PROJECT REPORT

An implementation of the approach as described in the paper **"What You Say and How You Say It Matters: Predicting Financial Risk Using Verbal and Vocal Cues"** by Yu Qin and Yi Yang.

Jagori Bandyopadhyay
Dept. of Metallurgical and Materials Engineering
Indian Institute of Technology (IIT), Kharagpur

Harshvardhan
Dept. of Metallurgical and Materials Engineering
Indian Institute of Technology (IIT), Kharagpur

TABLE OF CONTENTS

OVERVIEW

Financial risk prediction is a crucial task within the financial market. Previous studies have indicated that the textual information present in a company's financial statements can be utilized to forecast the risk level associated with its stock. In today's context, CEOs not only convey information verbally through press releases and financial reports but also non-verbally during investor meetings and earnings conference calls. Anecdotal evidence suggests that the **CEO's vocal characteristics**, including emotions and voice tones, can provide insights into the company's performance. However, the extent to which vocal features can be employed to predict risk levels remains unknown. To address this gap, we have used **audio recordings** and **textual transcripts** of earnings calls for **S&P 500 companies**. The approach involves proposing a **multimodal deep regression model (MDRM)** that simultaneously models the CEO's verbal information (from text) and vocal information (from audio) during conference calls. Empirical findings demonstrate that our model, which incorporates both verbal and vocal features, leads to a significant and substantial reduction in prediction errors.

DATA DESCRIPTION

The Dataset contains folders for different companies. The folders are titled by company codes and the date of release of the Earning's call conference. Each folder contains the transcript file and the sentence-wise audio features data frame. Both this information will be fed as input to the model.

The label values are the stock price volatility values that had to be collected for the specified time frame. The stock volatility prediction problem is formulated following (Kogan et al., 2009). The volatility is defined as

$$v_{[t-\tau,t]} = \ln\left(\sqrt{\frac{\sum_{i=0}^{\tau}(r_{t-i} - \bar{r})^2}{\tau}}\right)$$

where rt is the return price at day t and r¯ is the mean of the return price over the period of day t − τ to day t.
We choose τ value as **30 calendar days** to evaluate the effectiveness of volatility prediction

DATA COLLECTION

Stock Price Data

We already had the data for Audio and Transcript (link). Of these, 290+ Companies were chosen and Historical stock data was extracted from **Yahoo Finance** using Python's **pandas_datareader** library. Here is one use case:

```
from pandas_datareader import data as pdr
import matplotlib.pyplot as plt

# initializing Parameters
start = "2017-01-01"
end = "2017-03-31"
symbols = ["AMZN","TWTR"]

# Getting the data
data = pdr.get_data_yahoo(symbols, start, end)

# # Display
# plt.figure(figsize = (20,10))
# plt.title('Opening Prices from {} to {}'.format(start, end))
# plt.plot(data['Open'])
# plt.show()
```

The data was collected and organized into **CSV files** containing the stocks' closing prices for the period mentioned. This data was further divided into train and test datasets. Following is a snippet of the Stock data.

DQ	DR	DS	DT	DU	DV	DW	DX	DY	DZ	EA	EB	EC	ED	EE	EF	EG	EH
lose	Close	Close	Close	Close	Close	Close	Close	Close	Close	Close	Close	Close	Close	Close	Close	Close	Close
WKS	TFX	WRLD	XYL	AXL	CBOE	FE	LPNT	MCO	NSP	PEG	TDS	VFC	CEVA	CHUY	CMCSA	FN	L
92.25	122.96	84.62	37.02	23.96	55.51	36.25	71.44	107.22	26.875	41.96	28.81	67.84369	18.45	22.15	29.39	18.68	41.67
98.3	122.61	86.54	37.14	24.41	55.53	36.65	72.76	108.4	26.95	42.69	29.26	67.09982	18.66	26.06	28.99	18.57	41.27
99.11	124.61	90.61	36.9	24.01	55.5	35.31	70.97	108.24	25.835	41.56	28.97	67.3258	19.01	25.5	28.83	18.66	41.09
94.63	123.04	91.04	36.64	23.88	55.82	34.83	69.7	106.63	26.06	41.74	28.84	66.63842	19.03	25.53	28.96	18.74	41.22
94.57	122.71	92.42	36.53	24.51	56.84	34.75	71.875	107.01	26.525	41.96	28.9	66.82674	19.5	25.24	29.205	18.55	41.42
97.07	122.78	93.05	36.69	24.46	57.27	35.09	72.53	107.88	26.68	42.06	29.28	67.70245	19.9	25.81	28.885	18.41	41.15
97.71	124.3	95.1	36.9	24.4	58.09	34.94	73.09	106.7	26.66	42.05	29.26	67.6177	20.01	25.75	28.665	18.57	40.95
97.16	125.27	94.46	36.81	24.28	58.77	34.86	71.86	106.49	26.435	41.89	29.78	67.1469	20.02	25.88	28.14	18.41	40.57
96.79	126.13	94.41	36.49	24.64	58.85	34.54	72.85	107.2	26.41	41.68	29.86	66.14877	19.84	26.75	28.27	18.55	40.9
96.65	126.6	93.96	36.79	24.84	59.4	34.95	73.26	108.73	26.65	42.28	29.71	66.45951	19.77	26	28.32	18.28	40.64
98.48	129.27	89.24	37.09	24.92	58.69	35.24	73.18	109.98	26.515	42.54	29.99	67.09982	20.25	26.92	28.485	18.6	41.05
97.56	128.38	85	36.83	25.21	58.31	35.33	74.93	109.45	26.48	42.63	30.05	67.58004	19.8	27.31	28.615	18.25	41.1
103.03	128.64	84.3	37.01	25.09	58.45	35.5	75.83	110.1	26.525	43.03	29.34	67.27872	19.83	27.47	28.785	18.23	41.17
103.99	128	84.1	36.78	25.44	58.15	35.83	75.9	110.27	26.56	43.33	29.81	67.0339	20.06	27.62	28.77	18.94	41.05
103.74	127.67	84.15	36.81	25.48	58.06	35.92	75.77	109.89	26.47	43.76	29.85	67.11864	19.8	27.26	28.935	18.68	40.7
103.89	128.91	82.97	37.14	25.63	59.16	36.01	76.15	110.11	26.44	43.86	29.7	67.19398	19.79	26.71	29.295	18.32	40.05

Text and Audio Data

Web scrapping (**Beautiful Soup**) was used to automate the import of the raw text from GitHub for each company folder and store them in a Python list. The list was saved to the device as a **Pickle object**.

```
In [7]: master_list_fl=[]
```

```
In [2]: import urllib
        import bs4 as bs
```

```
In [12]: # keep changing the company code and date in the link (20150421_CMG)
```

```
In [136]: source = urllib.request.urlopen('https://raw.githubusercontent.com/Earnings-Call-Dataset/MAEC-A-Multimodal-Aligned-Earnings-C
```

```
In [137]: soup=bs.BeautifulSoup(source,'lxml')
          text=""
          for paragraph in soup.find_all('p'):
              text+=paragraph.text
```

```
In [1]: # text
```

```
In [139]: master_list_fl.append(text)
```

Similarly, Audio data was collected by reading the Dataframes from Github.

DATA CLEANING

Stock Data was clean.

