Predicting stock price movements based on earnings calls using natural language processing and machine learning models

Victor Shao

Word Count: 2492 (excluding Note)

***Note*

My submission for the high master's prize is divided into two parts:

1. Written report
2. Python programme

This document is the written report, the python programme is located in the GitHub link for reference, or of interest: **https://github.com/Vxtr10/NLP-Earnings-Transcript.**

Abstract

Publicly-traded companies are prohibited to fabricate or deceive investors in earnings calls so it's a useful tool for stock valuations. There may exist patterns from earnings calls that may be identified by a machine learning algorithm and used to extrapolate the direction of future stock movements. Various feature extraction techniques are used to convert earnings call transcripts (texts) to machine-readable formats (vectors). The main feature extraction methods include the use of TF-IDF and Cosine similarity; sentiment analysis using the Loughran-Mcdonalds Dictionary; and various text complexity metrics. The features are then processed through a Random Forest Classifier where both Binary (one vs rest approach) and Multi-class methods were implemented. The former achieved an average accuracy of 83%, whereas multi-class methods achieved 45% and 74% accuracy, depending on the label range.

Stage 1: Gathering Initial Datasets

Earnings calls begin with the **"Safe Harbour"** statements to discuss quarterly operating results and future outlook. This is presided by the **"Questions and Answers"** section which involves a Q&A exchange between the company and the public.

First, I used the Requests and BeautifulSoup4 libraries to web scrape the Wikipedia page that contained stock tickers[1] of all S&P500 companies. Next, each ticker is looped to extract 14 data points.

These 14 data points are categorised into two datasets:

1. ***arr_full_transcripts*** - 1 data point
2. ***arr_ticker_infos*** - 13 data points

The 2-dimensional[2] dataset ***arr_ticker_infos*** contains 13 data points ordered as:

1. The ticker name
2. Year of release
3. Quarter of release

   Price % change between day X and day Y:
4. Day 0[3] - Day 10
5. Day 0 - Day 30
6. Day 0 - Day 50
7. Day 0 - Day 70
8. Day 0 - Day 90
9. Day 1 - Day 11
10. Day 1 - Day 31
11. Day 1 - Day 51
12. Day 1 - Day 71
13. Day 1 - Day 91

[1] Stock ticker is teh symbol used to represent stocks (e.g. Apple = 'AAPL', sAmazon = 'AMZN')
[2] 2D arrays are arrays in array, such as [[0, 1], [2, 3]]
[3] Day 0 = earnings release date

```
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")
```

Figure 1 - requesting and parsing HTML

Roic.ai is used for web scraping since it's easy to customise: *https://roic.ai/transcripts/**{ticker_name}**?y=**{year}**&q=**{quarter}***.

```
script = soup.find_all('script')[15].text.strip()
data = json.loads(script)
transcript_data = data['props']['pageProps']['transcriptdata']['content']
```

Figure 2 - HTML → JSON → parse into the string *'transcript_data'*

transcript_data is then appended to *arr_full_transcript*, an array which contains all historical earnings transcript of a ticker.

```
hist = ntr.history(start=date1, end=date1 + timedelta(days=10))
percentage_change = ((hist['Open'][-1])/(hist['Open'][0])-1)
```

Figure 3

The independent variables (dataset 4 to 13) are the percentage changes in opening price between day-X and day-Y of the ticker, which is collected using the popular python library *'yfinance'*. *Figure 3* illustrates the calculation of the opening price percentage change between *Day 0* and *Day 10*.

