-looking statements".

Sentiment Analysis:

```
from transformers import BertTokenizer, BertForSequenceClassification, pipeline

sentiment_finbert =
BertForSequenceClassification.from_pretrained('yiyanghkust/finbert-tone',num_labels=3)
sentiment_tokenizer = BertTokenizer.from_pretrained('yiyanghkust/finbert-tone')
```

Figure 53 - Installing FinBERT machine learning libraries

FinBERT (Financial Bidirectional Encoder Representations from Transformers) is a pre-trained deep learning model that can analyze financial text data and provide sentiment analysis. The model is based on the Transformer architecture and is pre-trained on a large corpus of financial text data, allowing it to accurately classify financial sentiment in new text data. I decided to use this pre-trained model as they have already been proven to be highly accurate and effective since they have been developed and fine-tuned by experienced researchers and practitioners in the field, and have been tested on large and diverse datasets to ensure their accuracy and robustness. By using a battle-tested pre-trained model, I can have greater confidence in the results and reduce the risk of errors or biases that could arise from developing my own model.

```
mystr = """
I mean, I can comment a little bit about it. I mean, the corridor that we did very well
in with Cuba and there is a I don't know how else to explain it, but there's a black
market currency and a regular currency. And people are basically choosing to do business
in cash in Cuba because they can buy way more on the black market versus paying for
things here, where we have to obviously not do that and that's really the situation. And
it's and again, it's not just for us, it's for all of our competitors as well. They are
all seeing the same deterioration."
"""
sentiment_nlp = pipeline("text-classification", model=sentiment_finbert,
tokenizer=sentiment_tokenizer)
results = sentiment_nlp(mystr3)

--------------------------------------------------------------------------------
-
Output:
[{'label': 'Neutral', 'score': 0.9691150188446045}]
```

Figure 54 - Example of getting the sentiment value for “mystr” using the FinBERT pre-trained model

With reference to Figure 54, it classified “mystr” as Neutral, with a confidence level of 96.9%. However, we can improve this by breaking the paragraphs down into individual sentences (using the tokenising technique).

```
nltkstr = "Marcelo Fischer : Just to correct the income from operations, the loss from
operations for net2phone was minus $1.8 billion. Of EBITDA itself was about minus
$100,000, okay? So we are getting closer to reaching EBITDA profitability and based on
our 2023 projections, we do hope to exit 2023 with net2phone being an EBITDA-positive
company, even as it continues to grow at rates which are probably higher than the
average UCaaS play on basically."

nltkstr= (nltk_tokenizer.tokenize(nltkstr))
sentiment_results = sentiment_nlp(nltkstr)

--------------------------------------------------------------------------------
-
Output:
['Marcelo Fischer : Just to correct the income from operations, the loss from operations
for net2phone was minus $1.8 billion.',
 'Of EBITDA itself was about minus $100,000, okay?',
 'So we are getting closer to reaching EBITDA profitability and based on our 2023
projections, we do hope to exit 2023 with net2phone being an EBITDA-positive company,
even as it continues to grow at rates which are probably higher than the average UCaaS
play on basically.']

[{'label': 'Negative', 'score': 0.5338950753211975},
 {'label': 'Neutral', 'score': 0.999138355255127},
 {'label': 'Positive', 'score': 0.9999885559082031}]
```

Figure 55 - Getting the sentiment value after tokenising paragraphs

By tokenizing paragraphs into sentences, we can isolate individual units of text and analyze their sentiment independently, rather than treating the entire paragraph as a

single unit. This is important because the sentiment of a paragraph can be nuanced and include both positive and negative sentiment, and analyzing the paragraph as a whole may obscure or dilute these opposing sentiments, generalising it as neutral. By tokenizing the paragraph into individual sentences, we can analyze the sentiment of each sentence independently, hence get a more granular understanding of the overall sentiment. For example, in Figure 55, we tokenise the original string into 3 sentences, then gather the sentiment value for each of them as the output.

Forward Looking Statement (FLS) Classification:

```
# FLS classification
fls_finbert =
BertForSequenceClassification.from_pretrained('yiyanghkust/finbert-fls',num_labels=3)
fls_tokenizer = BertTokenizer.from_pretrained('yiyanghkust/finbert-fls')

fls_nlp = pipeline("text-classification", model=fls_finbert, tokenizer=fls_tokenizer)
fls_results = fls_nlp(nltkstr)

--------------------------------------------------------------------------------
-
Output:
[{'label': 'Not FLS', 'score': 0.9779328107833862},
 {'label': 'Not FLS', 'score': 0.9796538949012756},
 {'label': 'Specific FLS', 'score': 0.8796855211257935}]
```

Figure 56 - FLS Classification

We can also classify each of the sentence based on whether they are a forward looking statement or not. Forward-looking statements (FLS) inform investors of managers' beliefs and opinions about firm's future events or results. Identifying forward-looking statements from corporate reports can help us evaluate what their future prospects are. As mentioned prior:

Specific forward-looking statements (S_FLS) refer to statements that involve a precise estimation of future events (e.g. expected revenue).

Non-specific forward-looking statements (NS_FLS), on the other hand, are more general in nature and do not involve specific figures or estimates. They may include statements regarding future plans or strategies, but without committing to a precise outcome.

Not forward-looking statements (N_FLS) are statements that do not estimate future events or projections, but instead, they describe the current state of affairs or historical data.

In my programe, I mapped FLS classification with sentiment analysis to find a sentiment score for each FLS class. My hypothesis is that, if the forward looking statement has positive sentiment, this may mean higher business performances,

which then translates to future stock prices. There may also be similar relationships between stock prices and sentiment of other forward-looking statement classes.

For example if there are 4 sentences with positive sentiment [0.6, 0.7, 0.8, 0.9] respectively, the sentiment score for S_FLS will be 0.75 as this is the average sentiment value.

```
sentence1 = {'label': 'Negative', 'score': 0.5338950753211975}
sentence2 = {'label': 'Neutral', 'score': 0.999138355255127}
sentence3 = {'label': 'Positive', 'score': 0.9999885559082031}
```

Figure 57a - Example of sentiment for different sentences

```
sentence1 = -0.5338950753211975
sentence2 = 0
sentence3 = 0.9999885559082031
```

Figure 57b - After the mapping function

However, in Figure 57a, the sentiment scores for the three sentences are not clearly distinguishable as negative, neutral, or positive. To address this issue, I implemented a sentiment scale ranging from -1 to 1, where -1 represents negative sentiment, 0 represents neutral sentiment, and positive scores indicates positive sentiment. To achieve this, I created a mapping function that assigns a “minus sign” to the score if the label is negative; if the label is neutral, it will set the score to 0; and if the label is positive, the score will stay the same. Hence, after inputting the contents of Figure 57a into this mapping function, it will output the contents of Figure 57b.

```
[{'label': 'Not FLS', 'score': 0.9779328107833862},
{'label': 'Not FLS', 'score': 0.9796538949012756},
{'label': 'Specific FLS', 'score': 0.8796855211257935}]


[{'label': 'Negative', 'score': 0.5338950753211975},
{'label': 'Neutral', 'score': 0.999138355255127},
{'label': 'Positive', 'score': 0.9999885559082031}]

----------------------------------------------------------------------
sentence1 = ['Not FLS', -0.5338950753211975]
sentence2 = ['Not FLS', 0]
sentence3 = ['Specific FLS', 0.9999885559082031]

Sentiment score of Not FlS = (-0.5339+0)/2 = -0.26695
Sentiment score of Specific FLS = 0.99998
```

Figure 58 - Combining FLS classification and negative mapping

Figure 58 shows the result if we combine FLS classification with sentiment mapping. **Sentence1** and **Sentence2** are both classified as **'Not FLS'**, hence the sentiment score for the **'Not FLS'** class is the average sentiment between the two, hence **-0.26695**. Similarly, since there is only 1 sentence for **'Specific FLS'**, the sentiment score for this class is **0.99998.**

Text Complexity:

With the above functions in mind, we can also find the text complexity value and combine it with the forward looking statement classification method as well. If a particular forward looking statement has a high text complexity value, this may imply that the statement is attempting to obscure or mislead future results by using convoluted language.

I used the gunning fog index and the flesch score index for identifying text complexity as they are both widely used and extensively studied and tested. Also, the two indices provide different measures of complexity, the Gunning Fog Index for example, measures text complexity based on sentence length and the use of complex words. Whereas the Flesch Score Index measures complexity based on the average number of syllables per word and the average number of words per sentence. By using the two indicies, it can give us a more comprehensive understanding of the text complexity.

Feature_extract_5to96.ipynb:

This Jupyter notebook file extracts all features within the index 5-96 (see **Summary of the features extracted from each transcript** in the **NEA Design section**), for 2469 stocks, and 85620 earnings transcripts.

Some functions in **Feature_extract_5to96.ipynb** are modified from other files, such as from **Dividing_transcript.ipynb** and **Sentiment_FLS_classify.ipynb**. The reason for this is that **Dividing_transcript.ipynb** and **Sentiment_FLS_classify.ipynb** were primarily used for experimenting and testing purposes, and the final versions of these functions have been incorporated into **Feature_extract_5to96.ipynb.**

```
def get_transcript(path):
   mytranscript = pd.read_csv(path).iloc[[2]].values[0][0]
   mytranscript = re.sub(r'[^A-Za-z0-9.,:!\'\n ]', '', mytranscript)
   mytranscript = mytranscript.replace(".", ". ")
   mytranscript = re.sub('[^\S\n]+', ' ', mytranscript) #replaces multiple spaces to single space, without
deleting newlines \n in the process
   mytranscript = mytranscript.splitlines() # finds transcript
```

```
    return mytranscript

def split_transcript(mytranscript):
    transcript_safe_harbour, transcript_questions = "", ""
    for i in range(0, len(mytranscript)):
        speech_bubble = mytranscript[i].lower()
        speech_bubble = re.sub(r'[^\w\s:]', ' ', speech_bubble) # regex: replaces all punctuations (except for ":") with 1 open space so the IF condition below can run smoothly
        # finds the following condition (what operator says) and splits the transcript into 2)
        if (i > 1) and (("operator:" in speech_bubble) and (("question" in speech_bubble) or ("go ahead" in speech_bubble) or ("operator instructions" in speech_bubble))):
            transcript_safe_harbour = mytranscript[0:i]
            transcript_questions = mytranscript[i:]
            break
        elif (i > 1 ) and ("operator" in speech_bubble) and ("question" in speech_bubble):
            transcript_safe_harbour = mytranscript[0:i+1]
            transcript_questions = mytranscript[i+1:]
            break
        elif (i > 1 ) and ("operator:" in speech_bubble) and ("first" in speech_bubble):
            transcript_safe_harbour = mytranscript[0:i]
            transcript_questions = mytranscript[i:]
            break

    return transcript_safe_harbour, transcript_questions

def get_file_speaker_names(sector, stock):
    write_path = "sectors/"+sector+"/"+stock+"/"+"speaker names.csv"
    speaker_names = np.loadtxt(write_path, delimiter='\t', dtype=str)
    return speaker_names

# finds a list of analyst names for a single .csv file
def find_analyst_names(speaker_names, transcript_questions):
    analyst_names = []
    # the programme recognises the question is being asked by an analyst when the following conditions are met:
    for index in range(0, len(transcript_questions)-1):
        speech_bubble = transcript_questions[index].lower()
        speech_bubble = re.sub(r'[^\w\s:]', ' ', speech_bubble) # regex: replaces all punctuations (except for ":") with 1 open space
        if ("operator:" in speech_bubble) or ("operator :" in speech_bubble):
            for name in speaker_names:
                namelist = name.split()
                if (name.lower() != "operator") and ("representative" not in name.lower()) and ("corporate" not in name.lower()) and ("company" not in name.lower()):
                    for name_2 in namelist: # cycle through each name in the name_list
                        name_2 = name_2.lower()
                        # checks if the speaker name happens to be in the speech_bubble, if it is, then the person speaking is an analyst
                        # also len(name) > 2 is used to avoid the problem with single letters being registered as in the speech_bubble
                        # (e.g. the letter "A" in the name "A Gayn Erickson" will be in the speech_bubble, but Gayn Erickson is not an analyst, so "A" is not counted)
                        if ((" "+name_2+" " in speech_bubble) and len(name_2) > 2) and (("end" not in speech_bubble) and ("closing" not in speech_bubble) and (("turn" not in speech_bubble) or (("over" not in speech_bubble))))) and (name_2 in transcript_questions[index+1].lower()):
                            analyst_names.append(name)
```

```
                    if "unidentified" in name.lower().split(): # finds name such as "Unidentified Analyst"
                        analyst_names.append(name)

    analyst_names = list(set(analyst_names)) # replaces duplicates
    return analyst_names

def get_analyst_management_sentences(analyst_names, transcript_questions):
    analyst_sentences = []
    management_sentences = []

    # get analyst sentence
    for index in range(0, len(transcript_questions)-1):
        speech_bubble = transcript_questions[index]

        #finds the name of the speaker
        colon_pos = speech_bubble.find(":")
        speaker_name = speech_bubble[:colon_pos]
        if index > 3:
            for name in analyst_names:
                namelist = name.split()
                if (speech_bubble not in analyst_sentences):
                    # checks if the name of the current speaker is in the "analyst_names" list, if it is,
then this speaker is considered as an analyst
                    if speaker_name in analyst_names:
                        analyst_sentences.append(speech_bubble)

                    elif (speaker_name.lower() == "operator") or (speaker_name.lower() == "operator "):
                        pass

                    elif ((namelist[0] in speaker_name) or (namelist[-1] in speaker_name)) and
(("operator:" in transcript_questions[index-1].lower()) or ("operator :" in
transcript_questions[index-1].lower())):
                        # in the case where:
                        # Operator: [Operator Instructions]Our first question is from Jeffrey Van Sinderen
with B. Riley FBR. Please proceed.
                        # JeffreySinderen: Good morning, everyone. Can you speak a little bit more about
the sales progression you've seen in China...
                        # Jeffrey Van Sindere is an analyst, but is referenced as JeffreySinderen in the
text, the previous find_analyst_names() function did not pick up this
                        # However, now the name "JeffreySinderen" is registered as an analyst name through
this function.
                        analyst_names.append(speaker_name)
                        analyst_sentences.append(speech_bubble)

    # get management sentence
    for index in range(0, len(transcript_questions)-2):
        speech_bubble = transcript_questions[index]
        colon_pos = speech_bubble.find(":")
        speaker_name = speech_bubble[:colon_pos]

        # dont want operator's sentence
        if (speaker_name.lower() == "operator") or (speaker_name.lower() == "operator "):
            pass

        elif (speech_bubble not in analyst_sentences):
            management_sentences.append(speech_bubble)
```

```
    return analyst_sentences, management_sentences
```

Figure 59 - transcript pre-processing functions

With reference to Figure 59, the `get_transcript` will get the transcript, `Split_transcript` will split the transcript into safe_harbour and Q&A sections, `get_file_speaker_names` will get the list of speaker names, `find_analyst_names` will spot which speaker is the analyst, and with this information in mind, `get_analyst_management_sentences` will divide the Q&A section into the analyst's questions section and management's answers section.

```
class TrieNode:
    def __init__(self):
        self.children = {}
        self.is_word = False
        self.fail = None
        self.word = None

class AhoCorasick:
    def __init__(self, words):
        self.root = TrieNode()
        self.build_trie(words)
        self.build_ac_automata()

    def build_trie(self, phrase):
        for word in phrase:
            node = self.root
            for mychar in word:
                if mychar not in node.children:
                    node.children[mychar] = TrieNode()
                node = node.children[mychar]
            node.is_word = True
            node.word = word

    def build_ac_automata(self):
        queue = []
        for node in self.root.children.values():
            queue.append(node)
            node.fail = self.root

        while len(queue) > 0:
            node = queue.pop(0)
            for char, child in node.children.items():
                queue.append(child)
                fail_node = node.fail
                while (fail_node is not None) and (char not in fail_node.children):
                    fail_node = fail_node.fail
                if fail_node is None:
                    child.fail = self.root
                else:
                    child.fail = fail_node.children[char]
                child.is_word |= child.fail.is_word

    def remove_words(self, text):
        node = self.root
```

```
        new_text = text
        for char in text:
            index = text.index(char)
            while (node is not None) and (char not in node.children):
                node = node.fail
            if node is None:
                node = self.root
                continue
            node = node.children[char]
            if node.is_word:
                new_text = new_text.replace(node.word, '')
        return new_text
```

Figure 60 - Implementation of the Aho Corasick algorithm as specified in the design section

```
list1 = ['string1', 'string2', 'string3']
sentence = "string1 is string2 is string is string3 is string"
ac = AhoCorasick(list1)
new_sentence = ac.remove_words(sentence)
new_sentence = re.sub('[^\S\n]+', ' ', new_sentence)

Output:
' is is string is is string'
```

Figure 61 - Using the remove_words method from the AhoCorasick Class

With reference to Figure 61, running the **ac.remove_words** method will delete a particular string from **new_sentence** if that string is present in **list1**. Hence it's output will be ' is is string is is string'.

Functions for getting feature set 5 to 22:

```
def getFeature5to22(pre_release, speaker_names):
    new_speaker_names = [word + ':' for word in speaker_names]

    ac = AhoCorasick(new_speaker_names)

    net_sentiment_list = []
    flesch_list = []

    n_flslist = []
    s_flslist = []
    ns_flslist = []

    net_positive = 0
    net_negative = 0
    net_neutral = 0

    feature_extract_5 = 0
    feature_extract_6 = 0
    feature_extract_7 = 0
```

```
    feature_extract_8 = 0
    feature_extract_9 = 0
    feature_extract_10 = 0
    feature_extract_11 = 0
    feature_extract_12 = 0
    feature_extract_13 = 0
    feature_extract_14 = 0
    feature_extract_15 = 0
    feature_extract_16 = 0
    feature_extract_17 = 0
    feature_extract_18 = 0
    feature_extract_19 = 0
    feature_extract_20 = 0
    feature_extract_21 = 0
    feature_extract_22 = 0

    try:
        for speech_bubble in pre_release:
            try:
                new_speech_bubble = ac.remove_words(speech_bubble)
                new_speech_bubble = re.sub('[^\S\n]+', ' ', new_speech_bubble)

                if new_speech_bubble[0] == " ":
                    new_speech_bubble = new_speech_bubble.replace(" ", "", 1) # replace the first space bar
with an empty string, for example ' is is string is is string' to 'is is string is is string'

                # gets text complexity
                flesch_score = textstat.flesch_reading_ease(new_speech_bubble)
                flesch_list.append(flesch_score)

                new_speech_bubble_list = split_paragraph_into_sentences(new_speech_bubble)

                fls_results = fls_nlp(new_speech_bubble_list)

                for i in range(0, len(new_speech_bubble_list)):
                    sentence = new_speech_bubble_list[i]
                    if fls_results[i]['label'] == 'Not FLS':
                        n_flslist.append(sentence)
                    elif fls_results[i]['label'] == 'Specific FLS':
                        s_flslist.append(sentence)
                    elif fls_results[i]['label'] == 'Non-specific FLS':
                        ns_flslist.append(sentence)

            except:
                pass

        try:
            feature_extract_9, feature_extract_10, feature_extract_11, feature_extract_12,
fls1_sentiment_list, net1_positive, net1_negative, net1_neutral = get_fls_features(s_flslist)

        except:
            pass

        try:
            feature_extract_13, feature_extract_14, feature_extract_15, feature_extract_16,
fls2_sentiment_list, net2_positive, net2_negative, net2_neutral = get_fls_features(ns_flslist)
```

```
        except:
            pass

        try:
            feature_extract_17, feature_extract_18, feature_extract_19, feature_extract_20,
fls3_sentiment_list, net3_positive, net3_negative, net3_neutral = get_fls_features(n_flslist)

        except:
            pass

        try:
            numb_s_flslist = len(s_flslist)
            numb_ns_flslist = len(ns_flslist)
            numb_n_flslist = len(n_flslist)
            total = numb_s_flslist + numb_ns_flslist + numb_n_flslist

            feature_extract_21 = numb_s_flslist/total
            feature_extract_22 = numb_ns_flslist/total

        except:
            pass

        try:
            net_positive = net1_positive + net2_positive + net3_positive
            net_negative = net1_negative + net2_negative + net3_negative
            net_neutral = net1_neutral + net2_neutral + net3_neutral
            net_sentiment_list = fls1_sentiment_list + fls2_sentiment_list + fls3_sentiment_list
        except:
            pass

        try:
            feature_extract_5 = statistics.mean(net_sentiment_list)
        except:
            pass

        try:
            feature_extract_6 = net_positive/(net_negative+net_positive+net_neutral)
            feature_extract_7 = net_negative/(net_negative+net_positive+net_neutral)
        except:
            pass

        try:
            feature_extract_8 = statistics.mean(flesch_list)
        except:
            pass

        fea_ext_list5to22 = [feature_extract_5, feature_extract_6, feature_extract_7, feature_extract_8,
feature_extract_9, feature_extract_10, feature_extract_11, feature_extract_12, feature_extract_13,
feature_extract_14, feature_extract_15, feature_extract_16, feature_extract_17, feature_extract_18,
feature_extract_19, feature_extract_20, feature_extract_21, feature_extract_22]

    except:
        fea_ext_list5to22 = [0]*18

    return fea_ext_list5to22
```

```
def get_fls_features(flslist):
   fls_sentiment_list = []
   net_positive = 0
   net_negative = 0
   net_neutral = 0

   feature_extract_1 = 0
   feature_extract_2 = 0
   feature_extract_3 = 0
   feature_extract_4 = 0

   for each_fls in flslist:
       sentiment_result = sentiment_nlp(each_fls)
       sentiment_score, positivity_value = map_sentiments(sentiment_result)
       fls_sentiment_list.append(sentiment_score)

       if positivity_value == "positive":
           net_positive += 1

       elif positivity_value == "negative":
           net_negative += 1

       else:
           net_neutral += 1
   try:
       feature_extract_1 = statistics.mean(fls_sentiment_list)
   except:
       pass
   try:
       feature_extract_2 = net_positive/(net_positive+net_negative+net_neutral)
   except:
       pass
   try:
       feature_extract_3 = net_negative/(net_positive+net_negative+net_neutral)
   except:
       pass
   try:
       feature_extract_4 = textstat.flesch_reading_ease(' '.join(flslist))
   except:
       pass

   return feature_extract_1, feature_extract_2, feature_extract_3, feature_extract_4, fls_sentiment_list,
net_positive, net_negative, net_neutral

def map_sentiments(sentiment_result):
   sentiment_result = sentiment_result[0]
   if sentiment_result['label'] == 'Negative':
       return -1 * sentiment_result['score'], "negative"

   elif sentiment_result['label'] == 'Neutral':
       return 0, "neutral"

   elif sentiment_result['label'] == 'Positive':
       return sentiment_result['score'], "positive"
```

```
def split_paragraph_into_sentences(para):
   sentences = nltk.sent_tokenize(para)
   return sentences

def get_NLP_values(liststr):
   # further analysis includes finding sentiment and word complexity.
   if len(liststr) == 0:
       return 0, 0, 0

   else:
       # maps sentiment data so it outputs a single sentiment value
       sentiment_result = sentiment_nlp(liststr)
       # gets
       sentiment_score = map_sentiments(sentiment_result)

       # word complexity:
       flesch_score = textstat.flesch_reading_ease(liststr)
       gunning_fog_score = textstat.gunning_fog(liststr)

       return sentiment_score, flesch_score, gunning_fog_score
```

Figure 62 - Extracting Feature indices 5 to 22

With reference to Figure 62, the programme will call the functions in Figure 59 and the AhoCorasick algorithm in Figure 60, and find the features with the index 5 to 22 accordingly by using modified algorithms from the **Sentiment_FLS_classify.ipynb** file, this includes:

5. Whole pre-release's net sentiment index
6. Whole pre-release's positive sentiment index
7. Whole pre-release's negative sentiment index
8. Whole pre-release's average text complexity index

9. Specific forward-looking statement's net sentiment index
10. Specific forward-looking statement's positive sentiment index
11. Specific forward-looking statement's negative sentiment index
12. Specific forward-looking statements average text complexity index

13. Non-Specific Forward-looking statement's net sentiment index
14. Non-Specific Forward-looking statement's positive sentiment index
15. Non-Specific Forward-looking statement's negative sentiment index
16. Non-Specific Forward-looking statement's average text complexity index

17. Not Foward-looking statement's net sentiment index
18. Not Foward-looking statement's positive sentiment index
19. Not Foward-looking statement's negative sentiment index
20. Not Foward looking statement's average text complexity index

21: Specific forward-looking statement index (calculated as the number of sentences classed as specific forward-looking, over the total number of sentences)
22: Non-Specific forward-looking statement index (calculated as the number of sentences classed as non-specific forward-looking, over the total number of sentences)

Functions for getting feature set 23 to 43:

```
def getFeature23to43(analyst_speech, management_speech, speaker_names):
  new_speaker_names = [word + ':' for word in speaker_names]
  ac = AhoCorasick(new_speaker_names)

  n_flslist = []
  s_flslist = []
  ns_flslist = []

  questions_complex_list = []
  reply_complex_list = []
  net_text_complex_list  = []

  # list of sentiments for all S_FLS, N_FLS, NS_FLS classes
  s_fls_sentiment_list = []
  n_fls_sentiment_list = []
  ns_fls_sentiment_list = []

  # list of sentiments for all sentences that are identified as a "question"
  question_sentiment_list = []

  # list of sentiments for all sentences that are identified as a "reply"
  reply_sentiment_list = []

  # list of sentiments for all sentences in the Q&A section
  net_sentiment_list = []

  net_positive = 0
  net_negative = 0
  net_neutral = 0

  Qpositive = 0
  Qnegative = 0
  Qneutral = 0

  Rpositive = 0
  Rnegative = 0
  Rneutral = 0

  SFLSpositive = 0
  SFLSnegative = 0
  SFLSneutral = 0

  NSFLSpositive = 0
```

```
    NSFLSnegative = 0
    NSFLSneutral = 0

    NFLSpositive = 0
    NFLSnegative = 0
    NFLSneutral = 0

    feature_extract_23 = 0
    feature_extract_24 = 0
    feature_extract_25 = 0
    feature_extract_26 = 0

    feature_extract_27 = 0
    feature_extract_28 = 0
    feature_extract_29 = 0
    feature_extract_30 = 0

    feature_extract_31 = 0
    feature_extract_32 = 0
    feature_extract_33 = 0
    feature_extract_34 = 0

    feature_extract_35 = 0
    feature_extract_36 = 0
    feature_extract_37 = 0

    feature_extract_38 = 0
    feature_extract_39 = 0
    feature_extract_40 = 0

    feature_extract_41 = 0
    feature_extract_42 = 0
    feature_extract_43 = 0

    try:
        for speech_bubble in analyst_speech:
            try:
                new_speech_bubble = ac.remove_words(speech_bubble)
                new_speech_bubble = re.sub('[^\S\n]+', ' ', new_speech_bubble)

                if new_speech_bubble[0] == " ":
                    new_speech_bubble = new_speech_bubble.replace(" ", "", 1) # replace the first space bar
with an empty string, for example ' is is string is is string' to 'is is string is is string'

                # gets text complexity
                flesch_score = textstat.flesch_reading_ease(new_speech_bubble)
                questions_complex_list.append(flesch_score)
                net_text_complex_list.append(flesch_score)

                new_speech_bubble_list = split_paragraph_into_sentences(new_speech_bubble)

                for i in range(0, len(new_speech_bubble_list)):
                    sentence = new_speech_bubble_list[i]
                    sentiment_result = sentiment_nlp(sentence)
                    sentiment_score, positivity_value = map_sentiments(sentiment_result)
                    question_sentiment_list.append(sentiment_score)
```

```
                    net_sentiment_list.append(sentiment_score)

                    if positivity_value == "positive":
                        net_positive += 1
                        Qpositive += 1

                    elif positivity_value == "negative":
                        net_negative += 1
                        Qnegative += 1

                    else:
                        net_neutral += 1
                        Qneutral += 1
            except:
                pass

        for speech_bubble in management_speech:
            try:
                new_speech_bubble = ac.remove_words(speech_bubble)
                new_speech_bubble = re.sub('[^\S\n]+', ' ', new_speech_bubble)

                if new_speech_bubble[0] == " ":
                    new_speech_bubble = new_speech_bubble.replace(" ", "", 1) # replace the first space bar
with an empty string, for example ' is is string is is string' to 'is is string is is string'

                # gets text complexity
                flesch_score = textstat.flesch_reading_ease(new_speech_bubble)
                reply_complex_list.append(flesch_score)
                net_text_complex_list.append(flesch_score)

                new_speech_bubble_list = split_paragraph_into_sentences(new_speech_bubble)

                fls_results = fls_nlp(new_speech_bubble_list)

                for i in range(0, len(new_speech_bubble_list)):
                    sentence = new_speech_bubble_list[i]
                    if fls_results[i]['label'] == 'Not FLS':
                        n_flslist.append(sentence)
                    elif fls_results[i]['label'] == 'Specific FLS':
                        s_flslist.append(sentence)
                    elif fls_results[i]['label'] == 'Non-specific FLS':
                        ns_flslist.append(sentence)

            except:
                pass

        # for "n_flslist":
        n_fls_sentiment_list, reply_sentiment_list, Rpositive, Rnegative, Rneutral, NFLSpositive,
NFLSnegative, NFLSneutral, net_positive, net_negative, net_neutral =
get_SentimentLists_from_FLS(net_sentiment_list, n_flslist, n_fls_sentiment_list, reply_sentiment_list,
Rpositive, Rnegative, Rneutral, NFLSpositive, NFLSnegative, NFLSneutral, net_positive, net_negative,
net_neutral)

        # for "s_flslist":
        s_fls_sentiment_list, reply_sentiment_list, Rpositive, Rnegative, Rneutral, SFLSpositive,
SFLSnegative, SFLSneutral, net_positive, net_negative, net_neutral =
```

```
get_SentimentLists_from_FLS(net_sentiment_list, s_flslist, s_fls_sentiment_list, reply_sentiment_list,
Rpositive, Rnegative, Rneutral, SFLSpositive, SFLSnegative, SFLSneutral, net_positive, net_negative,
net_neutral)

       # for "ns_flslist":
       ns_fls_sentiment_list, reply_sentiment_list, Rpositive, Rnegative, Rneutral, NSFLSpositive,
NSFLSnegative, NSFLSneutral, net_positive, net_negative, net_neutral =
get_SentimentLists_from_FLS(net_sentiment_list, ns_flslist, ns_fls_sentiment_list, reply_sentiment_list,
Rpositive, Rnegative, Rneutral, NSFLSpositive, NSFLSnegative, NSFLSneutral, net_positive, net_negative,
net_neutral)

       feature_extract_23 = statistics.mean(net_sentiment_list)

       try:
           feature_extract_24 = net_positive/(net_positive+net_negative+net_neutral)
           feature_extract_25 = net_negative/(net_positive+net_negative+net_neutral)
       except:
           pass
       try:
           feature_extract_26 = statistics.mean(net_text_complex_list)
       except:
           pass
       try:
           feature_extract_27 = statistics.mean(question_sentiment_list)
       except:
           pass
       try:
           feature_extract_28 = Qpositive/(Qpositive+Qnegative+Qneutral)
           feature_extract_29 = Qnegative/(Qpositive+Qnegative+Qneutral)
       except:
           pass
       try:
           feature_extract_30 = statistics.mean(questions_complex_list)
       except:
           pass
       try:
           feature_extract_31 = statistics.mean(reply_sentiment_list)
       except:
           pass
       try:
           feature_extract_32 = Rpositive/(Rpositive+Rnegative+Rneutral)
           feature_extract_33 = Rnegative/(Rpositive+Rnegative+Rneutral)
       except:
           pass
       try:
           feature_extract_34 = statistics.mean(reply_complex_list)
       except:
           pass
       try:
           feature_extract_35 = statistics.mean(s_fls_sentiment_list)
       except:
           pass
       try:
           feature_extract_36 = SFLSpositive/(SFLSpositive+SFLSnegative+SFLSneutral)
           feature_extract_37 = SFLSnegative/(SFLSpositive+SFLSnegative+SFLSneutral)
       except:
```

```
            pass

        try:
            feature_extract_38 = statistics.mean(ns_fls_sentiment_list)
        except:
            pass
        try:
            feature_extract_39 = NSFLSpositive/(NSFLSpositive+NSFLSnegative+NSFLSneutral)
            feature_extract_40 = NSFLSnegative/(NSFLSpositive+NSFLSnegative+NSFLSneutral)
        except:
            pass

        try:
            feature_extract_41 = statistics.mean(n_fls_sentiment_list)
        except:
            pass
        try:
            feature_extract_42 = NFLSpositive/(NFLSpositive+NFLSnegative+NFLSneutral)
            feature_extract_43 = NFLSnegative/(NFLSpositive+NFLSnegative+NFLSneutral)
        except:
            pass

    except:
        pass
    return [feature_extract_23, feature_extract_24, feature_extract_25, feature_extract_26,
feature_extract_27, feature_extract_28, feature_extract_29, feature_extract_30, feature_extract_31,
feature_extract_32, feature_extract_33, feature_extract_34, feature_extract_35, feature_extract_36,
feature_extract_37, feature_extract_38, feature_extract_39, feature_extract_40, feature_extract_41,
feature_extract_42, feature_extract_43]

def get_SentimentLists_from_FLS(net_sentiment_list, THIS_flslist, THISfls_sentiment_list,
reply_sentiment_list, Rpositive, Rnegative, Rneutral, FLSpositive, FLSnegative, FLSneutral, net_positive,
net_negative, net_neutral):
    for each_fls_sentence in THIS_flslist:
        sentiment_result = sentiment_nlp(each_fls_sentence)
        sentiment_score, positivity_value = map_sentiments(sentiment_result)
        THISfls_sentiment_list.append(sentiment_score)
        reply_sentiment_list.append(sentiment_score)
        net_sentiment_list.append(sentiment_score)

        if positivity_value == "positive":
            net_positive += 1
            FLSpositive += 1
            Rpositive += 1
        elif positivity_value == "negative":
            net_negative += 1
            FLSnegative += 1
            Rnegative += 1

        else:
            net_neutral += 1
            FLSneutral += 1
            Rneutral += 1

    return THISfls_sentiment_list, reply_sentiment_list, Rpositive, Rnegative, Rneutral, FLSpositive,
FLSnegative, FLSneutral, net_positive, net_negative, net_neutral
```