Text Data

For cleaning the Text Data we did the following steps:

- Lower cased the entire paragraph
- Removed stop words
- Removed punctuations
- Fixed English contractions
- Lemmatized the text

In [9]:

```
def cleaning(paragraph):


    sentences= nltk.sent_tokenize(paragraph)
    for sentence in sentences:
        sentence.lower()

    #removing stopwords
    for i in range(len(sentences)):
        words = nltk.word_tokenize(sentences[i])
        words = [word for word in words if word not in set(stopwords.words('english'))]
        sentences[i] = ' '.join(words)

    punc = '''!()-[]{};:'"\,<>./?@#$%^&*_~'''

    # Removing punctuations in string
    #retaining the numbers as they are important to listeners
    sent=[]
    for sentence in sentences:
        for ele in sentence:
            if ele in punc:
                sentence = sentence.replace(ele, "")

        sent.append(sentence)

    return(sent)
```

```
                        "to've": "to have","wasn't": "was not","we'd": "we would",
                        "we'd've": "we would have","we'll": "we will","we'll've": "we will have",
                        "we're": "we are","we've": "we have", "weren't": "were not","what'll": "what will",
                        "what'll've": "what will have","what're": "what are", "what've": "what have",
                        "when've": "when have","where'd": "where did", "where've": "where have",
                        "who'll": "who will","who'll've": "who will have","who've": "who have",
                        "why've": "why have","will've": "will have","won't": "will not",
                        "won't've": "will not have", "would've": "would have","wouldn't": "would not",
                        "wouldn't've": "would not have","y'all": "you all", "y'all'd": "you all would",
                        "y'all'd've": "you all would have","y'all're": "you all are",
                        "y'all've": "you all have", "you'd": "you would","you'd've": "you would have",
                        "you'll": "you will","you'll've": "you will have", "you're": "you are",
                        "you've": "you have"}

# Regular expression for finding contractions
contractions_re=re.compile('(%s)' % '|'.join(contractions_dict.keys()))

# Function for expanding contractions
def expand_contractions(text,contractions_dict=contractions_dict):
  def replace(match):
    return contractions_dict[match.group(0)]
  return contractions_re.sub(replace, text)

# Expanding Contractions in the reviews
df['expanded']=df['transcript'].apply(lambda x:expand_contractions(x))
```

```
[ ] def lemmatization(paragraph):

      for i in range(len(paragraph)):
        words = nltk.word_tokenize(paragraph[i])
        words = [lemmatizer.lemmatize(word) for word in words if word not in set(stopwords.words('english'))]
        paragraph[i] = ' '.join(words)
      return(paragraph)
```

```
[ ] df['lemmatized']=df['cleaned'].apply(lemmatization)
```

Audio Data

For cleaning audio data following steps were done:

- Undefined or NaN were filled with zeros for further processing
- The data type was changed to floating values

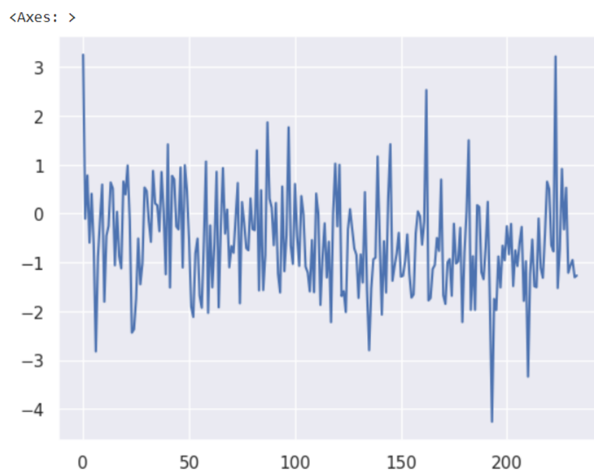
```
In [10]: def process_audio(mylist):

             for audio in mylist:

             # audio[audio == '--undefined-- '] = 0

             # replacing undefined with 0

                 df=pd.DataFrame(audio)
                 df=df.replace('--undefined--', 0)
                 df=df.replace('--undefined-- ', 0)
                 d=np.array(df)

             # change type to float
                 d=d.astype('float64')

             # filling in with zeros to maintain size
                 c = np.concatenate((d, np.zeros([348-d.shape[0], 29])),axis=0)

                 audio_data.append(c)

             return(audio_data)
```

EXPLORATORY DATA ANALYSIS

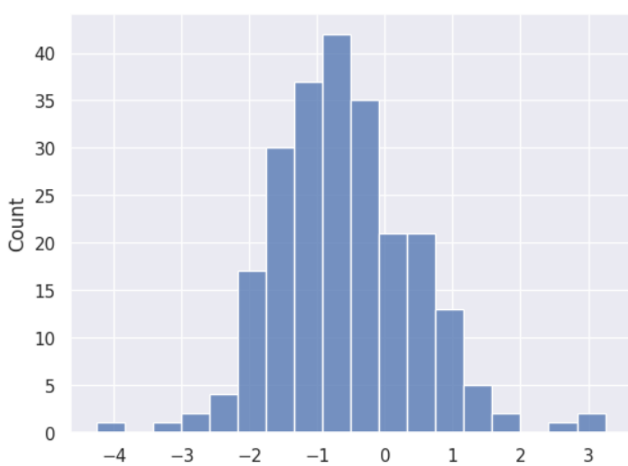
Stock Data:

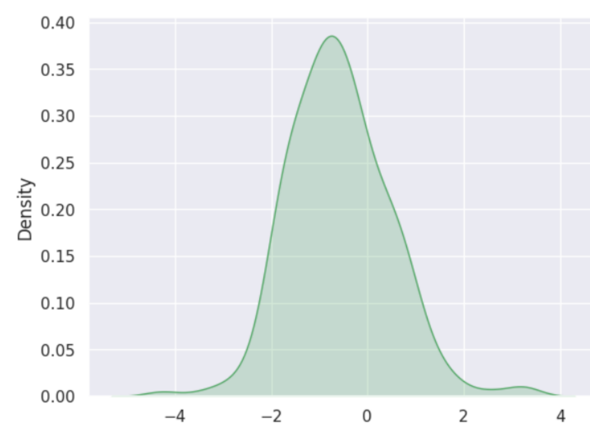
<u>3 DAY STOCK VOLATILITY</u>

Line plot

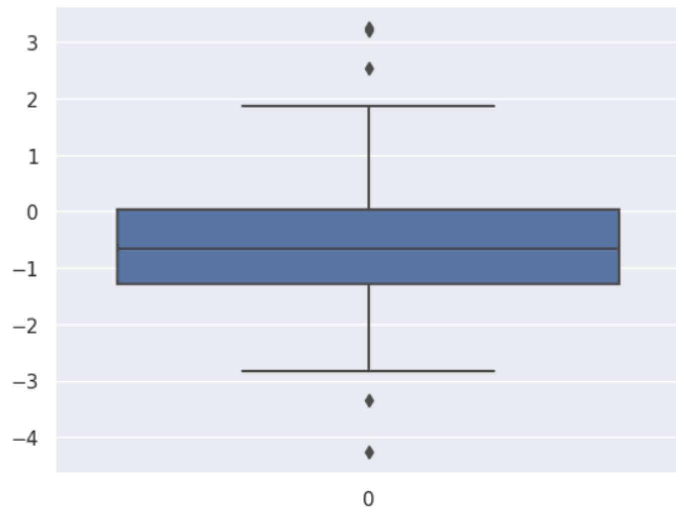


Histogram and KDEplot of the stock price volatility values

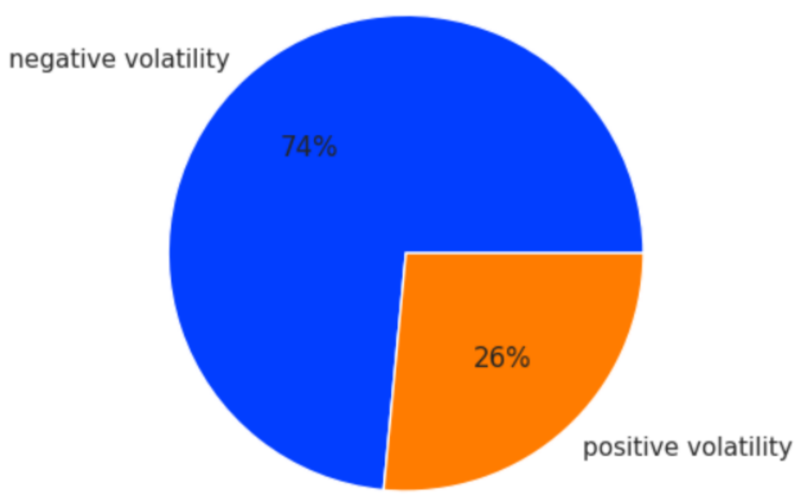




Boxplot



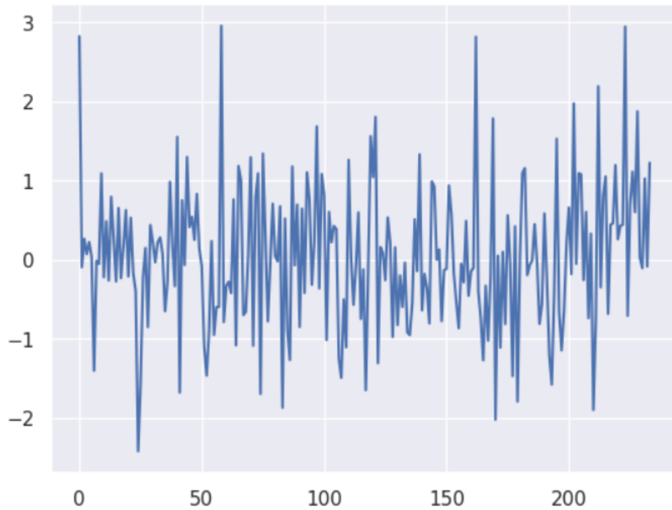
Checking the distribution of positive and negative stock volatility through pie plot



