Analisi del sentiment e dell'emotions in relazione al movimento dei titoli

Progetto di esame a cura di Alberto Mastromarino

Web Analytics e Analisi Testuale - A.A. 2021/2022
Docente: Prof. Marco Ortu
Università degli Studi di Cagliari

INDICE

1. Introduzione

Il progetto svolto cerca di sfruttare il più possibile le metodologie e le tecniche introdotte durante il corso di Web Analytics e Analisi testuale, in particolare si sofferma su:
web scraping e social mining, sentiment analysis, topic modelling (Latent Dirichlet Allocation) e Natural Language Processing.
L'attenzione è stata rivolta maggiormente al social network Reddit, dal quale sono stati estratti i dati testuali di un determinato subreddit chiamato WallStreetBets; si è deciso di estrarre sia i post contenenti soltanto testo, che i titoli. L'analisi è stata svolta su un periodo di tempo di 1 anno e 6 mesi, a partire dal dicembre 2020, fino a maggio 2022 compreso (per un totale di 677802 post).

1.1. Obiettivo del progetto

L'obiettivo finale del progetto è quello di costruire un modello capace di comprendere, tramite l'analisi testuale dei post o dei titoli presenti all'interno del subreddit WallStreetBets, se esista una correlazione tra il sentimento dei post/dei titoli e l'andamento di un determinato titolo azionario. In questo caso si utilizzeranno i post e i titoli di WallStreetBets e come titolo azionario quotato in borsa, GameStop (GME). Inizialmente, ci sarà una doppia fase di scraping, verranno estratti sia i dati contenuti all'interno di Reddit che i dati inerenti all'azione scelta (Open price, Close Price, Volume, Date). Successivamente ci sarà una fase di ripulitura dei dati collezionati e, dopo averli salvati separatamente in file formato .csv, da una parte i post e dall'altra i titoli, verrà eseguita una *sentiment analysis* tramite la libreria NLTK, utilizzando VADER, e un'analisi dell'*emotion* tramite la libreria NRCLex. Infine, tramite una *LDA* si cercherà di capire quali sono i principali topic.
La domanda a cui si è cercato di rispondere in questo progetto nella creazione del modello finale è stata: i post/titoli generati dagli utenti all'interno del social network Reddit riescono ad influenzare l'andamento del mercato? Quanto il sentimento dei post/titoli

degli utenti riesce a incidere sull'andamento di un titolo azionario, più precisamente, il Close Price?

1.2. Struttura del progetto

Il progetto è suddiviso in 3 branch sulla piattaforma BitBucket:

- Il branch "*master*" che comprende i due oggetti Scraper (Reddit_Scraper e Finance), quello per i post/titoli di Reddit e quello per i dati azionari. All'interno del branch troviamo i file formato .csv *total_stock* per le informazioni di GME e *total_period* per quanto riguarda i post/titoli, ottenuti attraverso il codice.
- Il branch "*data_cleaning*" che contiene lo script *data_cleaning.py* che andrà a ripulire il file total_period.csv e produrrà due file formato .csv, total_period_post e total_period_title. All'interno del branch sono presenti anche gli script *sentiment_analysis.py* e *topic_modelling.py*. Il primo produrrà i file formato .csv sentiment_emotion_total_post e sentiment_emotion_total_title e il secondo LDA_Visualization_title.html e LDA_Visualization_post.html.
- Il branch "analysis" che contiene lo script *Linear_Discriminant_Analysis.py* che prenderà in esame i file sentiment_emotion_total_post.csv e sentiment_emotion_total_title.csv insieme a total_stocks.csv

2. Svolgimento

2.1 Web scraping

2.1.1 Estrazione dei post/titoli da Reddit

Il branch "master" ha come obiettivo la raccolta di dati, tramite il web scraping. Nel codice sono state utilizzate delle classi, come si può notare subito nei due file reddit_scraper.py e stock_scraper.py.

Per quanto riguarda lo scraper di reddit, si è creata una classe chiamata "*Reddit_Scraper*"

e nell'inizializzazione si trovano before e after, due parametri che saranno utili quando si andrà a scegliere quale periodo temporale andare ad analizzare.
La classe è stata suddivisa in due parti, tramite due funzioni ben definite. La prima "*historical_posts*" che permette di andare ad effettuare lo scraping di un subreddit scelto dall'utente, impostando anche un limite di post che può essere evitato utilizzando la dicitura "None", in modo tale che vengano presi tutti i post di quel determinato periodo; questo a seconda del periodo preso in esame, rallenterà il processo ma permette di ottenere una grande quantità di dati.
Per questo scraper è stata utilizzata l'API di Reddit chiamata PushShift (PSAW) che permette di eseguire uno scraping con un datetime, a differenza dell'API ufficiale di Reddit PRAW che non permette di utilizzare dei parametri temporali e ha dei limiti di post giornalieri.
All'interno della funzione "*historical_posts*" si utilizza un blocco try and except, in modo tale che qualora dovesse entrare nell'except andrebbe a fornire l'errore che affronta il codice (es. Attribute Error), e la riga esatta in cui è presente.
Dopo la creazione dell'oggetto scraper all'esterno della classe, si utilizza la funzione search.submission all'interno della variabile posts. Successivamente tramite una list comprehension (i.d_ è un dizionario offerto dall'API PushShift che contiene tutti gli attributi dei dati che stiamo richiedendo) vengono convertiti tutti i dati ottenuti in un dataframe di pandas. Infine, sempre tramite pandas si crea una colonna '*date*' e si sceglie quali colonne salvare all'interno del dataframe. Si utilizza anche un print.df.shape e un print.df.head() per vedere la grandezza del dataframe e le prime 5 righe. Come ultimo task, la funzione salva il dataframe in un file .csv che potrà poi successivamente essere riaperto per essere ripulito.

```
class Reddit_Scraper(object):

    """..."""

    def __init__(self,before, after):
        self.before = before
        self.after = after

        print(f'Si inizia a raccogliere dati :-) Attendi...')
```

Figura 1 Classe Reddit_Scraper

```
def historical_posts(self,subreddit, limit):
    """..."""
    self.subreddit = subreddit
    self.limit = limit
    try:

        scraper = PushshiftAPI()
        posts = scraper.search_submissions(
            subreddit=self.subreddit,
            before=after,
            after=before,
            limit=self.limit
        )
        df = pd.DataFrame([i.d_ for i in posts])
        df['date'] = pd.to_datetime(df['created'], unit='s')
        df = df[['author', 'date', 'title', 'selftext', 'score', 'num_comments']]
        print(df.head())
        print(df.shape)
        df.to_csv('/Users/Alberto/PycharmProjects/00302_mastromarino/reddit_post.csv', index=False)

    except Exception as e:
        print("Unexpected error")
        exc_type, exc_obj, exc_tb = sys.exc_info()
        fname = os.path.split(exc_tb.tb_frame.f_code.co_filename)[1]
        print(exc_type, fname, exc_tb.tb_lineno)
```

Figura 2 Funzione historical_posts

La seconda funzione invece "*query posts*" ha un'impostazione simile alla precedente, ma con delle differenze importanti. Entrambe utilizzano l'oggetto scraper con la funzione search_submission, convertono i dati in un dataframe pandas, hanno un blocco try and except e salvano il dataframe in un file .csv, ma la funzione query_posts, ha due parametri in più, la query e lo score. In questo modo si potranno collezionare determinati post che presenteranno al loro interno la query che si andrà ad indicare; inoltre è possibile indicare

anche quanti upvotes dovranno avere i post che si andranno a ricercare tramite la funzione (es. è possibile ricercare i post con la parola “to the moon” con 10k upvotes).
Infine, nel ***__main__*** si assegnano a *subreddit*, *limit*, *before* e *after*, i parametri che si vogliono utilizzare per la funzione “*historical_posts*”; richiamando la classe “*Reddit_Scraper”* insieme a before e after, presenti nell’init, viene chiamata la funzione “historical_posts” con i parametri *subreddit* e *limit* indicati poco prima.
Stessa situazione abbiamo per la chiamata della funzione “*query_posts*” con la differenza che qui andremo ad indicare il valore dello score, e la query che si andrà a ricercare nei posts.

```
def query_posts(self, q, score):
    """..."""
    self.q = q
    self.score = score
    try:
        query_posts = scraper.search_submissions(
            q=self.q,
            before=after,
            after=before,
            score=self.score
        )
        df = pd.DataFrame([i.d_ for i in query_posts])
        df['date'] = pd.to_datetime(df['created'], unit='s')
        df = df[['author', 'date', 'title', 'selftext', 'score', 'num_comments']]
        print(df.head())
        print(df.shape)
        df.to_csv('/Users/Alberto/PycharmProjects/00302_mastromarino/query_post.csv', index=False)

    except Exception as e:
        print("Unexpected error")
        exc_type, exc_obj, exc_tb = sys.exc_info()
        fname = os.path.split(exc_tb.tb_frame.f_code.co_filename)[1]
        print(exc_type, fname, exc_tb.tb_lineno)
```

Figura 3 Funzione query_posts

```python
if __name__ == '__main__':
    # TODO To get historical posts

    subreddit = 'WallStreetBets'
    limit = 30000
    before = int(dt.datetime(2022, 5, 1).timestamp())
    after = int(dt.datetime(2022, 6, 1).timestamp())

    Reddit_Scraper(before, after).historical_posts(subreddit, limit)

    #TODO To get query posts

    q = "GME"
    score = ">10"
    before = int(dt.datetime(2022, 1, 1).timestamp())
    after = int(dt.datetime(2022, 5, 31).timestamp())

    Reddit_Scraper(before, after).query_posts(q, score)
```

Figura 4 Main della classe Reddit_Scraper

All'interno del file reddit_scraper.py, all'esterno della classe, viene utilizzata la funzione logging per tracciare gli eventi quando il codice viene eseguito. In questo modo si può tenere traccia degli eventuali errori, e si può vedere quando l'API PSAW effettua delle richieste al server per ottenere dati.

```python
handler = logging.StreamHandler()
handler.setLevel(logging.INFO)
logger = logging.getLogger('psaw')
logger.setLevel(logging.INFO)
logger.addHandler(handler)
```

Figura 5 Funzione logging built-in

```
https://api.pushshift.io/reddit/submission/search?subreddit=WallStreetBets&before=1653945497&after=1651356000&limit=1000&metadata=true
&sort=desc
```

Figura 6 Informazioni ottenute da logging

2.1.2 Estrazione dei dati finanziari

Per quanto riguarda l'estrazione dei dati finanziari è stato creato uno script denominato "*stock_scraper*" sempre object oriented, che ha come task principale l'estrazione di dati di un determinato stock.