Stage 2: Feature Extraction

It's necessary to vectorise earnings transcripts using different feature extraction methods to become machine-readable. Various NLP models are used to extract relevant features from *arr_full_transcripts*, which are subsequently appended onto *arr_ticker_infos*, these include:

TF-IDF and Cosine Similarity:

1. Tf-idf_cos_sim

Loughran-Mcdonald Sentiment Analysis:

2. LM_posA[4]
3. LM_negA
4. LM_polA
5. LM_subA
6. LM_posB
7. LM_negB
8. LM_polB
9. LM_subB

   Text complexity:

10. FK_safe_harbour
11. FK_Q&A
12. GF_safe_harbour
13. GF_Q&A
14. smog_safe_harbour
15. smog_Q&A
16. FE_safe_harbour
17. FE_Q&A

Miscellaneous Functions

Before looking at specific data points, three functions are applied to certain datasets, these are:

- *Clean_Text*
    - *For all feature extraction models*
- *Remove_Stop_Words*
    - *Only used for TF-IDF and Cosine Similarity*
- *Split_Transcript*
    - *Only used for LM, FK, GF, SMOG, and FE*

[4] pos = positive
neg = negative
pol = polarity
sub = subjectivity

The '*Clean_Text*' function builds the groundwork for removing unnecessary noise, therefore it's applied to all data points.

Additional noise filtration methods such as '*Remove_Stop_Words*' and '*Split_Transcript*' are also used for other NLP feature extraction models when necessary (*Reference the programme for more details).*

Clean_Text

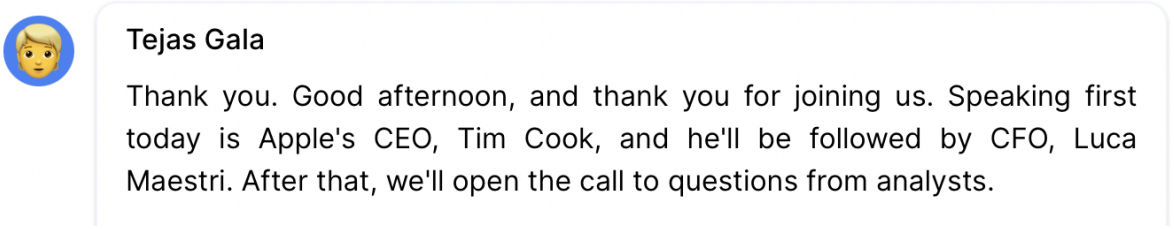

Figure 4 - Sample comment in earnings transcript

This function uses the Python RegEx library as it allows a wide range of string manipulation methods. The main features include:

- All characters to lowercase
- Deletes comments made by the Operator[5]
- Turns 'end sentence.start' to 'end sentence. start'
- Remove multiple spaced words to a single space (e.g. 'double space')
- Deletes all speaker names (e.g. Tejas Gala, Tim Cook) and any mentions of it throughout the entire transcript. It also makes sure embedded words are not deleted, such as 'tim' in the word 'estimate', or 'gala' in 'galaxy'.

Remove_Stop_Words

Stop words are common words such as 'the', 'and', 'of', 'to'. These words are often filtered out by NLP models since they themselves serve little to no value. Elimination of stop words is important for TF-IDF analysis[6] since it would only add noise to it. However, stop words are important for sentiment and text complexity analysis since a higher usage of them may hint at

[5] 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.
[6] See below for TF-IDF explanation

pessimism, or indicate higher complexity. For this reason, this function is only used for the TF-IDF function.

This function uses the Python Spacy library (i.e. subpackage *'en_core_web_sm'*) to remove stop words. Four additional words were also added to be removed including 'Operator', 'Analyst', 'Quarter', 'Year'.