Figure 63 - Code for extracting Feature indices 23 to 43

Figure 63 shows the function used to extract Features 23 to 43, the output will be a list format (i.e. [feature23, feature24 … feature 42, feature 43]). Feature 23 to 43 includes:

23. Whole Q&A's net sentiment index
24. Whole Q&A's positive sentiment index
25. Whole Q&A's negative sentiment index
26. Whole Q&A's average text complexity index

27. All Question's net sentiment index
28. All Question's positive sentiment index
29. All Question's negative sentiment index
30. All Question's average text complexity index

31. All Reply's net sentiment index
32. All Reply's positive sentiment index
33. All Reply's negative sentiment index
34 .All Reply's average text complexity index

35. Specific forward-looking statement's net sentiment index
36. Specific forward-looking statement's positive sentiment index
37. Specific forward-looking statement's negative sentiment index

38. Non-Specific Forward-looking statement's net sentiment index
39. Non-Specific Forward-looking statement's positive sentiment index
40. Non-Specific Forward-looking statement's negative sentiment index

41. Not Forward-looking statement's net sentiment index
42. Not Forward-looking statement's positive sentiment index
43. Not Forward-looking statement's negative sentiment index

Functions for getting feature set 44 to 73:

```
def getFeature44to73(mytranscript, speaker_names):
   marginSentimentList = []
   mar_positive = 0
   mar_negative = 0
   mar_neutral = 0

   costSentimentList = []
```

```
cost_positive = 0
cost_negative = 0
cost_neutral = 0

revenueSentimentList = []
rev_positive = 0
rev_negative = 0
rev_neutral = 0

earningsEBIDTASentimentList = []
ear_positive = 0
ear_negative = 0
ear_neutral = 0

growthSentimentList = []
gro_positive = 0
gro_negative = 0
gro_neutral = 0

leverageDebtSentimentList = []
lev_positive = 0
lev_negative = 0
lev_neutral = 0

IndSentimentList = []
ind_positive = 0
ind_negative = 0
ind_neutral = 0

operationSentimentList = []
ope_positive = 0
ope_negative = 0
ope_neutral = 0

cashflowSentimentList = []
cash_positive = 0
cash_negative = 0
cash_neutral = 0

dividendSentimentList = []
div_positive = 0
div_negative = 0
div_neutral = 0

feature_extract_44 = 0
feature_extract_45 = 0
feature_extract_46 = 0
feature_extract_47 = 0
feature_extract_48 = 0
feature_extract_49 = 0
feature_extract_50 = 0
feature_extract_51 = 0
feature_extract_52 = 0
feature_extract_53 = 0
feature_extract_54 = 0
feature_extract_55 = 0
```

```
feature_extract_56 = 0
feature_extract_57 = 0
feature_extract_58 = 0
feature_extract_59 = 0
feature_extract_60 = 0
feautre_extract_61 = 0
feature_extract_62 = 0
feature_extract_63 = 0
feature_extract_64 = 0
feature_extract_65 = 0
feature_extract_66 = 0
feature_extract_67 = 0
feature_extract_68 = 0
feature_extract_69 = 0
feature_extract_70 = 0
feature_extract_71 = 0
feature_extract_72 = 0
feature_extract_73 = 0

updatedTranscript = deepCleanTranscript(mytranscript, speaker_names)

updatedTranscriptList = split_paragraph_into_sentences(updatedTranscript)

for mysentence in updatedTranscriptList:
    if (" margin" in mysentence) or (" return" in mysentence):
        sentiment_result = sentiment_nlp(mysentence)
        sentiment_score, positivity_value = map_sentiments(sentiment_result)
        marginSentimentList.append(sentiment_score)
        if positivity_value == "positive":
            mar_positive += 1

        elif positivity_value == "negative":
            mar_negative += 1

        else:
            mar_neutral += 1


    if " cost" in mysentence:
        sentiment_result = sentiment_nlp(mysentence)
        sentiment_score, positivity_value = map_sentiments(sentiment_result)
        costSentimentList.append(sentiment_score)
        if positivity_value == "positive":
            cost_positive += 1

        elif positivity_value == "negative":
            cost_negative += 1

        else:
            cost_neutral += 1

    if (" revenue" in mysentence) or (" top line" in mysentence) or (" sales" in mysentence):
        sentiment_result = sentiment_nlp(mysentence)
        sentiment_score, positivity_value = map_sentiments(sentiment_result)
        revenueSentimentList.append(sentiment_score)
```

```
            if positivity_value == "positive":
                rev_positive += 1

            elif positivity_value == "negative":
                rev_negative += 1

            else:
                rev_neutral += 1

        if (" earning" in mysentence) or (" EBIT" in mysentence) or (" profit" in mysentence) or (" bottom
line" in mysentence) or (" net income" in mysentence):
            sentiment_result = sentiment_nlp(mysentence)
            sentiment_score, positivity_value = map_sentiments(sentiment_result)
            earningsEBIDTASentimentList.append(sentiment_score)
            if positivity_value == "positive":
                ear_positive += 1

            elif positivity_value == "negative":
                ear_negative += 1

            else:
                ear_neutral += 1

        if (" growth" in mysentence) or (" organic" in mysentence):
            sentiment_result = sentiment_nlp(mysentence)
            sentiment_score, positivity_value = map_sentiments(sentiment_result)
            growthSentimentList.append(sentiment_score)
            if positivity_value == "positive":
                gro_positive += 1

            elif positivity_value == "negative":
                gro_negative += 1

            else:
                gro_neutral += 1

        if (" leverage" in mysentence) or (" debt" in mysentence):
            sentiment_result = sentiment_nlp(mysentence)
            sentiment_score, positivity_value = map_sentiments(sentiment_result)
            leverageDebtSentimentList.append(sentiment_score)
            if positivity_value == "positive":
                lev_positive += 1

            elif positivity_value == "negative":
                lev_negative += 1

            else:
                lev_neutral += 1

        if (" industry" in mysentence) or (" industr" in mysentence) or (" economy" in mysentence) or ("
economi"" in mysentence) or (" sector" in mysentence):
            sentiment_result = sentiment_nlp(mysentence)
            sentiment_score, positivity_value = map_sentiments(sentiment_result)
            IndSentimentList.append(sentiment_score)
            if positivity_value == "positive":
                ind_positive += 1
```

```
            elif positivity_value == "negative":
                ind_negative += 1

            else:
                ind_neutral += 1

        if " operation" in mysentence:
            sentiment_result = sentiment_nlp(mysentence)
            sentiment_score, positivity_value = map_sentiments(sentiment_result)
            operationSentimentList.append(sentiment_score)
            if positivity_value == "positive":
                ope_positive += 1

            elif positivity_value == "negative":
                ope_negative += 1

            else:
                ope_neutral += 1

        if (" cashflow" in mysentence) or (" cash flow" in mysentence):
            sentiment_result = sentiment_nlp(mysentence)
            sentiment_score, positivity_value = map_sentiments(sentiment_result)
            cashflowSentimentList.append(sentiment_score)
            if positivity_value == "positive":
                cash_positive += 1

            elif positivity_value == "negative":
                cash_negative += 1

            else:
                cash_neutral += 1

        if (" dividend" in mysentence) or (" buyback" in mysentence) or (" repurchase" in mysentence):
            sentiment_result = sentiment_nlp(mysentence)
            sentiment_score, positivity_value = map_sentiments(sentiment_result)
            dividendSentimentList.append(sentiment_score)
            if positivity_value == "positive":
                div_positive += 1

            elif positivity_value == "negative":
                div_negative += 1

            else:
                div_neutral += 1

    try:
        feature_extract_44 = statistics.mean(marginSentimentList)
    except:
        pass
    try:
        feature_extract_45 = mar_positive/(mar_positive+mar_negative+mar_neutral)
        feature_extract_46 = mar_negative/(mar_positive+mar_negative+mar_neutral)
    except:
        pass
    try:
```

```
        feature_extract_47 = statistics.mean(costSentimentList)
    except:
        pass
    try:
        feature_extract_48 = cost_positive/(cost_positive+cost_negative+cost_neutral)
        feature_extract_49 = cost_negative/(cost_positive+cost_negative+cost_neutral)
    except:
        pass
    try:
        feature_extract_50 = statistics.mean(revenueSentimentList)
    except:
        pass
    try:
        feature_extract_51 = rev_positive/(rev_positive+rev_negative+rev_neutral)
        feature_extract_52 = rev_negative/(rev_positive+rev_negative+rev_neutral)
    except:
        pass
    try:
        feature_extract_53 = statistics.mean(earningsEBIDTASentimentList)
    except:
        pass
    try:
        feature_extract_54 = ear_positive/(ear_positive+ear_negative+ear_neutral)
        feature_extract_55 = ear_negative/(ear_positive+ear_negative+ear_neutral)
    except:
        pass

    try:
        feature_extract_56 = statistics.mean(growthSentimentList)
    except:
        pass
    try:
        feature_extract_57 = gro_positive/(gro_positive+gro_negative+gro_neutral)
        feature_extract_58 = gro_negative/(gro_positive+gro_negative+gro_neutral)
    except:
        pass

    try:
        feature_extract_59 = statistics.mean(leverageDebtSentimentList)
    except:
        pass
    try:
        feature_extract_60 = lev_positive/(lev_positive+lev_negative+lev_neutral)
        feature_extract_61 = lev_negative/(lev_positive+lev_negative+lev_neutral)
    except:
        pass

    try:
        feature_extract_62 = statistics.mean(IndSentimentList)
    except:
        pass
    try:
        feature_extract_63 = ind_positive/(ind_positive+ind_negative+ind_neutral)
        feature_extract_64 = ind_negative/(ind_positive+ind_negative+ind_neutral)
    except:
        pass
```

```
    try:
        feature_extract_65 = statistics.mean(operationSentimentList)
    except:
        pass
    try:
        feature_extract_66 = ope_positive/(ope_positive+ope_negative+ope_neutral)
        feature_extract_67 = ope_negative/(ope_positive+ope_negative+ope_neutral)

    except:
        pass

    try:
        feature_extract_68 = statistics.mean(cashflowSentimentList)
    except:
        pass
    try:
        feature_extract_69 = cash_positive/(cash_positive+cash_negative+cash_neutral)
        feature_extract_70 = cash_negative/(cash_positive+cash_negative+cash_neutral)

    except:
        pass

    try:
        feature_extract_71 = statistics.mean(dividendSentimentList)
    except:
        pass
    try:
        feature_extract_72 = div_positive/(div_positive+div_negative+div_neutral)
        feature_extract_73 = div_negative/(div_positive+div_negative+div_neutral)
    except:
        pass

    return [feature_extract_44, feature_extract_45, feature_extract_46, feature_extract_47,
feature_extract_48, feature_extract_49, feature_extract_50, feature_extract_51, feature_extract_52,
feature_extract_53, feature_extract_54, feature_extract_55, feature_extract_56, feature_extract_57,
feature_extract_58, feature_extract_59, feature_extract_60, feautre_extract_61, feature_extract_62,
feature_extract_63, feature_extract_64, feature_extract_65, feature_extract_66, feature_extract_67,
feature_extract_68, feature_extract_69, feature_extract_70, feature_extract_71, feature_extract_72,
feature_extract_73]

def deepCleanTranscript(mytranscript, speaker_names):
    updatedTranscript = ' '.join(mytranscript)
    new_speaker_names = [word + ':' for word in speaker_names]
    ac = AhoCorasick(new_speaker_names)

    updatedTranscript = ac.remove_words(updatedTranscript)
    if updatedTranscript[0] == " ":
        updatedTranscript = updatedTranscript.replace(" ", "", 1)

    updatedTranscript = re.sub('[^\S\n]+', ' ', updatedTranscript)
    updatedTranscript.lower()

    return updatedTranscript
```

Figure 64 - Code for extracting features 44 to 73

Figure 64 shows the code for extracting features 44 to 73 which includes:

44: "margin" - average sentiment (i.e. what's the average sentiment for sentences that mentioned the word "margin")
45: "margin" - positive sentiment index
46: "margin" - negative sentiment index
47: "cost" - average sentiment
48: "cost" - positive sentiment index
49: "cost" - negative sentiment index
50: "revenue" - average sentiment
51: "revenue" - positive sentiment index
52: "revenue" - negative sentiment index
53: "earnings/EBIDTA" - average sentiment
54: "earnings/EBIDTA" - positive sentiment index
55: "earnings/EBIDTA" - negative sentiment index
56: "growth" - average sentiment
57: "growth" - positive sentiment index
58: "growth" - negative sentiment index
59: "leverage/debt" - average sentiment
60: "leverage/debt" - positive sentiment index
61: "leverage/debt" - negative sentiment index
62: "industry/sector" – average sentiment
63: "industry/sector" – positive sentiment index
64: "industry/sector" – negative sentiment index
65: "operation" - average sentiment
66: "operation" - positive sentiment index
67: "operation" - negative sentiment index
68: "cashflow" - average sentiment
69: "cashflow" - positive sentiment index
70: "cashflow" - negative sentiment index
71: "dividend/share buyback" - average sentiment
72: "dividend/share buyback" - positive sentiment index
73: "dividend/share buyback" - negative sentiment index

The function `deepCleanTranscript(mytranscript, speaker_names)` essentially removes the speaker names from the transcript. For example, the sentence "`Stephen Mullery: Thanks, Tim. Some of the statements made on...`" will become "`Thanks, Tim. Some of the statements made on...`".

Functions for getting feature set 74 to 85:

As a reminder, features 74 to 85 includes:
74: Text similarity between the current and the 2nd most recent earnings transcript
75: Text similarity between the current and the 3rd most recent earnings transcript

76: Text similarity between the current and the 4th most recent earnings transcript

77: Text similarity between the current and the 2nd most recent earnings transcript (only looking at the pre-release materials)
78: Text similarity between the current and the 3rd most recent earnings transcript (only looking at the pre-release materials)
79: Text similarity between the current and the 4th most recent earnings transcript (only looking at the pre-release materials)

80: Text similarity between the current and the 2nd most recent earnings transcript (only looking at the management replies)
81: Text similarity between the current and the 3rd most recent earnings transcript (only looking at the management replies)
82: Text similarity between the current and the 4th most recent earnings transcript (only looking at the management replies)

83: Text similarity between the current and the 2nd most recent earnings transcript (only looking at the analyst questions)
84: Text similarity between the current and the 3rd most recent earnings transcript (only looking at the analyst questions)
85: Text similarity between the current and the 4th most recent earnings transcript (only looking at the analyst questions)

To determine the text similarity, I employed a text vectorisation technique involving using the term frequency-inverse document frequency (TF-IDF) and cosine similarity.

TF-IDF and Cosine Similarity

The term frequency-inverse document frequency (TF-IDF) is a vector that intends to reflect how important a given word/term is to a document in a collection or corpus[4].

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

tf_{ij} = number of occurrences of i in j
df_i = number of documents containing i
N = total number of documents

Figure 65 - TF-IDF formula, represented by $w_{i,j}$

With reference to Figure 65:

[4] Corpus = collection of documents

- ‘i’ is the specific word in question
- ‘j’ represents a single earnings transcript
- ‘Documents’ is synonymous with ‘collection’ or ‘corpus’ which in this case represents a rolling window of a company’s 3 most recent earnings calls.

	tfidf
had	0.493562
little	0.493562
tiny	0.493562
house	0.398203
mouse	0.235185
the	0.235185
ate	0.000000
away	0.000000
cat	0.000000
end	0.000000
finally	0.000000
from	0.000000
of	0.000000
ran	0.000000
saw	0.000000
story	0.000000

```
(0, 43)       0.01711525549397508
(0, 47)       0.020934414386550717
(0, 31)       0.10467207193275359
(0, 5)        0.10467207193275359
(0, 37)       0.06846102197590032
(0, 55)       0.041868828773101434
(0, 40)       0.020934414386550717
(0, 26)       0.01711525549397508
(0, 82)       0.03423051098795016
(0, 92)       0.03423051098795016
(0, 77)       0.020934414386550717
(0, 85)       0.020934414386550717
(0, 41)       0.06846102197590032
(0, 68)       0.020934414386550717
(0, 12)       0.03423051098795016
(0, 49)       0.03423051098795016
(0, 38)       0.051345766481925244
(0, 3)        0.01711525549397508
(0, 42)       0.01711525549397508
(0, 80)       0.041868828773101434
(0, 79)       0.0855762774698754
(0, 39)       0.041868828773101434
(0, 10)       0.10269153296385049
(0, 84)       0.0855762774698754
(0, 53)       0.08373765754620287
```

Figure 66a (left) - A visual example of a TF-IDF vector
Figure 66b (right) - Matrices example of a TF-IDF vector

The resulting matrix is a sparse matrix where each row (“x” value of the coordinate in Figure 66b) represents the particular document and each column represents a unique word in the entire corpus (“y” value of the coordinate). The value in each cell of the matrix is the product of the TF and IDF for the corresponding word and document. This value indicates the importance of the corresponding word in the given document. A higher value indicates that the word is more important in this document, while a lower value indicates that the word is less important.

For example:
(0, 85) 0.02093 means the word represented by the key index 85, in document “0” (aka the first document) has an importance value of 0.02093.

(1, 85) 0.05312 means this same word that appears in the second document (index “1”) has a more important value.

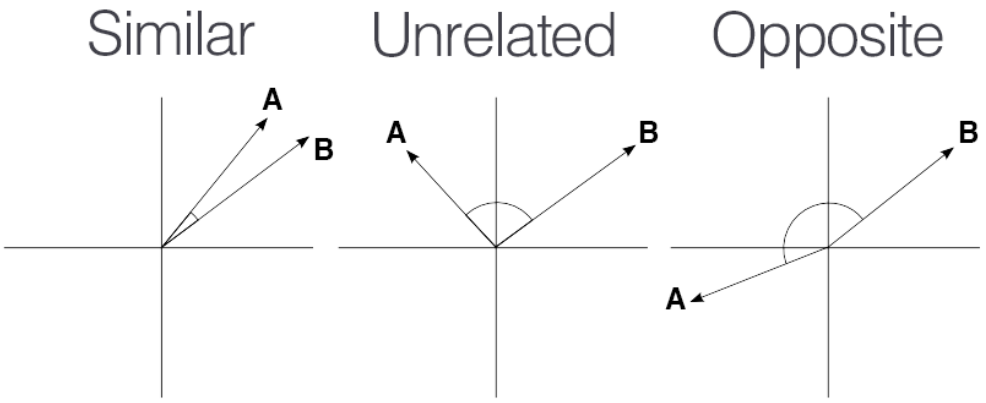
Note that the rolling frame includes a list of the three most recent earnings transcripts, and the current earnings transcript. So for example, if the current earnings transcript is 2022Q4, the rolling frame will look like **[2022Q4, 2022Q3, 2022Q2, 2022Q1].**

The reason I'm using a rolling frame is that by incorporating a corpus of a company's 4 most recent earnings calls, it will give a more balanced picture of how similar the current call is to the past 4 earnings calls, essentially, it smooths out the noise. However, using the TF-IDF vector itself as a feature holds little value since it can't be compared with others. And naturally, the TF-IDF vector must be compared to a relatable counterpart, for example, Apple may include words such as 'iPad' but that doesn't mean anything in the context of United Airlines. Hence, to obtain the text similarity value, the current earnings transcript's TF-IDF vector is compared to each of the other three transcripts from the rolling frame.

Cosine Similarity is used to compare the different TF-IDF vectors of the same company. This way, for example, if Apple suddenly reduces the mention of the word 'iPad' in 2022-Q2, it will be reflected by a low cosine similarity score, the intuition is that there should've been some operational change. Regardless of whether the sudden alteration in cosine similarity is good or bad, it is nevertheless an important statistic that may add weight to other values (e.g. sentiment[5]).

$$\cos\theta = \frac{a \cdot b}{|a||b|}$$

Figure 67a - Cosine Similarity formula



[5] If the mentioning of the word 'iPad' is reduced, cosine similarity will change. However if sentiment is high, the cosine similarity may act as a weight for the sentiment value and imply there is brighter future outlook.

Figure 67b - Visualising cosine similarity, where cosine similarity = Cos(θ)

Cosine similarity is calculated by dividing the dot product by the product of the two vector's magnitudes. Unlike Euclidean distance, cosine similarity is better suited for information retrieval and text mining and achieves higher accuracy according to OpenGenus IQ. Referencing Figure 67b, a smaller angle between the two vectors translates to a high value for the cosine similarity, and thus imply they are more similar.

```
def find_cosineSimilarity(thistranscript):
   tf_idf_value = tf_idf(thistranscript)
   cosineMatrix = linear_kernel(tf_idf_value) # finds the cosine similarity matrix

   cos1_2 = cosineMatrix[0,1] # (compares the first transcript with the second transcript)
   cos1_3 = cosineMatrix[0,2] # (compares the first transcript with the third transcript)
   cos1_4 = cosineMatrix[0,3] # (compares the first transcript with the fourth transcript)

   return cos1_2, cos1_3, cos1_4

def tf_idf(transcript):
   custom_stop_words = ['thanks', 'thank', 'really', 'said', 'say', 'yes', 'no', 've', 'll', 'don']
   all_stop_words = list(ENGLISH_STOP_WORDS) + custom_stop_words

   vectoriser = TfidfVectorizer(
       lowercase=True,
       max_features=100,
       ngram_range=(1, 3), # 1 to trigram as they are all common in finance (i.e. earnings per share, free
cash flow etc.)
       stop_words=all_stop_words # removes stop words (i.e. irrevelant day to day words)
   )

   #vectorises tfidf values into a vector
   tfidf_vec = vectoriser.fit_transform(transcript)

   return tfidf_vec

def getFeature74to85(sector, stock, rolling_path_of_four_transcript):

   WHOLE_rolling_frame_of_four_transcripts = []
   PRERELEASE_rolling_frame_of_four_transcripts = []
   MANAGEMENT_SENTENCES_rolling_frame_of_four_transcripts = []
   ANALYST_SENTENCES_rolling_frame_of_four_transcripts = []


   for path in rolling_path_of_four_transcript:
       wholeTranscript = get_transcript(path)
       transcript_safe_harbour, transcript_questions = split_transcript(wholeTranscript)
       speaker_names = get_file_speaker_names(sector, stock)
       analyst_names = find_analyst_names(speaker_names, transcript_questions)

       analyst_sentences, management_sentences = get_analyst_management_sentences(analyst_names,
transcript_questions)
```

```
        wholeTranscript = deepCleanTranscript(wholeTranscript, speaker_names)
        transcript_safe_harbour = deepCleanTranscript(transcript_safe_harbour, speaker_names)
        management_sentences = deepCleanTranscript(management_sentences, speaker_names)
        analyst_sentences = deepCleanTranscript(analyst_sentences, speaker_names)

        WHOLE_rolling_frame_of_four_transcripts.append(wholeTranscript)
        PRERELEASE_rolling_frame_of_four_transcripts.append(transcript_safe_harbour)
        MANAGEMENT_SENTENCES_rolling_frame_of_four_transcripts.append(management_sentences)
        ANALYST_SENTENCES_rolling_frame_of_four_transcripts.append(analyst_sentences)

    feature_extract_74, feature_extract_75, feature_extract_76 =
find_cosineSimilarity(WHOLE_rolling_frame_of_four_transcripts)

    feature_extract_77, feature_extract_78, feature_extract_79 =
find_cosineSimilarity(PRERELEASE_rolling_frame_of_four_transcripts)

    feature_extract_80, feature_extract_81, feature_extract_82 =
find_cosineSimilarity(MANAGEMENT_SENTENCES_rolling_frame_of_four_transcripts)

    feature_extract_83, feature_extract_84, feature_extract_85 =
find_cosineSimilarity(ANALYST_SENTENCES_rolling_frame_of_four_transcripts)


    return [feature_extract_74, feature_extract_75, feature_extract_76, feature_extract_77,
feature_extract_78, feature_extract_79, feature_extract_80, feature_extract_81, feature_extract_82,
feature_extract_83, feature_extract_84, feature_extract_85]
```

Figure 68 - Functions to extract features 74 to 85

```
[[1.         0.75076336 0.6806517  0.63510046]
 [0.75076336 1.         0.72706212 0.60725982]
 [0.6806517  0.72706212 1.         0.66643206]
 [0.63510046 0.60725982 0.66643206 1.        ]]
0.7507633631950922
0.6806517019015744
0.6351004595056162
```

Figure 69a - An example of the cosineMatrix

With reference to Figure 68, the `tf_idf` function and `find_cosineSimilarity` interacts with each other to create a matrix, defined as `cosineMatrix`. The `cosineMatrix` will look like Figure 69. The `cosineMatrix` is in the format:

Transcript Documents:	**2022 Q4**	**2022 Q3**	**2022 Q2**	**2022 Q1**
2022 Q4	1	0.75	0.68	0.64

2022 Q3	0.75	1	0.73	0.61
2022 Q2	0.68	0.73	1	0.67
2022 Q1	0.64	0.61	0.67	1

Figure 69b - An example of the cosineMatrix, table format

The cosine similarity between 2022 Q4 and 2022 Q4 is 1 because the TF-IDF vectors are the same. Row 2, column 3 of Figure 69b, holds a value of 0.75, this implies that the earnings transcript of 2022 Q4 is 75% "similar" with the transcript in 2022 Q3.

After parsing the relevant sentences as specified by the descriptions between the features 74 to 85, the `getFeature74to85` function will return a list that includes features between 74 to 85.