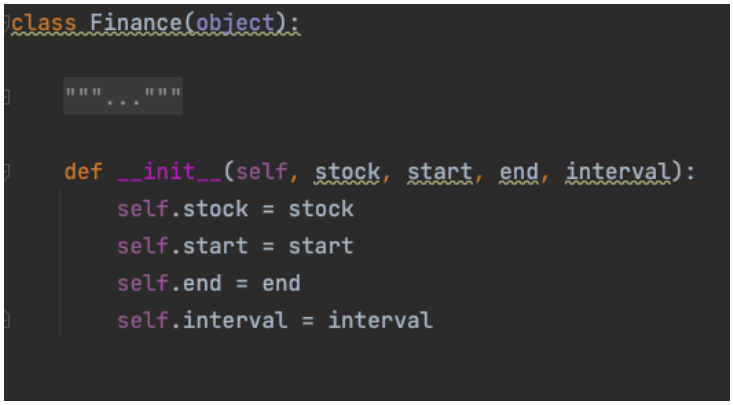

Figura 7 Classe Finance

Nel costruttore della classe, sono stati inseriti gli slot *stock*, *start*, *end* e *interval*. In questa classe è stata utilizzata l'API di Yahoo Finance, Yfinance , che ha permesso di poter scaricare qualsiasi dato inerente al mercato azionario senza nessun limite temporale. All'interno della classe è stata creata una funzione chiamata "*get_stock"*, formata da un blocco try and except che permetterebbe di capire quali siano gli errori nel codice, qualora entrasse nel blocco except. Nel blocco try, viene creato l'oggetto *info*, all'interno della quale si utilizza una funzione di Yahoo Finance, "*Ticker*" che permette di selezionare un determinato stock. Successivamente all'interno di *info_stock* si richiama l'oggetto *info* e al suo interno viene chiamata la funzione *history* che andrà a estrarre i dati ricercati.
A seguire *info_stock* viene convertito in un dataframe di pandas e si resetta l'indice poiché di default la colonna "Date" è un indice (impostazione predefinita di Yfinance). In aggiunta viene eseguito un print(df) per vedere cosa si ottiene, e per concludere si salva il dataframe in un file .csv, scegliendo quali colonne andare a mantenere nel file.

```python
def get_stock(self):

    """..."""

    try:
        info = yf.Ticker(self.stock)
        info_stock = info.history(start=self.start,
                                  end=self.end,
                                  interval=self.interval,


        )
        df = pd.DataFrame(info_stock)
        df = df.reset_index()
        print(df)
        plot = info_stock.plot(kind='line', figsize=(12,12))
        subplot = info_stock.plot(kind='line',figsize=(12,12), subplots=True)
        plt.show()
        df.to_csv('/Users/Alberto/PycharmProjects/00302_mastromarino/info_stocks.csv',
                  columns=['Date', 'Open', 'High', 'Low', 'Close', 'Volume'], index=False)

    except Exception as e:
        print("Unexpected error")
        exc_type, exc_obj, exc_tb = sys.exc_info()
        fname = os.path.split(exc_tb.tb_frame.f_code.co_filename)[1]
        print(exc_type, fname, exc_tb.tb_lineno)
```

Figura 8 Funzione get_stock

Con la chiamata di questa funzione si otterranno anche due grafici, il primo riguarda l'andamento del volume dello stock scelto, il secondo invece è composto da 7 subplot che riguardano i valori "Open, High, Low, Close, Volume, Dividends, Stock Splits"
Nel *__main__* si assegnano a *stock*, *start*, *end* e *interval*, i valori che si decide di utilizzare per la funzione "*get_stock*". L'intervallo che può essere utilizzato va da 1 minuto fino a 3 mesi (gli intervalli molto brevi che vanno da 1 minuto fino a 30 minuti possono essere utilizzati soltanto per un range che comprende gli ultimi 60 giorni).
Richiamando la classe Finance insieme alle variabili *stock*, *start*, *end* e *interval* presenti nel costruttore, si chiama la funzione "*get_stock*" per ottenere i dati azionari scelti inizialmente.

```
if __name__ == '__main__':

#TODO interval = "1m", "2m", "5m", "15m", "30m", "60m", "90m", "1h", "1d", "5d", "1wk", "1mo", "3mo"

    stock = 'GME'
    start = '2020-12-01'
    end = '2022-06-01'
    interval = '1d'

    Finance(stock, start, end, interval).get_stock()
```

Figura 9 Main della classe Finance

2.2 Data cleaning

Il secondo branch del progetto "*data_cleaning*" ha come task principale, ripulire il dataframe che si ottiene dopo lo scraping. Questo compito viene svolto dalla classe *Cleaner* che prenderà come attributo solamente il dataframe che verrà importato.

```
df = pd.read_csv('/Users/Alberto/PycharmProjects/00302_mastromarino/total_period.csv')
```

Figura 10 Lettura del dataframe da ripulire tramite pandas

```
class Cleaner:

    """..."""

    def __init__(self, dataframe):
        self.dataframe = dataframe
```

Figura 11 Classe Cleaner

La classe *Cleaner* è formata da quattro funzioni:

1. *"cleaning"* che viene applicata a tutto il dataframe, e rimuove i NaN, i caratteri speciali, i links, le emoji e i numeri. Infine, riordina il dataframe in ordine crescente, prendendo come riferimento la colonna "date".

```
def cleaning(self):
    """..."""
    self.dataframe.dropna(inplace=True)
    self.dataframe.sort_values(by='date', ascending=True, inplace=True)
    self.dataframe.replace(regex=True, inplace=True, to_replace=r'(@[A-Za-z0-9]+)|([^0-9A-Za-z \t])|(https?\S+)',
                           value=r'')
    self.dataframe['selftext'] = self.dataframe['selftext'].str.replace('\d+', '')
    self.dataframe['selftext'] = self.dataframe['selftext'].apply(str.lower)
    self.dataframe['title'] = self.dataframe['title'].str.replace('\d+', '')
    self.dataframe['title'] = self.dataframe['title'].apply(str.lower)
```

Figura 12 Funzione cleaning

2. *"cleaning_selftext"* che viene applicata soltanto alla colonna "selftext" (colonna che contiene i post degli utenti dopo lo scraping iniziale). La funzione elimina dalla

colonna selfext i post uguali (andando a rimuovere così l'intera riga), e va a rimuovere i post vuoti che contengono le parole "*removed*" e "*deleted*", post che sono stati precedentemente eliminati dagli utenti stessi o sono stati rimossi dai moderatori perché non rispettavano le regole del subreddit. "*Cleaning_selftext*" ha anche il compito di rimuovere le parole "ampx200b" e "ampxb", che non sono nient'altro delle lettere e numeri che è possibile ritrovare dopo lo scraping, incollate alla fine di una parola, poiché Reddit usa il markdown per formattare i commenti e i post; quindi, quando un utente vuole creare un extra spazio tra due paragrafi, nel nostro dataframe otterremmo questa dicitura (es. Gme to the moon**ampx200b** → Gme to the moon").

Successivamente si applica il regexp tokenizer alla colonna selftext andando a tokenizzare il tutto, e viene creata una nuova colonna "text_token"; utilizzando la lambda function insieme ad una list comprehension vengono rimosse le stopwords e i caratteri presenti in string.punctuation (!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~).

Inoltre, viene creata una colonna chiamata "text_string" all'interno della quale, utilizzando la lambda function e una list comprehension, si eliminano le parole con una lunghezza minore di due caratteri.

Infine alla colonna text_string viene applicato lo Snowball stemmer (stemming: processo di riduzione della forma flessa di una parola, alla sua forma radice, chiamata tema), creando così la colonna finale ripulita che verrà poi salvata nel file .csv con il nome di "*clean_post*"

```
def cleaning_selftext(self):
    """..."""
    self.dataframe.drop(self.dataframe.index[self.dataframe['selftext'] == 'removed'], inplace=True)
    self.dataframe.drop(self.dataframe.index[self.dataframe['selftext'] == 'deleted'], inplace=True)
    self.dataframe.drop_duplicates(subset=['selftext'],keep=False, inplace=True)
    self.dataframe['selftext'] = self.dataframe['selftext'].str.replace('ampx200b', '')
    self.dataframe['selftext'] = self.dataframe['selftext'].str.replace('ampxb', '')
    self.dataframe['text_token'] = self.dataframe['selftext'].apply(regexp.tokenize)
    self.dataframe['text_token'] = self.dataframe['text_token'].apply(
        lambda x: [item for item in x if item not in stop_words and item not in string.punctuation])
    self.dataframe['text_string'] = self.dataframe['text_token'].apply(
        lambda x: ' '.join([item for item in x if len(item) > 2]))
    self.dataframe['clean_post'] = self.dataframe['text_string'].apply(snowball.stem)
    # self.dataframe['is_equal'] = (self.dataframe['text_string'] == df['text_lemma'])
```