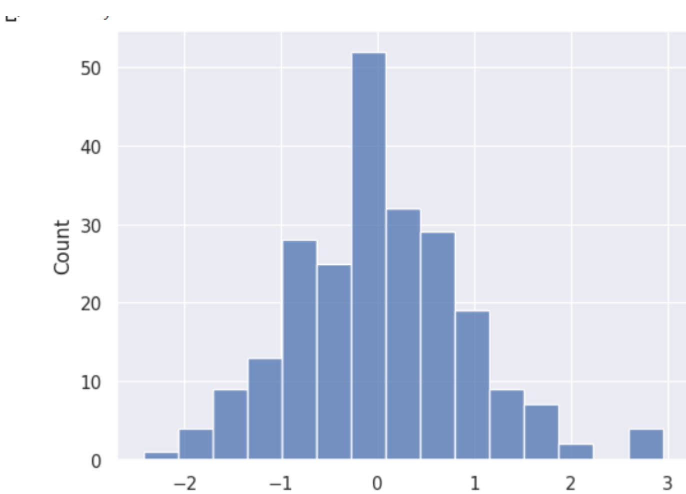
It is skewed data.

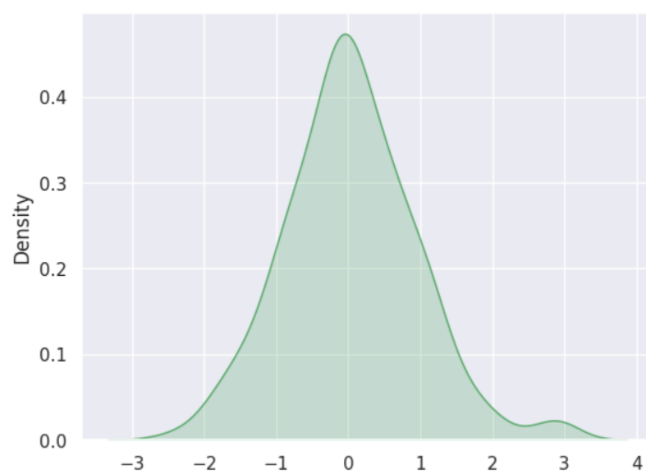
<u>30-DAY STOCK VOLATILITY</u>

Line plot

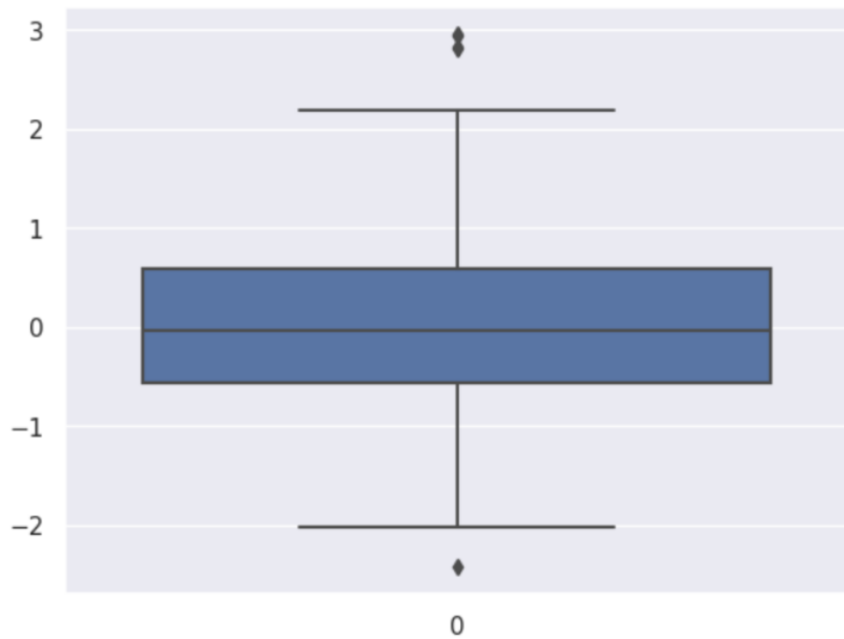


Histogram and KDEplot of the stock price volatility values

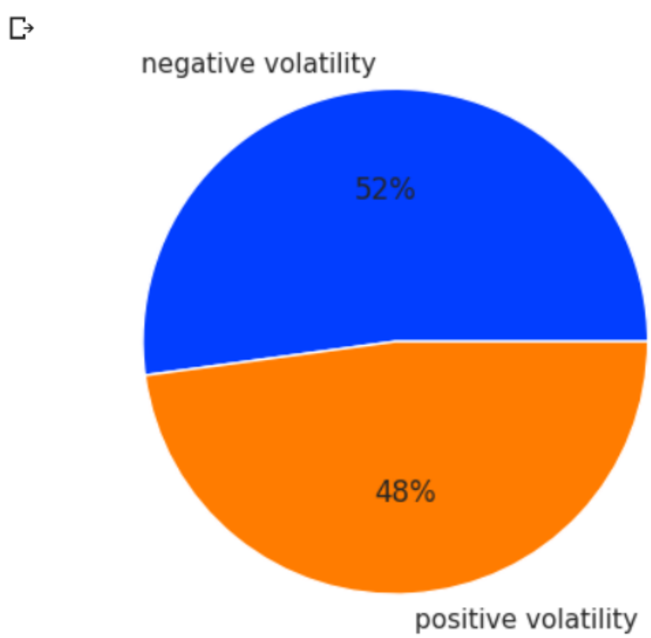




Boxplot

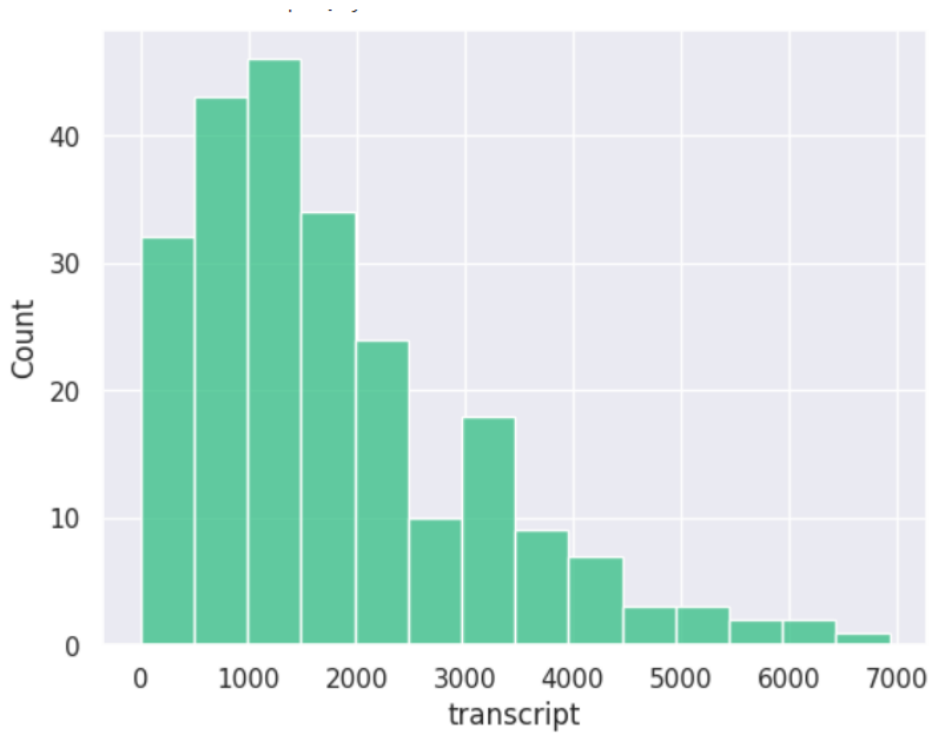


Checking the distribution of positive and negative stock volatility through pie plot