Functions for getting the dependent variable (aka Y-data):

```
from datetime import timedelta

def find_percentage_change(hist):
    try:
        percentage_change = ((hist['Open'][-1])/(hist['Open'][0])-1) # % change in price of day X to day X+Y
    except:
        return None

    return percentage_change

def get_stock_returns(path, stock):
    date1 = pd.read_csv(path).iloc[[1]].values[0][0]
    date1 = datetime.strptime(date1, '%Y-%m-%d %H:%M:%S').date()

    yfTicker = yf.Ticker(stock)
    TickerHistory = yfTicker.history

    hist1 = TickerHistory(start=date1, end=date1 + timedelta(days=10))  # day 0 to day 10
    hist2 = TickerHistory(start=date1, end=date1 + timedelta(days=30))   # day 0 to day 30
    hist3 = TickerHistory(start=date1, end=date1 + timedelta(days=50))
    hist4 = TickerHistory(start=date1, end=date1 + timedelta(days=70))
    hist5 = TickerHistory(start=date1, end=date1 + timedelta(days=90))

    hist6 = TickerHistory(start=date1 + timedelta(days=1), end=date1 + timedelta(days=10))   #day 1 to day 11
    hist7 = TickerHistory(start=date1 + timedelta(days=1), end=date1 + timedelta(days=30))   #day 1 to day 31
    hist8 = TickerHistory(start=date1 + timedelta(days=1), end=date1 + timedelta(days=50))
    hist9 = TickerHistory(start=date1 + timedelta(days=1), end=date1 + timedelta(days=70))
    hist10 = TickerHistory(start=date1 + timedelta(days=1), end=date1 + timedelta(days=90))

    percentage_change1 = find_percentage_change(hist1)
    percentage_change2 = find_percentage_change(hist2)
    percentage_change3 = find_percentage_change(hist3)
    percentage_change4 = find_percentage_change(hist4)
    percentage_change5 = find_percentage_change(hist5)
```

```
    percentage_change6 = find_percentage_change(hist6)
    percentage_change7 = find_percentage_change(hist7)
    percentage_change8 = find_percentage_change(hist8)
    percentage_change9 = find_percentage_change(hist9)
    percentage_change10 = find_percentage_change(hist10)

    return [percentage_change1, percentage_change2, percentage_change3, percentage_change4,
percentage_change5, percentage_change6, percentage_change7, percentage_change8, percentage_change9,
percentage_change10]
```

Figure 70 - Getting stock returns

Figure 70 shows the code for getting the stock returns, which is used as the dependent variable for the machine learning algorithm later. YFinance is used to find the historical stock return data.

Gathering Features 5 to 96:

```
csv.field_size_limit(sys.maxsize)

for sector in sectorlist:
    filelist = os.listdir("sectors/"+sector)
    try:
        filelist.remove('.DS_Store')
    except:
        pass
    for stock in filelist:
        print(stock)
        sector_files = glob.glob('sectors/'+str(sector)+'/'+str(stock)+'/'+str(stock)+'20*[0-9]**[0-9]*[1-4].*')
        sector_files.sort(reverse=True)
        ticker = yf.Ticker(stock)
        market_cap = ticker.fast_info['market_cap']
        for i in range (0, len(sector_files)): # for every .csv path of that stock
            path = sector_files[i]
            with open(path, 'r') as file:
                reader = csv.reader(file)
                lengthofList = len(list(reader))
            if lengthofList == 5:
                mytranscript = get_transcript(path)
                transcript_safe_harbour, transcript_questions = split_transcript(mytranscript)
                speaker_names = get_file_speaker_names(sector, stock)
                analyst_names = find_analyst_names(speaker_names, transcript_questions)

                analyst_sentences, management_sentences = get_analyst_management_sentences(analyst_names,
transcript_questions)

                fea_ext_list5to22 = getFeature5to22(transcript_safe_harbour, speaker_names)

                fea_ext_list23to43 = getFeature23to43(analyst_sentences, management_sentences, speaker_names)

                fea_ext_list44to73 = getFeature44to73(mytranscript, speaker_names)

                if i < len(sector_files)-3:
                    rolling_path_of_four_transcript = sector_files[i:i+4]
```

```
                    fea_ext_list74to85 = getFeature74to85(sector, stock, rolling_path_of_four_transcript)
                else:
                    fea_ext_list74to85 = [None]*12

                stock_return_list = get_stock_returns(path, stock)

                list_add_to_csv = fea_ext_list5to22 + fea_ext_list23to43 + fea_ext_list44to73+ fea_ext_list74to85 +
stock_return_list + [market_cap]

            #if there are 5 items in the list
                with open(path, 'a', newline='') as file:
                    writer = csv.writer(file)
                    for item in list_add_to_csv:
                        writer.writerow([item])
    break
```

Figure 71 - Code for gathering features 5 to 96

Figure 71 will cycle through each sector, stock, and transcript to run the relevant aforementioned functions. It will also query the market cap (feature 96) using the yfinance API `market_cap = ticker.fast_info['market_cap']`.

The function below will find the “rolling frame” to be used for the TF-IDF and Cosine Similarity analysis

```
if i < len(sector_files)-3:
    rolling_path_of_four_transcript = sector_files[i:i+4]
```

Finally, by running each of the `getFeature5to22`, `getFeature23to43`, `getFeature44to73`, and `fea_ext_list74to85` functions will get a list of features that is then written to the designated earnings transcript.csv. The earnings transcript.csv will then look like Figure 72 as shown below, consisting of the meta data, contents of the earnings transcript, EPS value, and features 5 to 96. This would be repeated for all 2469 stocks, and 85620 earnings transcripts:

Figure 72 - Earnings transcrip.csv example

Machine Learning:

```
import requests
from bs4 import BeautifulSoup
import json
import pandas as pd
import time
import numpy as np
from datetime import datetime
import yfinance as yf
import os
import glob
import regex as re
import csv
import statistics

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout
import matplotlib.pyplot as plt
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score


from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
```

Figure 73 - Importing relevant machine learning libraries

```
def remove_none_types(mydf):
   if mydf.isnull().values.any():
       return False
   else:
       return True

X_data = []
Y_data = []

sectors = ["consumer-durables"]
for sector in sectors:
   filelist = os.listdir("../sectors/"+sector)
   try:
       filelist.remove('.DS_Store')
   except:
       pass

   for stock in filelist:
       sector_files =
glob.glob('../sectors/'+str(sector)+'/'+str(stock)+'/'+str(stock)+'20*[0-9]**[0-9]*[1-4].*')
       sector_files.sort(reverse=True)

       testpath = sector_files[0]
```

```
        testdata_from_csv = pd.read_csv(testpath)
        try:
            market_cap = float(testdata_from_csv.iloc[[-1]].values[0][0])
            if (market_cap > 100000) and (market_cap < 1000000000000):
                for i in range(0, len(sector_files)): # for every .csv path of that stock
                    path = sector_files[i]
                    data_from_csv = pd.read_csv(path)
                    Checks_None = remove_none_types(data_from_csv)
                    if (Checks_None == True):
                        single_X_data = (np.array((data_from_csv[3:73].astype(float))).flatten())
                        if (single_X_data.shape[0]) == 70:
                            X_data.append(single_X_data)
                            single_Y_data = np.array(data_from_csv[85:95].astype(float)).flatten()
                            Y_data.append(single_Y_data)
        except:
            pass
```

Figure 73 - Reads features from .csv files

In the example in Figure 73, it cycles through all consumer durable stocks and their respective earnings transcripts. It also includes an adjustable market cap filter where you can tell the programme to only include stocks between 5 billion to 20 billion market cap, for example. From these files, it will split the data into an X_data and Y_data set, where the X_data represents the dependent variable, and Y_data represents the independent variable.

```
X_data_ORIGINAL = np.array(X_data)
Y_data_ORIGINAL = np.array(Y_data)
scaler = StandardScaler()
X_data = scaler.fit_transform(X_data)
Y_data = Y_data_ORIGINAL[:,9]
```

Figure 74 - Code for scaling and splitting data

The X_data values are normalised (scaled) within a 0 to 1 range. This is a necessary step of feature engineering since it improves the effectiveness and accuracy of the model as it will avoid numerical issues, allows algorithm to converge more efficiently, learn patterns more fluently, and prevent any feature from having too much influence on the model.

```
n_bins = 2
# Divide the data into n_bins intervals using quantiles
bin_edges = np.quantile(Y_data, np.linspace(0, 1, n_bins + 1))
# Assign each data point to a specific bin based on its value
bin_indices = np.digitize(Y_data, bin_edges) - 1
# Make sure the indices are within the range of the number of bins
bin_indices = np.minimum(bin_indices, n_bins - 1)
# Label each bin with a categorical label
bin_labels = ["bin_{}".format(i + 1) for i in range(n_bins)]
# Transform the continuous Y_data into a categorical variable
Y_categorical = np.array([bin_labels[i] for i in bin_indices])
```

Figure 75a - Code for binning data

```
    for i in range(n_bins):
        print("Bin {}: [{}, {}]".format(i + 1, bin_edges[i], bin_edges[i + 1]))
```

```
Bin 1: [-0.7640156453715776, 0.01531878240611495]
Bin 2: [0.01531878240611495, 2.9402985074626864]
```

Figure 75b - The range for each bins

The Y_data represents a continuous stream of stock returns. However, I have chosen to use a binary classification model (why?) to predict the stock returns as predicting purely the continuous form of the stock return accurately is inaccurate and subject to a lot of different factors. Hence I've used a binning method where a continuous variable is divided into a set of discrete intervals (bins). Each bin represents a range of values within the continuous variable, and the observations are then assigned to the respective bin based on their values.

Although Y_data reflects a continuous flow of stock returns, I opted to implement a binary classification model to predict these returns. The reason for this is that predicting the continuous form of stock returns is imprecise and influenced by numerous other variables (i.e. momentum, volume, etc.). Therefore, to apply a binary classification technique, I employed a binning technique that divides the continuous variable into distinct intervals or "bins". Each bin corresponds to a specific range of values within the continuous variable.

For example, in Figure 75b, we can see that stock returns that fall between -0.76 to 0.015 will fall under the label "Bin 1", while stock returns that fall between 0.015 to 2.94 will fall under the label "Bin 2".

```
X_train, X_test, Y_train, Y_test = train_test_split(X_data, Y_categorical, test_size=0.33,
random_state=42)

#Starting off with random forest classifer
rfc = RandomForestClassifier(max_depth=500, n_estimators=2000)

rfc.fit(X_train, Y_train)
score = rfc.score(X_test, Y_test)
Y_pred = rfc.predict(X_test)

print(metrics.classification_report(Y_test, Y_pred))

-------------------------------------------------------------------------------------
Output:
              precision    recall  f1-score   support

       bin_1       0.49      0.67      0.57       212
```

```
       bin_2       0.59      0.41      0.49       248

    accuracy                           0.53       460
   macro avg       0.54      0.54      0.53       460
weighted avg       0.55      0.53      0.52       460
```

Figure 76 - Random Forest Classifier

Next, I divided X_data and Y_data into two separate sets, with 67% of the data allocated to the training set, and the remaining 33% assigned to the testing set. The purpose of the training dataset is to facilitate the training of the algorithm, while the testing dataset will be used to evaluate the algorithm's effectiveness.

I then train the random forest classifier model with X_train and Y_train, then evaluate the algorithm's performance by inputting X_test to get the Y_predicted, and compare Y_predicted with Y_test. In the example above the model is trained using all consumer durables stocks.

$$
\begin{aligned}
precision &= \frac{TP}{TP + FP} \\
recall &= \frac{TP}{TP + FN} \\
F1 &= \frac{2 \times precision \times recall}{precision + recall} \\
accuracy &= \frac{TP + TN}{TP + FN + TN + FP}
\end{aligned}
$$

Figure 77 - precision, recall, F1, and accuracy formulas

The evaluation scores are all derived from True-Positive, False-Positive, True-Negative, and False-Positive.

Precision:
Number of times the model is correct in that category

Recall:
Number of times the model detects that category

F1-Score:
Harmonic average of precision and recall of that category

Accuracy:
The overall number of times the model is correct (all categories)

With reference to Figure 76, it can predict the label “bin_2” (which represents stock returns between +1% to +294%) with 59% precision. In the context of stock predictions, precision is a more useful indicator than recall rate to evaluate the overall performance of the model in predicting the correct direction of the stock price since we can afford to miss out on some gains, but we want to minimise losses.

```
Y_pred_proba = rfc.predict_proba(X_test)
Y_pred = np.where(Y_pred_proba[:,1] > 0.56, 'bin_2', 'bin_1')

print(metrics.classification_report(Y_test, Y_pred))

-------------------------------------------------------------------------------------
Output:
              precision    recall  f1-score   support

       bin_1       0.49      0.90      0.64       212
       bin_2       0.71      0.21      0.32       248

    accuracy                           0.53       460
   macro avg       0.60      0.56      0.48       460
weighted avg       0.61      0.53      0.47       460
```

Figure 77 - Add filters to the random forest classifier model

With reference to Figure 77, we can further improve on the precision rate by only allowing the algorithm to classify a particular instance to the “bin_2” label if it’s over 56% confident (instead of the default >50% confidence level). Now, we can see an improvement in terms of precision rate to 71%.

In other words, out of the 248 stock instances that fall under “bin_2”, it predicts 52 of these stocks to have positive returns. And out of the 52 predictions, it guesses 37 of them correctly.

```
import pickle
#save to file using pickle
with open("../info/rfc_consumer_durables_model.pkl", "wb") as f:
   pickle.dump(rfc, f)
```

Figure 78 - Saving the machine learning model as a .pkl file

As previously specified in the Design specification, the machine learning model will be saved as a .pkl file using pickle, for later use.

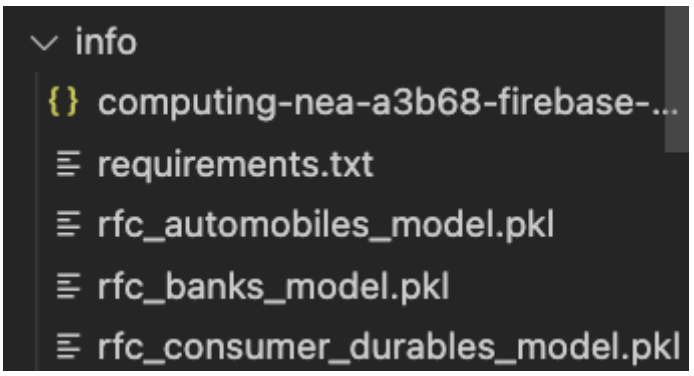


Figure 79 - .pkl files under "info" sub-directory

As such, I have stored each .pkl random forest classifier under the sub-directory "info". (note that these file sizes exceed 150mb and cannot be uploaded on Github). The user can later choose between each of these .pkl files and use the one that suits their needs.

Basic User Interface:

```
import transformers
from transformers import BertTokenizer, BertForSequenceClassification, pipeline
import torch
import numpy as np
import textstat

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel

import requests
from bs4 import BeautifulSoup
import json
import pandas as pd
import time
import numpy as np
from datetime import datetime
import yfinance as yf
import os
import glob
import regex as re
import csv
import statistics
import nltk
nltk.download('punkt')

sentiment_finbert =
BertForSequenceClassification.from_pretrained('yiyanghkust/finbert-tone',num_labels=3)
sentiment_tokenizer = BertTokenizer.from_pretrained('yiyanghkust/finbert-tone')

sentiment_nlp = pipeline("text-classification", model=sentiment_finbert,
tokenizer=sentiment_tokenizer)
```

```
fls_finbert =
BertForSequenceClassification.from_pretrained('yiyanghkust/finbert-fls',num_labels=3)
fls_tokenizer = BertTokenizer.from_pretrained('yiyanghkust/finbert-fls')
fls_nlp = pipeline("text-classification", model=fls_finbert, tokenizer=fls_tokenizer)

import firebase_admin
from firebase_admin import credentials
from firebase_admin import firestore
from firebase_admin import auth
import pickle
```

Figure 80 - Importing the relevant libraries

```
with open("info/rfc_banks_model.pkl", "rb") as f:
   banks_rfc = pickle.load(f)

with open("info/rfc_automobiles_model.pkl", "rb") as f:
   automobiles_rfc = pickle.load(f)

with open("info/rfc_consumer_durables_model.pkl", "rb") as f:
   consumer_durables_rfc = pickle.load(f)
```

Figure 81 - Loading the machine learning models using Pickle

```
# connect to firebase
cred = credentials.Certificate('info/computing-nea-a3b68-firebase-adminsdk-m2onq-44557de7c9.json')
app = firebase_admin.initialize_app(cred)

db = firestore.client()
```

FIgure 82 - Connecting to my Firebase database using an API key

```
def submit_user_details(user_id):
   year = int(input("What year were you born?"))
   month = int(input("What month were you born?"))
   day = int(input("What day were you born?"))

   dob = datetime(year=year, month=month, day=day)

   first_name = input("What's your first name?")
   last_name = input("What's your last name?")
   sex = input("What's your sex?")

   user_details = {
       'dob': dob,
       'first_name': first_name,
       'last_name': last_name,
       'sex': sex
   }

   doc_ref = db.collection(u'userdetails').document(user_id)
   doc_ref.set(user_details)
```

Figure 83 - Function that is used to submit user details, and stores it as a Python Object

```
email = input("Enter your email: ")
password = input("Enter your password: ")

try:
   user = auth.create_user(email=email, password=password)

except Exception as exception_message:
   print("There has been an error creating an account:", exception_message)

user_id = user.uid

print("User ID:", user_id)

submit_user_details(user.uid)
```

Figure 84 - Code for creating a user and store it in Firebase

```
email = input("Enter your email: ")
password = input("Enter your password: ")

try:
   user = auth.get_user_by_email(email)
   print("You have successfully logged in!")
except Exception as exception_message:
   print("There has been an error signing in:", exception_message)

user_id = user.uid

print("User ID:", user_id)
```

Figure 85 - Code to login and access their user information

Figure 85 shows the code to login to a user's account. After logging in, the user_id will help locate other records from the user, such as their "saved watchlists", "view history" and "user details".

```
# Update user email
new_email = 'replaceEmail@gmail.com'
try:
   user = auth.update_user(
       user.uid,
       email=new_email,
   )
   print("You have successfully updated your email address")
except Exception as exception_message:
```

```
    print("There has been an error updating your account:", exception_message)
```

Figure 86 - Allows user to update their emails

```
# delete user account
AreYouSure = input("Are you sure you want to delete this account? -- Y or N")
if AreYouSure == "Y":
    try:
        auth.delete_user(user.uid)
        # also deletes it from the userdetails collection
        doc_ref = db.collection('userdetails').document(user.uid)
        doc_ref.delete()
        print("You have successfully deleted this account")
    except Exception as exception_message:
        print("There has been an error deleting your account:", exception_message)

else:
    print("You have not deleted this account")
```

Figure 87 - Allows user to delete their user account

```
stock = input("enter stock")
year = input("enter year")
quarter = input("enter quarter")
```

Figure 89 - The user can enter what stock they want to predict, the year, and which quarter it belongs to.

```
stock = "JPM"
year = 2
quarter = 1
document_name, stock_data, stock_data_with_date = find_stock_data(stock, year, quarter,
user_id)
```

Figure 90a - For example if user wants to predict the stock returns for JP Morgan in 2022, Q1. (find_stock_data function will be expanded in Figure 92)

```
print(stock_data)
-------------------------------------------------------------------------------------
-
Output:
{'stock': 'JPM',
 'year': 2022,
 'quarter': 1,
 'direction': 'Down',
 'probabilityUp': 0.465,
 'probabilityDown': 0.535}
```

Figure 90b - What stock_data object looks like by running the code in Figure 90a

```
print("Your current watchlists are:")
# from userdetails collection
user_ref = db.collection('userdetails').document(user_id)
collections = user_ref.collections()
for collection in collections:
   print(collection.id)

watchlist_name = input("Please enter a watchlist to add this stock to, or enter 'N' if you don't want
to save it to your watchlists (the programme will automatically create a watchlist for you if it
doesn't exist)")
if watchlist_name == "N":
   exit()
else:
   try:
       watchlist_ref = user_ref.collection(watchlist_name)
       stock_ref = watchlist_ref.document(document_name)
       stock_ref.set(stock_data_with_date)
       print(f'Successfully added {document_name} to the watchlist {watchlist_name}!')
   except:
       print("There has been an error with adding to your watchlist:", exception_message)
```

Figure 91a - Code for users to save stock data to their watchlists

With reference to Figure 91, it allows users to save the current stock prediction object **("stock_data")** to a custom "watchlist" on Firebase. It will first tell the user what their available watchlists are, and the user can select which one he want's to add the stock_data object to. If the watchlist doesn't exist, it will be created.

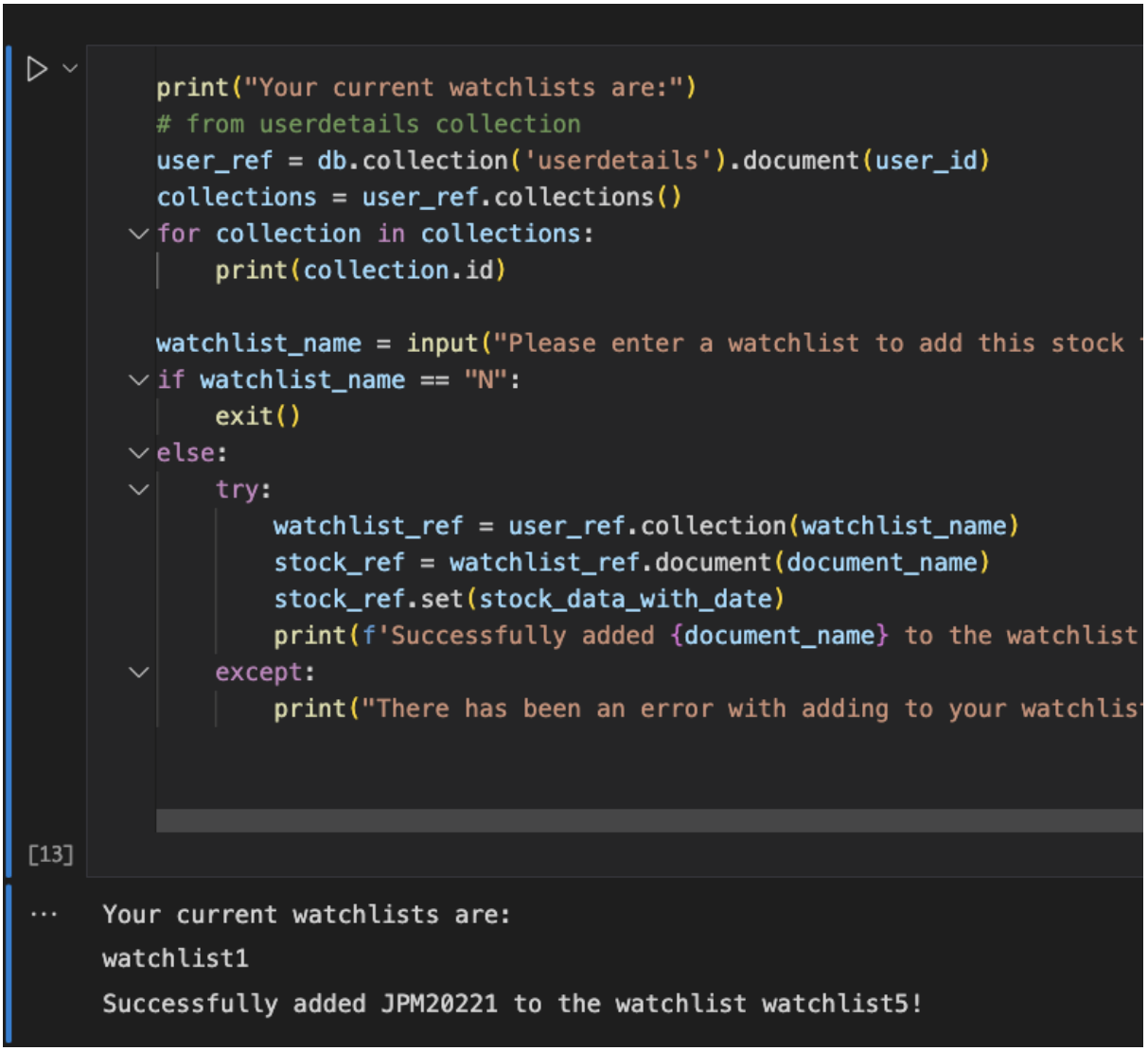


Figure 91b - Running the code in Figure 91a

Figure 91b shows what will happen when you save the stock_data from JPM 2022 Q1 to the user specified “watchlist1”.

```
def find_stock_data(stock, year, quarter, user_id):
   sectorlist = ["automobiles", "banks", "capital-goods", "commercial-services",
   "consumer-durables", "consumer-retailing", "consumer-services",
"diversified-financials",
   "energy", "food-beverage-tobacco", "healthcare", "household", "insurance",
"materials", "media",
   "pharmaceuticals-biotech", "real-estate", "retail", "semiconductors", "software",
   "tech", "telecom", "transportation", "utilities"]

   for sectorin in sectorlist:
       filelist = os.listdir("sectors/"+sectorin)
       try:
           filelist.remove('.DS_Store')
       except:
           pass
       for stockin in filelist:
```

```
            if stockin == stock:
                sector = sectorin

    if sector == "banks":
        loaded_rfc = banks_rfc

    elif sector == "automobiles":
        loaded_rfc = automobiles_rfc

    elif sector == "consumer-durables":
        loaded_rfc  = consumer_durables_rfc

    else:
        print("this sector doesn't have a preset model")
        exit()

    document_name = stock + str(year) + str(quarter)
    doc_ref = db.collection(u'stockpresets').document(document_name)

    # if the stock data has already been logged inside the "stockpresets" collection,
then it will fetch this data from firebase instead of computing everything all over
again
    if doc_ref.get().exists:
        print("The stock you are searching is already in the global stockpresets
collection, the program does not need to compute it again")

        stock_data = doc_ref.get().to_dict()

    else:
        print("The stock you are searching is not already in the global stockpresets
collection, please wait until the program computes the results.")

        document_name = stock + str(year) + str(quarter)

        doc_ref = db.collection(u'stockpresets').document(document_name)

        ticker = yf.Ticker(stock)
        market_cap = ticker.fast_info['market_cap']
        mytranscript, exact_date = get_transcript(stock, year, quarter)
        earnings_list = ticker.get_earnings_dates(limit=30) #uses yfinance and gets
historical eps consensus/reported data
        earnings_list.reset_index(inplace=True)
        earnings_list = earnings_list[earnings_list['Surprise(%)'].notna()]
        eps_list = earnings_list.reset_index(drop=True)
        for index in range(0, len(eps_list)):
            percentage_surprise = float(eps_list['Surprise(%)'][index])
            date = eps_list['Earnings Date'][index]
            date = date.replace(tzinfo=None)
            date = date.to_pydatetime()
            mycase = check_day_diff(date, exact_date)
            if mycase == True:
                myEPS = percentage_surprise
                break
        transcript_safe_harbour, transcript_questions = split_transcript(mytranscript)
        speaker_names = get_file_speaker_names(sector, stock)
        analyst_names = find_analyst_names(speaker_names, transcript_questions)
```

```
        analyst_sentences, management_sentences =
get_analyst_management_sentences(analyst_names, transcript_questions)
        fea_ext_list5to22 = getFeature5to22(transcript_safe_harbour, speaker_names)
        fea_ext_list23to43 = getFeature23to43(analyst_sentences, management_sentences,
speaker_names)
        fea_ext_list44to73 = getFeature44to73(mytranscript, speaker_names)
        X_dataset = [myEPS] + fea_ext_list5to22 + fea_ext_list23to43 + fea_ext_list44to73
+ [market_cap] + [stock] + [year] + [quarter] + [sector]

        X_data = np.array(X_dataset)
        Single_X_data = X_data[:70]
        Single_X_data = Single_X_data.reshape(1,-1)


        Y_pred = loaded_rfc.predict(Single_X_data)

        Y_pred_proba = loaded_rfc.predict_proba(Single_X_data)

        if Y_pred == "bin_2":
            direction = "Up"

        elif Y_pred == "bin_1":
            direction = "Down"

        stock_data = {
            "stock": stock,
            "year": year,
            "quarter": quarter,
            "direction": direction,
            "probabilityUp": Y_pred_proba[0][1],
            "probabilityDown": Y_pred_proba[0][0]
        }

        doc_ref = db.collection(u'stockpresets').document(document_name)
        doc_ref.set(stock_data)
        print("The stock information that will be uploaded to the database is:",
stock_data)
        print("Successfully added the contents to the global database")

    try:
        user_ref = db.collection('viewhistory').document(user_id)

        # checks if the userID document exists or not, if not, it will be created
        if not user_ref.get().exists:
            user_ref.set({})

        current_time = datetime.now()
        stock_view_filename = stock + current_time.strftime("%Y-%m-%d %H:%M:%S")

        # checks if the history collection exists
        history_ref = user_ref.collection('history')
        stock_ref = history_ref.document(stock_view_filename)

        stock_data_with_date = stock_data.copy()
        stock_data_with_date["DateViewed"] = current_time
        stock_ref.set(stock_data_with_date)
```

```
        print('Successfully updated the stock to your view history')
    except Exception as exception_message:
            print("There has been an error with updating your view history:",
exception_message)

    return document_name, stock_data, stock_data_with_date
```

Figure 92 - function for finding stock data and performing the relevant functions via Firebase

