

# Sentiment Analysis on publicly traded companies

J016 - Avneesh Dubey  
J054 - Aayush Talekar





# Table of Contents

**1**

**What is Sentiment  
Analysis**

**2**

**Lexicon Approach & ML  
Approach**

**3**

**The SEC**

**4**

**Initial Project**

**5**

**Document-Term  
Matrix**

**6**

**finBERT**

**7**

**Final Project**





# What is Sentiment Analysis?

- Sentiment analysis is the use of natural language processing, text analysis, computational linguistics, and biometrics to systematically identify, extract, quantify, and study affective states and subjective information.
- Sentiment analysis (in Finance) involves quantifying and exploiting ‘sentiment’/‘emotions’ for investment.
- ‘Sentiment’ can include (but is not limited to) positivity, negativity, uncertainty, narcissism, anxiety, panic, etc.
- There are broadly 2 ways of estimating ‘sentiment’:
  1. Lexicon / Dictionary based approach,
  2. “Machine learning” approach

# Lexicon / Dictionary Approach

- In the lexicon / dictionary based approach, we start with a ‘prior’ on what constitutes words relating to ‘sentiment’ (e.g.,
- ‘positivity’, ‘negativity’, etc).
- We use a “sentiment language” ( $\psi$ ).
- We then estimate ‘sentiment’ ( $\phi$ ) as a function of the (cleaned) words ( $\mathcal{W}$ ) in a given document ( $d$ ) which belong to a sentiment language ( $\psi$ ).
- As far as estimating sentiment in Finance goes, the literature largely relies on lexicon / dictionary based approach.

# Machine Learning Approach

- In the “Machine learning” approach, we start with a subsample of the ‘corpus’  $\mathcal{C}$  which displays ‘sentiment’ ( $\phi$ ).
- E.g., A range of movie reviews ‘labelled’ as either ‘positive’ or ‘negative’.
- We then apply ‘machine learning’ (e.g. classification) algorithms to categorise / classify other sample text on its level of ‘sentiment’ ( $\phi$ ).
- Applying this approach in Finance is problematic though, because we don’t have data with sentiment ‘labels’.
- E.g. there’s no database of companies that are
- already classified by ‘sentiment’

# The SEC

- The **U.S. Securities and Exchange Commission (SEC)** is a large independent agency of the United States federal government that was created following the stock market crash in the 1930s to protect investors and the national banking system.
- The primary purpose of the SEC is to enforce the law against market manipulation.
- The SEC has a three-part mission: to protect investors; maintain fair, orderly, and efficient markets; and facilitate capital formation.<sup>[5]</sup>
- To achieve its mandate, the SEC enforces the statutory requirement that **public companies** and other regulated companies submit quarterly and **annual reports**, as well as other periodic reports.
- In addition to annual **financial reports**, company executives must provide a narrative account, called the "**management discussion and analysis**" (MD&A), that outlines the previous year of operations and explains how the company fared in that time period.
- MD&A will usually also touch on the upcoming year, outlining future goals and approaches to new projects.
- In an attempt to level the playing field for all investors, the SEC maintains an online database called **EDGAR** (the Electronic Data Gathering, Analysis, and Retrieval system) **online** from which investors can access this and other information filed with the agency.





# The SEC's EDGAR Database

- Quarterly and semiannual reports from public companies are crucial for investors to make sound decisions when investing in the capital markets.
- Unlike banking, investment in the capital markets is not guaranteed by the federal government.
- The potential for big gains needs to be weighed against that of sizable losses.
- Mandatory disclosure of financial and other information about the issuer and the security itself gives private individuals as well as large institutions the same basic facts about the public companies they invest in, thereby increasing public scrutiny while reducing insider trading and fraud.
- The SEC makes reports available to the public through the EDGAR system. The SEC also offers publications on investment-related topics for public education.
- The same online system also takes tips and complaints from investors to help the SEC track down violators of the securities laws.
- The SEC adheres to a strict policy of never commenting on the existence or status of an ongoing investigation.





# SEC 10-K Filings:

- A 10-K is a comprehensive report filed annually by a publicly-traded company about its financial performance and is required by the U.S. Securities and Exchange Commission (SEC).
- The report contains much more detail than a company's annual report, which is sent to its shareholders before an annual meeting to elect company directors.
- Some of the information a company is required to document in the 10-K includes its history, organizational structure, financial statements, earnings per share, subsidiaries, executive compensation, and any other relevant data.





## More on 10-K filings:

Often, the most essential components of the annual 10-K filing include:

- Item 1: Business (a description of the company's operation)
- Item 1A: Risk Factors
- Item 3: Legal Proceedings
- Item 6: Selected Financial Data
- Item 7: Management's Discussion and Analysis of the Financial Condition



# The Components we focus on:

1

**Item 1A - Risk factors.** These outline any and all risks the company faces or may face in the future. The risks are typically listed in order of importance.

2

**Item 7 - Management's discussion and analysis** of financial condition and results of operations. Also known as MD&A, this gives the company an opportunity to explain its business results from the previous fiscal year. This section is where the company can tell its story in its own words.