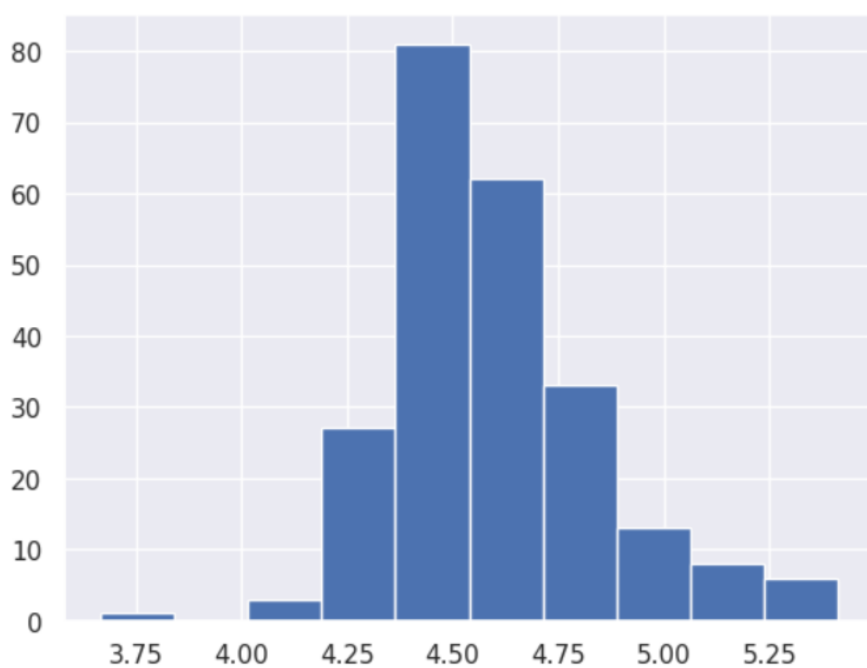


Text Data:

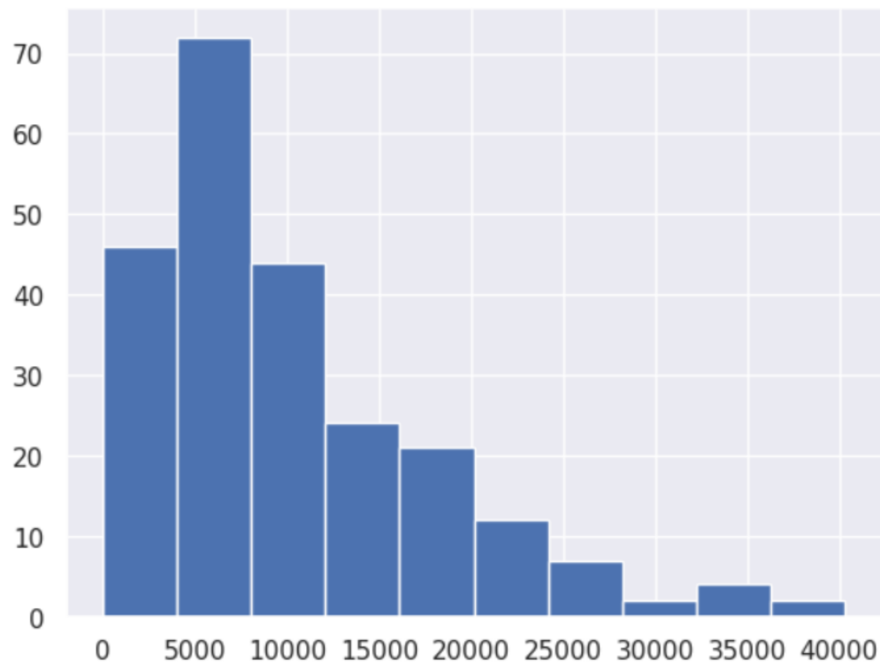
No. of words per transcript:



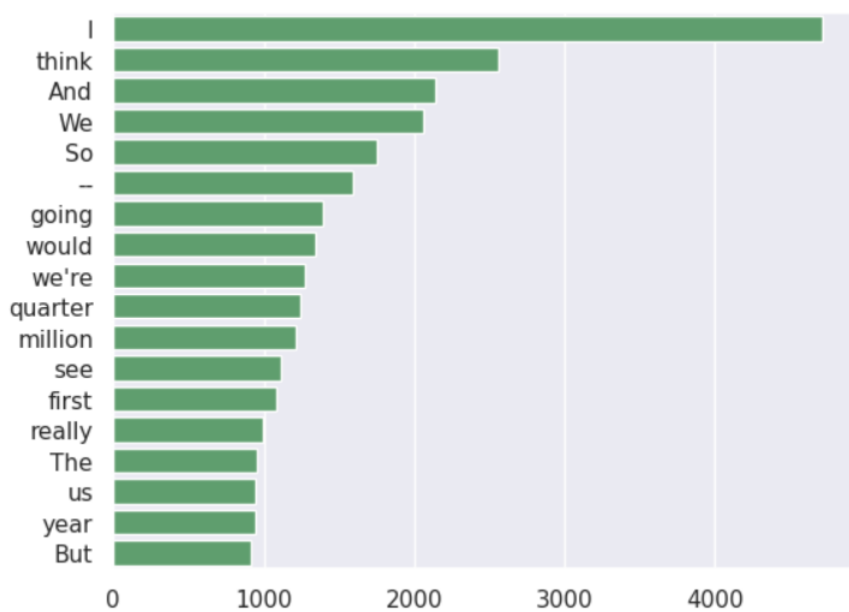
Mean word length ranges:



No. of characters per transcript:



Top 20 used words (Non stop-word):



DATA PREPROCESSING

The following points needed to be kept in mind while preprocessing the **text data**:

- Each sentence was required to be padded to the maximum length of the sentence specified.
- Each paragraph needed to be padded to the maximum length of all paragraphs.
- The paragraphs had to be retained in a sentence-wise format so as to be fed to the model along with their corresponding audio feature.
- The final text and audio features extracted by the model needed to be merged sentence wise

We used **pre-trained word embeddings** and calculated the **arithmetic mean of the word vector in each sentence as the sentence representation**. We chose the embedding **GloVe** pre-trained on Wikipedia and Gigaword 55. Therefore, each sentence is represented as a 50-dimension vector.
GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

Each word is represented as a 300-dimensional vector with GloVe. These vectors are then represented by their arithmetic average to form a concise embedding for the sentences.

```
In [10]: embeddings_dictionary = dict()
         glove_file = open('glove.6B.300d.txt', encoding="utf8")
```

```
In [11]: for line in glove_file:
             records = line.split()
             word = records[0]
             vector_dimensions = np.asarray(records[1:], dtype='float32')
             embeddings_dictionary [word] = vector_dimensions

         glove_file.close()
```

```
In [12]: def get_keys(n):

             key_list = list(word_tokenizer.word_index.keys())
             val_list = list(word_tokenizer.word_index.values())

             # print key with val 100
             position = val_list.index(n)
             return(key_list[position])
```

```
In [13]: def glove(padded_sentences):
             sn=[]
             for sentence in padded_sentences:
                 wn=[]
                 for word in sentence:
                     if(word==0):
                         wn.append(0)
                     else:
                         vector=embeddings_dictionary.get(get_keys(word))
                         if vector is not None:
                             avg=np.average(vector)

                             wn.append(avg)
                         else:
                             wn.append(0)

                 sn.append(wn)
             return(sn)
```

```
In [16]:  for paragraph in master_text:
              processed=cleaning(paragraph)

              # generating embedding
              word_tokenizer = Tokenizer()
              word_tokenizer.fit_on_texts(processed)
              vocab_length = len(word_tokenizer.word_index) + 1
              embedded_sentences = word_tokenizer.texts_to_sequences(processed)

              #padding
              word_count = lambda sentence: len(word_tokenize(sentence))
              longest_sentence = max(processed, key=word_count)
              length_long_sentence = len(word_tokenize(longest_sentence))
              padded_sentences = pad_sequences(embedded_sentences, 30, padding='post')

              #vectorizing
              para=glove(padded_sentences)
              arr=np.array(para)

              #stacking with zeros to maintain shape
              c = np.concatenate((arr, np.zeros([348-arr.shape[0], 30])),axis=0)

              text_arrays.append(c)
              print(count)
              count=count+1
```

For each sentence in an earnings conference call, we generate a 30-dimension text vector and a 29-dimension audio vector to represent verbal and vocal features separately