Figura 13 Funzione cleaning_selftext

3. *"cleaning_title"* che viene applicata alla colonna *title*; questa funzione esegue lo stesso task della funzione precedente *cleaning_selftext,* ovvero ripulisce la colonna *title,* eliminando i duplicati, e dopo aver tokenizzato il testo ed eliminato le stopwords e i caratteri speciali, in seguito verrà lemmatizzata tramite lo snowball stemmer. Infine, viene creata la colonna finale *clean_title* che successivamente verrà salvata nel file .csv.

```
def cleaning_title(self):
    """..."""
    self.dataframe.drop_duplicates(subset=['title'], keep=False, inplace=True)
    self.dataframe['title_token'] = self.dataframe['title'].apply(regexp.tokenize)
    self.dataframe['title_token'] = self.dataframe['title_token'].apply(
        lambda x: [item for item in x if item not in stop_words and item not in string.punctuation])
    self.dataframe['title_string'] = self.dataframe['title_token'].apply(
        lambda x: ' '.join([item for item in x if len(item) > 2]))
    self.dataframe['clean_title'] = self.dataframe['title_string'].apply(snowball.stem)
```

Figura 14 Funzione cleaning_title

4. *"csv"* funzione che si utilizza per andare a salvare in un file formato .csv, i dataframe ripuliti. Inizialmente tramite pandas viene riformattata la data (y-m-d -h-min-sec) poiché con la rimozione dei caratteri speciali, la data perde il trattino che era presente tra anno/mesi/giorni e tra ore/minuti/secondi (es. 2021-01-10 → 20210110); infine sempre tramite pandas con *dt.date* si decide di mantenere soltanto l'anno, il mese e il giorno, eliminando quindi il tempo.
   Nella funzione è possibile scegliere se salvare in un file formato .csv i titoli ripuliti o i post ripuliti, oppure entrambi. La classe però è stata costruita in modo tale da chiamare le funzioni in maniera alternata: si ripuliscono i post oppure si ripuliscono i titoli; questa decisione è stata presa poiché la maggior parte dei post degli utenti di Reddit, in particolare nel subreddit WallStreetBets, sono immagini/video o meme, quindi dopo la fase iniziale di scraping tanti post sono risultati vuoti oppure rimossi (deleted/removed), perciò quando si andrà a chiamare la funzione "*cleaning_title*" si noterà un numero maggiore di righe rispetto a quando si chiamerà la funzione

"*cleaning_selftext*".

```
def csv(self):
    """..."""
    self.dataframe['date'] = pd.to_datetime(self.dataframe['date'], format='%Y%m%d %H%M%S')
    self.dataframe['date'] = self.dataframe['date'].dt.date
    # self.dataframe.to_csv('/Users/Alberto/PycharmProjects/00302_mastromarino/total_period_cleaned_post.csv', columns=['date','clean_post'], index=False)
    self.dataframe.to_csv('/Users/Alberto/PycharmProjects/00302_mastromarino/total_period_cleaned_title.csv', columns=['date','clean_title'], index=False)
    print(f'Il dataframe dopo la ripulitura è composto da {df.shape} (righe/colonne)')
```

Figura 15 Funzione csv

```
if __name__ == '__main__':

    df_clean = Cleaner(df)

    df_clean.cleaning()
    # df_clean.cleaning_selftext()
    df_clean.cleaning_title()
    df_clean.csv()
```

Figura 16 Main della classe Cleaner

Prima della classe, dopo aver letto il dataframe che si andrà a ripulire, si utilizza una funzione print(f'{df.shape}') che stampa la grandezza del dataframe iniziale. Successivamente per vedere la differenza, questa print si trova anche all'interno della funzione "*csv*" nella classe Cleaner, e permette di vedere la differenza tra il dataframe iniziale e quello finale ripulito.

```
Inizialmente il dataframe è composto da (29980, 6) (righe/colonne)
Il dataframe dopo la ripulitura è composto da (6107, 9) (righe/colonne)
```

Figura 17 Esempio di print('f{df.shape}')

2.3 Sentiment analysis

Dopo aver ripulito il dataframe e salvato in formato .csv i post ripuliti o i titoli ripuliti si è passati allo script sentiment_analysis.py, che esegue una sentiment and emotion analysis.

```
# clean_df = pd.read_csv('/Users/Alberto/PycharmProjects/00302_mastromarino/total_period_cleaned_post.csv')
clean_df = pd.read_csv('/Users/Alberto/PycharmProjects/00302_mastromarino/total_period_cleaned_title.csv')
```

Figura 18 Lettura del dataframe

Per eseguire una sentiment and emotion analysis viene utilizzata una classe"*Sentiment*" che prende come attributo soltanto il dataframe ripulito. La classe è formata da sei funzioni, *get_sentiment_post* e *get_emotion_post* per quanto riguarda il sentiment e l'emotion dei post; *get_sentiment_title* e *get_emotion_title* per quanto riguarda il sentiment e l'emotion dei titoli; *high_sentiment* ricerca il sentimento più positivo/più negativo e *plotting* crea un countplot.

```
class Sentiment(object):


    def __init__(self, dataframe):
        self.dataframe = dataframe
```

Figura 19 Classe Sentiment

La funzione "*get_sentiment_post*" esegue una sentiment analysis dei post scaricati e si è scelto di utilizzare la librearia NLTK, il modulo nltk.sentiment. Creando l'oggetto *sid* contenente il SentimentIntensityAnalyzer, viene effettuato il calcolo del sentiment per ogni post della colonna "*clean_post*" utilizzando VADER (Valence Aware Dictionary and Sentiment Reasoner), sentiment analyzer che restituisce un dizionario con i valori del sentiment: negativo, neutrale e positivo. Dopo l'applicazione della funzione *polarity_scores* alla colonna *clean_post*, si crea la nuova colonna *polarity*, e tramite pandas usando la funzione pd.Series che permette di splittare in colonne separate il dizionario che si ottiene

nella colonna *polarity*, si avranno quattro colonne chiamate “neu”, “pos” ,“neg” e “compound”. Infine, la funzione crea una colonna chiamata *sentiment*, che tramite lambda function, conterrà al suo interno il valore “positive” qualora il compound sia > 0, “neutral” se = 0 e “negative” se < 0.

```
def get_sentiment_post(self):
    """..."""
    sid = SentimentIntensityAnalyzer()
    self.dataframe['polarity'] = self.dataframe['clean_post'].apply(lambda x: sid.polarity_scores(str(x)))
    self.dataframe = pd.concat([self.dataframe.drop(['polarity'],
            axis=1),self.dataframe['polarity'].apply(pd.Series)], axis=1)
    self.dataframe['sentiment'] = self.dataframe['compound'].apply\
        (lambda x: 'positive' if x > 0 else 'neutral' if x == 0 else 'negative')
    print(self.dataframe.head())
    # self.dataframe.to_csv('only_sentiment_post.csv', index=False)
    # if we want only positive and negative, we can choose to save this csv
```

Figura 20 Funzione get_sentiment_post

La funzione “*get_emotion_post*” esegue una emotion analysis dei post scaricati tramite la libreria NRCLex che è in grado di elaborare il testo e restituire 8 categorie di emozioni: fear, anger, anticipation, trust, surprise, sadness, disgust, joy e infine positive e negative che però non si terranno in considerazione (per quanto riguarda la sentiment anlysis). Tramite la funzione *affect_frequencies* come output si ottiene un dizionario, le cui chiavi sono le emozioni riconosciute dalla libreria, con un valore che va da 0.0 a 1.0. Come per il sentiment, attraverso la funzione pd.Series viene splittato il dizionario delle emotion in modo tale da avere una colonna singola per ogni tipo di emozione. L’ultimo task della funzione *get_emotion_post* prevede il salvataggio del dataframe in un file formato .csv, le cui colonne verranno specificate al momento del salvataggio.

```
def get_emotions_post(self):
    """..."""
    self.dataframe['emotions'] = self.dataframe['clean_post'].apply(lambda x: NRCLex(str(x)).affect_frequencies)
    self.dataframe = pd.concat([self.dataframe.drop(['emotions'], axis=1), self.dataframe['emotions'].apply
    (pd.Series)],axis=1)
    self.dataframe.to_csv('/Users/Alberto/PycharmProjects/00302_mastromarino/sentiment_emotion_total_post.csv',
                          columns=['date', 'clean_post', 'neg', 'neu', 'pos',
                                   'compound', 'sentiment',
                                   'fear', 'anger', 'anticipation', 'trust', 'surprise',
                                   'sadness', 'disgust', 'joy'], index=False)
```

Figura 21 Funzione get_emotion_post

La funzione "*get_sentiment_title*" insieme alla funzione "*get_emotion_title*" svolgono gli stessi task previsti per le funzioni precedenti, ovvero una sentiment analysis e una emotion analysis, con la sola differenza che vengono applicate ai titoli e non ai post.

```
def get_sentiment_title(self):
    """..."""
    self.dataframe['polarity'] = self.dataframe['clean_title'].apply(lambda x: sid.polarity_scores(str(x)))
    self.dataframe = pd.concat([self.dataframe.drop(['polarity'],
            axis=1),self.dataframe['polarity'].apply(pd.Series)], axis=1)
    self.dataframe['sentiment'] = self.dataframe['compound'].apply\
        (lambda x: 'positive' if x > 0 else 'neutral' if x == 0 else 'negative')
    print(self.dataframe.head())
    # self.dataframe.to_csv('only_sentiment_title.csv', index=False)
    # if we want only positive and negative, we can choose to save this csv
```

Figura 22 Funzione get_sentiment_title

```
def get_emotions_title(self):
    """..."""
    self.dataframe['emotions'] = self.dataframe['clean_title'].apply(lambda x: NRCLex(str(x)).affect_frequencies)
    self.dataframe = pd.concat([self.dataframe.drop(['emotions'], axis=1), self.dataframe['emotions'].apply
    (pd.Series)],axis=1)
    self.dataframe.to_csv('/Users/Alberto/PycharmProjects/00302_mastromarino/sentiment_emotion_total_title.csv',
                          columns=['date', 'clean_title', 'neg', 'neu', 'pos',
                                   'compound', 'sentiment',
                                   'fear', 'anger', 'anticipation', 'trust', 'surprise',
                                   'sadness', 'disgust', 'joy'], index=False)
```

Figura 23 Funzione get_emotion_title

Le ultime due funzioni della classe Sentiment sono "*high_sentiment*" e "*plotting*".
La prima stampa il post/titolo con il sentimento più positivo e successivamente il post/titolo con il sentimento più negativo.