# Initial Project:



NLP Attempt.ipynb - Colaboratory

colab.research.google.com/drive/13cUqDG0gNdJ3VfDGRxc5fVJ\_bN9wjrf#scrollTo=2BbA1JrjSWd

NLP Attempt.ipynb

File Edit View Insert Runtime Tools Help Cannot save changes

Table of contents

- EDGAR
- New Section
- Section

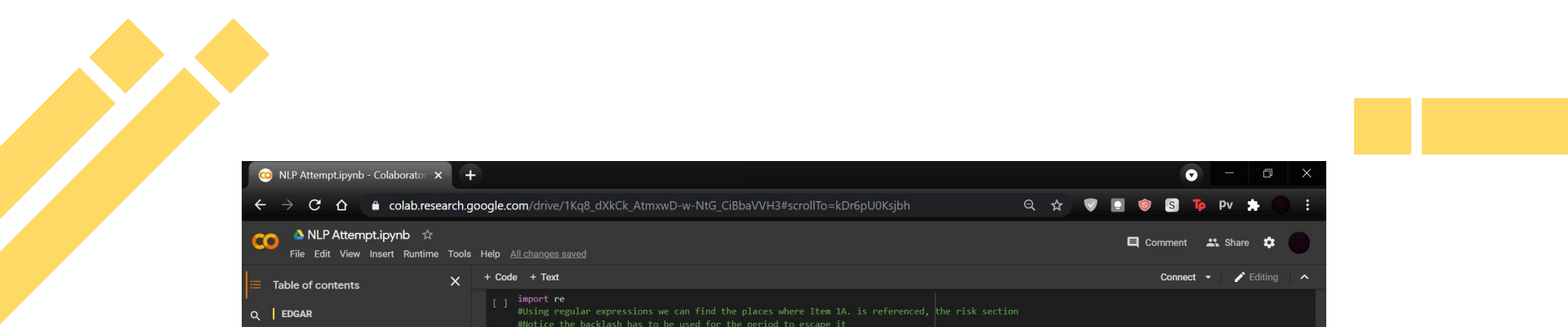
EDGAR

```
pip install edgar
```

```
Collecting edgar
  Downloading https://files.pythonhosted.org/packages/0f/55/89defc71453aa05a3b4beb2a251de54fd4438752d5f9c8022d35d02f3025/edgar-5.4.1-py3-none-any.whl
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from edgar) (4.41.1)
Collecting fuzzywuzzy[speedup]
  Downloading https://files.pythonhosted.org/packages/43/ff/74f23998ad2f93b945c0309f875be92e04e0348e062026998b5eeef4c33/fuzzywuzzy-0.18.0-py2.py3-none-any.whl
Requirement already satisfied: lxml in /usr/local/lib/python3.7/dist-packages (from edgar) (4.2.6)
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from edgar) (2.23.0)
Collecting python-Levenshtein>=0.12; extra == "speedup"
  Downloading https://files.pythonhosted.org/packages/2a/dc/97f2b63ef0fa1fd78dcb7195aca577804f6b2b51e712516cc0e902a9a201/python-Levenshtein-0.12.2.tar.gz
Requirement already satisfied: idna<3, >=2.5 in /usr/local/lib/python3.7/dist-packages (from requests->edgar) (2.10)
Requirement already satisfied: certifi>2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests->edgar) (2020.12.5)
Requirement already satisfied: urllib3<1.25.0, >=1.25.1, <1.26, >=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests->edgar) (1.24.3)
Requirement already satisfied: chardet<4, >=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests->edgar) (3.0.4)
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (from python-Levenshtein>=0.12; extra == "speedup") (51.1.0)
Building wheels for collected packages: python-Levenshtein
  Building wheel for python-Levenshtein (setup.py) ... done
  Created wheel for python-Levenshtein: filename=python-Levenshtein-0.12.2-cp37-cp37m-linux_x86_64.whl size=149811 sha256=9f8f7ab9e3fabb8f67c796a944465a7
  Stored in directory: /root/.cache/pip/wheels/b3/26/73/4b48503bac73f01cf18e52cd250947049a7f339e940c5df8fc
Successfully built python-Levenshtein
Installing collected packages: python-Levenshtein, fuzzywuzzy, edgar
Successfully installed edgar-5.4.1 fuzzywuzzy-0.18.0 python-Levenshtein-0.12.2
```

```
[2] import nltk
```

5s completed at 7:29 AM



NLP Attempt.ipynb - Colaborator x

colab.research.google.com/drive/1Kq8\_dXkCk\_AtmxwD-w-NtG\_CiBbaVVH3#scrollTo=kDr6pU0Ksjbh

NLP Attempt.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

Table of contents

- EDGAR
  - New Section
  - Section

+ Code + Text

```
[ ] import re
#Using regular expressions we can find the places where Item 1A. is referenced, the risk section
#Notice the backslash has to be used for the period to escape it
#There are two places we find it, the table of contents and the header
print(re.findall(re.compile('Item 1A\.'),sample_text))

['Item 1A.']
```