The code in Figure 92 will find the input features by calling functions from Figure 93. It will then load this list of input features to the machine learning model and receive an output. The output will be further processed to produce the "**stock_data"** object. It will also save the user's request in their "view history", as well as the "global database" by communicating with Firebase. The relevant exception handling makes sure the programme will tell the user the exception message.

```
def check_day_diff(date1, exact_date):
    day_difference = abs((exact_date - date1).days)
    if day_difference < 5:
        return True
    else:
        return False

def find_transcript(stock, date_yr, date_qtr):
    try:
        url = 'https://roic.ai/transcripts/' + stock + '?y=' + str(date_yr) + '&q=' +
str(date_qtr)
        html_text = requests.get(url).text
        soup = BeautifulSoup(html_text, "html.parser")
        script = soup.find_all('script')[15].text.strip()
        data = json.loads(script)
        transcript_data = data['props']['pageProps']['transcriptdata']['content']  #
loads the transcript content
        json_dates = data['props']['pageProps']['data']['data']['earningscalls']
        for info in json_dates:
            if info["year"] == year and info["quarter"] == quarter:
                exact_date = datetime.strptime(info["date"], '%Y-%m-%d %H:%M:%S')
    except:
        print("sorry, the transcript cannot be found, please enter a different date")
        exit()

    return transcript_data, exact_date

def get_transcript(stock, year, quarter):
    mytranscript = ""
    mytranscript, exact_date = find_transcript(stock, year, quarter)
    mytranscript = re.sub(r'[^A-Za-z0-9.,:!\'\n ]', '', mytranscript)
    mytranscript = mytranscript.replace(".", ". ")
    mytranscript = re.sub('[^\S\n]+', ' ', mytranscript) #replaces multiple spaces to
single space, without deleting newlines \n in the process
    mytranscript = mytranscript.splitlines() # finds transcript
    return mytranscript, exact_date
```

```
def split_transcript(mytranscript):
   transcript_safe_harbour, transcript_questions = "", ""
   for i in range(0, len(mytranscript)):
       speech_bubble = mytranscript[i].lower()
       speech_bubble = re.sub(r'[^\w\s:]', ' ', speech_bubble) # regex: replaces all
punctuations (except for ":") with 1 open space so the IF condition below can run
smoothly
       if (i > 1) and (("operator:" in speech_bubble) and (("question" in speech_bubble)
or ("go ahead" in speech_bubble) or ("operator instructions" in speech_bubble))):
           transcript_safe_harbour = mytranscript[0:i]
           transcript_questions = mytranscript[i:]
           break
       elif (i > 1 ) and ("operator" in speech_bubble) and ("question" in
speech_bubble):
           transcript_safe_harbour = mytranscript[0:i+1]
           transcript_questions = mytranscript[i+1:]
           break
       elif (i > 1 ) and ("operator:" in speech_bubble) and ("first" in speech_bubble):
           transcript_safe_harbour = mytranscript[0:i]
           transcript_questions = mytranscript[i:]
           break

   return transcript_safe_harbour, transcript_questions

def get_file_speaker_names(sector, stock):
   write_path = "sectors/"+sector+"/"+stock+"/"+"speaker names.csv"
   speaker_names = np.loadtxt(write_path, delimiter='\t', dtype=str)
   return speaker_names

def find_analyst_names(speaker_names, transcript_questions):
   analyst_names = []
   for index in range(0, len(transcript_questions)-1):
       speech_bubble = transcript_questions[index].lower()
       speech_bubble = re.sub(r'[^\w\s:]', ' ', speech_bubble) # regex: replaces all
punctuations (except for ":") with 1 open space
       if ("operator:" in speech_bubble) or ("operator :" in speech_bubble):
           for name in speaker_names:
               namelist = name.split()
               if (name.lower() != "operator") and ("representative" not in
name.lower()) and ("corporate" not in name.lower()) and ("company" not in name.lower()):
                   for name_2 in namelist: # cycle through each name in the name_list
                       name_2 = name_2.lower()
                       if ((" "+name_2+" " in speech_bubble) and len(name_2) > 2) and
(("end" not in speech_bubble) and ("closing" not in speech_bubble) and (("turn" not in
speech_bubble) or (("over" not in speech_bubble))))) and (name_2 in
transcript_questions[index+1].lower()):
                           analyst_names.append(name)
                   if "unidentified" in name.lower().split(): # finds name such as
"Unidentified Analyst"
                       analyst_names.append(name)

   analyst_names = list(set(analyst_names)) # replaces duplicates
   return analyst_names

def get_analyst_management_sentences(analyst_names, transcript_questions):
```

```
    analyst_sentences = []
    management_sentences = []
    for index in range(0, len(transcript_questions)-1):
        speech_bubble = transcript_questions[index]
        colon_pos = speech_bubble.find(":")
        speaker_name = speech_bubble[:colon_pos]
        if index > 3:
            for name in analyst_names:
                namelist = name.split()
                if (speech_bubble not in analyst_sentences):
                    if speaker_name in analyst_names:
                        analyst_sentences.append(speech_bubble)

                    elif (speaker_name.lower() == "operator") or (speaker_name.lower() ==
"operator "):
                        pass

                    elif ((namelist[0] in speaker_name) or (namelist[-1] in
speaker_name)) and (("operator:" in transcript_questions[index-1].lower()) or ("operator
:" in transcript_questions[index-1].lower())):

                        analyst_names.append(speaker_name)
                        analyst_sentences.append(speech_bubble)

    # get management sentence
    for index in range(0, len(transcript_questions)-2):
        speech_bubble = transcript_questions[index]
        colon_pos = speech_bubble.find(":")
        speaker_name = speech_bubble[:colon_pos]

        # dont want operator's sentence
        if (speaker_name.lower() == "operator") or (speaker_name.lower() == "operator "):
            pass

        elif (speech_bubble not in analyst_sentences):
            management_sentences.append(speech_bubble)
    return analyst_sentences, management_sentences

class TrieNode:
    def __init__(self):
        self.children = {}
        self.is_word = False
        self.fail = None
        self.word = None

class AhoCorasick:
    def __init__(self, words):
        self.root = TrieNode()
        self.build_trie(words)
        self.build_ac_automata()

    def build_trie(self, words):
        for word in words:
            node = self.root
            for char in word:
                if char not in node.children:
```

```
                    node.children[char] = TrieNode()
                node = node.children[char]
            node.is_word = True
            node.word = word

    def build_ac_automata(self):
        queue = []

        for node in self.root.children.values():
            queue.append(node)
            node.fail = self.root

        while len(queue) > 0:
            node = queue.pop(0)
            for char, child in node.children.items():
                queue.append(child)
                fail_node = node.fail
                while fail_node is not None and char not in fail_node.children:
                    fail_node = fail_node.fail
                if fail_node is None:
                    child.fail = self.root
                else:
                    child.fail = fail_node.children[char]
                child.is_word |= child.fail.is_word

    def remove_words(self, text):
        node = self.root
        new_text = text
        for i, char in enumerate(text):
            while node is not None and char not in node.children:
                node = node.fail
            if node is None:
                node = self.root
                continue
            node = node.children[char]
            if node.is_word:
                new_text = new_text.replace(node.word, '')
        return new_text


def getFeature5to22(pre_release, speaker_names):
    new_speaker_names = [word + ':' for word in speaker_names]

    ac = AhoCorasick(new_speaker_names)

    net_sentiment_list = []
    flesch_list = []

    n_flslist = []
    s_flslist = []
    ns_flslist = []

    net_positive = 0
    net_negative = 0
    net_neutral = 0
```

```
    feature_extract_5 = 0
    feature_extract_6 = 0
    feature_extract_7 = 0
    feature_extract_8 = 0
    feature_extract_9 = 0
    feature_extract_10 = 0
    feature_extract_11 = 0
    feature_extract_12 = 0
    feature_extract_13 = 0
    feature_extract_14 = 0
    feature_extract_15 = 0
    feature_extract_16 = 0
    feature_extract_17 = 0
    feature_extract_18 = 0
    feature_extract_19 = 0
    feature_extract_20 = 0
    feature_extract_21 = 0
    feature_extract_22 = 0

    try:
        for speech_bubble in pre_release:
            try:
                new_speech_bubble = ac.remove_words(speech_bubble)
                new_speech_bubble = re.sub('[^\S\n]+', ' ', new_speech_bubble)

                if new_speech_bubble[0] == " ":
                    new_speech_bubble = new_speech_bubble.replace(" ", "", 1) # replace
the first space bar with an empty string, for example ' is is string is is string' to
'is is string is is string'

                # gets text complexity
                flesch_score = textstat.flesch_reading_ease(new_speech_bubble)
                flesch_list.append(flesch_score)

                new_speech_bubble_list =
split_paragraph_into_sentences(new_speech_bubble)

                fls_results = fls_nlp(new_speech_bubble_list)

                for i in range(0, len(new_speech_bubble_list)):
                    sentence = new_speech_bubble_list[i]
                    if fls_results[i]['label'] == 'Not FLS':
                        n_flslist.append(sentence)
                    elif fls_results[i]['label'] == 'Specific FLS':
                        s_flslist.append(sentence)
                    elif fls_results[i]['label'] == 'Non-specific FLS':
                        ns_flslist.append(sentence)

            except:
                pass

        try:
            feature_extract_9, feature_extract_10, feature_extract_11,
feature_extract_12, fls1_sentiment_list, net1_positive, net1_negative, net1_neutral =
get_fls_features(s_flslist)
```

```
        except:
            pass

        try:
            feature_extract_13, feature_extract_14, feature_extract_15,
feature_extract_16, fls2_sentiment_list, net2_positive, net2_negative, net2_neutral =
get_fls_features(ns_flslist)

        except:
            pass

        try:
            feature_extract_17, feature_extract_18, feature_extract_19,
feature_extract_20, fls3_sentiment_list, net3_positive, net3_negative, net3_neutral =
get_fls_features(n_flslist)

        except:
            pass

        try:
            numb_s_flslist = len(s_flslist)
            numb_ns_flslist = len(ns_flslist)
            numb_n_flslist = len(n_flslist)
            total = numb_s_flslist + numb_ns_flslist + numb_n_flslist

            feature_extract_21 = numb_s_flslist/total
            feature_extract_22 = numb_ns_flslist/total

        except:
            pass

        try:
            net_positive = net1_positive + net2_positive + net3_positive
            net_negative = net1_negative + net2_negative + net3_negative
            net_neutral = net1_neutral + net2_neutral + net3_neutral
            net_sentiment_list = fls1_sentiment_list + fls2_sentiment_list +
fls3_sentiment_list
        except:
            pass

        try:
            feature_extract_5 = statistics.mean(net_sentiment_list)
        except:
            pass

        try:
            feature_extract_6 = net_positive/(net_negative+net_positive+net_neutral)
            feature_extract_7 = net_negative/(net_negative+net_positive+net_neutral)
        except:
            pass

        try:
            feature_extract_8 = statistics.mean(flesch_list)
        except:
            pass
```

```
        fea_ext_list5to22 = [feature_extract_5, feature_extract_6, feature_extract_7,
feature_extract_8, feature_extract_9, feature_extract_10, feature_extract_11,
feature_extract_12, feature_extract_13, feature_extract_14, feature_extract_15,
feature_extract_16, feature_extract_17, feature_extract_18, feature_extract_19,
feature_extract_20, feature_extract_21, feature_extract_22]

    except:
        fea_ext_list5to22 = [0]*18

    return fea_ext_list5to22


def get_fls_features(flslist):
    fls_sentiment_list = []
    net_positive = 0
    net_negative = 0
    net_neutral = 0

    feature_extract_1 = 0
    feature_extract_2 = 0
    feature_extract_3 = 0
    feature_extract_4 = 0

    for each_fls in flslist:
        sentiment_result = sentiment_nlp(each_fls)
        sentiment_score, positivity_value = map_sentiments(sentiment_result)
        fls_sentiment_list.append(sentiment_score)

        if positivity_value == "positive":
            net_positive += 1

        elif positivity_value == "negative":
            net_negative += 1

        else:
            net_neutral += 1
    try:
        feature_extract_1 = statistics.mean(fls_sentiment_list)
    except:
        pass
    try:
        feature_extract_2 = net_positive/(net_positive+net_negative+net_neutral)
    except:
        pass
    try:
        feature_extract_3 = net_negative/(net_positive+net_negative+net_neutral)
    except:
        pass
    try:
        feature_extract_4 = textstat.flesch_reading_ease(' '.join(flslist))
    except:
        pass

    return feature_extract_1, feature_extract_2, feature_extract_3, feature_extract_4,
fls_sentiment_list, net_positive, net_negative, net_neutral
```

```
def getFeature23to43(analyst_speech, management_speech, speaker_names):
    new_speaker_names = [word + ':' for word in speaker_names]
    ac = AhoCorasick(new_speaker_names)

    n_flslist = []
    s_flslist = []
    ns_flslist = []

    questions_complex_list = []
    reply_complex_list = []
    net_text_complex_list  = []

    # list of sentiments for all S_FLS, N_FLS, NS_FLS classes
    s_fls_sentiment_list = []
    n_fls_sentiment_list = []
    ns_fls_sentiment_list = []

    # list of sentiments for all sentences that are identified as a "question"
    question_sentiment_list = []

    # list of sentiments for all sentences that are identified as a "reply"
    reply_sentiment_list = []

    # list of sentiments for all sentences in the Q&A section
    net_sentiment_list = []

    net_positive = 0
    net_negative = 0
    net_neutral = 0

    Qpositive = 0
    Qnegative = 0
    Qneutral = 0

    Rpositive = 0
    Rnegative = 0
    Rneutral = 0

    SFLSpositive = 0
    SFLSnegative = 0
    SFLSneutral = 0

    NSFLSpositive = 0
    NSFLSnegative = 0
    NSFLSneutral = 0

    NFLSpositive = 0
    NFLSnegative = 0
    NFLSneutral = 0

    feature_extract_23 = 0
    feature_extract_24 = 0
    feature_extract_25 = 0
    feature_extract_26 = 0
```

```
    feature_extract_27 = 0
    feature_extract_28 = 0
    feature_extract_29 = 0
    feature_extract_30 = 0

    feature_extract_31 = 0
    feature_extract_32 = 0
    feature_extract_33 = 0
    feature_extract_34 = 0

    feature_extract_35 = 0
    feature_extract_36 = 0
    feature_extract_37 = 0

    feature_extract_38 = 0
    feature_extract_39 = 0
    feature_extract_40 = 0

    feature_extract_41 = 0
    feature_extract_42 = 0
    feature_extract_43 = 0

    try:
        for speech_bubble in analyst_speech:
            try:
                new_speech_bubble = ac.remove_words(speech_bubble)
                new_speech_bubble = re.sub('[^\S\n]+', ' ', new_speech_bubble)

                if new_speech_bubble[0] == " ":
                    new_speech_bubble = new_speech_bubble.replace(" ", "", 1) # replace
the first space bar with an empty string, for example ' is is string is is string' to
'is is string is is string'

                # gets text complexity
                flesch_score = textstat.flesch_reading_ease(new_speech_bubble)
                questions_complex_list.append(flesch_score)
                net_text_complex_list.append(flesch_score)

                new_speech_bubble_list =
split_paragraph_into_sentences(new_speech_bubble)

                for i in range(0, len(new_speech_bubble_list)):
                    sentence = new_speech_bubble_list[i]
                    sentiment_result = sentiment_nlp(sentence)
                    sentiment_score, positivity_value = map_sentiments(sentiment_result)
                    question_sentiment_list.append(sentiment_score)
                    net_sentiment_list.append(sentiment_score)

                    if positivity_value == "positive":
                        net_positive += 1
                        Qpositive += 1

                    elif positivity_value == "negative":
                        net_negative += 1
                        Qnegative += 1
```

```
                    else:
                        net_neutral += 1
                        Qneutral += 1
            except:
                pass

        for speech_bubble in management_speech:
            try:
                new_speech_bubble = ac.remove_words(speech_bubble)
                new_speech_bubble = re.sub('[^\S\n]+', ' ', new_speech_bubble)

                if new_speech_bubble[0] == " ":
                    new_speech_bubble = new_speech_bubble.replace(" ", "", 1) # replace
the first space bar with an empty string, for example ' is is string is is string' to
'is is string is is string'

                # gets text complexity
                flesch_score = textstat.flesch_reading_ease(new_speech_bubble)
                reply_complex_list.append(flesch_score)
                net_text_complex_list.append(flesch_score)

                new_speech_bubble_list =
split_paragraph_into_sentences(new_speech_bubble)

                fls_results = fls_nlp(new_speech_bubble_list)

                for i in range(0, len(new_speech_bubble_list)):
                    sentence = new_speech_bubble_list[i]
                    if fls_results[i]['label'] == 'Not FLS':
                        n_flslist.append(sentence)
                    elif fls_results[i]['label'] == 'Specific FLS':
                        s_flslist.append(sentence)
                    elif fls_results[i]['label'] == 'Non-specific FLS':
                        ns_flslist.append(sentence)

            except:
                pass

        # for "n_flslist":
        n_fls_sentiment_list, reply_sentiment_list, Rpositive, Rnegative, Rneutral,
NFLSpositive, NFLSnegative, NFLSneutral, net_positive, net_negative, net_neutral =
get_SentimentLists_from_FLS(net_sentiment_list, n_flslist, n_fls_sentiment_list,
reply_sentiment_list, Rpositive, Rnegative, Rneutral, NFLSpositive, NFLSnegative,
NFLSneutral, net_positive, net_negative, net_neutral)

        # for "s_flslist":
        s_fls_sentiment_list, reply_sentiment_list, Rpositive, Rnegative, Rneutral,
SFLSpositive, SFLSnegative, SFLSneutral, net_positive, net_negative, net_neutral =
get_SentimentLists_from_FLS(net_sentiment_list, s_flslist, s_fls_sentiment_list,
reply_sentiment_list, Rpositive, Rnegative, Rneutral, SFLSpositive, SFLSnegative,
SFLSneutral, net_positive, net_negative, net_neutral)

        # for "ns_flslist":
        ns_fls_sentiment_list, reply_sentiment_list, Rpositive, Rnegative, Rneutral,
NSFLSpositive, NSFLSnegative, NSFLSneutral, net_positive, net_negative, net_neutral =
get_SentimentLists_from_FLS(net_sentiment_list, ns_flslist, ns_fls_sentiment_list,
```

```
reply_sentiment_list, Rpositive, Rnegative, Rneutral, NSFLSpositive, NSFLSnegative,
NSFLSneutral, net_positive, net_negative, net_neutral)

        feature_extract_23 = statistics.mean(net_sentiment_list)

        try:
            feature_extract_24 = net_positive/(net_positive+net_negative+net_neutral)
            feature_extract_25 = net_negative/(net_positive+net_negative+net_neutral)
        except:
            pass
        try:
            feature_extract_26 = statistics.mean(net_text_complex_list)
        except:
            pass
        try:
            feature_extract_27 = statistics.mean(question_sentiment_list)
        except:
            pass
        try:
            feature_extract_28 = Qpositive/(Qpositive+Qnegative+Qneutral)
            feature_extract_29 = Qnegative/(Qpositive+Qnegative+Qneutral)
        except:
            pass
        try:
            feature_extract_30 = statistics.mean(questions_complex_list)
        except:
            pass
        try:
            feature_extract_31 = statistics.mean(reply_sentiment_list)
        except:
            pass
        try:
            feature_extract_32 = Rpositive/(Rpositive+Rnegative+Rneutral)
            feature_extract_33 = Rnegative/(Rpositive+Rnegative+Rneutral)
        except:
            pass
        try:
            feature_extract_34 = statistics.mean(reply_complex_list)
        except:
            pass
        try:
            feature_extract_35 = statistics.mean(s_fls_sentiment_list)
        except:
            pass
        try:
            feature_extract_36 = SFLSpositive/(SFLSpositive+SFLSnegative+SFLSneutral)
            feature_extract_37 = SFLSnegative/(SFLSpositive+SFLSnegative+SFLSneutral)
        except:
            pass

        try:
            feature_extract_38 = statistics.mean(ns_fls_sentiment_list)
        except:
            pass
        try:
            feature_extract_39 = NSFLSpositive/(NSFLSpositive+NSFLSnegative+NSFLSneutral)
```

```
            feature_extract_40 = NSFLSnegative/(NSFLSpositive+NSFLSnegative+NSFLSneutral)
        except:
            pass

        try:
            feature_extract_41 = statistics.mean(n_fls_sentiment_list)
        except:
            pass
        try:
            feature_extract_42 = NFLSpositive/(NFLSpositive+NFLSnegative+NFLSneutral)
            feature_extract_43 = NFLSnegative/(NFLSpositive+NFLSnegative+NFLSneutral)
        except:
            pass

    except:
        pass
    return [feature_extract_23, feature_extract_24, feature_extract_25,
feature_extract_26, feature_extract_27, feature_extract_28, feature_extract_29,
feature_extract_30, feature_extract_31, feature_extract_32, feature_extract_33,
feature_extract_34, feature_extract_35, feature_extract_36, feature_extract_37,
feature_extract_38, feature_extract_39, feature_extract_40, feature_extract_41,
feature_extract_42, feature_extract_43]

def get_SentimentLists_from_FLS(net_sentiment_list, THIS_flslist,
THISfls_sentiment_list, reply_sentiment_list, Rpositive, Rnegative, Rneutral,
FLSpositive, FLSnegative, FLSneutral, net_positive, net_negative, net_neutral):
    for each_fls_sentence in THIS_flslist:
        sentiment_result = sentiment_nlp(each_fls_sentence)
        sentiment_score, positivity_value = map_sentiments(sentiment_result)
        THISfls_sentiment_list.append(sentiment_score)
        reply_sentiment_list.append(sentiment_score)
        net_sentiment_list.append(sentiment_score)

        if positivity_value == "positive":
            net_positive += 1
            FLSpositive += 1
            Rpositive += 1
        elif positivity_value == "negative":
            net_negative += 1
            FLSnegative += 1
            Rnegative += 1

        else:
            net_neutral += 1
            FLSneutral += 1
            Rneutral += 1

    return THISfls_sentiment_list, reply_sentiment_list, Rpositive, Rnegative, Rneutral,
FLSpositive, FLSnegative, FLSneutral, net_positive, net_negative, net_neutral

def deepCleanTranscript(mytranscript, speaker_names):
    updatedTranscript = ' '.join(mytranscript)
    new_speaker_names = [word + ':' for word in speaker_names]
    ac = AhoCorasick(new_speaker_names)

    updatedTranscript = ac.remove_words(updatedTranscript)
```

```
    if updatedTranscript[0] == " ":
        updatedTranscript = updatedTranscript.replace(" ", "", 1)

    updatedTranscript = re.sub('[^\S\n]+', ' ', updatedTranscript)
    updatedTranscript.lower()

    return updatedTranscript

def getFeature44to73(mytranscript, speaker_names):
    marginSentimentList = []
    mar_positive = 0
    mar_negative = 0
    mar_neutral = 0

    costSentimentList = []
    cost_positive = 0
    cost_negative = 0
    cost_neutral = 0

    revenueSentimentList = []
    rev_positive = 0
    rev_negative = 0
    rev_neutral = 0

    earningsEBIDTASentimentList = []
    ear_positive = 0
    ear_negative = 0
    ear_neutral = 0

    growthSentimentList = []
    gro_positive = 0
    gro_negative = 0
    gro_neutral = 0

    leverageDebtSentimentList = []
    lev_positive = 0
    lev_negative = 0
    lev_neutral = 0

    IndSentimentList = []
    ind_positive = 0
    ind_negative = 0
    ind_neutral = 0

    operationSentimentList = []
    ope_positive = 0
    ope_negative = 0
    ope_neutral = 0

    cashflowSentimentList = []
    cash_positive = 0
    cash_negative = 0
    cash_neutral = 0

    dividendSentimentList = []
    div_positive = 0
```

```
div_negative = 0
div_neutral = 0

feature_extract_44 = 0
feature_extract_45 = 0
feature_extract_46 = 0
feature_extract_47 = 0
feature_extract_48 = 0
feature_extract_49 = 0
feature_extract_50 = 0
feature_extract_51 = 0
feature_extract_52 = 0
feature_extract_53 = 0
feature_extract_54 = 0
feature_extract_55 = 0
feature_extract_56 = 0
feature_extract_57 = 0
feature_extract_58 = 0
feature_extract_59 = 0
feature_extract_60 = 0
feautre_extract_61 = 0
feature_extract_62 = 0
feature_extract_63 = 0
feature_extract_64 = 0
feature_extract_65 = 0
feature_extract_66 = 0
feature_extract_67 = 0
feature_extract_68 = 0
feature_extract_69 = 0
feature_extract_70 = 0
feature_extract_71 = 0
feature_extract_72 = 0
feature_extract_73 = 0

updatedTranscript = deepCleanTranscript(mytranscript, speaker_names)

updatedTranscriptList = split_paragraph_into_sentences(updatedTranscript)

for mysentence in updatedTranscriptList:
    if (" margin" in mysentence) or (" return" in mysentence):
        sentiment_result = sentiment_nlp(mysentence)
        sentiment_score, positivity_value = map_sentiments(sentiment_result)
        marginSentimentList.append(sentiment_score)
        if positivity_value == "positive":
            mar_positive += 1

        elif positivity_value == "negative":
            mar_negative += 1

        else:
            mar_neutral += 1


    if " cost" in mysentence:
        sentiment_result = sentiment_nlp(mysentence)
        sentiment_score, positivity_value = map_sentiments(sentiment_result)
```

```
            costSentimentList.append(sentiment_score)
            if positivity_value == "positive":
                cost_positive += 1

            elif positivity_value == "negative":
                cost_negative += 1

            else:
                cost_neutral += 1

        if (" revenue" in mysentence) or (" top line" in mysentence) or (" sales" in
mysentence):
            sentiment_result = sentiment_nlp(mysentence)
            sentiment_score, positivity_value = map_sentiments(sentiment_result)
            revenueSentimentList.append(sentiment_score)

            if positivity_value == "positive":
                rev_positive += 1

            elif positivity_value == "negative":
                rev_negative += 1

            else:
                rev_neutral += 1

        if (" earning" in mysentence) or (" EBIT" in mysentence) or (" profit" in
mysentence) or (" bottom line" in mysentence) or (" net income" in mysentence):
            sentiment_result = sentiment_nlp(mysentence)
            sentiment_score, positivity_value = map_sentiments(sentiment_result)
            earningsEBIDTASentimentList.append(sentiment_score)
            if positivity_value == "positive":
                ear_positive += 1

            elif positivity_value == "negative":
                ear_negative += 1

            else:
                ear_neutral += 1

        if (" growth" in mysentence) or (" organic" in mysentence):
            sentiment_result = sentiment_nlp(mysentence)
            sentiment_score, positivity_value = map_sentiments(sentiment_result)
            growthSentimentList.append(sentiment_score)
            if positivity_value == "positive":
                gro_positive += 1

            elif positivity_value == "negative":
                gro_negative += 1

            else:
                gro_neutral += 1

        if (" leverage" in mysentence) or (" debt" in mysentence):
            sentiment_result = sentiment_nlp(mysentence)
            sentiment_score, positivity_value = map_sentiments(sentiment_result)
            leverageDebtSentimentList.append(sentiment_score)
```

```
            if positivity_value == "positive":
                lev_positive += 1

            elif positivity_value == "negative":
                lev_negative += 1

            else:
                lev_neutral += 1

        if (" industry" in mysentence) or (" industr" in mysentence) or (" economy" in
mysentence) or (" economi" in mysentence) or (" sector" in mysentence):
            sentiment_result = sentiment_nlp(mysentence)
            sentiment_score, positivity_value = map_sentiments(sentiment_result)
            IndSentimentList.append(sentiment_score)
            if positivity_value == "positive":
                ind_positive += 1

            elif positivity_value == "negative":
                ind_negative += 1

            else:
                ind_neutral += 1

        if " operation" in mysentence:
            sentiment_result = sentiment_nlp(mysentence)
            sentiment_score, positivity_value = map_sentiments(sentiment_result)
            operationSentimentList.append(sentiment_score)
            if positivity_value == "positive":
                ope_positive += 1

            elif positivity_value == "negative":
                ope_negative += 1

            else:
                ope_neutral += 1

        if (" cashflow" in mysentence) or (" cash flow" in mysentence):
            sentiment_result = sentiment_nlp(mysentence)
            sentiment_score, positivity_value = map_sentiments(sentiment_result)
            cashflowSentimentList.append(sentiment_score)
            if positivity_value == "positive":
                cash_positive += 1

            elif positivity_value == "negative":
                cash_negative += 1

            else:
                cash_neutral += 1

        if (" dividend" in mysentence) or (" buyback" in mysentence) or (" repurchase" in
mysentence):
            sentiment_result = sentiment_nlp(mysentence)
            sentiment_score, positivity_value = map_sentiments(sentiment_result)
            dividendSentimentList.append(sentiment_score)
            if positivity_value == "positive":
                div_positive += 1
```

```
        elif positivity_value == "negative":
            div_negative += 1

        else:
            div_neutral += 1

try:
    feature_extract_44 = statistics.mean(marginSentimentList)
except:
    pass
try:
    feature_extract_45 = mar_positive/(mar_positive+mar_negative+mar_neutral)
    feature_extract_46 = mar_negative/(mar_positive+mar_negative+mar_neutral)
except:
    pass
try:
    feature_extract_47 = statistics.mean(costSentimentList)
except:
    pass
try:
    feature_extract_48 = cost_positive/(cost_positive+cost_negative+cost_neutral)
    feature_extract_49 = cost_negative/(cost_positive+cost_negative+cost_neutral)
except:
    pass
try:
    feature_extract_50 = statistics.mean(revenueSentimentList)
except:
    pass
try:
    feature_extract_51 = rev_positive/(rev_positive+rev_negative+rev_neutral)
    feature_extract_52 = rev_negative/(rev_positive+rev_negative+rev_neutral)
except:
    pass
try:
    feature_extract_53 = statistics.mean(earningsEBIDTASentimentList)
except:
    pass
try:
    feature_extract_54 = ear_positive/(ear_positive+ear_negative+ear_neutral)
    feature_extract_55 = ear_negative/(ear_positive+ear_negative+ear_neutral)
except:
    pass

try:
    feature_extract_56 = statistics.mean(growthSentimentList)
except:
    pass
try:
    feature_extract_57 = gro_positive/(gro_positive+gro_negative+gro_neutral)
    feature_extract_58 = gro_negative/(gro_positive+gro_negative+gro_neutral)
except:
    pass

try:
    feature_extract_59 = statistics.mean(leverageDebtSentimentList)
```

```
    except:
        pass
    try:
        feature_extract_60 = lev_positive/(lev_positive+lev_negative+lev_neutral)
        feature_extract_61 = lev_negative/(lev_positive+lev_negative+lev_neutral)
    except:
        pass

    try:
        feature_extract_62 = statistics.mean(IndSentimentList)
    except:
        pass
    try:
        feature_extract_63 = ind_positive/(ind_positive+ind_negative+ind_neutral)
        feature_extract_64 = ind_negative/(ind_positive+ind_negative+ind_neutral)
    except:
        pass

    try:
        feature_extract_65 = statistics.mean(operationSentimentList)
    except:
        pass
    try:
        feature_extract_66 = ope_positive/(ope_positive+ope_negative+ope_neutral)
        feature_extract_67 = ope_negative/(ope_positive+ope_negative+ope_neutral)

    except:
        pass

    try:
        feature_extract_68 = statistics.mean(cashflowSentimentList)
    except:
        pass
    try:
        feature_extract_69 = cash_positive/(cash_positive+cash_negative+cash_neutral)
        feature_extract_70 = cash_negative/(cash_positive+cash_negative+cash_neutral)

    except:
        pass

    try:
        feature_extract_71 = statistics.mean(dividendSentimentList)
    except:
        pass
    try:
        feature_extract_72 = div_positive/(div_positive+div_negative+div_neutral)
        feature_extract_73 = div_negative/(div_positive+div_negative+div_neutral)
    except:
        pass

    return [feature_extract_44, feature_extract_45, feature_extract_46,
feature_extract_47, feature_extract_48, feature_extract_49, feature_extract_50,
feature_extract_51, feature_extract_52, feature_extract_53, feature_extract_54,
feature_extract_55, feature_extract_56, feature_extract_57, feature_extract_58,
feature_extract_59, feature_extract_60, feautre_extract_61, feature_extract_62,
feature_extract_63, feature_extract_64, feature_extract_65, feature_extract_66,
```

```
feature_extract_67, feature_extract_68, feature_extract_69, feature_extract_70,
feature_extract_71, feature_extract_72, feature_extract_73]

def map_sentiments(sentiment_result):
   sentiment_result = sentiment_result[0]
   if sentiment_result['label'] == 'Negative':
       return -1 * sentiment_result['score'], "negative"

   elif sentiment_result['label'] == 'Neutral':
       return 0, "neutral"

   elif sentiment_result['label'] == 'Positive':
       return sentiment_result['score'], "positive"


def split_paragraph_into_sentences(temp):
   sentences = nltk.sent_tokenize(temp)
   return sentences

def get_NLP_values(liststr):
   # further analysis includes finding sentiment and word complexity.
   if len(liststr) == 0:
       return 0, 0, 0

   else:
       # maps sentiment data so it outputs a single sentiment value
       sentiment_result = sentiment_nlp(liststr)
       # gets
       sentiment_score = map_sentiments(sentiment_result)

       # word complexity:
       flesch_score = textstat.flesch_reading_ease(liststr)
       gunning_fog_score = textstat.gunning_fog(liststr)

       return sentiment_score, flesch_score, gunning_fog_score
```

Figure 93 - Functions that finds the input features

Figure 93 is a slightly modified list of functions that help find the input list of features to be processed through the machine learning model.