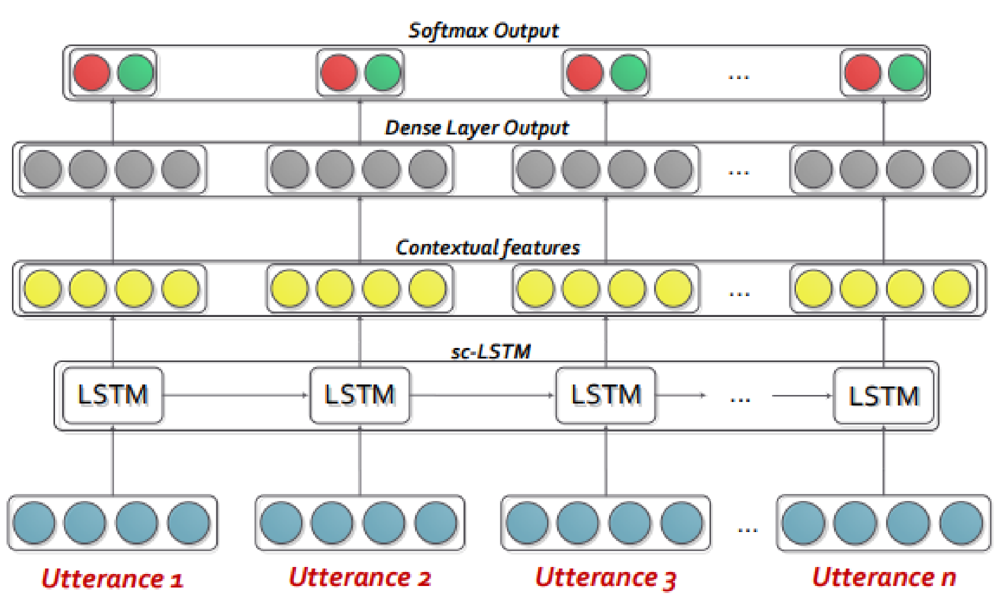
MODEL

Bidirectional LSTM

A Bidirectional Long Short-Term Memory (BiLSTM) is a type of recurrent neural network (RNN) architecture that **combines two LSTM networks**, one processing the input sequence in a forward direction and the other processing it in a backward direction. This bidirectional processing allows the network to capture both past and future context information for each time step in the sequence.
LSTM is designed to acquire key information from time series data while overcoming the defect that traditional RNN might lose information in long time series. BiLSTM is then developed from LSTM, considering not only the forward information transfer but backward transfer.
The bidirectional information transmission significantly improves model prediction power.

Contextual BiLSTM

A contextual Bidirectional Long Short-Term Memory (BiLSTM) is a variation of the standard BiLSTM architecture that incorporates additional contextual information into the network. It aims to enhance the model's ability to capture the context and semantics of the input sequence by considering external knowledge or information. **Following is the architecture of a simple contextual LSTM** **(source: Poria et al., 2017)**



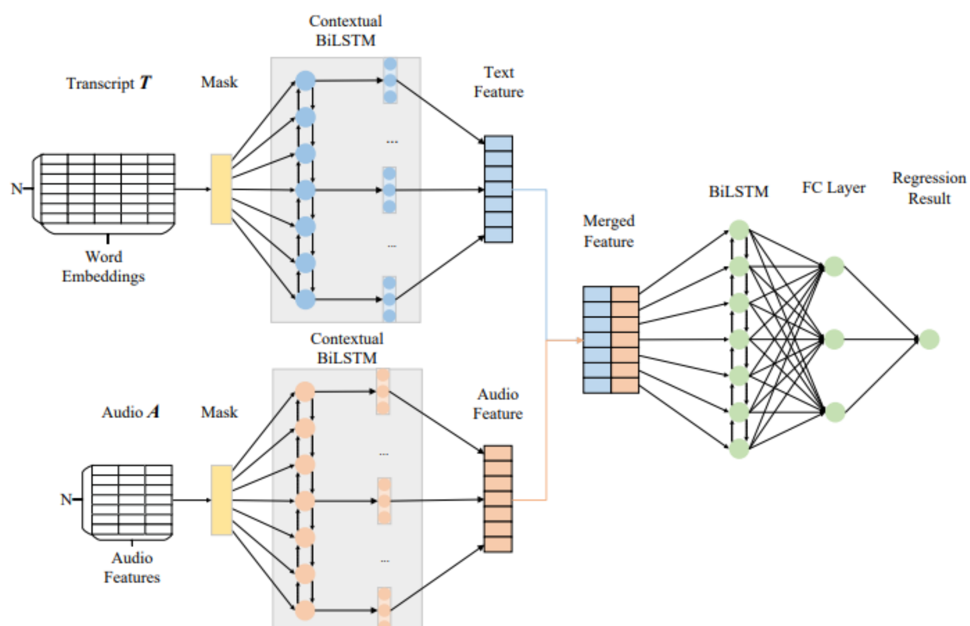
sc-LSTM variant of the contextual LSTM architecture consists of unidirectional LSTM cells. As this is the simple variant of the contextual LSTM, we termed it as simple contextual LSTM

We **replaced the regular LSTM with a bi-directional LSTM** and named the resulting architecture as **bi-directional contextual LSTM (bc-LSTM)**. The training process of this architecture is similar to sc-LSTM.

Hierarchical Fusion of Unimodal Features

Vectors T and A are represented by the matrices on the left. **Matrix T is 348 × 30 dimensional** and **matrix A is 348 × 29 dimensional**, while 348 is the length of the longest paragraph, 30 and 29 are the dimensions of textual features and audio features. The matrices are then fed into **Contextual BiLSTM** through a **Mask layer to screen the effect of zero-padding**.

Contextual BiLSTM extracts unimodal features for each matrix separately while keeping the original chronological order. After extraction, unimodal features are still **organized at the sentence level so they can be horizontally stitched as merged features**.



The merged features are then fed into a BiLSTM connected with a two-layer neural network.

Following is the architecture for processing the unimodal features. **(These architectures have been finalized after a lot of hits and trials to come to the one that gave the best results. These might be subject to further improvements):**

Text:

```
In [73]:  input_data = Input(shape=(text_arrays.shape[1],text_arrays.shape[2]))
          masked = Masking(mask_value =0)(input_data)
          lstm = Bidirectional(LSTM(50, activation='tanh', return_sequences = True, dropout=0.6))(masked)
          inter = Dropout(0.9)(lstm)
          inter1 = TimeDistributed(Dense(100,activation='relu'))(inter)
          inter2 = Dropout(0.9)(inter1)
          output = TimeDistributed(Dense(1,activation='linear'))(inter2)
```

Audio:

```
In [7]:   input_data = Input(shape=(audio_data_processed.shape[1],audio_data_processed.shape[2]))
          masked = Masking(mask_value =0)(input_data)
          lstm = Bidirectional(LSTM(300, activation='tanh', return_sequences = True, dropout=0.6))(masked)
          inter = Dropout(0.5)(lstm)
          lstm2 = Bidirectional(LSTM(300, activation='tanh', return_sequences = True, dropout=0.6))(inter)
          inter2 = Dropout(0.5)(lstm2)
          inter1 = TimeDistributed(Dense(100,activation='tanh'))(inter2)
          inter = Dropout(0.5)(inter1)
          intr=TimeDistributed(Dense(50))
          output = TimeDistributed(Dense(1,activation='linear'))(inter)
```

TRAINING

Optimizer: Adam
Loss function: Mean Squared error

Text Data:

```
In [74]:  model = Model(input_data, output)
          aux = Model(input_data, inter1)
          model.compile(optimizer='adam', loss='mean_squared_error', sample_weight_mode='temporal')
          early_stopping = EarlyStopping(monitor='val_loss', patience=10)
          model.fit(text_arrays, y_train,epochs=10,batch_size=10,shuffle=True,
                            callbacks=[early_stopping],
                            validation_split=0.2)

          Epoch 1/10
          19/19 [==============================] - 13s 233ms/step - loss: 0.2228 - val_loss: 0.2497
          Epoch 2/10
          19/19 [==============================] - 2s 115ms/step - loss: 0.2193 - val_loss: 0.2511
          Epoch 3/10
          19/19 [==============================] - 2s 117ms/step - loss: 0.2085 - val_loss: 0.2502
          Epoch 4/10
          19/19 [==============================] - 2s 119ms/step - loss: 0.2218 - val_loss: 0.2501
          Epoch 5/10
          19/19 [==============================] - 2s 121ms/step - loss: 0.2184 - val_loss: 0.2503
          Epoch 6/10
          19/19 [==============================] - 3s 153ms/step - loss: 0.2019 - val_loss: 0.2500
          Epoch 7/10
          19/19 [==============================] - 3s 150ms/step - loss: 0.2159 - val_loss: 0.2502
          Epoch 8/10
          19/19 [==============================] - 3s 153ms/step - loss: 0.2091 - val_loss: 0.2502
          Epoch 9/10
          19/19 [==============================] - 3s 152ms/step - loss: 0.2185 - val_loss: 0.2508
          Epoch 10/10
          19/19 [==============================] - 3s 151ms/step - loss: 0.2072 - val_loss: 0.2510

Out[74]: <keras.callbacks.History at 0x1a7efa3b5b0>
```