#A better way is to search and find the elements. This will give location as well as matched string

```
for match in re.finditer('Item 1A\.', sample_text):
    print(match)

<re.Match object; span=(26459, 26467), match='Item 1A.'>
```

#We can grab the span as so

```
print(match.span())

(26459, 26467)
```

#We can also generalize the regular expression to all items

```
maches = list(re.finditer(re.compile('Item [0-9][A-Z]*\.', sample_text))
print(maches)

[<re.Match object; span=(26459, 26467), match='Item 1A.'>, <re.Match object; span=(26480, 26488), match='Item 1B.'>, <re.Match object; span=(26515, 26522), match='Item 1C.'>]
```

#The first index of the matches will be the actual string

```
print(matches[1][0])

Item 1B.
```



# Where we faced issues.

The screenshot shows a Google Colab notebook titled "NLP Attempt.ipynb". The left sidebar contains a "Table of contents" with sections for "EDGAR", "New Section", and "Section". The main code area shows a function `get_threat_level_of_word` that takes `company_name`, `company_id`, and `word` as arguments. A `ValueError` is raised with the message "max() arg is an empty sequence". The error traceback shows the function call and the `pull_risk_section` function. Below the error, there is a code block that loads several CSV files from GitHub using `pd.read_csv`. The files are: `NLP/main/Negative.csv`, `NLP/main/Positive.csv`, `NLP/main/Constraining.csv`, `NLP/main/Uncertainty.csv`, `NLP/main/Weak%20Modal.csv`, `NLP/main/Strong%20Modal.csv`, and `NLP/main/Litigious.csv`. The code also defines columns for each category: `negative.columns = ['Word']`, `positive.columns = ['Word']`, `constraining.columns = ['Word']`, `uncertainty.columns = ['Word']`, and `weakmodal.columns = ['Word']`.

```
[ ] get_threat_level_of_word(company_name="Oracle Corp", company_id="0001341439", word='loss')

-----
ValueError                                Traceback (most recent call last)
<ipython-input-43-80b4aad5143> in <module>()
----> 1 get_threat_level_of_word(company_name="Oracle Corp", company_id="0001341439", word='loss')

-----
2 frames
<ipython-input-19-75957012d47b> in pull_risk_section(text)
      4 text = re.sub('\xa0+', ' ', text)
      5 matches = list(re.finditer(re.compile('Item [0-9][A-Z]*\.', text)))
----> 6 start = max([i for i in range(len(matches)) if matches[i][0] == 'Item 1B.'])
      7 end = start+1
      8 start = matches[start].span()[1]

ValueError: max() arg is an empty sequence

SEARCH STACK OVERFLOW

[ ] negative = pd.read_csv('https://raw.githubusercontent.com/AayushTalekar/NLP/main/Negative.csv', header=None)
positive = pd.read_csv('https://raw.githubusercontent.com/AayushTalekar/NLP/main/Positive.csv', header=None)
constraining = pd.read_csv('https://raw.githubusercontent.com/AayushTalekar/NLP/main/Constraining.csv', header=None)
uncertainty = pd.read_csv('https://raw.githubusercontent.com/AayushTalekar/NLP/main/Uncertainty.csv', header=None)
weakmodal = pd.read_csv('https://raw.githubusercontent.com/AayushTalekar/NLP/main/Weak%20Modal.csv', header=None)
strongmodal = pd.read_csv('https://raw.githubusercontent.com/AayushTalekar/NLP/main/Strong%20Modal.csv', header=None)
litigious = pd.read_csv('https://raw.githubusercontent.com/AayushTalekar/NLP/main/Litigious.csv', header=None)

[ ] negative.columns = ['Word']
positive.columns = ['Word']
constraining.columns = ['Word']
uncertainty.columns = ['Word']
weakmodal.columns = ['Word']
```

Link:

[https://colab.research.google.com/drive/1Kq8\\_dXkCk\\_AtmxwD-w-NtG\\_CiBbaVVH3?usp=sharing](https://colab.research.google.com/drive/1Kq8_dXkCk_AtmxwD-w-NtG_CiBbaVVH3?usp=sharing)



# Constructing a Document-Term Matrix via Sklearn, NLTK

- A Document-Term Matrix is used as a starting point for a number of NLP tasks. This short write up shows how to use Sklearn and NLTK python libraries to construct frequency and binary versions.