Split_Transcript

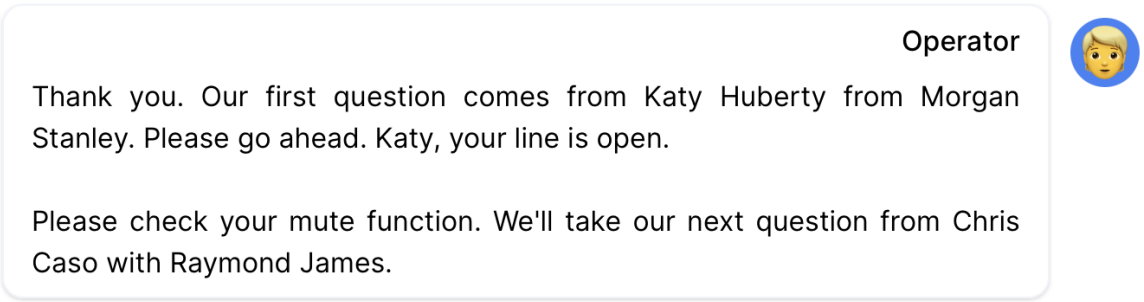

Figure 5 - Point of separation between 'Safe Harbour' and 'Questions and Answers' for Apple

As mentioned previously, an earnings call is divided into the **'Safe Harbour'** and **'Questions and Answers'**. *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, and potentially correlated with the stock's future outlook.

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[7].

[7] Corpus = collection of documents

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

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

Figure 6 - TF-IDF formula, represented by $w_{i,j}$.

With reference to Figure 6:

- ***'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 4 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

```
(1, 39)   0.14447035115215148
(1, 13)   0.1165577696502543
(1, 19)   0.14447035115215148
(1, 25)   0.14447035115215148
(1, 89)   0.14447035115215148
(1, 82)   0.14447035115215148
(1, 28)   0.14447035115215148
(1, 99)   0.14447035115215148
(1, 67)   0.14447035115215148
(1, 92)   0.14447035115215148
(1, 45)   0.14447035115215148
(1, 6)    0.1165577696502543
(1, 37)   0.1165577696502543
(1, 12)   0.1165577696502543
(1, 18)   0.14447035115215148
(1, 24)   0.14447035115215148
(1, 81)   0.14447035115215148
(1, 8)    0.1165577696502543
(1, 27)   0.14447035115215148
(1, 98)   0.14447035115215148
(1, 66)   0.14447035115215148
(1, 91)   0.14447035115215148
(1, 44)   0.14447035115215148
(1, 60)   0.14447035115215148
(1, 14)   0.1165577696502543
```

Figure 7a (left) - More visual example of a TF-IDF vector

Figure 7b (right) - Matrices example of a TF-IDF vector where each coordinate represents a word

I used Python's ML library Scikit Learn for TF-IDF vectorisation. The output will be in the form of a 100 featured vector (see figure 7b), this means the TF-IDF vector will include the top 100 highest TF-IDF valued words. The programme calculates the individual TF-IDF vector by comparing one earnings transcript with the 4 most-recent transcripts (whenever possible, otherwise return None).

The corpus of a company's 4 most recent earnings calls gives a more balanced picture of how similar the current call is to the past 4 earnings calls. However, using the TF-IDF vector itself as a feature holds little value since it can't be compared with others. For example, Apple may include words such as 'iPad' but that doesn't mean anything in the context of United Airlines.

Fortunately, *Cosine Similarity* can be used to compare different TF-IDF vectors of the same company. This way, 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[8]).

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

Figure 8

[8] 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.