Testing:

Technical Testing:

For the testing part of my programme, I've decided to partake in three factors:

1. **Technical Testing**
    a. **Unit testing**. Tests are defined for every major function in code and run regularly. This helps ascertain that the code works, and also helps to ensure that bugs are not introduced.
    b. **Testing for Rubustness** with exception Handlings
2. **Requirements Testing** to check with my initial requirements in the Analysis section
3. **End User/Human Testing**, where I ask my investing friends to use the service and asks for their comments. Also tests to make sure the user-firebase functions runs as it should.
4. **Performance Testing** to test whether the program runs at an acceptable speed.

Unit Testing

Basic User Interface:

First Example:

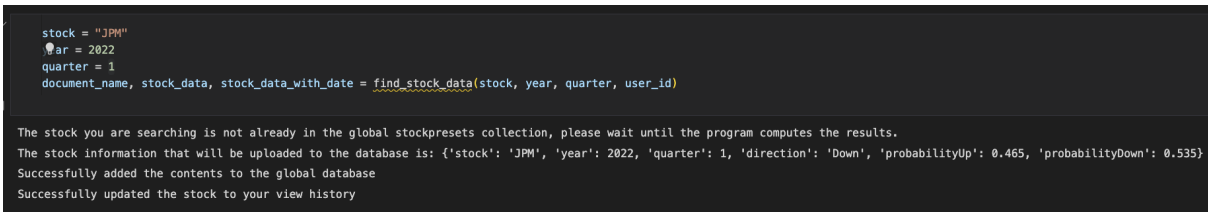


Figure 94 - Inputting JPM 2022 Q1 as query

Input:
Stock = JPM
Year = 2022
Quarter = 1

Expected Output meets actual output.

It says `The stock you are searching is not already in the global stockpresets collection, please wait until the program computes the results.` This means this stock (JPM2022Q1) has not been queried before by any

other users so the output prediction has not been registered in the **"global stockpresets database"**. The reason there is a **"global stockpresets database"** to begin with is that the programme takes time to pre-process and parse the earnings transcript, so instead, the programme can query this **"global stockpresets database"** and find the stock predictions almost immediately to save time. And since it's not in the global stockpresets database, the progrmame will parse all the relevant info in the earnings call and run it through the machine learning algorithm to get the output as shown below.

The second part, is the stock prediction object:

```
The stock information that will be uploaded to the database is:
{'stock': 'JPM', 'year': 2022, 'quarter': 1, 'direction': 'Down',
'probabilityUp': 0.465, 'probabilityDown': 0.535}
```

This also meets the expected output as each value has the correct datatypes

```
Successfully added the contents to the global database
```

This message means it has re-uploaded the contents to the global database (changes the datetime of which it was queried).

```
Successfully updated the stock to your view history
```

This message implies it has successfully updated the stock to their "view history" database collection, as expected.

Second Example for the User interaction process:

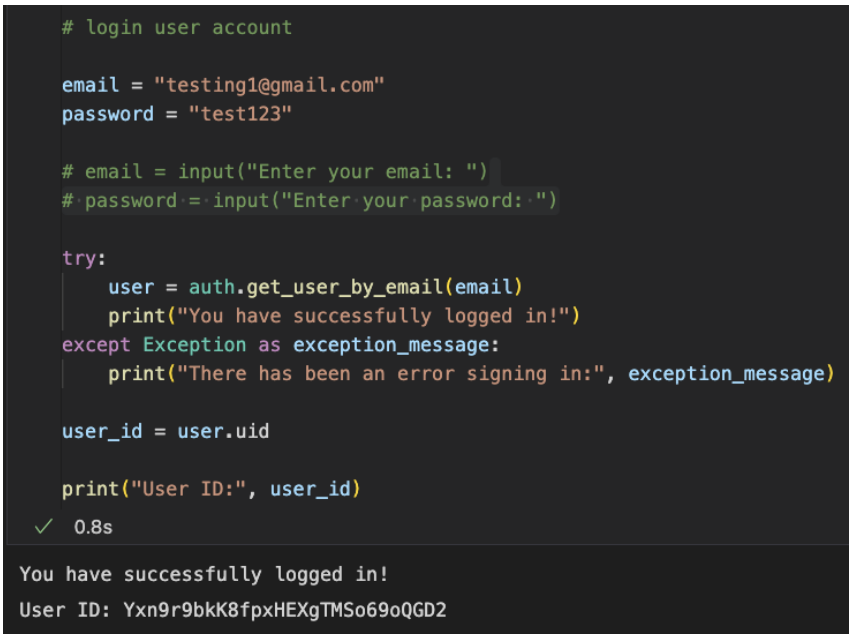


Figure 95 - Logging in to user with the email testing1@gmail.com

Figure 95 shows the code to log in to the user's account via firebase by inputting the email and password, the try and except exception handling also makes sure if they logged in successfully or not, and if there's an error, it will provide the exception message. Furthermore, it gets the userID after logging in (i.e. Yxn9r9bkK…) This userID makes sure that the user can only access their account's information.

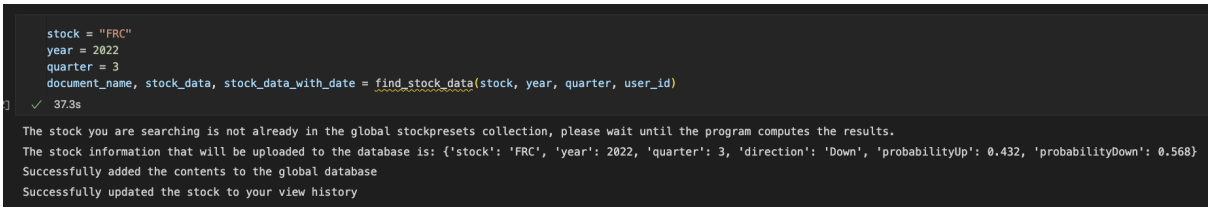
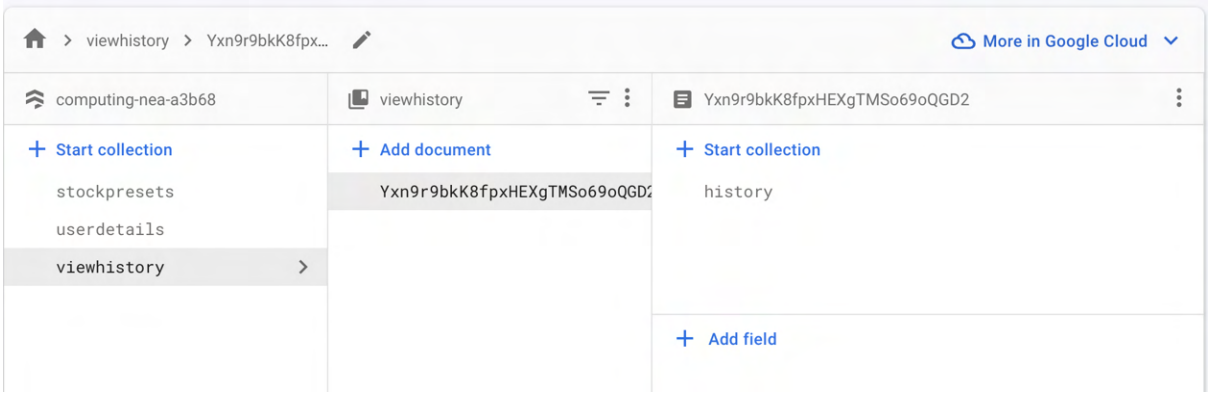


Figure 96 - Running an example

With reference to Figure 96, after loging in, this user can run by inputting their stock, year and quarter dates, in this example being FRC, 2022, Quarter 3. Since this stock hasn't been run before, it is not located in the global stockpresets database, therefore, it sends the message **"The stock you are searching is not already in the global stockpresets collection, please wait until the program computes the results."** and saves it in the global database, as well as their view history, as expected.

Furthermore, we should look at the contents of firebase to make sure that it's saved in the right format as well.



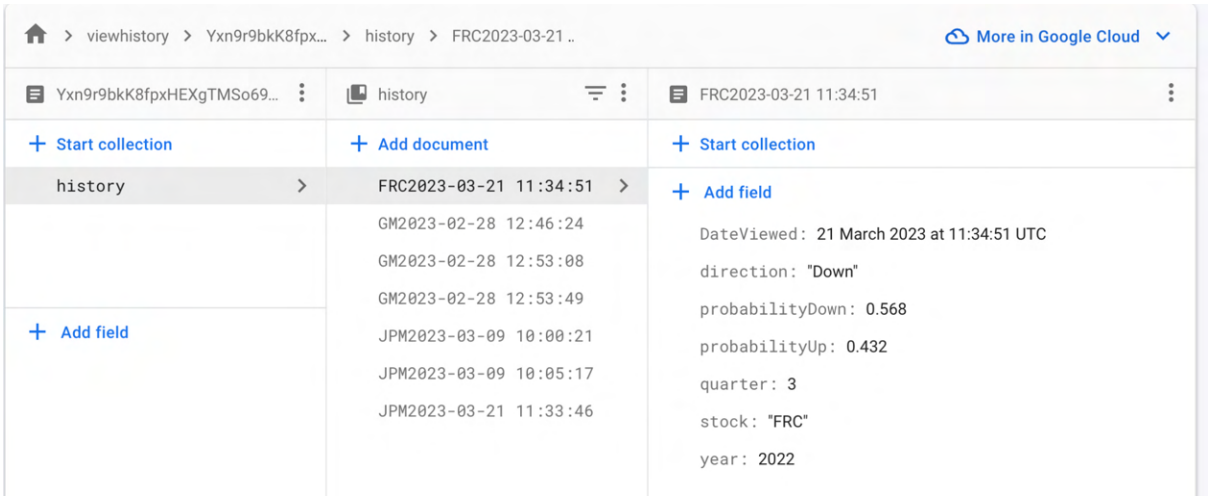


Figure 97a and Figure 97b- ViewHistory collection for the User with UserID

As seen in Figure 97a and Figure 97b, we can see that the FRC 2022 Q3 earnings call was analysed and saved in their “view history” with the file name “FRC2023-03-21 11:34:51” which is the date I queried FRC.

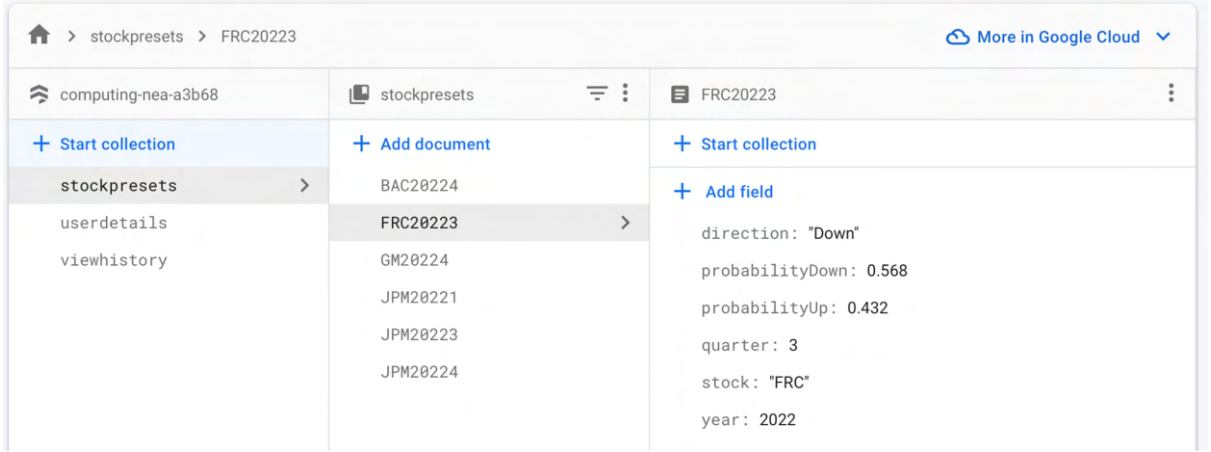


Figure 98 - StockPresets global database

As seen in Figure 98, the contents for FRC, 2022, Q3 is also uploaded to the global stock presets database as well, saved with the filename FRC20223. You can also see other stockpresets have also been saved, for example, JPM20221, the one we ran in the previous example.

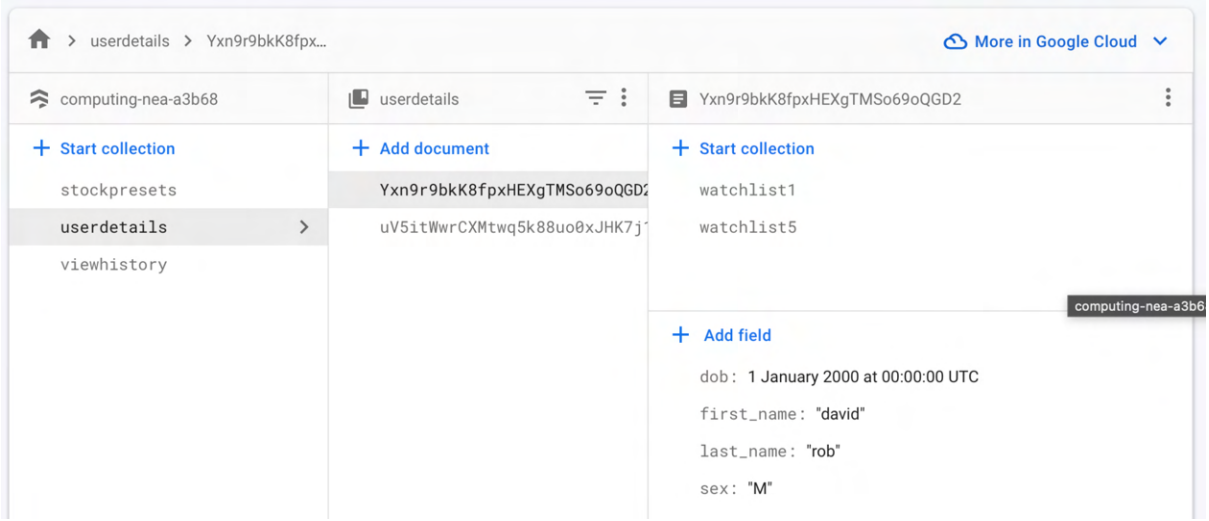


Figure 99 - Userdetails (before update)

Now looking at the userdetails collection, you can see this user currently has 2 “watchlists” named watchlist1 and watchlist5. The user can choose to upload the contents of FRC20223 that they’ve just run to an existing watchlist or create a new watchlist.

```
print("Your current watchlists are:")
# from userdetails collection
user_ref = db.collection('userdetails').document(user_id)
collections = user_ref.collections()
for collection in collections:
    print(collection.id)

watchlist_name = input("Please enter a watchlist to add this stock to, or enter 'N' if you dor
if watchlist_name == "N":
    exit()
else:
    try:
        watchlist_ref = user_ref.collection(watchlist_name)
        stock_ref = watchlist_ref.document(document_name)
        stock_ref.set(stock_data_with_date)
        print(f'Successfully added {document_name} to the watchlist {watchlist_name}!')
    except:
        print("There has been an error with adding to your watchlist:", exception_message)
```

```
[13]  ✓ 10.8s

...  Your current watchlists are:
watchlist1
watchlist5
Successfully added FRC20223 to the watchlist MyFavouriteWatchlist!
```

Figure 100 - Uploading stockdata contents to a watchlist

Figure 100 shows the code to upload stock data contents to the watchlist “MyFavouriteWatchlist”. However, since this watchlist does not exist in the userdetails’ watchlist collection, then it is created.

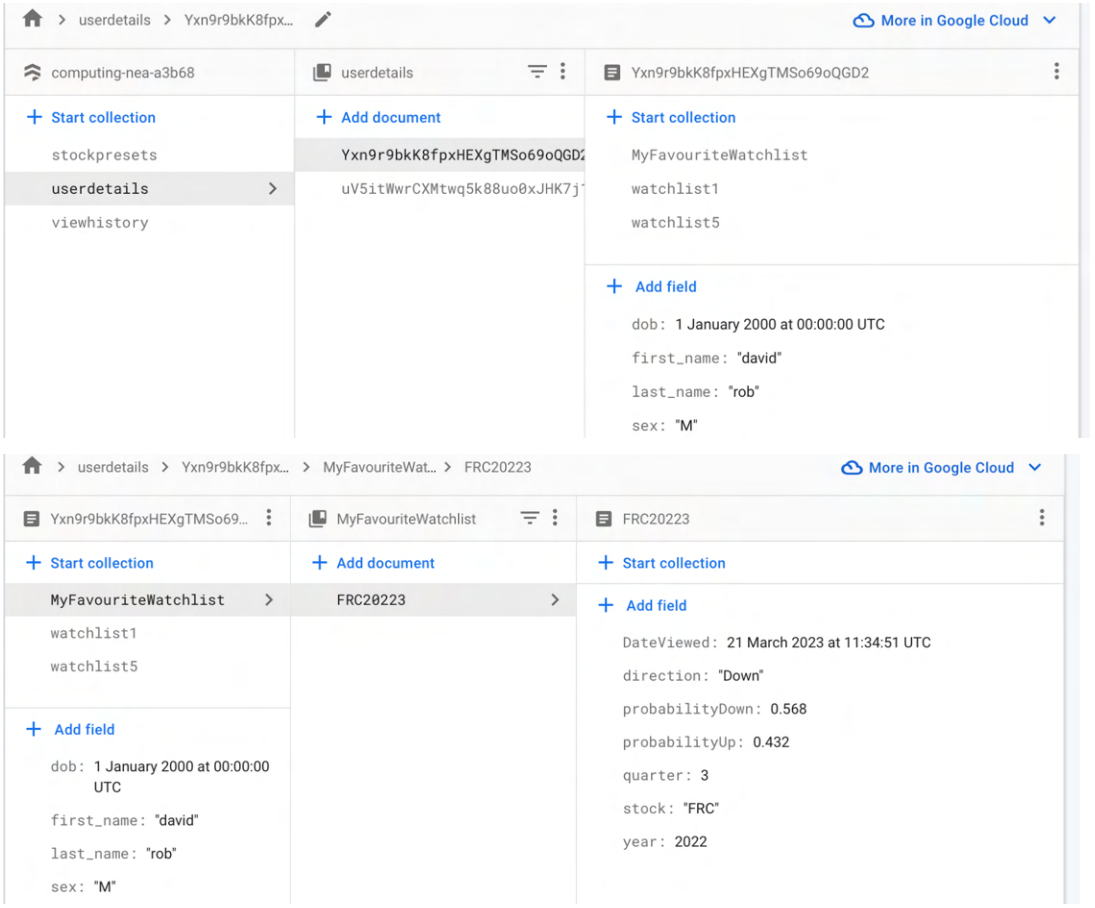


Figure 101a and Figure 101b - Userdetails (after update)

As seen in Figure 101a and 101b, the contents of FRC20223 are added to MyFavouriteWatchlist on Firebase. The userdetails also includes the date of birth of this user, his name, and sex. All of which are expected so we can conclude the testing here works perfectly.

Third Example:

Figure 102 - Querying stocks that are already in the global stockrpesets database

Looking at Figure 102, if we query JPM 2022 Q1 again, it says **"The stock you are searching is already in the global stockpresets collection, the program does not need to compute it again "** which means it can return the data instantaneously (0.3seconds) and update the view history accordingly too.

Feature Extraction:

One problem that I encountered when running through the programme is that some earnings transcript files were stored in the wrong format.

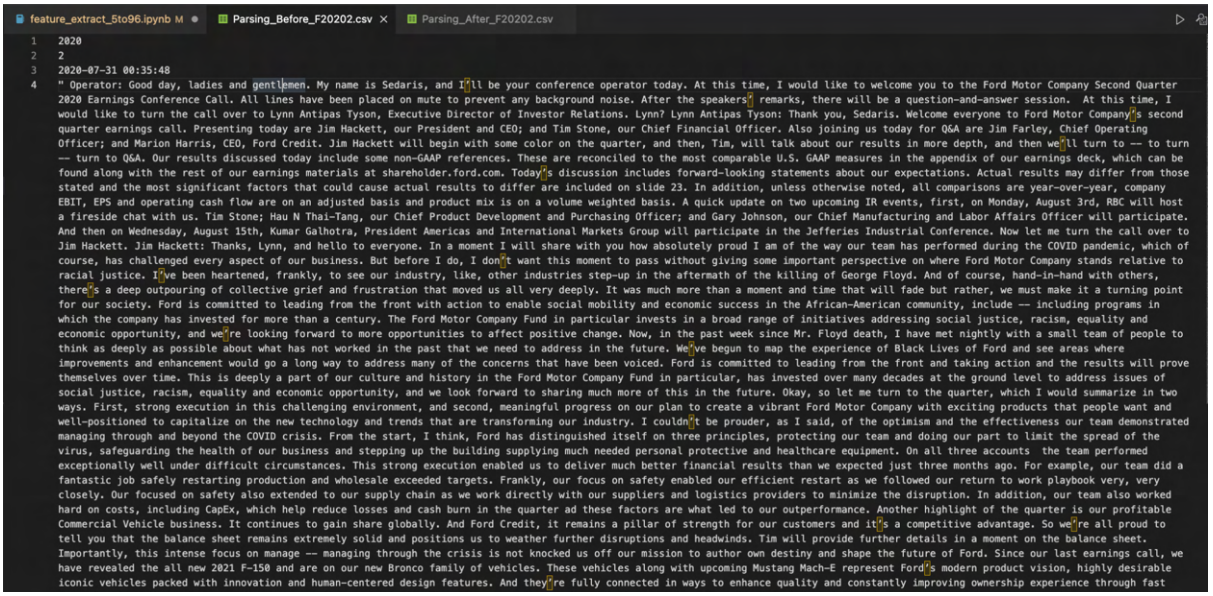


Figure 103 - Wrong format earnings transcript file

Figure 103 is one example of an earnings transcript in the wrong format.

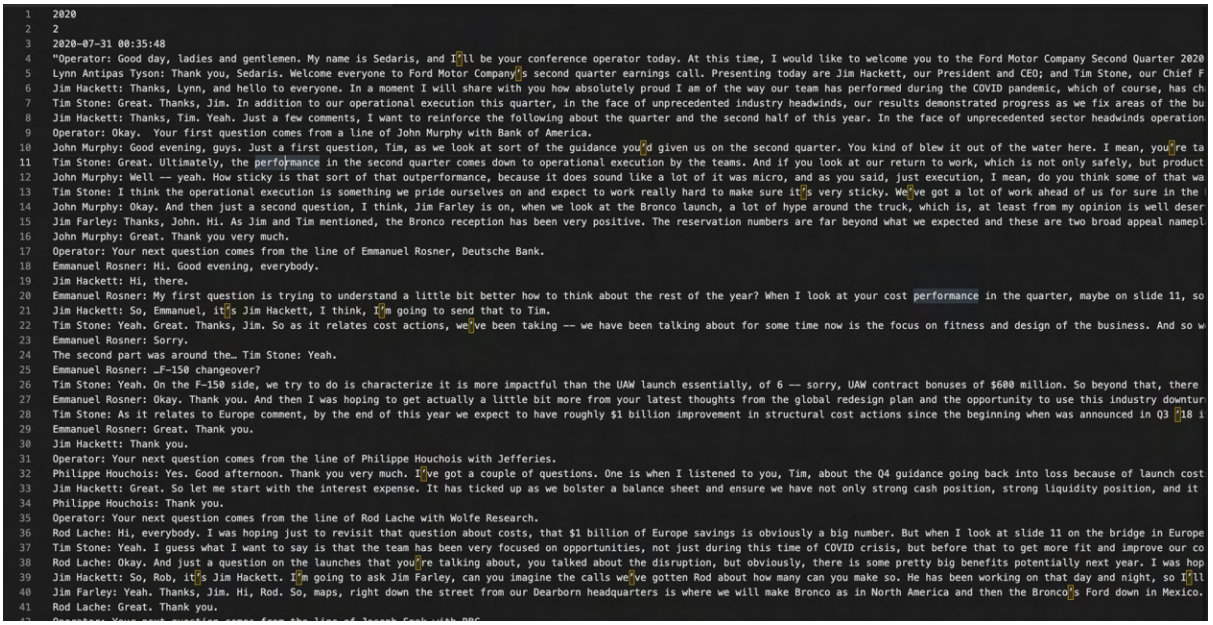


Figure 104 - Desired format for the same earnings transcript

Figure 104 is the desired format that the earnings transcript should be stored, where each new line starts with the speaker_name, followed by a colon.

```
def split_by_colon(string):
```

```
    result = []
    while ":" in string:
        name_start3 = string.rfind(".", 0, string.index(":"))
        name_start2 = string.rfind("?", 0, string.index(":"))
        name_start = max(name_start2, name_start3)
        if name_start == -1:
            name_start = 0
        else:
            name_start += 1
        name = string[name_start:string.index(":")].strip()
        result.append(name + ": " + string[string.index(":") + 1:].strip())
        string = string[string.index(":") + 1:]
    return (result)
```

Figure 105 - Split by colon function

Hence, I wrote a function called **split_by_colon** that separates “speech bubbles” by recognising when the speaker begins their speech.

```
sector = "automobiles"
stock = "F"

path = '../Feature_Extraction/Parsing_Before_F20202.csv'
input_string = pd.read_csv(path).iloc[[2]].values[0][0]

output_list = split_by_colon(input_string)

result = []
for i in range(0, len(output_list)-1):
    currentstring = output_list[i]
    nextstring = output_list[i+1]
    '\n'.join(currentstring)
    index = currentstring.find(nextstring)
    if index != -1:
        currentstring = currentstring[:index]
    result.append(currentstring)
result.append(nextstring)

mystr = "\n".join(result)

with open(path, "r") as f:
    reader = csv.reader(f)
    data = [row for row in reader]

data[3] = [mystr]

with open(path, "w", newline='') as f:
    writer = csv.writer(f)
    writer.writerows(data)
```

Figure 106 - Code that runs the split by colon function and correctly formats the earnings transcript

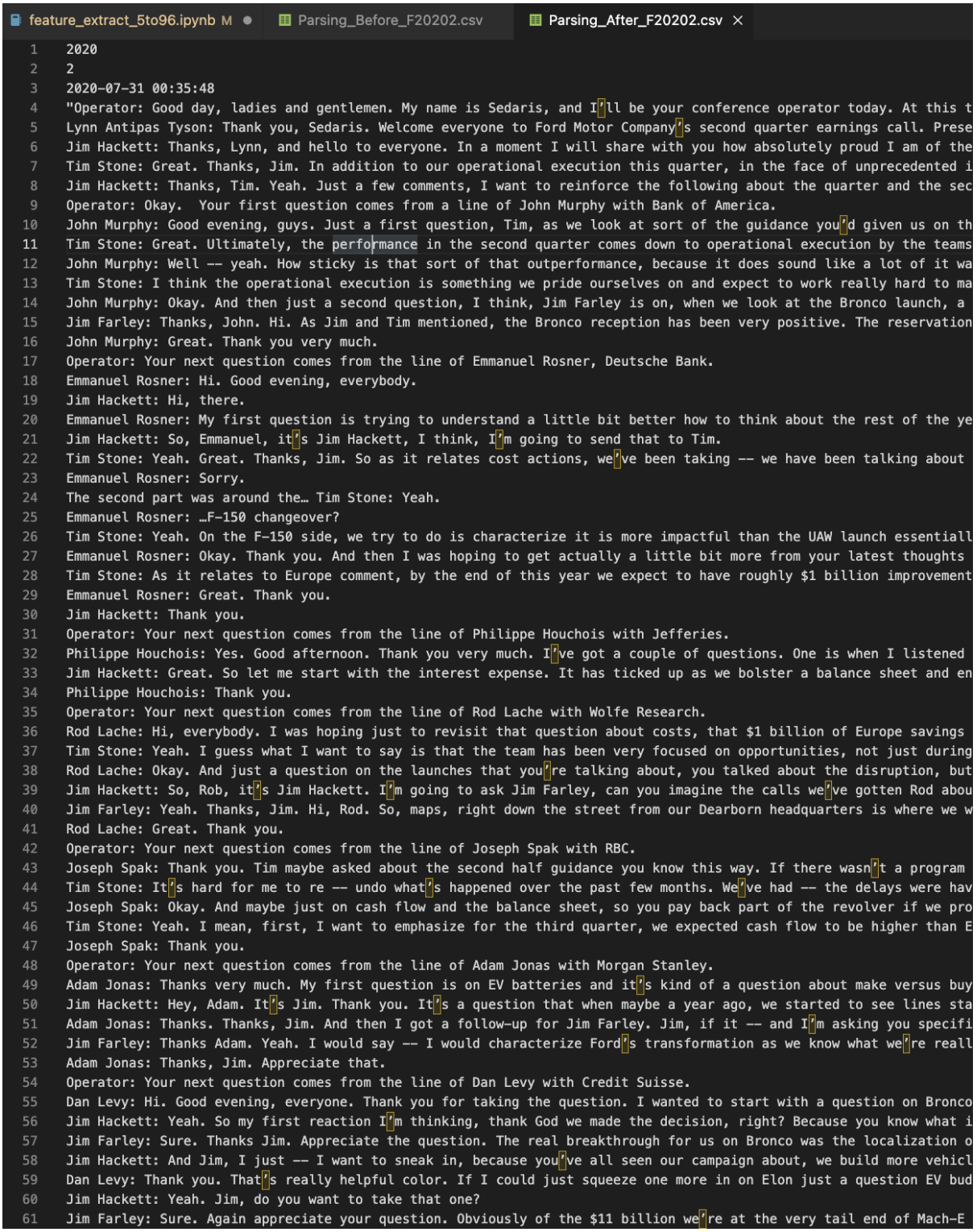


Figure 107 - Transcript result after being formatted correctly

Robustness:

Pre-defining Variables:

```
def getFeature44to73(mytranscript, speaker_names):
    marginSentimentList = []
    mar_positive = 0
    mar_negative = 0
    mar_neutral = 0

    costSentimentList = []
    cost_positive = 0
    cost_negative = 0
    cost_neutral = 0

    revenueSentimentList = []
    rev_positive = 0
    rev_negative = 0
    rev_neutral = 0

    earningsEBIDTASentimentList = []
    ear_positive = 0
    ear_negative = 0
    ear_neutral = 0

    growthSentimentList = []
    gro_positive = 0
    gro_negative = 0
```

Figure 108 - Pre-defining variables

With reference to Figure 108, I have been pre-defining these variables for every feature extraction technique to improve the robustness of the code. Pre-defining variables can improve the robustness of code by ensuring that they have a valid initial value, which can help prevent errors and unexpected behaviour. In many of these cases, this may involve setting default values, such as 0, for variables that could otherwise return None or other invalid values.