La funzione “*plotting*” invece crea un grafico, tramite la librearia seaborn, un countplot per vedere come è distribuito il sentiment per i titoli o per i post.

```
def high_sentiment(self):
    """..."""
    print('Post/Titolo con il sentimento più positivo: ')
    print(self.dataframe.loc[self.dataframe['compound'].idxmax()].values)
    print('Post/Titolo con il sentimento più negativo: ')
    print(self.dataframe.loc[self.dataframe['compound'].idxmin()].values)
```

Figura 24 Funzione high_sentiment

```
def plotting(self):
    """..."""
    sns.countplot(y='sentiment',
                  data=self.dataframe,
                  palette=['#b2d8d8', "#008080", '#db3d13']
                  )

    plt.show()
```

Figura 25 Funzione plotting

Nel main dopo che è stato creato l’oggetto *df_emotion*, passandogli come parametro il dataframe ripulito dopo la fase di data cleaning, è possibile richiamare le funzioni presenti nella classe Sentiment. La classe è stata creata in modo tale da chiamare le funzioni in maniera alternata, viene eseguita una sentiment ed emotion analysis sui post oppure sui titoli.

```
if __name__ == '__main__':

    df_emotion = Sentiment(clean_df)

    # df_emotion.get_sentiment_post()
    # df_emotion.get_emotions_post()
    df_emotion.get_sentiment_title()
    df_emotion.get_emotions_title()
    # df_emotion.high_sentiment()
    # df_emotion.plotting()
```

Figura 26 Main della classe Sentiment

2.4 Topic Modelling

Il terzo script presente nel branch "*data_analysis*" è stato denominato *topic_modelling.py.* Infatti, è stato usato l'algoritmo di topic modelling, in particolare la Latent Dirichlet Allocation (LDA), che permette di determinare gli argomenti(topics) che sono presenti in un corpus. In questo caso è possibile scegliere due corpus differenti, il primo è il dataframe ripulito dei post mentre il secondo è il dataframe ripulito dei titoli: in questo modo si avranno due differenti LDA che andranno a produrre differenti topics.
Per questo script si è deciso di creare una classe "*LDA*" che prende come attributo soltanto il dataframe ripulito; all'interno sono presenti due funzioni "*tokenize_column*" e "*topic*".
La prima funzione ha il compito di trasformare il documento (il corpus), in questo caso la colonna che si andrà a selezionare (clean_post oppure clean_title), in una lista di token.
Per fare ciò si utilizza una lambda fuction, applicando il tokenizer della libreria NLTK.
In questo caso non ci sarà bisogno di eliminare stopwords o lemmatizzare poiché il dataframe è già stato ripulito con lo script data_cleaning.py.

```
# df_cleaned = pd.read_csv('/Users/Alberto/PycharmProjects/00302_mastromarino/total_period_cleaned_post.csv')
df_cleaned = pd.read_csv('/Users/Alberto/PycharmProjects/00302_mastromarino/total_period_cleaned_title.csv')

class LDA(object):

    def __init__(self, dataframe):
        self.dataframe = dataframe


    def tokenize_column(self):
        """..."""

        # self.dataframe['tokens'] = self.dataframe['clean_post'].apply(lambda x: nltk.word_tokenize(str(x)))
        self.dataframe['tokens'] = self.dataframe['clean_title'].apply(lambda x: nltk.word_tokenize(str(x)))
```

Figura 27 Classe LDA - Funzione tokenize_column

Quindi si procede direttamente a creare il modello utilizzando l'implementazione della LDA, attraverso la libreria Gensim.
In primis, con la funzione "*topic*" viene costruito un vocabolario partendo dai dati scelti (post/titoli) che sono stati tokenizzati e salvati nella colonna *tokens.* Dopo aver trasformato

il corpus (bow_corpus) tramite list comprehension, viene addestrato il modello per la topic modelling.

Tra i parametri che sono presenti, vi sono:

- corpus: corpus di documenti da utilizzare per il training
- id2word: dizionario che definisce il mapping degli id con le parole
- num _topics: il numero di topics da estrarre.
- random_state: serve come seed, nel caso si volesse ripetere il processo di formazione.
- update_every: numero di documenti da iterare per ogni aggiornamento.
- chunk_size: controlla quanti documenti vengono elaborati alla volta nell'algoritmo di training; la sua grandezza può influenzare la qualità del modello.
- passes: numero di passaggi attraverso il corpus durante il training.
- alpha: può essere impostato su "auto" oppure su "asymmetric"
- eval_every: la log perplexity viene stimata in tanti aggiornamenti, impostando a uno, rallenta il training.
- iterations: numero di volte che si ripete un particolare ciclo su ogni documento.
- per_word_topic: se impostato su True, il modello calcola anche un elenco di argomenti più probabili ordinati in ordine decrescente per ogni parola; ogni parola verrà assegnata ad un argomento.

```
def topic(self):
    """..."""

    dataset = self.dataframe['tokens']
    dic = gensim.corpora.Dictionary(dataset)
    bow_corpus = [dic.doc2bow(x) for x in dataset]
    lda_model = gensim.models.ldamodel.LdaModel(corpus=bow_corpus,
                                                id2word=dic,
                                                num_topics=10,
                                                random_state=10,
                                                update_every=1,
                                                chunksize=200,
                                                passes=5,
                                                alpha='auto',
                                                eval_every=1,
                                                iterations=100,
                                                per_word_topics=True)

    for idx, topic in lda_model.print_topics():
        print('Topic: {} \nWords: {}'.format(idx, topic))


    vis = pyLDAvis.gensim_models.prepare(lda_model, bow_corpus, dic)
    # pyLDAvis.save_html(vis, 'LDA_Visualization_post.html')
    pyLDAvis.save_html(vis, 'LDA_Visualization_title.html')

    figure = plt.figure(figsize=(30, 30)) #15 30
    for i in range(10):
        df = pd.DataFrame(lda_model.show_topic(i), columns=['term', 'prob']).set_index('term')
        df = df.sort_values('prob')
```

Figura 28 Funzione topic

```
        sns.barplot(x='prob', y=df.index, data=df, label='Cities', palette='Reds_d')
        plt.xlabel('probability')

    plt.show()
```

Figura 29 Funzione topic

La funzione topic inoltre con un print, stampa a schermo le parole più frequenti e i topic.

Figura 30 Topic ottenuti per i titoli

Figura 31 Topic ottenuti per i post

Infine, tramite la librearia PyLDavis è possibile visualizzare i topic individuati dal Topic Model appena creato. Attraverso la funzione save_html si ottiene un file html che è possibile visualizzare dal browser, che successivamente viene salvato con il nome di LDA_Visualization.

Nel main si crea la variabile df, e dopo aver creato l'oggetto lda con all'interno la classe "*LDA*", passandogli df come parametro, vengono chiamate le due funzioni per ottenere il Topic Model.

```
if __name__ == '__main__':

    df = df_cleaned
    lda = LDA(df)
    lda.tokenize_column()
    lda.topic()
```

Figura 32 Main della classe LDA

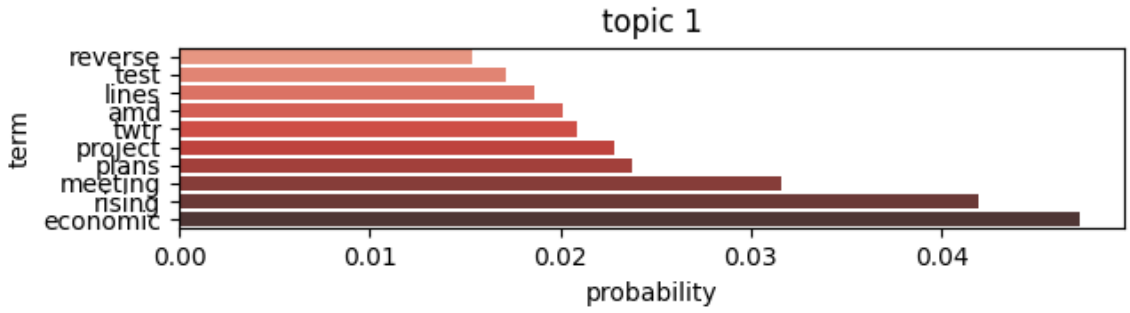

Figura 33 Plot che otteniamo per ogni topic

Utilizzando la classe "*LDA*", si ottiene anche un plot per ogni topic, come in Figura 33, che mostra i termini, insieme alla loro probabilità.

Nella Figura 34 invece, è possibile vedere il file formato .html prodotto dalla libreria PyLDavis.

L'area del cerchio rappresenta l'importanza di ogni argomento sull'intero corpus, la distanza tra il centro dei cerchi indica la somiglianza tra i topics. Per ogni argomento, l'istogramma sul lato destro elenca i 30 termini più rilevanti; in questo caso nella Figura 34 viene mostrato LDA_Visualization_post.html e si possono notare tra i termini più importanti: stock, price, shares, short, company see, earnings, gme, buy, week.