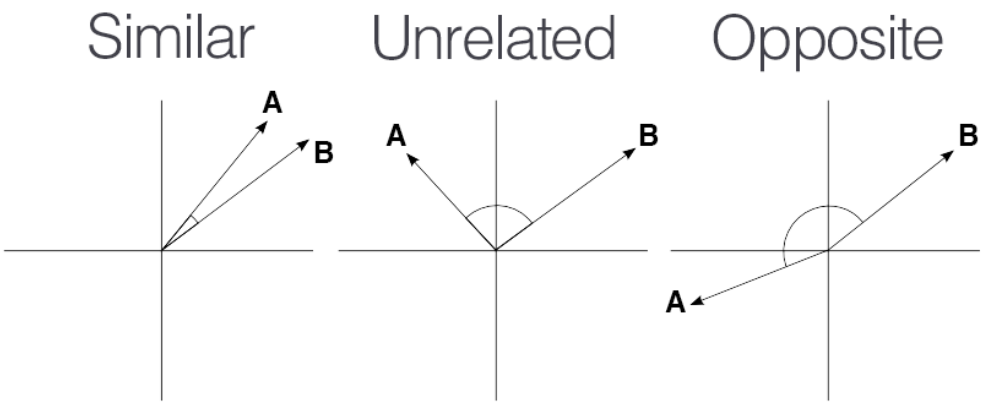

Figure 9 - Visualising cosine similarity, where cosine similarity = Cos(θ)

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 9*, a smaller angle between the two vectors translates to a high value for the cosine similarity.

LM - Loughran-McDonalds Sentiment Dictionary (LMSD)

The LMSD holds a lexicon of words and phrases that are specially trained for financial texts, each associated with different levels of sentiment intensity so it can identify industry jargon more accurately.

Word Complexity

This function uses the py-readability-metrics library to calculate the four text complexity indices, which include:

- FK - Flesch Kincaid Grade Level
- GF - Gunning Fog Index
- SMOG - Smog index
- FE - Flesch Reading Ease

These four indices vary in methodology but are all suited to financial contexts and offer unique benefits. For example, the Gunning Fog index is more suitable for shorter passages whereas the SMOG index is better for longer passages.

Feature Extraction Process

The feature extraction of all S&P500 stocks will take 28 to 40 hours since each ticker spans 3 to 5 minutes of run time, depending on the size and number of transcripts. Therefore, I divided the spy_tickers_list into batches, looped it, and then appended the features onto a text file called 'Feature_extracted_data.csv'.

Stage 3: Machine Learning

```
from numpy import loadtxt
file = open('Feature_extracted_data.csv', 'rb')
data = np.loadtxt(file, delimiter=",",dtype='str')
```

Figure 10

The programme reads the feature extracted data from the text file and stores it as a NumPy array.

```
Y_data = data[:,5]
```

Figure 11

After deleting rows consisting of 'NaN' values, it's separated into ***X_data*** and ***Y_data*** datasets, being the *independent* and *dependent* variables respectively. The ***Y_data*** dataset consists of the percentage price change between 2 days so it can be manually changed depending on what you want to look for (e.g. data[:,6], data[:,7] etc.).

```
scaler = MinMaxScaler()
X_data = scaler.fit_transform(X_data)
```

Figure 12

X_data values are normalised (scaled) between 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
- Prevents any feature from having too much influence on the model

```
X_train, X_test, Y_train, Y_test = train_test_split(X_data, Y_data, test_size=0.33)
```

Figure 13

The dataset is then split in a random order of 33% test data, and 67% training data. The training dataset is used to train the algorithm, while the testing dataset will validate the effectiveness of the algorithm via a scoring system.

Scoring System:

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

Figure 14

These 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)

Classification

```
df.loc[(df[0] >= 0.1)] = 1
```

Figure 15

First, ***Y_data*** is translated from floats to integers using the pandas' .loc function. As an example, the .loc function in *figure 15* swaps all values greater than 0.1 to a category labelled as '***1.0***'.

**note that The Y_values for the data below are according to the percentage change of stock prices between Day 0 and Day 50.*

```
rfc = RandomForestClassifier(max_depth=1000, n_estimators=1000)

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))
```

Figure 16

I used a Random Forest Regressor algorithm for all classification problems out of simplicity as my goal was to assess what classification method is most beneficial.