# Constructing a Document-Term Matrix via Sklearn, NLTK

- Importing Libraries:

```
import re
```

```
import pandas as pdfrom nltk import word_tokenize
```

```
from nltk.corpus import stopwords
```

```
from nltk.tokenize import word_tokenize
```

```
from nltk.stem.porter import PorterStemmerfrom sklearn.feature_extraction.text import  
CountVectorizer
```

# Constructing a Document-Term Matrix via Sklearn, NLTK

- Importing Libraries:

```
import re
```

```
import pandas as pdfrom nltk import word_tokenize
```

```
from nltk.corpus import stopwords
```

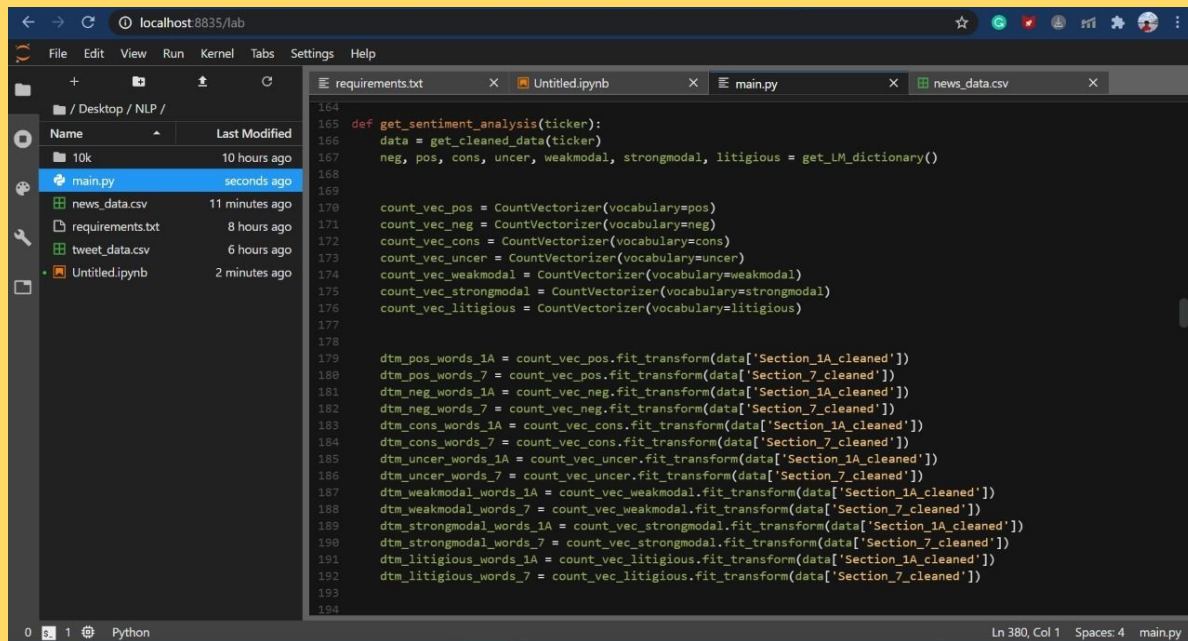
```
from nltk.tokenize import word_tokenize
```

```
from nltk.stem.porter import PorterStemmerfrom sklearn.feature_extraction.text import  
CountVectorizer
```



# Constructing a Document-Term Matrix via Sklearn, NLTK

- Setting up Tokenizer:



The screenshot shows a JupyterLab environment with a file browser on the left and a code editor on the right. The file browser displays a directory structure with files like `news_data.csv`, `requirements.txt`, `tweet_data.csv`, and `Untitled.ipynb`. The code editor shows a Python script with the following content:

```
164
165 def get_sentiment_analysis(ticker):
166     data = get_cleaned_data(ticker)
167     neg, pos, cons, uncer, weakmodal, strongmodal, litigious = get_lm_dictionary()
168
169
170     count_vec_pos = CountVectorizer(vocabulary=pos)
171     count_vec_neg = CountVectorizer(vocabulary=neg)
172     count_vec_cons = CountVectorizer(vocabulary=cons)
173     count_vec_uncer = CountVectorizer(vocabulary=uncer)
174     count_vec_weakmodal = CountVectorizer(vocabulary=weakmodal)
175     count_vec_strongmodal = CountVectorizer(vocabulary=strongmodal)
176     count_vec_litigious = CountVectorizer(vocabulary=litigious)
177
178
179     dtm_pos_words_1A = count_vec_pos.fit_transform(data['Section_1A_cleaned'])
180     dtm_pos_words_7 = count_vec_pos.fit_transform(data['Section_7_cleaned'])
181     dtm_neg_words_1A = count_vec_neg.fit_transform(data['Section_1A_cleaned'])
182     dtm_neg_words_7 = count_vec_neg.fit_transform(data['Section_7_cleaned'])
183     dtm_cons_words_1A = count_vec_cons.fit_transform(data['Section_1A_cleaned'])
184     dtm_cons_words_7 = count_vec_cons.fit_transform(data['Section_7_cleaned'])
185     dtm_uncer_words_1A = count_vec_uncer.fit_transform(data['Section_1A_cleaned'])
186     dtm_uncer_words_7 = count_vec_uncer.fit_transform(data['Section_7_cleaned'])
187     dtm_weakmodal_words_1A = count_vec_weakmodal.fit_transform(data['Section_1A_cleaned'])
188     dtm_weakmodal_words_7 = count_vec_weakmodal.fit_transform(data['Section_7_cleaned'])
189     dtm_strongmodal_words_1A = count_vec_strongmodal.fit_transform(data['Section_1A_cleaned'])
190     dtm_strongmodal_words_7 = count_vec_strongmodal.fit_transform(data['Section_7_cleaned'])
191     dtm_litigious_words_1A = count_vec_litigious.fit_transform(data['Section_1A_cleaned'])
192     dtm_litigious_words_7 = count_vec_litigious.fit_transform(data['Section_7_cleaned'])
193
194
```