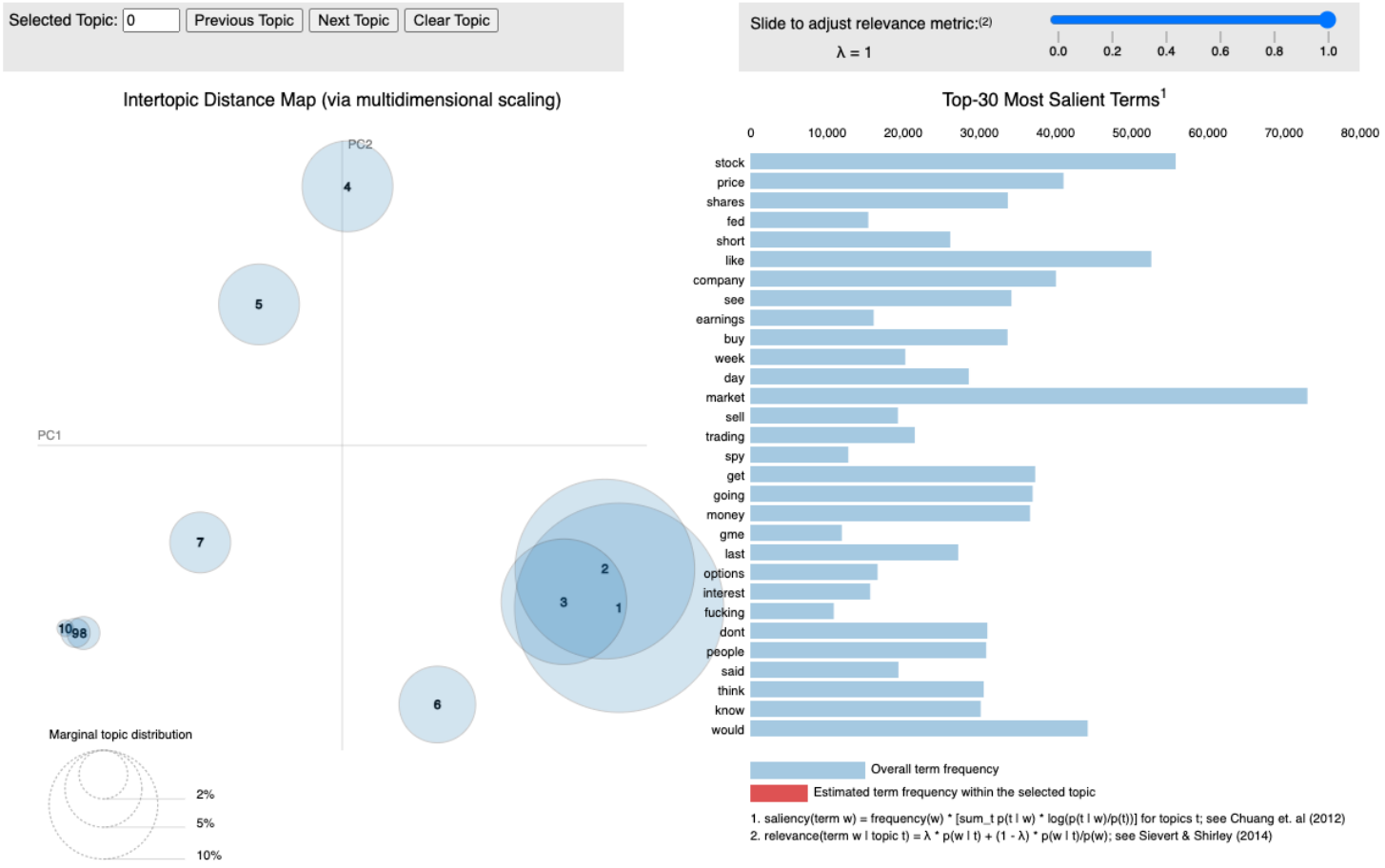

Figura 34 Esempio di LDA_Visualization.html

2.5 Data Visualization

2.5.1 Data visualization per lo stock scelto: GME

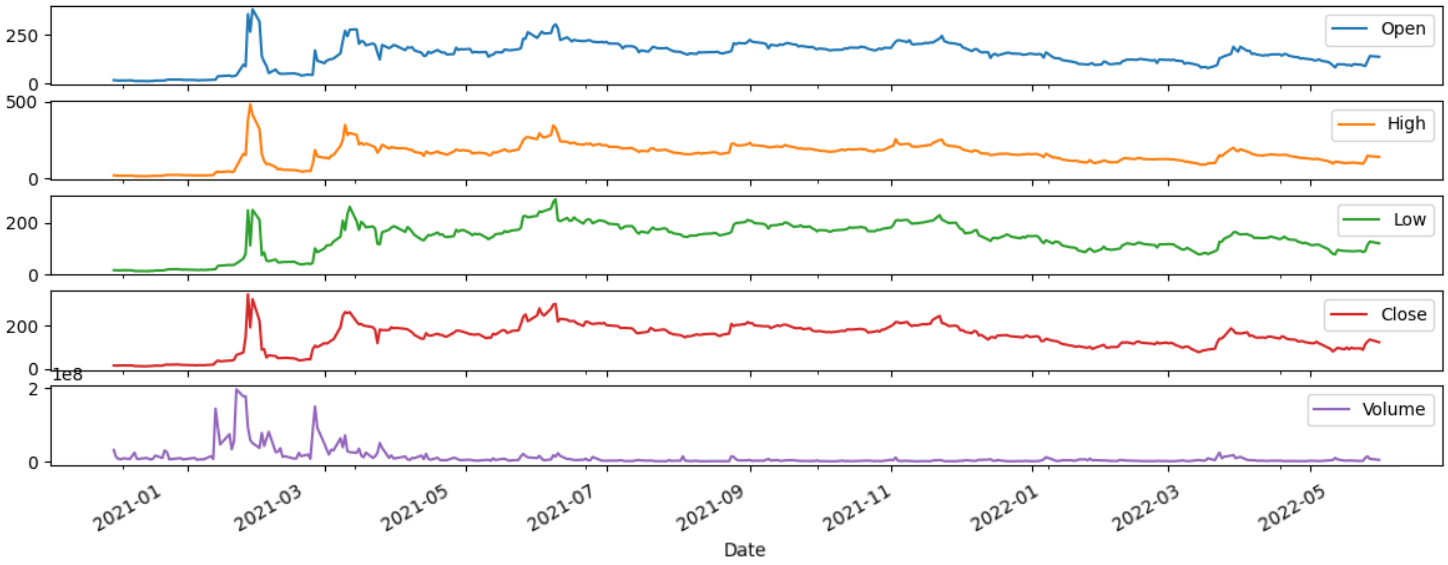

Figura 35 Sublot per lo stock scelto ottenuto dallo stock scraper

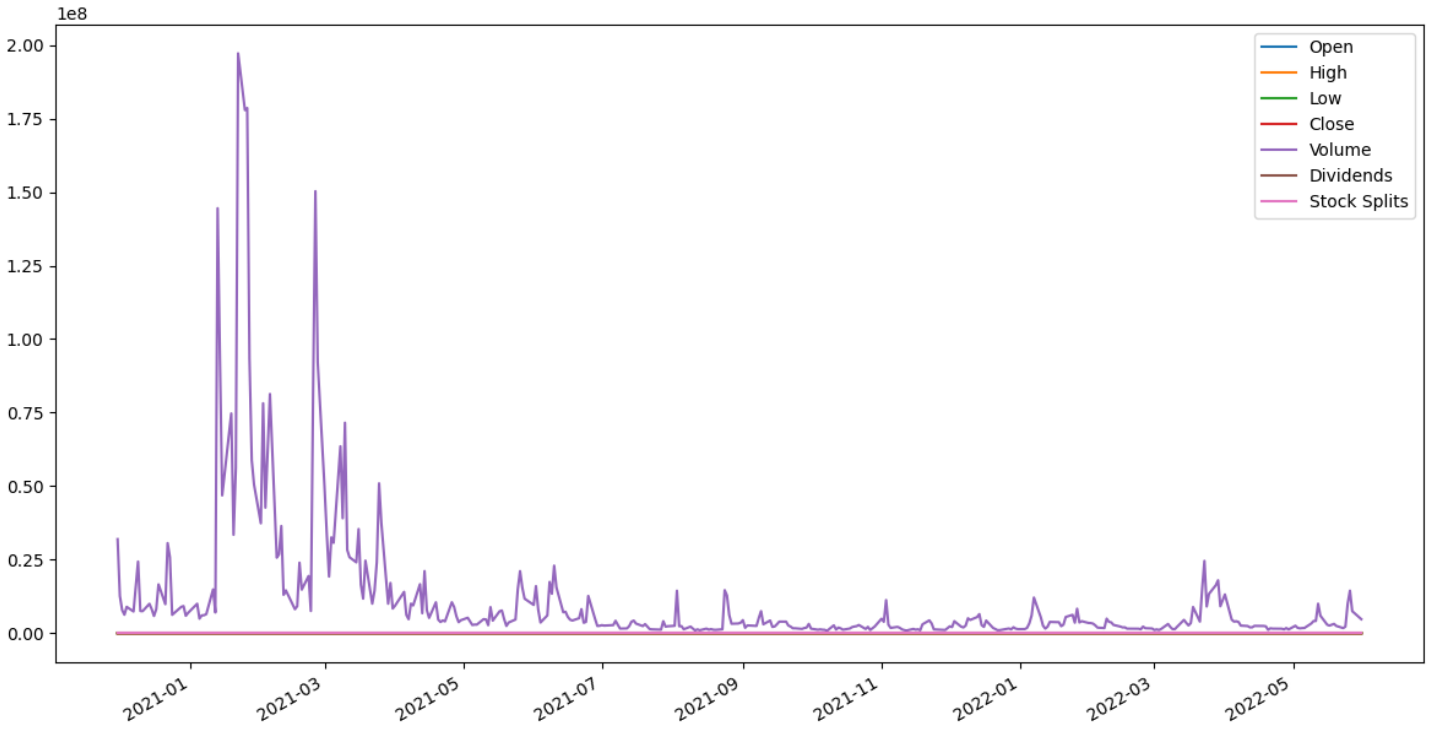

Figura 36 Plot ottenuto dallo stock scraper

Quello che si può osservare dai grafici in Figura 35 e Figura 36 è l'andamento del titolo scelto inizialmente. Nel primo grafico in Figura 35, sono presenti diversi subplot, ognuno che indica una determinata area:

- Open: il prezzo del titolo al momento dell'apertura del mercato.
- Close: il prezzo del titolo al momento della chiusura del mercato.
- High: il prezzo più alto che tocca il titolo durante la giornata.
- Low: il prezzo più basso che tocca il titolo durante la giornata.
- Volume: il volume di scambi del titolo durante la giornata.
- Dividends: dividendi.
- Stock Splits: frazionamento dei propri titoli azionari in più azioni.