Multiclass Classification

After dozens of trial and error, I noticed that the ML model has low accuracy when:

1. Too many classifications (i.e. more than 3 different labels)
2. Too specific classification

```
df.loc[(df[0] >= 0.03)] = 1
df.loc[(df[0] < -0.03)] = -1
df.loc[((df[0] < 0.03) & (df[0] >= -0.03))] = 0
```

	precision	recall	f1-score	support
-1.0	0.35	0.06	0.10	1628
0.0	0.37	0.05	0.09	1672
1.0	0.46	0.93	0.61	2733
accuracy			0.45	6033
macro avg	0.39	0.34	0.26	6033
weighted avg	0.40	0.45	0.33	6033

Figure 17a, 17b[9]

```
df.loc[(df[0] >= 0.10)] = 1
df.loc[(df[0] < -0.10)] = -1
df.loc[((df[0] < 0.10) & (df[0] >= -0.10))] = 0
```

	precision	recall	f1-score	support
-1.0	0.88	0.01	0.02	596
0.0	0.74	1.00	0.85	4446
1.0	0.60	0.02	0.03	991
accuracy			0.74	6033
macro avg	0.74	0.34	0.30	6033
weighted avg	0.73	0.74	0.63	6033

[9] Note that 1.0, -1.0, and 0.0 are labels for each category

Figure 18a, 18b

The ML model has lower precision when classifying labels that are too specific, such as classifying 3% ≥ y > -3% (class 0.0) compared to classifying 10% ≥ y > -10% (*figure 17b vs figure 18b*). Although there's the possibility that the data are skewed due to the low recall value.

Binary Classification

```
df.loc[(df[0] >= 0.1)] = 1
df.loc[(df[0] < 0.1)] = 0
```

Figure 19a

Then, I tried to use a binary classification set instead of multiclass, I separated the Y-values with the intent of 'one vs rest':

- Class 1: < 10%
- Class 2: >= 10%

I wanted to obtain the tickers predicted to increase by more than 10%, whereas those performed lower than 10% are considered *'the rest'*.

I chose binary classification because:

- Sentiment and text complexity analysis are ill-suited to define rules for a large number of categories due to their relative ambiguity.
- Achieves higher accuracy scores in the limited dataset (18280 data after deducting NaN values)

	precision	recall	f1-score	support
0.0	0.83	1.00	0.91	4991
1.0	1.00	0.02	0.04	1042
accuracy			0.83	6033
macro avg	0.91	0.51	0.47	6033
weighted avg	0.86	0.83	0.76	6033

Figure 19b

	precision	recall	f1-score	support
0.0	0.83	1.00	0.91	5027
1.0	0.91	0.01	0.02	1006
accuracy			0.83	6033
macro avg	0.87	0.50	0.46	6033
weighted avg	0.85	0.83	0.76	6033

Figure 19c

	precision	recall	f1-score	support
0.0	0.84	1.00	0.91	5047
1.0	0.85	0.02	0.03	986
accuracy			0.84	6033
macro avg	0.84	0.51	0.47	6033
weighted avg	0.84	0.84	0.77	6033

Figure 19d

The results were better than anticipated as they achieved high precision for both classes. I ran three trials with different test/training splits (figure 19b, 19c, 19d). The average accuracy is 0.83˙, and the average precision for category '*1.0*' is 0.92. One caveat to this model is the low recall value which may be caused by an imbalance of data, it may be beneficial to widen the initial dataset by including stocks outside the S&P500 (which may triple/quadruple the initial 18280 datasets). Then group the percentage change in stock price evenly to reduce the problem of data imbalance. However as the dataset expands and the recall rate increases, the precision may lower.
For this 'one vs rest' model, a real-world strategy is to solely go Long[10] on those situated in '1.0' and ignore the '0.0' category, even with 2% recall, which is 21 stocks, in this case, it can obtain decent returns. To validify this model, the initial dataset must be expanded (by including stocks outside S&P500), although this may prove the model to be ineffective as the recall rate increases and precision decreases. Thus calling for expansion and improvement of feature extraction techniques.

Nonetheless, even if precision remains above 60% when the recall rate increases, this shows there is a correlation between the features and stock performances, thus calling for improvements.