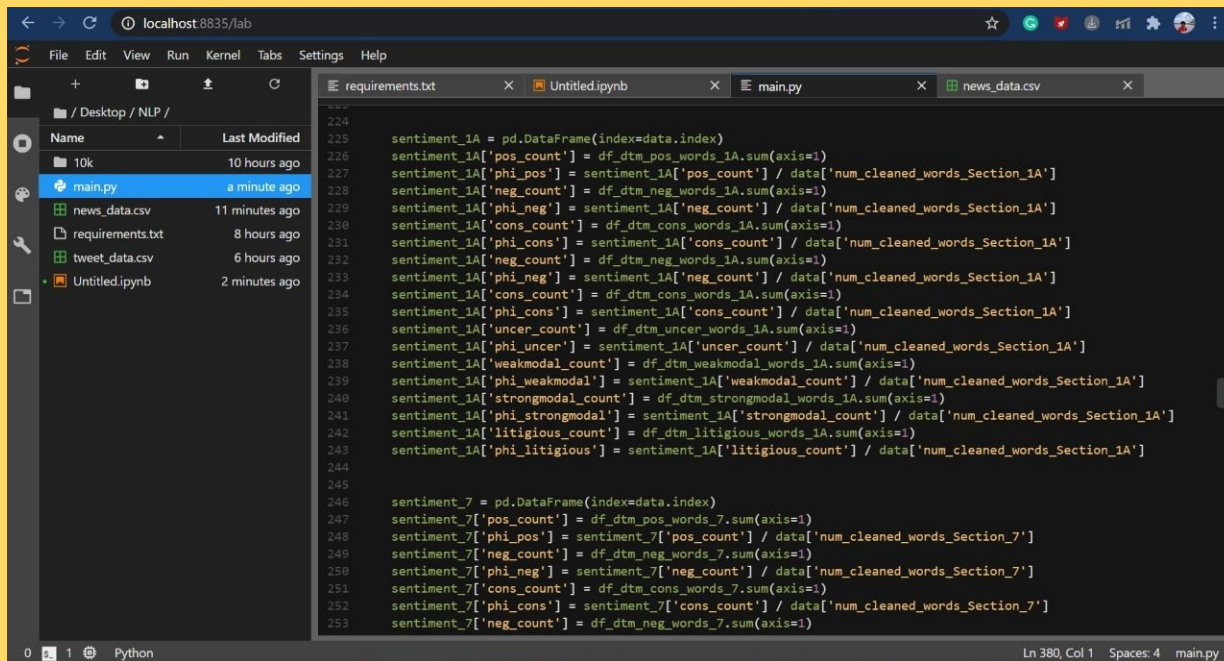


# Constructing a Document-Term Matrix via Sklearn, NLTK

- Document-Term Frequency Matrix:

```
194 df_dtm_pos_words_1A = pd.DataFrame(dtm_pos_words_1A.toarray(), index=data.index)
195 df_dtm_pos_words_1A.columns = count_vec_pos.vocabulary_.keys()
196 df_dtm_pos_words_7 = pd.DataFrame(dtm_pos_words_7.toarray(), index=data.index)
197 df_dtm_pos_words_7.columns = count_vec_pos.vocabulary_.keys()
198 df_dtm_neg_words_1A = pd.DataFrame(dtm_neg_words_1A.toarray(), index=data.index)
199 df_dtm_neg_words_1A.columns = count_vec_neg.vocabulary_.keys()
200 df_dtm_neg_words_7 = pd.DataFrame(dtm_neg_words_7.toarray(), index=data.index)
201 df_dtm_neg_words_7.columns = count_vec_neg.vocabulary_.keys()
202 df_dtm_cons_words_1A = pd.DataFrame(dtm_cons_words_1A.toarray(), index=data.index)
203 df_dtm_cons_words_1A.columns = count_vec_cons.vocabulary_.keys()
204 df_dtm_cons_words_7 = pd.DataFrame(dtm_cons_words_7.toarray(), index=data.index)
205 df_dtm_cons_words_7.columns = count_vec_cons.vocabulary_.keys()
206 df_dtm_uncer_words_1A = pd.DataFrame(dtm_uncer_words_1A.toarray(), index=data.index)
207 df_dtm_uncer_words_1A.columns = count_vec_uncer.vocabulary_.keys()
208 df_dtm_uncer_words_7 = pd.DataFrame(dtm_uncer_words_7.toarray(), index=data.index)
209 df_dtm_uncer_words_7.columns = count_vec_uncer.vocabulary_.keys()
210 df_dtm_weakmodal_words_1A = pd.DataFrame(dtm_weakmodal_words_1A.toarray(), index=data.index)
211 df_dtm_weakmodal_words_1A.columns = count_vec_weakmodal.vocabulary_.keys()
212 df_dtm_weakmodal_words_7 = pd.DataFrame(dtm_weakmodal_words_7.toarray(), index=data.index)
213 df_dtm_weakmodal_words_7.columns = count_vec_weakmodal.vocabulary_.keys()
214 df_dtm_strongmodal_words_1A = pd.DataFrame(dtm_strongmodal_words_1A.toarray(), index=data.index)
215 df_dtm_strongmodal_words_1A.columns = count_vec_strongmodal.vocabulary_.keys()
216 df_dtm_strongmodal_words_7 = pd.DataFrame(dtm_strongmodal_words_7.toarray(), index=data.index)
217 df_dtm_strongmodal_words_7.columns = count_vec_strongmodal.vocabulary_.keys()
218 df_dtm_litigious_words_1A = pd.DataFrame(dtm_litigious_words_1A.toarray(), index=data.index)
219 df_dtm_litigious_words_1A.columns = count_vec_litigious.vocabulary_.keys()
220 df_dtm_litigious_words_7 = pd.DataFrame(dtm_litigious_words_7.toarray(), index=data.index)
221 df_dtm_litigious_words_7.columns = count_vec_litigious.vocabulary_.keys()
222
223
224
```