Audio Data:

```
In [8]:  model_a = Model(input_data, output)
         aux_a = Model(input_data, inter1)
         model_a.compile(optimizer='adadelta', loss='mean_squared_error', sample_weight_mode='temporal')
         early_stopping = EarlyStopping(monitor='val_loss', patience=10)
         model_a.fit(audio_data_processed, y_train,epochs=10,batch_size=10,shuffle=True,
                           callbacks=[early_stopping],
                           validation_split=0.2)

         Epoch 1/10
         19/19 [==============================] - 50s 2s/step - loss: 0.3477 - val_loss: 0.4133
         Epoch 2/10
         19/19 [==============================] - 37s 2s/step - loss: 0.3348 - val_loss: 0.4061
         Epoch 3/10
         19/19 [==============================] - 38s 2s/step - loss: 0.3392 - val_loss: 0.4003
         Epoch 4/10
         19/19 [==============================] - 37s 2s/step - loss: 0.3489 - val_loss: 0.3947
         Epoch 5/10
         19/19 [==============================] - 37s 2s/step - loss: 0.3424 - val_loss: 0.3891
         Epoch 6/10
         19/19 [==============================] - 38s 2s/step - loss: 0.3330 - val_loss: 0.3840
         Epoch 7/10
         19/19 [==============================] - 39s 2s/step - loss: 0.3343 - val_loss: 0.3791
         Epoch 8/10
         19/19 [==============================] - 40s 2s/step - loss: 0.3337 - val_loss: 0.3739
         Epoch 9/10
         19/19 [==============================] - 41s 2s/step - loss: 0.3296 - val_loss: 0.3705
         Epoch 10/10
         19/19 [==============================] - 42s 2s/step - loss: 0.3391 - val_loss: 0.3670
```

The activations from the last layer of each of these unimodal architectures are taken and merged together:

In [78]:
```
train_data = np.concatenate((train_activations_30t, train_activations_30), axis=2)
```

In [106]:
```
test_data = np.concatenate((test_activations_30t, test_activations_30), axis=2)
```

Final Training of the multi-modal features:

In [95]:
```
input_data = Input(shape=(train_data.shape[1],train_data.shape[2]))
masked = Masking(mask_value =0)(input_data)
lstm = Bidirectional(LSTM(50, activation='tanh', dropout=0.4))(masked)
inter = Dropout(0.9)(lstm)
lstm2 = (Dense(50, activation='relu'))(inter)
inter2 = Dropout(0.9)(lstm2)
output = Dense(1)(inter2)
```

In [96]:
```
model2 = Model(input_data, output)
# aux2 = Model(input_data, inter1)
model2.compile(optimizer='adam', loss='mean_squared_error', sample_weight_mode='temporal')
early_stopping = EarlyStopping(monitor='val_loss', patience=10)
model2.fit(train_data, y_train,
                epochs=30,
                batch_size=10,
#                 sample_weight=train_mask,
                shuffle=True,
                callbacks=[early_stopping],
                validation_split=0.2)
```

```
Epoch 1/30
19/19 [==============================] - 10s 237ms/step - loss: 2.0479 - val_loss: 1.0568
Epoch 2/30
19/19 [==============================] - 2s 119ms/step - loss: 1.4057 - val_loss: 1.0440
Epoch 3/30
19/19 [==============================] - 2s 122ms/step - loss: 1.1553 - val_loss: 1.0423
Epoch 4/30
19/19 [==============================] - 2s 126ms/step - loss: 1.0282 - val_loss: 1.0431
Epoch 5/30
```

PREDICTIONS AND EVALUATIONS

We report the performance using the Mean Squared Error (MSE) between the predicted volatility and true volatility:

$$MSE = \frac{1}{M'} \sum_{i=1}^{M'} (f(\boldsymbol{X}'_i) - y'_i)^2$$

where M' is the size of the test set, and y 0 i is the true volatility associated with testing example X'i .

Evaluations of prediction on Test set:

```
In [26]:  from sklearn.metrics import mean_squared_error

In [110]: mean_squared_error(y_true,result_test)
Out[110]: 1.0610654063071112
```

MSE= 1.0610

Evaluations of predictions on Train set:

```
In [99]:  mean_squared_error(y_train,result_train)
Out[99]:  0.8264861277222201
```

MSE= 0.8265

<u>On 3-day stock volatility</u>

Test predictions:

```
In [77]:  mean_squared_error(y_res_true,result_test)
Out[77]:  1.148580639154286
```

MSE=1.148

OTHER MODELS

Roberta

The RoBERTa model, pre-trained on a large corpus of text data, is fine-tuned using a dataset consisting of speeches given by CEOs of various companies. The model is trained to understand the nuances of language, contextual information, and sentiment expressed in these speeches. By capturing the underlying sentiment and key information from the speeches, the model can identify potential factors affecting stock variance.

Steps followed

1. Preprocess the training data:

```
In [11]: import re
import torch
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from transformers import RobertaTokenizer, RobertaModel
import string

# Load pre-trained RoBERTa model and tokenizer
model_name = 'roberta-base'
tokenizer = RobertaTokenizer.from_pretrained(model_name)
model = RobertaModel.from_pretrained(model_name)



# Preprocess the text data
def preprocess_text(text):
    # Remove special characters and digits
    text = re.sub(r"[^a-zA-Z]", " ", text)
    text = re.sub(r'http\S+', '', text)
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    text = text.translate(str.maketrans('', '', string.punctuation))
    text = text.lower()

    # Convert to lowercase
    text = text.lower()

    return text
```

```
                                                  Speech    target
1    □  I don't think  it will turn negative.But wh...  0.905962
2     □  It's sophisticated options traders' accoun...  1.294760
3    Z$  I will take the first one and I will throw...  1.070070
4    3.  Yes I think the other key market I would c...  1.244030
5    □  Thanks.Our -- if you look at our business a...  1.031935
..                                                 ...       ...
226  □□  Thank you.Yes .We're glad we found a front...  3.029982
227  5  Okay thank you.Good afternoon and thank you...  1.818684
228  □  This is  and I'll tell you there's certain ...  6.495490
229  5□  Thanks .On the Edge take rate look this is...  1.025679
230  &  The first place I think we are going to bec...  0.895575

[230 rows x 2 columns]
```

2. Convert the text data into contextual embeddings:

```
# Tokenize the text dataset and generate contextual embeddings
embeddings = []
for text in df['Speech']:
    # Preprocess the text
    text = preprocess_text(text)

    tokens = tokenizer.encode(text, add_special_tokens=True, truncation=True, max_length=512)
```

3. Aggregate the contextual embeddings:

```
    input_ids = torch.tensor([tokens])
    with torch.no_grad():
        outputs = model(input_ids)
        last_hidden_states = outputs[0]
    avg_embeddings = torch.mean(last_hidden_states, dim=1).numpy().tolist()[0]
    test_embeddings.append(avg_embeddings)
```

4. if the token length is greater than 512 token, we split it further

```
    if len(tokens) > 512:
        tokens = tokens[:511] + [tokenizer.sep_token_id]
```

5. Train the regression model, using random forest:

```
In [13]: # Train a random forest regression model on the aggregated embeddings and target variable
         regression_model = RandomForestRegressor()

         regression_model.fit(embeddings, df['target'])
```