Stage 4: Conclusions

There are several weaknesses this model face. Firstly, it doesn't have a large dataset (18280). Secondly, it has limited features for training (only various sentiment, prose, and text complexity) which means it doesn't consider other elements that may impact the price movement of a stock. Due to the limited and qualitative nature of existing features, it means the model cannot give a precise quantitative percentage estimation, thus explaining the difficulty with a regression model. Since the movement of share prices is not solely dependent on earnings calls and instead also relies on external information (press releases, financials, sentiment etc.), improvements must be made to ensure higher accuracies.

[10] Go Long means to buy a stock

The model may improve by including the ability to read financial statements and spot irregularities with past statements (e.g. spotting new footnotes, and new risk factors in 10K[11] and 10Qs). Another improvement is to include analyst rating[12] as well as the general sentiment towards a specific ticker (and its changes over time) by scraping content from Twitter, Reddit, and StockTwits. Technical analysis can also be applied (e.g. data on Volume, MACD[13], RSI[14]) by scraping off yahoo finance using the yFinance plugin, to gain higher accuracies for multi-class classification algorithms, potentially even enough information for regression models.

Nonetheless, at present, an effective strategy is to go long on shares predicted to overperform (labelled 1.0); and to short those predicted to underperform (by reversing the labels, i.e. switching '<' to '>=' for the binary classification model); as well as adding empathise on companies who operate in the same industry. Howbeit, it's prudent to expand on the current dataset as well as to 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 data I selected for classification measures the Day 0 to 50 percent change. This is unrealistic as stocks often move pre-market/after-hours almost immediately after the earnings are announced. This means we cannot capitalise on the day 0 bounce, since transcripts often take 1-2 days before being published, nonetheless, it's a good starting point.

[11] 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.
[12] Analyst ratings includes: sell, underperform, hold, outperform, buy. Most analysts issue ratings four times a year so we can spot any changes made and process it. In addition, a price target is accompanied by the release, which we can also use.
[13] MACD = Moving Average Convergence/Divergence. It provides information on the stock price momentum
[14] RSI = Relative Strength Index. Momentum indicator used to signal oversold/overbought status.

Bibliography:

Borcan, M. (2020). *TF-IDF Explained And Python Sklearn Implementation*. [online] Medium. Available at: https://towardsdatascience.com/tf-idf-explained-and-python-sklearn-implementation-b020c5e83275.

Dubiel, C. (2022). *Stock Price Predictions from Earnings Calls*. [online] GitHub. Available at: https://github.com/cdubiel08/Earnings-Calls-NLP [Accessed 25 Aug. 2022].

Edmiston, D. and Park, Z. (n.d.). *Unsupervised Discovery of Firm-Level Variables in Earnings Call Transcript Embeddings*. [online] Available at: https://aclanthology.org/2020.finnlp-1.6.pdf [Accessed 25 Aug. 2022].

Financial Modeling Prep. (n.d.). *Earnings Call Natural Language Processing Analysis*. [online] Available at: https://site.financialmodelingprep.com/how-to/how-to-use-natural-language-processing-to-analyze-earning-call-transcripts-with-python [Accessed 25 Aug. 2022].

Ganesan, K. (2019). *How to Use Tfidftransformer & Tfidfvectorizer - A Short Tutorial*. [online] Kavita Ganesan, PhD. Available at: https://kavita-ganesan.com/tfidftransformer-tfidfvectorizer-usage-differences/#.Yu_IEC8RqLc [Accessed 25 Aug. 2022].

Jha, V., Blaine, J. and Alphasense, W. (2015). *Finding Value in Earnings Transcripts Data with AlphaSense*. [online] Available at: https://extractalpha.com/wp-content/uploads/2015/06/Finding-Value-in-Earnings-Transcripts-Data-with-AlphaSense1.pdf.