In figura 35 si può notare come il volume delle azioni abbia un forte picco a fine Gennaio 2021, esattamente il 27 Gennaio, quando è avvenuto il famoso "GME short squeeze" per merito degli utenti del subreddit WallStreetBets.

2.5.2 Data visualization per il sentiment

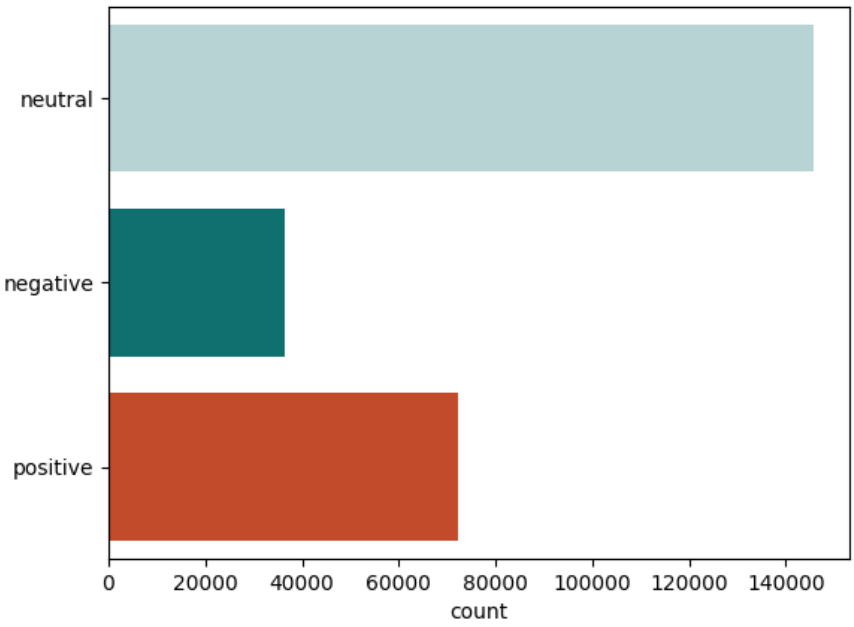

Figura 37 Countplot per il sentiment dei title

Dalla Figura 37 invece è possibile notare il sentiment per quanto riguarda i titoli, prevalentemente neutrale, con 145.908 titoli con sentiment neutro, mentre si possono contare 72236 titoli con sentiment positivo e 36410 titoli con sentiment negativo.

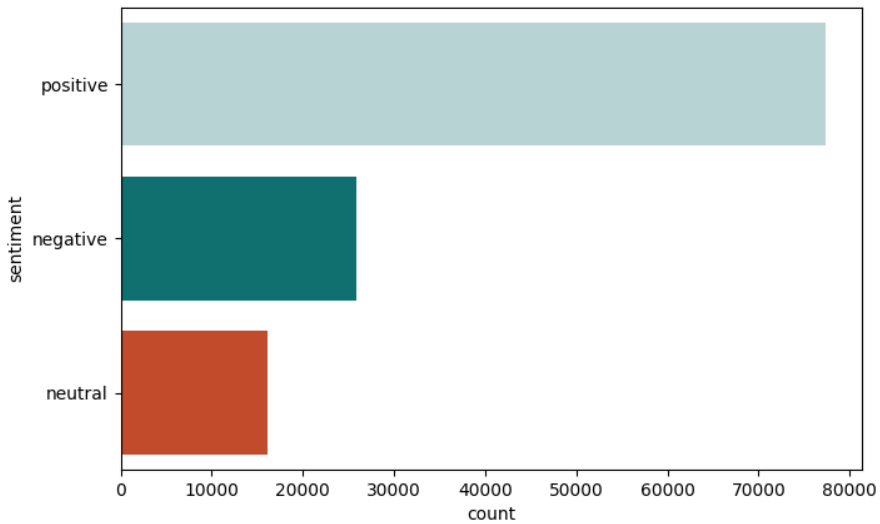

Figura 38 Countplot per il sentiment dei post

In Figura 38 invece si notano delle differenze, poiché in questo grafico il sentiment prevalente è quello positivo; infatti, si contano 77412 post con sentiment positivo, mentre sono presenti 25876 post con un sentiment negativo e 16074 post con sentiment neutrale.

2.5.3 Data visualization per le emotion

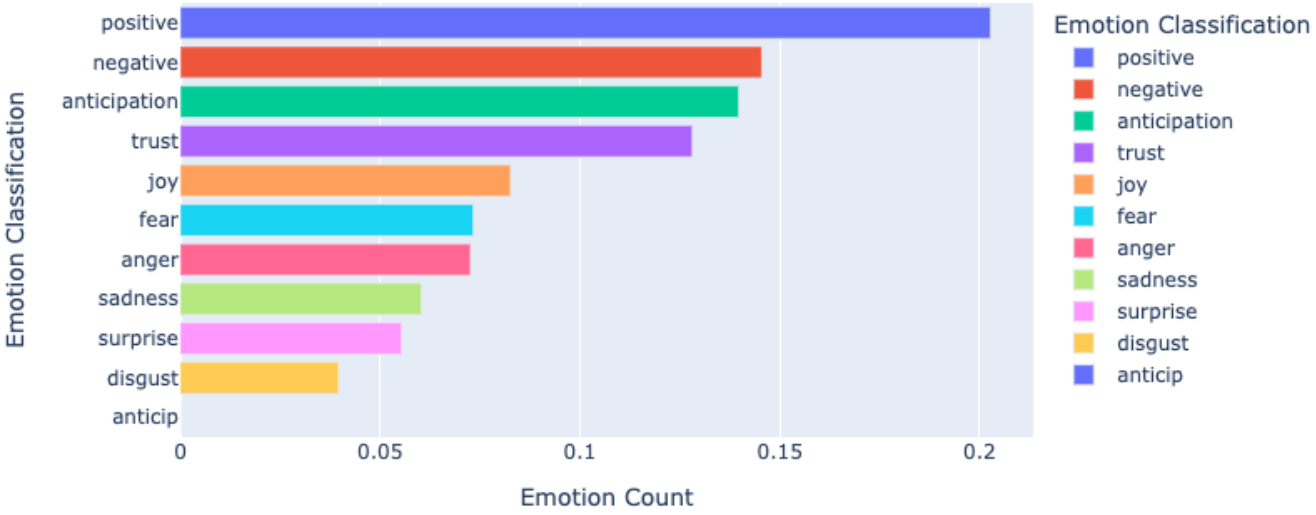

Figura 39 Affect Frequencies title

Per quanto riguarda le emotion invece, si è scelto di utilizzare NRCLex, che misura le emozioni dal corpo di un testo; questa libreria è stata costruita su circa 27.000 parole ed è basata sul lessico del National Research Council Canada (NRC) e sui sinonimi WordNet della libreria NLTK.

Si ottengono 6 diverse emozioni unite alle emotion positive e negative, per un totale quindi di 8 emozioni riconosciute dalla libreria.

Nella Figura 39 è possibile vedere le frequenze di ogni emozione all'interno di ogni titolo; qui si può vedere che l'emozione positiva ha le frequenze più alte all'interno dei titoli scritti dagli utenti. Le frequenze per ogni emozione sono uguali a:

- positive: 0,20
- negative: 0,15
- anticipation: 0,14

- trust: 0,13
- joy: 0,09
- fear: 0,07
- anger: 0,07
- sadness: 0,06
- surprise: 0,05
- disgust: 0,04

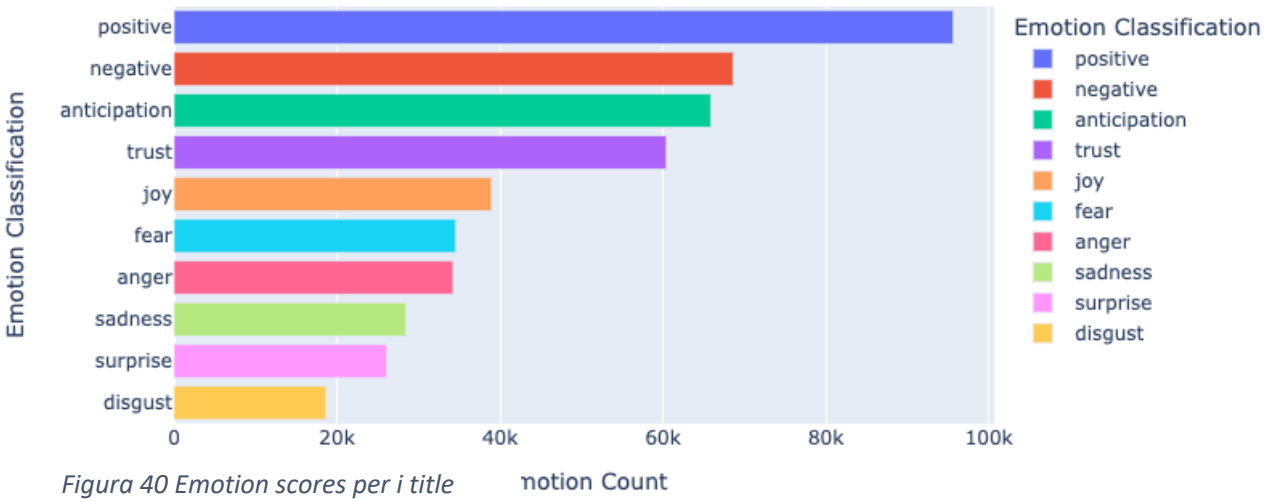

Figura 40 Emotion scores per i title

Nella Figura 40 invece è possibile vedere l'Emotion Count per quanto riguarda i title:

- positive: 95619
- negative: 68623
- anticipation: 65902
- trust: 60423
- joy: 38972
- fear: 34562
- anger: 34250
- sadness: 28447
- surprise: 26118