Try Except Blocks:

```
feature_extract_44 = 0
feature_extract_45 = 0
feature_extract_46 = 0
feature_extract_47 = 0
feature_extract_48 = 0
feature_extract_49 = 0
feature_extract_50 = 0
feature_extract_51 = 0
```

Figure 109a - Pre-defining the feature sets variables

```
try:
    feature_extract_44 = statistics.mean(marginSentimentList)
except:
    pass
try:
    feature_extract_45 = mar_positive/(mar_positive+mar_negative+mar_neutral)
    feature_extract_46 = mar_negative/(mar_positive+mar_negative+mar_neutral)
except:
    pass
try:
    feature_extract_47 = statistics.mean(costSentimentList)
except:
    pass
try:
    feature_extract_48 = cost_positive/(cost_positive+cost_negative+cost_neutral)
    feature_extract_49 = cost_negative/(cost_positive+cost_negative+cost_neutral)
except:
    pass
try:
    feature_extract_50 = statistics.mean(revenueSentimentList)
except:
    pass
try:
    feature_extract_51 = rev_positive/(rev_positive+rev_negative+rev_neutral)
    feature_extract_52 = rev_negative/(rev_positive+rev_negative+rev_neutral)
except:
    pass
try:
    feature_extract_53 = statistics.mean(earningsEBIDTASentimentList)
except:
    pass
try:
    feature_extract_54 = ear_positive/(ear_positive+ear_negative+ear_neutral)
    feature_extract_55 = ear_negative/(ear_positive+ear_negative+ear_neutral)
except:
    pass
```

Figure 109b - Try, Except block

To prevent errors from occurring and to avoid skipping values that cannot be processed by the machine learning algorithm, every feature (e.g. in Figure 109a) is pre-defined as 0. If an error does occur, the program will return the default value of 0 instead of None or NaN.

Figure 109b demonstrates the use of try and except blocks to ensure that errors do not end the code. Since each earnings transcript takes 1-2 minutes to complete, errors can erase progress, making it unproductive to constantly pause the code and fix the error. In such cases, the program will return a pre-defined value of 0 instead of abrupting the program.

For example, if the denominator is 0, the programme will return an error, but instead of ending the programme, it will pass on to the next line of code and use the predefined value of that particular feature, in this case, 0. There are many reasons why the denominator can be 0, for example with reference to feature_extract_54 in Figure 109b, if there are no sentences that contain the word “earnings”, then the denominator will be 0 as the number of sentences is 0.

```
def find_percentage_change(hist):
    try:
        percentage_change = ((hist['Open'][-1])/(hist['Open'][0])-1) # % change in price of day X to day X+Y
    except:
        return None

    return percentage_change
```

Figure 110 - Robustness in the stock return percentage change.

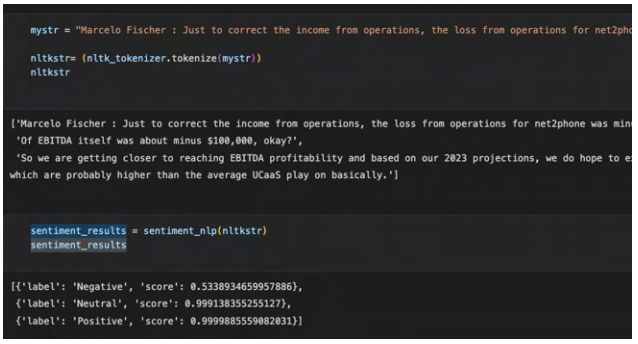
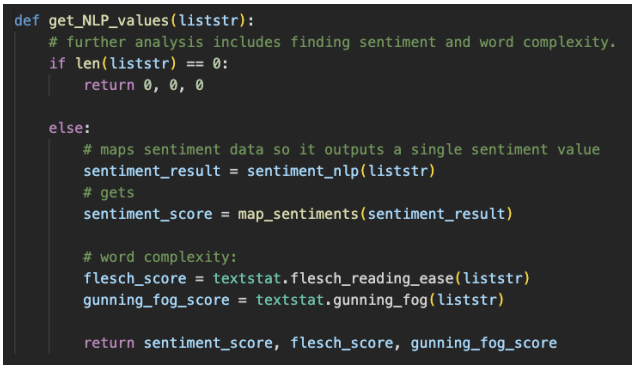
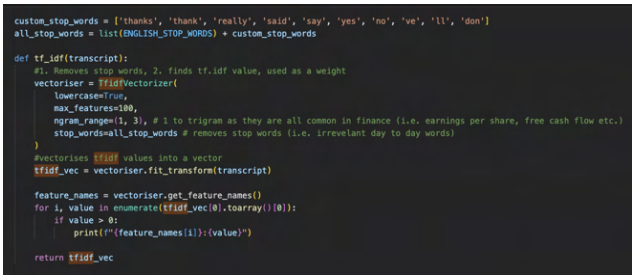
There are instances where returning None is necessary instead of 0, such as when calculating the percentage of stock returns when the historical stock price API becomes unavailable (i.e. Figure 110). In such cases, relying on a single value is not optimal as accuracy is essential for the dependent variable of the model, which cannot be smoothed out by other features. This is in contrast to the independent variables as it consists of over 80 features, so a single default value would not significantly alter the accuracy.

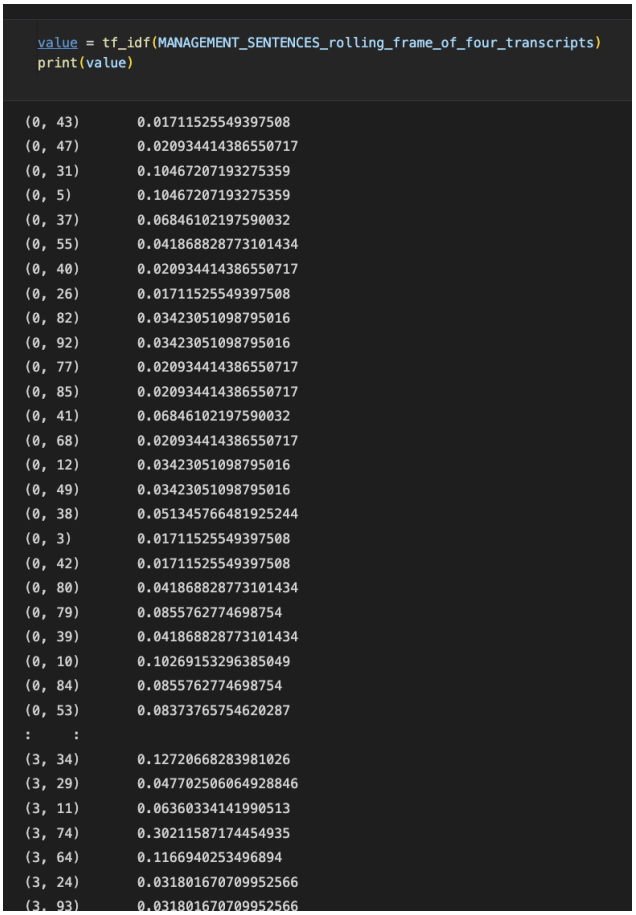
Requirements Testing:

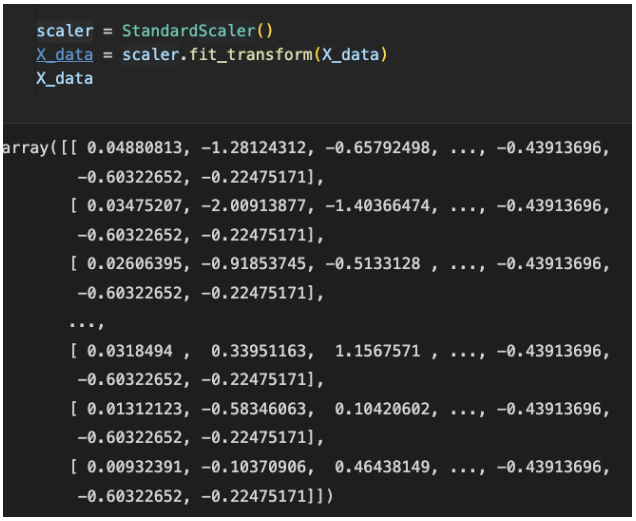
In the requirements testing section, I will reference the specifications I have listed previously in the NEA analysis part. One thing to note is that my project is divided between the “**Machine Learning**” section, and the “**User Interface**” section. The machine learning section is in charge of providing the features for the machine learning algorithm, whereas the “user interface” acts as a bridge between the user, database, and the finalised machine learning model.

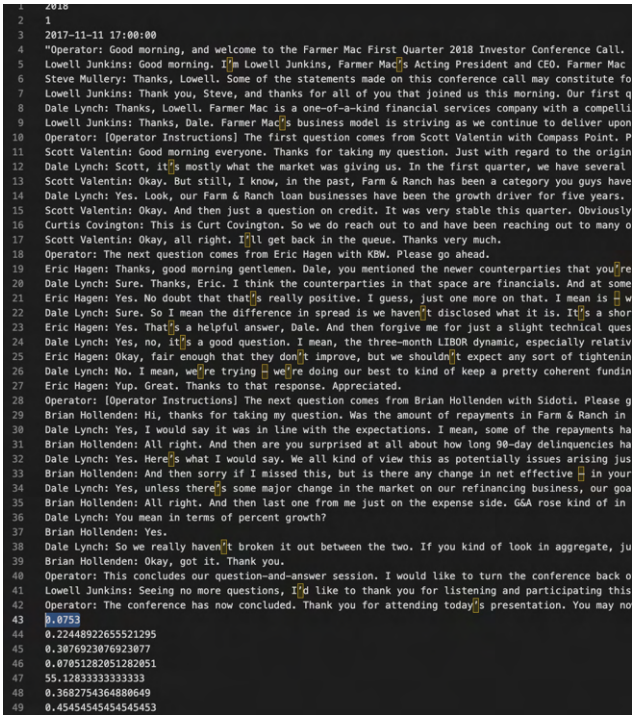
Machine Learning:

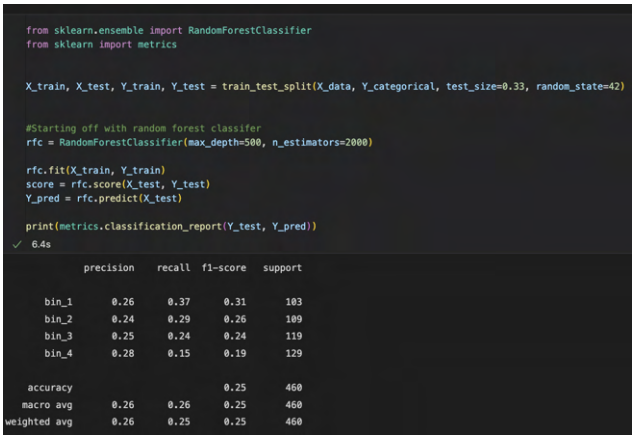
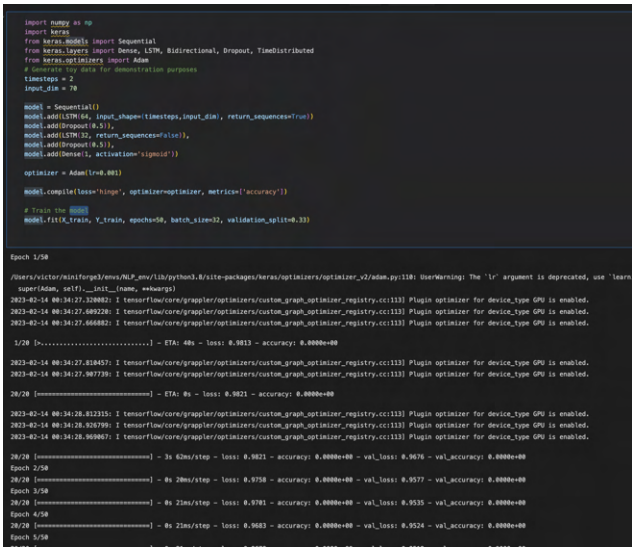
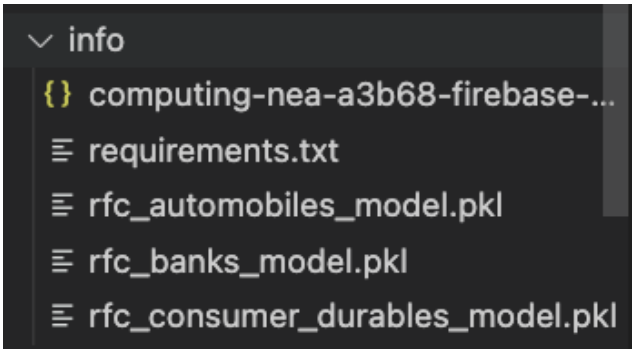
Test Number	Description	Met (Yes, Partial, No)	Evidence/comment

1	Sentiment analysis on earnings calls using FinBert Models	Yes	
2	Text complexity analysis on earnings calls	Yes	
3	Text similarity analysis on earnings calls	Yes	 

			 Vectorise it using the TF-IDF formula, then finds the cosine similarity between these TF-IDF vectors to find the text similarity value.
4	Moving Average Convergence Divergence (MACD)	No	Unable to find historical momentum data on stocks, only real-time data so I can't use it to train.
5	Volume buy/sell ratio, and its deviation to average	No	Same reason why I can't find MACD as there aren't any historical data to train from so I decided not to include it.

6	Normalise all above values using MinMaxScaler on Scikit Learn	Yes	
7	If time allows: Word2Vec word embedding on annual reports analysis.	No	Unfortunately, this specification was overly ambitious and I didn't have time to include it.
8	If time allows: Spotting variances in new annual/quarterly reports (i.e. new footnotes, and measure their sentiment using L&M dictionary)	No	Unfortunately, this specification was overly ambitious and I didn't have time to include it.
9	If time allows: Intrinsic value estimator	Partial	There isn't any historical information on the intrinsic value, however, there are for earnings per share (EPS) data, so I used EPS data instead. See **feature_extract_EPS.ipynb** file for more evidence.

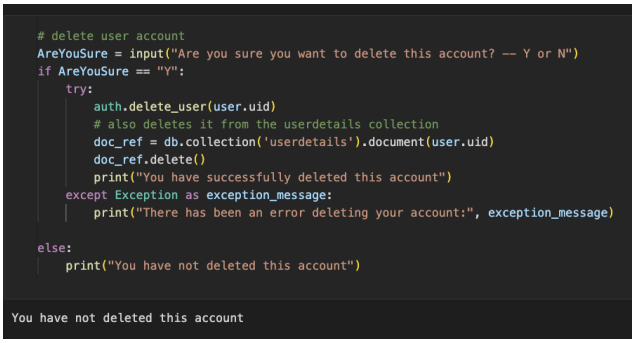
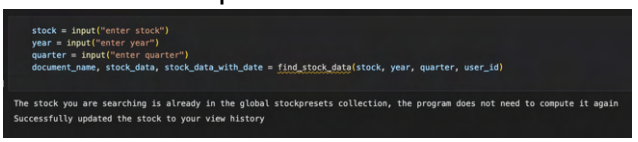
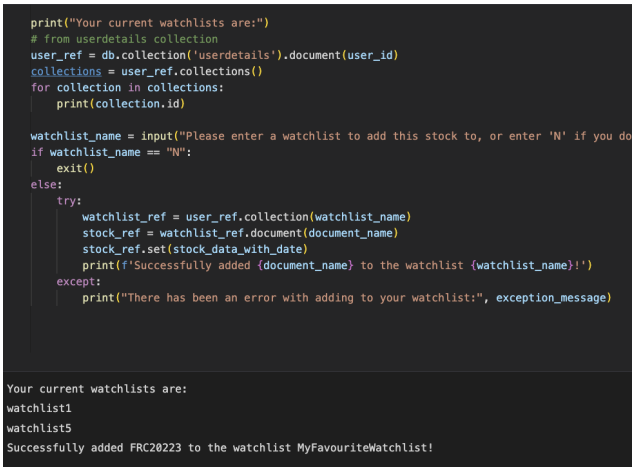
			 Highlighted is the EPS value.
10	It will train using the data from all S&P 500 companies' historical data, and Russell 2000 companies, if time allows.	Partial	I included features for 2469 stocks, and 85620 earnings transcripts. The 85620 transcripts and EPS data are there, however, the rest of the features were not compiled as it takes roughly 30 seconds to 1:30 minute for each transcript to iterate through. That means it will take around 1427 hours (60 days) to cycle through the 85620 transcript.
11	Use a classification method (Random Forest Classifier) for the stock prediction. Including both a Binary and a Multiclass approach	Yes	 Binary Classification

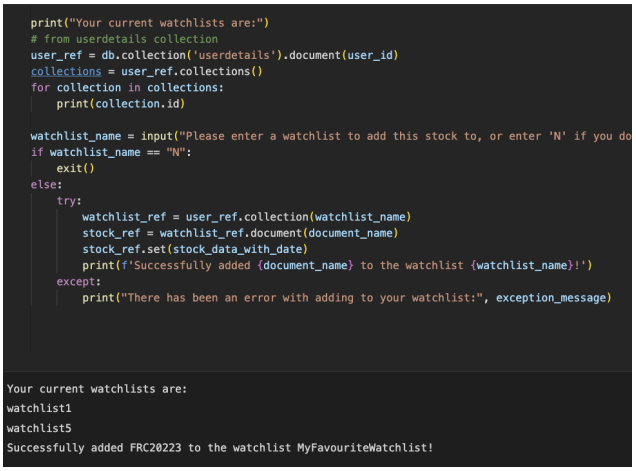
			 Multi-class classification, the reason why I didn't choose this is because it has low accuracy.
12	Using deep learning models like using Long Short-Term Memory (LSTM) networks	Partial	 Attempting to create a LSTM model, but the data structure is not suitable for timeseries models. So by the time I realised this, I already finished extracting the features so the LSTM did not achieve high accuracies, thus I opted for the binary classification.
13	Machine learning model serialised using the Python Pickle library to pickle (.pkl) files.	Yes	

			`` with open("info/rfc_banks_model.pkl", "rb") as f: banks_rfc = pickle.load(f) with open("info/rfc_automobiles_model.pkl", "rb") as f: automobiles_rfc = pickle.load(f) with open("info/rfc_consumer_durables_model.pkl", "rb") as f: consumer_durables_rfc = pickle.load(f) ```

User Interface

Test Number	Description	Met (Yes, Partial, No)	Evidence/comment
14	Users have the option to register, login, delete their account, and edit their user information.	Yes	``` # create a user account email = "testing1@gmail.com" password = "test123" email = input("Enter your email: ") password = input("Enter your password: ") try: user = auth.create_user(email=email, password=password) except Exception as exception_message: print("There has been an error creating an account:", exception_message) user_id = user.uid print("User ID:", user_id) submit_user_details(user.uid) ``` ``` # login user account email = "testing1@gmail.com" password = "test123" # email = input("Enter your email: ") # password = input("Enter your password: ") try: user = auth.get_user_by_email(email) print("You have successfully logged in!") except Exception as exception_message: print("There has been an error signing in:", exception_message) user_id = user.uid print("User ID:", user_id) ``` ``` You have successfully logged in! User ID: Yxn9r9bkK8fpxHEXgTMSo69oQGD2 ``` ``` # Update user email new_email = 'replaceEmail@gmail.com' try: user = auth.update_user(user.uid, email=new_email,) print("You have successfully updated your email address") except Exception as exception_message: print("There has been an error updating your account:", exception_message) ``` ``` You have successfully updated your email address ```

			
15	Users can search for specific stock tickers through a search bar	Partial	I do not have a search bar as I didn't have time to create a website, however, the user can enter the "stock", "year", and "quarter", then choose to predict. 
16	If the user wants to "predict" a stock, the programme will query, and compute the specific stock's features into a "feature vector", this is then fed into the deserialised machine learning algorithm for predictions.	Yes	See the functions in the bottom of the file **"Basic_User_Interface.ipynb"**
17	Users have the option to save the results of the stock information tab into a "favourites" list as a dataframe. Each dataframe will include information such as: - Stock ticker name - Date model is generated - Date model is saved - Prediction of the direction of stock's movement - Probability of stock's movement prediction		

			stock_data {'stock': 'JPM', 'year': 2022, 'quarter': 1, 'direction': 'Down', 'probabilityUp': 0.465, 'probabilityDown': 0.535}
18	Users can delete particular stock dataframes from their “favourites list”	No	Due to time constraints
19	Users can create new “favourites” lists and name it as a string.	Yes	Yes  print("Your current watchlists are:") # from userdetails collection user_ref = db.collection('userdetails').document(user_id) collections = user_ref.collections() for collection in collections: print(collection.id) watchlist_name = input("Please enter a watchlist to add this stock to, or enter 'N' if you do if watchlist_name == "N": exit() else: try: watchlist_ref = user_ref.collection(watchlist_name) stock_ref = watchlist_ref.document(document_name) stock_ref.set(stock_data_with_date) print(f'Successfully added {document_name} to the watchlist {watchlist_name}!') except: print("There has been an error with adding to your watchlist:", exception_message) Your current watchlists are: watchlist1 watchlist5 Successfully added FRC20223 to the watchlist MyFavouriteWatchlist!
20	The main page of the website will show the various “favourites” or other lists the user has when he logs in, as well as a stock search bar in the middle.	Partial	The user can see what favourites they have, however I did not make a website due to time constraints.

Performance Testing:

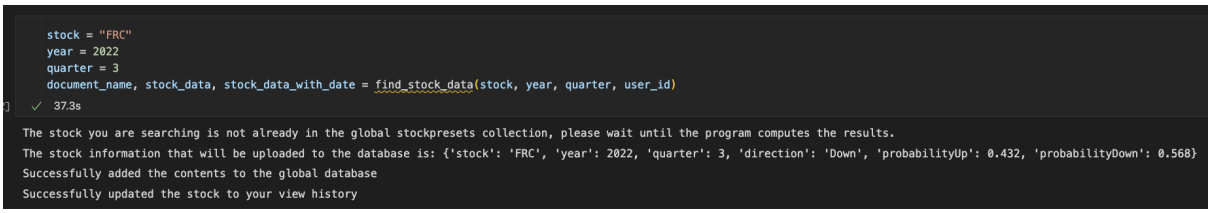


Figure 111a - Testing running time for FRC 2022 Q3

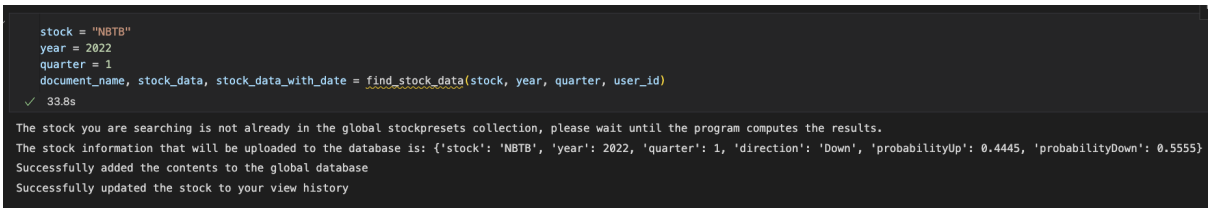


Figure 111b - Testing running time for NBTB 2022 Q1

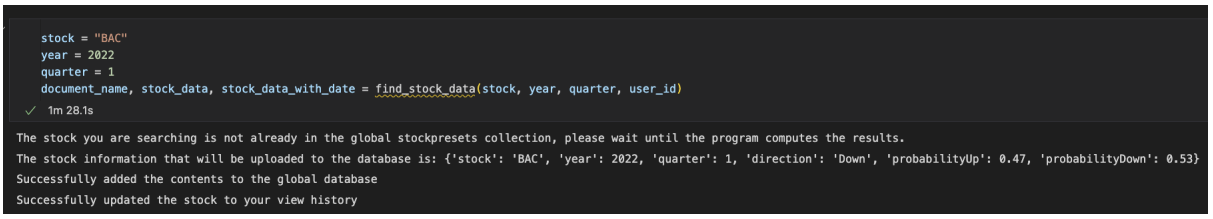


Figure 111c - Testing running time for BAC 2022 Q1

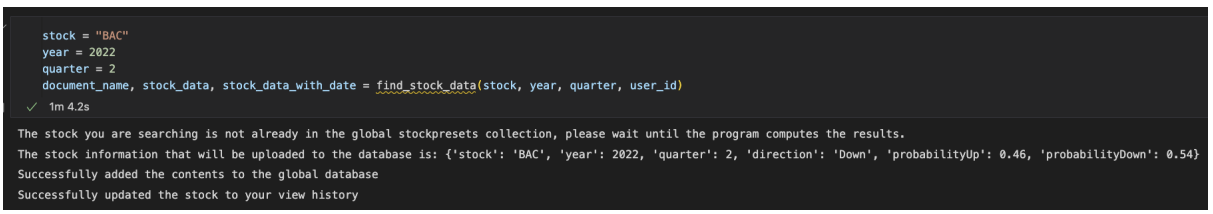


Figure 111d - Testing running time for BAC 2022 Q2

It will take 37.3 seconds for the user to predict the stock returns for FRC 2022 Q3, 33.8 seconds for NBTB 2022 Q1, 1 minute 28.1 seconds for BAC 2022 Q1, and 1 minute 4.2 seconds for BAC 2022 Q2. The disparity in running time between these dates is determined by how long the earnings transcript is (longer the earnings transcript, the more sentences to parse, hence more time required). After testing it through 10 examples, the range of processing time falls between 30 and 1 min 30 seconds, with an average of 48 seconds.

Figure 112- Global stockpresets database

To address this, I have created a "global stockpresets database" that stores all previously made predictions. This allows for instant access without the need for re-processing and parsing. As seen in the figure above, the code is completed within 0.7 seconds. First, the programme checks whether there's a record of the prediction for BAC 2022 Q2 in the database, and calls it if it exists, saving time.

As for getting the independent data (feature extraction sections, see **Feature_extract_5to96.ipynb**), each iteration through a single earnings transcript takes approximately 30 seconds to 1:30 minutes too (sample size: 8). This is a significant improvement compared to a month ago where it took 2-3 minutes to find the features of a single transcript. This improvement was achieved by enhancing the efficiency of the code through the elimination of redundancies in the data processing and optimising memory management between functions. For example, the Q&A section is parsed only once, and this parsed data is saved for subsequent processes, rather than being re-parsed when it's needed in the next section.

End User Testing:

To evaluate the usability and functionality of my stock prediction website, I enlisted the help of a friend who has some experience in stock investing. My friend Roman, used the website for around 40 mins and provided feedback on his experience. I gave him a brief introduction to the website and its features, including how to register, login, search for stocks, and save them to a favourites list. I then asked him to perform several tasks on the website.

Due to time constraints, I did not make a fully fledged website, instead, I created a Jupyter Notebook file that allowed the user to achieve basic interactions with it. He found it easy to create an account, login, update user information and search for stocks. However, when he attempted to log out, he was unable to locate the option and consequently exited the program instead. The reason for this was because I did not specify in the NEA analysis specification to include a logout feature, so I did not code it. Nonetheless, he mentioned that this was a minor problem.

Roman was able to easily query for a prediction by inputting the ticker symbol for a stock he was interested in. The search results were displayed after around 50 seconds, so he said it was quite boring to wait. Nonetheless, he felt the data that the programme provided was very useful about the stock's future potential performance.

He tested the programme on several stocks he was familiar with and by referencing yahoo finance's historical prices, he found some of the predictions to be accurate, but others not so much. I told him that the accuracy is between 60-70% and he suggested that combining this data with other types of analysis, such as a discounted cash flow model or utilising momentum technical analysis techniques like the relative strength index or MACD, may increase accuracy, which I agree with. He also appreciated that you can save these stock data to a favourites list and check your view history as well. Overall, Roman was quite interested with my stock prediction website and said he will most likely use it in his own analysis too as it was very informative. He also suggested a few minor improvements, such as the ability to sort and filter the favourites list by listing different criteria, but overall he was very satisfied with the website's performance.

Evaluation:

In addition to Roman's review, 5 other people took part to test out my programme, the average score for usability was 6.5/10 as they cited it lacked a well-designed user interface, however, it was rated 8.2/10 for functionality as it effectively fulfils the intended purpose as specified by the specification.