# Constructing a Document-Term Matrix via Sklearn, NLTK



```
224
225 sentiment_1A = pd.DataFrame(index=data.index)
226 sentiment_1A['pos_count'] = df_dtm_pos_words_1A.sum(axis=1)
227 sentiment_1A['phi_pos'] = sentiment_1A['pos_count'] / data['num_cleaned_words_Section_1A']
228 sentiment_1A['neg_count'] = df_dtm_neg_words_1A.sum(axis=1)
229 sentiment_1A['phi_neg'] = sentiment_1A['neg_count'] / data['num_cleaned_words_Section_1A']
230 sentiment_1A['cons_count'] = df_dtm_cons_words_1A.sum(axis=1)
231 sentiment_1A['phi_cons'] = sentiment_1A['cons_count'] / data['num_cleaned_words_Section_1A']
232 sentiment_1A['neg_count'] = df_dtm_neg_words_1A.sum(axis=1)
233 sentiment_1A['phi_neg'] = sentiment_1A['neg_count'] / data['num_cleaned_words_Section_1A']
234 sentiment_1A['cons_count'] = df_dtm_cons_words_1A.sum(axis=1)
235 sentiment_1A['phi_cons'] = sentiment_1A['cons_count'] / data['num_cleaned_words_Section_1A']
236 sentiment_1A['uncer_count'] = df_dtm_uncer_words_1A.sum(axis=1)
237 sentiment_1A['phi_uncer'] = sentiment_1A['uncer_count'] / data['num_cleaned_words_Section_1A']
238 sentiment_1A['weakmodal_count'] = df_dtm_weakmodal_words_1A.sum(axis=1)
239 sentiment_1A['phi_weakmodal'] = sentiment_1A['weakmodal_count'] / data['num_cleaned_words_Section_1A']
240 sentiment_1A['strongmodal_count'] = df_dtm_strongmodal_words_1A.sum(axis=1)
241 sentiment_1A['phi_strongmodal'] = sentiment_1A['strongmodal_count'] / data['num_cleaned_words_Section_1A']
242 sentiment_1A['litigious_count'] = df_dtm_litigious_words_1A.sum(axis=1)
243 sentiment_1A['phi_litigious'] = sentiment_1A['litigious_count'] / data['num_cleaned_words_Section_1A']
244
245
246 sentiment_7 = pd.DataFrame(index=data.index)
247 sentiment_7['pos_count'] = df_dtm_pos_words_7.sum(axis=1)
248 sentiment_7['phi_pos'] = sentiment_7['pos_count'] / data['num_cleaned_words_Section_7']
249 sentiment_7['neg_count'] = df_dtm_neg_words_7.sum(axis=1)
250 sentiment_7['phi_neg'] = sentiment_7['neg_count'] / data['num_cleaned_words_Section_7']
251 sentiment_7['cons_count'] = df_dtm_cons_words_7.sum(axis=1)
252 sentiment_7['phi_cons'] = sentiment_7['cons_count'] / data['num_cleaned_words_Section_7']
253 sentiment_7['neg_count'] = df_dtm_neg_words_7.sum(axis=1)
```