- disgust: 18666

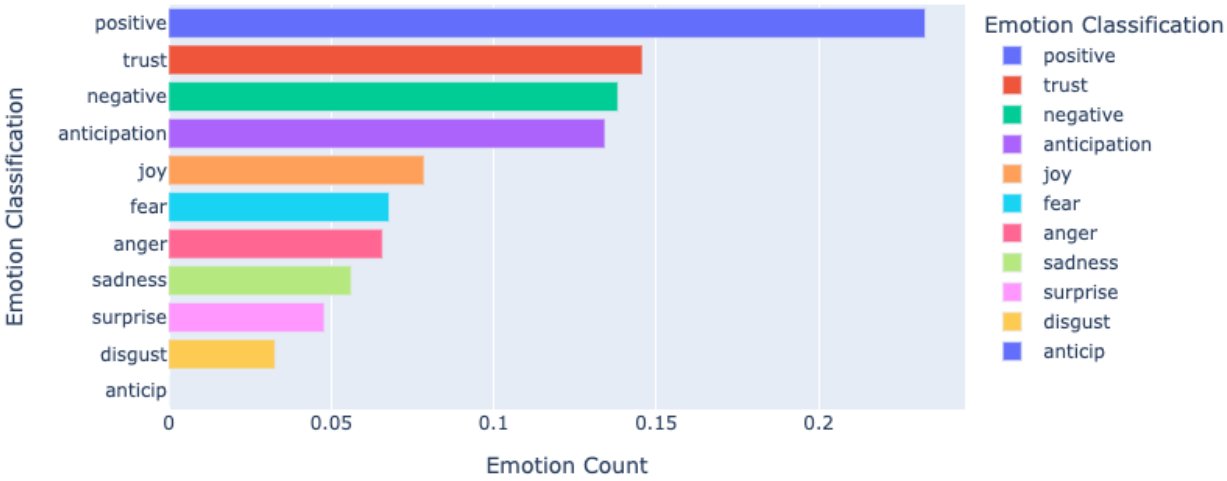

Figura 41 Affect frequencies post

Nella Figura 41 è possibile vedere le frequenze di ogni emozione all'interno di ogni post; anche qui si può notare che l'emozione positiva ha le frequenze più alte all'interno dei post scritti dagli utenti ed è seguita dall'emotion trust. Le frequenze per ogni emozione sono uguali a:

- positive: 0.231
- trust: 0.146
- negative: 0.139
- anticipation: 0.134
- joy: 0.079
- fear: 0.068
- anger: 0.066
- sadness: 0.056
- surprise: 0.048
- disgust: 0.033

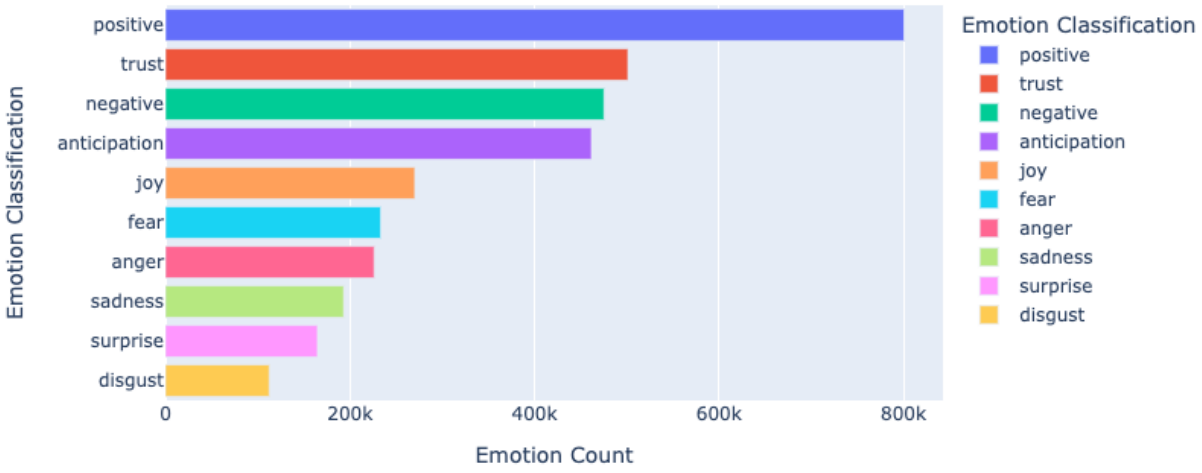

Figura 42 Emotion scores per i post

Nella Figura 42 invece è possibile vedere l'Emotion Count per quanto riguarda i post:

- positive: 800589
- trust: 501320
- negative: 475392
- anticipation: 461741
- joy: 270341
- fear: 233279
- anger: 226265
- sadness: 193163
- surprise: 164777
- disgust: 112488

N.B. Si possono evincere delle differenze tra i sentimenti positivi e negativi tra i grafici presenti in Figura 37-38 con quelli invece che si trovano in Figura 40-42, questo perché vengono utilizzate due librerie differenti, di cui una tiene soltanto conto di sentiment positivo, neutro e negativo, facendo una media del sentiment presente all'interno di una frase, mentre la seconda ha 6 emotion in più, e tiene conto delle singole parole.

2.6 Creazione e validazione modello

Nella scelta del modello, per determinare se il prezzo di uno stock diminuisce o aumenta sulla base del sentiment dei post/titoli, è stata utilizzata una Linear Discrimant Analysis. Si è cercato quindi di capire se il sentiment dei post/dei titoli vada a influenzare l'andamento di uno determinato stock.

Il procedimento che segue è stato applicato sia per il sentiment dei post, che per il sentiment dei titoli; quindi, sono stati ottenuti due risultati differenti.

Inizialmente dopo che sono stati letti i due file formato .csv tramite pandas, per evitare la conversione automatica in stringa della colonna 'date', effettuata dalla libreria pandas, viene utilizzato il parametro parse_dates, e successivamente viene convertita la colonna stessa come tipo intero.

Stessa cosa avviene per l'altro file contenente tutti i dati azionari dal dicembre 2020 fino a maggio 2022. In aggiunta vengono scelte quali colonne mantenere in entrambi i dataframe; per il df1, contenente la sentiment emotion, si effettua un cambio di nome della colonna, questo servirà per effettuare il merge poco dopo.

Prendendo in esame il primo dataframe(df1), si effettua un groupby & mean, passandogli come parametro per effettuare il groupby, la colonna '*Date*': in questo modo si ottiene una media del sentiment giornaliera per ogni post scritto nell'intervallo che va dal 1 dicembre 2020 fino al 31 maggio 2022, e questo permette di diminuire la grandezza del dataframe.

Dopo aver modellato il df1, viene creata una nuova colonna nel df2, chiamata '*Closing_difference'*, contenente i valori provenienti dalla differenza tra il Close Price e l'Open Price. Questa colonna è servita poi come riferimento, per la creazione della colonna '*Trend*'; tramite l'utilizzo della libreria numpy e della sintassi where, viene assegnato il valore 1, se il Closing_Difference è >= 0 oppure, viene assegnato il valore 0; in questo modo è possibile notare quindi un trend vero e proprio nel dataframe, e si può leggere nella colonna, il valore 1, quando c'è un aumento nello stock, perché il prezzo di chiusura è maggiore di quello di apertura , e il valore 0, quando il prezzo di chiusura è inferiore a quello di apertura.

Infine, dopo aver modellato entrambi i dataframe, si procede con un merge, per ottenere un singolo dataframe, usando come colonna di riferimento 'Date' presente in entrambi. In questo modo si ottiene un dataframe ridotto di 365 righe (dal 1° dicembre 2020 fino al 31 Maggio 2022 si contano in totale 547 giorni; non contando però i giorni in cui il mercato è chiuso è necessario sottrarre 164 giorni, arrivando ad un numero pari a 383; bisogna tenere in considerazione anche il fatto che possa mancare qualche giornata nel file iniziale dopo aver effettuato lo scraping iniziale, in questo caso 18 giorni).

```
df1 = pd.read_csv('sentiment_emotion_total_title.csv',parse_dates=['date'])
df1['date'] = df1['date'].astype('int')

df2 = pd.read_csv('total_stocks.csv', parse_dates=['Date'])
df2['Date'] = df2['Date'].astype('int')


df1 = df1[['date', 'compound', 'sentiment', 'neg', 'neu', 'pos']]
df1 = df1.rename(columns={'date': 'Date'})
df1 = df1.groupby(['Date']).mean()


df2 = df2[['Date', 'Open', 'Close', 'Volume']]
df2['Closing_Difference'] = df2['Close'] - df2['Open']
df2['Trend'] = np.where(df2['Closing_Difference'] >= 0,  1, 0)


merge = df1.merge(df2, how='inner', on='Date')
# merge.to_csv('merge_post.csv',index=False)
merge.to_csv('merge_title.csv',index=False)
```

Figura 43 Modellazione dei dataframe e merge

Dopo aver creato un dataset unico chiamato merge, si procede all'assegnazione della variabile target e delle features, splittando il dataset in training set e test set, in proporzione 0,80/0,20 con la funzione *train_test_split().*
Infine, dopo aver settato il modello, si stampano a schermo le metriche che si ottengono.

```
#Create the feature data set
X = merge
X = np.array(X.drop(['Trend'], 1))

#Create the target data set
y = np.array(merge['Trend'])

#Split the data into 80% training and 20% testing data sets
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

model = LinearDiscriminantAnalysis().fit(x_train, y_train)

#Get the models predictions/classification
predictions = model.predict(x_test)
print(predictions)

#Show the models metrics
print(classification_report(y_test, predictions))
```

Figura 44 LDA model

Come si può evincere in Figura 45, il modello ha un'accuracy del 81%, una precision del 76% per quanto riguarda il valore 0 (trend in cui il Close Price è minore dell'Open Price) e una precision del 91% per quanto riguarda il valore 1 (trend in cui il Close Price è >= dell'Open Price)

```
              precision    recall  f1-score   support

           0       0.76      0.95      0.84        40
           1       0.91      0.64      0.75        33

    accuracy                           0.81        73
   macro avg       0.84      0.79      0.80        73
weighted avg       0.83      0.81      0.80        73
```

Figura 45 Report LDA title

	precision	recall	f1-score	support
0	0.78	0.95	0.85	40
1	0.92	0.67	0.77	33
accuracy			0.82	73
macro avg	0.85	0.81	0.81	73
weighted avg	0.84	0.82	0.82	73

Figura 46 Report LDA post

Per quanto riguarda invece i risultati ottenuti per i post, si ottiene un'accuracy del 82%, più alta soltanto di un punto percentuale, e una precision del 78% per quanto riguarda il valore 0, e del 92% per quanto riguarda il valore 1.