Overall, the outcome met the majority of its requirements. I intended to extract features from 2469 stocks, and 85620 earnings calls, then used the model to make predictions about the stock market. I did manage to extract 85620 earnings calls, however, only around 5000 earnings transcripts have been fully analysed, this is largely due to the time that's required to extract a single earnings call (each takes approx. 1 min). Nonetheless, these 5000 earnings calls represent all of the stocks for the sectors "banks", "automobiles", and "consumer durables". Hence, I was able to create a model for each of these sectors, and the user can select between them.

However, there were a few areas where the outcome could be improved if the problem were revisited. One notable area for improvement would be to incorporate more advanced deep learning models, such as LSTM, to improve the accuracy of the predictions. To do this, I must restructure the data for it to be a timeseries. I tried transforming the current data into a format where each observation represents a specific point in time, in my case, each point of time represents one single earnings call (hence quarterly intervals). However, the accuracy of the model was lower than

the binary classification model. This suggests I would have to include further time series data that integrates daily or weekly intervals into the analysis. Next, the data would need to be resampled or aggregated at the chosen time interval to create a time series dataset. Once the data has been transformed into a time series format, it would be suitable for use within a LSTM model. The reason that I may want a LSTM model is that they are very useful for predicting time series data because they can learn long-term dependencies between data points, making them well-suited for predicting stock market trends over time. Overall, incorporating more advanced deep learning models into the project would require a significant amount of additional work, but could potentially lead to more accurate predictions and a more sophisticated model overall.

The feedback from Roman also proved very helpful, however one thing that I could improve if I had more time is to include a front-end website for the user to interact with, improving user experience, like how I described in my initial specification. One thing that I realised is that my initial specification was quite too ambitious so some features were dropped due to time constraints. Furthermore, some data such as Moving average convergence/divergence is not equipt with historical data so it becomes difficult to train my model. If I do find a way to incorporate these technical/momentum trading techniques, such as by purchasing a more advanced API, I would be able to add additional time series information and improve the accuracy.

There are also other weaknesses this model face. Firstly, the machine learning model is only trained using earnings per share, and earnings transcript data (e.g. sentiment, prose, and text complexity) which means it doesn't consider other elements that may impact the price movement of a stock such as press-releases, annual/quarterly reports, economic news, momentum of the stock.

Furthermore, since I'm analysing based on qualitative data (i.e. earnings transcripts instead of quantitative data such as moving average, relative strength index etc.) this may mean the model cannot give a precise quantitative percentage estimation. In addition the features I extracted from earnings transcript at this stage may not be very useful to feed into a LSTM network as stated before, which means finding new ways to feature engineering will benefit the accuracy significantly.

The model may improve by including some of the specification points I didn't meet, such as the ability to read financial statements and spot irregularities with past statements (e.g. spotting new footnotes, and new risk factors in 10K[6] and 10Qs). To do this, I need to find a reliable API that provides this information. Another

[6] 10Ks are annual reports, 10Qs are quarterly reports filed to the SEC - Securities and Exchange Commission by publicly traded companies, typically released after the earnings report.

improvement is to include the general sentiments of retail investors towards a specific stock (and measure its changes over time) by scraping content from Twitter, Reddit, and StockTwits.

Nonetheless, at present, using a binary classification model, an effective strategy is to go long on shares predicted to overperform (e.g. labelled bin1); and to short those labelled (e.g. labelled bin2). If I was given more time, I would also feature extraction all stocks and evaluate which sector presents the highest accuracies. Another improvement is to handle feature extraction differently for each sector, for example, for the tech sector, we can add custom features such as emphathising “growth rate” in the analysis, whereas for banks, we can emphathise “balance sheet” or “interest rate”. Furthermore, we can perhaps avoid data imbalances using methods such as over/under-sampling, or by using a classification model that features a weighted loss function.

Furthermore, the conclusions I received may not be applicable in the real world since the dependent data (Y_data) is measured as the stock return between Day 0 to Day X. This may be unrealistic as stocks often move pre-market/after-hours almost immediately after the earnings call are announced. This means we cannot capitalise on the day 0 bounce. Furthermore, some transcripts may take 1-2 days before being published so it will hinder it’s real life effectiveness. Nonetheless, it’s a good starting point.

In conclusion, while the project met most of its requirements, there were areas for improvement, and independent feedback was obtained and evaluated to identify potential areas for future development.

Appendix

An Example of the website reference links path: “hrefs/ref_banks.csv”:

```
href,ticker,fair_value
/stocks/us/banks/nyse-jpm/jpmorgan-chase,JPM, ($182.25)
/stocks/us/banks/nyse-bac/bank-of-america,BAC, ($53.44)
/stocks/us/banks/nyse-wfc/wells-fargo,WFC, ($71.4)
/stocks/us/banks/nyse-c/citigroup,C, ($93.85)
/stocks/us/banks/nyse-usb/us-bancorp,USB, ($84.21)
/stocks/us/banks/nyse-pnc/pnc-financial-services-group,PNC, ($264.42)
/stocks/us/banks/nyse-tfc/truist-financial,TFC, ($82.27)
/stocks/us/banks/nyse-mtb/mt-bank,MTB, ($373.15)
/stocks/us/banks/nasdaq-fitb/fifth-third-bancorp,FITB, ($64.62)
/stocks/us/banks/nyse-frc/first-republic-bank,FRC, ($156.3)
/stocks/us/banks/nasdaq-hban/huntington-bancshares,HBAN, ($29.69)
/stocks/us/banks/nyse-rf/regions-financial,RF, ($51.41)
/stocks/us/banks/nyse-nu/nu-holdings,NU, ($0.71)
/stocks/us/banks/nyse-cfg/citizens-financial-group,CFG, ($79.6)
/stocks/us/banks/nyse-key/keycorp,KEY, ($32.4)
/stocks/us/banks/nasdaq-sivb/svb-financial-group,SIVB, ($399.16)
/stocks/us/banks/nyse-fhn/first-horizon,FHN, ($41.1)
/stocks/us/banks/nyse-rkt/rocket-companies,RKT, ($0.14)
/stocks/us/banks/nyse-bap/credicorp,BAP, ($200.87)
/stocks/us/banks/nasdaq-fcnc.a/first-citizens-bancshares,FCNC.A, ($1495.08)
/stocks/us/banks/nasdaq-ewbc/east-west-bancorp,EWBC, ($199.54)
/stocks/us/banks/nyse-cfr/cullen-frost-bankers,CFR, ($207.71)
/stocks/us/banks/nyse-cma/comerica,CMA, ($150.25)
/stocks/us/banks/nyse-wbs/webster-financial,WBS, ($118.39)
/stocks/us/banks/nasdaq-cbsh/commerce-bancshares,CBSH, ($90.17)
/stocks/us/banks/nasdaq-sbny/signature-bank,SBNY, ($519.89)
/stocks/us/banks/nasdaq-zion/zions-bancorporation-national-association,ZION, ($114.56)
/stocks/us/banks/nyse-wal/western-alliance-bancorporation,WAL, ($208.61)
/stocks/us/banks/nasdaq-bokf/bok-financial,BOKF, ($170.38)
/stocks/us/banks/nasdaq-ssb/southstate,SSB, ($160.67)
/stocks/us/banks/nyse-pb/prosperity-bancshares,PB, ($114.08)
/stocks/us/banks/nyse-gbci/glacier-bancorp,GBCI, ($63.37)
/stocks/us/banks/nyse-uwmc/uwm-holdings,UWMC, ($17.57)
/stocks/us/banks/nasdaq-vly/valley-national-bancorp,VLY, ($27.45)
/stocks/us/banks/nasdaq-pnfp/pinnacle-financial-partners,PNFP, ($160.3)
/stocks/us/banks/nyse-snv/synovus-financial,SNV, ($79.84)
/stocks/us/banks/nasdaq-onb/old-national-bancorp,ONB, ($32.5)
/stocks/us/banks/nasdaq-ubsi/united-bankshares,UBSI, ($54.91)
/stocks/us/banks/nasdaq-wtfc/wintrust-financial,WTFC, ($183.05)
/stocks/us/banks/nasdaq-ffin/first-financial-bankshares,FFIN, ($35.87)
/stocks/us/banks/nasdaq-ozk/bank-ozk,OZK, ($103.8)
/stocks/us/banks/nyse-homb/home-bancshares-conway-ar,HOMB, ($42.2)
/stocks/us/banks/nyse-cade/cadence-bank,CADE, ($55.16)
/stocks/us/banks/nasdaq-bpop/popular,BPOP, ($220.99)
/stocks/us/banks/nyse-fnb/fnb,FNB, ($27.26)
/stocks/us/banks/nasdaq-hwc/hancock-whitney,HWC, ($132.45)
/stocks/us/banks/nasdaq-fibk/first-interstate-bancsystem,FIBK, ($78.55)
/stocks/us/banks/nasdaq-umpq/umpqua-holdings,UMPQ, ($53.02)
/stocks/us/banks/nasdaq-act/enact-holdings,ACT, ($64.61)
/stocks/us/banks/nyse-nycb/new-york-community-bancorp,NYCB, ($11.06)
/stocks/us/banks/nyse-esnt/essent-group,ESNT, ($123.97)
/stocks/us/banks/nyse-sfbs/servisfirst-bancshares,SFBS, ($109)
/stocks/us/banks/nasdaq-ucbi/united-community-banks,UCBI, ($64.25)
/stocks/us/banks/nasdaq-umbf/umb-financial,UMBF, ($164.18)
/stocks/us/banks/nasdaq-cvbf/cvb-financial,CVBF, ($39.65)
/stocks/us/banks/nasdaq-indb/independent-bank,INDB, ($109.93)
/stocks/us/banks/nyse-mtg/mgic-investment,MTG, ($48.46)
/stocks/us/banks/nasdaq-tfsl/tfs-financial,TFSL, ($3.11)
/stocks/us/banks/nyse-asb/associated-banc-corp,ASB, ($35.65)
/stocks/us/banks/nasdaq-abcb/ameris-bancorp,ABCB, ($110.15)
/stocks/us/banks/nasdaq-ppbi/pacific-premier-bancorp,PPBI, ($60.14)
/stocks/us/banks/nyse-cbu/community-bank-system,CBU, ($78.18)
/stocks/us/banks/nasdaq-caty/cathay-general-bancorp,CATY, ($121.81)
/stocks/us/banks/nasdaq-banf/bancfirst,BANF, ($147.05)
/stocks/us/banks/nasdaq-fhb/first-hawaiian,FHB, ($44.5)
/stocks/us/banks/nasdaq-iboc/international-bancshares,IBOC, ($65.24)
/stocks/us/banks/nyse-rdn/radian-group,RDN, ($67.28)
/stocks/us/banks/nasdaq-fult/fulton-financial,FULT, ($39.08)
/stocks/us/banks/nyse-boh/bank-of-hawaii,BOH, ($122.61)
/stocks/us/banks/nasdaq-ebc/eastern-bankshares,EBC, ($6.68)
/stocks/us/banks/nasdaq-sfnc/simmons-first-national,SFNC, ($42.12)
```

```
/stocks/us/banks/nasdaq-coop/mr-cooper-group,COOP, ($242.76)
/stocks/us/banks/nasdaq-wsfs/wsfs-financial,WSFS, ($59.03)
/stocks/us/banks/nasdaq-tcbi/texas-capital-bancshares,TCBI, ($58.73)
/stocks/us/banks/nasdaq-pacw/pacwest-bancorp,PACW, ($66.52)
/stocks/us/banks/nyse-fbp/first-bancorp,FBP, ($38.7)
/stocks/us/banks/nyse-bku/bankunited,BKU, ($49.62)
/stocks/us/banks/nasdaq-colb/columbia-banking-system,COLB, ($44.77)
/stocks/us/banks/nyse-pfsi/pennymac-financial-services,PFSI, ($65.22)
/stocks/us/banks/nasdaq-frme/first-merchants,FRME, ($88.21)
/stocks/us/banks/nasdaq-ibtx/independent-bank-group,IBTX, ($103.56)
/stocks/us/banks/nyse-wd/walker-dunlop,WD, ($167.65)
/stocks/us/banks/nasdaq-aub/atlantic-union-bankshares,AUB, ($47.49)
/stocks/us/banks/nasdaq-wafd/washington-federal,WAFD, ($51.55)
/stocks/us/banks/nasdaq-ffbc/first-financial-bancorp,FFBC, ($44.33)
/stocks/us/banks/nasdaq-banr/banner,BANR, ($129.79)
/stocks/us/banks/nasdaq-wsbc/wesbanco,WSBC, ($42.78)
/stocks/us/banks/nasdaq-clbk/columbia-financial,CLBK, ($10.07)
/stocks/us/banks/nyse-ax/axos-financial,AX, ($97.3)
/stocks/us/banks/nysemkt-prk/park-national,PRK, ($188.66)
/stocks/us/banks/nasdaq-town/townebank,TOWN, ($50.35)
/stocks/us/banks/nasdaq-trmk/trustmark,TRMK, ($55.04)
/stocks/us/banks/nasdaq-rnst/renasant,RNST, ($61.76)
/stocks/us/banks/nasdaq-sybt/stock-yards-bancorp,SYBT, ($81.55)
/stocks/us/banks/nasdaq-htlf/heartland-financial-usa,HTLF, ($106.34)
/stocks/us/banks/nasdaq-lkfn/lakeland-financial,LKFN, ($92.56)
/stocks/us/banks/nasdaq-nbtb/nbt-bancorp,NBTB, ($79.28)
/stocks/us/banks/nyse-fbk/fb-financial,FBK, ($66.36)
/stocks/us/banks/nasdaq-efsc/enterprise-financial-services,EFSC, ($116.3)
/stocks/us/banks/nasdaq-sbcf/seacoast-banking-corporation-of-florida,SBCF, ($45.35)
/stocks/us/banks/nasdaq-nwbi/northwest-bancshares,NWBI, ($21.69)
/stocks/us/banks/nyse-fbc/flagstar-bancorp,FBC, ($85.53)
/stocks/us/banks/nasdaq-tcbk/trico-bancshares,TCBK, ($93.26)
/stocks/us/banks/nyse-hth/hilltop-holdings,HTH, ($60.64)
/stocks/us/banks/nyse-nbhc/national-bank-holdings,NBHC, ($77.26)
/stocks/us/banks/nasdaq-stel/stellar-bancorp,STEL, ($61.96)
/stocks/us/banks/nasdaq-nmih/nmi-holdings,NMIH, ($77.6)
/stocks/us/banks/nasdaq-vbtx/veritex-holdings,VBTX, ($62.98)
/stocks/us/banks/nasdaq-wabc/westamerica-bancorporation,WABC, ($61.15)
/stocks/us/banks/nyse-pfs/provident-financial-services,PFS, ($33.84)
/stocks/us/banks/nasdaq-tbbk/bancorp,TBBK, ($50.51)
/stocks/us/banks/nasdaq-fbnc/first-bancorp,FBNC, ($91.26)
/stocks/us/banks/nasdaq-hope/hope-bancorp,HOPE, ($30.55)
/stocks/us/banks/nasdaq-sasr/sandy-spring-bancorp,SASR, ($63.35)
/stocks/us/banks/nyse-ntb/bank-of-nt-butterfield-son,NTB, ($94.12)
/stocks/us/banks/nasdaq-buse/first-busey,BUSE, ($53.9)
/stocks/us/banks/nasdaq-egbn/eagle-bancorp,EGBN, ($97.71)
/stocks/us/banks/nasdaq-chco/city-holding,CHCO, ($145.98)
/stocks/us/banks/nasdaq-stba/st-bancorp,STBA, ($67.32)
/stocks/us/banks/nasdaq-srce/1st-source,SRCE, ($77.73)
/stocks/us/banks/nasdaq-ocfc/oceanfirst-financial,OCFC, ($55.13)
/stocks/us/banks/nasdaq-tbk/triumph-bancorp,TBK, ($69.46)
/stocks/us/banks/nasdaq-lob/live-oak-bancshares,LOB, ($56.27)
/stocks/us/banks/nyse-fcf/first-commonwealth-financial,FCF, ($34.43)
/stocks/us/banks/nyse-ofg/ofg-bancorp,OFG, ($83.52)
/stocks/us/banks/nasdaq-dcom/dime-community-bancshares,DCOM, ($12.74)
/stocks/us/banks/nyse-bhlb/berkshire-hills-bancorp,BHLB, ($45.33)
/stocks/us/banks/nasdaq-obnk/origin-bancorp,OBNK, ($89.36)
/stocks/us/banks/nyse-agm/federal-agricultural-mortgage,AGM, ($61.84)
/stocks/us/banks/nasdaq-cash/pathward-financial,CASH, ($89.03)
/stocks/us/banks/nasdaq-lbai/lakeland-bancorp,LBAI, ($36.02)
/stocks/us/banks/nysemkt-tmp/tompkins-financial,TMP, ($109.84)
/stocks/us/banks/nasdaq-hfwa/heritage-financial,HFWA, ($58.87)
/stocks/us/banks/nasdaq-gabc/german-american-bancorp,GABC, ($69.04)
/stocks/us/banks/nyse-nic/nicolet-bankshares,NIC, ($162.99)
/stocks/us/banks/nasdaq-sbsi/southside-bancshares,SBSI, ($61.53)
/stocks/us/banks/nasdaq-intr/inter-co,INTR, ($0.92)
/stocks/us/banks/nasdaq-cffn/capitol-federal-financial,CFFN, ($4.81)
/stocks/us/banks/nyse-si/silvergate-capital,SI, ($118.53)
/stocks/us/banks/nasdaq-pfbc/preferred-bank,PFBC, ($210.69)
/stocks/us/banks/nasdaq-mbin/merchants-bancorp,MBIN, ($119.32)
/stocks/us/banks/nasdaq-pfc/premier-financial,PFC, ($61.77)
/stocks/us/banks/nasdaq-brkl/brookline-bancorp,BRKL, ($21.53)
```

```
/stocks/us/banks/nyse-cubi/customers-bancorp,CUBI, ($125.99)
/stocks/us/banks/nyse-banc/banc-of-california,BANC, ($31.39)
/stocks/us/banks/nasdaq-amtb/amerant-bancorp,AMTB, ($37.72)
/stocks/us/banks/nasdaq-cnob/connectone-bancorp,CNOB, ($49.32)
/stocks/us/banks/nasdaq-rbca.a/republic-bancorp,RBCA.A, ($96.46)
/stocks/us/banks/nasdaq-qcrh/qcr-holdings,QCRH, ($125.42)
/stocks/us/banks/nasdaq-htbk/heritage-commerce,HTBK, ($27.49)
/stocks/us/banks/nyse-by/byline-bancorp,BY, ($39.65)
/stocks/us/banks/nasdaq-abtx/allegiance-bancshares,ABTX, ($40.51)
/stocks/us/banks/nasdaq-ctbi/community-trust-bancorp,CTBI, ($88.93)
/stocks/us/banks/nasdaq-wash/washington-trust-bancorp,WASH, ($59.11)
/stocks/us/banks/nasdaq-pebo/peoples-bancorp,PEBO, ($82.46)
/stocks/us/banks/nasdaq-bfc/bank-first,BFC, ($165.04)
/stocks/us/banks/nasdaq-uvsp/univest-financial,UVSP, ($46.69)
/stocks/us/banks/nasdaq-ffwm/first-foundation,FFWM, ($23.68)
/stocks/us/banks/nasdaq-fbms/first-bancshares,FBMS, ($72.59)
/stocks/us/banks/nasdaq-hafc/hanmi-financial,HAFC, ($49.16)
/stocks/us/banks/nasdaq-gsbc/great-southern-bancorp,GSBC, ($101.45)
/stocks/us/banks/nasdaq-osbc/old-second-bancorp,OSBC, ($43.3)
/stocks/us/banks/nasdaq-amal/amalgamated-financial,AMAL, ($35.21)
/stocks/us/banks/nasdaq-nfbk/northfield-bancorp-staten-island-ny,NFBK, ($14.51)
/stocks/us/banks/nyse-mcb/metropolitan-bank-holding,MCB, ($180.19)
/stocks/us/banks/nasdaq-fmbh/first-mid-bancshares,FMBH, ($71.65)
/stocks/us/banks/nasdaq-pgc/peapack-gladstone-financial,PGC, ($100.74)
/stocks/us/banks/nasdaq-trst/trustco-bank-corp-ny,TRST, ($52.99)
```

An Example of the contents of one earnings transcript.csv file (ADNT 2021 Q3):

```
2021
3
2021-08-07 21:47:09
"Operator: Welcome, and thank you all for standing by.  Today's call is also being recorded. If anyone does have any objections, you may disconnect at this time. And I would now like to turn the call over to Mr. Mark Oswald. Thank you. You may begin.
Mark Oswald: Thank you, Sue. Good morning, and thank you for joining us as we review Adient's results for the third quarter of fiscal year 2021. The press release and presentation slides for our call today have been posted to the Investors section of our website at adient.com. This morning, I'm joined by Doug Del Grosso, Adient's President and Chief Executive Officer; Jeff Stafeil, our Executive Vice President and Chief Financial Officer; and Jerome Dorlack, Executive Vice President, Head of Americas Seating. On today's call, Doug will provide an update on the business followed by Jeff, who will review our Q3 financial results and outlook for the remainder of the fiscal year. After our prepared remarks, we will open the call to your questions. Before I turn the call over to Doug and Jeff, there are a few items I'd like to discuss. First, today's conference call will include forward-looking statements. These statements are based on the environment as we see it today, and therefore, involve risks and uncertainties. I would caution you that our actual results could differ materially from these forward-looking statements made on the call. Please refer to Slide 2 of the presentation for our complete safe harbor statement. In addition to the financial results presented on a GAAP basis, we will be discussing non-GAAP information that we believe is useful in evaluating the company's operating performance. Reconciliations for these non-GAAP measures to the closest GAAP equivalent can be found in the appendix of our full earnings release. This concludes my comments. I'll now turn the call over to Doug. Doug?
Doug Del Grosso: Great. Thanks, Mark. Good morning. Thank you to our investors, prospective investors and analysts joining the call this morning as we review our third quarter results for fiscal 2021. Turning to Slide 4. Let me begin with a few comments related to our third quarter. As we anticipated heading into the quarter, the ongoing supply chain disruptions related to semiconductors and the resulting customer production stoppages, combined with escalating commodity prices, provided a difficult operating landscape for the industry and Adient. I'll dig deeper to those macro pressures in just a minute. Despite the many challenges, the Adient team remained focused and successfully executed items within our control, while at the same time took proactive actions to lessen the impact of the production stoppages and lost sales. Starting on the left-hand side of the slide. Adient's revenue for the quarter was $3.2 billion, up significantly from the $1.6 billion reported in Q3 fiscal '20. Last year's results, as you're aware, were significantly impacted by vehicle production shutdowns across Europe and Americas due to COVID-19. Although revenue increased year-over-year, this year's top line was also impacted by significant headwinds, specifically the numerous production stoppages at our customers due to semiconductor supply disruptions. Adjusted EBITDA for the quarter totaled $118 million. And as pointed out on the slide, this included just over $50 million in premiums and temporary operating inefficiencies, again, primarily driven by chip shortages and unplanned production stoppages. Adient's June 30 cash balance totaled $1 billion. I'll point out that the cash balance does not include about $270 million held in other assets as a deposit related to certain assets Adient is acquiring from the YFAS joint venture as part of our China strategic transformation. In addition, about $190 million of cash was used during the quarter to repay a portion of the company's debt. Finally, speaking of debt, Adient's gross and net debt totaled just over $3.7 billion and $2.7 billion, respectively. Approximately $180 million of principal debt prepayment occurred during Q3. Jeff will provide additional color on Adient's financial results, including our capital structure in just a few minutes. Achieving the results just discussed was hard-fought, especially considering the near-perfect storm of temporary headwinds that impacted the industry and Adient. These included supply chain disruptions and unplanned production stoppages, increased freight costs,
```

significant increases in commodity prices and continued costs associated with COVID-19, just to name a few. Rather than dwell on the negatives, the team looked ahead and began to implement actions to help lessen the impact. On the right-hand side of the slide, we've highlighted certain of those proactive measures. Actions taken included: pulling ahead preventive maintenance, reallocating our resources between plants, implementing involuntary/voluntary layoffs at our plants in the U.S. and Canada, forced vacations for impacted salaried team members and keeping very tight controls on discretionary spending, including travel, which is essentially limited to launch activities and hiring. The list shown is not exhaustive, but should provide you an idea of how the company continues to drive the business forward, essentially executing on actions that are within our control. Turning to Slide 5. Let me shift gears slightly and spend a few minutes discussing the current operating environment as the factors shown on the slide are really dominating and influencing the business near term as evidenced in our Q3 results. First, despite the many negative headlines, there are a number of positive influence impacting the industry, including: consumer demand, which remains extremely strong; vehicle inventories, which are at near or historic levels and will continue to drive strong production for several quarters as manufacturers look to rebuild inventory; vehicle mix, which remains strong as manufacturers continue to protect reduction of their most profitable trucks and SUVs; and finally, COVID treatments and vaccines are generally driving the reopening of many economies. We continue to watch closely as certain variants such as the Delta variant could result in disruptions along the way. That said, and focusing on the headwinds column, there are a number of headwinds, which we view as temporary that are masking or limiting the full benefit of the positive influence just discussed. They include ongoing supply chain semiconductor shortages, which continue to result in production downtime at our customers. As seen with our Q2 and Q3 results, these unplanned production stoppages are leading to premiums and operating inefficiencies across our network. That said, for fiscal 2021, we estimate the supply chain disruptions and the resulting lost production, operating efficiencies, premium freight, et cetera, will have a net impact on the top line of about $1.1 billion and adjusted EBITDA by approximately $300 million. I wish I could tell you this will be limited to 2021, unfortunately, as shown in our call-out box on the far right, the visibility of our customers' production schedules has not improved in the course of the past few months. Discussions and commentary from our customers suggest improving supplies of chips and production schedules in the coming weeks and months. Unfortunately, we continue to experience very short notice of production downtime, and it's likely this trend will continue through the rest of our fiscal year and even into 2022. Also having a significant impact on the business, as mentioned in our Q2 earnings call, continues to be escalating steel and chemical prices. As noted on the slide, steel prices in the Americas are up approximately 3x versus the beginning of the year. And although Adient has certain mechanisms and pass-through agreements with our customer, they were never intended to cover movements of this magnitude for this long of duration. For 2021, we estimate the net impact to be about $80 million, consistent with what we shared in our Q2 call. Although these items appear to be temporary in nature and should lessen over time, they are placing a significant amount of strain on the near-term business. The question remains, when will these pressures subside? Our best guess is the supply chain disruption and increased commodity costs will likely persist as we enter 2022. Jeff will dig deeper into the impact the rising commodity prices is having on Adient in his prepared remarks. Turning to Slide 6. Shifting back to the theme of executing actions within our control, let me provide a few comments related to Adient's long-term objectives. First, we are on track with our strategic transformation in China. This includes, among other agreements, the termination of the YFAS joint venture and the acquisition of an additional 50% equity interest in CQYFAS, resulting in us owning 75% interest in CQ and 100% equity interest in the YFAS Langfang. I hope to share news of the completion of the agreements in the not-so-distant future. As a reminder, Adient is expecting cash proceeds of about $1.4 billion after tax with the first tranche of proceeds collected at the closing and the remaining proceeds collected prior to the 2021 calendar year-end. The completed transaction will enable Adient to drive our strategy in China independently, which is expected to result in a variety of benefits, including capturing growth in profitable and expanding segments, improving the integration of the company's China operations and allowing for more certain value realization relative to the status quo, where cash and value are generated from dividends at entities not in Adient's control. As mentioned in March, we expect to remain a leader in the China market, essentially a position on par with and among the top 3 seat suppliers in the market. Based on our estimates, this equates to a market share of just under 20%. Equally exciting and in part enabled by the China transformation is the work underway to transform Adient's capital structure. As mentioned earlier, during Q3, we paid down the final $160 million in principal of the company's 7% first lien notes. In total, so far in fiscal '21, the company has prepaid approximately $840 million in debt. When combining the 7% note paydown and the prepayments made to the European Investment Bank loan, I'd also remind you the company made $100 million of voluntary debt prepayment at the end of fiscal 2020, bringing our total voluntary debt paydown to $940 million since the beginning of the capital structure transformation in Q4 fiscal year '20. Adient's capital structure has a lot of flexibility and callability remaining. We expect to continue our deleveraging efforts on completion of the China transaction and collection of the related proceeds. Whether it's the longer-term strategies just discussed or the day-to-day initiatives, such as improving our plant operation and flawlessly executing on launches, Adient is executing actions within our control despite the macro headwinds that are temporarily impacting the industry. The continued push to improve the business has had a positive impact on our relationships with customers as well. We've seen Adient's customers recognize value-added product and process initiatives. At the bottom of the slide, we've included a few awards Adient recently received, including: Stellantis' Best Supplier Competitiveness award, Toyota's Superior Supplier Diversity award, GM's Excellence in On-Time Shipping award, and finally, the Outstanding Quality Launch Performance award from Nissan. I mention this recognition only to provide proof points that the team continues to execute on many fronts, continually looking for ways to drive the business forward. Turning to Slide 7. Let me provide a few comments on Adient's business wins and how we're strengthening our leading market position. What hopefully has been clear as we reported our key wins over the last several quarters is the fact that Adient continues to focus on capital allocation, return on capital when targeting new or incumbent business wins and has not limited our ability to secure new business. Our year-to-date win rate for both targeted new business and targeted incumbent business is tracking in line with our expectations. We've highlighted a number of recent program wins here, including a number of key replacement wins this past quarter, including: Toyota Tacoma, Mercedes A-Class, Mercedes GLA and the Nissan Patrol. In addition to these replacement wins, Adient also secured new SUV program with XPeng in China and the new EV CUV platform at Honda. It should be noted that our recent wins, awards include a good mix, a combination of JIT, foam, trim and metals business. As our new book of business continues to launch, we expect to balance in and balance out platforms to further enable margin expansion. Turning to Slide 8. As we typically do, we've highlighted several critical launches that are complete, in process or scheduled to begin in the near term. I'm happy to report we're heading into the final few months of the fiscal year, and the team continues to focus on process discipline around launch readiness and has driven a very high level of performance, especially considering the launch load and the complexity of launches that were planned for this year. In addition to the number of launches and complexity, the disruption to production schedules presented another layer of challenges that the team successfully managed through though. Again, a testament to the discipline around our processes. We have no intention of letting up and look forward to finishing the year strong with the launches that are in process and scheduled to begin entering fiscal 2022. Before turning the call over to Jeff, flipping to Slide 9, let me just conclude my remarks with a few comments about Adient's guiding principles. These principles are intended to drive Adient forward while focusing on what's most important. The key drivers, customer, quality, people, community and financial discipline, guide and inform our business strategy and our culture. As pointed

out today, remaining focused on these drivers enable the team to drive the business forward even while operating in a challenging environment we're currently facing. I'm proud of what we've accomplished and excited about what lies ahead. And with that, I'll turn the call over to Jeff to take us through Adient's third quarter 2021 financial performance and provide a little bit more color on what's expected as we wrap up 2021.