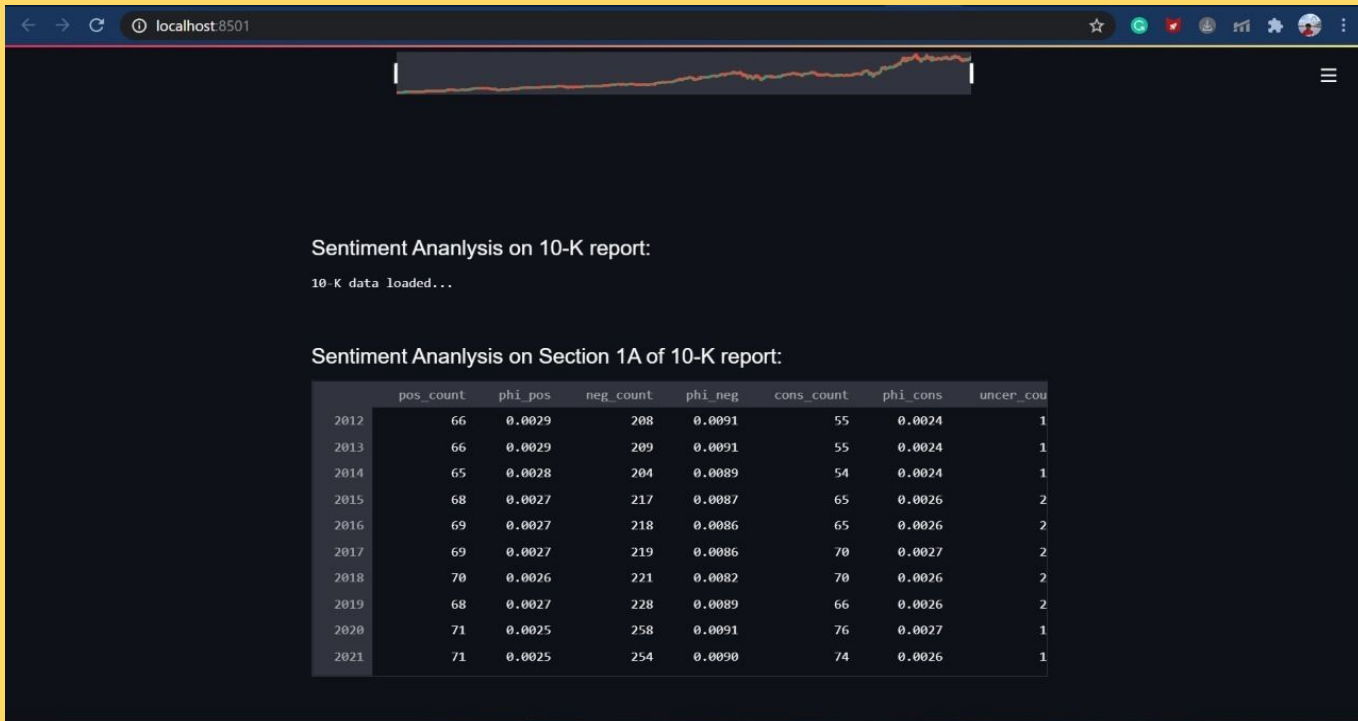


# Sentiment Analysis using finBERT

- BERT was the language model that made the whole concept of pre-training and fine-tuning flow very popular.
- BERT brought two core innovations to language modelling: (1) It borrowed the **transformer** (T of BERT) architecture from machine translation, which does a better job of modelling long-term dependencies than RNN-based ones (excellent overview here). (2) It introduced the Masked Language Modelling (MLM) task, where a random 15% of all tokens are masked and the model predicts them, enabling true **bi-directionality** (B of BERT).
- BERT was perfect for our task of financial sentiment analysis. Even with a very small dataset, it was now possible to take advantage of state-of-the-art NLP models. But since our domain — finance is very different from the general purpose corpus BERT was trained on, we wanted to add one more step before going for sentiment analysis. Pre-trained BERT knew how to talk, but now it was time to teach it how to *talk like a trader*. We took the pre-trained BERT and then further trained it on a purely financial corpus.