John Watson Rooney (2020). *Python Web Scraping: JSON in SCRIPT tags*. *YouTube*. Available at: https://www.youtube.com/watch?v=QNLBBGWEQ3Q [Accessed 16 Jul. 2022].

JsonFormatter (n.d.). *Best JSON Formatter and JSON Validator: Online JSON Formatter*. [online] jsonformatter.org. Available at: https://jsonformatter.org/.

Labs, D.D. (2020). *Earnings Call Sentiment Analysis (Part I)*. [online] Medium. Available at: https://medium.com/@deephavendatalabs/earnings-call-sentiment-analysis-part-i-e3e7aafe2cab [Accessed 25 Aug. 2022].

Liang, D. (n.d.). *Predicting Stock Price Changes with Earnings Call Transcripts*. [online] cdr.lib.unc.edu. Available at: https://cdr.lib.unc.edu/concern/masters_papers/w9505408x [Accessed 25 Aug. 2022].

Ma, Z., Bang, G., Wang, C. and Liu, X. (n.d.). *Towards Earnings Call and Stock Price Movement*. [online] Available at: https://arxiv.org/pdf/2009.01317.pdf.

Natural Language Processing -Part I: Primer Unveiling the Hidden Information in Earnings Calls. (2017). [online] Available at: https://www.spglobal.com/marketintelligence/en/documents/sp-global-market-intelligence-nlp-primer-september-2018.pdf.

Natural Language Processing -Part II: Stock Selection Alpha Unscripted: The Message within the Message in Earnings Calls. (2018). [online] Available at: https://www.spglobal.com/marketintelligence/en/documents/mi-research-qr-nlp-part-ii-180912-new.pdf [Accessed 25 Aug. 2022].

Natural Language Processing -Part III: Feature Engineering. (2020). [online] Available at: https://www.spglobal.com/marketintelligence/en/documents/nlp-iii-final-013020-10a.pdf.

netpeaksoftware.com. (n.d.). *What Is TF-IDF, and Why Does It Matter for SEO?* [online] Available at: https://netpeaksoftware.com/blog/what-is-tf-idf-and-why-does-it-matter-for-seo [Accessed 25 Aug. 2022].

Ng, A. (2019). *Machine Learning | Coursera*. [online] Coursera. Available at: https://www.coursera.org/learn/machine-learning.

Nicholson, C. (n.d.). *A Beginner's Guide to Neural Networks and Deep Learning*. [online] Pathmind. Available at: https://wiki.pathmind.com/neural-network.

OpenGenus IQ: Computing Expertise & Legacy. (2019). *Find similarity between documents using TF IDF*. [online] Available at: https://iq.opengenus.org/document-similarity-tf-idf/. [Accessed 25 Aug. 2022].

ResearchGate. (n.d.). *What is the best metric (precision, recall, f1, and accuracy) to evaluate the machine learning model for imbalanced data?* [online] Available at:

https://www.researchgate.net/post/What_is_the_best_metric_precision_recall_f1_and_accuracy_to_evaluate_the_machine_learning_model_for_imbalanced_data.

roic.ai. (n.d.). *Roic.ai · your first step in analyzing a company*. [online] Available at: https://roic.ai [Accessed 25 Aug. 2022].

Wikipedia Contributors (2019). *Cosine similarity*. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Cosine_similarity.

www.youtube.com. (n.d.). *TF-IDF in Python with Scikit Learn (Topic Modeling for DH 02.03)*. [online] Available at: https://www.youtube.com/watch?v=i74DVqMsRWY [Accessed 25 Aug. 2022].

www.youtube.com. (n.d.). *Web Scraping with Python - Beautiful Soup Crash Course*. [online] Available at: https://www.youtube.com/watch?v=XVv6mJpFOb0&t=2039s [Accessed 25 Aug. 2022].

www3.cs.stonybrook.edu. (n.d.). *Lydia/TextMap Publications*. [online] Available at: https://www3.cs.stonybrook.edu/~skiena/lydia/.