Jeff Stafeil: Thanks, Doug. Good morning, everyone. Before jumping into the financial results, which, to a certain extent, provides less insight into the ongoing operations given the abnormal operating environments in both Q3 of last year and this year, let me spend a few minutes discussing commodity inflation. Specifically, what we're seeing today, how it's impacting the business and steps the company is taking to lessen the impact, especially as it relates to the out-years. First, on Slide 11, we've provided a chart illustrating price movements and expected price movements for hot-rolled steel in North America. As you can see, despite repeating forecasts that prices will fall, prices have continued to increase while forecast for price decreases continue to be pushed out. For Adient, this has resulted in more than a 3x increase in the cost over the last 12 months for steel. Assuming prices remain constant, the impact on our cost for steel and chemicals would be approximately $650 million higher or more in 2022 than prices paid in 2020. The $650 million is based on current market conditions, such as hot-rolled steel prices in excess of $1,800 a ton. If the current market conditions hold and we do nothing commercially to pass through this increase to our customers beyond our current commercial agreements, net commodity price headwind for fiscal '22 would be approximately $200 million versus 2021. However, as you would expect, we're taking aggressive actions to mitigate the impact, which I'll discuss further on Slide 12. And for that reason, it's premature for us to know how much of this headwind will be realized in 2022. We've spoken at length on prior calls with regard to Adient's recovery mechanisms that are in place to recover material cost changes. As a reminder, our pass-through agreements differ by customers in 2 key attributes, lag and percentage of contractual recovery. For lag, certain customers are nearly immediate. Others, such as our Japanese-headquartered customers, tend to be trued up on an annual basis or essentially a 5-quarter lag. This results in an average lag for Adient of a little more than 2 quarters. For the contractual piece, some customers have 100% pass-through with lag, while others might have only a 60% contractual pass-through. These pass-throughs have generally been effective despite the lag in protecting the company over the cycle as commodity prices have fluctuated. The mechanisms tend to work best against minor, short duration price movements. They are not designed for the extreme increases impacting the business today. Turning to Slide 12. On the left-hand side of the slide, we've provided an illustrative example of how movements in commodity prices impact Adient's financial results. In general, as you would expect, Adient's financial results are negatively impacted as prices rise, but positively impacted as prices decline. Important to point out, prices through the cycle have generally reverted to the mean. As material cost goes up, the revenue change, a true-up from the customer, will lag, resulting in reduced EBITDA. Only when the cost goes down and when revenue change again lags will we recover the lost profits and vice versa. Adient generally recaptures approximately 70% of commodity inflation through automatic mechanisms, while the remaining 30% must be negotiated through hard-fought, roll up the sleeves type negotiations. You'll also see on the chart, we have included a pair of dotted lines, which we view as normal movements in commodity prices over a cycle, call it, somewhere around 15%, plus or minus. As discussed in the prior slide, Adient's mechanisms in place today, combined with manual commercial negotiations, have enabled the company to recapture nearly all cost increases over time in this environment. Unfortunately, this is not the environment we're in today and is the reason the company is executing aggressive actions in an attempt to mitigate the impact of rising commodity cost. These include, but are not limited to, renegotiating commercial agreements with our customers to reduce time lags associated with true-ups and ideally eliminating Adient's portion of the pain share agreements, recognizing Adient's position as a value-add supplier. And finally, making sure the underlying indices that underpin the true-ups are actually aligned with the commodities being purchased. For example, we currently have an agreement in place with one customer where the true-up is based on a scrap steel index, which is not reflective of the steel being purchased. As you might expect, the discussions are not one-size-fits-all. We're tailoring the discussions based on the size of the exposure, relationship with the customer, et cetera. Given Adient's timing of commodity purchases and continued escalation in steel prices and to a lesser extent, chemical prices, it appears commodity prices will remain a headwind entering 2022. That's why it's imperative we move quickly to address this headwind. That said, the outcomes of these ongoing discussions will ultimately determine the magnitude of the headwind next year and that's why it's premature for us to dimension the risk at this time. Speaking of 2022, the team is in process of developing our fiscal '22 plan. We've discussed one of the big unknowns, commodities. The other driver will be vehicle production. As Doug mentioned earlier, our visibility into our customer production schedule is currently quite low. We hope as we exit fiscal 2021 and enter fiscal '22 visibility improves, especially as it relates to the semiconductor supply chain. I anticipate sharing our 2022 planning assumptions with you as we typically do during our Q4 financial results conference call in early November. Now let's move to Slide 14 and shift gears to Adient's Q3 financial results. Adhering to our typical format, the page is formatted with the reported results in the left and our adjusted results on the right-hand side of the page. We will focus our commentary on the adjusted results, which exclude special items that we view as either onetime in nature or otherwise skew important trends in underlying performance. For the quarter, the biggest drivers of the difference between our reported and adjusted results relate to indirect tax recoveries in Brazil, write-off of deferred financing charges resulting from debt repayment, transaction costs, restructuring costs and purchase accounting amortization. One other significant adjustment important to note relates to a derivative loss on the Yanfeng transaction. As a reminder, the proceeds associated with the China strategic transformation were negotiated in RMB. Adient executed a hedge to protect our cash proceeds, movements between the USD and CNY resulted in a noncash loss. Details of these adjustments are in the appendix of the presentation. For the quarter, sales were $3.2 billion, up significantly compared to our third quarter results last year, where much of our production across Europe and America was shut down due to COVID. Although up year-over-year, Adient's sales in the most recent quarter were significantly impacted by lost production, primarily driven by supply chain disruptions related to semiconductors. Adjusted EBITDA for the quarter was $118 million, up $240 million year-on-year, more than explained by an increase in volume and mix. Unfortunately, the benefits from increased volume were significantly offset by numerous temporary operating inefficiencies, which, by the way, masked business performance improvements made to the core ongoing operations. For example, of the approximate $600 million in lost sales in our Americas operation, we received notification of the reduction less than 3 days in advance, while we received more than 7 days notice in less than 20% of the time. This is obviously a difficult environment to run a JIT operation. In addition, rising commodity costs and lower equity income also had a negative impact in the quarter. I'll expand on these key drivers in just a minute. Finally, at the bottom line, Adient reported a net loss of $50 million or a loss of $0.53 per share. Now let's break down our third quarter results in more detail, starting with revenue on Slide 15. We reported consolidated sales of $3.2 billion, an increase of $1.6 billion compared to the same period a year ago. The primary driver of the year-over-year increase was attributed to higher volume and mix, call it, $1.5 billion, and to a much lesser extent, the positive impact of currency movements between the 2 periods of about $68 million. Portfolio adjustments executed in fiscal '20 provided a very minor offset to the benefits of volume and FX, call it, about $16 million. Focusing on the table on the right-hand side of the slide, you can see our consolidated sales failed to keep up with production in both the Americas and EMEA. The primary driver was Adient's customer mix, which was heavily weighted to manufacturers that were significantly impacted by the semiconductor shortages. These customers included Ford, Daimler, Stellantis, Renault and VW. Looking at the various third-party production forecast, it's estimated Adient's portion of the lost Q3 production volume was about 30% and 45% in the

Americas and EMEA, respectively. Of course, we view this as temporary and should reverse as the supply chain stabilizes. With regard to Adient's unconsolidated seating revenue, year-over-year results were up approximately 4%, adjusting for FX and executed portfolio changes. Significant year-over-year increases were recorded in both Americas and EMEA, again, largely driven by the fact that these operations were all but shuttered in Q3 of last year. In China, unconsolidated sales were relatively in line with industry production despite an approximate 7% decline at YFAS. Similar to what we saw with our consolidated results, sales at YFAS were heavily impacted by supply chain disruptions at their customers. Moving to Slide 16. We've provided a bridge of adjusted EBITDA to show the performance of our segments between periods. The bucket labeled corporate represents central costs that are not allocated back to the operation, such as executive office, communications, corporate finance, legal and marketing. Big picture, adjusted EBITDA was $118 million in the current quarter versus a loss of $122 million last year. The primary driver of the increase is detailed on the page, but effectively compares 2 very suboptimal quarters. Last year was obviously overwhelmed by COVID-related shutdowns, while this quarter was materially impacted by external factors outside of Adient's control, such as the semiconductor shortage and the storm in Texas earlier this year and its related fallout impact to the chemical supply and cost to name just a couple of the factors. Therefore, I do not plan to go into in-depth discussion on this page, but we've outlined the movements year-over-year in detail for your information. And that said and to give comfort or some proof points on Adient's turnaround in the last couple of years, let me give some statistics comparing Q3 '21 to Q3 2019 or the last Q3 period without significant externally generated shock to our business. Compared to that 2019 Q3, our admin expense was approximately $20 million lower this past quarter, while launch expense, ops waste and premium freight were also better by $30 million. We also realized approximately $70 million of improvements in other operating metrics, but these were unfortunately more than offset by approximately $125 million due to lower volume, $30 million in net commodities, $15 million in FX, $30 million in inefficiencies from chemical and semiconductor supply chain issues, among others. Clearly, the operating environment has been challenging and masked many of these performance improvements. One area worth mentioning in the Q3 2021 versus Q3 2020 comparison is the decline in equity income of roughly $24 million. The year-over-year decline was primarily related to performance and material economics at YFAS and to a lesser extent, the divestiture of our SJA joint venture. In the end, given all the moving pieces, the team worked hard to lessen the impact of the temporary headwinds to deliver the $244 million year-over-year improvement. To ensure enough time is allocated to the Q&A portion of the call, we've provided our detailed segment performance slides in the appendix of the presentation. High level, improved volume and mix benefited each of the regions. Ongoing business performance continued to trend in a positive direction, however, in Americas and EMEA, temporary operating inefficiencies resulting from unplanned production stoppages masked the overall improvement. Let me now shift to our cash, liquidity and capital structure on Slide 17 and 18. Starting with cash on Slide 17, I'll focus on year-to-date results as the longer time frame helps smooth some of the volatility in working capital movements. Adjusted free cash flow, defined as operating cash flow less CapEx, was $176 million. The $445 million improvement in adjusted EBITDA, net of equity, $72 million reduction in cap spending and significant improvement in trade working capital was partially offset by an expected increase in restructuring of $57 million, increase in interest paid of $36 million, elevated non-income-related taxes, specifically VAT payments and the timing of commercial activity. With regard to VAT payments, while some of the year-to-date outflow will continue to reverse as we progress through Q4, we'd expect larger-than-normal outflows in fiscal '21 related to government approved delays out of fiscal 2020 due to COVID accommodations. As noted on the right-hand side of the page, we ended the quarter with more than $1.8 billion in total liquidity comprised of cash on hand of about $1 billion and approximately $850 million of undrawn capacity under Adient's revolving line of credit. Also noted in the call-out, the June 30 cash balance excludes approximately $270 million held as other assets related to funds on deposit to acquire certain assets of YFAS. Cash used during the quarter to voluntarily pay down debt totaled about $190 million, which includes both principal and premium paid. Speaking of debt and flipping to Slide 18. In addition to showing our debt and net debt positions, which totaled just over $3.7 billion and $2.7 billion, respectively, at June 30, we've also provided a snapshot of Adient's capital structure. As you can see, the 7% first lien notes were entirely repaid at quarter end as the final $160 million in principal was paid during the quarter. In addition, $20 million of principal of the European Investment Bank loan was also repaid in Q3. It's clear the transformation of Adient's balance sheet is well underway with approximately $940 million of debt prepayment complete going back to Q4 of fiscal '20. The good news is, we're not done. Additional voluntary paydown of debt is expected as we progress through calendar 2021 with proceeds expected to be received from Adient's China transformation. With that, let's flip to Slide 19 and review our outlook for the remainder of fiscal '21. Adient's fiscal 2021 guidance has been updated to reflect the company's year-to-date results through June, our completed portfolio transactions, executed debt paydown and the current market conditions. Expectations for consolidated sales have been reduced to between $14.3 billion and $14.5 billion. The significant impact of production stoppages at our customers resulting from semiconductor shortages is driving the decline. The impact of lower-than-expected sales, combined with temporary operating inefficiencies driven by continued unplanned production stoppages, elevated commodity costs and increased freight, is expected to place downward pressure on Adient's adjusted EBITDA. Our current forecast is between $925 million and $975 million. Moving on to equity income, which is included in our adjusted EBITDA, continues to be forecast at about $230 million for the year. Interest expense, based on our debt and cash position, is expected to be $215 million, which is consistent to earlier expectations. Cash taxes in fiscal '21 have been revised down modestly, call it, about $80 million. To assist with your modeling, although volatile with fluctuations between quarters and as mentioned earlier, we continue to expect Adient's effective tax rate to be in the mid-20% range. Based on our customer launch plans, we expect capital expenditures to total about $310 million for the year. And finally, one last item for your modeling. We now expect free cash flow to be approximately $100 million in fiscal '21, which is in line with our previous range of between $50 million and $150 million. The team has worked hard to offset the top line and EBITDA pressures. In addition, our cash restructuring, which was previously forecasted at $200 million for the year, has now been reduced to about $150 million. The total remains elevated compared to our typical spend of around $100 million or less. In addition to an elevated restructuring spend, the 2021 free cash flow is negatively impacted by approximately $30 million of VAT payments that were deferred from last year into 2021. Stripping out these one-offs, Adient's free cash flow guide would have been approximately $180 million. With that, let's move to the question-and-answer portion of the call. Operator, first question, please.

Operator: Our first question is from Rod Lache with Wolfe Research.

Rod Lache: A couple of things. One is, I was just hoping you can clarify the $200 million commodity headwind that you described for next year, that's just 30% of the $650 million, I believe. Just to confirm, is that relative to 2021 or is that relative to 2020 because you are absorbing about $80 million this year? And then secondly, can you just discuss the nature of the inefficiencies? You mentioned on one slide, $300 million for the year, which I think included some revenue impact. And separately, you talked about $53 million for the quarter, which may not have included anything. I'm not sure if that's apples-and-apples. But basically, should we view the $300 million impact is a likely tailwind at some point once production normalizes?

Jeff Stafeil: Yes, Rod, I'll start on those. The first question as it relates to the material inflation and the net impact, that $200 million was compared to 2021. And you're right, 2021 had $80 million in it. It doesn't exactly, I think the computation was about that 30%. It's sort of a mix of the lag we have and then the timing of the recoveries. So that would be if we just ran through with the mechanical agreements as they stand today. As I mentioned, we're working, just with the size of the increases, those mechanisms don't make

sense for a supplier like ourselves that essentially is putting value add to materials. So we're working with our teams and our customers to help improve that, but that's where it would fall out if no actions were put forward, about $200 million in 2022 versus 2021. As it relates to the second part of your question on the $300 million, about $200 million of it relates to volume. And you can say the rest, there's the chemical supply chain issues were a bit over 40-ish. There's some container and freight type of issues, which were north of $25 million. There are still some COVID-related issues that were sort of the balance there to get you up to the $300 million.
Rod Lache: Okay. And what is a reasonable kind of aspirational target for recovery of commodities? And if you could just maybe help us a little bit with bridging beyond 2022. I mean, you'll do $950 million of EBITDA this year. You'll lose about maybe $200 million or so from the China equity income sale, but you're going to recover at least this $100 million of inefficiency and I presume this $80 million of commodities. And then in your last 10-Q, there was some disclosure about 5,000 remaining headcount reduction, which might be $300 million. So just kind of high level, what are you sort of aspiring towards as far as margin recovery as we think beyond this? I know you've got the longer-term mid-8% margin, but as we think about maybe 2023.
Jeff Stafeil: Yes. No. All good questions, Rod, and we're in the process of putting a lot of that together, but I'll give you a few crumbs here and we'll obviously work to expand. But right now, I guess I'd start with volume running at $14.3 billion to $14.5 billion or so in volume this year. We think that's a depressed level. And timing of chip production coming back is a bit uncertain here, we expect it probably hits at least our first quarter of 2022 still in some fashion, if not further. But we would expect volume to improve. We certainly would expect a lot of those inefficiencies to improve, if not fully go away. We'll continue to chip away at the material inflation issues. We'd also get a big benefit if we start to see some reductions in material prices. I highlight the steel forecast, everyone you look at continues to show that the next month or next couple of months, it's going to start to go down. It just hasn't done that yet, and that's created some of the challenges for us. We continue to drive the things that are internally within our control. I see nice improvements on our cost structure, see nice improvements on our manufacturing efficiencies, our CI benefits, et cetera, the VA/VE program we've put in place. So the margin expansion and the benefits we have from the restructuring are still there. So it's really going to be largely dependent upon the success we have of going after that material inflation and the pace of recovery on the underlying market.
Doug Del Grosso: Rod, this is Doug Del Grosso. I would just add to that, relative to material inflation, I would characterize that it is somewhat concentrated across a small group of customers. As Jeff's mentioned, we've got a wide variety of agreements. Some are working reasonably well and others are working less well, and that's why we've had the impact that we've talked about for this year. And so as we go after that, we're approaching it kind of 3 ways. One is, it can naturally correct itself. We're not sitting on our hands waiting for that to happen. We're heavily engaged with the concentrated customer base where we have those issues. I would say it's a customer base that I feel pretty confident that we can make some ground in those discussions. I'm not here today to peg a number, but it's customers that we've got excellent relationships with. And our level of patience is going to be measured to a certain degree on a longer-term outlook with those customers. Right now, we've not done anything I would call super aggressive as we deal with this issue because we always have to keep in mind, we want a sustained relationship with these customers and taking into account our backlog and new business opportunities is on our mind as we engage in those discussions.
Operator: Our next question is from Brian Johnson with Barclays.
Brian Johnson: Yes. Obviously, margins being the topic of the day. How do we get a sense of in EMEA? It still looks like, even if we exclude the $18 million business inefficiencies, a step down from last quarters. You talked about commodities. And obviously, that will help in Americas where you, of course, have your metal footprint that you're working on winding down. But what has to happen in EMEA? And what's your level of confidence in getting that segment back to a 7% and eventually peer margins?
Doug Del Grosso: Well, I'll start. And obviously, Jeff can comment as well. I think when you look at EMEA, what we talked about last quarter was, I don't want to say a bit, an unusually high level of commercial settlements that occurred. They occur every year, so it's not completely unusual. It was just calendarized at a higher level than normal that I think distorts a little bit of the margin picture. And I would suggest you have to look at EMEA margin over a longer period than a quarter-to-quarter because of that phenomenon where we just settle up on a lot of issues, and that tends to occur in Q2.
Jeff Stafeil: Yes, Q1, Q2. As I think you look at the EMEA picture, the SS&M business has been improving. I think we continue to hit all the operating metrics and other portions there. The metal movements are not helpful for that, so margin certainly there. And it's a region that's been, I highlighted on in my comments the short time frame that we've had to react to call-offs in volume in Americas, but the Europe situation is very similar and their ability to react to those is even less from a worker standpoint to flex cost base. So the operating environment has not been helpful for EMEA in where we're sitting. But from an overall metric standpoint, I think we've been pleased on the operating performance with that region. The restructuring that we're continuing to put through there will continue to drive some improvements as well. But we need a much more stable operating environment. And ideally, a quicker and better sort of mechanisms to deal with the current commodity inflation issues.
Brian Johnson: Okay. And then secondly, when I toured your plants, one point the plant manager made, it was actually one of your competitors, was that the JIT plant is just the tip of the iceberg. There's a whole supply chain, including freight, cross-docking facilities upstream from that. So to what extent is the nature of your supply chain driving premium freight? And then can that be part of your recoveries?
Doug Del Grosso: Sure. With the exception of, I'll say, ocean freight, which has had a pretty significant impact on us when we're bringing components in from namely Asia, both in the U.S., Americas and EMEA region, I would say the impact has been relatively small at this stage. Now that as these mounting pressures continue, certainly, our suppliers will want to come in and talk about it. But we've not experienced a lot of supply chain. In some cases, we've got direct suppliers who were completely immune from any issue associated with it. So I would characterize the issue, the inefficiencies mainly in our internal operation and the fact that we've got a fair amount of vertical integration, so it's all-inclusive. Yes. It hits us hard in the JIT environment because when the customer shuts down with 1 or 2 days notice, we just have to stop immediately. As we go further down the supply chain, we have an ability to manage that a little bit more effectively. We can build inventory. I think you've seen a little bit of increase in inventory as we moved into this quarter. So we can offset some of that. And we've got Jerome here who runs our operations in the Americas region. Anything you want to add to that, Jerome?
Jerome Dorlack: Yes. I mean, I think as Doug said, within the JIT site and Jeff quoted some numbers earlier around 1 to 3 days. And in that even maybe smaller window that flows down into our component plants, we're at 60% of notification falls into that, and that really puts the challenge on JIT. But as you flow down in because of our vertical integration, it's not such an impact really. The West Coast, specifically the West Coast and the port delays there for parts coming in from Asia, has been a real constraint for us. And we've actively worked to reroute freight away from the West Coast. So bring it in either on the East Coast of the U.S. now or bring it in further north into Canada and moving it down. So I think, Brian, we've done everything we can to really actively manage those aspects of it to mitigate the impacts further down in the supply chain. JIT has really been the bigger issue for us.
Jeff Stafeil: But we have paid a lot of...
Brian Johnson: Those are the backstop and dedicated?

Jeff Stafeil: Yes. I'd say the biggest issue to us has been on cost. Freight expense has gone up, driver shortages, all kinds of things. And I mentioned the $27 million bump in our overall freight. Also, when the chemical supply issues hit us, we were sending airplanes of chemicals to meet customer demand. So freight has definitely been an issue, but it's been more on the cost side.
Doug Del Grosso: Yes. And that's embedded in that $300 million number that we put out.
Jeff Stafeil: Yes, in the beginning.
Doug Del Grosso: So that was very much a specific incident that drove those inefficiencies into the supply base, but not really characteristic, I'd say, of overall chip shortage.
Operator: The next question is from Joe Spak with RBC Capital Markets.
Joe Spak: As we look through some of the walks in the back, I get the operational impact from these temporary inefficiencies you're talking about. I also noticed, I guess, like in Americas, for instance, and I believe in EMEA as well, there was a year-over-year headwind from launch. So I guess you guys have performed better on launch. Is this just like a lack of execution on launch or just actual like launch costs are higher because you're launching more programs?
Jerome Dorlack: Yes. So this is really around, if you take the Americas, in particular, it's the volume of launch that's running through the system. So it isn't inefficiency associated with launch, it's just year-over-year launch load. And so if you look at like P42QR and the programs we're launching in Tennessee for Nissan, as an example, that's a full value chain program for us. So we have the JIT, the trim, the foam and the metals. And so it's a good business. It's a significant amount of launch load coming into the business. But in terms of launch inefficiency and successive launch, we continue to trend very well from that standpoint. And it's reflected, Doug talked a little bit about the customer awards. Nissan was one in particular for a recent launch that we had. And so I'd say we have a high level of control on our launches. It's just the launch load that's coming into the business on a year-over-year standpoint.
Jeff Stafeil: And no, a year ago in COVID, there were no launches.
Joe Spak: Yes. I guess that's what I was sort of getting at. And then sorry if I missed this, a little bit busy morning. But is there sort of any more refined timing update on all the China transactions when that can close and when we can sort of see that new balance sheet?
Jeff Stafeil: Yes. No. The good news is no expected changes at this point. We're still on target, we think, to close the transaction in September, by the end of September. And the proceeds dynamic is still the same as we outlined on March 12 when we announced the transaction. So roughly half the dollars come in on that date and then the rest is paid by the end of December. And we'd still expect that time line.
Joe Spak: December of '21?
Jeff Stafeil: Correct. Yes, this December.
Operator: Our next question is from Dan Levy with Credit Suisse.
Dan Meir: I apologize if I missed this earlier, but I think you mentioned in your slides about the commercial negotiations. There's a little bit of roll up your sleeves. So maybe you could give us a sense of how the dynamics work on the commercial negotiations given the higher input cost. What's the typical timing of when you might be able to recover that? And then maybe you could just give us the latest on how that would comp versus typical recoveries, which I think you said is like $300 million in a typical year. So just trying to size out what the commercial recoveries could be. And then what's left on the, call it, lower hanging fruit, which I think you've said is mostly exhausted?
Doug Del Grosso: Yes. So I'll start at a macro level. Dan, this is Doug Del Grosso. And then, obviously, Jeff and Jerome can provide some additional details. So I'd first start out, we signaled commodity economic inflation was going to be an issue during our Q2 call. So this isn't anything that we didn't see coming. What's been challenging is what we felt the impact was going to be versus what it is now has been continually changing. And that's the one slide we have in the deck that just shows every preceding month forecast is at a higher level for an extended period of time. That said, we've gotten in front of every single one of our customers. This has not been, I'll say, a traditional discussion that we've had. We've gone to extremely high levels of our customer, explaining with a great amount of transparency the impact that it has relative to each one of their recovery mechanisms and why this is such an issue for us, not only in fiscal year '21, but if inflationary costs continue into '22, and it needs to be addressed. As I mentioned, it's concentrated across a few customers. In some cases, we have mechanisms that are in place that better manage the situation. The issue that they have is, these mechanisms are historic mechanisms that have been in place for quite some time in most cases. And to change these is a pretty fundamental change for our customers to enact. It's not like a specific commercial issue. They felt that they've got a mechanism in place, and it's just this really erratic situation we have right now that's causing it not to work, but they understand it. I think what I feel good about is with those customers, they're customers that we're pretty well positioned with. I think we've got outstanding relationships with, and I think they appreciate the value we bring. The only reason I would say I'm reluctant to put a time on it is, it's still a pretty fluid situation. And as commodity costs continue to increase, it's hard to really peg what the final impact is going to be and put some brackets around it. So we'll be working on it through this quarter. We'll have better visibility of where we're at as we go to Q4 and we start talking about our '22 outlook, but it's more than an issue that will take a quarter to resolve. But I'm fairly confident that over time, we'll be able to work this out with the customer base where we have the most significant problems with.
Dan Meir: Great. And then just as a follow-up, I know that once your transaction on the China site closes, the pro forma leverage is going to be far below the, I think the 1.5x to 2x range that you typically flagged. What would be the timing on potentially pursuing other capital allocation options once you get below that target leverage or could there be a period where you're a bit more conservative on the capital allocation side given some of the supply chain issues that we're seeing?
Jeff Stafeil: Yes, great question. I think as we finish this year, we'll come to you guys with our guide for expectations for 2022, and we'll give you a better definition around where we see things. But to your point, the more volatile the market, the more, I guess, uncertain things are, the more we'd probably delay a little bit or under those situations, I could see us delaying a little bit. But at the same time, we do think the big element here and what we earmarked a lot of that cash for is to deleverage. That is our primary path forward here. But you're right, I think we have a real good path to get under those metrics that we've outlined in the past and we'll start to outline what the intentions are on capital allocation here. We don't expect the disruption in the market to continue indefinitely, for sure.
Mark Oswald: Thank you. And it looks like we're at the bottom of the hour. So again, if there's anybody that has additional questions, please feel free to reach out to me. I'll be available for follow-up calls. Operator, thank you very much. This concludes the conference call for today.
Jeff Stafeil: Thanks, everyone.
Operator: Thank you for participating in today's conference. This concludes today's conference and you may disconnect at this time."
-4.1928
-0.019494067862689257
0.18326693227091634

```
0.199203187250996
62.3
0.010298657006230848
0.1724137931034483
0.1724137931034483
44.88
-0.339449633251537
0.09090909090909091
0.45454545454545453
57.91
-0.00690865460165304
0.1895734597156398
0.1895734597156398
51.18
0.11553784860557768
0.043824701195219126
-0.02436887507611606
0.12953367875647667
0.16580310880829016
71.48758620689655
-0.07982181708017985
0.03333333333333333
0.13333333333333333
65.73875
-0.014162812130582845
0.147239263803681
0.17177914110429449
73.67761904761905
0.14200886752870348
0.4444444444444444
0.3333333333333333
0.1662378802019007
0.17647058823529413
0.0
-0.046807752908581364
0.12408759124087591
0.18248175182481752
0.0318972733285692
0.2222222222222222
0.2222222222222222
-0.2234093655239452
0.22727272727272727
0.4090909090909091
-0.27623346918507624
0.21052631578947367
0.47368421052631576
0.0417174451491412
0.35294117647058826
0.29411764705882354
1.0
1.0
0.0
0.06661372582117717
0.06666666666666667
0
-0.0001718282699584961
0.4
0.4
-0.23009382052855057
0.2727272727272727
0.5454545454545454
-0.24995970726013184
0.0
0.25
1.0
1.0
0.0
0.8168417431895268
0.7382800063820921
0.8231021174321869
0.7232868890121237
0.664334854983298
```

```
0.7638387712058993
0.658518931220543
0.746386322884297
0.6679618374361949
0.4580748702261801
0.2862829531952669
0.32208997548230706
0.08108821038911529
-0.002641370502012519
0.055467470688495446
0.23560480784848248
0.2216058968407537
0.08108821038911529
-0.002641370502012519
0.055467470688495446
0.23560480784848248
0.2216058968407537
4404884131.829224
```

Not_found_list_stocks.txt:

```
automobiles
TEN
automobiles
XL
banks
ABTX
banks
FBC
banks
TBK
capital-goods
FBHS
commercial-services
MANT
commercial-services
NLSN
consumer-services
HLG
consumer-services
TMX
consumer-services
STON
food-beverage-tobacco
LNDC
healthcare
HNGR
healthcare
CHNG
healthcare
CVET
materials
GCP
pharmaceuticals-biotech
GBT
pharmaceuticals-biotech
CCXI
real-estate
BRG
real-estate
CTT
```

```
real-estate
APTS
semiconductors
CYBE
software
PING
software
AVLR
software
SWCH
software
ZEN
software
CTXS
tech
MOVE
transportation
GLG
```