# Final Project:



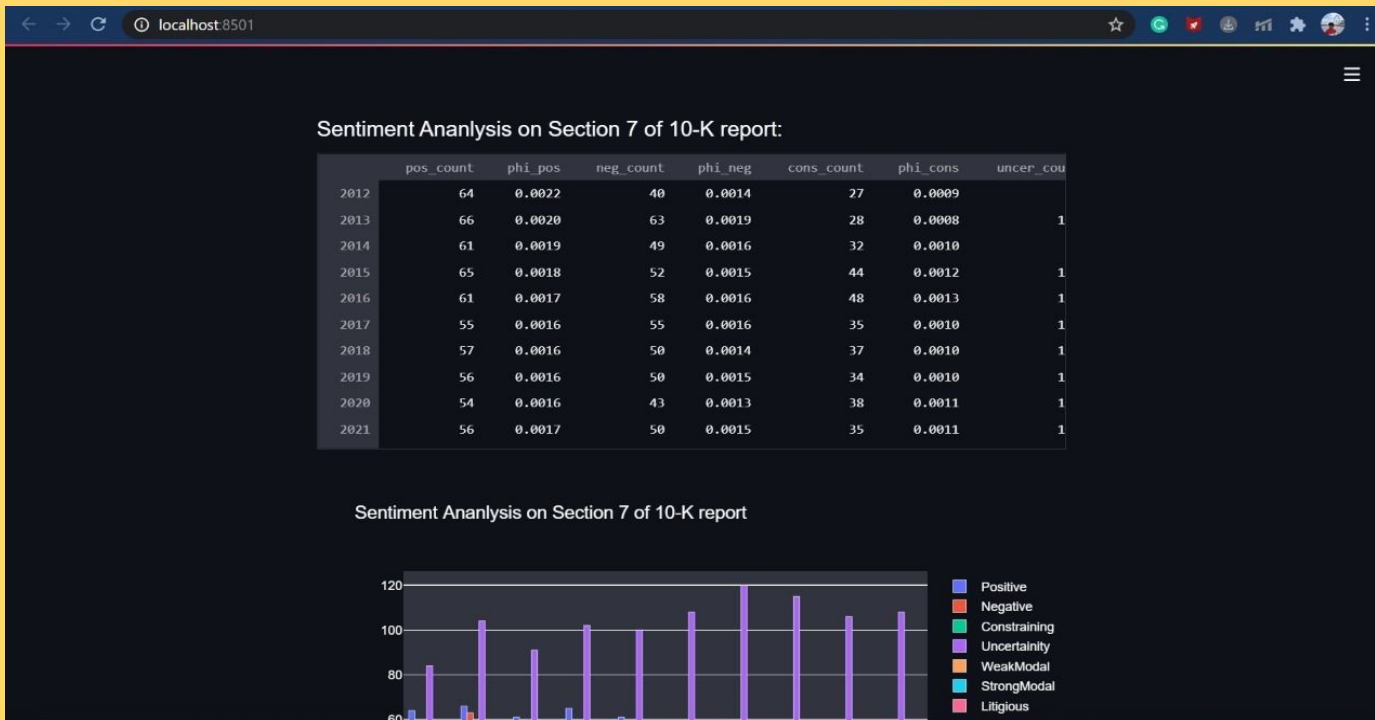


# Final Project:



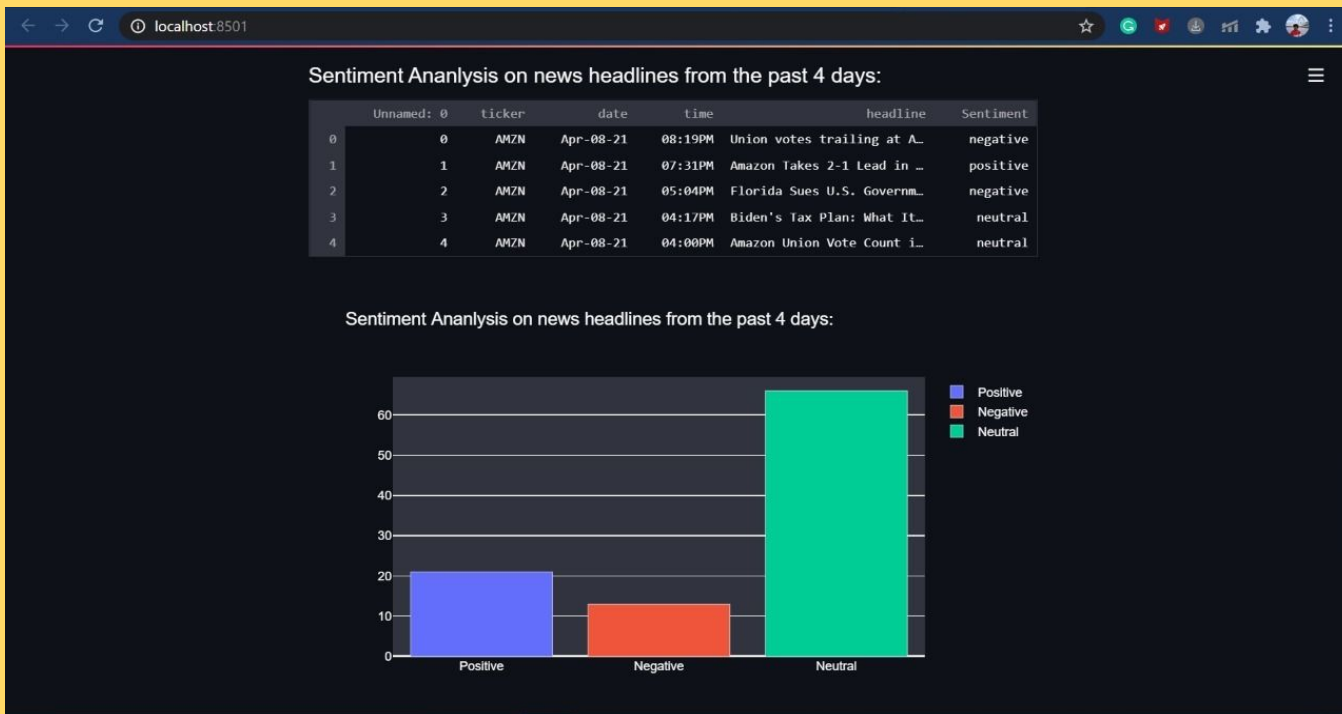


# Final Project:





# Final Project:







**Thank You.**