6. Evaluate the model, we have test data of 61 speeches on which we check the model.

```
In [15]: import pandas as pd

         # Read the CSV file into a DataFrame
         df3 = pd.read_csv('Y-VAL (1).csv')

         # Extract values from the third column of the CSV file
         column_values = df3['std dev 30']

         # Create a new column in the DataFrame
         df2['target'] = column_values

         # Print the updated DataFrame
         print(df2)
```

```
                                                Speech    target
1   &  Well all fair questions .We went into 2015 ...   1.537117
2   □  Yes.In the third quarter I talked about 70 ...   0.318935
3   T  Good day ladies and gentlemen and welcome t...   0.505257
4   □  Again  fair question.Clearly the best inves...   3.820940
5   l□  Good afternoon.You know I would tell you t...   3.556293
6   □  And again that really does speak to -- back...   3.113278
7   h□  Given that we're approaching the top of th...   2.466278
8    Thank you .Dan this is  .Biofuels is absolu...   0.991578
9   "   Yes thanks for your question.We have not b...   1.842056
10  □  No I guess what we're saying  is we will be...   1.346607
11  %  On the Stream product we didn't have to obt...   3.441009
12  b3  Okay  let me start with the interchange.Th...   2.350453
13   million to spend on digital and that's it.Wha...   0.171264
14  T□  Yes I think Matt right now we expect it to...   2.803862
15  □□  Yes that's a great question .That's -- we ...   1.908405
16  □□  What we've shared is that some portion of ...   3.080126
17  \n□  Thanks Shannon.Thanks . we feel very good...   0.587223
18  [(  Good morning .You've got two items outside...   0.689017
19  □  Good morning and welcome to our quarterly e...   1.147616
20  F  Thank you  and good morning.Since joining t...   4.413306
```

```
In [16]: test_data = df2['Speech'].tolist()
         preprocessed_test_data = [preprocess_text(text) for text in test_data]
```

```
In [17]: test_embeddings = []
         for text in preprocessed_test_data:
             tokens = tokenizer.encode(text, add_special_tokens=True, truncation=True, max_length=512)
             if len(tokens) > 512:
                 tokens = tokens[:511] + [tokenizer.sep_token_id]
             input_ids = torch.tensor([tokens])
             with torch.no_grad():
                 outputs = model(input_ids)
                 last_hidden_states = outputs[0]
             avg_embeddings = torch.mean(last_hidden_states, dim=1).numpy().tolist()[0]
             test_embeddings.append(avg_embeddings)
```

```
In [18]: predicted_varience = regression_model.predict(test_embeddings)
```

FINBert

The FINBERT model is trained on a specialized dataset consisting of financial texts, including CEO and MD speeches. This training enables the model to understand the unique language and context-specific to financial discussions. By analyzing the sentiment expressed in these speeches, the model can uncover insights that may impact stock variance.

Steps followed

1. Downloading the important Libraries:

```
In [10]: import re
         import torch
         import pandas as pd
         from sklearn.ensemble import RandomForestRegressor

         import string
         from transformers import BertModel
         from transformers import BertTokenizer, BertForSequenceClassification
```

2. loading a pre-trained FinBERT model for sequence classification and its corresponding tokenizer. FinBERT is a specialized variant of BERT that has been fine-tuned on financial text data for sentiment analysis and other financial NLP tasks:

```
finbert = BertForSequenceClassification.from_pretrained('yiyanghkust/finbert-tone',num_labels=3)
tokenizer = BertTokenizer.from_pretrained('yiyanghkust/finbert-tone')
# Load pre-trained RoBERTa model and tokenizer
model_name = 'yiyanghkust/finbert-tone'

model = BertModel.from_pretrained(model_name)
```

3. Again, we do the preprocessing:

```
# Preprocess the text data
def preprocess_text(text):
    # Remove special characters and digits
    text = re.sub(r"[^a-zA-Z]", " ", text)
    text = re.sub(r'http\S+', '', text)
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    text = text.translate(str.maketrans('', '', string.punctuation))
    text = text.lower()

    # Convert to lowercase
    text = text.lower()

    return text
```

4. Convert the text data into contextual embeddings:

```
# Tokenize the text dataset and generate contextual embeddings
embeddings = []
for text in df['Speech']:
    # Preprocess the text
    text = preprocess_text(text)

    tokens = tokenizer.encode(text, add_special_tokens=True, truncation=True, max_length=512)
```

5.if the token length is greater than 512 token, we split it further:

```
    # If the sequence length is still longer than 512 after truncation, you can further split it
    if len(tokens) > 512:
        tokens = tokens[:511] + [tokenizer.sep_token_id]
```

6. Aggregate the contextual embeddings:

```
    input_ids = torch.tensor([tokens])
    with torch.no_grad():
        outputs = model(input_ids)
        last_hidden_states = outputs[0]

    avg_embeddings = torch.mean(last_hidden_states, dim=1).numpy().tolist()[0]
    embeddings.append(avg_embeddings)
```

7. Train the regression model, using random forest:

```
In [11]: # Train a random forest regression model on the aggregated embeddings and target variable
         regression_model = RandomForestRegressor()

         regression_model.fit(embeddings, df['target'])
```

8..Evaluate the model, we have test data of 61 speeches on which we check the model.

```
In [13]: import pandas as pd

         # Read the CSV file into a DataFrame
         df3 = pd.read_csv('Y-VAL (1).csv')

         # Extract values from the third column of the CSV file
         column_values = df3['std dev 30']

         # Create a new column in the DataFrame
         df2['target'] = column_values

         # Print the updated DataFrame
         print(df2)

                                                   Speech    target
         1    &  Well all fair questions .We went into 2015 ...  1.537117
         2    ⍰  Yes.In the third quarter I talked about 70 ...  0.318935
         3    T  Good day ladies and gentlemen and welcome t...  0.505257
         4    ⍰  Again  fair question.Clearly the best inves...  3.820940
         5    1⍰  Good afternoon.You know I would tell you t...  3.556293
         6    ⍰  And again that really does speak to -- back...  3.113278
         7    h⍰  Given that we're approaching the top of th...  2.466278
         8     Thank you .Dan this is  .Biofuels is absolu...  0.991578
         9    "   Yes thanks for your question.We have not b...  1.842056
         10   ⍰  No I guess what we're saying  is we will be...  1.346607
         11   %  On the Stream product we didn't have to obt...  3.441009
         12   b3  Okay  let me start with the interchange.Th...  2.350453
         13    million to spend on digital and that's it.Wha...  0.171264
         14   T⍰  Yes I think Matt right now we expect it to...  2.803862

In [14]: test_data = df2['Speech'].tolist()
         preprocessed_test_data = [preprocess_text(text) for text in test_data]

In [15]: test_embeddings = []
         for text in preprocessed_test_data:
             tokens = tokenizer.encode(text, add_special_tokens=True, truncation=True, max_length=512)
             if len(tokens) > 512:
                 tokens = tokens[:511] + [tokenizer.sep_token_id]
             input_ids = torch.tensor([tokens])
             with torch.no_grad():
                 outputs = model(input_ids)
                 last_hidden_states = outputs[0]
             avg_embeddings = torch.mean(last_hidden_states, dim=1).numpy().tolist()[0]
             test_embeddings.append(avg_embeddings)

In [16]: predicted_prices = regression_model.predict(test_embeddings)
```

Evaluation:

Roberta -

We try to do simple prediction on a non financial text line and its is giving very high variance relative to the target data, which is good, since line is not giving much information about the company conditions

```
In [13]: # Train a random forest regression model on the aggregated embeddings and target variable
regression_model = RandomForestRegressor()

regression_model.fit(embeddings, df['target'])

# Make predictions on new data
new_data = ["This is a new text to predict the average stock price using RoBERTa embeddings."]
new_embeddings = []
for text in new_data:
    # Preprocess the text
    text = preprocess_text(text)

    tokens = tokenizer.encode(text, add_special_tokens=True)
    input_ids = torch.tensor([tokens])
    with torch.no_grad():
        outputs = model(input_ids)
        last_hidden_states = outputs[0]
    avg_embeddings = torch.mean(last_hidden_states, dim=1).numpy().tolist()[0]
    new_embeddings.append(avg_embeddings)
predicted_price = regression_model.predict(new_embeddings)

# Print the predicted stock price
print(predicted_varience)

[5.76444759]
```

The MSE score on the 61 test data speeches is as follows

```
In [19]: from sklearn.metrics import mean_squared_error

actual_prices = df2['target'].tolist()
mse_score = mean_squared_error(actual_varience, predicted_varience)

print("MSE Score:", mse_score)

MSE Score: 4.563866846508192
```