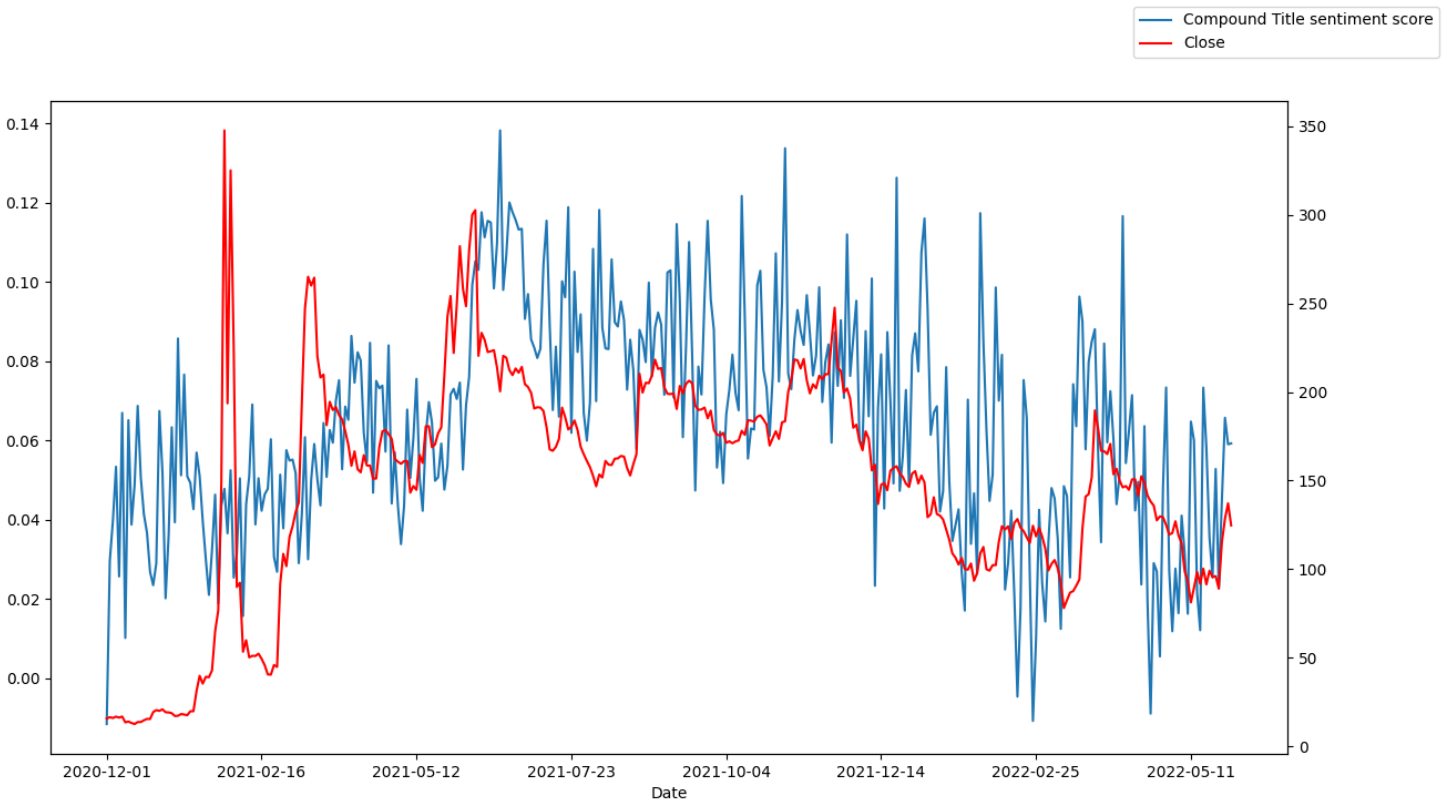

Figura 47 Sentiment title and Close Price

In via definitiva sono stati creati due grafici, in Figura 47 è possibile notare il cambiamento del Close Price nell'arco temporale di un anno e 6 mesi, insieme al cambiamento del sentiment per i titoli. Nell'asse delle ascisse è stato inserito l'arco temporale, mentre

nell'asse delle ordinate a sinistra è stato inserito il compound score, e sulla destra il prezzo raggiunto dal Close Price.

Si può notare che ad accezione di fine gennaio 2021, per esattezza il 27, giorno in cui il Close Price ha un forte picco verso l'alto, distaccandosi dal sentiment dei title, durante tutto l'arco temporale, il Close Price tenda a seguire il sentiment dei titoli.

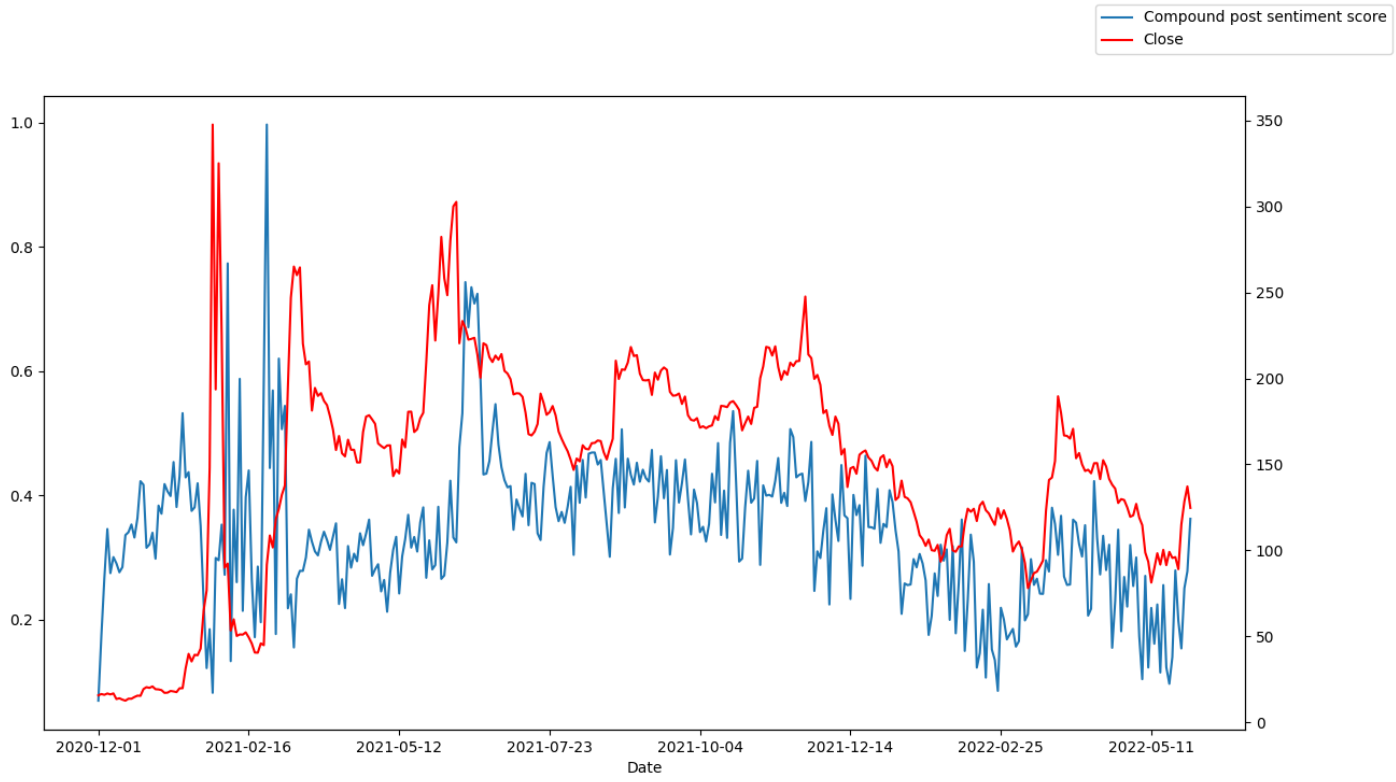

Figura 48 Sentiment post and Close Price

Per quanto riguarda invece il grafico presente in Figura 48, si nota una forte correlazione del sentiment dei post con il Close Price a fine gennaio, e un forte sentiment positivo anche durante tutto il mese di febbraio, seguito da un Close Price poco più basso. Anche in questo grafico è possibile notare che l'andamento del Close Price, è molto simile all'andamento del sentiment dei post. Facendo un confronto con il grafico precedente, è importante dire che su Reddit, ma soprattutto nel subreddit WallStreetBets, gli utenti si esprimono attraverso l'utilizzo di meme divertenti o video e si tende quindi a scrivere di meno nei post e ad esprimere la propria opinione nel titolo.

3. Conclusioni

In conclusione, per rispondere alla research question di questo progetto, è possibile affermare che, per quanto riguarda l'analisi del sentiment, sia dei post che dei titoli, tenendo conto dei risultati ottenuti secondo il modello utilizzato, il sentiment tende ad influenzare l'andamento del titolo preso in esame, in questo caso GameStop.
Bisogna però mettere in chiaro che sono diversi i fattori che influenzano l'andamento di un titolo azionario, non soltanto il sentimento degli utenti permette di prevedere come si muoverà il mercato, ma ci sono tanti elementi esterni che possono andare a influenzare totalmente e in maniera improvvisa il mercato azionario (es. tweet di Elon Musk, shorting da parte di Hedge Funds etc.).