FINBert

We try to do simple prediction on a non-financial text line and its is giving very high variance relative to the target data, which is good, since line is not giving much information about the company conditions

```
# Make predictions on new data
new_data = ["This is a new text to predict the average stock varience using BERTa embeddings."]
new_embeddings = []
for text in new_data:
    # Preprocess the text
    text = preprocess_text(text)

    tokens = tokenizer.encode(text, add_special_tokens=True)
    input_ids = torch.tensor([tokens])
    with torch.no_grad():
        outputs = model(input_ids)
        last_hidden_states = outputs[0]
    avg_embeddings = torch.mean(last_hidden_states, dim=1).numpy().tolist()[0]
    new_embeddings.append(avg_embeddings)
predicted_price = regression_model.predict(new_embeddings)

# Print the predicted stock varience
print(predicted_varience)

[7.17447542]
```

The MSE score on the 61 test data speeches is as follows

```
In [17]: from sklearn.metrics import mean_squared_error

actual_prices = df2['target'].tolist()
mse_score = mean_squared_error(actual_varience, predicted_varience)

print("MSE Score:", mse_score)

MSE Score: 5.464426334254537
```

Bert Base Uncased

<u>We try this model on 3-day stock price volatility to judge short-term effectiveness</u>

Import necessary libraries

```
[14] import transformers
     from transformers import BertModel, BertTokenizer, AdamW, get_linear_schedule_with_warmup
     import torch

     from torch import nn, optim
     from torch.utils.data import Dataset, DataLoader

     from sklearn.model_selection import train_test_split
     from sklearn.metrics import confusion_matrix, classification_report
     from collections import defaultdict
     from textwrap import wrap
```

```
[15] from transformers import BertPreTrainedModel, BertModel
     from transformers import AutoConfig, AutoTokenizer

     from sklearn import metrics
     from sklearn.model_selection import train_test_split
     from tqdm import tqdm, trange
```

Set the parameters

```
[44] MAX_LEN_TRAIN = 512
     MAX_LEN_VALID = 512
     MAX_LEN_TEST = 512
     BATCH_SIZE = 16
     LR = 5e-5
     NUM_EPOCHS = 5
     NUM_THREADS = 1  ## Number of threads for collecting dataset
     MODEL_NAME = 'bert-base-uncased'
```

Create a function for preparing data

```
class Excerpt_Dataset(Dataset):

    def __init__(self, data, maxlen, tokenizer):
        #Store the contents of the file in a pandas dataframe
        self.df = data.reset_index()
        #Initialize the tokenizer for the desired transformer model
        self.tokenizer = tokenizer
        #Maximum length of the tokens list to keep all the sequences of fixed size
        self.maxlen = maxlen

    def __len__(self):
        return self.df.shape[0]

    def __getitem__(self, index):
        #Select the sentence and label at the specified index in the data frame
        excerpt = self.df.loc[index, 'excerpt']
        try:
            target = self.df.loc[index, 'target']
        except:
            target = 0.0
        # identifier = self.df.loc[index, 'id']
        #Preprocess the text to be suitable for the transformer
        tokens = self.tokenizer.tokenize(excerpt)
        tokens = ['[CLS]'] + tokens + ['[SEP]']
        if len(tokens) < self.maxlen:
            tokens = tokens + ['[PAD]' for _ in range(self.maxlen - len(tokens))]
        else:
            tokens = tokens[:self.maxlen-1] + ['[SEP]']
        #Obtain the indices of the tokens in the BERT Vocabulary
        input_ids = self.tokenizer.convert_tokens_to_ids(tokens)
        input_ids = torch.tensor(input_ids)
        #Obtain the attention mask i.e a tensor containing 1s for no padded tokens and 0s for padded ones
        attention_mask = (input_ids != 0).long()

        target = torch.tensor(target, dtype=torch.float32)

        return input_ids, attention_mask, target
```

The Model

```
[46] class BertRegresser(BertPreTrainedModel):
         def __init__(self, config):
             super().__init__(config)
             self.bert = BertModel(config)
             #The output layer that takes the [CLS] representation and gives an output
             self.cls_layer1 = nn.Linear(config.hidden_size,128)
             self.relu1 = nn.ReLU()
             self.ff1 = nn.Linear(128,128)
             self.tanh1 = nn.Tanh()
             self.ff2 = nn.Linear(128,1)

         def forward(self, input_ids, attention_mask):
             #Feed the input to Bert model to obtain contextualized representations
             outputs = self.bert(input_ids=input_ids, attention_mask=attention_mask)
             #Obtain the representations of [CLS] heads
             logits = outputs.last_hidden_state[:,0,:]
             output = self.cls_layer1(logits)
             output = self.relu1(output)
             output = self.ff1(output)
             output = self.tanh1(output)
             output = self.ff2(output)
             return output
```

Training Function

```python
def train(model, criterion, optimizer, train_loader, val_loader, epochs, device):
    best_acc = 0
    for epoch in trange(epochs, desc="Epoch"):
        model.train()
        train_loss = 0
        for i, (input_ids, attention_mask, target) in enumerate(iterable=train_loader):
            optimizer.zero_grad()

            input_ids, attention_mask, target = input_ids.to(device), attention_mask.to(device), target.to(device)

            output = model(input_ids=input_ids, attention_mask=attention_mask)

            loss = criterion(output, target.type_as(output))
            loss.backward()
            optimizer.step()

            train_loss += loss.item()

        print(f"Training loss is {train_loss/len(train_loader)}")
        val_loss = evaluate(model=model, criterion=criterion, dataloader=val_loader, device=device)
        print("Epoch {} complete! Validation Loss : {}".format(epoch, val_loss))
    return model
```

Other Utility Functions

```python
def evaluate(model, criterion, dataloader, device):
    model.eval()
    mean_err, mean_loss, count = 0, 0, 0

    with torch.no_grad():
        for input_ids, attention_mask, target in (dataloader):

            input_ids, attention_mask, target = input_ids.to(device), attention_mask.to(device), target.to(device)
            output = model(input_ids, attention_mask)

            mean_loss += criterion(output, target.type_as(output)).item()
            mean_err += get_rmse(output, target)
            count += 1

    return mean_loss/count
```

```python
[49] def get_rmse(output, target):
    err = torch.sqrt(metrics.mean_squared_error(target, output))
    return err
```

```python
def predict(model, dataloader, device):
    predicted_label = []
    actual_label = []
    with torch.no_grad():
        for input_ids, attention_mask, target in (dataloader):

            input_ids, attention_mask, target = input_ids.to(device), attention_mask.to(device), target.to(device)
            output = model(input_ids, attention_mask)

            predicted_label += output
            actual_label += target

    return predicted_label
```

Final Evaluation:

```
from sklearn.metrics import mean_squared_error

mean_squared_error(op, df['target'])
```

```
0.9682033609532474
```

Mean squared error on test data:
MSE= 0.968

FURTHER ENHANCEMENTS

The given models used like Roberta and Finbert have some limitations like
1.they are trained on 512 embeddings and text data used is most of the time bigger than this limit.
2. They can be biased, reflecting the biases in the data they were trained on.
3. Roberta is a large language model, and it is not specifically designed for predicting stock variance.

To overcome these issues we have some newly introduced models , which are introduced in recent years (2021-22). For example, RWKV , Switch Transformer , XLM Roberta XL , These model are not trained specifically for financial data , but provide more accurate results , with general NLP tasks and hence can be expected it also works for Financial dataset in same way.
Here is a detail comparison

Model	Size	Parameters	Training Data	Tasks	Strengths	Weaknesses
Roberta	137B	340M	BooksCorpus and English Wikipedia	Natural language understanding, question answering, text generation	High accuracy, good performance on long-range dependencies	Can be slow for some tasks
FinBERT	340M	110M	Financial news articles	Extract sentiment from financial news and predict stock prices	Good performance on financial tasks	Not as well-rounded as Roberta
rwkv	1.37T	1.37B	BooksCorpus, English Wikipedia, and Common Crawl	Natural language understanding, question answering, text generation, code summarization	State-of-the-art performance on many tasks	Can be slow for some tasks

Switch Transformer	175 B	600M	BooksCorpus, English Wikipedia, and Common Crawl	Natural language understanding, question answering, text generation, code summarization	High accuracy and good performance on long-range dependencies	Can be slow for some tasks
XLM Roberta XL	1.37 T	1.37B	BooksCorpus, English Wikipedia, and Common Crawl	Natural language understanding, question answering, text generation, machine translation	High accuracy and good performance on long-range dependencies, multilingual support	Can be slow for some tasks

CONCLUSION

Model	30-Day stock price volatility (MSE)	3-Day stock price volatility (MSE)
Contextual BiLSTM (text+audio features)	1.06	1.14
RoBerta (text only)	4.54	-
FinBert (text only)	5.46	-
Bert Base Uncased (text only)	-	0.968